

HairHaven BarberShop

*Software Engineering
Software Requirements Specification
(SRS) Document*

[04/29/2024]

[Version 1.0]

By:

Team Member:

Muhammad Asad Khan

Kevin Ornelas

Romario Barahona

[Honor Code]

Table of Contents

1. Introduction	5
1.1. Purpose	5
1.2. Document Conventions	5
1.3. Definitions, Acronyms, and Abbreviations	5
1.4. Intended Audience	6
1.5. Project Scope	6
1.6. Technology Challenges	6
1.7. References	6
2. Project Requirements Analysis	7
2.1. Functional	7
2.2. Usability	7
2.3. User Interface (Diagram and Description)	8
2.3.1. Sign Up Interface -	8
2.3.2. Sign In Interface -	8
2.3.3. Admin HomePage Interface	9
2.3.4. Admin create a barber profile Interface	9
2.3.5. Barber Homepage Interface -	10
2.3.6. Barber Profile Detail Interface -	10
2.3.7. Barber Add Services Interface -	11
2.3.8. Barber Booking Page Interface -	11
2.3.9. User HomePage Interface -	12
2.3.10. User Profile Interface -	12
2.3.11. User Edit Profile Interface -	13
2.3.12. User Viewing Barbers Profile Interface -	13
2.3.13. User booking an appointment Interface	14
2.3.14. User Booking Page Interface	14
2.4. Performance	15
2.5. System- System Architecture and Data Flow	15
2.6. Hardware	15
2.7. Software	15
2.8. Database	15
2.9. Security and Authentication	15
3. Project Specification	15
3.1. Focus/ Domain/ Area	16
3.2. Libraries/ Frameworks/ Development Environment	16
3.3. Platform (Mobile, Desktop, Gaming, Etc.)	18
3.4. Genre	18
3.5. Teamwork division	18

4.	System - Design Perspective	19
4.1.	Identify Sub-Systems Design:	19
4.2.	Sub-System Communication (Diagram and Description)	19
4.2.1.	Controls	19
4.2.2.	I/O	19
4.2.3.	DataFlow	19
4.3.	Entity Relationship Model (ER Model)	19
4.4.	Database Schema	20
4.5.	Overall Operation - System Model	20
4.5.1.	Big-O analysis of overall system and subsystem	20
5.	System - Analysis Perspective	20
5.1.	User Management Sub-System- Analysis Perspective (Muhammad Khan)	20
5.2.	System (Tables and Description)	20
5.2.1.	Data analysis	20
5.2.2.	Data Dictionary (Table - Name, Data Type, Description)	20
5.2.3.	Process Models	21
5.3.	Algorithm Analysis	21
5.3.1.	Simplified Sub-System to System interaction.	21
5.3.2.	Implementation Details	21
5.4.	Booking Sub-System - Analysis Perspective (Muhammad Khan)	22
5.5.	System (Tables and Description)	22
5.5.1.	Data analysis	22
5.5.2.	Data Dictionary (Table - Name, Data Type, Description)	22
5.5.3.	Process Models	22
5.6.	Algorithm Analysis	23
5.6.1.	Simplified Sub-System to System interaction.	23
5.6.2.	Implementation Details	23
5.7.	User Authentication Sub-Systems - Analysis Perspective (Muhammad Khan)	23
5.8.	System (Tables and Description)	23
5.8.1.	Data analysis	23
5.8.2.	Data Dictionary (Table - Name, Data Type, Description)	23
5.8.3.	Process Models	24
5.8.4.	Simplified Sub-System to System interaction.	24
5.8.5.	Implementation Details	24
5.9.	Service Management Sub-System - Analysis Perspective (Kevin)	25
5.10.	System (Tables and Description)	25
5.10.1.	Data analysis	25
5.10.2.	Data Dictionary (Table - Name, Data Type, Description)	25
5.10.3.	Process Models	25
5.10.4.	Simplified Sub-System to System interaction.	25

5.10.5.	Implementation Details	26
5.11.	Barber Management Sub-System - Analysis Perspective (Romario)	26
5.12.	System (Tables and Description)	26
5.12.1.	Data analysis	26
5.12.2.	Data Dictionary (Table - Name, Data Type, Description)	26
5.12.3.	Process Models	26
5.12.4.	Simplified Sub-System to System interaction.	27
5.12.5.	Implementation Details	27
6.	Project Scrum Report	28
6.1.	Product Backlog	28
7.	Sub-Systems	30
7.1.1.	Initial Design and Model	30
7.1.2.	Data Dictionary	31
	User:	31
	• id (INT, PK)	31
	• firstName (VARCHAR)	31
	• lastName (VARCHAR)	31
	• email (VARCHAR)	31
	• password(VARCHAR)	31
	• phone(VARCHAR)	31
	• location (VARCHAR)	31
	• profilePic (VARCHAR)	31
	• availableHours(TEXT)	31
7.1.3.	Refined model and design	31
	• Reason for Refinement: To improve security and usability.	31
	• Changes from Initial Model: Added two-factor authentication for enhanced security.	31
	• Refined Model Analysis: Ensured that user data is more secure while maintaining ease of access.	31
	• Refined Design: Updated UML diagrams to reflect new authentication flow.	31
7.1.4.	Scrum Backlog (Product and Sprint- link to section)	31
7.1.5.	Coding - approach(functional, OOP) and Language	31
	• Approach: Object-Oriented Programming (OOP)	31
	• Language: TypeScript with NestJS	31
7.1.6.	User Training - training/user manual	31
	• simple instructions on managing user roles and secure login procedures.	31
7.1.7.	Testing	31
	• Automated tests for login, registration, and profile updates.	31
	• Security tests to validate new authentication measures.	31
7.1.8.	Initial Design and Model	32

7.1.9.	Data Dictionary	32
	booking_service_services:	32
•	bookingID (INT, PK)	32
•	serviceID (INT, FK)	32
7.1.10.	Refined model and design	32
7.1.11.	Scrum Backlog (Product and Sprint- link to section)	32
•	Refer to section 6	32
7.1.12.	Coding - approach(funtional, OOP) and Language	32
7.1.13.	User Training - training/user manual	32
•	Guidelines on managing and optimizing appointment schedules.	32
7.1.14.	Testing	32
•	Load testing to ensure system handles high traffic.	32
•	Functional testing for all scheduling features.	32
8.	Complete System	33
8.1.	Final Software/Hardware Product	33
8.2.	Source code and demonstration video- screenshots - technical report githublink	33
8.3.	Evaluation by client and instructor	33
8.4.	Team Member Descriptions	33

1. Introduction

1.1. Purpose

- The goal of our project is to develop a comprehensive barbershop scheduling application that facilitates efficient management of appointments, staff schedules, and customer interactions. The app aims to streamline the booking process, reduce waiting times, and improve the overall customer experience, while providing barbershop owners with valuable insights into their business operations.

1.2. Document Conventions

- The purpose of this Software Requirements Document (SRD) is to provide a detailed framework for the development of the Barbershop Scheduling App. In it, we will outline the application's specifications, detail the functional and non-functional requirements, and provide a roadmap for the development process using agile methodology.

1.3. Definitions, Acronyms, and Abbreviations

Next.js	A React framework capable of server-side rendering to enhance performance and SEO, which will be used to build the front end of our Barbershop Scheduling App.
Nest.js	A progressive Node.js framework for building efficient and scalable server-side applications, selected for our back-end development.
TypeORM	An Object-Relational Mapping library that can be used with TypeScript (or JavaScript) to interact with various databases in an object-oriented manner.
React	A JavaScript library for building user interfaces, especially dynamic single-page applications, used in conjunction with Next.js.
Redux	A predictable state management library for JavaScript apps, helping to manage the state across all components of our application.
MySQL	An open-source relational database management system, used to store and manage the application's data persistently.
API	Application Programming Interface, a set of rules that allow different software entities to communicate with each other. In our app, APIs are used to connect the front end and the back end.
JWT (JSON Web Tokens)	A compact, URL-safe means of representing claims to be transferred between two parties, allowing us to implement secure authentication in our app.
TypeScript	Application Programming Interface. This will be used to implement a function within the software where the current date and time is displayed on the homepage.

ESLint	A static code analysis tool for identifying problematic patterns found in JavaScript code, helping to ensure code quality and consistency across the team.
Prettier	An opinionated code formatter that enforces a consistent style by parsing code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.

1.4. Intended Audience

- This document is intended for the development team, barbers, stakeholders, and the end users which include both barbers and clients.

1.5. Project Scope

- The software goals align with the business goals of enhancing operational efficiency and improving customer satisfaction in barbershops. The project will deliver a system that simplifies appointment management and promotes better business for our clients.

1.6. Technology Challenges

- Integration Complexity: Ensuring seamless integration of new technologies with existing systems.
- Scalability: Designing the backend to handle a growing amount of user data and interactions without performance lag.
- Data Security: Implementing robust security measures to protect sensitive user data and prevent unauthorized access.

1.7. References

- [Docs | Next.js \(nextjs.org\)](https://nextjs.org/docs)
- [Documentation | NestJS - A progressive Node.js framework](#)
- [MySQL :: MySQL Documentation](#)

2. Project Requirements Analysis

2.1. Functional

- The product will enable users to create and manage bookings with their preferred barber and also view their booking history. It allows barbers to check their bookings, set their schedules, and add their services with prices.

2.2. Usability

- Barbershop system has three different types of profiles: User, Barbers, and Administrator.

User Dashboard:

- Reservations: Users can effortlessly book appointments with available barbers, choosing from a variety of services during preferred shifts.
- Profile Management: Users have the ability to view and edit their profiles, ensuring their personal information is always up-to-date.
- Service and Barber Selection: A comprehensive view allows users to browse barbers and their specific services, facilitating informed booking decisions.
- Session Management: Users can securely sign out of their accounts at any convenience, maintaining their privacy and security.

Barber Dashboard:


- Service Display: The homepage prominently displays all the services offered by the barber, providing clear options to clients.
- Profile Customization: Barbers can update and manage their profiles to reflect their current skills and specialties.
- Service Management: Barbers have the flexibility to add or modify the services they offer, tailoring their offerings to meet client demands.
- Scheduling: Barbers can set and adjust their work hours, providing them control over their work-life balance.
- Booking Overview: A dedicated section for managing bookings allows barbers to handle their appointments efficiently.
- Analytics: Barbers can access real-time data on the number of bookings they have, aiding in business management and planning.

Administrator Dashboard:

- Barber Oversight: Administrators have comprehensive access to view all barbers' profiles, work schedules, and the services they offer, ensuring operational consistency and quality control.
- Barber Registration: Administrators can onboard new barbers, integrating them into the platform seamlessly and expanding service capacity.


2.3. User Interface (Diagram and Description)

2.3.1. Sign Up Interface -



Sign up

Profile Picture:



First Name

Last Name

Email Address

+1 336 9012321

Password

Select Role:


☐ User

☐ Barber

SIGN UP

[Already have an account? Sign in](#)

2.3.2. Sign In Interface -



Sign in

Email Address

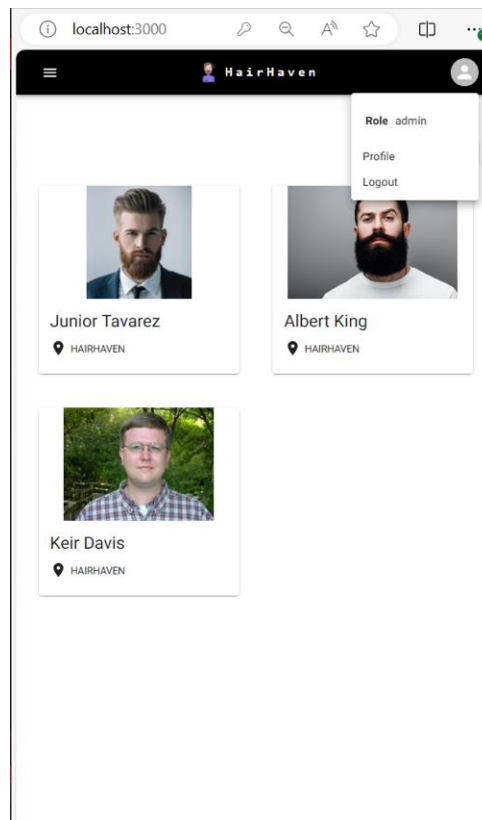
Password

☐ Remember me

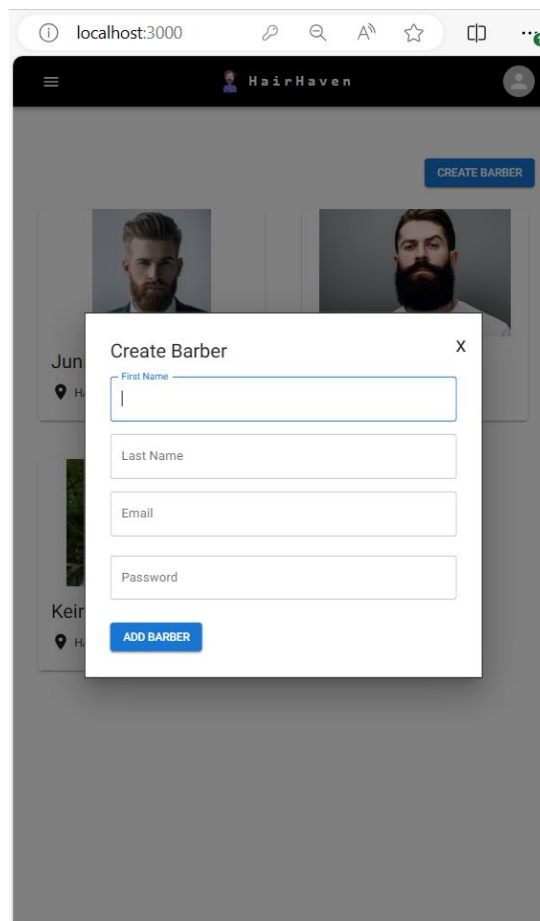
SIGN IN

[Don't have an account? Sign Up](#)

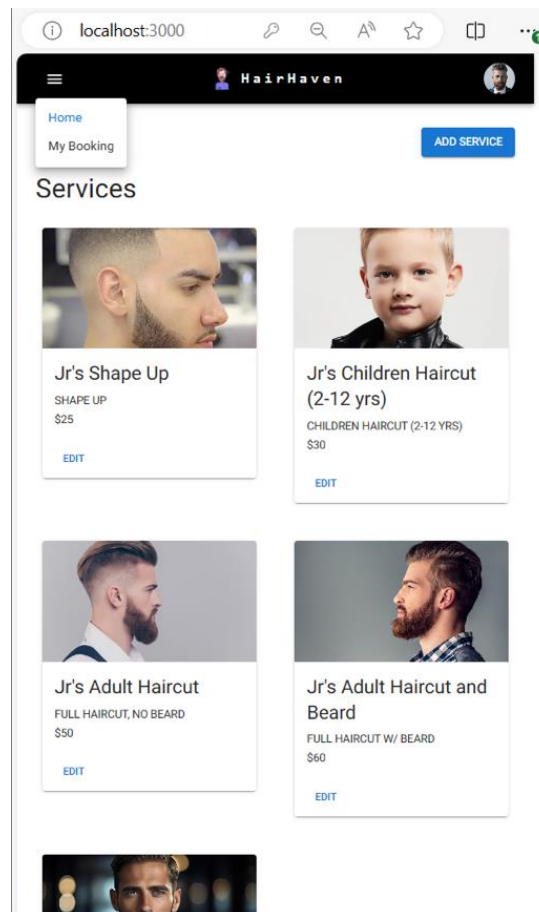
2.3.3. Admin HomePage Interface



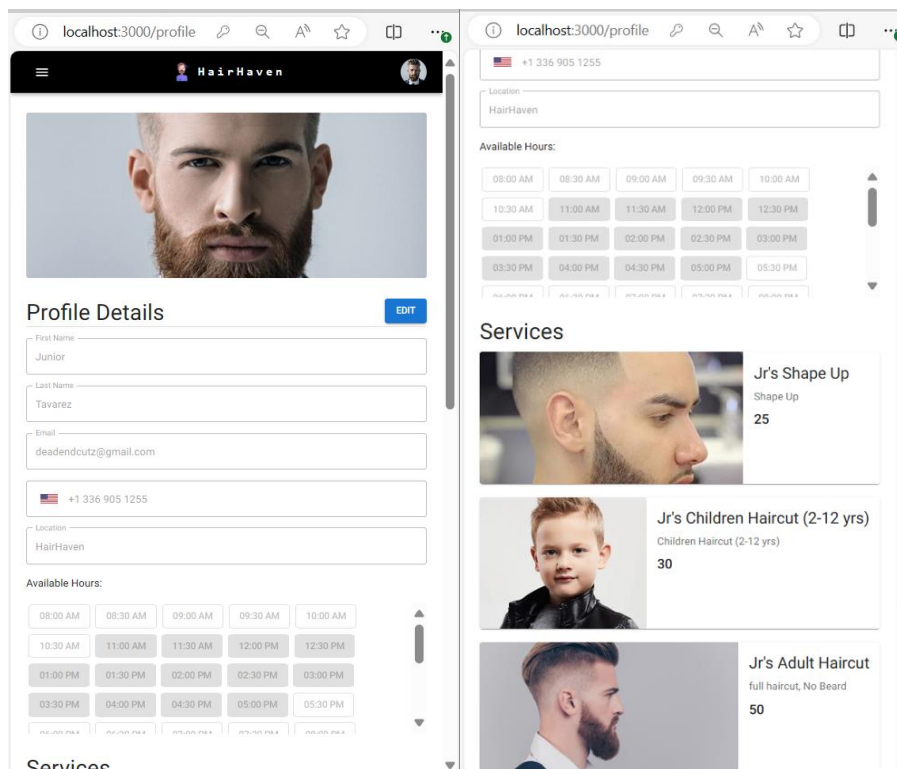
2.3.4. Admin create a barber profile Interface



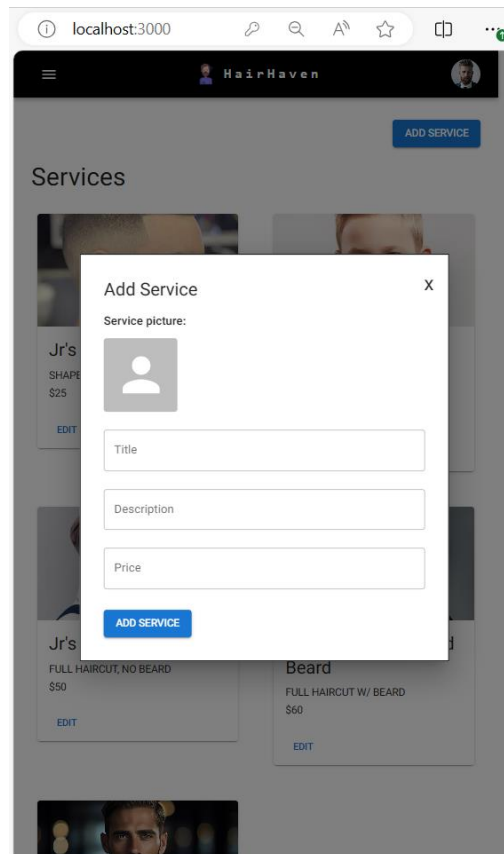
2.3.5. Barber Homepage Interface -



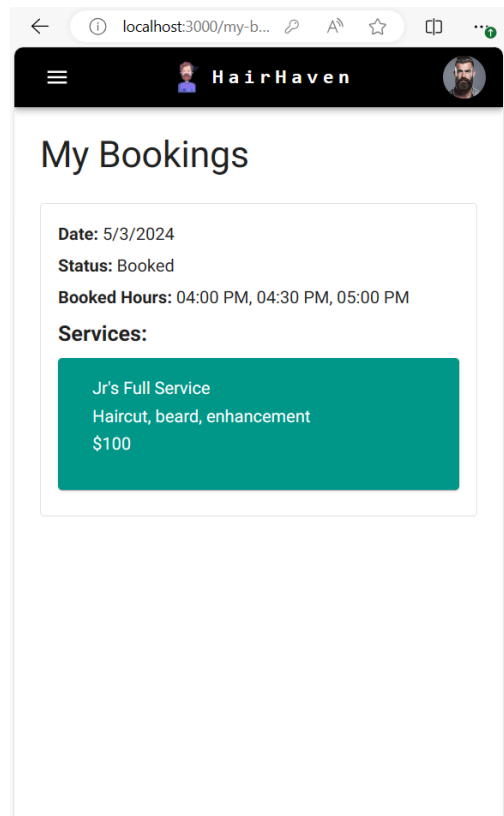
2.3.6. Barber Profile Detail Interface -



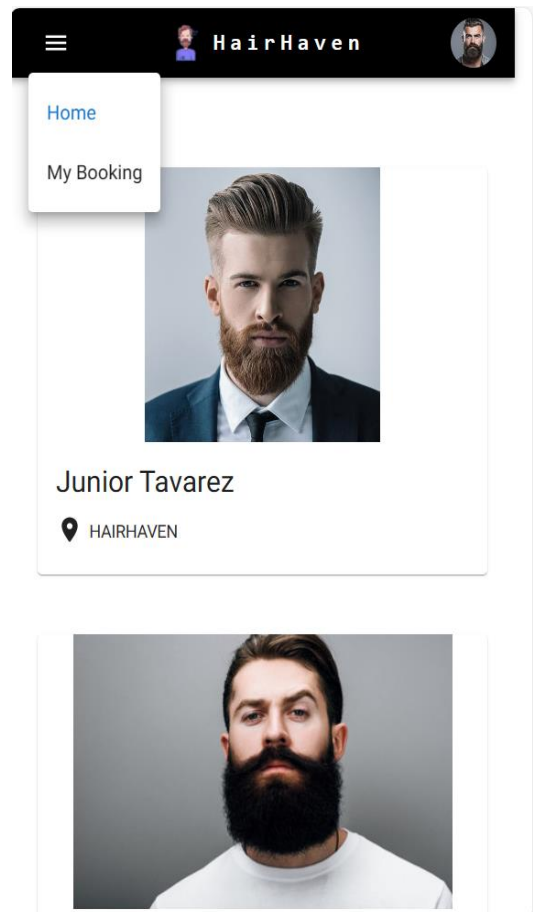
2.3.7. Barber Add Services Interface -



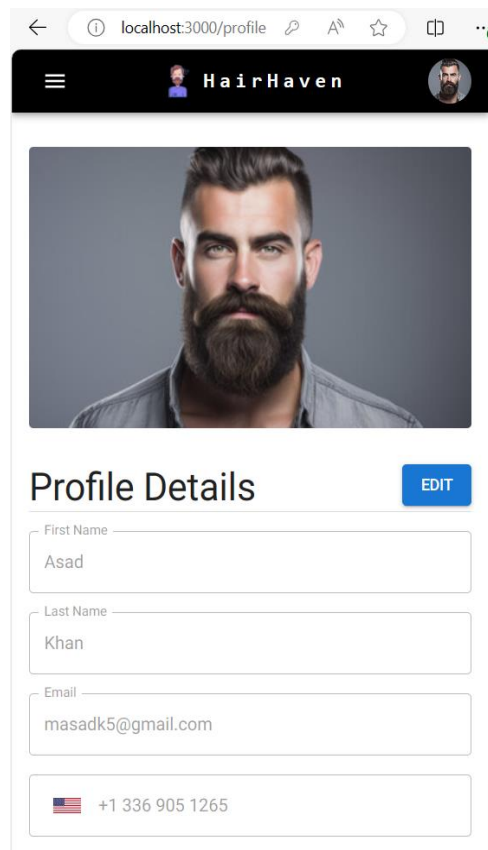
2.3.8. Barber Booking Page Interface -



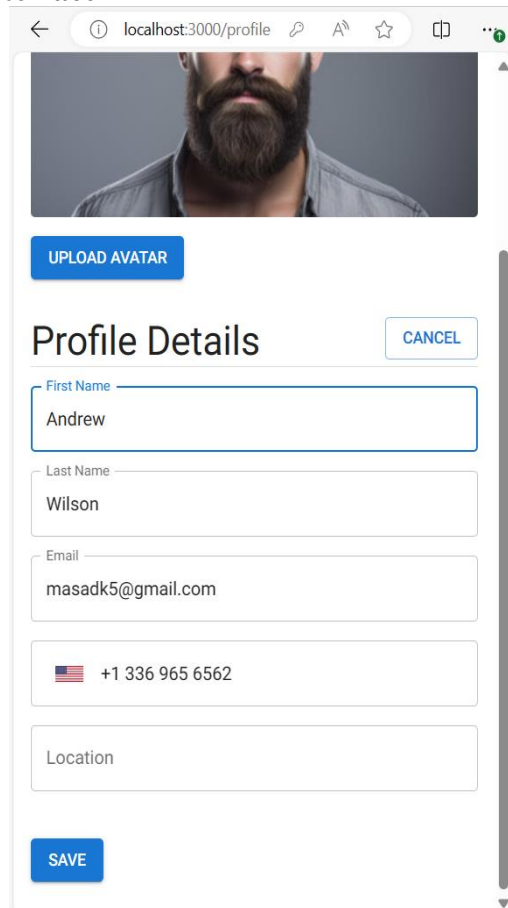
2.3.9. User HomePage Interface -



2.3.10. User Profile Interface -

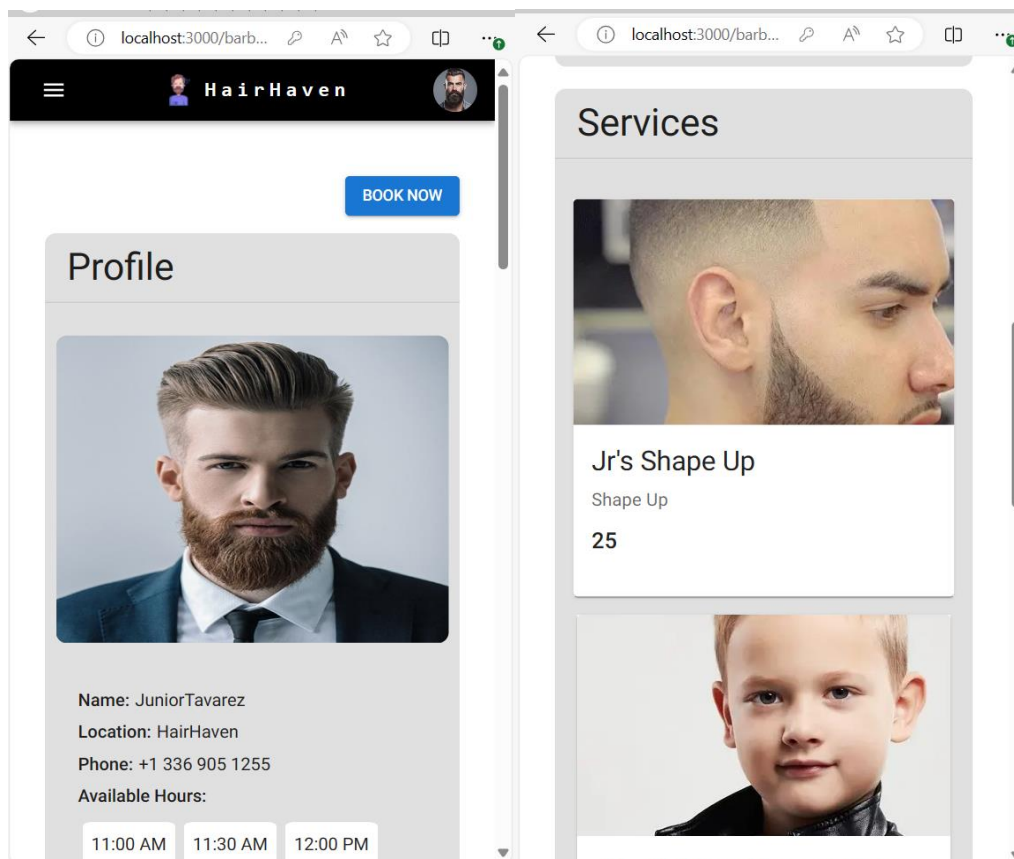


2.3.11. User Edit Profile Interface -

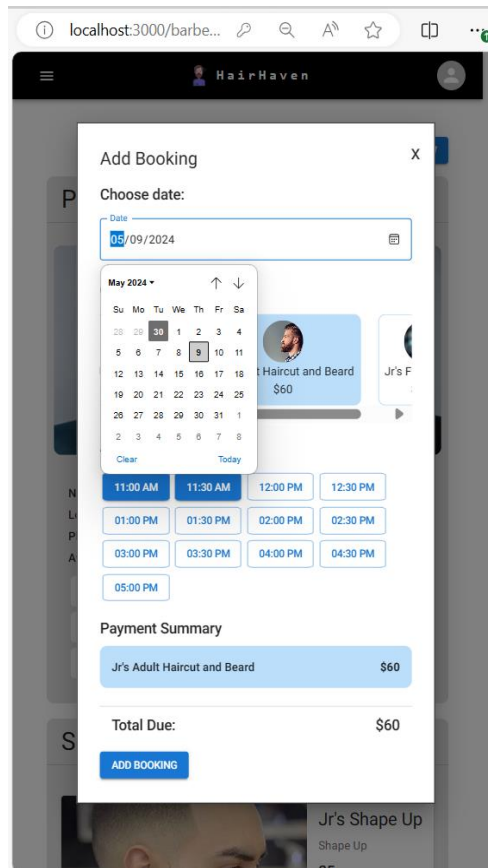


A screenshot of a web browser showing the 'User Edit Profile' interface. The browser address bar displays 'localhost:3000/profile'. The interface features a large profile picture of a man with a beard at the top. Below it is a blue button labeled 'UPLOAD AVATAR'. The main section is titled 'Profile Details' and includes a 'CANCEL' button. It contains several input fields: 'First Name' with the value 'Andrew', 'Last Name' with 'Wilson', 'Email' with 'masadk5@gmail.com', a phone number field with a US flag icon and '+1 336 965 6562', and a 'Location' field. A blue 'SAVE' button is at the bottom.

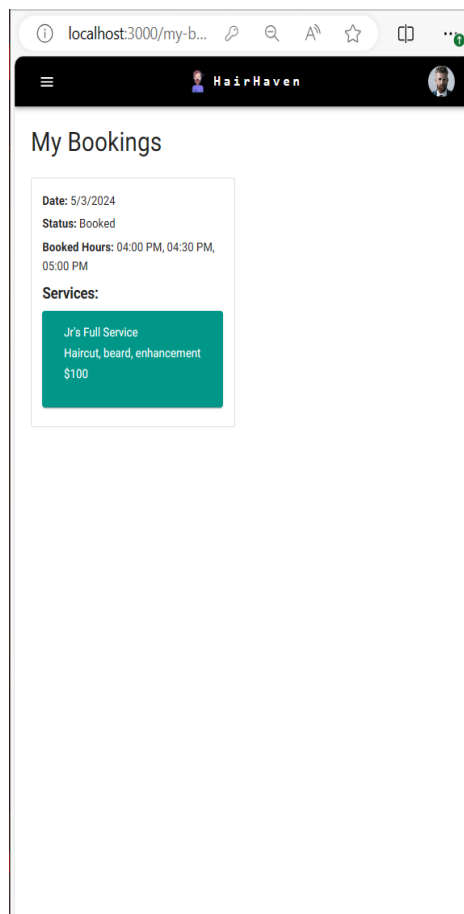
2.3.12. User Viewing Barbers Profile Interface -



2.3.13. User booking an appointment Interface



2.3.14. User Booking Page Interface



2.4. Performance

- Fast load times and quick response to user inputs.
- Efficient handling of simultaneous users without performance degradation.

2.5. System- System Architecture and Data Flow

- The frontend communicates with the backend via RESTful APIs, with data stored in a MySQL database.
- Frontend components interact with the Redux store for state management, using actions and reducers to handle the application state based on user interactions.
- Backend services handle business logic, database interactions, and API responses. Nest.js modules and controllers orchestrate these operations, ensuring a clean separation of concerns.

2.6. Hardware

- Server with adequate processing power, memory, and storage to handle the app's data processing and storage needs.
- Compatible with various devices (smartphones, tablets, computers).

2.7. Software

- Web (e.g., Apache, Axios).
- Backend framework (e.g., Node.js, Nest.js).
- Frontend technology (e.g., React, Next.js, typescript,).]

2.8. Database

- **MySQL:** Chosen for its robustness and support for complex queries and transactions. It's used for storing user, service, and booking data efficiently.
- **Database Schema:** Our schema includes tables for user, service, booking, and a join table `booking_services_service` to manage the many-to-many relationship between bookings and services.

2.9. Security and Authentication

- **@nestjs/jwt and jsonwebtoken:** Facilitate JSON Web Tokens (JWT) handling for secure authentication.
- **bcrypt:** Provides functions for hashing and checking passwords, essential for secure user authentication.

3. Project Specification

- Barbershops often experience challenges in managing appointments and client expectations, leading to inefficient operations and customer dissatisfaction. An automated scheduling system can streamline the booking process, reduce wait times, and improve the overall customer experience.

3.1. Focus/ Domain/ Area

- The app focuses on providing an efficient, user-friendly appointment scheduling system for barbershops. It serves both the barbers and their clients, offering features such as appointment booking, booking management, service selection, barber management, and customer profiles
- The goal is to develop a user-friendly, efficient, and scalable online scheduling system specifically for barbershops that allows clients to book appointments. The system will be built using Next.js for the frontend to deliver a responsive and accessible user interface. Nest.js will serve as the backend framework for robust API management, and MySQL with TypeORM will be used for data persistence. The combination ensures a scalable and maintainable application structure

3.2. Libraries/ Frameworks/ Development Environment

Frontend Technologies

Frameworks and Libraries:

- **Next.js:** Our primary framework for building the frontend. It enables server-side rendering and generates static websites for faster page loads, improving the overall user experience.
- **React:** A JavaScript library for building user interfaces, used alongside Next.js to manage views and state within the application.
- **Material-UI (@mui/material and @mui/icons-material):** Used for designing the UI with pre-built components that can be customized to fit our application's design requirements, ensuring a modern and responsive interface.
- **@emotion/react and @emotion/styled:** CSS-in-JS libraries that let us style our app components using JavaScript, enhancing our ability to use dynamic styling solutions.

State Management:

- **Redux and React-Redux (@reduxjs/toolkit and react-redux):** Provides predictable state container for JavaScript apps, making it easier to manage state across the application.
- **Redux Thunk (redux-thunk):** Middleware that allows us to write action creators that return a function instead of an action, useful for handling asynchronous operations.

Form Handling:

- **React Hook Form (@hookform/error-message and react-hook-form):** Simplifies form handling in React, offering an easy way to collect, validate, and submit form data.

Utilities:

- **Axios:** Promise-based HTTP client for making API requests to communicate with the backend.
- **Yup:** JavaScript schema builder for value parsing and validation, defining a schema to ensure all user inputs meet our requirements.

Additional Components:

- **mui-tel-input:** A Material-UI component for inputting and validating international telephone numbers.
- **React Toastify (react-toastify):** Allows displaying customizable and dismissible alert notifications to the user.
- **@fontsource/roboto:** Provides the Roboto font, ensuring text consistency across all app components.

Development Tools:

- **TypeScript:** Used for developing the application with static types, offering potential for more reliable code and better developer tooling.
- **ESLint, Prettier, and related plugins:** Ensure code quality and consistent formatting across the development team.

Backend Technologies

Frameworks and Libraries:

- **Nest.js:** A progressive Node.js framework for building efficient and scalable server-side applications.
- **TypeORM:** ORM tool used to interact seamlessly with our MySQL database, managing relationships and performing database operations using object-oriented programming.
- **@nestjs/typeorm, @nestjs/common, @nestjs/core:** Core dependencies for Nest.js applications to manage modules, features, and configurations.
- **Reflect-metadata:** Allows using decorators and metadata reflection in TypeScript.

Security and Authentication:

- **@nestjs/jwt and jsonwebtoken:** Facilitate JSON Web Tokens (JWT) handling for secure authentication.
- **bcrypt:** Provides functions for hashing and checking passwords, essential for secure user authentication.

Email Handling:

- **@nestjs-modules/mailer, handlebars, mailgun.js, nestjs-mailgun:** These libraries and modules provide the capabilities to send emails, using templates and integrating with the Mailgun service for email delivery.

Additional Utilities:

- **class-validator and class-transformer:** Used for input validation and to transform plain objects into instances of classes, and vice versa.
- **dotenv:** Loads environment variables from a .env file into process.env, managing sensitive configuration separately from code.

- **multer**: Middleware for handling multipart/form-data, primarily used for uploading files.

Database Technology:

- **MySQL**: Chosen for its robustness and support for complex queries and transactions. It's used for storing user, service, and booking data efficiently.
- **Database Schema**: Our schema includes tables for user, service, booking, and a join table `booking_services_service` to manage the many-to-many relationship between bookings and services.

System Architecture and Data Flow:

- The frontend communicates with the backend via RESTful APIs, with data stored in a MySQL database.
- Frontend components interact with the Redux store for state management, using actions and reducers to handle the application state based on user interactions.
- Backend services handle business logic, database interactions, and API responses. Nest.js modules and controllers orchestrate these operations, ensuring a clean separation of concerns.

Development Environment and Tools:

- The development environment includes version control via Git, with GitHub as the repository hosting service.
- Continuous Integration/Continuous Deployment (CI/CD) pipelines are set up using GitHub Actions to automate testing and deployment processes.

3.3. Platform (Mobile, Desktop, Gaming, Etc.)

- **Mobile Web App**: A fully responsive web application for managing appointments on the go.
- **Desktop**: This is more oriented towards the barbers who manage the schedule from a workstation.

3.4. Genre

- Web App Application
- More specifically, it's a utility and productivity application designed to facilitate service scheduling and business management.

3.5. Teamwork division

- **Muhammad Khan**: Lead Backend Developer - Responsible for setting up the Nest.js backend, database integration, and server-side logic. User Management Sub-System and Appointment Booking Sub-System
- **Romario**: Lead Frontend Developer - In charge of designing the UI/UX with Next.js and Material-UI, implementing frontend logic, and integrating APIs. Barber Management Sub-System
- **Kevin**: Database Administrator/Backend Support - Focuses on database schema design, optimizing queries with TypeORM, and supporting backend development with necessary API integrations. Service Management Sub-System

4. System - Design Perspective

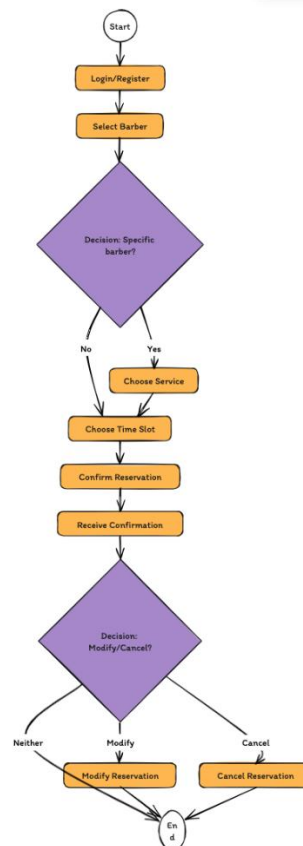
4.1. Identify Sub-Systems Design:

- Booking Management System: Handles user bookings, cancellations, and modifications.
- Barber Management System: Manages barber profiles, schedules, and availability.
- User Management System: Maintains customer profiles, preferences, and history.
- Service Management System: Maintains customer profiles, preferences, and history.

4.2. Sub-System Communication (Diagram and Description)

4.2.1. Controls

- Control mechanisms ensure that only valid users can book, barbers can only manage their schedules, and administrators have control over system settings.



4.2.2. I/O

- Inputs include user data (e.g., name, service choice), barber availability, and p. Outputs are confirmation of bookings.

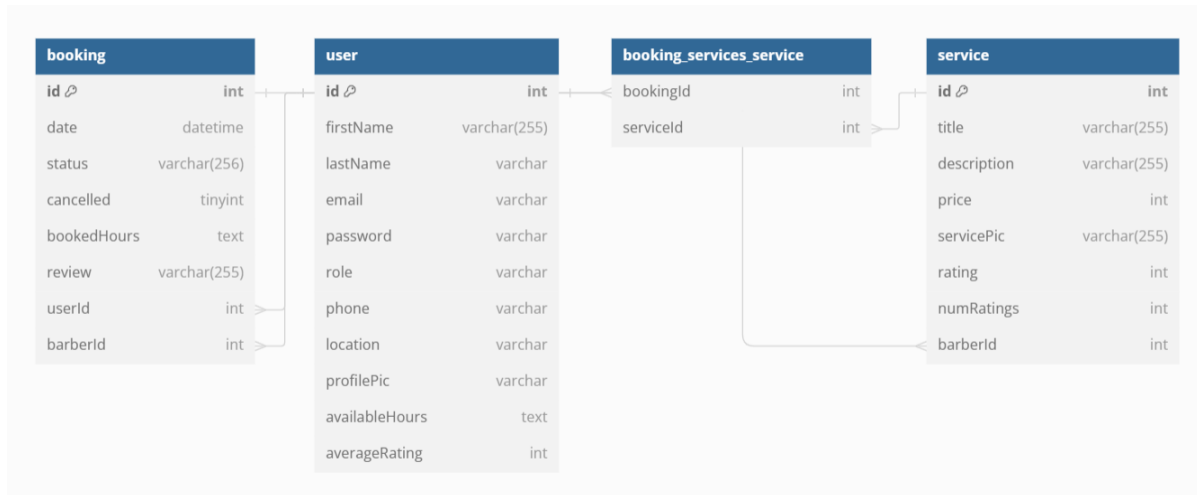
4.2.3. DataFlow

- Data flows between sub-systems, with the user management system sending data to both the Service and Booking, and all systems updating the central database.

4.3. Entity Relationship Model (ER Model)

- Entities include User, Booking, Service, and booking_services_service.
- Relationships include Users booking Reservations, and Reservations involving Services.

4.4. Database Schema



4.5. Overall Operation - System Model

4.5.1. Big-O analysis of overall system and subsystem

- The performance can be assessed in terms of response time (how quickly the system responds to user actions), which should ideally be $O(1)$ for fetching data and $O(\log n)$ for inserting or updating data, depending on the indexing of the database.

5. System - Analysis Perspective

5.1. User Management Sub-System- Analysis Perspective (Muhammad Khan)

- The User Profile Management subsystem is an integral component of a larger system that handles user interactions and data management within the application.
- This subsystem is responsible for:
 - User Authentication and Authorization: Ensures that user credentials are verified and that users have the correct permissions to access their profile.
 - Data Validation and Storage: Validates user input for profile updates and persists these changes in the database.
 - User Interface Management: Provides and updates the user interface where users can view and modify their profile details.

5.2. System (Tables and Description)

5.2.1. Data analysis

This subsystem interacts primarily with the `User` table in the database, where it performs CRUD (Create, Read, Update, Delete) operations on user data.

5.2.2. Data Dictionary (Table - Name, Data Type, Description)

User Table

- Id (INT, PK) Unique identifier for each user.
- firstName (VARCHAR): First name of the user.
- lastName (VARCHAR): Last name of the user.
- email (VARCHAR): Email address of the user.
- password(VARCHAR): Hashed password for user login.
- phone (VARCHAR): Contact number of the user.
- location (VARCHAR): Physical address or location of the user.
- profilePic(VARCHAR): URL to the user's profile picture.
- availableHours (text): Physical address or location of the user.
- averageRating(INT): URL to the user's profile picture.

5.2.3. Process Models

- Update Profile Process:
- Start: User logs into their profile.
- Input: User edits details such as name, email, phone number, and location.
- Validation: System checks new data for correctness (e.g., email format, phone number validity).
- Process: If valid, the system updates the user's details in the database.
- End: User receives a confirmation message; updated information is displayed.

5.3. Algorithm Analysis

5.3.1. Simplified Sub-System to System interaction.

- Big-O Analysis of Overall System and Sub-Systems
- User Data Retrieval (Read): $O(1)$ - Retrieving user data based on the primary key (userId) from the database is a constant time operation.
- User Data Update (Update): $O(1)$ - Updating user data in the database where the primary key is known also operates in constant time.
- Validation Algorithms: $O(n)$ - Validation of user inputs such as email and phone number typically runs in linear time relative to the length of the input.

5.3.2. Implementation Details

- Algorithm Efficiency: Emphasis on ensuring that data retrieval and update operations are optimized for performance, leveraging indexed database queries to achieve constant time complexity.
- Security Considerations: Use of secure hashing algorithms for passwords and secure protocols for data transmission (HTTPS).

5.4. Booking Sub-System - Analysis Perspective (Muhammad Khan)

- The Booking subsystem is a critical component of the barbershop scheduling application. It interacts with several key subsystems:
 1. User Management Subsystem: Ensures that user data and preferences are up-to-date and accessible.
 2. Service Management Subsystem: Maintains details about available services, pricing, and scheduling.

5.5. System (Tables and Description)

5.5.1. Data analysis

- The Booking subsystem is primarily focused on managing appointments, involving scheduling, modifications, cancellations, and confirmations.

5.5.2. Data Dictionary (Table - Name, Data Type, Description)

Booking Table:

- id (INT, PK): A unique identifier for each booking.
- date (DATETIME): Identifier linking to the User table to identify the customer.
- status (VARCHAR): Identifier linking to the Service table that details the type of service booked.
- cancelled(TINYINT): Scheduled date and time for the appointment.
- bookedHours (TEXT): Current status of the booking (e.g., confirmed, cancelled).
- review (VARCHAR): Total cost of the booking.
- userId (INT): userId number.
- barberId (INT): Identifier linking to the User table to identify the customer.

5.5.3. Process Models

Booking Creation Process

- Start: User selects a service and chooses a date and time.
- Input: Service details, selected date and time, and user information.
- Validation: Check for any scheduling conflicts or unavailable slots.
- Process: If no conflicts, save the booking details in the Booking table.
- End: Confirm the booking with the user and update the user interface.

Booking Modification and Cancellation Process

- Start: User requests changes to or cancellation of an existing booking.
- Input: Updated details or cancellation request.
- Validation: Verify permissible changes and check cancellation policies.
- Process: Update the Booking table accordingly.
- End: Notify the user of successful update or cancellation

5.6. Algorithm Analysis

5.6.1. Simplified Sub-System to System interaction.

- Big-O Analysis of Overall System and Sub-Systems
- Data Retrieval (Read): $O(\log n)$ for retrieving booking details based on indexed search criteria like `bookingId` or `userId`.
- Insertion (Create): $O(1)$ assuming a direct insert operation into an efficiently indexed table.
- Update/Deletion (Update/Delete): $O(\log n)$ due to the need to first locate the record before updating or deleting, based on an indexed column.

5.6.2. Implementation Details

- Algorithm Efficiency: Focus on optimizing read operations since booking retrieval will be frequent and should be fast to ensure a smooth user experience.
- Concurrency Handling: Implement transactional control to manage simultaneous booking attempts, ensuring data integrity and consistency.
- Security Measures: Enforce authentication and authorization checks to ensure that users can only manage their own bookings.

5.7. User Authentication Sub-Systems - Analysis Perspective (Muhammad Khan)

- The User Authentication subsystem is a fundamental component that interacts closely with various parts of the application, such as:
 1. User Management Subsystem: Manages user profiles and access rights.
 2. Session Management Subsystem: Manages user sessions post-authentication to maintain a secure and persistent user state during interactions.
 3. Security Subsystem: Provides underlying security mechanisms like encryption and token generation that are critical for authenticating user credentials and maintaining session integrity.

5.8. System (Tables and Description)

5.8.1. Data analysis

This subsystem focuses on user registration and login processes, ensuring secure and efficient handling of user credentials.

5.8.2. Data Dictionary (Table - Name, Data Type, Description)

User Table

- Id (INT, PK) Unique identifier for each user.
- firstName (VARCHAR): First name of the user.
- lastName (VARCHAR): Last name of the user.
- email (VARCHAR): Email address of the user.
- password(VARCHAR): Hashed password for user login.
- phone (VARCHAR): Contact number of the user.
- location (VARCHAR): Physical address or location of the user.

- profilePic(VARCHAR): URL to the user's profile picture.
- availableHours (text): Physical address or location of the user.
- averageRating(INT): URL to the user's profile picture.

5.8.3. Process Models

Registration Process

- Start: The user fills out the registration form.
- Input: User details such as name, email, password, and optionally a profile picture.
- Validation: Checks if the email is already registered and if the password meets the security criteria.
- Process: Encrypt the password, create a new user entry in the User table.
- End: Confirm registration and direct the user to either log in or directly into the application.

Login Process

- Start: The user enters their email and password.
- Input: Email and password.
- Validation: Verify if the email exists and if the password matches the stored hash.
- Process: Generate a session token if the credentials are valid.
- End: Grant access to the user and redirect to the homepage or user dashboard.

Algorithm Analysis

5.8.4. Simplified Sub-System to System interaction.

- Big-O Analysis of Overall System and Sub-Systems
- Registration (Insertion): $O(1)$ - Inserting a new user is a constant time operation assuming the email uniqueness is enforced by an index.
- Login (Search): $O(\log n)$ - Searching for a user by email involves a logarithmic time complexity if indexed properly.
- Password Hashing: $O(1)$ - Hashing the password typically occurs in constant time, independent of the database size.

5.8.5. Implementation Details

- Security Measures: Utilizes best practices such as SSL/TLS for data transmission, bcrypt for hashing passwords, and HTTPS-only cookies for session management.
- Scalability Considerations: Designed to handle a high number of registrations and logins simultaneously through efficient use of database indices and connection pooling.
- Reliability and Fault Tolerance: Implements error handling to manage failed login attempts and provide useful feedback to users.

5.9. Service Management Sub-System - Analysis Perspective (Kevin)

- The Service Management subsystem plays a crucial role within the broader system architecture, specifically designed for barbers or salon managers to manage their service offerings. It interfaces with:
 1. Database Management Subsystem: Stores and retrieves service details.
 2. User Interface Subsystem: Provides a user-friendly interface for barbers to add or edit service details.
 3. Authorization Subsystem: Ensures that only authenticated users can modify service information.

5.10. System (Tables and Description)

5.10.1. Data analysis

This subsystem is responsible for the creation, modification, and viewing of service details by authenticated barbers.

5.10.2. Data Dictionary (Table - Name, Data Type, Description)

Service Table

- serviceId (INT, PK): Unique identifier for each service.
- barberId (INT, FK): Identifier linking the service to a specific barber.
- title (VARCHAR): Name or title of the service.
- description (TEXT): Detailed description of the service.
- price (DECIMAL): Cost of the service.
- serviceImageUrl (VARCHAR): URL to an image representing the service.

5.10.3. Process Models

Service Addition Process

- Start: Barber selects the "Add Service" option.
- Input: Inputs for service image, title, description, and price.
- Validation: Validate inputs to ensure data integrity and adherence to business rules (e.g., non-empty fields, valid pricing format).
- Process: Save new service details in the Service table.
- End: Confirmation message displayed, and new service appears on the barber's service list.

Service Editing Process

- Start: Barber chooses to edit an existing service.
- Input: Updated details for the selected service.
- Validation: Checks similar to the addition process.
- Process: Update details in the Service table for the given serviceId.
- End: Updated service details are confirmed and displayed.

5.10.4. Simplified Sub-System to System interaction.

- Big-O Analysis of Overall System and Sub-Systems
- Insert Operation (Adding Service): $O(1)$ - Assuming efficient handling by the database management system with appropriate indexing on the `serviceId`.
- Update Operation (Editing Service): $O(1)$ - Direct updates to a single row identified by the `serviceId` also benefit from efficient index usage.
- Query Operation (Retrieving Service List): $O(n)$ - Depending on the implementation, retrieving all services for a specific barber could scale linearly with the number of services offered.

5.10.5. Implementation Details

- Algorithm Efficiency: Emphasizes the need for efficient data manipulation operations to ensure quick response times for end users, especially important for real-time updates in service information.
- Security and Integrity: Robust authentication and authorization checks to prevent unauthorized access or modifications, along with input validation to prevent SQL injection and other common security vulnerabilities.
- Usability and Accessibility: The interface is designed to be intuitive, allowing easy navigation and manipulation of service data, crucial for user satisfaction and operational efficiency.

5.11. Barber Management Sub-System - Analysis Perspective (Romario)

- The Barber Profile Display subsystem is part of a broader system that integrates with the following:
 1. User Authentication and Authorization Subsystem: Ensures only authenticated users can view and book appointments.
 2. Booking Subsystem: Users can directly book appointments with the barber from their profile.

5.12. System (Tables and Description)

5.12.1. Data analysis

This subsystem revolves around displaying detailed profiles of barbers, including their available times, services, and contact information

5.12.2. Data Dictionary (Table - Name, Data Type, Description)

Service Table

- serviceId (INT, PK): Unique identifier for each service.
- barberId (INT, FK): Identifier linking the service to a specific barber.
- title (VARCHAR): Name or title of the service.
- description (TEXT): Detailed description of the service.
- price (DECIMAL): Cost of the service.
- serviceImageUrl (VARCHAR): URL to an image representing the service.

5.12.3. Process Models

1. Service Addition Process

- Start: Barber selects the "Add Service" option.
- Input: Inputs for service image, title, description, and price.

- Validation: Validate inputs to ensure data integrity and adherence to business rules (e.g., non-empty fields, valid pricing format).
- Process: Save new service details in the Service table.
- End: Confirmation message displayed, and new service appears on the barber's service list.

2. Service Editing Process

- Start: Barber chooses to edit an existing service.
- Input: Updated details for the selected service.
- Validation: Checks similar to the addition process.
- Process: Update details in the Service table for the given serviceId.
- End: Updated service details are confirmed and displayed.

5.12.4. Simplified Sub-System to System interaction.

- Big-O Analysis of Overall System and Sub-Systems
- Insert Operation (Adding Service): $O(1)$ - Assuming efficient handling by the database management system with appropriate indexing on the `serviceId`.
- Update Operation (Editing Service): $O(1)$ - Direct updates to a single row identified by the `serviceId` also benefit from efficient index usage.
- Query Operation (Retrieving Service List): $O(n)$ - Depending on the implementation, retrieving all services for a specific barber could scale linearly with the number of services offered.

5.12.5. Implementation Details

- Algorithm Efficiency: Emphasizes the need for efficient data manipulation operations to ensure quick response times for end users, especially important for real-time updates in service information.
- Security and Integrity: Robust authentication and authorization checks to prevent unauthorized access or modifications, along with input validation to prevent SQL injection and other common security vulnerabilities.
- Usability and Accessibility: The interface is designed to be intuitive, allowing easy navigation and manipulation of service data, crucial for user satisfaction and operational efficiency.

6. Project Scrum Report

6.1. Product Backlog

Table for Product Backlog:

ID	User Story	Priority	Estimate (days)	Notes
1	User can create a profile	High	3	Includes photo upload
2	User can book appointments	High	5	Calendar integration
3	User can view appointments	Medium	4	Changes allowed up to 24 hours prior
4	User edits profile	High	2	Email and SMS notifications
5	Barbers can set their availability	High	3	Sync with personal calendar
6	Admin dashboard for managing appointments	Medium	6	Real-time data updates
7	Implement user authentication and authorization	High	4	Secure user logins and appropriate access
8	Frontend and backend basic setup	High	5	Setup development environment and initial codebase
9	Advanced appointment management features	Medium	7	Including status updates and cancellations
10	Database integration and schema implementation	High	4	Proper data storage and retrieval
11	Notification system for reminders and alerts	High	3	Push notifications and email integration
12	UI consistency and responsiveness	Medium	4	Ensure compatibility across devices
13	Comprehensive testing and deployment	High	5	Includes unit, integration, and usability testing

6.2. Sprint Backlog

- The Sprint Backlog is the list of tasks pulled from the Product Backlog to be completed during a sprint. It provides a clear picture of the work that the development team commits to complete within a sprint

Table for Sprint Backlog: Week 1

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
1	Create wireframes for the UI	Romario	In Progress	16	0
2	Develop UX flowcharts for UI pages	Romario	Not Started	20	0
3	Design database schema	Muhammad	Planned	12	0

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
4	Set up development environment	Muhammad	In Progress	8	0
5	Begin coding UI skeleton	Kevin	Not Started	15	0
6	Design app architecture and select tools	Muhammad	Not Started	10	0

Sprint Backlog: Week 2

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
1	Design user profile interface	Muhammad	Complete	16	0
2	Implement calendar integration for bookings	Muhammad	Not Started	20	0
3	Set up SMS notification system	N/a	Not Started	0	24
4	Test profile creation and editing features	Kevin	Not Started	8	0
5	Develop barber availability settings	Romario	Planned	15	0

Sprint Backlog for Week 3:

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
7	Implement front-end functionality (calendar view, appointment booking form)	Romario	Complete	12	0
8	Link frontend views with backend endpoints	Muhammad	Complete	15	0
9	Develop controller classes for frontend-backend communication	Muhammad	Complete	10	0
10	Implement database schema	Kevin	Complete	8	0
11	Develop controller classes for backend-database communication	Kevin	Complete	12	0

Sprint Backlog for Week 4:

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
12	Code business logic for managing appointments	Muhammad	Complete	20	0
13	Continue frontend development for appointment status and cancellations	Romario	Complete	18	0
14	Implement user authentication and authorization	Muhammad	Complete	15	0
15	Test and refine user-specific views and access controls	Kevin	Complete	12	0
16	Optimize and debug backend and frontend linkage	Kevin	Complete	10	0

Sprint Backlog for Week 5

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
17	Integrate push notifications and email reminders for appointments	Jeese	Not Started	12	12
18	Finish any incomplete tasks from previous sprints	Everyone	N/A	Variable	Variable
19	Polish UI visuals and ensure view integrity on various screen sizes and orientations	Everyone	Complete	16	0
20	Conduct preliminary user testing and gather feedback for final adjustments	Muhammad	Complete	10	0

Sprint Backlog for Week 6:

ID	Task	Assigned To	Status	Estimated Hours	Remaining Hours
21	Perform unit testing, integration testing, and usability testing	Muhammad	Complete	10	0
22	Fix any identified bugs and issues	Everyone	N/A	Variable	Variable
23	Deploy the web app and monitor for any deployment issues	Muhammad	Complete	12	0
24	Conduct final user testing to verify project is fully functional	Muhammad	Complete	10	0

7. Sub-Systems

7.1 Subsystem 1 (Muhammad Khan): User Management Subsystem

- The User Management subsystem is an integral component of a larger system that handles user interactions and data management within the application.
- This subsystem is responsible for:
 1. User Authentication and Authorization: Ensures that user credentials are verified and that users have the correct permissions to access their profile.
 2. Data Validation and Storage: Validates user input for profile updates and persists these changes in the database.
 3. User Interface Management: Provides and updates the user interface where users can view and modify their profile details.

7.1.1. Initial Design and Model

Diagrams:

- Class Diagram: Shows User, Admin, Barber, each with methods for account management.
- Use-Case Diagram: Outlines user registration, login, profile management.
- Sequence Diagram: Details the process from user login to profile updating.

Design Choices:

- Used a modular design to allow scalability.
- Implemented role-based access control for different user types.

7.1.2. Data Dictionary

User:

- id (INT, PK)
- firstName (VARCHAR)
- lastName (VARCHAR)
- email (VARCHAR)
- password(VARCHAR)
- phone(VARCHAR)
- location (VARCHAR)
- profilePic (VARCHAR)
- availableHours(TEXT)

7.1.3. Refined model and design

- Reason for Refinement: To improve security and usability.
- Changes from Initial Model: Added two-factor authentication for enhanced security.
- Refined Model Analysis: Ensured that user data is more secure while maintaining ease of access.
- Refined Design: Updated UML diagrams to reflect new authentication flow.

7.1.4. Scrum Backlog (Product and Sprint- [link to section](#))

- Refer to Section 6

7.1.5. Coding - approach(funtional, OOP) and Language

- Approach: Object-Oriented Programming (OOP)
- Language: TypeScript with NestJS

7.1.6. User Training - training/user manual

- simple instructions on managing user roles and secure login procedures.

7.1.7. Testing

- Automated tests for login, registration, and profile updates.
- Security tests to validate new authentication measures.

7.2 Subsystem 2 (Muhammad Khan): Booking Subsystem

- The Booking subsystem is a critical component of the barbershop scheduling application. It interacts with several key subsystems:
 1. User Management Subsystem: Ensures that user data and preferences are up-to-date and accessible.

2. Service Management Subsystem: Maintains details about available services, pricing, and scheduling.

7.1.8. Initial Design and Model

Diagrams:

- Class Diagram: Includes classes like user, booking, service, and book_services_service.
- Use-Case Diagram: Captures booking and cancellation.
- Sequence Diagram: Visualizes the appointment booking flow.

Design Choices:

- Integrated real-time scheduling to accommodate dynamic availability changes.

7.1.9. Data Dictionary

booking_service_services:

- bookingID (INT, PK)
- serviceID (INT, FK)

7.1.10. Refined model and design

- Reason for Refinement: To enhance real-time data processing.
- Changes from Initial Model: Optimized the scheduling algorithm to reduce latency.
- Refined Model Analysis: Faster appointment processing and improved user experience.
- Refined Design: Sequence diagrams updated to show optimized data flows.

7.1.11. Scrum Backlog (Product and Sprint- [link to section](#))

- Refer to section 6

7.1.12. Coding - approach(functional, OOP) and Language

- Approach: Functional Programming for state management.
- Language: JavaScript with Redux for state control in a React environment.

7.1.13. User Training - [training/user manual](#)

- Guidelines on managing and optimizing appointment schedules.

7.1.14. Testing

- Load testing to ensure system handles high traffic.
- Functional testing for all scheduling features.

8. Complete System

8.1. Final Software/Hardware Product

- Complete Code in GitHub

8.2. Source code and demonstration video- screenshots - technical report githublink

- In Github along with the video. Also emailing this report and video to Professor Davis

8.3. Evaluation by client and instructor

- Professor Davis

8.4. Team Member Descriptions

- **Muhammad Khan:** Lead Backend Developer - Responsible for setting up the Nest.js backend, database integration, and server-side logic. User Management Sub-System and Appointment Booking Sub-System
- **Romario:** Lead Frontend Developer - In charge of designing the UI/UX with Next.js and Material-UI, implementing frontend logic, and integrating APIs. Barber Management Sub-System
- **Kevin:** Database Administrator/Backend Support - Focuses on database schema design, optimizing queries with TypeORM, and supporting backend development with necessary API integrations. Service Management Sub-System.