Software Design Specification (SDS)

Project Name: [GlowVibe beauty clinic Web Application]
Prepared By: [Rofida, Menna, Abeer, Sara, Manar]
Date: [30/11/2024]

---

1. Introduction

1.1 Purpose

The purpose of this document is to describe the design, architecture, and technical specifications of the **GlowVibe** platform. It outlines the functionality, system components, and design decisions to be followed during development. The goal is to create a user-friendly and efficient platform that simplifies appointment scheduling and consultation management for beauty services, enhancing the experience for clients and professionals alike.

1.2 Scope

This SDS covers the design and implementation details of the **GlowVibe** platform. The software will perform the following major tasks:

- **User Registration and Profile Management**: Allow clients and professionals to register, log in, and manage their profiles.
- **Appointment Booking and Scheduling**: Enable clients to view available times and book, modify, or cancel appointments with beauty professionals.
- **Service Catalog and Reviews**: Display a categorized list of services, allow clients to filter services by category and price, and enable clients to leave reviews and ratings for professionals.

---

2. System Overview

The system consists of the following components:

- **Frontend:** React for a responsive and interactive user interface.

- **Backend:** Node js for handling server-side logic.

- **Database:** MongoDB to manage customer data, bookings, and service information.

---

3. System Architecture

3.1 Architectural Design

This project follows the [Insert architecture type: e.g., client-server, microservices, monolithic] architecture, where:

- **Frontend** communicates with the backend using [API/Protocol].

- Restful API and HTTPS protocol to ensure secure and reliable data transmission.
- **Backend** interacts with the database to manage and retrieve data.

- the backend interacts with my SQL database to store and manage data related to the users

3.2 Data Flow

1. **User Interaction**: The user interacts with the UI to perform an action (e.g., create a new entry, update an existing record).

   the users interact with the frontend interface by selecting the service, booking an appointment, or viewing a profile

2. **Request Processing**: The frontend sends an API request to the backend server.
   the frontend users send API requests to the backend with any necessary data

3. **Data Handling**: upon receiving the request the backend processes the data validations and then interacts with the database (my SQL), for example: if the user wants to booking will check the availability first and then store the details

4. **Response**: The backend sends the response back to the frontend, updating the UI.

   the backend sends the response to the frontend, providing the result of the action (confirmation or error message)the frontend updates the user UI.

---

4. Database Design

4.1 Database Schema

The database is designed using the **MongoDB NoSQL** model, with collections representing various entities in the system. The schema ensures efficient querying, relationships between collections using references, and optimal storage of hierarchical data.

1. Users Collection

The Users collection stores information about all types of users in the system, including
clients, beauty professionals, doctors, and administrators.

- clientid: A unique identifier for each user. This field is automatically generated by MongoDB as an ObjectId.
- userType: A string that defines the type of user. It can be one of the following: client, beautyProfessional, doctor, or admin. This field is required to determine the role of each user.
- email: A string representing the user's email address. This field is required and must be unique to ensure no duplicate accounts.
- passwordHash: A string that stores the encrypted password for secure authentication. This field is mandatory to ensure user security.
- name: A string containing the full name of the user. This field is optional but recommended for personalized services.
- phone: A string representing the contact phone number of the user. This field is optional and can be used for communication purposes.

2. Appointments Collection
The Appointments collection tracks all the bookings made by clients with beauty professionals or doctors.

- appointmentid: A unique identifier for each appointment, automatically generated by MongoDB as an ObjectId.
- userId: An ObjectId that references the Users collection, linking the appointment to a specific client. This field is required.
- serviceId: An ObjectId that references the Services collection, indicating the service being booked. This field is mandatory.
- appointmentTime: A date and time indicating when the appointment is scheduled. This field is required.
- status: A string representing the current status of the appointment. Possible values include scheduled, canceled, or other custom statuses. This field is required for tracking appointment progress.

3. Services Collection
The Services collection contains detailed information about the various services offered by beauty professionals and doctors.

- serviceid: A unique identifier for each service, generated automatically by MongoDB as an ObjectId.
- professionalId: An ObjectId that references a beauty professional in the Users collection. This field is required to associate services with specific providers.
- serviceName: A string representing the name of the service. This field is mandatory.
- description: A string containing a detailed description of the service. This field is optional, providing more context about the service.
- price: A number indicating the cost of the service in USD. This field is required to display pricing to clients.

4. Reviews Collection
The Reviews collection stores client feedback regarding their experiences with services and professionals.

- reviewid: A unique identifier for each review, auto-generated by MongoDB as an ObjectId.
- userId: An ObjectId referencing the Users collection, identifying the client who submitted the review. This field is required.
- professionalId: An ObjectId referencing the Users collection to identify the reviewed beauty professional. This field is required.
- rating: A numeric rating given by the client, typically between 1 and 5. This field is required to provide quantitative feedback.
- comment: A string containing textual feedback from the client. This field is optional but useful for qualitative insights.

5. Products Collection
The Products collection stores information about products available for sale at the clinic.

- productid: A unique identifier for each product, generated by MongoDB as an ObjectId.
- productName: A string representing the name of the product. This field is required for product identification.
- description: A string providing a detailed description of the product. This field is optional but helpful for informing clients.
- price: A number indicating the product's cost in USD. This field is required for displaying pricing to customers.
- stockQuantity: A number representing the available stock for the product. This field is mandatory to manage inventory effectively.

Relationships :

- **Users ↔ Appointments**: One-to-Many
  A user can have multiple appointments, but each appointment belongs to a single user.
- **Users ↔ Services**: One-to-Many
  A beauty professional or doctor can offer multiple services, but each service belongs to one professional.
- **Users ↔ Reviews**: One-to-Many
  A client can write multiple reviews, and each review is tied to one service provider.
- **Appointments ↔ Services**: Many-to-One
  Each appointment involves a single service, but a service can be booked in multiple appointments.

---

5. Technology Stack

- **Frontend**: React.
- **Backend**: Node.js.
- **Database**: MongoDB.
- **Hosting**: Azure, heroku.

---

6. Testing Plan

6.1 Unit Testing

Each module and function will undergo unit testing to ensure that individual components are working as expected.

Tools: Jest for frontend (React), Pytest for any Python-based backend components, or Mocha for Node.js.
test individual UI components, backend logic, for example: (booking functionality, authentication) and database interactions independently to confirm that every tiny component operates as planned.
Testing will continue during development and after every significant code update.

6.2 Integration Testing

Integration tests will validate that different modules (frontend and backend, or backend and database) work together as expected.
We will Focus on scenarios where multiple components must work together, such as when a user books an appointment (frontend requests the backend, backend retrieves data from the database
Tools: Postman for API endpoint testing and Supertest for HTTP assertions in Node.js.
We will test after each feature or module integration and before any deployment.

6.3 User Acceptance Testing (UAT)

End users will be involved in testing the system to verify that it meets their requirements and expectations.
Test important user processes like registering for an account, scheduling appointments, looking up professional profiles, and giving feedback.
Tools: Test scripts and questionnaires for feedback and Google form for collecting feedback

6.4 Performance Testing

Stress and load testing will be conducted to ensure the system can handle the required number of users and operations without degradation in performance.
Tools: Apache JMeter or Gatling for load testing we will test scenarios with high user concurrency and examine server load, stability, and performance.

---

7. Conclusion

GlowVibe is designed to satisfy the functional and non-functional requirements detailed in this document. Its system architecture ensures that it will be dependable, user-friendly, and capable of adapting to future needs. By simplifying the appointment booking and service management processes, **GlowVibe** aims to offer a valuable and intuitive platform for beauty and wellness clients and professionals.