Real-Time Sports Scoreboard using AWS AppSync and DynamoDB

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY
(22SDCS07A)

by

B. SHAMITHA SRI

(2210030478)

Under the esteemed guidance of

Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075

April 2025

K L Deemed to be UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is Certified that the project entitled **"Real-Time Sports Scoreboard using AWS AppSync and DynamoDB"** which is a experimental &/ Simulation work carried out by B.SHAMITHA SRI (2210030478), in partial fulfillment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi	**Dr. Arpita Gupta**
Course Coordinator	**Head of the Department**

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

	Page No.
1. Introduction	1
2. AWS Services Used as part of the project	2
3. Steps involved in solving project problem statement	5
4. Stepwise Screenshots with brief description	6
5. Learning Outcomes	18
6. Conclusion	19
7. References	20

1. INTRODUCTION

In the modern sports industry, real-time score updates are essential for enhancing fan engagement and providing instant access to game statistics. Traditional systems often rely on REST APIs with polling mechanisms, which introduce latency and consume excessive server resources. To address these limitations, AWS AppSync and Amazon DynamoDB offer a scalable, low-latency solution for real-time sports scoreboards [1].

AWS AppSync is a managed GraphQL API service that enables applications to fetch, update, and subscribe to live data seamlessly. Unlike traditional REST-based architectures, GraphQL allows clients to retrieve only the necessary data, improving efficiency. AppSync's subscription feature ensures that users receive instant updates without manually refreshing their screens, making it an ideal choice for live sports applications [2].

Amazon DynamoDB, a serverless NoSQL database, provides high-performance data storage for sports scores, team details, and player statistics. With DynamoDB Streams, data modifications trigger real-time events, allowing applications to process and propagate updates instantly. AWS Lambda integrates with these streams to handle game events dynamically and push updates to AppSync subscribers [3].

By utilizing GraphQL subscriptions with AWS AppSync, sports applications can achieve real-time synchronization, ensuring fans receive updates as soon as they happen. This architecture eliminates the inefficiencies of traditional polling-based methods while reducing server costs. It also enhances scalability, as AppSync and DynamoDB can support millions of concurrent users without performance degradation [4].

Real-world implementations of AWS AppSync and DynamoDB have been successfully used in various live sports platforms. Companies leverage this serverless architecture to provide seamless and real-time updates on web and mobile apps, ensuring an engaging user experience. The combination of low-latency data delivery, scalability, and cost-effectiveness makes AWS AppSync and DynamoDB an ideal choice for building a next-generation sports scoreboard system [5].

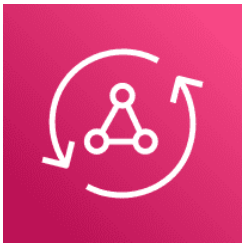
2. AWS SERVICES USED AS PART OF THE PROJECT

To implement Real Time Sports Scoreboard using AWS AppSync and DynamoDB, several AWS services have been utilized to ensure a seamless, efficient, and scalable solution. Below are the key AWS services used in the project:

AWS AppSync

AWS AppSync is the backbone of the real-time data delivery system in this project. It is a managed service that uses GraphQL APIs to allow clients to query and subscribe to specific pieces of data. AppSync supports GraphQL subscriptions, enabling the system to push real-time updates directly to clients whenever data in the backend changes, without needing continuous polling.

AppSync integrates seamlessly with Amazon DynamoDB and AWS Lambda, allowing live data changes to be reflected instantly on web or mobile client interfaces. It also provides built-in support for caching, authorization (via AWS Cognito or IAM), and offline access, which enhances overall system performance and reliability [1].



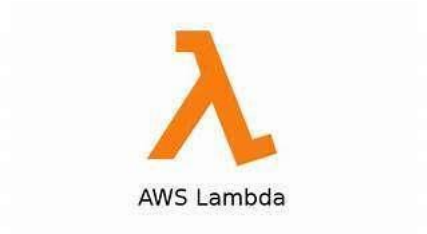
Amazon DynamoDB

Lambda DynamoDB is a fully managed, highly available NoSQL database service used to store all real-time sports data, including match scores, team info, and event logs. Its ability to handle millions of read/write requests per second makes it suitable for high-demand use cases like live score updates during peak match ours.

This project utilizes DynamoDB Streams to capture changes to the database in real time. These streams allow the application to react to new updates—such as a goal being scored—by triggering further processing using AWS Lambda functions [2].

AWS Lambda

AWS Lambda is used to perform backend logic in response to database changes or API events. For example, when a DynamoDB stream detects that a score has changed, a Lambda function processes the update and forwards it to AppSync for broadcasting.Create and edit Lambda functions.Lambda's event-driven architecture is ideal for serverless real-time applications, as it reduces operational overhead while ensuring fast response times. Additionally, Lambda functions can be integrated with third-party APIs (e.g., sports data providers) to fetch external match data and update the system [3].



AWS IAM (Identity and Access Management)

In this project, AWS Identity and Access Management (IAM) is used to define security policies and control access to AWS services, ensuring a secure and well-managed real-time sports scoreboard system. IAM roles are assigned to AWS Lambda functions to grant necessary permissions for accessing Amazon DynamoDB, AppSync, and CloudWatch Logs, enabling seamless data processing and monitoring. Additionally, IAM roles and policies are configured for users and developers to manage and deploy Lambda functions securely, ensuring that only authorized personnel can modify or execute critical backend logic [7].

Amazon CloudWatch

In this project, Amazon CloudWatch is utilized for monitoring and logging to ensure the seamless operation of the real-time sports scoreboard system. It tracks AWS Lambda function executions and performance, providing insights into execution time, memory usage, and error rates. CloudWatch also collects logs and error messages, which are essential for debugging issues in AppSync resolvers, Lambda functions, and DynamoDB triggers. Additionally, alarms and notifications are configured to detect failures or performance bottlenecks, ensuring that real-time score updates remain accurate and responsive. Set up alarms and notifications in case of failures or performance issues [5]

By integrating CloudWatch with other AWS services, the project maintains a highly efficient, modular, and scalable architecture, enabling effective monitoring and management of multiple serverless functions within the system.

3. STEPS INVOLVED IN SOLVING PROJECT PROBLEM STATEMENT

To implement a Real-Time Sports Scoreboard using AWS AppSync and DynamoDB, follow these steps:

1. Set Up DynamoDB for Data Storage:
    - Create a DynamoDB table to store match scores, teams, and timestamps [1].
    - Enable DynamoDB Streams to track real-time data changes [2].
2. Configure AWS AppSync:
    - Navigate to AWS AppSync in the AWS Management Console.
    - Create a GraphQL API and define the schema for querying and updating scores [3].
    - Link DynamoDB as a data source and configure resolvers [4].
3. Create AWS Lambda for Data Processing:
    - Develop a Lambda function to process real-time score updates.
    - Use DynamoDB Streams to trigger Lambda when a score changes [5].
    - Deploy the function and ensure it has the required IAM permissions [6].
4. Use AWS AppSync Subscriptions for Real-Time Updates:
    - Configure GraphQL subscriptions to notify clients of score changes [7].
    - Ensure WebSocket connections are enabled for instant updates [8].
5. Deploy AWS Lambda Layers for Code Reusability:
    - Write reusable functions for score validation and formatting.
    - Package them into a Lambda Layer and attach them to the function [9].
6. Monitor and Optimize with Amazon CloudWatch:
    - Enable CloudWatch logs to track Lambda executions and API performance.
    - Set up alarms for failures or performance issues.
7. Integrate Frontend for Live Score Display:
    - Build a web or mobile application that queries AppSync for scores.
    - Use GraphQL subscriptions to receive real-time updates.

4. STEPWISE SCREENSHOTS WITH BRIEF DESCRIPTION

Step 1: Log in to AWS Management Console

- Search for "AppSync" and open the service.
- Create the GraphQL API.

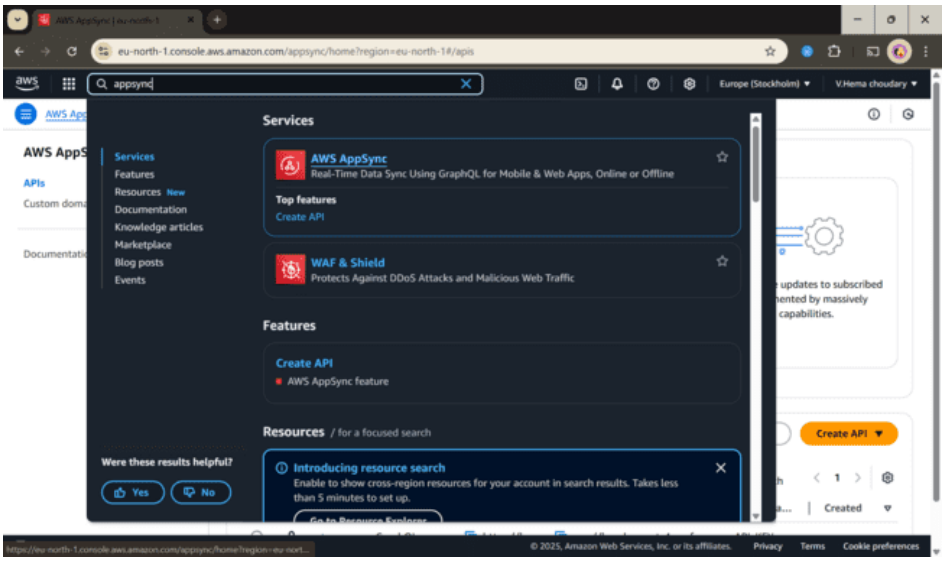


Fig 4.1: Open Aws AppSync in the Console.

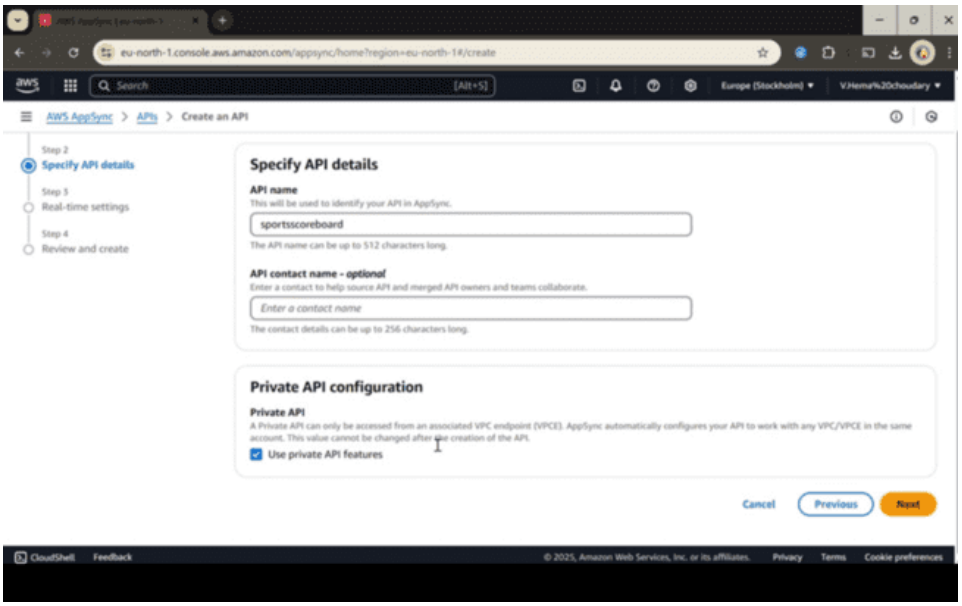


Fig 4.2 : Create an AppSync API.

Step 2: Define GraphQL Schema in AppSync Console

Paste this code in the Schema of AppSync and save Schema

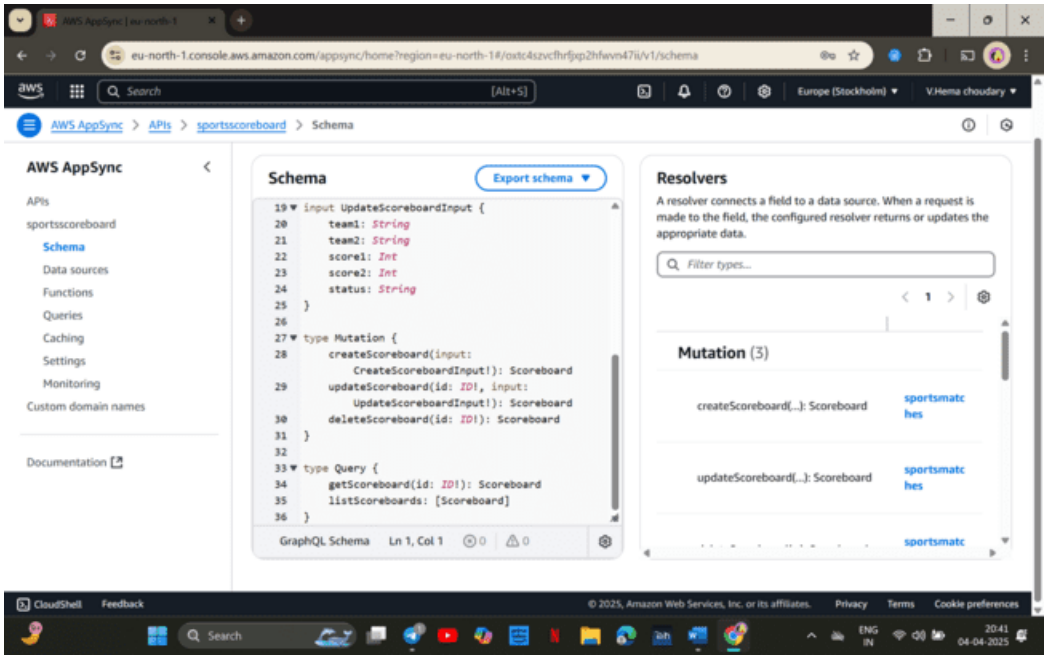


Fig 4.3 : Define GraphQL Schema in AppSync Console.

Step 3: Create DynamoDB Table for Match Data

Open the DynamoDB service & create a table in it (i.e sportsmatches) and create items for the data.

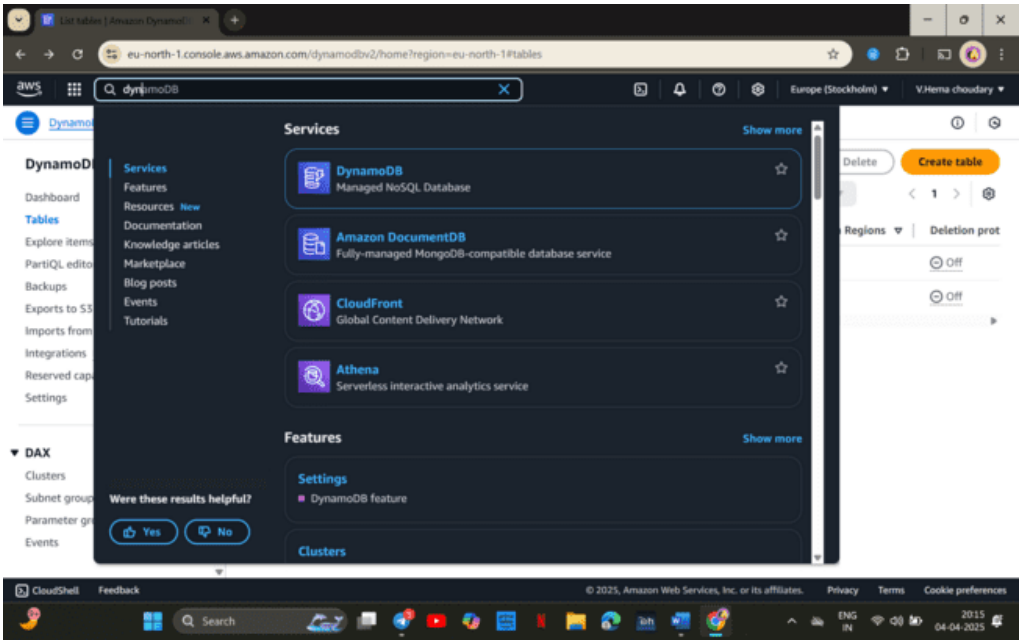


Fig 4.4 : Search DynamoDB service in the Console.

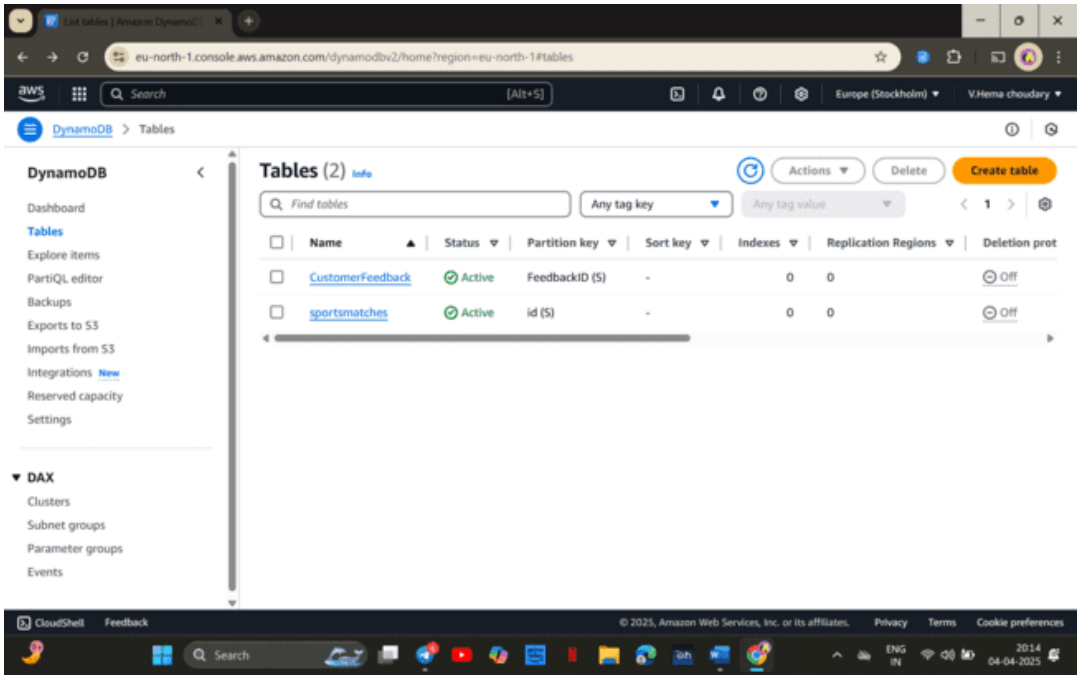


Fig 4.5 : Create a table in the DynamoDB.

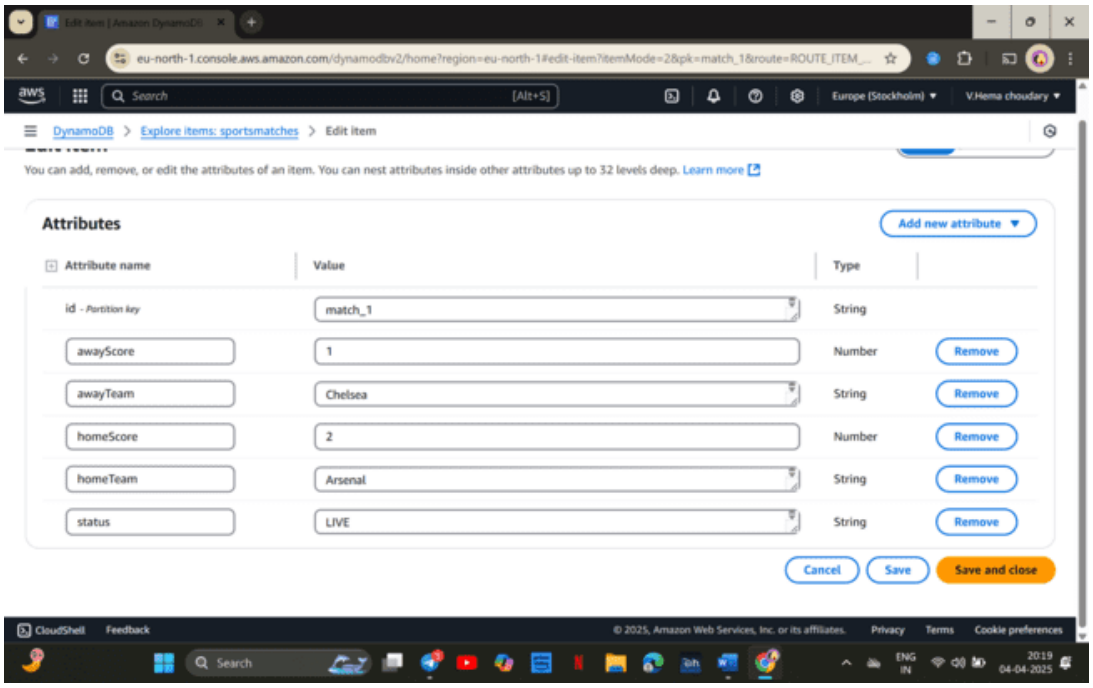


Fig 4.6 : Create New Attributes in DynamoDB.

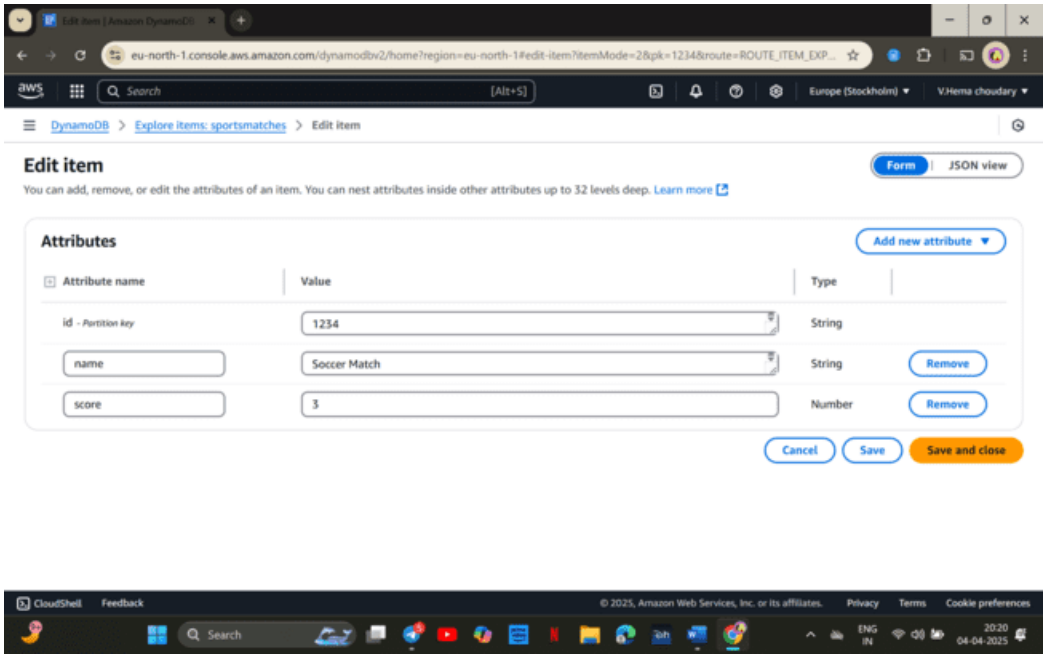


Fig 4.7 : Create New Attribute in DynamoDB.

Step 4: Add DynamoDB as a Data Source in AppSync

- Go to AWS Console → Click "AppSync".
- Click "Datasources".
- Enter DataSource and name it as sportsmatches.
- Click "Create" to finalize the Datasource.

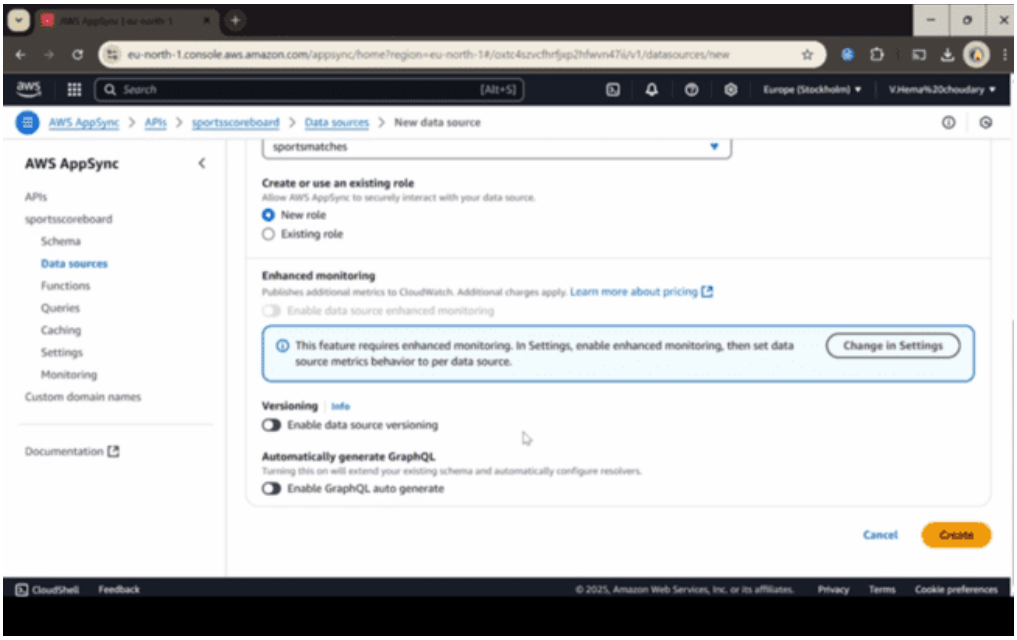


Fig 4.8 : Create a New Data Source in the AppSync.

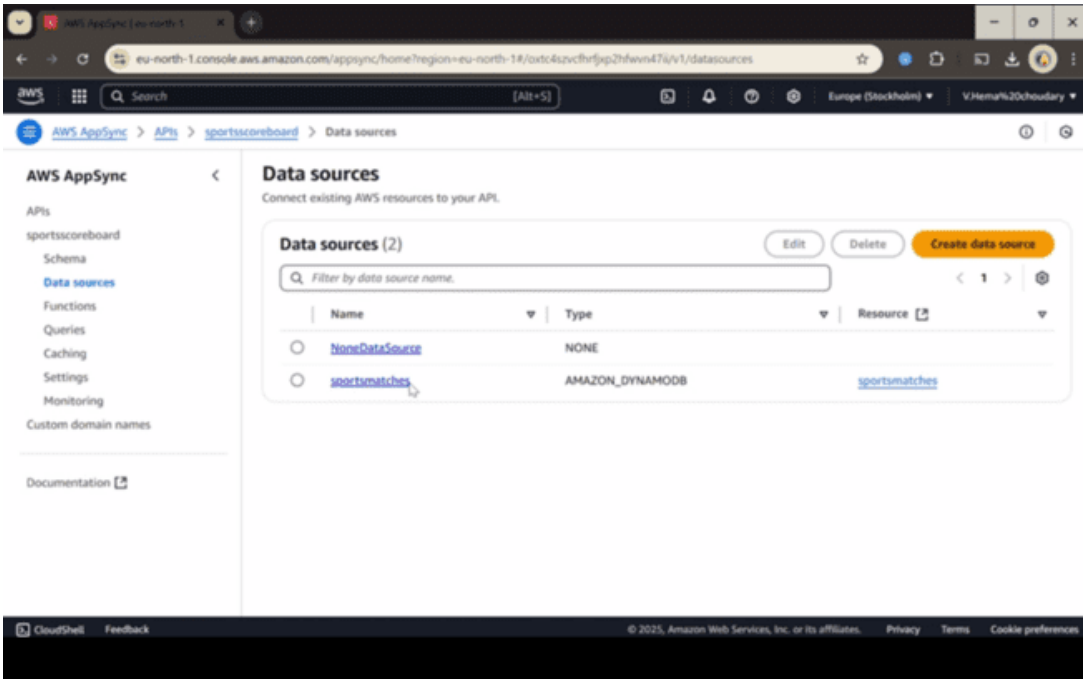


Fig 4.9 : Sportsmatches Data Source is created.

Step 5: Configure Query Resolvers for Data Retrieval

- Go to AWS AppSync → Click "Schema" in the sidebar.
- Go to Queries to attach in the Resolver.
- Paste the code in the Response-Request Mapping Template of the Queries.
- Click "Create" to finalize the function.

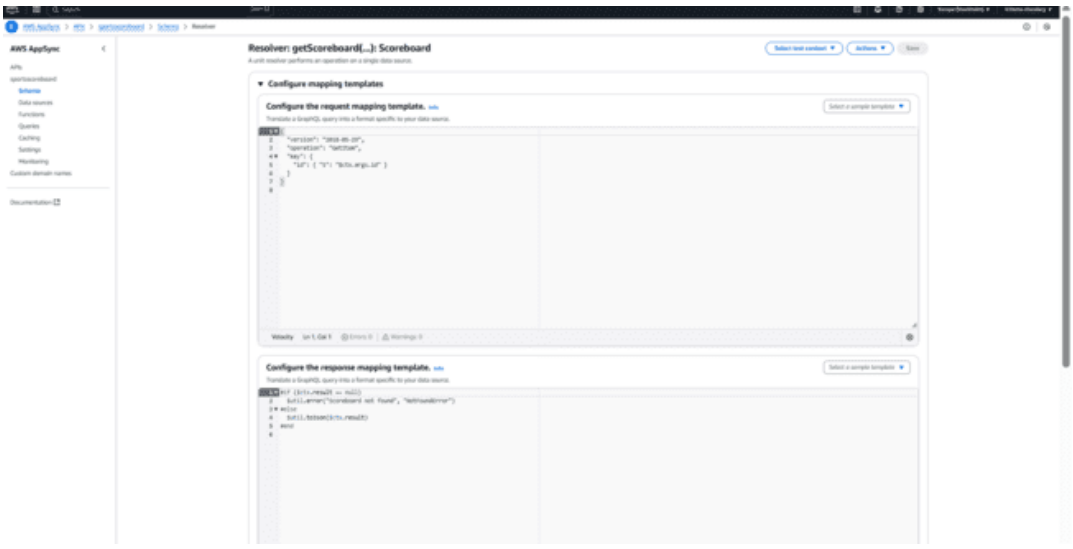


Fig 4.10 : The request and response mapping template for getScoreboard.

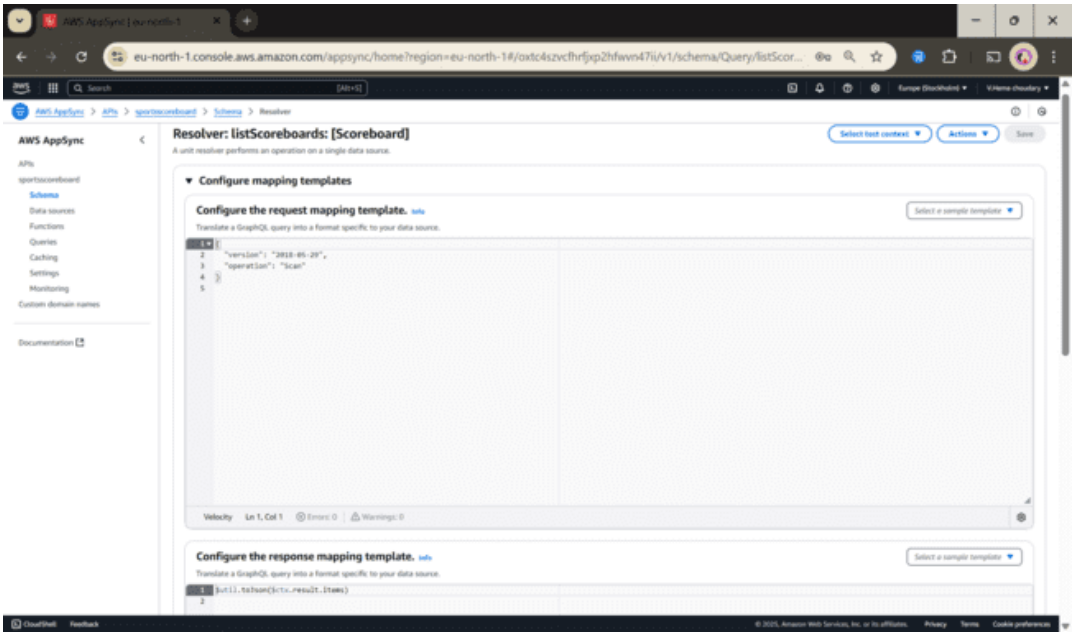


Fig 4.11 :The request and response mapping template for listScoreboard.

Step 6: Paste the code in the Response-Request Mapping Template of the Mutations

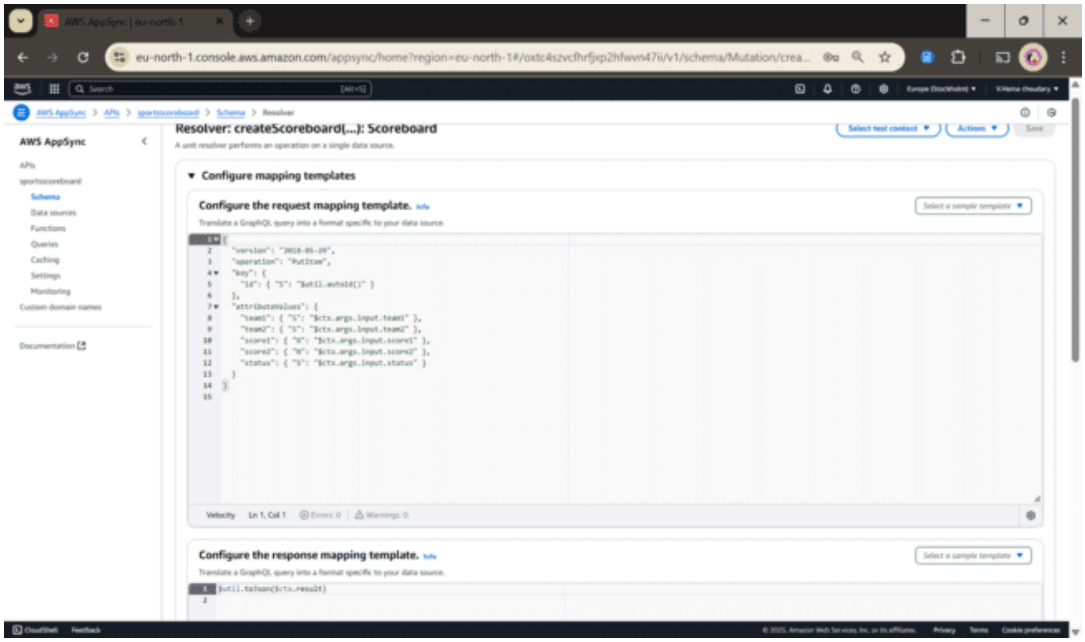


Fig 4.12 : The request and response mapping template for Mutations.

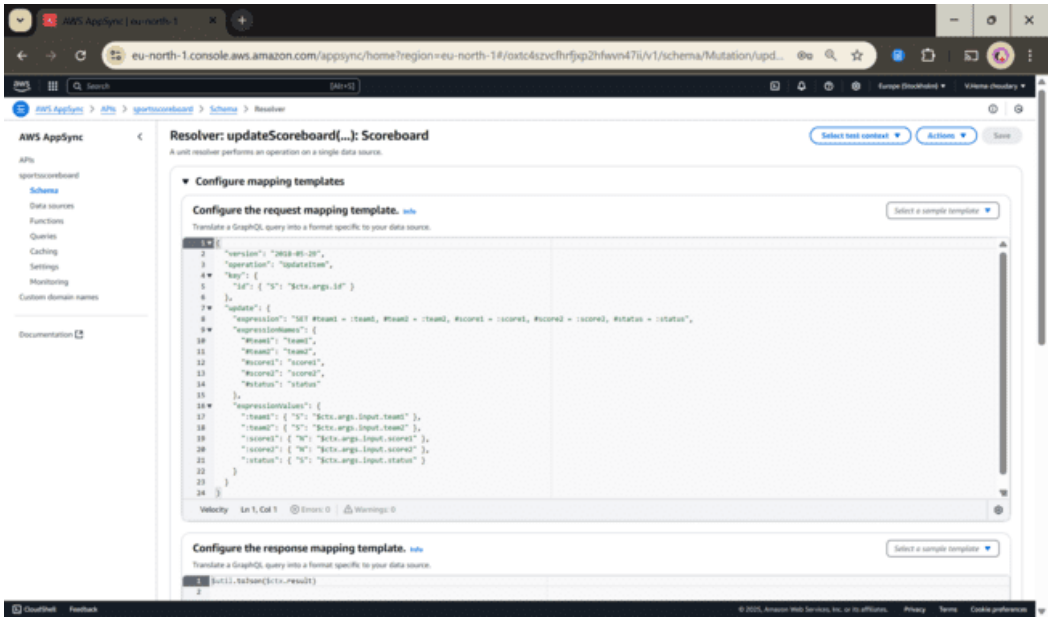


Fig 4.13 :The request and response mapping template for Updatescoreboard.

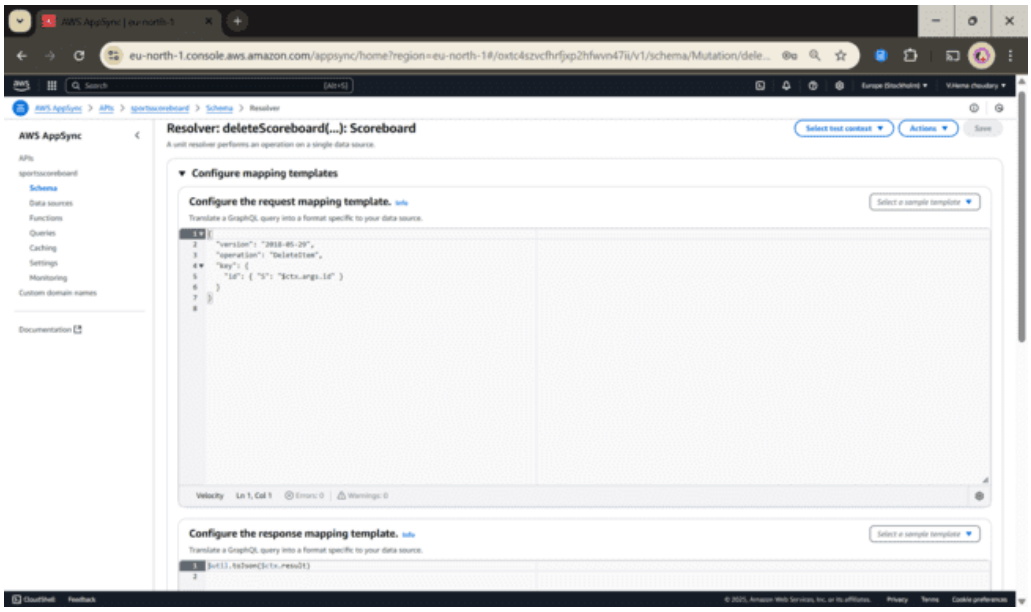


Fig 4.14 :The request and response mapping template for deletescoreboard.

Step 7: Create AWS Lambda Function for Score Updates

- Open Lambda Service in AWS Console.
- Click "Create a Function".
- Name it as UpdateScoreboards and click Create

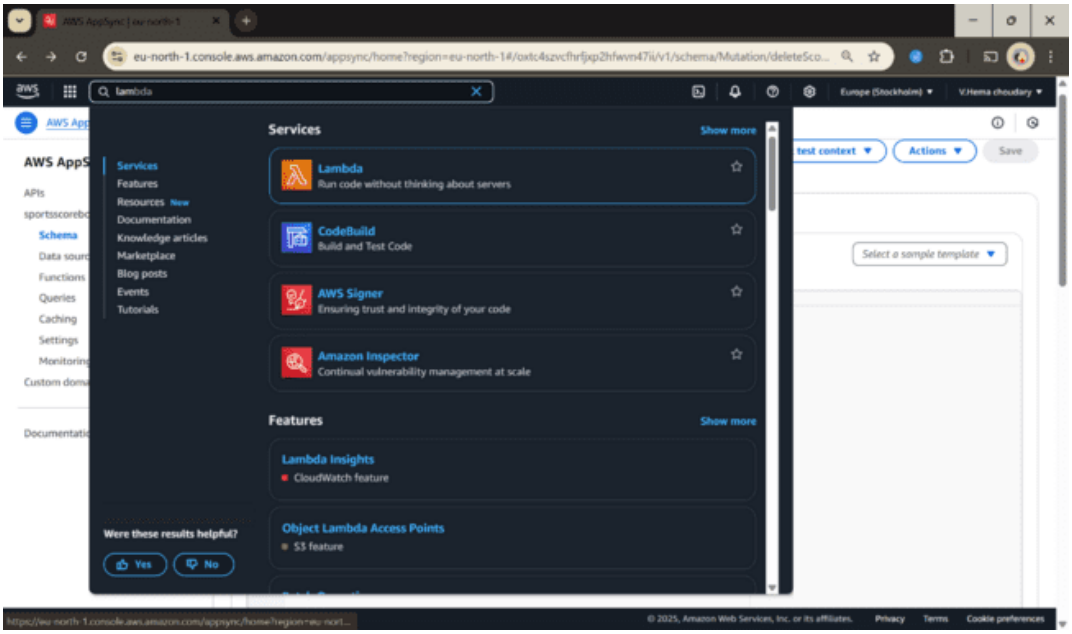


Fig 4.15 : Open Lambda Service from the Console.

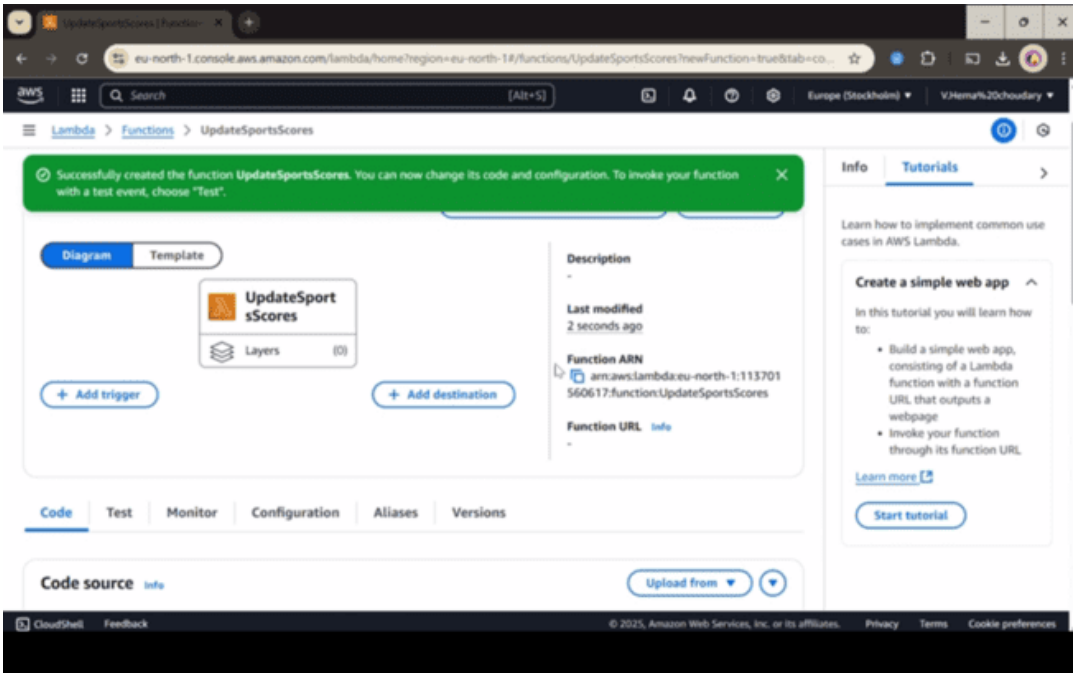


Fig 4.16 : Create a function in the Lambda.

Step 8: Test GraphQL Queries in the AppSync Console

Do a sample test with a simple Query.

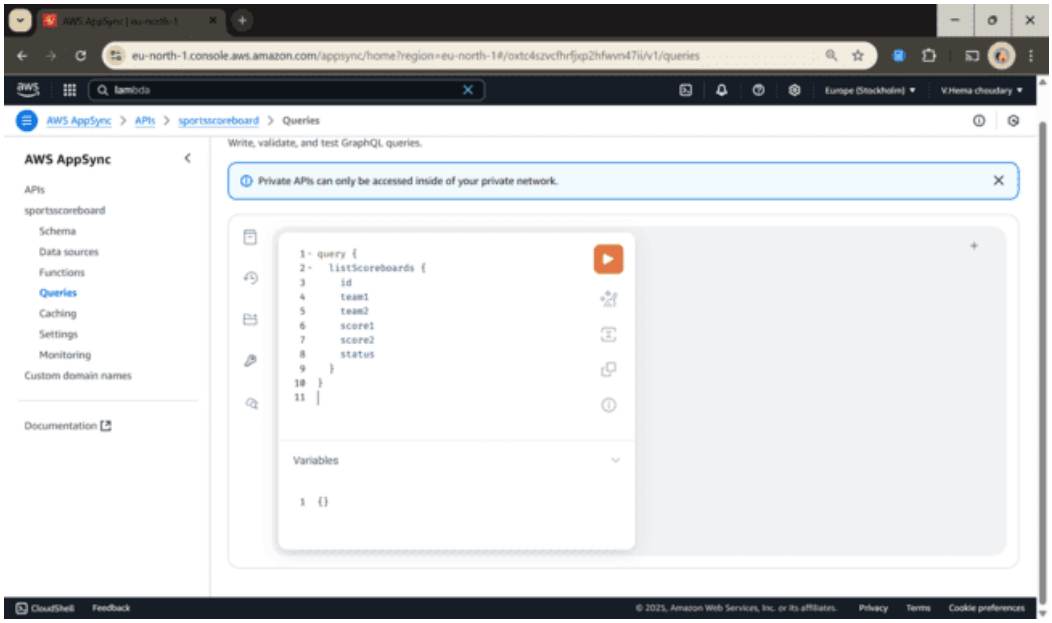


Fig 4.17 : Testing the API with a Query

Step 9: Integrate GraphQL API with React Frontend Using Amplify

- Now create a front end for this by installing amplify and create a project in vscode.
- Now Connect the Graphql API with the frontend.

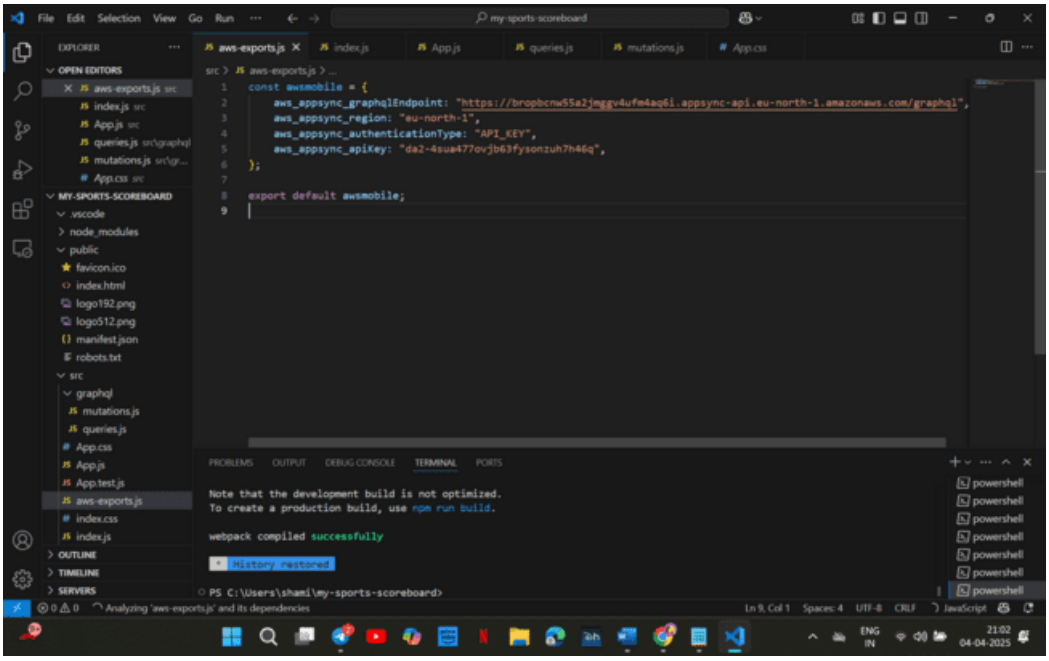


Fig 4.18: Connecting Frontend with GraphQL API.

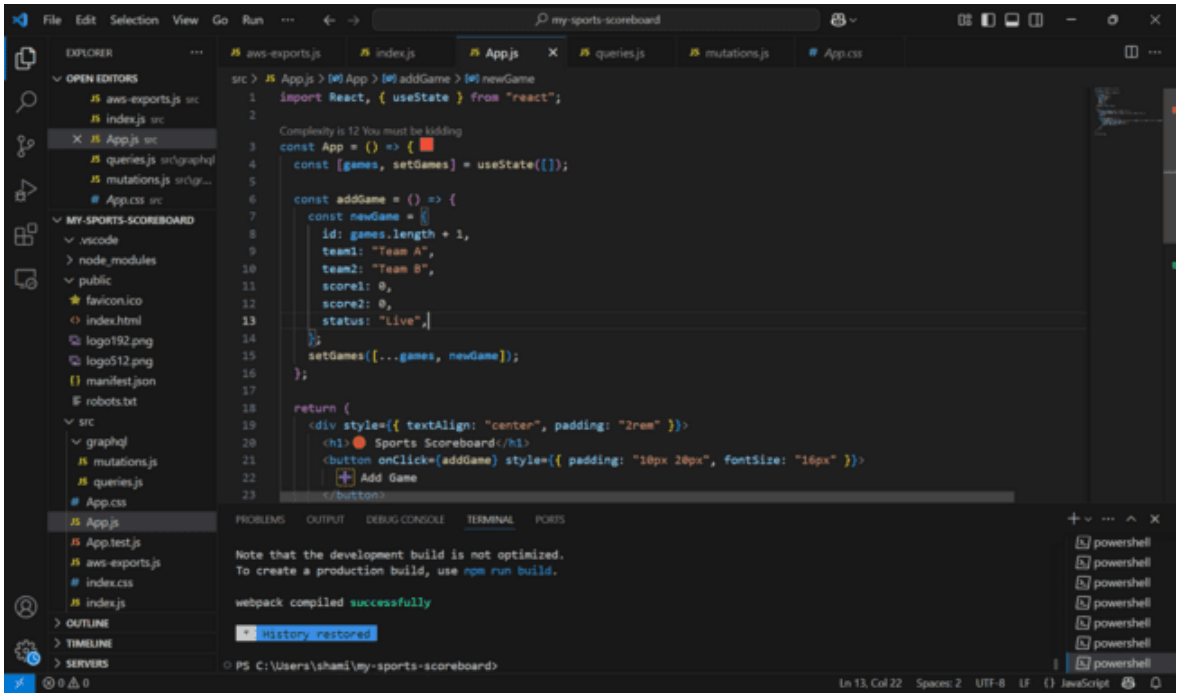


Fig 4.19 : Code for the app.js.

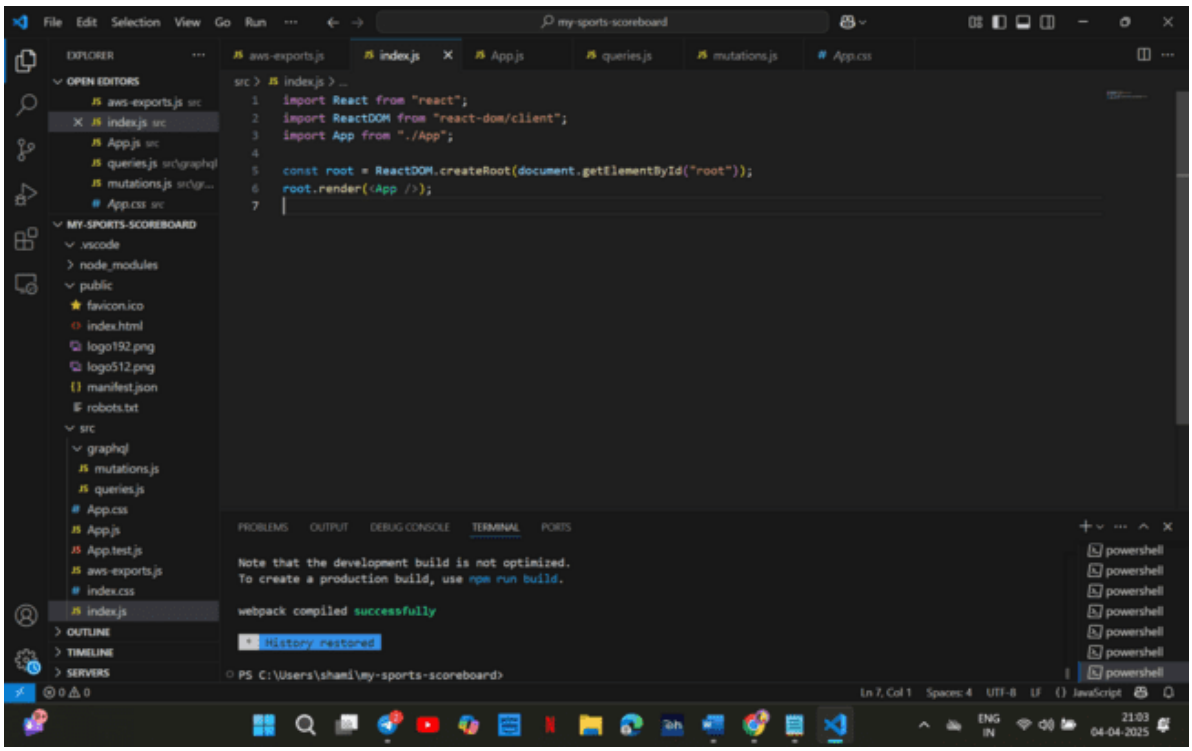


Fig 4.20 : Code for index.js.

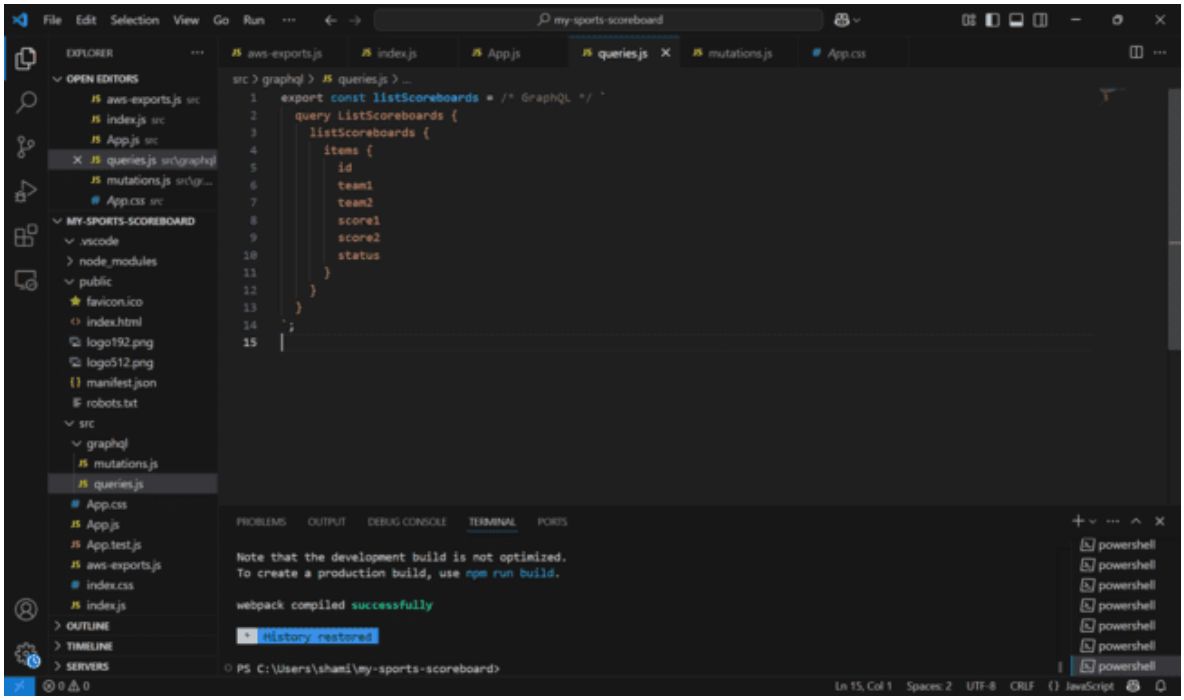


Fig 4.21 : Code for the queries.

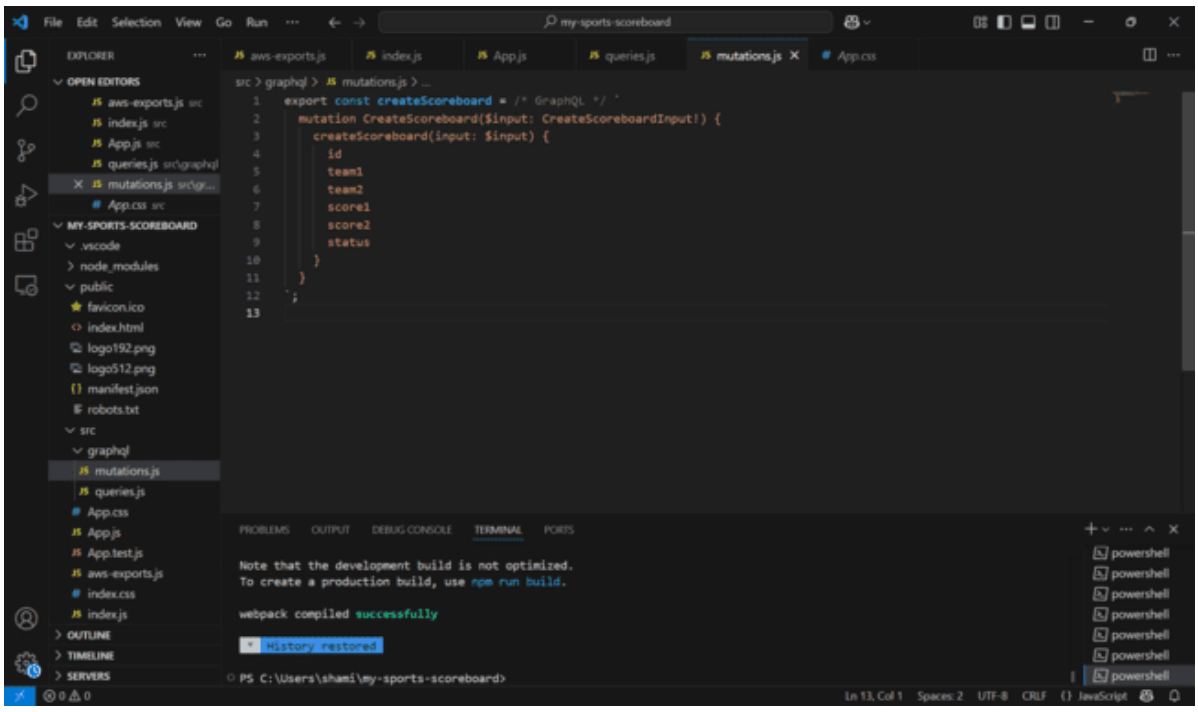


Fig 4.22 : Code for Mutations.

Step 10: Display Final Output of Live Sports Scoreboard

Here is the Final Output of the Project i.e Sports Scoreboard .

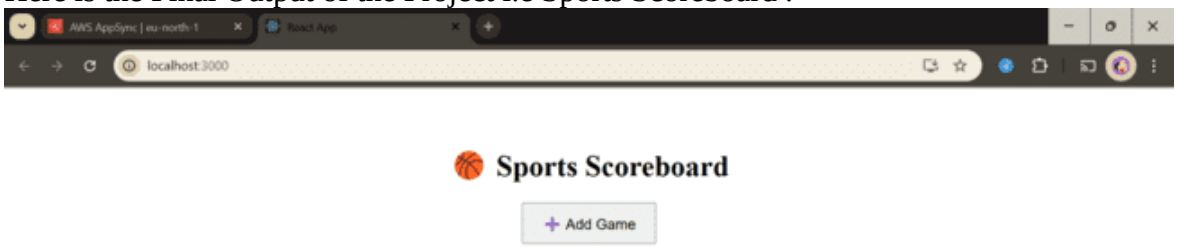


Fig 4.23 : Final Output of the project.

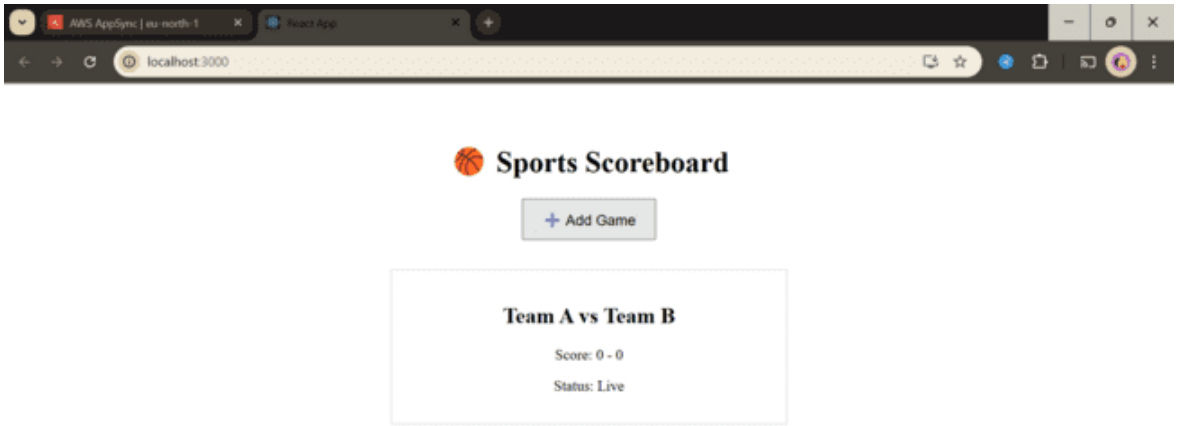


Fig 4.24 : Live Score of Team A vs Team B.

5. LEARNING OUTCOMES

- **Understanding AWS AppSync and DynamoDB Integration:** Gained hands-on experience in building real-time applications using AppSync with DynamoDB as the backend to provide low-latency data access and real-time updates via GraphQL [1].
- **Mastery of Real-Time Data Flow with Subscriptions:** Learned how to implement GraphQL subscriptions in AppSync to push real-time score updates to connected clients, ensuring a dynamic user experience [2].
- **Lambda Function and Layer Utilization:** Understood how AWS Lambda Layers promote code reuse, enhance modularity, and improve deployment efficiency when used to process and validate match data [3].
- **Efficient Event-Driven Architecture:** Explored event-driven programming using DynamoDB Streams to trigger Lambda functions for processing live game data automatically [4].
- **Security Best Practices with IAM:** Learned how to define and apply IAM roles and policies to restrict and manage access between AWS services securely within the scoreboard system [5].
- **Monitoring with Amazon CloudWatch:** Developed skills to monitor Lambda executions, set up alarms, and debug errors using CloudWatch metrics and logs for better observability and performance tuning [6].
- **Frontend Integration Using GraphQL APIs:** Gained experience in consuming GraphQL APIs from a frontend application and implementing real-time listeners for live scoreboard updates [7].
- **Optimized Serverless Workflow:** Improved development workflow by using tools like the AWS Management Console, CLI, and Cloud9, streamlining the deployment and debugging of serverless functions [8].
- These outcomes reflect a comprehensive understanding of building a scalable, secure, and real-time sports scoreboard system using AWS serverless technologies.

6. CONCLUSION

The Real-Time Sports Scoreboard project showcases the power and flexibility of AWS serverless architecture in building scalable, event-driven applications. By leveraging AWS AppSync, DynamoDB, and Lambda, the system ensures real-time data updates, seamless performance, and low-latency responses—essential for live sports tracking. GraphQL subscriptions enabled real-time push notifications to clients, enhancing user engagement and experience. Security and operational efficiency were strengthened through IAM policies, Lambda Layers, and CloudWatch monitoring, ensuring secure access, code reusability, and robust observability. Additionally, the use of DynamoDB Streams for event-driven automation minimized manual intervention and improved system responsiveness. Overall, this project offered practical experience in developing a real-time cloud-native application while deepening the understanding of serverless best practices, allowing for future scalability and adaptability across various real-time data use cases beyond sports scoreboards.

7. REFERENCES

[1] Amazon Web Services. Getting started with AWS AppSync. Available at: https://docs.aws.amazon.com/appsync/latest/devguide/what-is-appsync.html

[2] Amazon Web Services. Real-time Data with GraphQL Subscriptions. Available at: https://docs.aws.amazon.com/appsync/latest/devguide/graphql-subscriptions.html

[3] Amazon Web Services. Using AWS Lambda with AWS AppSync. Available at: https://docs.aws.amazon.com/lambda/latest/dg/services-appsync.html

[4] Amazon Web Services. DynamoDB Streams and AWS Lambda Triggers. Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.Lambda.html

[5] Amazon Web Services. IAM Roles and Policies. Available at: https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html

[6] Amazon Web Services. Monitoring AWS Lambda with Amazon CloudWatch. Available at: https://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions.html

[7] Amazon Web Services. Connecting to AWS AppSync from a client application. Available at: https://docs.aws.amazon.com/appsync/latest/devguide/building-a-client-app.html

[8]Amazon Web Services. AWS Cloud9 – Cloud-based IDE. Available at: https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html

[9] Amazon Web Services. Serverless Architectures with AWS Lambda. Available at: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html