Deploy a Serverless Web App using AWS Lambda and API Gateway

Rajdeep Sen   
*Techno International New Town*

*Kolkata, West Bengal*  
*Email: rjdiips@gmail.com*

*Abstract*—This report presents the design, development, and deployment of a serverless student record web application utilizing Amazon Web Services (AWS). The application integrates multiple AWS services, including AWS Lambda, API Gateway, DynamoDB, and S3, to provide a scalable, cost-effective, and highly available solution. Furthermore, Amazon CloudFront is incorporated to enhance security, optimize content delivery, and reduce latency. The report provides a detailed, step-by-step explanation of the implementation process, covering critical aspects such as database setup, API configuration, frontend hosting, security enhancements, and performance optimization. The adoption of a serverless architecture minimizes infrastructure management overhead, making this solution ideal for dynamic and growing workloads. The implementation follows best practices to ensure efficiency, security, and maintainability.

Keywords—AWS, Serverless, Lambda, API Gateway, DynamoDB, S3, CloudFront, Scalability, Security, Performance Optimization

# Introduction

Serverless computing has fundamentally transformed the way applications are deployed and managed by removing the complexities associated with traditional server provisioning and maintenance. Instead of allocating and overseeing infrastructure resources, developers can focus entirely on writing and optimizing application code while cloud providers handle the underlying operational aspects, including scaling, fault tolerance, and security.

This project focuses on designing and implementing a fully functional student record web application utilizing various AWS services to ensure high availability, enhanced security, and cost efficiency. The primary objective is to create a seamless and responsive platform that allows users to add, store, and retrieve student records efficiently. The frontend of the application is hosted on AWS S3, ensuring fast and scalable content delivery, while AWS Lambda serves as the backend, enabling serverless computing with dynamic execution. API Gateway acts as the communication bridge, handling HTTP requests and ensuring secure interactions between the frontend and backend.

By harnessing AWS’s suite of serverless technologies, this system achieves rapid scalability, cost reduction, and operational efficiency through a pay-per-use pricing model. The implementation of CloudFront significantly enhances security by mitigating Distributed Denial-of-Service (DDoS) attacks and optimizing content delivery. Additionally, AWS Web Application Firewall (WAF) is integrated to provide additional layers of security against malicious traffic.

This report provides a comprehensive overview of the project’s development lifecycle, covering critical areas such as system architecture, database design, API configuration, frontend deployment, security enforcement, and performance optimizations. Furthermore, it discusses best practices for designing serverless applications, ensuring sustainability, maintainability, and future scalability. The insights from this project highlight the advantages of adopting a serverless approach for modern cloud-native applications, demonstrating how AWS services collectively create a powerful, resilient, and efficient solution.

# System Architecture

The architecture of the student record web application is structured using a modular and scalable approach, ensuring optimal performance and security. It comprises four primary components:

### **Frontend:** A static website hosted on AWS S3, providing an intuitive and user-friendly interface for interacting with the student record database. The frontend consists of HTML, CSS, and JavaScript files designed for responsive access across multiple devices. The S3 bucket is configured with static website hosting capabilities, ensuring that users can easily access the application via a browser.

### **Backend:** The application logic is handled by AWS Lambda functions, which execute on-demand to process database transactions and API requests. These functions are responsible for handling GET and POST requests, allowing users to retrieve and insert student records into the DynamoDB database. The backend is designed with scalability in mind, allowing Lambda functions to automatically scale with demand while maintaining efficiency and performance.

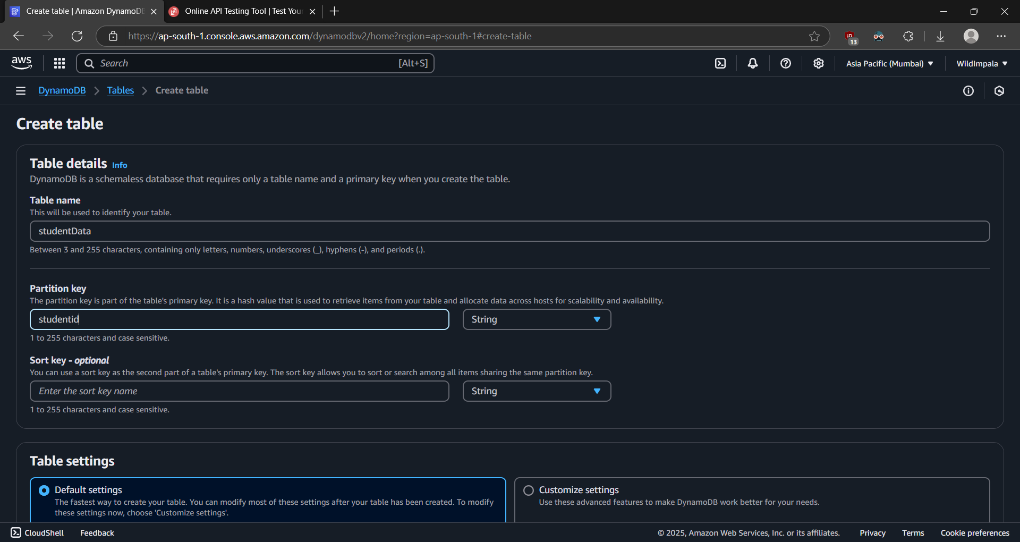
### **API Gateway:** Serving as the intermediary between the frontend and backend, API Gateway facilitates communication between users and the application logic. It is configured to process HTTP requests securely, applying request validation, throttling, and authentication mechanisms. Additionally, it translates incoming API calls into executable commands for the Lambda functions, ensuring seamless data flow.

### **CloudFront:** A content delivery network (CDN) that enhances the security and performance of the application. CloudFront reduces latency by caching frequently accessed content at edge locations worldwide. It also serves as a security layer, mitigating threats such as DDoS attacks by integrating with AWS Shield and AWS WAF. Additionally, CloudFront provides SSL/TLS encryption to ensure secure data transmission between users and the application.

The combination of these components enables the application to deliver high performance, cost efficiency, and security, making it a robust solution for managing student records in a serverless environment.

# Implementation Steps

## DynamoDB Table Creation



### Steps to Create DynamoDB Table:

### Open AWS Management Console

### Navigate to DynamoDB

### Click "Create Table"

### Configure table details:

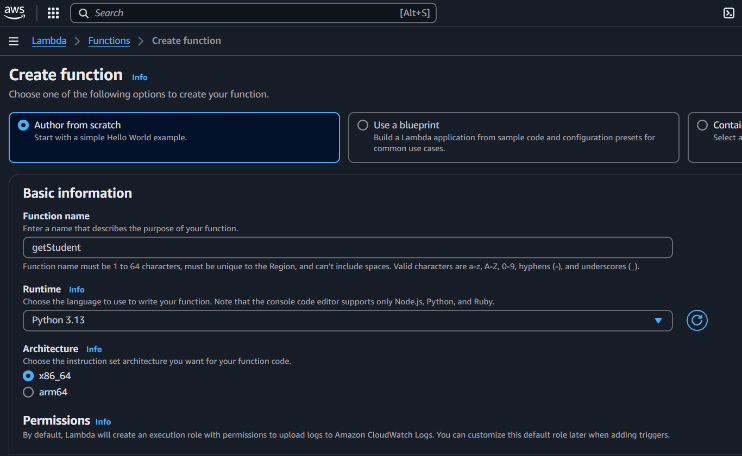
#### Table Name: student-data

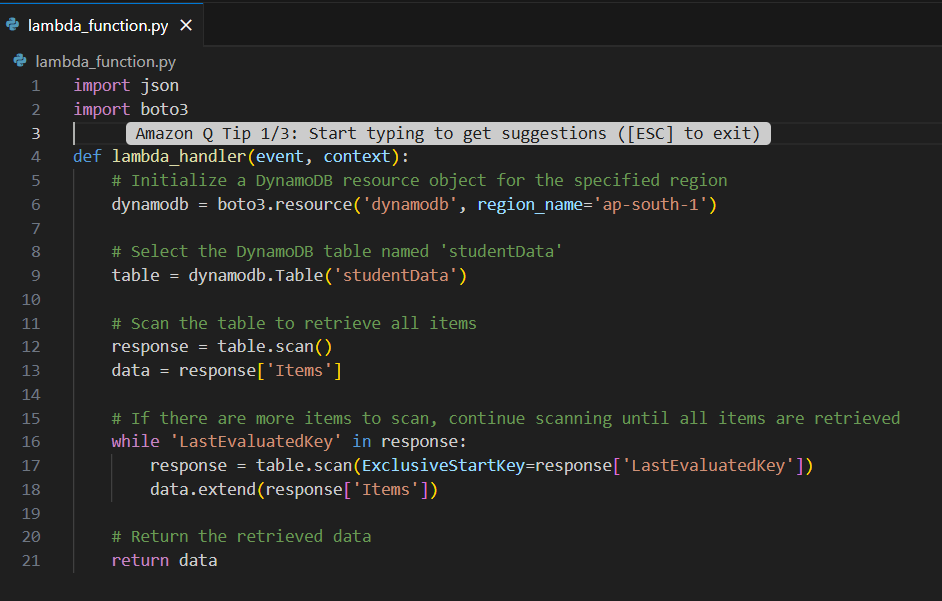
#### Partition Key: student\_id (String)

## Lambda Function Development

* + 1. *Get Student Data Lambda Function*

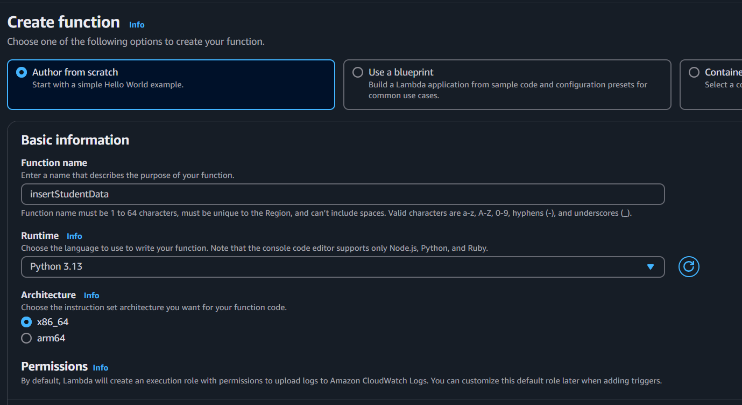
1. *Runtime: Python 3.12*
2. *Function Purpose: Retrieve student records from DynamoDB*
3. *IAM Role: LambdaDynamoDB with full DynamoDB access*

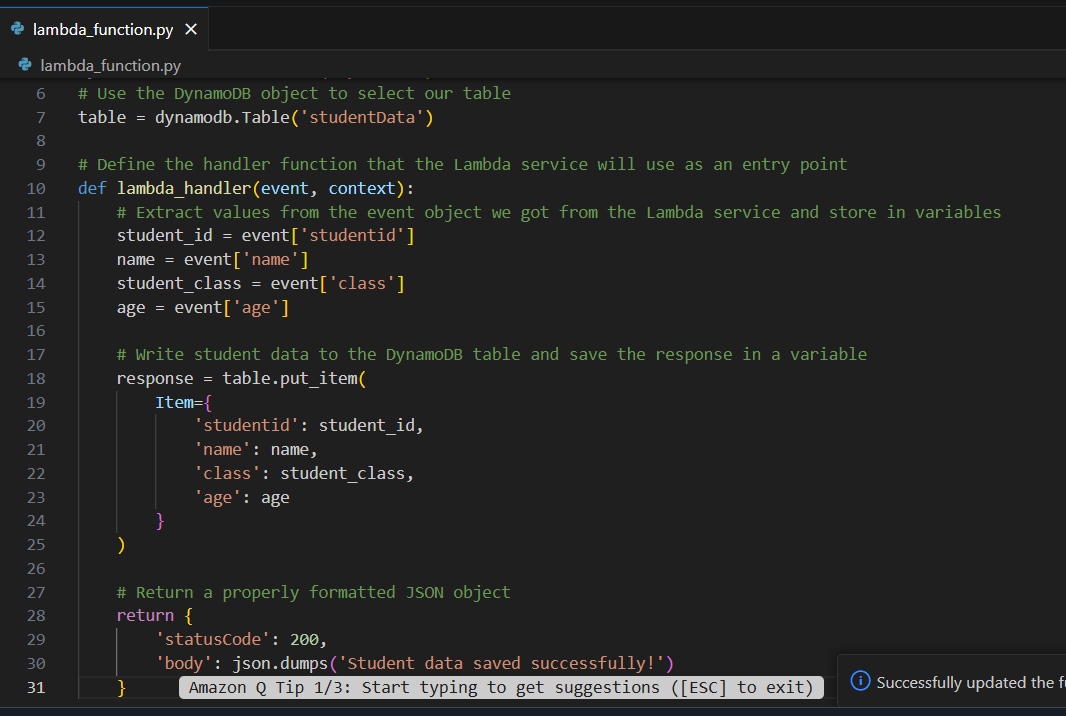
**

**

* + 1. *Insert Student Data Lambda Function*

1. *Runtime: Python 3.12*
2. *Function Purpose: Insert new student records into DynamoDB*
3. *IAM Role: LambdaDynamoDB with full DynamoDB access*

**



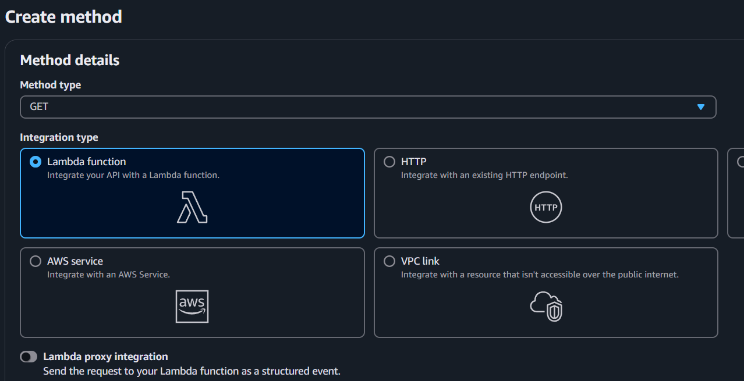
* 1. *API Gateway Configuration*

1. *Create REST API named "student"*

A screenshot of a computer

AI-generated content may be incorrect.

1. *Create GET and POST methods*



A screenshot of a computer

AI-generated content may be incorrect.

1. *Integrate with respective Lambda functions*
2. *Deploy API to production stage*

A screenshot of a software

AI-generated content may be incorrect.

1. *Enable CORS for GET and POST*
   1. *S3 Static Website Hosting*
      1. *Create bucket "student-record-5150"*

A screenshot of a computer

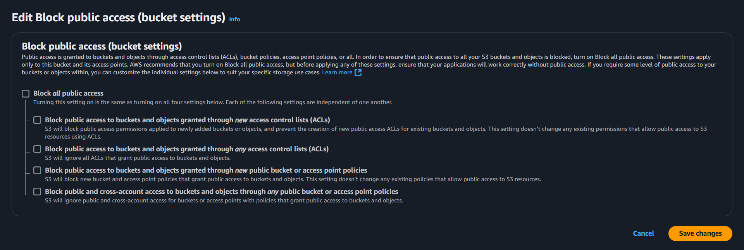
AI-generated content may be incorrect.

* + 1. *Upload index.html and scripts.js*

A screenshot of a computer

AI-generated content may be incorrect.

* + 1. *Disable Block all public access*

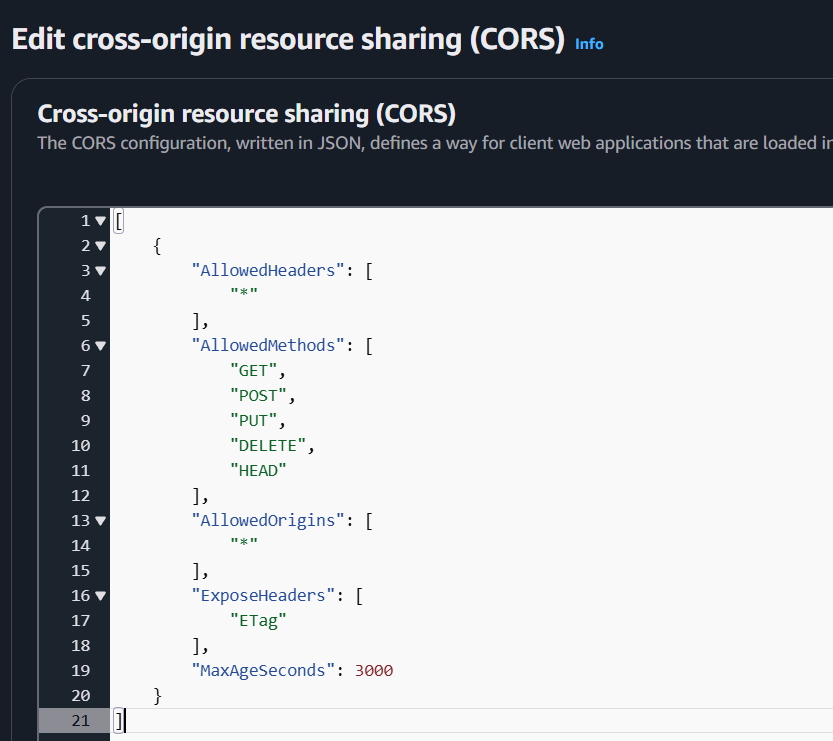


* + 1. *Enable static website hosting*
    2. *Configure public access policy*
    3. *Set default document to index.html*

A screenshot of a computer

AI-generated content may be incorrect.

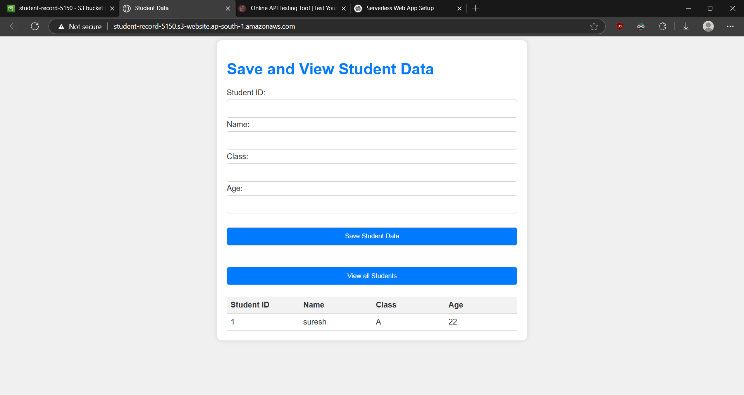
* + 1. *Edit CORS for the S3 bucket*



* 1. *CloudFront Distribution*

1. *Create CloudFront distribution*
2. *Set S3 bucket as origin*
3. *Configure default root object (index.html)*
4. *Generate secure HTTPS endpoint*

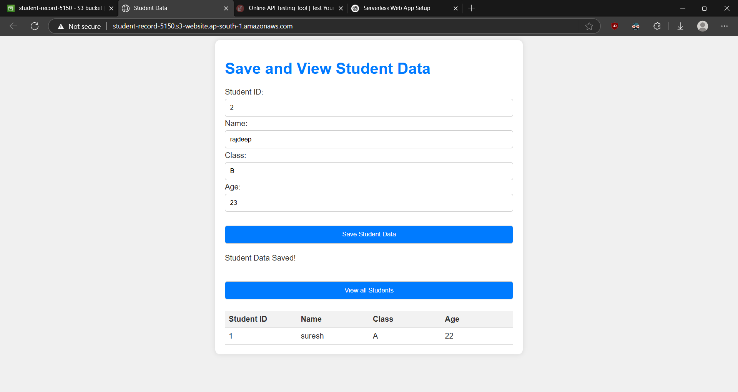
# Webapp Functionality and Testing



## User Interface Overview

The web application provides a simple, intuitive interface for managing student data with two primary functions:

1. Save Student Data



1. View All Student Data

A screenshot of a computer

AI-generated content may be incorrect.

# Results

Successfully hosted the web application on AWS S3, ensuring availability and accessibility.

1. Verified seamless interaction between the frontend and backend through API Gateway, enabling secure communication.
2. Integrated CloudFront to enhance security by restricting unauthorized access and improving content delivery speed.
3. Conducted functional testing to confirm that student records can be successfully added and retrieved through the web interface.
4. Observed and validated API requests and responses via browser developer tools to ensure proper data transmission.
5. Ensured that security configurations, such as CORS policies and HTTPS enforcement, were correctly implemented to prevent potential vulnerabilities

# Cost Efficiency Evaluation

Monthly Estimated Costs:

* Lambda Compute: $0.20 per 1M requests
* DynamoDB Operations: $1.25 per million read/write units
* API Gateway Calls: $3.50 per million API requests
* S3 Storage: $0.023 per GB
* CloudFront Data Transfer: $0.085 per GB

Total Estimated Monthly Cost: $5-10 for low to moderate traffic

# Future Work and Recommendations

## Proposed Enhancements

### Security Improvements

* Integrate Amazon Cognito for user management
* Implement fine-grained IAM role policies
* Add comprehensive input validation

### Functional Expansions

#### Advanced Search and Filtering Capabilities

* Implement complex query mechanisms
* Add pagination for large datasets
* Create advanced filtering options

#### Reporting and Analytics

* Generate student performance reports
* Implement data visualization dashboards
* Create exportable analytics

## Technical Improvements

#### Implement Comprehensive Error Handling

* Develop robust error logging
* Create custom error response mechanisms
* Implement retry strategies for transient failures

#### Performance Optimization

* Implement caching strategies
* Optimize Lambda cold start times
* Use DynamoDB DAX for read-heavy workloads

## Potential Extensions

* Mobile application development
* Integration with external student management systems
* Real-time notification systems
* Machine learning-based student performance prediction

# CONCLUSION

This project successfully demonstrates the design, development, and deployment of a serverless student record web application using AWS services such as Lambda, API Gateway, DynamoDB, S3, and CloudFront. The serverless architecture ensures scalability, cost-efficiency, and minimal maintenance, eliminating the need for traditional infrastructure management.

The integration of AWS Lambda and API Gateway provides a secure and efficient mechanism for handling student records, while DynamoDB ensures fast and scalable data storage. Hosting the frontend on Amazon S3, combined with CloudFront for security and performance, enhances reliability and accessibility. Key security measures, including CORS policies and IAM role-based access control, were implemented to prevent vulnerabilities.

The final system enables seamless record management and serves as a cost-effective, scalable solution for educational institutions. Future enhancements, such as authentication with AWS Cognito, improved logging with CloudWatch, and AI-driven analytics, could further extend its functionality. This project highlights the power of serverless computing, offering valuable insights into cloud-native application development and real-world AWS implementations.

##### Acknowledgments

##### I would like to express my sincere gratitude to Mainak Bannerjee for their invaluable guidance and support throughout this project. Their insights and feedback played a crucial role in shaping the implementation and enhancing the overall quality of the system.

##### I would also like to acknowledge Amazon Web Services (AWS) for providing comprehensive documentation and robust cloud services, which facilitated the seamless development of this serverless web application. Special thanks to online resources, tutorials, and technical communities that provided useful insights and troubleshooting support.

##### Lastly, I extend my appreciation to friends, colleagues, and peers who offered their encouragement and constructive feedback, making this project a rewarding learning experience.

##### References

1. Amazon Web Services, “AWS Lambda Documentation,” AWS, [Online]. Available: https://docs.aws.amazon.com/lambda/.
2. Amazon Web Services, “Amazon API Gateway Documentation,” AWS, [Online]. Available: https://docs.aws.amazon.com/apigateway/.
3. Amazon Web Services, “Amazon DynamoDB Documentation,” AWS, [Online]. Available: https://docs.aws.amazon.com/dynamodb/.
4. Amazon Web Services, “Amazon Simple Storage Service (S3) Documentation,” AWS, [Online]. Available:https://docs.aws.amazon.com/s3/.
5. Amazon Web Services, “Amazon CloudFront Documentation,” AWS, [Online]. Available: https://docs.aws.amazon.com/cloudfront/.
6. Amazon Web Services, “AWS Identity and Access Management (IAM) Documentation,” AWS, [Online]. Available: https://docs.aws.amazon.com/iam/.

[7] Amazon Web Services, “AWS Security Best Practices,” AWS Whitepaper, [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-security-best-practices/.