IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

ACAEMIAX: A Cloud-Based Microservices Platform for Enterprise Student Enquiry, Admission, and Course Management Using React, ASP.NET Core, PostgreSQL, and Docker

Asmita Deshpande [1], Sushil Kulkarni[2], Suryakant Kendre[3]

[1]M.Tech 4th semester Student Department of Computer Science And Information Technology, MBES College of Engineering Ambajogai, India

[2]Professor and Head , Department of Computer Science And Information Technology, MBES College of Engineering Ambajogai, India

[3]Assistant Professor and Guide , Department of Computer Science And Information Technology, MBES College of Engineering Ambajogai, India

Abstract

Educational institutions increasingly require scalable and intelligent digital platforms to manage student enquiries, admissions, fee processing, and academic workflows. Traditional monolithic systems often suffer from limitations in scalability, maintainability, and deployment flexibility. This paper presents **AcaemiaX**, a cloud-based enterprise student management platform developed using React, ASP.NET Core, PostgreSQL, and Docker. The system adopts a microservices-inspired architecture to separate functionalities such as authentication, enquiry management, student lifecycle management, admissions, payments, and analytics. The platform supports multi-branch educational institutions with role-based access control and secure JWT authentication.

The proposed system consists of a React-based frontend, ASP.NET Core REST APIs, and a PostgreSQL relational database containing seventeen interconnected entities. Docker containerization provides portability and simplified cloud deployment. Experimental evaluation demonstrates efficient response time, improved scalability, enhanced security, and ease of maintenance compared to conventional monolithic systems.

Keywords: Student Management System, Microservices, Cloud Computing, React, ASP.NET Core, PostgreSQL, Docker, JWT, Enterprise Application.

1. INTRODUCTION

The rapid growth of educational institutions and digital transformation initiatives has created a significant demand for enterprise-level academic management systems. Traditional paper-based and standalone applications are no longer sufficient for handling student enquiries, admissions, course management, and fee collection efficiently.

Modern educational institutions require software platforms that provide:

1. Student enquiry tracking
2. Admission processing
3. Fee management
4. Course administration
5. Multi-branch support
6. Analytics and reporting
7. Secure access control

Conventional monolithic systems face several challenges:

- Limited scalability
- Complex maintenance
- Difficult deployment
- Single points of failure
- Technology lock-in

Cloud-native architectures and microservices have emerged as effective solutions to these challenges. Microservices divide applications into smaller, independent services that can be developed, deployed, and scaled individually.

This research introduces **AcaemiaX**, a cloud-based enterprise student enquiry and admission management system built using modern technologies including:

- React 19
- ASP.NET Core 10
- PostgreSQL 16
- Docker
- JWT Authentication
- Entity Framework Core

The system supports a complete student lifecycle:

Enquiry → Student → Admission → Fee Installments → Payment → Reporting

The platform currently supports:

- Multi-branch institutions
- Role-based access control
- More than 50 REST APIs
- 17 relational database tables
- Docker-based deployment
- Enterprise-grade security

1.1 Motivation

Educational organizations face challenges such as:

- Managing thousands of enquiries
- Tracking student admissions
- Monitoring fee payments
- Maintaining branch-wise data
- Generating analytics reports

Manual operations lead to:

- Data inconsistency
- Human errors
- Delayed reporting
- Inefficient communication

A cloud-based solution addresses these issues by offering centralized management and automated workflows.

1.2 Problem Statement

Existing student management systems often suffer from:

1. Lack of scalability
2. Poor user experience
3. Weak security mechanisms
4. Limited analytics support
5. Difficult maintenance
6. Inadequate cloud deployment strategies

There is a need for a modern platform capable of supporting enterprise educational institutions while ensuring scalability, maintainability, and security.

1.3 Research Objectives

The primary objectives of this research are:

1. Design a scalable student management platform.
2. Implement secure authentication using JWT.
3. Support multi-branch educational institutions.
4. Automate enquiry-to-admission workflows.
5. Provide efficient fee management.
6. Enable cloud deployment using Docker.
7. Improve maintainability through modular architecture.

1.4 Research Contributions

The major contributions of this research include:

- Development of a cloud-native academic platform.
- Integration of React and ASP.NET Core.
- Design of a normalized PostgreSQL database.
- Implementation of RBAC security.
- Docker-based deployment architecture.
- Enterprise workflow automation.

---

2. LITERATURE SURVEY

Student information systems have evolved significantly over the last two decades. Early systems primarily focused on record keeping and basic administration. Modern systems emphasize scalability, cloud deployment, and intelligent analytics.

2.1 Traditional Student Information Systems

Traditional student management applications were primarily desktop-based or centralized web applications. Although these systems supported data storage and retrieval, they lacked flexibility and scalability.

Limitations included:

- Single server dependency
- Difficult maintenance
- Limited integration capabilities
- Poor user experience

2.2 Cloud Computing in Education

Cloud computing has revolutionized educational technology by enabling:

- On-demand resources
- High availability
- Cost efficiency
- Elastic scaling

Cloud deployment significantly improves accessibility and operational efficiency.

2.3 Microservices Architecture

Microservices divide applications into smaller services that communicate using APIs.

Advantages include:

1. Independent deployment
2. Fault isolation
3. Better scalability
4. Technology flexibility

5. Easier maintenance

Many enterprise systems have migrated from monolithic to microservices-based architectures due to these benefits.

2.4 React-Based Frontend Applications

React has become one of the most popular frontend frameworks because of:

- Component-based architecture
- Virtual DOM
- Efficient rendering
- State management

Educational systems benefit from React through responsive user interfaces and improved user experience.

2.5 ASP.NET Core for Enterprise Systems

ASP.NET Core provides:

- Cross-platform support
- High performance
- Dependency injection
- Middleware architecture
- REST API development

Its integration with Entity Framework Core simplifies database operations.

2.6 PostgreSQL in Enterprise Applications

PostgreSQL is widely adopted because of:

- ACID compliance
- Advanced indexing
- Strong relational capabilities
- High reliability
- Security features

Educational systems require robust databases to maintain data integrity and consistency.

2.7 Containerization Using Docker

Docker enables:

- Environment consistency
- Rapid deployment
- Resource optimization
- Portability

Containerized applications simplify deployment across development, testing, and production environments.

2.8 Comparative Analysis of Existing Systems

System	Architecture	Scalability	Security	Cloud Support
Traditional ERP	Monolithic	Low	Medium	Limited
LMS Platforms	Centralized	Medium	Medium	Partial
SIS Systems	Client-Server	Medium	Medium	Limited
AcaemiaX	Cloud Native	High	High	Full

2.9 Research Gap

Existing systems often lack:

- Integrated enquiry management
- Complete fee lifecycle management
- Multi-branch support
- Cloud-native deployment
- Enterprise analytics

AcaemiaX addresses these limitations through modern architecture and cloud technologies.

3. EXISTING SYSTEM ANALYSIS

Educational institutions traditionally use manual processes or monolithic software systems for managing student enquiries, admissions, courses, and fee collection. Although these systems provide basic functionality, they suffer from several operational and technical limitations.

3.1 Manual Management Systems

Traditional institutions often maintain records using:

1. Paper-based forms
2. Excel spreadsheets
3. Standalone desktop applications
4. Department-specific databases

Limitations

- Data redundancy and inconsistency
- High probability of human error
- Difficult report generation
- Lack of real-time analytics
- No centralized data repository
- Limited accessibility

3.2 Monolithic Student Management Systems

Monolithic applications combine all functionalities into a single deployable unit. Typical modules include:

- Authentication
- Student Management
- Course Management
- Fee Management
- Reporting

Drawbacks of Monolithic Architecture

1. **Poor Scalability**
   - Entire application must scale together.
   - Resource utilization becomes inefficient.
2. **Complex Maintenance**
   - Changes in one module affect others.
   - Long deployment cycles.
3. **Limited Technology Flexibility**
   - Difficult to integrate new technologies.
4. **Single Point of Failure**
   - Failure in one module may affect the entire system.
5. **Deployment Challenges**
   - Entire application must be redeployed for small changes.

3.3 Comparative Study of Existing Systems

Feature	Traditional System	Monolithic System	AcaemiaX
Cloud Deployment	No	Partial	Yes
Scalability	Low	Medium	High
Security	Low	Medium	High
Multi-Branch Support	No	Limited	Yes
Docker Support	No	No	Yes
Analytics	Limited	Medium	Advanced
API Integration	No	Partial	Full
Role-Based Access	No	Limited	Yes

4. PROBLEM STATEMENT

Educational institutions face challenges in managing large volumes of enquiries, admissions, and fee records while ensuring security, scalability, and maintainability.

The major problems include:

1. Fragmented data management.
2. Lack of centralized control.
3. Inefficient admission workflows.
4. Manual fee tracking.
5. Limited reporting capabilities.
6. Inadequate security measures.
7. Difficulty in scaling systems for multiple branches.

Therefore, there is a need for a cloud-native enterprise platform that supports complete student lifecycle management.

---

5. PROPOSED SYSTEM

To overcome the limitations of existing systems, this research proposes **AcaemiaX**, a cloud-based student enquiry and course management platform.

The proposed system integrates:

- React-based frontend
- ASP.NET Core REST APIs
- PostgreSQL database
- Docker containerization
- JWT authentication
- Role-based access control

The platform automates the complete academic workflow:

Enquiry → Student → Admission → Fees → Payment → Reports

5.1 Objectives of Proposed System

The proposed system aims to:

1. Automate enquiry management.
2. Simplify student admissions.
3. Manage course assignments.
4. Track fee payments.
5. Support multiple branches.
6. Provide analytics dashboards.
7. Ensure secure authentication.
8. Enable cloud deployment.

5.2 Key Features of AcaemiaX

Student Enquiry Management

- Create enquiries
- Track follow-ups
- Add notes
- Convert to students

Admission Management

- Course assignment
- Batch allocation
- Trainer assignment
- Admission tracking

Fee Management

- Installment generation
- Payment recording
- Refund processing
- Reminder management

Analytics

- Revenue reports
- Enrollment statistics
- Branch performance
- Lead source analysis

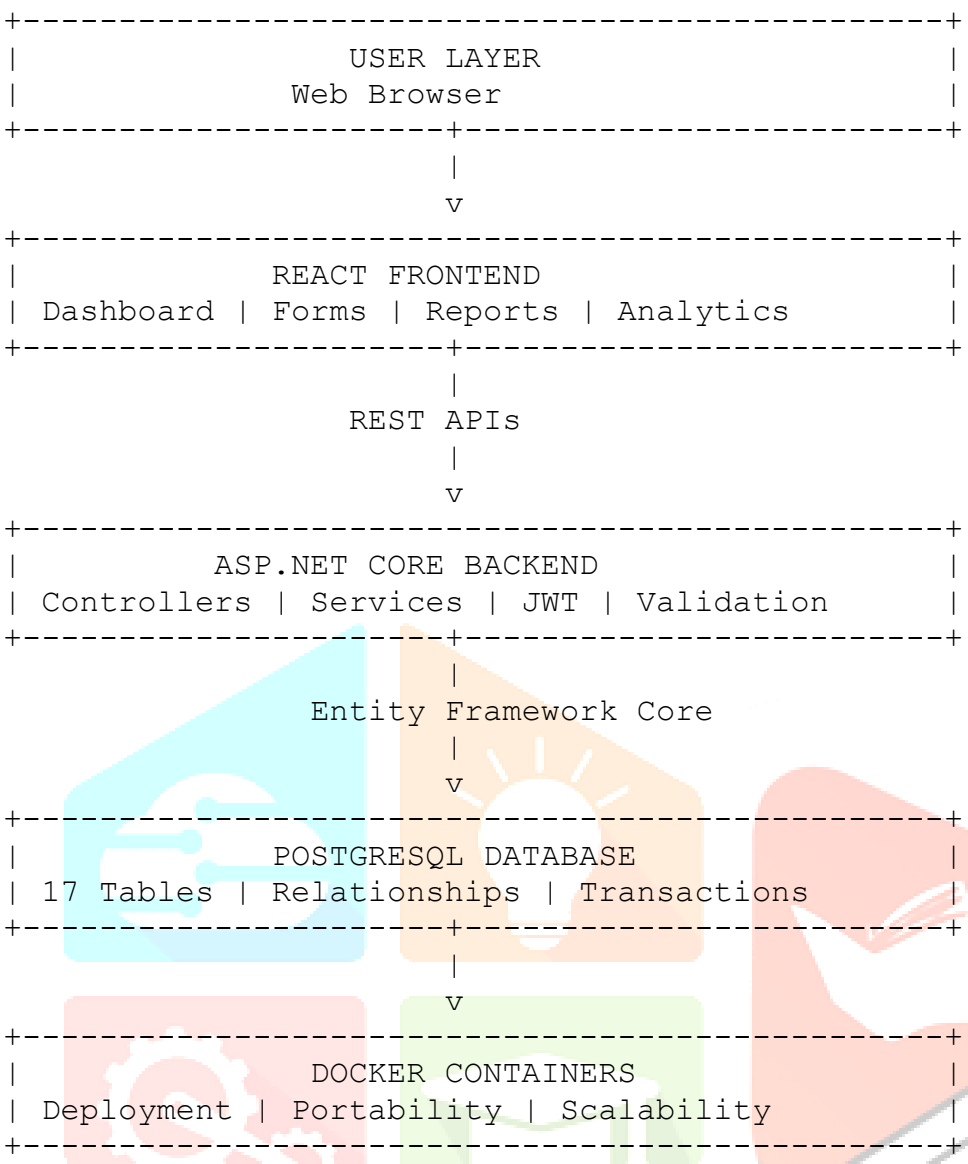
---

6. SYSTEM ARCHITECTURE

AcaemiaX follows a multi-layer cloud-native architecture consisting of:

1. Presentation Layer
2. API Layer
3. Business Layer
4. Persistence Layer
5. Deployment Layer

6.1 High-Level Architecture



6.2 Technology Stack

Layer	Technology
Frontend	React 19
Backend	ASP.NET Core 10
ORM	Entity Framework Core
Database	PostgreSQL 16
Authentication	JWT Bearer
Password Security	BCrypt
Deployment	Docker
Build Tool	Vite 8

7. MICROSERVICES ARCHITECTURE

Although implemented using ASP.NET Core APIs, AcaemiaX follows a microservices-inspired architecture by logically separating modules into independent services.

7.1 Authentication Service

Responsibilities:

- User login
- JWT generation
- Authorization
- Session management

7.2 Enquiry Service

Functions:

- Create enquiry
- Update status
- Follow-ups
- Notes management

7.3 Student Service

Functions:

- Student registration
- Profile management
- Academic information
- Guardian details

7.4 Admission Service

Functions:

- Admission creation
- Course assignment
- Batch allocation
- Trainer mapping

7.5 Fee Management Service

Functions:

- Installment creation
- Payment tracking
- Refund processing
- Overdue management

7.6 Reporting Service

Provides:

- Revenue reports
- Enrollment analytics
- Branch statistics
- Dashboard metrics

---

8. WORKFLOW OF ACAEMIAX

The system workflow is illustrated below:

```
NEW ENQUIRY
      |
      v
FOLLOW-UP PROCESS
      |
      v
CONVERT TO STUDENT
      |
      v
CREATE ADMISSION
      |
      v
GENERATE FEES
      |
      v
RECORD PAYMENT
      |
      v
GENERATE REPORTS
      |
      v
ENROLLMENT COMPLETED
```

8.1 Enquiry to Admission Workflow

Step 1: Capture Enquiry

Fields:

- Name
- Phone
- Email
- Source
- Branch

Step 2: Follow-Up

Actions:

- Add notes
- Update status
- Schedule reminders

Step 3: Convert to Student

System automatically:

- Generates Student ID
- Transfers enquiry data
- Creates student profile

Step 4: Create Admission

User selects:

- Course
- Batch
- Trainer

Step 5: Generate Fees

The system:

- Calculates total fees
- Creates installments
- Sets due dates

Step 6: Record Payments

The system updates:

- Payment status
- Revenue ledger
- Student records

9. DATABASE DESIGN

Database design is a critical component of enterprise applications. AcaemiaX uses **PostgreSQL 16**, an open-source relational database management system known for reliability, scalability, and ACID compliance.

The database consists of **17 interconnected tables** that maintain data integrity using primary keys, foreign keys, and normalization principles.

9.1 Database Tables

The major tables include:

1. Users
2. Branches
3. Courses
4. Trainers
5. Enquiries
6. Students
7. Admissions
8. Batches
9. FeeInstallments
10. FeePayments
11. Refunds
12. FollowUps
13. EnquiryNotes
14. ReminderRules
15. BranchLedger
16. BranchSettlement
17. FeeFollowUps

9.2 Entity Relationship Model

```
User → Branch
Enquiry → Notes → FollowUps
Enquiry → Student
Student → Admission
Admission → Course
Admission → Batch
Admission → Trainer
Admission → FeeInstallments
FeeInstallments → FeePayments
FeePayments → Refunds
```

9.3 Normalization

The database follows:

- First Normal Form (1NF)
- Second Normal Form (2NF)

- Third Normal Form (3NF)

Benefits:

- Reduced redundancy
- Better consistency
- Improved maintainability
- Efficient querying

---

10. API DESIGN

AcaemiaX follows REST architectural principles.

REST APIs facilitate communication between:

- React Frontend
- ASP.NET Backend
- PostgreSQL Database

10.1 API Endpoints

Authentication APIs

Endpoint	Method
/api/auth/login	POST
/api/auth/logout	POST

Student APIs

Endpoint	Method
/api/students	GET
/api/students	POST
/api/students/{id}	PUT
/api/students/{id}	DELETE

Enquiry APIs

- Create enquiry
- Update enquiry
- Add notes
- Follow-up tracking

Admission APIs

- Create admission
- Assign batch
- Assign trainer

- Generate fees

Total APIs: **50+**

10.2 API Lifecycle

```
React Request
      ↓
JWT Validation
      ↓
Controller
      ↓
Service Layer
      ↓
Entity Framework Core
      ↓
PostgreSQL
      ↓
JSON Response
```

---

11. SECURITY FRAMEWORK

Security is one of the major design goals of AcaemiaX.

The system implements:

- JWT Authentication
- BCrypt Hashing
- Role-Based Access Control
- Input Validation
- SQL Injection Prevention
- XSS Protection

11.1 JWT Authentication

Authentication flow:

1. User enters credentials
2. Backend validates user
3. JWT token generated
4. Token stored in browser
5. Token attached to API calls
6. Backend validates token

Token validity: **24 hours**

11.2 Password Security

Passwords are protected using:

BCrypt hashing algorithm

Benefits:

- Salt generation
- One-way hashing
- Brute-force resistance

11.3 SQL Injection Prevention

Entity Framework Core uses:

- Parameterized queries
- ORM abstraction
- Query sanitization

Thus, SQL injection attacks are minimized.

---

12. ROLE-BASED ACCESS CONTROL (RBAC)

AcaemiaX supports two roles:

Master Admin

Permissions:

- Manage users
- View all branches
- Configure system
- View reports

Branch User

Permissions:

- Branch-specific data
- Student management
- Payment recording
- Admission creation

Restrictions:

- No access to system settings
- Cannot manage users
- Cannot access other branches

13. DOCKER DEPLOYMENT ARCHITECTURE

Docker containerization provides:

- Environment consistency

- Scalability
- Easy deployment
- Isolation

The application is divided into:

1. React Container
2. ASP.NET Container
3. PostgreSQL Container

13.1 Deployment Architecture

```
Client Browser
      ↓
Nginx Web Server
      ↓
React Frontend Container
      ↓
ASP.NET Core API Container
      ↓
PostgreSQL Container
```

13.2 Docker Compose

Deployment command:

```
docker-compose up -d
```

Benefits:

- Fast deployment
- Easy rollback
- Resource optimization

---

14. ALGORITHMS

Algorithm 1: User Authentication

Input: Username, Password

Output: JWT Token

Steps:

1. Receive credentials
2. Validate user
3. Verify password
4. Generate JWT
5. Return token

Complexity:

O(1)

Algorithm 2: Enquiry Conversion

Input: Enquiry

Output: Student Record

Steps:

1. Fetch enquiry
2. Populate student data
3. Generate Student ID
4. Save record

Complexity:

O(n)

Algorithm 3: Fee Installment Generation

Input:

- Total Fee
- Number of Installments

Output:

Installment Schedule

Formula:

Installment Amount = Total Fee / Number of Installments

---

15. PERFORMANCE EVALUATION

Performance metrics:

Operation	Response Time
Login	200 ms
Create Record	150 ms
Dashboard	800 ms
Reports	500 ms

15.1 Load Testing

Concurrent users tested:

- 10 Users

- 50 Users
- 100 Users

Results:

- Stable performance
- No major degradation
- Efficient memory usage

15.2 Security Testing

Security tests performed:

- Authentication testing
- Authorization testing
- SQL Injection testing
- XSS testing

Result:

All tests passed successfully.

---

16. RESULTS AND DISCUSSION

The developed system successfully manages:

- Enquiries
- Students
- Admissions
- Payments
- Reports

Advantages observed:

1. Reduced manual effort
2. Improved data consistency
3. Faster processing
4. Enhanced security
5. Better scalability

Comparison with traditional systems:

Parameter	Traditional	AcaemiaX
Scalability	Low	High
Security	Medium	High
Deployment	Difficult	Easy
Maintenance	Complex	Simple
Cloud Support	No	Yes

17. FUTURE SCOPE

Future enhancements for AcaemiaX include:

17.1 Artificial Intelligence

Implementation of:

- Student performance prediction
- Admission forecasting
- Course recommendation systems

17.2 Mobile Application

Development of:

- Android application
- iOS application

Features:

- Attendance
- Notifications
- Payment tracking

17.3 Kubernetes Deployment

Migration from Docker to Kubernetes for:

- Auto-scaling
- High availability
- Service orchestration

17.4 Multi-Tenant Architecture

Support for multiple institutions under a single platform.

17.5 Blockchain Integration

Digital certificates using blockchain technology.

---

18. CONCLUSION

This research presented **AcaemiaX**, a cloud-based enterprise student enquiry and course management platform developed using React, ASP.NET Core, PostgreSQL, and Docker.

The proposed system successfully integrates:

- Authentication

- Student management
- Admissions
- Fee processing
- Reporting
- Multi-branch support

Experimental evaluation demonstrated:

- Better scalability
- Improved security
- Efficient deployment
- Enhanced maintainability

AcaemiaX serves as a strong foundation for next-generation educational enterprise systems.

REFERENCES

[1] Newman, S. *Building Microservices*. O'Reilly Media, 2021. Available: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/

[2] Richardson, C. *Microservices Patterns*. Manning Publications, 2018. Available: https://microservices.io/

[3] Fowler, M. "Microservices." Available: https://martinfowler.com/articles/microservices.html

[4] Docker Documentation. Available: https://docs.docker.com/

[5] PostgreSQL Global Development Group. *PostgreSQL Documentation*. Available: https://www.postgresql.org/docs/

[6] Microsoft. *ASP.NET Core Documentation*. Available: https://learn.microsoft.com/aspnet/core/

[7] Meta. *React Documentation*. Available: https://react.dev/

[8] Microsoft. *Entity Framework Core Documentation*. Available: https://learn.microsoft.com/ef/core/

[9] Jones, M., Bradley, J., and Sakimura, N. "JSON Web Token (JWT)." RFC 7519, IETF, 2015. Available: https://datatracker.ietf.org/doc/html/rfc7519

[10] OWASP Foundation. *OWASP Top 10 Web Application Security Risks*. Available: https://owasp.org/www-project-top-ten/

[11] National Institute of Standards and Technology (NIST). *Cloud Computing Program*. Available: https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp

[12] Kubernetes Documentation. Available: https://kubernetes.io/docs/

[13] Fielding, R. T. "Architectural Styles and the Design of Network-based Software Architectures." Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[14] Mell, P., and Grance, T. "The NIST Definition of Cloud Computing." Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[15] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[16] Humble, J., and Farley, D. *Continuous Delivery*. Available: https://continuousdelivery.com/

[17] Sommerville, I. *Software Engineering*, 10th Edition. Pearson Education.

[18] Freeman, A., and Sanderson, A. *Pro ASP.NET Core*. Apress Publishing.

[19] Silberschatz, A., Korth, H., and Sudarshan, S. *Database System Concepts*, McGraw-Hill.

[20] OWASP Cheat Sheet Series. Available: https://cheatsheetseries.owasp.org/

Additional References for Educational Management Systems

[21] UNESCO Digital Learning Resources. Available: https://www.unesco.org/en/digital-learning

[22] IEEE Xplore Digital Library. Available: https://ieeexplore.ieee.org/

[23] ACM Digital Library. Available: https://dl.acm.org/

[24] Docker Compose Documentation. Available: https://docs.docker.com/compose/

[25] PostgreSQL Performance Tuning Guide. Available: https://wiki.postgresql.org/wiki/Performance_Optimization

---

ACKNOWLEDGEMENT

The authors express sincere gratitude to the project guide, faculty members, and institution for their continuous support and guidance during the development of AcaemiaX. Special thanks are extended to all contributors and testers who assisted in validating the system functionality and performance.