BEAUTAI MVP DEVELOPMENT MANUAL

TECHNICAL DOCUMENTATION

Comprehensive Guide to BEAUT-AI Development

Version 1.0

Prepared by: BEAUT-AI Development Team

January 21, 2025

Contents

1 Overview

This document provides a comprehensive outline of the key features and requirements necessary for the successful launch of BeautAI's Minimum Viable Product (MVP). It serves as the primary reference guide for the development team, ensuring alignment and clarity throughout the project lifecycle. Given that the team is navigating its first full-scale product launch, this document is designed to offer structured guidance and facilitate effective project tracking.

As the app evolves and expands, so will this document. It will continue to grow in tandem with new features, technical requirements, and strategic directions. This ensures that all contributors have the latest updates, specifications, and best practices at their fingertips, creating a single source of truth for the entire development lifecycle. By adhering to the outlined roadmap, the team can maintain focus and momentum as we work collaboratively to bring BeautAI to market

2 Tech Stack

2.1 Frontend

- **Expo:**
  - Use Expo for building cross-platform mobile applications with React Native.
  - Leverage Expo's managed workflow for easy setup and development.
  - Utilize Expo's libraries and APIs for accessing native device features.
- **React Native:**
  - Implement a component-based architecture for reusable UI components.
  - Use React Hooks for managing state and side effects in functional components.
  - Ensure responsive design and adaptation to various screen sizes and orientations.
- **Styling:**
  - Use Styled Components or React Native's StyleSheet for styling components.
  - Implement a consistent design system with reusable styles and themes.
  - Ensure accessibility and usability across different devices and platforms.
- **State Management:**
  - Use Redux or Context API for managing global state in the application.
  - Implement middleware like Redux Thunk for handling asynchronous actions.
  - Optimize state updates to improve performance and reduce re-renders.

2.2 Backend

- **NestJS:**
  - Use NestJS for building scalable and maintainable server-side applications.
  - Implement a modular architecture with controllers, services, and modules.
  - Use decorators and dependency injection for clean and organized code.
- **GraphQL:**
  - Use GraphQL for flexible and efficient data querying.
  - Implement a schema-first approach with TypeScript for type safety.

  - Use Apollo Server for GraphQL API implementation and integration.
- **Authentication:**
  - Implement JWT-based authentication for secure session management.
  - Use Passport.js for integrating various authentication strategies.
  - Ensure secure handling of authentication tokens and user credentials.

2.3 Database

- **MySQL:**
  - Use MySQL as the primary relational database for structured data storage.
  - Implement normalization and indexing for optimized query performance.
  - Use ORM tools like TypeORM for database interaction and migration management.
- **Data Backup and Recovery:**
  - Set up automated backups and snapshots for data protection.
  - Implement disaster recovery strategies to ensure data availability.
  - Regularly test backup and recovery processes to ensure reliability.

2.4 Cloud

- **AWS:**
  - Use AWS for scalable cloud infrastructure and services.
  - Deploy applications using AWS ECS or EKS for container orchestration.
  - Use AWS S3 for storing static assets and user-uploaded files.
- **DevOps:**
  - Implement CI/CD pipelines using AWS CodePipeline or Jenkins.
  - Use Infrastructure as Code (IaC) tools like Terraform for resource provisioning.
  - Monitor application performance and resource utilization with AWS CloudWatch.

3 Authentication & User Management

3.1 User Authentication System

- **Customer Authentication:**
  - **JWT-based Authentication:** Implement JSON Web Tokens for secure session management, with refresh tokens for session renewal.
  - **Email Verification:** Use SendGrid to send verification emails upon signup, ensuring valid email addresses.
  - **Rate Limiting:** Add rate limiting on login attempts to prevent brute force attacks, allowing a maximum of 5 attempts per 15 minutes.
  - **Push Notifications:** Integrate push notifications to alert users about account activity, such as login from a new device or password changes, enhancing security and engagement.
- **Profile Management:**
  - **Profile Picture Upload:** Integrate with AWS S3 for storing and retrieving user profile pictures.

- **Contact Information Management:** Allow users to update their contact details, including phone numbers and addresses.
  - **Notification Preferences:** Enable users to set preferences for receiving notifications via email.

- **Security Features:**
  - **Password Strength Requirements:** Enforce strong passwords with a minimum of 8 characters, including special characters and numbers.
  - **Password Hashing:** Use bcrypt for hashing passwords, ensuring secure storage.
  - **Session Management:** Implement token expiration and renewal mechanisms for secure session handling.

3.2 Deferred for Future Implementation

- **Business Authentication:**
  - **Separate Authentication Flow:** Provide a simple, tailored authentication process for businesses.
  - **Business Verification:** Implement a basic admin approval process for business accounts.
- **OAuth Integration:** Integration with Google and Facebook for social login.
- **Multi-factor Authentication:** Provide additional security options for businesses.
- **Advanced Admin Panel:** Develop a comprehensive admin interface for managing business approvals and monitoring account status.
- **Notification Preferences via SMS:** Extend notification preferences to include SMS.
- **CSRF Protection:** Use anti-CSRF tokens for cross-site request forgery prevention.
- **Account Deletion:** Implement a fully compliant process for users to request account deletion in line with data protection regulations.

4 Business Management

4.1 Business Profile System

- **Manually Stored Information:**
  - **Business Name:** Store the business name in the database for reference.
  - **Business Address:** Maintain a record of the physical address of businesses.
  - **Contact Details:** Store contact numbers manually in the database.

4.2 Deferred for Future Implementation

- **Profile Information:**
  - **Business Name and Description:** Allow businesses to enter a name and a brief description that highlights their services and specialties.
  - **Multiple Contact Numbers:** Enable businesses to provide multiple contact numbers for different departments or purposes.
  - **Business Category Selection:** Provide a list of predefined categories (e.g., Hair, Makeup, Spa) for businesses to classify their services.
  - **Business Hours with Timezone Support:** Enable businesses to set their operating hours, including support for different time zones.

  - **Physical Address with Geocoding:** Integrate with a geocoding API to convert physical addresses into geographical coordinates for map-based features.
  - **Social Media Links:** Allow businesses to add links to their social media profiles to increase customer engagement.
  - **Holiday Schedule Management:** Allow businesses to specify holidays or special closing days to manage customer expectations.

- **Team Management:**
  - **Staff Profiles with Roles:** Create profiles for each staff member, assigning roles such as admin, stylist, or receptionist.
  - **Service Assignment to Staff Members:** Allow businesses to assign specific services to staff members based on their expertise.
  - **Individual Staff Calendars:** Provide each staff member with a personal calendar to manage their appointments and availability.
  - **Staff Availability Management:** Enable businesses to manage staff availability, including setting working hours and break times.

- **Service Management:**
  - **Service Categories and Subcategories:** Organize services into categories and subcategories for easier navigation and selection by customers.
  - **Duration Settings with Buffer Time:** Enable businesses to specify the duration of services, including buffer time for preparation or cleanup.
  - **Service Description and Requirements:** Provide fields for detailed service descriptions and any prerequisites or requirements for customers.
  - **Pricing Tiers:** Allow businesses to set different pricing tiers (e.g., basic, premium) for their services.
  - **Before/After Photos for Services:** Allow businesses to upload before and after photos to showcase the effectiveness of their services.
  - **Bulk Service Import/Export:** Facilitate the import and export of service lists for businesses with extensive catalogs.

5 Service Management

5.1 Service Type

- Each service is associated with a static business profile and includes key details such as:
  - **Name**: The name of the service (e.g., Massage, Consultation).
  - **Description**: A detailed description of what the service offers.
  - **Duration**: The time required to complete the service (in minutes).
  - **Price**: The cost of the service.
  - **Category**: The category the service falls under (e.g., Health, Beauty, Fitness).
  - **Availability**: A list of available times for the service, linked to business availability.
- Services are linked to static business profiles, allowing customers to browse available services per business.
- Services are categorized to make browsing easier for users, enabling filtering and sorting by service type, category, or business.

5.2 Service Profile Linked to Business

- Each service is linked to a static business profile which includes:
  - **Business Name**: The business offering the service.
  - **Business Location**: Physical address or online availability.
  - **Business Rating**: Customer reviews and ratings for the business.
  - **Service Hours**: Availability of services based on the business hours.
- Services are dynamically displayed within business profiles to provide clear navigation for customers.
- Businesses may offer different services within the same category, with each service being fully customizable.

6 Appointment System

6.1 Current Implementation

- Instead of an appointment booking system, users will be shown basic business information, including:
  - **Business Name:** Display the name of the business.
  - **Contact Details:** Provide phone numbers or email addresses for direct communication.
  - **Address:** Show the physical location of the business.
- Customers can directly contact businesses to inquire about services or book appointments manually.

6.2 Deferred Features

- **Appointment Booking Flow:**
  - **Date/Time Selection:**
    - Implement a calendar interface for customers to select dates.
    - Display available time slots based on staff availability and service duration.
    - Allow customers to choose preferred staff members if desired.
  - **Service Selection:**
    - Provide a list of available services with detailed descriptions and pricing.
    - Enable filtering by service categories and subcategories.
    - Allow customers to select multiple services in a single booking session.
  - **Basic Availability Checking:**
    - Real-time availability checking to prevent double bookings.
    - Consideration of staff working hours, breaks, and holidays.
    - Automatic updates to availability upon booking confirmation.
- **Appointment Management:**
  - **View, Modify, or Cancel Appointments:**
    - Provide customers with a dashboard to view upcoming and past appointments.
    - Allow modifications to appointment details, including rescheduling and service changes.
    - Implement cancellation policies and fees, if applicable.
  - **Status Updates:**
    - Appointment statuses include Pending, Confirmed, Completed, and Cancelled.
    - Automatic status updates based on business rules (e.g., auto-confirmation after a set period).

    * Notifications sent to customers and staff upon status changes.

- **Email Notifications:**
  - **Booking Confirmation:**
    * Send confirmation emails to customers upon successful booking.
    * Include appointment details, such as date, time, services, and staff.
  - **Appointment Reminders:**
    * Schedule reminder emails to be sent 24 hours and 1 hour before appointments.
    * Allow customers to customize reminder settings.
  - **Status Changes:**
    * Notify customers of any changes in appointment status (e.g., cancellations or rescheduling).
    * Provide links for quick access to appointment management in the email.

7 Search & Discovery

7.1 Basic Business Search

- **List View of Businesses:**
  - Display businesses in a list format with key information such as name, category, and rating.
  - Include business logos or images for visual appeal.
  - Implement pagination or infinite scroll for large result sets.
- **Simple Filters:**
  - Filter businesses by service type, location, and category.
  - Provide options for filtering by business hours and availability.
  - **Deferred:** Allow users to save filter preferences for future searches.
- **Basic Sorting:**
  - Sort businesses by distance from user's location, ratings, and popularity.
  - **Deferred:** Implement sorting by price range and service availability.
  - **Deferred:** Allow users to toggle between ascending and descending order.

7.2 Deferred Features

- **Service Browsing:**
  - **Filter by Category:**
    * Display service categories prominently for easy navigation.
    * Allow users to explore services within selected categories.
    * Provide subcategory filters for more granular browsing.
  - **Price Range Filtering:**
    * Enable users to set minimum and maximum price ranges for services.
    * Display average pricing for selected services.
    * Highlight promotions or discounts within the price range.
  - **Simple List View:**
    * Present services in a clean, organized list with essential details.
    * Include service descriptions, durations, and pricing.
    * Allow users to add services to their favorites for quick access.

8 Basic Customer Features

8.1 Service History

- **View Past Services:**
  - Provide customers with a history of all services availed.
  - Include details such as service names, dates, staff members, and costs.
  - Allow customers to reorder past services with a single click.
- **Service Feedback:**
  - Enable customers to provide feedback or ratings for past services.
  - Display average ratings and reviews for each service.
  - Allow customers to edit or delete their reviews within a set timeframe.

8.2 Consultation Form

- **Basic Consultation Form:**
  - Provide a form for customers to fill out prior to appointments.
  - Include fields for personal preferences, allergies, and special requests.
  - **Deferred:** Allow businesses to customize the form based on service type.
- **Form Submission and Review:**
  - Enable customers to submit the form online before appointments.
  - Allow staff to review and update form details as needed.
  - Store form submissions securely with customer profiles.

8.3 Rating/Review System

- **Simple Rating System:**
  - Allow customers to rate services and staff members on a 5-star scale.
  - Display average ratings prominently on service and staff profiles.
  - Enable businesses to respond to customer reviews.
- **Review Moderation [Deferred]:**
  - Implement moderation tools to filter inappropriate content.
  - Allow flagged reviews to be reviewed by admin before publication.
  - Notify customers when their review is published or flagged.

8.4 Upcoming Appointments

- **Appointment Dashboard:**
  - Provide a dashboard for customers to view upcoming appointments.
  - Include options to modify, cancel, or reschedule appointments.
  - Display appointment details such as date, time, services, and staff.
- **Calendar Integration [Deferred]:**
  - Allow customers to sync appointments with personal calendars (e.g., Google Calendar).
  - Send calendar invites with appointment details and reminders.
  - Provide options to update calendar entries upon appointment changes.

9 Business Dashboard

9.1 Appointment Calendar View [Deferred]

- **Interactive Calendar:**
  - Provide an interactive calendar view for businesses to manage appointments.
  - Display appointments in a daily, weekly, or monthly view.
  - Allow color-coding of appointments based on status (e.g., confirmed, pending).
- **Drag-and-Drop Rescheduling :**
  - Enable drag-and-drop functionality for easy rescheduling of appointments.
  - Automatically update appointment details and notify customers of changes.
  - Implement conflict detection to prevent overlapping appointments.

9.2 Daily Schedule View [Deferred or Not Included]

- **Staff Schedules:**
  - Display individual staff schedules with assigned appointments and breaks.
  - Allow staff to update their availability directly from the dashboard.
  - Highlight any gaps or overbooked slots for quick adjustments.
- **Service Overview:**
  - Provide a summary of services scheduled for the day, including customer details.
  - Allow businesses to print or export daily schedules for offline access.
  - Include quick access links to customer profiles and service details.

9.3 Basic Customer List [Deferred]

- **Customer Directory:**
  - Maintain a directory of all customers with key information such as contact details and service history.
  - Enable search and filter options for quick retrieval of customer data.
  - Allow businesses to add notes or tags to customer profiles for personalized service.
- **Customer Engagement:**
  - Provide tools for sending personalized messages or promotions to customers.
  - Track customer interactions and engagement metrics.
  - Enable integration with CRM systems for advanced customer management.

9.4 Simple Service Management Interface [Deferred]

- **Service Catalog Management:**
  - Allow businesses to add, edit, or remove services from their catalog.
  - Provide tools for updating service descriptions, pricing, and availability.
  - Implement bulk editing features for managing multiple services at once.
- **Performance Analytics:**
  - Display analytics on service performance, including booking trends and revenue.
  - Highlight top-performing services and identify areas for improvement.
  - Allow businesses to export analytics reports for further analysis.

10 Community Page

The Community Page within the BEAUT AI app serves as a dedicated social platform for users to engage with medical aesthetic treatments, businesses offering these services, and other members. Users can share personal experiences, posts, and photos about medical aesthetic treatments, interact with businesses, and explore beauty-related content. This section will include features for discussions, sharing content, user engagement, and business visibility.

10.1 User Profiles

- **Customizable User Profiles:**
  - Each user can create and maintain a personalized profile showcasing their treatment preferences, past medical aesthetic treatments, and favorite businesses.
  - Profiles can include a photo, short bio, and a gallery of shared medical aesthetic treatment experiences (images, videos, reviews).
  - Users can set privacy settings for each profile section (e.g., whether their shared content is public or restricted to certain users).
- **Past Treatments and Businesses:**
  - Display a history of medical aesthetic treatments that users have received, including ratings and reviews.
  - Business names and profiles providing treatments will be linked to posts or reviews made by users.
  - Allow users to tag businesses in their posts to improve visibility and business engagement.

10.2 Sharing Content and Interaction

- **Treatment Posts:**
  - Users can share posts that include images, videos, and descriptions of the medical aesthetic treatments they have received or are recommending.
  - Posts will allow users to mention specific medical aesthetic treatments (e.g., facials, skin rejuvenation, body contouring), and link them to businesses offering these services.
  - Each post can include user-generated content like ”Before & After” images to show the effectiveness of treatments.
- **Business Visibility:**
  - When a user posts about a treatment, the business that provided the treatment is automatically tagged and visible in the post.
  - Businesses can claim their profiles to showcase their services, pricing, location, and customer reviews directly on the posts.
  - Users can follow businesses to receive updates on promotions, new treatments, or special offers shared by the business on the community page.
- **Likes, Comments, and Shares:**
  - Users can like, comment, and share other users’ posts.
  - Each post will have an option to add comments, allowing users to interact and discuss treatments, businesses, or medical aesthetic topics.
  - Shared posts allow users to spread content to their followers or in relevant community groups (e.g., treatment seekers, beauty enthusiasts).
- **Hashtags and Categories:**

  - Users can add hashtags to their posts to categorize treatments or trends (e.g., #SkinRejuvenation, #BodyContouring).
  - Posts with common hashtags will appear in trending sections, making it easier for users to discover popular treatments or trending topics.

10.3 Interaction with Businesses

- **Business Profiles:**
  - Businesses offering medical aesthetic treatments can create detailed profiles showcasing their services, images, reviews, pricing, and available slots.
  - Businesses can interact with customers by responding to comments, reviews, and questions posted by users.
  - Provide businesses with the ability to post updates, promotions, or special offers that will be visible to users.
- **Appointment Booking Integration:**
  - Directly link from business posts or profiles to booking pages where users can schedule appointments for medical aesthetic treatments.
  - Businesses can display availability and offer discounts or promotions through the community platform.
- **User Reviews and Ratings:**
  - Users can rate and leave feedback on businesses and treatments they've experienced.
  - Review sections will include star ratings, textual feedback, and the ability to upload pictures of treatments.
  - Businesses can reply to reviews, addressing customer feedback or thanking users for their comments.

10.4 Content Discovery and Trending Topics

- **Explore Section:**
  - A curated "Explore" section allows users to discover the latest medical aesthetic trends, new treatments, and top businesses based on their interests.
  - Use personalized recommendations to highlight posts and businesses based on a user's activity, preferences, and engagement history.
- **Trending Posts and Hashtags:**
  - Trending posts will appear on the main feed, showcasing the most popular or highly engaged content, based on likes, shares, and comments.
  - Trending hashtags will allow users to follow specific medical aesthetic trends, services, or treatment types (e.g., #AntiAging, #FacialRejuvenation).
- **Follow Users and Businesses:**
  - Users can follow other community members who share interesting content or have valuable insights.
  - Businesses can be followed for updates about new services, promotions, or events.

10.5 Event and Webinar Integration

- **Medical Aesthetic Events:**
  - Users can participate in community-hosted medical aesthetic events, webinars, or live Q&A sessions with beauty experts and professionals.
  - Events can be promoted via posts and business profiles, with details on how to register, topics covered, and speakers.
- **Live Streaming:**
  - Businesses can stream live medical aesthetic tutorials, product demonstrations, or customer reviews directly on the community page.
  - Users can interact with live sessions by commenting, asking questions, and engaging in real-time.

10.6 Privacy and Moderation

- **Content Moderation:**
  - All posts and comments are subject to moderation to ensure the content is appropriate and follows community guidelines.
  - Users can report inappropriate or harmful content that will be reviewed by administrators.
- **Privacy Controls:**
  - Users can manage the visibility of their content and choose to share posts only with selected followers, specific groups, or keep them private.
  - Profile privacy settings will allow users to control who can view their shared content, send messages, or comment on their posts.
- **User Blocking and Reporting:**
  - Users can block other members or report content that violates community rules, including spam, offensive language, or misleading information.

10.7 User Engagement and Rewards

- **Badges and Recognition:**
  - Users can earn badges and rewards for participating actively in the community (e.g., ”Top Reviewer,” ”Aesthetic Guru”).
  - Badges can be displayed on user profiles to highlight their contributions to the community.
- **Incentives for Content Creators:**
  - Reward users who consistently post high-quality content with exclusive benefits, such as discounts on treatments or access to VIP events.
  - Offer incentives like referral bonuses or points for sharing content that leads to bookings or customer referrals.

10.8 Prerequisite Functionalities for Community Page Development

- **User Authentication and Profile Management:**
  - Implement secure user authentication using `OAuth 2.0`, `JWT (JSON Web Tokens)`, or `Session-based Authentication`.
  - Use frameworks like `Firebase Authentication`, `Passport.js` (for Node.js), or `Django Allauth` (for Python).
  - Develop customizable user profiles stored in a database like `MongoDB`, `PostgreSQL`, or `MySQL`.
  - Use libraries like `Cloudinary` or `Amazon S3` for secure media storage (e.g., profile photos).
  - Integrate role-based access controls (`RBAC`) for privacy level management.
- **Content Posting and Sharing System:**
  - Use a rich text editor such as `Quill.js` or `TinyMCE` for creating posts.
  - Store multimedia content securely using `AWS S3`, `Google Cloud Storage`, or `Cloudinary`.
  - Implement tagging with a database schema that supports many-to-many relationships (e.g., in `PostgreSQL` or `MongoDB`).
  - Use `GraphQL` or `RESTful APIs` to handle media uploads and linking posts to businesses.
- **Interaction Features:**
  - Enable liking and commenting using real-time databases like `Firebase Firestore` or `Redis`.
  - Implement discussion threads with hierarchical data models in `NoSQL` or tree structures in `SQL`.
  - Develop notification systems using `WebSockets` (e.g., `Socket.IO`) or third-party services like `OneSignal`.
- **Business Integration and Profiles:**
  - Use `Google Maps API` or `Mapbox` for business location integration.
  - Store business information and reviews in a relational database such as `PostgreSQL` or `MySQL`.
  - Implement authentication and verification for businesses using `SendGrid` or `Twilio`.
  - Enable interactions with user posts and reviews using APIs with `GraphQL` or `REST`.
- **Content Discovery and Categorization:**
  - Build an explore section powered by recommendation algorithms using `Python` libraries like `scikit-learn` or `TensorFlow`.
  - Store hashtags and trending data in a NoSQL database like `MongoDB` or `Elasticsearch`.
  - Implement personalized feeds using `Collaborative Filtering` or `Content-based Filtering`.
- **Event and Webinar Support:**
  - Use `WebRTC` for live streaming capabilities or integrate platforms like `Zoom API`.
  - Enable event scheduling and tracking using `Google Calendar API` or `Microsoft Graph API`.
  - Store event details and user participation data in a relational database such as `PostgreSQL`.
- **Privacy and Moderation Tools:**
  - Implement content moderation using `AI-based tools` like `AWS Rekognition` or `Google Vision API` for detecting inappropriate images.
  - Add user reporting and blocking functionalities with flagging systems managed in `Redis` or `MongoDB`.
  - Enable privacy settings using a rules engine with `Spring Security` (Java) or `Express Middleware` (JavaScript).
- **User Engagement and Rewards System:**
  - Develop a badge system using `DynamoDB` for quick reads and writes or `Firestore` for scalable storage.
  - Create incentive programs integrated with payment gateways like `Stripe API` or `PayPal SDK`.

11 Technical Requirements

11.1 Mobile-Responsive Web Application

- Optimized for various screen sizes (desktop, tablet, and mobile).
- Cross-browser compatibility (Chrome, Safari, Firefox, Edge).
- Performance-focused design with a load time under 3 seconds.

11.2 Expo Mobile App Requirements

- Platform Support:
  - iOS (support for iOS 12 and above).
  - Android (support for Android 8.0 and above).
- Responsiveness:
  - Seamless layout scaling for various screen sizes and resolutions.
  - Device orientation support (portrait and landscape).
- API Integration:
  - Secure integration with backend services via GraphQL.
  - Real-time updates for appointment status and notifications.
- Notifications:
  - Push notifications using Expo's notification service.
  - Configurable reminders for appointments and key updates.
  - Push notification settings for users to manage their preferences.
- Offline Functionality:
  - Basic offline support for critical features (e.g., appointment viewing, service browsing).
  - Synchronization of data once the device reconnects to the internet.
- Expo-Specific Features:
  - Built-in support for Expo modules like secure storage for sensitive data.
  - Integration with Expo Updates for over-the-air updates.

11.3 Basic Security Implementation

- Input validation for all user inputs to prevent common vulnerabilities like SQL injection or XSS.
- Authentication tokens (JWT or similar) to maintain secure sessions.
- Basic data encryption (HTTPS for data in transit, encryption for sensitive data at rest).

11.4 Email Integration for Notifications

- Automated email notifications for sign-up confirmations, password resets, and appointment reminders.
- Customizable email templates to maintain brand consistency.
- Error-handling mechanisms for failed email deliveries.

11.5 Error Handling and Logging

- Centralized logging for backend and mobile errors (e.g., using Sentry or similar services).
- User-friendly error messages for known issues.
- Detailed logs for developers to debug and address technical problems effectively.

12 Database Schema and Requirements

12.1 Core Database Schema

```sql
-- Users Table
CREATE TABLE users (
    id UUID PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    user_type ENUM('customer', 'business', 'admin') NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    status ENUM('active', 'inactive', 'suspended') DEFAULT 'active'
);

-- Business Profiles
CREATE TABLE businesses (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    business_name VARCHAR(255) NOT NULL,
    description TEXT,
    address_line1 VARCHAR(255),
    address_line2 VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    postal_code VARCHAR(20),
    country VARCHAR(100),
    phone VARCHAR(20),
    email VARCHAR(255),
    website VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Services
CREATE TABLE services (
    id UUID PRIMARY KEY,
    business_id UUID REFERENCES businesses(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    duration_minutes INTEGER NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    category VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

--user's preferences
CREATE TABLE usersPreferences (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id), -- Assuming a 'users' table exists
    service_id UUID REFERENCES services(id), -- Links preference to a specific service (optional)
    prompt_id INTEGER, -- To identify which prompt the preference relates to (e.g., 1, 2, 3)
    preference_value VARCHAR(255) NOT NULL, -- Stores the selected option, e.g., 'Acne and
        Blemishes'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


** Community page specific tables
-- Posts Table
CREATE TABLE posts (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id), -- User who created the post
    business_id UUID REFERENCES business_profiles(id), -- Associated business providing the
        treatment
    content TEXT NOT NULL, -- Text content of the post
    image_url VARCHAR(255), -- Optional image URL (e.g., treatment image)
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Comments Table
CREATE TABLE comments (
    id UUID PRIMARY KEY,
    post_id UUID REFERENCES posts(id), -- The post the comment belongs to
    user_id UUID REFERENCES users(id), -- User who commented
    content TEXT NOT NULL, -- Comment text
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Likes Table
CREATE TABLE likes (
    id UUID PRIMARY KEY,
    post_id UUID REFERENCES posts(id), -- The post that received the like
    user_id UUID REFERENCES users(id), -- User who liked the post
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Post-Business Association Table
CREATE TABLE post_business_association (
    id UUID PRIMARY KEY,
    post_id UUID REFERENCES posts(id),
    business_id UUID REFERENCES business_profiles(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

12.2 Database Requirements

- **User Data Storage:**
  - Securely store user credentials and personal information.
  - Implement encryption for sensitive data such as passwords and personal identifiers.
  - Ensure compliance with data protection regulations (e.g., GDPR, CCPA).
- **Business Profiles:**
  - Maintain detailed profiles for each business, including contact and operational information.
  - Support dynamic updates to business profiles with audit trails for changes.
  - Enable relational mapping for businesses to their services and appointments.
- **Services Catalog:**
  - Store comprehensive details for each service offered by businesses.
  - Allow for categorization and tagging of services for easy retrieval.
  - Implement versioning for service descriptions and pricing changes.

- **Appointment Records:**
  - Maintain a history of all appointments, including customer and staff involvement.
  - Track appointment statuses and changes over time.
  - Provide analytics on appointment trends and utilization rates.
- **Basic Customer Records:**
  - Store essential customer details and service history.
  - Enable integration with CRM systems for enhanced customer relationship management.
  - Support data export for reporting and analysis purposes.

13 API Specifications

13.1 GraphQL Schema

```
# User Type
type User {
  id: ID!
  email: String!
  userType: UserType!
  profile: UserProfile!
  createdAt: DateTime!
  updatedAt: DateTime!
}

# Business Type
type Business {
  id: ID!
  name: String!
  description: String
  services: [Service!]!
  team: [TeamMember!]!
  operatingHours: [OperatingHours!]!
  location: Location!
}

# Service Type
type Service {
  id: ID!
  name: String!
  description: String!
  duration: Int!
  price: Float!
  category: Category!
  availability: [Availability!]!
}

# Queries
type Query {
  getUser(id: ID!): User
  getBusiness(id: ID!): Business
  getService(id: ID!): Service
  listAppointments(customerId: ID): [Appointment!]!
}

# Community Post Type
type Post {
  id: ID!
  user: User!
  business: Business
  content: String!
  imageUrl: String
  createdAt: DateTime!
  updatedAt: DateTime!
}

# Comment Type
type Comment {
  id: ID!
  post: Post!
  user: User!
  content: String!
  createdAt: DateTime!
}

# Like Type
type Like {
  id: ID!
  post: Post!
  user: User!
  createdAt: DateTime!
}

# Post-Business Association Type
type PostBusinessAssociation {
  id: ID!
  post: Post!
  business: Business!
  createdAt: DateTime!
}

# Queries
type Query {
  getPost(id: ID!): Post
  listPosts: [Post!]!
  getPostComments(postId: ID!): [Comment!]!
  getPostLikes(postId: ID!): [Like!]!
}


# Mutations
type Mutation {
  createUser(email: String!, userType: UserType!): User
  createBusiness(name: String!, description: String): Business
  createService(name: String!, duration: Int!, price: Float!): Service
  createAppointment(input: CreateAppointmentInput!): Appointment
  updateAppointmentStatus(id: ID!, status: AppointmentStatus!): Appointment

** Mutations for Community Page
  createPost(content: String!, imageUrl: String, businessId: ID): Post
  updatePost(id: ID!, content: String, imageUrl: String): Post
  deletePost(id: ID!): Boolean

  createComment(postId: ID!, content: String!): Comment
  updateComment(id: ID!, content: String!): Comment
  deleteComment(id: ID!): Boolean

  createLike(postId: ID!): Like
  deleteLike(id: ID!): Boolean

  associatePostWithBusiness(postId: ID!, businessId: ID!): PostBusinessAssociation
}
```

13.2 API Endpoints and Operations

- **User Management:**
  - `getUser(id: ID!)`: Retrieve user details by ID.

  - `createUser(email: String!, userType: UserType!)`: Create a new user with specified email and type.

- **Business Management:**
  - `getBusiness(id: ID!)`: Fetch business details by ID.
  - `createBusiness(name: String!, description: String)`: Register a new business with name and description.

- **Service Management:**
  - `getService(id: ID!)`: Get service details by ID.
  - `createService(name: String!, duration: Int!, price: Float!)`: Add a new service with specified details.

- **Appointment Management:**
  - `listAppointments(customerId: ID)`: List all appointments for a customer.
  - `createAppointment(input: CreateAppointmentInput!)`: Schedule a new appointment.
  - `updateAppointmentStatus(id: ID!, status: AppointmentStatus!)`: Change the status of an existing appointment.

- **Community Page Management:**
  - `getPost(id: ID!)`: Fetch a specific post by ID.
  - `listPosts`: Retrieve a list of all community posts.
  - `createPost(content: String!, imageUrl: String, businessId: ID)`: Create a new post in the community, optionally associating it with a business.
  - `updatePost(id: ID!, content: String, imageUrl: String)`: Update an existing post's content or image.
  - `deletePost(id: ID!)`: Delete a post from the community.
  - `createComment(postId: ID!, content: String!)`: Add a comment to a post.
  - `updateComment(id: ID!, content: String!)`: Edit an existing comment.
  - `deleteComment(id: ID!)`: Remove a comment from a post.
  - `createLike(postId: ID!)`: Like a post.
  - `deleteLike(id: ID!)`: Remove a like from a post.
  - `associatePostWithBusiness(postId: ID!, businessId: ID!)`: Link a post to a business to indicate which business provided the treatment.

14 Testing Strategy

14.1 Unit Testing

- **Backend Testing:**
  - Use Jest as the primary testing framework for unit tests.
  - Achieve a minimum of 70% code coverage across all modules.
  - Mock external services and dependencies to isolate unit tests.
  - Test individual functions and methods for expected behavior and edge cases.
  - Implement database transaction tests to ensure data integrity.

- **Frontend Testing:**

  - Utilize React Native Testing Library for component testing.
  - Write tests for component rendering, state management, and event handling.
  - Ensure coverage for Redux actions, reducers, and selectors.
  - Test custom hooks and context providers for expected outcomes.

14.2 Integration Testing

- **GraphQL API Integration:**
  - Use libraries like `Apollo Server Testing` or `GraphQL Playground` for testing GraphQL queries and mutations.
  - Validate request and response payloads for all GraphQL operations (queries, mutations, subscriptions).
  - Test authentication and authorization flows for secure GraphQL endpoints (e.g., ensuring proper token handling).
  - Ensure proper error handling, including GraphQL-specific errors (e.g., validation errors, permissions issues).
  - Verify correct response structure and data consistency according to the schema (e.g., ensuring the presence of required fields).
  - Test edge cases by providing invalid data inputs to ensure proper error messages and status codes.
- **Database Integration:**
  - Set up a test database environment for integration tests.
  - Verify data consistency and relationships across tables.
  - Test complex queries and data retrieval operations.
  - Ensure rollback of transactions in case of test failures.

14.3 End-to-End (E2E) Testing

- **Mobile App Testing:**
  - Use Detox or Appium for end-to-end testing of mobile applications.
  - Automate critical user journeys, such as login, navigation, and booking.
  - Test on real devices and simulators/emulators for comprehensive coverage.
  - Ensure cross-platform compatibility and responsive design.
- **Web Testing:**
  - Use Cypress for end-to-end testing of web components.
  - Validate integration between frontend and backend components.
  - Ensure smooth navigation and data flow across pages.
  - Test for performance bottlenecks and loading times.

14.4 Continuous Testing and CI/CD Integration

- **Automated Testing Pipeline:**
  - Integrate testing suites into the CI/CD pipeline for automated execution.
  - Run unit and integration tests on every pull request to catch regressions early.
  - Schedule E2E tests to run on staging environments before deployment.
- **Reporting and Monitoring:**
  - Generate test reports with coverage metrics and test results.
  - Monitor test execution times and identify slow tests for optimization.
  - Set up alerts for test failures to ensure quick response and resolution.

15 Deployment

15.1 Mobile App Distribution

- **App Store Deployment:**
  - Use Expo's build services to generate platform-specific binaries (APK for Android, IPA for iOS).
  - Follow guidelines for submitting apps to the Apple App Store and Google Play Store.
  - Prepare necessary assets and metadata, such as app icons, screenshots, and descriptions.
- **Versioning and Updates:**
  - Implement a versioning strategy for releasing updates and new features.
  - Use Expo's over-the-air (OTA) updates to push minor changes without requiring a full app store release.
  - Ensure backward compatibility with previous app versions.
- **Beta Testing:**
  - Use TestFlight for iOS and Google Play's Internal Testing for Android to conduct beta testing.
  - Gather feedback from beta testers to identify and fix issues before public release.
  - Implement a feedback loop to continuously improve the app based on user input.

15.2 AWS Setup

- **Infrastructure as Code:**
  - Use Terraform or AWS CloudFormation to define and manage infrastructure.
  - Version control infrastructure code alongside application code.
  - Automate provisioning of AWS resources, including EC2, RDS, and S3.
- **Compute Resources:**
  - Deploy application containers using AWS ECS or EKS for orchestration.
  - Configure auto-scaling groups to handle variable traffic loads.
  - Use Elastic Load Balancers (ELB) for distributing incoming traffic.
- **Database Setup:**
  - Use Amazon RDS for MySQL as the primary database service.
  - Enable automated backups and snapshots for data recovery.
  - Configure read replicas for improved read performance and availability.

15.3 Domain Configuration

- **Domain Registration:**
  - Register domain names using AWS Route 53 or another domain registrar.
  - Set up DNS records for mapping domain names to application endpoints.
- **SSL/TLS Implementation:**
  - Use AWS Certificate Manager to provision SSL/TLS certificates.
  - Enforce HTTPS for all web traffic to ensure secure communication.
  - Configure automatic certificate renewal to maintain security compliance.

15.4 CI/CD Pipeline

- **Continuous Integration:**
  - Use Jenkins, GitHub Actions, or AWS CodeBuild for continuous integration.
  - Automate build and test processes for every code commit.
  - Integrate code quality checks and linting into the CI pipeline.
- **Continuous Deployment:**
  - Use AWS CodeDeploy or Jenkins for automated deployment to staging and production environments.
  - Implement blue/green or canary deployments to minimize downtime and risk.
  - Rollback changes automatically in case of deployment failures.

15.5 Monitoring and Logging

- **Monitoring Tools:**
  - Use AWS CloudWatch for monitoring application performance and resource utilization.
  - Set up alerts for critical metrics such as CPU usage, memory, and response times.
  - Integrate with third-party monitoring tools like Datadog or New Relic for enhanced insights.
- **Logging and Auditing:**
  - Use AWS CloudWatch Logs for centralized logging of application events.
  - Implement log rotation and retention policies to manage log data efficiently.
  - Enable auditing of user actions and system changes for security compliance.

16 Development Standards

16.1 Version Control

- **Git Workflow:**
  - Use Git for version control, hosted on GitHub or GitLab.
  - Follow a feature branch workflow to isolate development efforts.
  - Use meaningful commit messages that describe changes clearly.
- **Branching Strategy:**
  - Maintain separate branches for development, staging, and production.
  - Use feature branches for new features and bugfix branches for hotfixes.
  - Merge changes into the main branch only after code review and testing.
- **Pull Request Process:**
  - Require pull requests for all changes to the main branch.
  - Enforce code reviews with at least two approving reviewers.
  - Use automated checks for code quality, linting, and testing.

16.2 Code Quality and Standards

- **Coding Standards:**
  - Follow language-specific style guides (e.g., PEP 8 for Python, AirBnB for JavaScript).
  - Use consistent naming conventions for variables, functions, and classes.
  - Write self-documenting code with clear and descriptive variable names.
- **Code Quality Tools:**
  - Use linters (e.g., ESLint, Pylint) to enforce coding standards.
  - Integrate static analysis tools (e.g., SonarQube) for code quality assessment.
  - Perform regular code refactoring to improve readability and maintainability.

16.3 Testing Requirements

- **Unit Testing:**
  - Achieve a minimum of 70% unit test coverage for all modules.
  - Write tests for all critical paths and edge cases.
  - Use mocking frameworks to isolate unit tests from external dependencies.
- **Integration Testing:**
  - Test interactions between different components and services.
  - Validate data flow and integration points with external systems.
  - Use test doubles to simulate external service responses.
- **End-to-End Testing:**
  - Automate testing of user flows using tools like Cypress or Selenium.
  - Ensure cross-browser compatibility and responsive design.
  - Test performance and load handling under realistic scenarios.

16.4 Continuous Integration and Deployment

- **CI/CD Pipeline:**
  - Automate build, test, and deployment processes with CI/CD tools.
  - Run automated tests on every code commit to catch regressions early.
  - Deploy to staging environments for manual testing before production release.
- **Environment Management:**
  - Use environment variables to manage configuration settings.
  - Separate configuration for development, staging, and production environments.
  - Automate environment provisioning and teardown for consistency.

17 Infrastructure Details

17.1 AWS Services

- **Compute Services:**
  - Use Amazon EC2 for scalable virtual server instances.
  - Deploy containerized applications using Amazon ECS or EKS.
  - Implement auto-scaling groups to adjust resources based on demand.
- **Storage Services:**
  - Use Amazon S3 for storing static assets and user-uploaded files.
  - Implement lifecycle policies for efficient storage management.
  - Enable versioning and cross-region replication for data redundancy.
- **Database Services:**
  - Use Amazon RDS for MySQL as the primary relational database.
  - Configure automated backups and multi-AZ deployments for high availability.
  - Utilize Amazon DynamoDB for NoSQL storage needs, if applicable.
- **Networking Services:**
  - Use Amazon VPC to create a secure network environment.
  - Implement Elastic Load Balancing (ELB) for distributing traffic.
  - Configure AWS CloudFront as a content delivery network (CDN) for faster content delivery.

17.2 External Services

- **Email Notifications:**
  - Use SendGrid or AWS SES for sending transactional emails.
  - Implement email templates for notifications, confirmations, and alerts.
  - Monitor email delivery rates and handle bounces and complaints.
- **Push Notifications:**
  - Use Expo's push notification service for sending notifications to mobile devices.
  - Implement user preferences for managing notification settings.
  - Track notification delivery and engagement metrics.
- **Monitoring and Logging:**
  - Use AWS CloudWatch for real-time monitoring of application performance.
  - Integrate with third-party tools like Datadog for enhanced monitoring capabilities.
  - Implement centralized logging with AWS CloudWatch Logs or ELK Stack.

17.3 Security and Compliance

- **Access Management:**
  - Use AWS IAM for managing user access and permissions.
  - Implement role-based access control (RBAC) for fine-grained permissions.
  - Enable multi-factor authentication (MFA) for all administrative access.
- **Data Protection:**
  - Encrypt data at rest using AWS KMS or similar encryption services.
  - Use SSL/TLS for encrypting data in transit.
  - Regularly audit and review security policies and configurations.
- **Compliance:**
  - Ensure compliance with industry standards such as GDPR and CCPA.
  - Conduct regular security assessments and vulnerability scans.
  - Maintain documentation and records for compliance audits.

18 Success Metrics

18.1 Performance Metrics

- **App Load Time:**
  - Target an average app load time of less than 3 seconds on mobile devices.
  - Optimize asset delivery using Expo's asset management and caching strategies.
  - Use performance profiling tools to identify and address bottlenecks.
- **API Response Time:**
  - Ensure API response times are consistently under 500 milliseconds.
  - Implement query optimization and efficient data retrieval techniques.
  - Use monitoring tools to track response times and identify slow endpoints.
- **Uptime:**
  - Achieve a service uptime of 99.9% or higher.
  - Use AWS CloudWatch and third-party services for uptime monitoring and alerts.
  - Implement redundancy and failover strategies to minimize downtime.

18.2 Business Metrics

- **User Acquisition:**
  - Target 100 business signups and 1,000 customer signups in the first month.
  - Track acquisition channels and conversion rates using analytics tools.
  - Implement referral programs and promotions to boost signups.
- **Booking Completion Rate:**
  - Aim for an 80% booking completion rate.
  - Analyze drop-off points in the booking process and optimize user experience.

- Use A/B testing to test and refine booking flow enhancements.

- **Customer Retention:**
  - Monitor retention rates and identify factors influencing customer loyalty.
  - Implement loyalty programs and personalized offers to increase retention.
  - Use customer feedback to continuously improve service offerings.

18.3 User Engagement Metrics

- **Active Users:**
  - Track daily, weekly, and monthly active users.
  - Analyze user engagement patterns and identify key usage trends.
  - Use engagement metrics to inform feature development and prioritization.
- **Session Duration:**
  - Monitor average session duration to gauge user engagement.
  - Identify content or features that drive longer sessions.
  - Optimize user interface and content to encourage longer interactions.
- **Feature Usage:**
  - Track usage of key features such as booking, reviews, and messaging.
  - Identify underutilized features and explore reasons for low engagement.
  - Use feature usage data to prioritize updates and enhancements.
- **App Downloads:**
  - Monitor app download numbers from the Apple App Store and Google Play Store.
  - Analyze download trends and the impact of marketing campaigns.
  - Use download data to assess market reach and penetration.

19 Compliance & Accessibility

19.1 Accessibility Standards

- **WCAG Compliance:**
  - Adhere to WCAG 2.1 Level A and AA standards to ensure accessibility for users with disabilities.
  - Implement keyboard navigation and screen reader compatibility across the application.
  - Use semantic HTML and ARIA roles to enhance accessibility features.
- **Mobile Accessibility:**
  - Ensure touch targets are appropriately sized and spaced for mobile users.
  - Support voiceover and screen reader functionalities on both iOS and Android.
  - Implement gesture-based navigation and interactions where applicable.
- **Color Contrast:**
  - Ensure sufficient color contrast between text and background elements.
  - Use tools like the WebAIM Color Contrast Checker to validate compliance.

  - Provide alternative text for images and non-text content.

- **Responsive Design:**
  - Design the application to be fully responsive across devices and screen sizes.
  - Use flexible grid layouts and media queries to adapt to different viewports.
  - Test responsiveness on a range of devices, including mobile, tablet, and desktop.

19.2 Data Protection and Privacy Compliance

- **GDPR Compliance:**
  - Implement mechanisms for obtaining user consent for data collection and processing.
  - Provide users with the ability to access, rectify, and delete their personal data.
  - Maintain records of processing activities and data protection impact assessments.
- **CCPA Compliance:**
  - Allow California residents to opt-out of the sale of their personal information.
  - Provide clear and transparent privacy notices detailing data collection practices.
  - Implement processes for responding to consumer rights requests within mandated timeframes.
- **Data Encryption:**
  - Encrypt sensitive data both at rest and in transit using industry-standard protocols.
  - Use secure key management practices to protect encryption keys.
  - Regularly audit encryption implementations to ensure compliance with best practices.

19.3 Security Measures

- **Vulnerability Management:**
  - Conduct regular vulnerability assessments and penetration testing.
  - Use automated tools to scan for common security vulnerabilities.
  - Implement a patch management process to address identified vulnerabilities promptly.
- **Access Controls:**
  - Implement role-based access control (RBAC) to restrict access to sensitive data and functions.
  - Use multi-factor authentication (MFA) for administrative access.
  - Regularly review and update access permissions based on user roles and responsibilities.
- **Incident Response:**
  - Develop an incident response plan to address security breaches and data incidents.
  - Conduct regular incident response drills to ensure preparedness.
  - Maintain a communication plan for notifying affected parties and regulatory bodies in case of a breach.

20 Not Included in MVP

These features are not essential for the initial launch and can be deferred for future phases. Below is a list with reasons for deferring each feature:

- **Language Toggle**: While multilingual support can broaden the user base, it requires significant translation efforts and additional testing to ensure proper functionality across all supported languages. It can be introduced once the primary features are stable and localized demand is established.
- **Advanced Animations**: While advanced animations enhance user experience, they can increase development time and are not vital to delivering core functionality. Basic, responsive design will ensure usability for MVP.
- **Full HIPAA Compliance**: Achieving full compliance involves rigorous testing, auditing, and infrastructure adjustments, which can be resource-intensive. Basic data protection and secure handling practices will suffice for the MVP to ensure a quick go-to-market strategy.
- **ML/Computer Vision Features**: Implementing advanced features like machine learning or computer vision requires a significant amount of data and development resources. These can be explored in the future when the platform has a robust dataset and user adoption.
- **Advanced Map Integration**: While features like dynamic routing or geolocation can improve user experience, they require third-party integrations and additional testing. Basic location-based search is sufficient for the MVP.
- **Social Features**: Social elements like sharing services or connecting with other users add complexity without directly impacting the core functionality. These can be introduced as value-added features in subsequent updates.
- **Advanced Analytics**: Advanced analytics for user behavior and trends require significant backend support and data infrastructure. For the MVP, basic analytics will provide sufficient insights into user engagement and system performance.

By focusing on core functionality for the MVP, we can ensure timely delivery while keeping the product scalable for future enhancements.