IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

Cloud-Native Student Information Management: A Serverless NoSQL Architecture with Infrastructure-as-Code and Continuous Deployment on AWS

Vasamsetti Bala sridevi[1], K.Lakshmi Sai Sri*[2]

PG Scholar Department of Computer Science, S.V.K.P & Dr. K.S. Raju Arts and Science College(Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[1]

*Lecturer, Department of Master of Computer Applications, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[2]

Abstract: Higher education institutions increasingly face challenges in managing fragmented student data across disparate systems, resulting in operational inefficiencies and delayed access to academic information. This paper presents the design, implementation, and evaluation of a cloud-native Student Information System (SIS) built on a fully serverless NoSQL backend using Amazon Web Services (AWS) DynamoDB, provisioned through AWS CloudFormation Infrastructure-as-Code (IaC) templates, and delivered via an automated continuous integration and continuous deployment (CI/CD) pipeline. The system employs Python and Flask as the application framework, integrates Flask-SocketIO for real-time WebSocket-based data broadcasting, and enforces security through bcrypt-based password hashing and session management via Flask-Login. A five-table DynamoDB schema operating in PAY_PER_REQUEST billing mode eliminates capacity planning overhead while ensuring high availability. The infrastructure stack provisions a Virtual Private Cloud (VPC), EC2 compute instance, IAM roles, security groups, and an Elastic IP through a single declarative CloudFormation template. An AWS CodePipeline orchestrates CodeBuild and CodeDeploy stages triggered automatically by CodeCommit repository events via Amazon EventBridge. Experimental evaluation demonstrates sub-100 ms response times for core academic operations and WebSocket event delivery latencies below 30 ms. The proposed architecture reduces manual deployment effort and infrastructure drift, providing an open, replicable template for cloud-native academic information systems in resource-constrained institutional contexts.

Keywords: Student Information System; AWS DynamoDB; CloudFormation; Infrastructure-as-Code; CI/CD; Flask; WebSocket; SocketIO; Cloud-Native; Serverless NoSQL.

I. INTRODUCTION

A. Background

The global expansion of higher education enrolment has amplified the operational complexity of academic record management. Student Information Systems (SIS) serve as the operational backbone of educational institutions, consolidating diverse academic data-student demographics, course registrations, attendance records, and examination resultsinto a unified, accessible repository [1]. Traditional SIS architectures have historically relied on monolithic application servers paired with relational database management systems (RDBMS). While functionally adequate in homogeneous, on-premises environments, such designs exhibit well-documented limitations including rigid schema structures, horizontal scaling constraints, and the operational burden of dedicated infrastructure teams managing upgrades, backups, and failover mechanisms [2].

The proliferation of cloud computing paradigms has fundamentally altered the feasibility landscape for educational software. Managed cloud servicesparticularly serverless databases, declarative infrastructure provisioning tools, and automated deployment pipelinesnow enable small development teams to operate production-grade systems without maintaining physical hardware [3]. Amazon Web Services (AWS), as the leading cloud provider, offers a

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

comprehensive suite of services that, when composed correctly, can deliver a complete application stack from database to compute to deployment automation using only configuration-driven, version-controlled templates [4].

B. Problem Statement

Despite the availability of cloud services, a significant proportion of institutional SIS deployments remain anchored to legacy on-premises architectures. These systems suffer from three principal deficiencies. First, the absence of real-time data synchronisation forces administrators and faculty to rely on periodically refreshed views, introducing latency between system events and user awareness [5]. Second, infrastructure provisioning continues to depend on manual processes, introducing human error, configuration drift, and extended recovery times during incidents [6]. Third, software update cyclesfrom code commit to production deploymentare often manual and multi-step, creating a disjoint between development velocity and operational stability [7].

These deficiencies collectively result in reduced operational efficiency, elevated administrative costs, and constrained institutional responsiveness. There is therefore a clear need for architecturally sound, reproducible, and cloud-native SIS implementations that address each of these gaps in an integrated manner.

C. Research Objectives

This research pursues four primary objectives:

- To design and implement a full-featured SIS covering student management, course administration, attendance tracking, and result processing using Python and Flask on AWS.
- To integrate AWS DynamoDB as a scalable, managed NoSQL data tier provisioned through declarative CloudFormation templates, eliminating manual schema and capacity management.
- To incorporate WebSocket-based real-time event propagation via Flask-SocketIO, enabling all connected clients to receive immediate reflections of system state changes.
- To establish a fully automated CI/CD pipeline using AWS CodeCommit, CodeBuild, and CodeDeploy orchestrated by CodePipeline, triggered by EventBridge repository events.

D. Contributions

The principal contributions of this work are:

- A complete, open-architecture SIS implementation integrating five coordinated DynamoDB tables with real-time WebSocket notification, requiring no relational database infrastructure.
- A reusable, parameterised AWS CloudFormation template provisioning VPC networking, EC2 compute, IAM roles, DynamoDB tables, and Elastic IP in a single atomic operation.
- A companion pipeline CloudFormation stack encoding a three-stage CodePipeline (Source-Build-Deploy) with EventBridge-triggered automatic execution and rollback-on-failure policy.
- An empirical performance characterisation of the system across core operations, demonstrating sub-100 ms API latency and sub-30 ms WebSocket broadcast latency.

II. LITERATURE REVIEW

A. Related Work

Student information systems have been the subject of sustained research attention across multiple technology generations. Early systems employed client-server architectures with centralised RDBMS backends. Llorente et al. [3] described a Java EE-based SIS using a MySQL relational schema, which, while functionally comprehensive, lacked scalable deployment mechanisms or real-time capabilities. Similarly, Okonkwo et al. [5] developed a PHP-based system incorporating PostgreSQL and demonstrated functional completeness for African higher education contexts but documented significant manual administrative effort in system maintenance.

The transition toward cloud-hosted architectures is documented by several studies. Rawat and Barthwal [7] explored a Django-based SIS hosted on shared cloud infrastructure but retained SQLite as the persistence layer, limiting horizontal scalability. Prasetyo et al. [9] proposed a Laravel framework system with partial real-time elements using polling mechanisms rather than persistent WebSocket connections, resulting in increased server load under concurrent use.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

Serverless and NoSQL approaches have gained traction in adjacent application domains. Mahmoud et al. [11] implemented an educational management platform using Node.js and MongoDB containerised with Docker, demonstrating real-time capability but employing custom deployment scripts rather than managed cloud IaC tooling. Research directly targeting Infrastructure-as-Code for educational systems remains limited; prior studies by Chen et al. [13] and Raj et al. [15] addressed IaC in enterprise contexts but did not apply these methodologies specifically to academic information management systems.

AWS DynamoDB as an application persistence layer has been studied in e-commerce and IoT contexts [17], but its application to academic information managementparticularly with concurrent real-time synchronisation and automated deploymenthas not been previously addressed in the literature.

B. Research Gap Analysis

A synthesis of existing literature reveals three principal research gaps. First, no prior SIS study integrates serverless NoSQL cloud persistence with real-time WebSocket event propagation and Infrastructure-as-Code provisioning within a unified architecture. Second, fully automated CI/CD pipelines implemented using managed cloud services (rather than self-hosted tools such as Jenkins) have not been documented for academic SIS deployments. Third, the operational and performance characteristics of DynamoDB-backed academic management systems under realistic concurrent load conditions have not been empirically characterised.

C. Comparative Study

Table I presents a structured comparison of the proposed system against representative prior implementations across key technical dimensions.

TABLE I

Comparison of SIS Implementations in Related Literature

Reference	Platform	Database	Real-time	IaC Deployment	CI/CD Pipeline
Llorente et al. [3]	Java EE	MySQL	No	No	No
Okonkwo et al. [5]	PHP	PostgreSQL	No	No	Partial
Rawat & Barthwal [7]	Django	SQLite	No	No	No
Prasetyo et al. [9]	Laravel	MySQL	Partial	No	No
Mahmoud et al. [11]	Node.js	MongoDB	Yes	Docker	Partial
	Flask/Python	AWS DynamoDB	Yes (WebSocket)	CloudFormation	Full AWS Pipeline

III. PROPOSED METHODOLOGY

A. System Architecture

The proposed system adopts a three-tier cloud-native architecture comprising a presentation tier, an application tier, and a managed data tier. The presentation tier consists of server-rendered Jinja2 HTML templates enhanced with Bootstrap 5.3 for responsive layout, Chart.js 4 for interactive dashboard visualisations, DataTables for searchable and sortable record grids, and the Socket.IO client library for WebSocket connectivity. The application tier is implemented as a Python 3.11 Flask application structured around the application factory pattern, with Flask-Login managing session-based authentication and Flask-SocketIO providing the WebSocket server operating over an Eventlet asynchronous worker. The data tier consists of five AWS DynamoDB tables provisioned in PAY_PER_REQUEST billing mode, accessed through the boto3 AWS SDK.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

All compute and networking resources are defined declaratively in an AWS CloudFormation template and provisioned as a named stack. The deployment workflow is encoded in a companion CloudFormation pipeline template that creates the full CI/CD pipeline as managed AWS resources. Figure 1 illustrates the overall three-tier system architecture.

B. Workflow

The operational workflow of the system follows a request-event-broadcast cycle. Upon receiving an HTTPS request, the Flask routing layer authenticates the session via Flask-Login, queries or mutates the appropriate DynamoDB table through the boto3 resource interface, renders a Jinja2 template with the updated data, and simultaneously emits a named Socket.IO event to all connected WebSocket clients. Clients receiving the event update the relevant UI components—dashboard counters, activity feeds, toast notificationswithout requiring a page reload.

The deployment workflow follows a source-build-deploy cycle. A developer commits code to the AWS CodeCommit repository on the main branch. Amazon EventBridge detects the referenceUpdated event and triggers the CodePipeline execution. The Source stage packages the repository contents as an S3 artifact. The Build stage invokes the CodeBuild project, which executes the buildspec.yml specification: it installs Python 3.11, installs dependencies from requirements.txt, validates imports, removes compiled artefacts, and packages the clean application. The Deploy stage delivers the build artifact to the EC2 instance via CodeDeploy, executing the AppSpec lifecycle hooks (stop service, install dependencies, start service, validate health). Automatic rollback is configured to activate on deployment failure or CloudWatch alarm breach. Figure 2 depicts the CI/CD pipeline flow.

C. Database Schema Design

The DynamoDB schema comprises five independent tables, each using a UUID-based string partition key to ensure globally unique item identification and uniform key distribution. DynamoDB's schema-less nature permits each record to carry arbitrary attributes without prior declaration, enabling the addition of optional fields without migration overhead. Table II details the five tables and their core attributes. The students table enables Point-in-Time Recovery (PITR), protecting enrolled student records against accidental deletion or corruption.

TABLE II

DynamoDB Schema Overview

Table Name	Partition Key	Billing Mode	PITR	Key Attributes
users	user_id (S)	PAY_PER_REQUEST	—	username, password_hash, role, full_name
students	student_id (S)	PAY_PER_REQUEST	Enabled	student_code, full_name, program, enrolled_courses
courses	course_id (S)	PAY_PER_REQUEST	—	course_name, credits, schedule, faculty
attendance	attendance_id (S)	PAY_PER_REQUEST	—	student_id, course_id, date, status (Present/Absent/Late)
results	result_id (S)	PAY_PER_REQUEST	—	student_id, course_id, marks, max_marks, grade, exam_type

Inter-table relationships are resolved at the application layer rather than via foreign key constraints, consistent with DynamoDB's document-oriented model. When an attendance or result record is created, the application queries student and course names from their respective tables and denormalises these values directly into the record item. This design trades normalisation for read performance, eliminating the need for JOIN operations and permitting each record to be retrieved with a single GetItem call.

D. Automated Grade Computation

The result management module incorporates an automated grade computation function that maps numeric marks to letter grades according to the following piecewise threshold function:

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

$$G(m) = \{A+ \text{ if } m \geq 90,\ A \text{ if } 80 \leq m < 90,\ B+ \text{ if } 70 \leq m < 80,$$

$$B \text{ if } 60 \leq m < 70,\ C \text{ if } 50 \leq m < 60,\ D \text{ if } 40 \leq m < 50,\ F \text{ if } m < 40\}$$

where G denotes the assigned grade and m denotes the student's numerical score. This function is implemented in the calculate_grade() utility in results.py and is invoked both at record creation and during any subsequent update, ensuring grade consistency without client-side computation. Table III presents the full grade mapping applied by the system.

TABLE III

Automated Grade Mapping Function

Marks Range (%)	Grade	Classification
≥ 90	A+	Distinction
80 – 89	A	Excellent
70 – 79	B+	Very Good
60 – 69	B	Good
50 – 59	C	Average
40 – 49	D	Pass
< 40	F	Fail

E. Real-Time Event Architecture

Real-time synchronisation is implemented through Flask-SocketIO operating in Eventlet asynchronous mode. Upon each Create, Update, or Delete operation in any route, the server emits a named event carrying a minimal payload identifying the affected entity. The full set of broadcast events includes: student_added, student_updated, student_deleted, attendance_marked, attendance_updated, attendance_deleted, result_added, result_updated, result_deleted, and server_message. Connected browser clients subscribe to these events through the Socket.IO JavaScript client and update the interface locally, eliminating polling overhead and reducing perceived latency. Figure 3 illustrates the WebSocket event lifecycle.

F. Infrastructure-as-Code Design

The infrastructure is defined across two CloudFormation YAML templates. The primary cloudformation.yaml template provisions: a VPC with a /16 CIDR block (10.0.0.0/16), a public subnet (/24), an internet gateway, route table, and subnet association; an application security group permitting inbound TCP on ports 80, 443, 5000, and 22; an IAM instance role granting fine-grained DynamoDB table access (GetItem, PutItem, UpdateItem, DeleteItem, Scan, Query, CreateTable, DescribeTable, ListTables) scoped to the table prefix ARN pattern; five DynamoDB tables with PAY_PER_REQUEST billing; an EC2 instance (t3.micro) running Amazon Linux 2023 with a UserData bootstrap script that installs Python, clones the application, writes the environment file, and registers a systemd service; and an Elastic IP address associated with the instance.

The companion pipeline.yaml template provisions: a versioned S3 artifact bucket with 30-day lifecycle expiry; an AWS CodeCommit repository; CodeBuild, CodeDeploy, CodePipeline, and EventBridge IAM roles with least-privilege policies; a CodeBuild project targeting Python 3.11 with CloudWatch Logs integration; a CodeDeploy application and deployment group configured for in-place EC2 deployment with automatic rollback; a three-stage CodePipeline; and an EventBridge rule monitoring CodeCommitreferenceCreated and referenceUpdated events to trigger the pipeline automatically. Figure 4 illustrates the IaC resource dependency graph.

IV. EXPERIMENTAL SETUP

A. Tools and Technologies

The system was developed and evaluated using the technology stack summarised in Table IV. Local development employed DynamoDB Local (version 2.5, Apache-licenced) as an offline emulator, allowing full application testing

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

without AWS credentials. The production environment was provisioned exclusively through the CloudFormation templates described in Section III.

TABLE IV

Technology Stack Summary

Category	Technology	Version	Purpose
Backend Framework	Flask	3.0.3	Web application core
Real-time Engine	Flask-SocketIO	5.3.6	WebSocket events
Authentication	Flask-Login + bcrypt	0.6.3 / 4.1.3	Session & password security
AWS SDK	boto3	1.34.69	DynamoDB operations
Database	AWS DynamoDB	Managed	NoSQL data persistence
Async Server	Eventlet	0.36.1	Async WebSocket support
Frontend	Bootstrap 5.3 + Chart.js	5.3 / 4.x	UI components & charts
IaC	AWS CloudFormation	—	Infrastructure provisioning
CI/CD	CodeBuild + CodeDeploy	—	Continuous deployment
Compute	AWS EC2 (t3. micro)	Amazon Linux 2023	Application hosting

B. Environment Configuration

Production deployment targets an AWS EC2 t3. micro instance (2 vCPUs, 1 GiB RAM) running Amazon Linux 2023 in the us-east-1 region. The application is managed as a systemd service (studentcore.service) restarting automatically on failure. Environment variables are written by the CloudFormation UserData bootstrap and include the AWS region, DynamoDB table prefix, and a 32-byte cryptographically random Flask secret key. No static AWS credentials are embedded; the EC2 IAM instance role satisfies all DynamoDB authentication requirements through the instance metadata service.

C. Dataset Description

Performance evaluation employed synthetically generated datasets representative of a medium-scale academic institution. The evaluation dataset comprised 250 student records across 20 academic programmes, 30 course records, 1,200 attendance records spanning a 12-week semester (4 sessions per week per student sample), and 600 result records across two examination periods (midterm and final). This scale is consistent with departmental-level deployments at small to medium higher education institutions. All synthetic records were generated using UUID-based identifiers to replicate production key distributions.

D. Evaluation Metrics

The system was evaluated across four metric categories: (1) API response latency—the round-trip time from HTTP request submission to response receipt, measured as mean and 95th percentile values; (2) WebSocket event broadcast latency—the elapsed time from server-side socketio.emit() invocation to client-side event handler execution; (3) pipeline cycle time—the elapsed time from git commit to successful service restart on EC2; and (4) CloudFormation stack provisioning time—the elapsed time from stack creation initiation to CREATE_COMPLETE status. Latency measurements were collected using the Locust load testing framework with concurrent user counts of 1, 10, and 50, with reported values representing the 50-user concurrency scenario to reflect operational load conditions.

V. RESULTS AND DISCUSSION

A. Performance Analysis

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

Table V presents the measured performance characteristics of core system operations under 50-concurrent-user load. All CRUD operations exhibited mean latencies well below 100 ms, with the exception of the dashboard statistics endpoint (/api/stats), which performs five parallel DynamoDB table scans and returned a mean latency of 128 ms. Login operations exhibit higher latency (310 ms mean) due to the intentional computational cost of bcrypt password verification (work factor 12), which is a security requirement rather than an architectural deficiency.

TABLE V

System Performance Metrics Under 50-Concurrent-User Load

Operation	Avg Latency (ms)	95th Pct (ms)	Throughput (req/s)	Error Rate (%)
Student Record Fetch	42	67	95	0.0
Attendance Submission	55	88	78	0.0
Result Entry + Grade Calc	48	72	85	0.0
Dashboard API (/api/stats)	128	198	42	0.0
WebSocket Event Broadcast	18	28	220	0.0
User Login (bcrypt auth)	310	370	22	0.0
CloudFormation Stack Deploy	—	—	1 stack / 8 min	0.0
CI/CD Pipeline Cycle	—	—	1 deploy / 4 min	0.0

B. WebSocket Performance

WebSocket event broadcast exhibited a mean latency of 18 ms and a 95th percentile of 28 ms across all monitored event types. This performance is attributable to the Eventlet asynchronous worker model, which eliminates per-request thread blocking and allows the SocketIO server to process and emit events concurrently with ongoing HTTP request handling. The broadcast architecture ensures that regardless of the triggering user's geographic proximity to the server, all connected clients receive state updates within sub-30 ms windows, satisfying the operational requirement for real-time academic system synchronisation.

C. CI/CD Pipeline Efficiency

The full CI/CD cycle—from git push to production service restartcompleted in a mean of 4 minutes and 12 seconds across 10 test deployments. This duration is decomposed as: CodeCommit upload (12 s), CodeBuild install and validation phase (2 min 40 s), CodeDeploy deployment and validation (1 min 20 s). The buildspec.yml cache configuration for the pip dependency directory reduced the install phase by approximately 45 seconds on cached builds. Automatic rollback was verified by intentionally introducing a dependency error; the pipeline correctly detected the ApplicationStop hook failure and rolled back to the previous revision without manual intervention.

D. Infrastructure Provisioning

The primary CloudFormation stack (cloudformation.yaml) completed provisioning in a mean of 7 minutes 48 seconds across three test deployments, with DynamoDB table creation completing in under 15 seconds and EC2 instance initialisation (including UserData script execution) accounting for the majority of provisioning time. Stack deletion completed in under 3 minutes, cleanly removing all provisioned resources. The pipeline stack provisioned in under 4 minutes.

E. Comparative Evaluation

Comparing against the prior systems identified in Table I, the proposed system offers distinct advantages across all

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135116

evaluated dimensions. Unlike the Java EE and PHP systems reviewed in the literature, which required manual server provisioning and lacked real-time capability, the proposed system eliminates both deficiencies through CloudFormation and SocketIO respectively. The Node.js/MongoDB system of Mahmoud et al. [11] achieved real-time capability but relied on Docker Compose rather than managed cloud services, requiring operators to manage container infrastructure. The proposed system eliminates this operational layer by targeting managed DynamoDB and EC2, reducing the operational surface to application code alone.

F. Limitations

The current implementation presents several limitations that constrain its applicability at scale. First, DynamoDB Scan operations are used extensively for list queries, which introduce O(N) read cost and increasing latency as table sizes grow. Production-scale deployments should introduce Global Secondary Indexes (GSIs) on frequently queried attributes such as student program, course enrollment, and attendance date. Second, the dashboard statistics endpoint executes five sequential table scans, creating a bottleneck under high concurrent access. Introducing ElastiCache for Memcached or DynamoDB Accelerator (DAX) as a caching layer would substantially reduce this latency. Third, the single-EC2 deployment topology lacks redundancy; integration of an Application Load Balancer with an Auto Scaling Group would be required for production availability guarantees.

VI. CONCLUSION AND FUTURE WORK

This paper has presented a comprehensive cloud-native Student Information System demonstrating the practical integration of serverless NoSQL persistence, Infrastructure-as-Code provisioning, real-time WebSocket event broadcasting, and automated CI/CD pipeline execution within a single deployable architecture. The system addresses the three principal deficiencies identified in legacy SIS implementations—absence of real-time synchronisation, manual infrastructure management, and fragmented deployment workflows—through coordinated use of AWS DynamoDB, CloudFormation, Flask-SocketIO, and CodePipeline.

Experimental evaluation confirmed sub-100 ms API response latency for primary academic operations, sub-30 ms WebSocket broadcast latency, a complete CI/CD cycle time of approximately four minutes, and full infrastructure provisioning in under eight minutes from a single declarative template deployment. The zero-error rate across all evaluated operations demonstrates the operational reliability of the DynamoDB-backed architecture under representative concurrent load conditions.

Future work will address the identified limitations through four planned enhancements. First, DynamoDB Global Secondary Indexes will be introduced on student_program, course_id (in attendance and results tables), and attendance_date attributes to replace full-table Scan operations with efficient Query operations. Second, a DynamoDB Accelerator (DAX) cluster will be evaluated as an in-memory caching layer for the dashboard statistics API. Third, the deployment topology will be extended to a multi-AZ architecture using an Application Load Balancer and Auto Scaling Group to deliver production-grade availability and elastic horizontal scaling. Fourth, the authentication layer will be enhanced with AWS Cognito integration to support federated identity, multi-factor authentication, and institutional SSO, replacing the current bcrypt-hash-in-DynamoDB credential model.

REFERENCES

[1] R. Almunawar and M. Anshari, "Student information system and its adoption in higher education institutions," in Proc. Int. Conf. Educ. Technol. Sci., 2022, pp. 45-52.

[2] T. Connolly and C. Begg, Database Systems: A Practical Approach to Design, Implementation, and Management, 6th ed. Harlow, UK: Pearson, 2015.

[3] J. Llorente, A. Rodriguez, and M. Fernandez, "A Java EE-based student information management system for universities," J. Syst. Softw., vol. 85, no. 3, pp. 612-624, Mar. 2012.

[4] Amazon Web Services, AWS Well-Architected Framework, AWS Whitepaper, 2023. [Online]. Available: https://aws.amazon.com/architecture/well-architected/

[5] C. Okonkwo, B. Eke, and U. Eze, "Design and implementation of an online student academic record management system," Int. J. Comput. Appl., vol. 174, no. 12, pp. 18-26, 2021.

DOI: 10.17148/IARJSET.2026.135116

[6] M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Boston, MA: Addison-Wesley, 2013.

[7] S. Rawat and A. Barthwal, "A cloud-based student management system using Django framework," in Proc. IEEE Int. Conf. Comput. Power Commun. Technol. (GUCON), 2021, pp. 1-6.

[8] W. Kim, "Cloud computing: Today and tomorrow," J. Object Technol., vol. 8, no. 1, pp. 65-72, Jan. 2009.

[9] D. Prasetyo, A. Suharto, and W. Suharso, "Real-time student information system using Laravel and WebSocket," in Proc. Int. Conf. Inform. Commun. Technol. (ICoICT), 2022, pp. 314-319.

[10] A. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, 3rd ed. Upper Saddle River, NJ: Pearson, 2017.

[11] A. Mahmoud, H. Hassan, and R. Abdel-Fattah, "Node.js and MongoDB-based academic management platform with real-time notifications," in Proc. Int. Conf. Comput. Inf. Sci. (ICCIS), 2023, pp. 1-7.

[12] Amazon Web Services, Amazon DynamoDB Developer Guide, 2024. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/

[13] Y. Chen, J. Lin, and H. Zheng, "Infrastructure as code for enterprise cloud environments: A comparative study," IEEE Access, vol. 10, pp. 23441-23455, 2022.

[14] Amazon Web Services, AWS CloudFormation User Guide, 2024. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/

[15] S. Raj, A. Periwal, and R. Singh, "Continuous delivery practices in cloud-native applications: A systematic review," J. Cloud Comput., vol. 11, no. 1, pp. 1-22, 2022.

[16] A. Kleppmann, Designing Data-Intensive Applications. Sebastopol, CA: O'Reilly Media, 2017.

[17] N. Al-Fayoumi, M. Hussain, and A. Al-Saleh, "Performance evaluation of AWS DynamoDB for IoT event streaming applications," IEEE Internet Things J., vol. 10, no. 5, pp. 4112-4122, 2023.

[18] P. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," Linux J., vol. 2014, no. 239, p. 2, 2014.

[19] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.

[20] C. Richardson, Microservices Patterns: With Examples in Java. Shelter Island, NY: Manning, 2018.

AUTHORS' BIOGRAPHIES

Vasamsetti Bala Sridevi received the B.Sc. degree in Computer Science from Sri Samhitha Degree college Ravulapalem, in 2022, She is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts and Science College(Autonomous), Penugonda, West Godavari, India. Her research interests include cloud computing, web application development, and educational technology systems.

K.Lakshmi Sai Sri Working as Lecturer in S.V.K.P& Dr.K.S.Raju Arts and Science College (Autonomous), Penugonda, West Godavari District, AP. Master's Degree in Computer Applications from Adikavi Nannaya University. Her areas of interest Applications of Artificial intelligence, Mobile application development, PHP, MySQL, Object Oriented Programming languages.