Trade-offs in Tinder-like App Architecture

1. Database Approach: Hybrid (Location-Based Primary Databases + Sharding & Replication)

To optimize performance, we employ a **hybrid database strategy**. The **primary database is location-based**, ensuring users retrieve matches with low latency. We further enhance scalability through **sharding and replication**:

- **Sharding:** Users are partitioned by region to minimize cross-region queries.
- **Replication:** Read replicas handle read-heavy traffic.
- **Hybrid read/write approach:** Read traffic is distributed across replicas, while writes use a leader-follower system to balance consistency and performance.

2. Eventual Consistency vs. Strong Consistency

To reduce costs and improve scalability, the system follows an **eventual consistency model**, except in critical areas such as payments:

- **Match swipes, user profiles, and messages** use eventual consistency to allow high availability.
- **Payment transactions, premium subscriptions, and purchases** require **strong consistency** to prevent fraud and ensure reliability.

3. Rate Limiting for Request Control

Rate limiting is enforced based on **subscription tiers** to balance server load and ensure fair usage:

- **Free users** have stricter request caps (e.g., limited swipes per minute).
- **Premium users** have higher limits for swipes, messages, and profile views.
- Implemented via **token bucket algorithm** to allow bursty traffic while preventing abuse.

4. Load Balancing for High Availability

A **global load balancer** distributes user traffic efficiently across regions. Key considerations:

- **Geolocation-based routing:** Directs users to the nearest data center.
- **Sticky sessions for WebSockets:** Ensures real-time messaging continuity.
- **Round-robin or least connections balancing** for backend APIs.

5. Content Delivery Network (CDN) for Static Assets

To optimize delivery of static assets (images, CSS, JavaScript):

- **CDN caches user profile pictures and static UI elements** to reduce load on origin servers.
- **Optimized image formats (WebP, AVIF) and compression techniques** improve load times.

6. Machine Learning (ML) for Personalized Matching

ML models improve user experience through:

- **Personalized ranking algorithms** based on swiping behavior and interactions.
- **Spam and bot detection models** to prevent misuse.
- **Real-time image moderation and NSFW content filtering** using deep learning.

7. WebSockets for Real-Time Messaging

- WebSockets enable **low-latency, bidirectional messaging** between users.
- Persistent connections ensure instant message delivery and read receipts.
- **Falling back to polling/long polling** for older devices or network restrictions.

8. Indexing & Partitioning for Fast Query Retrieval

Efficient data retrieval is achieved through:

- **Indexing user profiles on key attributes (age, location, interests).**
- **Partitioning message storage** based on sender-receiver pairs for fast lookups.

9. Global Capacity Planning & Scalability

To handle **80 million MAUs and 20 million DAUs**, with a **5x buffer for peak loads**, infrastructure is designed with:

- **Horizontally scalable databases** to handle query load.
- **Auto-scaling compute clusters** for real-time ML inference and API processing.
- **Edge caching of frequently accessed content** to offload backend queries.

10. Hybrid Read-Optimized and Write-Optimized System

Since Tinder is primarily **read-heavy**, the system is optimized accordingly:

- **Profile & match queries use read replicas** to scale horizontally.
- **Writes (likes, super likes, new messages) are batched and processed asynchronously** to reduce database contention.
- **Caching strategies (Redis, Memcached) are used** to reduce read latency on frequently accessed data.