Dermatologist Review & Recommendation API

1. Architecture Overview

The system follows a layered architecture pattern with clear separation of concerns. To view the system architecture kindly visit this link.

Core Components:

1.1. API Layer: FastAPI handles requests, routing and response formatting

1.2. Authentication: JWT-based token system for secure access control

1.3. Data Layer: SQLAlchemy ORM with SQLite for data persistence

1.4. Validation: Pydantic models ensure data integrity and API contract compliance

1.5. External Integration: HTTP client for fetching product data from DummyJSON API

2. Data Models and Relationships

To view the ER Diagram, please visit this link.

2.1 Entity Definitions:

- **User**: Base entity for authentication (patients and doctors)
- **Doctor**: Extended profile for medical practitioners with specialization
- **Rating**: Numerical rating (1-5) linking users to doctors
- **Review**: Text feedback with word limit validation
- **Recommendation**: Product suggestions with UUID-based sharing and expiry

2.2 Key Relationships:

- One-to-many: Doctor → Ratings, Reviews, Recommendations
- Many-to-one: Ratings, Reviews → User
- One-to-one: User → Doctor (for medical practitioners)

3. Key Design Decisions

3.1. Authentication Strategy

Decision: JWT token-based authentication with role differentiation
Rationale:

- Stateless authentication suitable for REST APIs
- Built-in expiry mechanism
- Role-based access control for doctor-specific operations

3.2. Database Choice

Decision: SQLite in-memory with SQLAlchemy ORM
Rationale:

- Meets assessment requirement for in-memory storage
- SQLAlchemy provides database abstraction for future migration
- Relational model suits the structured data relationships

3.3. Rating System

Decision: Allow rating updates, calculate averages dynamically
Rationale:

- Users can modify their ratings as opinions change
- Real-time average calculation ensures data accuracy
- Simple aggregation suitable for small-scale operations

3.4. Recommendation Sharing

Decision: UUID-based public links with 7-day expiry
Rationale:

- UUIDs provide unpredictable, secure identifiers
- Public access eliminates authentication barriers for patients
- Expiry mechanism prevents indefinite access to potentially outdated recommendations

3.5. Product Integration

Decision: Fetch products synchronously from external API
Rationale:

- Real-time data ensures product information accuracy
- Graceful fallback for missing products
- Caching consideration for production environments

4. Input Validation and Business Rules

4.1. Validation Rules:

- Ratings: Integer range 1-5 with comprehensive error handling
- Reviews: 100-word limit with automatic word counting
- Authentication: Password hashing with SHA-256
- Recommendation expiry: Automatic 7-day TTL with server-side validation.

4.2. Business Logic:

- Doctors can only create recommendations (role-based authorization)
- Users can update existing ratings for the same doctor
- Public recommendation access without authentication requirements
- Expired recommendations return HTTP 410 Gone status

5. Assumptions and Limitations

5.1. Assumptions

1. **User Trust**: Users provide honest ratings and reviews
2. **Product API Reliability**: DummyJSON API maintains consistent availability
3. **Single-Doctor Rating**: Users rate each doctor only once (with updates allowed)
4. **English Content**: All text content is in English for word counting

5.2. Technical Limitations

1. **Scalability**: In-memory database unsuitable for production loads
2. **Concurrency**: No optimistic locking for concurrent rating updates
3. **Caching**: No caching layer for frequently accessed doctor lists
4. **Search**: Limited filtering capabilities (only by minimum rating)