Microservices Architecture Design - Skinfinity

Team Members:

Priyanka Jammu
Aayushi Nirajkumar Desai
Ankita Tripathy

Step 1: Identify Core Services

Skinfinity is a skincare recommendation system powered by AI. The following microservices manage distinct parts of the application flow:

1. **User Profile Service –** Handles registration, login, profile editing, authentication and user preferences.
2. **Questionnaire Service:** Collects lifestyle and skincare input from users.
3. **Image Analysis Service**: Detects skin conditions using uploaded images.
4. **Recommendation Engine Service**: Generates personalized skincare suggestions.
5. **Product Catalog Service**: Manages product information and availability.
6. **Online Consultation Service**: Manages dermatologist booking and session data.
7. **Payment Service**: Handles transactions for consultations and premium access.
8. **Progress Tracking Service**: Tracks user improvement over time.
9. **Notification & Feedback Service**: Sends reminders and gathers user feedback.

Step 2: Define Service Responsibilities

- **User Profile Service:** Authenticates users, stores personal details, and maintains user-specific settings and history.
- **Questionnaire Service**: Provides dynamic forms for users to enter details about their lifestyle, skin concerns, allergies, and current skincare practices.
- **Image Analysis Service:** Accepts uploaded facial images, runs AI/ML models to extract features like acne level, dryness, pigmentation, and stores extracted results.
- **Recommendation Engine Service:** Consumes outputs from the Image Analysis and Questionnaire services, runs matching algorithms, and produces product and routine suggestions.
- **Product Catalog Service:** Maintains a database of skincare products with metadata such as brand, ingredients, compatibility, and links to retailers.
- **Online Consultation Service:** Allows users to schedule virtual dermatologist sessions, view appointment history, and receive post-session recommendations.
- **Payment Service:** Processes user payments using third-party gateways, manages billing history, and handles refunds if applicable.

- **Progress Tracking Service:** Monitors before-and-after comparisons using images and responses over time and displays visual improvements or changes.
- **Notification & Feedback Service:** Sends scheduled notifications and appointment reminders and collects product or session feedback from users.

Step 3: Service Interaction

In order to keep Skinfinity's microservices independent but collaborate well, we adopt a hybrid communication strategy that combines both synchronous APIs and asynchronous messaging systems.

1. Synchronous Communication (APIs)

This method is utilized when a service needs an instantaneous response from another. It is most suitable for real-time, user-initiated interactions.

Implementation:

- RESTful APIs with clearly defined request and response formats
- Secure authentication using user tokens
- Versioned endpoints to reduce tight dependencies

Examples:

- The Recommendation Engine calls the Product Catalog Service to fetch product metadata (e.g., ingredients, suitability).
- The Consultation Service retrieves health and lifestyle data from the Questionnaire Service before an appointment.

2. Asynchronous Communication (Messaging Systems)

This model is used when services exchange information via events and do not need to send immediate responses. It is most suitable for background processes and event-based workflows.

Implementation:

- Messaging tools like Kafka or RabbitMQ
- Event publishing and subscription model
- Message validation and retry mechanisms for fault tolerance

Examples:

- The Image Analysis Service publishes an analysis complete event after processing user photos. The Recommendation Engine listens for this to generate recommendations.

- The Payment Service emits a payment success event. The Notification Service subscribes to this event and sends a confirmation message to the user.

This approach ensures that services are loosely coupled, scalable, and resilient. Each service communicates through well-defined contracts without relying on internal logic or shared databases.

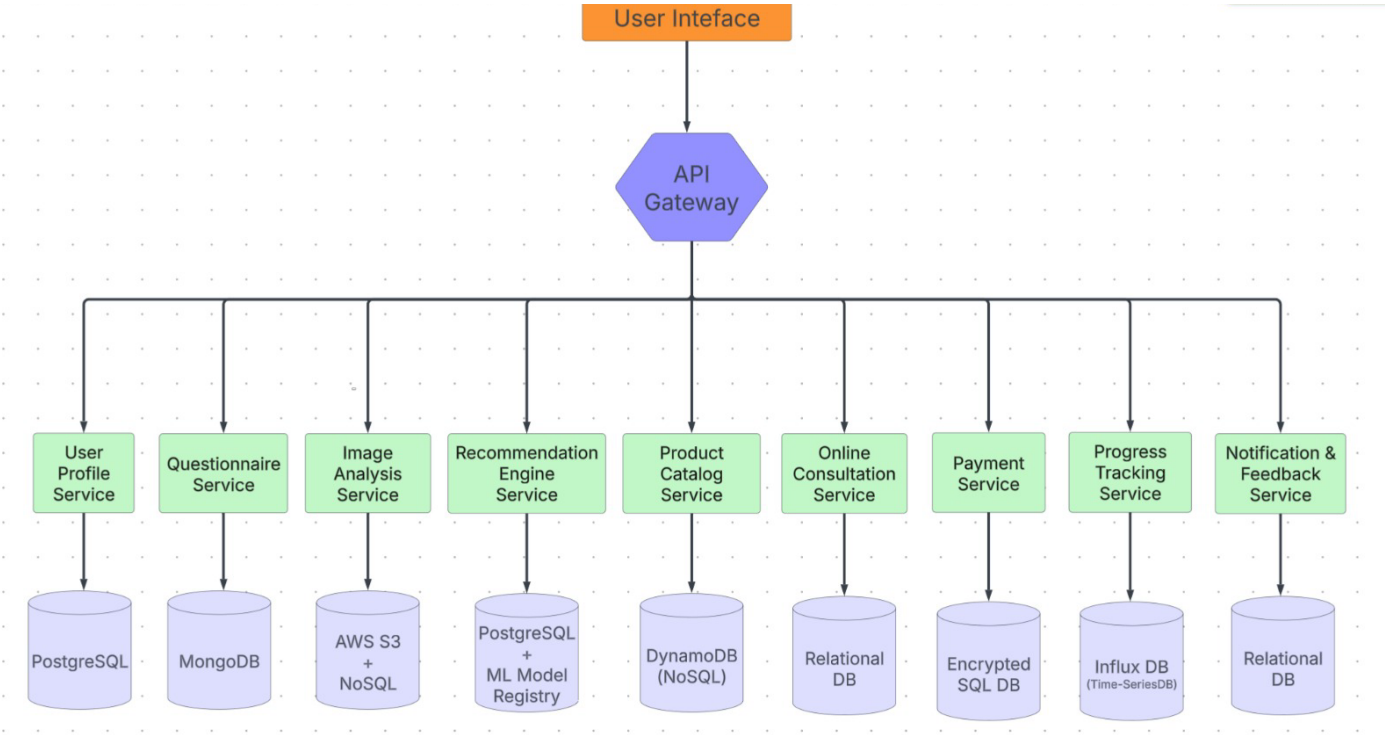
Step 4: Data Management – Real-Time and Decentralized Approach

At Skinfinity, each microservice is responsible for managing its own data. This "database per service" approach keeps services independent, reduces cross-service dependencies, and allows each service to choose the best-fit database for its specific needs — supporting real-time performance and scalability.

- **User Profile Service:** User credentials and preferences are stored in a PostgreSQL database, enabling secure login and quick profile retrieval.
- **Questionnaire Service:** Skincare questionnaire responses such as lifestyle habits and allergies are stored in MongoDB, allowing flexible and evolving data structures.
- **Image Analysis Service:** Uploaded photos are stored in AWS S3, while extracted features (e.g., acne levels, dryness scores) are stored in a NoSQL database for fast access.
- **Recommendation Engine Service:** Recommendations are logged in PostgreSQL for traceability and the service interacts with ML models stored in a model registry for real-time predictions.
- **Product Catalog Service:** Product data like ingredients, compatibility, and pricing is stored in a NoSQL database (e.g., DynamoDB) to support fast reads.
- **Online Consultation Service:** Appointment details and dermatologist notes are stored in a relational database to ensure consistency and reliability.
- **Payment Service:** Transactions are logged in an encrypted SQL database to securely store financial data and meet compliance standards.
- **Progress Tracking Service:** Time-based user data such as before-and-after photos and routine completion is logged in a time-series database like InfluxDB.
- **Notification & Feedback Service:** Messages, alerts, and user feedback are stored in a relational database, making it easy to manage delivery history and user responses

Step 5: Visualization

To understand better about the Skinfinity's microservices architecture, the below diagram diagrammatically presents all the essential components, how they interact, and data transfer. Visual representation of the system architecture facilitates it to recognize easily the dependencies, service boundaries, and possible system bottlenecks.



This architecture diagram consists of:

- Nine important microservices depicted as independent ones.
- Respective databases for all microservices, which represent a database-per-service approach.
- Centralized API Gateway to integrate the User Interface to all microservices via RESTful APIs.
- Synchronous communication is represented by solid arrows, indicating real-time API calls.
- Asynchronous communication, such as event-based communication (e.g., from Image Analysis to Recommendation Engine), is represented by dashed arrows.
- Messaging technologies (e.g., Kafka, RabbitMQ) and cloud storage technologies (e.g., AWS S3) are indicated as well.
- Through the representation of service-to-database relationships and communication patterns, this visualization facilitates scalability, independence, and fault isolation — the core principles of microservices architecture.