Build Serverless SaaS APIs for DiningByFriends Social Network Project using Relational Database and OpenAPI Specifications (Version 1.0.0)

You will build a building a fictitious restaurant review and recommendation API we call "DiningByFriends." As we move through the software development life cycle from planning, to analysis, to design, and on to implementation, this API project will help us how to think about and teach us how to build Serverless SaaS APIs.

Source:
Chapter 2 Graph Databases in Action
https://livebook.manning.com/book/graph-databases-in-action/chapter-2/

Required Learning:
1. Event Driven Architecture

2. AWS Technologies and Services using TypeScript:
   CDK, Cognito, VPC, Aurora Serverless, API Gateway, Lambda, and EventBridge

3. SQL:
https://www.w3schools.com/sql/

4. Entity Relationship Diagraming Tool: DB Diagram
https://dbdiagram.io

5. OpenAPI 3.0 Tutorial
https://support.smartbear.com/swaggerhub/docs/tutorials/openapi-3-tutorial.html

6a. CDK OpenAPI Support
https://www.npmjs.com/package/@aws-cdk/aws-apigateway?activeTab=readme#openapi-definition

6b. Exploring An OpenAPI/Swagger First Approach to Serverless Development on AWS (Theory)
https://itnext.io/exploring-an-openapi-swagger-first-approach-to-serverless-development-on-aws-d19f0e9ca257

7a. Swagger OpenAPI Editor:
https://swagger.io/tools/swagger-editor/

7b. Open API Visualization and Documentation: Swagger UI
https://swagger.io/tools/swagger-ui/

8. Generate Client SDKs and Server Stubs from Open API
https://swagger.io/tools/swagger-codegen/
https://github.com/anttiviljami/openapi-backend

9. Dynamic Test Generation
https://www.readysetcloud.io/blog/allen.helton/dynamic-test-generation-with-oas/
https://opensource.com/article/18/6/better-api-testing-openapi-specification
https://github.com/stoplightio/prism

10. CI/CD Pipeline: Github Actions
https://docs.github.com/en/actions

11. OpenAPI Tools Reference
https://openapi.tools/

12. Working with OpenAPI with AWS API Gateway
https://advancedweb.hu/how-to-use-openapi-with-api-gateway-rest-apis/
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-import-api.html
https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-open-api.html
https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-export.html
https://support.smartbear.com/swaggerhub/docs/integrations/amazon-api-gateway.html
https://docs.aws.amazon.com/cdk/api/latest/docs/@aws-cdk_aws-apigateway.SpecRestApi.html
https://stackoverflow.com/questions/62179893/aws-cdk-how-to-create-an-api-gateway-backed-by-lambda-from-openapi-spec

I. UNDERSTANDING THE PROBLEM AND DOMAIN

First we will try to understand the problem, the domain, and the scope of the work we are addressing by asking questions.

1.1 WHAT WILL DININGBYFRIENDS API DO FOR ITS USERS?

DiningByFriends provides developers API to build Serverless SaaS application with personalized restaurant recommendations. When using DiningByFriends based SaaS applications the users have three main needs that the application must satisfy and the API must provide:

1. **Social network—**Users want to connect with friends who are also using the application. This functionality is similar to the way people connect with friends on any social network such as Twitter, LinkedIn, or Facebook.
2. **Restaurant recommendations—**Users want to create and look at reviews of restaurants and then get recommendations for a restaurant based on these reviews. This is the central service the API provides.

3. **Personalization—**Users want to rate the reviews of restaurants to indicate whether the review was helpful or not. Then they want to combine these reviews with their friends' ratings to receive personalized recommendations based on the restaurants their friends also like.

1.2 WHAT TYPES OF INFORMATION DOES THE API NEED TO RECORD TO PERFORM THESE TASKS?

To answer this question, DiningByFriends API should provide at least the following information:

1. All the basic identifying information about users, such as a name and a unique ID, so people can find and connect with them on the social network. (In a realworld scenario, this would likely include many additional attributes, but we keep it limited for this API.)
2. Restaurant identifiers and details, such as the name, address, and cuisine, to provide location-specific recommendations.
3. The text of the review, along with the rating and a timestamp of the rating in order to get personalized recommendations.
4. Reviews need to include ratings of its helpfulness (for example, up/down thumbs) so that friends know if a user agreed or disagreed with those reviews.

1.3 WHO ARE THE USERS OF THE APPLICATION DEVELOPERS WILL BUILD USING THESE APIs?

We have one type of user for the application the developer will build using our API. This includes users of the application who connect with friends, enter reviews, and receive recommendations.

NOTE: We know that nearly all complex applications have internal or system users of some sort. These can include system administrators, customer service personnel, and others responsible for the maintenance and operation of a complex technical solution. We have elected to ignore such requirements in an effort to streamline the design of the use case. We therefore only focus on the traditionally understood end user.

1.4 WHAT QUESTIONS DOES DININGBYFRIENDS APIs NEED TO ANSWER?

These questions about functionality fill in the details of how a user is going to interact with the system and what APIs will need to be built:

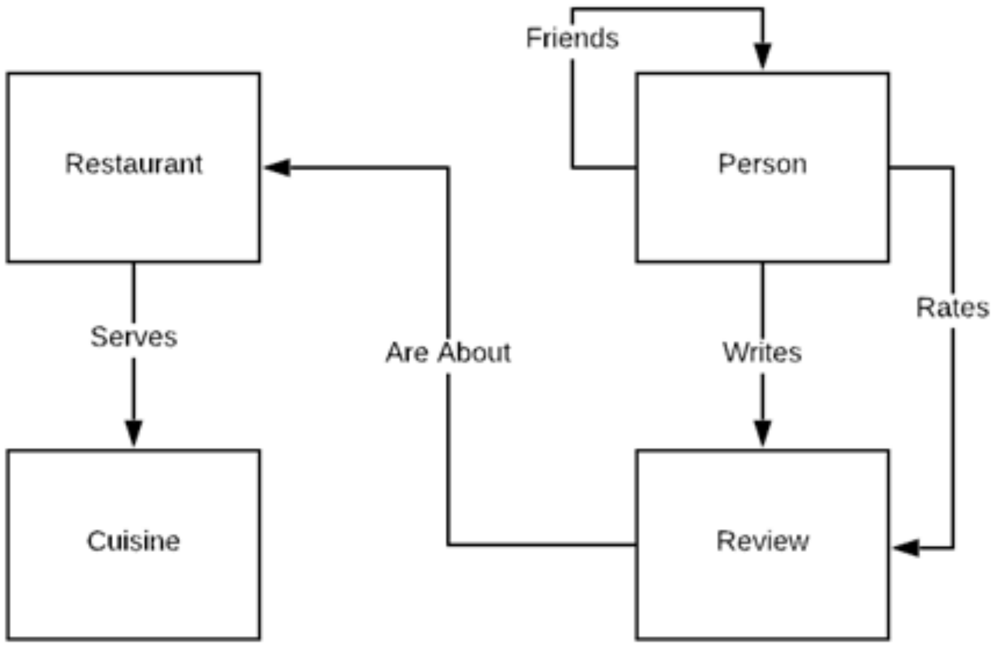
1. Who are my friends?
2. Who are the friends of my friends?
3. How is user X associated with user Y?
4. What restaurant near me with a specific cuisine is the highest rated?
5. Which restaurants are the ten highest-rated restaurants near me?
6. What are the newest reviews for this restaurant?
7. What restaurants do my friends recommend?
8. Based on my friends' review ratings, what are the best restaurants for me?
9. What restaurants have my friends reviewed or rated in the past X days?

For more details read the chapter 2 of the book Graph Databases in Action.

II. Developing the Database Model

2.1 Developing the Whiteboard Model

Using Hackolade (https://hackolade.com/help/TinkerPop.html) build the following white board conceptual data model shows the entities (boxes) and relationships (arrows) for DiningByFriends.



2.2 Constructing the Entity Relationship Diagram

Using DB Diagram (https://dbdiagram.io) now we are ready to build our entity relationship and translate those entities and relationships (developed in section 2.1) to the relational concepts of entities and relationships. The outcome of this process is another diagram, but this one contains sufficient detail to provide the necessary schema information for us to begin coding an implementation using a relational database i.e. AWS Aurora Serverless

III. Developing the OpenAPI Definition

Using Swagger OpenAPI Editor (https://swagger.io/tools/swagger-editor/) define a OpenAPI definition for DiningByFriends.

Note: The API at least must answer all the questions documented in section 1.4

IV. Implement, Test, and Continuously Deploy the GraphQL API

Using AWS CDK (Infrastructure as Code), Cognito (Authentication), VPC (Security), Aurora Serverless (Relational Database), API Gatway (GraphQL API), Lambda (Compute), EventBridge (Event Bus), and SQL to implement the DiningByFriends API using CDK (https://www.npmjs.com/package/@aws-cdk/aws-apigateway?activeTab=readme#openapi-definition).
As you add each API write automation test using:
https://www.readysetcloud.io/blog/allen.helton/dynamic-test-generation-with-oas/
https://opensource.com/article/18/6/better-api-testing-openapi-specification
https://github.com/stoplightio/prism
Build a CI/CD pipeline using these tests and GitHub Actions:
https://docs.github.com/en/actions

V. Visualize, Explore and Document the API

https://swagger.io/tools/swagger-ui/

VI. Generate the API SDKs for your Favorite Languages

https://swagger.io/tools/swagger-codegen/
https://github.com/anttiviljami/openapi-backend

This API Cloud Project will teach us how to build modern Serverless SaaS OpenAPIs and will help in thinking about and working with relational databases in conjunction with OpenAPI.