DALHOUSIE UNIVERSITY

CSCI 5410: Serverless Data Processing

Project: Trivia Titans

Multi-Cloud Serverless Collaborative Trivia Challenge Game

Final Report

4th August 2023

Group: 03

Keyur Pradipbhai Khant	B00935171
Viral Siddhapura	B00913032
Kush Sutaria	B00928066
Shreyas Balaji Nagaraja	B00928044
Nilesh Kopparty	B00922710

Course Instructor: Dr. Saurabh Dey

Table of Contents:

1. Abstract:

"Trivia Titans" is an online trivia game platform designed to foster collaborative learning and engagement. The initiative is designed for fans of trivia who compete in real-time quizzes in teams. Along with other crucial features, the system includes user identification, profile administration, team management, and interactive gameplay.

Users can sign up using social media accounts or email addresses, ensuring an easy and secure login process. The project employs a 2-factor authentication system for increased security. User profiles provide personalized experiences, and team affiliations allow users to collaborate and compete together.

One of the project's core modules is the Trivia Game Lobby, where players can browse and join available trivia games created by administrators. During the games, team members can collaborate in real-time, answering multiple-choice trivia questions within a specified time frame. The leaderboard module showcases top-performing teams and players globally and in category-specific areas.

Administrators can create and manage trivia games with custom settings and monitor gameplay data and user engagement. This project's unique features include automated question tagging and virtual assistance via chatbots, offering navigation support and dynamic database search.

The project extensively uses cloud services and serverless technologies, including AWS and GCP for various modules. These services ensure scalability, fault tolerance, and data security. Key services include AWS Cognito, GCP - Firebase Authentication, DynamoDB, Firestore, AWS Lambda, GCP Cloud Functions, Pub/Sub, SQS, SNS, and Open AI's ChatGPT for AI-generated team names.

Overall, the "Trivia Titans" project blends trivia gaming with modern technology, providing a competitive, collaborative, and engaging platform for trivia enthusiasts worldwide.

2. Feature Specifications:

2.1 User Authentication:

User Authentication Module

The User Authentication module is a crucial component of our application, responsible for ensuring secure access to user accounts and safeguarding sensitive information. For this module, we have chosen Option 2 with GCP (Google Cloud Platform) services for its robust and reliable authentication capabilities.

1. Authentication Flow:

The authentication flow begins when a user accesses the application and clicks on the "Sign Up" or "Log In" option. GCP Firebase Authentication handles the sign-up and login process seamlessly, allowing users to create new accounts or log in using their existing credentials, including social media accounts and email addresses.

2. User Data Storage:

Upon successful authentication, GCP Firestore is utilized to store user profile information, including user names, email addresses, and other relevant personal details. Firestore is a scalable NoSQL database that ensures efficient data storage and retrieval for our application.

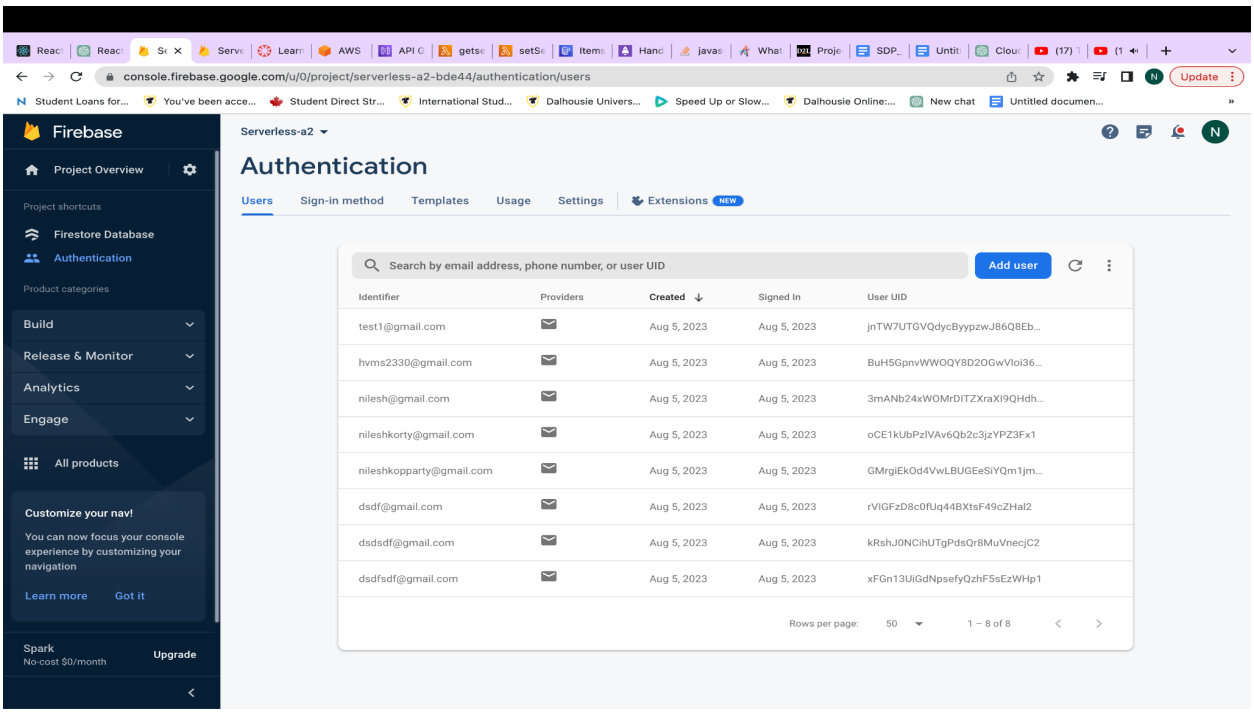


Figure 1: Authentication firebase storage

3. Second-Factor Authentication (2FA):

To enhance security, our application incorporates a second-factor authentication (2FA) mechanism. The user is required to provide answers to three predefined questions during the sign-up process. The question-answer pairs are stored securely in AWS DynamoDB, a fast and scalable NoSQL database offered by AWS (Amazon Web Services).

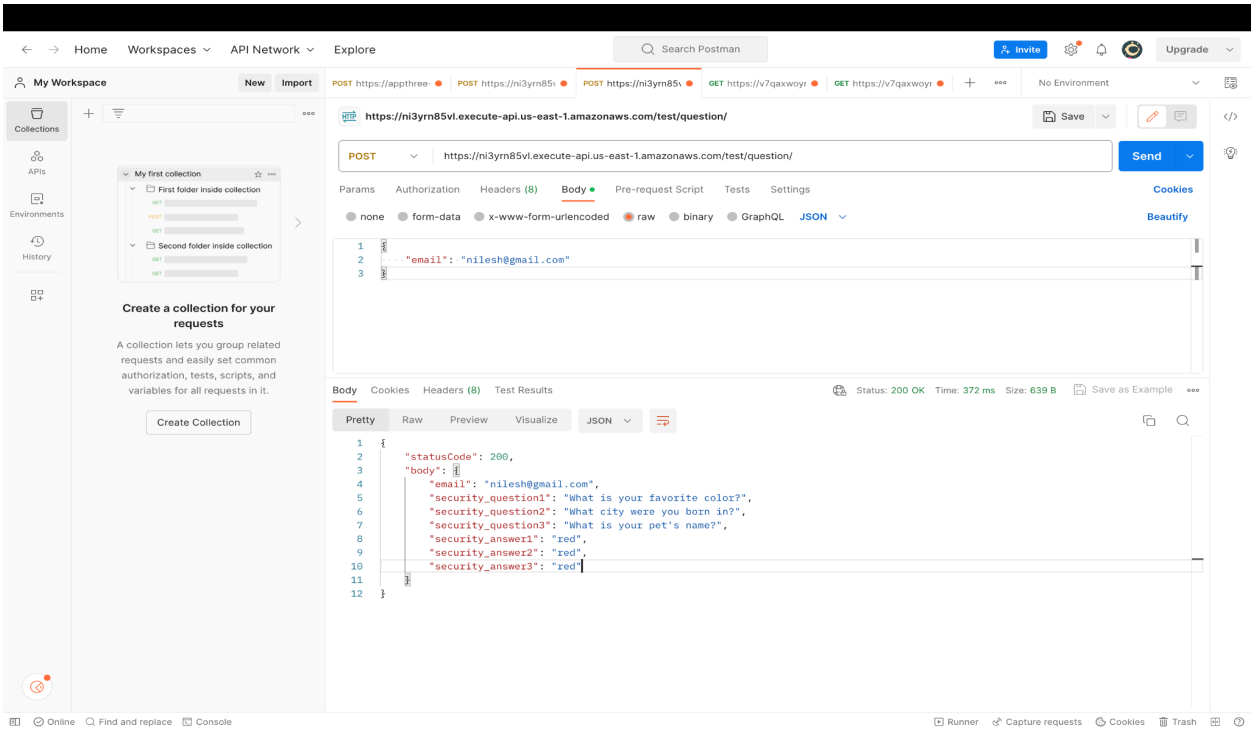


Figure 2: Postman API testing

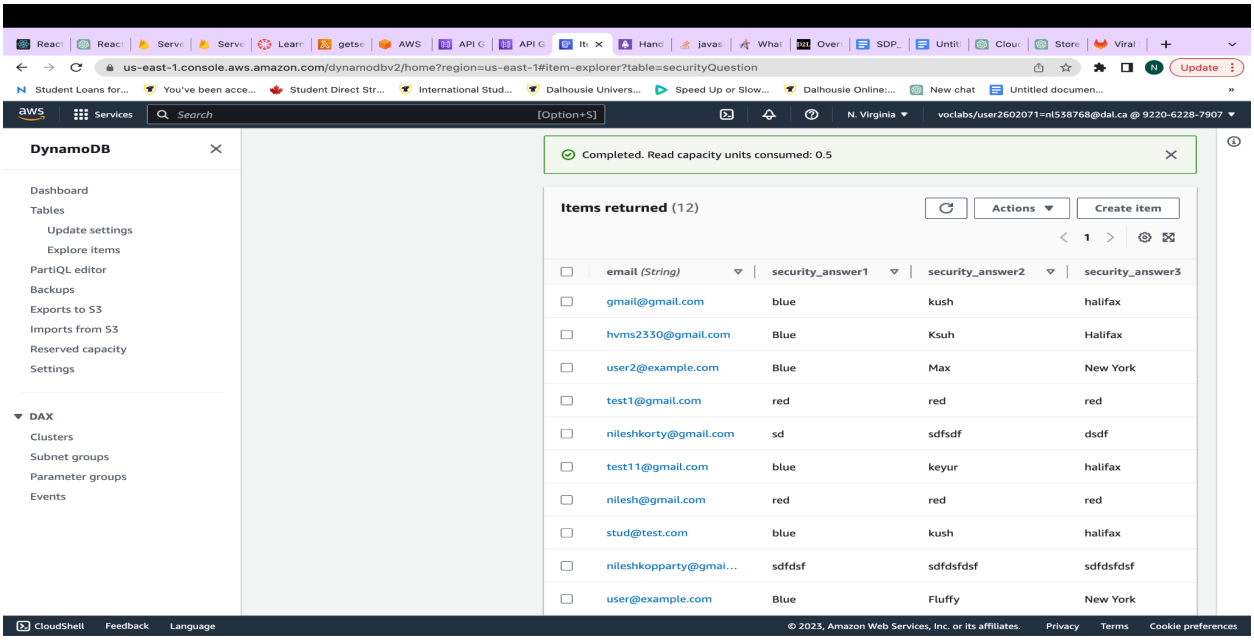


Figure 3: DynamoDB storage for user authentication

4. Second-Factor Validation:

When a user attempts to log in after successful email and password verification, our application triggers an AWS Lambda function to validate the second-factor questions and answers from the DynamoDB database. This process ensures that the user has correctly set up the 2FA and provides an additional layer of security for their account.

5. Implementation:

The implementation of the User Authentication module is achieved through integration with GCP Firebase Authentication and AWS DynamoDB and Lambda. The necessary APIs and triggers have been set up to facilitate smooth user authentication and 2FA validation processes.

6. Test Cases:

To ensure the robustness and reliability of our User Authentication module, we have performed extensive testing with various scenarios:

- Test User Sign Up with email and password.
- Test User Sign Up with social media accounts (e.g., Google, Facebook).
- Test User Login with email and password.
- Test User Login with social media accounts.
- Test 2FA setup during Sign Up.
- Test 2FA validation during Log In.

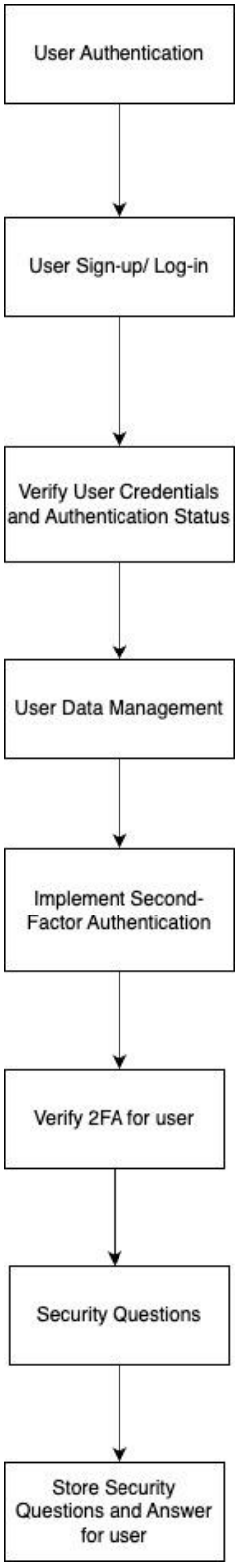


Figure 4: Application Flow diagram

The User Authentication module plays a crucial role in providing a secure and seamless user experience for our application. With the integration of GCP Firebase Authentication for primary authentication and AWS DynamoDB and Lambda for 2FA validation, we have established a robust and reliable authentication mechanism, ensuring the protection of user accounts and data. Through extensive testing, we have verified the module's effectiveness, and it stands ready to provide a secure foundation for our application's overall functionality.

2.2 User Profile Management:

User profile management involves using GCP firestore and cloud functions to fetch the details of the user who are currently registered and are actively playing.
This offers 4 main features.

1. User stats display
2. User details display and edit
3. Team stats display
4. Comparisons with users

To view the user stats we have to click the profile option,and it will be first thing displayed, it will send a post request to gcp firestore with the logged in mail and fetch the details regarding that id and display, it involves games played, games won, games lost and points earned.
User info contains each user's name, email, number and profile photo. Profile picture of the user will be stored in gcp cloud storage and the link will be supplied to user json document as a value, when user profile will be displayed, name and number will be fetched from firesstore and picture will be fetched from cloud storage.
Team stats can be viewed on the right side of profile page, will be fetched from gcp firestore and will supply all the team name the user is part of, after that another post request will go to teams collection with each team name and extract its specifics.
Comparison with other users will be visible on the bottom of page,it is just all data fetched and matched against the current user.

Used Cloud Services:

GCP – Firestore + Cloud Function

Justification:

There are 2 main reasons to prefer Firestore over DynamoDB

1. **Real time updates**

Real time update in firestore comes out of the box and changes to the data will be immediately pushed to connected clients while DynamoDB supports real-time updates through DynamoDB Streams, integrating it with clients can be a tedious task.

2. **Structure Querying Simplicity**

Firestore document structure is simple you can store any kind and any structure of data which will be identified by document name while DynamoDB involves creation of partition key to identify the document

Two reasons to prefer Cloud function over lambda are -

1. **Language support**

Cloud Functions provides larger language support for even languages like Ruby and .NET while lambda function language support is limited.

2. Billing details

The billing details are more descriptive with the invocation pattern graph and the cost of hosting it in cloud function is less compared to lambda.

Implementation:

Only thing needed from the user side here is to click the profile option, once clicked the data regarding user stats, team stats, user details and comparison details will be fetched by calling 3 different cloud functions.

To get user, teams data:

Once the request is sent it will fetch data from both user and team collection and the details are stored in the hook for further retrieval and display.

Firestore structure:

Only difference here is that the user document consists of a key named pic whose value is the link of a photo in GCP cloud storage; others are just key-value pairs.

Editing user details:

Inorder to edit user details, user has to give input for both name and number but has to upload a picture to change the profile picture which can be chosen using select prompt, once update request is submitted it will change the name and number in firstore and will upload the photo to cloud storage and supply that link to the firestore document so that every time it is fetched it will extract the correct image.

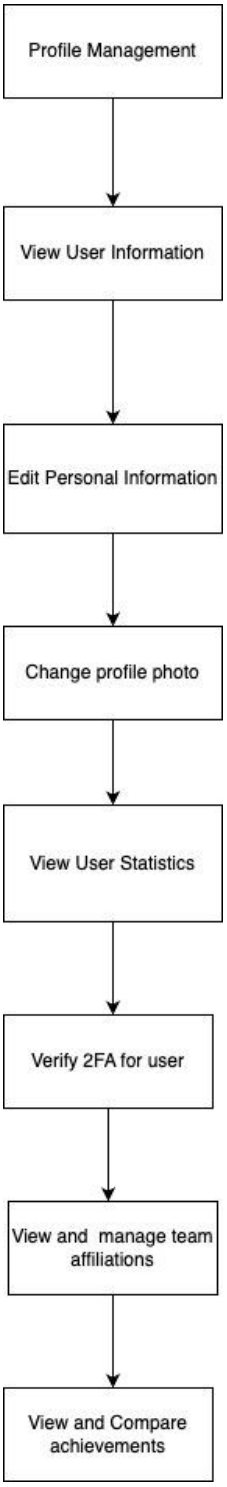


Figure 5: Application Flow diagram for user profile management

Test Case:

Table 1: User Profile Management Test Case

Test	Description	Figures
User clicks on Profile option	User should be able to access profile info	[10]
User will view the User details	Users should be able to view all data concerning them	
User Team association visibility	User must be able to see the performances of the teams	
User compares stats	User must be able to compare their stats with others	
User edits details	User can change the name and number	
User changes profile photo	User must be able to change profile photo	

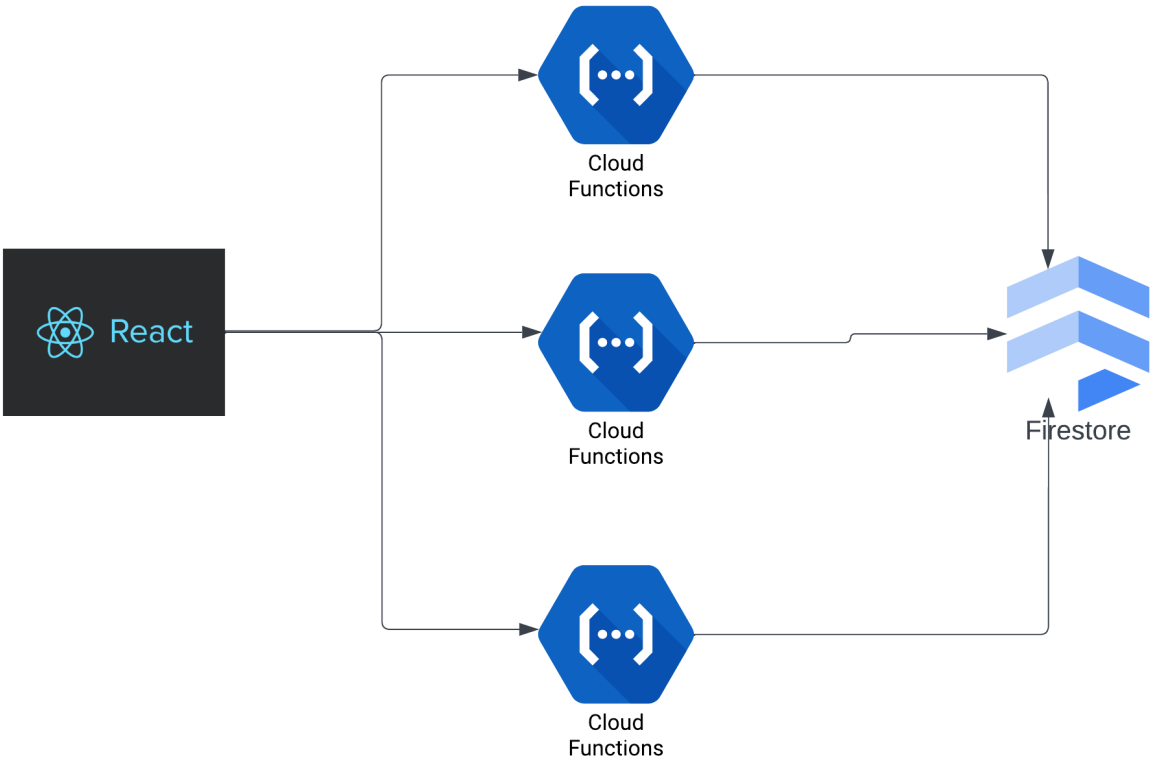


Figure 6: User Profile Management Architecture

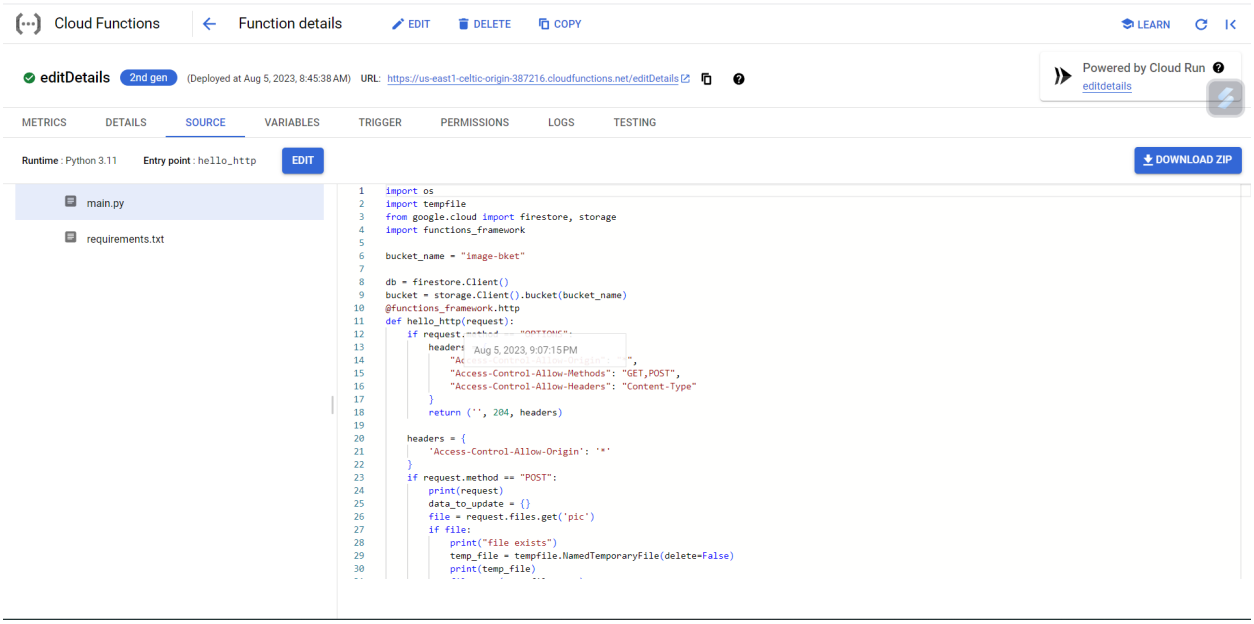


Figure 7: Cloud Function for user profile for edit profile

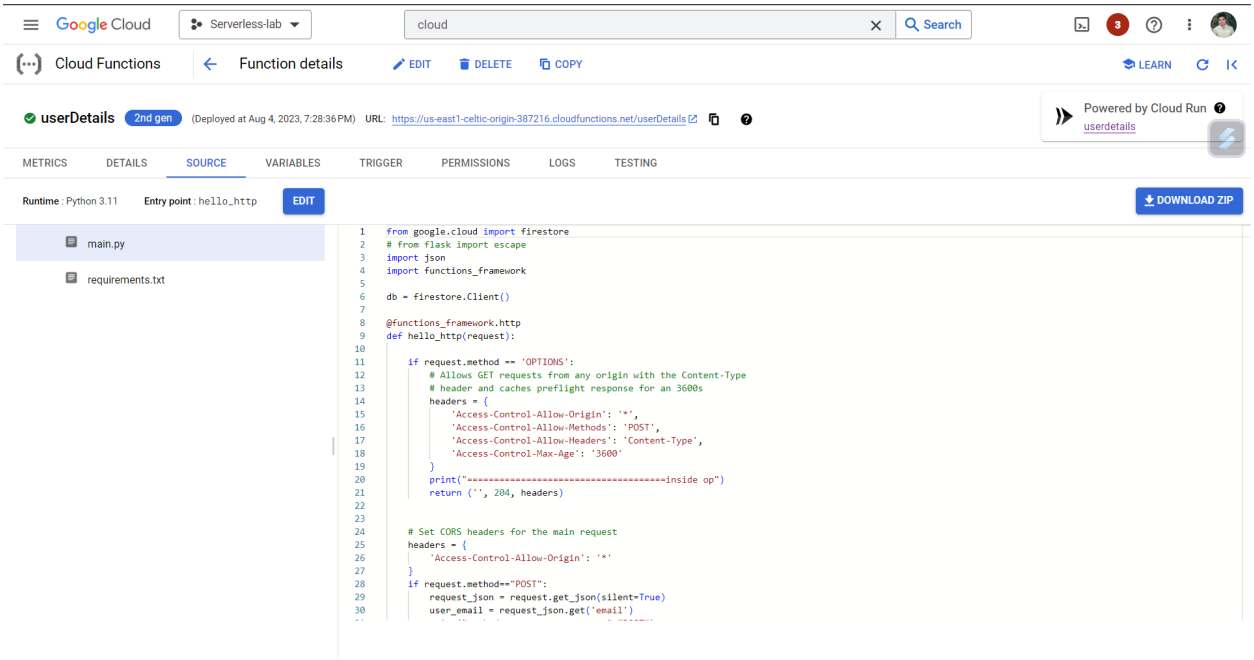


Figure 8: Cloud Function for user profile for user details

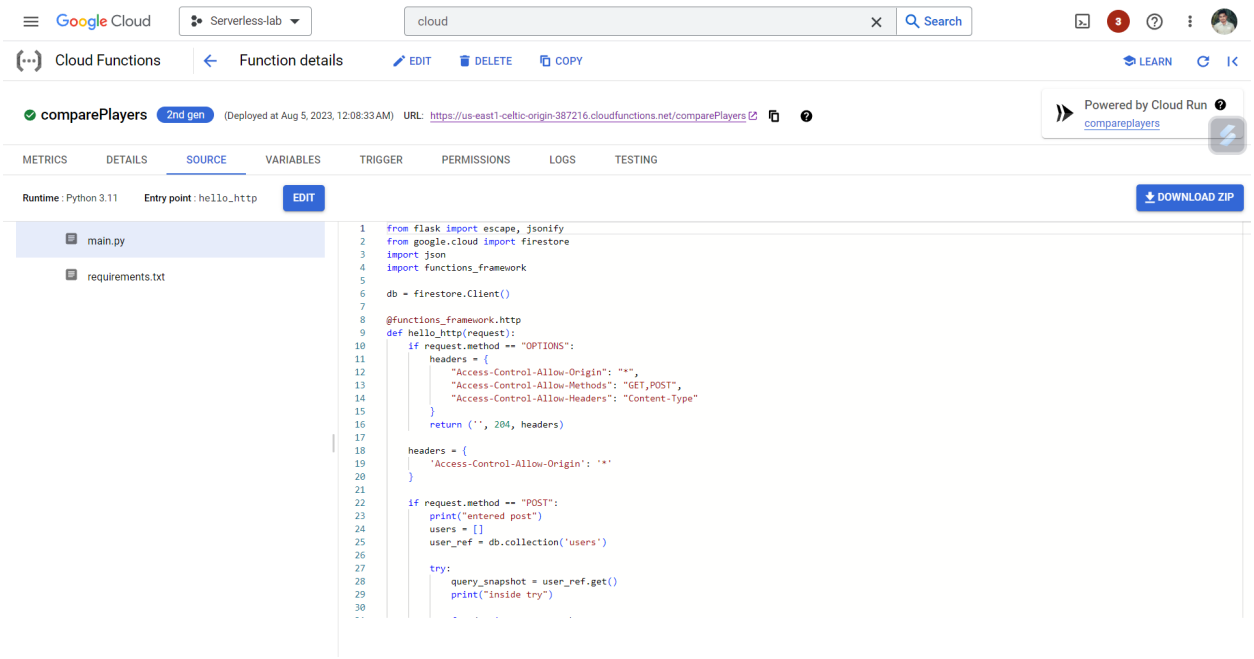


Figure 9: Cloud Function for user profile for compare players

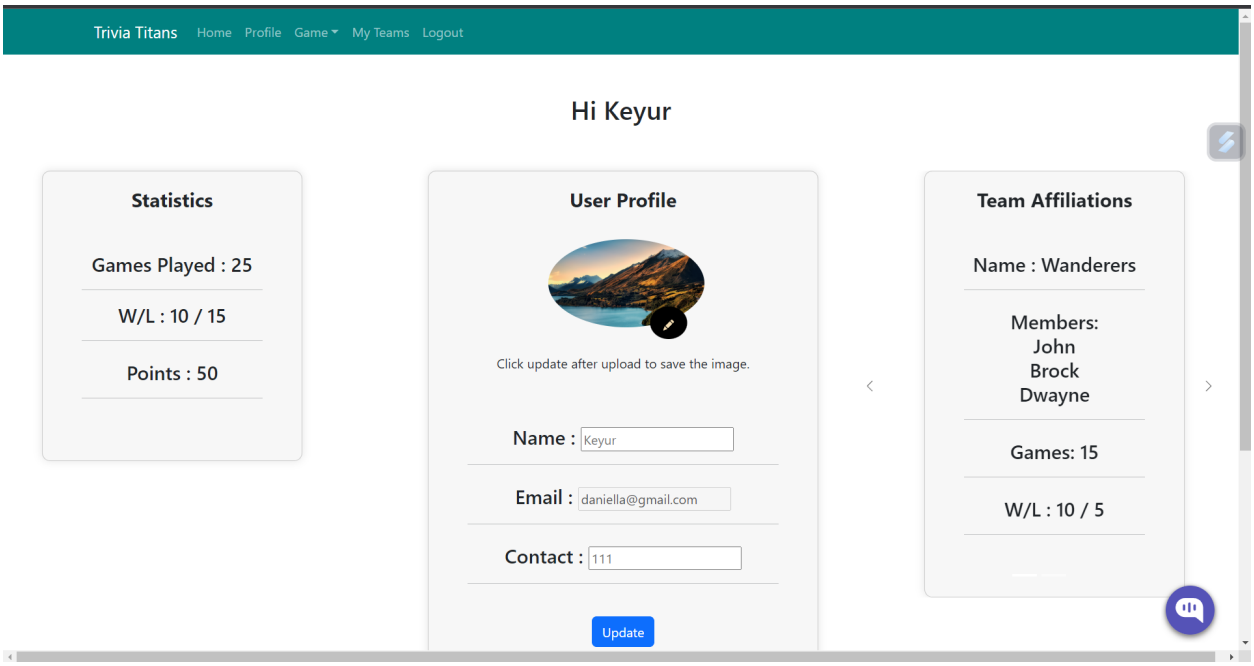


Figure 10: User Profile page

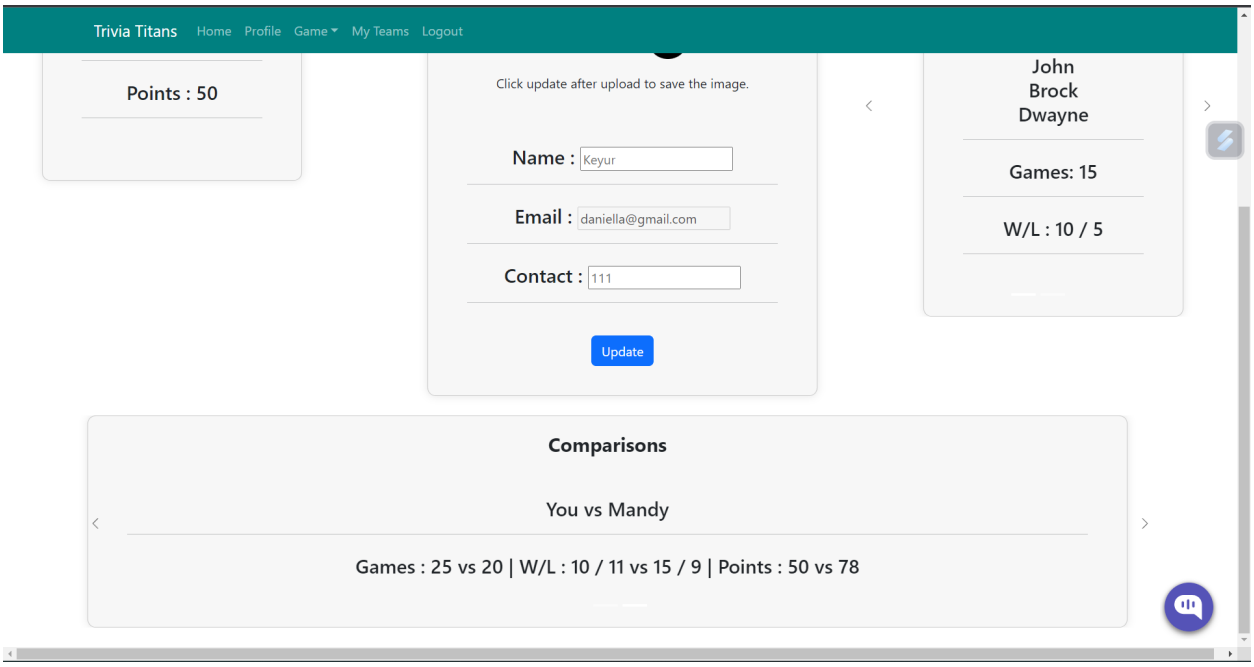


Figure 11: Comparison details in user profile details

2.3 Team Management:

The Team Management module consists of generating a unique team name with OpenAI API [5], Creating a team and inviting players to join them through pub/sub, tracking the score of all the teams the logged-in user is in, and some features like removing players from a team or leaving the team. The user can go to the Team Stats Page by clicking on the “My Teams” button from the navbar. This will redirect the user to the team stats page where the user can select the teams he/she is in and the team details such as Win/Loss ratio, total games played, all the players in the teams, etc will be displayed. In addition to this, the users can remove the players or leave the team from this page.

To create a new team or invite others, the user can click on the ‘Click here to create a new team’ button at the bottom of the page. This will redirect the user to the Manage Teams Page where the user has 2 options:
1) Create a new team and 2) Invite a player to join the existing teams.

To create a team, a user can either write their own team name or generate a team name with the openAI API.

For invitation, a user can write the email id of the players they want to invite and select the team from the dropdown list. On clicking the submit button, an email for an SNS notification subscription will be sent. On confirming the subscription, the user can send the invite again which will send a link to the player.[2,3,4] The link will open a form wherein the player can write their email ID and either accept or reject the invitation. On accepting the invite, the player will be added to the team.

Used Cloud Services:

AWS – DynamoDB + Lambda Functions + SQS + SNS + ChatGPT (Open AI)

Justification:

There are two reasons for choosing this option of cloud services. I had decided to approach one module with AWS stack and the second module with GCP stack. The second reason was, as I was doing module 3 first and had little experience using AWS, I proceeded to do this with AWS. Furthermore, the documentation was easier to understand, and using the AWS stack was much easier as compared to GCP. The second module I had done was module 9 which I did using GCP.

Implementation:

On clicking the “Generate Team Name” button, a prompt: "Generate a unique team name of 15 characters or less for a quiz game like Kahoot, last 4 characters are numbers. Space is allowed. The first 11 characters should be meaningful" is sent via OpenAI API. This returns a name such as Brainiacs5214, Quizwar2351, etc.

On clicking submit button, the lambda is triggered via the API gateway which creates the following things:

1: Team in 2 Database tables: Team stats (ratio, games played, etc) and Team Details (users in the team)

2: SNS Topic of the respective Team.

3: Team name added in the User table.

On clicking the invite team after entering the email ID of a player, the corresponding lambda is triggered which will add the email ID to the Topic subscription list on confirmation. After this, on sending the email invite again, a form link is sent (the form is stored in the S3 bucket) which will have the team name written in it. On submitting the form, it will close after 3 seconds. The user response is sent to a lambda which will process the email id and invite response. If the response is “accept”, it will publish the data to an SNS Topic which will send it to an SQS queue. When a message arrives in the queue, another lambda will be triggered which will do the following things:
1. Add the player to the Team Details table and Add the team name to User Table.
2. Remove the message from the queue and unsubscribe the user from the team Topic.

For team stats:
When a user goes to the team stats page, lambda APIs will be triggered which will fetch the user email from the localStorage and all the teams in which the user is present will be fetched by a lambda using the API Gateway.

To remove a player from the team:
When a user selects a team, all the corresponding team members will be shown below. Besides them, there will be a button to remove the players from the team. On clicking the button, the users will be removed from the Team Details table and the team name will be removed from the Users table for the corresponding user.

Test Case:

Table 2: Team Management Test Case

Test	Description	Figures
User creates a new team with the AI-generated name	Players should be able to use an AI-generated name for creating a team	Figure 16
User creates a new team with a name written by them.	Players should be able to create a team with a name created by them	Figure 38
The team already exists in the database.	If the user enters a team name that already exists in the database.	Figure 37

The team invite isn't sent to all the players in the game.	Only the players whose emails are written in the invite players field should get the team invitation link.	Figure 39
The team members are shown in their respective teams.	If there are n number of players in a team, all the players' emails should be displayed	Figures 31, and 32
The team members are removed from the team when a player clicks on the remove player button	A player can remove other team members if they click on the remove player button next to a player's email	Figure 32,33
The user can leave a team	A player should be able to leave the team if they click on Leave Team button	Figure 36

What was planned vs what was built:

Initially, the plan was to use SNS to fan out the email with custom/personalized fields for every user such as team name in the email id and a unique URL link to join the team. Unfortunately, SNS fan out doesn't let users have different fields for different endpoints. Every endpoint gets the same content in the email. I even tried generating a unique number using UUID but even then the different users had the same message content.

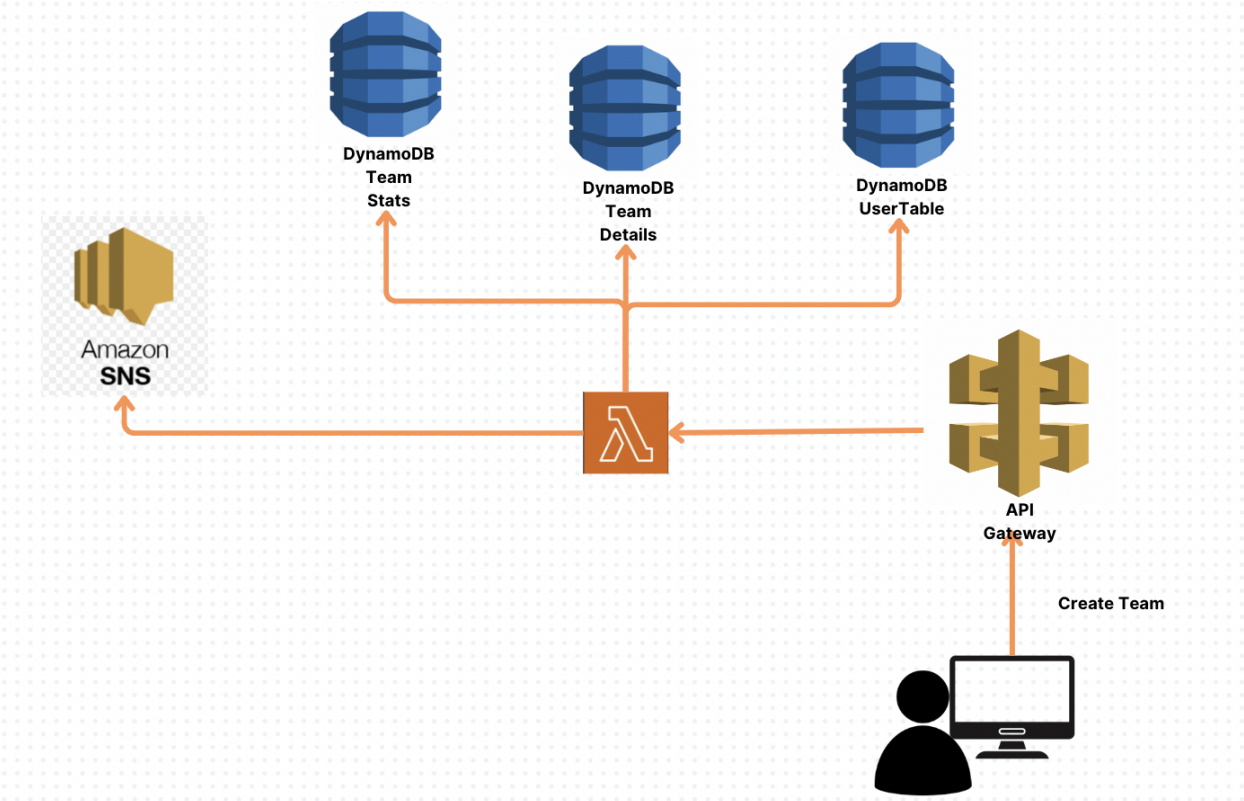


Figure 12: Service architecture for team creation

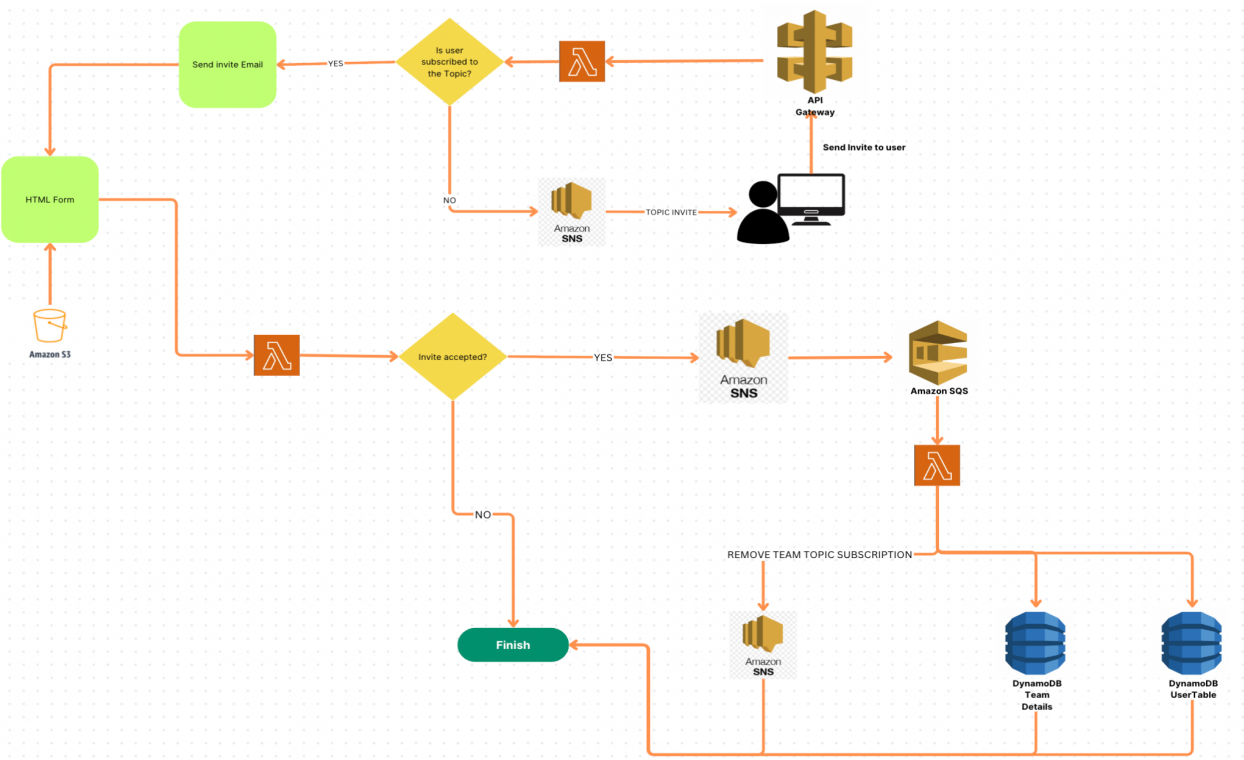


Figure 13: Service architecture for team invitation

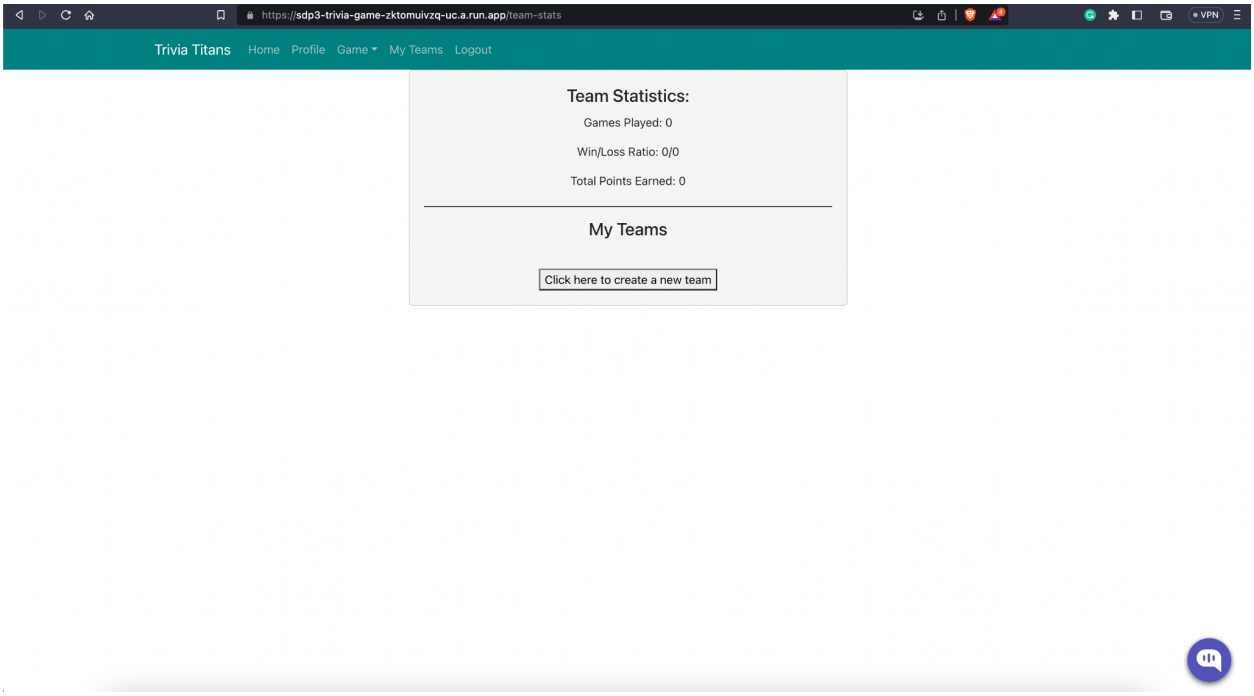


Figure 14: Team details home page

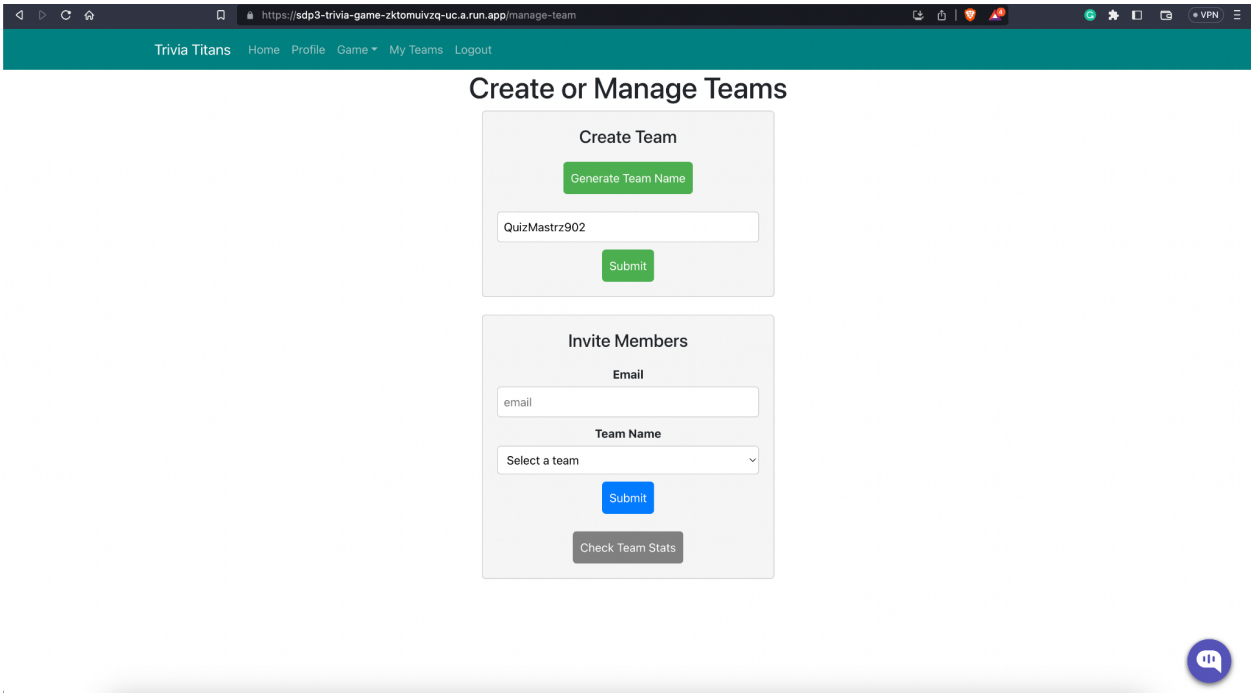


Figure 15: Creating new team

Generate team name using OpenAI API:

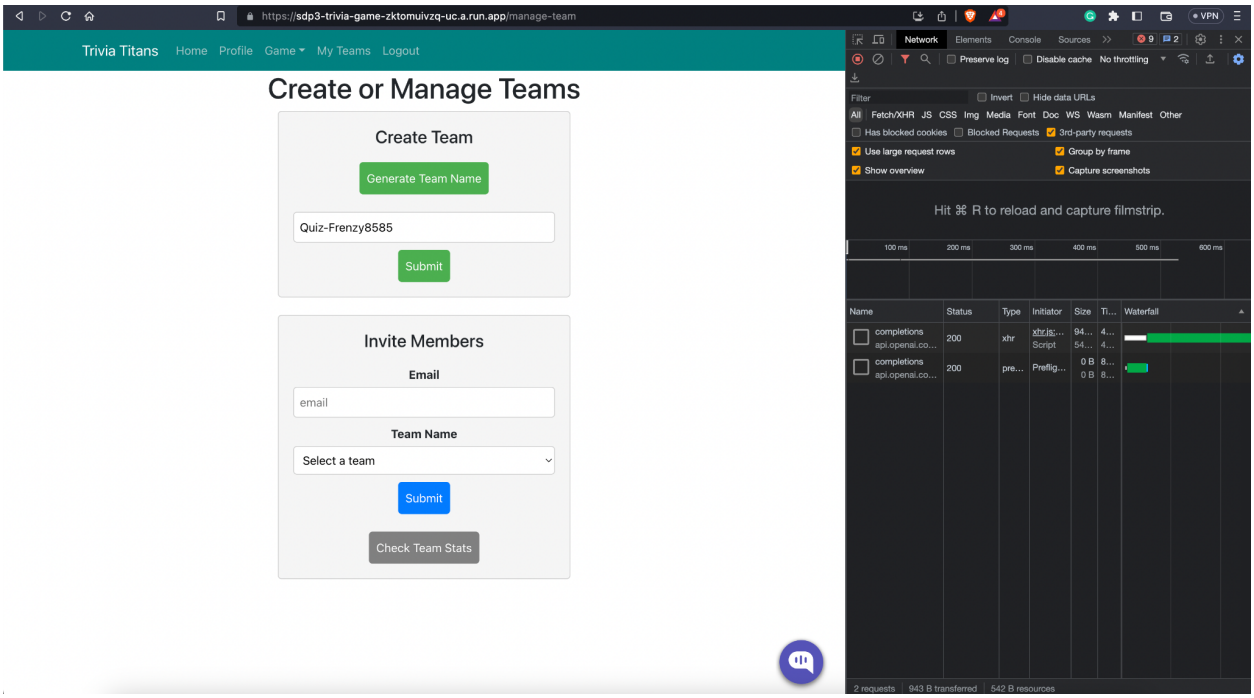


Figure 16: Generating team name using OpenAI APi

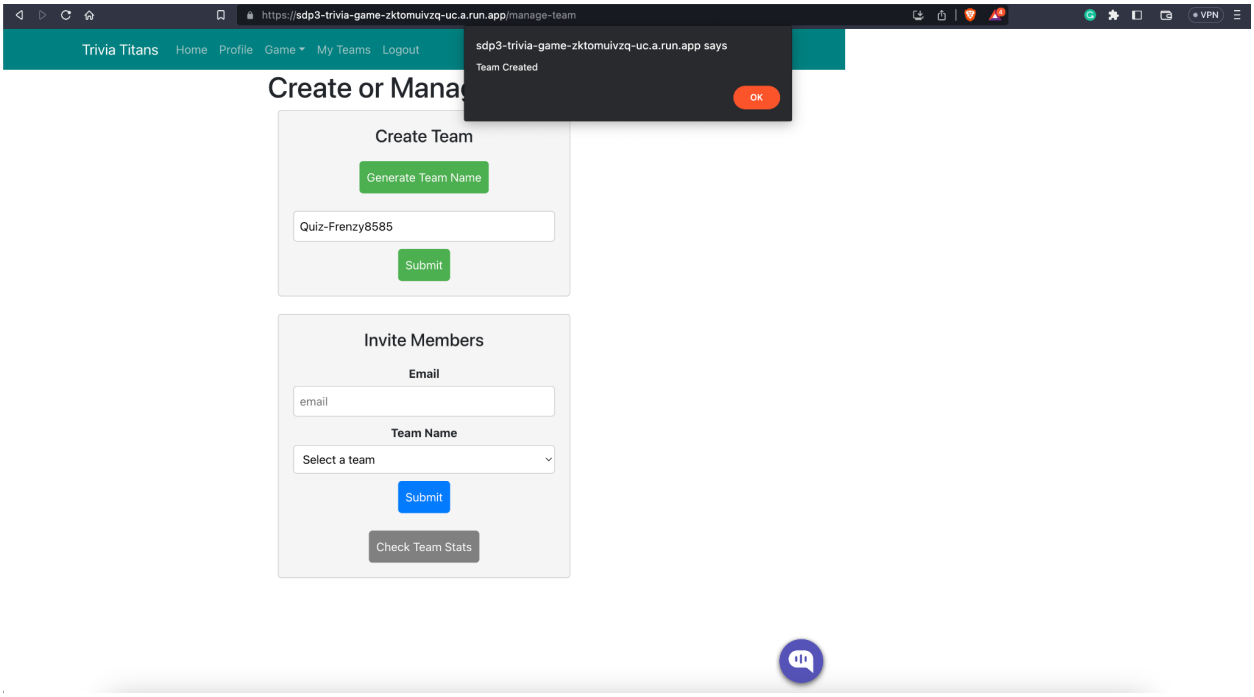


Figure 17: Team Creation popup message

All teams that the player is in:

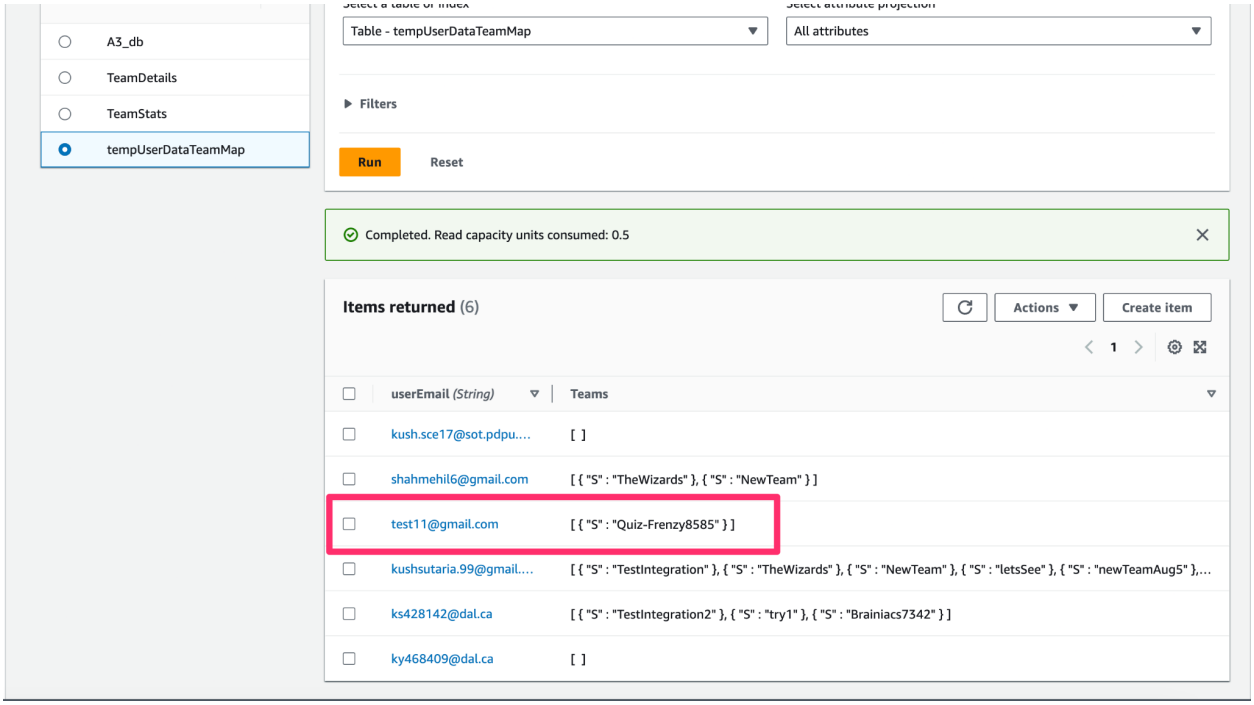


Figure 18: New team creation entry into DynamoDB

Team details (all players in a team):

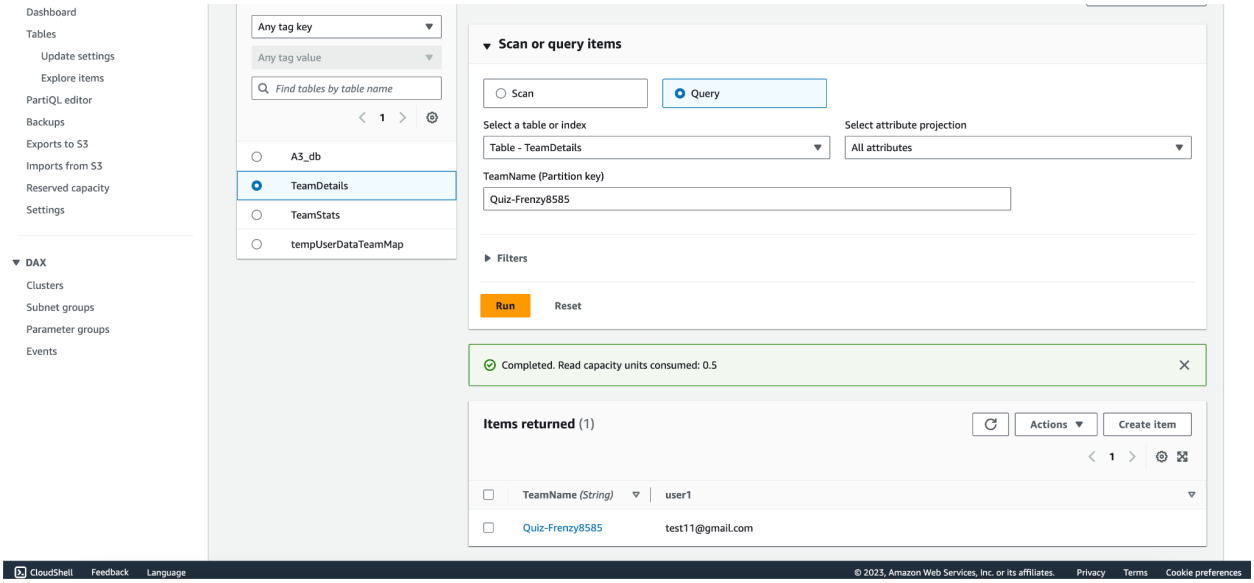


Figure 19: All players details in DynamoDB

Team Stats:

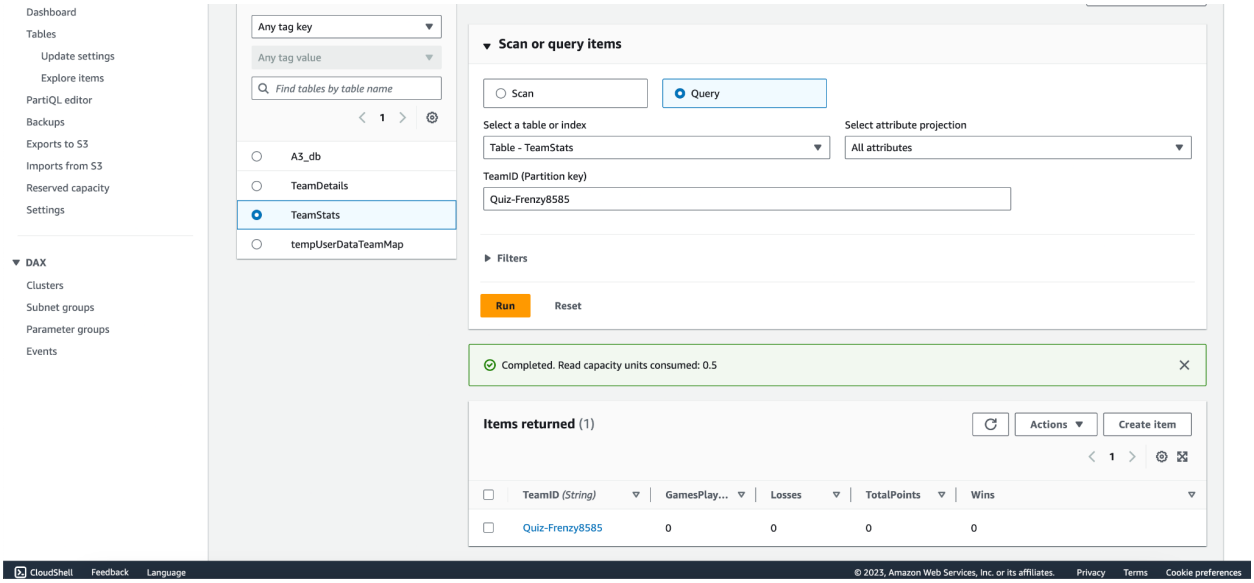


Figure 20: Team statistics in the DynamoDB

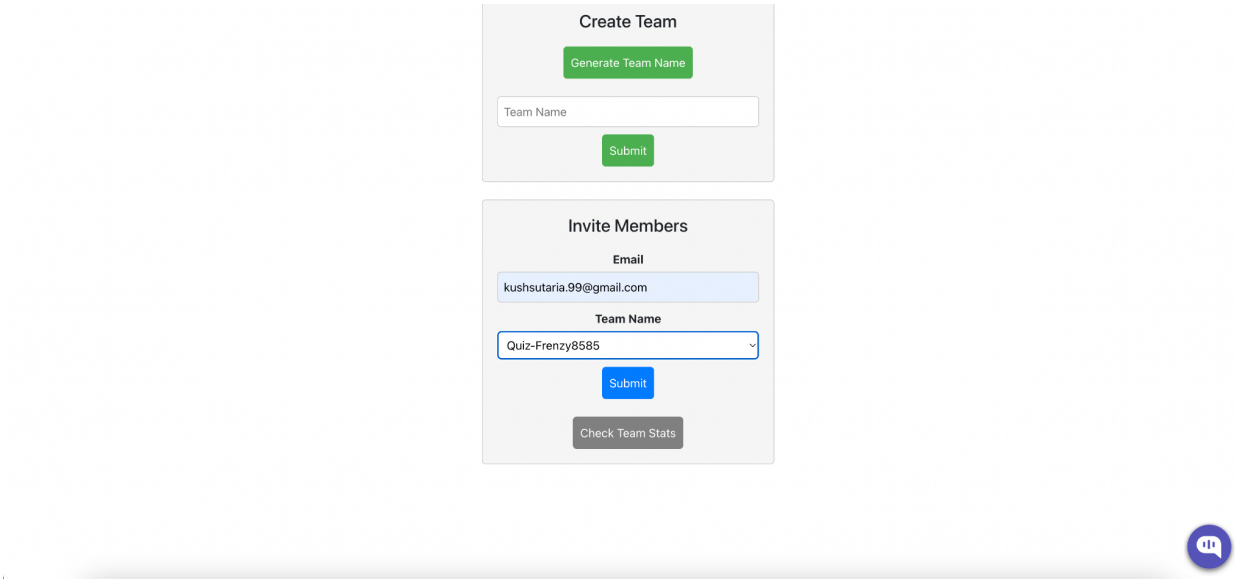


Figure 21: Inviting new team member into existing team

Subscription mail:

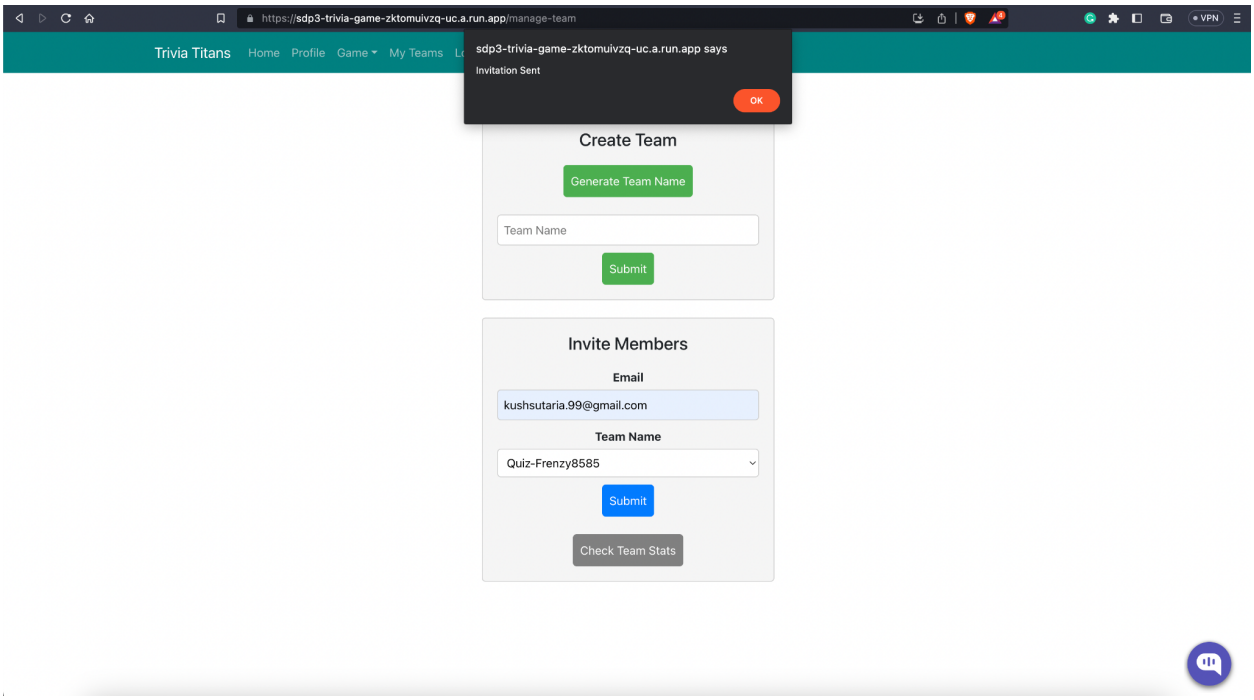


Figure 22: Sending notification of subscription to the invited user

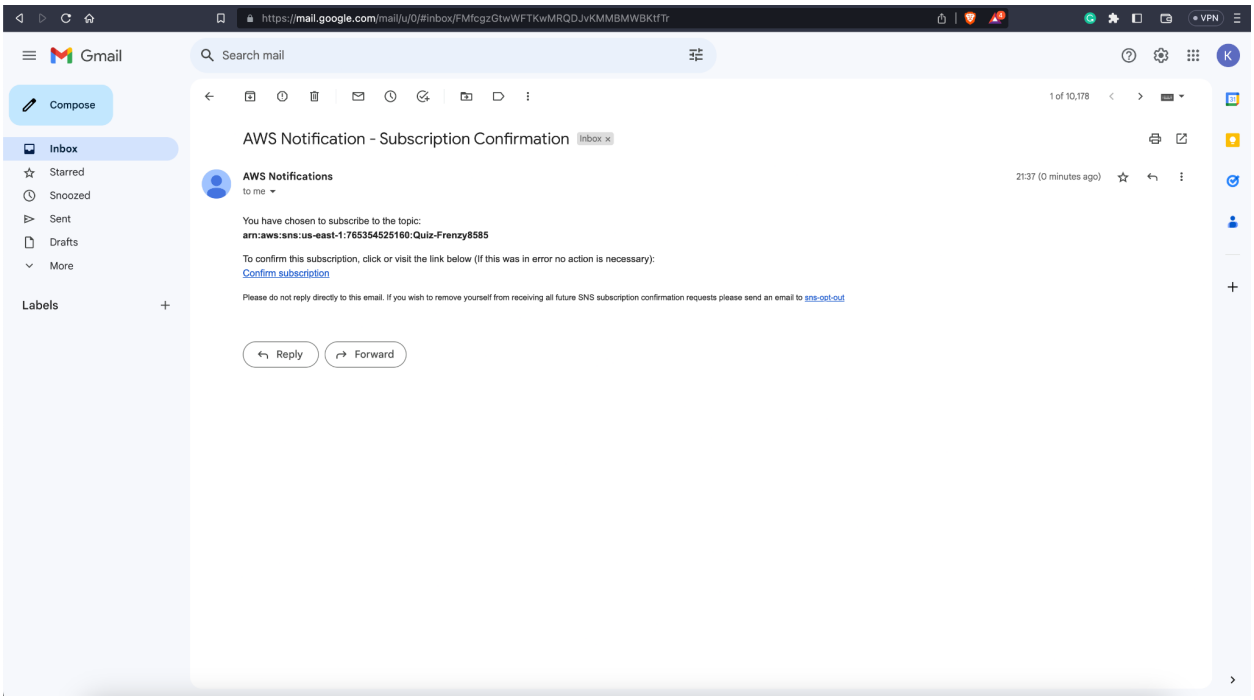


Figure 23: Subscription mail to invited users

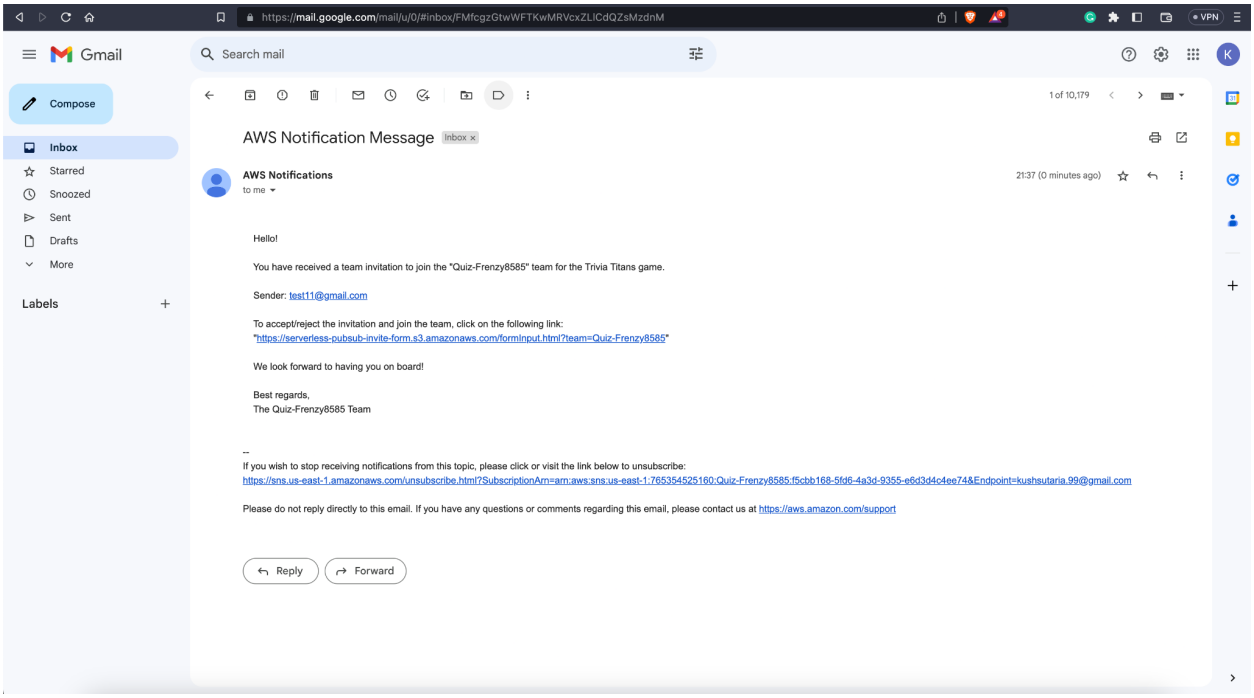


Figure 24: Notification email to invited users

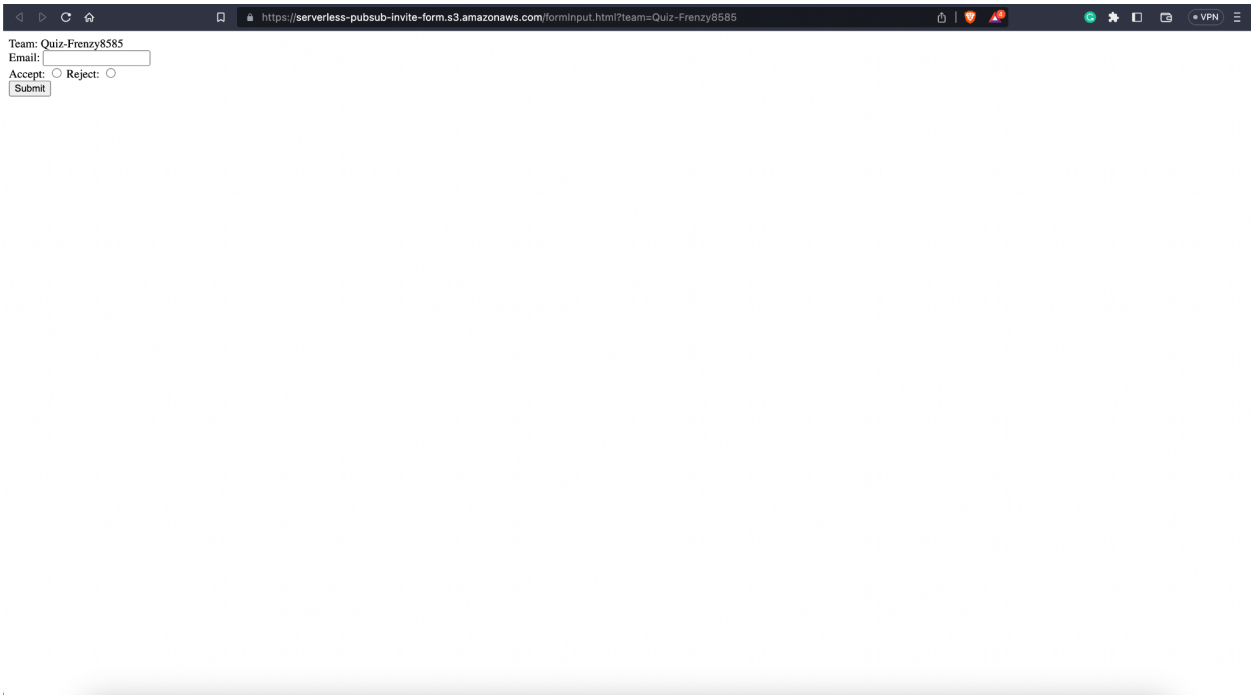


Figure 25: Accepting the invitation

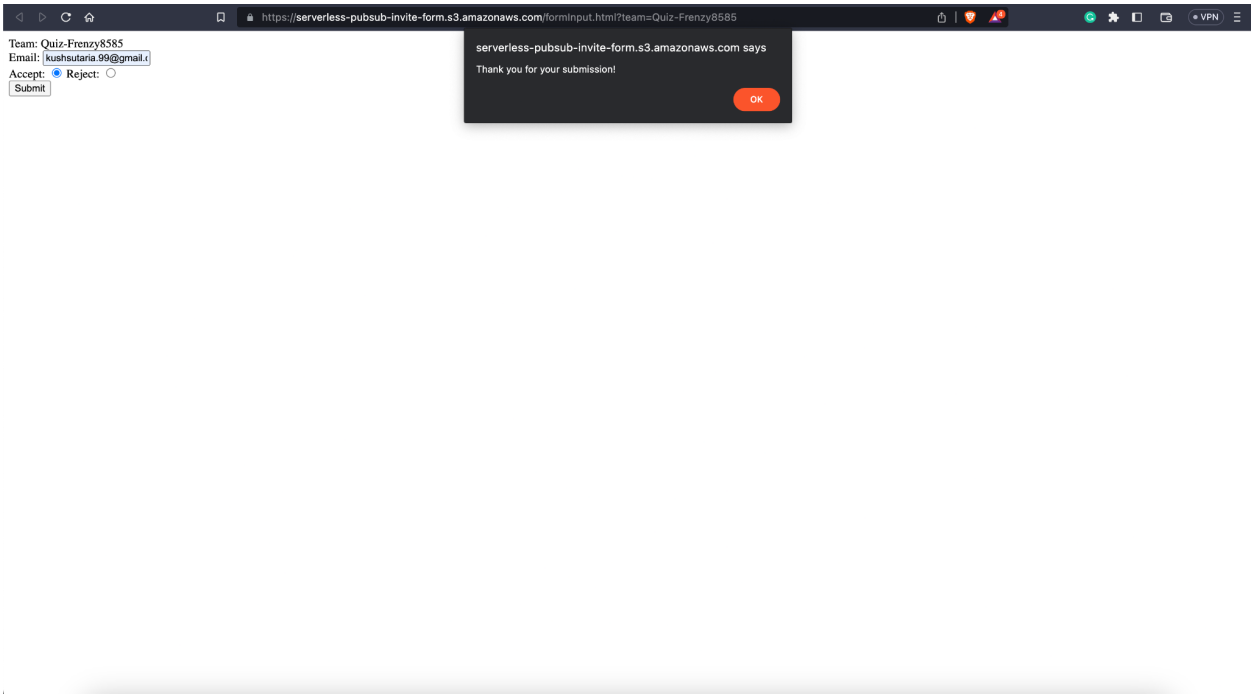


Figure 26: Acceptance pop up message

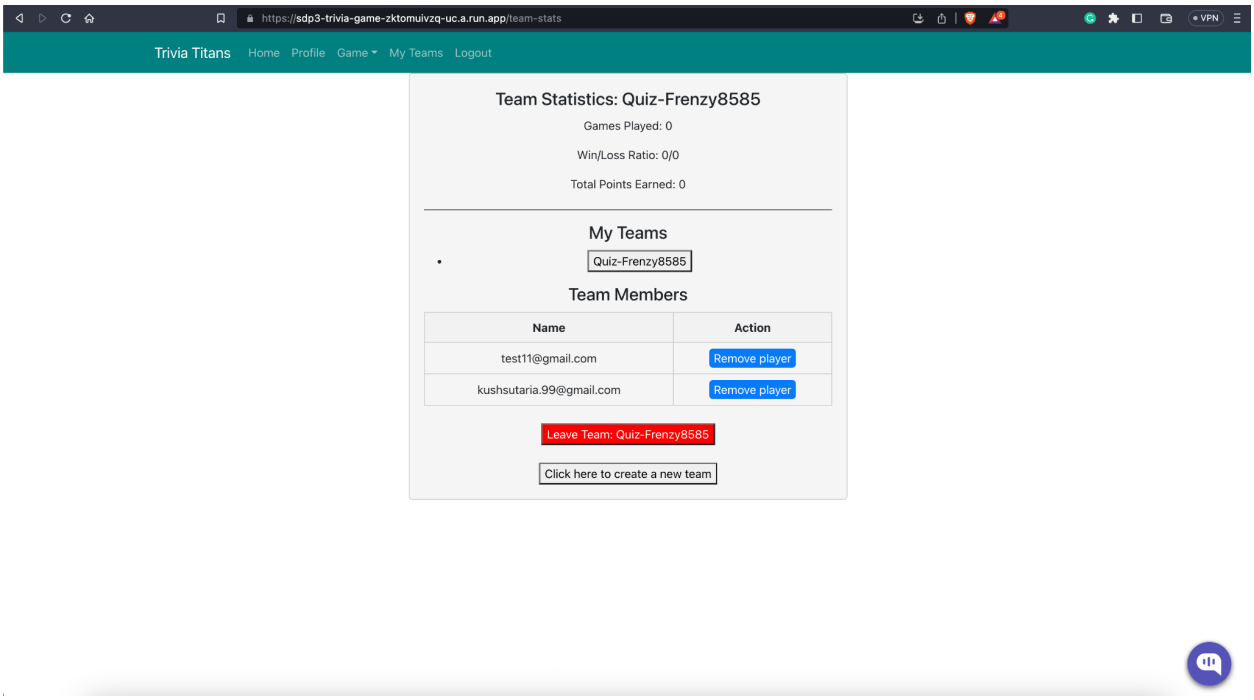


Figure 27: Team statistics in the DynamoDB

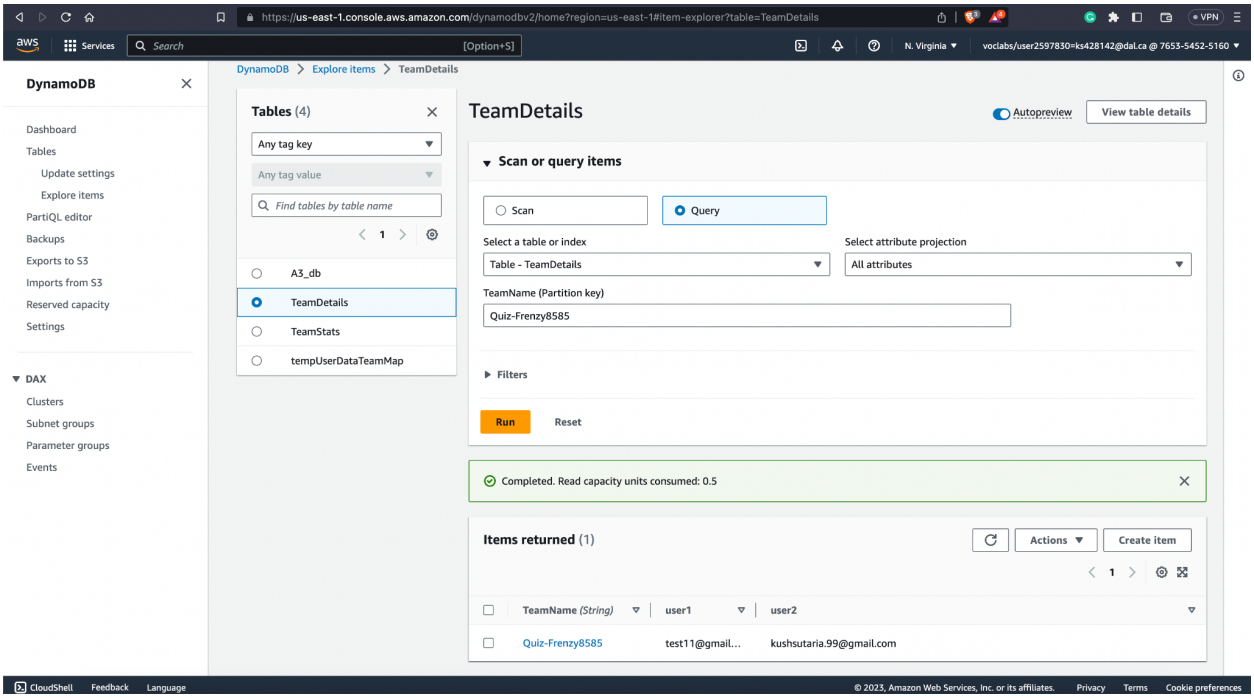


Figure 28: Team Details in DynamoDB

Shown in edit view because all the values weren't visible in normal table view:

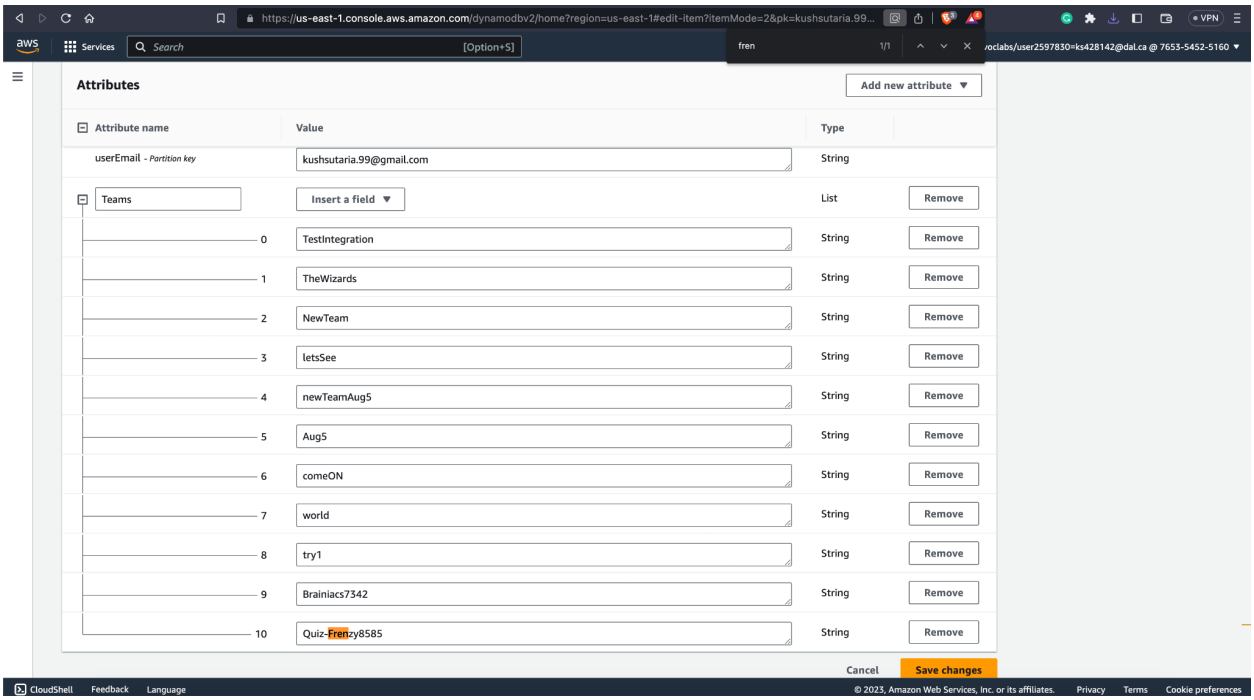


Figure 29: Team statistics in the DynamoDB

Logged in to kushsutaria.99@gmail.com account:

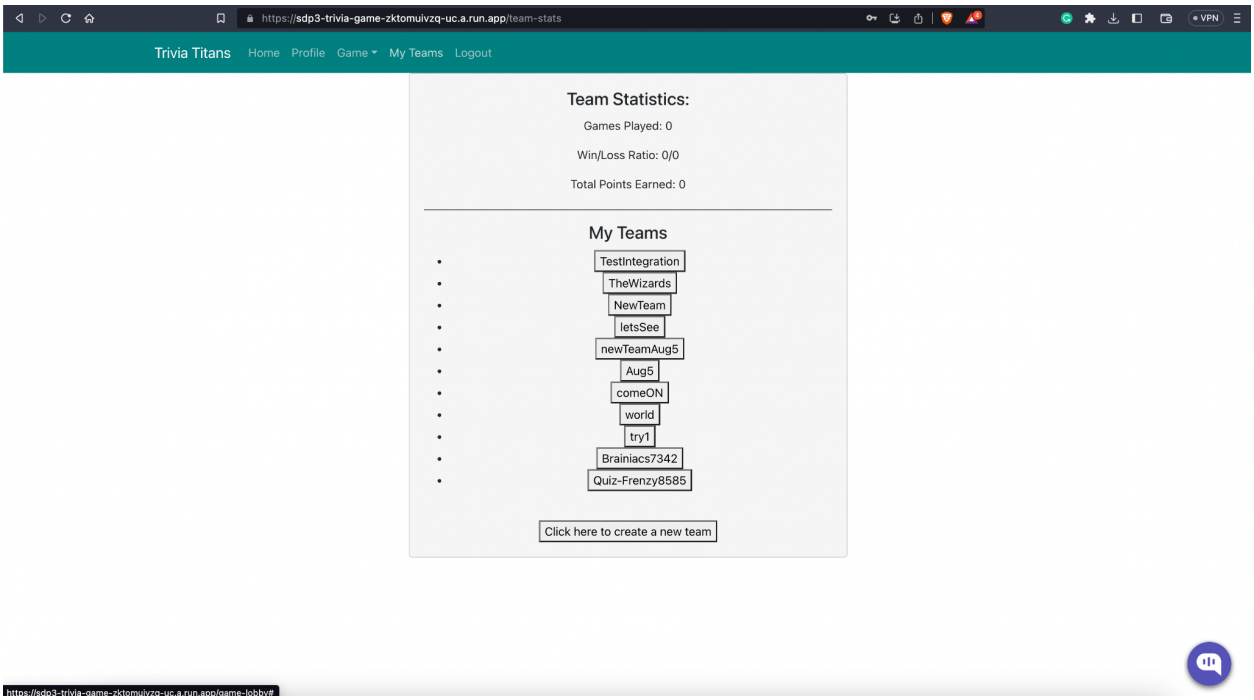


Figure 30: All available teams

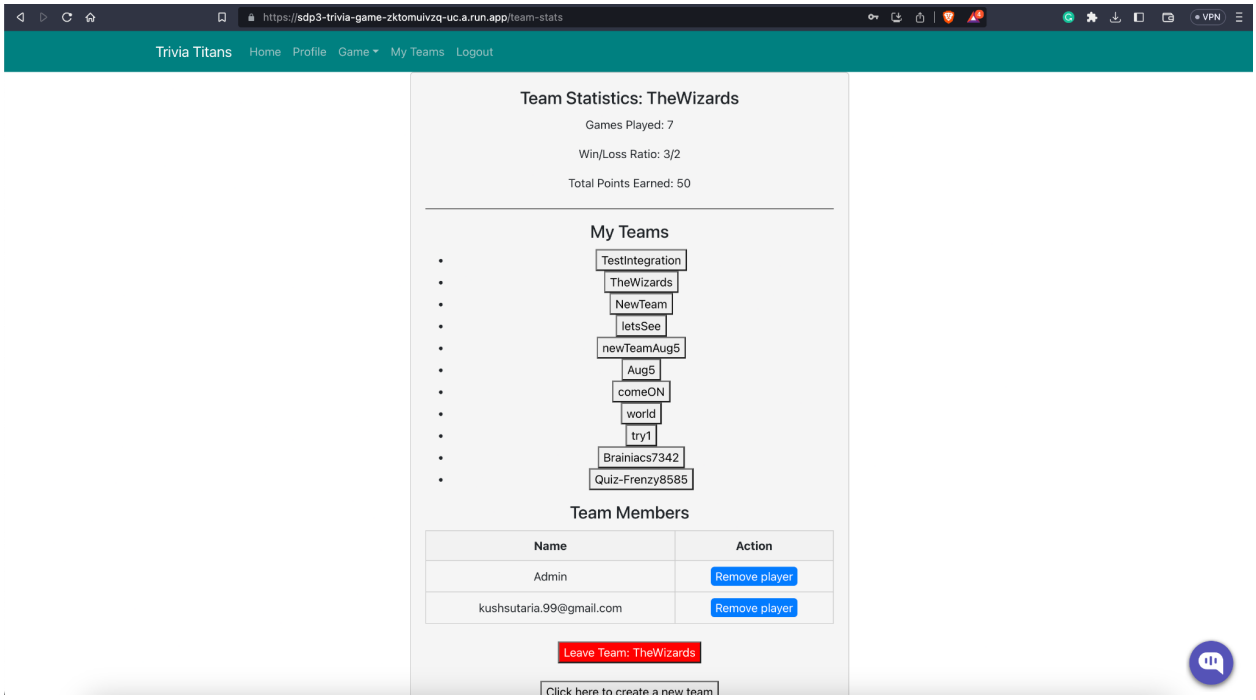


Figure 31: Team members of a team

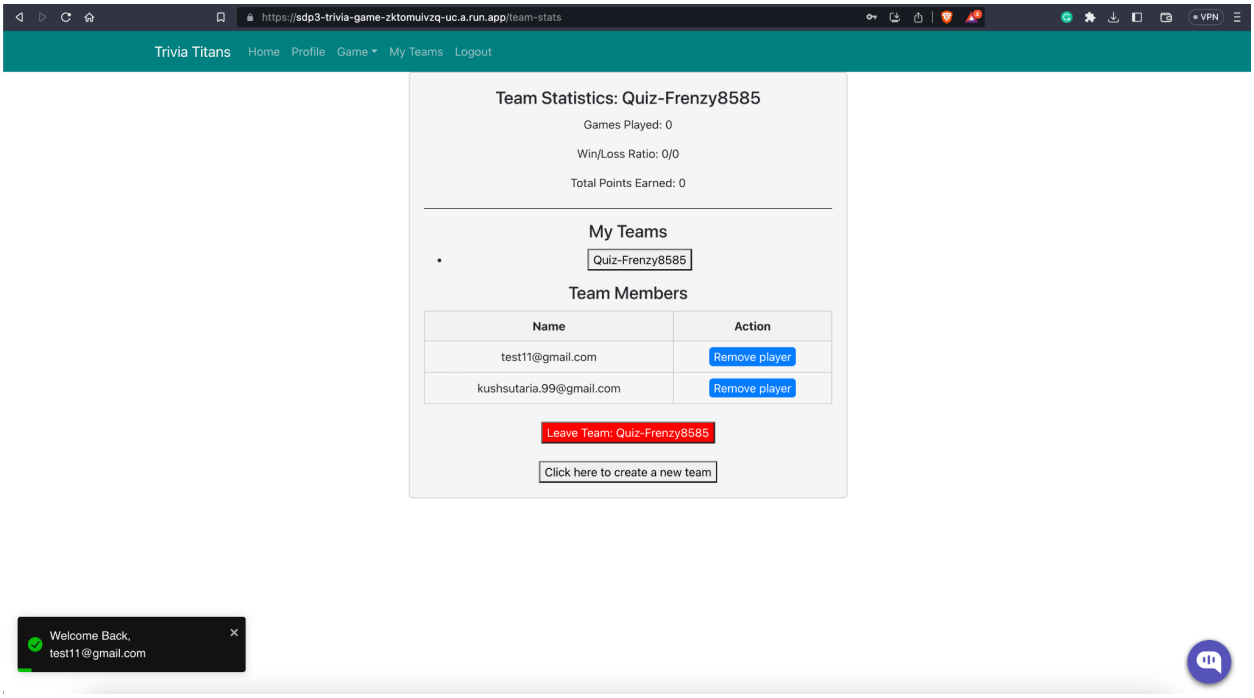


Figure 32: Team operations

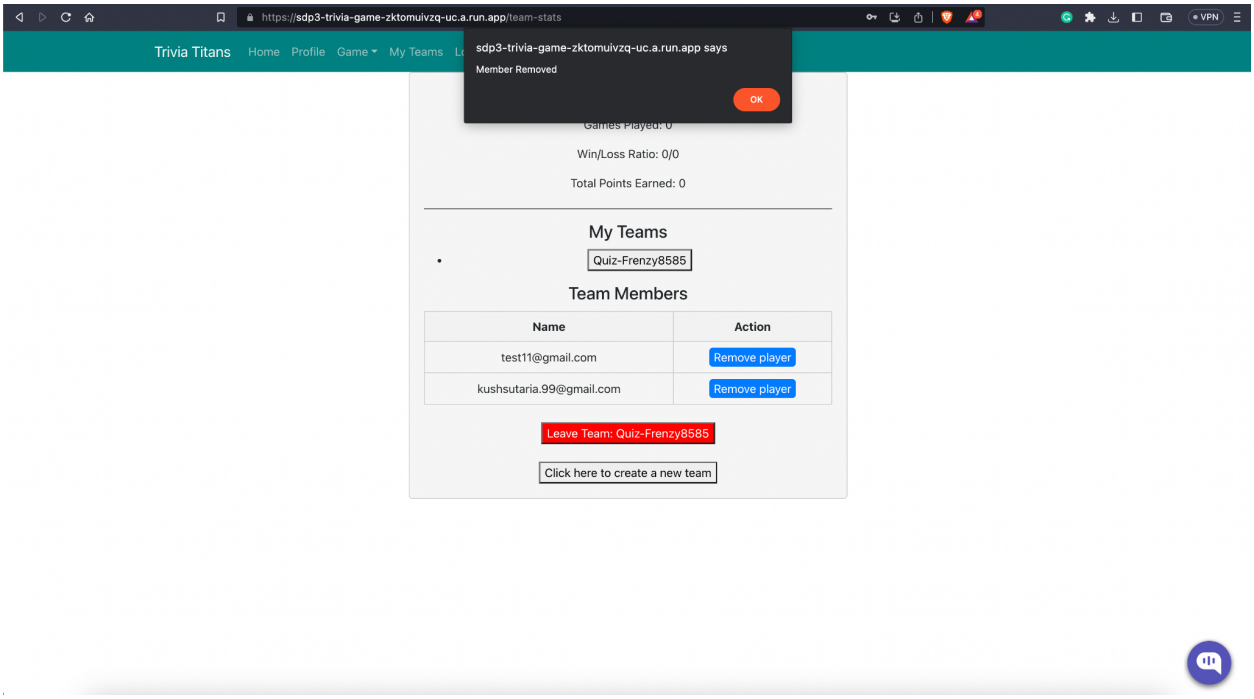


Figure 33: Member removed from team

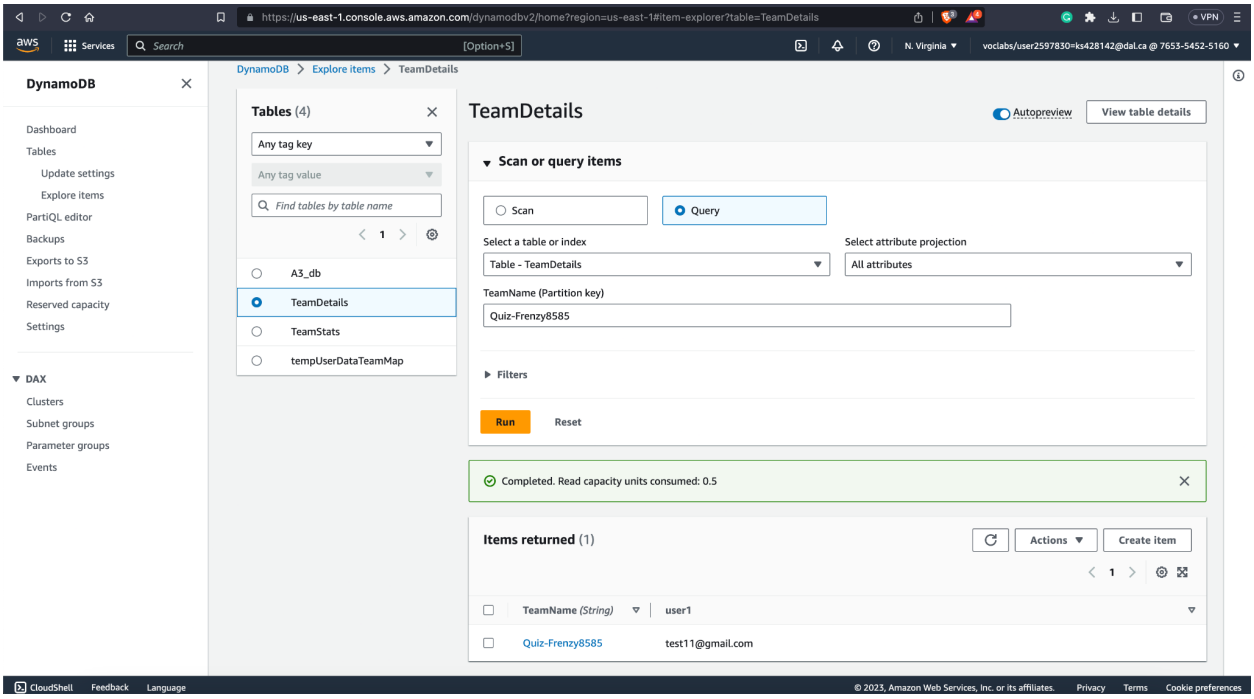


Figure 34: Team statistics in the DynamoDB

Leave team:

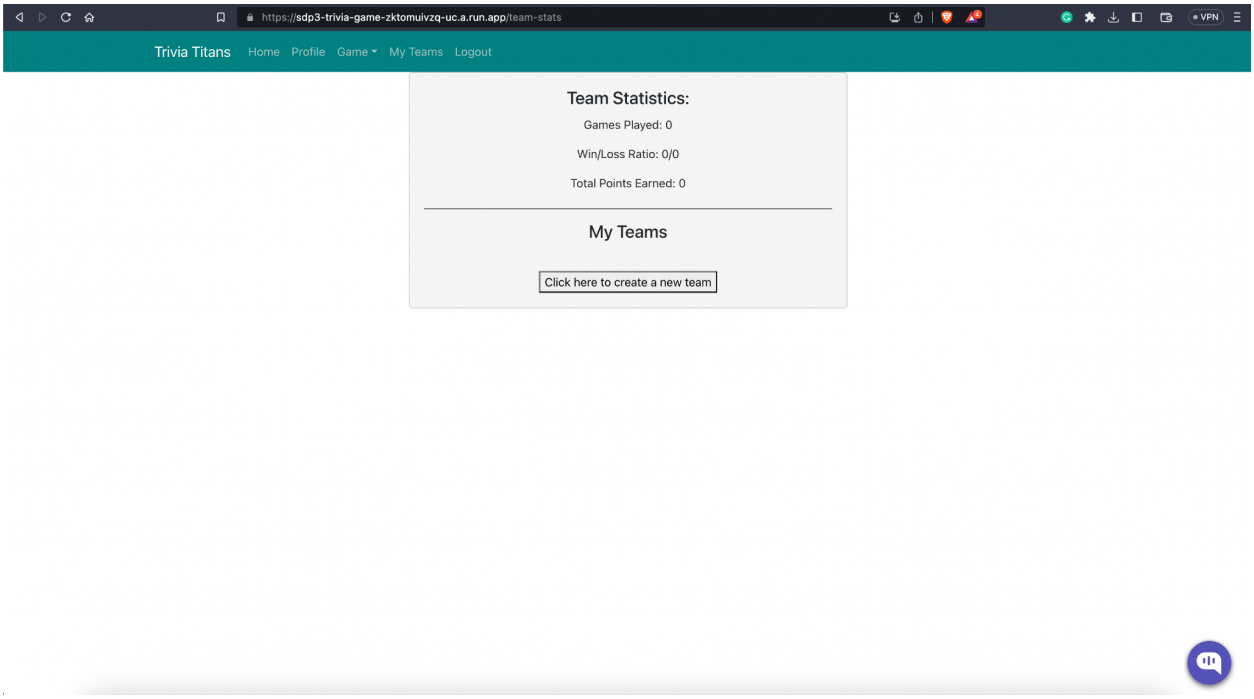


Figure 35: Team Statistics Page

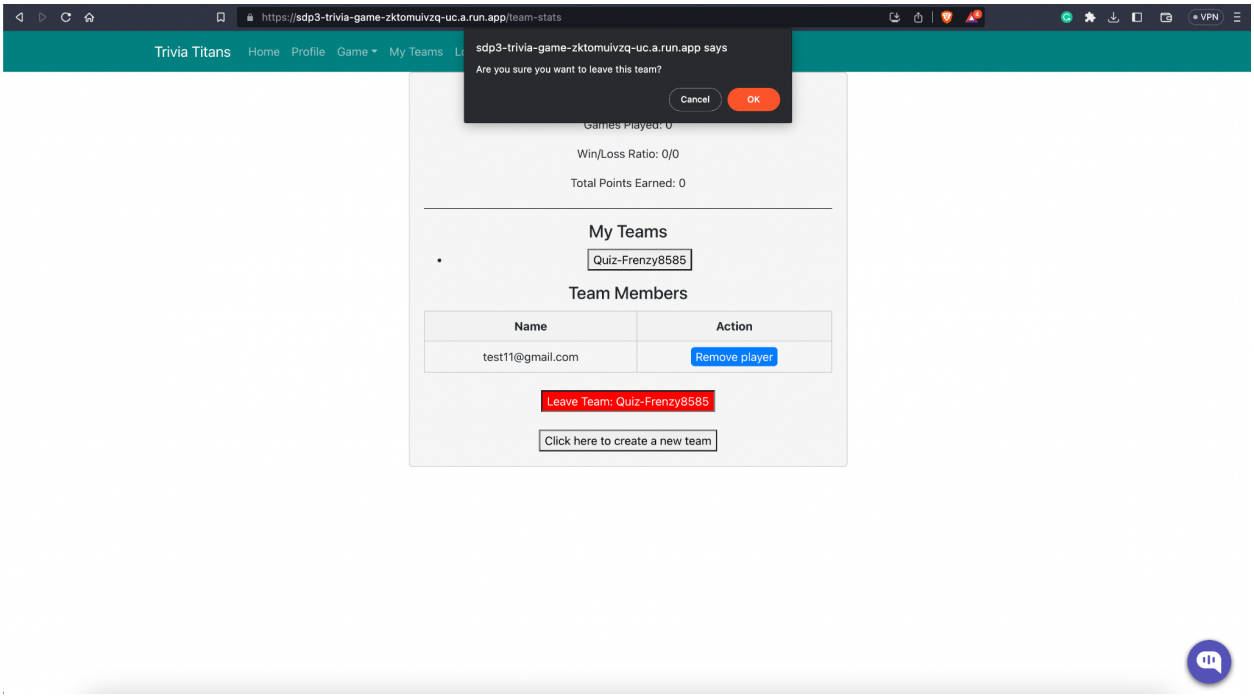


Figure 36: Leaving the team

If a team name already exists:

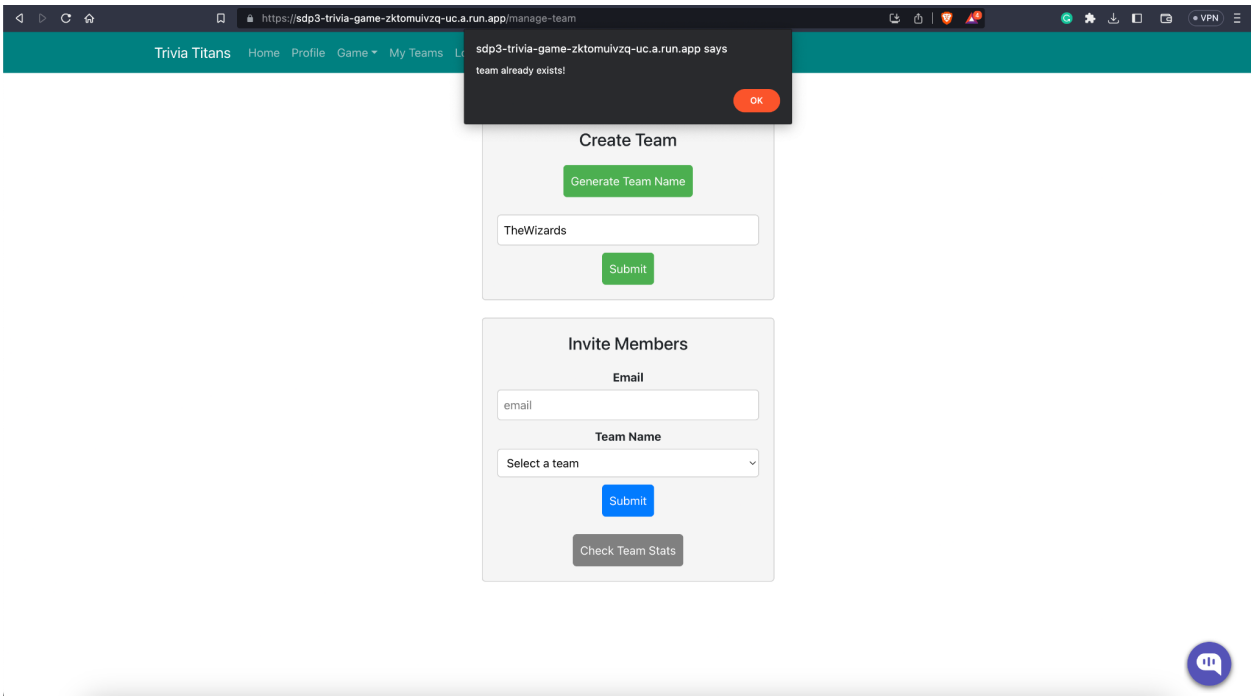


Figure 37: Team existence check

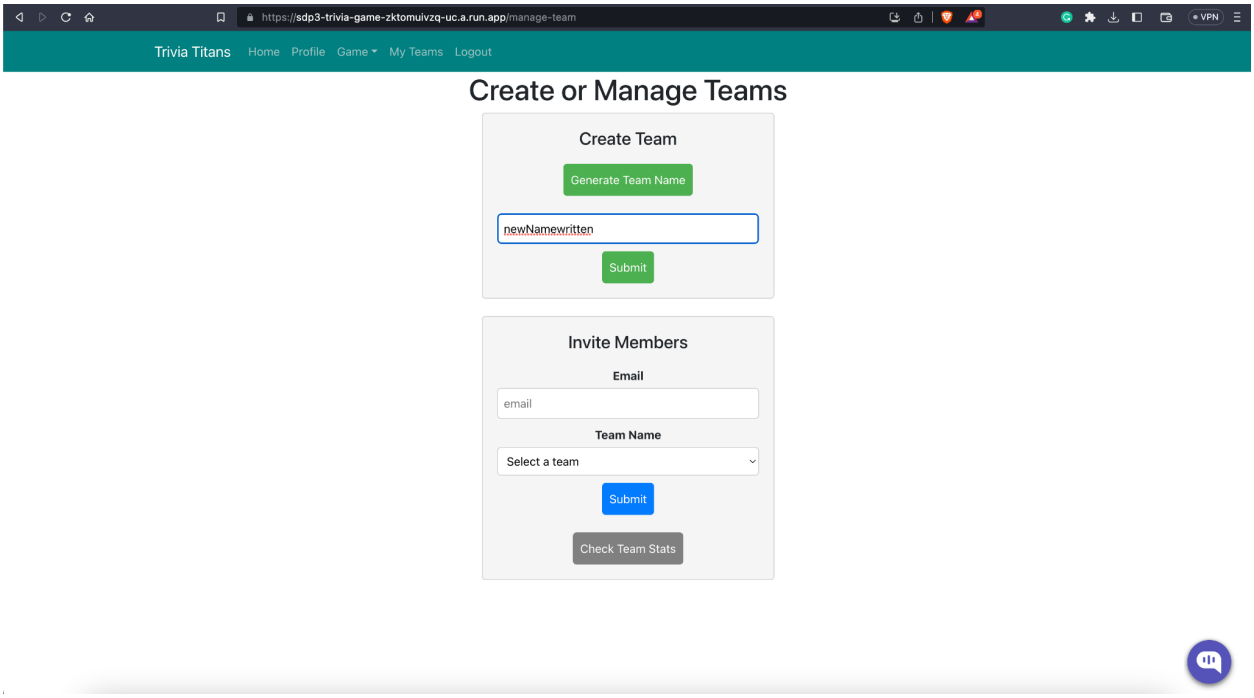


Figure 38: Manually written team name

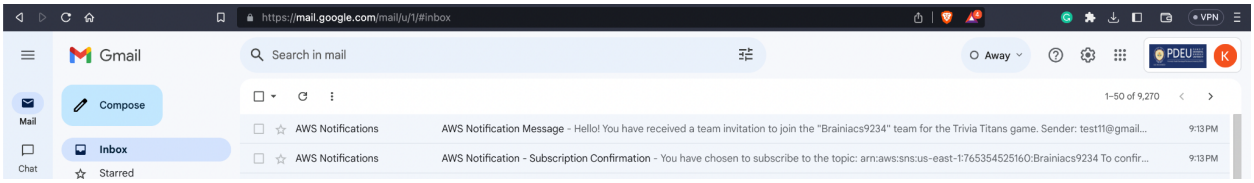


Figure 39: Email for 2nd invite didn't come to this user.

2.4 Trivia Game Lobby:

A key element of the "Trivia Titans" game is the Trivia Game Lobby, which acts as a hub for players to discover and take part in different trivia games. The lobby displays the games that are accessible and have been made by the administrators, allowing players to freely navigate through these options. The game lobby's intuitive filtering system, which enables users to select and join games that best match their interests and skill levels, is a major feature. The games can be filtered by users based on a number of criteria, including genre (for example, science, history, or entertainment), difficulty level (beginning, intermediate, or advanced), and time frame (short duration or longer sessions).

Upon selecting a game, the lobby provides more detailed information about it, enhancing user understanding and decision-making. These specifics include the number of players, the amount of time left before the game begins, and a brief statement detailing the game's purpose and any applicable rules. This thorough information guarantees that users are fully aware of what they are agreeing to, enabling them to align their tastes and expectations before deciding to join a game.

The Trivia Game Lobby will be developed using cloud services suitable for managing real-time data and enabling smooth, scalable game operations. Depending on the chosen technology stack, in this project, a trivia game lobby is developed using Google Cloud Platform services, such as Cloud Functions.

Proposed Cloud Services:

AWS – DynamoDB || Lambda Functions || SQS || SNS || GCP – Firestore || Cloud Functions || Pub/Sub

Used Cloud Services:

GCP Cloud Functions, AWS DynamoDB

Implementation:

Implementing the Trivia Game Lobby using GCP Cloud Functions and AWS DynamoDB would require several steps. Here are detailed implementation steps in Python:

Step 1: Setup Google Cloud Functions & AWS DynamoDB:

Set up Google Cloud Functions in the GCP Console. Ensure that the Google Cloud SDK is installed and authenticated with your Google Cloud account. Handling DynamoDB access tokens in cloud functions.

Step 2: Implement the Filter Function:

Create a new Python file for your Google Cloud Function. Import the necessary modules, including the Boto3 module for interacting with DynamoDB.

Write a function to connect to your DynamoDB table. This will require the AWS access key ID, secret access key, and the region where your DynamoDB table is located. This filter function will get query parameters from the URLs and filter according to its data. If there aren't any filter parameters, it will provide all game data.

Service Architecture:

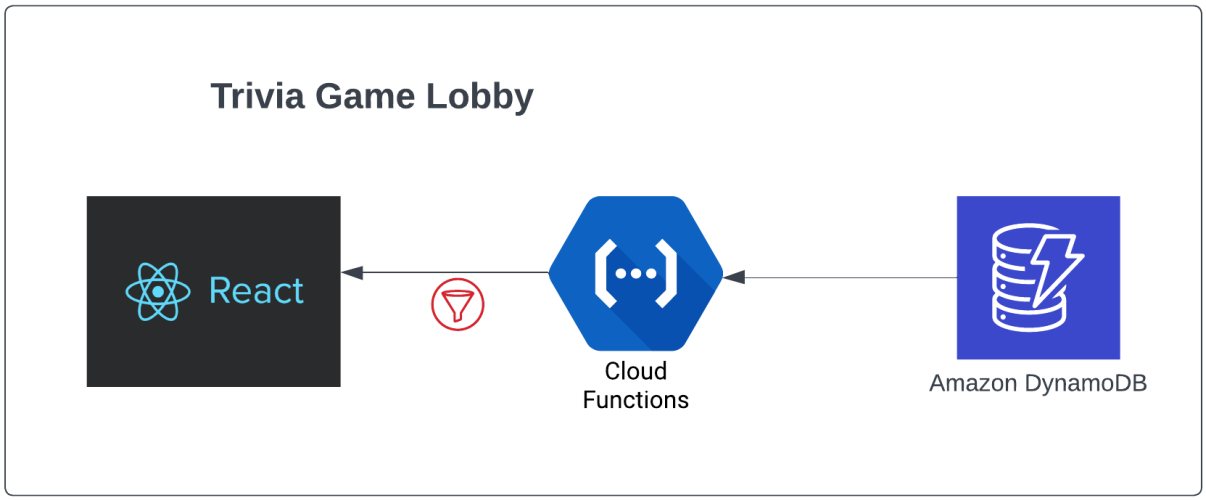


Figure 40: Trivia game lobby architecture

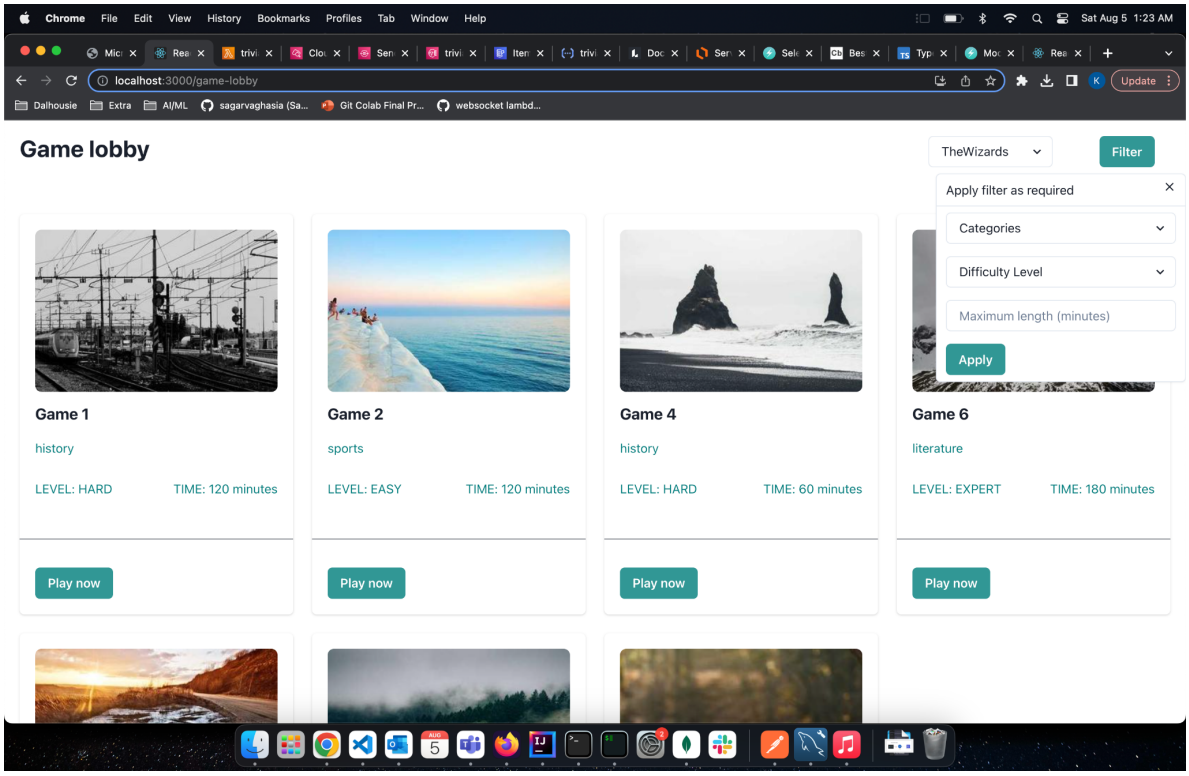


Figure 41: Game Lobby page with team selection and filtration functionality

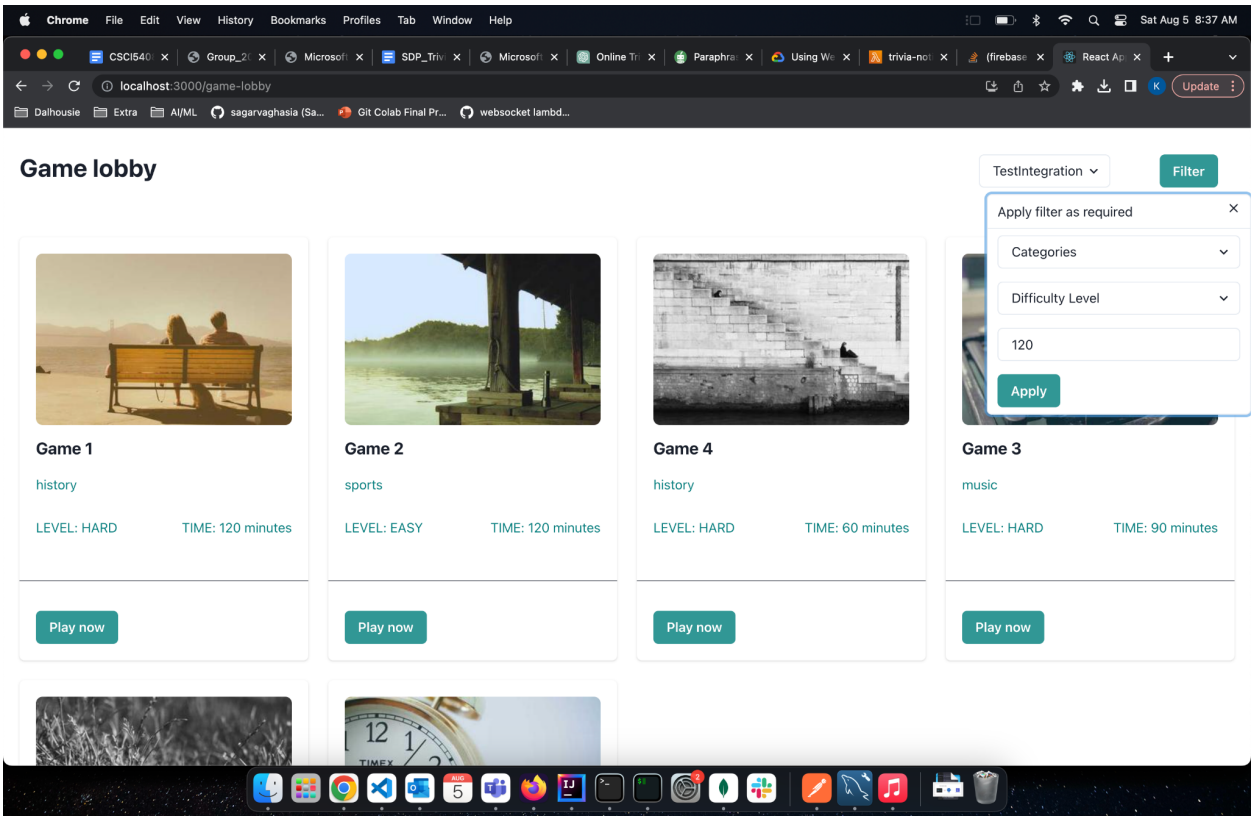


Figure 42: Game Lobby page maximum time frame filter

Figure 43: API for handling game data fetch with filter options

This endpoint URL represents an API that provides access to the trivia game data stored on a serverless backend, specifically targeting the **triviaGameLobby** function. The function is designed to fetch trivia game data, filtered by specific parameters such as category, timeframe, and difficulty level.

The base URL: **https://us-central1-serverless-392821**.cloudfunctions.net corresponds to the Google Cloud Function deployed on the Google Cloud Platform. us-central1 is the region where the cloud function is hosted.

Users have to select a team from the top dropdown. Once a user selects the game, game invitations will be sent to all the team members via email notification. It will contain one unique URL to join into the Game lobby. Users will be waiting for a specified amount of time in the wait lobby.

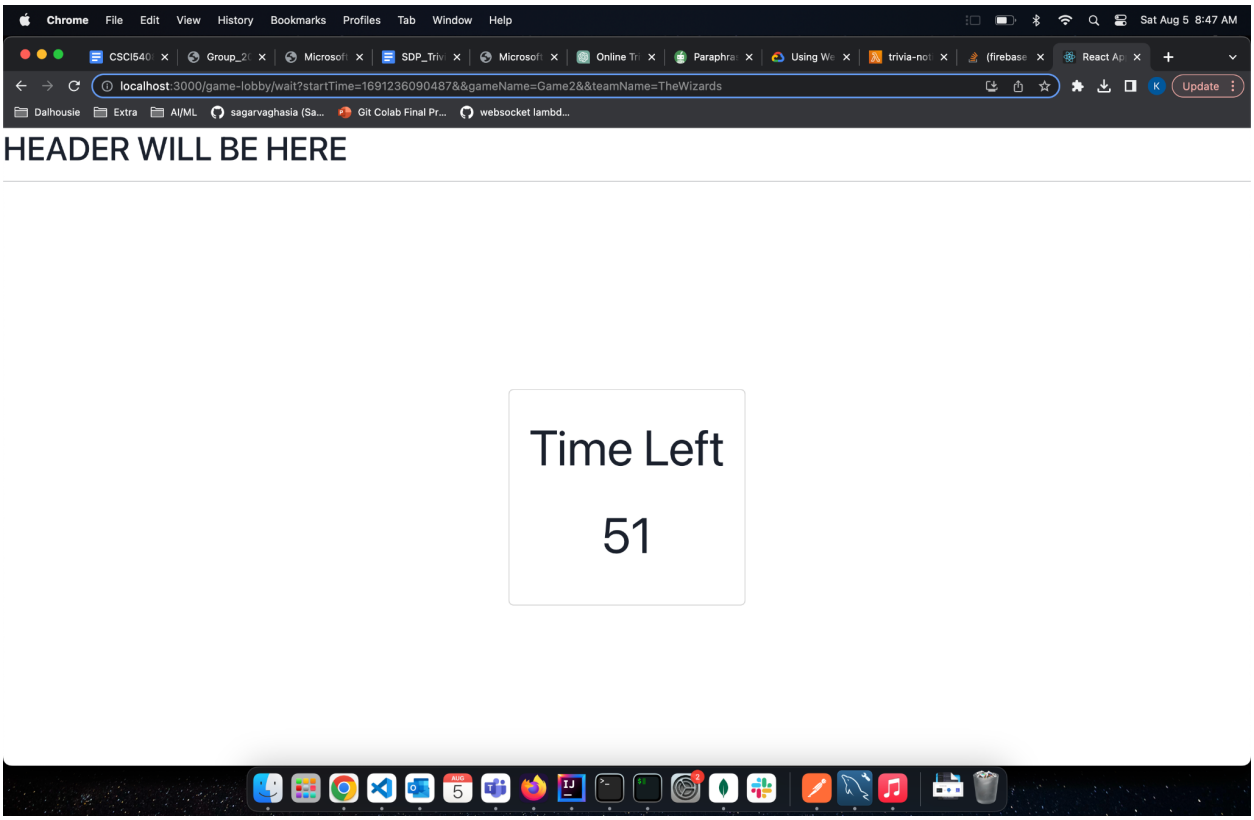


Figure 44: Wait lobby, where user will be waiting for other users to join

Once, wait time is over, all users who joined with the URL will be redirected to the game experience or quiz lobby, where they can play together.

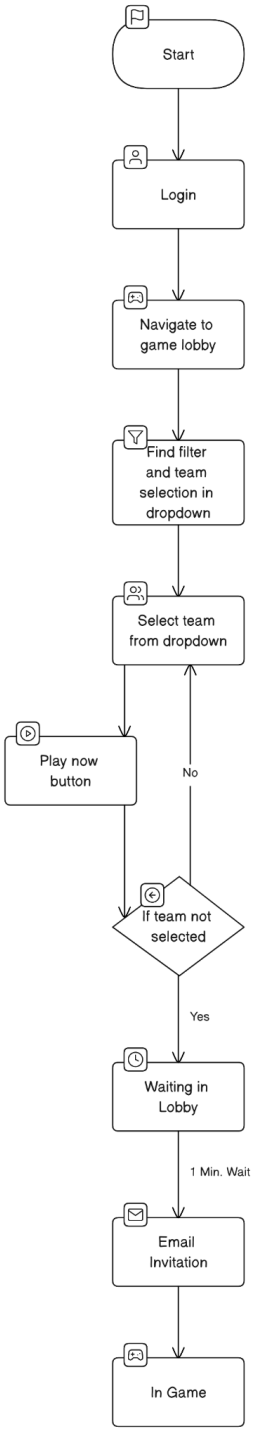


Figure 45: Game Lobby flow diagram

Test Case:

Table 3: Trivia Game Lobby Test Case

Test	Description	Figure
Quiz lobby with all available games.	Users can view all available games from the database.	Figure 43
Filter options with category, timeframe and difficulty level.	User can customize game views as per the various 3 filter options: Category wise filter, Maximum Timeframe, Difficulty wise filter	Figure 42
Team selection	Users can select teams from top dropdown, which indicates users want to play a game with a specific team.	Figure 47
Playing the game	Users will be redirected to the waiting lobby, after clicking the “Play Now” button, which will parallel send the email notification to all team members.	Figure 48

2.5 In-Game Experience:

The scenario presented here depicts a multiplayer trivia game in which participants answer multiple-choice trivia questions within a set time limit. The game includes real-time team scores, a chat function for team communication, the opportunity to request and share hints with team members, and the display of right answers and explanations once each question's time limit has expired. Furthermore, during the game, the system monitors and analyses both individual and team performance.

Used Cloud Services:

GCP – Firestore + AWS Api Gateway + Aws Lambda + AWS SNS + Dynamo DB

Justification:

There are 2 main reasons to prefer Firestore over DynamoDB

1. Real time updates

Real time update in firestore comes out of the box and changes to the data will be immediately pushed to connected clients while DynamoDB supports real-time updates through DynamoDB Streams, integrating it with clients can be a tedious task.

2. Structure Querying Simplicity

Firestore document structure is simple you can store any kind and any structure of data which will be identified by document name while DynamoDB involves creation of partition key to identify the document

3. Simplicity and ease of use: When compared to DynamoDB, Firestore delivers a simpler and more straightforward data model. It employs collections and documents, similar to a hierarchical structure, to help developers understand and deal with data.

Integrated Querying: Firestore supports compound queries, filtering, and sorting, allowing for more expressive and versatile data querying. This makes complex queries easier to execute without the need for additional secondary indexes.

Two reasons to prefer Cloud function over lambda are.

1. Language support

Cloud Functions provides larger language support for even languages like Ruby and .NET while lambda function language support is limited.

2. Billing details

The billing details are more descriptive with the invocation pattern graph and the cost of hosting it in cloud function is less compared to lambda.

Implementation:

The scenario's central aspect is the presenting of multiple-choice trivia questions to the participants. Each question is accompanied by a set of answer alternatives, and players must choose the proper answer within a time limit.

Each question in the trivia game has a time limit for answering. Players must respond before the timer runs out, which adds a sense of urgency and challenge to the game.

Players' points are tallied in real time as they answer questions. These scores add to the overall performance of the squad. Players may track their progress and competitiveness throughout the game thanks to the real-time score display.

The game includes a chat component that allows team members to speak in real time. During the trivia game, players can use instant messaging to strategize, discuss answers, and collaborate.

Participants can either request hints from their teammates or share hints to help others. The hints can be sent using the chat feature, which promotes teamwork and group problem-solving.

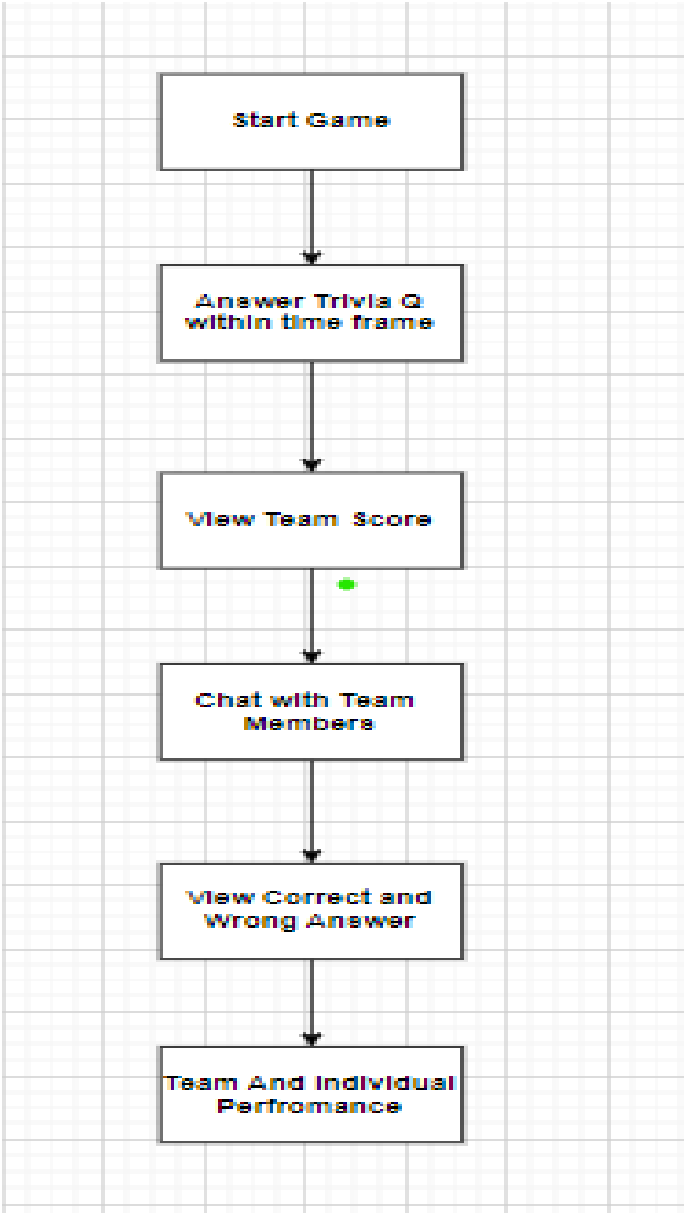


Figure 46: Flow chart

The scenario describes a fun and dynamic multiplayer trivia game that encourages collaboration, communication, and information exchange. The mix of time limitations, real-time scoring, chat functionality, suggestions, and performance monitoring provides participants with an exciting and difficult experience while also developing a sense of camaraderie among team members. Furthermore, the system's capacity to display correct answers and explanations improves the game's instructional value.

2.6 Leaderboards:

There are two primary components of the Leaderboard: overall rankings and category-specific rankings of both individual and team. The overall top-scoring teams and players in each category are shown in descending order The category-specific scoreboard, on the other hand, promotes the top teams and players in particular trivia categories, catering to diverse hobbies and areas of expertise.

Players can quickly access the Leaderboard from the navbar of the landing page, where they can view their current positions of each individual and each team and assess how they stack up against other players. Users can browse ranks for both categories, which creates a dynamic and updated leaderboard experience.

The Leaderboard shows the team name and the matching score for rankings that are based on teams. Additionally, it displays the quantity of games played, win/loss percentage, and total points scored, enabling users to evaluate the team's overall performance and development over time.

Individual player rankings are also displayed on the Leaderboard along with team rankings. Users can view their own individual statistics, such as the number of games they've played, their win-to-loss percentage, and the overall number of points they've accrued. Both leaderboards can be toggled using a dropdown menu on the top.

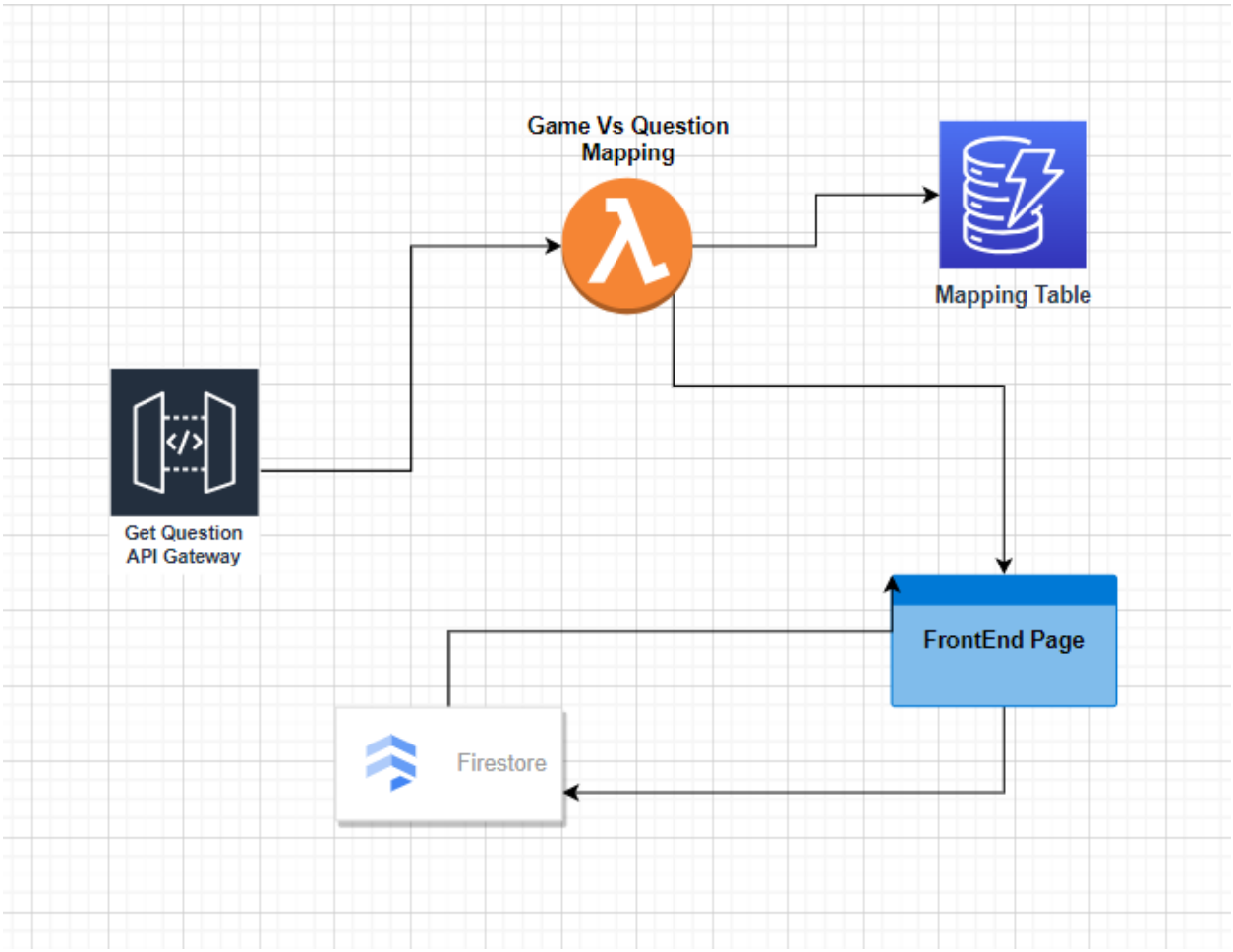


Figure 47: Integration flow chart

The above diagram describes a fun and dynamic multiplayer trivia game that encourages collaboration, communication, and information exchange. The mix of time limitations, real-time scoring, chat functionality, suggestions, and performance monitoring provides participants with an exciting and difficult experience while also developing a sense of camaraderie among team members. Furthermore, the system's capacity to display correct answers and explanations improves the game's instructional value. This flow diagram depicts the sequence of events and interactions that occur when playing a trivia game. Within the time limit, players answer questions, check real-time scores, communicate via chat, request/share tips, view right answers, and track their individual and team performances.

Used Cloud Services:

GCP – Firestore + Cloud Function

Justification:

There are 2 main reasons to prefer Firestore over DynamoDB

3. Real time updates

Real time update in firestore comes out of the box and changes to the data will be immediately pushed to connected clients while DynamoDB supports real-time updates through DynamoDB Streams, integrating it with clients can be a tedious task.

4. Structure Querying Simplicity

Firestore document structure is simple you can store any kind and any structure of data which will be identified by document name while DynamoDB involves creation of partition key to identify the document

Two reasons to prefer Cloud function over lambda are.

3. Language support

Cloud Functions provides larger language support for even languages like Ruby and .NET while lambda function language support is limited.

4. Billing details

The billing details are more descriptive with the invocation pattern graph and the cost of hosting it in cloud function is less compared to lambda.

Implementation:

The "Trivia Titans" platform provides a dynamic and engaging leaderboard experience by utilizing React JS and cloud functionalities, inspiring players to keep playing trivia games, improving their knowledge, and aiming to be at the top of the scoreboard.

This feature is implemented using React JS for frontend which uses cloud functions to fetch the data of both teams and individual scores and it will sort the data according to the points scored of each individual or team.

Leader boards of both kind are conditionally rendered inside the program so in ui, user can toggle between both using a dropdown menu.

The fetching will be done using 2 separate serverless functions and both will be accessing the firestore.The use of cloud functions ensures optimal performance and scalability, as the data retrieval process is handled efficiently by the cloud infrastructure. This ensures a smooth and seamless user experience even during peak usage times.

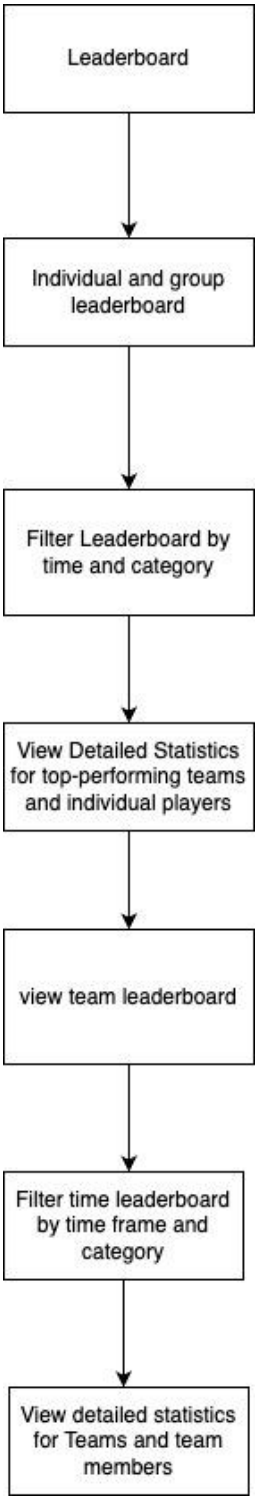


Figure 48: user management flow diagram

Test Case:

Table 4: Leader Board Test Case

Test	Description	Figures
User accesses the leaderboard	Leaderboard option is visible in the navbar.	Figure 45
Toggle Leaderboards	There is a dropdown from which user can toggle the leaderboards	Figure 46
View individual leaderboard	User can toggle to individual leaderboard and view it	Figure 47
View team leaderboard	Users can toggle to view team leaderboard.	Figure 48

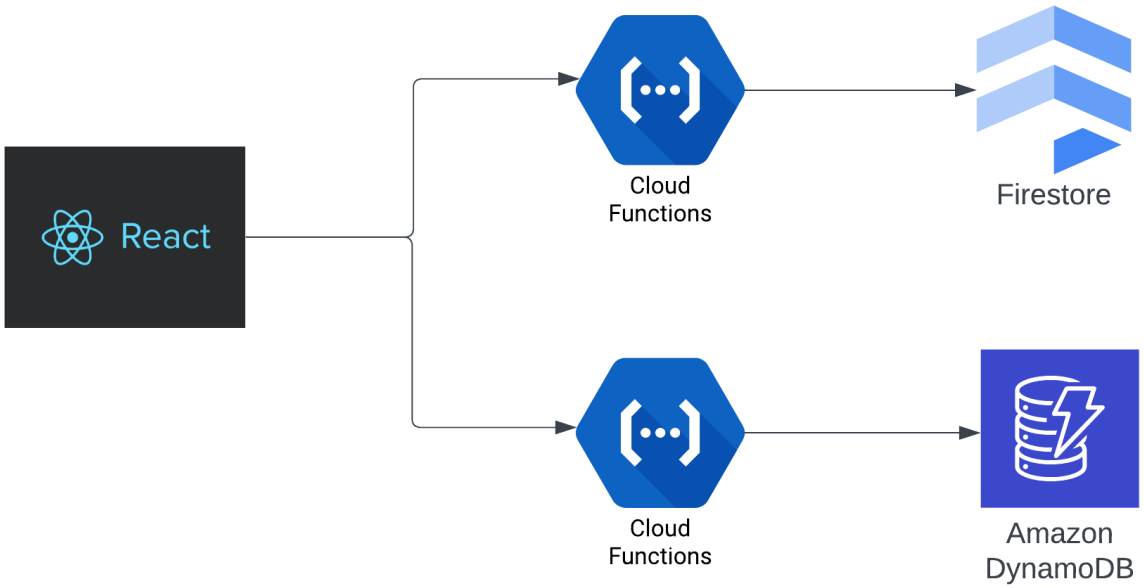


Figure 49: user module architecture

Individual leaderboard

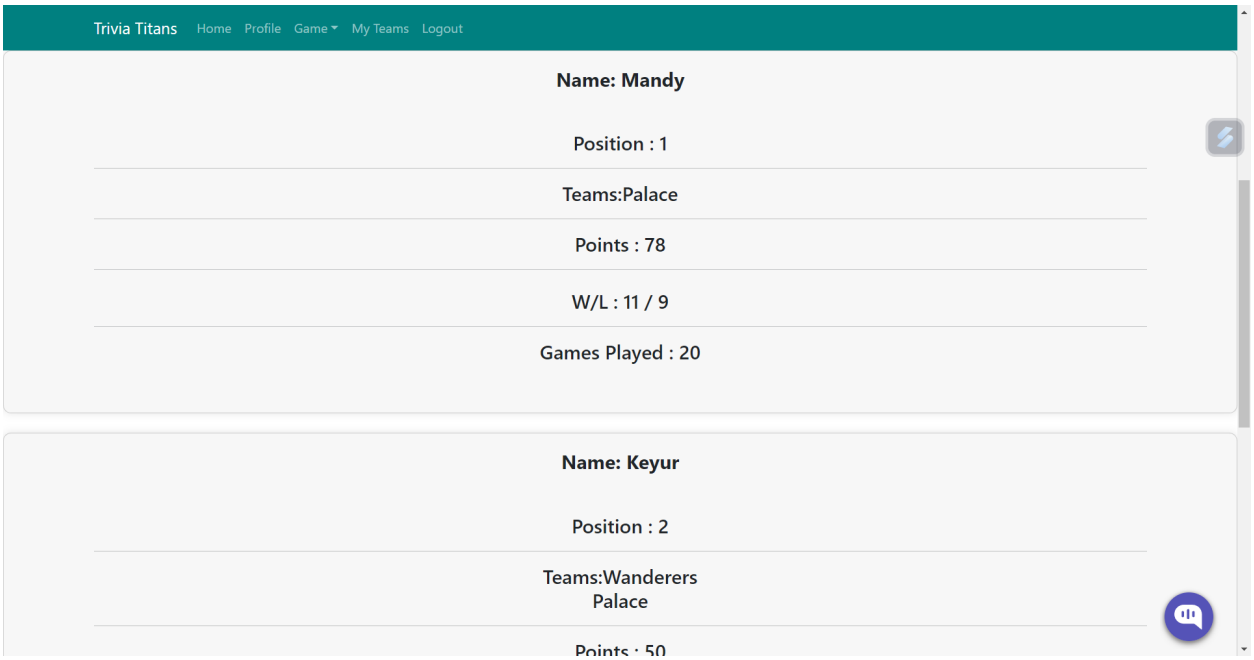


Figure 50: userInfo team

Team leaderboard

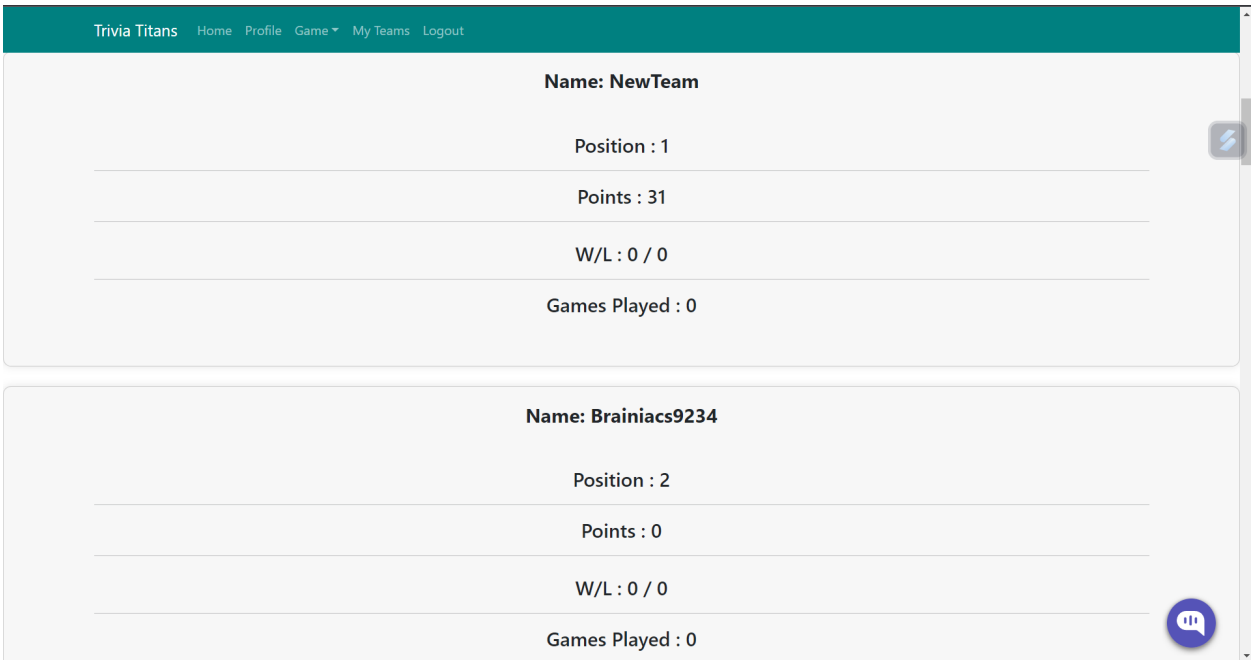


Figure 51: User info Team

Cloud Function:

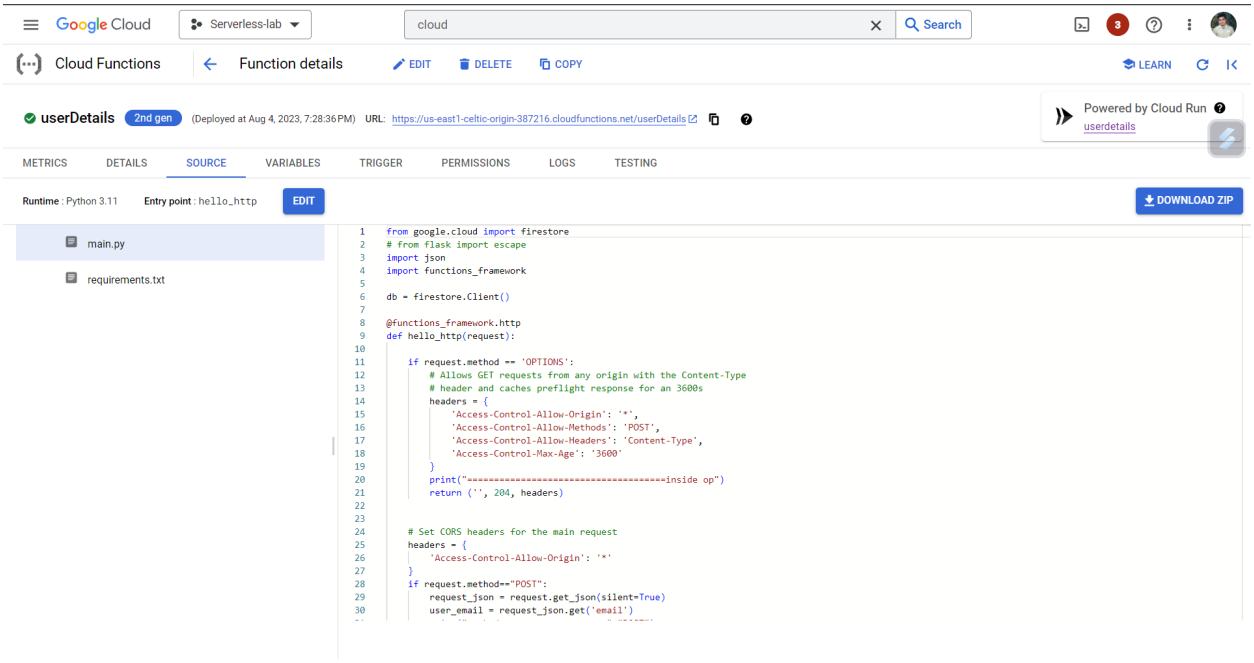


Figure 52: userDetails cloud function

2.7 Trivia Content Management (Administrators):

Proposed Cloud Services:

AWS – DynamoDB || Lambda Functions || SQS || SNS || Looker Studio || GCP – Firestore || Cloud Functions || Pub/Sub || AWS QuickSight

Used Cloud Services:

AWS Lambda, SQS, SNS, DynamoDB, SNS

Implementation:

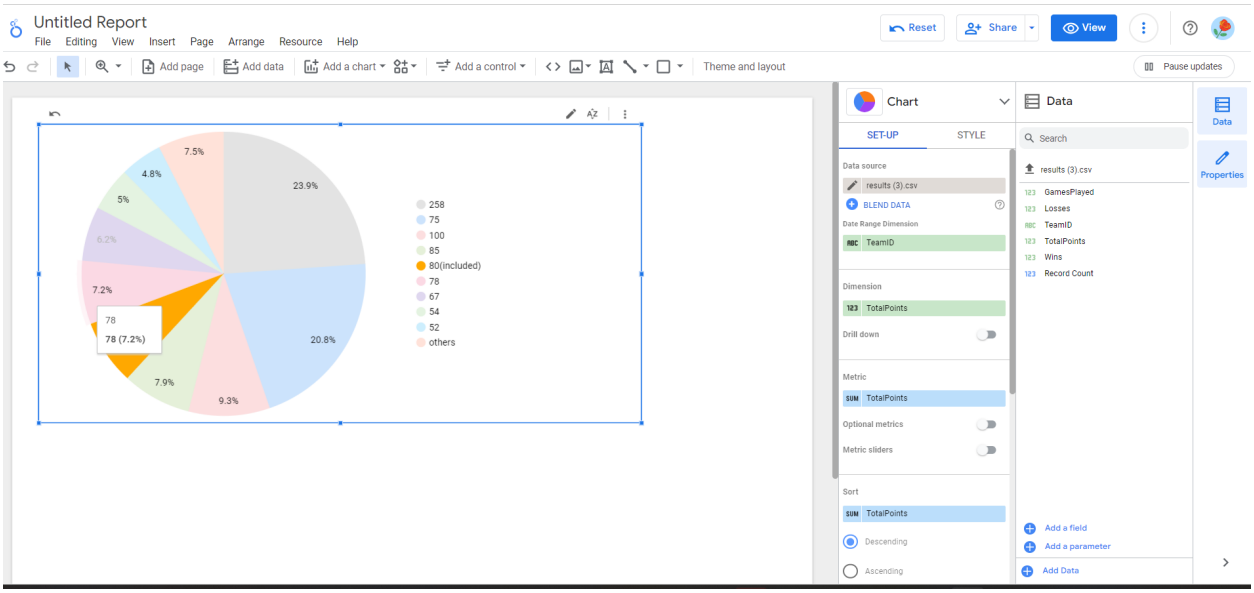


Figure 53: Trivia Content Looker Studio

Looker Studio has an intuitive and user-friendly interface that enables non-technical people to examine data, make reports, and create visualisations without the need for complex scripting or SQL skills. This openness allows a broader range of people to gain valuable insights from data.

Looker Studio provides a wide range of data visualisations, such as charts, graphs, and dashboards, making it easier for users to visually understand data patterns, trends, and correlations. Interactive visualisations improve data exploration and create a better knowledge of complex datasets.

Looker Studio interacts with a variety of data sources and consolidates data into a centralised repository. This one source of truth assures data integrity and reduces data silos, allowing teams to access trustworthy and up-to-date information more easily.

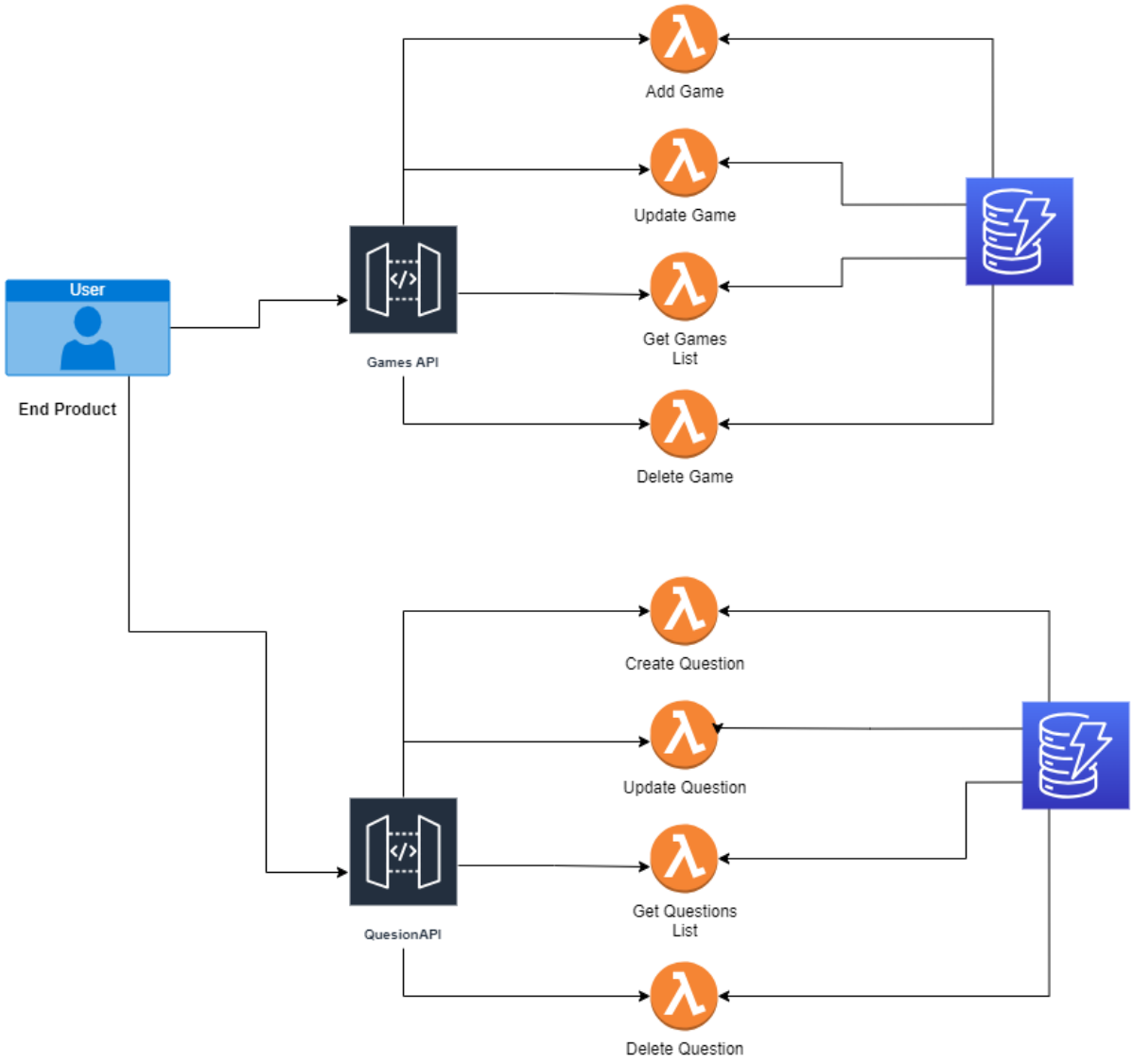


Figure 54: Cloud Architetcure diagram for Games and Questions CRUD

for the above diagram, I am using aws services like AWS API Gateway and AWS lambda such that it is handling the data from backend part in AWS management console. Additionally, "Add Trivia Question" is the first stage in the procedure. This is symbolised by a process symbol (rectangle with rounded sides) labelled "Add Trivia Question." When this stage is completed, the user is prompted to enter the new trivia question's question, answer alternatives, right answer, category, and difficulty level.

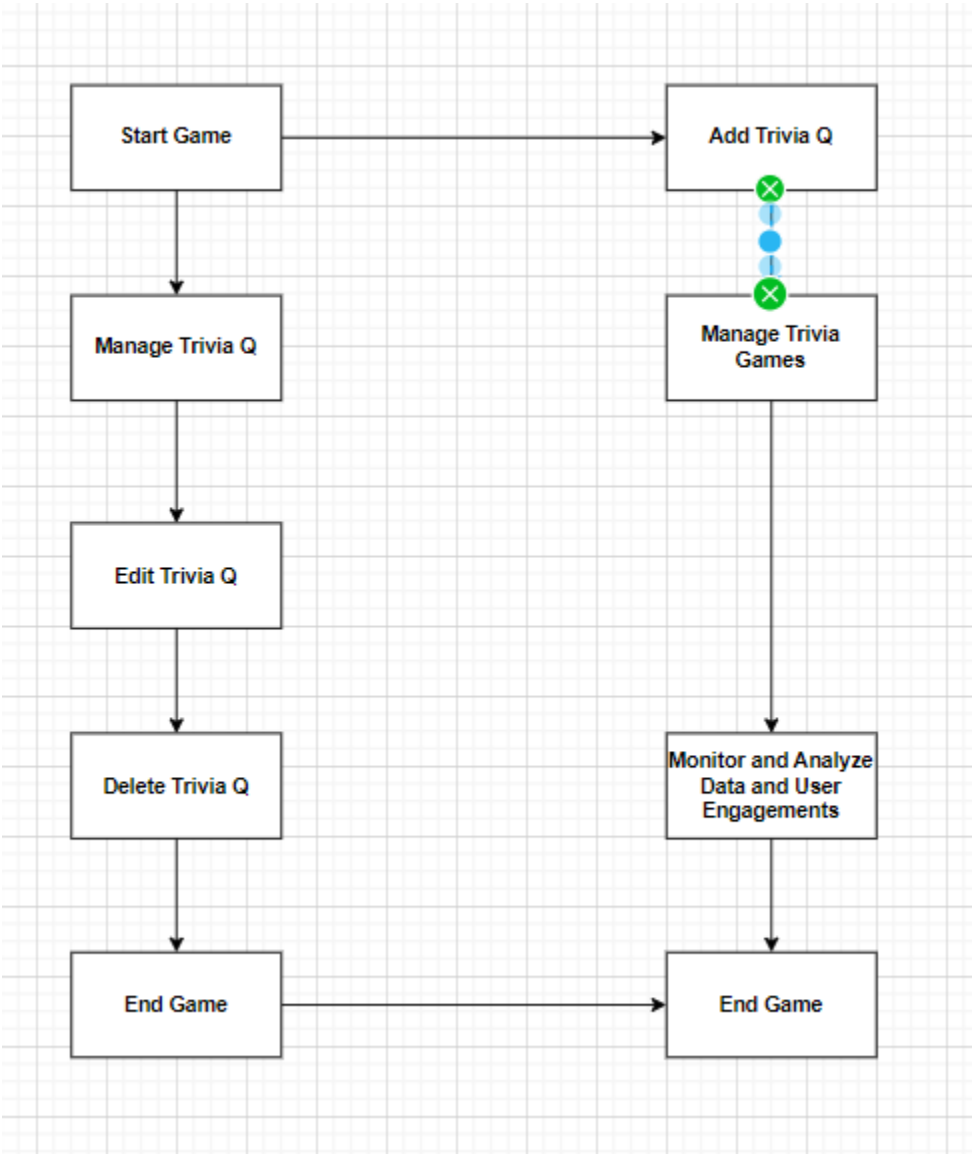


Figure 55: Worflow Diagram

Test Case:

Add, Edit, and Delete Trivia Questions as a Test Case
Verify your ability to add, amend, and delete trivia questions, as well as their category and difficulty level.
Prerequisite: The user has logged into the trivia app and has access to the question management options.
Steps:
a. Introduce a new trivia question:
b. Press the "Add Question" button.
c. Fill in the blanks with the question, answer choices, right answer, category, and difficulty level.
d. Press the "Save" button.
e. Revise an already existent trivia question:
f. From the list of questions, choose a previously inserted question.
g. Press the "Edit" button
h. Change the question, response options, right answer, category, or level of difficulty.
i. Press the "Save" button.
j. Remove the following trivia question:

Verify your ability to add, amend, and delete trivia questions, as well as their category and difficulty level.
Prerequisite: The user has logged into the trivia app and has access to the question management options.

Steps:

a. Introduce a new trivia question:
b. Fill in the blanks with the question, answer choices, right answer, category, and difficulty level.
c. Press the "Save" button.
d. Revise an already existent trivia question:
e. From the list of questions, choose a previously inserted question.
f. Press the "Edit" button.
g. Change the question, response options, right answer, category, or level of difficulty.
h. Press the "Save" button
i. Remove the following trivia question:

2.8 Notifications and Alerts:

The Notifications and Alerts feature plays a crucial role in the **Trivia Titans** application, ensuring users stay up-to-date with all essential events and updates. Real-time notifications for game invites, team updates, the availability of new trivia games, achievements unlocked, and leaderboard rank changes are provided through this function. Users are able to respond quickly to game invitations, monitor their progress, interact with their teams more successfully, and stay up to date on new trivia games by receiving these timely messages.

This feature ensures effective, immediate communication, improving the user experience within the application. It does this by utilizing cloud services like AWS DynamoDB, Lambda Functions, SQS, SNS, and AWS API Gateway Websocket.

Proposed Cloud Services:

AWS – DynamoDB || Lambda Functions || SQS || SNS || GCP – Firestore || Cloud Functions || Pub/Sub

Used Cloud Services:

AWS Lambda, SQS, SNS, API Gateway WebSocket

Implementation:

Steps for handling notifications and alerts are as below:

Step 1: Queueing Notifications:

A common SQS (Simple Queue Service) queue will be created to aggregate notification messages from all application modules. This queue will serve as a centralized point for collecting all notification and alert data. The queued message will contain a "type" field, which can take the values "PUSH" or "EMAIL," indicating the type of notification to be sent.

Step 2: Lambda Trigger:

A Lambda function is set up to get triggered whenever a new message arrives in the SQS queue. This function serves two primary events: one for handling messages from the queue and the other for managing WebSocket connections via the API Gateway.

Step 3: Email Notifications:

When a new message enters the queue and triggers the Lambda function, the function checks the message type. If the type is "EMAIL," the function calls the SNS (Simple Notification Service) to send an email notification to the user. The SNS is capable of sending these notifications to all users who have subscribed to this particular notification type.

Step 4: Push Notifications:

If the type is "PUSH," the Lambda function checks for an existing WebSocket connection. If a connection is found, the function sends a push notification to the user via this WebSocket connection.

Step 5: WebSocket Disconnection:

In the case where there is no active WebSocket connection (for instance, if the user is offline or the client is closed), the function stores the message temporarily instead of discarding it.

Step 6: Reconnecting WebSockets:

Whenever a new WebSocket connection is established, the function checks for any stored messages for that user. If any are found, they are immediately sent to the user as push notifications. This ensures that the user does not miss any crucial notifications, even when they are temporarily offline.

Service Architecture:

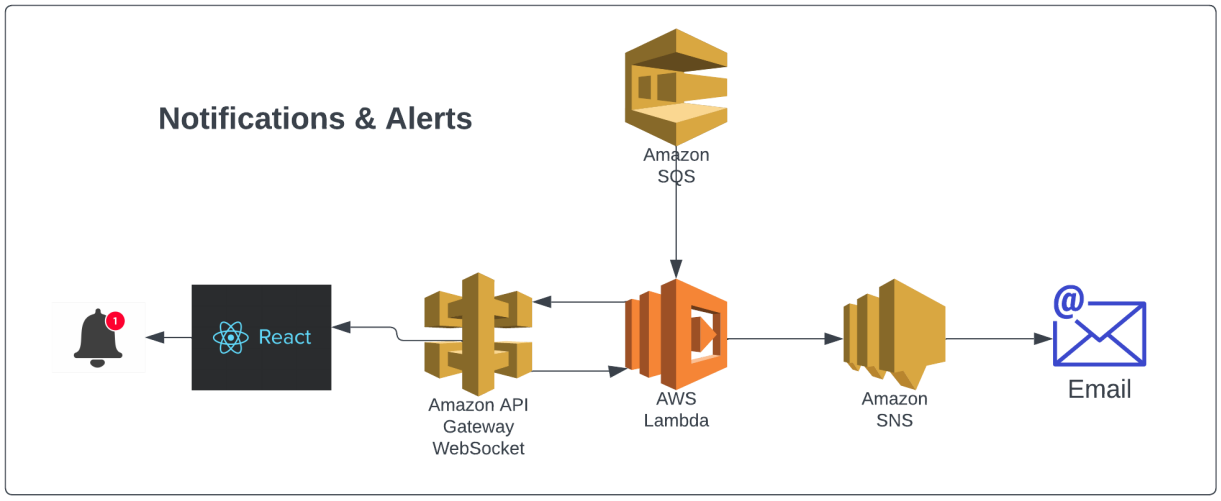


Figure 56: Notification queue to handle notification messages

This architecture design ensures a seamless, user-centric notification and alert system that can effectively scale to handle increased user loads and provide an exceptional user experience.

Using AWS services over GCP because of a few factors. Firstly, AWS SQS handles a high number of requests over GCP Pub/Sub. Hence, it provides higher scalability for handling multiple messages in real time. Furthermore, AWS has inbuilt WebSocket integration support with API Gateway. It provides reliable integration functionality over GCP.

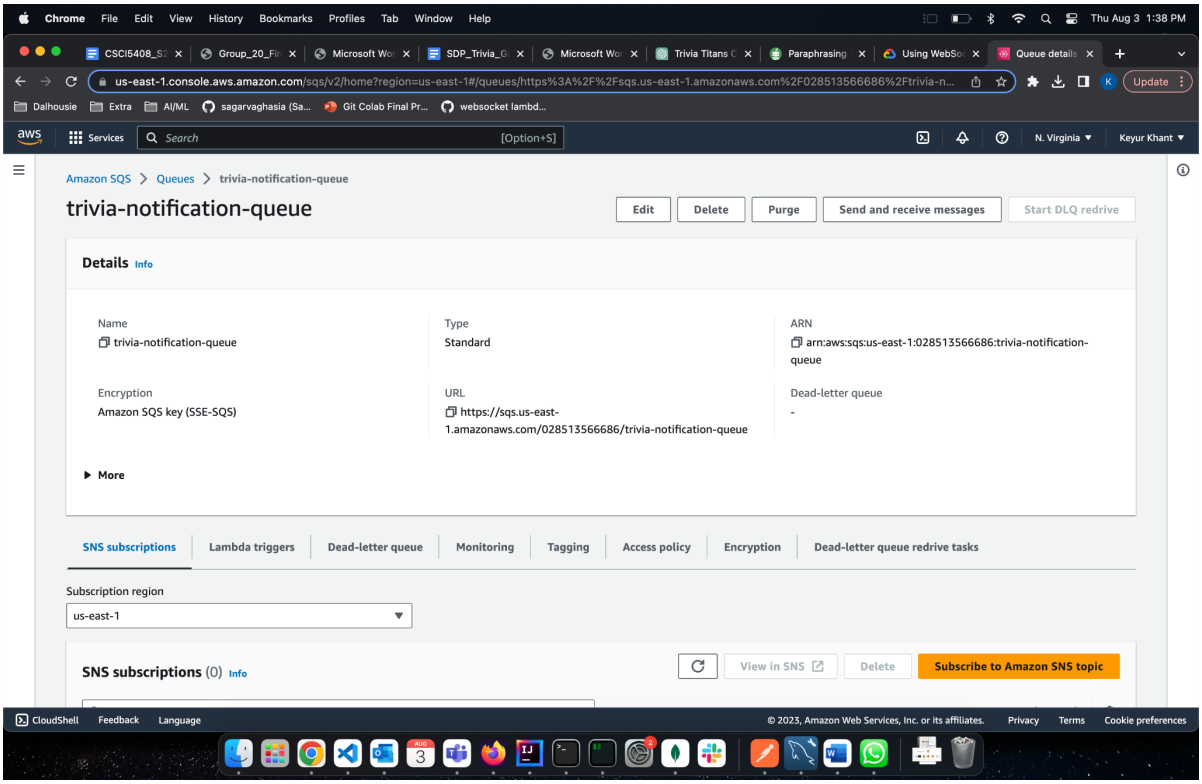


Figure 57: Notification queue to handle notification messages

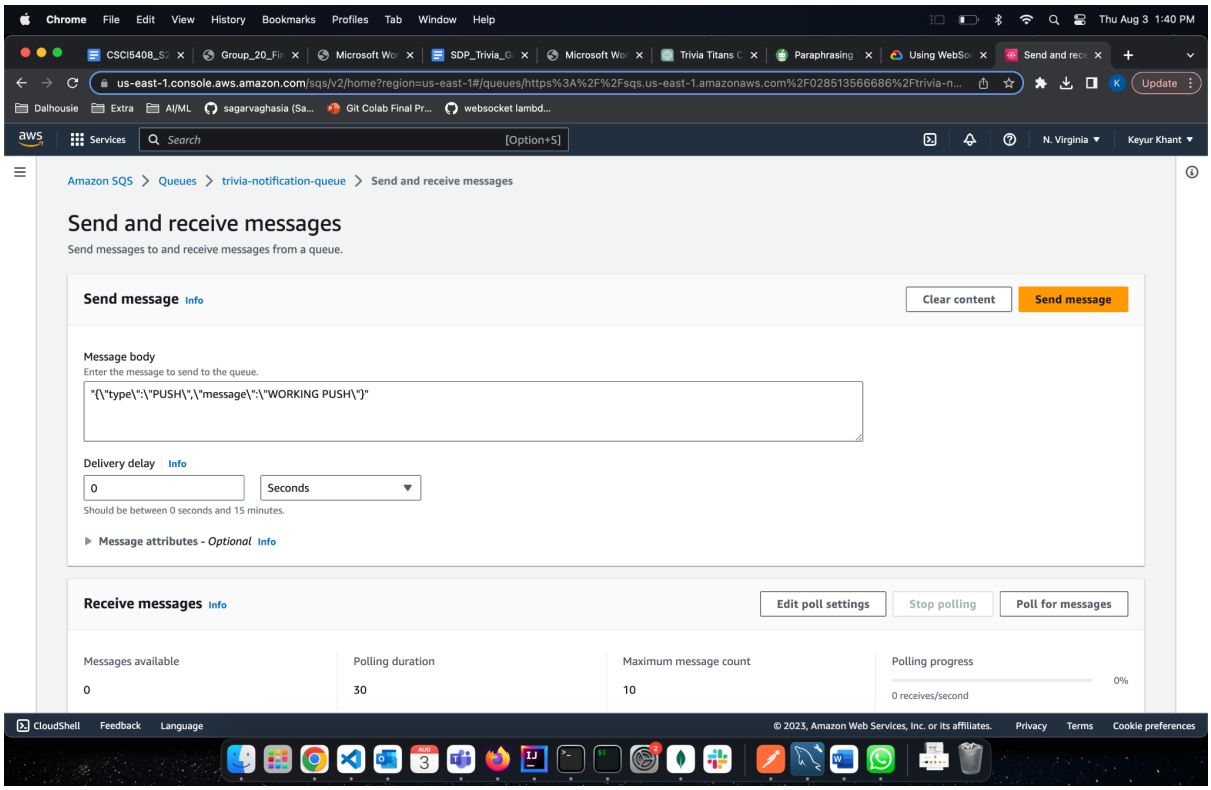


Figure 58: Sample message type for sending PUSH or EMAIL notification

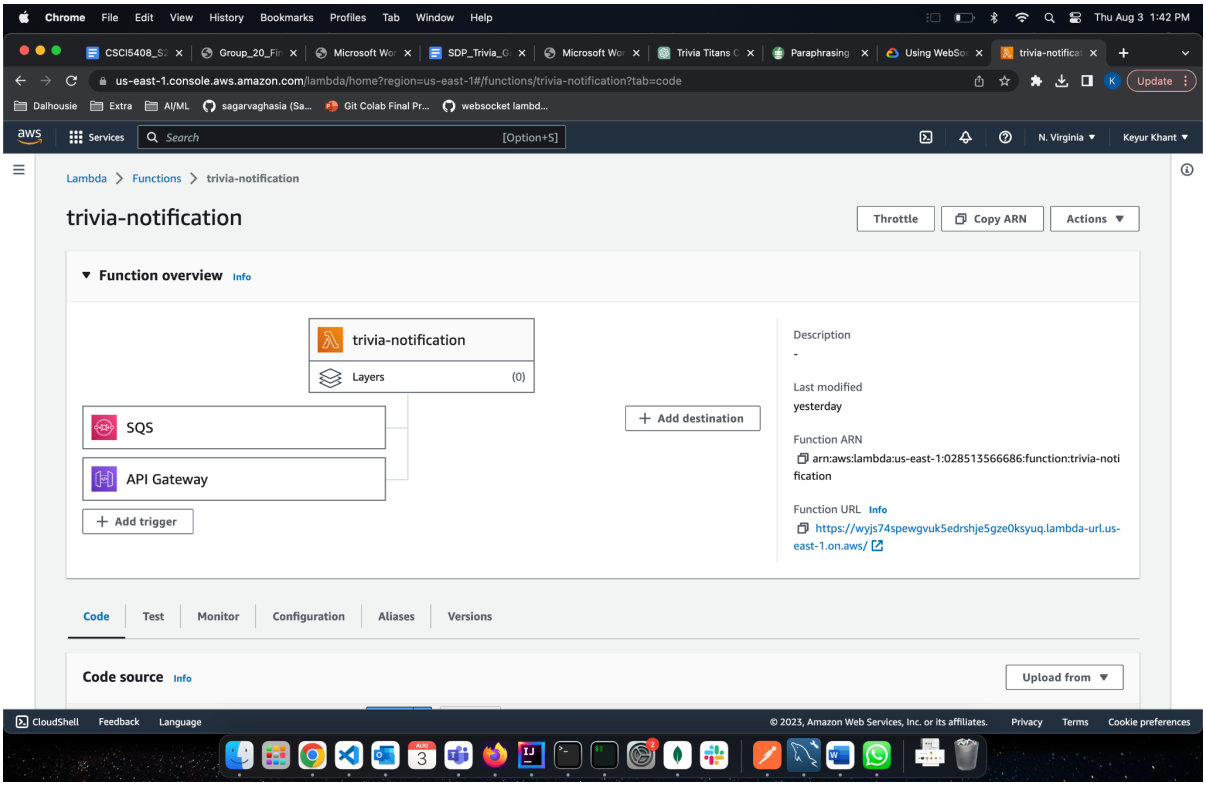


Figure 59: Notification Lambda trigger over SQS or WebSocket

The Notification Lambda function is triggered through either AWS SQS (Simple Queue Service) or a WebSocket connection. When a new notification is queued in SQS, this acts as an event source for the Lambda function, activating it to process the message and determine the type of notification to be sent. Similarly, for real-time push notifications, the Lambda function is triggered when it detects an active WebSocket connection. The Lambda function then sends the notification directly through this connection, ensuring an immediate, real-time alert for the user.

```
// Notification type 1
{
    type: "PUSH",
    message: "Welcome to Trivia Lobby"
}

// Notificatio type 2
{
    type: "EMAIL",
    message: "New achievement unlocked"
}
```

Figure 60: Two types of notification messages

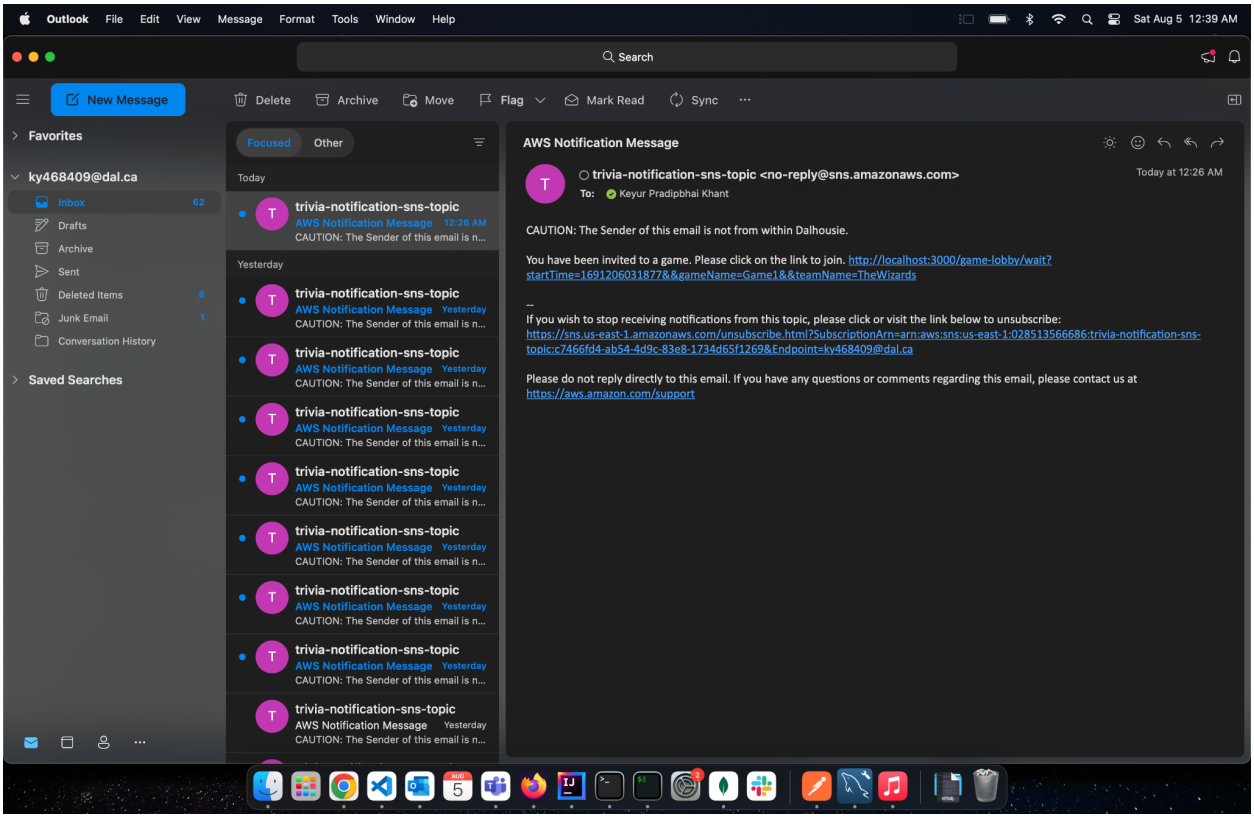


Figure 61: Email typed notification for game invitation

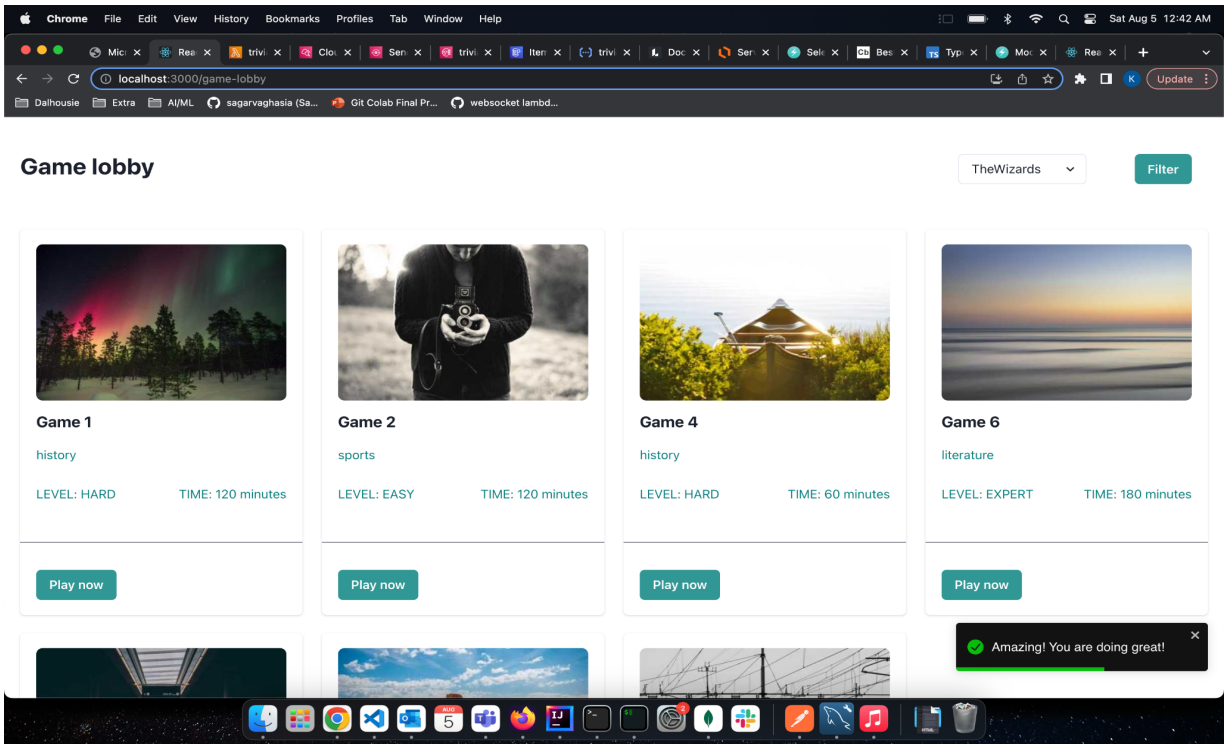


Figure 62: PUSH notification in bottom right corner

Test Case:

Table 5: Notification and Alerts Test Case

Test	Description	Figures
Game Invitation Sending to the team member	Whenever a new game is created by any team member, an email notification is getting sent to all the users in the team.	Figure 56
User welcome message on game lobby	User is getting notified via PUSH notification over navigating on the game lobby page.	Figure 57
PUSH notification on the game started	New PUSH notification is getting shot over user joining and game start.	Figure 58
Email for achievement unlock on leaderboard	User is getting notified over new achievement unlock and any update on the leaderboard.	Figure 59

2.9 Automated Question Tagging:

The automated question tagging uses Google Cloud Natural Language API's classifyText function. It takes the question into the document parameter and executes the classifyText function. It returns a list of confidence numbers and a list of tags associated with the confidence number.

To just get one tag for a question, only the highest confidence number's corresponding tag is stored.

Used Cloud Services:

GCP - Natural Language API + Cloud Function + Firestore/ DynamoDb

Justification:

There are again 2 reasons for choosing this stack of cloud services for this module.

1. As I had done module 3 using AWS, I planned on doing this using GCP services.
2. The AWS Comprehend is not available in the academy account so I went with GCP only.

Implementation:

The implementation was quite straightforward. The scripts in NL API documentation are used and modified to fetch the root tag with the highest corresponding number. By root tag, I mean the general tag. If the tag is like 'science/physics/thermodynamics', the root tag would be 'science' [1].

What was planned vs what was built:

Initially, the code was written in Python. But on deploying it on cloud function, it gave TypeError and no proper traceback log. As a result, I switched to node.js and it worked perfectly with cloud function.

Test Case:

Table 6: Automated Question Tagging Test Case

Test	Description	Figures
The function returns a tag for the question input	The output should be a relevant tag	Shown below
The function returns only one tag and it is the root tag	The output should be only one tag and it should not have any subsection	Shown below

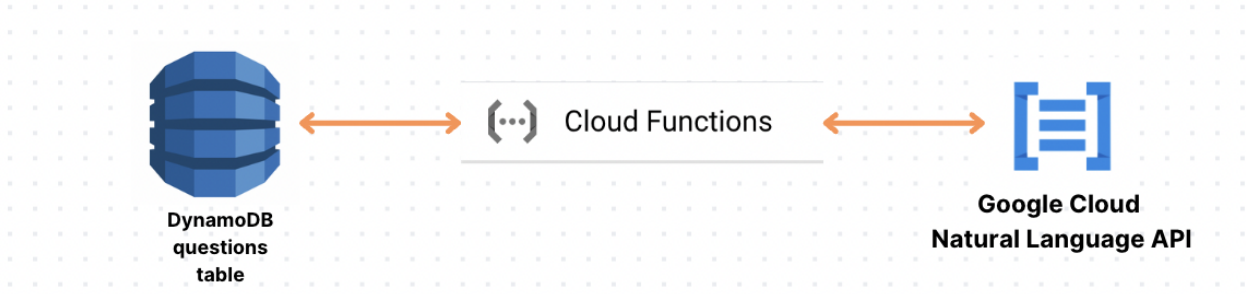


Figure 63: Diagram

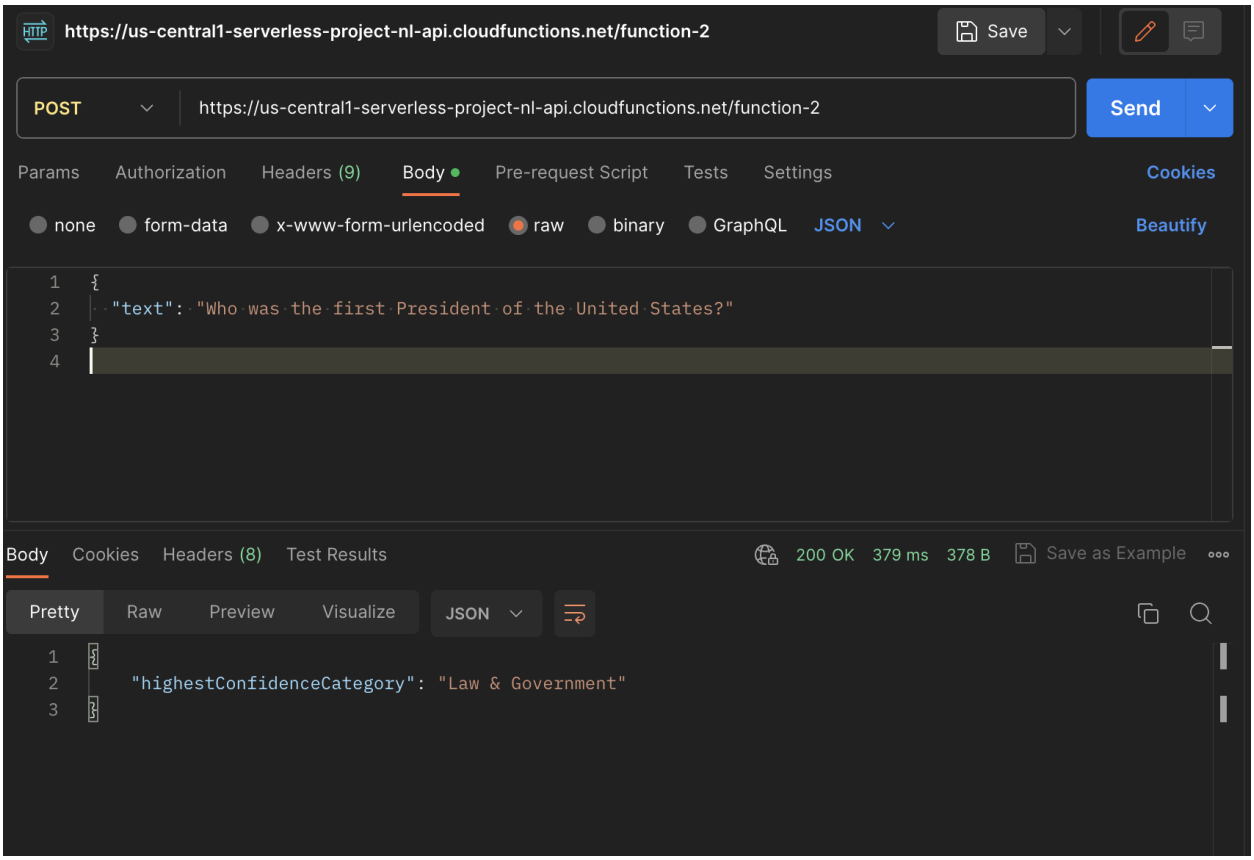


Figure 64: Response payload (Output of cloud function NL API)

2.10 Virtual Assistance:

Introduction:

The Virtual Assistance module is essential to improving user interaction and satisfaction with our product. It offers two key chatbot features that act as virtual assistants and help users with a range of activities and inquiries. The implementation and justification of the selected cloud services are covered in detail in this paper, with a special emphasis on the decision to use AWS Lex for the chatbot features.

1. Navigation Support Chatbot:

Implementation:

The Navigation Support Chatbot is integrated into the application's front end to offer real-time assistance to users. When users interact with the chatbot, it listens for natural language queries and processes them to understand the user's intent.

Cloud Services:

We have selected AWS Lex, a natural language processing technology from Amazon Web Services, for this chatbot. We are able to easily build conversational interfaces with AWS Lex. We create unique intents, utterances, and slots to mold the chatbot's actions to the demands of our particular application.

Justification:

1. Advanced Natural Language Understanding:

The extensive natural language processing capabilities of AWS Lex are one of the main justifications for using it. Lex uses powerful machine learning algorithms, allowing the chatbot to effectively comprehend and interpret human inquiries. As a result, the user and the chatbot can have more productive and human-like conversations.

2. Seamless Integration with AWS Services:

AWS Lex seamlessly integrates with other AWS services such as AWS Lambda and DynamoDB. This integration allows the chatbot to access backend data and perform various operations efficiently. The seamless communication between Lex and other services streamlines the development process and enhances overall efficiency.

3. Cost-Effectiveness:

AWS Lex operates on a pay-as-you-go model, where we are charged only for the resources used. Additionally, the serverless nature of AWS Lambda means that we do not need to manage servers, leading to cost savings and reduced operational overhead.

2. Dynamic Database Search Chatbot:

Implementation:

The Dynamic Database Search Chatbot is designed to enable users to retrieve game scores and related statistics for specific teams. It is integrated into the application's front-end and interacts with backend APIs to access the database.

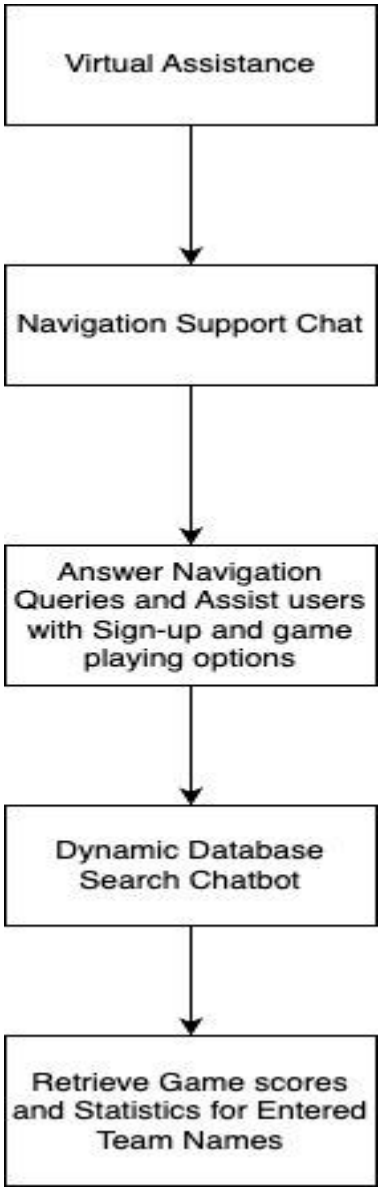


Figure 65: Flow Diagram

Cloud Services:

For this chatbot, we continue to leverage AWS Lex for natural language understanding. Additionally, AWS Lambda Functions and DynamoDB are used as backend services to process user queries and retrieve data from the database.

Justification:

1. Seamless Integration with Backend Services: The integration of AWS Lex with AWS Lambda and DynamoDB ensures a smooth flow of data between the chatbot and the database. This allows the

chatbot to fetch real-time scores and statistics for the entered team names, providing users with accurate and up-to-date information.

2. Scalability and Reliability: AWS Lambda and DynamoDB are highly scalable and reliable services. As our application's user base grows, these services can handle increasing traffic and maintain consistent performance, ensuring a seamless user experience.

Conclusion:

The Virtual Assistance module, powered by AWS Lex, adds significant value to our application, creating a user-friendly and engaging experience for our users. The Navigation Support Chatbot allows users to access sign-up and game-playing options effortlessly, while the Dynamic Database Search Chatbot empowers users to retrieve game scores based on entered team names. By leveraging AWS Lex's advanced natural language understanding capabilities and its integration with AWS Lambda and DynamoDB, we ensure a robust and scalable virtual assistance solution for our application.

Benefits of Using AWS Lex:

1. Advanced Natural Language Understanding: AWS Lex's sophisticated machine learning algorithms enable accurate interpretation of user queries, leading to more effective conversations between users and chatbots.

2. Seamless Integration with AWS Services: The smooth integration with other AWS services streamlines the development process and enhances overall efficiency, allowing the chatbots to access backend data and perform operations seamlessly.

3. Cost-Effectiveness: The pay-as-you-go model and serverless nature of AWS Lex, Lambda, and DynamoDB ensure cost savings and reduced operational overhead.

4. Scalability and Reliability: AWS services are designed to handle high loads and maintain consistent performance, making them ideal for accommodating potential future expansions of our application.

Overall, the Virtual Assistance Module powered by AWS Lex reinforces our commitment to providing a user-centric, efficient, and engaging experience for our application's users.

3. Service Architecture:

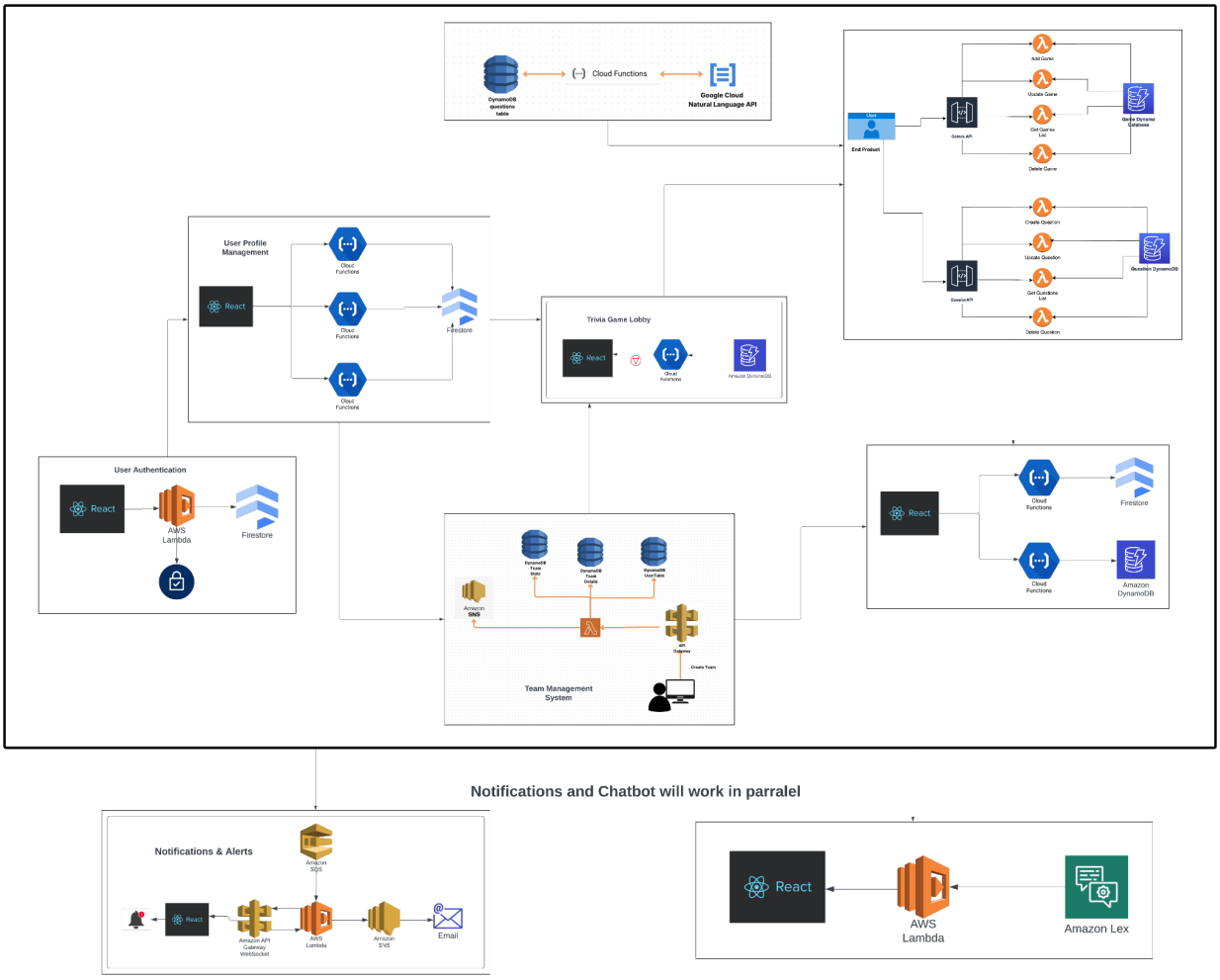


Figure 66: Architecture

The Trivia Game's cloud architecture spans various layers including the user interface, server-side processing, and data storage. It uses a frontend application for user interaction, serverless functions for processing requests, and a NoSQL database for maintaining game, user, and team data. Additionally, it leverages AI and messaging services for enhancing user experience and providing real-time updates. This architecture is scalable, resilient, and cost-effective.

Worksheet:

Table 7: Worksheet

Name	1st Module	2nd Module
Keyur Khant	Module 4	Module 8
Viral Siddhpura	Module 5	Module 7
Kush Sutaria	Module 3	Module 9
Nilesh Kopparty	Module 1	Module 10
Shreyas Nagaraja	Module 2	Module 6

Member tasks done:

1. Keyur Khant

I have worked on module 4 (Trivia Game Lobby) and module 8 (Notification and Alerts). Both the features are completed by me. Furthermore, in the end, I have fully contributed to the whole team to integrate all features, solving bugs for chatbot integration and leaderboard. I have worked on a deployment task, where I have created a docker image for frontend application and deployed that containerized application to Google Cloud Run.

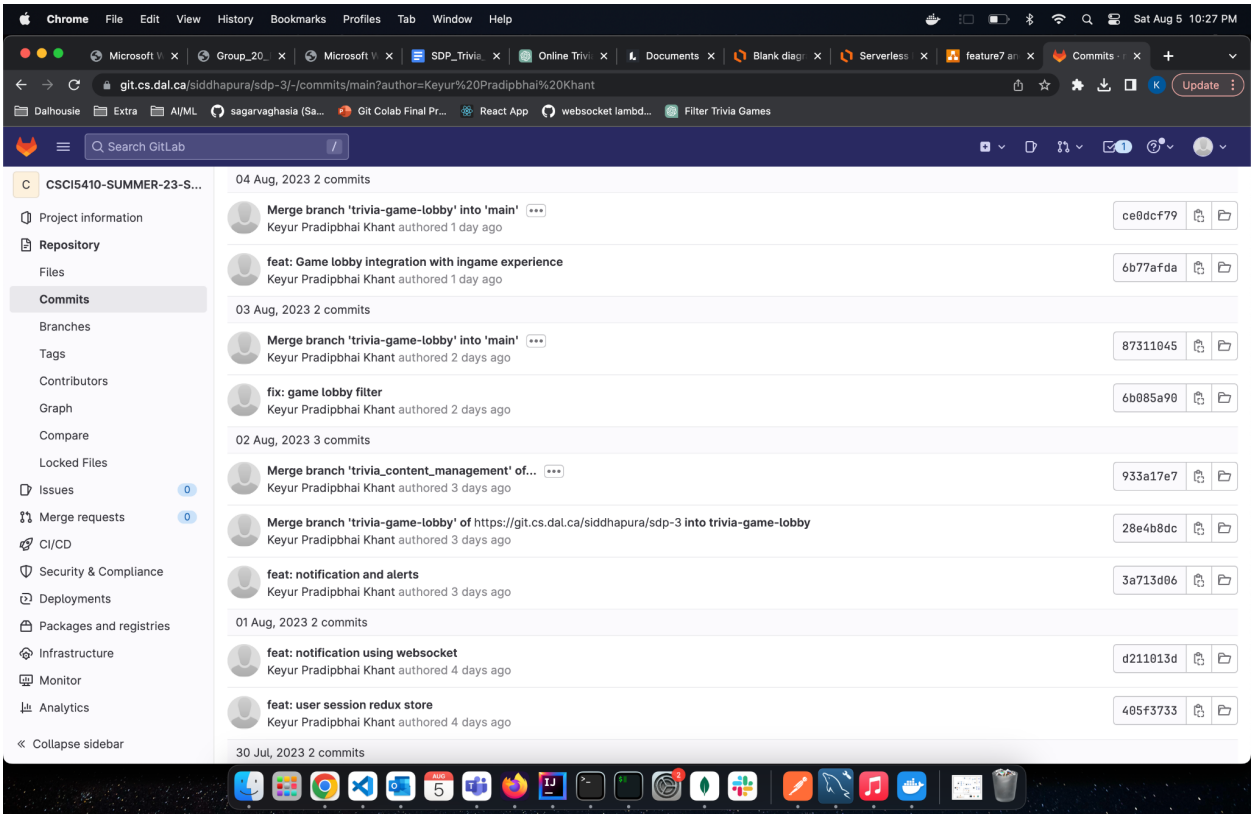


Figure 67: Project commit

2. Viral Siddhpura

I had module 5 and module 7 assigned to me. Both the modules are entirely done by me. In addition to this, at the last moment there were some issues in module 1, and as a result 2 members were working simultaneously on module 1 to finish as soon as possible. I do have my code in a branch but it wasn't merged. Apart from this, I've also helped debug and suggest potential solutions to my team members.

Code commits:

Final: Cdaa80538c58bd1ca2b8d1a66a55718e35379dc5 (date:08/05/2023)

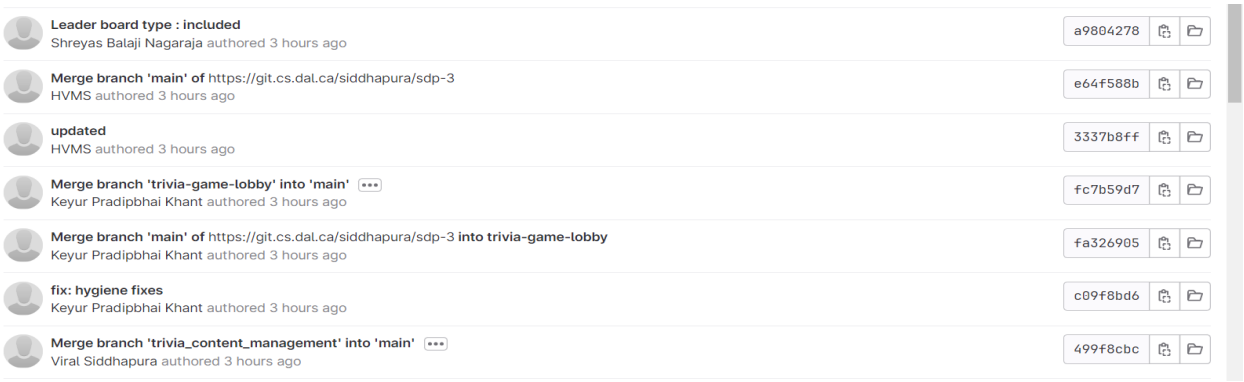


Figure 68: Gitlab Commits

3. Kush Sutaria
I had module 3 and module 9 assigned to me. Both the modules are entirely done by me. In addition to this, at the last moment there were some issues in module 1, and as a result 2 members were working simultaneously on module 1 to finish as soon as possible. I do have my code in a branch but it wasn't merged. Apart from this, I've also helped debug and suggest potential solutions to my team members.
Final Code commit: 499f8cbc90b07fa66cfdb43efe9034994d073441 (date:08/05/2023)

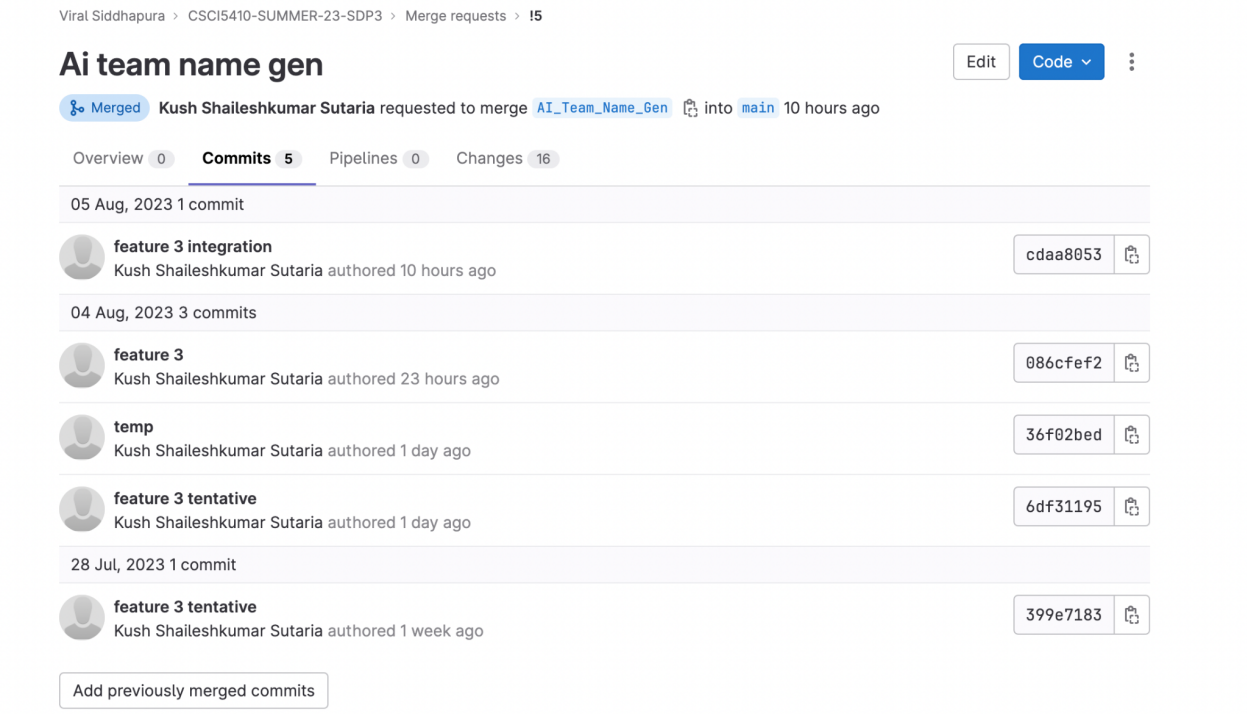


Figure 69: commit image

4. Nilesh Kopparty
I have been assigned module1 (user authentication) and module 10(virtual assistance). I have tried to work most of the work assigned on my own but however I have taken help from my team members.I have explained my both modules in the video attached wherein i have explained the code as well. I haven't committed wherein i worked on the code and sent it to my teammates and they merged it. Since I was having merge conflicts, I apologized at the last minute because I couldn't commit.

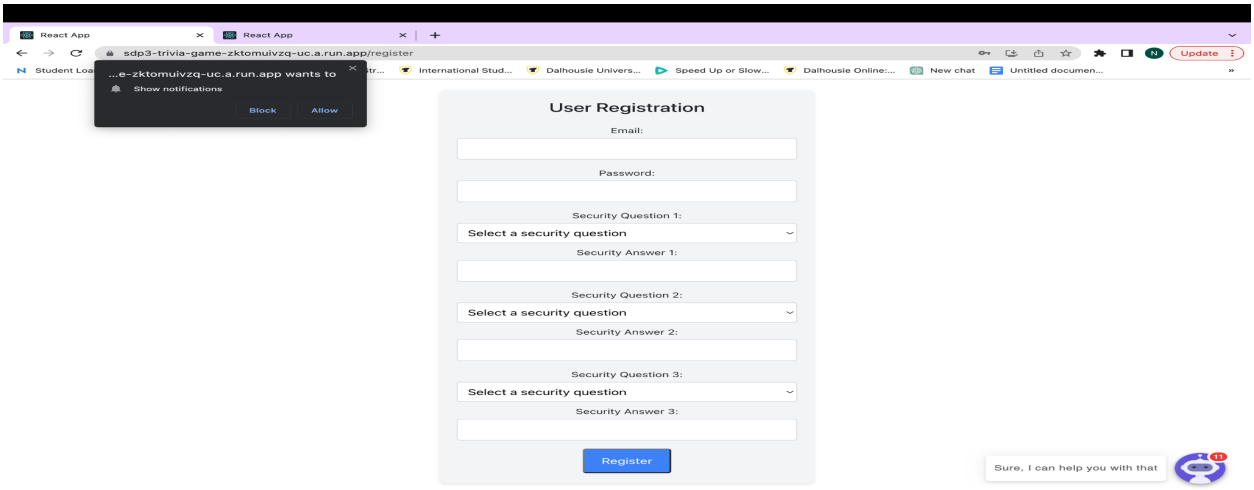


Figure 70: Registration

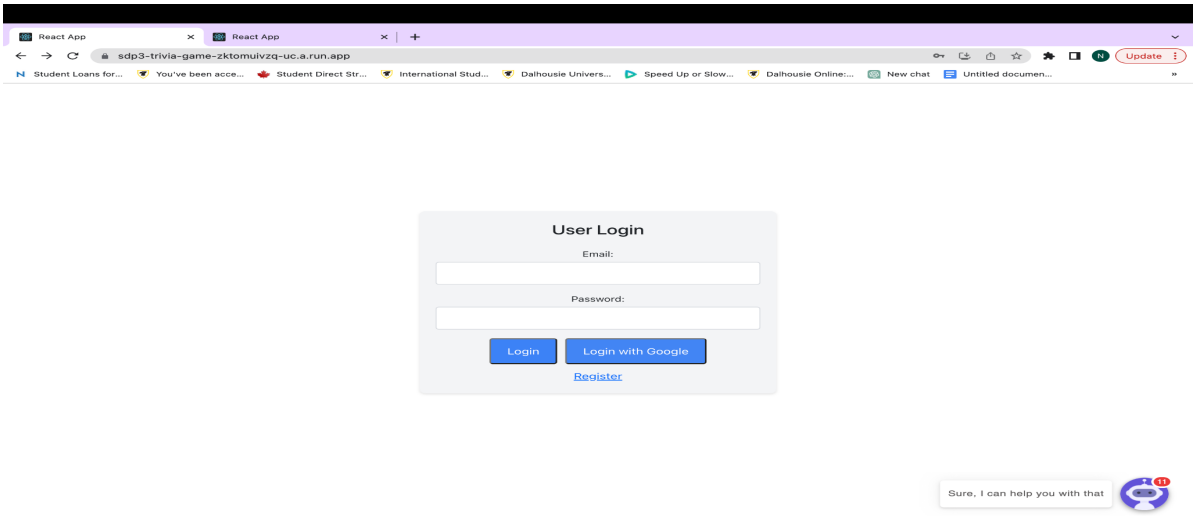
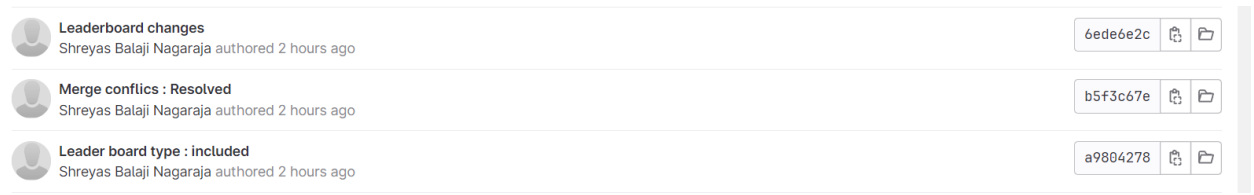


Figure 71: login

5. Shreyas Nagaraja

I had module 2 (User profile management) and module 6 (Leaderboards) assigned to me and is entirely done by me, some help was taken from teammates when it came to some exception handling and doubts. the commits are attached for reference and it is explained in the video with deployment.



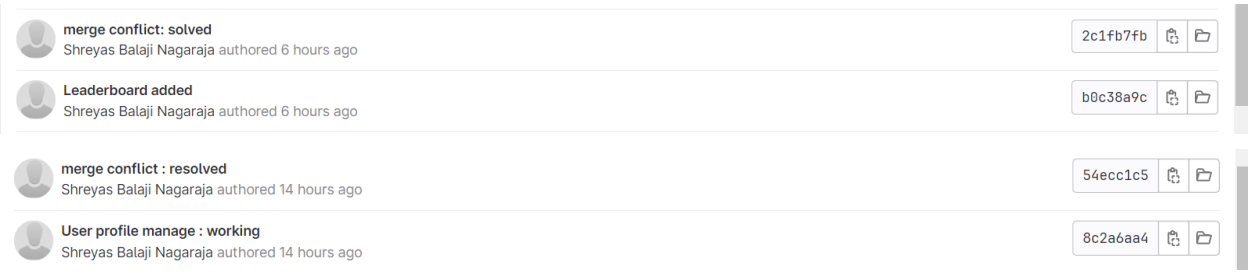


Figure 72: commit page

Git Repository:

https://git.cs.dal.ca/siddhapura/sdp-3

Sprint Plan:

Sprint 1 (June 1 - June 14): Initial Setup and Authentication

- Set up project infrastructure in the cloud (GCP and AWS)
- Implement User Authentication module (sign up, login, password recovery, 2-factor authentication)
- Design and implement the user profile management system
- Initial setup of the frontend application

Sprint 2 (June 15 - June 28): User Profile and Team Management

- Enhance user profile system with user statistics
- Implement team creation, invitation, and management
- Integration of AI for team name generation
- Set up in-app messaging for team communication

Sprint 3 (June 29 - July 12): Trivia Game Lobby and In-Game Experience

- Design and implement the Trivia Game Lobby
- Implement game browsing and filtering
- Develop the in-game experience (question answering, hint requesting, real-time scoring)
- Set up real-time communication for in-game collaboration

Sprint 4 (July 13 - July 31): Leaderboards, Admin Functions, and Final Touches

- Implement leaderboard system (global and category-specific leaderboards)
- Develop admin functions for trivia content management
- Setup automated question tagging and virtual assistance

- Integrate notification and alert system
- Perform final testing and troubleshooting
- Prepare for deployment

During the first sprint, we unfortunately missed our deadline to complete the feature specification. After a thorough discussion, we identified the root cause: a lack of communication. To address this issue in the next sprint, we have decided to include an additional meeting alongside our regular weekly meetings. This change has proven beneficial, as it has facilitated better integration and synchronization within the team, making our work more efficient.

The work wasn't done on a consistent basis during the break and after a week. Post that, the team started focusing less on recording meetings consistently and met/talked whenever they were stuck or needed help/info.

Meeting Logs:

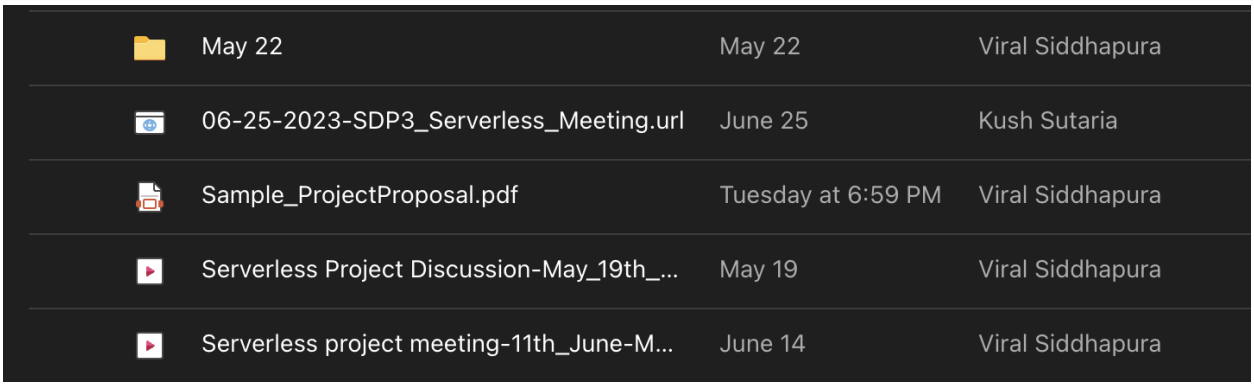


Figure 73: Meeting log

Figure: Recorded meetings

In addition to this, the team met on certain occasions in person or in a hybrid way.

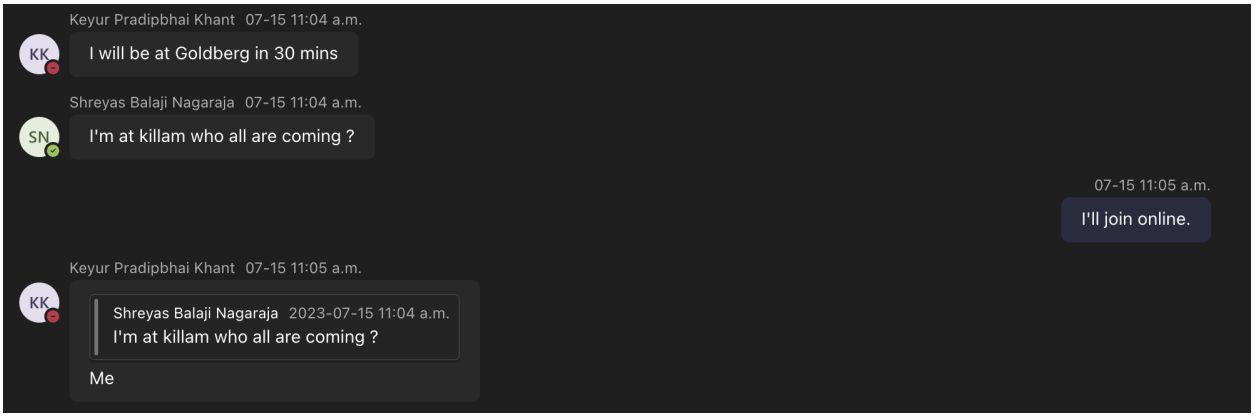


Figure 74: Meeting chat

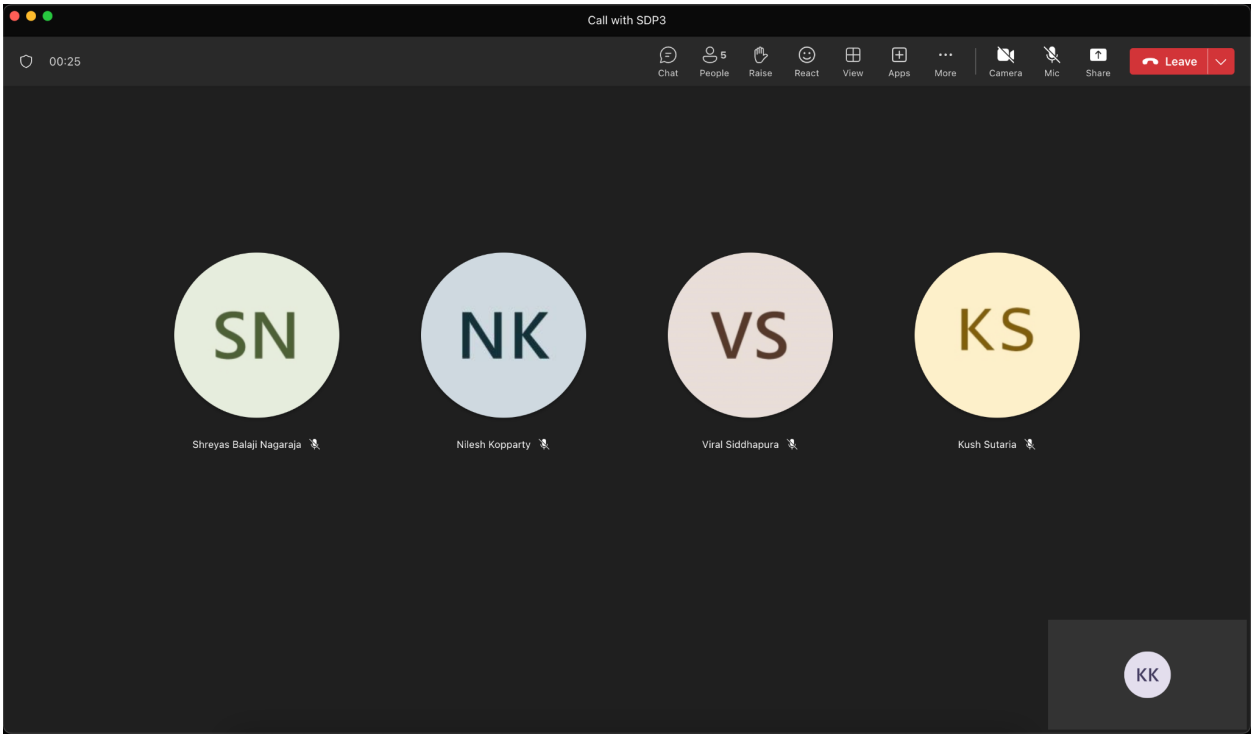


Figure 75: Meeting logs

Individual Contribution & Novelty

Keyur Pradipbhai Khant:

Novelty in Module 8:

1. Notification and Alerts, there are multiple ways to handle notification and there are many possibilities to handle notification such as keeping individual SQS and SNS services from each module.
2. In this project, there is one generic notification handler with only one SQS queue. All developers in application use single SQS to send notification throughout the application.
3. Furthermore, in this module, both PUSH notification and EMAIL notifications are getting handled.

Kush Sutaria:

Novelty in Module 3:

1. Created individual team topics so one team's invitation doesn't go to everyone. In addition to this, if a user has already accepted the invitation, they won't receive the invitation as they would have been unsubscribed from the topic.

2. As the SNS and SQS are not suitable for sending custom emails in a fanned-out process, I created an HTML form that is hosted on an S3 bucket. The form will have the team name fetched from the URL and the user will just have to provide an email id and their response to the invitation (accept or reject).

3. For optimization, while removing a user, I swap the user to be removed to the last place (if user 3 is to be removed from 10 users, user 3 is moved to user 10's place and vice versa, and user 10 is then removed.

Novelty in Module 9:

1. Only the tag with the highest confidence number is fetched. As the tag has subsections by default, only the root tag (the tag before the first '/') is stored.

Shreyas Balaji Nagaraja

Novelty in Module 2:

1. There are many ways to facilitate a profile picture change but I used the cloud storage way, it was efficient and used python blob, new image will just be uploaded to cloud storage and only the reference will be passed to document structure.

References:

[1] "Classifying content," *Google Cloud*. [Online]. Available: https://cloud.google.com/natural-language/docs/classifying-text. [Accessed: 06-Aug-2023].

[2] "Amazon DynamoDB - Boto3 1.28.20 documentation," *Amazonaws.com*. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html. [Accessed: 06-Aug-2023].

[3] "SNS - Boto3 1.28.20 documentation," *Amazonaws.com*. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sns.html. [Accessed: 06-Aug-2023].

[4] "OpenAI platform," *Openai.com*. [Online]. Available: https://platform.openai.com/docs/api-reference. [Accessed: 06-Aug-2023].

[5] "Firestore," *Firebase*. [Online]. Available: https://firebase.google.com/docs/firestore. [Accessed: 06-Aug-2023].

[6] "Cloud functions," *Google Cloud*. [Online]. Available: https://cloud.google.com/functions. [Accessed: 06-Aug-2023].

[7] *Amazon.com*. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 06-Aug-2023].

[8] "Cloud run," *Google Cloud*. [Online]. Available: https://cloud.google.com/run. [Accessed: 06-Aug-2023].

[9] "Fun trivia categories (20 trivia category ideas)," *Watercoolertrivia.com*. [Online]. Available: https://www.watercoolertrivia.com/blog/fun-trivia-categories. [Accessed: 06-Aug-2023].

[10] A. Garrity and S. Lemire, "185 best trivia questions with answers 2023," *TODAY*, 21-Jul-2022. [Online]. Available: https://www.today.com/life/inspiration/trivia-questions-rcna39101. [Accessed: 06-Aug-2023].

[11] "A Sample Tutorial - Boto3 1.28.20 documentation," *Amazonaws.com*. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html.

[Accessed: 06-Aug-2023].

[12] *Amazon.com*. [Online]. Available: https://aws.amazon.com/sqs/features/. [Accessed: 06-Aug-2023].

[13] *Amazon.com*. [Online]. Available: https://aws.amazon.com/lex/. [Accessed: 06-Aug-2023].

[14] “React,” *React.dev*. [Online]. Available: https://react.dev/. [Accessed: 06-Aug-2023]

[15] “Firestore,” *Firebase*. [Online]. Available: https://firebase.google.com/docs/firestore. [Accessed: 06-Aug-2023].

[16] *Amazon.com*. [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed: 06-Aug-2023].

[17] R. Venati, “Introduction to Amazon S3,” *Medium*, 28-Sep-2019. [Online]. Available: https://medium.com/@rahulvenati/day-5-introduction-to-amazon-s3-5d47d2962a3. [Accessed: 06-Aug-2023].

[18] “Cloud natural language,” *Google Cloud*. [Online]. Available: https://cloud.google.com/natural-language. [Accessed: 06-Aug-2023].