



VIT[®]

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

M. Tech. (Integrated) Software Engineering
School of Computer Science and Engineering (SCOPE)

**Online Learning Management System using Amazon
Web Services**

SWE4002 – Cloud Computing

Final Report Submission

submitted by

Dogiparthi Harini (22MIS1174)

Pamala Thanmai (22MIS1030)

Subireddy Gari Harshitha Reddy (22MIS1139)

Objective

The objective of this project is to develop a cloud-based **Learning Management System (LMS) using Amazon Web Services (AWS)** that offers a secure, scalable, and serverless platform for online education and management. The system enables educators to create and manage courses, conduct live classes, track attendance, and assess students through quizzes, while students can access course materials, join live sessions, and participate in evaluations seamlessly. By integrating services such as Cognito, Lambda, DynamoDB, S3, API Gateway, SNS, IAM, CloudWatch, and Amplify, the LMS ensures efficient automation, real-time communication, high performance, and cost optimization. The overall aim is to deliver a reliable and interactive e-learning platform powered entirely by cloud infrastructure.

List of Modules

- (i) User Authentication and Management
- (ii) Course Management
- (iii) Live Class Management
- (iv) Quiz Management
- (v) Attendance Management
- (vi) Notification Service
- (vii) Profile Management
- (viii) Monitoring and Logging

Detailed description of the module

1. User Authentication and Management

- User authentication and management were achieved using Amazon Cognito for authentication with two separate user pools for Students and Educators.
- Students will have the ability to sign themselves up, but educator accounts must be created by an Administrator in Cognito.
- After an account is confirmed, we use a post-confirmation Lambda trigger to save user details to a DynamoDB with the user-supplied information.
- Additionally, a post-confirmation Lambda trigger sends an activation email (via SNS) to guide the users through the unlocking of their accounts.
- This allows for safe and seamless login to your safe, secure system with data that is available across the platform

2. Course Management

- Course management will utilize the frontend which the Educator can create and upload with the frontend and connected to a Lambda function via API Gateway.

- Students will only be able to see the list of course links and videos that were uploaded via pre-signed URLs provided by Lambda before being uploaded to Amazon S3.
- All course metadata (course titles, course description, URLs, course IDs) are saved in the DynamoDB instance so students and educators can access these items in the courses created in Fowler.

3. Live Class Management

- Educators can schedule and start scheduled live classes from the frontend to the Zoom API (Server-to-Server OAuth App) via a Lambda function.
- The class metadata (class name, class links, class scheduled time) are saved using the identifiers in the dynamoDB.
- Educators and Students can fetch the information and link into the classes.
- Lastly, there are Lambda functions (dedicated) created for tracking attendance for classes scheduled and manage attendance to be saved on the IAM Role in the DynamoDB.

4. Quiz Management

- The quiz system leverages AWS Lambda, API Gateway, and DynamoDB to function.
- Educators will be able to create quizzes by entering a title, start time, end time, and multiple-choice questions (MCQs) with their corresponding answers.
- Each quiz will be assigned a unique Quiz ID; all quiz data is saved in DynamoDB in array format.
- Once the quiz is created, a Simple Notification Service (SNS) email will be automatically sent to students with the quiz information (Quiz ID, title, timings).Students will be able to take the quiz by entering the unique Quiz ID; once the quiz is submitted, the results (name, email, Quiz ID, score) will be saved to a Quiz Results table in DynamoDB.
- Educators will have access to see all quizzes they have created and the students' results through the Lambda functions.

5. Management of Attendance

- Each live class has attendance taken by the operation of Lambda and DynamoDB.
- Each live class has a Class ID saved in the LMS_Attendance table.
- When a student joins a live class, the Lambda function marks the student's attendance with the class information.
- Educators will be able to track attendance for each class by pulling that data from DynamoDB with the Class ID.

6. Notification Service

- Once again, we took advantage of Amazon Simple Notification Service (SNS). SNS is used to send notification emails to users when events occur that are important to the user, such as, quiz notifications, and account activations of any new user.
- We use Lambda functions to trigger an SNS topic depending on backend events, it allows us to notify users when we feel it is necessary.

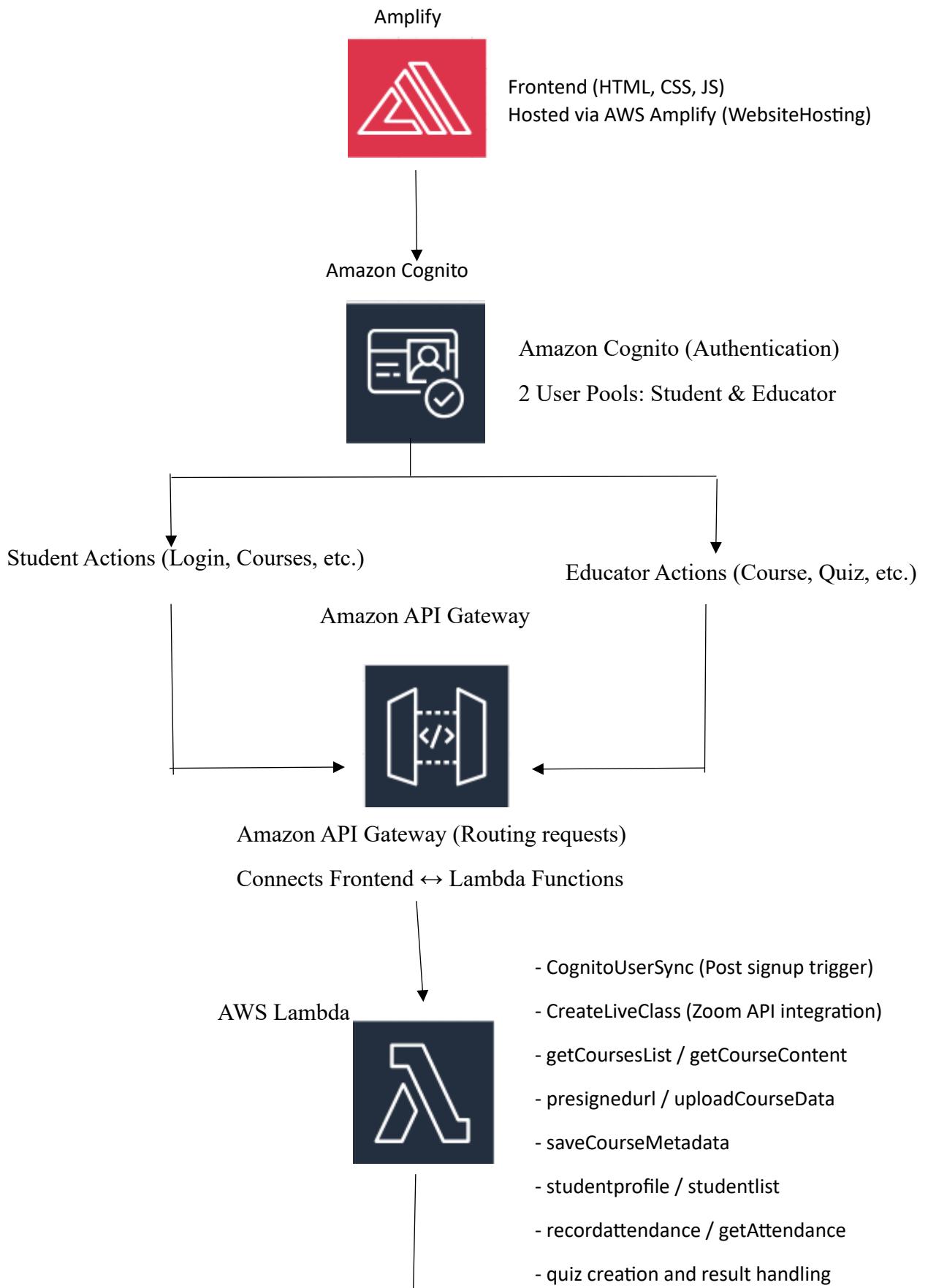
7. Profile Management

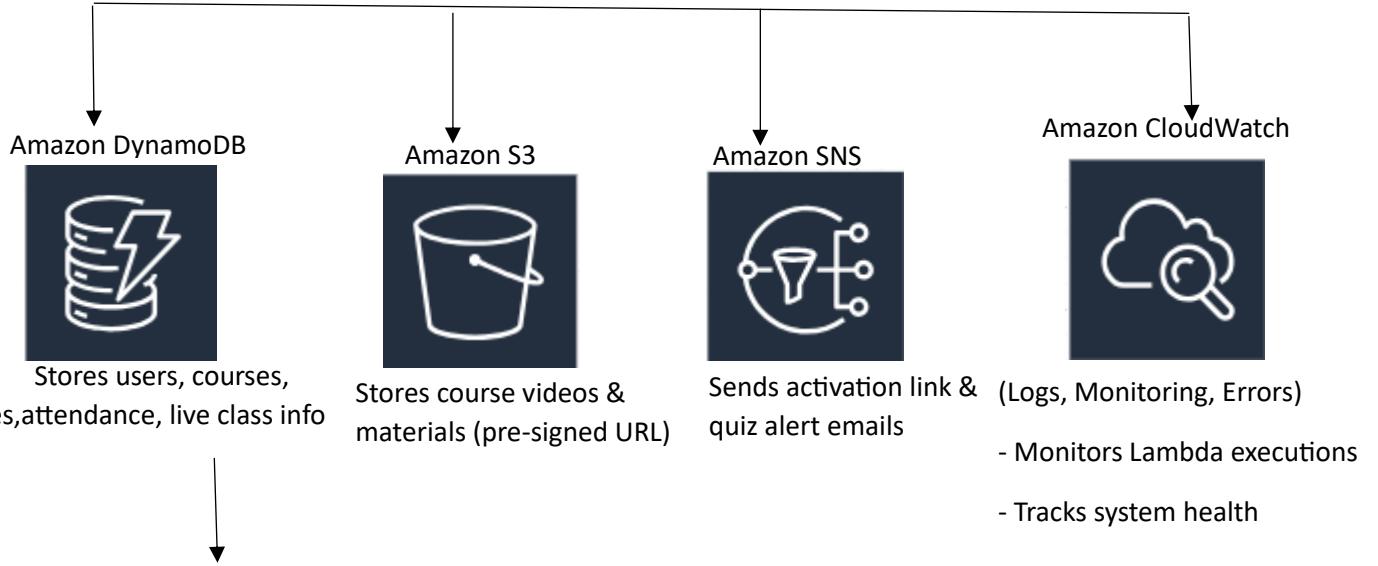
- We are using Amazon Cognito and Lambda to manage profiles of educators and students.
- Once the user has signed in successfully, the studentprofile Lambda function retrieves the user's information (name, email etc.) from Cognito.
- This greatly contributes to the notion of data consistency and the safe retrieval of the users profile through the sign-in process.
- In a similar fashion, the educator is able to see a full listing of all registered students through the use of another lambda to pull student details from DynamoDB.

8. Monitoring and Logging

- We utilized the uses of Amazon Cloudwatch.
- We will use Cloudwatch to monitor performance of the lambda functions and log execution for bug tracking.
- Cloudwatch monitors and logs the invocation from the API Gateway, log any error codes returned from API calls, and monitor latency capabilities to reliably pass workloads.
- This module provides developers a means to debug runtime issues as it happens whether on an issues page, url or id.

Architecture





Zoom API

Zoom API (Live Classes)

(via Server-to-Server OAuth)

Used by Lambda to create & manage live class links

Individual Contribution

Pamala Thanmai

- Designed and coded the Frontend in HTML, CSS and JavaScript.
- Combined the frontend and backend services via API gateway endpoints.
- Deployed the LMS site on AWS Amplify and controlled its deployment pipeline (based on GitHub).
- Established SNS on Amazon to receive notification of quiz and user activation via email.
- Designed the Quiz Management Module:
 - Creation of quizzes, management of questions and storage of their results using Lambda and DynamoDB.
 - SNS to post quiz notifications among students automatically.

SubireddyGari Harshitha Reddy

- AWS Cognito User Authentication and Authorization, which is implemented.
- Developed two distinct user groups, one being Students and another one being Educators.
- Install Lambda triggers (Post Confirmation) to perform automatic synchronization of data to DynamoDB.
- Assured secure user access and both role-based authentication.

Dogiparthi Harini

- Developed and deployed the back-end logic in AWS Lambda and API Gateway in all the core functionalities.
- Stored DynamoDB to save and get course, quizzes data, attendance data, and user data.
- Developed the Course Management Module (upload, fetch, and store content on the basis of pre-signed URLs of S3 + Lambda).
- Zoom Server to Server OAuth Application to create and manage Live Classes.
- Applied CloudWatch to Introduce Attendance Tracking, Profile Management and Monitoring.
- Set IAM Roles in order to grant access to Lambda, S3, and DynamoDB safely.
- CloudWatch Logs End to end system testing and debugging.

Tools/Software requirement

Frontend:

- HTML, CSS, JavaScript: Applied to create interactive and responsive web pages on the interface of students and educators.
- AWS Amplify: An app that is utilized to host and roll out the LMS web application directly out of GitHub.

Backend:

- AWS Lambda: Serverless computing service that is applied to implement the logic of the backend and interlink among various services in AWS.
- Amazon API Gateway: API Gateway is utilized to develop, publish and administer REST and HTTP APIs, which interlink the frontend to the Lambda functions.
- Amazon DynamoDB: NoSQL database that is used to store user information, courses, quizzes, attendance, classes.
- Amazon Cognito: This is employed to authenticate and manage user-securely (two different user pools: Student and Educator).
- Amazon Simple Notification Service (SNS): Sends notification emails (e.g. quiz notifications, account activation) in an automated fashion.
- Amazon S3: It will be used to store videos and materials of courses and retrieve them based on pre-sign URLs.
- Amazon CloudWatch: Utilized to monitor Lambda functions and monitor logs along with debugging errors.
- AWS Identity and Access Management (IAM): Applied to roles and permissions between AWS services.

Other Tools:

- Zoom Developer Account (Server-to-Server OAuth App): It is used to make and manage live classes.
- Visual Studio Code: The code development tool used when writing frontend and Lambda functions.
- GitHub: Utilised in version control and in integration with AWS Amplify to allow continuous deployment.

AWS services used

1) Amazon Cognito (User Authentication/ Authorization)

- Transactions with users of two pools of secured access and registration of students and teachers.
- Cognito has a post-confirmation trigger, which invokes a Lambda function to store the information of new users into DynamoDB.
- Sends the users email messages with SNSs when he is registered.
- Ensures access control and secure session management of both roles.

2) AWS lambda (Serverless Backend Logic)

- Conducts all the back-end operations such as user synchronization, course management, quiz management, attendance management and real time classes management.
- Cognito events, API gateway calls or internal processes.
- Gathers and reads data in DynamoDB, sends pre-signed URLs to S3 and calls Zoom API to retrieve live classes.
- Eliminates the usage of dedicated servers meaning that it is scalable and cost-effective.

3) Amazon API Gateway (Frontend Backend Bridge)

- Connects the web frontend with Lambda backend functions using REST and HTTP API.
- Course uploading, content acquisition, quiz creation, and accessing attendance are some of the processes involved.
- Granted access to APIs appropriately and authorization.
- Manages as the communication support of all frontend activities.

4) Amazon DynamoDB (Database Storage)

- Stores all the system data including the users, courses, quizzes, live classes and attendance.
- It has 4 major tables, namely: Cognitousers, Courses, LMS_LiveClasses, LMS_Attendance.
- Offers well-scaling, high-speed, and serverless data storage by the use of the NoSQL technology in order to obtain rapid access to data.
- The Lambda functions use it to store all the information about the application in an effective way.

5) Amazon S3 (Storage repository of course material)

- Stores course videos and study materials uploaded by educators.
- Upload and download via pre-signed URLs (generated by Lambda).
- A separate bucket to store files (edu-platform-course), and a separate bucket to store links (DynamoDB).
- Provide safe, unlimited and trustworthy content storage.

6) Amazon Simple Notification service (SNS) (Email Notifications)

- Sends email messages to learners and faculty.
- Appears in the notification of a quiz, as well as in the activation of an account.
- Together with Lambda to automatically e-mail when specific events were monitored.
- Offers real time communication across the platform.

7) AWS Identity and Access management (IAM) (Permissions Control)

- The permissions to the access of Lambda, S3, DynamoDB, and SNS.
- Every Lambda function has an IAM role which permits a list of actions.
- Ensures good service to service communication and the removal of unauthorized access.
- Secures and compliance of the system overall.

8) Amazon CloudWatch(Monitoring and debugging)

- Monitoring performance and log of execution of every Lambda function.
- Identify and remove errors in API calls and back-end processes.
- Optimize using tracking and track performance information.
- The product is very reliable and it is easily troubleshootable.

9) AWS Amplify(Monitoring and debugging)

- Just a service, the open source LMS written in HTML, CSS and JavaScript.
- Automatic deploying and construction on GitHub.
- Provides an online address to a web site:
<https://main.dijffme8w1boe.amplifyapp.com>
- Offers scalable, fast and secure hosting offering.

Screen shots of all services and features of project

(i) COGNITO :

Educator userpool

The screenshot shows the 'Overview' page for the 'Educators' user pool in Amazon Cognito. The left sidebar shows navigation options like 'Overview', 'Applications', 'User management', and 'Authentication'. The main panel displays 'User pool information' including the name 'Educators', ARN, Token signing key URL, and estimated number of users (3). It also includes 'Recommendations' for setting up an app and applying branding to login pages.

Self registration disabled since only admin can create the educator

The screenshot shows the 'Sign-up' configuration page for the 'Educators' user pool. The left sidebar shows 'Sign-up' selected under 'Authentication'. The main panel shows 'Required attributes' (email, name) and 'Custom attributes' (0). Under 'Self-service sign-up', it is set to 'Disabled'.

App client for Educator

The screenshot shows the 'App client information' section for the 'LMSEducator' app client. Key details include:

- App client name:** LMSEducator
- Client ID:** Tumutsnskbkrdl23mk6jun0kdv
- Client secret:** (redacted)
- Authentication flows:** Choice-based sign-in, Secure remote password (SRP)
- Authentication flow session duration:** 3 minutes
- Refresh token expiration:** 5 day(s)
- Access token expiration:** 60 minutes
- ID token expiration:** 60 minutes
- Advanced authentication settings:** Enable token revocation, Enable prevent user existence errors

Other tabs visible include: Quick setup guide, Attribute permissions, Login pages, Threat protection, Analytics.

Students userpool

The screenshot shows the 'User pool information' section for the 'Students' user pool. Key details include:

- User pool name:** Students
- User pool ID:** us-east-1_tj9jiNyqX
- ARN:** arn:aws:cognito-idp:us-east-1:073017371430:userpool/us-east-1_tj9jiNyqX
- Token signing key URL:** https://cognito-idp.us-east-1.amazonaws.com/us-east-1_tj9jiNyqX/well-known/jwks.json
- Estimated number of users:** 3
- Feature plan:** Essentials

Other tabs visible include: Recommendations, Set up your app: LMSStudent, Apply branding to your managed login pages.

Self registration enabled for students

The screenshot shows the 'Sign-up' configuration for the 'Students' user pool. Under 'Required attributes', 'email' and 'name' are listed. In the 'Custom attributes' section, there are no custom attributes defined. Under 'Self-service sign-up', 'Self-registration' is set to 'Enabled'. The left sidebar shows navigation options for 'Amazon Cognito', 'User pools', 'Applications', 'User management', and 'Authentication'.

App client for Student

The screenshot shows the 'App client: LMSStudent' configuration. The 'App client information' section includes details like Client ID (7cdmtqipvq30pdeo17hv7h6n0s), Client secret (*****), and Authentication flows (Choice-based sign-in, Secure remote password (SRP)). Other settings include Authentication flow session duration (3 minutes), Refresh token expiration (5 day(s)), Access token expiration (60 minutes), ID token expiration (60 minutes), and Advanced authentication settings (Enable token revocation, Enable prevent user existence errors). The 'Login pages' tab is selected at the bottom. The left sidebar shows navigation options for 'Amazon Cognito', 'User pools', 'Applications', 'User management', and 'Authentication'.

(ii) AWS S3

Edu-platform-course bucket with courses folder to store video and material uploaded by educator

The screenshot shows the AWS S3 console interface. The top navigation bar includes the AWS logo, a search bar, and account information (Account ID: 0730-1737-1430, United States (N. Virginia), Harini Dogiparthi). The left sidebar under 'Amazon S3' lists various bucket categories: General purpose buckets, Directory buckets, Table buckets, Vector buckets, Access Grants, Access Points (General Purpose Buckets, FSx file systems), Access Points (Directory Buckets), Object Lambda Access Points, Multi-Region Access Points, Batch Operations, and IAM Access Analyzer for S3. The main content area is titled 'edu-platform-course' and shows the 'Objects' tab selected. It displays one object, 'courses/' (Type: Folder). There are buttons for Copy S3 URI, Copy URL, Download, Open, Delete, Actions, and Create folder. A note at the bottom says: 'Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions.' A 'Find objects by prefix' search bar and a 'Show versions' toggle are also present.

(iii) SNS (Simple Notification Service)

SNS quiz-notifications topic created to send notifications to students when quiz created by educator

The screenshot shows the AWS SNS console interface. The top navigation bar includes the AWS logo, a search bar, and account information (Account ID: 0730-1737-1430, United States (N. Virginia), Harini Dogiparthi). The left sidebar under 'Amazon SNS' lists Topics, Subscriptions, and Mobile (Push notifications, Text messaging (SMS)). The main content area is titled 'quiz-notifications' and shows the 'Details' tab selected. It displays the topic's Name (quiz-notifications), ARN (arn:aws:sns:us-east-1:073017371430:quiz-notifications), and Type (Standard). The 'Subscriptions' tab is selected, showing three confirmed email subscriptions:

ID	Endpoint	Status	Protocol
21e8171c-52d4-4f31-ad2c-9...	harinilakshaya6150@gmail.com	Confirmed	EMAIL
6132f011-ee42-4bf4-8e5f-8...	tharmai784@gmail.com	Confirmed	EMAIL
b1c6f881-727a-4387-a972-2...	harshithareddy.sweety28@g...	Confirmed	EMAIL

(iv) AWS Amplify

Used to host the website

The screenshot shows the AWS Amplify LMS Overview page. On the left, there's a sidebar with 'Overview', 'Hosting', 'Monitoring', and 'App settings'. The main area has a heading 'lms' and an 'App ID: dijffme8w1boe'. It features a 'Get to production' section with three steps: 'Add a custom domain', 'Enable firewall protections', and 'Connect new branches'. Below this is a 'Branches' section with one branch named 'main' (Deployed). The URL is listed as <https://main.dijffme8w1boe.amplifyapp.com>. At the bottom, there are links for 'cloudShell', 'Feedback', and 'Console Mobile App'.

(v) API GATEWAY

6 APIs are created to connect with frontend pages

The screenshot shows the AWS API Gateway APIs list page. On the left, there's a sidebar with 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', 'Usage plans', 'API keys', 'Client certificates', and 'Settings'. The main area shows a table titled 'APIs (6/6)' with columns for Name, Description, ID, Protocol, API endpoint type, and Created. The APIs listed are: CognitoProfileAPI, CoursesAPI, CreateLiveClass, lms-materials, quiz, and studentprofile. Each entry includes a 'Delete' button and a 'Create API' button.

Name	Description	ID	Protocol	API endpoint type	Created
CognitoProfileAPI		brn328k062	REST	Regional	2025-11-06
CoursesAPI		vc3atpev41	REST	Regional	2025-11-06
CreateLiveClass		3qfr9chuv8	HTTP	Regional	2025-11-06
lms-materials		oyt8yc3le9	HTTP	Regional	2025-11-05
quiz		vwnzyha9dd	HTTP	Regional	2025-11-10
studentprofile		43jqxl3q3k	HTTP	Regional	2025-11-06

CognitoProfileAPI

The screenshot shows the AWS API Gateway Resources page for the 'CognitoProfileAPI' API. On the left, the navigation sidebar shows the API Gateway and the specific API 'CognitoProfileAPI'. The main panel displays a resource named '/profile' with methods: GET, OPTIONS, and POST. To the right, a detailed view of the '/profile - GET - Method execution' is shown, illustrating the flow from a 'Client' to a 'Method request', then to an 'Integration request' (handled by a Lambda integration), and finally a 'Method response' back to the client. The ARN of the resource is listed as arn:aws:execute-api:us-east-1:073017371430:brn328k062/*GET/profile. The Resource ID is gkxpnh. Buttons for 'Update documentation' and 'Delete' are visible at the top right.

CoursesAPI

The screenshot shows the AWS API Gateway Resources page for the 'CoursesAPI' API. The left sidebar shows the API Gateway and the specific API 'CoursesAPI'. The main panel displays a resource named '/' with several methods: GET, OPTIONS, and others like /courses, /generate-presigned-url, /meta, and /upload-course. To the right, the 'Resource details' section shows the path '/courses' and the resource ID 0d3zix. Below it, the 'Methods (2)' section lists two entries: a GET method using Lambda integration and a OPTIONS method using Mock integration. Buttons for 'Delete', 'Update documentation', and 'Enable CORS' are visible at the top right.

Routes for CreateLiveClass

POST /create-class

The screenshot shows the AWS API Gateway Routes page for the 'CreateLiveClass' route. The left sidebar shows the API Gateway and the specific API 'CreateLiveClass... (3qfr9chuv8)'. The main panel displays the route 'Routes for CreateLiveClass' with a search bar and a 'Create' button. Below it is a list of routes: /create-class, /get-attendance, /get-classes, and /get-studentclasses. To the right, the 'Route details' section shows the method POST /create-class (ID: pxdvaxq). It includes fields for 'ARN' (arn:awsapigateway:us-east-1::apis/3qfr9chuv8/routes/pxdvaxq), 'Authorization' (described as 'Authorizers protect your API against unauthorized requests. Routes with no authorization attached are open.'), and 'Integration' (described as 'The integration is the backend resource that this route calls when it receives a request.'). A note states 'No authorizer attached to this route.' and a 'Configure' button is available. Buttons for 'Stage: -' and 'Deploy' are visible at the top right.

GET /get-attendance

The screenshot shows the AWS API Gateway console. On the left, the sidebar has 'APIs' selected under 'Custom domain names'. The main area shows a 'Routes' section for an API named 'CreateLiveClass... (3qfr9chuv8)'. A route named '/get-attendance' is selected, showing its details. The 'Route details' pane indicates it's a GET request to '/get-attendance' (ID: 914meri). It shows the ARN as arn:aws:apigateway:us-east-1::apis/3qfr9chuv8/routes/914meri and notes that no authorizer is attached. There are 'Delete' and 'Edit' buttons at the top right.

GET /get-classes

This screenshot shows the same API configuration as the previous one, but with a different route selected: '/get-classes'. The 'Route details' pane now shows it's a GET request to '/get-classes' (ID: 2v8rynn). The ARN is arn:aws:apigateway:us-east-1::apis/3qfr9chuv8/routes/2v8rynn. No authorizer is attached. There are 'Delete' and 'Edit' buttons at the top right.

GET /get-studentclasses

This screenshot shows the API configuration with a third route selected: '/get-studentclasses'. The 'Route details' pane shows it's a GET request to '/get-studentclasses' (ID: 1519d3g). The ARN is arn:aws:apigateway:us-east-1::apis/3qfr9chuv8/routes/1519d3g. No authorizer is attached. There are 'Delete' and 'Edit' buttons at the top right.

Routes for lms-materials

POST /materials

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with 'APIs' selected. Under 'APIs', there are sections for 'Custom domain names', 'Domain name access associations', and 'VPC links'. Below this, it says 'API: lms-materials... (oyt8yc3le9)'. On the right, the main area is titled 'Routes' and contains a section for 'Routes for lms-materials'. It shows a single route: 'POST /materials'. The 'Route details' panel on the right shows the ARN as 'arn:aws:apigateway:us-east-1::apis/oyt8yc3le9/routes/6fnbk4i'. It also indicates that no authorizer is attached to this route and provides a 'Configure' button for integration.

GET /materials

This screenshot is similar to the previous one, but it shows a GET route instead of a POST. The 'Route details' panel on the right shows the ARN as 'arn:aws:apigateway:us-east-1::apis/oyt8yc3le9/routes/ytaci43'. The rest of the interface is identical, including the sidebar and the overall layout.

Routes - quiz

This screenshot shows the AWS API Gateway console for a different API named 'quiz'. The sidebar shows 'Custom domain names', 'Domain name access associations', and 'VPC links'. Below this, it says 'API: quiz(vwnzyha9dd)'. The main area shows two routes: 'POST /createQuiz' and 'GET /getAllQuizzes'. The 'Route details' panel on the right shows the ARN for the POST route as 'arn:aws:apigateway:us-east-1::apis/vwnzyha9dd/routes/vnlsbcs'. It also indicates that no authorizer is attached to this route and provides a 'Configure' button for integration.

Routes for studentprofile

The screenshot shows the AWS API Gateway console. On the left, the sidebar has 'APIs' selected under 'API Gateway'. The main area is titled 'Routes' and shows a single route named 'Routes for studentprofile'. This route has a single endpoint 'callback' with a 'GET' method. To the right, there's a 'Route details' panel. It shows the ARN as 'arn:aws:apigateway:us-east-1::apis/43jqxl3q3k/routes/j411eek'. Under 'Authorization', it says 'No authorizer attached to this route.' and has a 'Attach authorization' button. Under 'Integration', it says 'The integration is the backend resource that this route calls when it receives a request.' and shows 'fas8zl' with a 'Configure' button. At the top right, there are 'Delete' and 'Edit' buttons.

(vi) DYNAMO DB

The screenshot shows the AWS DynamoDB console. On the left, the sidebar has 'Tables' selected under 'DynamoDB'. The main area is titled 'Tables (8)' and lists eight tables: cognitousers, Courses, LMS_Attendance, LMS_LiveClasses, LMS-Courses, Quiz-Submissions, QuizResults, and Quizzes. Each table row includes columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, and Favorite. The 'cognitousers' table is currently selected. At the bottom, there are links for CloudShell, Feedback, and Console Mobile App, along with copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' and links for Privacy, Terms, and Cookie preferences.

(vii) LAMBDA

The screenshot shows the AWS Lambda Functions list page. At the top, there's a search bar and a 'Create function' button. Below is a table with columns: Function name, Description, Package type, Runtime, and Last modified. The table lists 21 functions, all of which are Zip packages running on Python 3.12, except for one Node.js 22.x function. Most functions were last modified 3 days ago, except for a few like 'GetCurrentAssistant' and 'GetCourseContent' which were last modified 7 days ago.

Function name	Description	Package type	Runtime	Last modified
getQuizResult	-	Zip	Python 3.12	3 days ago
getQuizResult	-	Zip	Python 3.12	3 days ago
getCourseContent	-	Zip	Python 3.12	7 days ago
submitQuiz	-	Zip	Python 3.12	3 days ago
studentprofile	-	Zip	Python 3.12	7 days ago
recordattendance	-	Zip	Python 3.12	7 days ago
currentallassistant	-	Zip	Python 3.12	2 days ago
LMS-CourseMaterials	-	Zip	Node.js 22.x	1 week ago
getCoursesList	-	Zip	Python 3.12	7 days ago
CreateLiveClass	-	Zip	Python 3.12	7 days ago
studentlist	-	Zip	Python 3.12	7 days ago

(viii) IAM

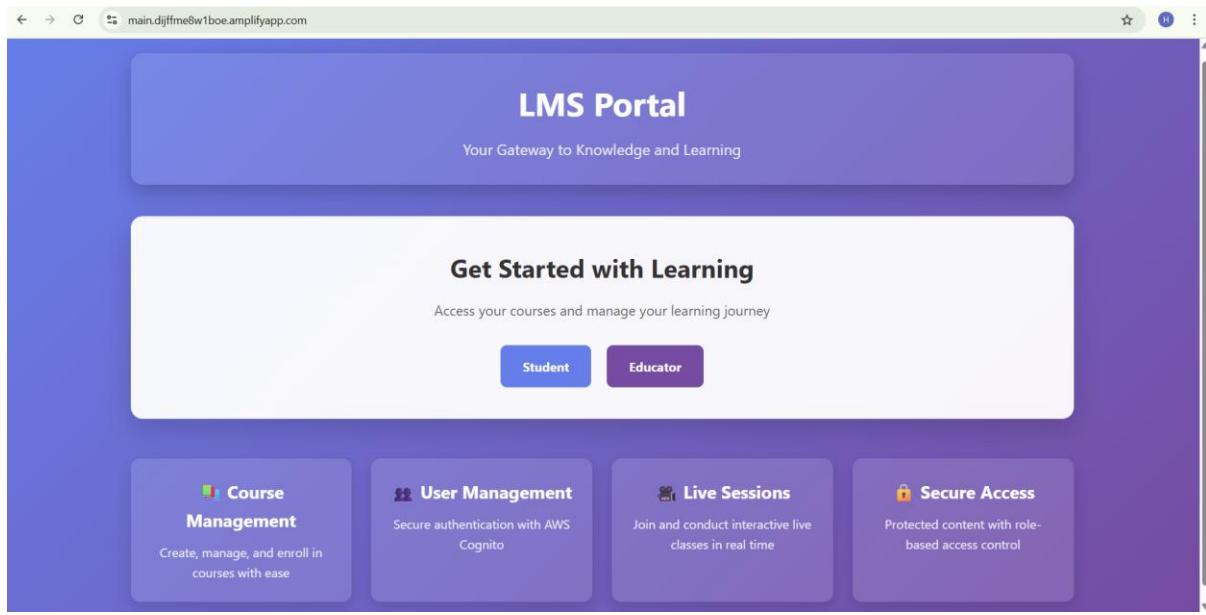
The screenshot shows the AWS IAM Roles list page. On the left, there's a navigation sidebar for Identity and Access Management (IAM). The main area displays a table of roles, each with a role name, trusted entities, and last activity. Most roles have AWS services as their trusted entities and were last active within the last 24 hours. One role, 'CreateQuizLambda-role-btnezelw', was last active 3 hours ago.

Role name	Trusted entities	Last activity
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service)	-
AWSServiceRoleForLexV2Bots_D43SH4KSQ2K	AWS Service: lexv2 (Service-Linked R)	23 hours ago
AWSServiceRoleForResourceExplorer	AWS Service: resource-explorer-2 (Service)	22 minutes ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linked)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-
currentallassistant-role-by2f9j6p	AWS Service: lambda	23 hours ago
CloudTrail_CloudWatchLogs_Role	AWS Service: cloudtrail	14 minutes ago
CreateLiveClass-role-2brsu85	AWS Service: lambda	3 hours ago
CreateQuizLambda-role-btnezelw	AWS Service: lambda	3 hours ago
GetAttendance-role-no4vqoag	AWS Service: lambda	3 hours ago
GetClassesLambda-role-a1546e8c	AWS Service: lambda	3 hours ago
getCourseContent-role-w74euued4	AWS Service: lambda	7 hours ago

(ix) CLOUDWATCH

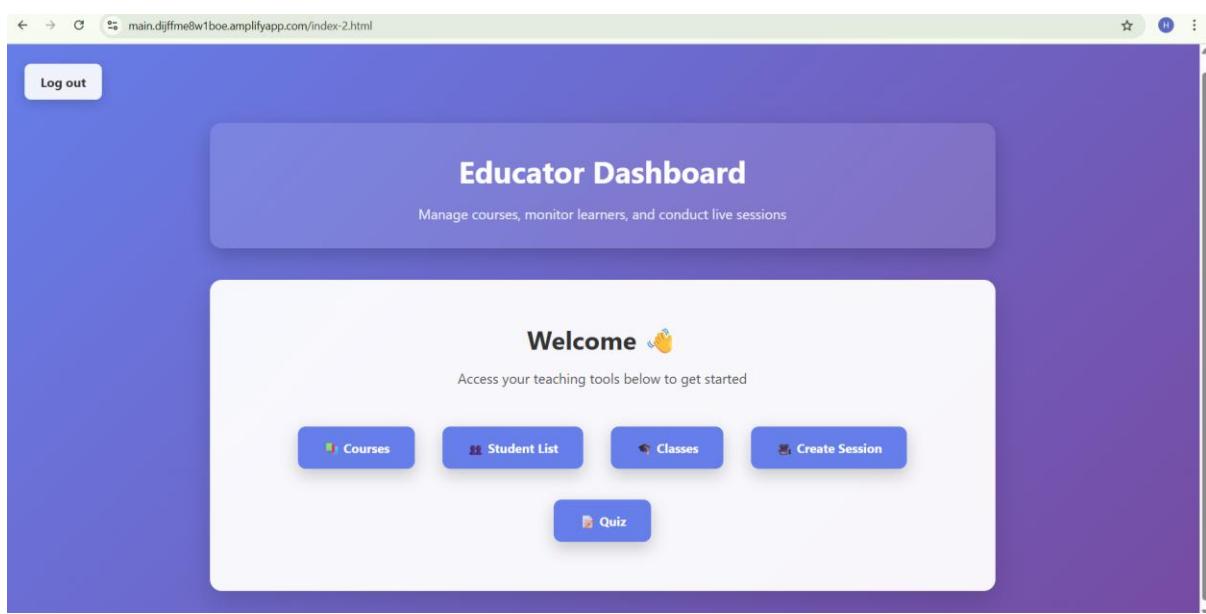
The screenshot shows the AWS CloudWatch Log groups list page. The left sidebar includes sections for AI Operations, Alarms, Logs (Log groups, Log anomalies, Live Tail, Logs Insights, Contributor Insights), Metrics, and Application Signals. The main area shows a table of log groups, each with a log group name, log class, and configuration options. All log groups are of the Standard type and have a retention period of 'Never expire'.

Log group	Log class	Anomaly d...	Data pr...	Sensitiv...	Retention	Metric fi...
/aws/lambda/CognitoUserSync	Standard	Configure	-	-	Never expire	-
/aws/lambda/CreateLiveClass	Standard	Configure	-	-	Never expire	-
/aws/lambda/CreateQuizLambda	Standard	Configure	-	-	Never expire	-
/aws/lambda/GetAttendance	Standard	Configure	-	-	Never expire	-
/aws/lambda/GetClassesLambda	Standard	Configure	-	-	Never expire	-
/aws/lambda/GetStudentClasses	Standard	Configure	-	-	Never expire	-
/aws/lambda/currentallassistant	Standard	Configure	-	-	Never expire	-
/aws/lambda/getCoursesList	Standard	Configure	-	-	Never expire	-
/aws/lambda/getQuiz	Standard	Configure	-	-	Never expire	-
/aws/lambda/getQuizResult	Standard	Configure	-	-	Never expire	-
/aws/lambda/time-source-ani	Standard	Configure	-	-	Never expire	-



The screenshot shows the LMS Portal homepage with a purple header and footer. The main content area has a white background. At the top center is a large blue button labeled "LMS Portal" and "Your Gateway to Knowledge and Learning". Below this is a section titled "Get Started with Learning" with a sub-instruction "Access your courses and manage your learning journey". Two buttons, "Student" (blue) and "Educator" (purple), are present. The footer contains four cards: "Course Management" (Create, manage, and enroll in courses with ease), "User Management" (Secure authentication with AWS Cognito), "Live Sessions" (Join and conduct interactive live classes in real time), and "Secure Access" (Protected content with role-based access control).

EDUCATOR SIDE



The screenshot shows the Educator Dashboard with a purple header and footer. The main content area has a white background. A "Log out" button is in the top-left corner. A central blue button is labeled "Educator Dashboard" and "Manage courses, monitor learners, and conduct live sessions". Below this is a white box titled "Welcome" with a smiling emoji. Inside the box is the instruction "Access your teaching tools below to get started". Five buttons are listed: "Courses", "Student List", "Classes", "Create Session", and "Quiz".

Back Courses + Add Course

Certified Courses

Python
Learn Python for general-purpose programming.
Start

C++
Understand the fundamentals of C++ programming.
Start

Uploaded Courses

cloud
cloud
Start

XYZ

← → ⌂ main.dijffme@w1boe.amplifyapp.com/course-content-new.html?courseId=course-8bfa5c0db1

cloud

Video:



The image shows two screenshots of a web browser interface.

The top screenshot displays a course content page titled "KNOWLEDGE MANAGEMENT SYSTEMS" with a "FINAL REVIEW" button. The URL is main.dijffme8w1boe.amplifyapp.com/course-content-new.html?courseId=course-8bfa5c0db1. The browser tab bar includes multiple open tabs related to course management and attendance.

The bottom screenshot shows a "Create New Course" form. The form fields are:

- Course Title: Enter course title
- Description: Enter description
- Educator Name: Enter your name
- Course Video (MP4): Choose File (No file chosen)
- Course Material (PDF): Choose File (No file chosen)

A large blue "Create Course" button is at the bottom of the form.

[← Back](#) **Student List** [Search by name...](#)

Name	Email	User ID	Created At	Role	Status
Harini Akshaya	hariniakshaya6130@gmail.com	34987478-1031-70fb-638c-23c1a6f3fa0f	11/11/2025, 4:50:44 AM	student	Enabled
Harshitha	harshithareddy.sweety28@gmail.com	c45894b8-50a1-7001-62ef-22f23ba37d5e	11/13/2025, 6:11:04 AM	student	Enabled
thanmai784@gmail.com	thanmai784@gmail.com	54987478-a031-7050-1baa-c7abba760109	11/13/2025, 4:11:31 AM	student	Enabled

[← Back](#) **Sessions List**

Class Name	Schedule Time	Attendance	Join Link	Start Link
liveclass	2025-11-13 16:15	1 View	Join	Start
ABC		1 View	Join	Start

Educator - Class Attendance [← Back](#)

Attendance List
Class ID: 88445391593

Student Name †	Join Time (IST) †
harshitha	2025-11-13 04:15:12 PM

[← Back](#)

Create Live Class

Schedule and start your live sessions effortlessly

Class Details

Class Name

Schedule Time

Duration (minutes)

[Create Class](#)

After creation, you'll get Join / Start links below.

[← Back](#)

Quiz Dashboard

Manage quizzes, view results, and keep track of student progress

[+ Create Quiz](#) [All Quiz Data](#) [Quiz Results](#)

[← Back](#)

Create a New Quiz

Quiz Title

Start Time

End Time

Question 1

Question Text

Option A

Option B

Option C

Option D

Correct Answer (A/B/C/D)

[Add Question](#) [Create Quiz](#)

main.dijffme8w1boe.amplifyapp.com/quiz-data.html

All Quizzes

Quiz ID	Title	Created At	Scheduled From	Scheduled To	Action
quiz_f38f7a2d	cloud quiz	2025-11-13T10:40:38.973958	2025-11-13T16:10	2025-11-13T16:20	<button>View</button>
quiz_8e98faaa	TTT	2025-11-13T04:27:31.309917	2025-11-13T10:00	2025-11-13T10:30	<button>View</button>

main.dijffme8w1boe.amplifyapp.com/quiz-data.html

All Quizzes

Quiz ID	Title	Created At	Scheduled From	Scheduled To	Action
quiz_f38f7a2d	cloud quiz	2025-11-13T10:40:38.973958	2025-11-13T16:10	2025-11-13T16:20	<button>View</button>
quiz_8e98faaa	TTT	2025-11-13T04:27:31.309917	2025-11-13T10:00	2025-11-13T10:30	<button>View</button>

Quiz Details

cloud quiz
1. 2+3
4
5
6
7

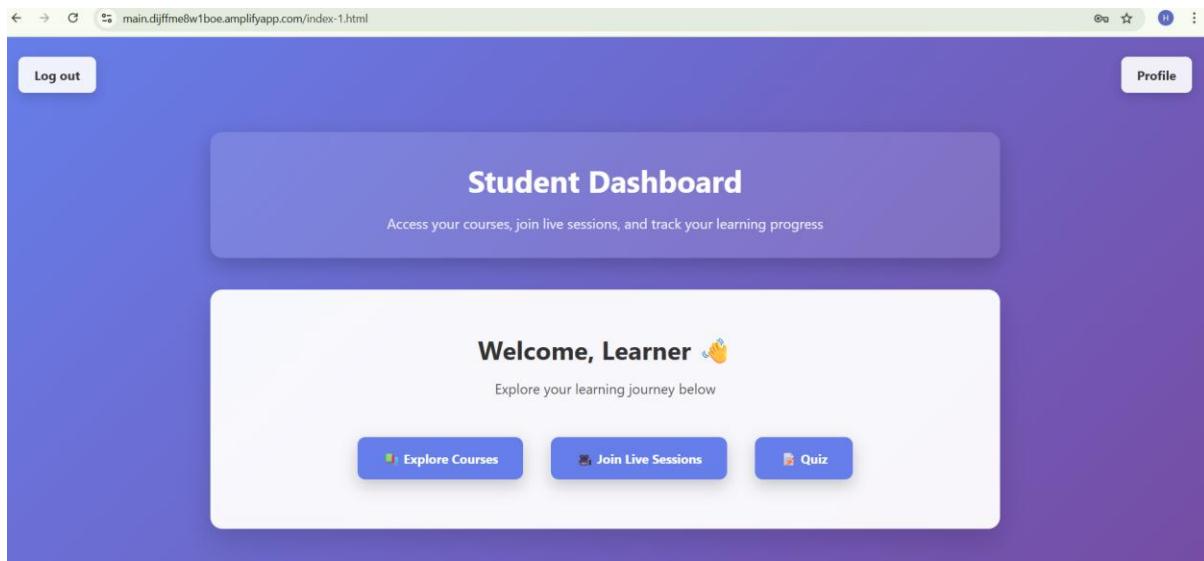
main.dijffme8w1boe.amplifyapp.com/quiz-results.html

Quiz Results

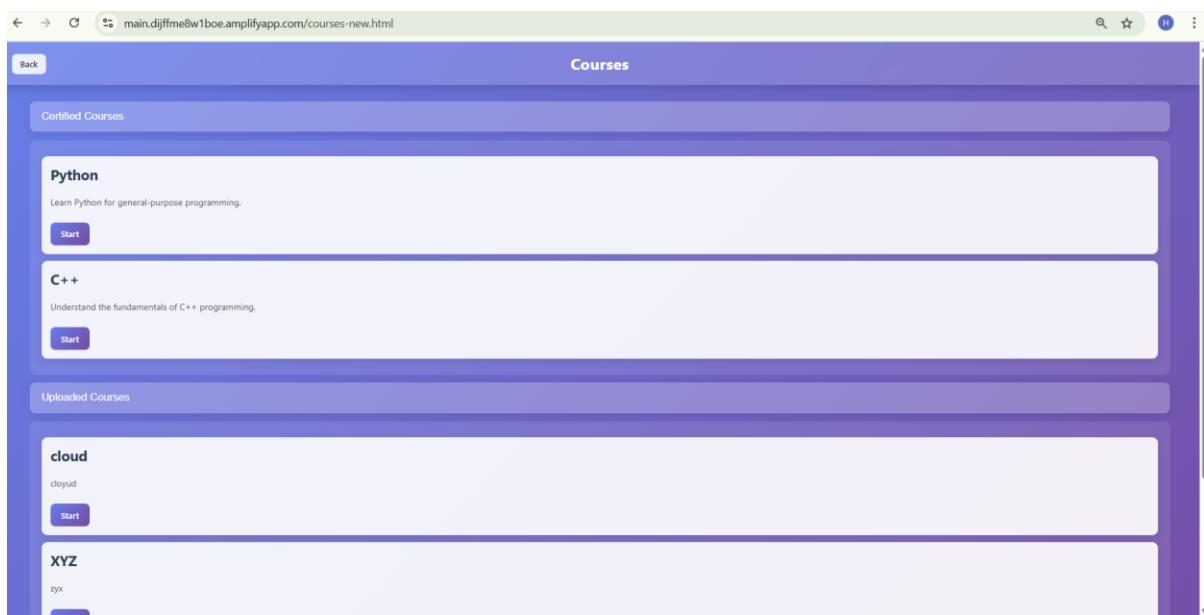
quiz_f38f7a2d Get Results

Student Name	Email	Score
Harini	hariniakshaya6130@gmail.com	1

STUDENT SIDE



The screenshot shows the Student Dashboard. At the top, there are 'Log out' and 'Profile' buttons. Below that is a purple header with the title 'Student Dashboard' and the sub-instruction 'Access your courses, join live sessions, and track your learning progress'. The main area has a white background with a central box containing the text 'Welcome, Learner' with a hand icon, followed by the instruction 'Explore your learning journey below'. Three buttons are present: 'Explore Courses' (with a globe icon), 'Join Live Sessions' (with a video camera icon), and 'Quiz' (with a document icon).



The screenshot shows the 'Courses' page. At the top, there is a 'Back' button and a 'Courses' title. Below that is a 'Certified Courses' section with two items: 'Python' (described as 'Learn Python for general-purpose programming.' with a 'Start' button) and 'C++' (described as 'Understand the fundamentals of C++ programming.' with a 'Start' button). Below this is an 'Uploaded Courses' section with two items: 'cloud' (uploaded by 'cloyd' with a 'Start' button) and 'XYZ' (uploaded by 'zyx' with a 'Start' button).

XYZ

Video:

United States (N. Virginia) ▾ Account ID: 0730-1737-1430 ▾ Harini Dogiparthi

lish (US) > Intents

Test Draft version
Last build submitted: Now

Inspect

Type a message

Material:

Amazon AWS: A Detailed Overview

Introduction

Amazon Web Services (AWS) is the world's most comprehensive and widely adopted cloud platform, offering over 200 fully featured services from data centers globally. Businesses, governments, and individuals use AWS for cloud computing, storage, networking, security, and AI/ML capabilities.

Key AWS Services and Offerings

1. Compute Services

AWS provides scalable compute power for various applications:

- Amazon EC2 (Elastic Compute Cloud) – Virtual servers (instances) with flexible computing capacity.
- AWS Lambda – Serverless computing to run code without provisioning infrastructure.
- Amazon ECS & EKS – Container services for deploying and managing Docker and Kubernetes workloads.
- AWS Batch – Fully managed service for running batch computing jobs.

2. Storage Services

← Back

Available Classes

Class #	Schedule Time #	Join
ABC	2025-11-13 04:21	Join
liveclass	2025-11-13 16:15	Join

← → ⌛ main.dijffme8w1boe.amplifyapp.com/join-class.html

main.dijffme8w1boe.amplifyapp.com says
Attendance marked successfully

OK

Available Classes

Class #	Schedule Time #	Join
ABC	2025-11-13 04:21	Join
liveclass	2025-11-13 16:15	Join

← → ⌛ main.dijffme8w1boe.amplifyapp.com/take-quiz.html

Student Quiz Portal
Enter Details

Your Name
Your Email
Quiz ID (provided by educator)

Start Quiz

← → ⌛ main.dijffme8w1boe.amplifyapp.com/take-quiz.html

Student Quiz Portal
Quiz

1. $2+3$

4
 5
 6
 7

Submit Quiz

← → ⌛ main.dijffme8w1boe.amplifyapp.com/take-quiz.html

Student Quiz Portal

✓ Your score: 0 / 1

Coding of the project

I. CognitoUserSync

```
import boto3
import os
from datetime import datetime

# Initialize AWS clients
dynamodb = boto3.resource('dynamodb')
sns = boto3.client('sns')

# Environment variables
table_name = os.environ.get('USER_TABLE', 'cognitousers')
sns_topic_arn = os.environ.get('SNS_TOPIC_ARN')

table = dynamodb.Table(table_name)

def lambda_handler(event, context):
    print("EVENT:", event)

    try:
        user_email = event['request']['userAttributes'].get('email')
        user_name = event['request']['userAttributes'].get('name', "")
        user_id = event['userAttributes']

        # 1. Save user to DynamoDB
        table.put_item(
            Item={
                'email': user_email,
                'user_id': user_id,
                'name': user_name,
```

```

        'role': 'student',
        'status': 'Enabled',
        'created_at': datetime.utcnow().isoformat()
    }
)

print(f" User {user_email} stored in DynamoDB")

# 2. Subscribe new student to SNS topic (email notification)
if user_email and sns_topic_arn:
    sns.subscribe(
        TopicArn=sns_topic_arn,
        Protocol='email',
        Endpoint=user_email
    )
    print(f"Subscribed {user_email} to SNS topic")
else:
    print(" Missing email or SNS_TOPIC_ARN")

return event

except Exception as e:
    print(f" Error: {str(e)}")
    raise e

```

II. studentlist

```

import json

import boto3

from boto3.dynamodb.conditions import Attr

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('cognitousers')

```

```

def lambda_handler(event, context):

    print("🟢 Incoming event:", json.dumps(event)) #👉 ADD THIS LINE

    try:
        # Scan all users where role = 'student'
        response = table.scan(
            FilterExpression=Attr('role').eq('student')
        )
        students = response.get('Items', [])
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }

    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods': 'GET, OPTIONS',
            'Access-Control-Allow-Headers': 'Content-Type'
        },
        'body': json.dumps(students)
    }

```

III. presignedurl

```

import json
import boto3
import uuid

```

```
# Initialize S3 client
s3 = boto3.client("s3")

BUCKET_NAME = "edu-platform-course" # 👈 change if your bucket name is different

def lambda_handler(event, context):
    try:
        # Generate unique course ID
        course_id = f"course-{uuid.uuid4().hex[:10]}"

        # Define file keys
        video_key = f"courses/{course_id}/video.mp4"
        material_key = f"courses/{course_id}/material.pdf"

        # Generate presigned URLs
        video_url = s3.generate_presigned_url(
            "put_object",
            Params={
                "Bucket": BUCKET_NAME,
                "Key": video_key,
                "ContentType": "video/mp4"
            },
            ExpiresIn=604800
        )

        material_url = s3.generate_presigned_url(
            "put_object",
            Params={
                "Bucket": BUCKET_NAME,
                "Key": material_key,
                "ContentType": "application/pdf"
            },
        )
```

```
ExpiresIn=604800
}

# Prepare final response
response_body = {
    "course_id": course_id,
    "urls": {
        "video": {"key": video_key, "url": video_url},
        "material": {"key": material_key, "url": material_url}
    }
}

# Return success
return {
    "statusCode": 200,
    "headers": {
        "Access-Control-Allow-Origin": "*",
        "Access-Control-Allow-Headers": "*",
        "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
    },
    "body": json.dumps(response_body)
}
```

```
except Exception as e:
    print("Error generating presigned URL:", str(e))
    return {
        "statusCode": 500,
        "headers": {"Access-Control-Allow-Origin": "*"},
        "body": json.dumps({"error": str(e)})
    }
```

IV. UploadCourseData

```
import json
import boto3
import os
import time

s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')

BUCKET_NAME = os.environ.get('COURSES_BUCKET', 'edu-platform-course')
TABLE_NAME = os.environ.get('COURSES_TABLE', 'Courses')
table = dynamodb.Table(TABLE_NAME)

def lambda_handler(event, context):
    print("Received event:", event)

    body = event.get("body")
    if isinstance(body, str):
        try:
            body = json.loads(body)
        except Exception:
            pass
    elif isinstance(event, dict):
        body = event

    if not body or not isinstance(body, dict):
        return error_response("Missing request body")

    required = ["course_id", "title", "description", "educator", "video_key", "material_key"]
    missing = [k for k in required if k not in body or not body[k]]
    if missing:
```

```
        return error_response(f"Missing required fields: {missing}")

    try:
        # Generate accessible URLs for playback
        video_url = s3.generate_presigned_url(
            'get_object',
            Params={'Bucket': BUCKET_NAME, 'Key': body["video_key"]},
            ExpiresIn=604800 # 7 days
        )

        material_url = s3.generate_presigned_url(
            'get_object',
            Params={'Bucket': BUCKET_NAME, 'Key': body["material_key"]},
            ExpiresIn=604800
        )

        course_item = {
            "course_id": body["course_id"],
            "title": body["title"],
            "description": body["description"],
            "educator": body["educator"],
            "video_key": body["video_key"],
            "material_key": body["material_key"],
            "video_url": video_url,
            "material_url": material_url,
            "created_at": time.strftime("%Y-%m-%dT%H:%M:%S")
        }

        table.put_item(Item=course_item)
        print("✓ Course saved:", course_item)

    return {
```

```

    "statusCode": 200,
    "headers": cors_headers(),
    "body": json.dumps({
        "message": "Course saved successfully",
        "course_id": body["course_id"]
    })
}

except Exception as e:
    print("✖ Error:", str(e))
    return error_response(str(e), 500)

```

```

def error_response(msg, code=400):
    return {
        "statusCode": code,
        "headers": cors_headers(),
        "body": json.dumps({"error": msg})
    }

```

```

def cors_headers():
    return {
        "Access-Control-Allow-Origin": "*",
        "Access-Control-Allow-Headers": "*",
        "Access-Control-Allow-Methods": "OPTIONS,POST"
    }

```

V. saveCourseMetadata

```

import json
import boto3
import time

dynamodb = boto3.resource('dynamodb')
TABLE_NAME = 'Courses' # replace with your table

```

```
def lambda_handler(event, context):

    try:
        body = json.loads(event['body'])

        required_fields = ["course_id", "title", "description", "video_url", "material_url"]

        if not all(field in body for field in required_fields):
            return {
                "statusCode": 400,
                "body": json.dumps({"error": "Missing required fields"})
            }

        table = dynamodb.Table(TABLE_NAME)
        body["created_at"] = str(time.strftime("%Y-%m-%dT%H:%M:%S"))

        table.put_item(Item=body)

        return {
            "statusCode": 200,
            "headers": {"Access-Control-Allow-Origin": "*"},
            "body": json.dumps({"message": "Course metadata saved"})
        }

    except Exception as e:
        return {
            "statusCode": 500,
            "headers": {"Access-Control-Allow-Origin": "*"},
            "body": json.dumps({"error": str(e)})
        }
```

VI. GetAttendance

```
import json

import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('LMS_Attendance')

def lambda_handler(event, context):

    try:
        class_id = event['queryStringParameters'].get('classId')

        if not class_id:
            return {"statusCode": 400, "body": json.dumps({"error": "classId required"})}

        response = table.query(
            KeyConditionExpression=boto3.dynamodb.conditions.Key('classId').eq(class_id)
        )

        items = response.get('Items', [])

        return {
            "statusCode": 200,
            "headers": {"Access-Control-Allow-Origin": "*"},
            "body": json.dumps(items)
        }

    except Exception as e:
        return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
```

VII. CreateQuizLambda

```
import boto3

import os
import json
import uuid
from datetime import datetime

# Initialize AWS clients
dynamodb = boto3.resource('dynamodb')
sns = boto3.client('sns')

# Environment variables
QUIZZES_TABLE = os.environ.get('QUIZZES_TABLE', 'Quizzes')
SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')

quizzes_table = dynamodb.Table(QUIZZES_TABLE)

def lambda_handler(event, context):
    print("EVENT:", event)

    try:
        # Parse incoming request body
        body = json.loads(event['body'])

        title = body['title']
        questions = body['questions'] # List of {question, options, answer}
        start_time = body['start_time']
        end_time = body['end_time']
        created_by = body.get('created_by', 'educator@example.com')

        # Generate unique quiz ID
        quiz_id = f"quiz_{uuid.uuid4().hex[:8]}"

        # Create quiz item
        quiz_item = {
            'id': quiz_id,
            'title': title,
            'questions': questions,
            'start_time': start_time,
            'end_time': end_time,
            'created_by': created_by
        }

        # Put item into dynamoDB
        quizzes_table.put_item(Item=quiz_item)

        # Publish SNS message
        sns.publish(TopicArn=SNS_TOPIC_ARN, Message=quiz_id)
    except Exception as e:
        print(f"Error: {e}")
        raise
```

```

#  Save quiz in DynamoDB
quizzes_table.put_item(
    Item={
        'quizId': quiz_id,
        'title': title,
        'questions': questions,
        'start_time': start_time,
        'end_time': end_time,
        'created_by': created_by,
        'created_at': datetime.utcnow().isoformat()
    }
)

#  Send SNS Notification to all students, include quizId
if SNS_TOPIC_ARN:
    message = (
        f"🔴 New Quiz Created!\n\n"
        f"Title: {title}\n"
        f"Quiz ID: {quiz_id}\n"
        f"Start Time: {start_time}\n"
        f"End Time: {end_time}\n\n"
        "Use this Quiz ID to take the quiz in your portal."
    )
    sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Subject="New Quiz Available!",
        Message=message
    )
    print(f"✉️ SNS notification sent for quiz {quiz_id}")
else:

```

```

print("⚠ SNS_TOPIC_ARN not set. Skipping notification.")

# Return success response
return {
    'statusCode': 200,
    'headers': {'Content-Type': 'application/json'},
    'body': json.dumps({'quizId': quiz_id, 'message': 'Quiz created and notification sent!'})
}

except Exception as e:
    print("✖ Error:", str(e))
    return {
        'statusCode': 500,
        'body': json.dumps({'error': str(e)})
    }

```

VIII. GetStudentClasses

```

import json
import boto3
from decimal import Decimal

def convert_decimal(obj):
    if isinstance(obj, list):
        return [convert_decimal(i) for i in obj]
    elif isinstance(obj, dict):
        return {k: convert_decimal(v) for k, v in obj.items()}
    elif isinstance(obj, Decimal):
        return int(obj) if obj % 1 == 0 else float(obj)
    return obj

```

```

def lambda_handler(event, context):
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table("LMS_LiveClasses")

    response = table.scan()
    items = response.get("Items", [])
    items = sorted(items, key=lambda x: x.get("schedule_time", x.get("created_at", "")))
    safe_items = convert_decimal(items)

    student_view = []
    for it in safe_items:
        student_view.append({
            "meetingId": it.get("meetingId"),
            "class_name": it.get("class_name") or it.get("topic") or "",
            "join_url": it.get("join_url", ""),
            "schedule_time": it.get("schedule_time", ""),
            "created_at": it.get("created_at", "")
        })

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
        "body": json.dumps(student_view, default=str)
    }

```

IX. GetClassesLambda

```

import json
import boto3
from decimal import Decimal

def convert_decimal(obj):

```

```
if isinstance(obj, list):
    return [convert_decimal(i) for i in obj]

elif isinstance(obj, dict):
    return {k: convert_decimal(v) for k, v in obj.items()}

elif isinstance(obj, Decimal):
    return int(obj) if obj % 1 == 0 else float(obj)

return obj


def lambda_handler(event, context):
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table("LMS_LiveClasses")

    response = table.scan()
    items = response.get("Items", [])
    items = sorted(items, key=lambda x: x.get("created_at", ""), reverse=True)
    safe_items = convert_decimal(items)

    educator_view = []
    for it in safe_items:
        attendance = it.get("attendance") or []
        educator_view.append({
            "meetingId": it.get("meetingId"),
            "class_name": it.get("class_name") or it.get("topic") or "",
            "join_url": it.get("join_url", ""),
            "start_url": it.get("start_url", ""),
            "schedule_time": it.get("schedule_time", ""),
            "created_at": it.get("created_at", ""),
            "attendance_count": len(attendance)
        })
    return {
```

```
        "statusCode": 200,  
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},  
        "body": json.dumps(educator_view, default=str)  
    }  
  
}
```

X. getQuiz

```
import boto3  
  
import os  
  
import json  
  
  
dynamodb = boto3.resource('dynamodb')  
  
QUIZZES_TABLE = os.environ.get('QUIZZES_TABLE', 'Quizzes')  
  
quizzes_table = dynamodb.Table(QUIZZES_TABLE)  
  
  
def lambda_handler(event, context):  
    try:  
        quiz_id = None  
  
        if event.get('queryStringParameters'):  
            quiz_id = event['queryStringParameters'].get('quizId')  
  
  
        if not quiz_id:  
            return {  
                'statusCode': 400,  
                'headers': {'Access-Control-Allow-Origin': '*'},  
                'body': json.dumps({'error': 'Missing quizId'})  
            }  
  
  
        response = quizzes_table.get_item(Key={'quizId': quiz_id})  
        item = response.get('Item')  
  
  
        if not item:
```

```
return {

    'statusCode': 404,
    'headers': {'Access-Control-Allow-Origin': '*'},
    'body': json.dumps({'error': f'Quiz {quiz_id} not found'})

}

questions = item.get('questions', [])
if not isinstance(questions, list):
    questions = []

# ✅ Convert options to array if they are dict
for q in questions:
    opts = q.get('options', [])
    if isinstance(opts, dict): # from createQuiz, options may be {"A": "4", "B": "1"}
        q['options'] = list(opts.values())
    elif isinstance(opts, str):
        q['options'] = opts.split(',')
    elif not isinstance(opts, list):
        q['options'] = []

quiz_data = {
    'quizId': item['quizId'],
    'title': item.get('title', ''),
    'questions': questions
}

return {
    'statusCode': 200,
    'headers': {
        'Access-Control-Allow-Origin': '*',
        'Content-Type': 'application/json'
    }
}
```

```
        },
        'body': json.dumps(quiz_data)
    }
}
```

except Exception as e:

```
    print("Error:", str(e))
    return {
        'statusCode': 500,
        'headers': {'Access-Control-Allow-Origin': '*'},
        'body': json.dumps({'error': str(e)})
    }
```

XI. CreateLiveClass

```
import os
import json
import uuid
import boto3
import requests
from datetime import datetime
from requests.auth import HTTPBasicAuth

ZOOM_ACCOUNT_ID = os.getenv("ZOOM_ACCOUNT_ID")
ZOOM_CLIENT_ID = os.getenv("ZOOM_CLIENT_ID")
ZOOM_CLIENT_SECRET = os.getenv("ZOOM_CLIENT_SECRET")
TABLE_NAME = os.getenv("DYNAMODB_TABLE", "LMS_LiveClasses")

dynamodb = boto3.resource("dynamodb")
table = dynamodb.Table(TABLE_NAME)

def lambda_handler(event, context):
    try:
        body = json.loads(event.get("body", "{}"))
    
```

```
class_name = body.get("class_name") or body.get("topic") or "Untitled Class"

# schedule_time expects ISO 8601 (e.g. "2025-11-04T10:00:00")

schedule_time = body.get("schedule_time", "")

# If schedule_time provided, use it as Zoom start_time, else create instant meeting (type 1)

meeting_type = 2 if schedule_time else 1

token_url = "https://zoom.us/oauth/token"

data = {"grant_type": "account_credentials", "account_id": ZOOM_ACCOUNT_ID}

token_resp = requests.post(
    token_url,
    data=data,
    auth=HTTPBasicAuth(ZOOM_CLIENT_ID, ZOOM_CLIENT_SECRET),
    timeout=10
)

if token_resp.status_code != 200:
    return {"statusCode": token_resp.status_code, "body": json.dumps({"error": "Failed to get token", "details": token_resp.text})}

access_token = token_resp.json().get("access_token")

if not access_token:
    return {"statusCode": 400, "body": json.dumps({"error": "Missing Zoom access token"})}

headers = {"Authorization": f"Bearer {access_token}", "Content-Type": "application/json"}

meeting_data = {
    "topic": class_name,
    "type": meeting_type,
    "duration": int(body.get("duration", 45)),
    "timezone": body.get("timezone", "UTC"),
    "settings": {
```

```
        "host_video": True,
        "participant_video": True,
        "join_before_host": False
    }
}

if schedule_time:
    # Zoom expects RFC3339 (with timezone); assume schedule_time is local ISO without Z
    # If schedule_time already contains timezone info, leave it; otherwise append Z to treat as UTC
    if schedule_time.endswith("Z") or "+" in schedule_time or "-" in schedule_time[-6:]:
        meeting_data["start_time"] = schedule_time
    else:
        meeting_data["start_time"] = schedule_time + "Z"

zoom_resp = requests.post("https://api.zoom.us/v2/users/me/meetings", headers=headers,
json=meeting_data, timeout=10)

if zoom_resp.status_code not in (201, 200):
    return {"statusCode": zoom_resp.status_code, "body": json.dumps({"error": "Zoom meeting creation failed", "details": zoom_resp.text})}

meeting = zoom_resp.json()
meeting_id = int(meeting.get("id", 0))
join_url = meeting.get("join_url", "")
start_url = meeting.get("start_url", "")

# store item in DynamoDB
item = {
    "meetingId": meeting_id,
    "class_name": class_name,
    "join_url": join_url,
    "start_url": start_url,
    "schedule_time": schedule_time,
    "created_at": datetime.utcnow().isoformat(),
```

```

        "attendance": [] # initialize empty attendance list
    }

    table.put_item(Item=item)

    return {"statusCode": 200, "body": json.dumps({"message": "Meeting created successfully!",
"meetingId": meeting_id, "join_url": join_url, "start_url": start_url})}

except Exception as e:

    return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
```

XII. getCoursesList

```

import json

import boto3

from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource("dynamodb")

table_name = "Courses" # <-- change to your exact table name

table = dynamodb.Table(table_name)

def lambda_handler(event, context):

    try:

        # Scan the table to get all courses

        response = table.scan()

        courses = response.get("Items", [])

        return {

            "statusCode": 200,

            "headers": {

                "Access-Control-Allow-Origin": "*",

                "Access-Control-Allow-Methods": "GET,OPTIONS",

                "Access-Control-Allow-Headers": "Content-Type"
            }
        }
    
```

```
        },
        "body": json.dumps(courses)
    }
}
```

except Exception as e:

```
    return {
        "statusCode": 500,
        "headers": {"Access-Control-Allow-Origin": "*"},
        "body": json.dumps({"error": str(e)})
    }
```

XIII. recordattendance

```
import json
import boto3
from datetime import datetime, timedelta

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('LMS_Attendance')

def lambda_handler(event, context):
    try:
        body = json.loads(event.get('body', '{}'))
        class_id = str(body.get('classId'))
        student_name = body.get('studentName')

        if not class_id or not student_name:
            return {"statusCode": 400, "body": json.dumps({"error": "Missing classId or studentName"})}

        # Convert current UTC time to IST
        ist_time = datetime.utcnow() + timedelta(hours=5, minutes=30)
```

```

join_time_ist = ist_time.strftime("%Y-%m-%d %I:%M:%S %p")

table.put_item(Item={
    "classId": class_id,
    "studentName": student_name,
    "joinTimeIST": join_time_ist
})

return {
    "statusCode": 200,
    "headers": {
        "Access-Control-Allow-Origin": "*"
    },
    "body": json.dumps({"message": "Attendance recorded", "joinTimeIST": join_time_ist})
}

except Exception as e:
    return {"statusCode": 500, "body": json.dumps({"error": str(e)})}


```

XIV. studentprofile

```

import json

import base64

import urllib.parse

import urllib.request

import urllib.error

import boto3

# === Configure these ===

DOMAIN = "https://us-east-1tj9jinyqx.auth.us-east-1.amazoncognito.com"

CLIENT_ID = "7cdmtqipvq30pdoe17hv7h6n0s"

```

```
CLIENT_SECRET = "1kdv9d9mb5taha5cf8qun0b0hg2hqt6gmh2rr2a5h5korh1hsbdn" # keep secret in  
env; None if public client  
  
REDIRECT_URI = "https://main.dijffme8w1boe.amplifyapp.com/index-1.html"  
  
# DynamoDB setup  
dynamodb = boto3.resource("dynamodb")  
table = dynamodb.Table("cognitousers")  
  
# ======  
  
def decode_jwt_payload(token):  
    try:  
        parts = token.split('.')  
        if len(parts) < 2:  
            return {}  
        payload_b64 = parts[1] + '=' * (-len(parts[1]) % 4)  
        decoded_bytes = base64.urlsafe_b64decode(payload_b64)  
        return json.loads(decoded_bytes.decode('utf-8'))  
    except Exception:  
        return {}  
  
def fetch_userinfo(access_token):  
    try:  
        url = f"{DOMAIN.rstrip('/')}/oauth2/userInfo"  
        headers = {"Authorization": f"Bearer {access_token}"}  
        req = urllib.request.Request(url, headers=headers, method="GET")  
        with urllib.request.urlopen(req, timeout=10) as resp:  
            return json.loads(resp.read().decode())  
    except Exception:  
        return {}
```

```
def make_response(status, body):
    return {
        "statusCode": status,
        "headers": {
            "Access-Control-Allow-Origin": "*",
            "Access-Control-Allow-Methods": "GET,POST,OPTIONS",
            "Access-Control-Allow-Headers": "Content-Type,Authorization"
        },
        "body": json.dumps(body)
    }

def lambda_handler(event, context):
    try:
        # Support API Gateway style request
        code = None
        if isinstance(event, dict):
            qs = event.get("queryStringParameters") or {}
            code = qs.get("code") or event.get("code")

        if not code:
            return make_response(400, {"error": "Missing authorization code"})

        token_url = f"{DOMAIN.rstrip('/')} oauth2/token"
        data = {
            "grant_type": "authorization_code",
            "client_id": CLIENT_ID,
            "code": code,
            "redirect_uri": REDIRECT_URI
        }
        encoded_creds = base64.b64encode(f"{CLIENT_ID}:{CLIENT_SECRET}".encode()).decode()
        headers = {
```

```
"Content-Type": "application/x-www-form-urlencoded",
"Authorization": f"Basic {encoded_creds}"
}

req = urllib.request.Request(token_url, data=urllib.parse.urlencode(data).encode(),
headers=headers, method="POST")

with urllib.request.urlopen(req, timeout=10) as resp:
    tokens = json.loads(resp.read().decode())

    id_token = tokens.get("id_token")
    access_token = tokens.get("access_token")

if not id_token:
    return make_response(400, {"error": "Missing id_token"})

decoded = decode_jwt_payload(id_token)
email = decoded.get("email")

profile = {
    "name": decoded.get("name"),
    "email": email,
    "username": decoded.get("cognito:username") or decoded.get("username") or
    decoded.get("sub")
}

# Fallback to userInfo if name missing
if not profile.get("name") and access_token:
    ui = fetch_userinfo(access_token)
    if ui.get("name"):
        profile["name"] = ui["name"]

# ♦ Fallback to DynamoDB lookup if still missing name
```

```

if profile.get("email") and not profile.get("name"):

    try:

        response = table.get_item(Key={"email": profile["email"]})

        if "Item" in response:

            profile["name"] = response["Item"].get("name", "Unknown")

        else:

            print(f"No DynamoDB record found for {profile['email']}")

    except Exception as e:

        print(f"DynamoDB lookup failed: {e}")

return make_response(200, {"profile": profile, "decoded_token": decoded})

except Exception as e:

    return make_response(500, {"error": str(e)})

```

XV. submitQuiz

```

import boto3

import os

import json

from datetime import datetime

dynamodb = boto3.resource('dynamodb')

QUIZZES_TABLE = os.environ.get('QUIZZES_TABLE', 'Quizzes')

QUIZ_RESULTS_TABLE = os.environ.get('QUIZ_RESULTS_TABLE', 'QuizResults')

quizzes_table = dynamodb.Table(QUIZZES_TABLE)

results_table = dynamodb.Table(QUIZ_RESULTS_TABLE)

```

```

def lambda_handler(event, context):

    try:

        body = json.loads(event['body'])

        quiz_id = body['quizId']

```

```

student_name = body['studentName']
student_email = body['studentEmail']
answers = body['answers'] # list of selected options

# Fetch quiz
quiz_data = quizzes_table.get_item(Key={'quizId': quiz_id}).get('Item')
if not quiz_data:
    return {'statusCode': 404, 'body': json.dumps({'error':'Quiz not found'})}

questions = quiz_data['questions']

# Compute score
score = 0
for i, q in enumerate(questions):
    correct_option_index = q['answer'] # assuming 'answer' stores index of correct option
    if i < len(answers) and answers[i] == q['options'][correct_option_index]:
        score += 1

# Store result
results_table.put_item(
    Item={
        'quizId': quiz_id,
        'studentEmail': student_email,
        'name': student_name,
        'score': score,
        'submitted_at': datetime.utcnow().isoformat()
    }
)

return {
    'statusCode': 200,

```

```
'headers': {'Access-Control-Allow-Origin': '*' , 'Content-Type': 'application/json'},  
'body': json.dumps({'score': score})  
}  
  
except Exception as e:  
  
    print("Error:", str(e))  
  
    return {  
  
        'statusCode': 500,  
  
        'headers': {'Access-Control-Allow-Origin': '*'},  
  
        'body': json.dumps({'error': str(e)})  
    }
```

XVI. getCourseContent

```
import json  
  
import boto3  
  
  
dynamodb = boto3.resource("dynamodb")  
table = dynamodb.Table("Courses") # your table name  
  
  
def lambda_handler(event, context):  
  
    print("Received event:", event) # helpful for debugging  
  
  
    # Get course_id from pathParameters  
    course_id = event.get("pathParameters", {}).get("id")  
  
  
    if not course_id:  
  
        return {  
            "statusCode": 400,  
            "headers": {  
                "Access-Control-Allow-Origin": "*",  
                "Access-Control-Allow-Methods": "GET,OPTIONS",
```

```
        "Access-Control-Allow-Headers": "Content-Type"
    },
    "body": json.dumps({"error": "Missing course_id"})
}

try:
    response = table.get_item(Key={"course_id": course_id})
    item = response.get("Item")

    if not item:
        return {
            "statusCode": 404,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Methods": "GET,OPTIONS",
                "Access-Control-Allow-Headers": "Content-Type"
            },
            "body": json.dumps({"error": "Course not found"})
        }

    return {
        "statusCode": 200,
        "headers": {
            "Access-Control-Allow-Origin": "*",
            "Access-Control-Allow-Methods": "GET,OPTIONS",
            "Access-Control-Allow-Headers": "Content-Type"
        },
        "body": json.dumps(item)
    }

except Exception as e:
```

```
        return {  
            "statusCode": 500,  
            "headers": {"Access-Control-Allow-Origin": "*"},  
            "body": json.dumps({"error": str(e)})  
        }
```

XVII. getQuizResult

```
import boto3  
  
import os  
  
import json  
  
from decimal import Decimal  
  
  
dynamodb = boto3.resource('dynamodb')  
  
QUIZ_RESULTS_TABLE = os.environ.get('QUIZ_RESULTS_TABLE', 'QuizResults')  
  
results_table = dynamodb.Table(QUIZ_RESULTS_TABLE)  
  
  
def decimal_default(obj):  
    if isinstance(obj, Decimal):  
        return float(obj)  
    raise TypeError  
  
  
def lambda_handler(event, context):  
    try:  
        quiz_id = None  
  
        if event.get('queryStringParameters'):   
            quiz_id = event['queryStringParameters'].get('quizId')  
  
        if not quiz_id:  
            return {  
                'statusCode': 400,  
                'headers': {'Access-Control-Allow-Origin': '*'},
```

```

        'body': json.dumps({'error': 'Missing quizId'})

    }

# Query all results for this quiz
response = results_table.query(
    KeyConditionExpression=boto3.dynamodb.conditions.Key('quizId').eq(quiz_id)
)

results = response.get('Items', [])

return {
    'statusCode': 200,
    'headers': {'Access-Control-Allow-Origin': '*','Content-Type':'application/json'},
    'body': json.dumps({'results': results}, default=decimal_default)
}

except Exception as e:
    print("Error:", str(e))
    return {
        'statusCode': 500,
        'headers': {'Access-Control-Allow-Origin': '*'},
        'body': json.dumps({'error': str(e)})
    }

```

XVIII. getAllQuizes

```

import boto3

import os

import json

from boto3.dynamodb.conditions import Key, Attr

from decimal import Decimal

```

```
dynamodb = boto3.resource('dynamodb')
QUIZZES_TABLE = os.environ.get('QUIZZES_TABLE', 'Quizzes')
quizzes_table = dynamodb.Table(QUIZZES_TABLE)

# Helper to convert DynamoDB items to JSON (Decimal -> int/float)
def decimal_to_native(obj):

    if isinstance(obj, list):
        return [decimal_to_native(i) for i in obj]

    elif isinstance(obj, dict):
        return {k: decimal_to_native(v) for k, v in obj.items()}

    elif isinstance(obj, Decimal):
        if obj % 1 == 0:
            return int(obj)
        else:
            return float(obj)

    else:
        return obj

def lambda_handler(event, context):

    try:
        # Scan all items in the Quizzes table
        response = quizzes_table.scan()
        items = response.get('Items', [])

        # Convert any Decimal types to native Python types
        quizzes = [decimal_to_native(q) for q in items]

        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
            },
            'body': quizzes
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': str(e)
        }


```

```
'Content-Type': 'application/json'  
},  
'body': json.dumps({'quizzes': quizzes})  
}  
  
except Exception as e:
```

```
    print("Error:", str(e))  
    return {  
        'statusCode': 500,  
        'headers': {'Access-Control-Allow-Origin': '*'},  
        'body': json.dumps({'error': str(e)})  
    }
```

References

- (i) https://docs.amplify.aws/react/reference/amplify_outputs/
- (ii) <https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html?>
- (iii) <https://aws.amazon.com/textract/resources/>
- (iv) <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.Lambda.html?>
- (v) <https://dev.to/frosnerd/event-handling-in-aws-using-sns-sqs-and-lambda-2ng?>