



DALHOUSIE
UNIVERSITY

Project Report

CSCI 5410 – SERVERLESS DATA PROCESSING

GROUP – 25

Harshil Shah	hs@dal.ca
Khushi Shah	kh482021@dal.ca
Mohit Sojitra	mh546676@dal.ca
Mansi Vaghasiya	mansi.vaghasiya@dal.ca
Edwin Adams	ed743899@dal.ca

Contents

LIST OF FIGURES	4
LIST OF TABLES.....	7
Feature Specification	8
1. User Authentication.....	8
2. User Profile Management	13
3. Team Management.....	17
4. Trivia Game Lobby.....	26
5. In-Game Experience.....	33
6. Leaderboards.....	39
7. Trivia Content Management.....	44
8. Notifications and Alerts	51
9. Automated Question Tagging	52
10. Virtual Assistance	57
Application Roadmap.....	64
User Authentication	65
User Management.....	69
Team Management	72
Trivia Game Lobby	72
In-game experience.....	73
Leaderboards	73
Automated Question Tagging	74
Virtual Assistant.....	74
Architecture.....	75
Worksheet.....	76
Module Owners.....	76
Paragraph on Individual Component:	76
1. User Authentication (Harshil Shah).....	76
2. User Profile Management (Harshil Shah)	77
3. Team Management (Khushi Shah)	77
4. Trivia Game Lobby (Mansi Vaghasiya)	77
5. In-game experience (Mohit Sojitra)	78
6. Leaderboard (Mohit Sojitra)	78

9. Automated Question Tagging (Mansi Vaghasiya)	78
10. Virtual Assistance (Khushi Shah).....	79
Sprint Plan	80
Individual experience:	81
Merge Requests Report	83
Code Commits Report	85
Meeting Logs	89
Screenshots of the meeting	92
Individual Contribution.....	94
Harshil Shah	94
Khushi Shah	94
Mohit Sojitra.....	94
Mansi Vaghasiya.....	95
Novelty.....	95
Harshil Shah	95
Khushi Shah	95
Mohit Sojitra.....	95
Mansi Vaghasiya.....	96
Cloud Run Hosted URL.....	96
References	97

LIST OF FIGURES

Figure 1: Firebase Authentication	8
Figure 2: Lambdas	9
Figure 3: Users DynamoDB table	9
Figure 4: GCP bucket for user profile image storage.....	10
Figure 5: API testing via Postman.....	10
Figure 6: API testing via Postman 2	11
Figure 7: Questions DynamoDB table.....	14
Figure 8: Lambdas	15
Figure 9: API Testing 1.....	15
Figure 10: API Testing 2	16
Figure 11: Tag Generation API.....	18
Figure 12: DynamoDB table – User Roles.....	18
Figure 13: Store Generated Team Names using AI API - DynamoDB table.....	19
Figure 14: Functional Testing 1.....	19
Figure 15: Invite entry mail on DynamoDB entry addition.....	20
Figure 16: Functional Testing 2 - Accept/Decline Invitation.....	20
Figure 17: Functional Testing 3 - Invite Status	21
Figure 18: Team management.....	21
Figure 19: Functional Testing for Team management 1	22
Figure 20:Functional Testing for Team management 2	22
Figure 21: Functional Testing for Team management 3	22
Figure 22:Functional Testing for Team management 4	23
Figure 23: Functional Testing for Team management 5	23
Figure 24: Functional Testing for Team management 6	23
Figure 25: Functional Testing for Team management 7	24
Figure 26: API testing 1	24
Figure 27: API testing 2	25
Figure 28: Game Type DynamoDB Table.....	28
Figure 29: Filtered GameType DynamoDB table	28
Figure 30: Lambda Functions	29
Figure 31: Games Dashboard	29
Figure 32: Filters - 1	30
Figure 33: Filter - 2.....	30
Figure 34: Dynamic Participants shown	31
Figure 35: Waiting Lobby	31
Figure 36: Waiting Lobby.....	32
Figure 37: Time Left	32
Figure 38: Lambda Functions	35
Figure 39: SQS Queue	36

Figure 40: DynamoDB	37
Figure 41: WebSocket Monitoring	37
Figure 42: Postman Testing.....	38
Figure 43: Buckets	40
Figure 44: AWS Athena	40
Figure 45: Quick Sight (Login Required)	41
Figure 46: QuickSight 2.....	42
Figure 47: QuickSight 3.....	42
Figure 48: Graph Testing Script	43
Figure 49: QuickSight provider-managed role	44
Figure 50: Services which QuickSight natively supports as data sources for analysis	44
Figure 51: Default QuickSight Access and Management console.....	45
Figure 52: AWS Glue Data Catalogs.....	46
Figure 53: AWS Glue Crawlers (Dump to S3)	46
Figure 54: Athena Catalog integrations (From Glue to S3 to Athena).....	46
Figure 55: HTTP API.....	47
Figure 56: DynamoDB Tables	47
Figure 57: AWS SNS Topics	47
Figure 58: Events Bridge recurring (4 hours) Trigger for Data-Source Lambda	48
Figure 59:GoogleSheets API Datasource.....	48
Figure 60: LookerStudio Data Source	48
Figure 61: GET all questions.....	49
Figure 62:GET all Games	50
Figure 63: PATCH – Updates game data upon completion of a live game for data collection....	50
Figure 64: Admin Email Notifications.....	51
Figure 65: Postman Testing 1	53
Figure 66:Postman Testing 2	53
Figure 67: GCP Firestore table – questions-tagging	54
Figure 68: Questions tagging	54
Figure 69: Cloud Functions	55
Figure 70: Game lobby	55
Figure 71: Question tags generation from frontend	56
Figure 72: Question tagging 2	56
Figure 73: Chatbot	58
Figure 74: Chatbot response 2	58
Figure 75: Chatbot response 3	59
Figure 76: Chatbot response 3	59
Figure 77: Chatbot fulfillment	60
Figure 78: Call Lambda from chatbot	60
Figure 79: Called Lambda	61
Figure 80: Intent calling lambda	61
Figure 81: Condition in chatbot execution	62
Figure 82; API testing	62

Figure 83: Application Roadmap [10]	64
Figure 84: Feature 1 Flow diagram [10]	65
Figure 85: Functional Testing Feature 1.....	66
Figure 86: Functional Testing 2 - Feature 1	66
Figure 87: Functional Testing 3 - Feature 1	67
Figure 88: Functional Testing 4 - Feature 1	67
Figure 89: Functional Testing 5 - Feature 1	68
Figure 90: Functional Testing 6 - Feature 1	68
Figure 91: Functional Testing 7 - Feature 1	69
Figure 92: User Management - Feature 2 Workflow [10]	69
Figure 93: User Dashboard	70
Figure 94: Update User Details.....	70
Figure 95: Verification mail on User register using Email ID	71
Figure 96: Flow Diagram Feature 3 [10].....	72
Figure 97: Flow Diagram Feature 10 [10].....	74
Figure 98: Architecture Diagram [10].....	75
Figure 99: Sprint Planning Excel	79
Figure 100: Sprint Commit 1	85
Figure 101: Sprint Commit 2	86
Figure 102: Sprint Commit 3	86
Figure 103: Sprint Commit 4	87
Figure 104: Sprint Commit 5	87
Figure 105: Sprint Commit 6	88
Figure 106: Sprint Commit 7	88
Figure 107: Sprint Commit 8	89
Figure 108: Meeting 1	89
Figure 109: Meeting 2	89
Figure 110: Meeting Minutes 1	90
Figure 111: Meeting 3 with meeting minutes.....	90
Figure 112: Meeting 4 with meeting minutes.....	91
Figure 113: MOM 3.....	91
Figure 114: MOM 4.....	92
Figure 115: Meeting 1	92
Figure 116: Meeting 2	93
Figure 117: Meeting 3	93
Figure 118: Meeting 4	94

LIST OF TABLES

Table 1: Functional Testing Result	11
Table 2: Planned vs Used services.....	12
Table 3: Functional Testing	12
Table 4: Functional Test Case Results.....	16
Table 5: Used Service VS Planned Services.....	16
Table 6: API endpoints	25
Table 7: Planned Vs Used Services	25
Table 8: Functional Testing Status	25
Table 9: Planned Service VS Used service	33
Table 10: Functional Testing	33
Table 11: API Gateway	34
Table 12: Lambda Functions	35
Table 13: Test Case	38
Table 14: Test case	38
Table 15: Planned Vs Used Services	39
Table 16: Planned Feature VS Used Features	43
Table 17: Planned VS Used	56
Table 18: Testing	57
Table 19: Planned VS Used Services	62
Table 20: Chatbot Functional Testing	62
Table 21: Module Owners	76
<i>Table 22: Sprint Planning</i>	80
Table 23: Merge Request Details	83

Feature Specification

1. User Authentication

How I planned it:

In this implementation, I planned to use AWS Cognito for user authentication and authorization [1].

User details, such as email, name, phone number, user type, and key (plain text), will be stored in AWS DynamoDB, which is a NoSQL database service. AWS Lambda, an event-driven serverless computing service, will handle storing user details to the database and generating and verifying cipher keys [2].

For the question-and-answer details, I will use GCP Firestore, a NoSQL document database provided by Google Cloud Platform (GCP). GCP Cloud Function, a serverless computing service, will be used to set and get question and answer details from the database.

How I built it:

Instead of AWS Cognito for user authentication and authorization, I've decided to use Firebase Authentication, which is a service provided by Google for user authentication. It is easier to implement and can work well with multiple sign-in services [2].

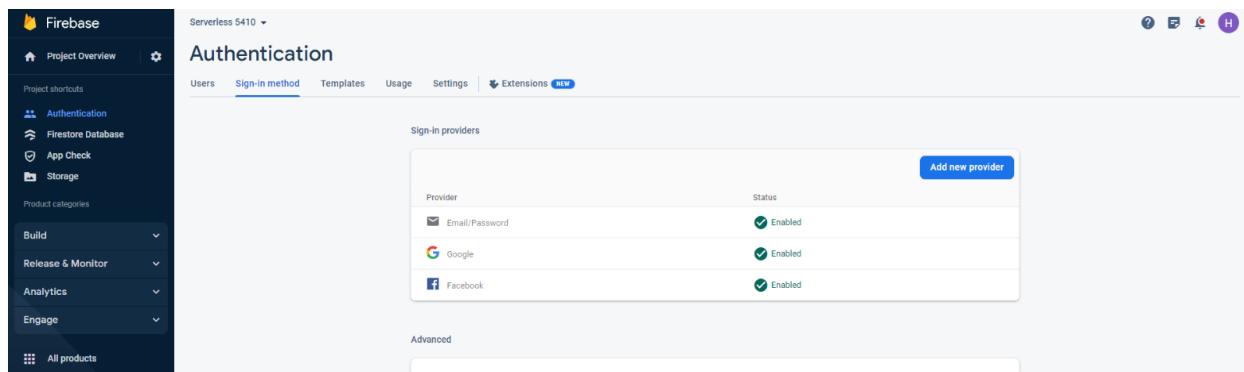


Figure 1: Firebase Authentication

User details, such as email, name, question answer and other user details, will be stored in AWS DynamoDB, which is a NoSQL database service. AWS Lambda, an event-driven serverless computing service, will handle storing user details to the database and generating and verifying cipher keys [3].

The screenshot shows the AWS Lambda console interface. At the top, there's a search bar labeled "Find APIs". To the right are buttons for "Actions" and "Create API". Below the header is a table with the following columns: Name, Description, ID, Protocol, Endpoint type, and Created. The table lists eight APIs:

Name	Description	ID	Protocol	Endpoint type	Created
check_security_question_dynamo		j8kaa89tm2	REST	Regional	2023-07-30
fetchQuesFirstNameLastNameByID		v8vw4gos0	REST	Regional	2023-07-31
gameCompletionUserDetailsUpdate		ca9bklt3k6	REST	Regional	2023-08-01
getAllUsers		akabqf470a	REST	Regional	2023-07-28
getQuestionsFromDynamo		30quej290j	REST	Regional	2023-07-31
increaseGamesPlayed		tedxdxrnrob	REST	Regional	2023-08-01
storeUserToDynamoDB		csxvr7woxf	REST	Regional	2023-07-31
UpdateUserDetails		4ipyn6vwh6	REST	Regional	2023-07-31

Figure 2: Lambdas

For the question-and-answer details, I wanted to use GCP Firestore, a NoSQL document database provided by Google Cloud Platform (GCP). However, I've decided to use AWS DynamoDB for storing the question-and-answer details as well, to maintain consistency within the AWS environment and reduce complexity [2].

The screenshot shows the AWS DynamoDB console for the "USERS" table. At the top, there's a button for "Autopreview" and a link to "View table details". Below the header is a message: "Completed. Read capacity units consumed: 2". The main area is titled "Items returned (16)". It contains a table with the following columns: firebase_user_id, answer1, answer2, answer3, email, first_name, games played, last_name, loss, question1, question2, and question3. The table lists 16 items, each with a unique ID and various user details.

firebase_user_id	answer1	answer2	answer3	email	first_name	games played	last_name	loss	question1	question2	question3
cZxx4MuuGKeQM...	<empty>	<empty>	<empty>	hexeho730...	hexeho7301	0	hexeho7301	0	What is you...	What is you...	What is you...
uOr6GBGTeufc3xFir...	friend	born	car	sharshil129...	Harshil	0	Shah	0	What is the ...	In what city...	What is the ...
o98s9yjTPhQQ84d...	<empty>	<empty>	<empty>	alt.xo-ce2s...	ALex	0	ROman	0	What is you...	What is you...	What is you...
630dZaZM2lcPiu6D...	pizza	viral	shah	harshil@yo...	harshil	0	shah	0	What is you...	What is the ...	What is you...
7mPJ8deAhJa9sDid...	<empty>	<empty>	<empty>	mansi2233...	Mansi	0	Patel	0	What is you...	What is you...	What is you...
5OFGn0lDHPyazBy...	a	d	s	mail4fb4@...	Harshil	0	Shah	0	What is the ...	What is the ...	What is the ...
Io869OQGt6frte1...	a	s	d	harshil.gro...	Harshil	0	Shah	0	What is you...	In what city...	What was y...
2af5karkNMOnPrR	<empty>	<empty>	<empty>	mansil@unn...	&n	<empty>	&n	<empty>	What is unus...	What is unus...	What is unus...

Figure 3: Users DynamoDB table

Finally, I've used GCP Buckets to store profile picture images and stored its URL to DynamoDB [4].

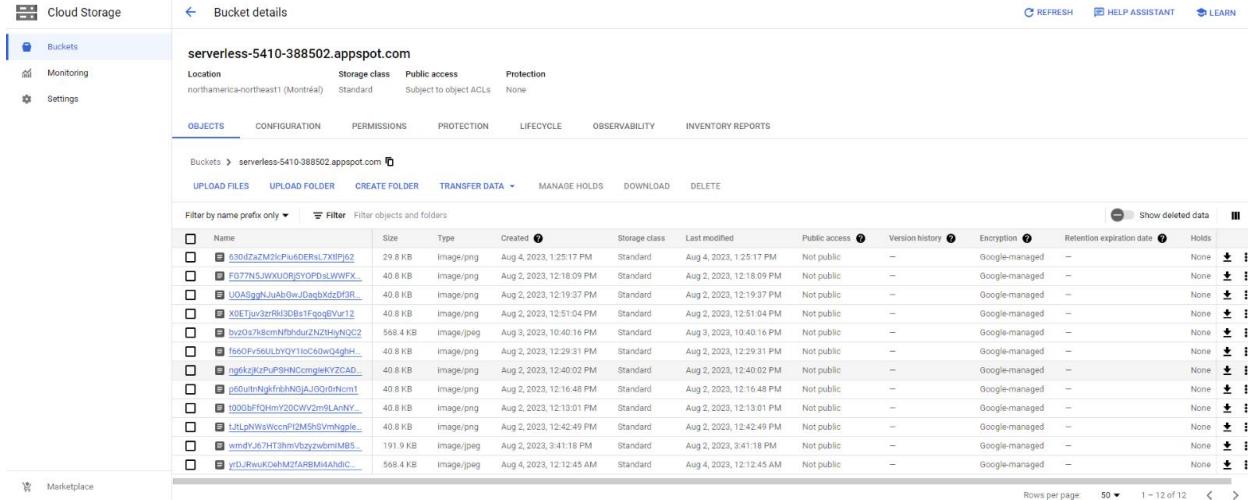


Figure 4: GCP bucket for user profile image storage

API Testing -

All API endpoints were tested using Postman with appropriate inputs of all possible

Figure 5: API testing via Postman

Requests		Tests	
<code>GET getAllUsers</code>	200 OK	<ul style="list-style-type: none"> > Response status code is 200 > Email is in a valid format > Response headers include 'Access-Control-Allow-Origin' header > Response time is less than 500ms 	PASSED
↳ Body			FAILED
<code>POST check_security_question_dynamo</code>	502 Bad Gateway	<ul style="list-style-type: none"> > Response status code is 502 > Response has the required field - message > Message is a non-empty string > Response time is less than 200ms > Request URL is valid and accessible 	PASSED
↳ Body			PASSED
↳ Body			FAILED
↳ Body			FAILED
<code>POST fetchQuesFirstNameLastNameBy...</code>	404 Not Found	<ul style="list-style-type: none"> > Response status code is 404 > Response body is null > Response time is less than 200ms 	PASSED
↳ Body			FAILED
↳ Body			FAILED
<code>GET getQuestionsFromDynamo</code>	200 OK	<ul style="list-style-type: none"> > Response status code is 200 > Response has the required fields > statusCode is a non-negative integer > Body is a non-empty string > headers.Access-Control-Allow-Origin is a string 	PASSED
↳ Body			PASSED
↳ Body			PASSED
PUT storeUserToDynamoDB	200 OK	<p>⚠ Test generation is supported only for JSON responses.</p>	
↳ Body			

Figure 6: API testing via Postman 2

Table 1: Functional Testing Result

Request	Type	Result
getAllUsers	GET	PASS
check_security_question_dynamo	POST	PASS
fetchQuesFirstNameLastNameByUserID	POST	PASS
getQuestionsFromDynamo	GET	PASS
storeUserToDynamoDB	PUT	PASS
UpdateUserDetails	PATCH	PASS
increase games played count	POST	PASS
games Completion status update	POST	PASS
get games details by userid	GET	PASS
KHUSHI_sendmailonregister	POST	PASS

mohit_get_user_details	GET	PASS
khushi_mohit_req	GET	PASS
pubsub	POST	PASS
getteamsByUserId	GET	PASS
removefromteam	DEL	PASS

Table 2: Planned vs Used services

	Planned Service	Used Service
Auth	Cognito	Firebase Authentication
Database	Firestore	DynamoDB
Compute	Cloud Function	Lambda

Functional Test

Table 3: Functional Testing

Test Case	Description	Expected	Results
1	User doesn't provide any credentials	System doesn't let users register	PASS

2	User provides an already registered email	System doesn't let users proceed	PASS
3	User enters correct registration details	User is taken to setting question form	PASS
4	User provides incorrect credentials during login	User is given invalid credentials error from backend	PASS
5	User provides correct email and password	System takes user to question verify pop up	PASS
6	User provides incorrect answer	System shows error to the end user	PASS
7	User provides correct answer	System takes user to cipher key verification	PASS
8	User provides incorrect cipher key	System shows error to the user	PASS

2. User Profile Management

How I planned it:

For our user interaction system, I have chosen to utilize GCP Firestore and Cloud Functions to build a scalable and efficient solution. To enable users to edit their personal information, I will create a Firestore collection to store user profiles. This collection will include fields for profile picture, name, contact information, and other relevant details. Additionally, Cloud Functions will be implemented to handle user profile updates securely, ensuring data integrity and authentication. For user statistics, such as games played, win/loss ratio, and total points earned, I plan to set up another Firestore collection dedicated to storing this data. Cloud Functions will be leveraged to calculate and update these statistics in real-time, ensuring users have access to up-to-date information. Furthermore, I will create a Firestore collection to

manage team affiliations, allowing users to join or leave teams seamlessly. Cloud Functions will play a vital role in handling these team-related interactions while enforcing necessary authorization checks. Lastly, to provide users with a way to view and compare achievements, I will design a Firestore collection to store achievement data. Cloud Functions will be responsible for retrieving and comparing achievements between users efficiently, offering an interactive and engaging experience.

How I built it:

During the implementation phase, I had to deviate from my original plan to use Firestore and cloud functions, as I have used DynamoDB in the User Authentication module, utilizing AWS DynamoDB and Lambda Functions to build our user interaction system. For editing personal information, I updated the DynamoDB table that I had set up in the first module that stores user profiles, which includes fields for profile picture, name, and security questions.

AWS Lambda Functions were developed to handle user profile updates securely, ensuring that only authorized users can modify their data via API gateway. To address user statistics, I created a DynamoDB table to store relevant metrics such as games played, win/loss ratio, and total points earned for each user. All predefined security questions are loaded from the following DynamoDB “questions” table at run time.

Items returned (10)	
<input type="checkbox"/>	question (String)
<input type="checkbox"/>	What is your favorite food?
<input type="checkbox"/>	What is the name of your first pet?
<input type="checkbox"/>	What is the name of your favorite childhood friend?
<input type="checkbox"/>	In what city were you born?
<input type="checkbox"/>	What was your childhood nickname?
<input type="checkbox"/>	What is the model of your first car?
<input type="checkbox"/>	What was the name of your first school?
<input type="checkbox"/>	What is your favorite movie or book?
<input type="checkbox"/>	What is your mother's maiden name?
<input type="checkbox"/>	What is your favorite sports team?

Figure 7: Questions DynamoDB table

AWS Lambda Functions were leveraged to perform calculations and update the statistics when games are completed or points are earned. For managing team affiliations, I designed a DynamoDB table for teams and another one to manage team affiliations for each user.

Functions (17)					
Last fetched 7 seconds ago					
<input type="button" value="Actions"/> <input type="button" value="Create function"/>					
Function name	Description	Package type	Runtime	Last modified	
CheckSecurityQuestions	-	Zip	Python 3.10	2 days ago	
fetchQuesFirstNameLastNameByUserID	-	Zip	Python 3.10	2 days ago	
storeUserToDynamoDB	-	Zip	Python 3.10	2 days ago	
getGameDetailsByUserID	-	Zip	Python 3.11	3 days ago	
gameCompletionUserDetailsUpdate	-	Zip	Python 3.11	3 days ago	
increaseGamesPlayed	-	Zip	Python 3.11	3 days ago	
UpdateUserDetails	-	Zip	Python 3.11	4 days ago	
getQuestionsFromDynamo	-	Zip	Python 3.10	4 days ago	
getAllUsersSDP	-	Zip	Python 3.10	4 days ago	

Figure 8: Lambdas

I called feature 3's API gateway to handle team leaving operations, ensuring data consistency and security. Lastly, to enable users to view and compare achievements, I stored the achievement data in DynamoDB. AWS Lambda Functions were responsible for fetching and comparing achievements between users efficiently, offering a seamless and rewarding experience to our users. With the AWS stack, our user interaction system is now scalable, robust, and well-integrated, providing an optimal solution to meet our requirements [4].

API Testing:

Used Postman to test all APIs whenever something is updated to ensure everything is working properly at all times.

Requests	Tests
PATCH UpdateUserDetails 200 OK ↳ Body	> Response status code is 200 PASSED > firebase_user_id is a non-empty string PASSED > First name is a non-empty string PASSED > last_name is a non-empty string PASSED
POST increase games played count 203 Non-Authoritative Information ↳ Body	⚠ Test generation is supported only for JSON responses.
POST games Completion status update 203 Non-Authoritative Information ↳ Body	⚠ Test generation is supported only for JSON responses.
GET get games details by userid 404 Not Found ↳ Body	> Response status code is 404 PASSED > Error message is present in the response PASSED > Error message is a non-empty string PASSED > Response time is less than 500ms PASSED > Response body is not empty PASSED
POST KHUSHI_sendmailonregister 200 OK ↳ Body	> Response status code is 200 PASSED > statusCode should be 0 FAILED > Body is an empty string FAILED > Response time is less than 500ms PASSED > The subscription was sent successfully FAILED

Figure 9: API Testing 1

<code>POST</code> KHUSHI_sendmailonregister	200 OK	<ul style="list-style-type: none"> ➤ Response status code is 200 ➤ statusCode should be 0 ➤ Body is an empty string ➤ Response time is less than 500ms ➤ The subscription was sent successfully 	PASSED FAILED FAILED PASSED FAILED
<code>GET</code> mohit_get_user_details	200 OK	<ul style="list-style-type: none"> ➤ Response status code is 200 ➤ Response is an array with at least one element ➤ startTime is a non-negative integer ➤ Score is a non-negative integer 	PASSED PASSED PASSED PASSED
<code>GET</code> khushi_mohit_req	200 OK	<ul style="list-style-type: none"> ➤ Response status code is 200 ➤ Response body is an array ➤ startTime is a positive integer ➤ Score is a non-negative integer 	PASSED PASSED PASSED PASSED
<code>POST</code> pubsub	200 OK	<ul style="list-style-type: none"> ➤ Response status code is 200 ➤ Response has the required field - data ➤ Data is a non-empty string ➤ Response time is less than 500ms ➤ Data is successfully created and published 	PASSED PASSED PASSED FAILED PASSED
<code>GET</code> getteamsByUserId	200 OK	<ul style="list-style-type: none"> ➤ Response status code is 200 	PASSED

Figure 10: API Testing 2

Table 4: Functional Test Case Results

Test Case	Description	Expected	Results
1	User doesn't provide proper first name	System doesn't let users update profile	PASS
2	User doesn't provide proper last name	System doesn't let users proceed	PASS

Table 5: Used Service VS Planned Services

	Planned Service	Used Service
Database	Firestore	DynamoDB
Compute	Cloud Function	Lambda

3. Team Management

The central functionality of this module is to be able to create team and select users to add the users to that team and on clicking invite, all the selected users will receive an email notification to accept the invitation to the team. Upon selecting accept, that user will be added to the team and if declined that user will not be added to the team.

After the user is added to the team, there is a table where the Your Teams and Your participation teams are displayed. Here multiple teams will be displayed for the logged in user. Now selecting a particular team, the user will be navigated to the dashboard of that team. Here the team leader and team members are displayed. Along with that if the logged in user is a normal participant, they will get an option to leave the team. Similarly, if the logged in user is a team creator, they can remove anyone from the team and promote someone to admin but can't leave the team. And if the logged in user admin, they can remove anyone and can leave the team. Moreover, there is an option to view different team statistics. The team statistics displays data regarding the total games played by the team and a user-specific bifurcation regarding the games played by each user. Moreover, the win -loss ratio for the user is displayed and the total score by the team is displayed.

Furthermore, there is an option to navigate to different pages like explore games and view leaderboards.

The frontend of this feature is built in ReactJS, and the backend is serverless. All the logic is written in lambda with python programming language.

Planning of the feature:

The planning of this feature can be broken down in several tasks.

Requirements gathering:

Understanding this feature was quite complex and tricky. First, I developed a proper understanding of the module.

Then I looked for the services provided by the cloud provider which will help me to fulfill the expectations of this module.

Database design schema:

For this module, I've used DynamoDB to store data.

Building of the feature:

The building of the feature can be broken in several tasks.

Create Team:

This create team feature is built with Lambda, API Gateway and DynamoDB.

The requirement states that the team's name should be created dynamically every time.

For this, I have used the OpenAI API, which will be called by the Lambda associated with the API invoked at the create team button click.

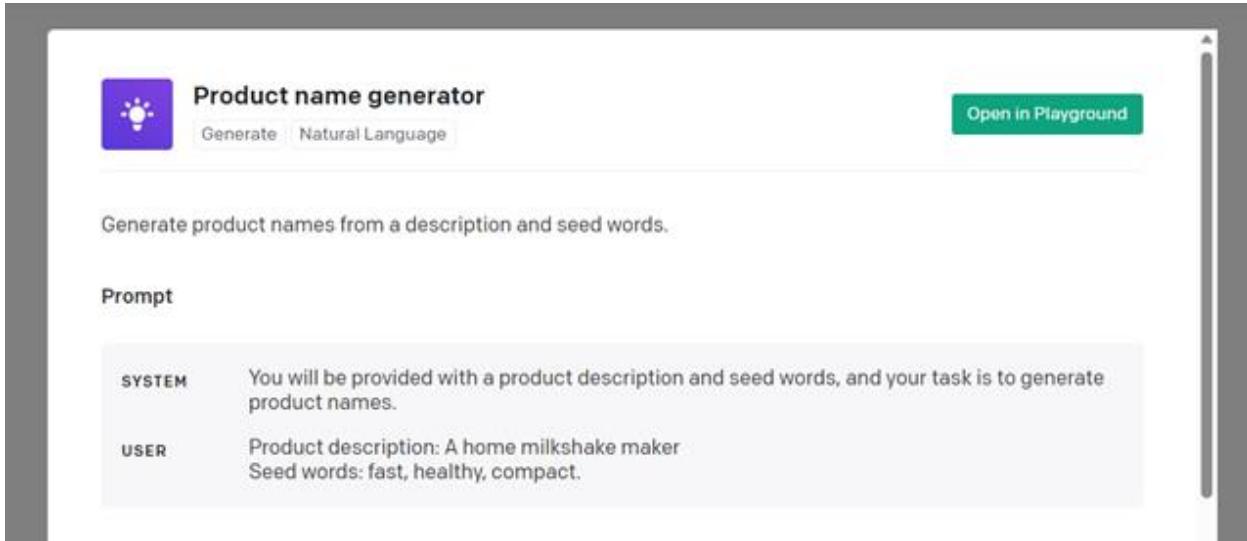


Figure 11: Tag Generation API

This API will always generate a unique one-word team name, and return the name, which will be shown in the dialog [5].

Later, the user will get the option to add the users to the team. Only the registered SNS users will be displayed in the dropdown and will be asked an email invite to accept/decline the team invite.

Items returned (22)									<input type="button" value="C"/>	<input type="button" value="Actions ▾"/>	<input type="button" value="Create item"/>
	invitedId (String)	declined	email	isAdmin	status	teamCreator	teamId	teamName	▲		
<input type="checkbox"/>	e71c0d8a-8854-4cbb-af95-9dc04522a882	false	khushi.shah...	false	true	true	e536e769...	Cougars			
<input type="checkbox"/>	1236bf4d-ca28-4d54-8c5c-1fdcd59c2654	false	mansi4455...	true	true	false	e536e769...	Cougars			
<input type="checkbox"/>	213d67b4-dc2a-4893-8e47-b59edae74cb2	false	mohitsojitr...	false	true	false	37b0fc5d...	Hustle			
<input type="checkbox"/>	82a8e822-72c0-46e4-9b63-104773799c91	false	harshil@yo...	false	true	true	37b0fc5d...	Hustle			
<input type="checkbox"/>	296cf16-8b5e-452c-a32f-fd123e8ce188	false	mohitsojitr...	false	true	false	95879cfb...	Nomads			
<input type="checkbox"/>	67ea1dc7-1afd-4c4a-b58e-8f0843344e5a	false	khushi.shah...	false	true	false	95879cfb...	Nomads			
<input type="checkbox"/>	b97c00fd-244f-4ac1-a983-c3e73c6d7ca5	false	hexeho730...	false	true	true	95879cfb...	Nomads			
<input type="checkbox"/>	5348f3f9-d336-4c2d-9674-fd3cd2013b64	false	mansi4455...	false	false	false	95879cfb...	Nomads			
<input type="checkbox"/>	f2c7e6aa-dacb-46da-9a95-7c7156f069a8	false	khushi.shah...	false	false	false	449d412b...	Rad			
<input type="checkbox"/>	ea7d79e1-874f-4ef2-bdc6-1a3ca8a42600	false	sharshil129...	false	true	true	449d412b...	Rad			
<input type="checkbox"/>	d2dc2fcf-7eed-48be-b176-e9eaff016718	false	wonekiy69...	false	false	false	449d412b...	Rad			

Figure 12: DynamoDB table – User Roles

teamName
Cougars
Cougars
Hustle
Hustle
Nomads
Nomads
Nomads
Nomads
Rad
Rad

Figure 13: Store Generated Team Names using AI API - DynamoDB table

In the below screenshot you will demonstrate the functional testing of this feature.

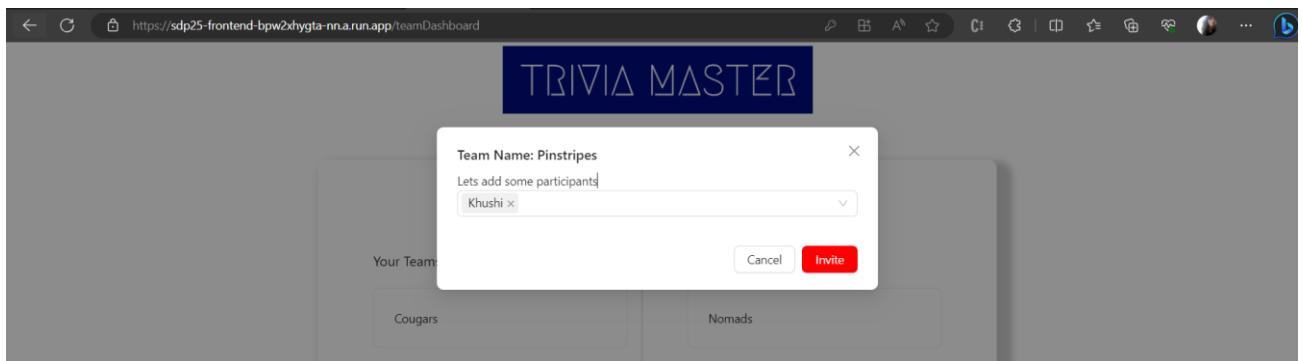


Figure 14: Functional Testing 1

In DynamoDB table teamNames and invite entry is added. And at the same time email will be given to the invitee [6].

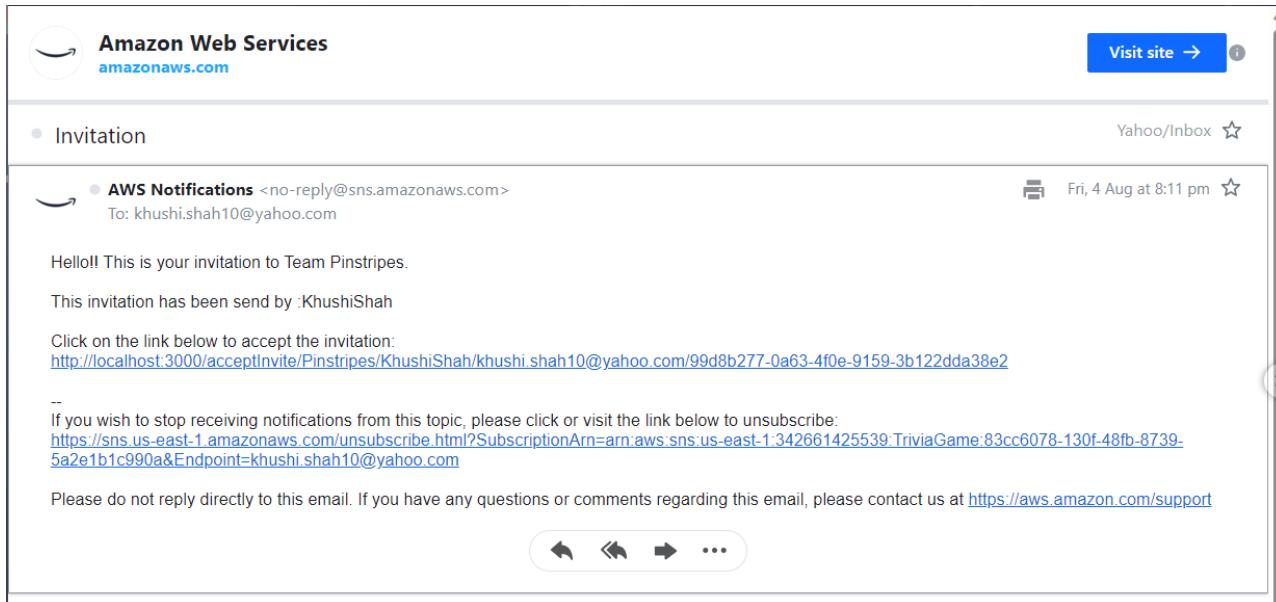


Figure 15: Invite entry mail on DynamoDB entry addition

Accepting/Declining Invitations:

To implement this feature, I've used Lambda, SNS, API gateway, DynamoDB.

Requested users will receive email with a link which will redirect to the accept/decline page.

There will be 2 buttons to accept or decline, and based on the users' choice the data will be updated in DynamoDB. If accepted then it will add them to the respective team.

Below are the steps where users will either accept or decline the team invite.

In the below screenshot you will demonstrate the functional testing of this feature [7].



Figure 16: Functional Testing 2 - Accept/Decline Invitation



Items returned (1/24)							<input type="button" value="C"/>	Actions ▾	<input type="button" value="Create item"/>	<input type="button" value="?"/>
	invited (String)	declined	email	isAdmin	status	teamCreator	teamId	teamName	▲	
<input checked="" type="checkbox"/>	99d8b277-0a63-4f0e-9159-3b122dda38e2	false	khushi.shah...	false	true	false	1ed6aef8-...	Pinstripes		

Figure 17: Functional Testing 3 - Invite Status

If the users decline then they won't be added to the team and here in the status it would be false and declined would be true.

Team Dashboard

To build the team dashboard, I used services like Lambda, API gateway and DynamoDB.

In this team dashboard, users will be able to see the teams they have created under "Your Teams" and the teams they have participated in "Your participations".

By clicking on the team names, they will be navigated to the team page [6].

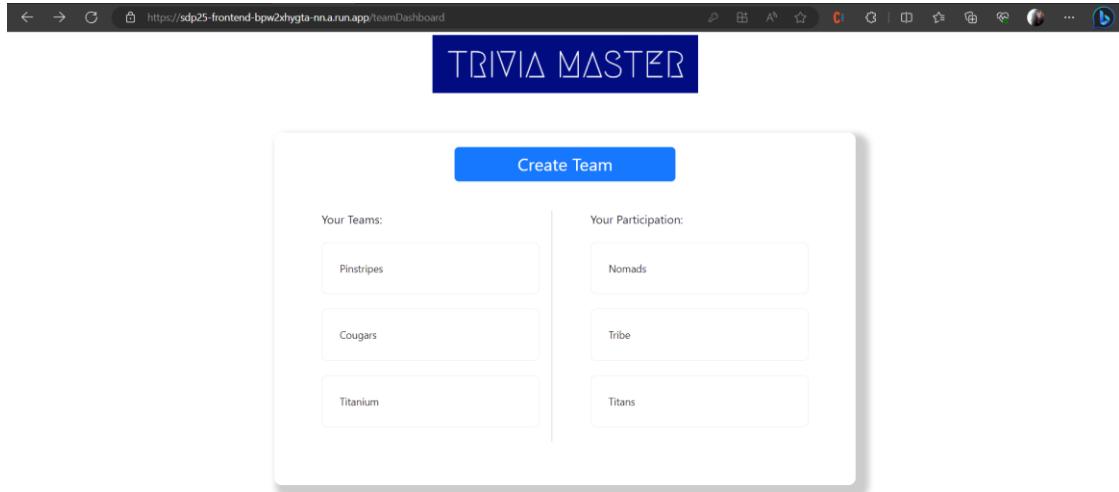


Figure 18: Team management

On the team dashboard, the team creator and other team participants details will be visible. And the team creator will get the option to promote the users to the admin or remove them. Also there is a view leaderboard option, which will store Team statistics like, games played, win-loss ratio and total points earned.

In the below screenshot you will demonstrate the functional testing of this feature.



Figure 19: Functional Testing for Team management 1

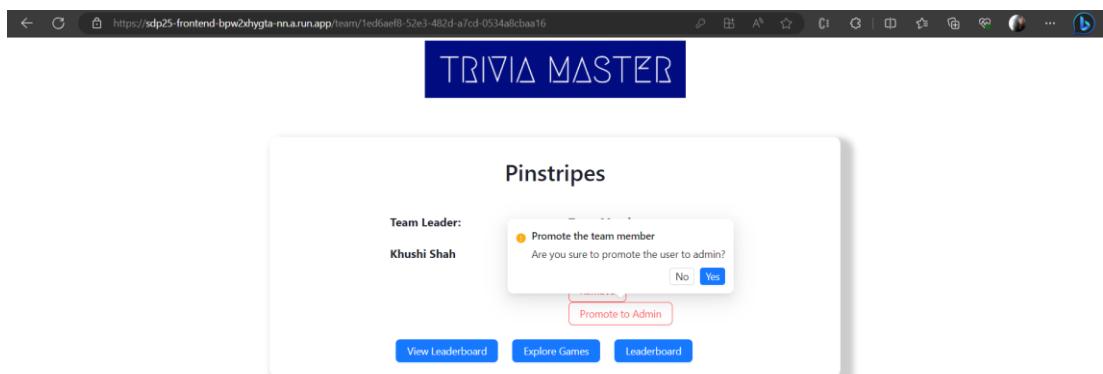


Figure 20: Functional Testing for Team management 2

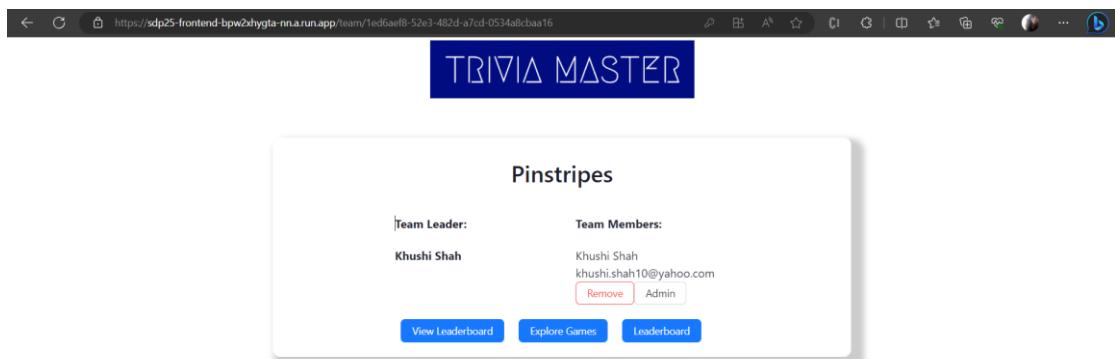


Figure 21: Functional Testing for Team management 3

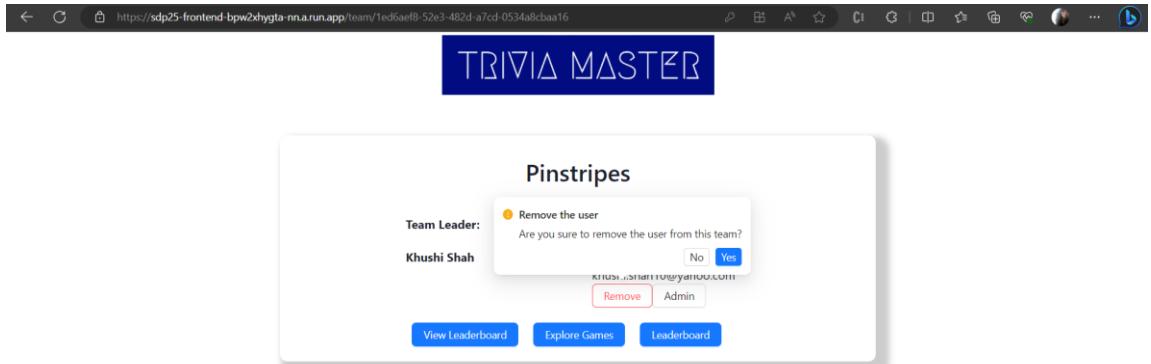


Figure 22: Functional Testing for Team management 4

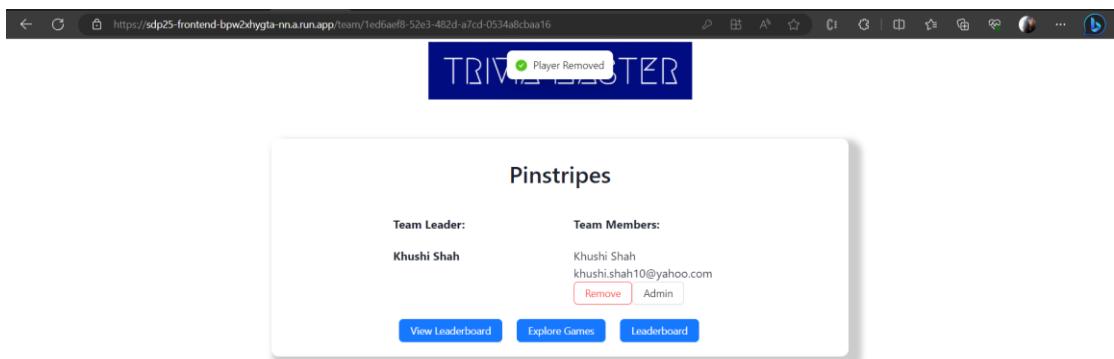


Figure 23: Functional Testing for Team management 5

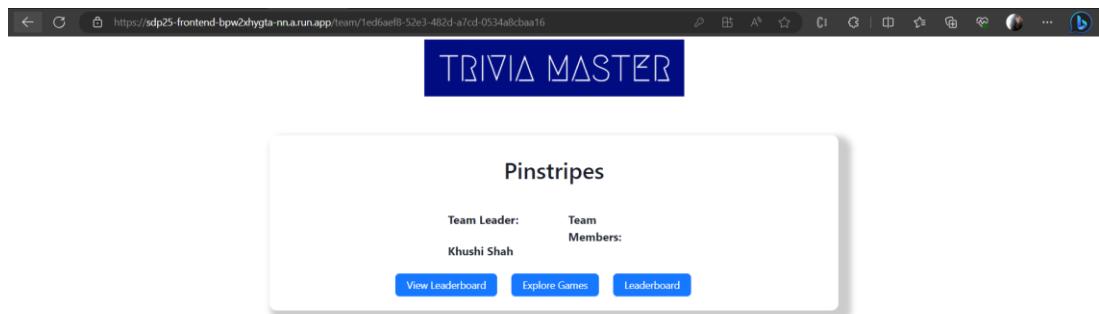


Figure 24: Functional Testing for Team management 6

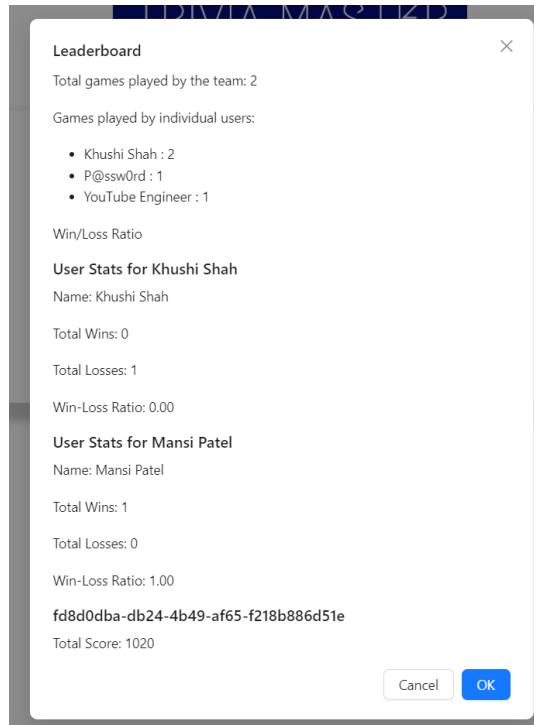


Figure 25: Functional Testing for Team management 7

API Testing (from postman):

Method	Endpoint	Status	Time
POST	/getTeamMembers	200 OK	352 ms 345 B
POST	/getYourTeams	200 OK	327 ms 345 B
POST	/giveLexData	200 OK	430 ms 546 B
POST	/inviteResponse	200 OK	56 ms 573 B
POST	/leaveremoveTeam	400 Bad Request	63 ms 738 B
POST	/promoteUser	200 OK	2109 ms 385 B

Figure 26: API testing 1

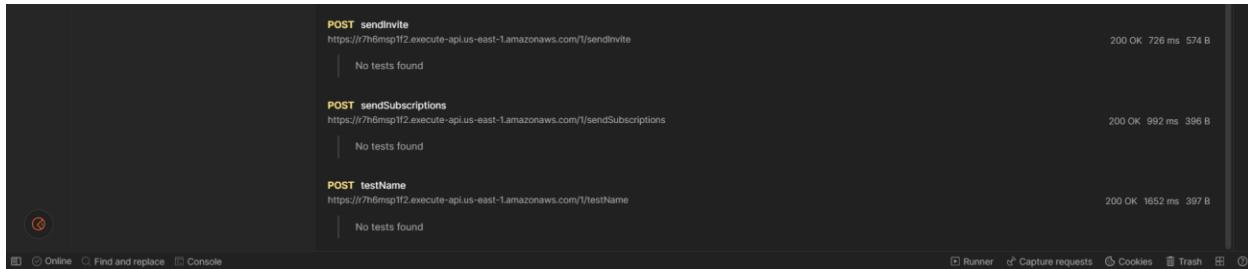


Figure 27: API testing 2

API used for this service:

Table 6: API endpoints

Method	Endpoint
post	/getTeamMembers
post	/getYourTeam
post	/inviteResponse
post	/leaveRemoveTeam
post	/promoteUser
post	/sendInvite
post	/sendSubscriptions
post	/testName

Table 7: Planned Vs Used Services

	Planned Service	Used Service
Database	DynamoDB	DynamoDB
Compute	Lambda, API Gateway	Lambda, API Gateway
Subscriptions	SNS	SNS

Functional Testing:

Table 8: Functional Testing Status

TestName	Expectations	Result	Status

Generate a new team name dynamically	The button click should generate new team names everytime	It creates new names	PASS
Display users to add on team	Should only display users that are confirmed	No other users are displayed except the one that are confirmed	PASS
Send emails	Send invitations to the added users	Email received	PASS
Accept/Reject Invite	User decides to accept/reject and accordingly DB must be updated and they must be added to the team	Users who accepted are added to the team, declined are not	PASS
View my teams and my participations	Only the logged in user's teams must be visible	Yes, only logged in users team are visible	PASS
Remove User	Admin and the team creator can remove the users	Users are removed from the team and DB	PASS
Promote to admin	Team creator can promote any user to the admin	Users are promoted to admin only by team creator	PASS
View Leaderboard	Display team stats for the opened team	Only statistics for the team are visible	PASS

4. Trivia Game Lobby

This feature contains Trivia Game Lobby with Category Filtering and Game Scheduling. The feature retrieves game data from a centralized datatable and presents it to players along with Name, Category, Difficulty level and time frame.

Fetching Game Data: The trivia game lobby will retrieve game information, including game title, category, time frame, and difficulty level, from the centralized datatable. This datatable will be handled by the admin portal to render information.

Category Filtering: Upon loading the trivia game lobby, players will have the option to filter the available games by category, difficulty level and time frame.

Email Notifications: Upon joining a game, players will receive an email notification that the game is scheduled to start in X minutes (X being a configurable value). The email will contain details such as the game title, scheduled start time,

Countdown Timer: Once a game's start time is scheduled, a countdown timer will be displayed in nextpage. The timer will indicate the time remaining until the game begins. Players can see the timer update in real-time.

Gather Questions: The questions which is stored with Category information in Question table will be called and gather data, With correct time stamp the start game will be triggered.

Choice of Services

During the initial stages of developing this feature, we carefully considered various options for building it. We performed a brief analysis to determine which cloud provider would be the most suitable. After evaluating Firestore, we realized that managing additional document-like structures for each game entry and question could be a challenge. Consequently, we opted to use DynamoDB as the database for this feature [8].

Once the database was selected, we found that combining DynamoDB with Cloud Functions might introduce portability and design issues. To avoid these potential complications, we made the decision to focus on implementing the game and question entry functionality within the AWS ecosystem instead of GCP. This choice allowed us to streamline development and ensure a smoother integration of the feature.

How I planned

I planned to do everything systematically, the steps include requirement analysis, database design, user interface design, technology stack selection [5], task breakdown, API design, game scheduling logic, email notification configuration.

How I did

After planning everything, I started working on following the steps. In between we had a meeting too, to validate the requirement of each module is being satisfied with another modules. I started with creating a lambda function to get the information from DynamoDB and process that data to make it useful for filtration. After completing his I created logical way to manage wait time and coordinated with team member working on in game experience module. Provided the needful data via lambda to start game [9].

To occupies this, started with designing the database structure and parameters

game_id	game_name	category	description	difficulty_level	participants	time_frame
fd8d0dba-db24-4b49-af65-f218b886...	General Knowledge Trivia	General Knowledge	This game includes various general knowledge questions across multiple categories.	Moderate	0	120
b2665a6b-658-4412-9787-854fbef...	Celebrity Trivia	Celebrity	This game includes questions about famous celebrities from various fields.	easy	0	90
fd8d0dba-db24-4b49-af65-f218b886...	Music Trivia	Music	This trivia game covers various genres of music and artists.	Moderate	0	90
fd8d0dba-db24-4b49-af65-f218b886...	Movie Trivia	Movie	This trivia game covers various movies and actors.	difficult	0	60
32644245-0039-4c0c-8e71-553b818...	History	History	This game includes various historical events and figures.	easy	0	90
32644245-0039-4c0c-8e71-553b818...	Science Trivia	Science	This game includes various science concepts and facts.	Hard	0	90
98ff9d96-15fe-416a-b7a2-d7921af2...	Geography Trivia	Geography	This game includes various geographical locations and facts.	Hard	5	60

Figure 28: Game Type DynamoDB Table

This DynamoDB data are being used to render the information on the exploreGame page. These attribute values are responsible for filtering the game values [9].

difficulty_level	Type	Condition	Value
Moderate	String	Equal to	Moderate

Items returned (2)
Moderate
Moderate

Figure 29: Filtered GameType DynamoDB table

I have created lambda functions to communicate between frontend and database [3].

The screenshot shows the AWS Lambda Functions page. The left sidebar includes links for Dashboard, Applications, Functions, Additional resources, and Related AWS resources. The main content area displays a table of functions with the following data:

Function name	Description	Package type	Runtime	Last modified
triviaGamePool	-	Zip	Node.js 18.x	22 days ago
GetGameDetails	-	Zip	Python 3.10	17 days ago
getJoinedGameDetails	-	Zip	Python 3.9	2 days ago
joinGame	-	Zip	Python 3.8	14 hours ago
CreateTriviaGame	-	Zip	Python 3.10	17 days ago

Figure 30: Lambda Functions

The module is designed in a way that It will access data from kinds of database mapping the game id, and render that according to database data [10].

The screenshot shows the Games Dashboard. It features two main sections: "General Knowledge Trivia" and "Celebrity Trivia".

General Knowledge Trivia

- This game implies question related to General Knowledge
- Difficulty :** Moderate
- Time Frame :** 120
- Category :** General Knowledge
- Participants :** 0
- Game ID :** fd8d0dba-db24-4b49-af65-f218b886d51e

Celebrity Trivia

- This game implies question related to Celebrity
- Difficulty :** easy
- Time Frame :** 90
- Category :** Celebrity
- Participants :** 0
- Game ID :** b2665a6b-a658-4412-9787-854fbeb1a275

Figure 31: Games Dashboard

Here the details are shown like

Name

Difficulty Level

Time Frame

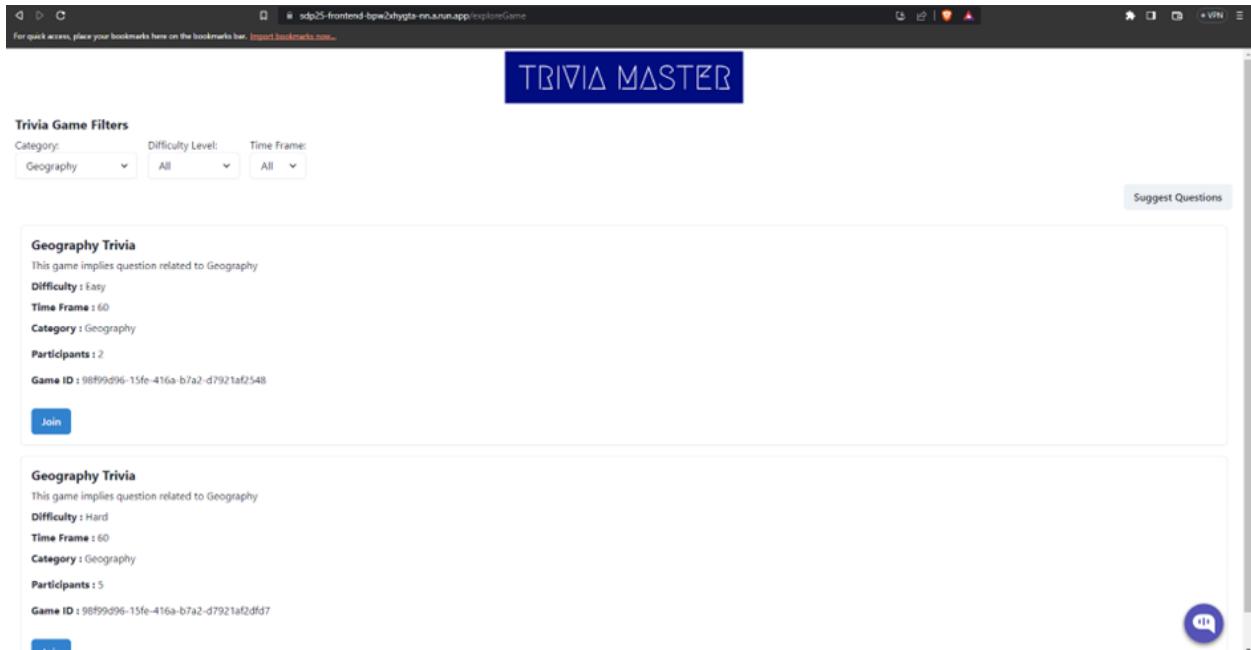
Category

Current Participants

Game Id

Here the games are being filtered by Category, Difficulty Level and Time Frame. This filter will be applied on AND policy, so if one Category is selected and one Difficulty level is selected than only entry will be shown which is satisfying both the conditions.

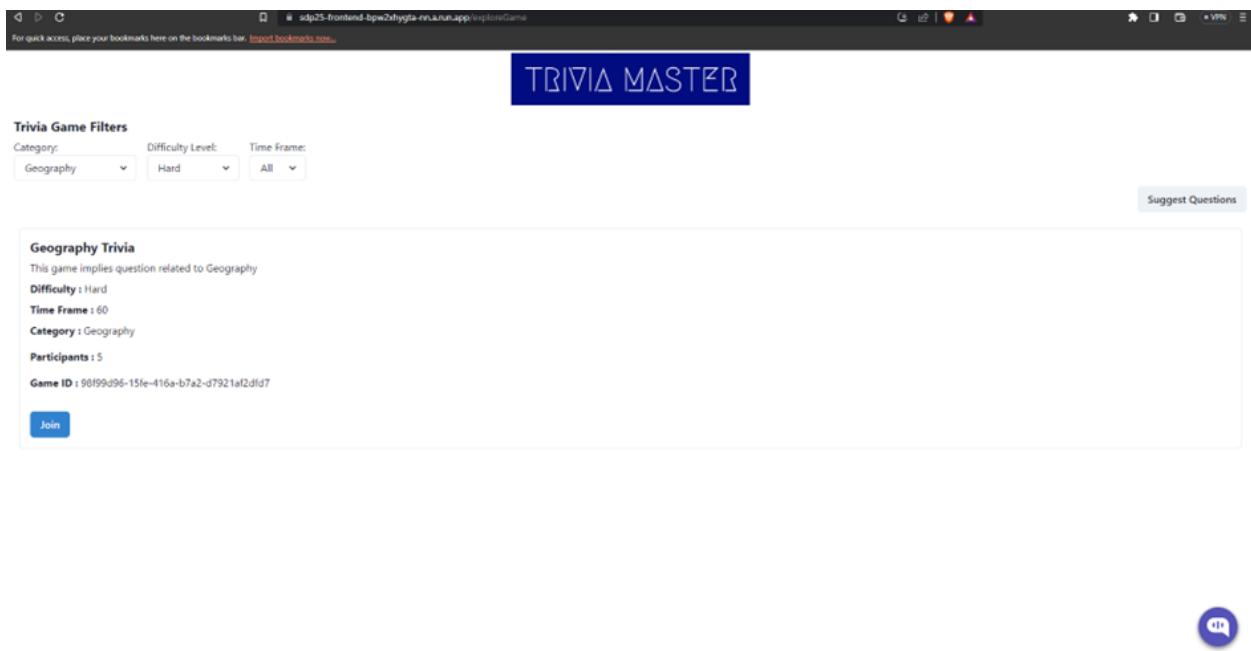
With 1 filter only



The screenshot shows the Trivia Master application interface. At the top, there is a header bar with the title "Trivia Master". Below the header, there is a section titled "Trivia Game Filters" with three dropdown menus: "Category" (set to "Geography"), "Difficulty Level" (set to "All"), and "Time Frame" (set to "All"). To the right of these filters is a "Suggest Questions" button. Below the filters, there is a card for a game titled "Geography Trivia". The card displays the following details:
- This game implies question related to Geography.
- Difficulty : Easy
- Time Frame : 60
- Category : Geography
- Participants : 2
- Game ID : 98f99d96-15fe-416a-b7a2-d7921af2548
A blue "Join" button is located at the bottom left of the card. On the right side of the card, there is a small circular icon with a speech bubble and a plus sign.

Figure 32: Filters - 1

With 2 filters



The screenshot shows the Trivia Master application interface. At the top, there is a header bar with the title "Trivia Master". Below the header, there is a section titled "Trivia Game Filters" with three dropdown menus: "Category" (set to "Geography"), "Difficulty Level" (set to "Hard"), and "Time Frame" (set to "All"). To the right of these filters is a "Suggest Questions" button. Below the filters, there is a card for a game titled "Geography Trivia". The card displays the following details:
- This game implies question related to Geography.
- Difficulty : Hard
- Time Frame : 60
- Category : Geography
- Participants : 5
- Game ID : 98f99d96-15fe-416a-b7a2-d7921af2dfdf7
A blue "Join" button is located at the bottom left of the card. On the right side of the card, there is a small circular icon with a speech bubble and a plus sign.

Figure 33: Filter - 2

Here the participants key is dynamic, which will be 0 at the time when game is just created. As the people will join the game, it will be updated by either number of participants waiting in the lobby or by number of participants playing the game.

Before anybody has joined the game

The screenshot shows a web browser window for the 'Trivia Master' application. The title bar reads 'sdp25-frontend-bpw2hygta-nn.a.run.app exploreGame'. The main content area is titled 'Trivia Master' with a blue header. Below it, there's a section for 'Trivia Game Filters' with dropdown menus for 'Category' (History), 'Difficulty Level' (All), and 'Time Frame' (All). A 'Suggest Questions' button is in the top right. The main content area displays a 'History' section with the following details:
This game implies question related to History
Difficulty : Easy
Time Frame : 90
Category : History
Participants : 0
Game ID : 32644245-0039-4c0c-8e71-553b81871rf3
A blue 'Join' button is at the bottom left of this section. The bottom right corner features a blue speech bubble icon with a white person icon inside.

Figure 34: Dynamic Participants shown

After I join the game, the number of participants will be changed

The screenshot shows the same web browser window for the 'Trivia Master' application. The title bar and filters remain the same. The main content area now shows:
Participants : 1
The 'Join' button is still present. The bottom right corner features a blue speech bubble icon with a white person icon inside.

Figure 35: Waiting Lobby

After clicking on join button, the user will be redirected to waiting lobby and there will be reverse timer.

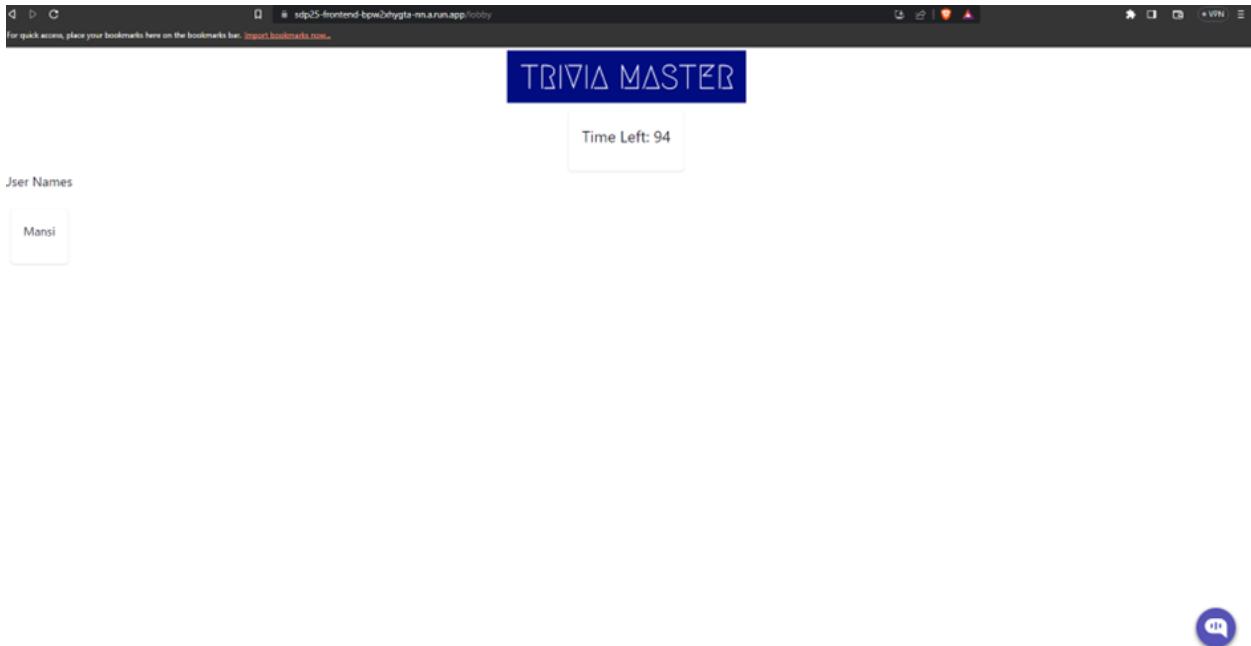


Figure 36: Waiting Lobby

When any other user will join the game, both the user and remaining time will be shown to user.

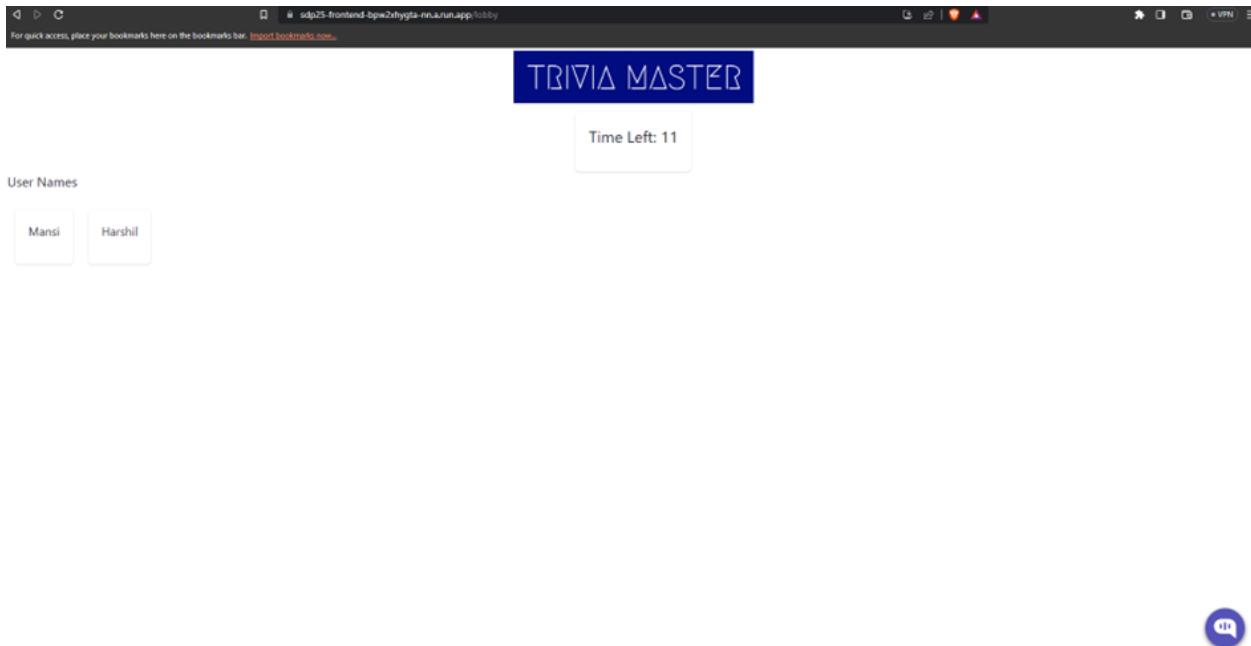


Figure 37: Time Left

By mean time I will gather data from API of Admin module, reformat that data and share it to In Game Experience module.

Planning

Table 9: Planned Service VS Used service

Planned	Used
AWS DynamoDb	AWS DynamoDB
AWS Lambda	AWS Lambda
AWS SNS	API Gateway

Tests performed

Table 10: Functional Testing

Test Name	Expectation	Pass
Filtering the data by Category.	It should filter the data properly	Pass
Testing two filters together	Work on AND condition	Pass
Testing the wait lobby functionality in a normal scenario.	It should show remaining time	Pass
Testing the wait lobby after refreshing the page.	It should start with continues time	Pass
Verifying the participant variable updates dynamically.	Another joined user should get same time	Pass

5. In-Game Experience

How I planned :

In the development of the game experience module, the requirement to answer multiple questions within specific time constraints, along with the need for real-time chat functionality, necessitated the adoption of a suitable two-way communication protocol. After careful consideration, the decision was made to employ the socket protocol. The evaluation process

involved weighing two alternatives: Firebase Realtime Database and API Gateway with WebSocket. Ultimately, the selection was made in favor of API Gateway with WebSocket API, particularly due to the integration of AWS DynamoDB for storing the questions.

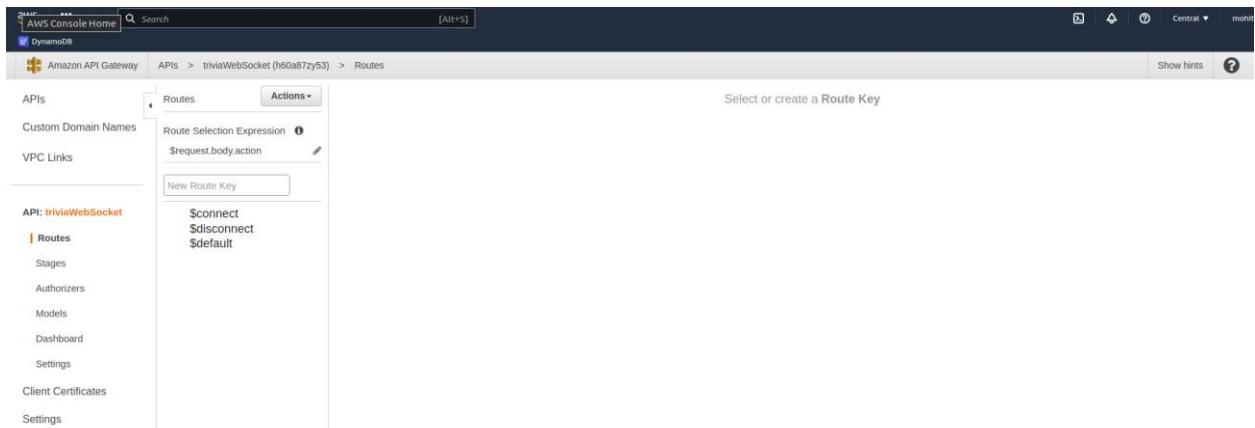
The chosen solution involved a combination of AWS services to achieve a cohesive and efficient system. AWS API Gateway with WebSocket functionality enabled real-time chat capabilities, allowing seamless communication between users. Moreover, the use of AWS DynamoDB ensured the effective storage and retrieval of questions. To address the handling of multiple questions with specific time periods, AWS SQS (Simple Queue Service) and Lambda functions were employed, facilitating a responsive and smooth user experience.

How I built it:

I used Serverless Application Model to build the entire infrastructure from the scratch. I created two api gateway to one for the websocket api and other for the rest api.

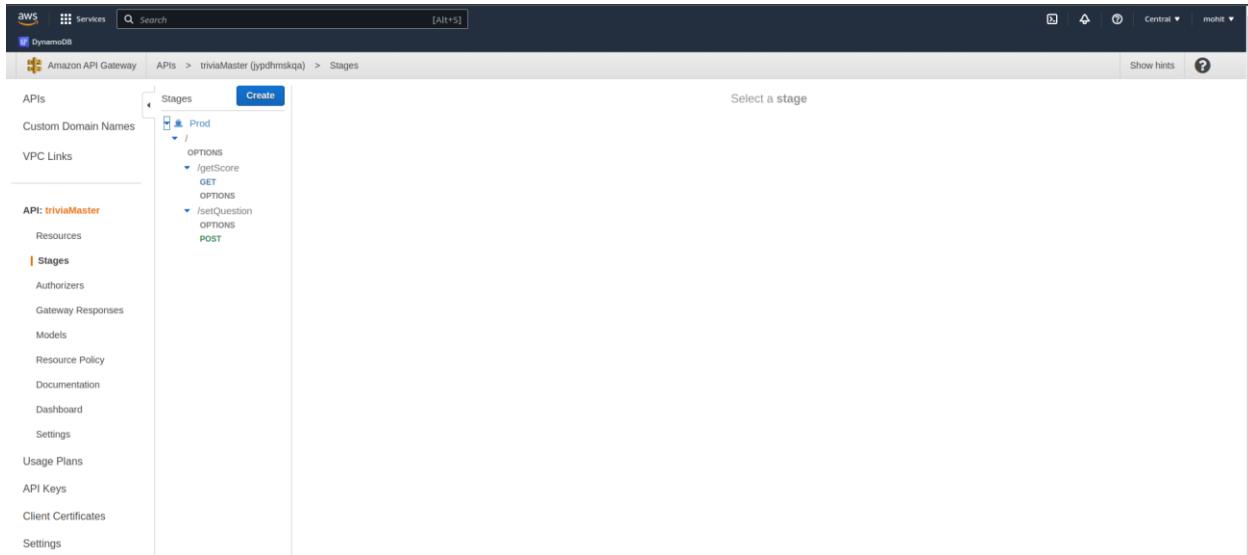
APIGateway (Websocket API)

Table 11: API Gateway



APIGateway (RestAPI)

Table 12: Lambda Functions



Lambda functions to handle the requests:

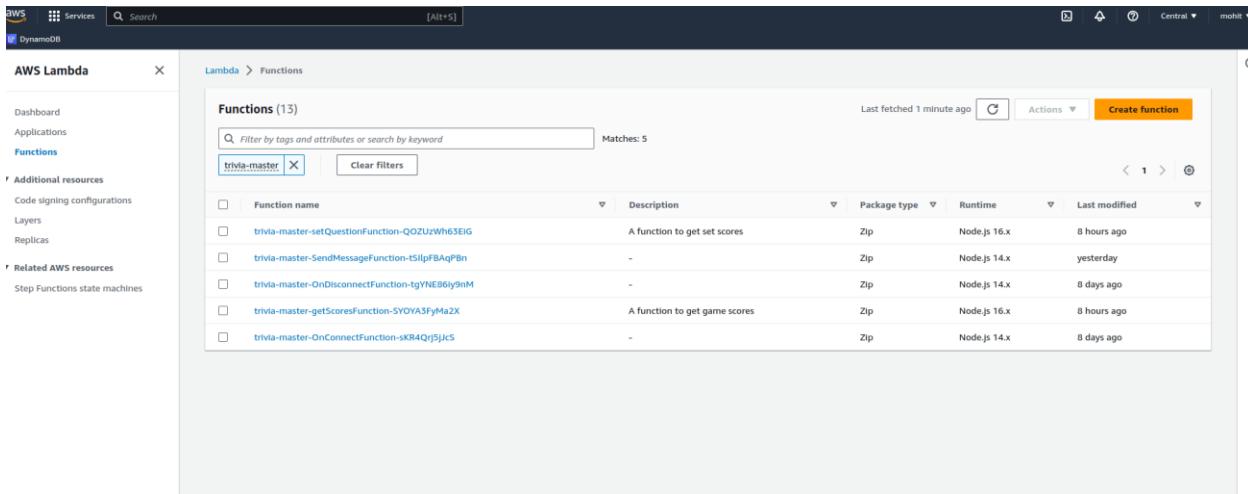


Figure 38: Lambda Functions

3.1 trivia-master-setQuestionFunction-QOZUzWh63EiG

The purpose of this Lambda function is to facilitate the creation and scheduling of games in a DynamoDB table while ensuring that each game is associated with a specific question. When invoked, the function takes relevant input data, such as the game details and the scheduled time, and then inserts this information into the DynamoDB table. Additionally, it associates a corresponding question with each game entry. By executing this function, games can be efficiently organized, and players will receive the appropriate question for the respective scheduled game at the designated time.

3.2 trivia-master-SendMessageFunction-tSIIpFBAqPBn

This Lambda function serves as the message handler for a Socket API, effectively managing various functionalities by breaking down the process into distinct components. It is responsible for handling incoming messages sent to the Socket API and is structured into separate segments to handle the functionality of sending messages, saving user answers, providing hints to users, and distributing questions among connected users within the game. This modular approach allows for efficient and organized management of real-time interactions and ensures smooth communication between users and the game system through the Socket API.

3.2 [trivia-master-OnDisconnectFunction-tgYN86iy9nM](#)

This is a straightforward Lambda function designed to handle disconnections from the Socket API. Its primary objective is to remove the userdata associated with a connected user from the connected user table. When a disconnection event occurs, the function is triggered, and it efficiently removes the relevant user data from the table, ensuring that the user's presence is no longer tracked within the Socket API system.

3.3 [trivia-master-OnConnectFunction-sKR4Qrj5jJcS](#)

Upon a user's connection to the socket, this Lambda function executes to efficiently store the user's information, including `user_id`, `connection_id`, and `team_id`, into a dedicated database table. The function is triggered whenever a new user establishes a connection, and it promptly captures and records the relevant user details into the table. By doing so, it ensures that crucial user information is accurately preserved and readily accessible for seamless real-time interactions within the Socket API system.

SQS Queue:

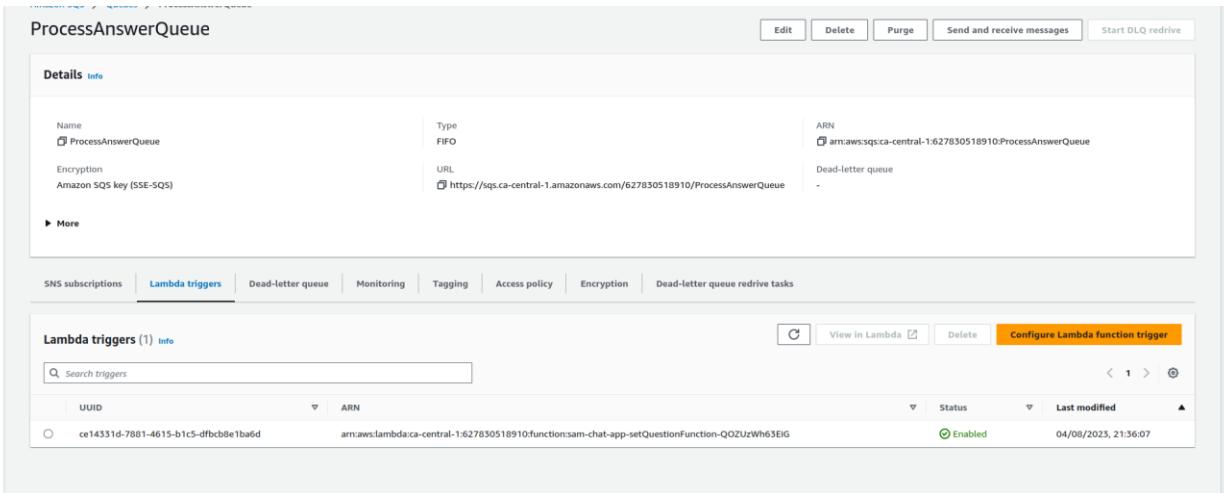


Figure 39: SQS Queue

This queue system serves as a mechanism to efficiently manage and prioritize answers submitted by users within a game or application. It adopts a First-In-First-Out (FIFO) approach, ensuring that the user who submits an answer first receives higher points for the correct response. As users submit their answers, the queue stores the question-answer ranks, preserving the order in which answers are received. This orderly arrangement allows for

accurate determination of the first responder, who is rewarded with higher points in accordance with their timely submission. By employing this queue system, the application can effectively handle and process user answers in a fair and equitable manner, promoting engagement and competition among users while maintaining a streamlined and organized flow of responses.

DynamoDB table:

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size	Table class
gameQuestions	Active	gameId (\$)	-	o <input type="radio"/> off	No	Provisioned (5)	Provisioned (5)	2.9 kilobytes	Standard
gameScores	Active	id (\$)	-	o <input type="radio"/> off	No	Provisioned (5)	Provisioned (5)	11.8 kilobytes	Standard
simplechat_connections	Active	connectionId (\$)	-	o <input type="radio"/> off	No	Provisioned (5)	Provisioned (5)	0 bytes	Standard

Figure 40: DynamoDB

The "gameQuestions," "gameScores," and "simplechat_application" tables serve as essential components of the application's data storage and management system. The "gameQuestions" table is designed to store the questions used in the game, enabling easy access and retrieval of questions during gameplay. Meanwhile, the "gameScores" table is responsible for storing user scores and related information, allowing for the tracking and updating of individual and team scores as the game progresses. Lastly, the "simplechat_application" table is dedicated to storing socket connection data, recording user details such as user_id, connection_id, and team_id upon their connection to the socket.

Testing:

Websocket: To monitor WebSocket connections and trace received messages, I utilized a Chrome extension called "WebSocket Testing." This extension facilitated the observation of message exchanges over the WebSocket protocol.

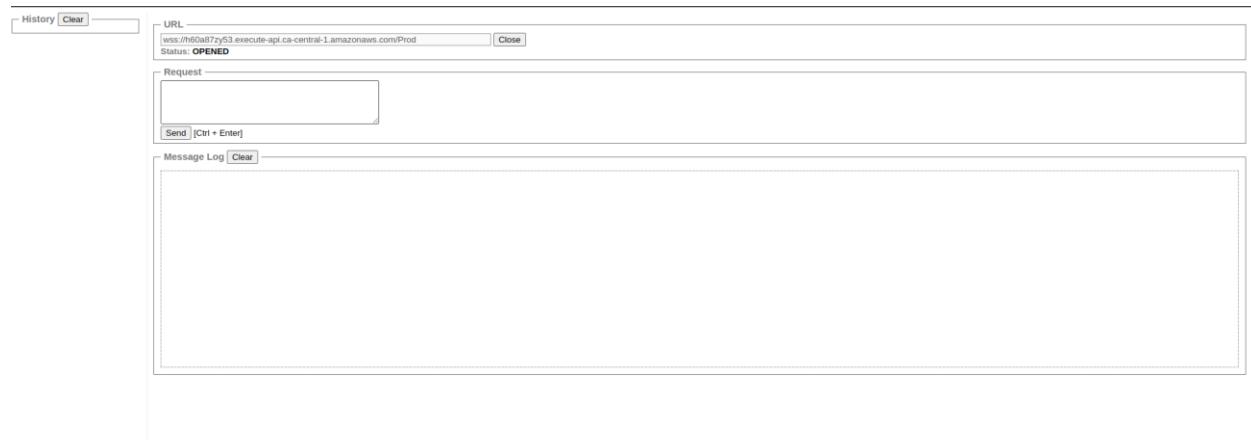


Figure 41: WebSocket Monitoring

Testcases:

Table 13: Test Case

Input	Expected Behavior
saveldeneity with User data JSON	Save user details on table
sendMessage with message	Publish the message to all team member
getQuestion with current timeframe and gameid	Fetch and send the question to that user

RestAPIS: To inspect and interact with RESTful APIs, I employed the widely-used tool Postman. With Postman, I was able to send HTTP requests (such as GET, POST, PUT, DELETE, etc.) to the API endpoints, examine the responses, and trace the data flow during the testing process.

The screenshot shows the Postman application interface. At the top, there are tabs for Home, Workspaces, and Explore, along with a search bar and a sign-in button. Below the header, a sidebar on the left lists a 'History' section with requests from 'Today' and 'August 3'. The main workspace displays a collection titled 'https://rv7nztjhc.execute-api.ca-central-1.amazonaws.com/Prod/getScore'. It contains a single GET request with the URL 'https://rv7nztjhc.execute-api.ca-central-1.amazonaws.com/Prod/getScore'. The 'Body' tab is selected, showing a JSON response with two data points. The first point has a 'start_time' of 1690140200697, a 'user_id' of 'user_5', a 'category' of 'Nature', a 'score' of 100, a 'team_id' of 'team_5', an 'id' of '2y12v9lvab31dddt8xozc', a 'game_id' of 'game_5', and a 'status' of 'win'. The second point has a 'start_time' of 1691349800697, a 'user_id' of 'user_10', a 'category' of 'Science', a 'score' of 820, a 'team_id' of 'team_3', an 'id' of '2nmao09pge7tvf2k41', a 'game_id' of 'game_8', and a 'status' of 'loss'. The bottom right corner of the interface shows a status bar with 'Status: 200 OK', 'Time: 1688 ms', and 'Size: 16.62 KB'.

Figure 42: Postman Testing

Testcases:

Table 14: Test case

Endpoint	Input	Expected Output
/getScore (GET)	user_id or team_id as QueryParams	Return the score array of that user or team
/setScore (POST)	JSON object or submitted answer, time	Return the Score

/setQuestions	JSON array of question for scheduled game	Return the success response
---------------	---	-----------------------------

All the test cases were run as per requirement

Cloud service planned vs Cloud Service Used:

Table 15: Planned Vs Used Services

Planned	Used
AWS DynamoDB	AWS DynamoDB
AWS Lambda	AWS Lambda
API Gateway	API Gateway
AWS SNS	AWS SQS

6. Leaderboards

How I planned :

In the context of implementing the leaderboard module, the requirement entails access to user details, team details, and individual user scores achieved per team. Additionally, the integration of this data into a Business Intelligence (BI) technology is essential. In this regard, two options were considered: Looker Studio and AWS QuickSight.

After careful evaluation, the decision was made to utilize AWS QuickSight for generating reports. This choice aligns well with the existing data infrastructure, as user details and scores are stored in an AWS DynamoDB table. To facilitate the data integration process, an AWS free tier account will be utilized, along with AWS Lambda for Extract, Transform, Load (ETL) operations from DynamoDB to AWS QuickSight. By leveraging AWS QuickSight's robust features, the goal is to create dynamic leadership graphs and insightful visualizations based on the accumulated data.

This approach ensures seamless integration of user and team data into AWS QuickSight, enabling the generation of comprehensive and meaningful reports. By effectively utilizing AWS Lambda for data ETL operations and leveraging the capabilities of AWS QuickSight, the leaderboard module will deliver valuable insights and actionable analytics to enhance the gaming experience.

How I built it:

aws-athena-query-results-ca-central-1-627830518910

The screenshot shows the AWS S3 Buckets page. On the left, there's a sidebar with links like 'Batch Operations', 'IAM Access Analyzer for S3', 'Block Public Access settings for this account', 'Storage Lens', 'Dashboards', 'AWS Organizations settings', and 'Feature spotlight'. The main area has a header 'BUCKETS (0) Info' with a note 'Buckets are containers for data stored in S3. Learn more'. Below is a search bar 'Find buckets by name'. A table lists four buckets:

Name	AWS Region	Access	Creation date
aws-athena-query-results-ca-central-1-627830518910	Canada (Central) ca-central-1	Bucket and objects not public	July 27, 2023, 20:04:41 (UTC-03:00)
aws-sam-cli-managed-default-samcliSourcebucket-193s94fbrytx	Canada (Central) ca-central-1	Bucket and objects not public	May 9, 2023, 23:58:38 (UTC-03:00)
tags-b00940478	Canada (Central) ca-central-1	Bucket and objects not public	July 16, 2023, 21:30:48 (UTC-03:00)
team-score940478	Canada (Central) ca-central-1	Bucket and objects not public	July 27, 2023, 19:59:04 (UTC-03:00)

Figure 43: Buckets

You need the aws-Athena-query-results S3 bucket for Amazon Athena because it serves as the default location where Athena stores the query results and query execution metadata. When you run a query in Athena, the results are typically large and can be stored in multiple files. Athena data source.

The screenshot shows the AWS Athena Data sources page. On the left, there's a sidebar with links like 'Query editor', 'Workgroups', 'Data sources' (which is selected), 'Jobs', 'Workflows', and 'Powered by Step Functions'. The main area shows a data source named 'TeamScoreConnector' under 'Data source details'. It includes fields for 'Data source name' (TeamScoreConnector), 'Data source type' (Data source connector), 'Lambda function' (athena_dynamo_dbconnector), and 'Lambda function ARN' (arn:aws:lambda:ca-central-1:627830518910:function:athena_dynamo_dbconnector). There are sections for 'Associated databases (1)' (with one database named 'default') and 'Tags (0) Info' (with a 'Manage tags' button).

Figure 44: AWS Athena

Amazon Athena is an intermediary data source for Amazon DynamoDB in Amazon QuickSight, offering advantages for data analytics and reporting. By transforming NoSQL data into a tabular format, Athena's SQL querying capabilities enable complex queries, joins, and aggregations. Its serverless architecture eliminates infrastructure management and costs for queries run. This integration allows users to create interactive dashboards, gain deeper insights, and make data-driven decisions using QuickSight's robust visualization and reporting features.

Connect Amazon QuickSight and Amazon Athena with Federated Query

Amazon QuickSight supports Federated Query, enabling cross-data source analysis between Amazon QuickSight and Amazon Athena. To connect, set up an AWS Glue Data Catalog, create IAM Policies, configure data sources, connect to AWS Glue Data Catalog, define query options, import data, and create visualizations and dashboards. This enables sophisticated insights and decision-making by combining data from multiple sources. For up-to-date information, consult the official AWS documentation.

Create QuickSight dataset of DynamoDB via Athena

In the Amazon QuickSight console, go to the "Manage Data" screen, and click on "New data set." Select "Athena" as the data source and connect it to your AWS Glue Data Catalog, which maps to your DynamoDB data. Additionally, you can connect other data sources, such as Amazon S3, Amazon Redshift, Amazon RDS, etc., if you want to combine data from multiple sources.

Graph's creation in Quicksight analysis

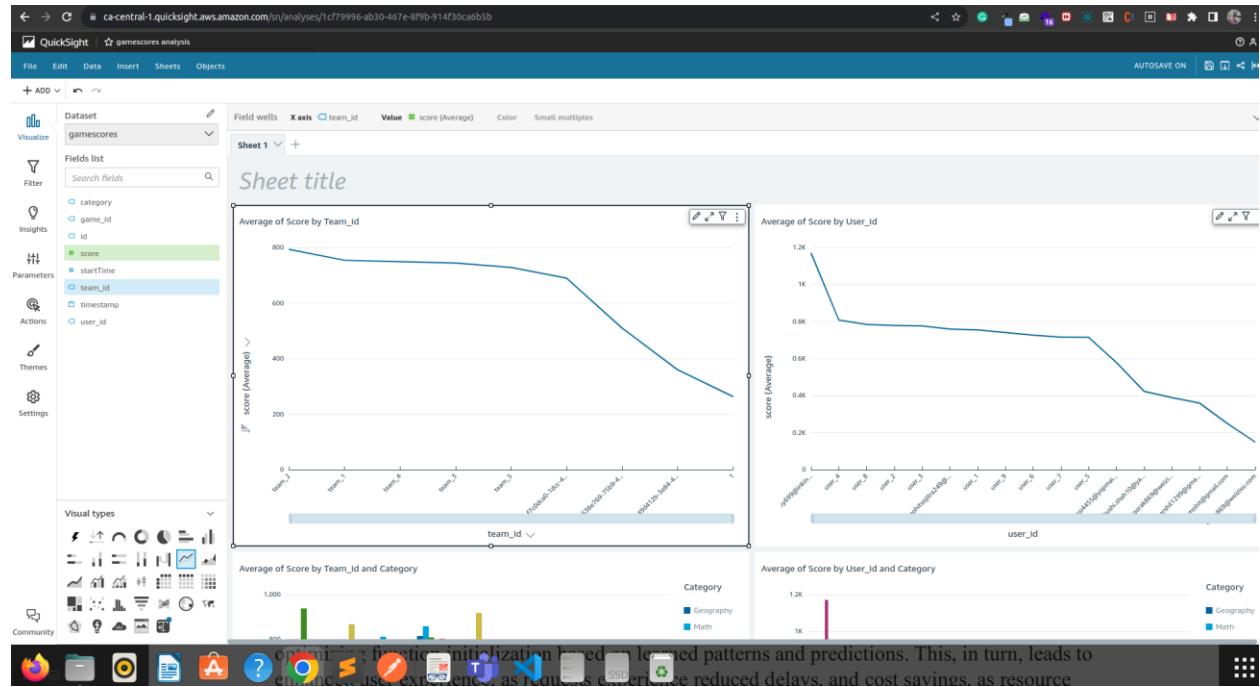


Figure 45: Quick Sight (Login Required)

By creating custom filters in Amazon QuickSight, We can efficiently filter leaderboards to display data based on different time frames such as weekly, monthly, and daily intervals. These filters enable dynamic data selection, as you can pass specific parameters through the dashboard URL to control the time range for the leaderboard. By configuring the custom filters, users can interactively choose our desired time frames, adjusting the displayed data accordingly without the need for manual query modifications.

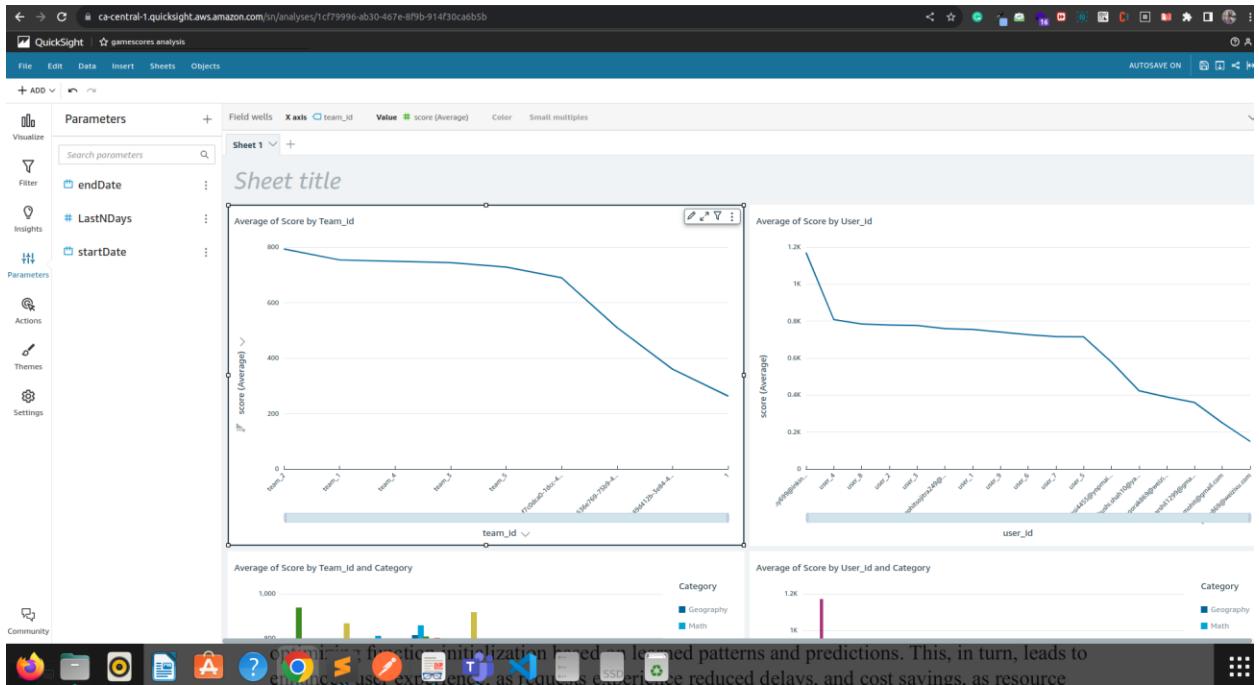


Figure 46: QuickSight 2

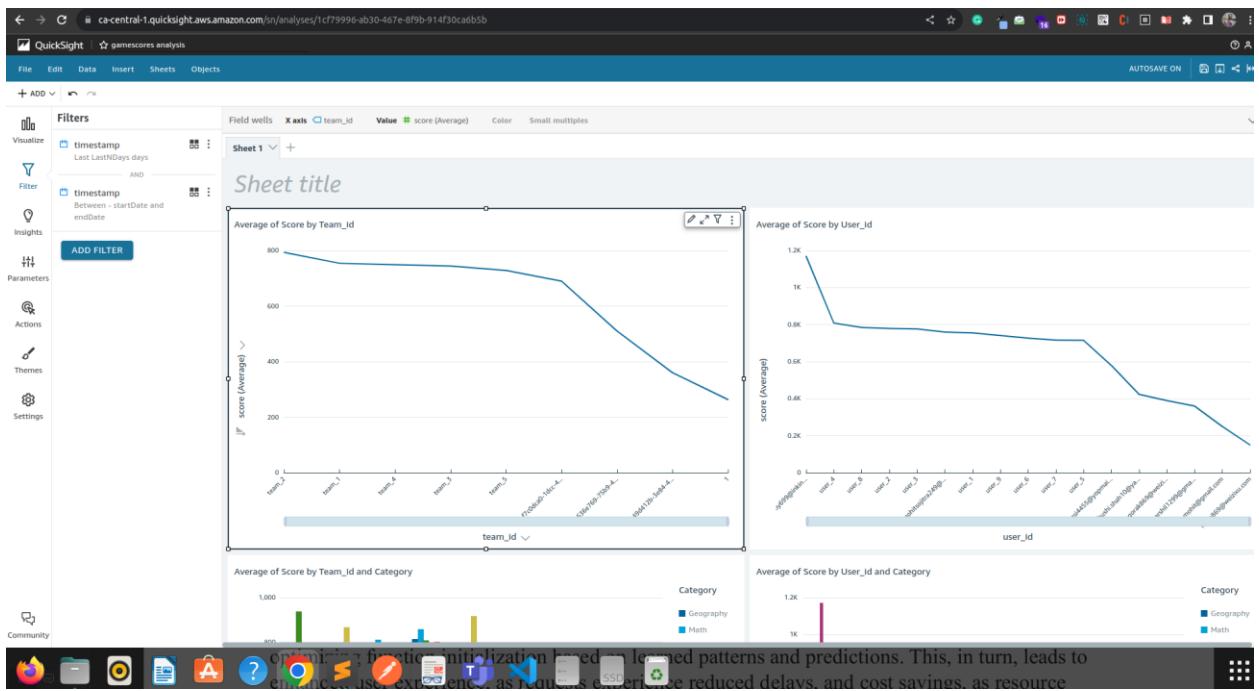


Figure 47: QuickSight 3

Additionally, Amazon QuickSight allows users to export the created dashboard and integrate it into their applications as an iframe source. This capability enables seamless embedding of QuickSight dashboards within external applications, offering a cohesive user experience. By exporting the dashboard as an iframe, developers can present data-driven visualizations

directly within their applications, enriching the user interface and providing valuable insights to end-users without requiring them to navigate to the QuickSight interface separately.

Testing:

To test the graphs and its authenticity I created a script which generate random score of 5 teams and 10 users for 10 games. So my script has generated 100 fake records which I store over dynamoDB table and then mapped that table to amazon quicksight.

```

for (let i = 0; i < 100; i++) {
  const user_id = Math.floor(Math.random() * 10) + 1;
  const game_id = Math.floor(Math.random() * 10) + 1;
  const team_id = Math.floor(Math.random() * 5) + 1;
  const score = Math.floor(Math.random() * 501) + 500; // Generates a random number between 500 and 1000 (inclusive)
  const startTime = gameIdStartTimeMapping[user_id-1];
  const category = gameCategoryMapping[user_id-1];
  const params = {
    TableName: "gameScores",
    Item: {
      id: {
        S: Math.random().toString(36).substring(2, 15) +
          Math.random().toString(36).substring(2, 15),
      },
      user_id: {
        S: `user_${user_id}`,
      },
      game_id: [
        {
          S: `game_${game_id}`,
        },
        team_id: {
          S: `team_${team_id}`,
        },
        score: {
          N: score.toString(),
        },
        startTime: {
          N: startTime.toString(),
        },
        category: {
          S: category,
        }
      ],
    };
  };
  randomData.push(params);
}
randomData.push(params);

```

Figure 48: Graph Testing Script

This test worked as expected and now I was able to see the graphs, However I don't have that dashboard now So I cannot share a snapshot over here.

Cloud service planned vs Cloud Service Used:

Table 16: Planned Feature VS Used Features

Planned	Used
Looker Studio	AWS Quicksight
AWS Firebase	AWS DynamoDB
-	AWS Athena
-	AWS S3

7. Trivia Content Management

Initial Plan

The initial plan for implementing the trivia contact management feature was to use a combination of AWS DynamoDB, Lambda Functions, SQS and SNS plus AWS QuickSight. The initial thinking was that keeping as many of our services on the same platform and within similar services common among features would simplify the development process. While this remained true for the core services, as development proceeded, the structure had to be adapted to circumvent the limitations of the LabRole account provided to us for access to AWS services. AWS QuickSight in particular caused a lot of problems for me because a provider-managed role as shown below:

QuickSight access to AWS services

By configuring access to AWS services, QuickSight can access the data in those services. Access by users and groups can be controlled through the options below.

IAM role in use

Quicksight-managed role (default)

Access granted to 1 services



Manage

Figure 49: QuickSight provider-managed role

QuickSight does not offer any out-of-the-box integration with DynamoDB, which I used consistently to integrate with all the other services in my architecture, with a preference being for relation databases as shown below:

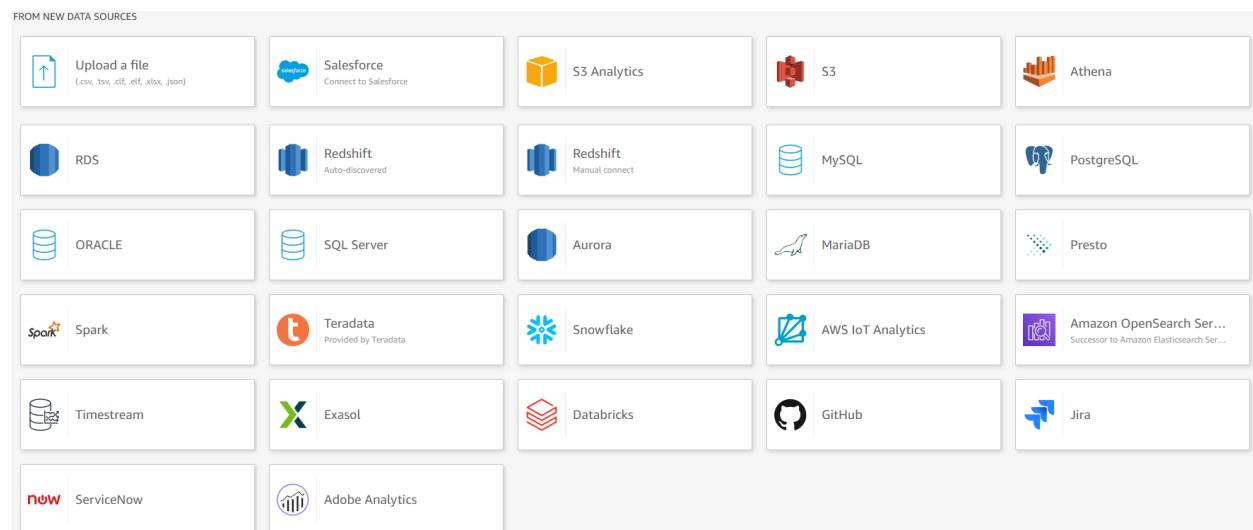


Figure 50: Services which QuickSight natively supports as data sources for analysis

However, although not as straight forward, it is possible to feed data from a DynamoDB Table into QuickSight for Analysis using an alternative method involving the ETL processes provided by AWS GLUE, Storage in S3 and Data preparation in AWS Athena as highlighted in this amazon article. I followed this method successfully and create a crawler for my DB and performed simple ETL with a spark script to extract my data and place it in Athena. Unfortunately, at the final step, Athena was not able to access schema I crawled with Glue had to be stored in s3, which by default cannot be accessed by QuickSight. Enabling this access is just a matter of toggling a selection option and hitting ok, but the lab role blocks the ability to modify permissions.

QuickSight access to AWS services

Make your existing AWS data and users available in QuickSight. [Learn more](#)

Allow access and autodiscovery for these resources

-  Amazon Redshift
-  Amazon RDS
-  IAM
-  Amazon S3
[Select S3 buckets](#)

-  Amazon Athena
Make sure you've chosen the right Amazon S3 buckets for QuickSight access
-  Amazon S3 Storage Analytics
-  AWS IoT Analytics
-  Amazon OpenSearch Service
-  Amazon SageMaker
-  Amazon Timestream
-  AWS SecretsManager
[Select secrets](#)

[Save](#)

[Cancel](#)

Figure 51: Default QuickSight Access and Management console

AWS Glue > Tables

Tables

A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Name	Database	Location	Classification	Deprecated	View data
trivia_games	5410-glue-db	arn:aws:dynamodb:us-east-1:5124841	dynamodb	-	Table data
trivia_games_of286b702437916b77f	5410-glue-db	s3://5410-project-bucket/trivia-game	JSON	-	

Figure 52: AWS Glue Data Catalogs

AWS Glue > Crawlers

Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Name	State	Schedule	Last run	Last run timestamp	Log	Table changes from last r...
5410-project-crawler	Ready		Succeeded	August 2, 2023 at 14:15:09	View log	-
5410-s3-crawler	Ready		Succeeded	August 2, 2023 at 22:26:39	View log	1 created

Figure 53: AWS Glue Crawlers (Dump to S3)

Amazon Athena > Data sources > AwsDataCatalog > 5410-glue-db

5410-glue-db

[Edit](#) [Delete](#)

Database details

Database name	5410-glue-db	Description	-
---------------	--------------	-------------	---

[Tables](#) [Views](#)

Associated tables (1)

Table name	Create time	Location
trivia_games_of286b702437916b77f9f1bc7a4aa694	2023-08-02T19:27:41.000-03:00	s3://5410-project-bucket/trivia-games/

Figure 54: Athena Catalog integrations (From Glue to S3 to Athena)

What I actually built:

I was able to complete the feature using a combination of an API Gateway, four lambda functions, four DynamoDB tables, two SNS Topics for segregating notifications for administrators and users, EventBridge for automated cloud triggers and a combination of the GoogleSheet API and LookerStudio to satisfy part 3 of feature 7. Below are screenshots of the provisioned resources used in the project.

APIs (1)							Actions	Create API
							Actions	Create API
	Name	Description	ID	Protocol	Endpoint type	Created		
<input type="radio"/>	5410-project-api		tunjietnw4	HTTP	Regional	2023-07-16		

Figure 55: HTTP API

Tables (4) Info								Actions	Delete	Create table
								Actions	Delete	Create table
	Name	Status	Partition key	Sort key	Indexes	Deletion protection				
<input type="checkbox"/>	trivia-admin-email-subscriptions	Active	email (S)	-	0	Off				
<input type="checkbox"/>	trivia-games	Active	gameId (S)	-	0	Off				
<input type="checkbox"/>	trivia-questions	Active	id (S)	-	0	Off				
<input type="checkbox"/>	trivia-user-email-subscriptions	Active	email (S)	-	0	Off				

Figure 56: DynamoDB Tables

Topics (4)				Edit	Delete	Publish message	Create topic
				Edit	Delete	Publish message	Create topic
	Name	Type	ARN				
<input type="radio"/>	halifax-taxi-notification	Standard	arn:aws:sns:us-east-1:5124841644...				
<input type="radio"/>	RedshiftSNS	Standard	arn:aws:sns:us-east-1:5124841644...				
<input type="radio"/>	trivia-admin-topic	Standard	arn:aws:sns:us-east-1:5124841644...				
<input type="radio"/>	trivia-user-topic	Standard	arn:aws:sns:us-east-1:5124841644...				

Figure 57: AWS SNS Topics

The screenshot shows the AWS Lambda console interface. At the top, there's a header with the function name "5410-analytics" and a "Layers" section indicating "(0)". Below the header, there's a "Triggers" section containing one entry: "EventBridge (CloudWatch Events)" with a sub-item "push-data". A button "+ Add trigger" is visible. To the right, there are several status indicators: "Last 4 hr" (Fun), "Add destination" (Fun), and a dropdown menu with options like "Des", "Last", "Fun", and "Fun". Below the triggers, a navigation bar has tabs: "Code", "Test", "Monitor", "Configuration" (which is selected and highlighted in blue), "Aliases", and "Versions". On the left, a sidebar lists "General configuration" and "Triggers" (which is also selected and highlighted in blue). The main content area displays the "Triggers (1)" list with the details of the "push-data" trigger.

Figure 58: Events Bridge recurring (4 hours) Trigger for Data-Source Lambda

	A	B	C	D	E	F	G	H
1	gameld	averageDuration	gameName	playerCount	playSessions	timePlayed		
2	66a519e6-b80f-4c61-9cab-0434a5f3d06b	21000	Mixed Science	7	2	42000		
3	ddf2ae33-a275-4834-b26d-9495797e0329	25000	Intermediate History	5	2	50000		
4	218de7c3-1ff8-40f8-9571-64dc34fe8a80	12833.33333	Intermediate Science	6	3	38500		
5								
6								

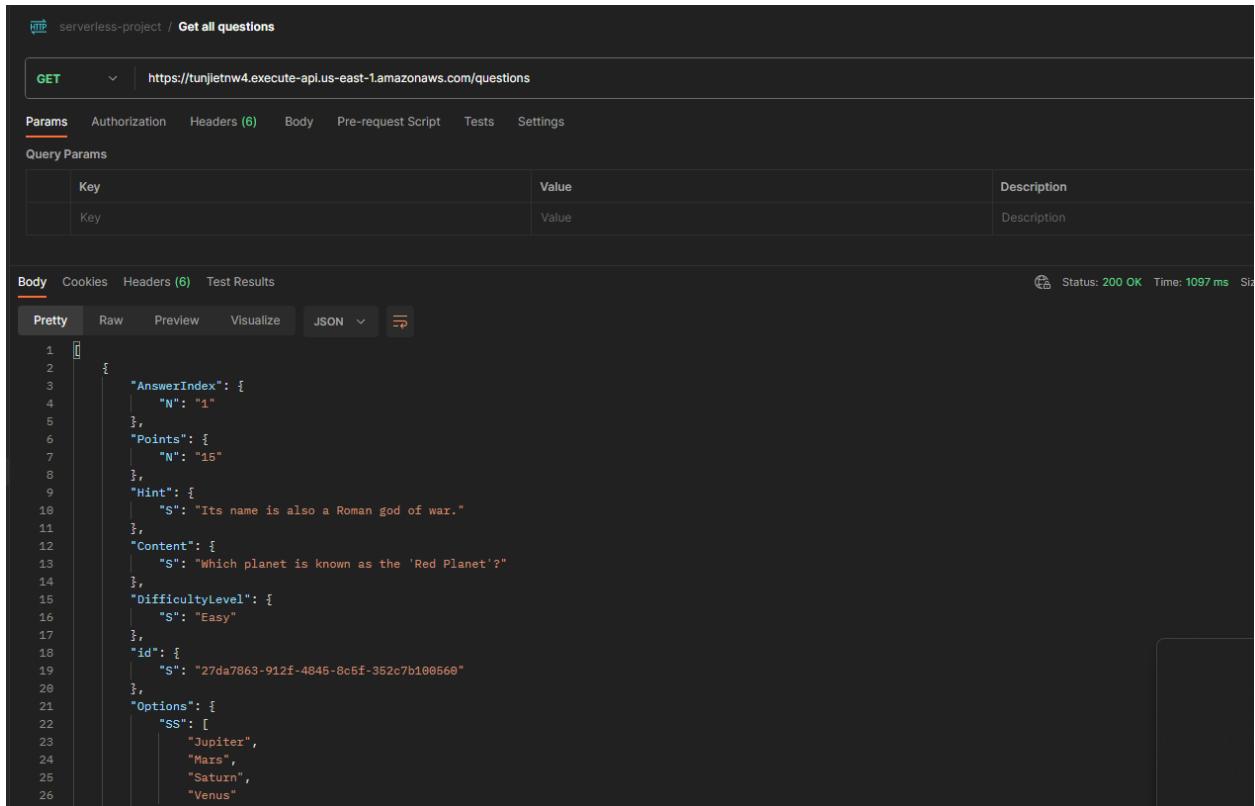
Figure 59: GoogleSheets API Datasource

The screenshot shows the Looker Studio interface. At the top, there's a header with "Looker Studio" and a search bar "Search Looker Studio". Below the header, there are tabs: "Recent", "Reports", "Data sources" (which is selected and highlighted in blue), and "Explorer". The main content area shows a list of data sources under the "Recent" tab. One item is listed: "gameplay-data - gameplay-data.csv", which is owned by "Edwin Adams" and was last opened by "me" at "12:26 PM". There are also other sections like "Shared with me", "Owned by me", and "Trash". At the bottom, there's a "Templates" section.

Figure 60: LookerStudio Data Source

Testing

Testing of the main functionality was done using API Postman calls to the endpoints I exposed on API Gateway and integrated with my Lambda Functions. Below are a few screenshots from API Tests.



The screenshot shows a Postman collection named "serverless-project" with a single test case titled "Get all questions". The request method is GET, and the URL is <https://tunjetnw4.execute-api.us-east-1.amazonaws.com/questions>. The "Params" tab is selected, showing a table with one row and columns for Key, Value, and Description. The "Body" tab is selected, showing the response body in Pretty JSON format. The response status is 200 OK, and the time taken is 1097 ms. The JSON response is as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
756
757
758
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
776
777
778
778
779
779
780
781
782
783
784
785
786
786
787
788
788
789
789
790
791
792
793
794
795
796
796
797
798
798
799
799
800
801
802
803
804
805
806
806
807
808
808
809
809
810
811
812
813
814
815
816
816
817
818
818
819
819
820
821
822
823
824
825
826
826
827
828
828
829
829
830
831
832
833
834
835
836
836
837
838
838
839
839
840
841
842
843
844
845
846
846
847
848
848
849
849
850
851
852
853
854
855
856
856
857
858
858
859
859
860
861
862
863
864
865
866
866
867
868
868
869
869
870
871
872
873
874
875
876
876
877
878
878
879
879
880
881
882
883
884
885
886
886
887
888
888
889
889
890
891
892
893
894
895
896
896
897
898
898
899
899
900
901
902
903
904
905
906
906
907
908
908
909
909
910
911
912
913
914
915
916
916
917
918
918
919
919
920
921
922
923
924
925
926
926
927
928
928
929
929
930
931
932
933
934
935
936
936
937
938
938
939
939
940
941
942
943
944
945
946
946
947
948
948
949
949
950
951
952
953
954
955
956
956
957
958
958
959
959
960
961
962
963
964
965
966
966
967
968
968
969
969
970
971
972
973
974
975
976
976
977
978
978
979
979
980
981
982
983
984
985
986
986
987
988
988
989
989
990
991
992
993
994
995
996
996
997
998
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1016
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1026
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1036
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1056
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1107
1107
1108
1108
1109
1110
1111
1112
1113
1114
1115
1115
1116
1117
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1125
1126
1127
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1135
1136
1137
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1145
1146
1147
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1155
1156
1157
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1165
1166
1167
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1197
1198
1198
1199
1200
1201
1202
1203
1204
1205
1205
1206
1207
1207
1208
1208
1209
1210
1211
1212
1213
1214
1215
1215
1216
1217
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1225
1226
1227
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1245
1246
1247
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1255
1256
1257
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1265
1266
1267
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1285
1286
1287
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1297
1298
1298
1299
1300
1301
1302
1303
1304
1305
1305
1306
1307
1307
1308
1308
1309
1310
1311
1312
1313
1314
1315
1315
1316
1317
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1325
1326
1327
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1335
1336
1337
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1345
1346
1347
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1355
1356
1357
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1365
1366
1367
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1375
1376
1377
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1385
1386
1387
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1397
1398
1398
1399
1400
1401
1402
1403
1404
1405
1405
1406
1407
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1415
1416
1417
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1425
1426
1427
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1435
1436
1437
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1445
1446
1447
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1455
1456
1457
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1465
1466
1467
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1475
1476
1477
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1485
1486
1487
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1497
1498
1498
1499
1500
1501
1502
1503
1504
1505
1505
1506
1507
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1515
1516
1517
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1525
1526
1527
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1535
1536
1537
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1545
1546
1547
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1555
1556
1557
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1565
1566
1567
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1575
1576
1577
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1585
1586
1587
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1597
1598
1598
1599
1600
1601
1602
1603
1604
1605
1605
1606
1607
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1615
1616
1617
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1625
1626
1627
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1635
1636
1637
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1645
1646
1647
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1655
1656
1657
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1665
1666
1667
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1675
1676
1677
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1685
1686
1687
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1705
1706
1707
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1715
1716
1717
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1725
1726
1727
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1735
1736
1737
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1745
1746
1747
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1755
1756
1757
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1765
1766
1767
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1775
1776
1777
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1785
1786
1787
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1805
1806
1807
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1815
1816
1817
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1825
1826
1827
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1835
1836
1837
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1845
1846
1847
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1855
1856
1857
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1865
1866
1867
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1875
1876
1877
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1885
1886
1887
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1905
1906
1907
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1915
1916
1917
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1925
1926
1927
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1935
1936
1937
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1945
1946
1947
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1955
1956
1957
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1965
1966
1967
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1975
1976
1977
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1985
1986
1987
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1997
1998
1998
1999
2000
2001
2002
2003
2004
2005
2005
2006
2007
2007
2008
2008
2009
2010
2011
2012
2013
2014
2015
2015
2016
2017
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2025
2026
2027
2027
2028
2028
2029
2030
2031
2032
2033
2034
2035
2035
2036
2037
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2045
204
```

The screenshot shows the Postman interface for a GET request to <https://tunjetnw4.execute-api.us-east-1.amazonaws.com/games>. The 'Body' tab is selected, showing the response body in JSON format:

```

1  [
2   {
3     "averageDuration": 21000,
4     "gameName": "Mixed Science",
5     "questions": [
6       {
7         "Options": [
8           "Respiration",
9           "Germination",
10          "Transpiration",
11          "Photosynthesis"
12        ],
13        "Points": 20,
14        "Category": "Science",
15        "Hint": "It involves sunlight.",
16        "Content": "What is the process by which plants prepare their food?"
17      }
18    ]
19  ]

```

Figure 62: GET all Games

The screenshot shows the Postman interface for a PATCH request to <https://tunjetnw4.execute-api.us-east-1.amazonaws.com/games/218de7c3-1ff8-40f8-9571-64dc34f...>. The 'Body' tab is selected, showing the request body in JSON format:

```

1  {
2   ...
3   "playerIds": ["p1234", "x2345", "p3456", "p8372", "MA342", "redhawk12"],
4   ...
5   "playTime": 9500
6 }

```

Figure 63: PATCH – Updates game data upon completion of a live game for data collection

8. Notifications and Alerts

Initial Plan

Initially I intended to complete this feature simple using SNS attached to a lambda function. However, during development, it became apparent that this would not be enough, because a user needs to subscribe to the notification list first, before they can receive notification messages. Another problem I faced was that even after a user already subscribed, my lambda function would send repeated emails requesting subscription rather than the actual email message.

Actual Implementation

To resolve the issues of recurring subscriptions, DynamoDB tables were implemented to store the lists of subscribed email addresses to the SNS Topic. What this allowed us to do then was to tie user subscriptions to the registration process, so that after that we could simply push notifications as needed. Similarly on the admin side, I set up monitoring alerts to be sent to my personal email whenever changes were made regarding trivia content management.

Testing

Below are some screenshots of email notifications I have been receiving as a result of API calls from the frontend application and postman to indicate that changes are being made:

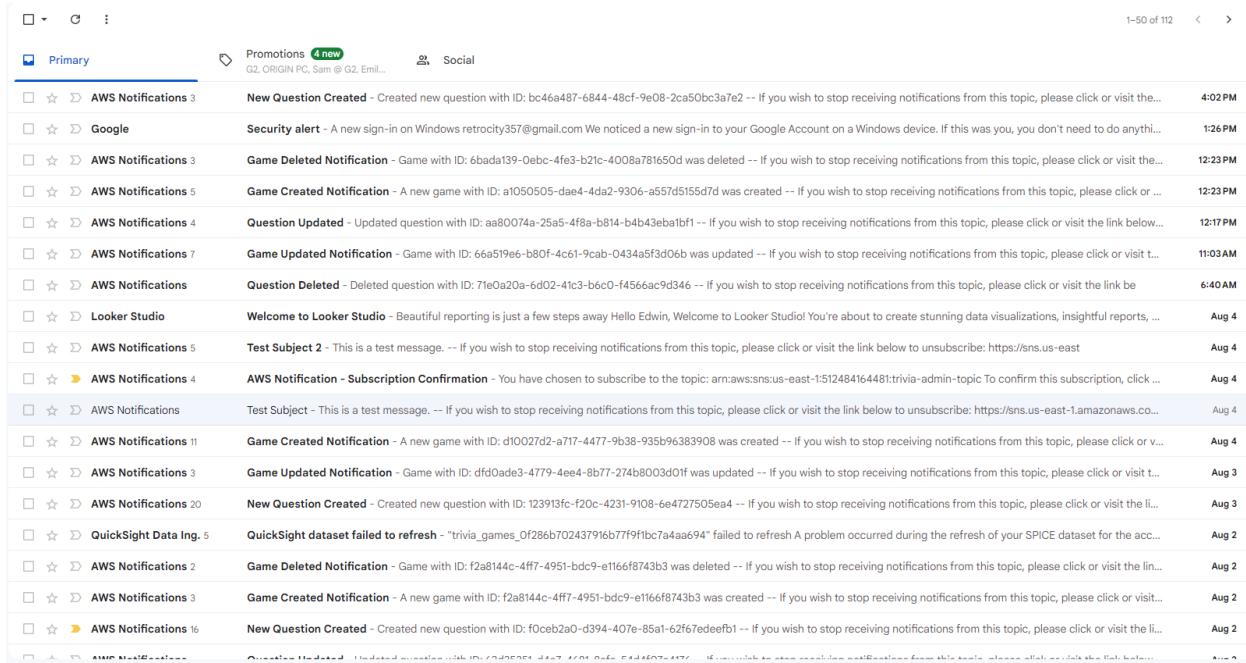


Figure 64: Admin Email Notifications

9. Automated Question Tagging

The Question Tagging Module is used to allow administrators to tag trivia questions with relevant categories. For now, the tagging option will be enabled for all the registered users as well as for admin. The user can suggest to us the question which can be added according to them in our system. The tagging mechanism will store the data given by the user in a database with its associated tags. The thought behind this is that, admin can choose and add questions to the question bank stored according to the tags associated with that.

Choice of Technology

My first choice for this task was Comprehend and Lambda. When I started browsing for the solution, I found that comprehend provides services like Sentiment Analysis, Entity Detection, Keyword Extraction and there was one service for tagging the document from S3 buckets. When I explored more about tagging of S3, it was more about classifying and not about providing actual suggestions. Also tried the approach of gathering the keywords from the comprehend API and then checking those words into a list of categories, but then this solution was not really good.

After this I moved to Natural Language API, and that is providing many services like Sentiment Analysis, Entity Detection, Syntax analysis, classifying content etc. After analyzing and testing I found Classifying Content API as best suited for our use case.

What I decided

For this task I decided to use the Comprehend API, so the plan was that there will be a lambda function which will be called by frontend with a question. That lambda will trigger AWS Comprehend service and find relevant words. I will maintain a list of words which can be associated with categories according to words suggested by comprehend. So we will store question along with tag in DynamoDB

What I did

After figured that this approach will not work, I transferred my focus from AWS to GCP for this task. NLP API of Google is working really well

I created a cloud function in python and tested with Postman API

The screenshot shows the Postman application interface. At the top, the URL is https://us-central1-serverless-project-390013.cloudfunctions.net/function-1. A POST request is selected. In the 'Body' tab, the raw JSON payload is:

```
1: { "question": "Which TV series features characters named Ross, Rachel, and Joey?" }
```

The 'Test Results' section shows a successful response with status 200 OK, time 104 ms, and size 365 B. The response body is displayed in JSON format:

```
1: { "responses": [ 2: { "category": "Arts & Entertainment", 3: { "subcategory": "TV & Video", 4: { "subcategory": "TV Shows & Programs", 5: { "subcategory": "Arts & Entertainment", 6: { "subcategory": "Fun & Trivia", 7: { "subcategory": "Fandom & Silly Surveys", 8: { "subcategory": "Arts & Entertainment", 9: { "subcategory": "TV & Video", 10: { "subcategory": "TV Guides & Reference" } } } } } } } ] }
```

Figure 65: Postman Testing 1

The screenshot shows the Postman application interface. At the top, the URL is http://ec2-100-27-30-89.compute-1.amazonaws.com/question-tagging. A POST request is selected. In the 'Body' tab, the raw JSON payload is:

```
1: { "question": "What is the largest planet in our solar system?" }
```

The 'Test Results' section shows a successful response with status 200 OK, time 335 ms, and size 202 B. The response body is displayed in JSON format:

```
1: { "responses": [ 2: { "category": "Science", 3: { "subcategory": "Astronomy" } } ] }
```

Figure 66: Postman Testing 2

The screenshot shows the Google Cloud Firestore interface. The left sidebar includes sections for Database (Data, Indexes, Import/Export, Disaster recovery, Time-to-live (TTL), Security Rules), Insights (Usage, Key Visualizer), and Release Notes. The main area displays a document in the 'question-tagging' collection. The document structure is as follows:

```

{
  "question": "Who played the character of Jack Dawson in the movie Titanic?",
  "tag": [
    "Arts & Entertainment",
    "Movies",
    "Movie Reference",
    "Arts & Entertainment",
    "Movies",
    "Drama Films",
    "Arts & Entertainment",
    "Movies",
    "Classic Films",
    "Arts & Entertainment",
    "Movies",
    "Romance Films",
    "Arts & Entertainment",
    "Film & Trivia"
  ]
}

```

Figure 67: GCP Firestore table – questions-tagging

This screenshot shows another document in the 'question-tagging' collection. The document structure is identical to the one in Figure 67, containing a question and a list of tags. The tags listed are:

```

{
  "question": "What is the chemical symbol for water?",
  "tag": [
    "Science",
    "Chemistry",
    "Reference",
    "General Reference",
    "Educational Resources"
  ]
}

```

Figure 68: Questions tagging

Google Cloud		serverless-project	cloud	X	Search	1	?	m
(...)	Cloud Functions	Functions	CREATE FUNCTION	REFRESH		LEARN	RELEASE NOTES	
Filter Filter functions								
	Environment	Name ↑	Last deployed	Region	Recommendation	Trigger	Runtime	Memory allocated
<input type="checkbox"/>	2nd gen	function_1	Aug 5, 2023, 11:33:56 AM	us-central1		HTTP	Python 3.8	256 MB
<input type="checkbox"/>	2nd gen	function_2	Aug 5, 2023, 10:04:16 AM	us-central1		HTTP	Python 3.8	256 MB
								Executed function
								Actions

Figure 69: Cloud Functions

The screenshot shows the Trivia Master game lobby. At the top, there's a header bar with a bookmark link and a 'SUGGEST QUESTIONS' button. Below the header is a dark blue banner with the 'TRIVIA MASTER' logo.

Trivia Game Filters:

- Category: All
- Difficulty Level: All
- Time Frame: All

General Knowledge Trivia:

- This game implies question related to General Knowledge.
- Difficulty : Moderate
- Time Frame : 120
- Category : General Knowledge
- Participants : 0
- Game ID : fd8d0dba-db24-4b49-af65-f218b886d51e

Celebrity Trivia:

- This game implies question related to Celebrity.
- Difficulty : Easy
- Time Frame : 90
- Category : Celebrity
- Participants : 0
- Game ID : b2065a6b-a650-4412-9787-854fbcb1a275

Figure 70: Game lobby

TRIVIA MASTER

Question Taging

Who co-founded Microsoft with Bill Gates?

Generate Tags

Tags generated

Computers & Electronics Software Operating Systems News Business News



Figure 71: Question tags generation from frontend

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now.](#)

TRIVIA MASTER

Question Taging

What type of hat is traditionally associated with Sherlock Holmes?

Generate Tags

Tags generated

Arts & Entertainment Fun & Trivia Fun Tests & Silly Surveys Books & Literature Literary Classics



Figure 72: Question tagging 2

Planning

Table 17: Planned VS Used

Planned	Used
AWS DynamoDb	Firestore

AWS Lambda	Cloud Function
AWS Comprehend	Natural Language API

Tests performed

Table 18: Testing

Test Name	Expectation	Pass
Performance of every question	It should provide relevant tags	Pass
If no question tag returned	It should not store anything to DB	Pass
Many category questions	It should only give max 5 suggestion	Pass

10. Virtual Assistance

The virtual assistance is developed using AWS service Lex. This virtual assistance gives basic information on important navigations like, login, create team, explore game and view score. Here I have integrated a lambda function to get the score of the team dynamically from the database and give it to the user. This lambda will take the input from the lex bot and then perform a filter on the database and will calculate the total score and display the results to the user.

This lex bot is integrated with the frontend by using Kommunicate Library. This library directly integrates the bot with the UI.

Below are the screenshots for the functional testing of the VA.

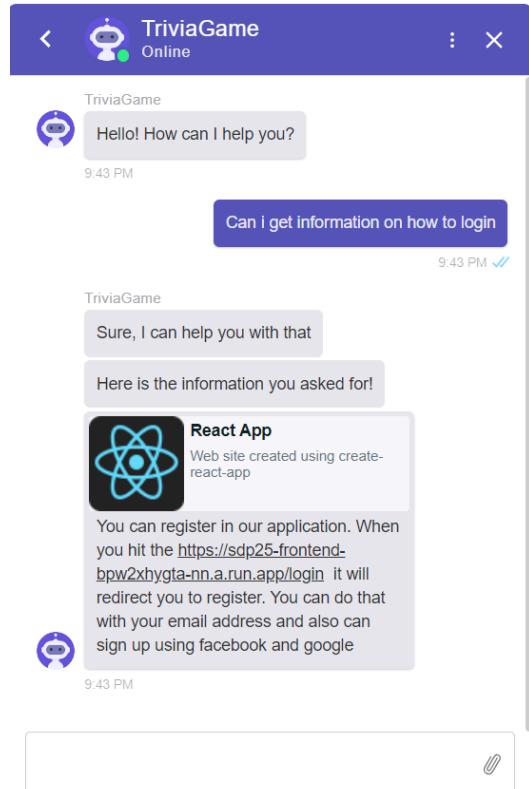


Figure 73: Chatbot

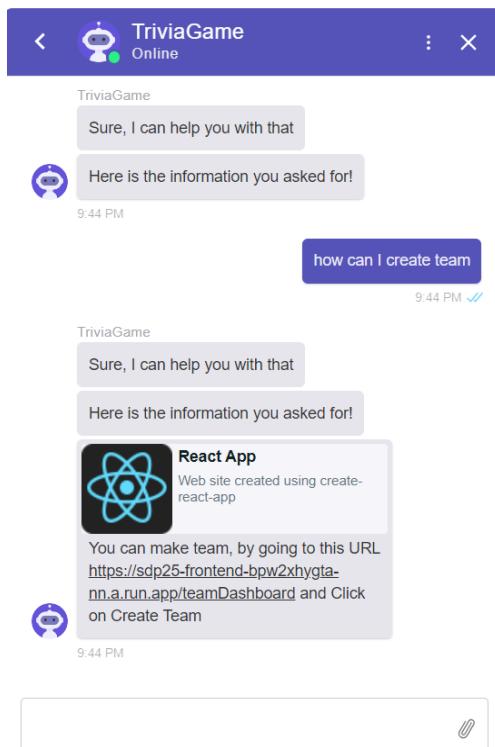


Figure 74: Chatbot response 2

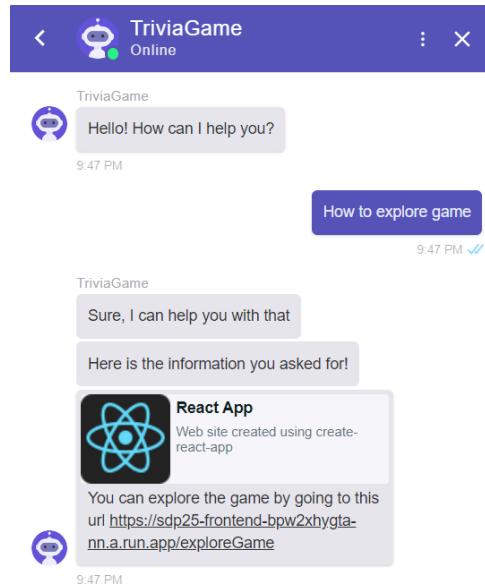


Figure 75: Chatbot response 3

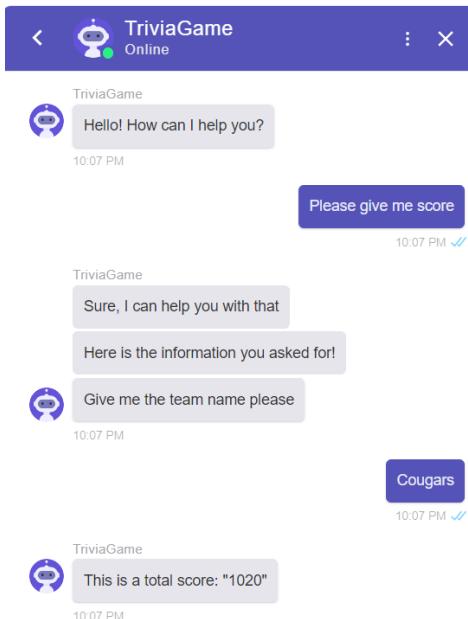


Figure 76: Chatbot response 3

The above chatbot answers 4 different questions, related to login, create team, explore games and score of teams.

For getting the team scored dynamically a lambda function is invoked. For that I had to enable the fulfillment of the intent that wants to calculate the score.

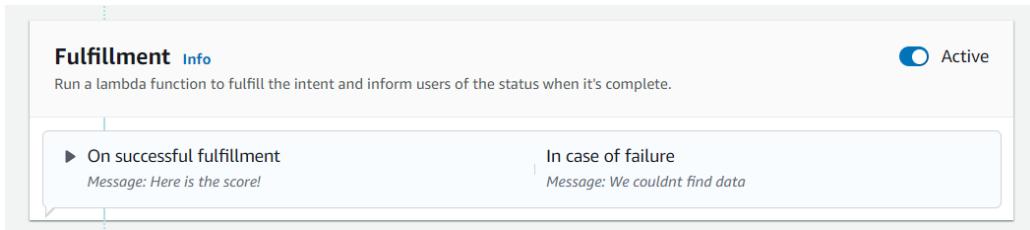


Figure 77: Chatbot fulfillment

This is the option where I can give the lambda i want to invoke.

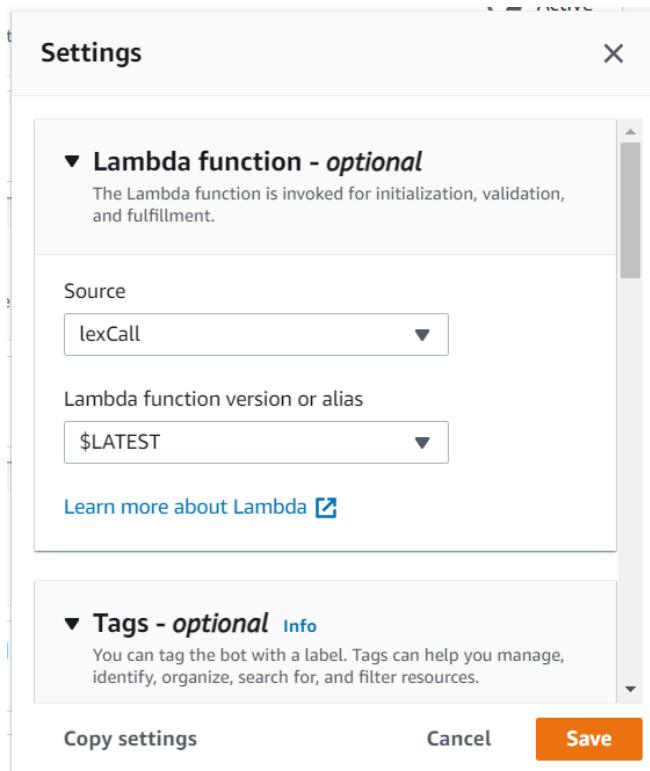


Figure 78: Call Lambda from chatbot

```

1 import json
2 import requests
3 import urllib.parse
4
5 def lambda_handler(event, context):
6
7     print(event)
8     total_score = 0
9
10    headers = {
11        "Content-Type": "application/json",
12        "Authorization": "Bearer YOUR_ACCESS_TOKEN"
13    }
14
15    payload = {
16        "teamName": event['inputTranscript'],
17    }
18
19    response = requests.post("https://7h6msp1f2.execute-api.us-east-1.amazonaws.com/1/giveLexData", headers=headers, json=payload)
20
21    if response.status_code == 200:
22        response_content = response.json()
23        print(response_content)
24
25        base_url = "https://jydpdnksqa.execute-api.ca-central-1.amazonaws.com/Prod/getScore"
26        url = f'{base_url}?teamId={response_content["teamId"]}'
27        print(url)
28        message = f"Hello world {response_content}"
29        print(message)
30
31        response_teamData = requests.get("https://rv7nufzjhc.execute-api.ca-central-1.amazonaws.com/Prod/getScore?teamId=team_5", headers=headers)
32
33        response_teamData.raise_for_status()
34        responseID = response_teamData.text
35        responseID = json.loads(responseID)
36        print(responseID)
37        for item in responseID:

```

(7 Bytes) 59.28 Python Spaces: 4

Figure 79: Called Lambda

The above snapshot has the code that calculates the total_score and returns it to the lex bot.
The below screenshot displays the intentions used to make this bot.

- StartGameIntent
- MakeTeamIntent
- LoginIntent
- GetTeamScore** Unsaved
- DecideTopic
- NewIntent
- LogoutIntent
- FallbackIntent

Figure 80: Intent calling lambda

The below screenshot demonstrates the conditional branching feature of the bot, it shows how the intent changes on the user's response.

Note: The AWS Lex only allowed 4 conditional branches so, I just implemented 4 conditions for the bot.

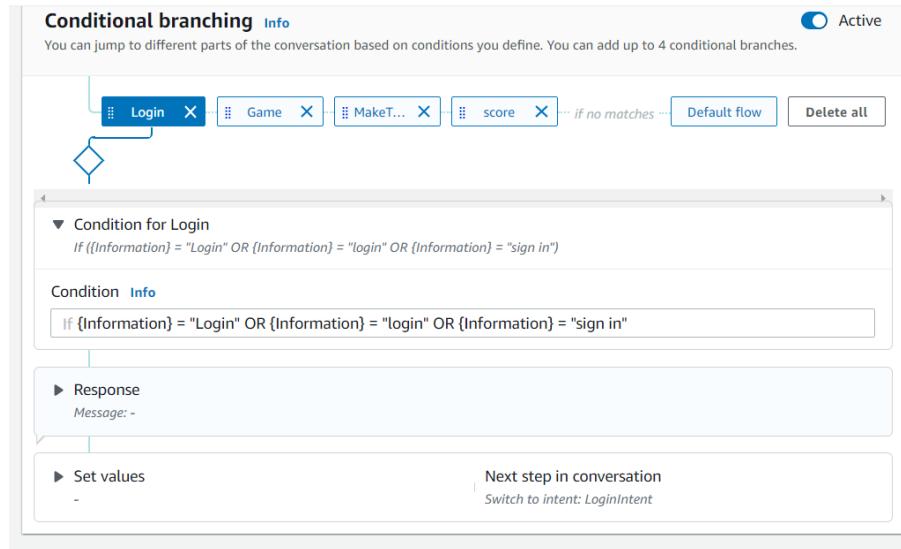


Figure 81: Condition in chatbot execution

API Testing (from postman):

The screenshot shows the Postman interface with a collection named 'New Collection - Run results'. It displays a single iteration with one test case named 'giveLexData'. The test case details are as follows: Method: POST, URL: https://i7h6msplf2.execute-api.us-east-1.amazonaws.com/1/giveLexData, Status: 200 OK, Duration: 334 ms, Body Size: 581 B. The test case result is listed as 'Passed (0)'.

Figure 82; API testing

Method: Post Endpoint: /giveLexData.

Table 19: Planned VS Used Services

	Planned	Implemented
Virtual Assistant	AWS Lex	AWS Lex
Compute	Lambda	Lambda

Functional Testing:

Table 20: Chatbot Functional Testing

Test Name	Expectations	Results	Status

Start conversation with bot	Every time new conversation must be started	With the option to start the conversation, every time the new conversation is started	PASS
Previous conversation is preserved	Past conversations must be saved	Users can see their past conversations	PASS
Assistance	The virtual assistant should be able to answer the questions correctly and promptly	Yes, In no time the bot answers and resolves the users query	PASS

Application Roadmap

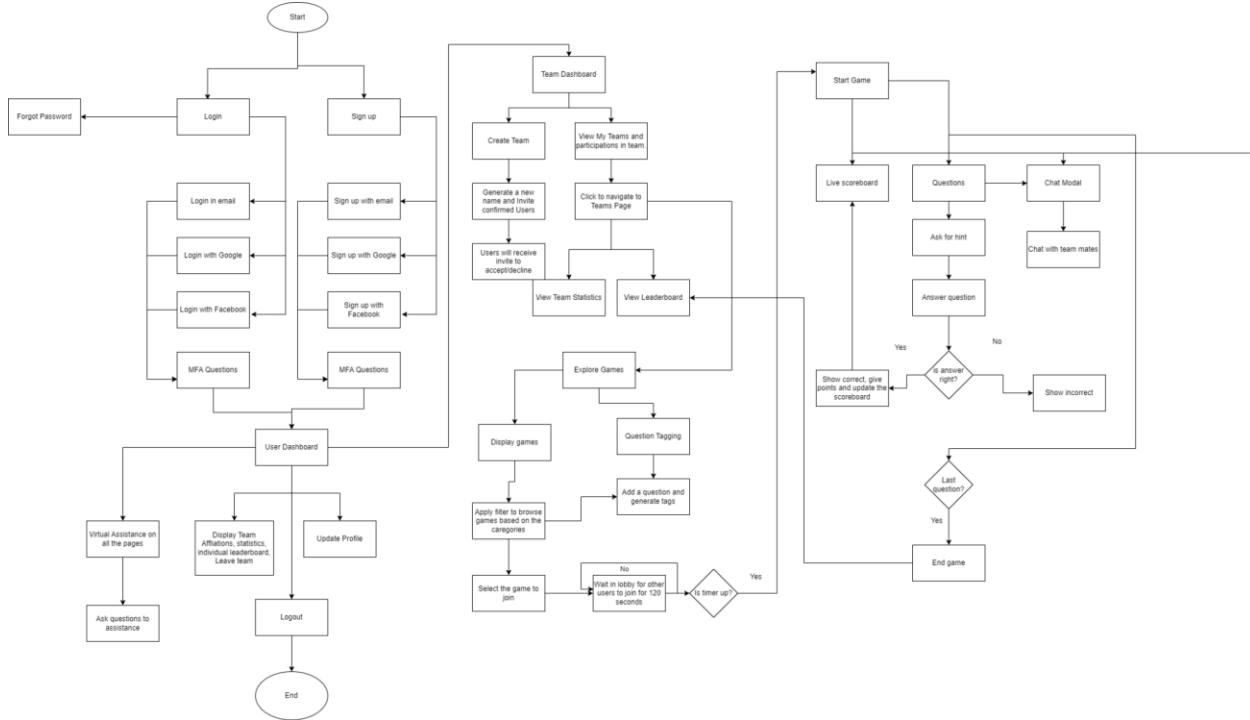


Figure 83: Application Roadmap [10]

We planned it exactly as we created it. However, we changed a few cloud services we wanted to use earlier, nothing was changed from the functionalist perspective. We implemented separate components following microservices architecture such that each component can work separately and statelessly. Then we integrated each other's components by giving proper routing and providing the required data.

User Authentication

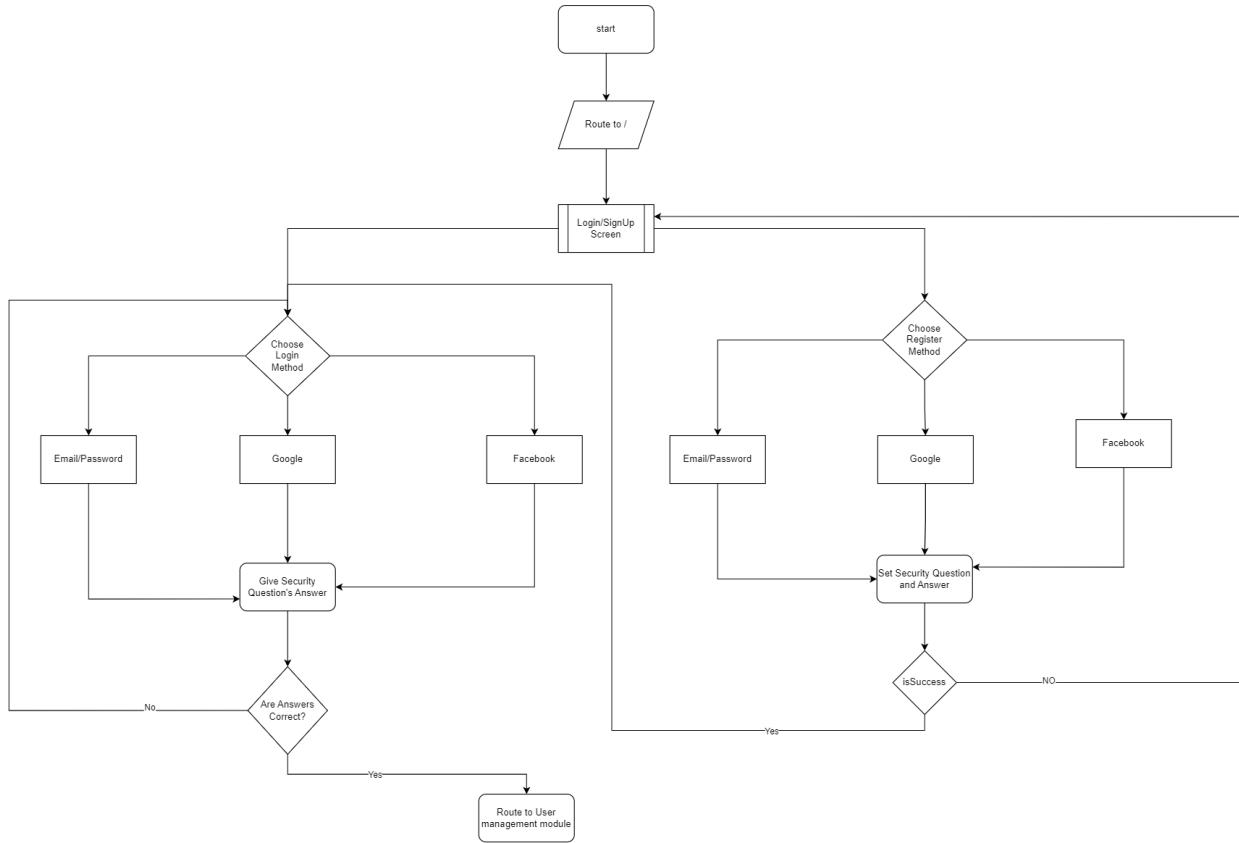


Figure 84: Feature 1 Flow diagram [10]

User Scenarios:

The User Authentication Module is designed with 2-factor authentications, and during the registration process, users are required to set up details for all three levels of verification. When a user attempts to log in, each level of verification is sequentially applied. The verification process follows these steps:

User provides email and password during login.

If the email and password are correct, the system proceeds to the next level.

If the email or password is incorrect, the user is prompted to provide the correct details and try again.

User is asked to answer the pre-set security question.

If the answer is correct, the system proceeds to the dashboard page.

If the answer is incorrect, the user is prompted to provide the correct answer and try again.

The planning and implementation of the 2-factor authentication process have been straightforward and consistent, ensuring that the user must provide the correct input at each level to proceed further. If the user gives a wrong input at any level, they are required to retry until the correct input is received by the system.

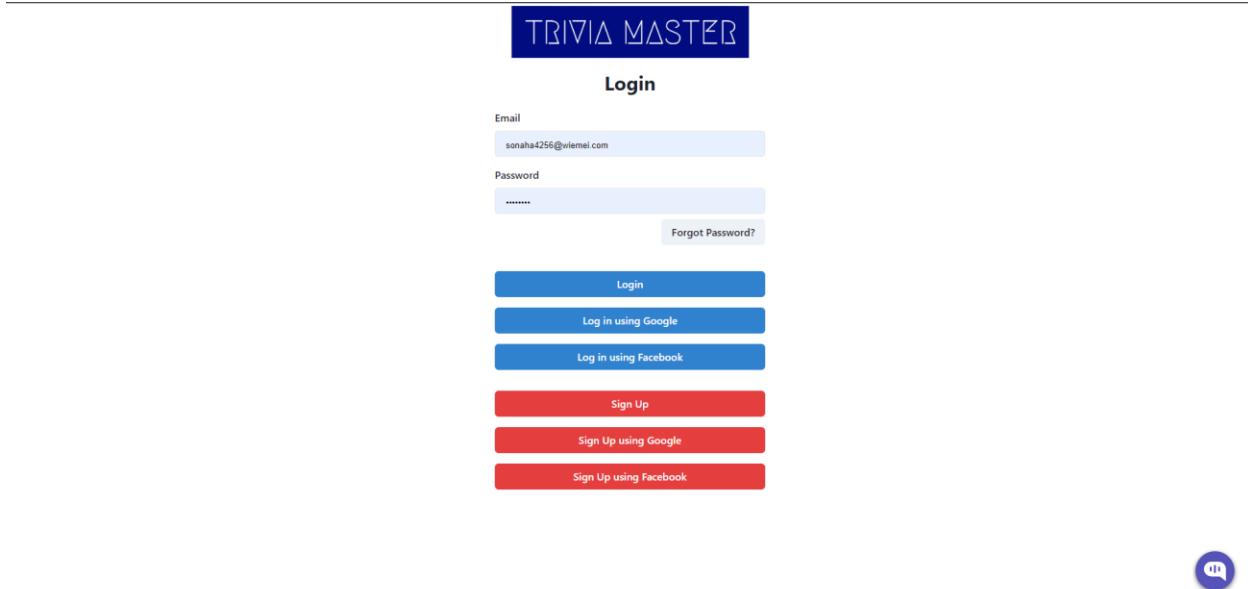


Figure 85: Functional Testing Feature 1

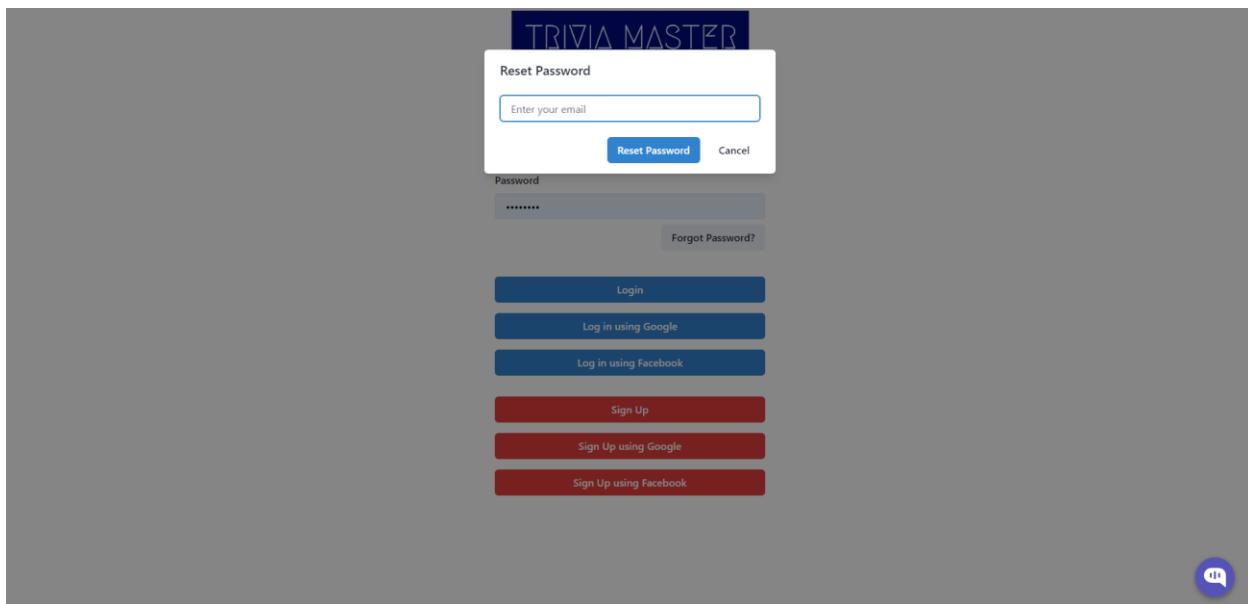


Figure 86: Functional Testing 2 - Feature 1

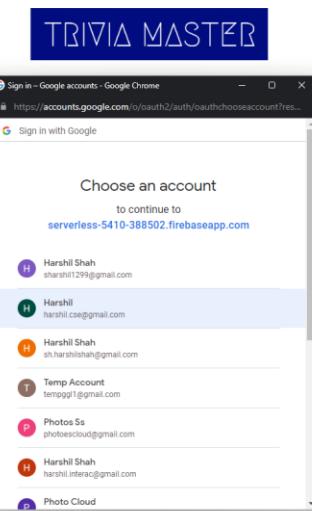


Figure 87: Functional Testing 3 - Feature 1

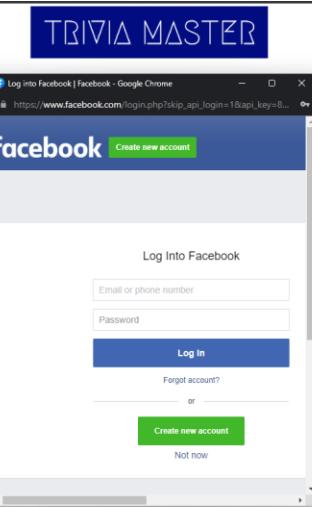


Figure 88: Functional Testing 4 - Feature 1



Sign Up

Email:

sonaha4256@wiemei.com

Password:

First Name:

Last Name:

Upload Profile Picture:

Choose File No file chosen

Question 1:

What is your favorite food?

Answer 1:

Question 2:

What is your favorite food?

Answer 2:

Question 3:

What is your favorite food?

Answer 3:

Sign Up



Figure 89: Functional Testing 5 - Feature 1



Sign Up

Upload Profile Picture:

Choose File No file chosen

Question 1:

What is your favorite food?

Answer 1:

Question 2:

What is your favorite food?

Answer 2:

Question 3:

What is your favorite food?

Answer 3:

Sign up using Google



Figure 90: Functional Testing 6 - Feature 1

The screenshot shows the 'Sign Up' page of the Trivia Master application. At the top, there's a blue header bar with the text 'TRIVIA MASTER'. Below it, the page title 'Sign Up' is displayed. There's a field for 'Upload Profile Picture' with a 'Choose File' button and a message 'No file chosen'. Below this are sections for 'Question 1' and 'Answer 1', 'Question 2' and 'Answer 2', and 'Question 3' and 'Answer 3', each consisting of a dropdown menu. At the bottom of the form is a blue button labeled 'Sign up using Facebook'. To the right of the form, there's a small circular icon with a speech bubble symbol.

Figure 91: Functional Testing 7 - Feature 1

User Management

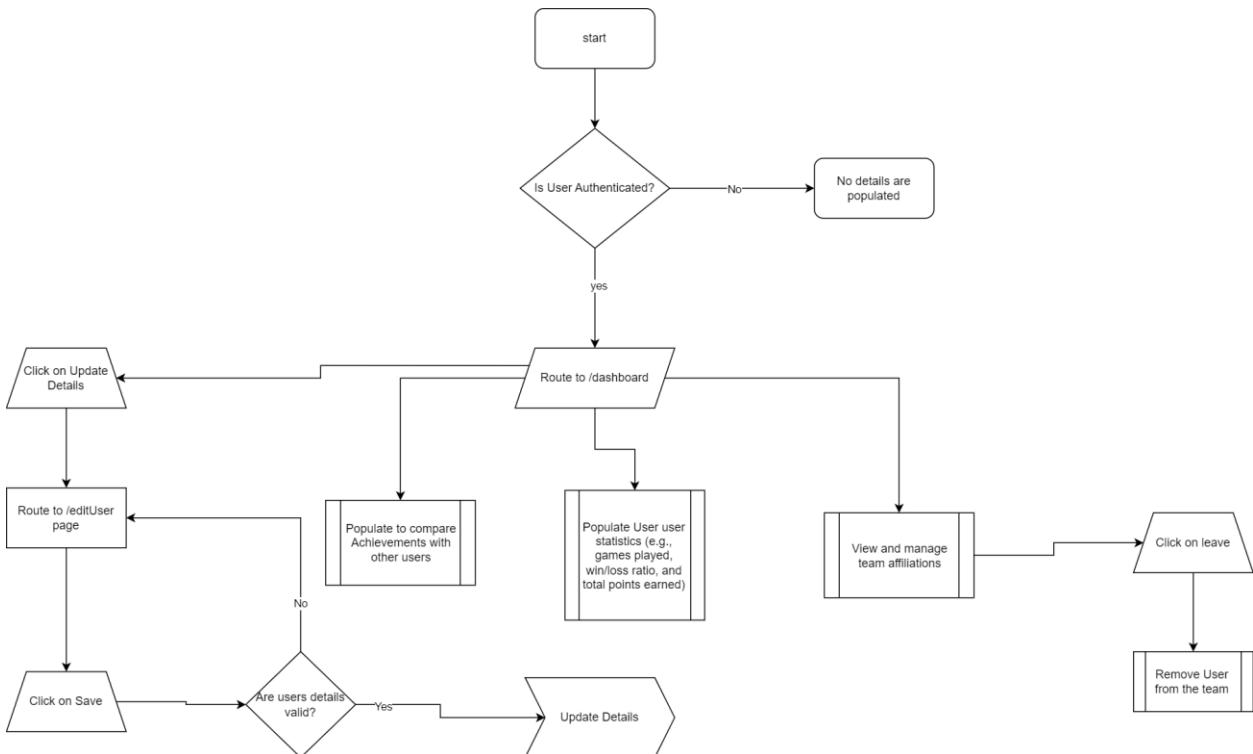


Figure 92: User Management - Feature 2 Workflow [10]

User Scenarios:

Obtain User information as well as user statistics and comparable other user statistics if the user is authenticated.

Otherwise, do not show user specific details.

To update user details, click on the Update Details button.

Reroute to /editUser

Validate all fields when use press on save details

The screenshot shows the Trivia Master User Dashboard. At the top, it says "Logged in sharshil1299@gmail.com" and "Hi Harshil Shah". Below this is a profile picture icon. There are two tabs: "Update Profile" (which is active) and "Team Dashboard".
Game Stats
Win/Loss Ratio: 1.00
Total Points Earned: 720
Games Played: 2
Win: 1
Loss: 1
Team Affiliations
Fusion (Leave Team)
Sprockets (Leave Team)
Top Achievers

	USER ID	ACHIEVEMENT
LOWEST LOSS	user_1	0
HIGHEST WIN	user_1	59
HIGHEST WIN/LOSS	user_1	1.1346153846153846
TOP SCORER	user_1	80615

Figure 93: User Dashboard

The screenshot shows the "Update Profile" section of the Trivia Master application. It includes fields for User ID (u0r6GBGTaufc3xFirMvdNSm9ft2), Email ID (sharshil1299@gmail.com), First Name (Harshil), and Last Name (Shah). A "Save Changes" button is at the bottom right. A blue speech bubble icon is located in the bottom right corner of the page.

Figure 94: Update User Details

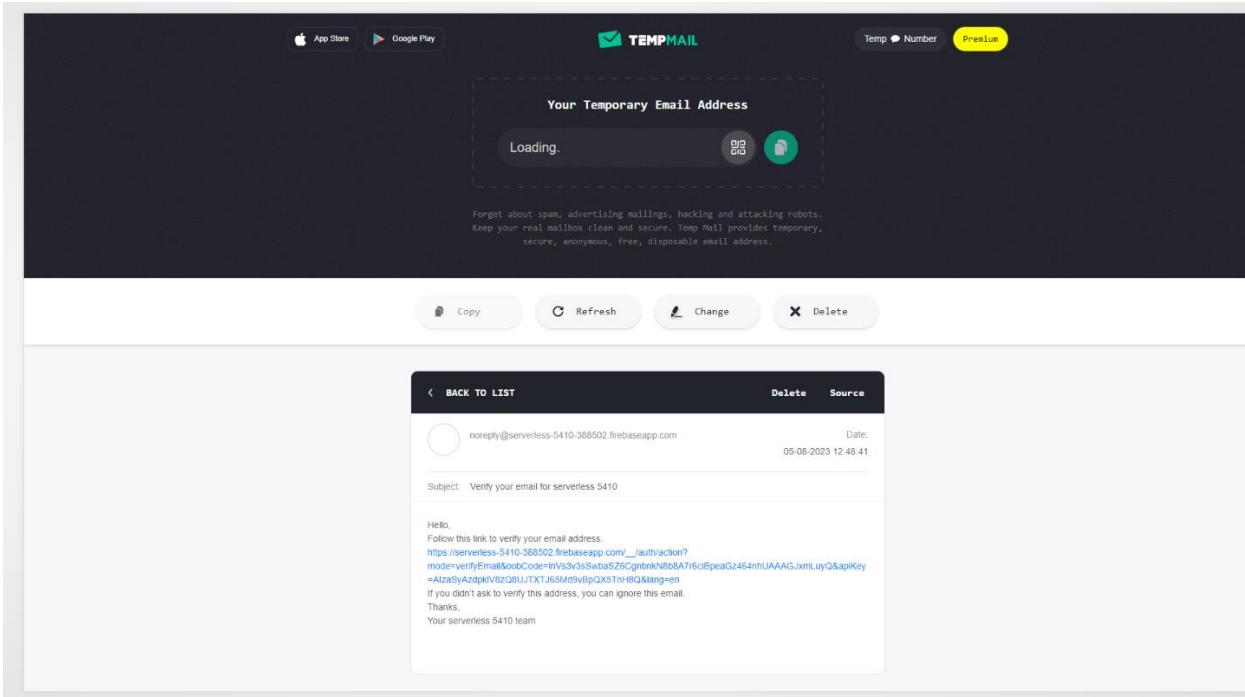


Figure 95: Verification mail on User register using Email ID

Team Management

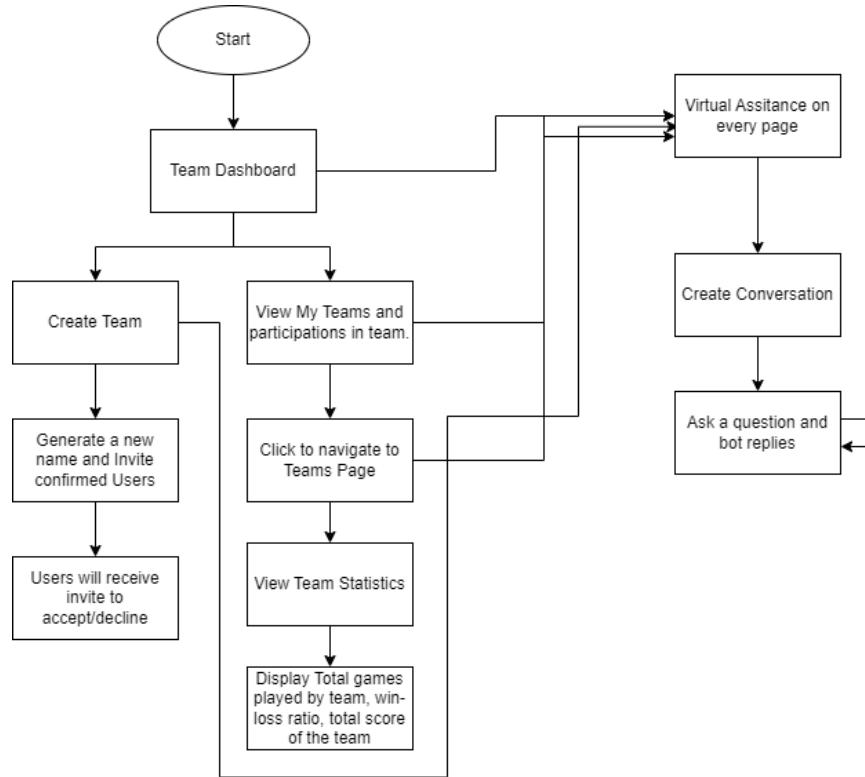


Figure 96: Flow Diagram Feature 3 [10]

The logged in user wants to play a game but does not have teammates nor is a part of any team. In this scenario, the user will create a team and invite people to join the team, just by clicking on the create team button and selecting the users, the selected users will get email notification and on accepting they will be added to the team. Now something happens and the logged in user does not want to play with a particular user, being the team owner, they can remove any user. And can promote anyone to admin so the admin will also have access to remove people from the team.

As a normal participant, they have the option to leave the game. And if any logged in user wants to see team statistics, they can view them by clicking on the View Leaderboard option.

Trivia Game Lobby

In the Trivia Game Lobby, users are presented with a list of available trivia games they can join. The lobby fetches essential game data, including the game title, category, difficulty level, and time frame, from a centralized data table. Users have the option to filter the games based on their preferred categories, difficulty levels, and time frames, allowing them to find games that suit their interests. The filtered games are displayed with relevant details, such as the game

name, category, difficulty level, and the number of participants already in each game. Users can then join a game they find appealing and wait for it to start. For scheduled games, a countdown timer is visible, indicating the time remaining until the game begins. On this screen if any new user joins the game, already joined person can see the list of user who have joined the game. At the scheduled start time, the game server fetches questions from the database based on the selected category, and the game starts, allowing all participants to answer trivia questions within a specified time frame.

In-game experience

Upon completion of the lobby registration and timer expiration, all users will be directed to a central screen where they will encounter questions presented one by one. Users have the opportunity to submit their answers within their respective teams. The win-loss ratio for each team will be computed based on the collective performance of its members. Each question will be displayed for approximately 20 seconds, after which the correct answer, along with an explanation, will be revealed for 5 seconds.

On the left side of the screen, a column will showcase the scores of all participating users in real-time, updating after each answer submission. Simultaneously, a chat window on the right side allows teams to communicate and discuss their answers. Additionally, a "hint" button is available to provide assistance with the questions. Once all the questions have been answered, participants will be redirected to the leaderboard screen to view the final rankings.

The scoring is calculated based on the promptness of the participants' responses, the submission status of their teammates, and the use of hints during the quiz. Faster answer times, synchronized team submissions, and non-reliance on hints contribute to higher marks. The scoring system takes into account individual and collaborative performances, emphasizing the significance of timely and well-coordinated contributions from team members.

Leaderboards

In the leaderboard screen, users are presented with graphical representations showcasing the scores of individual players and teams throughout the entire duration of the game. The interface offers filtering options, enabling users to view data based on daily, weekly, monthly, or all-time statistics. Two graphs are dedicated to displaying the static leaderboards, one for teams and another for individual players. Additionally, two more graphs exhibit the top-performing teams and players, highlighting their achievements within the game.

Automated Question Tagging

The user enters a question into the application that incorporates the question tagging model, which uses the Natural Language API. The model processes the input question, analyzing its context and semantics, and predicts relevant tags or categories. The user then sees these predicted tags displayed on the frontend of the application.

Virtual Assistant

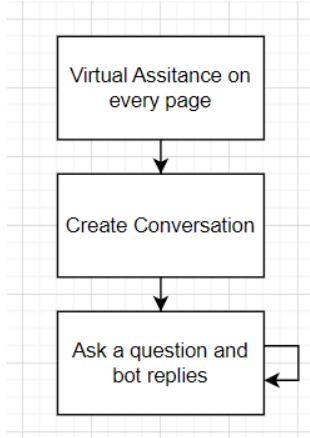


Figure 97: Flow Diagram Feature 10 [10]

The logged in user faces difficulty in navigation, they can take help from the VA and ask them proper questions and the VA will provide assistance to navigate through the web application. If they want to know the current total score of a team, the user can ask VA for the score of a particular team and the assistant will reply with the total score of that team. These are user flow diagrams as to how the user will be using every feature can be explained using flow charts feature by feature and possible user scenarios can be explored in the flow diagram.

Architecture

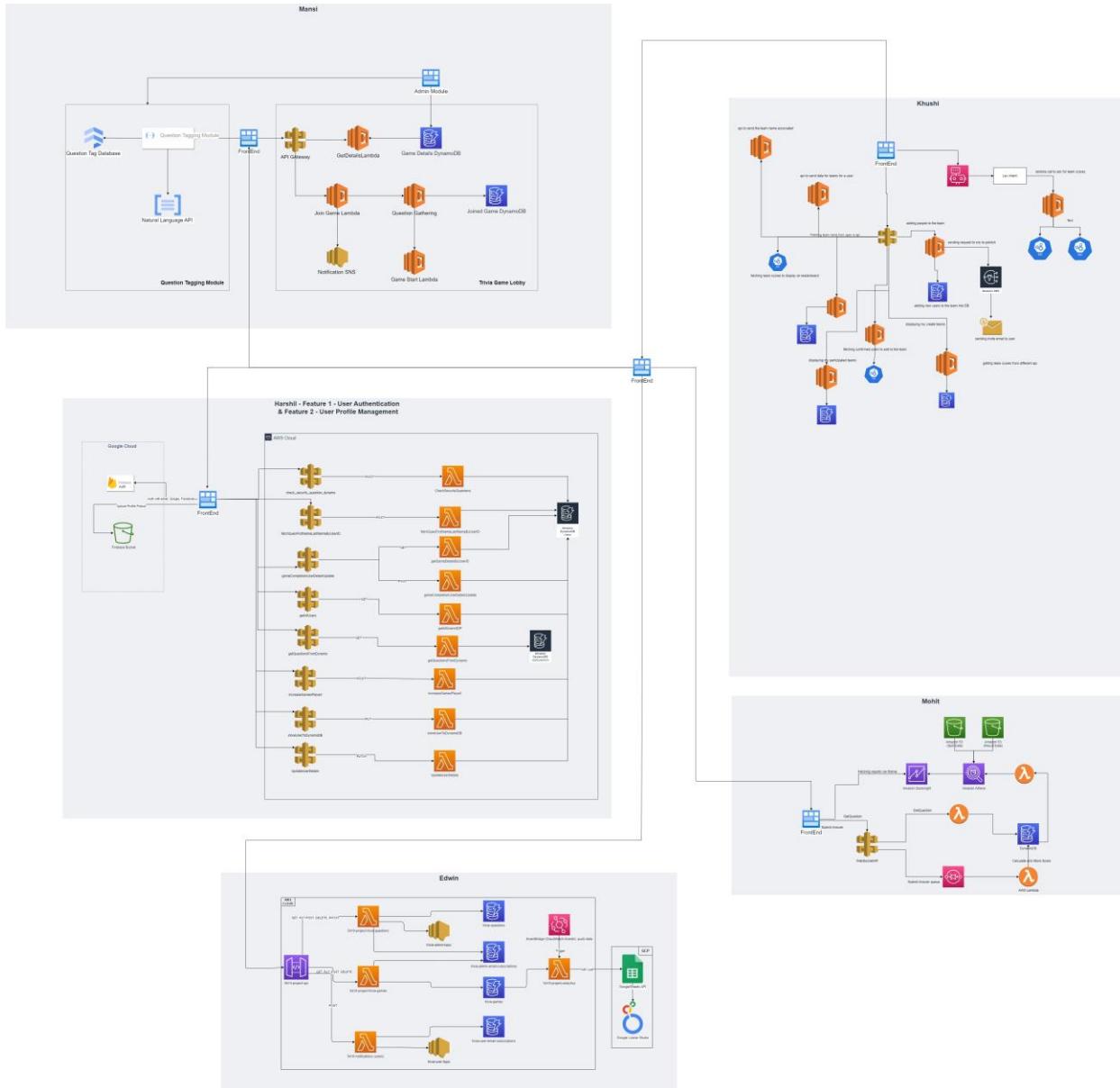


Figure 98: Architecture Diagram [10]

Worksheet

Module Owners

Table 21: Module Owners

Module	Module Owners
1. User Authentication	Harshil Shah
2. User Profile Management	Harshil Shah
3. Team Management	Khushi Shah
4. Trivia Game Lobby	Mansi Vaghasiya
5. In-game Experience	Mohit Sojitra
6. Leaderboards	Mohit Sojitra
7. Trivia Content Management (Administrators)	Edwin Adams
8. Notifications and Alerts	Edwin Adams
9. Automated Question Tagging	Mansi Vaghasiya
10. Virtual Assistance	Khushi Shah

There were no changes in the allocation of work within the team members.

Paragraph on Individual Component:

1. User Authentication (Harshil Shah)

In this implementation, the initial plan was to utilize AWS Cognito for user authentication and authorization, with user details stored in AWS DynamoDB and question-and-answer details in GCP Firestore. AWS Lambda would handle data storage and cipher key generation. However, during the build process, the decision was made to switch to Firebase Authentication for user authentication due to its ease of implementation and compatibility with multiple sign-in services. User details, including email, name, question-answer pairs, and other information, continue to be stored in AWS DynamoDB using AWS Lambda for data management. The initial idea of using GCP Firestore for question-and-answer details was dropped in favor of

maintaining consistency within the AWS environment and reducing complexity. Additionally, GCP Buckets were employed to store profile picture images, with the image URLs stored in DynamoDB.

2. User Profile Management (Harshil Shah)

In the initial plan, the user interaction system was intended to be built using GCP Firestore and Cloud Functions for scalability and efficiency. Firestore collections were proposed for user profiles, user statistics, team affiliations, and achievements, with Cloud Functions handling interactions and data updates. However, during the implementation, the decision was made to switch to AWS DynamoDB and Lambda Functions. User profiles, statistics, and achievements were stored in DynamoDB, and Lambda Functions were developed to handle profile updates, calculate statistics, and manage team affiliations securely. Despite the deviation, the AWS stack successfully delivered a scalable, robust, and well-integrated user interaction system, meeting the project's requirements effectively.

3. Team Management (Khushi Shah)

Team management is developed using AWS Lambda, API Gateway, DynamoDB and SNS. It has the requirement for dynamically generating unique team names by integrating open ai API which generates new names. Once the name is generated, users who have SNS already subscribed will be displayed in the dropdown. When the user selects the invite option, the system automatically sends personalized emails with SNS. When a user accepts an invitation, the corresponding data is promptly updated in DynamoDB. The email will have an accept or decline option. And based on the user selection, the DB will be updated accordingly. Finally, the team dashboard is the central page for team management. It displays various team names and by clicking on any team, the system navigates to the team page. The Team Dashboard also grants team creators more rights, such as the ability to make users admins or kick them off the team. The "View Leaderboard" feature enhances the gameplay by giving players access to crucial team information including games played, win-loss percentages, and overall point totals.

4. Trivia Game Lobby (Mansi Vaghasiya)

Trivia game lobby is developed using AWS Lambda, DynamoDB and SNS. Using this module the admin generated game and In Game Experience will be coordinate. Lambda function will be called for fetching the information from DynamoDB. This data will be filtered by criteria and visible to the end-user. When the user clicks on the join button the game will be started and users will get an email in the general channel saying that the game is getting started in our lobby. Also we will make an entry in DynamoDB for the first user and count down will be

started from the first join click. When other users will be joined they will see the name of users already joined in the game and when the counter goes to 0 all users will be redirected to In game experience page. Meanwhile the data will be fetched for category questions to make requests in In Game Experience to start the game.

5. In-game experience (Mohit Sojitra)

During the in-game experience, pertinent data is extracted from the trivia game lobby component, where game selection and scheduling occur. After the game schedule is finalized, the required information, such as game questions, correct answers, start time, number of teams, and hints, is retrieved from Amazon DynamoDB.

Once the game commences, backend communication with the frontend is established through WebSocket connections. The backend sends questions to the frontend every 25 seconds, allowing the frontend to display each question for 20 seconds, followed by a 5-second display of the correct answer and its explanation. Upon the frontend's submission of answers and corresponding timings, the data is forwarded to AWS SQS from Amazon DynamoDB.

The backend then proceeds to calculate the scores based on the responses, storing the results back into DynamoDB. Simultaneously, the updated scores are transmitted to the frontend in real-time for display. Upon game completion, a game-over message is dispatched to the frontend via Lambda, prompting the frontend to redirect to the leaderboard screen for final rankings and results.

6. Leaderboard (Mohit Sojitra)

The leaderboard feature retrieves real-time data directly from the Amazon DynamoDB table. To accomplish this, it leverages AWS Athena serverless, which is responsible for querying the data stored in DynamoDB. AWS Lambda and AWS S3 play essential roles in this process, aiding in the execution and storage of the query results obtained from Athena.

Once the data is successfully queried from DynamoDB and processed through AWS Athena, it is seamlessly integrated into an Amazon QuickSight dashboard. This dashboard provides a comprehensive and visual representation of the leaderboard, allowing users to access up-to-date scores and statistics. The combination of AWS Athena, AWS Lambda, and AWS S3 empowers the system to efficiently manage and display dynamic leaderboard data, providing a smooth and effective user experience.

9. Automated Question Tagging (Mansi Vaghasiya)

In the Question Tagging module, the Natural Language API is used to find associative tags for questions. Additionally, Cloud Function and Firestore are employed to store question data. This

module has an individual page for user experience. When a user adds any question from the page, the Cloud Function will be triggered, which in turn calls the NLP API for classification. The Language module of the NLP API classifies the text content and suggests some category tags that may be relevant to the user's given data. There are various tags that can be associated with a question, but for simplicity, only 5 will be considered.

10. Virtual Assistance (Khushi Shah)

Virtual Assistance, developed using AWS Lexi, is a tool designed to provide users with basic information and facilitate navigations such as login, team creation, game exploration and score viewing. One of the information and standout features of this bot is that it provides a dynamic scoring system. This feature is made possible by integrating it with Lambda function and the lambda function is connected with the database. When users ask for the score, the bot asks the team's name and when the users give the team name it will take that value and invoke the lambda function and will fetch the total score of the team and display it in the response of the bot. This lex bot is integrated with the front end with the help of Kommunicate library which makes the integration easy and user experience seamless. With functional testing, the virtual assistant has proven its efficacy, providing accurate and timely responses to user queries.

The screenshot shows an Excel spreadsheet titled "Bi-Weekly Sprints". The table structure is as follows:

	A	B	C	D	E	F	G	H
1	Project Member Name	Tickets Assigned	Status	Assigned Date	Completion Date			
2		Analysis for Task 1 (authentication) Which option to choose	On going					
3	Harshil Shah	Basic UI creation	On going					
4		Continuous Deployment on Netlify	Done					
5		Analysis for Task 2	Backlog					
6		Analysis/Research for Task 7						
7	Edwin Adams							
8								
9								
10	Mohit Sojitra	Working on In-game experience: task: Created basic UI in React Analyze the architecture Working on in-game chat feature	On going	30-May				
11	Khushi Shah	Analysis/Research for task 3 and 10	On going	30-May	6-Jun			
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								

Legend:

Status	What it means
On going	Currently in progress
Backlog	Not started yet
Done	Completed

Figure 99: Sprint Planning Excel

Code commit and Merge Request details with the screenshot is under Spring Plan Section.

Sprint Plan

During the planning phase to launch our task, we had a series of meetings to get organized. We carefully reviewed the project requirements in one meeting and then assigned tasks to team members in the next one. Each person got two specific tasks, and each team member was responsible for choosing the right cloud service (AWS or GCP) and documenting it in a sheet for their assigned task.

Once we knew what we needed to do, we split the work into a two-week sprint cycle. We had regular meetings on weekends to see what tasks were done and what was left. Sometimes, we had extra meetings to handle new requirements or things from other parts of the project. We planned to finish the first part of the project after two sprints, which was about a month. The second part would also take two sprints. The rest of the time was dedicated to putting everything together and making sure we had all the reports we needed for the launch.

Table 22: Sprint Planning

Planned Sprint	Team Member	About to finish
29th May - 12 June	Khushi	Team Management - Architecture planning and requirement gathering
	Harshil	User Authentication - Architecture planning and requirement gathering
	Mansi	Trivia Game Lobby - Architecture planning and requirement gathering
	Mohit	In-game Experience - Architecture planning and requirement gathering
	Adwin	Content MAnagement - Architecture planning and requirement gathering
12 June - 3 July	Khushi	Team Management - Completion
Extended due to reading week	Harshil	User Authentication - Completion
	Mansi	Trivia Game Lobby - Completion

	Mohit	In-game Experience - Nearly Completed
	Adwin	Content Management - Completed
3 July - 17 July	Khushi	Virtual Assistance
	Harshil	User Profile Management
	Mansi	Question tagging
	Mohit	Ingame Lobby complete & Leaderboard
	Adwin	Notification module
17 July - 31st July	Khushi	Virtual Assistance - complete
	Harshil	User Profile Management - complete
	Mansi	Question tagging - complete
	Mohit	Leaderboard - completed
	Adwin	Notification - completed
31st July - 4th August	Team	Integration and deployment

At the beginning, we made a timeline considering mid-term exams and the features' complexity. The first two parts went well and were completed on time. However, after mid-term, we faced deadline issues of other subjects, and the second feature took longer than expected, finishing on 2nd August instead of the planned 31st July. Luckily, because we communicated well, the integration part was faster, taking only one day instead of the 2-3 days we thought it would take.

Individual experience:

Harshil Shah:

Since the beginning of the project, we divided the features based on individual comfort and expertise. I started my journey by conducting thorough research to determine the best approach

for implementation, considering the right services for each specific problem. Initially, I opted for Amazon Cognito due to its extensive features and control. However, upon further exploration, I found that Firebase Authentication offered easy integration with the application and provided all the necessary details like profile picture URLs, names, and email IDs. Thus, I switched to Firebase Authentication, which required dedicated effort and about a month to successfully integrate Google, Facebook, and email-password logins, overcoming several challenges along the way.

Simultaneously, I worked on setting up a CI/CD pipeline for deploying the application on Netlify. However, later in the process, I discovered the cloud run requirement, which led me to revisit the deployment strategy. With midterms and assignments piling up, my progress temporarily slowed down, but I managed to find time during the end term to focus on my modules.

After completing my module, the team gathered to integrate our individual modules and create the required APIs for seamless communication between the modules. To streamline the deployment process, I developed a run.bat script to build and deploy the frontend to cloud run. Finally, with rigorous testing and deployment, we successfully completed a fully functional project just before the deadline. It was a rewarding experience to witness the project come together cohesively, and we celebrated the successful implementation of the features we had envisioned.

Khushi Shah:

In the beginning, everything was on track and research and analysis of the feature was going as expected, but the logic I wrote previously only allowed me to add a user to one team only. But that was not the requirement so I lagged behind for that module. So, I missed that sprint. Later on when I was trying to figure out that problem and how to solve it, mid-term exams and a lot of assignment submissions came so it slowed me down a bit. But once I was done with the prior submissions, I took responsibility and completed the module which was pending. And started working on the second module feature. I faced some challenges in this module but eventually finished that document.

Mansi:

In the beginning, everything went well for the first two sprints. But when I was creating the question tagging module, it took me more time than I thought. Testing with Comprehend took a lot of time. So, I decided to switch to GCP to finish the work. Because of these delays, I missed the deadline for sprint 3. However, I didn't give up and managed to complete the module within the original timeframe we had planned. Even though there were some challenges, I believe the extra effort I put in will make the question tagging module better in the end.

Mohit:

Initially, I anticipated that the in-game experience module would be straightforward and efficient. However, as the project progressed, I encountered the complexity of maintaining user sessions and synchronizing data across multiple users, leading to unexpected challenges.

Consequently, I had to invest additional time in learning how to establish two-way communication in the cloud, which caused a delay in my progress, and I missed the first sprint milestone.

Despite the initial setbacks, I remained dedicated to the task and managed to complete it within the defined sprint timeline. The integration of user sessions and real-time data synchronization was successfully implemented. Additionally, I faced a challenge when I discovered that LookerStudio lacked support for my credit account, prompting me to switch to AWS QuickSight for creating the leaderboard.

Thankfully, my experience with LookerStudio proved beneficial, as it prepared me well for working with ETL tools and real-time BI tools. This background knowledge enabled a smooth transition to AWS QuickSight, and I was able to complete the leaderboard integration on time. Throughout the process, I learned valuable lessons about cloud-based two-way communication and the versatility of cloud services in data analysis and visualization.

Merge Requests Report

Table 23: Merge Request Details

Title	State	Source Branch	Target Branch	Target Project ID	Author	Author Username	Created At (UTC)
Login Signup	closed	login-signup	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-06-02 20:41:29 UTC
Automated deployment to netlify	merged	ci-cd	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-06-02 20:49:18 UTC
Cd	merged	cd	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-06-02 21:35:11 UTC
Login Signup	merged	login-signup	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-06-02 22:43:26 UTC

converted to typescript	merged	login-signup	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-06-04 15:27:50 UTC
In game window	merged	in-game-window	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-08-02 18:52:09 UTC
Khushi	merged	khushi	master	76362	Khushi Hirenbai Shah	khshah	2023-08-02 19:42:26 UTC
forget password	merged	login-signup	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-08-03 14:07:52 UTC
Khushi URL working fine	merged	f8_notification	master	76362	Harshil Kalpeshkumar Shah	harshils	2023-08-03 22:04:44 UTC
Mv gamelobby dev	merged	mv-gamelobby-dev	master	76362	Mansi Vaghasiya	mvaghasiya	2023-08-03 23:26:29 UTC
Khushi	merged	khushi	master	76362	Khushi Hirenbai Shah	khshah	2023-08-04 01:34:22 UTC
Khushi	merged	khushi	master	76362	Khushi Hirenbai Shah	khshah	2023-08-04 02:05:57 UTC
Minor change	merged	khushi	master	76362	Khushi Hirenbai Shah	khshah	2023-08-04 02:10:12 UTC
In game window	merged	in-game-window	master	76362	Mohit Rajeshkumar Sojitra	msojitra	2023-08-04 02:19:50 UTC

CHanging routes and adding buttons	merged	khushi	master	76362	Khushi Hirenbai Shah	khshah	2023-08-04 02:55:22 UTC
Mv gamelobby dev	merged	mv-gamelobby-dev	master	76362	Mansi Vaghasiya	mvaghasiya	2023-08-04 02:58:37 UTC
Khushi	merged	khushi	master	76362	Khushi Hirenbai Shah	khshah	2023-08-04 13:39:30 UTC

Code Commits Report

Mohit Rajeshkumar Sojitra > CSCI5410-SUMMER-23-SDP25 > Commits

master serverless-fe-team25 Author Search by message

04 Aug, 2023 19 commits

- added redirect Harshil authored 18 hours ago 63ec0e44
- Dockerfile updated Harshil authored 19 hours ago 7073f498
- Merge branch 'mv-gamelobby-dev' into 'master' Mohit Rajeshkumar Sojitra authored 19 hours ago 27479c97
- changed button name mansi authored 19 hours ago f10d1d62
- Merge branch 'khushi' into 'master' Khushi Hirenbai Shah authored 20 hours ago 5403a974
- changing url Khushi Shah authored 20 hours ago 050323b6
- Merge branch 'master' of git.cs.dal.ca:msojitra/serverless-fe-team25 into khushi Khushi Shah authored 20 hours ago fd22c81a
- Pushing docerfile Khushi Shah authored 20 hours ago 6634cdd2
- Merge branch 'master' of https://git.cs.dal.ca/msojitra/serverless-fe-team25 Mohit Rajeshkumar Sojitra authored 20 hours ago 2c55faff
- game changed 4d5c9790

Figure 100: Sprint Commit 1

 Khushi Shah authored 1 day ago		
03 Aug, 2023 37 commits		
 Merge branch 'mv-gamelobby-dev' into 'master' 	 f59007a2	  
Harshil Kalpeshkumar Shah authored 1 day ago		
 Merge branch 'master' of https://git.cs.dal.ca/msojitra/serverless-fe-team25 into 'mansi'	696730d5	  
mansi authored 1 day ago		
 MC	a663c8db	  
mansi authored 1 day ago		
 Merge branch 'khushi' into 'master' 	 16450ef6	  
Khushi Hirenbai Shah authored 1 day ago		
 CHanging routes and adding buttons	c0b34a6d	  
Khushi Shah authored 1 day ago		
 Merge branch 'in-game-window' into 'master' 	 425baa68	  
Mohit Rajeshkumar Sojitra authored 1 day ago		
 Merge branch 'master' into in-game-window	3c82fc12	  
Mohit Rajeshkumar Sojitra authored 1 day ago		
 Merge branch 'in-game-window' of... 	f2b7d2e0	  
Mohit Rajeshkumar Sojitra authored 1 day ago		
 api urls	8cb182d1	  
Mohit Rajeshkumar Sojitra authored 1 day ago		
 Merge branch 'khushi' into 'master' 	 455347d3	  
Khushi Hirenbai Shah authored 1 day ago		
 Minor change	77b1fa24	  
Khushi Shah authored 1 day ago		

Figure 101: Sprint Commit 2

 Harshil Kalpeshkumar Shah authored 1 day ago		
 forget password done	49e71cec	  
Harshil authored 1 day ago		
02 Aug, 2023 16 commits		
 calls from khushi added	6b7b5571	  
Harshil authored 2 days ago		
 team affiliations added	 99acc272	  
Harshil authored 2 days ago		
 Merge branch 'khushi' into 'master' 	 3c3b99a5	  
Khushi Hirenbai Shah authored 2 days ago		
 Adding code	d2c707a4	  
Khushi Shah authored 2 days ago		
 merging master to main	86b58bfb	  
Khushi Shah authored 2 days ago		
 Adding team management	fd24cbab	  
Khushi Shah authored 2 days ago		
 filter page created	d9ab6d3b	  
mansi authored 2 days ago		
 Merge branch 'in-game-window' into 'master' 	 9eb4ad4f	  
Harshil Kalpeshkumar Shah authored 2 days ago		
 from mohit	348107ef	  
Harshil authored 2 days ago		
 router	8f7bc1d2	  
Harshil authored 2 days ago		

Figure 102: Sprint Commit 3

 EDIT USER changes	Harshil authored 3 days ago	 449630e1		
01 Aug, 2023 12 commits				
 merging with khushi	Khushi Shah authored 3 days ago	b8ebb6b1		
 Facebook done	Harshil authored 3 days ago	 41ee8012		
 Adding integrated bot	Khushi Shah authored 3 days ago	7c9f4270		
 Adding chatbot	Khushi Shah authored 3 days ago	8f9dd5d7		
 chatbot added	Harshil authored 3 days ago	61a9312a		
 Integrating bot	Khushi Shah authored 3 days ago	434271d7		
 check now chatbot should work	Harshil authored 3 days ago	e425a8f3		
 Merge branch 'khushi' of git.cs.dal.ca:msojitra/serverless-fe-team25 into khushi	Khushi Shah authored 3 days ago	108a80a5		
 Adding module 3rd and Lex	Khushi Shah authored 3 days ago	1b627128		
 SNS notification subscription sent on google registration	Harshil authored 3 days ago	 c2d35310		
449630e1174c41cd328f0d55b14f1cf344083f95				

Figure 103: Sprint Commit 4

31 Jul, 2023 5 commits				
 Merge branch 'master' of https://git.cs.dal.ca/msojitra/serverless-fe-team25	Harshil authored 4 days ago	 2ab518b1		
 Edit feature is wrking fine	Harshil authored 4 days ago	174f7e20		
 Update .gitlab-ci.yml file	Harshil Kalpeshkumar Shah authored 4 days ago	 84a84b5a		
 Merge branch 'master' of https://git.cs.dal.ca/msojitra/serverless-fe-team25	Harshil authored 4 days ago	 1ad3468a		
 Added all backend code to Lambda	Harshil authored 4 days ago	1a59acdd		
28 Jul, 2023 6 commits				
 Update .gitlab-ci.yml file	Harshil Kalpeshkumar Shah authored 1 week ago	 4caccc419		
 Update .gitlab-ci.yml file	Harshil Kalpeshkumar Shah authored 1 week ago	 b26dccfb		
 Update .gitlab-ci.yml file	Harshil Kalpeshkumar Shah authored 1 week ago	 50105e85		
 Edit user done	Harshil authored 1 week ago	 e67c7c7c		

Figure 104: Sprint Commit 5

27 Jul, 2023 5 commits	
 Social media done Harshil authored 1 week ago	✓ 04e232a2  
 email password auth done Harshil authored 1 week ago	✓ 1f2ae889  
 removed .idea from remote Harshil authored 1 week ago	✓ 47414d5f  
 login sign up success Harshil authored 1 week ago	✓ 8d32c756  
 game exp Mohit Rajeshkumar Sojitra authored 1 week ago	✓ 297c780a  
08 Jun, 2023 1 commit	
 in game experience ui fixed Mohit Rajeshkumar Sojitra authored 1 month ago	✓ 73a5e298  
04 Jun, 2023 2 commits	
 Merge branch 'login-signup' into 'master'  Harshil Kalpeshkumar Shah authored 2 months ago	✓ 37a43a38  
 converted to typescript Harshil Kalpeshkumar Shah authored 2 months ago	✓ a4d0f564  
02 Jun, 2023 12 commits	
 Merge branch 'ci-cd' into 'master'  Harshil Kalpeshkumar Shah authored 2 months ago	✓ 23084d0b  

Figure 105: Sprint Commit 6

02 Jun, 2023 12 commits	
 Merge branch 'ci-cd' into 'master'  Harshil Kalpeshkumar Shah authored 2 months ago	✓ 23084d0b  
 Merge branch 'cd' into 'master'  Harshil Kalpeshkumar Shah authored 2 months ago	✗ 02a66c04  
 Merge branch 'login-signup' into 'master'  Harshil Kalpeshkumar Shah authored 2 months ago	✓ 2e907aa8  
 removed all warnings Harshil Kalpeshkumar Shah authored 2 months ago	✓ e60a2f0a  
 Login, Sign Up and verification mail working successfully. Harshil Kalpeshkumar Shah authored 2 months ago	✓ b6e283b2  
 Update .gitlab-ci.yml file Harshil Kalpeshkumar Shah authored 2 months ago	✓ 5658e3d6  
 changing directory is not required Harshil Kalpeshkumar Shah authored 2 months ago	✓ 29543020  
 changing directory is not required Harshil Kalpeshkumar Shah authored 2 months ago	✓ 68825d12  
 Automated deployment to netlify Harshil Kalpeshkumar Shah authored 2 months ago	✓ 17d5a5c8  
 login/Sign up Harshil Kalpeshkumar Shah authored 2 months ago	✓ 824ad69d  
 Login.Signup  Harshil Kalpeshkumar Shah authored 2 months ago	✓ 2e00f5e4  

Figure 106: Sprint Commit 7

Harshil Kalpeshkumar Shah authored 2 months ago

Login, Sign Up and verification mail working successfully.
Harshil Kalpeshkumar Shah authored 2 months ago

Update .gitlab-ci.yml file
Harshil Kalpeshkumar Shah authored 2 months ago

changing directory is not required
Harshil Kalpeshkumar Shah authored 2 months ago

changing directory is not required
Harshil Kalpeshkumar Shah authored 2 months ago

Automated deployment to netlify
Harshil Kalpeshkumar Shah authored 2 months ago

login/Sign up
Harshil Kalpeshkumar Shah authored 2 months ago

Login Signup
Harshil Kalpeshkumar Shah authored 2 months ago

initial commit of react project
Mohit Rajeshkumar Sojitra authored 2 months ago

01 Jun, 2023 1 commit

Initialize project using Create React App
Mohit Rajeshkumar Sojitra authored 2 months ago

Figure 107: Sprint Commit 8

Meeting Logs



Figure 108: Meeting 1



Figure 109: Meeting 2

May 31, 2023

KS Khushi Hirenbai Shah 05-31 10:23 p.m.
Team, below are the features:

Features:
1 - Harshil Shah
2 - ...
See more

Reply

Meeting ended: 19m 54s

Reply

Figure 110: Meeting Minutes 1

May 24, 2023

HS Harshil Kalpeshkumar Shah 05-24 11:45 a.m.

Confirmed
You are scheduled with Bharat Shankaranarayanan.

CSCI5410-SUMMER-PROJECT
4:45pm - 5:00pm, Wednesday, May 24, 2023
Atlantic Time
Web conferencing details to follow.
hs@dal.ca, ed743899@dal.ca, kh482021@dal.ca, mansi.vaghaniya@dal.ca, mhs546676@dal.ca

A calendar invitation has been sent to your email address.

Reply

Meeting ended:

Attendance report
Click here to download attendance report

Reply

BS Bharat Shankaranarayanan 05-24 4:35 p.m. Edited
Hi SDP25, is it fine to move the meeting for your group to 5.15?

7 replies from you, Bharatwaaj, and Harshil Kalpeshkumar

Reply

Meeting started

4 replies from you

Reply

Figure 111: Meeting 3 with meeting minutes

May 18, 2023

Harshil Kalpeshkumar Shah 05-18 9:42 a.m. Edited
We can meet virtually, if everyone want.
Let's decide using this:

<https://lettucemeet.com/l/2Yw0r>

 **LettuceMeet - Easy Group Scheduling**
The easiest way to schedule group meetings.
lettucemeet.com

Reply

Mansi Vaghasiya 05-17 12:17 p.m.
Hi,
Let's schedule a meeting to discuss about the project, as professor has posted the definition.
Kindly share your availabilities in this week.

18 replies from you, Harshil Kalpeshkumar, Edwin, and Mohit Rajeshkumar

Reply

Meeting started

5 replies from Harshil Kalpeshkumar

Reply

Figure 112: Meeting 4 with meeting minutes

Minutes of Meeting:

May 31, 2023

Khushi Hirenbai Shah 05-31 10:23 p.m.
Team, below are the features:

Features:
 1 - Harshil Shah
 2 - Harshil Shah
 3 - Khushi Shah
 4 - Mansi
 5 - Mohit
 6 -
 7 - Edwin A
 8 -
 9 -
 10 -
[See less](#)

Reply

Figure 113: MOM 3

Friday 20, 2023

KS
Khushi Hirenbai Shah 05-24 6:40 p.m. Edited
Hi Team,
I have a question.

We all have different GCP and AWS accounts
Now while we create features individually in our own accounts, how will we share code with each other?
See less

HS
Harshil Kalpeshkumar Shah 05-24 9:54 p.m. Edited
We don't have to share the code, I guess
For GCP we can use something like IAM to create users to collaborate
For AWS we have to rely on the person who is implementing the feature and have to use the public url/API gateway
I am not 100% sure

Mohit Sojitra 05-26 10:43 p.m.
If we use IAM users anyway, the credit of only root account will get deducted.

Reply

Figure 114: MOM 4

Screenshots of the meeting

00:54

Khushi Hirenbai Shah, Mohit Sojitra

Take control Pop out Chat People Raise React View Apps More Camera Mic Share Leave

KS
Mohit Sojitra

5410 SDP 25

File Edit View Insert Format Tools Extensions Help

Search Bing

Worksheet

Module	Module Owners
1. User Authentication	Harshil Shah
2. User Profile Management	Harshil Shah
3. Team Management	Khushi Shah
4. Trivia Game Lobby	Mansi Vaghasiya
5. In-game Experience	Mohit Sojitra
6. Leaderboards	Mohit Sojitra
7	
8	
9. Automated Question Tagging	Mansi Vaghasiya
10. Virtual Assistance	Khushi Shah

There were no changes in the allocation of work within the team members.
Paragraph on Individual Component:

Figure 115: Meeting 1

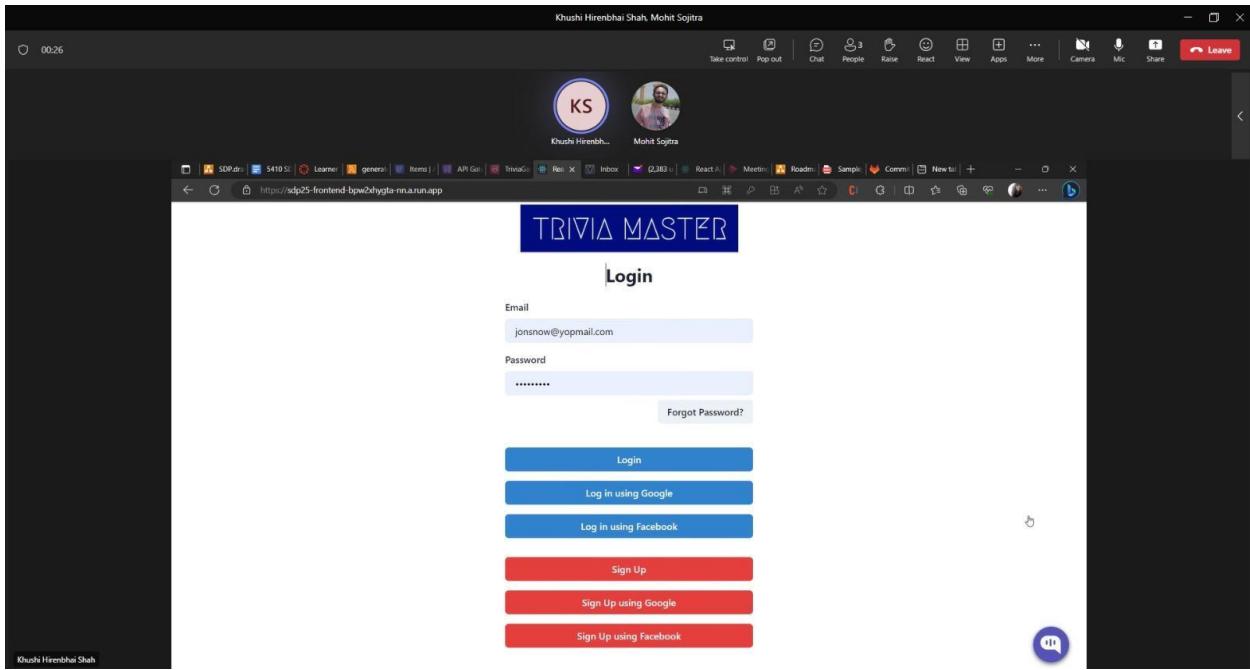


Figure 116: Meeting 2

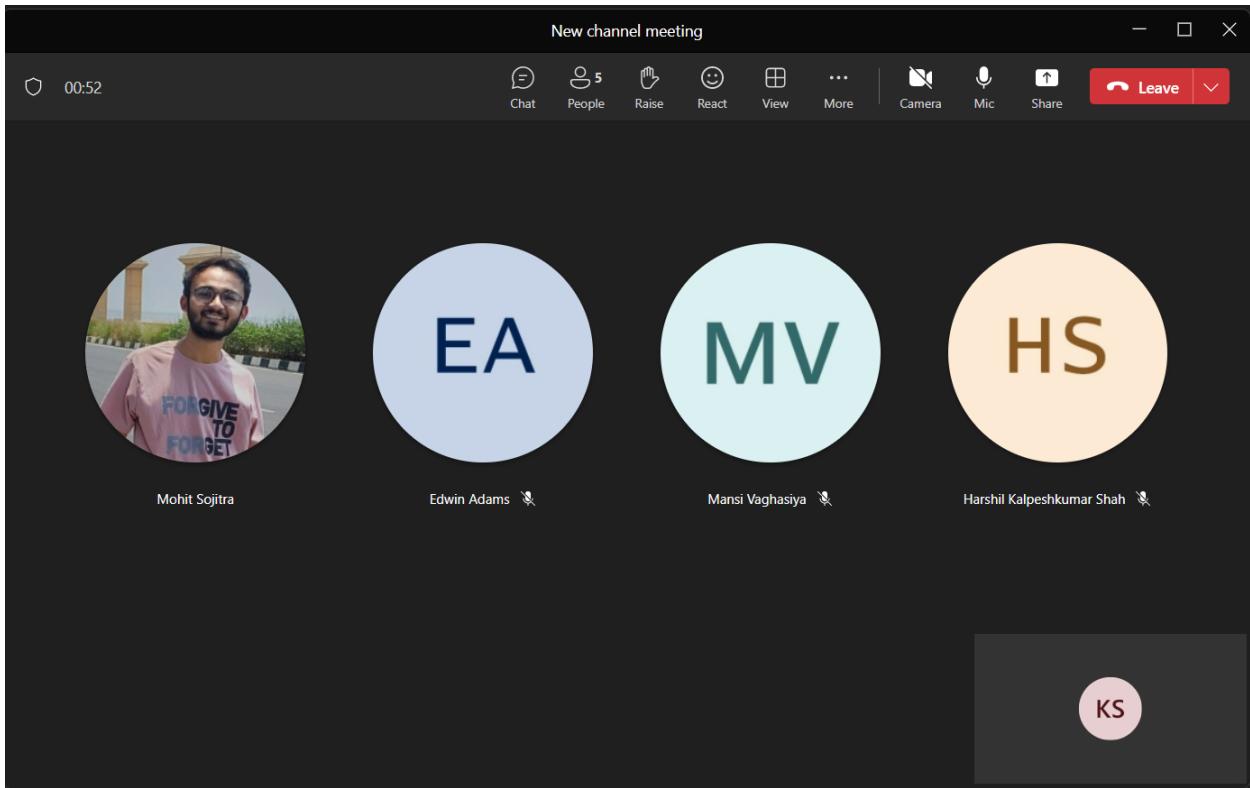


Figure 117: Meeting 3

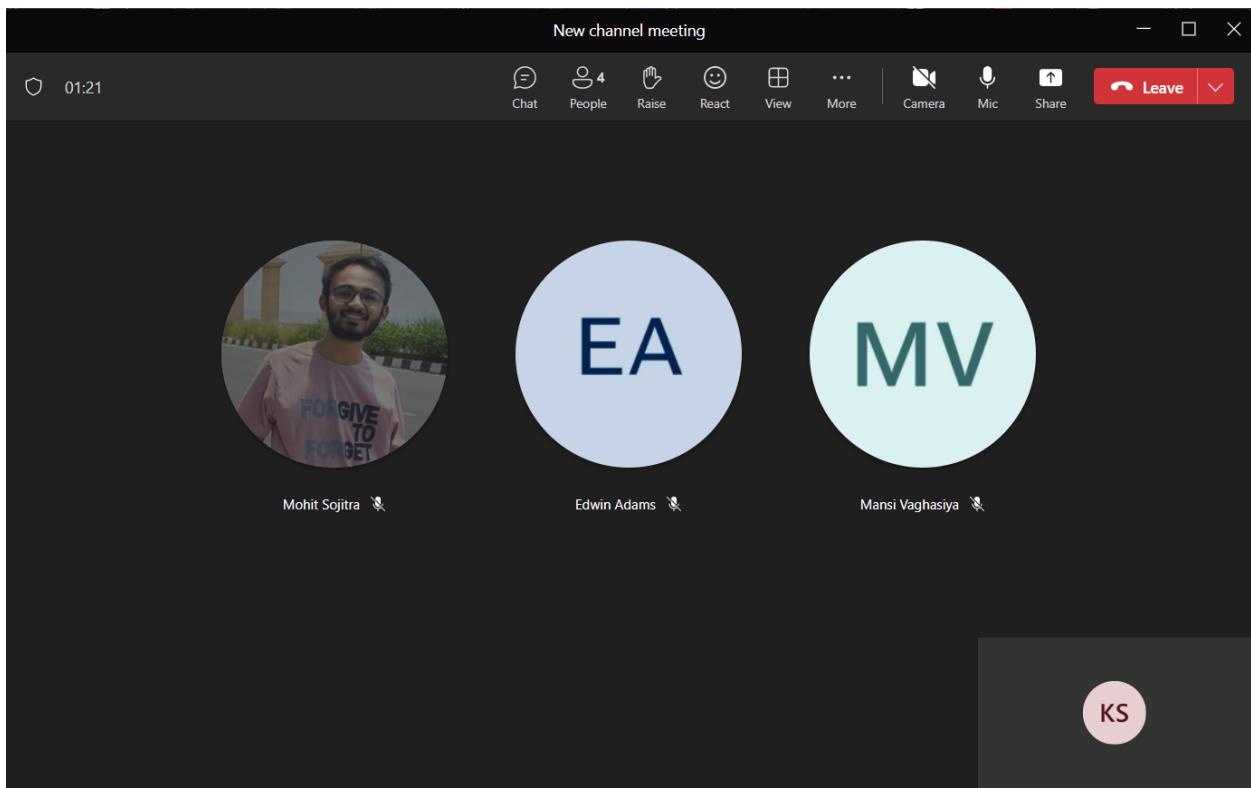


Figure 118: Meeting 4

Individual Contribution

Harshil Shah

Features built: User Authentication, User Management

For both the features, there are no deviations. The final output is the same as planned for requirements.

Khushi Shah

Features built: Team management, Virtual Assistant

For both the features, there are no deviations. The final output is the same as requirements and expectations.

Mohit Sojitra

Features built: Ingame experience, leaderboard

For both the features, there are no deviations. The final output is the same as requirements and expectations.

Mansi Vaghasiya

Features built: Game Lobby, Question tagging

For both the features, there are no deviations. The final output is the same as requirements and expectations.

Edwin Adams

Novelty

Harshil Shah

The novel aspect of my features lies in its independence and microservices architecture, which enables each feature to function autonomously.

Utilizing stateless communications with APIs, the modules are seamlessly integrated, allowing smooth interactions between them.

What sets this architecture apart is its multi-cloud multi-account approach, where each feature is deployed individually on distinct AWS accounts, yet collaborates harmoniously as a unified system.

This collaboration is facilitated through robust authentication methods, implemented via API Gateways, ensuring secure and efficient data exchange.

Khushi Shah

Code comments are present for both the modules, Team management and Virtual Assistance. For virtual assistance, the users can create multiple chats and can save them and can refer to them later on.

For a smoother experience of the virtual assistance, I have created the new AWS free tier account which gives me more permissions and makes the assistance experience seamless. But my database was present in the AWS Academy provided by the course. So to integrate both the things together I created an api in a new account which calls another account and fetches the data from the DB and returns the result.

Mohit Sojitra

Used serverless application development framework through the development to protect the architecture from accidental loss.

For a smoother experience of the virtual assistance, I have created the new AWS free tier account which gives me more permissions and makes the assistance experience seamless.

Mansi Vaghasiya

Make is commented thoroughly in the development.

For exploring the possibilities with comprehend, I created a free tier account.

In game lobby, all the user will be redirected to their In game experience after start time of the game approaches

Edmin Adams

Cloud Run Hosted URL

[React App \(sdp25-frontend-bpw2xhygta-nn.a.run.app\)](#)

References

- [1] "Amazon Cognito," Amazon, 2023. [Online]. Available: <https://aws.amazon.com/cognito/>.
- [2] "Amazon Lambda," Amazon, 2023. [Online]. Available: <https://aws.amazon.com/lambda/>.
- [3] "Chatbot run lambda function remotely tutorial," Amazon, 2023. [Online]. Available: <https://docs.aws.amazon.com/chatbot/latest/adminguide/chatbot-run-lambda-function-remotely-tutorial.html>.
- [4] "Google Sheets: Online Spreadsheet Editor," Google.ca, 2023. [Online]. Available: <https://www.google.ca/sheets/about/>.
- [5] "Create buckets," Google Cloud, 2023. [Online]. Available: <https://cloud.google.com/storage/docs/creating-buckets>.
- [6] "Looker Studio Overview," Google.com, 2023. [Online]. Available: <https://lookerstudio.google.com/u/0/navigation/reporting>.
- [7] "How to connect Google Cloud Functions to Google Sheets," Actiondesk.io, 2023. [Online]. Available: <https://www.actiondesk.io/google-sheets-integration/google-cloud-functions>.
- [8] "Firebase authentication," Firebase, 2023. [Online]. Available: <https://firebase.google.com/docs/auth>.
- [9] "Develop Chatbot using Amazon Lex and integration with React JS web application," Medium, 2021. [Online]. Available: <https://medium.com/@ideepaksharma/develop-and-integrate-amazon-lex-chatbot-into-website-47f50fe4cfed>.
- [10] "Flowchart maker and online diagram software," Diagrams.net, 2023. [Online]. Available: <https://app.diagrams.net/>.
- [11] "Amazon API Gateway," Amazon, 2023. [Online]. Available: <https://aws.amazon.com/api-gateway/>.
- [12] "AWS QuickSight," Amazon, 2023. [Online]. Available: <https://aws.amazon.com/quicksight/>.

NOTE: We have added the access date in the word citation but it is not showing up here

