Health Metric Tracker
Service Layer Design

1. Overview

- This document outlines the service layer (API endpoints) required for the Health Metric Tracker and Analysis application.
- The endpoints defined here are designed to support the interactions of patients, caregivers, and doctors with the underlying PostgreSQL database via the web interface.
- All endpoints follow RESTful principles, using appropriate HTTP verbs and message structures.

2. API Endpoints Overview

HTTP Verb	Endpoint Path	Description	Purpose (Persona-Based)
POST	/api/auth/login	User login	Email/password-based login for all user types (patients, caregivers, doctors).
GET	/api/users/:id	Get user profile	Fetch profile details based on user ID.
POST	/api/health-metrics	Add health metric	Patients log a new health metric (e.g., BP, glucose).
GET	/api/users/:id/health-metrics	View health metrics	Caregivers/doctors view a user's metric history.
POST	/api/medications	Add medication entry	Patients log when medication was taken.
GET	/api/users/:id/medications	Get medication history	View medication logs for a patient.

POST	/api/alerts	Create alert	System or caregiver manually creates an alert.
GET	/api/users/:id/alerts	Get user alerts	View active/resolved alerts.
PUT	/api/alerts/:id/resolve	Resolve alert	Resolve an active alert.
POST	/api/notes	Add caregiver note	Caregiver adds a note about a patient.
GET	/api/users/:id/notes	Get patient notes	Retrieve all notes about a patient.
POST	/api/reminders	Create reminder	Set a reminder for follow-ups or medication.
GET	/api/users/:id/reminders	Get reminders	Retrieve scheduled reminders.

3. Sample Requests and Responses

POST /api/health-metrics

- Successful Request Example:
  - *{"user_id": "abc123", "metric_type": "BP", "value": "140/90", "timestamp": "2025-03-21T10:00:00Z"}*
- Successful Response Example:
  - *{"message": "Health metric recorded successfully.", "id": "metric789"}*
- Error Response Example:
  - *{"error": "Invalid input: missing required fields."}*

GET /api/users/:id/health-metrics

- Successful Request Example:
  - *{}*
- Successful Response Example:
  - *[{"metric_type": "BP", "value": "130/85", "timestamp": "2025-03-19T08:00:00Z"}]*
- Error Response Example:
  - *{"error": "User not found."}*

PUT /api/alerts/:id/resolve

- Successful Request Example:
  - *{}*
- Successful Response Example:
  - *{"message": "Alert resolved successfully."}*
- Error Response Example:
  - *{"error": "Alert already resolved or not found."}*

4. Page-to-Endpoint Mapping

- [Login Page] ----------> POST /api/auth/login
- [Log Data Page] ----------> POST /api/health-metrics
  GET /api/users/:id/medications
  POST /api/medications
- [Reports Page] ----------> GET /api/users/:id/health-metrics
  GET /api/users/:id/medications
- [Alerts Page] ----------> GET /api/users/:id/alerts
  PUT /api/alerts/:id/resolve
  POST /api/alerts
- [Add Note Popup] ----------> POST /api/notes
- [Caregiver Dashboard] ----------> GET /api/users/:id/notes
  GET /api/users/:id/health-metrics
  [Symptom Log Page] ----------> POST /api/symptoms (Stretch Feature)
  [Admin Assignment Page] ----------> POST /api/assignments (Stretch Feature)
- [Reminder Setup] ----------> POST /api/reminders
  GET /api/users/:id/reminders

1.1 Backend Architecture (Route-Controller-Service)

The backend of the Health Metric Tracker application is built using Node.js with Express.js and follows a modular route-controller-service pattern for clean separation of concerns:

- Route Layer: Defines all RESTful endpoint paths and HTTP methods. These are the entry points for frontend-to-backend communication.
- Controller Layer: Handles request validation, extracts parameters, and calls the appropriate service functions. This is where HTTP logic and error handling live.
- Service Layer: Contains core business logic and interacts with the PostgreSQL database. It ensures that all database operations are isolated from the controller logic.

  This structure allows for easier testing, maintainability, and scalability while keeping each layer focused on its responsibility.

(Made using Mermaid.js)

Figure 1: Login Page API Mapping

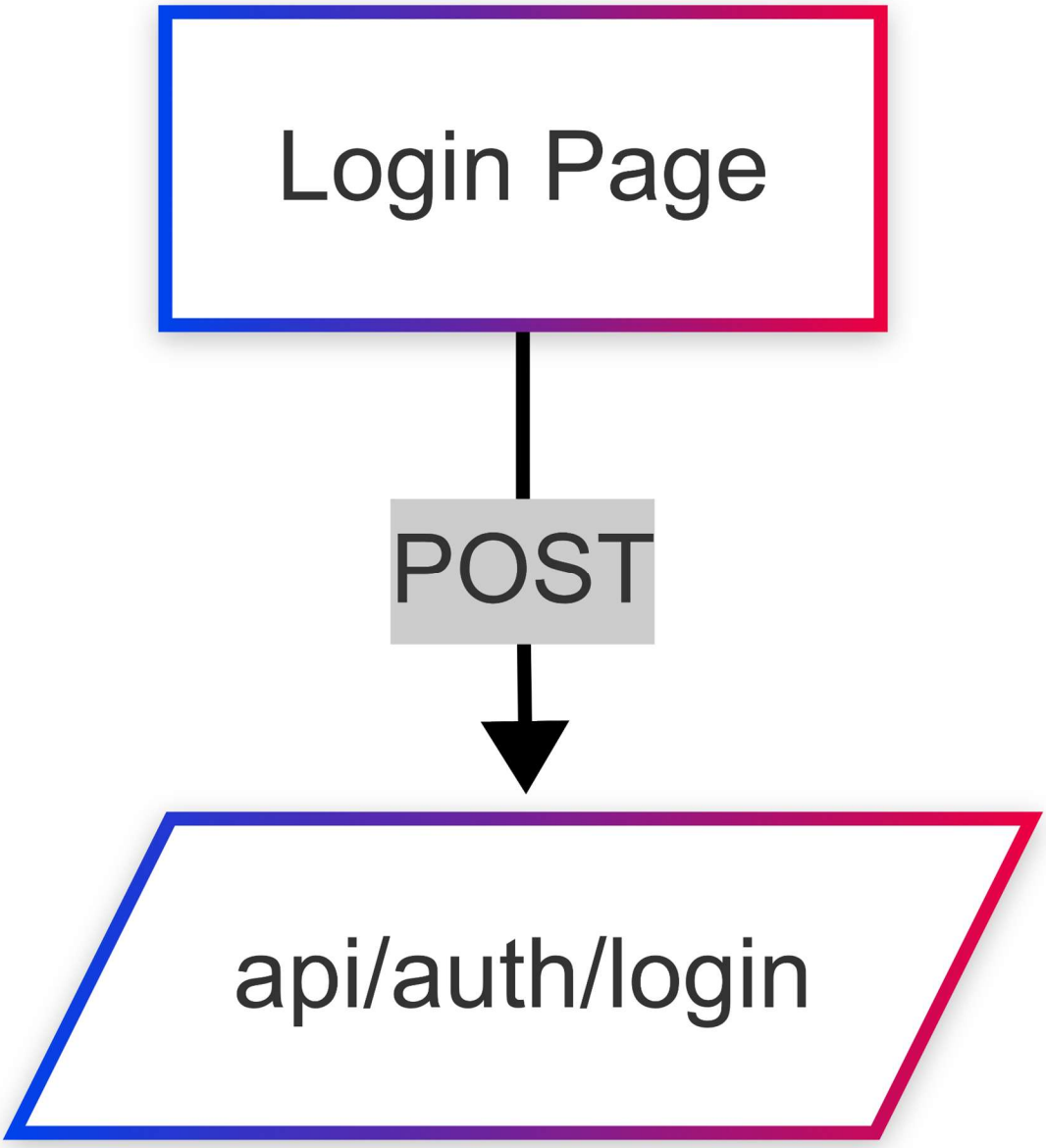


Figure 2: Log Data Page API Mapping

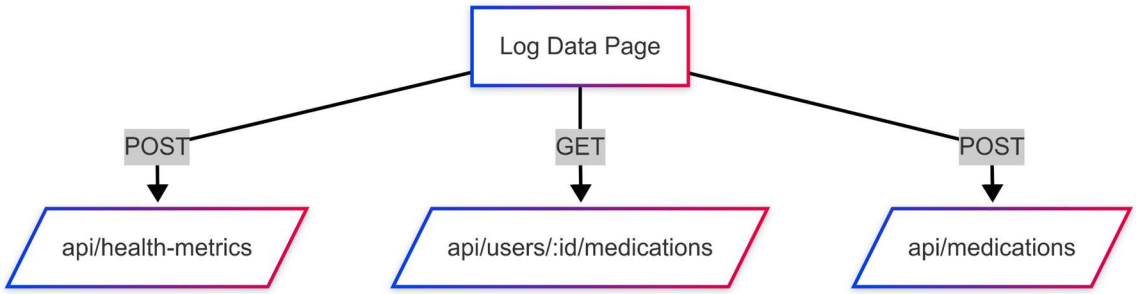


Figure 3: Reports Page API Mapping

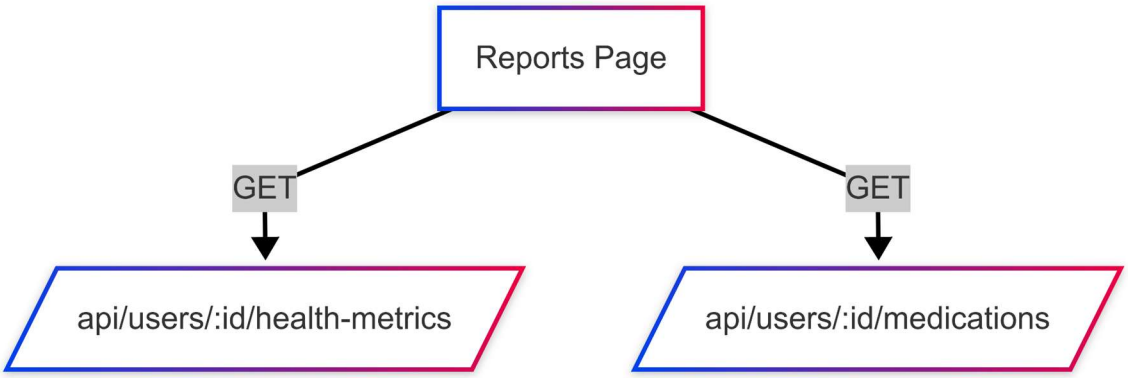


Figure 4: Alerts Page API Mapping

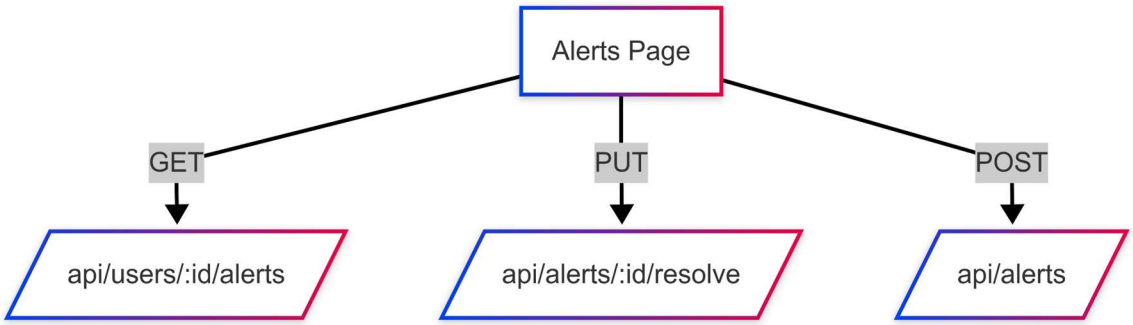


Figure 5: Caregiver Notes API Mapping

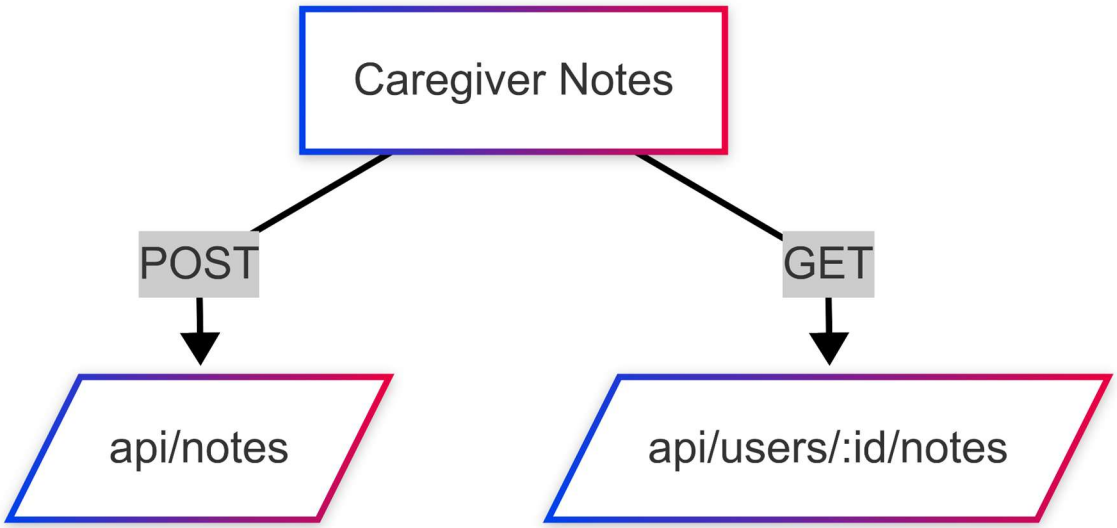


Figure 6: Reminder Setup Page API Mapping

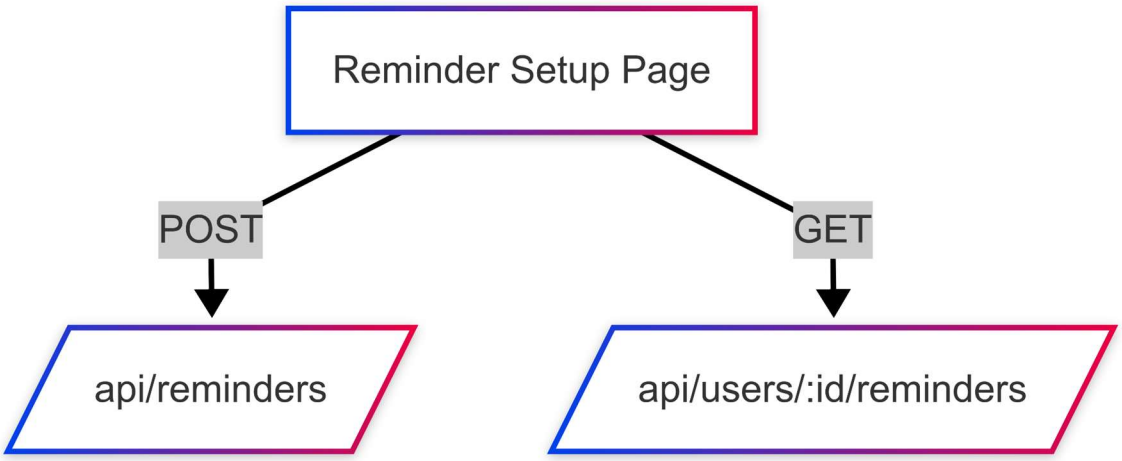


Figure 7: Health Metric ENUM Validation Sequence

This sequence diagram illustrates how the backend validates the `metric_type` field against a PostgreSQL ENUM before inserting into the `health_metrics` table.

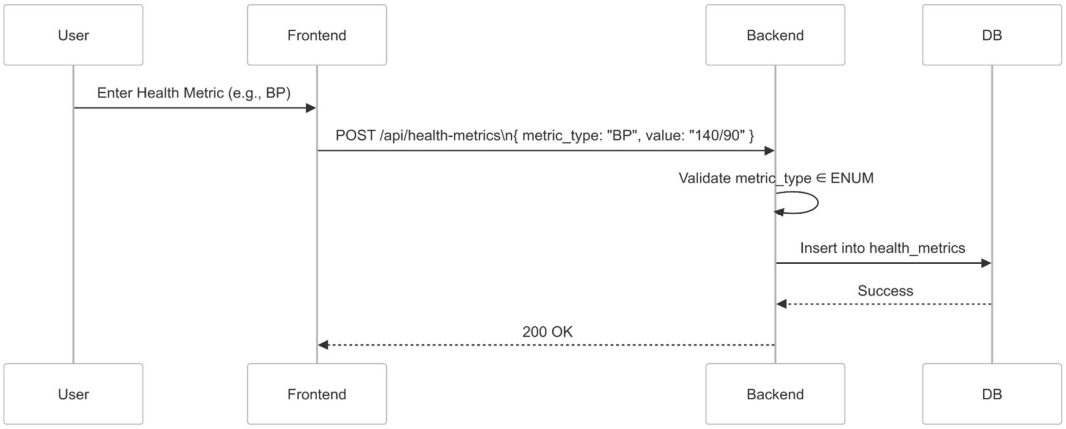
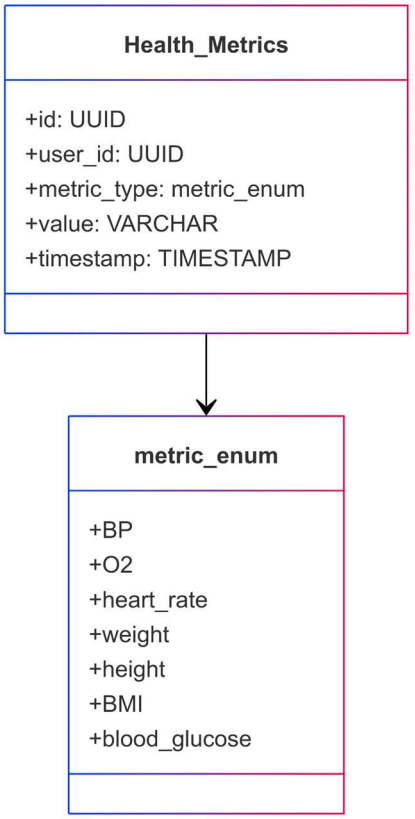


Figure 8: ENUM Class Mapping

This class diagram illustrates how the `Health_Metrics` table uses the `metric_enum` ENUM type to restrict valid input types. The backend logic uses this mapping to validate user inputs during POST requests.



5. Stretch Feature Endpoints

POST /api/symptoms

Purpose: Patients can log symptoms (e.g., nausea, fatigue) to add contextual data alongside health metrics.

Successful Request Example:

```
{
 "user_id": "abc123",
 "symptom_type": "nausea",
 "severity": 6,
 "notes": "Felt queasy after breakfast",
 "timestamp": "2025-03-27T09:15:00Z"
}
```

Successful Response Example:

```
{
 "message": "Symptom logged successfully.",
 "id": "symptom456"
}
```

Error Response Example:

```
{
 "error": "Missing required fields: symptom_type"
}
```

POST /api/assignments

Purpose: Admins or doctors assign caregivers to patients for access to notes, metrics, and reminders.

Successful Request Example:

```
{
 "caregiver_id": "caregiver789",
 "patient_id": "patient123",
 "assigned_on": "2025-03-27T12:00:00Z"
}
```

Successful Response Example:

```
{
 "message": "Caregiver assigned to patient.",
```

```
  "assignment_id": "assign101"
}
```


Error Response Example:


```
{
  "error": "Assignment already exists or user not found."
}
```