System design from sample paper

Enhanced Personalized Restaurant Recommendation Engine

1. Data Stores

To build an enhanced personalized recommendation engine for the Dining Concierge chatbot, several data stores are integrated to handle various aspects of data management, personalization, and real-time adaptability. Below is a detailed list and description of each data store, including their schemas and indexing mechanisms.

a. User Profile Store (Amazon DynamoDB)

- **Purpose**: Stores user-specific data, including preferences, past interactions, and feedback.
- **Data Stored**:
  - **UserID** (Primary Key): Unique identifier for each user.
  - **Preferences**: JSON object containing preferred cuisines, locations, dining times, etc.
  - **Past Searches**: Array of search queries with timestamps.
  - **Feedback**: Array of liked/disliked restaurants with timestamps.
- **Schema Example**:

json

Copy code

```
{ "UserID": "user123", "Preferences": { "cuisines": ["Japanese", "Italian"], "locations": ["Manhattan",
"Brooklyn"], "diningTime": "7 pm" }, "PastSearches": [ {"query": "Japanese in Manhattan",
"timestamp": "2024-10-25T19:00:00Z"} ], "Feedback": [ {"restaurantID": "rest001", "liked": true,
"timestamp": "2024-10-25T20:00:00Z"} ] }
```

- **Indexing Mechanisms**:
  - **Primary Key**: UserID ensures unique and fast retrieval of user profiles.
  - **Global Secondary Index (GSI)**: Index on Preferences.cuisines to facilitate quick querying based on cuisine preferences.

b. Restaurant Data Store (Amazon DynamoDB & Amazon Elasticsearch Service)

- **Purpose**: Stores comprehensive restaurant information and facilitates efficient searching and filtering.

i. DynamoDB Table: yelp-restaurants

- **Data Stored**:
  - **BusinessID** (Primary Key): Unique identifier for each restaurant.
  - **Name**: Name of the restaurant.

    - **Address**: Physical address.
    - **Coordinates**: Geolocation data (latitude and longitude).
    - **NumberOfReviews**: Total number of reviews.
    - **Rating**: Average rating.
    - **ZipCode**: Postal code.
    - **InsertedAtTimestamp**: Timestamp of data insertion.
- **Schema Example**:

json

Copy code

```
{ "BusinessID": "rest001", "Name": "Sushi Nakazawa", "Address": "23 Commerce St", "Coordinates":
{"lat": 40.71427, "lon": -74.00597}, "NumberOfReviews": 1500, "Rating": 4.8, "ZipCode": "10014",
"InsertedAtTimestamp": "2024-04-01T12:00:00Z" }
```

- **Indexing Mechanisms**:
    - **Primary Key**: BusinessID for unique identification.
    - **GSI on** Cuisine: To enable quick retrieval based on cuisine types.

ii. Elasticsearch Index: restaurants

- **Data Stored**:
    - **RestaurantID**: Same as BusinessID in DynamoDB.
    - **Cuisine**: Type of cuisine offered.
- **Schema Example**:

json

Copy code

```
{ "RestaurantID": "rest001", "Cuisine": "Japanese" }
```

- **Indexing Mechanisms**:
    - **Inverted Index** on Cuisine to facilitate full-text search and quick filtering based on cuisine types.
    - **Geo Indexing** on Coordinates for efficient geographical proximity searches.

c. Recommendation Queue (Amazon SQS)

- **Purpose**: Acts as a buffer for recommendation requests to ensure asynchronous processing.
- **Data Stored**:
    - Messages containing user requests with relevant parameters (e.g., UserID, Cuisine, Location).

- **Schema Example**:

json

Copy code

```
{ "UserID": "user123", "Cuisine": "Japanese", "Location": "Manhattan", "NumberOfPeople": 2,
"DiningTime": "7 pm", "Email": "user@example.com" }
```

- **Indexing Mechanism**:
  - SQS does not require traditional indexing; it handles message ordering and visibility internally.

d. Trending Data Store (Amazon DynamoDB)

- **Purpose**: Stores trending restaurants data based on user interactions.
- **Data Stored**:
  - **RestaurantID**: Unique identifier.
  - **TrendScore**: Calculated based on the number of likes.
  - **LastUpdated**: Timestamp of the last update.
- **Schema Example**:

json

Copy code

```
{ "RestaurantID": "rest001", "TrendScore": 150, "LastUpdated": "2024-10-31T10:00:00Z" }
```

- **Indexing Mechanisms**:
  - **Primary Key**: RestaurantID.
  - **GSI on** TrendScore: To enable quick retrieval of top trending restaurants.

e. State Management Store (Amazon DynamoDB)

- **Purpose**: Maintains the state of user interactions for personalized experiences.
- **Data Stored**:
  - **UserID** (Primary Key): Unique identifier.
  - **LastSearch**: Details of the last search performed.
- **Schema Example**:

json

Copy code

```
{ "UserID": "user123", "LastSearch": { "Cuisine": "Japanese", "Location": "Manhattan" } }
```

- **Indexing Mechanisms**:
  - **Primary Key**: UserID for quick state retrieval.

2. APIs

To support the enhanced functionalities, several new APIs are integrated into the system. These APIs facilitate personalized recommendations, trending data retrieval, and state management.

a. New APIs

1. **GetPersonalizedRecommendations API**
    - **Endpoint**: /recommendations/personalized
    - **Method**: GET
    - **Description**: Fetches personalized restaurant recommendations based on user preferences and past interactions.
    - **Parameters**: UserID, optional filters (e.g., location, cuisine).
2. **GetTrendingRecommendations API**
    - **Endpoint**: /recommendations/trending
    - **Method**: GET
    - **Description**: Retrieves trending restaurant recommendations based on collective user interactions.
    - **Parameters**: Location, optional filters.
3. **SubmitFeedback API**
    - **Endpoint**: /feedback/submit
    - **Method**: POST
    - **Description**: Allows users to submit feedback (like/dislike) on recommended restaurants.
    - **Payload**: UserID, RestaurantID, FeedbackType.
4. **GetLastSearch API**
    - **Endpoint**: /user/last-search
    - **Method**: GET
    - **Description**: Retrieves the last search parameters of a user for automatic recommendations.
    - **Parameters**: UserID.

b. Low-Level Backend Design

To handle high traffic efficiently, the backend is designed using AWS services that support scalability, event-driven architecture, and asynchronous processing.

1. **API Gateway**
    - **Role**: Acts as the entry point for all API requests.

- **Features**:
    - **Throttling and Rate Limiting**: Ensures API stability under high traffic.
    - **Authentication**: Secures APIs using AWS Cognito or API keys.

2. **AWS Lambda Functions**
    - **Functionality**:
        - **PersonalizedRecommendationsHandler**: Processes requests for personalized recommendations.
        - **TrendingRecommendationsHandler**: Processes requests for trending recommendations.
        - **FeedbackHandler**: Handles user feedback submissions.
        - **LastSearchHandler**: Retrieves the last search state for a user.
    - **Scalability**: Automatically scales based on incoming request volume.
    - **Statelessness**: Ensures that each invocation is independent, enhancing reliability.
3. **Amazon DynamoDB**
    - **Role**: Serves as the primary data store for user profiles, state management, and trending data.
    - **Features**:
        - **DAX (DynamoDB Accelerator)**: Optional in-memory caching for read-heavy operations.
        - **Auto Scaling**: Automatically adjusts read/write capacity based on traffic.
4. **Amazon Elasticsearch Service**
    - **Role**: Facilitates efficient searching and filtering based on restaurant attributes.
    - **Features**:
        - **Shard and Replica Configuration**: Ensures high availability and performance.
        - **Kibana Integration**: For monitoring and visualizing search queries.
5. **Amazon SQS**
    - **Role**: Manages asynchronous processing of recommendation requests.
    - **Features**:
        - **FIFO Queues**: Ensures ordered processing if required.
        - **Dead-Letter Queues**: Handles failed message processing.
6. **Amazon SNS (Simple Notification Service)**
    - **Role**: Notifies other services or triggers workflows based on specific events (e.g., new feedback submission).

7. **Amazon SES (Simple Email Service)**
    - **Role**: Sends personalized recommendation emails to users.
8. **Amazon EventBridge (formerly CloudWatch Events)**
    - **Role**: Manages scheduled tasks and event-driven triggers.

c. Scalability and Resilience Features

- **Auto Scaling**: Both API Gateway and Lambda automatically scale to handle varying loads.
- **Load Balancing**: API Gateway efficiently distributes incoming requests.
- **Retry Mechanisms**: Implemented in SQS and Lambda to handle transient failures.
- **Monitoring and Logging**: Amazon CloudWatch provides metrics and logs for all services, enabling proactive scaling and issue resolution.

3. System Design Architecture (High-Level Backend Design)

Below is a high-level architecture diagram illustrating the integration of the personalized recommendation engine and dynamic data system with the existing chatbot.

Note: As this is a text-based response, please visualize the architecture as described below.

Architecture Components and Flow

1. **User Interaction**:
    - Users interact with the frontend hosted on **Amazon S3**, which communicates with the backend via **API Gateway**.
2. **API Gateway**:
    - Routes requests to appropriate **AWS Lambda** functions:
        - **PersonalizedRecommendationsHandler**
        - **TrendingRecommendationsHandler**
        - **FeedbackHandler**
        - **LastSearchHandler**
3. **Lambda Functions**:
    - **PersonalizedRecommendationsHandler**:
        - Retrieves user preferences and past interactions from **DynamoDB**.
        - Queries **Amazon Elasticsearch Service** for matching restaurants.
        - Fetches detailed information from **DynamoDB**.
        - Compiles and sends recommendations via **Amazon SES**.
    - **TrendingRecommendationsHandler**:
        - Queries **Trending Data Store** in **DynamoDB**.

     - Retrieves trending restaurants from **Elasticsearch** and **DynamoDB**.
     - Sends recommendations via **Amazon SES**.
   - **FeedbackHandler**:
     - Processes user feedback and updates **Trending Data Store**.
   - **LastSearchHandler**:
     - Retrieves the last search parameters for a user to offer automatic recommendations.

4. **Data Stores**:
   - **DynamoDB**:
     - **User Profile Store**: Stores user-specific data.
     - **Restaurant Data Store**: Comprehensive restaurant information.
     - **Trending Data Store**: Tracks trending restaurants.
     - **State Management Store**: Maintains user interaction states.
   - **Amazon Elasticsearch Service**:
     - Facilitates efficient search and filtering of restaurants based on cuisine and geographic proximity.
5. **Asynchronous Processing**:
   - **Amazon SQS**:
     - Manages recommendation request queues.
     - **Lambda Workers** (e.g., **LF2**) process queue messages to fetch and send recommendations.
6. **Real-Time Data Integration**:
   - **External Data Sources** (e.g., Yelp API) feed updates into **Elasticsearch** and **DynamoDB** via scheduled **Lambda** functions or **EventBridge** triggers.
7. **Notification Service**:
   - **Amazon SES** sends personalized recommendation emails to users based on processed data.

4. Feedback Loop, Real-Time Processing, and Adaptability

a. Feedback Loop

The system incorporates a robust feedback loop to continuously refine and personalize recommendations based on user interactions and preferences.

1. **User Feedback Collection**:

   - Users interact with the frontend and can "like" recommended restaurants.
   - These interactions are sent to the **SubmitFeedback API**, which invokes the **FeedbackHandler Lambda**.

2. **Feedback Processing**:
   - **FeedbackHandler Lambda** updates the **User Profile Store** in **DynamoDB** with the feedback.
   - It also updates the **Trending Data Store** by incrementing the TrendScore for liked restaurants.
3. **Recommendation Refinement**:
   - Future personalized recommendations consider updated user preferences and trending data.
   - This ensures that recommendations evolve with user behavior and broader trends.

b. Real-Time Data Handling and Adaptability

The system is designed to adapt to real-time data changes, such as restaurant availability or new openings, ensuring that recommendations remain relevant and up-to-date.

1. **External Data Integration**:
   - **Data Source Assumption**: Assume integration with the **Yelp API** for real-time restaurant data updates.
   - **Lambda Function**: Scheduled **Lambda** functions fetch updates from the Yelp API at regular intervals (e.g., every hour).
2. **Data Update Pipeline**:
   - **Lambda** fetches new or updated restaurant data.
   - Updates are pushed to both **DynamoDB** and **Elasticsearch**:
     - **DynamoDB**: Ensures comprehensive and up-to-date restaurant details.
     - **Elasticsearch**: Facilitates efficient searching and filtering based on the latest data.
3. **Handling Restaurant Availability**:
   - **Lambda Workers** periodically check restaurant statuses (e.g., open/closed) via the Yelp API.
   - Updates are reflected in the data stores to exclude unavailable restaurants from recommendations.
4. **Dynamic Recommendations**:
   - **Lambda Handlers** dynamically query the latest data from **Elasticsearch** and **DynamoDB** to provide current recommendations.

    - **Trending Recommendations** are recalculated based on the latest user interactions and feedback.

c. AWS Components and Services for Real-Time Management

- **Amazon Kinesis** (Optional):
    - For handling high-throughput real-time data streams from external APIs.
- **AWS Lambda**:
    - Processes real-time data updates and user feedback.
- **Amazon EventBridge**:
    - Orchestrates event-driven workflows, triggering Lambda functions based on specific events or schedules.
- **Amazon DynamoDB Streams**:
    - Captures changes in DynamoDB tables for real-time processing and integration with other services.

5. Data Pipeline / Event Flow

The data pipeline ensures seamless flow from user interaction to the delivery of personalized recommendations, incorporating feedback and real-time data updates.

a. Data Pipeline Steps

1. **User Interaction**:
    - User engages with the chatbot via the frontend hosted on **S3**.
    - Requests for recommendations are sent through **API Gateway** to the respective Lambda handlers.
2. **Recommendation Request Processing**:
    - **PersonalizedRecommendationsHandler** Lambda retrieves user data from **DynamoDB**.
    - Queries **Elasticsearch** for matching restaurants based on preferences and geographic proximity.
    - Fetches detailed restaurant information from **DynamoDB**.
    - Compiles a list of 5 personalized and 5 trending recommendations.
    - Sends the recommendations via **Amazon SES**.
3. **Feedback Submission**:
    - User "likes" a restaurant, triggering the **SubmitFeedback API**.
    - **FeedbackHandler Lambda** updates user profiles and trending data.
4. **Trending Data Update**:

- **FeedbackHandler** updates the **Trending Data Store** in **DynamoDB**.
- The **TrendingRecommendationsHandler** Lambda uses this data to prioritize trending restaurants.

5. **Real-Time Data Updates**:
   - **Scheduled Lambda Functions** fetch updates from the Yelp API.
   - Updates are pushed to **DynamoDB** and **Elasticsearch**.
   - Ensures that recommendations reflect the latest restaurant data.
6. **State Management**:
   - **LastSearchHandler** Lambda retrieves the last search parameters.
   - Provides automatic recommendations based on past searches when the user returns.
7. **Asynchronous Processing**:
   - **Amazon SQS** queues handle high-volume recommendation requests.
   - **Lambda Workers** process queue messages, ensuring scalability and reliability.

b. Event Flow Diagram

1. **User Requests Recommendations**:
   - Frontend → API Gateway → PersonalizedRecommendationsHandler Lambda
2. **Lambda Processes Request**:
   - Fetch user data from DynamoDB
   - Query Elasticsearch for restaurants
   - Retrieve details from DynamoDB
   - Send email via SES
3. **User Provides Feedback**:
   - Frontend → SubmitFeedback API → FeedbackHandler Lambda
4. **FeedbackHandler Updates Data Stores**:
   - Update User Profile Store and Trending Data Store in DynamoDB
5. **Scheduled Data Updates**:
   - EventBridge triggers DataUpdate Lambda
   - Fetch and update data from Yelp API to DynamoDB and Elasticsearch
6. **Queue Processing**:
   - New recommendation requests are placed in SQS
   - Lambda Workers poll SQS and process messages

Assumptions

- **External Data Source**: The Yelp API is used for fetching real-time restaurant data, including new openings and availability statuses.
- **User Authentication**: Users are authenticated via AWS Cognito to secure API endpoints and manage user identities.
- **Data Volume**: The system is designed to handle a large user base with high-frequency interactions and data updates.
- **Email Sending Limits**: Amazon SES is configured to handle the expected email volume, with appropriate sending limits and verified domains.

Conclusion

The enhanced Personalized Restaurant Recommendation Engine leverages AWS services to deliver a dynamic, user-centric experience. By integrating personalized recommendations, trending data, and real-time adaptability, the system ensures that users receive relevant and up-to-date restaurant suggestions. The architecture emphasizes scalability, resilience, and efficient data management, making it robust enough to handle a growing user base and evolving data landscapes.