Deploying Serverless Web Application For Student registration on AWS: S3, API Gateway, Lambda, DynamoDB, and CloudFrontTeam

Team Members:

Mandha Sai Kumar Reddy - 700765227

Rohit Bandarupalli - 700762469

Ganesh Kumar Korra – 700761716(CRN30651)

Motivation

The deployment of a serverless web application on AWS using services like S3, API Gateway, Lambda, DynamoDB, and CloudFront represents a significant leap towards modern, scalable, and cost-efficient cloud architecture. By leveraging these cutting-edge technologies, our team has harnessed the power of serverless computing, which not only reduces the operational burden but also allows us to focus more on innovation and delivering value to our users.

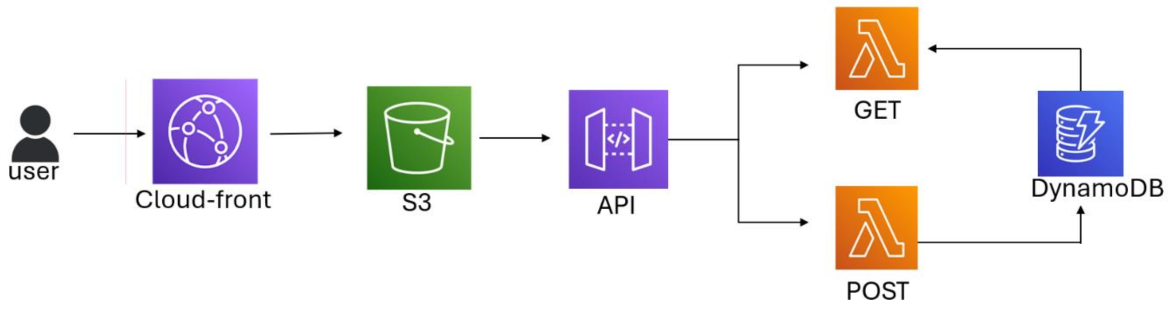
This project showcases our commitment to staying ahead in the rapidly evolving tech landscape. The choice of serverless architecture ensures that our application can scale seamlessly in response to varying user demands, while also optimizing costs by charging only for actual usage. This flexibility and efficiency are crucial in today’s competitive environment, where agility and resource management can make all the difference.

Moreover, the integration of DynamoDB for scalable and fast data storage, along with CloudFront for content delivery, ensures that our users experience high performance and low latency, regardless of their location. By choosing AWS’s robust ecosystem, we have ensured that our application is built on a foundation of reliability, security, and global reach.

In essence, this project is not just a technical achievement but a strategic decision that aligns with our vision of delivering top-notch, user-centric solutions. It reflects our dedication to leveraging the latest technologies to create innovative, reliable, and scalable applications that can grow and evolve with our users' needs.

This is what made me curious to take up this topic for my project.

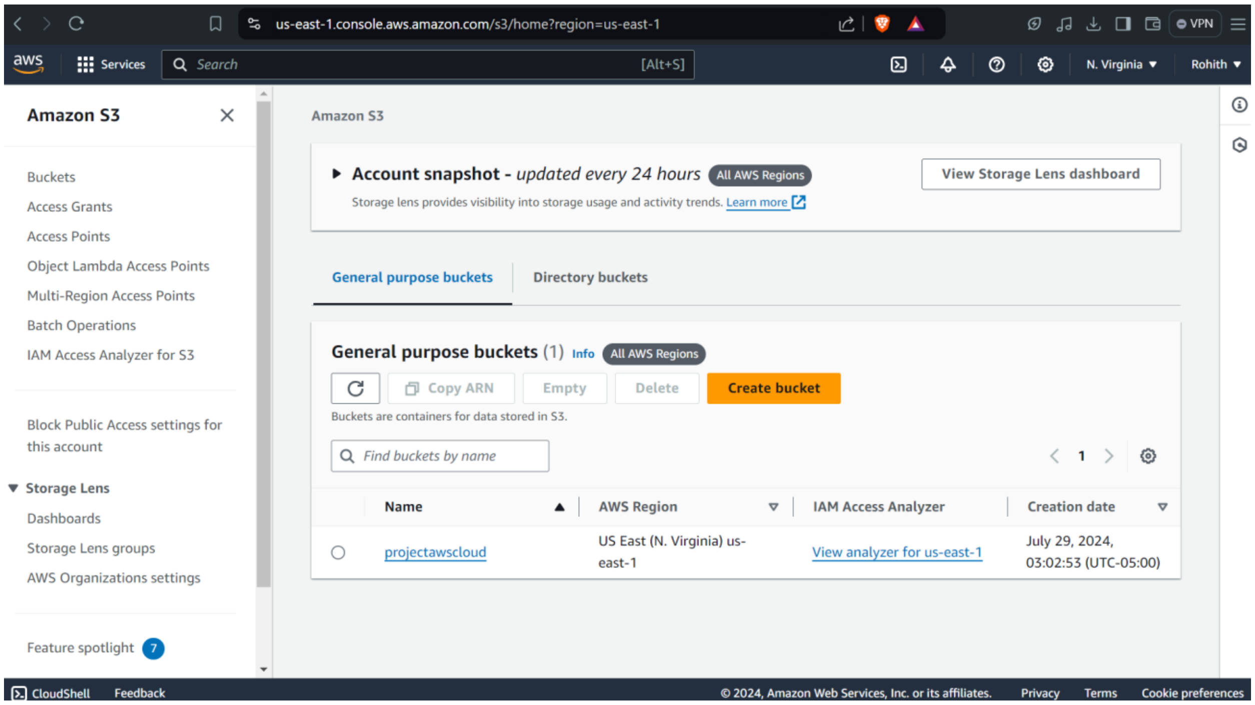
System Architecture



1. Amazon S3 (Simple Storage Service):

Purpose: Serves as the static file storage and hosting solution for the web application's frontend assets, such as HTML, CSS, JavaScript, images, and other media.

Benefits: Provides scalable, durable, and highly available storage with integrated security features. The content is distributed globally, offering high availability and low latency access.



Amazon S3

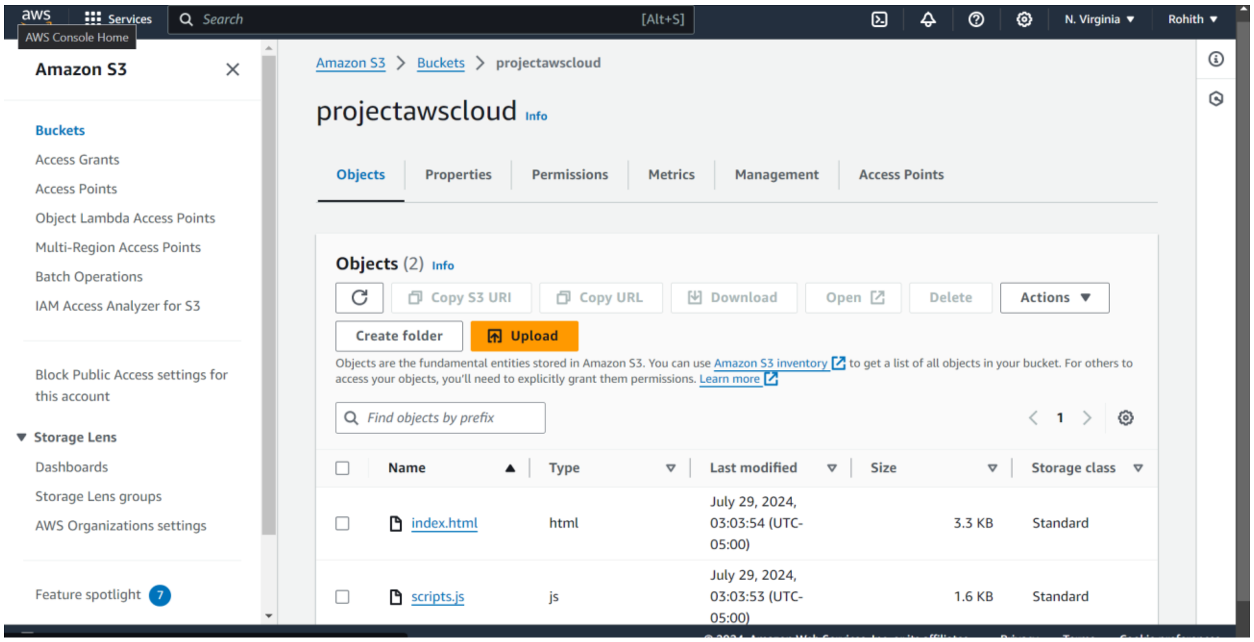
Account snapshot - *updated every 24 hours* All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

	Name	AWS Region	IAM Access Analyzer	Creation date
○	projectawscloud	US East (N. Virginia) us-east-1	View analyzer for us-east-1	July 29, 2024, 03:02:53 (UTC-05:00)



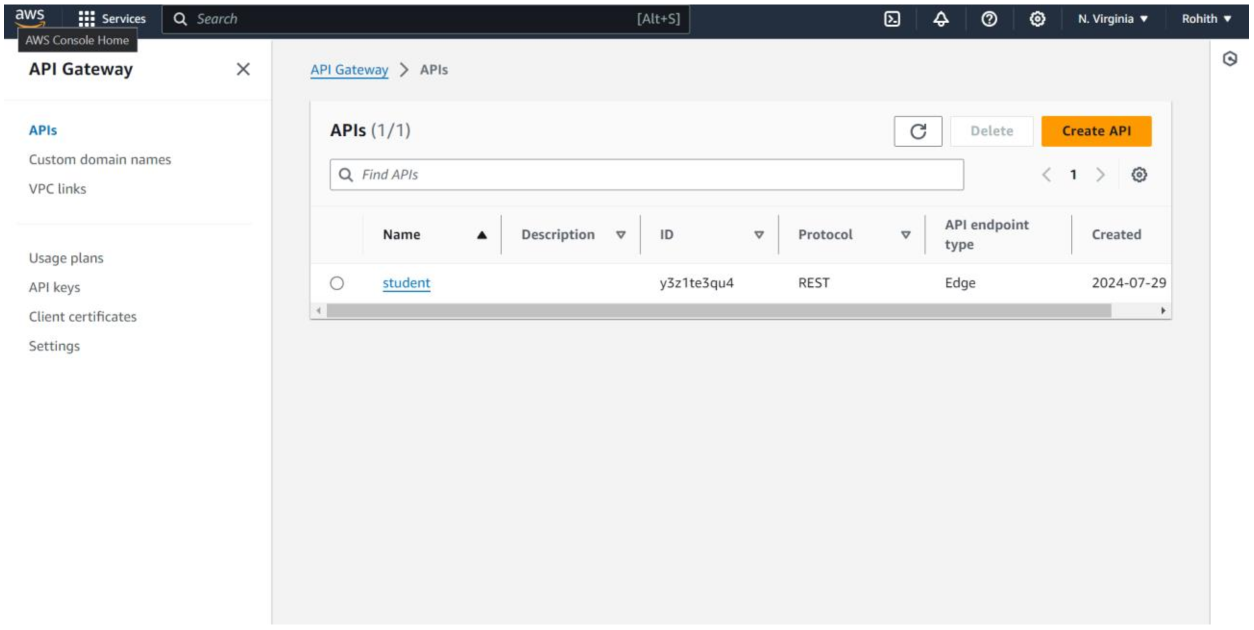
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	July 29, 2024, 03:03:54 (UTC-05:00)	3.3 KB	Standard
scripts.js	js	July 29, 2024, 03:03:53 (UTC-05:00)	1.6 KB	Standard

2. Amazon API Gateway:

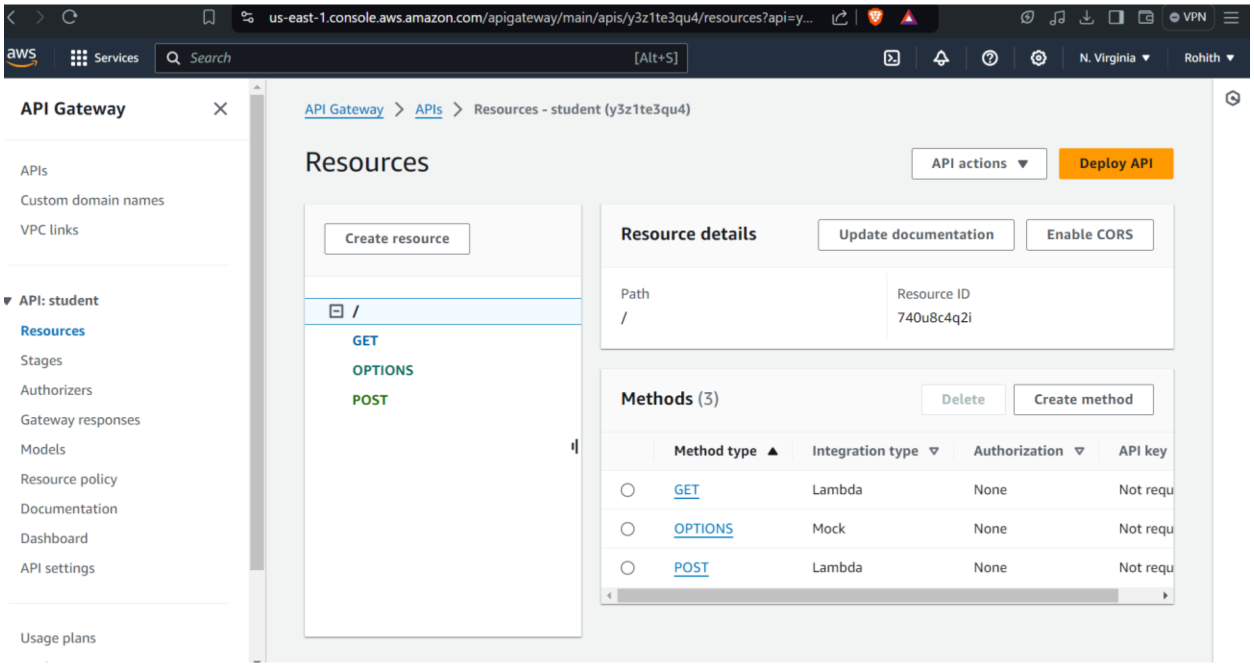
Purpose: Acts as the front door for the application's backend logic. It handles all the API requests from the frontend, routing them to the appropriate AWS Lambda functions.

Benefits: Provides a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It supports various protocols and integrates seamlessly with AWS services.


API Gateway > APIs

APIs (1/1)

	Name	Description	ID	Protocol	API endpoint type	Created
○	student		y3z1te3qu4	REST	Edge	2024-07-29


us-east-1.console.aws.amazon.com/apigateway/main/apis/y3z1te3qu4/resources?api=y...

API Gateway > APIs > Resources - student (y3z1te3qu4)

Resources

/
GET
OPTIONS
POST

Resource details

Path	Resource ID
/	740u8c4q2i

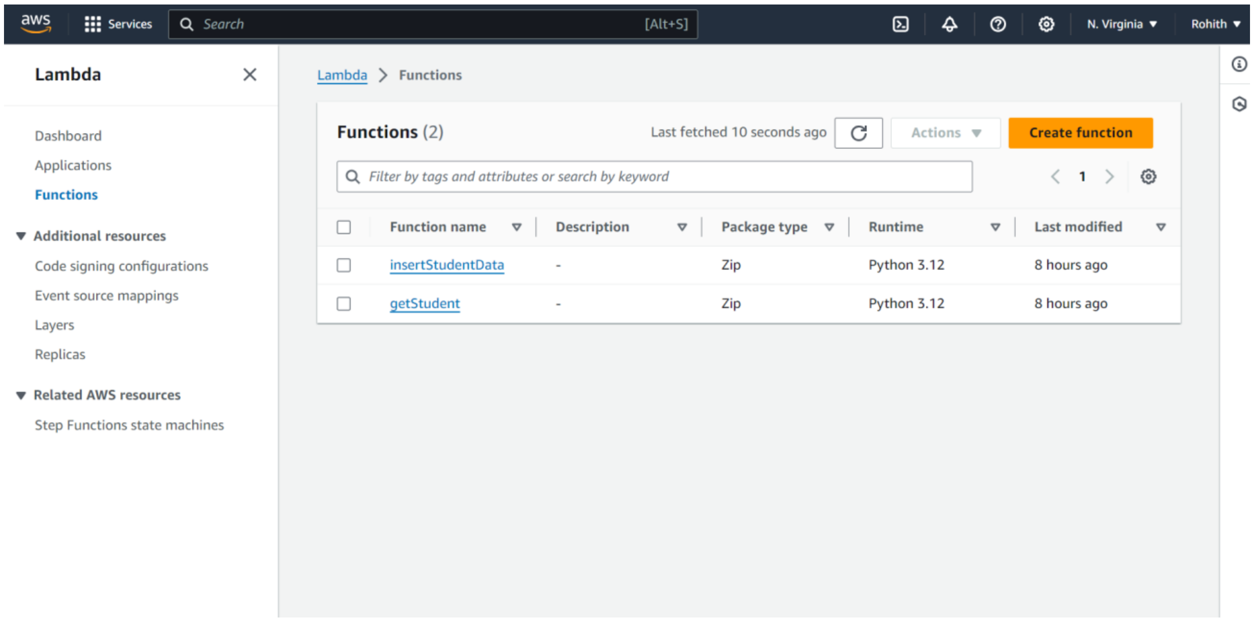
Methods (3)

	Method type	Integration type	Authorization	API key
○	GET	Lambda	None	Not requ
○	OPTIONS	Mock	None	Not requ
○	POST	Lambda	None	Not requ

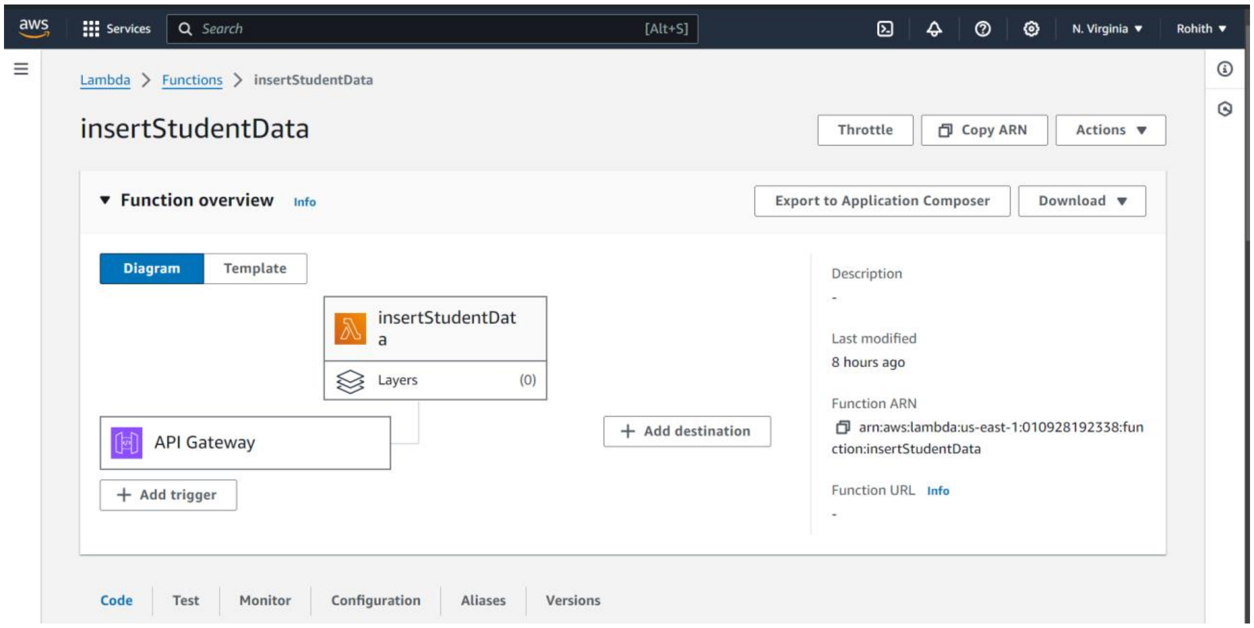
3. AWS Lambda:

Purpose: Contains the business logic of the application. Lambda functions are triggered by API Gateway requests or other AWS services to execute specific tasks, such as processing requests, interacting with the database, or performing backend calculations.

Benefits: Serverless compute service that automatically scales with the load, with a pay-as-you-go pricing model. It eliminates the need to manage servers, allowing the team to focus on code and functionality.



Function name	Description	Package type	Runtime	Last modified
insertStudentData	-	Zip	Python 3.12	8 hours ago
getStudent	-	Zip	Python 3.12	8 hours ago

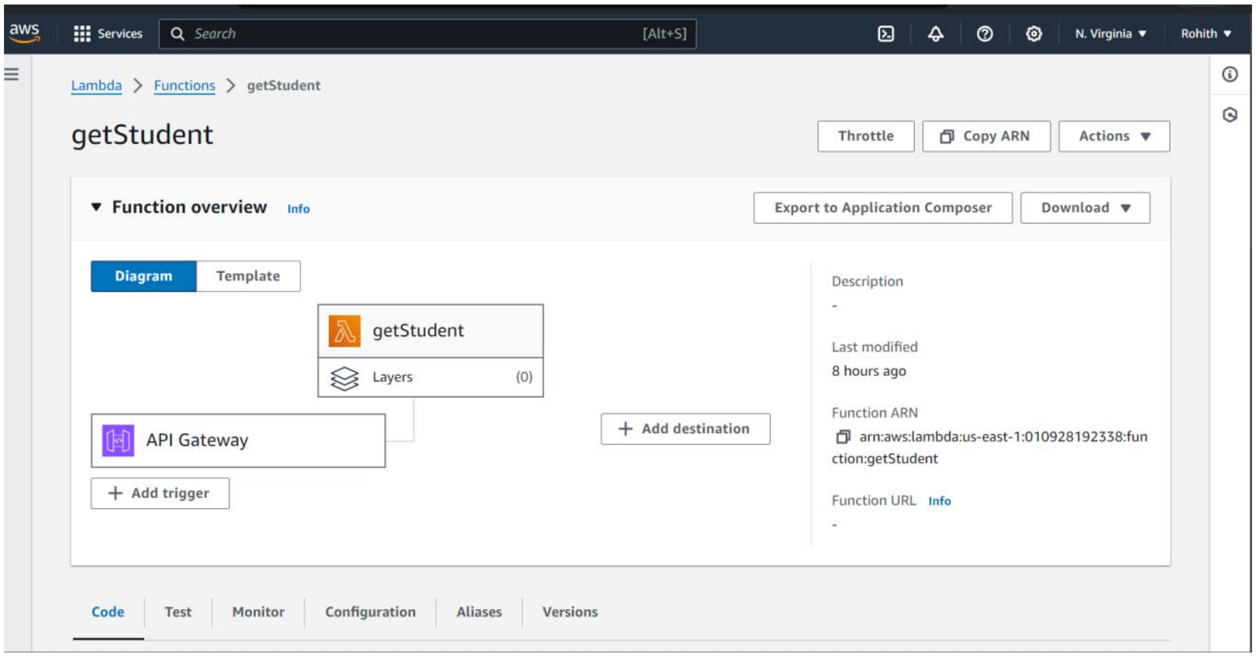


Description
-

Last modified
8 hours ago

Function ARN
arn:aws:lambda:us-east-1:010928192338:function:insertStudentData

Function URL Info
-



getStudent

Description
-

Last modified
8 hours ago

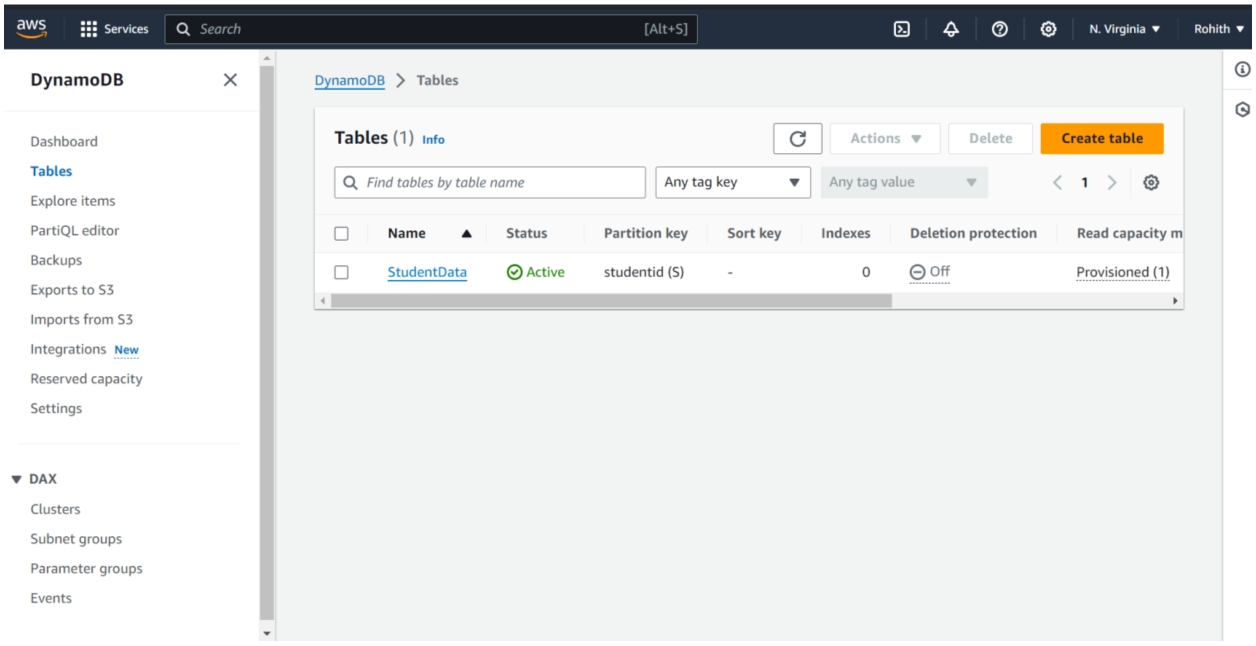
Function ARN
arn:aws:lambda:us-east-1:010928192338:function:getStudent

Function URL Info
-

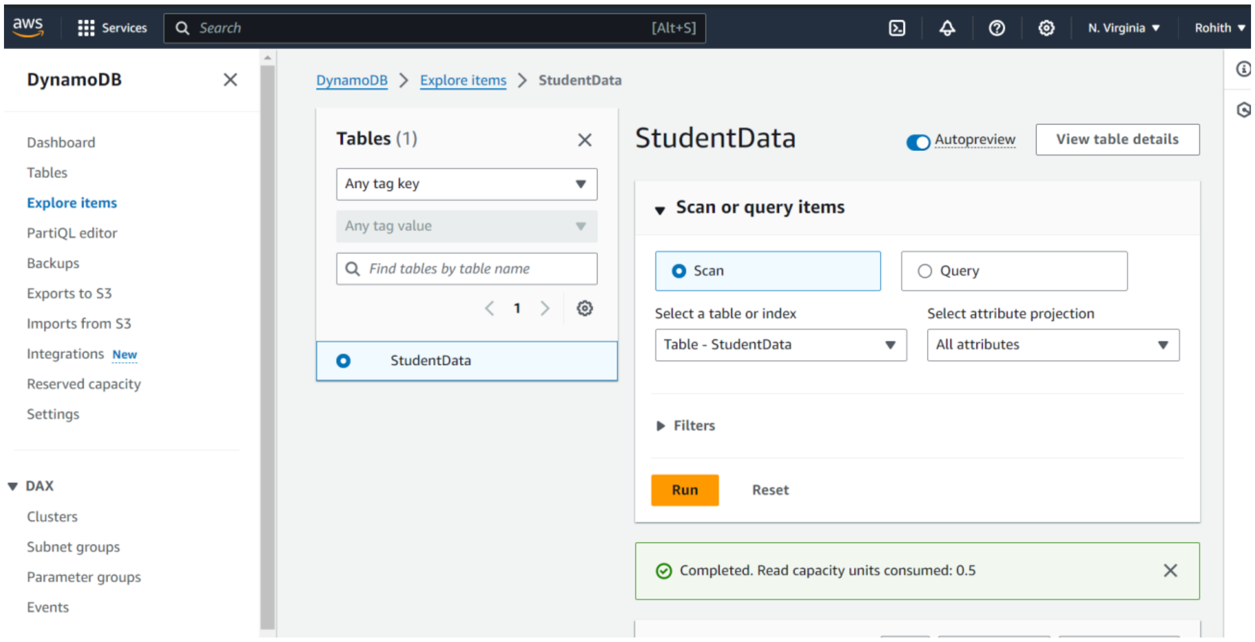
4. Amazon DynamoDB:

Purpose: Provides a NoSQL database solution to store and manage application data. It handles user data, application state, and other dynamic information required by the application.

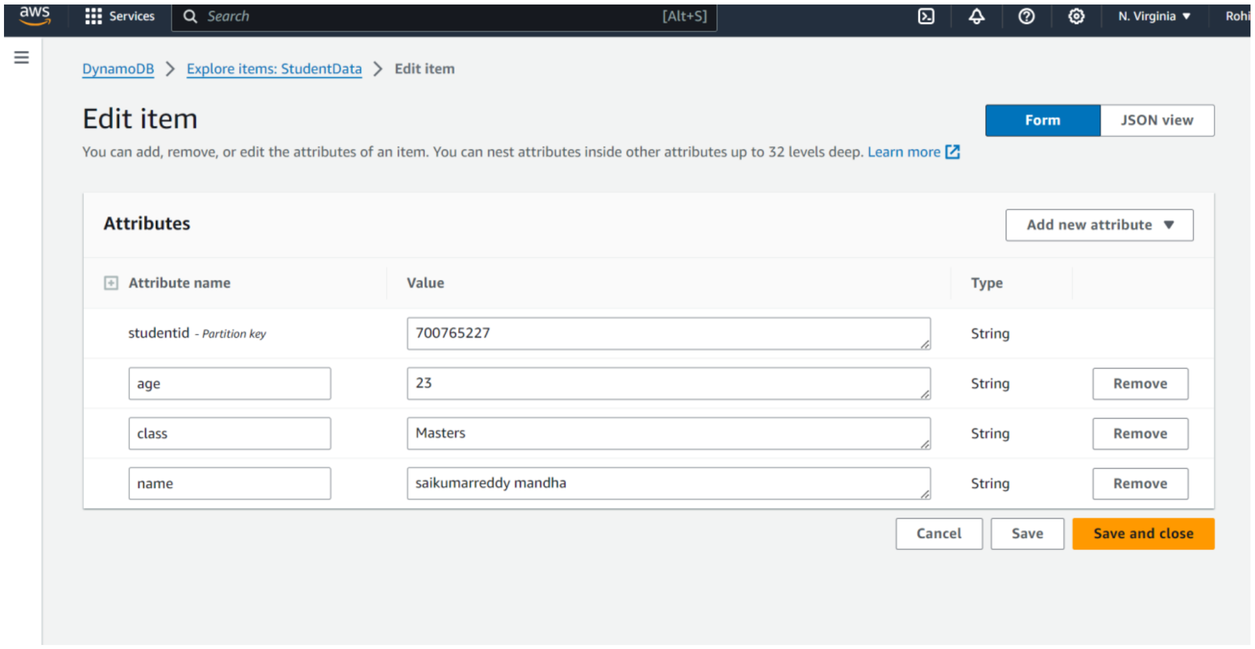
Benefits: A fully managed, scalable, and highly available database service that offers low latency and supports automatic scaling. It is designed for high performance and seamless integration with other AWS services.



Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity m
StudentData	Active	studentid (S)	-	0	Off	Provisioned (1)



Completed. Read capacity units consumed: 0.5



You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. Learn more

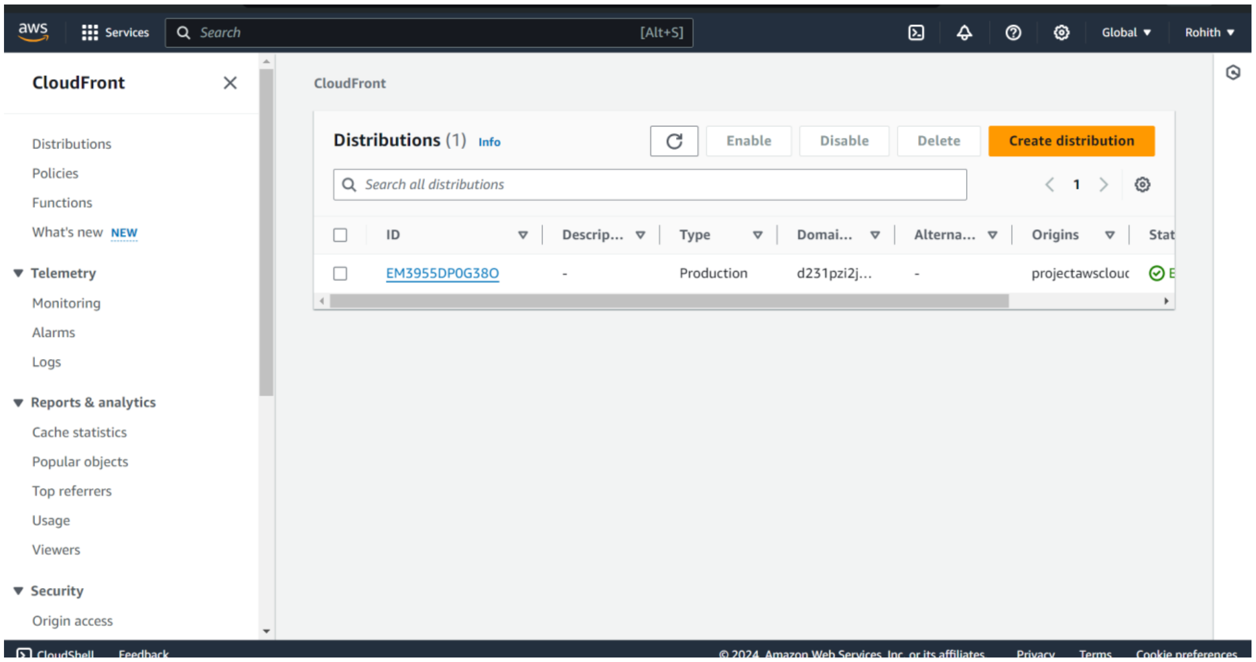
Attributes

Attribute name	Value	Type	
studentid - *Partition key*	700765227	String	
age	23	String	Remove
class	Masters	String	Remove
name	saikumarreddy mandha	String	Remove

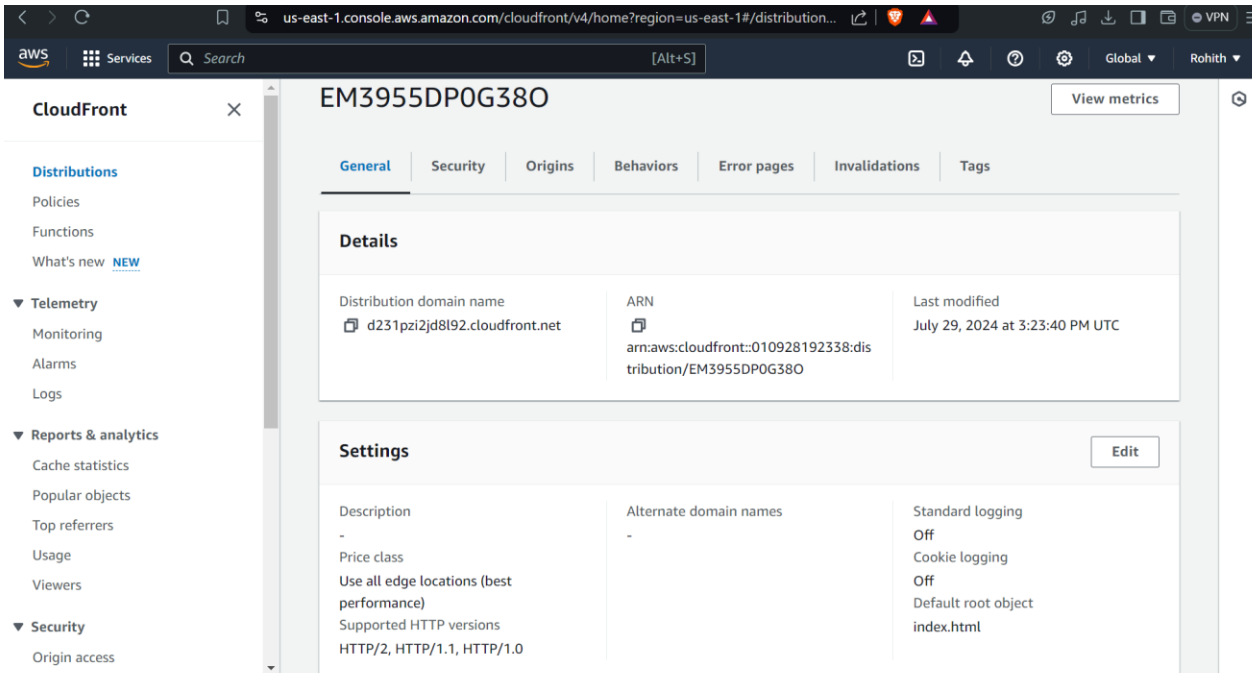
5. Amazon CloudFront:

Purpose: Content Delivery Network (CDN) that distributes content from S3 or dynamic content served by API Gateway and Lambda. It accelerates the delivery of static and dynamic web content to users around the globe.

Benefits: Provides a secure and scalable way to deliver content with low latency and high transfer speeds. It integrates with S3, API Gateway, and Lambda, offering comprehensive security features such as DDoS protection and encryption.



ID	Descrip...	Type	Domai...	Alterna...	Origins	Stat
EM3955DP0G38O	-	Production	d231pzi2j...	-	projectawscloud	E



EM3955DP0G38O

Details

Distribution domain name: d231pzi2jd8l92.cloudfront.net
ARN: arn:aws:cloudfront::010928192338:distribution/EM3955DP0G38O
Last modified: July 29, 2024 at 3:23:40 PM UTC

Settings

Description: -
Price class: Use all edge locations (best performance)
Supported HTTP versions: HTTP/2, HTTP/1.1, HTTP/1.0
Alternate domain names: -
Standard logging: Off
Cookie logging: Off
Default root object: index.html

6. Security and Monitoring:

Services Involved: AWS Identity and Access Management (IAM), AWS WAF (Web Application Firewall), Amazon CloudWatch, AWS CloudTrail.

Purpose: Ensure security and compliance, monitor application performance, and track user activities. IAM controls access and permissions, WAF protects against common web exploits, CloudWatch provides monitoring and logging, and CloudTrail tracks API usage for auditing.

7. Additional Considerations:

Scalability: The architecture is inherently scalable due to its serverless nature, allowing each component to scale independently based on demand.

Cost-Effectiveness: The pay-per-use model for services like Lambda, API Gateway, and DynamoDB ensures that costs are aligned with actual usage, minimizing waste and optimizing budget.

Features

1. Static Website Hosting: Feature:

Host and deliver static content such as HTML, CSS, JavaScript, images, and other media files.

Implementation: Using Amazon S3 for storage and CloudFront for content delivery, ensuring fast and reliable access for users globally.

2. Dynamic Content Handling:

Feature: Serve dynamic content and process user requests.

Implementation: API Gateway routes user requests to AWS Lambda functions, which execute the business logic and interact with other services like DynamoDB.

3. Serverless Compute:

Feature: Execute backend logic without managing servers.

Implementation: AWS Lambda functions are triggered by events such as API Gateway requests, providing scalable and cost-efficient compute power.

4. Data Management and Storage:

Feature: Store and retrieve application data.

Implementation: Amazon DynamoDB provides a scalable NoSQL database solution, allowing for flexible data modeling and high performance.

5. Content Delivery and Caching:

Feature: Distribute static and dynamic content efficiently to users around the world.

Implementation: Amazon CloudFront acts as a global CDN, reducing latency and ensuring fast delivery of content by caching it at edge locations.

6. Security and Access Control:

Feature: Secure the application and manage user access.

Implementation: AWS Identity and Access Management (IAM) for user permissions, AWS WAF for web application firewall protection, and encryption for data in transit and at rest.

7. Scalability and High Availability:

Feature: Automatically scale resources to handle varying traffic loads.

Implementation: The serverless architecture with Lambda, API Gateway, S3, and DynamoDB inherently supports automatic scaling and high availability, ensuring reliability even under peak loads.

8.Monitoring and Logging:

Feature: Monitor application performance and track events.

Implementation: Amazon CloudWatch for real-time monitoring, logging, and alerting, and AWS CloudTrail for auditing API calls and changes to the environment.

9.Cost-Optimization:

Feature: Minimize costs while maintaining performance and scalability.

Implementation: Pay-as-you-go pricing model for AWS services, with automated scaling and resource management to prevent over-provisioning and reduce waste.

10. User Authentication and Authorization:

Feature: Securely manage user identities and permissions.

Implementation: Integration with Amazon Cognito or third-party identity providers for authentication and authorization, enabling features like user registration, login, and multi-factor authentication.

11. Automated Deployments and CI/CD:

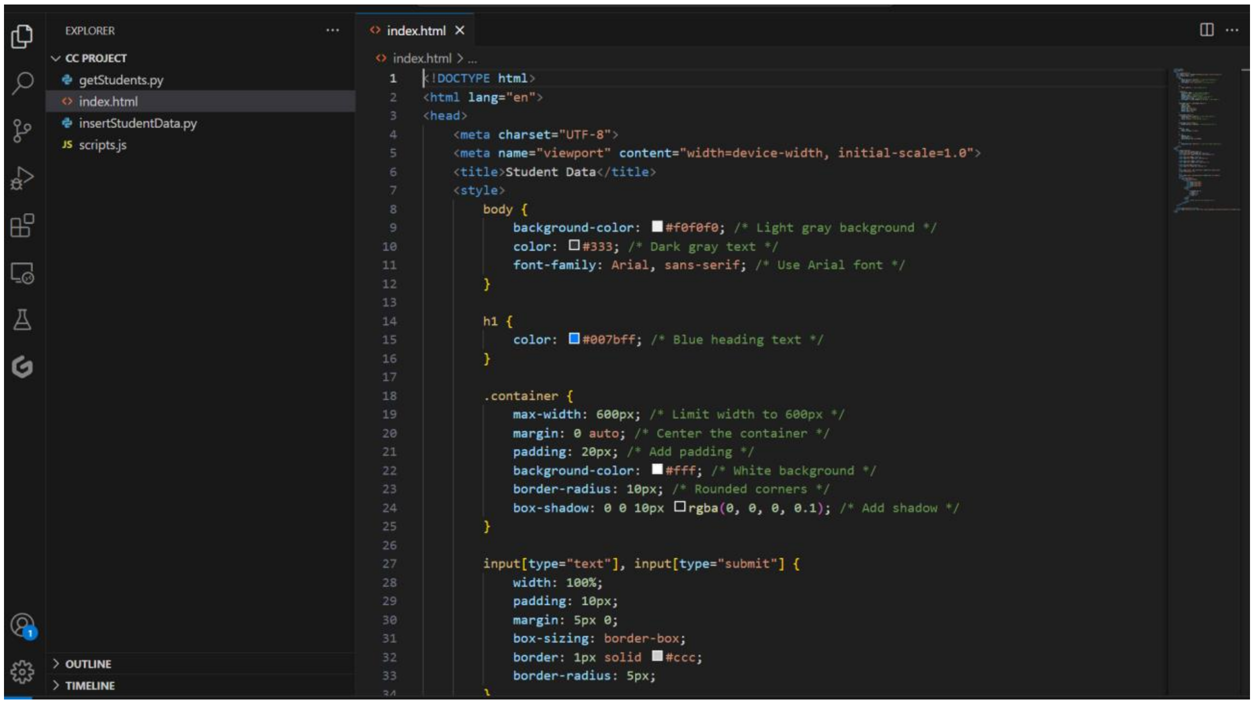
Feature: Streamline deployment and updates.

Implementation: Use AWS CodePipeline, CodeBuild, and CodeDeploy for continuous integration and continuous deployment (CI/CD), automating the release process and ensuring quick, reliable updates.

Technical Details

1. Frontend (Client-Side)

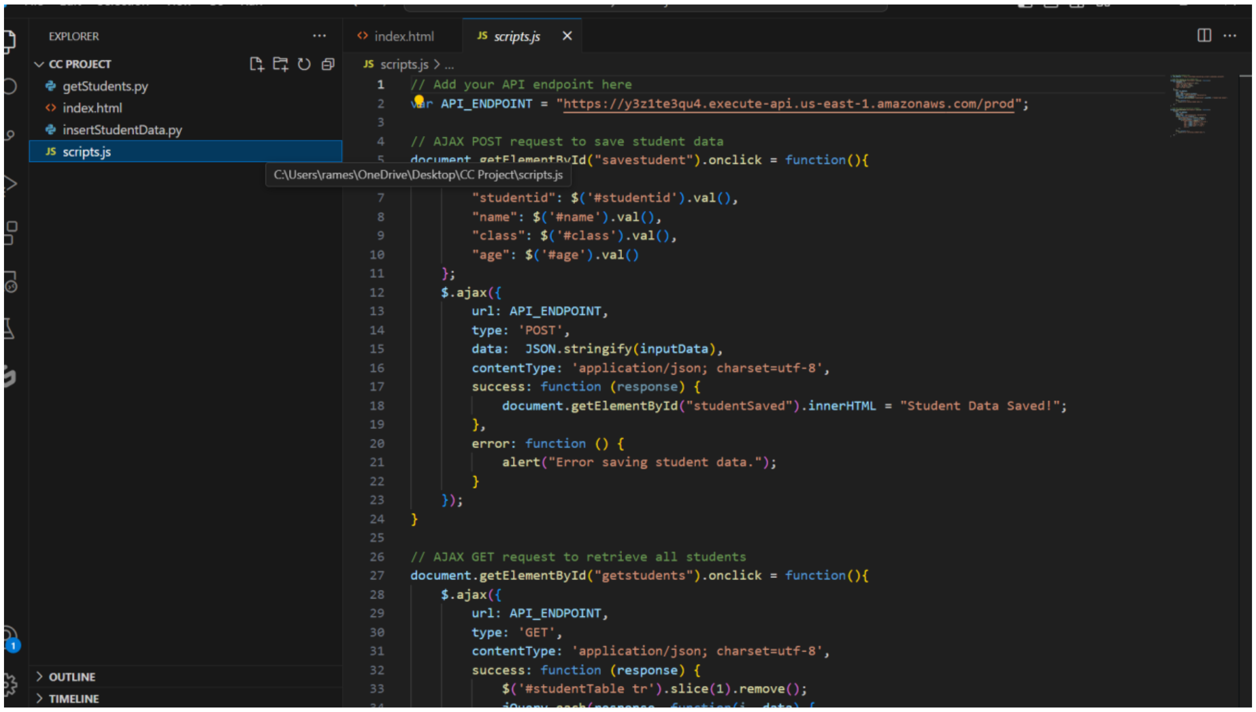
- **Technologies Used:** HTML, CSS, JavaScript, React/Vue/Angular (choose one or your preferred frontend framework).
- **Hosting:** Static files hosted on Amazon S3.
- **Deployment:** Automated using AWS CodePipeline and CodeDeploy or manual upload via the AWS Management Console.
- **Content Delivery:** Amazon CloudFront CDN for fast and secure content delivery to users globally.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Data</title>
    <style>
        body {
            background-color: #f0f0f0; /* Light gray background */
            color: #333; /* Dark gray text */
            font-family: Arial, sans-serif; /* Use Arial font */
        }

        h1 {
            color: #007bff; /* Blue heading text */
        }

        .container {
            max-width: 600px; /* Limit width to 600px */
            margin: 0 auto; /* Center the container */
            padding: 20px; /* Add padding */
            background-color: #fff; /* White background */
            border-radius: 10px; /* Rounded corners */
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Add shadow */
        }

        input[type="text"], input[type="submit"] {
            width: 100%;
            padding: 10px;
            margin: 5px 0;
            box-sizing: border-box;
            border: 1px solid #ccc;
            border-radius: 5px;
```

2. Amazon S3 (Simple Storage Service)

- **Purpose:** Storage and hosting of static assets like HTML, CSS, JavaScript, and media files.
- **Configuration:**
    - **Bucket Configuration:** Public access settings for static website hosting.
    - **Security:** Bucket policies and IAM roles to control access.
    - **Versioning:** Enabled for rollback and recovery.



```
// Add your API endpoint here
var API_ENDPOINT = "https://y3z1te3qu4.execute-api.us-east-1.amazonaws.com/prod";

// AJAX POST request to save student data
document.getElementById("savestudent").onclick = function(){

        "studentid": $('#studentid').val(),
        "name": $('#name').val(),
        "class": $('#class').val(),
        "age": $('#age').val()
    };
    $.ajax({
        url: API_ENDPOINT,
        type: 'POST',
        data:  JSON.stringify(inputData),
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            document.getElementById("studentSaved").innerHTML = "Student Data Saved!";
        },
        error: function () {
            alert("Error saving student data.");
        }
    });
}

// AJAX GET request to retrieve all students
document.getElementById("getstudents").onclick = function(){
    $.ajax({
        url: API_ENDPOINT,
        type: 'GET',
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            $('#studentTable tr').slice(1).remove();
```

3. Amazon CloudFront

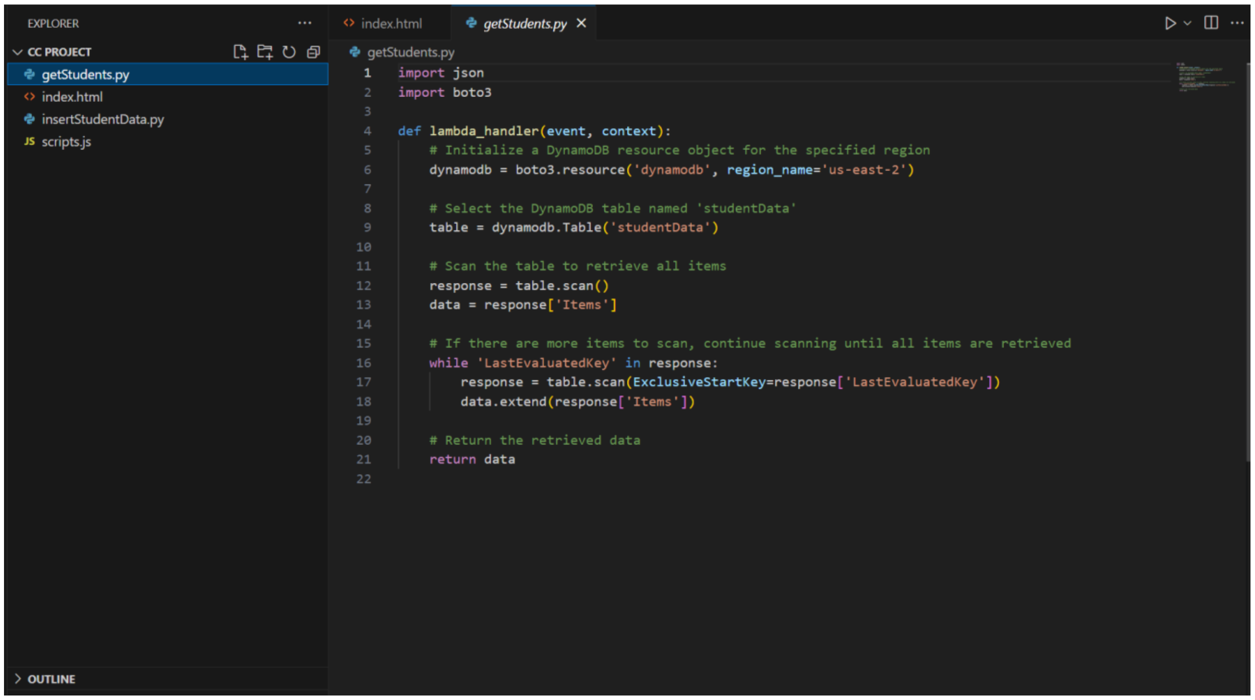
- **Purpose:** Global content delivery and caching for low-latency access.
- **Configuration:**
    - **Origin:** S3 bucket or API Gateway.
    - **Distribution Settings:** Edge locations for content caching.
    - **Security:** HTTPS/SSL for secure data transmission, WAF for protection against common web exploits.

4. Amazon API Gateway

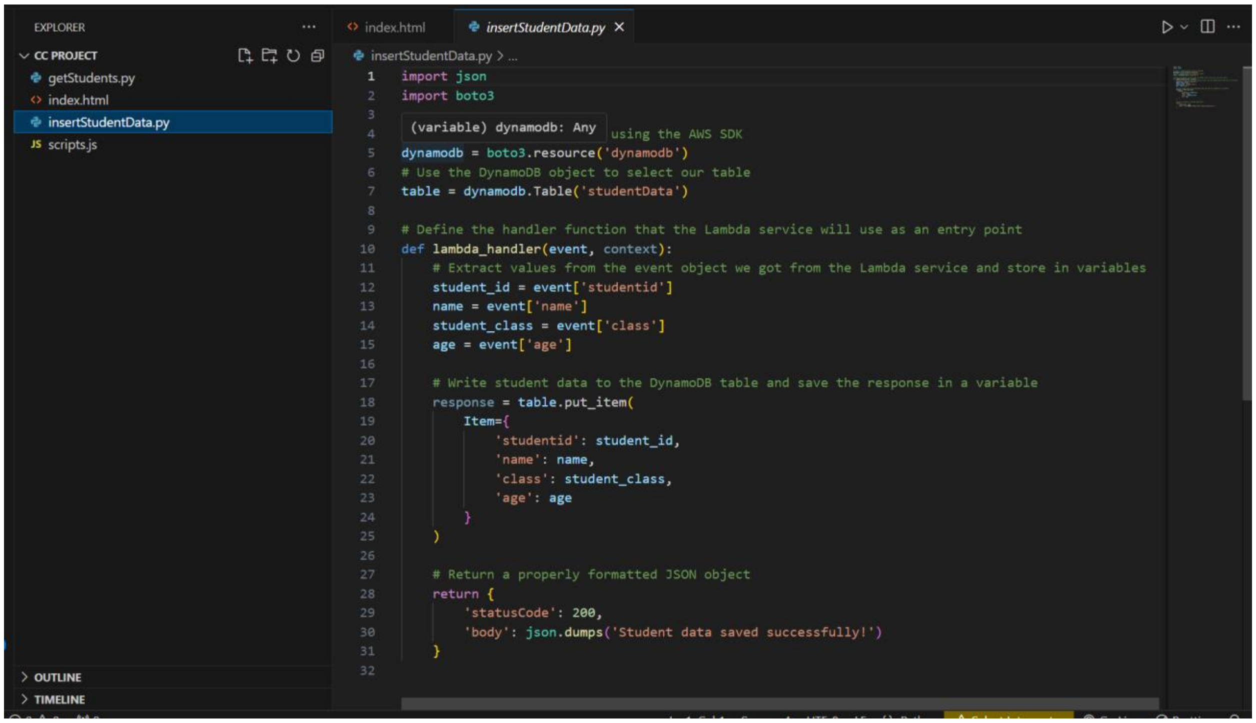
- **Purpose:** API management, request routing, and transformation.
- **Configuration:**
    - **API Type:** REST or HTTP APIs, depending on use case.
    - **Stages:** Separate stages for development, testing, and production.
    - **Integration:** Proxy integration with AWS Lambda.
    - **Security:** API keys, IAM roles, and Cognito user pools for authentication and authorization.

5. AWS Lambda

- **Purpose:** Serverless compute for backend logic.
- **Configuration:**
    - **Function Code:** Written in Python, Node.js, Java, or another supported language.
    - **Triggers:** API Gateway, S3 events, DynamoDB streams, etc.
    - **Execution Role:** IAM role with permissions to access required AWS services.
    - **Timeout and Memory Settings:** Configured based on function requirements to optimize performance and cost.



```python
import json
import boto3

def lambda_handler(event, context):
    # Initialize a DynamoDB resource object for the specified region
    dynamodb = boto3.resource('dynamodb', region_name='us-east-2')

    # Select the DynamoDB table named 'studentData'
    table = dynamodb.Table('studentData')

    # Scan the table to retrieve all items
    response = table.scan()
    data = response['Items']

    # If there are more items to scan, continue scanning until all items are retrieved
    while 'LastEvaluatedKey' in response:
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])

    # Return the retrieved data
    return data
```



```python
import json
import boto3

(variable) dynamodb: Any using the AWS SDK
dynamodb = boto3.resource('dynamodb')
# Use the DynamoDB object to select our table
table = dynamodb.Table('studentData')

# Define the handler function that the Lambda service will use as an entry point
def lambda_handler(event, context):
    # Extract values from the event object we got from the Lambda service and store in variables
    student_id = event['studentid']
    name = event['name']
    student_class = event['class']
    age = event['age']

    # Write student data to the DynamoDB table and save the response in a variable
    response = table.put_item(
        Item={
            'studentid': student_id,
            'name': name,
            'class': student_class,
            'age': age
        }
    )

    # Return a properly formatted JSON object
    return {
        'statusCode': 200,
        'body': json.dumps('Student data saved successfully!')
    }
```

6. Amazon DynamoDB

- **Purpose:** NoSQL database for storing application data.
- **Configuration:**

    - **Tables:** Structured with primary keys (partition key and sort key) for efficient querying.
    - **Indexes:** Global Secondary Indexes (GSI) and Local Secondary Indexes (LSI) for additional querying capabilities.
    - **Provisioning:** On-demand or provisioned capacity mode based on anticipated workload.
    - **Security:** IAM policies and encryption at rest.

7. Security and Access Control

- **AWS Identity and Access Management (IAM):**
    - **Purpose:** Secure access and permissions management.
    - **Configuration:** Roles and policies to control access to AWS services.
- **AWS Web Application Firewall (WAF):**
    - **Purpose:** Protect against common web exploits.
    - **Configuration:** Rules and conditions to filter traffic.
- **SSL/TLS Certificates:**
    - **Purpose:** Secure data transmission.
    - **Configuration:** Managed by AWS Certificate Manager (ACM) and applied to CloudFront distributions and API Gateway endpoints.

8. Monitoring and Logging

- **Amazon CloudWatch:**
    - **Purpose:** Monitoring, logging, and alerts.
    - **Configuration:** Metrics, logs, and alarms set for API Gateway, Lambda, and other services.
- **AWS CloudTrail:**
    - **Purpose:** Logging and auditing of API calls.
    - **Configuration:** Trail setup for capturing AWS account activity and API usage.

9. Continuous Integration and Continuous Deployment (CI/CD)

- **AWS CodePipeline, CodeBuild, CodeDeploy:**
    - **Purpose:** Automate the build, test, and deployment process.

    - **Configuration:** Pipelines set up for different environments (development, staging, production).
    - **Source Control Integration:** Integration with Git repositories like GitHub or AWS CodeCommit.

10. User Authentication and Authorization

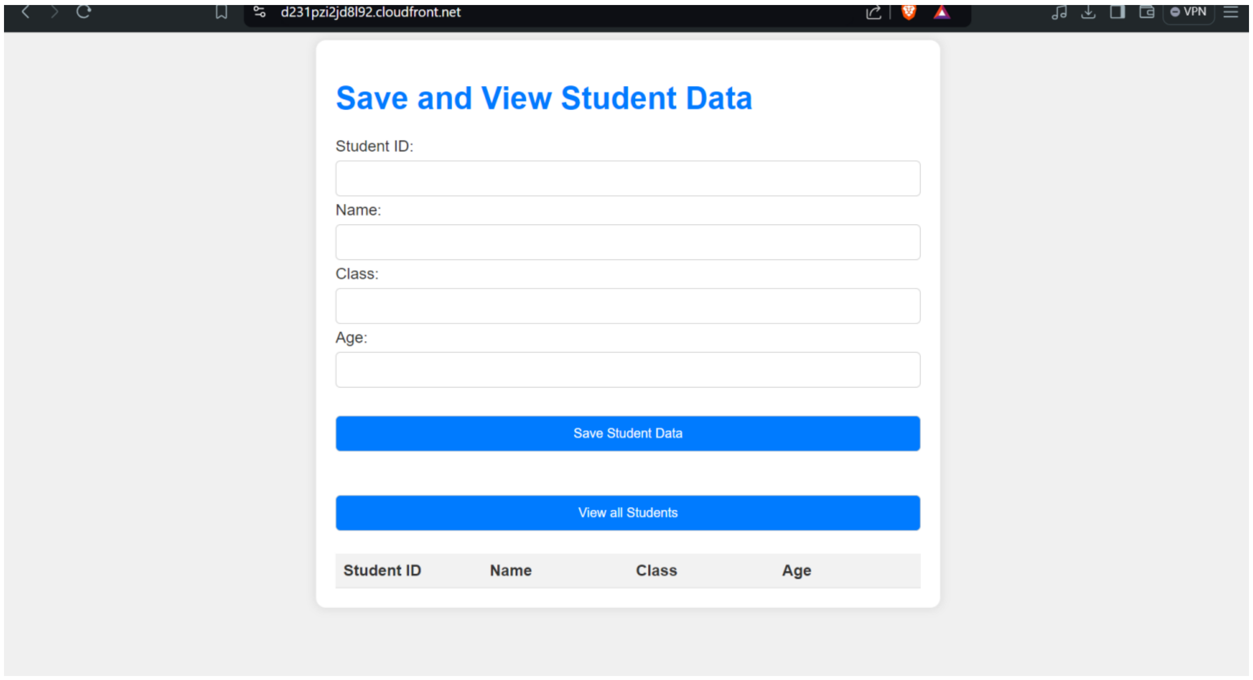
- **Amazon Cognito:**
    - **Purpose:** User sign-up, sign-in, and access control.
    - **Configuration:** User pools and identity pools for managing user authentication and authorization.
    - **Integration:** Integrated with API Gateway for secure access to backend services.

11. Additional Configurations

- **Backup and Recovery:**
    - **DynamoDB Backup:** Point-in-time recovery enabled for data resilience.
- **Cost Management:**
    - **Budgets and Alerts:** Set up AWS Budgets and Cost Explorer to monitor and manage costs effectively

Results:

- Web page


d231pzi2jd8l92.cloudfront.net

Save and View Student Data

Student ID:

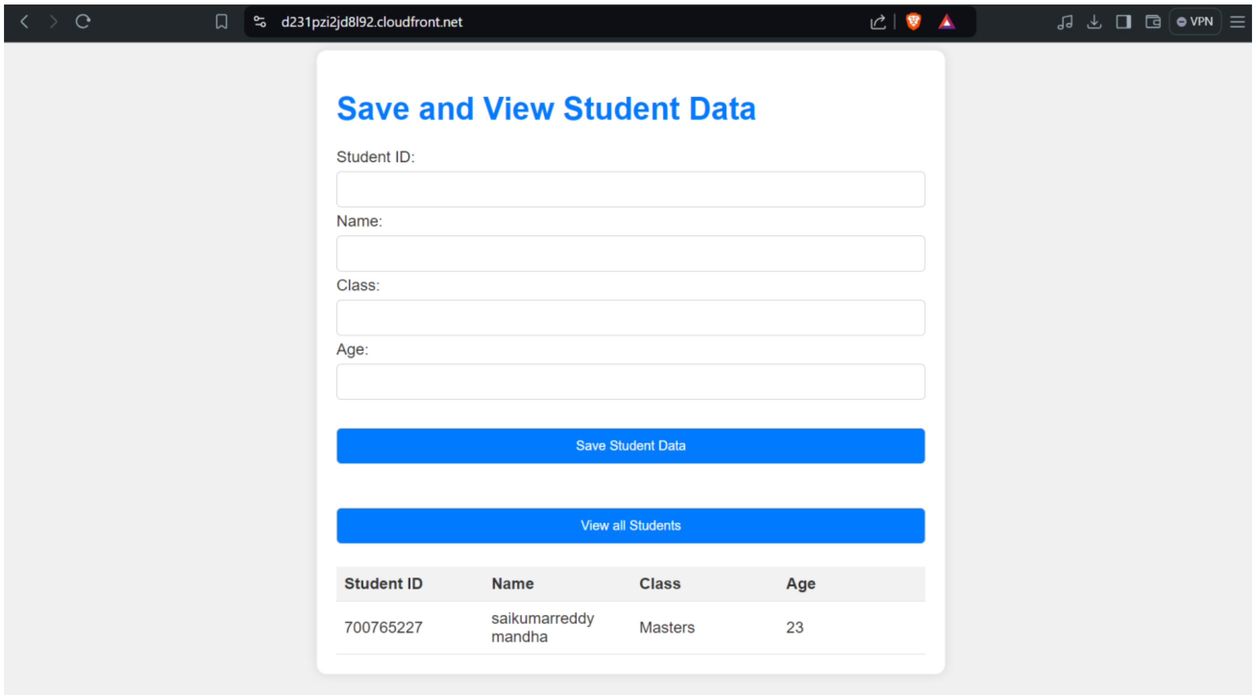
Name:

Class:

Age:

Student ID	Name	Class	Age

- After entering the student data


d231pzi2jd8l92.cloudfront.net

Save and View Student Data

Student ID:

Name:

Class:

Age:

Student ID	Name	Class	Age
700765227	saikumarreddy mandha	Masters	23

- Accessing the website at **https://d231pzi2jd8l92.cloudfront.net/**

END OF THE REPORT