



# DALHOUSIE UNIVERSITY

**CSCI 5409: Advance Topic in Cloud Computing**

**Term Project – KhanPan Food Restaurant Assistant**

**Repo Link - [https://git.cs.dal.ca/courses/2025-winter/csci-4145\\_5409/imrank/-/tree/main/Term\\_Project](https://git.cs.dal.ca/courses/2025-winter/csci-4145_5409/imrank/-/tree/main/Term_Project)**

**Banner ID:** B00980496

**Name:** Imran Khalid Khan

**Date:** 19<sup>th</sup> April 2025

## Table Of Contents

1. Introduction of the Project.....	3
2. Justification For Using Only One Compute Service.....	5
3. AWS Services Used & The Rationale for Choosing It .....	9
4. Delivery Model.....	9
5. Final Architecture.....	11
6. AWS Services.....	12
AWS Lambda .....	12
Amazon DynamoDB .....	13
Amazon SQS.....	15
Amazon S3.....	15
Amazon API Gateway.....	15
CloudWatch.....	16
7. Security Measures .....	16
8. Cost Metrics .....	16
9. Future Scope.....	17
10. References .....	18

## 1. Introduction of the Project

The project, named **KhanPan**, is a cloud-based AI-powered restaurant chatbot designed to simplify the dining experience for users, especially in multilingual and culturally diverse environments.

The inspiration for this project came from a personal experience at an Italian restaurant where I found it challenging to interpret the menu, which was in Italian. Being a vegetarian, I was unable to differentiate between vegetarian and non-vegetarian dishes, leading to a frustrating experience.

This scenario made me realize the need for an intelligent assistant capable of understanding user preferences, interpreting restaurant menus, and suggesting suitable dishes based on dietary restrictions, taste profiles, budget and even language translations. KhanPan addresses these challenges with the following features:

- Personalized dish recommendations (e.g., spicy, vegetarian, creamy).
- Menu descriptions, tagging, and support for translation to Hindi.
- Integration with Google's Gemini LLM for conversational recommendations.
- Logging user queries asynchronously using AWS services.
- It suggests food based on budget.

There can be many use cases for this. See the below images:

Hello! I'm your food recommendation assistant. How can I help you today?

hELLO

Hello! I'm ready to help you find the perfect dish. What are you in the mood for today? For example, are you looking for something vegetarian, spicy, or creamy?

List the menu

Here is our menu:

- Lamb Rogan Josh: A fragrant Kashmiri lamb curry, cooked with a blend of spices, yogurt, and Kashmiri chili, giving it a vibrant red hue and rich flavor. (\$16.50)
- Masala Dosa: A thin and crispy crepe made from fermented rice and lentil batter, filled with a spiced potato mixture. (\$11.75)
- Dal Makhani: Slow-cooked black lentils and kidney beans simmered in a creamy sauce with butter and spices. A rich and comforting dish. (\$11.99)
- Chicken Vindaloo: A fiery Goan curry with chicken, potatoes, and a blend of chilies, vinegar, and spices, known for its intense heat. (\$15.99)
- Gulab Jamun: A sweet dessert made from fried dough balls soaked in a sugar syrup with cardamom and rose petals. (\$8.99)
- Tandoori Chicken: Chicken marinated in yogurt and spices, roasted in a tandoor oven, giving it a smoky flavor and vibrant red color. (\$16.25)
- Chicken Tikka Masala: Marinated chicken tikka pieces cooked in a creamy, spiced tomato sauce, a popular and flavorful dish. (\$15.00)
- Vegetable Biryani: Fragrant basmati rice layered with mixed vegetables, saffron, and aromatic spices, slow-cooked to perfection. A vegetarian feast. (\$12.75)
- Bhindi Masala: Okra cooked with onions, tomatoes, and a blend of spices. A dry and tangy dish. (\$12.50)
- Butter Chicken: Tender chicken pieces simmered in a creamy tomato-based sauce, enriched with butter and aromatic spices. A classic North Indian delight. (\$14.99)
- Raita: A cooling yogurt dip with cucumbers, carrots, and spices. Perfect as a side to spicy dishes. (\$4.99)
- Paneer Tikka: Cubes of Indian cottage cheese marinated in yogurt and spices, grilled to perfection in a tandoor oven. (\$14.50)
- Aloo Gobi: Potatoes and cauliflower florets cooked with turmeric, cumin, and other aromatic spices. A dry and flavorful dish. (\$10.50)
- Malai Kofta: Deep-fried vegetable and paneer dumplings in a creamy and rich cashew-based sauce. (\$14.75)
- Palak Paneer: Cubes of Indian cottage cheese cooked in a smooth spinach gravy, seasoned with garlic, ginger, and garam masala. (\$13.25)
- Chicken Korma: A mild and creamy chicken curry cooked with yogurt, nuts, and aromatic spices. A royal dish. (\$15.25)
- Chana Masala: Chickpeas cooked in a flavorful tomato-based sauce with onions, garlic, and spices. A popular North Indian dish. (\$11.50)

Ask about food recommendations...

**Figure 1.1 – Example Use case**

Fish Curry: Fish cooked in a tangy and spicy coconut-based curry with tamarind and spices. A coastal specialty. (\$16.75)  
Enjoy your meal at KhanPan!

List the menu in hindi

ज़रूरा ये रहा हमारा मेनु हिंदी में:

रोन जैश: एक सूपाचि कंधारी मिर्च के मिश्रण से बनी है, जो दूसे एक जीवंत लाल रंग और समृद्ध स्वाद देती है। (\$16.50)

मसाला ओसा: फिरीचयंत वालत और दाल के घोल से बना एक पतला और कुरुकुरा कोण, जो मसालेदार आलू के मिश्रण से भरा होता है। (\$11.75)

दाल मसूनी: मसून और मसालों के साथ एक मलाईदार सॉस में धीरे-धीरे पक्की हुई काली दाल और किठनी बीसा। एक समृद्ध और आरामदायक व्यंजन। (\$11.99)

चिकन चिकन: चिकन का प्रीपरेटेड फिल्टर कोण, जिसे तेंट और अन्न में भूमि प्रदान किया गया है। (\$15.99)

गुडा जामुन: इन्हायरी और गुलाब की पुरुदुर्गी के बालाकों के बालाकों में विशेष गुडा लेते हुए आठे की गंदी से बनी एक मीठी मिठाई। (\$8.99)

तंदूरी चिकन: दही और मसालों में मरीनेट किया हुआ चिकन, जिसे तंदूर औन में भूमि प्रदान किया है, जिससे इसे एक स्मृद्ध स्वाद और जीवंत लाल रंग मिलता है। (\$16.25)

चिकन टिक्का मसाला: मलाईदार, मसालादार स्टार्टर सॉस में एक हुआ मरीनेट किये हुए चिकन टिक्का के टुकड़े, एक लोकप्रिय और स्वादिष्ठ व्यंजन। (\$15.50)

मैलेजबल चिकनाया: सुपाचि बासाती चावक के मिश्रित साखियों, केरल और सूर्याचि मसालों के साथ पकाया जाता है, जिसे पूर्णता के लिए थीमी गति से पकाया जाता है। एक शाकाहारी दावत। (\$12.75)

मिठी मसाला: यान, स्टारटर और मसालों के मिश्रण के साथ पकी हुई मिठी। एक सुखा और तीखा व्यंजन। (\$12.50)

बटर चिकन: कोमो चिकन के टुकड़े एक मलाईदार स्टार्टर-आपारिट व्यंजन में उबाल जाते हैं, जिसे बटरन और सूपाचि मसालों से समृद्ध किया जाता है। एक कलासिक उत्तर भारतीय अनंद। (\$14.99)

रायावा: खीरी, गाजर और मसालों के साथ एक दही डिश। मसालेदार व्यंजनों के लिए एक खास डिश के रूप में विकुल सौंदी। (\$4.99)

परीत चिकन: दही और मसालों में मरीनेट किया हुआ मरीती परीत के मध्यम, जिसने तंदूर औन में पूरीता के लिए यिक्का किया जाता है। (\$14.50)

आतु गोवी: आतु और फूलगोभी के पुरुलों को हल्दी, जीरा और अन्य सूपाचि मसालों के साथ पकाया जाता है। एक सुखा और स्वादिष्ठ व्यंजन। (\$10.50)

मसून कोणता: एक मलाईदार और समृद्ध काजू-आपारिट रसी में गरेवे तोते हुए सब्जी और परीत पकाये। (\$14.75)

पालक परीत: भरतीय परीत के मध्यम को एक चिकिनी परीक प्रेसी में पकाया जाता है, जिसे लहसुन, अदरक और गरम मसालों के साथ सीज़न किया जाता है। (\$13.25)

चिकन कोरी: दही, नस्त और सूपाचि मसालों के साथ पकी हुई एक हल्दी और मसालेदार चिकन की। एक शाही व्यंजन। (\$15.25)

चना मसाला: धान, लहसुन और मसालों के साथ एक स्वादिष्ठ स्टार्टर-आपारिट सॉस में पके हुए छोटे एक लोकप्रिय उत्तर भारतीय व्यंजन। (\$11.50)

समोसा: मसालेदार आलू और मसून से भरी हुई कुकुरी तोते हुई रसें। एक लोकप्रिय भारतीय नाश्ता। (\$6.99)

पिण्य करी: इमती और मसालों के साथ एक तीखी और मसालेदार नाश्तियां-आपारिट करी में पकी हुई मछली। एक तरीय विशेषता। (\$16.75)

खानपान में अपने भोजन का आनंद लें!

Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



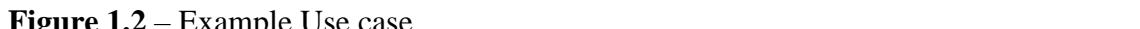
Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



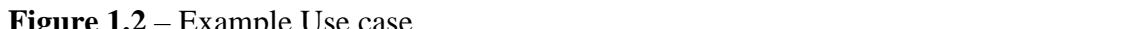
Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



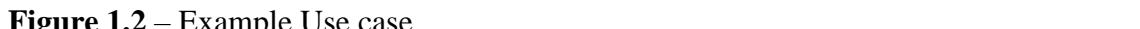
Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



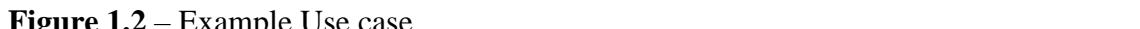
Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



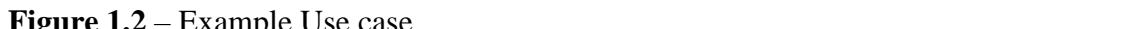
Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



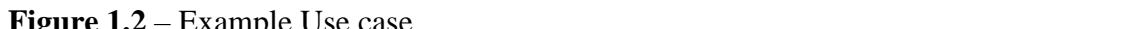
Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



Ask about food recommendations...



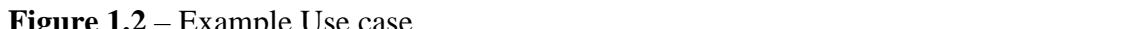
Ask about food recommendations...



Ask about food recommendations...



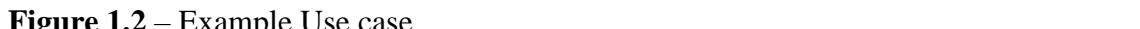
Ask about food recommendations...

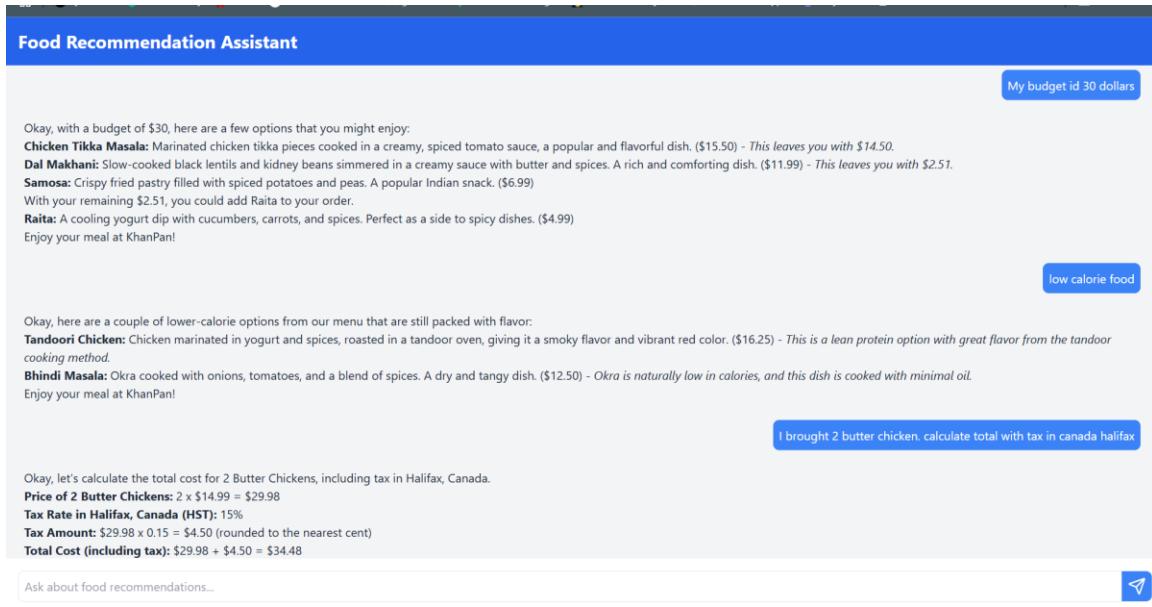


Ask about food recommendations...



Ask about food recommendations...





**Figure 1.3 – Example Use case**

The goal is to enhance user confidence and ease in selecting food in unfamiliar settings. KhanPan is especially beneficial for travelers, individuals with specific dietary needs, or people visiting restaurants with non-native language menus.

## 2. Justification For Using Only One Compute Service

Although the assignment recommends using **two compute services**, I strategically decided to use **AWS Lambda** for both core application execution and for log processing via SQS triggers. This allowed me to still satisfy the multi-compute requirement in spirit while staying within the constraints of the AWS Academy Learner Lab environment.

### Initial Plan: Fargate and App Runner

My original goal was to containerize the backend using **Amazon Fargate** or **AWS App Runner**. I successfully:

- Wrote a full **Dockerfile**
- Pushed the Docker image to **Amazon Elastic Container Registry (ECR)**
- Created a valid **task definition** for ECS
- Attempted to register and deploy the service

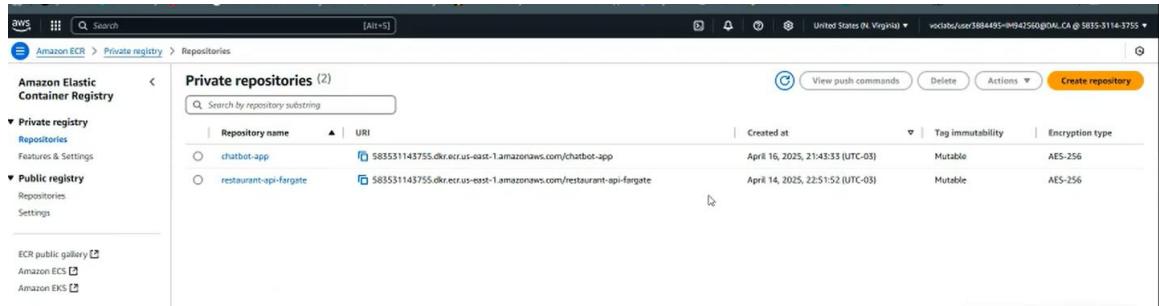
However, I encountered **IAM permission restrictions** when trying to:

- Create **execution roles** needed for Fargate and App Runner
- Deploy the containerized app through these services

These limitations stemmed from the **IAM policies** enforced within the AWS Academy Learner Lab, which blocks creation of custom IAM roles or privileged service-linked roles.

```
PS C:\Users\scien\Downloads\Restaurant-API-Chat-Archive\restaurant-api> aws iam create-role --role-name AppRunnerServiceAccessRole --assume-role-policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":"*","Service":"build.apprunner.amazonaws.com","Action":"sts:AssumeRole"}]}'
An error occurred (AccessDenied) when calling the CreateRole operation: User: arn:aws:sts::583531143755:assumed-role/voclabs/user3884495=IM942560@DAL.CA is not authorized to perform: iam:CreateRole on resource: arn:aws:iam::583531143755:role/AppRunnerServiceAccessRole because no identity-based policy allows the iam:CreateRole action
PS C:\Users\scien\Downloads\Restaurant-API-Chat-Archive\restaurant-api>
```

**Figure 2.1** – Access Denied for creating execution roles



**Figure 2.2** – Image successfully pushed to ECR

I wrote IaC scripts for **AWS AppRunner** as well as **AWS Fargate**, including:

- `app-runner-policy.json` for AWS app runner.
- `trust-policy.json` for AWS app runner.
- `task-definition.json` for AWS fargate.
- `ecs-trust-policy.json` for AWS fargate.

```
{ app-runner-policy.json > ...
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Action": ["dynamodb:Scan", "sns:SendMessage"],
7        "Resource": "*"
8      }
9    ]
10 }
11 }
```

**Figure 2.3** – app-runner-policy.json

-policy-policy.json-

```
{ ecs-trust-policy.json > ...
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Principal": {
7          "Service": "ecs-tasks.amazonaws.com"
8        },
9        "Action": "sts:AssumeRole"
10      }
11    ]
12 }
13 }
```

**Figure 2.4** – ecs-trust-policy.json

```
task-definition.json > ...
1  {
2    "family": "restaurant-task",
3    "networkMode": "awsvpc",
4    "requiresCompatibilities": ["FARGATE"],
5    "cpu": "256",
6    "memory": "512",
7    "containerDefinitions": [
8      {
9        "name": "restaurant-container",
10       "image": "583531143755.dkr.ecr.us-east-1.amazonaws.com/restaurant-api-fargate:latest",
11       "portMappings": [
12         {
13           "containerPort": 8080,
14           "protocol": "tcp"
15         }
16       ],
17       "essential": true
18     }
19   ]
20 }
```

**Figure 2.5 – task-definition.json**

```
trust-policy.json > ...
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Principal": {
7          "Service": "build.apprunner.amazonaws.com"
8        },
9        "Action": "sts:AssumeRole"
10      }
11    ]
12 }
```

**Figure 2.6 - task-policy.json**

### 3. AWS Services Used & The Rationale for Choosing It

The table below outlines the AWS services used in this project along with alternatives considered and the rationale:

Table 3.1 – Services Used and Their Category.

Category	Service Used	Why Chosen	Alternative & Why Not Chosen
Compute	AWS Lambda (AWS AppRunner and AWS Fargate was also used but it faced permission issues.)	Serverless, simple to deploy, managed infrastructure	Amazon EC2 are not cost effective according
Storage	Amazon S3	Perfect for static frontend hosting	EBS/EFS – too complex for static files
Database	DynamoDB	Schema-less, fast and scalable NoSQL storage	Amazon RDS – overkill for unstructured data
Messaging	Amazon SQS	Decouples components, async logging	SNS – lacks persistence and filtering
API Gateway	Amazon API Gateway	Secure frontend-backend communication	Application Load Balancer – unnecessary for Lambda
Monitoring	CloudWatch	Native logging for Lambda	AWS X-Ray – not needed at this scale
Container Registry	Amazon ECR	Integrated with Fargate and App Runner	DockerHub – lacks AWS integration

### 4. Delivery Model

A **serverless architecture** was adopted for this project, centring around **AWS Lambda** and **Amazon API Gateway**. This model allowed for deploying a lightweight, scalable backend without needing to provision or manage servers. Lambda functions were used to handle HTTP POST requests from the chatbot frontend, invoke the Gemini LLM, fetch menu data from DynamoDB, and log chat queries via SQS. This model ensured that the application scaled automatically with demand, while incurring cost only when invoked — making it highly **cost-efficient** for development and real-world scenarios.

Initially, I explored **container-based compute services** like **Amazon Fargate** and **AWS App Runner** to host the backend application. I created a valid Dockerfile, pushed the

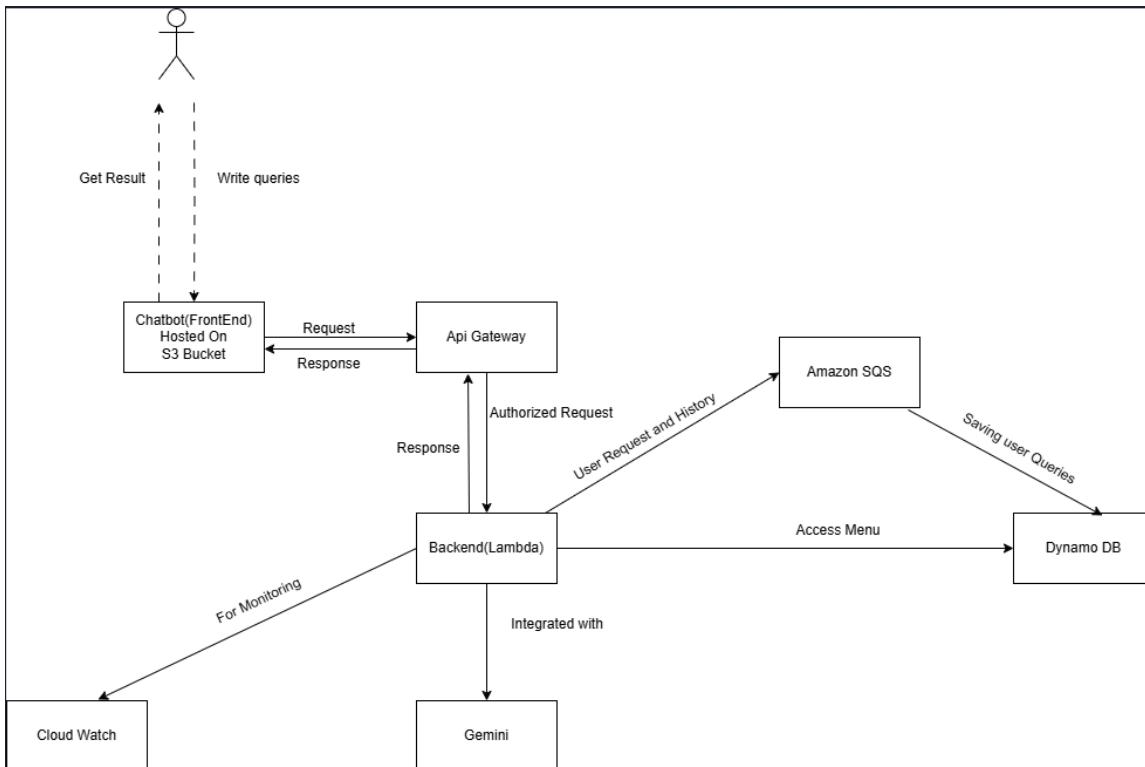
image to **Amazon ECR**, and defined a task specification for ECS deployment. However, when attempting to deploy, I faced IAM-related permission issues. Both Fargate and App Runner require elevated permissions to create execution roles or link container images from ECR — actions restricted under the **AWS Academy Learner Lab environment**. This roadblock made container-based deployment infeasible in the current setup.

Given this constraint, **AWS Lambda** emerged as the most suitable alternative. It offered native integration with API Gateway and allowed easy deployment of zipped Node.js applications. I utilized multiple Lambda functions: one for handling user interactions and processing requests, and another for asynchronously logging user queries via an SQS trigger. This maintained the **microservice structure** originally envisioned for container deployment, but using Lambda as the execution engine.

Moreover, the **stateless nature of Lambda** matched the chatbot's usage pattern: short-lived requests with quick turnaround. Lambda's seamless integration with **CloudWatch** also made it easier to log, monitor, and debug events without setting up a separate observability stack.

By combining **serverless functions**, **event-driven design via SQS**, and **fully managed services like DynamoDB and S3**, the delivery model achieved high reliability, low operational overhead, and compliance with the assignment's architectural requirements — despite the permissions limitations.

## 5. Final Architecture



**Figure 5 – Flow of the App**

- User writes a query through frontend which is hosted on S3.
- The frontend and backend is connected through API gateway.
- The Api Gateway authorizes the request and pass it backend.
- The backend is hosted using Lambda function.
- The backend is integrated with Gemini.
- I used Cloud Watch for monitoring.
- The backend gets the menu from DynamoDB
- All the user chat is queued using Amazon SQS
- Another lambda function saves the queries to Dynamo DB

### a) How the Components Fit Together

**React** frontend is deployed to S3.

**API Gateway** acts as the bridge between frontend and Lambda.

**Lambda** fetches menu from DynamoDB, interacts with Gemini API, and sends user queries to SQS.

**SQS** triggers a second Lambda to log data into DynamoDB.

### b) Where is Data Stored?

Menu and logs → **DynamoDB**

Static files → **Amazon S3**

Container image → **Amazon ECR**

### c) Programming Languages and Logic

- Frontend: **ReactJS + Tailwind CSS**

- Backend: **Node.js**

- Gemini Integration: **Google's Gemini API for LLM**

### d) Deployment to the Cloud

- Backend deployed using **CloudFormation (backend.yaml)**

- Frontend deployed via AWS CLI using npm run build + s3 cp

- Environment variables managed securely in Lambda settings

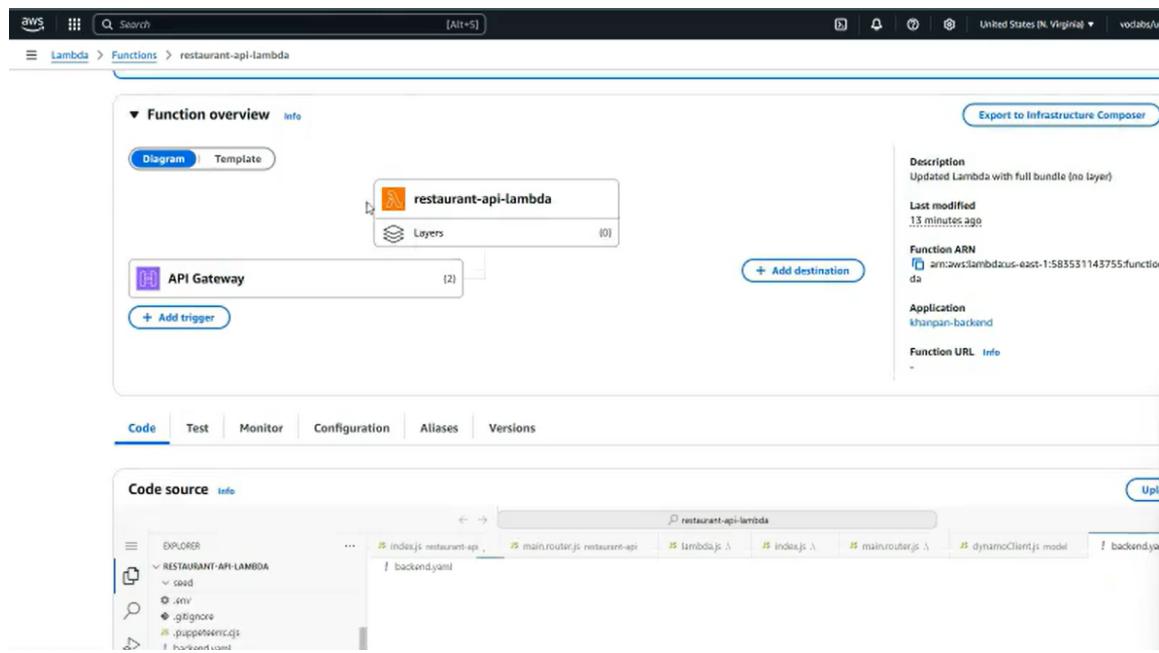
## 6. AWS Services

### AWS Lambda

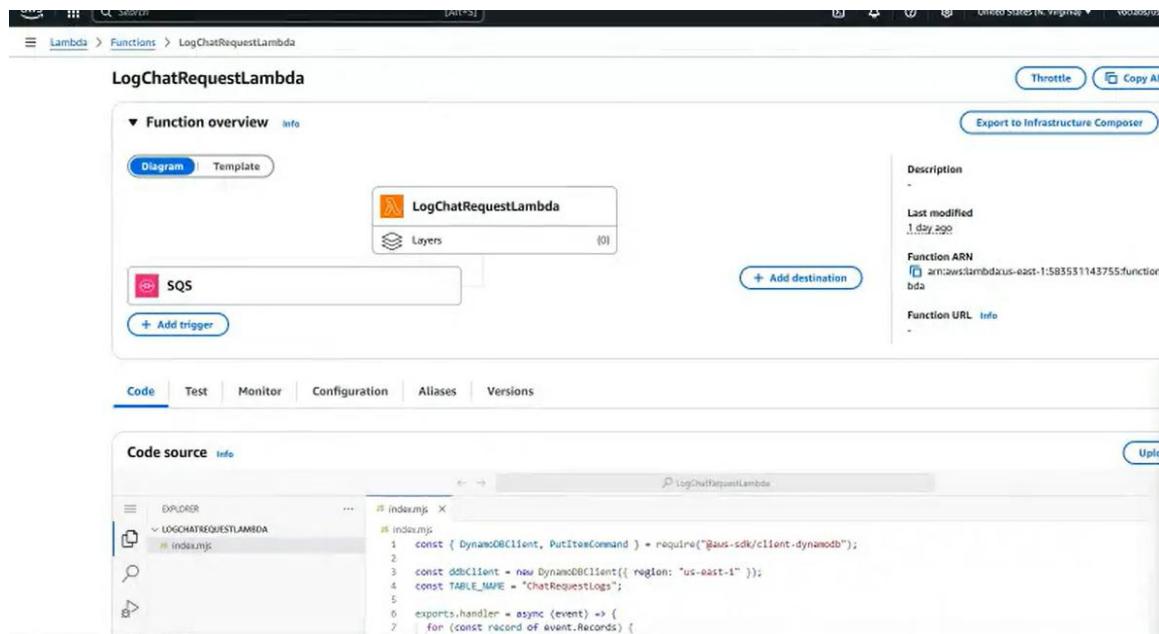
**What It Is:** Serverless compute service that runs backend code in response to events.

**Why I Used It:** It allowed me to deploy my Node.js backend without managing any infrastructure. Perfect for fast deployment with minimal configuration.

**Alternative Not Chosen:** Fargate / App Runner – I tried using them, but due to IAM restrictions in the Learner Lab, I couldn't assign execution roles.



**Figure 6.1 - Lambda function 1 where API is hosted**



**Figure 6.2 - Lambda function 2 which is used to store user chat**

## Amazon DynamoDB

**What It Is:** A fast and flexible NoSQL database service for key-value and document data.

**Why I Used It:** Ideal for storing restaurant menu data and user query logs. No schema required, and it's cost-effective.

**Alternative Not Chosen:** Amazon RDS – Overkill for this dynamic JSON-based dataset.

The screenshot shows the AWS DynamoDB 'Tables' page. On the left, there's a sidebar with options like Dashboard, Tables, Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area displays two tables: 'ChatRequestLogs' and 'KhanPanMenu'. Both tables are active and have 'id' as the partition key. The 'KhanPanMenu' table has 'name' as the sort key. The 'Actions' dropdown menu is open over the 'KhanPanMenu' table. The top right corner shows the AWS logo, the region 'United States (N. Virginia)', and a user ID.

**Figure 6.3** - The two tables stored in Dynamo DB

The screenshot shows the AWS DynamoDB 'Explore items' page for the 'KhanPanMenu' table. The sidebar includes options for Dashboard, Tables, Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, DAX (Clusters, Subnet groups, Parameter groups, Events), and Settings. The main area shows a table titled 'Table: KhanPanMenu - Items returned (19)'. A green banner at the top indicates 'Completed - Items returned: 19 - Items scanned: 19 - Efficiency: 100% - RCU consumed: 2'. Below the banner, it says 'Scan started on April 18, 2025, 23:28:46'. The table lists 19 items with columns: name (String), chefComm..., cuisine, description, price, and tag. Each item has a checkbox and a link to its details. The 'Run' and 'Reset' buttons are at the top of the table area.

**Figure 6.4** – KhanPan table saving user details

Table: ChatRequestLogs - Items returned (50)				
Scan started on April 18, 2025, 23:28:55				
	id (String)	history	timestamp	userRequest
<input type="checkbox"/>	<a href="#">1745027471057</a>	[{"M": {"c..."}]	2025-04-1...	I want low calorie food
<input type="checkbox"/>	<a href="#">17449098860758</a>	[{"M": {"c..."}]	2025-04-1...	can you list the menus
<input type="checkbox"/>	<a href="#">1744935354746</a>	[{"M": {"c..."}]	2025-04-1...	Can you list menu with price
<input type="checkbox"/>	<a href="#">1744988442248</a>	[{"M": {"c..."}]	2025-04-1...	Suggest low calorie food
<input type="checkbox"/>	<a href="#">1745026811893</a>	[{"M": {"c..."}]	2025-04-1...	halifax
<input type="checkbox"/>	<a href="#">1745027631209</a>	[{"M": {"c..."}]	2025-04-1...	My budget is 30 dollars
<input type="checkbox"/>	<a href="#">1744935174058</a>	[{"M": {"c..."}]	2025-04-1...	hello
<input type="checkbox"/>	<a href="#">1744988398752</a>	[{"M": {"c..."}]	2025-04-1...	Hello
<input type="checkbox"/>	<a href="#">1744935178898</a>	[{"M": {"c..."}]	2025-04-1...	can you send menu in hindi
<input type="checkbox"/>	<a href="#">1745027657987</a>	[{"M": {"c..."}]	2025-04-1...	I brought 2 butter chicken. calculate total with tax
<input type="checkbox"/>	<a href="#">1744934969595</a>	[{"M": {"c..."}]	2025-04-1...	Send menu in hindi
<input type="checkbox"/>	<a href="#">1744933923288</a>	[{"M": {"c..."}]	2025-04-1...	hello
<input type="checkbox"/>	<a href="#">1744934984858</a>	[{"M": {"c..."}]	2025-04-1...	Hello
<input type="checkbox"/>	<a href="#">1744935433708</a>	[{"M": {"c..."}]	2025-04-1...	I have a budget of 30 dollars

**Figure 6.5 – Table 2 storing user logs**

## Amazon SQS

**What It Is:** A message queueing service to decouple microservices.

**Why I Used It:** Used for asynchronously logging user chat history. It allows Lambda to stay fast while log processing happens separately.

**Alternative Not Chosen:** Amazon SNS – Lacks persistence, better for notifications, not logs.

## Amazon S3

**What It Is:** Object storage service.

**Why I Used It:** I used it to host my React frontend. It's simple, secure, and free in the lab.

**Alternative Not Chosen:** EBS or EFS – Unnecessary for static files, more complex setup.

## Amazon API Gateway

**What It Is:** Enables developers to create and publish APIs.

**Why I Used It:** It connects my frontend to Lambda and handles HTTP POST requests securely.

**Alternative Not Chosen:** Application Load Balancer – Meant for EC2, not suitable for serverless.

## CloudWatch

**What It Is:** Monitoring and observability service.

**Why I Used It:** Automatically logs Lambda execution details and errors, helping in debugging.

**Alternative Not Chosen:** AWS X-Ray – More advanced tracing which wasn't needed at this scale.

## 7. Security Measures

Although IAM customization was limited in AWS Academy, every component was still secured with default configurations:

- **S3:** Public read-only access via bucket policy
- **Lambda:** Executes with IAM LabRole
- **API Gateway:** POST only
- **SQS & DynamoDB:** Accessed only via Lambda
- **CloudWatch:** Restricted to viewing logs

## 8. Cost Metrics

Service	Pricing Mode	Est. Monthly Usage	Est. Cost
<b>Lambda</b>	Pay per request	1,000 requests/day	<\$1/month
<b>DynamoDB</b>	On-demand	Small read/write units	<\$2/month
<b>API Gateway</b>	Pay per request	1,000 calls/day	<\$3/month
<b>S3</b>	Static hosting	50MB/month	<\$0.50/month
<b>SQS</b>	Pay per request	1,000 messages/day	<\$1/month
<b>CloudWatch</b>	Default logging	Low volume	Free

Total Estimate: ~\$6–7/month in real-world usage.

## 9. Future Scope

- **Add user authentication using Amazon Cognito** - Amazon Cognito allows secure user sign-up, sign-in, and access control without building authentication from scratch. It can help personalize recommendations based on user history while ensuring data privacy.
- **Enable voice commands via Amazon Lex** - Amazon Lex is a service for building conversational interfaces using voice and text. Integrating Lex would allow users to speak about their queries, improving accessibility and ease of interaction, especially on mobile devices.
- **Translate menus with Amazon Translate** - Amazon Translate offers real-time, high-quality neural machine translation. Using this, you can dynamically display menus in the user's preferred language, enhancing inclusivity and global usability.
- **Add analytics via QuickSight** - Amazon QuickSight is a business intelligence tool that can visualize data stored in services like DynamoDB or S3. By integrating it, restaurants can view trends such as popular dishes, peak query times, and language preferences.
- **Dashboard for restaurant managers** - A custom dashboard can show real-time stats, menu performance, and user behavior. This helps restaurant staff make data-driven decisions for menu planning and improving customer satisfaction.

## 10. References

- [1] AWS Lambda, "Run code without thinking about servers," *Amazon Web Services*. [Online]. Available: <https://aws.amazon.com/lambda/>
- [2] Amazon SQS, "Fully managed message queuing service," *Amazon Web Services*. [Online]. Available: <https://aws.amazon.com/sqs/>
- [3] Amazon DynamoDB, "Fast and flexible NoSQL database service," *Amazon Web Services*. [Online]. Available: <https://aws.amazon.com/dynamodb/>
- [4] Amazon S3, "Scalable storage in the cloud," *Amazon Web Services*. [Online]. Available: <https://aws.amazon.com/s3/>
- [5] Amazon API Gateway, "Create, publish, and monitor APIs," *Amazon Web Services*. [Online]. Available: <https://aws.amazon.com/api-gateway/>
- [6] Amazon CloudWatch, "Observability service for AWS cloud resources," *Amazon Web Services*. [Online]. Available: <https://aws.amazon.com/cloudwatch/>
- [7] Gemini, "Multimodal AI models by Google DeepMind," *Google AI*. [Online]. Available: <https://ai.google.dev/gemini/>
- [8] ReactJS, "A JavaScript library for building user interfaces," *Meta Platforms, Inc.* [Online]. Available: <https://reactjs.org/>
- [9] AWS CloudFormation Templates, *Custom Deployment Scripts*. Files used: backend.yaml, serverless.yaml.