Design Yelp

Design a nearby business finding application like Yelp, google maps near me feature

Practiced On: 13 Feb 2025

Time taken: 85 min (1 H 25 min)

Qs for the Interviewer

1. Application will be on India level or global level?
2. What are the capabilities we are giving to business owners, is it something similar to Google Business?
3. How many business our application needs to manage?
4. How many DAU we have? What is peak load expectation?
5. Is user allowed to put KM distance range as well to find nearby businesses?
6. Application will be taking user's location?
7. Is it mandatory for the user to be logged in?
8. How many reviews we needs to persist and for how long?

Requirements

Functional Requirements:
1. User should be able to search a business via TEXT search, he can have filters available for search.
2. User will get list of business for choose from.
3. User can select any business to see its page and details like rating, reviews, open close days & hours etc.
4. User can post review for a business with photos and videos.
5. Business owners can upload details regarding their business (Secondary)

Non Functional Requirements:
1. System should have high availability for all the customers. Availability > Consistency
2. System should have low latency for the search
3. System should scale well for other new geo locations and during peak hours
4. System should be reliable - provide nearby businesses at correct distance from the user

System Entities

1. User
2. Business Page
3. Review
4. Business Location

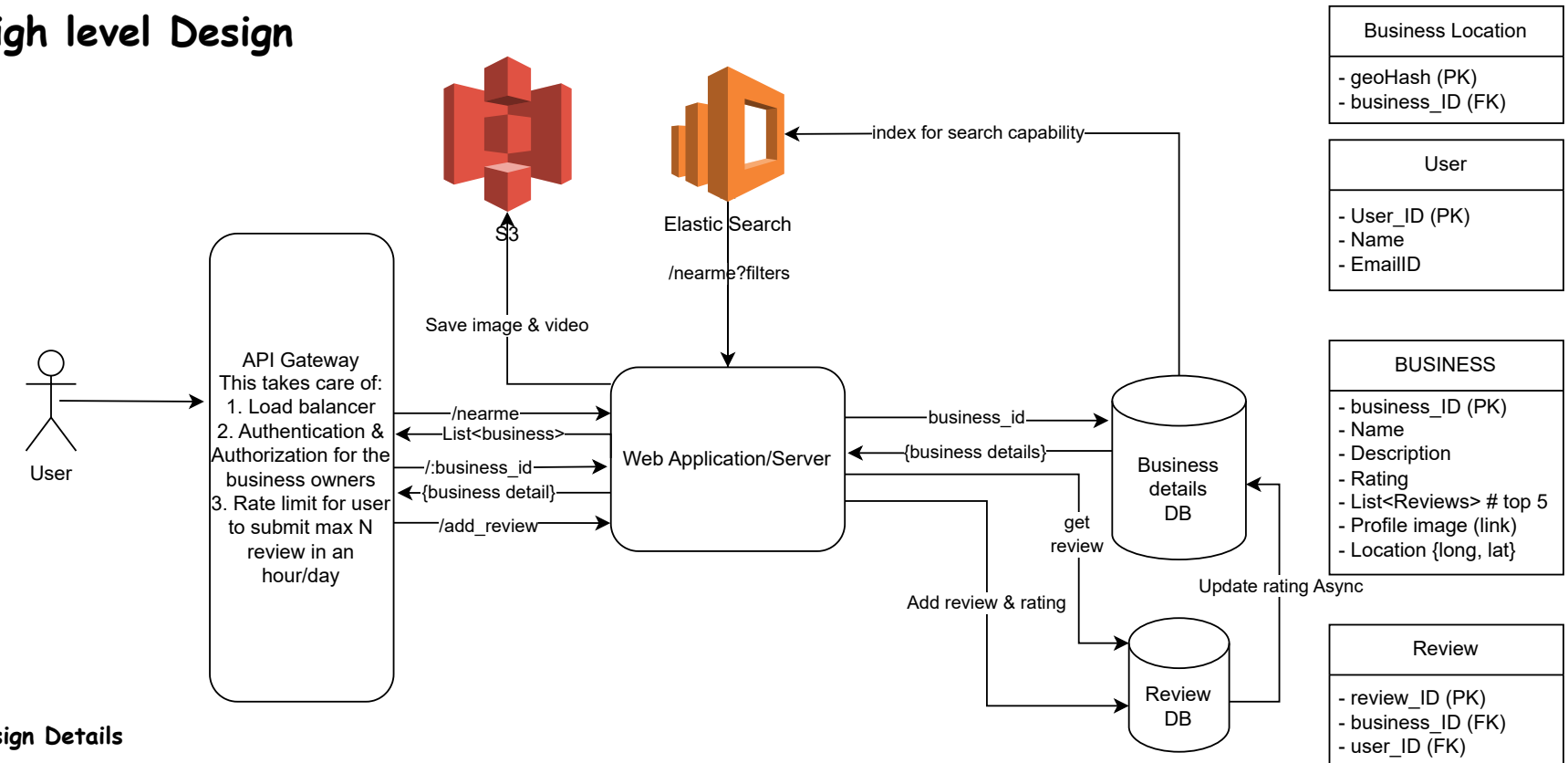
API Design

1. User should able to search for businesses with filters
GET HTTPS/near?text="SEARCH_BAR_TEXT"&?open=""&?distance=""&?rating=""
body {
	long:""
	latt:""
}
RESPONSE {
list<business> # at this point we will be returning a partial business details to show as a list
}
single business obj will contain: Name, Business ID, What it is?, Rating, Distance, Single image, Offer(added by business owners - like KFC added 10% Off)

2. User click on a single business card from the list
GET HTTPS /near/business/:business_id
RESPONSE {
Name, Title, Description about this business, images, videos, rating, customer reviews, menu (in case of restaurant)
}

3. User submit a review for a business
POST HTTP/near/business/:business_id
body {
	rating: "" #mandatory
	review: "" #optional upto 500 char
}
RESPONSE {
	rating: ""
	review: ""
}

High level Design



Design Details

1. User hit with his request to API gateway / Load balancer
2. Request move to web server
3. User searched "burger" with filter isOpen="true", distance_radius=5km, rating>4
	3.1. web server pass on the request to elastic search

3.2. based of filters and keyword elastic search returned a list of businesses.
ES returned partial data as it have not all the data like review and description
3.3. list of businesses is shown to the user
4. user click on one of the line item from the list (business_id)
4.1. Request web server for business_id
4.2. request sent to DB for this business_id
4.3. DB return business details with top 5 review (At this point only top informative comments are shown to the user)
4.4. In case user clicks on more reviews - paginated API call to review DB is made
4.5. Reviews API response can be sort via different filters
5. User added a review and rating
5.1. review passed to review DB
5.2. Async - review will be examine for bad word / authenticity.
5.3. Async - rating for the business will be updated in Business DB

Note: In inicial days one can keep review in same DB (diff table), but at time grows one can have separate that out.
TODO - need to think how to decide wether to keep tables in same DB or different DB instances

Design Deep Dive

How to find the nearby businesses

To find the near by businesses under a specific distance provide by the user can be done via geoHash data
1. When a business owner adds a business on the platform, he needs to add {log, lat} for location
2. A geohash can be derived from {log, lat} via libraries like: python-geohash
3. This geohash will be stored with business_id in [business location] table
4. As user makes an request for nearby businesses - we get their {log, lat} in the request body > we can have user's geohash
5. Now we have both user's geohash #UGH and list of all business geohash List<#BGH>
6. We can query the business geohash List<#BGH> with condition for at least 4 length prefix should match with #UGH
6.1. 4 length prefix match is at least
6.2 longest prefix match will be nearest
6.3 we return list of #BGH with longest prefix match upload 4 length - sorted manner
6.4 using #BGH one can query and get business_id and with that all other business details

How to make sure a single user provides single review only

1. We can make this a mandate that if one needs to add review, one needs to be logged-in user
2. If is user click on [Add review] - we can query the review DB and match if this user_ID has written a review for this business_ID.
2.1. If YES - return the same review - that can be edited.
2.2. If NO - show an empty form to submit a review

How to have high availability? & Scaling?

1. For peak hours when load is high we can have auto scaling enabled for our web service.
2. As our system is read heavy - we would keep a READ REPLICA for our database, so that READ query are fast
3. Business Location DB is not updating very frequently, can be cached for fast retrieval

Design Review By ChatGPT

Strengths of Your Design

✅ Clear Requirements & API Design

- You have asked the right questions about **scale (India vs. Global), business owner capabilities, user authentication, and data retention.**
- Functional & Non-functional requirements are **well defined.**
- **RESTful APIs** are structured logically, with query parameters for filters.

✅ Efficient Search via ElasticSearch + Geohash Indexing

- **ElasticSearch** for text-based business search ensures **low latency** and fast lookups.
- **Geohash-based indexing** is a solid way to handle **"near me"** queries efficiently.
- Using **prefix matching on Geohashes** is an excellent way to optimize **spatial queries.**

✅ Scalability & High Availability Considerations

- **API Gateway** with **rate-limiting** and **auth checks** protects the system.
- **Read Replicas** for the database ensure fast reads.
- **Auto-scaling of web services** to handle peak loads.
- **Caching business locations** reduces DB lookups for frequently searched areas.

✅ User Review Handling & Moderation

- Prevents duplicate reviews by **checking if a user has already reviewed a business.**
- Asynchronous **bad-word filtering & review authenticity checks** improve quality.
- **Paginated reviews API** ensures **efficient loading** of older reviews.

✅ Optimized Review Storage Strategy

- Initially, reviews are **stored in the same DB as businesses** but can be **migrated later to a separate service** as traffic increases.
- **Incremental rating updates** in business DB **asynchronously** prevent performance bottlenecks.

Areas for Improvement & Optimizations

🔷 1. ElasticSearch + Geohash Query Optimization

- **Current Approach:** Geohash prefix matching works well but may return **false positives at larger distances.**

- **Suggested Improvement:**
  - Use a **Hybrid Approach**:
    1. **First, use Geohash prefix matching** for a **quick shortlist** of businesses.
    2. **Then, apply Haversine formula** (great-circle distance) for **precise filtering** based on the user's radius.

🔷 2. Improve Caching Strategy for Hot Locations

- **Current Approach:** Business Location DB **caching is mentioned**, but **no details** on eviction policies.

- **Suggested Improvement:**
  - **Use Redis for caching high-traffic locations** (e.g., downtown areas).
  - **Store precomputed popular searches** (e.g., "best coffee shops in Mumbai") for **fast retrieval.**

🔷 3. Handling High-Volume Review Updates Efficiently

- **Current Approach:** Updating ratings **asynchronously** in Business DB.
- **Potential Problem:High-frequency reviews** may cause **race conditions** in rating updates.
- **Suggested Improvement:**Use **event-driven architecture** (Kafka or RabbitMQ) to **batch-process reviews** before updating Business DB.

DeepSeek: Areas for Improvement & Optimizations

Areas for Improvement

1. Geolocation Search

- **Issue**: Relying solely on geohash prefix matching (4-length) may miss edge cases (e.g., businesses near geohash boundaries).
- **Improvement**:
  - Use Elasticsearch's built-in geo queries for accurate distance filtering.
  - Combine geohash with Haversine formula for precise distance calculations.

2. Data Consistency

- **Issue**: Async rating updates could lead to temporary inconsistencies.
- **Improvement**:
  - Acknowledge eventual consistency in non-functional requirements.
  - Use idempotent operations for rating updates to avoid duplicates.

3. Security & Anti-Abuse

- **Issue**: Limited details on securing APIs or preventing spam.
- **Improvement**:
  - Add HTTPS, input validation, and parameterized queries to prevent SQL injection.
  - Implement CAPTCHA or rate-limiting for reviews to deter bots.

4. Media Handling

- **Issue**: No mention of optimizations for images/videos.
- **Improvement**:
  - Use CDN (e.g., CloudFront) for fast media delivery.
  - Add image compression/video transcoding during upload.

5. Scalability Deep Dive

- **Issue**: Sharding and partitioning strategies are missing.
- **Improvement**:
  - Shard databases by geohash or business ID.
  - Consider CQRS (Command Query Responsibility Segregation) for read/write separation.

7. Edge Cases

- **Issue**: Handling overlapping businesses (e.g., same geohash but far apart).
- **Improvement**: Add a secondary filter using actual distance after geohash-based pre-filtering.

8. API Details

- **Issue**: Missing authentication headers in review submission API.
- **Improvement**: Include JWT tokens or API keys in headers for authenticated requests.