Functional Requirements:

- **User Accounts:** Users can sign up, log in, and manage their profiles.
- **Business Listings:** Businesses can be listed with details like name, address, photos, and hours.
- **Search & Discovery:** Users can search for businesses using keywords, filters (category, price), and location.
- **Reviews & Ratings:** Authenticated users can post reviews with a star rating and text.
- **Photo Uploads:** Users can upload photos for businesses.

Non-Functional Requirements:

- **Low Latency:** Search and content loading must be very fast (under 200ms).
- **High Availability:** The system must be resilient to failures and maintain high uptime.
- **Scalability:** The architecture must scale to handle millions of users and businesses globally.
- **Eventual Consistency:** It is acceptable for new reviews and business updates to have a short delay before being visible everywhere.

---

2. Scale Estimation

To guide our design, we'll use estimations based on a large user base.

- **Users:** 200 million monthly active users, with ~20 million daily active users (DAU).
- **Businesses:** 50 million listings.
- **Traffic Pattern:** Highly read-heavy, with an estimated **100:1 read-to-write ratio**.
- **Read QPS (Queries Per Second):** Assuming 20M DAU perform ~30 reads/day = 600M reads/day.
    - Peak Read QPS: (600,000,000 / 86400s) * 2 (peak factor) ≈ 14,000 QPS. We'll design for **15,000 QPS**.
- **Write QPS:** 15,000 / 100 = 150 QPS.
- **Storage Estimation (5 years):**
    - **Businesses:** 50M businesses * 1 KB/business = 50 GB.
    - **Reviews:** 50M businesses * 40 reviews/business * 1 KB/review = 2 TB.
    - **Photos:** 50M businesses * 20 photos/business * 500 KB/photo = 500 TB.

Conclusion: The system must handle high read traffic and store petabytes of media data. The search functionality needs a specialized, high-performance solution.

---

3. High-Level Architecture

A decoupled microservices architecture is the ideal choice for this scale.

Core Components:

1. **API Gateway:** A single, managed entry point that handles client requests, authentication, rate limiting, and routing to the appropriate microservice.
2. **User Service:** Manages user profiles, authentication, and sessions.
3. **Business Service:** The source of truth for all business listing data.
4. **Review Service:** Handles the creation and retrieval of user reviews.
5. **Photo Service:** Manages the upload, storage, and delivery of all media content.
6. **Search Service:** A highly optimized service dedicated to handling complex search queries.
7. **Recommendation Service:** Provides personalized business suggestions to users.
8. **Ad Service:** Manages and serves targeted ads within search results and on business pages.

Technology Stack:

- **Databases:** A **sharded PostgreSQL/MySQL** cluster for the User, Business, and Review services. A relational database provides the transactional integrity needed for this structured data.
- **Search: Elasticsearch** is used by the Search Service for its powerful full-text and geospatial indexing capabilities.
- **Caching: Redis** is used extensively for caching database queries, user sessions, and hot search results to reduce latency.
- **Media Storage: Amazon S3** (or another object store) is used by the Photo Service.
- **Asynchronous Processing: Kafka** is used as a message bus to decouple services and handle write operations asynchronously.
- **Content Delivery:** A **CDN** is placed in front of S3 to deliver photos quickly to users worldwide.

---

4. Deep Dive: Core System Flows

Flow 1: Advanced Search & Ranking

A simple database query is not enough for a good search experience. The **Search Service** uses a sophisticated multi-stage process.

1. **Query Understanding:** The service first uses Natural Language Processing (NLP) techniques to parse the user's query (e.g., "cheap sushi downtown"). It extracts key intents and entities like `cuisine: sushi`, `price: cheap`, `location: downtown`.
2. **Candidate Generation:** These entities are used to construct an initial query to **Elasticsearch**. Elasticsearch quickly returns a broad list of candidate businesses based on keyword matches and geospatial filters.
3. **Machine Learning Ranking:** This is the crucial step. The list of candidates is passed to a **ranking model**. This model scores each business based on a variety of signals to determine the final order:

    - **Business Signals:** Average rating, number of reviews, price tier.
    - **User Signals:** Distance from the user, match with the user's past preferences (provided by the **Recommendation Service**).
    - **Contextual Signals:** Time of day (e.g., ranking restaurants open for dinner higher in the evening).
    - **Business Model Signals:** Whether the business is a paid advertiser (provided by the **Ad Service**).
4. **Results Presentation:** The final, ranked list of businesses is returned to the user.

Flow 2: Asynchronous Review Processing

To ensure a fast user experience and a resilient system, new reviews are processed asynchronously.

1. A user submits a review. The request is sent to the **Review Service**.
2. The Review Service performs basic validation and immediately publishes the review data to a **Kafka** topic called `new_reviews`. It then returns a `202 Accepted` response to the client.
3. Multiple, independent consumer services subscribe to this Kafka topic to perform different tasks in parallel:
    - **DB Writer Consumer:** Writes the review content to the primary **PostgreSQL** database.
    - **Search Indexer Consumer:** Pushes the new review text into the **Elasticsearch** index so it becomes searchable.
    - **Rating Aggregator Consumer:** Updates the `average_rating` and `review_count` fields in the `businesses` table for the reviewed business.
    - **Anti-Fraud Consumer:** Analyzes the review using a machine learning model to detect spam or fake reviews.

---

5. API & Database Schema

Example REST APIs:

- `GET /v1/search?term=pizza&location=nyc`: Main endpoint for the Search Service.
- `GET /v1/businesses/{business_id}`: Retrieves details for a specific business.
- `GET /v1/businesses/{business_id}/reviews`: Fetches reviews for a business.
- `POST /v1/businesses/{business_id}/reviews`: Submits a new review.

Simplified SQL Database Schema:

- **businesses:** `business_id (PK)`, `name`, `address`, `location (geopoint)`, `avg_rating`, `review_count`.
- **reviews:** `review_id (PK)`, `business_id (FK)`, `user_id (FK)`, `rating`, `text`, `created_at`.
- **users:** `user_id (PK)`, `username`, `email`, `hashed_password`.