UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Faculty of Information and Communication Technology

BACHELOR THESIS

By

Đỗ Minh Tú

BA11-094

Information and Community Technology (ICT)

Title:

Development Mobile Application for booking Personal Trainer

Internal supervisor: Huynh Vinh Nam

External supervisor: Le Thanh Nghi

Hanoi, July, 2025

To whom it may concern,

I, Mr. Le Thanh Nghi, certify that the thesis report of Mr. Do Minh Tu is qualified to be presented in the Internship Jury 2024-2025.

Hanoi, 07/2025

Supervisor's signature

ACKNOWLEDGEMENTS

Firstly, I would like to thank everyone who helped me and guided me during my internship and completing my thesis.

I would like to thank my Internal Supervisor, MSc. Huynh Vinh Nam, for support and encouragement in completing the thesis as well as the internship. He played a very important role during my internship.

I would like to thank my External Supervisor, Mr. Le Thanh Nghi for always giving me advice and guiding me very enthusiastically during my internship at BIDV. He also helped me to complete my graduate project.

I would like to thank the BIDV company for giving me the opportunity to intern here. I had the chance to learn, develop and gain more experience for myself during my internship.

LIST OF ABBREVIATIONS

USTH	University of Science and Technology of Hanoi
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interface
JSON	JavaScript Object Notation
ID	Identification
NoSQL	Not only SQL (Not only Structured Query Language)
PT	Personal Trainer

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

This thesis presents the design and implementation of a cross-platform mobile application to solve the problem of connecting customers with personal trainers (PT) in the Vietnamese market. I want to target freelance PTs and customers who want to find PTs to work out without having to go through gyms. This both increases connectivity and improves pricing.

The application was developed using React Native for the frontend, Node.js and Express.js for the backend, and MongoDB is aNoSQL database. The system provides core functionalities including user authentication with email verification, PT search and filtering, availability management, a two-way booking process (client requests and PT confirms), and a chat feature.

The end product is a stable, secure and user-friendly application that creates a reliable platform for better connection between clients and personal trainers. Thereby, improving the quality of training and health.

Keywords: personal trainer, booking system, React Native, Node.js, MongoDB, cross-platform app, better connection clients-PT.

CHAPTER I: INTRODUCTION

1.1 Context and Motivation

Nowadays, diseases tend to be younger, so people have more and more needs to exercise. However, when exercising, we cannot lack a leader, a teacher, a person who encourages us every time we exercise, that is the personal trainer. Gym models are increasingly developing, along with the lack of discipline as well as too many low-quality PTs and prices are also unclear and much higher than reality. There is a lot of information related to fraudulent sales of exercise packages.

At the same time, Vietnam has seen explosive growth in mobile technology, with extremely high smartphone usage and a population that is very comfortable using apps for work, entertainment and service. The mobile phone is now the primary way people access information and services. This creates a perfect opportunity to use technology to solve the problems in the fitness market.

That's why I want to build a mobile application that helps clients and personal trainers easily connect together with clear information. Both clients and personal trainers are my customers. My application focuses on helping Vietnamese people easily go to the gym, play sports and improve public health.

1.2 Literature Review

To understand the current market, we conducted a literature review of existing solutions in Vietnam. The market is currently dominated by two main platforms: ModunFit and LEEP.APP.

ModunFit operates as a 'super app' marketplace on Zalo, offering a wide variety of services from many partner gyms. While it provides flexibility, it is not a specialized tool for connecting with individual trainers."

LEEP.APP offers a more direct connection using technology like its 'PT-iMatch' algorithm. However, it is also a broad ecosystem that includes group classes and gym passes, not focusing exclusively on the one-on-one client-trainer relationship."

Table 1. Comparative Analysis of Existing Platform

Criteria	**Modun Fit**	**LEEP.APP**	**My project's goal**
Platform	Web	Mobile App	Mobile App
Model	Marketplace for partner gyms	Fitness Ecosystem	Specialized Platform
Target trainers	Trainers at parter gyms	Network of PTs & partner gyms	Freelance Personal Trainers
Key focus	Flexible booking at many places	Smart matching and multiservice	Simplifying the direct connection

1.3 Thesis Structure

This thesis is organized into six chapters:

- **Chapter I - Introduction:** This chapter explains why this project is necessary. It provides the general context, analyzes the problems in the current market, and presents the main goals.
- **Chapter II - Objectives:** This chapter lists the specific, detailed goals of the project. It breaks down the main goal into smaller, measurable objectives and defines the project's scope.
- **Chapter III - Requirement Analysis:** This chapter defines what the application must do. It details all the functional requirements, non-functional requirements and use cases for the system.
- **Chapter IV - Materials and Methods:** This chapter explains how the application was designed. It describes the technology stack that was chosen and presents the detailed system design, including the architecture, database schema, and use cases.
- **Chapter V - Testing and Result:** This chapter presents the evidence that the application works. It describes the testing process and shows the results with screenshots to verify that all key features function correctly.
- **Chapter VI - Conclusion and Future Work**: This final chapter summarizes the entire project. It provides a conclusion on what was achieved, discusses the project's limitations, and suggests ideas for future improvements.

CHAPTER II: OBJECTIVES

2.1 Project Objective

The primary objective of this project is to design, implement, and evaluate a functional, cross-platform mobile application that provides a reliable and efficient solution to the problem of connecting clients with freelance personal trainers.

I will consider this project successful when I have a stable application that actually works as designed. This means it must pass all the basic tests for its main features and qualities, like being secure and easy to use. In the end, the goal is to create an application that can work well and help society be better.

2.2 Desired Features

To achieve the project's main goal, the application must be implemented with a set of core features. These features are grouped into logical modules, each containing specific sub-features designed to meet the needs of both Clients and Personal Trainers.

2.2.1 User Authentication

This is the foundational feature that handles user access and security.

Sub-features:

- **Account Registration with Email Verification:** New users can create an account by providing their basic information. A verification code is sent to their email to confirm their identity and prevent fake accounts.
- **Secure User Login:** Registered users can log in to the system using their email and a securely stored (hashed) password.
- **Role Selection:** During registration, users must select their role as either a "Client" or a "Personal Trainer (PT)," which determines their access level.

2.2.2 Trainer Discovery and Search

This feature allows clients to find the right trainer for their needs.

Sub-features:

- **Search and Filter System:** A powerful search interface allows clients to find trainers. They can apply filters based on criteria like specialty (e.g., Yoga, Gym), price range, and location.
- **Detailed Trainer Profiles:** Clients can view comprehensive profiles for each trainer, which include their professional bio, photos, qualifications, hourly rates, and work schedule.

- **Ratings and Reviews:** To build trust, clients can see ratings and read reviews left by other users for each trainer.

2.2.3 Booking and Scheduling System

This is the core business logic of the application, managing the entire booking lifecycle.

Sub-features:

- **Availability Management (for PTs):** Trainers have a dedicated interface to set and update their weekly work schedule, defining the exact time slots they are available for booking.
- **Session Booking (for Clients):** Clients can request a session by selecting an available time slot directly from a trainer's calendar.
- **Booking Management (for both):** Both clients and trainers have a dashboard to view and manage their bookings. This includes:
    - PTs can **accept** or **decline** new booking requests.
    - Clients can see the status of their bookings (pending, confirmed, completed).
    - Both users can cancel a booking (subject to cancellation rules).

2.2.4 User Interaction and Communication

This feature focuses on managing user profiles and enabling communication.

Sub-features:

- **Personal Profile Management:** Both clients and trainers can edit their personal information, such as their name, profile picture, and password.
- **Direct Messaging (Chat):** A built-in chat system allows a client and their booked trainer to communicate directly and privately to discuss session details.
- **Notification System:** The application sends automated push notifications to keep users informed about important events, such as new booking requests, confirmations, cancellations, and new chat messages.

2.3 Expected Outcome

My main goal with this project was always to build more than just code. The expected outcome for me is a finished, stable application that people can actually use. I want clients to feel confident when they look for a trainer, and I want good trainers to have a fair chance to grow their careers. If the app can do that, then I'll consider it a success.

CHAPTER III: REQUIREMENT ANALYSIS

3.1 Introduction:

This chapter explains the rules for our application. Before we can design or build anything, we first need to understand exactly what the app must do. This analysis will be the guide for all the design and development work in the next chapters.

3.2 Functional Requirement

Functional requirements are a list of all the actions and features that the application must have. We have two main types of users: the Client and the Personal Trainer (PT). The table below shows the key functions for each user. The app has 2 types of users: Client and PT

Table 2. List of App Functions

User	Function Description
All users	A new user can register for an account as either a "Client" or a "Personal Trainer". A registered user can log in and log out of the system. A logged-in user can view and edit their own profile information.
Client	The Client can search for personal trainers based on filters (by name, specialization, ..). The Client can view a trainer's detailed profile, including their schedule. The Client can send a booking request for an available time slot. The Client can view a list of their upcoming and past bookings. The Client can cancel a booking. The Client can chat with their PT.
PT	The Personal Trainer can create and manage their professional profile. The Personal Trainer can set and update their weekly availability schedule. The Personal Trainer can view and manage incoming booking requests (confirm/reject). The Personal Trainer can view their schedule of confirmed bookings. The Personal Trainer can chat with their clients.

3.3 Non-Functional Requirements

- **Performance:** The app must be fast. It should not make users wait.
- **Usability:** The app must be very easy to use. People should understand it without help.
- **Reliability:** The app must work all the time and not crash.
- **Security:** User information, like passwords, must be kept very safe.
- **Compatibility:** The app must work well on both Android phones and iPhones.

3.4 Use Case Analysis

To better understand how users will interact with the system, we used Use Case diagrams and scenarios.

3.4.1 Use Case Diagram

Use Case diagram helps to visualize the main functions of the system from the user's perspective. I will show key features of 2 user roles: client and PT.

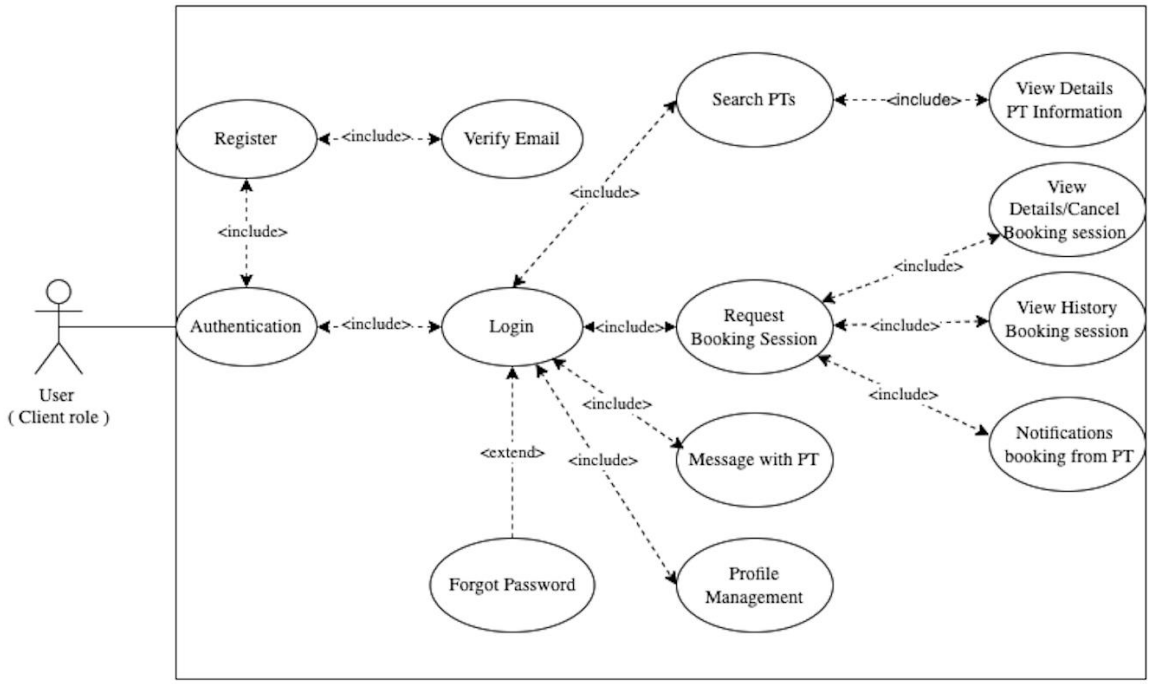

Figure 1. Use Case diagram for client role

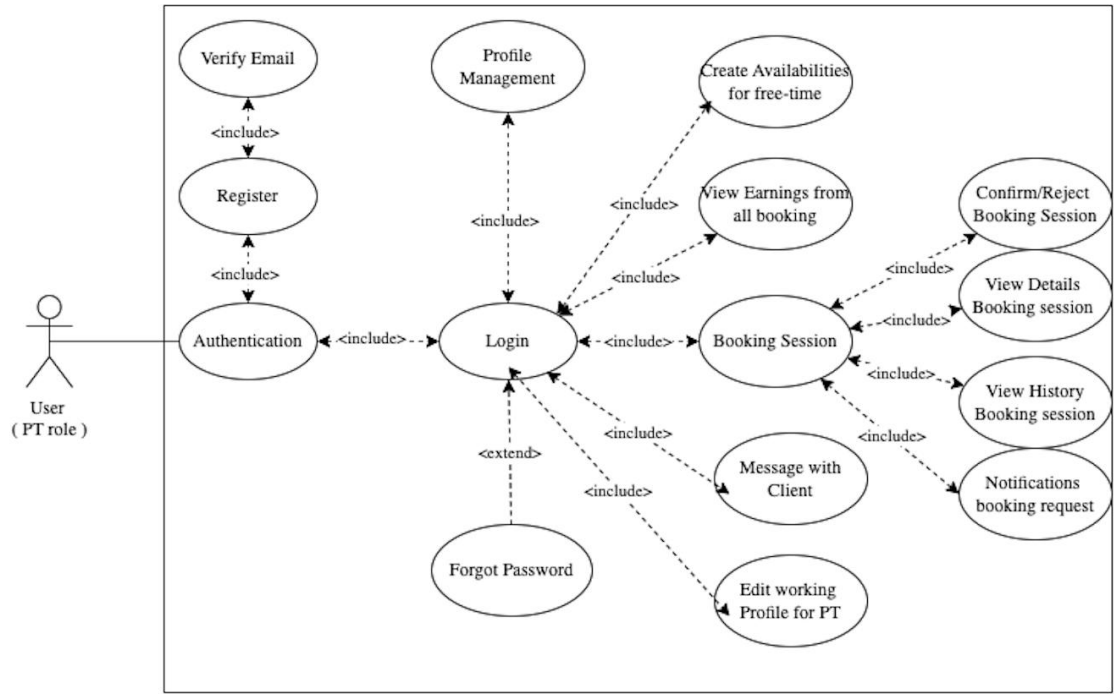

Figure 2. Use Case diagram for PT role

3.4.2 Use Case Scenario

This section provides detailed descriptions for the main use cases in the system to clarify the step-by-step interactions between the user and the application.

Use Case 1: Register and Email Verification

User case for Register initiation

Table 3 . Use Case scenario for register

Use Case	Register
Actor	New User (Client or Personal Trainer)
Description	Describes how a new user creates a personal account
Preconditions	The user opens the application and clicks on “Sign Up” text in the login screen.
Main Success Scenario	1. The user enters their full name, a valid email address, username, password, date of birth and confirm password. 2. The user selects their desired role: “Client” or “Personal Trainer”. 3. The user click “Register: button 4. The system validates the input (checks if the email is in valid

Use Case	Register
	format) 5. The system checks if the email already exists in the database. 6. The system hashes the password. 7. The system creates a new user record in the database with status of “pending verification”. 8. The system generates a random verification code (4-digit number) and sets expiration time. 9. The system sends an email contain verification code to the user’s email address 10. The application navigates the user to the “Verification Screen”.
Alternative	6a. Email Already Exists: 1. If the system finds the email is already registered, it displays an error message “This email already exists”. 2. The use case returns to step 2.
Postconditions	A new user record with "pending_verification" status is created. A verification code has been sent to the user's email.

Use Case for Verification Email

Table 4. Use case scenario for verification email

Use Case	Verify email and active account
Actor	New user (Client or PT)
Description	Describes how the user enters the verification code received via email to activate their account.
Preconditions	1. The user fills all the fields and successfully clicks on the “Sign Up” button. 2. The user is on “Verification Screen”
Main Success Scenario	1. The user checks their email inbox and finds the verification code. 2. The user enters the 4-digit verification code into the input field on the screen. 3. The user taps the "Verify" button. 4. The system sends the code and the user's email to the server. 5. The system validates verify code 6. The code is correct. The server updates the user's account status from "pending_verification" to "active". 7. The server generates a JWT access token for the user. 8. The server sends the JWT token back to the mobile app. 9. The app saves the token, navigates the user to the home screen.
Alternative	5a. Incorrect Code:

Use Case	Verify email and active account
	1. If the submitted code does not match the stored code, the server returns an error. 2. The application displays an error message: "Invalid verification code." 5b. Expired Code: 1. If the submitted code has passed its expiration time, the server returns an error. 2. The application displays an error message: "Verification code has expired. Server auto generates new code for the user.
Postconditions	The user is authentication and navigate to home screen based on their role

Use Case 2: Login

Table 5. Use Case for Login

Use Case	Login
Actor	Registered user (Client or PT)
Description	Describes how a user login into their account
Preconditions	The user opens an application and has an account.
Main Success Scenario	1. The user enters their registered email and password. 2. The user taps the "Login" button. 3. The system sends the credentials to the server for verification. 4. The server finds the user account by email. 5. The server compares the submitted password with the securely stored (hashed) password. 6. The credentials are correct. The server generates a secure JWT access token. 7. The server sends the JWT access token back to the mobile app. 8. The app saves the token and navigates the user to their appropriate home screen.
Alternative	4a. User Not Found / Incorrect Credentials: 1. If the email does not exist or the password does not match, the server returns an authentication error. 2. The application displays an error message: "Invalid email or password." 3. 3. The use case returns to step 1.
Postconditions	The user is successfully authenticated and can access the application.

Use Case 3: Create booking request for client role

Table 6. Use Case scenario for client booking

Use Case	Create Booking Request
Actor	Client
Description	This use case describes the process for a client to request and book a training session with a personal trainer.
Preconditions	1. The Client must be logged into the system. 2. The Client is viewing the detailed profile of a specific trainer.
Main Success Scenario	1. The system displays the trainer's availability schedule with open time slots. 2. The Client selects a desired date and an available time slot. 3. The system shows a summary of the booking (trainer, time, price). 4. The Client taps the "Booking" button to send the request. 5. The system validates that the time slot is still available. 6. The system creates a new booking record in the database with the status set to "pending". 7. The system sends a notification to the trainer about the new booking request. 8. The system displays a success message to the Client, confirming the request has been sent.
Alternative	5a. Time Slot is no longer available: 1. If the server finds that the selected time slot was just booked by another user, it returns an error. 2. The application refreshes the schedule, and the use case returns to step 1. 5b. Other Validation Errors: 1. If the system detects other issues (e.g., client has a conflicting booking, network error), it displays an appropriate error message. 2. The user can then correct the issue or try again.
Postconditions	A new booking record with a "pending" status is created in the database, and the selected trainer is notified of the request.

Use Case 4: Management booking request for PT

Table 7. Use Case scenario for PT manages request booking

Use Case	Manage booking request
Actor	PT
Description	This use case describes how a personal trainer reviews, accepts, or declines an incoming booking request from a client.
Preconditions	1. The PT is logged into the system. 2. There is at least one "pending" booking request from a client.
Main Success Scenario	1. The PT navigates to their "Booking Requests" screen. 2. The system displays a list of all pending requests. 3. The PT selects a request to view its details (client name, requested time). 4. After reviewing, the PT taps the "Accept" button. 5. The system sends a request to the server to update the booking's status. 6. The server changes the booking status from "pending" to "confirmed". 7. The server updates the trainer's schedule, making that time slot unavailable for others. 8. The system sends a confirmation notification to the client. 9. The system refreshes the PT's booking list, moving the booking to the "Confirmed" tab.
Alternative	4a. Trainer Declines the Request: 1. From step 4 of the Main Success Scenario, the PT taps the "Reject" button instead. 2. The system sends a request to the server. 3. The server changes the booking status to "rejected". 4. The server keeps the time slot available for other clients. 5. The system sends a notification to the client informing them of the decline. 6a. Booking Already Processed: 1. If the booking was already canceled by the client or processed, the server returns an error. 2. The application displays an error message like: "This booking is no longer available to be actioned." 3. The application refreshes the booking list.
Postconditions	The booking's status is updated to "confirmed" or "reject" in the database. The client is notified of the decision, and the trainer's schedule is updated accordingly.

Use case 5: Manage Availability Schedule for PT

Table 8. Use Case scenario for PT manages availability

Use Case	Manage Availability Schedule
Actor	PT
Description	This use case describes how a personal trainer sets and updates their weekly work schedule to show clients when they are available for booking.
Preconditions	1. The PT is logged into the system. 2. The PT has completed their working profile.
Main Success Scenario	1. The PT navigates to the "My Schedule" or "Availability" screen. 2. The system displays a weekly or monthly calendar interface. 3. The PT selects a specific date they want to add availability for. 4. The PT taps an "Add Time Slot" button. 5. The system displays a form to input the start time and end time for a new available slot. 6. The PT confirms the new time slot. 7. The system adds the new time slot to the schedule with the status "available". 8. The PT can repeat steps 3-7 to add multiple slots for different days. 9. The new available time slots are immediately visible to clients who view the PT's profile.
Alternative	5a. Time Slot Conflict: 1. If the PT tries to add a time slot that overlaps with an existing one (either another available slot or a confirmed booking), the system displays an error message: "This time slot conflicts with an existing entry." 2. The use case returns to step 5. 9a. Remove an Available Time Slot: 1. The PT selects an existing "available" time slot on their schedule. 2. The PT taps a "Delete" button. 3. The system removes the time slot from the PT's availability. 4. The time slot is no longer visible to clients.
Postconditions	The trainer's availability schedule in the database is updated with the new time slots, which are now open for clients to book.

Use case 6: Client Search Personal Trainer

Table 9. Use Case scenario for Searching Personal Trainer

Use Case	Search PT
Actor	Client
Description	This use case describes how a client searches for and filters personal trainers to find one that meets their needs.
Preconditions	The Client must be logged into the system.
Main Success Scenario	1. Client opens the app and navigates to “Home Screen” and clicks a search bar. 2. The system updates filter indicator, show selected specialization, show recommendation PT. 3. The clients click the “Search” button. 4. The system queries PTProfile collection with filters, checking availability slots for each PT. 5. The system displays PT cards with key info (username, spcialization, ..) 6. The client clicks on the PT card they want. 7. The system navigates to a detailed PT profile.
Alternative	4a. No PTs match criteria 1. 1. If no trainers in the database match the filter criteria, the server returns an empty list. 2. 2. The application displays a message to the user, such as: "No trainers found matching your criteria. Try broadening your search."
Postconditions	The Client can see a list of relevant personal trainers and can tap on any result to view a detailed profile.

Use case 7: Personal Trainer update their work profile

Table 10. Use Case for PT updating their work profile

Use Case	Update Work Profile
Actor	Personal Trainer
Description	This use case describes how a personal trainer edits and updates their professional profile information.
Preconditions	The PT must be logged into the system.
Main Success Scenario	1. The PT navigates to their "My Profile" screen. 2. The PT taps the "Edit Profile" button. 3. The system displays a form with the PT's current information (e.g., bio, specialties, hourly rate, experience). 4. The PT modifies one or more fields. 5. The PT taps the "Save Changes" button. 6. The application sends a PUT request to the server with the updated profile data. The server validates the new data. The server updates the corresponding document in the PTProfiles collection in the database. 9. The system displays a success message: "Your profile has been updated successfully.
Alternative	7a. Invalid Data Input: 1. If the PT enters invalid data (e.g., a non-numeric value for the hourly rate), the server returns a validation error. 2. The application displays an error message next to the invalid field, such as: "Please enter a valid number for the hourly rate." 3. The use case returns to step 4.
Postconditions	The PT's profile information is successfully updated in the database and is immediately visible to any clients viewing their profile.

CHAPTER IV: MATERIALS AND METHODS

4.1 Introduction

This chapter describes the "materials" (the technologies) and the "methods" (the design process) that were used to build the personal trainer booking application. The first part, Materials, will explain the technology stack we chose. The second part, Methods, will detail the step-by-step design process, from analyzing user needs to creating the final technical blueprint for the system.

4.2 The technology stack

Choosing the right technologies is a critical step. Our goal was to select a modern, efficient, and cohesive "tech stack" for building our application.

4.2.1 Frontend: React Native

React Native is a framework created by Facebook for building mobile applications. It allows developers to write code once and run it on both Android and IOS devices.

The main benefits of React Native include:

- **Cross-platform Development:** One codebase works on both Android and iOS
- **Fast Development:** Developers can see changes instantly while coding
- **Native Performance:** Apps run almost as fast as native applications
- **Large Community:** Many developers use React Native, so help is available

For my project, these benefits allowed us to develop for both platforms quickly, creating an app without the high cost of native development. The component-based architecture was particularly useful for building a clean and reusable user interface for features like trainer profiles and booking lists

4.2.2 Backend: NodeJs and ExpressJs

Node.js Runtime Environment

Node.js allows us to run JavaScript code on a server, not just in web browsers. This means I can use the same programming language for both the mobile app and the server.

Key features of Node.js:

- **Event-driven:** Handles many requests at the same time efficiently.
- **Non-blocking:** The server doesn't wait for slow operations to finish.
- **Fast development:** JavaScript developers can work on both frontend and backend.

ExpressJs Web Framework

Express.js is a web framework that makes it easier to build web servers with Node.js. It helps us create APIs that our mobile app can communicate with.

Express provides:

- **Routing:** Defines which code runs for different web addresses.
- **Middleware:** Code that runs before handling requests.
- **Error handling:** Manages problems that occur during requests.
- **Static files:** Serves images, documents, and other files.

This event-driven and non-blocking architecture is crucial for our booking app. It ensures the server stays responsive and fast, even when many users are searching for trainers or sending messages at the same time.

4.2.3 Database: MongoDB

MongoDB is a NoSQL database that stores data in a flexible, document-based format. Unlike traditional databases that use tables and rows, MongoDB uses collections and documents.

Benefits of MongoDB:

- **Flexible Schema**: Easy to change data structure as the app grows.
- **JSON-like Format**: Data looks similar to JavaScript objects.
- **Scalable**: Can handle large amounts of data and users.
- **Fast Queries**: Good performance for mobile applications.

This flexibility was the key reason for our choice, as it allows us to easily store complex trainer profiles with varied information, such as multiple skills and custom schedules, which would be difficult to manage in a traditional SQL database.

4.2.4 API Design and Communication

The communication foundation between my mobile application (the client) and the backend server is the standard HTTP (Hypertext Transfer Protocol) request-response model. In this model, the client sends an HTTP request to the server to ask for data or perform an action. The server then processes the request and sends back an HTTP response containing the result. To structure and standardize this communication, we designed our API following the REST (Representational State Transfer) architectural style. REST is not a technology itself, but a set of guiding principles for building web services that are simple, scalable, and reliable. It uses standard HTTP methods to perform actions on data "resources".

The main HTTP methods used in our project are:

- **GET:** The app asks the server to get or read information.
- **POST:** The app tells the server to create something new.
- **PUT:** The app tells the server to update existing information.
- **DELETE:** The app tells the server to delete something.

I chose this REST approach because it is a very popular standard. It is simple, reliable, and makes it easy for the mobile app and the server to understand each other perfectly. This ensures that data is sent and received correctly.

In conclusion, the choice of React Native, Node.js, and MongoDB forms a highly efficient, full-stack JavaScript solution. The key advantage is that the entire system works with a JSON-like data format from the MongoDB documents to the Node.js API and finally to the React Native components. This consistency reduces complexity and made it the ideal choice for building my mobile application

4.2.5 Development Environment

Table 11. Development Tools and Environment

Category	Tool/Technology	Purpose
Code Editor	Visual Studio Code	The main editor for writing project code
Frontend	React Native CLI	Framework for building the "bare" cross-platform app.
	Xcode	To build, run, and manage the iOS simulator.
Backend	NodeJs	JavaScript runtime for building the server.
	npm (Node Package Manager)	For managing server-side libraries.
Database	MongoDB Atlas	Cloud-based database service for data storage.
API Testing	Postman	To test the backend API endpoints manually.
Version Control	Github	For tracking code changes and project backup.

4.3 System Design

We followed a structured design process before writing any code to ensure the final application was well-planned and logical.

4.3.1 System Architecture

A 3-Tier Client-Server Model

The application is designed using a Client-Server model, and more specifically, it is structured as a 3-Tier Architecture. This is a standard and reliable way to build modern applications because it separates the system into three logical layers, making it organized, scalable, and easy to maintain.

The Three Tiers:

- **The Presentation Tier (The Client)**
  - This is the React Native mobile app that runs on the user's phone.
  - Its only job is to show the user interface (the screens, buttons, and text) and to send the user's requests to the server.
- **The Logic Tier (The Server)**
  - This is the Node.js backend system. It is the "brain" of our application.
  - It handles all the important rules and work, like checking a user's password, creating a new booking, or figuring out a trainer's schedule.
- **The Data Tier (The Database)**
  - This is our MongoDB database.
  - Its only job is to store and give back data when the server asks for it. It doesn't do any thinking; it just keeps the information safe.

The Client and Server talk to each other over the internet using a REST API. The diagram below shows how these three tiers are connected.

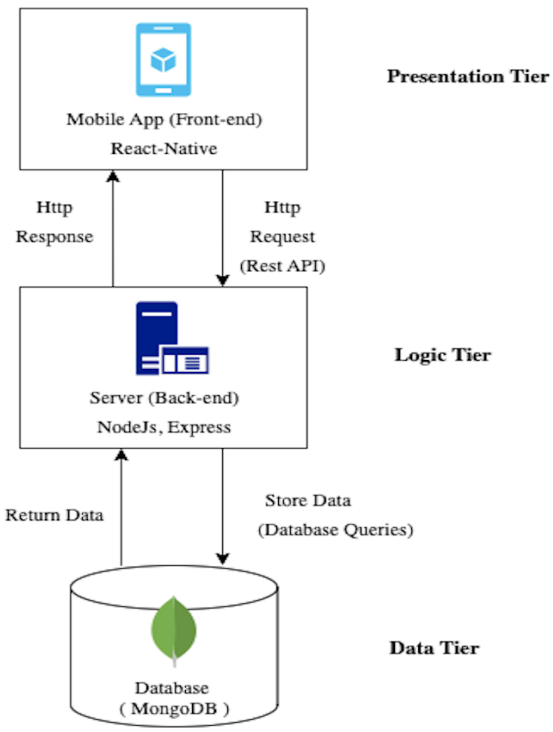

Figure 3. 3-tier Architecture

4.3.2 Backend is Organized: The MVC Pattern

To keep the server code organized, we followed the MVC (Model-View-Controller) architectural pattern. The MVC pattern is a popular design choice that separates an application's logic into three interconnected components, making the code more modular and easier to manage.

Here is what each part does:

- **Model:**
    - The Model is in charge of the data. It is the only part that is allowed to talk directly to our MongoDB database. For example, a booking model knows how to save a new booking or find an old one.
- **View:**
    - In my project, the "View" is not a screen that you can see. Instead, it is the JSON data that the server sends back to the mobile app. The React Native app then uses this JSON data to show the real screens to the user.
- **Controller:**
    - The Controller is the manager. When the mobile app sends a request, the Controller is the first to receive it. It tells the Model what to do (like "get all trainers"). After the Model gives the data back, the Controller prepares the View (the JSON data) and sends it back to the app.

Using the MVC pattern was very helpful. It made our code much easier to read, test, and update in the future.

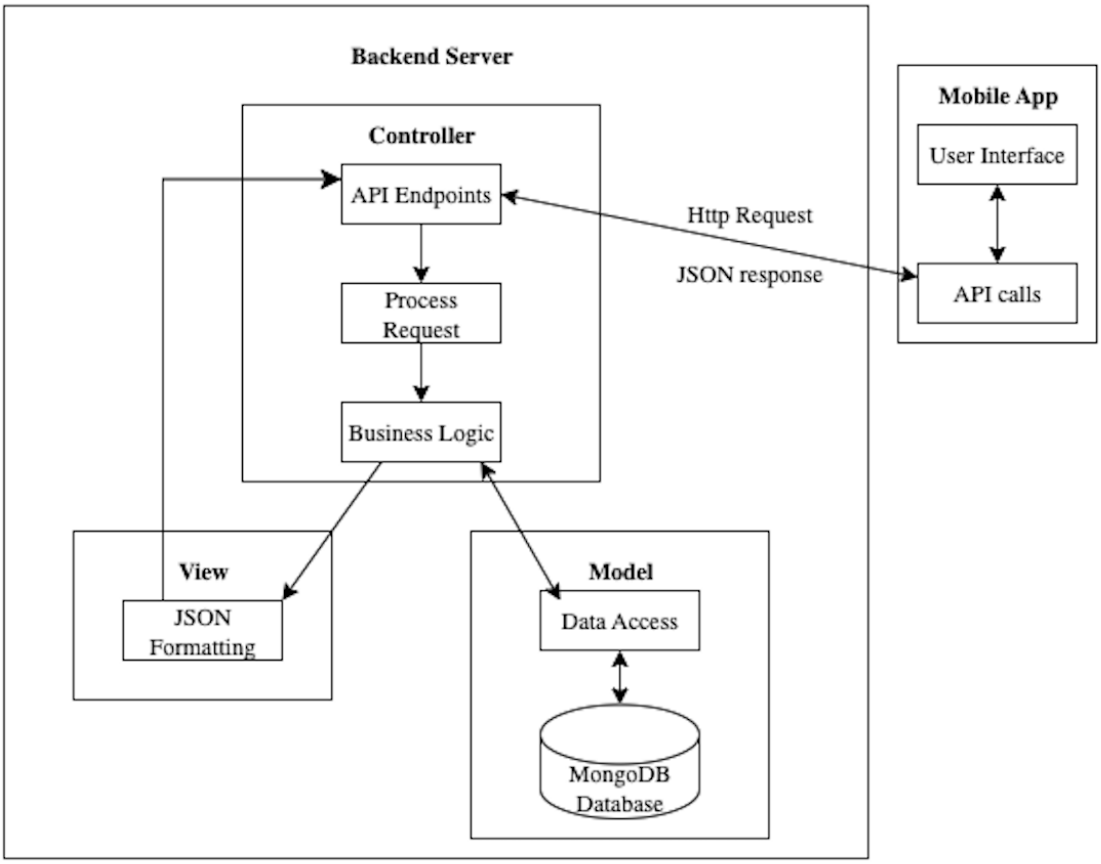

Figure 4. MVC pattern

4.4 Database Design

To store all application data, we designed a database using MongoDB. I chose this NoSQL database for its flexible, document-based model, which easily handles the application's varied data, such as user profiles and schedules. The database is organized into several main "collections," which are groups of JSON-like documents. The schema for these collections is illustrated in the diagram below.

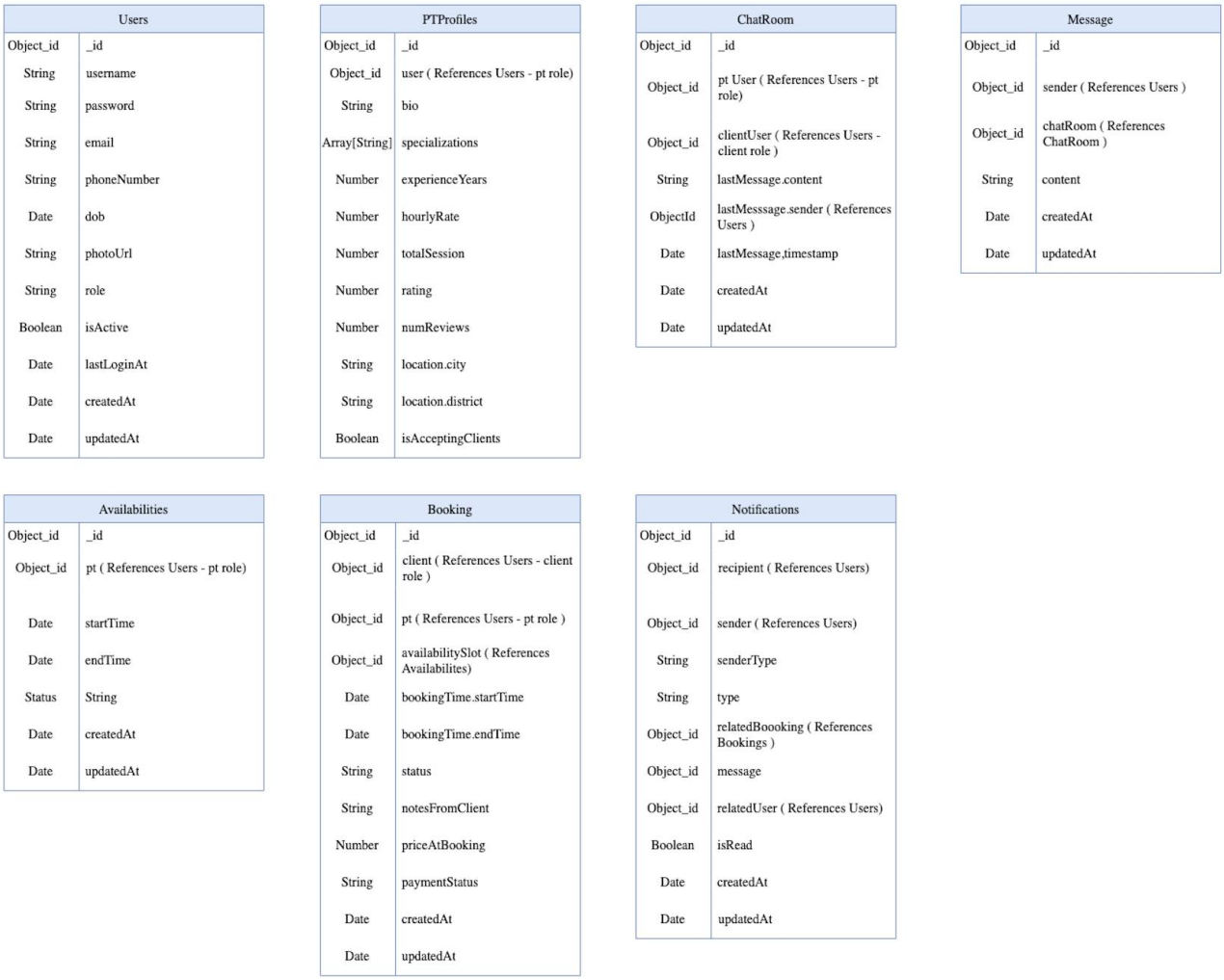

Figure 5. Overall Database Schema Design

4.4.1 Users collection:

Table 12. Users Collection

Field	**Data type**	**Description**
_id	Object_id	Unique identifier for user
username	String	User's login name
password	String	User's encrypted password
email	String	User's email address
phoneNumber	String	User's contact phone number
dob	Date	User's date of birth
photoUrl	String	URL to user's profile picture

Field	Data type	Description
createdAt	Date	Timestamp when user account was created
updatedAt	Date	Timestamp when user account was last updated

4.4.2: PT Profiles Collection

Table 13. PT Profiles Collection

Field	Data type	Description
_id	Object_id	Unique identifier for the PT profile document
user	Object_id	Reference to the user in Users collection with PT role
specializations	Array[String]	List of trainer's specialization
experienceYears	Number	Years of experience
hourlyRate	Number	Trainer's hourly price
totalSession	Number	Total number of sessions
rating	Number	Average rating score of session conducted
numRivews	Number	Number of reviews received
location.city	String	City where trainer work
location.district	String	District where trainer work

4.4.3 Availabilities Collection

Table 14. Availabilities Collection

Field	Data type	Description
_id	Object_id	Unique identifier for the availabilities
pt	Object_id	Reference to the PT in Users collection
startTime	Date	Start time of availability slot

Field	Data type	Description
endTime	Date	End time of availability slot
status	String	Status of the availability
createdAt	Date	Timestamp when the availability was created
updatedAt	Date	Timestamp when availability was updated

4.4.4 Booking Collection

Table 15. Booking Collection

Field	Data type	Description
_id	Object_id	Unique identifier for the booking
client	Object_id	Reference to the client in Users collection
pt	Object_id	Reference to the PT in Users collection
availabilitySlot	Object_id	Reference to the availability entry
bookingTime.startime	Date	Start time of booking
bookingTime.endtime	Date	End time of booking
status	String	Status of booking
priceAtBooking	Number	Price set at the time of booking
paymentStatus	String	Status of payment
createdAt	Date	Timestamp when booking was created
updatedAt	Date	Timestamp when booking was updated

4.4.5 Notification Collection

Table 16. Notifications Collection

Field	Data type	Description
_id	Object_id	Unique identifier for the notification
recipient	Object_id	Reference to the user receiving notification
sender	Object_id	Reference to the user who triggered notification
senderType	String	Type of sender (system, user)
type	String	Type of notification (booking, message)
relatedBooking	Object_id	Reference to related booking
message	Object_id	Reference to related message
relatedUser	Object_id	Reference to related user if applicable
isRead	Boolean	Indicates if notification has been read
createdAt	Date	Timestamp when notification was created
updatedAt	Date	Timestamp when notification was updated

4.4.6 ChatRoom Collection

Table 17. ChatRoom collection

Field	Data type	Description
_id	Object_id	Unique identifier for the chat room
pt	Object_id	Reference to the PT user in Users collection
clientUser	Object_id	Reference to Client user in Users
lastMessage.content	String	Content of the last message in chat
lastMessage.sender	Object_id	Reference to the sender last message

Field	Data type	Description
lastMessage.timestamp	Date	Timestamp of the last message
createdAt	Date	Timestamp when the chat room was created
updatedAt	Date	Timestamp when the chat room was last updated

4.4.7 Message Collection

Table 18. Message Collection

Field	Data type	Description
_id	Object_id	Unique identifier for the message
sender	Object_id	Reference to the sender in Users collection
chatRoom	Object_id	Reference to the associated ChatRoom
content	String	Content of the message
createdAt	Date	Timestamp when message was created
updatedAt	Date	Timestamp when message was updated

Description of Key Collections:

- **users collection**: This is the central collection for storing information about all users, whether they are a Client or a Personal Trainer (PT). It contains essential data for authentication (email, hashed password) and basic profile information. A role field is used to distinguish between the two types of users.
- **pt_profiles collection**: To keep the users collection clean and efficient, I created a separate collection to store information specific only to trainers. This collection is linked to the users collection via a **userId** reference. It holds professional details such as specializations, certifications, hourly rates, and a professional bio. This design allows me to easily expand the trainer's profile in the future without affecting the core user data.
- **availability collection:** This collection is dedicated to managing a trainer's work schedule. Each document represents a block of time that a trainer has marked as

"available" for booking. This separation makes it very efficient to query for a trainer's free time slots without having to search through all their bookings.

- **bookings collection:** This is a main collection that records every session booked through the application. Each booking document acts as a link between a client and a trainer for a specific time. It contains references to the **clientId** and **ptId** (from the users collection), the session time, the price, and the current status of the booking (e.g., pending, confirmed, completed).
- **notifications collection:** These collections are designed to handle specific functionalities. Notifications stores a record of all automated messages sent within the system, such as booking confirmations or chat alerts.

4.5 API Design

The API (Application Programming Interface) acts as the communication bridge between the mobile app (client) and the server. To ensure this communication is structured, predictable, and secure, we designed a RESTful API. This architectural style uses standard HTTP methods to perform operations on data resources. The main API endpoints designed for the application, each endpoint represents a specific action that the client can request from the server.

Authentication endpoint:

Base URL: /auth

Table 19. Authentication API endpoint

Method	**Endpoint**	**Description**
POST	/register	User registration with email verification
POST	/login	User authentication
POST	/verification	Email verification
POST	/forgotPassword	Password reset request

Client endpoint:

Base URL: /api/client

Table 20. Client API endpoint

Method	**Endpoint**	**Description**
GET	/pt/:ptId/profile	View PT detailed profile.
GET	/pt/:ptId/profile	Get PT availability slots

POST	/booking	Create booking request
GET	/my-bookings	Get client’s booking
POST	/my-bookings/:bookingId/ca-ncel	Cancel booking
GET	/pt	Search PT

Personal Trainer endpoint:

Base URL: api/pt

Table 21. Personal Trainer API endpoint

Method	**Endpoint**	**Description**
PUT	/profile	Update PT profile
DELETE	/profile	Delete PT profile
POST	/availability	Add availability slots
PUT	/availability/:id	Update availability slots
DELETE	/availability/:id	Delete availability slots
PUT	/bookings/:id/confirm	Confirm booking
PUT	/bookings/:id/Reject	Reject booking
PUT	/bookings/:id/complete	Mark booking as completed

4.6 Use Case Implementation

This section provides a deeper look into how the most important features of the system were implemented. I will use the sequence diagrams to help visualize the process.

4.6.1 User Authentication

The user authentication workflow is the foundation of the application's security. The implementation follows the steps detailed in the User Registration and User Login Sequence Diagram.

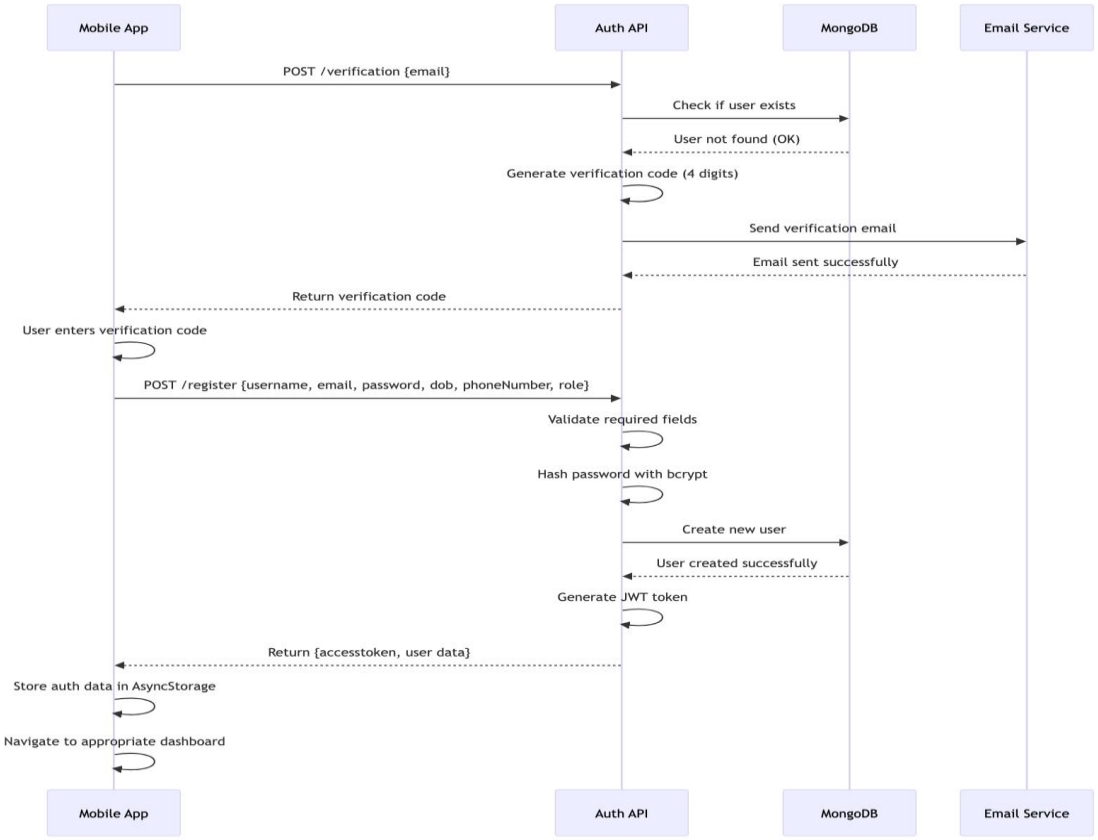

Figure 6. Registration Sequence Diagram

Implementation Logic:

1. **(Client):** The process starts when a user fills out the registration form in my application.
2. **(Client to Server):** the app sends a POST request with the user's data to the /auth/verification endpoint on the Node.js server.
3. **(Server):** The server checks if the email already exists in the database. If it is a new email, the server generates a random 4-digit verification code.
4. **(Server to Client):** The server sends the verification code to the user's email address and sends a success message back to the app. The app then shows the verification screen.
5. **(Client to Server):** The user enters the code, and the app sends a final POST request to /auth/register with all the original data plus the code.
6. **(Server):** The server checks if the code is correct. If it is, it hashes the user's password using bcrypt and saves the new user to the MongoDB database with a "verified" status.

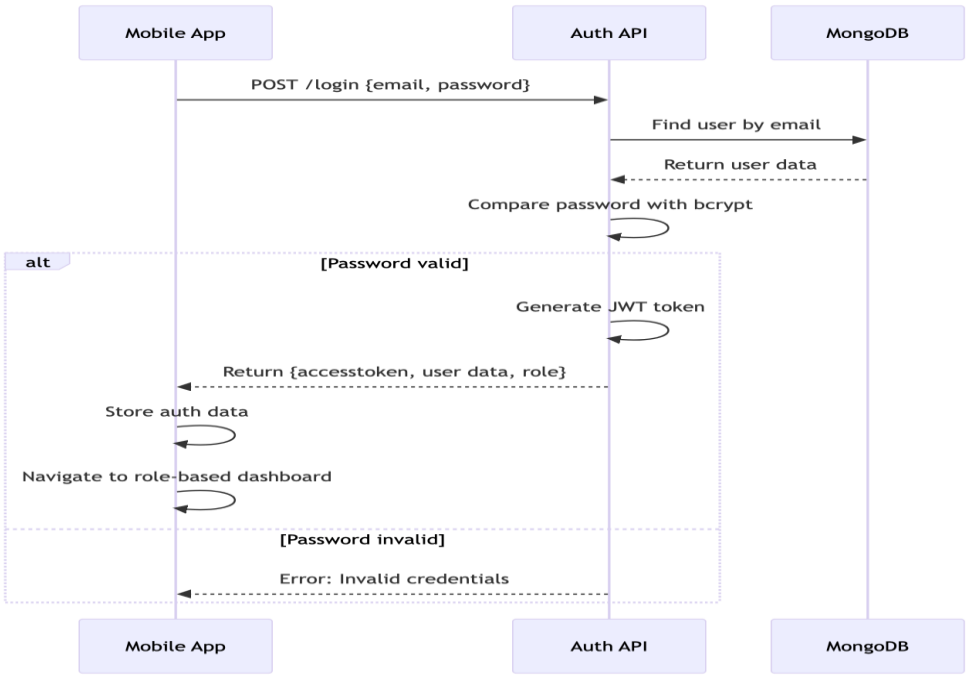

Figure 7. Login Sequence Diagram

Implementation Logic

1. **(Client):** A user enters their email and password on the Login Screen.
2. **(Client to Server):** The app sends a POST request to the /auth/login endpoint.
3. **(Server):** The server finds the user in the database by their email.
4. **(Server):** It then uses bcrypt.compare() to check if the password submitted by the user matches the secure, hashed password stored in the database.
5. **(Server to Client):** If the passwords match, the server creates a JSON Web Token (JWT). This token is a special, secure key. The server sends this token back to the app.
6. **(Client)**: The React Native app saves this token. For all future API requests, the app will send this token along to prove that the user is logged in.

4.6.2 Personal Trainer Profile and Schedule Management

This section describes how a Personal Trainer (PT) sets up their professional profile and manages their availability.

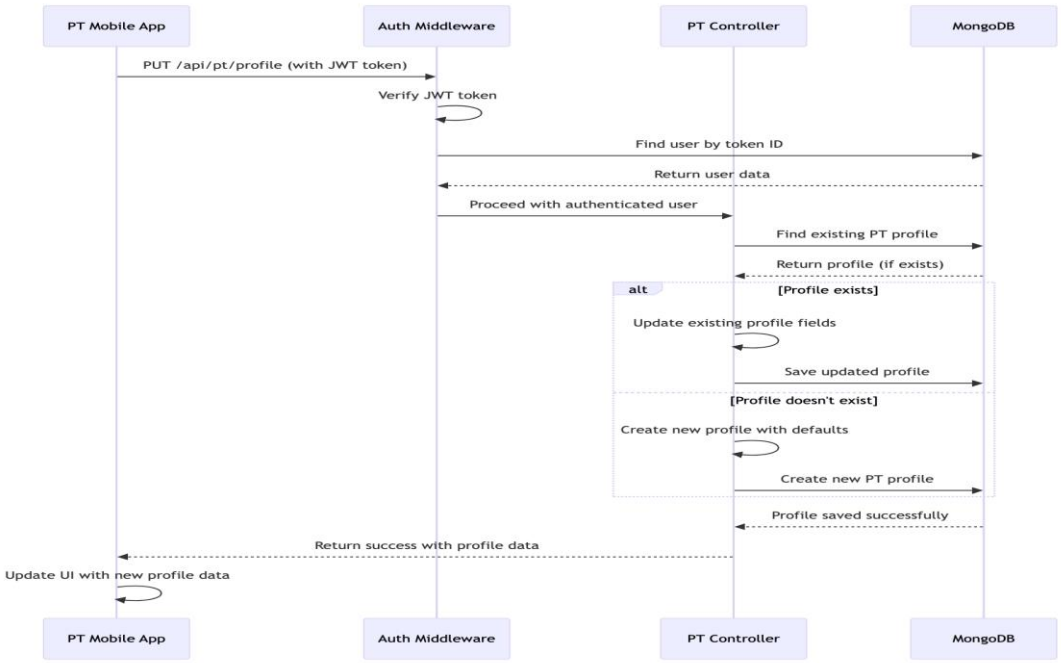

Figure 8. Sequence Diagram for PT Profile Creation

Implementation Logic

- **(Client):** When a logged-in PT edits their profile information (like their bio, skills, price and specialization) and taps the "Save" button, the React Native app sends a PUT request to an API endpoint /api/pt/profile.
- **(Server):** The authMiddleware first checks the user's token to make sure they are a logged-in PT. The profileController then receives the new profile data, validates it, and updates the correct document in the pt_profiles collection in the database.

PT Availability Management

This workflow allows a PT to show clients when they are free to be booked.

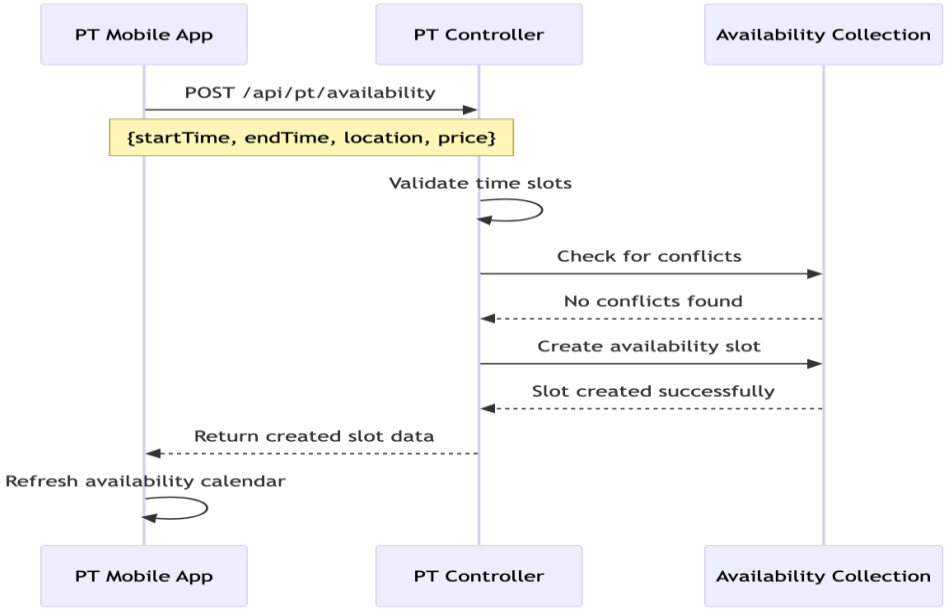

Figure 9. PT add their available sequence diagram

Implementation Logic:

- **(Client):** The PT clicks to the "availability screen" and uses a form in the app to choose start-time and end-time, and click add available.
- **(Server):** When the PT saves their schedule, the app sends a POST request to /api/pt/availability. The backend controller receives this list of available times and saves it to the availability collection in MongoDB, linked to the PT's ID. This new schedule is then immediately available for clients to see when they view the trainer's profile

4.6.3 The complete Booking Process Flow

This is the core business workflow of the application, involving a two-way interaction between the Client and the PT.

Client Creates a Booking Request

This sequence shows how a client requests a training session with a personal trainer.

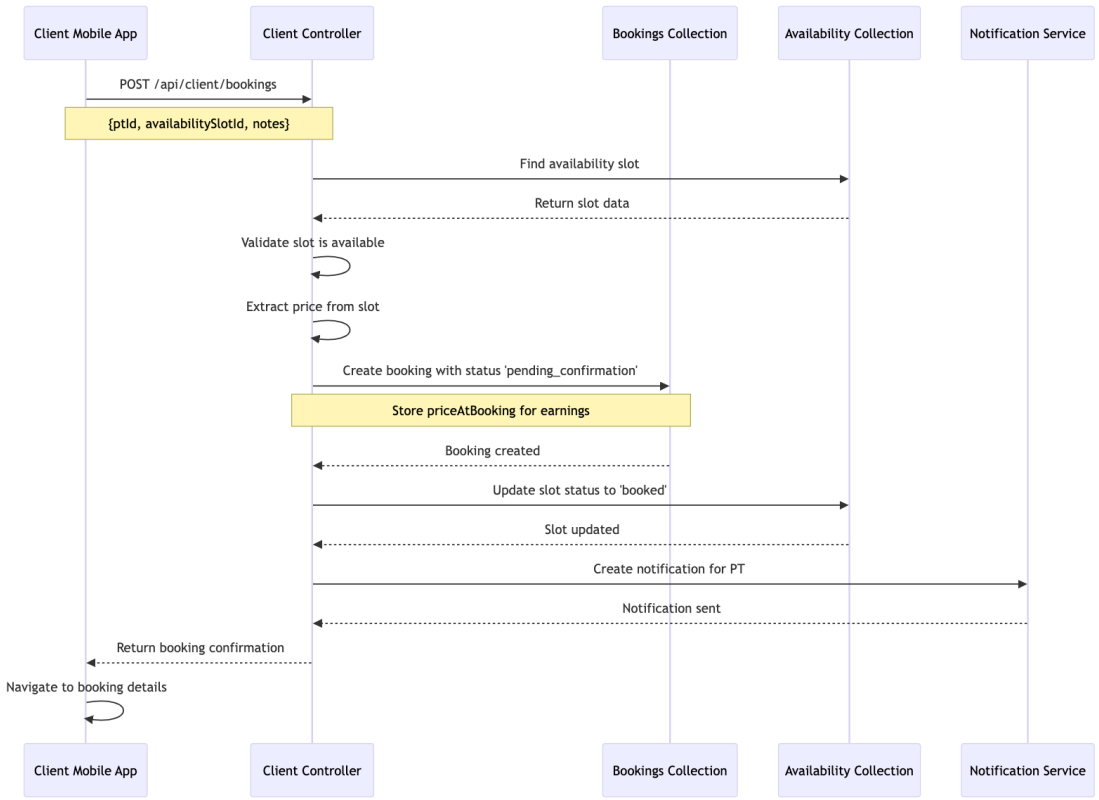

Figure 10. Client Creates a Booking Request Sequence Diagram

Implementation Logic

(Client - Client App) sends a POST request to the /api/client/bookingswith the details of the booking (like the ptID and availability SlotId, notes).
(Server) validates the request to make sure the time slot is still available and there are no conflicts. If everything is okay, the server creates a new booking in the database with a "pending" status. Finally, the server sends a success message back to the client and sends a notification to the trainer about the new request.

PT Manages the Booking Request

This sequence shows how a Personal Trainer can manage an incoming booking request by confirming or rejecting it.

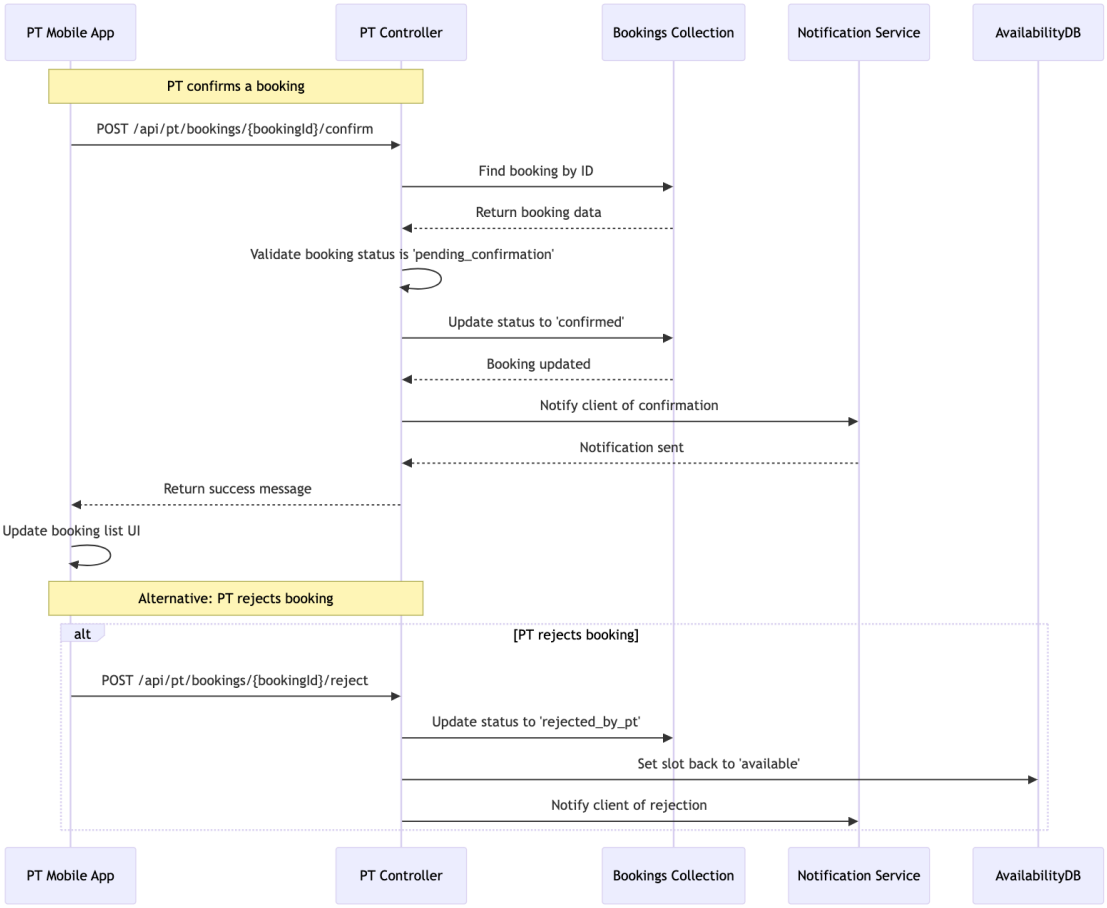

Figure 11. Figure 4.6: PT Manages the Booking Request Diagram

Implementation Logic

(Client - PT app) The trainer views a pending booking request.When the trainer taps "Confirm" or "Reject", the app sends a PUT request to the appropriate server endpoint (e.g., /api/pt/bookings/:bookingId/confirm).

(Server) The server receives the request and validates that the user is the correct PT for this booking. If the booking is confirmed: The server updates the booking's status in the database to "confirmed". It also updates the availability slot to "booked" so no one else can book it. If the booking is rejected: The server updates the booking's status to "rejected". It then changes the availability slot's status back to "available" so it can be booked by other clients. In both cases, the server sends a notification to the client to inform them of the decision.

CHAPTER V: TESTING AND RESULT

5.1 Introduction

This chapter is all about testing the app I built. The main goal was simple: to make sure that everything works correctly and that there are no big problems or bugs. Here, I will explain how I tested the app. Then, I will show you the results for the most important features, with pictures from the app to prove it. Finally, I will share my overall thoughts on how well the final application works.

5.2 How I tested the app

I did not use any automatic testing software. Instead, the main method was Manual Testing. This means I tested the app myself by using it like a real person would. I pretended to be a Client looking for a trainer, and then I pretended to be a Personal Trainer (PT) managing their bookings. I went through all the main steps from start to finish. I did my testing on both an Android and IOS simulator.

5.2.1 Testing the Login and Register Feature

First, I needed to make sure that people could sign up and log in safely.

- **Registration:** I tried creating a new account for a "Client" and another one for a "PT". The process worked perfectly. When I tried to use an email that was already registered, the app correctly showed an error message. After signing up, the app took me to the new email verification screen, just like I planned.
- **Login:** I tested logging in with a correct password, and it worked – it took me to the home screen. Then, I tried logging in with a wrong password. The app correctly showed an error message "Invalid email or password" and did not let me in.

This shows that the basic user authentication is working well and is secure.

Table 22. User testing account

Role	**Email**	**Password**	**username**
Client	mhtu1213@gmail.com	Lol123040602	mhtu202
Personal Trainer	tudm.ba11-094@st.usth.edu.vn	Lol123040602	dotu123

Results: The user authentication system performed exactly as designed. New users were able to register, and the system correctly validated their information. Existing users could log in

securely with their credentials. the system correctly navigates the user to the home screen after a successful login

5.2.2 Testing the Booking Feature (From the Client's View)

Next, I tested the most important feature for a client: finding and booking a trainer.

- **Searching:** The search screen displayed all the trainers correctly. I was able to use the filters to find trainers based on their skills, and the results were accurate.
- **Booking a Session:** As a Client, I could tap on a trainer to see their profile and schedule. I selected a free time slot and pressed the "Book" button. After that, the new booking immediately appeared in my "My Bookings" list with a "pending" status. This was the correct behavior.

- **Results:** The booking flow for the client was successful. The application correctly displayed the trainer's schedule. After a booking was made, the request immediately appeared in the client's "My Bookings" list with the correct "pending" status, waiting for the trainer's approval. This is demonstrated in **Figure 5.1**

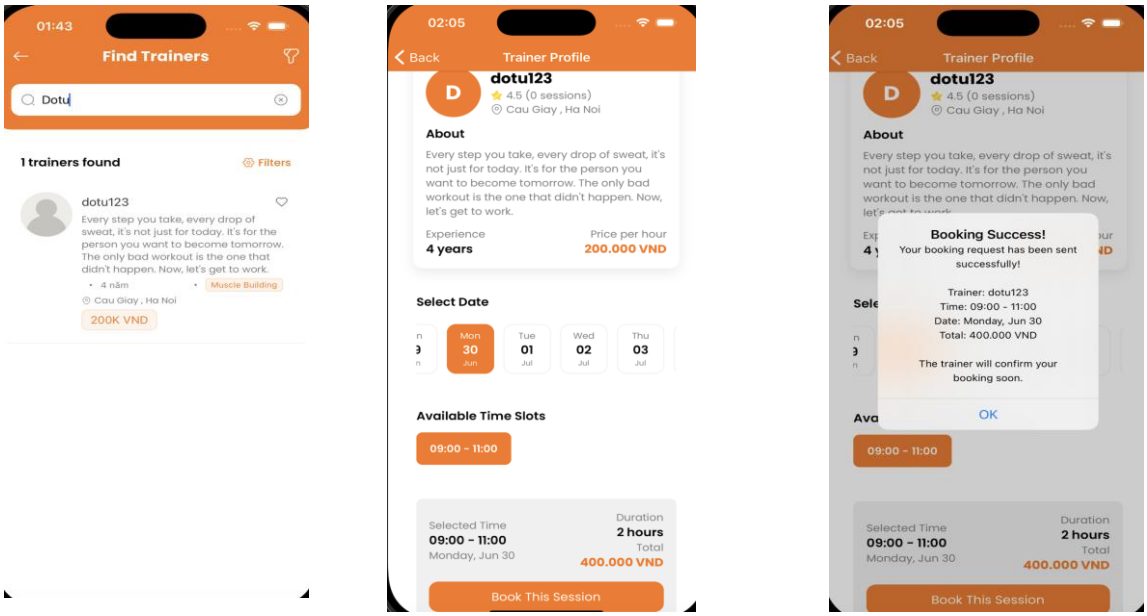

Figure 12. Client search and create the booking

5.2.3 Testing Booking Management (From the Trainer's View)

Finally, I needed to make sure the trainer could manage the requests from clients.

- **Receiving a Request:** After the Client made a booking in the previous step, I logged in with the PT's account. I saw a notification for the new booking request in the "Booking Requests" screen.
- **Accepting a Request:** I tapped on the request to see the details and then pressed the "Accept" button. The system worked perfectly. The booking status changed to

"confirmed," and that time slot was no longer available for other people to book. The Client's app also showed the updated "confirmed" status.

Results: The trainer's booking management system also worked well. The new request was visible on the PT's booking screen. Upon clicking the "Confirm" button, the booking's status was updated to "confirmed". I logged back in as the client and verified that the booking status had also changed to "confirmed" on their screen. This successful two-way interaction, shown in **Figure 5.2**, confirms that the entire booking lifecycle is functional.

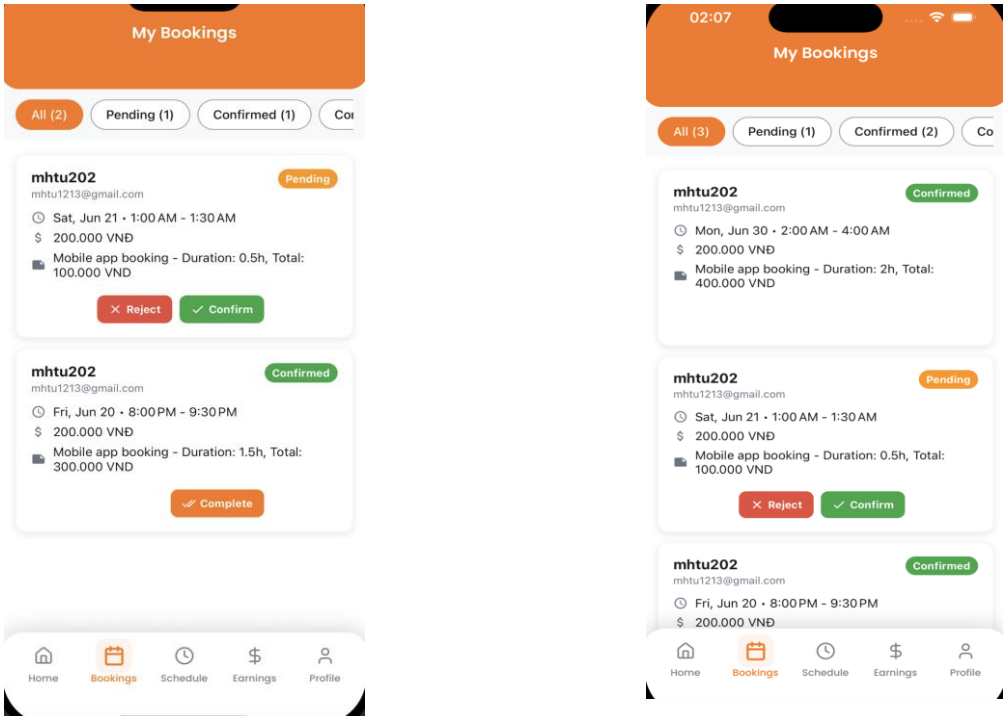

Figure 13. Booking management screen PT role

CHAPTER VI: CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis has presented the entire journey of building a mobile application, from the first idea to the final working product. After all the planning, designing, coding and testing, I can say that the project has completed the main functions set out, however, there is still a lot that can be improved.

The main goal was to create a mobile application to help clients easily connect with personal trainers. I am happy to report that this goal has been achieved. The final result is a working cross-platform application that runs on both Android and iOS. It has all the core features that we planned, such as user registration, a search function, and a complete booking system.

In short, the project successfully turned an idea into a real, functional application that solves the problem it was designed for.

6.2 Limitations

Even though the project was a success, it is important to be honest about its limitations. Because this was a student project with limited time, there are some features that I could not build.

- **No real-time booking:** This is the biggest limitation, the user has to refresh the application for updating information of the booking session.
- **No Online Payments:** The big limitation is that there is no payment system in the app. Clients and trainers have to handle payments in person.
- **Simple User Interface:** The app's design is clean and easy to use, but it is still very basic. It could be made to look more modern and beautiful.
- **Limited Testing:** I only tested the app on a few devices. There might be small bugs on different screen sizes or older phones.

6.3 Future work

This project has a lot of potential to grow. If I had more time, here are the first things I would add to make the app even better:

1. **Real-time booking:** I really need to improve this feature, for improving user experience, I will focus on this feature first.
2. **Add a Payment System:** The number one priority would be to add online payments using popular Vietnamese services like MoMo or VNPay. This would complete the entire booking experience inside the app.
3. **Improve the Chat:** I would upgrade the chat feature so that users can send photos and maybe even voice messages. This would make communication much better.
4. **Create a Review System**: Review is one of the most important things, to help users find a good Personal trainer, but for now, it's not really good in my application.
5. **Subscription:** I will develop these features when my application has an amount of users.

REFERENCES

1. *Auth0. (n.d.). JSON Web Tokens from https://jwt.io/*
2. *Express.js Team. (n.d.). Express.js - Node.js web application framework from https://expressjs.com/*
3. *Meta Platforms, Inc. (n.d.). React Native Documentation from https://reactnative.dev/docs/getting-started*
4. *MongoDB, Inc. (n.d.). MongoDB Documentation from https://docs.mongodb.com/*
5. *Mongoose Team. (n.d.). Mongoose ODM v8.4.1 documentation from https://mongoosejs.com/docs/guide.html*
6. *Mozilla Developer Network. (n.d.). Client-Server overview. MDN Web Docs. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview*
7. *freeCodeCamp. (2021, May 26). The Model-View-Controller (MVC) Pattern – MVC Architecture and Frameworks Explained from https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/*
8. *npm, Inc. (n.d.). bcryptjs - npm. Retrieved from https://www.npmjs.com/package/bcryptjs*
9. *OpenJS Foundation. (n.d.). Node.js Documentation. Retrieved from https://nodejs.org/en/docs/*
10. *Postman, Inc. (n.d.). Postman API Platform. Retrieved from https://www.postman.com/*
11. *Xu, A. (2020). System Design Interview – An Insider's Guide.*

APPENDICES

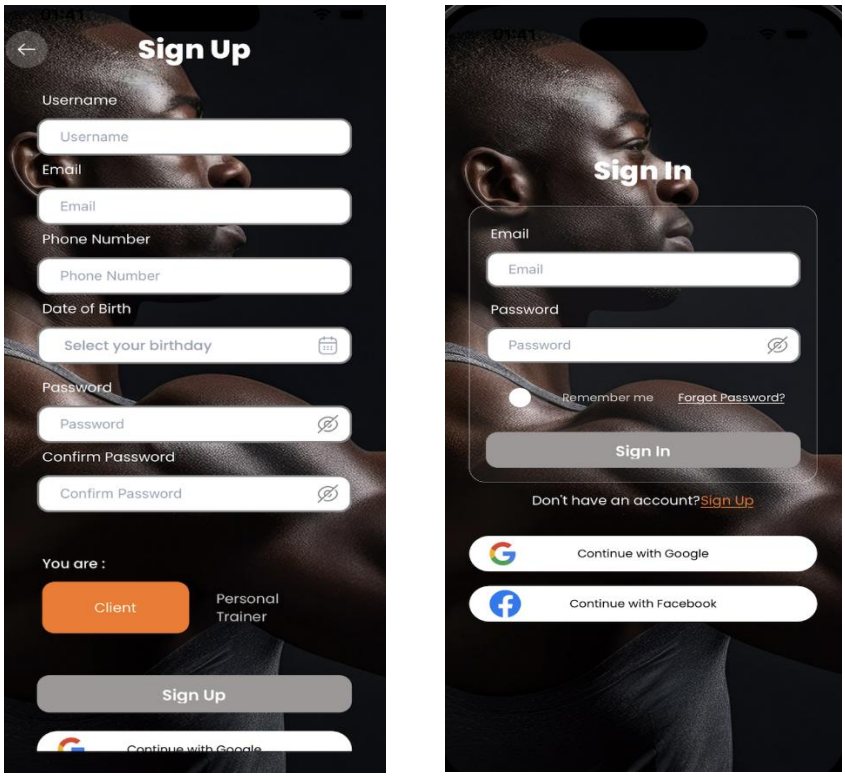

Figure 14. SignIn and SignUp Screen

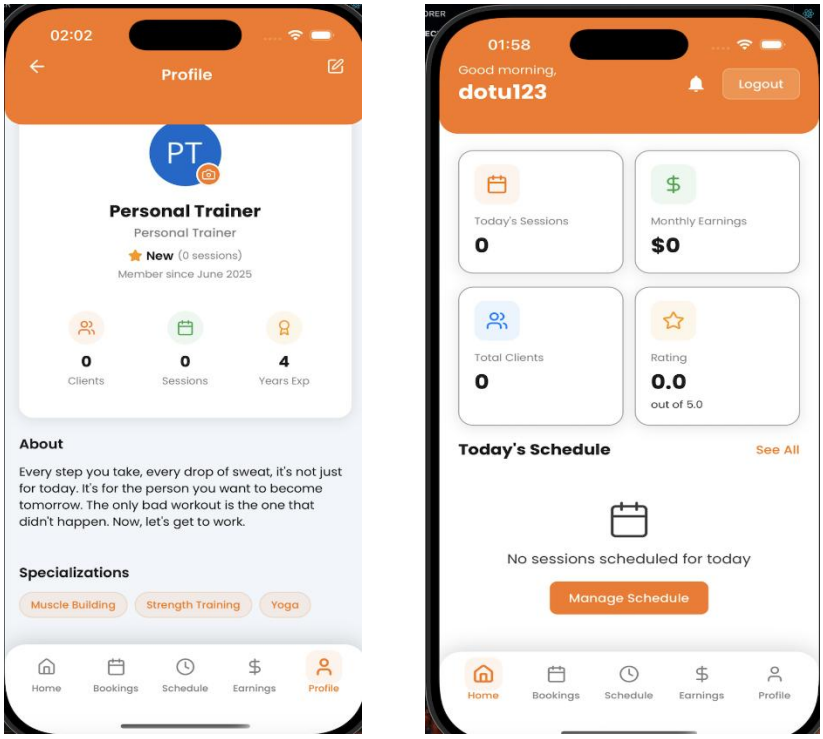

Figure 15. Personal Trainer HomeScreen and work profile

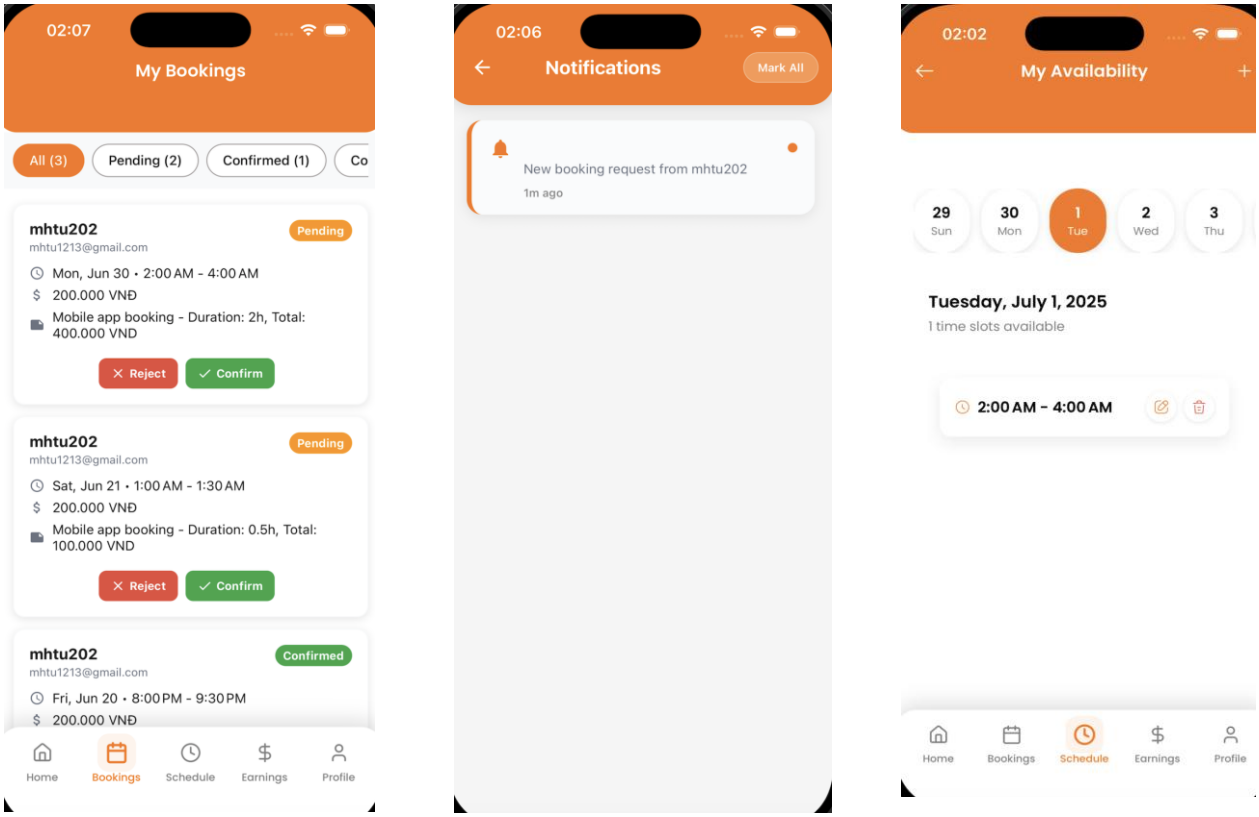

Figure 16. Personal Trainer Booking Screen, Notification and Availability Screen

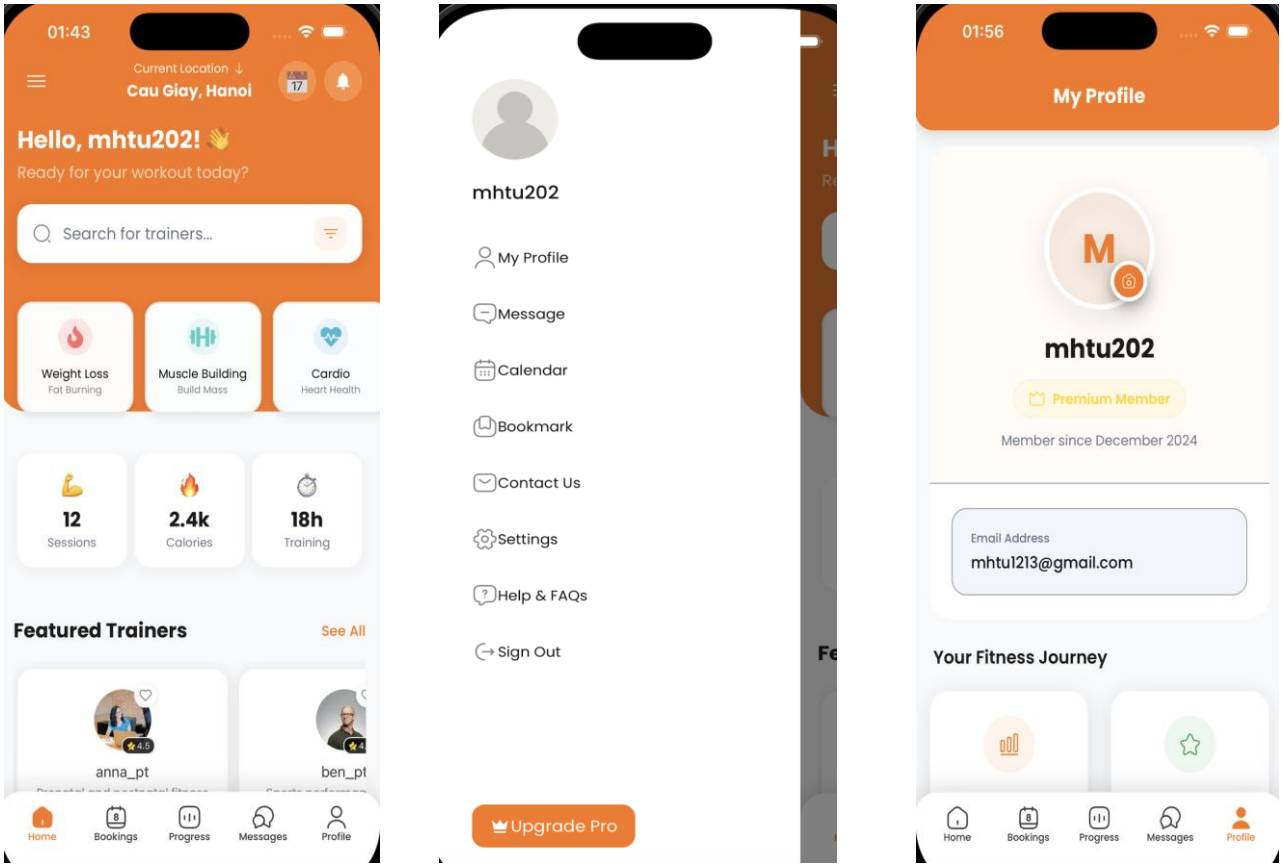

Figure 17. Client Home Screen, Drawer menu and Profile Screen

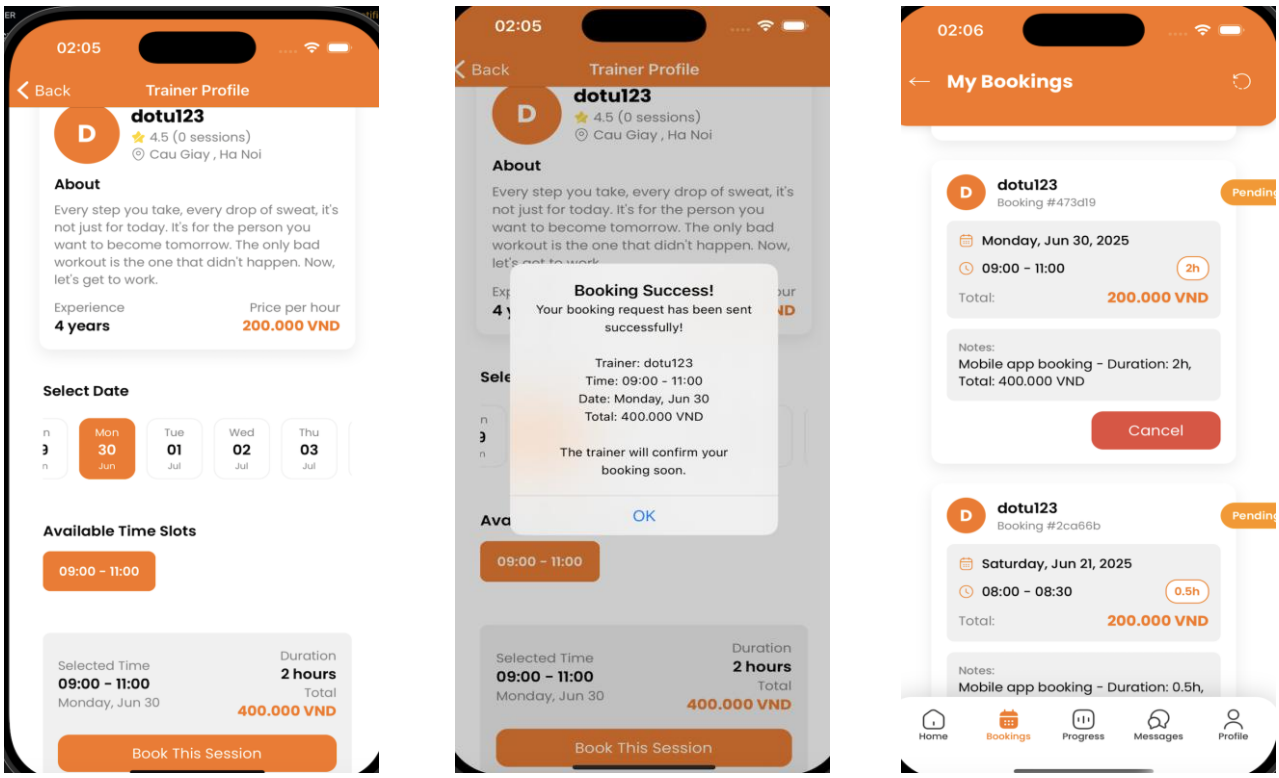

Figure 18. Trainer Profile, Booking Success and Booking Status Screen