API Documentation

1. Health Record Service API

Overview

The **Health Record Service API** is designed to manage patient health records, including creating, retrieving, updating, and deleting health records. The service leverages MongoDB for data storage and follows RESTful principles.

Base URL

http://<domain>:<port>/api/health-records

Endpoints

a. Create a Health Record

- **URL**: /
- **Method**: POST
- **Description**: Creates a new health record.
- **Request Body**: (JSON)

```
{ "patientId": "string",
  "name": "string",
  "dateOfBirth": "YYYY-MM-DD",
  "prescriptions": [
   { "medication": "string",
     "dosage": "string",
     "startDate": "YYYY-MM-DD",
     "endDate": "YYYY-MM-DD"
     }
   ],
 "labResults": [
{ "testName": "string",
"result": "string",
"date": "YYYY-MM-DD" }
],
"medicalHistory": ["string"] }
```

- Response:
  1. 201 Created: Returns the created health record.
  2. 500 Internal Server Error: If an error occurs.

b. Get All Health Records

- **URL:** /
- **Method:** GET
- **Description:** Retrieves all health records.
- Response:
  1. **200 OK:** Returns an array of health records.
  2. **500 Internal Server Error:** If an error occurs.

c. Get a Health Record by Patient ID

- **URL:** /:patientId
- **Method:** GET
- **Description:** Retrieves a specific health record using the patientId.
- Response:
  1. **200 OK:** Returns the health record.
  2. **404 Not Found:** If the record does not exist.
  3. **500 Internal Server Error:** If an error occurs.

d. Update a Health Record

- **URL:** /:patientId
- **Method:** PUT
- **Description:** Updates a health record using the patientId.
- **Request Body:** (Partial or full record update)

```
{
"name": "string",
"prescriptions": [
 {
"medication": "string",
"dosage": "string",
"startDate": "YYYY-MM-DD",
"endDate": "YYYY-MM-DD"
 }
],
"labResults": [
 {
"testName": "string",
"result": "string",
"date": "YYYY-MM-DD"
 }
],
"medicalHistory": ["string"]
}
```

- Response:
    1. **200 OK:** Returns the updated health record.
    2. **404 Not Found:** If the record does not exist.
    3. **500 Internal Server Error:** If an error occurs.

e. Delete a Health Record

- URL: /:patientId
- **Method:** DELETE
- **Description:** Deletes a health record using the patientId.
- Response:
    1. **200 OK:** Returns a confirmation message.
    2. **404 Not Found:** If the record does not exist.
    3. **500 Internal Server Error:** If an error occurs.

Data Model

```
{ "patientId": "string",
  "name": "string",
  "dateOfBirth": "YYYY-MM-DD",
  "prescriptions": [
   { "medication": "string",
     "dosage": "string",
     "startDate": "YYYY-MM-DD",
     "endDate": "YYYY-MM-DD"
     }
   ],
 "labResults": [
{ "testName": "string",
"result": "string",
"date": "YYYY-MM-DD" }
],
"medicalHistory": ["string"] }
```

Error Handling

The API uses standard HTTP status codes to indicate the success or failure of operations. Common status codes include:

- **201 Created:** Resource successfully created.
- **200 OK:** Request successful.
- **404 Not Found:** Resource not found.
- **500 Internal Server Error:** Server encountered an unexpected error.

Setup Instruction

1. Clone the repository:

   git clone <repository-url>

2. Install dependencies:

   npm install

3. Configure MongoDB connection in *config/db.js.*
4. Start the server:

   npm start

5. Access the API at:

   http://localhost:5000/api/health-records

2. Nearby Hospitals Location

Overview

The **Nearby Hospitals Location API** allows users to find hospitals within a specified radius of a given geographical location (latitude and longitude). It uses geospatial queries to fetch nearby hospital data stored in a MongoDB database.

Base URL

http://< domain>:<port>/api/hospitals

Endpoints

Get Nearby Hospitals

- **URL:** /nearby
- **Method:** GET
- **Description:** Retrieves a list of hospitals within a given radius of specified coordinates.
- **Query Parameters:**
  - **latitude** (required): The latitude of the user's location.

- **longitude** (required): The longitude of the user's location.
- **radius** (optional): The search radius in kilometers (default: 5 km).

- Example Request:
  GET /api/hospitals/nearby?latitude=26.9124&longitude=75.7873&radius=10
- Response:
  1. 200 OK: Returns an array of nearby hospitals

```
[
{
 "name": "Jaipur Medical Center",
 "amenity": "hospital",
 "healthcare": "hospital",
 "address": {
           "city": "Jaipur",
            "street": "MI Road",
            "housenumber": "123",
            "postcode": 302001
           },
 "location": {
           "latitude": 26.9124,
           "longitude": 75.7873
           },

"externalId": "node/2694200840"

 }

]
```

  2. 400 Bad Request: If latitude or longitude is missing.
     { "error": "Latitude and Longitude are required" }
  3. 500 Internal Server Error: If there's a server-side issue.
     { "error": "Server error" }

Data Model

Hospital schema

```
{
 "name": "Jaipur Medical Center",
 "amenity": "hospital",
 "healthcare": "hospital",
 "address": {
           "city": "Jaipur",
            "street": "MI Road",
            "housenumber": "123",
            "postcode": 302001
            },
    "location": {
            "latitude": 26.9124,
            "longitude": 75.7873
            },

   "externalId": "node/2694200840"

    }
```

Error Handling

The API uses standard HTTP status codes to indicate the result of operations:

- **200 OK:** Successful operation.
- **400 Bad Request:** Missing required query parameters.
- **500 Internal Server Error:** Unexpected server error.

Setup Instructions

Prerequisites

- Node.js
- MongoDB (ensure MONGO_URI is set in .env)

Steps

1. Clone the repository:

   git clone <repository-url>

2. Install dependencies:

   npm install

3. Create a .env file and add your MongoDB connection string:

   MONGO_URI=mongodb+srv://<username>:<password>@cluster0.mongodb.net/<database>?retryWrites=true&w=majority

4. Start the server:

   npm start

5. Access the API at:

   http://localhost:3000/api/hospitals

3. JWT Authentication API

Overview

The **JWT Authentication API** provides a robust and secure way to handle user authentication, including user registration, login, logout, and placeholder support for password recovery. It uses JWT tokens for session management and MongoDB for user data storage.

Base URL

http://<your-domain>:<port>/api/auth

Endpoints

1. Register User

- **URL:** /register
- **Method:** POST
- **Description:** Registers a new user in the system.
- **Request Body:**

```
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "password123"
}
```

- **Response:**
  - **201 Created:** User successfully registered.

```
{
  "user": {
    "_id": "64dbf91d8f8b9c001f0e2c88",
    "name": "John Doe",
    "email": "johndoe@example.com",
    "createdAt": "2025-01-22T08:00:00Z",
    "updatedAt": "2025-01-22T08:00:00Z"
  },
  "token": "your_jwt_token" }
```

- **400 Bad Request:** User already exists.

  { "message": "User already exists" }

- **500 Internal Server Error:** Any unexpected error.

  { "message": "Error message" }

2. Login User

- **URL:** /login
- **Method:** POST
- **Description:** Authenticates an existing user.
- **Request Body:**

  ```
  {
    "email": "johndoe@example.com",
    "password": "password123"
  }
  ```

- **Response:**
  - **200 OK:** Successful login.

    ```
    {
      "user": {
        "_id": "64dbf91d8f8b9c001f0e2c88",
        "name": "John Doe",
        "email": "johndoe@example.com"
      },
      "token": "your_jwt_token"
    }
    ```

  - **401 Unauthorized:** Invalid credentials.

    { "message": "Invalid email or password" }

3. Logout User

- **URL:** /logout

- **Method:** POST
- **Description:** Logs the user out.
- **Response:**

  { "message": "Logout successful" }

4. Forgot Password (Stub)

- **URL:** /forgot-password
- **Method:** POST
- **Description:** Placeholder for password recovery functionality.
- **Response:**

  { "message": "Forgot password functionality not implemented yet" }

Data Model

User Schema

```
{
  "name": "string",
  "email": "string",
  "password": "string (hashed)",
  "createdAt": "date",
  "updatedAt": "date"
}
```

Authentication Middleware

Protect Route

The protect middleware checks for a valid JWT in the Authorization header. If valid, the user object is added to req.user. Otherwise, it returns:

- **401 Unauthorized:** Missing or invalid token.

  { "message": "Not authorized, token failed" }

JWT Token Utility

Generate Token

The generateToken function creates a JWT token for authenticated users. Tokens expire in **1 hour** by default.

Setup Instructions

Prerequisites

- Node.js
- MongoDB (ensure MONGO_URI and JWT_SECRET are set in .env)

Steps

1. Clone the repository:

   git clone <repository-url>

2. Install dependencies:

   npm install

3. Create a .env file and add the following:

   MONGO_URI=mongodb+srv://<username>:<password>@cluster0.mongodb.net/<database>?retryWrites=true&w=majority

   JWT_SECRET=your_jwt_secret

4. Start the server:

   npm start

5. Access the API at:

   http://localhost:5000/api/auth