**Design Document**

**INTRODUCTION**

**Purpose**

The Doctor Appointment Web App aims to provide a user-friendly platform for patients to browse and book appointments with specialized doctors. It also facilitates doctor registration and detailed information submission while enabling administrators to manage hospitals.

**Scope**

The scope includes patient and doctor registration, appointment booking, hospital management, and integration with an external API for fetching nearby hospitals.

**FUNCTIONAL REQUIREMENTS**

**User Authentication**

* Patients, doctors, and administrators must be able to register and log in.

**Patient Features**

* Patients can browse doctors based on specialization, location, and other criteria.
* View doctor profiles, including detailed information about fees, specialization, and availability.
* Book appointments with selected doctors.
* View and manage their appointment history.

**Doctor Features**

* Doctors can register with detailed information, including fees, specialization, and availability.
* Manage their appointment schedule.
* View patient details for upcoming appointments.
* Update their profile information.

**Hospital Management**

* Admins can add, edit, and delete hospitals.
* Admins can associate doctors with specific hospitals.

**External API Integration**

* Integrate with an external API to fetch nearby hospitals based on location.

**User Account Management**

* Users can update their profile information.
* Users can view their booking history.

**NON-FUNCTIONAL REQUIREMENTS**

* Implement secure user authentication and authorization.
* Protect sensitive user data.

**USER TYPES**

There are three types of users:

* Admin
* Doctors
* Patients

**TECHNICAL ARCHITECTURE**

**1. Folder Structure**

Pages Folder

* Login.js: Handles user login.
* Register.js: Manages user registration.
* Home.js: Displays the main dashboard after successful login.
* ApplyDoctor.js: Handles doctor application.
* Notifications.js: Displays notifications with seen/unseen options for all types of users.

Components Folder

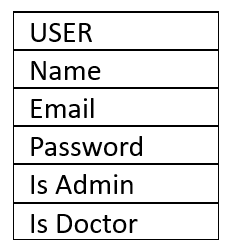
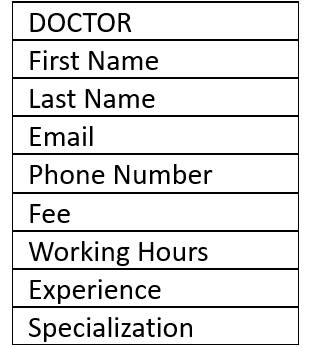
* PublicRoute.js: Redirects to the login page if the user is not authenticated.
* ProtectedRoute.js: Redirects to the home page if the user is authenticated.

Routes Folder

* UserRoute.js: Navigates to different pages based on user roles.
* Handles routes like /register, /login, /get-user-info-by-id.

Models Folder

* UserModel.js: Represents the structure of user data in the database.
* DoctorModel.js: Defines the data model for doctor information.

Config Folder

* dbConfig.js: Manages the configuration for connecting to the MongoDB database.

**2. Backend Structure (server.js)**

* Express Setup: Configures and sets up the Express application.
* Database Connection: Utilizes dbConfig for establishing a connection to MongoDB.
* API Routes: Defines routes for user interactions, such as /api/user.

**3. User Authentication and Authorization**

* JWT Authentication: Generates and verifies JSON Web Tokens for user authentication.
* Role-Based Authorization: Assigns roles (Admin, Doctor, Normal User) to users.

**4. Database (MongoDB)**

* Overview: MongoDB is used to store doctor and users data.
* Collections:

1. Doctors: Stores doctor information.
2. Users: Stores users information.

**5. Components and Modules**

* Authentication Module:Implements user authentication using JWT tokens.

**API DESIGN**

User Authentication and Registration

1. User Registration

* Endpoint: /api/user/register
* Method: POST
* Request Body: { "name": "John", "email": "john@example.com", "password": "secure" }
* Response: Success or Error message

2. User Login

* Endpoint: /api/user/login
* Method: POST
* Request Body: { "email": "john@example.com", "password": "secure" }
* Response: Token and User details or Error message

Doctor Certification

Apply for Doctor

* Endpoint: /api/user/apply-doctor
* Method: POST
* Request Body: Doctor details
* Response: Success or Error message

Notification Management

Get Notifications

* Endpoint: /api/user/notifications
* Method: GET
* Response: List of Notifications

**DEVELOPMENT ENVIRONMENT**

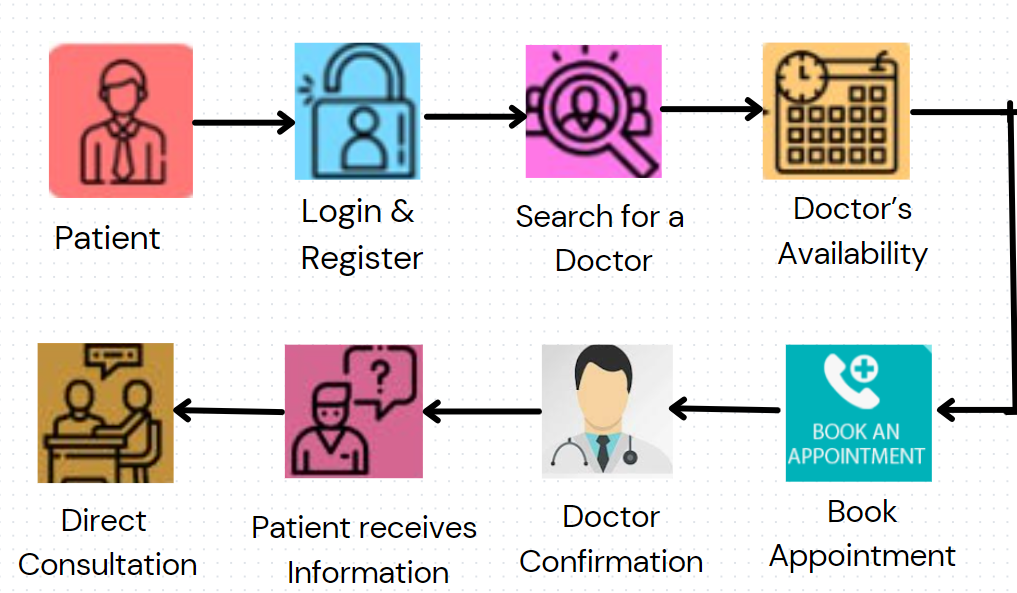
* Version Control: Git for version control.

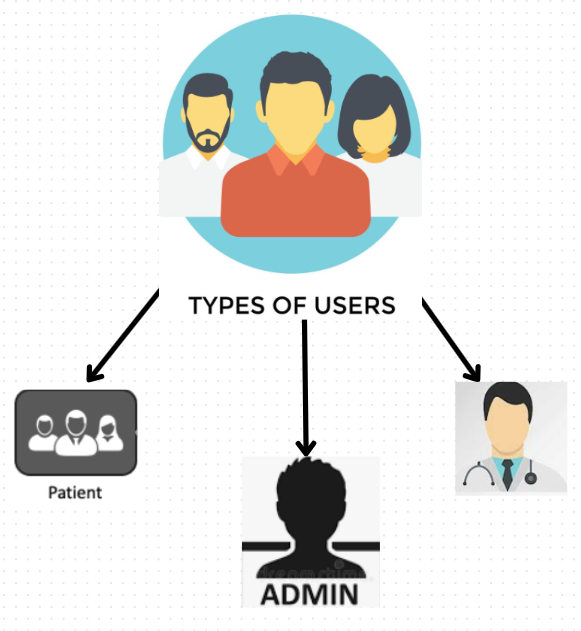
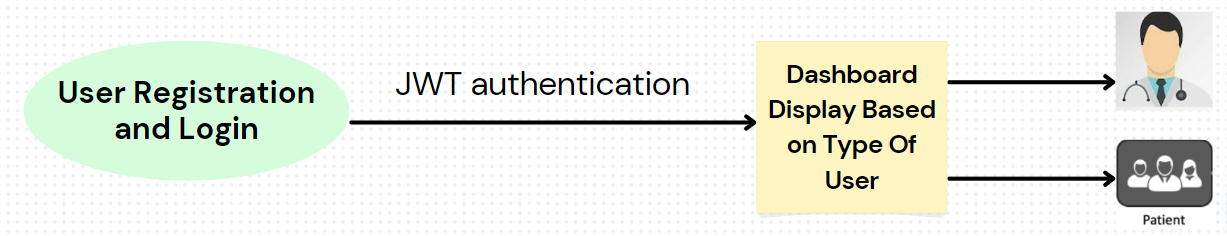
**DEPENDENCIES**

Frontend Dependencies : react, axios, react-router-dom, react-hot-toast ,.

Backend Dependencies : express, mongoose, jsonwebtoken, nodemon, dotenv, bcryptjs.

**MODEL**



* 
* 
* 