Rahati Platform API Design Documentation

Rahati Platform Team

March 22, 2025

Contents

1 Overview

The Rahati Platform API is designed as a RESTful service accessible via HTTPS. This document outlines a high-level API design covering resource endpoints, authentication mechanisms, error handling, security considerations, and example flows. The base URL for all API endpoints is:

`https://api.rahati.com/v1/`

All requests and responses use the JSON format. Versioning is handled via the URL path (e.g., `v1`).

2 Authentication & Authorization

2.1 Login Endpoint

Endpoint: `POST /auth/login`

Request Body:

```
{
  "email": "user@example.com",
  "password": "secret"
}
```

Response:

```
{
  "token": "JWT_OR_BEARER_TOKEN",
  "userId": 123,
  "role": "Patient"
}
```

Subsequent requests require the token in the `Authorization` header:

```
Authorization: Bearer JWT_OR_BEARER_TOKEN
```

User roles include `Patient`, `Provider`, and `Admin`.

3 Endpoints

Each resource supports standard RESTful methods (`GET`, `POST`, `PUT/PATCH`, `DELETE`). Below is a high-level listing.

3.1 1. Users

- `GET /users`: (Admin-only) Retrieve a list of users.
- `GET /users/{id}`: Retrieve a specific user (accessible by Admin or the user themself).

- `POST /users`: Create a new user. (Registration for patients or Admin creation for other roles.)
- `PUT /users/{id}`: Update user details (e.g., name, phone, address, caregiver info).
- `DELETE /users/{id}`: Delete or deactivate a user (Admin-only).

Example: Creating a new patient.

```
POST /users
Content-Type: application/json

{
  "name": "Alice",
  "email": "alice@example.com",
  "password": "secret",
  "role": "Patient",
  "phone": "555-1234",
  "address": "123 Main St",
  "caregiver_name": "Bob",
  "caregiver_phone": "555-5678"
}
```

3.2 2. Centers

- `GET /centers`: List all healthcare centers.
- `GET /centers/{id}`: Retrieve details of a specific center.
- `POST /centers`: Create a new center (Admin-only).
- `PUT /centers/{id}`: Update center information (Admin-only).
- `DELETE /centers/{id}`: Remove a center (preferably soft delete, Admin-only).

3.3 3. Appointments

- `GET /appointments`:
  - Patients see their own appointments.
  - Providers see appointments assigned to them.
  - Admins can see all appointments.
- `GET /appointments/{id}`: Retrieve details for a specific appointment.
- `POST /appointments`: Create a new appointment.
- `PUT /appointments/{id}`: Update an appointment (for changes, reassignments, etc.).
- `DELETE /appointments/{id}`: Cancel or delete an appointment.

Example: Booking an appointment.

```
POST /appointments
Content-Type: application/json

{
  "patientId": 123,
  "centerId": 10,
  "appointmentDatetime": "2025-05-01T09:00:00Z",
  "appointmentDuration": 60
}
```

3.4 4. Consultations

- `GET /consultations`: List consultations (filtered by provider or appointment).
- `GET /consultations/{id}`: Retrieve a specific consultation's details.
- `POST /consultations`: Create a consultation record (typically when an appointment starts).
- `PUT /consultations/{id}`: Update consultation details (e.g., add provider notes, mark end time).
- `DELETE /consultations/{id}`: Delete a consultation record (Admin-only).

3.5 5. Payments

- `GET /payments`: List payments (patients see their own; Admin sees all).
- `GET /payments/{id}`: Retrieve details for a specific payment.
- `POST /payments`: Submit a payment.
- `PUT /payments/{id}`: Update payment status/details.
- `DELETE /payments/{id}`: Delete a payment record (Admin-only).

Example: Creating a payment.

```
POST /payments
Content-Type: application/json

{
  "appointmentId": 456,
  "amount": 49.99,
  "paymentMethod": "CreditCard",
  "transactionId": "TX-2025-0001"
}
```

3.6 6. Meal Options

- `GET /meal-options`: List all available meal plans.
- `GET /meal-options/{id}`: Retrieve details for a specific meal option.
- `POST /meal-options`: Create a new meal option (Admin-only).
- `PUT /meal-options/{id}`: Update meal option details.
- `DELETE /meal-options/{id}`: Delete a meal option (Admin-only).

3.7 7. Accommodation Reservations

- `GET /accommodations`: List accommodation reservations (patients see their own; Admin sees all).
- `GET /accommodations/{id}`: Retrieve a specific accommodation reservation.
- `POST /accommodations`: Create a new reservation. If meals are included, specify `mealOptionId`.
- `PUT /accommodations/{id}`: Update reservation details.
- `DELETE /accommodations/{id}`: Cancel a reservation.

3.8 8. Rooms

- `GET /rooms`: List rooms (Admin or internal use).
- `GET /rooms/{id}`: Retrieve details for a specific room.
- `POST /rooms`: Create a new room (Admin-only).
- `PUT /rooms/{id}`: Update room details.
- `DELETE /rooms/{id}`: Delete a room record (Admin-only).

3.9 9. Transportation Requests

- `GET /transportation-requests`: List transportation requests (filtered by user or Admin).
- `GET /transportation-requests/{id}`: Retrieve a specific transportation request.
- `POST /transportation-requests`: Create a new transportation request.
- `PUT /transportation-requests/{id}`: Update the status of a request.
- `DELETE /transportation-requests/{id}`: Cancel a transportation request.

3.10 10. Feedback

- `GET /feedback`: List feedback entries (patients see their own; Admin sees all).
- `GET /feedback/{id}`: Retrieve a specific feedback entry.
- `POST /feedback`: Submit new feedback for a completed appointment.
- `PUT /feedback/{id}`: Update feedback (rarely used).
- `DELETE /feedback/{id}`: Delete a feedback entry (Admin or author).

3.11 11. Notifications

- `GET /notifications`: List notifications for the logged-in user.
- `GET /notifications/{id}`: Retrieve a specific notification.
- `POST /notifications`: Create a new notification (system or Admin).
- `PUT /notifications/{id}`: Update a notification (e.g., mark as read).
- `DELETE /notifications/{id}`: Delete a notification.

3.12 12. Service Capacity

- `GET /service-capacity`: List capacity rules (Admin-only).
- `GET /service-capacity/{id}`: Retrieve details for a specific capacity entry.
- `POST /service-capacity`: Create a new capacity rule (Admin-only).
- `PUT /service-capacity/{id}`: Update a capacity rule (Admin-only).
- `DELETE /service-capacity/{id}`: Delete a capacity rule (Admin-only).

4 Error Handling

Standard HTTP status codes are used:

- **200 OK**: Successful operation.
- **201 Created**: Resource successfully created.
- **400 Bad Request**: Invalid data.
- **401 Unauthorized**: Authentication required.
- **403 Forbidden**: Access denied.
- **404 Not Found**: Resource does not exist.

- **409 Conflict**: Data conflict (e.g., capacity overbook).
- **500 Internal Server Error**: Server error.

Example Error Response:

```
{
  "error": "ValidationError",
  "message": "appointmentDatetime is required"
}
```

5 Security Considerations

- **Token-Based Authentication:** All secure endpoints require a valid JWT token.
- **Role-Based Access Control:** Endpoints verify user roles (Patient, Provider, Admin) to enforce permissions.
- **Input Validation:** All inputs are validated to prevent injection and other attacks.
- **HTTPS Enforcement:** All API calls use HTTPS.

6 Example Flows

6.1 Patient Booking an Appointment with Accommodation

1. **Authentication:** `POST /auth/login` returns a JWT token.
2. **Create Appointment:** `POST /appointments` with details including `patientId`, `centerId`, `appointmentDatetime`, and `appointmentDuration`.
3. **Reserve Accommodation:** `POST /accommodations` referencing the created `appointmentId`, specifying `roomId` and optionally `mealOptionId` (e.g., ordered meals "BREAKFAST+LUNCH").
4. **Submit Payment:** `POST /payments` referencing the same `appointmentId`.
5. **Notifications:** The system sends notifications to the patient and provider.

6.2 Provider Managing a Consultation

1. **Provider Login:** `POST /auth/login` returns a token for the provider.
2. **View Appointments:** `GET /appointments?providerId={id}` lists appointments.
3. **Start Consultation:** `POST /consultations` with `appointmentId` creates a consultation record and sets $\texttt{start}_t ime$.`Complete Consultation:PUT /consultations/{id}`$updates\texttt{end}_t imeandsetss$

7 Conclusion

This API design provides a robust and scalable foundation for the Rahati Platform. It covers authentication, resource management, error handling, and security considerations, along with example flows that illustrate typical use cases. This design is intended to evolve as the platform grows and additional requirements emerge.