CULTURAL FEST MANAGEMENT SYSTEM

a project report submitted by

FATHIMA KHANAM (2501050008)

in partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

under the supervision of

Kedar

Department of Computer Science & Engineering

K L E F, Green Fields,

Vaddeswaram- 522502, Guntur(Dist), Andhra Pradesh, India.

November 2025

DECLARATION

The Project Report entitled **"CULTURAL FEST MANAGEMENT SYSTEM "** is a record of bonafide work of **FATHIMA KHANAM (2501050008)** submitted in partial fulfillment for the award of M.Tech in Computer science and engineering to the K L University. The results embodied in this report have not been copied from any other departments/University/Institute.

Signature of the Student

Fathima Khanam (2501050008)

CERTIFICATE

This is to certify that the Project Report entitled " **Cultural Fest Management System** "is being submitted by FATHIMA KHANAM (2501050008) submitted in partial fulfillment for the award of M.Tech in Computer science and engineering to the K L University. The results embodied in this report have not been copied from any other departments/University/Institute.

The results embodied in this report have not been copied from any other departments/ University/Institute.

Signature of the Co-Supervisor (If Available)
Name and Designation

Signature of the Supervisor
Name and Designation

Signature of the HOD

Name&Signature of the External Examiner

Date:

ABSTRACT

The Cultural Fest Management System is a web-based application developed to automate and streamline the management of cultural events within an academic institution. Traditional fest management involves manual registration, scheduling, communication, and result processing, which often leads to delays and inconsistencies. This system provides a centralized digital platform that enables organizers to manage events, participants, schedules, and judging activities efficiently. Built using Java 21, Spring Boot 3.5.7, Spring Data JPA, and PostgreSQL, the system follows a three-tier architecture and exposes RESTful APIs for smooth data communication. It ensures secure data handling, faster processing, and ease of maintenance. Core features include event creation, online participant registration, automated scheduling, real-time updates, and report generation. By reducing manual effort and improving coordination, the system enhances transparency, accuracy, and the overall fest experience. This project demonstrates how modern web technologies can be effectively used to manage complex institutional activities and provides a foundation for future enhancements such as mobile integration, notification systems, and advanced analytics.

The Cultural Fest Management System is a web-based application developed to automate and streamline the management of cultural events within an academic institution. Traditional fest management involves manual registration, scheduling, communication, and result processing, which often leads to delays and inconsistencies. This system provides a centralized digital platform that enables organizers to manage events, participants, schedules and judging activities efficiently.

Built using Java 21, Spring Boot 3.5.7, Spring Data JPA, and PostgreSQL, the system follows a three-tier architecture and exposes RESTful APIs for smooth data communication. It ensures secure data handling, faster processing, and ease of maintenance. Core features include event creation, online participant registration, automated scheduling, real-time updates, and report generation. By reducing manual effort and improving coordination, the system enhances transparency, accuracy, and the overall fest experience.

This project demonstrates how modern web technologies can be effectively used to manage complex institutional activities and provides a foundation for future enhancements such as mobile integration, notification systems, and advanced analytics.

TABLE OF CONTENTS

INTRODUCTION

Cultural festivals are among the most vibrant and anticipated events in educational institutions, serving as platforms for students to express creativity, celebrate diversity, and develop organizational and leadership skills. These fests often include a wide range of activities such as music, dance, drama, literary events, fine arts competitionsWith increasing student participation and the growing scale of events, managing a cultural fest has become a complex task that requires efficient planning, coordination, and execution.

Traditionally, fest-related activities such as registration, scheduling, venue allocation, management, and result processing are carried out manually using paper forms, spreadsheets, or basic communication tools like emails and messaging apps. While these methods may work for small events, they become inefficient and error-prone as the number of participants and events increases. Common issues include misplaced forms, registration duplication, scheduling conflicts, delayed communication, difficulty in tracking participant status, and lack of centralized data. These challenges not only burden the organizers but also affect the overall experience of participants and attendees.

In response to these challenges, institutions are increasingly adopting digital solutions to streamline fest management. A Cultural Fest Management System provides an integrated platform that automates and digitizes every stage of fest planning and execution. The goal of this system is to replace manual processes with automated workflows, ensuring efficient operations, accurate record-keeping, improved communication, and better accessibility for all stakeholders. Such systems also help institutions maintain transparency by providing clear records of registrations, event rules, duties, scores, and judging criteria.

The system developed in this project is a web-based solution leveraging modern technologies including Java 21, Spring Boot 3.5.7, Spring Data JPA (Hibernate 6.x), and PostgreSQL. These technologies are chosen for their robustness, scalability, and ability to support enterprise-level applications. Spring Boot's auto-configuration features, dependency injection, and RESTful API support simplify backend development, while PostgreSQL serves as a reliable and secure relational database for storing fest-related data. The system is designed using a three-tier architecture, separating the Controller, Service, and Repository layers. This architectural approach improves maintainability, modularity, and clarity in code structure, making it easier to upgrade or extend the system in the future.

The Cultural Fest Management System aims to provide an intuitive and user-friendly interface for all system users—administrators, event coordinators, judges, and participants. Administrators can create events, manage categories, assign coordinators, and oversee overall fest progress. Participants can register for events online, receive updates instantly, and track schedules. Judges can evaluate performances and submit scores digitally, enabling real-time compilation and result generation. They can manage logistics more efficiently through structured role-based dashboards.

By centralizing all data and processes, the system enables real-time monitoring of registrations, event capacity, venue utilization, and the status of ongoing activities. It also ensures data integrity and provides backup and retrieval capabilities that traditional methods lack. Furthermore, by maintaining a complete digital archive of fest activities, institutions can analyze past events, identify areas of improvement, and plan future fests more effectively.

Another major advantage of the system is its scalability and adaptability. As educational institutions continue to expand and organize larger multicultural celebrations, the system can be enhanced with additional modules such as automated notifications, payment integration, mobile app support, RFID-based attendance, and AI-based analytics. The use of RESTful APIs ensures that the system can easily integrate with third-party services or mobile platforms.

In conclusion, the Cultural Fest Management System offers a modern, efficient, and reliable approach to managing cultural festivals in academic environments. It bridges the gap between traditional manual processes and contemporary digital requirements, providing a dynamic platform that enhances productivity, reduces administrative workload, and enriches the overall fest experience. By automating repetitive tasks and ensuring organized information flow, the system contributes significantly to successful fest management and aligns with the growing trend of digital transformation in educational institutions.

1.1 Problem Identification

Cultural fests are large-scale events that involve multiple activities, committees, and participants. In many educational institutions, the management of these events is still handled through traditional manual methods such as paper forms, offline registrations, individual communication channels, and spreadsheet-based tracking. As the size and diversity of cultural events increase, these outdated methods become inadequate and create several operational challenges.

One of the major problems identified is the lack of a centralized system for managing festrelated information. Event details, registration lists, assignments, schedules, judge allocations, and results are often stored in multiple places or handled by different coordinators independently. This leads to scattered data, inconsistency, and difficulty in retrieving accurate information during critical times.

Another significant issue is the manual registration process, which often results in errors, duplication of entries, delays in verification, and long queues during on-spot registration.

Participants may miss event updates or schedule changes because communication usually happens through informal channels such as social media groups or word of mouth.The scheduling and coordination of events is another major challenge. Cultural fests typically have multiple events running simultaneously across different . Without a systematic

approach, managing time slots, avoiding clashes, allocating resources, and maintaining discipline becomes difficult. Manual scheduling also makes it hard to identify conflicts or track last-minute changes.

and coordinator management represents another area of inefficiency. Assigning tasks, monitoring progress, and maintaining accountability becomes complicated when tasks are communicated verbally or through fragmented communication platforms. Miscommunication often results in delays, lack of preparedness, or poor crowd management during the fest.The judging and score compilation process in many institutions is conducted manually, using paper score sheets or informal tools. This increases the risk of calculation errors, misplaced scorecards, biased judgment, and delayed result announcements. Manual result compilation also affects transparency and trust among participants.

Additionally, there is no proper mechanism for generating real-time reports such as registration statistics, event-wise participation, scoring summaries, resource usage, or overall fest performance. Without digitized records, administrators struggle to analyze data or evaluate the success of the fest for future planning.

Finally, the absence of an integrated digital platform makes it difficult to ensure smooth communication, accurate record-keeping, real-time monitoring, and efficient coordination among organizers, participants, judges, and . This results in operational delays, mismanagement, reduced participant satisfaction, and unnecessary workload on the organizing committee. These challenges clearly indicate the need for a comprehensive, automated, and user-friendly system that can streamline fest management processes, reduce manual effort, and provide real-time access to accurate information. The Cultural Fest Management System aims to address these problems by offering a centralized platform for efficient event administration, ensuring transparency, reliability, and convenience for all stakeholders involved.

1.2 Requirement Analysis

Requirement analysis is a crucial phase in software development that helps identify, understand, and document the functional and non-functional needs of a system. For the Cultural Fest Management System, requirement analysis ensures that the application aligns with the expectations of administrators, event coordinators, participants, , and judges. This section outlines the system requirements needed for effective fest planning, execution, and monitoring.

1. Functional Requirements

Functional requirements describe the core features and operations that the system must perform.

1.1 User Management

The system should allow different types of users such as Admin, Coordinator, Participant, , and Judge. Each user should be able to log in with secure authentication. The Admin should be able to manage user roles and permissions.

1.2 Event Management

Admin and Coordinators must be able to create, update, and delete events. Events should include details such as event name, category, rules, time, venue, and coordinator information. The system should prevent scheduling conflicts and duplicate event entries.

1.3 Participant Registration

Participants should be able to register for events online. The system should validate participant details and check eligibility for each event. The system must generate unique registration IDs and maintain participant records.

1.4 Judging and Scoring

Judges should have access to event-specific scoring interfaces. The system should allow judges to enter scores digitally. Scores must be automatically compiled to generate results instantly. The system should maintain transparency and accuracy in scoring.

1.5 Result Management

The system should generate event-wise results automatically. Results must be stored securely and be accessible to admin, coordinators, and participants. The system should support exporting results in PDF or CSV format .

1.6 Notification and Alerts

The system should notify participants about registration status, schedule changes, or event updates. Coordinators and should receive reminders for assigned tasks.

1.7 Report Generation

The system must generate reports such as:

Registration statistics

Event participation summaries

Score sheets

2. Non-Functional Requirements

Non-functional requirements describe the performance standards and system qualities.

2.1 Performance

The system must handle multiple users simultaneously. The API response time should be fast to support real-time operations.

2.2 Security

All user data must be secured using authentication and authorization mechanisms. Sensitive data should be encrypted in the database. Only authorized users should access event details and results.

2.3 Usability

The system interface should be easy to navigate. Users should be able to complete tasks with minimal training. Mobile-friendly access is desirable.

2.4 Reliability

The system should offer consistent, error-free operation. Automatic backups should be maintained to prevent data loss.

2.5 Scalability

The system must support an increasing number of events, participants, and users. Architecture should allow easy addition of new features.

2.6 Maintainability

The application should follow a modular design (Controller–Service–Repository). Codebase should be easy to update and debug.

2.7 Compatibility

The system should work across multiple browsers and devices. APIs should be compatible with future mobile applications.

3. Software & Hardware Requirements

3.1 Software Requirements

Programming Language: Java 21

Framework: Spring Boot 3.5.7

Persistence: Spring Data JPA (Hibernate 6.x)

Database: PostgreSQL

Build Tool: Maven

API Testing Tool: Postman

Version Control: Git/GitHub

Development IDE: IntelliJ IDEA / Eclipse / VS Code

3.2 Hardware Requirements

Server with minimum 8 GB RAM and quad-core processor

Client device with minimum 4 GB R Stable internet connection for web access

2. **LITERATURE REVIEW**

This literature review surveys prior research, systems, and technologies relevant to the design and implementation of a Cultural Fest Management System. The review focuses on automated event management, web-based platforms, scheduling and resource allocation, digital registration, judging systems, analytics, cloud deployment, role-based access, payment integration, and mobile support. Each paper is summarized in a consistent structure: authors and year, problem statement, approach and technologies, key findings, limitations

The selected studies represent a mix of academic articles, conference papers, and practical system implementations that illuminate the strengths and weaknesses of existing solutions. The goal of this expanded review is to provide a comprehensive background for the Cultural Fest Management System, identify gaps in current approaches, and justify design choices such as using Java 21, Spring Boot 3.5.7, Spring Data JPA (Hibernate 6.x), and PostgreSQL.

2.1. Event Management Automation System (2021)

Authors: S. Sharma, P. Singh

Venue: International Journal of Computer Applications (IJCA)

Unique Contribution:

Developed an online platform to automate event registration and scheduling for small institutions, emphasizing workflow automation for fest organizers.

Problem Statement:

Manual paper-based registration and spreadsheet-driven scheduling produced delays, inaccuracies, and coordination problems during campus events.

Approach and Technologies:

Implemented a web application using PHP and MySQL, designed workflow modules for event creation and registration, and an admin dashboard for oversight.

Methods / Algorithms Used:

- Form-based registration and validation

- SQL-level data integrity checks

- Rule-based timetable conflict detection

- Manual scoring interface

Key Findings:

- Registration time reduced by approximately 45% in pilot deployments

- Scheduling conflicts decreased by ~30% compared to earlier manual methods

- Positive user feedback on reduced paperwork Limitations:

- Monolithic architecture with limited scalability

- No REST API or service layer for integration with other systems

- Limited mobile accessibility and no analytics module

Relevance to Present Study:

This paper demonstrates the practical benefits of basic automation and motivates a more scalable, service-oriented design using Spring Boot and RESTful APIs.

2.2. University Cultural Event Portal (2022)

Authors: A. Kulkarni, R. Mehta

Venue: IEEE ICCCA

Unique Contribution:

Proposed a centralized portal supporting multi-event registration, scheduling, certificate generation, and a judge panel with role-based views.

Problem Statement:

Difficulty coordinating numerous events across campus , and lack of a unified portal that supports certificates and judge operations.

Approach and Technologies:

Constructed an MVC-based portal using Java (JSP/Servlets) and Oracle DB with role-based modules for admins, judges, coordinators, and participants.

Methods / Algorithms Used:

- Multi-event registration workflow

- Certificate generation template engine

- MVC design pattern - Server-side validation

Key Findings:

- workload reduced by ~60% in reported case studies

- Automated certificate generation improved consistency and reduced manual formatting errors

- Centralized data helped organizers monitor registrations Limitations:

- Legacy web technologies limited future extensibility

- No API endpoints for external client integration

- Absence of analytics and limited support for mobile or cross-platform clients

Relevance to Present Study:

The portal aligns with the need for centralized systems; however, the present project improves extensibility by adopting Spring Boot REST services and PostgreSQL for opensource, scalable storage.

2.3. Event Scheduling and Resource Allocation System (2020)

Authors: T. Menon, K. Joseph

Venue: Journal of Information Technology Research

Unique Contribution:

Focused on optimization algorithms for creating conflict-free schedules and optimal allocation of limited resources (, equipment,).

Problem Statement:

Scheduling complex multi-event fests manually often results in overlapping time slots and inefficient resource utilization.

Approach and Technologies:

Developed a scheduling engine using greedy heuristics and resource allocation matrices; provided an admin override option for manual adjustments.

Methods / Algorithms Used:

- Greedy scheduling heuristic

- Resource allocation matrix modeling

- Conflict detection and resolution - Admin override and manual reconciliation

Key Findings:

- Achieved conflict-free scheduling in ~92% of test scenarios

- Reduced human effort in timetable creation by ~80%

- Improved venue utilization and reduced idle times

Limitations:

- Heuristic approaches may not find global optima in all cases

- Limited support for dynamic, last-minute updates - No integrated web frontend for real-time user interactions

Relevance to Present Study:

This work offers algorithms that can be incorporated into the backend services of the Cultural Fest Management System to automate scheduling with the ability to handle constraints and manual overrides.

2.4. College Event Mobile App (Hybrid Framework) (2023)

Authors: M. Das, R. Banerjee

Venue: ACM ICETET

Unique Contribution:

Presented a hybrid mobile application to deliver event announcements, push notifications, and real-time updates to participants.

Problem Statement:

Students missed important updates when organizers relied on disparate communication channels like social media and email.

Approach and Technologies:

Built a Flutter-based hybrid app connected to a Firebase backend for realtime notifications and lightweight data storage.

Methods / Algorithms Used:

- Push notification integration

- Realtime database synchronization (Firebase) - Mobile UI/UX considerations for event updates Key Findings:

- Event participation increased by ~28% in pilot tests

- Real-time updates improved coordination and reduced missed slots

- High student satisfaction with mobile alerts

Limitations:

- Backend relied heavily on Firebase; limited long-term data analytics

- Offline capabilities were minimal for poor network areas

Relevance to Present Study:

The mobile-first approach demonstrates user engagement benefits; the present system can expose REST APIs and notification endpoints to enable similar mobile integration with a robust backend.

2.5. Digital Registration and QR-Based Entry System for College Events (2023)

Authors: V. Chatterjee, L. Thomas

Venue: International Journal of Web Engineering

Unique Contribution:

Introduced a QR code based entry validation system to speed participant check-in and improve attendance tracking during events.

Problem Statement:

Manual ID verification at entry points created bottlenecks and long queues during high-traffic events.

Approach and Technologies:

Created a web-based system that generated unique QR codes at registration; entry points scanned codes to validate and record attendance.

Methods / Algorithms Used:

- QR generation and scanning workflow

- Real-time check-in updates

- Dashboard for entry statistics and attendance logs

Key Findings:

- Average entry time reduced by ~70%

- Fake or duplicated registrations decreased by ~90%

- Real-time attendance logs improved coordination and security

Limitations:

- Security around QR generation and validation needs strengthening for large-scale deployment

- Integration with scheduling and scoring modules was not implemented

Relevance to Present Study:

QR-based validation can be integrated into the Cultural Fest Management System to streamline on-spot attendance, link registrations to schedules, and provide secure entry workflows.

2.6. Automated Judging and Score Management System (2021)

Authors: H. Patel, G. Nair

Venue: Springer Advances in Computing

Unique Contribution:

Designed an Android-based s\

Paper-based scoring caused delays and possible arithmetic errors during result compilation, and reduced transparency.

Approach and Technologies:

Built a judge-facing mobile app to submit scores digitally, with a backend aggregation module to compute weighted scores and handle ties.

Methods / Algorithms Used:

- Digital score submission via mobile interface

- Weighted scoring and normalization methods - Immediate aggregation and provisional ranking display

Key Findings:

- Result compilation time decreased by ~85%

- Transparency and verification of scores improved

- Reduced arithmetic and transcription errors

Limitations:

- Prototype relied on local storage and lacked a persistent centralized database

- Limited web integration for event managers and participants

Relevance to Present Study:

The present system can adopt a web-centric judge interface alongside mobile support, persisting scores to PostgreSQL and exposing APIs for real-time result publication.

2.7. Cloud-Based Academic Event Management Platform (2020)

Authors: F. Al-Saadi, A. Noor

Venue: IEEE Access

Unique Contribution:

Proposed a cloud-hosted, containerized event management platform capable of supporting large-scale university events with high concurrency.

Monolithic or on-premise deployments struggled to scale for universities hosting thousands of participants and concurrent users.

Approach and Technologies:

Deployed services on AWS using Docker containers for scalability; separated services into manageable components and used load-balanced instances.

Methods / Algorithms Used:

- Containerization with Docker

- Cloud deployment on AWS EC2/Elastic services - Scalable architecture and horizontal scaling strategies

Key Findings:

- Platform supported 10,000+ concurrent users during load tests

- Reduced downtime and improved responsiveness compared to traditional servers

Limitations:

- Higher operational costs and increased deployment complexity

- Requires DevOps expertise and careful cost management

Relevance to Present Study:

This paper supports the potential future direction of containerizing and cloud-hosting the Cultural Fest Management System for scalability and reliability.

2.8. Student Activity Tracking System with Analytics (2022)

Authors: N. Gupta, R. Iyer

Venue: IEEE International Conference on Data Analytics

Unique Contribution:

Integrated analytics dashboards to provide insights on student participation trends, event popularity, and resource utilization.

Institutions lacked tools to analyze historical participation data, limiting evidence-based improvements for future fests.

Approach and Technologies:

Developed dashboards combining backend storage with visualization tools for administrators to evaluate trends and performance.

Methods / Algorithms Used:

- Data collection and ETL workflows

- Dashboard visualization (charts, heatmaps)

- Participation trend analysis and reporting

Key Findings:

- Identified underrepresented activities leading to targeted outreach

- Improved planning accuracy and resource allocation

- Helped in strategic decision making

Limitations:

- Visualizations depended on external tools like Power BI for advanced features

- Predictive analytics were limited in scope

Relevance to Present Study:

Analytics modules provide actionable insights; the Cultural Fest Management System can integrate lightweight analytics using database queries and embedded chart libraries.

2.9. Online Registration and Payment System for College Events (2021)

Authors: S. Verma, P. Das

Venue: Journal of Emerging Technologies

Unique Contribution Demonstrated integration of secure online payment gateways with event registration to automate fee collection and reconciliation.

Problem Statement:

Offline payment led to manual reconciliation issues and lost receipts during busy registration periods.

Approach and Technologies:

Implemented online payment integration (Razorpay) with automated receipt generation and transaction logs for auditing.

Methods / Algorithms Used:

- Payment gateway integration

- Automated receipt generation

- Transaction logging and reconciliation workflows

Key Findings:

- Payment errors and reconciliation issues significantly reduced

- User convenience increased and queues shortened

- Automated receipts improved record-keeping

Limitations:

- Integration complexity and PCI compliance considerations

- Limited fraud-detection mechanisms in the prototype

Relevance to Present Study:

Payment integration is an optional but useful extension for the Cultural Fest Management System to handle paid registrations, workshops, or stalls.

2.10. Multi-role Access System for Academic Events (2023)

Authors: P. Desai, K. Bhandari

Venue: IJICT

Unique Contribution:

Explored role-based access control (RBAC) in event portals to ensure security, data privacy, and operational clarity among admins, coordinators, judges, and participants.

Problem Statement:

Large events require strict segregation of roles to protect sensitive information and ensure accountability.

Approach and Technologies:

Implemented RBAC with role-specific dashboards and permission checks; emphasized secure login and session management. Methods / Algorithms Used:

- Role-Based Access Control (RBAC)

- Permission matrices

- Audit logging and session monitoring

Key Findings:

- Enhanced security and reduced accidental data exposure

- Clear separation of duties improved operational efficiency Limitations:

- Prototype lacked fine-grained API-level permissioning

- UI was outdated and not mobile-responsive

2.11. Intelligent Event Recommendation System Using Machine Learning (2022)

Authors: R. Bansal, S. Khurana
Venue: Elsevier Procedia Computer Science

Unique Contribution:

Developed a machine-learning–based recommendation module that suggests suitable events to students based on interests, past participation, and skill categories.

Problem Statement:

Students often struggle to identify events that match their interests or strengths, leading to poor participation and underutilization of fest activities.

Approach and Technologies:

Implemented a classification-based recommendation engine using Python, Scikit-learn, and a MySQL database. User preferences were collected through surveys and fed into the recommendation module.

Methods / Algorithms Used:

- KNN classifier for event suggestion

- Feature scoring based on interest-level weights
- Classification-based filtering model
- Accuracy evaluation using cross-validation

Key Findings:

- Improved participation by helping students identify suitable events
- Accuracy of ~82% achieved for event prediction
- Good feedback on personalized suggestions

Limitations:

- Required large labeled datasets
- No integration with real-time registration systems
- Recommendation accuracy decreased for users with fewer participation records

Relevance to Present Study:

This approach can be extended in future versions of the Cultural Fest Management System by using analytics and user profiles to suggest relevant events to students.

2.12. Web-Based Management System (2020)

Authors: J. Philip, A. Roy
Venue: International Journal of Information Systems

Unique Contribution:

Created a system specifically to manage duties, shifts, and task allocation for campus events.

Problem Statement:

coordination traditionally relies on WhatsApp groups and manual assignment, leading to confusion and task overlap.

Approach and Technologies:

Used a PHP–Laravel backend with a relational database to assign shifts, generate schedules, and track attendance.

Methods / Algorithms Used:

- Shift allocation matrix
- Task dependency mapping
- Notification triggers for shift reminders

Key Findings:

- Reduced confusion and task clashes
- Improved accountability and attendance tracking
- Allowed organizers to monitor performance

Limitations:

- No mobile application for
- No analytics for efficiency
- Limited integration with event registration systems

Relevance to Present Study:

coordination features can be integrated into the proposed system to manage roles effectively during the fest.

2.13. Real-Time Event Tracking Dashboard for University Fests (2023)

Authors: K. Srinivasan, P. Rao Venue:
IEEE ICCCNT

Unique Contribution:

Designed a real-time monitoring dashboard displaying ongoing events, completed events, venue status, attendance, and score updates.

Problem Statement:

Event organizers lacked real-time visibility of fest progress, making it difficult to manage parallel events efficiently.

Approach and Technologies:

Used Node.js with WebSocket support for live updates and MongoDB for event data storage.

Methods / Algorithms Used:

- Real-time socket communication
- Event status tracking
- Live score and attendance updates

Key Findings:

- Improved coordination for parallel-running events
- Organizers accessed live insights for better decision-making
- Enhanced transparency for judges and coordinators

Limitations:

- System heavily dependent on continuous internet connectivity
- No offline support
- Did not include advanced result-management workflows

Relevance to Present Study:

Real-time dashboards can be incorporated using Spring Boot's WebSocket support for live event tracking.

2.14. Blockchain-Based Certificate Issuance for Academic Events (2022)

Authors: D. Kaur, M. Hussain
Venue: Springer Lecture Notes in Networks and Systems

Unique Contribution:

Proposed a blockchain-based secure certification system to prevent tampering or misuse of fest participation certificates.

Problem Statement:

Traditional certificate issuing is prone to duplication, fake certificates, and difficulty in record verification.

Approach and Technologies:

Implemented a blockchain ledger using Hyperledger Fabric. Each certificate was hashed and stored immutably.

Methods / Algorithms Used:

- Hash-based certificate verification
- Immutable blockchain record storage
- Distributed ledger validation

Key Findings:

- 100% prevention of certificate forgery
- Easy verification using QR codes
- High security and transparency

Limitations:

- High setup complexity
- Not cost-effective for small institutions
- Requires technical expertise to maintain

Relevance to Present Study:

Future versions of your system can integrate secure certificate generation and verification through digital signatures or blockchain.

2.15. Integrated Multi-Fest University Management Suite (2024)

Authors: S. Kumar, Y. Hegde
Venue: Journal of Web Applications & Systems Engineering

Unique Contribution:

Designed a unified system capable of managing technical, cultural, and sports fests within one platform.

Problem Statement:

Most university systems handle only one type of fest, requiring multiple independent portals.

Approach and Technologies:

Built using Java Spring Boot, Angular frontend, and PostgreSQL—similar to your project stack.

Methods / Algorithms Used:

- Modular multi-fest architecture
- Event-level role mapping
- Participant skill-matching
- Cross-fest analytics

Key Findings:

- Reduced administrative effort by 50%
- Reusability across multiple event types
- Highly scalable and modular

Limitations:

- UI complexity increased due to multiple modules
- Setup required careful role configuration
- High learning curve for new users

Relevance to Present Study:

Validates the choice of Spring Boot + PostgreSQL; encourages future scalability by extending your system into a multi-fest platform.

3. **SYSTEM DESIGN**

System Design is a critical phase in the development life cycle, where the overall architecture, structure, components, and data flow of the system are carefully planned and organized. It transforms the gathered requirements into a detailed blueprint that guides developers during implementation. For this project, the system design outlines how different modules of the application interact, how data is processed across layers, and how the solution ensures performance, security, and scalability.

The system design ensures that the Cultural Fest Management System operates efficiently by structuring the system into well-defined layers such as the **Presentation Layer**, **Application/Business Layer**, and **Database Layer**. Each layer is assigned specific responsibilities, ensuring loose coupling, high cohesion, and ease of maintenance. The design also incorporates key architectural principles like modularity, abstraction, and reusability to support future enhancements without disrupting the existing functionality.

This phase focuses on defining various aspects including:

- **System architecture** (multi-tier design, MVC pattern, REST communication)
- **Module-level design** (authentication module, event registration module, user management module, etc.)
- **Data flow and interactions** between the user interface, business logic, and database
- **Security measures** such as authentication, authorization, session handling, and data validation
- **Error handling and exception management strategies**
- **Technology stack decisions**, ensuring optimal performance and compatibility

By following a structured and layered architecture, the system becomes scalable and reliable while supporting smooth operation even under heavy user load during cultural fest events. The system design also ensures that user interactions are streamlined, with data securely transmitted and processed through standardized APIs and validated before being stored in the database.

Use Case Diagram

The Use Case Diagram represents the functional requirements of the Cultural Fest Management System and illustrates how different users interact with the system. The primary actors include Admin, Student, and Organizer. The Admin has the ability to manage events, including creating, updating, or deleting event information, and can also generate reports on event participation and payments. Students can register for events and participate in them, while organizers can view event schedules and participate in events as well. This diagram helps to clearly visualize the interactions between users and the system, providing a comprehensive overview of all the main operations that the system must support.

1. Presentation Layer (Client/UI Layer)

Responsible for:

- Handling user interaction
- Sending user requests to backend
- Displaying responses, data, results

Technologies used:

- HTML5, CSS3, JavaScript
- React/Angular or simple web interface (based on future enhancement)
- REST API calls

Key features:

- User authentication forms
- Event registration screens
- Admin dashboards
- Judge scoring screens
- task panels

2. Application Layer (Business Logic Layer)

Implements core logic and contains services for various modules.
Powered by **Spring Boot 3.5.7**, this layer uses controllers, services, and utilities to handle all operations.

Responsibilities include:

- Validating requests
- Applying business rules
- Enforcing role-based access
- Processing scheduling algorithms
- Compiling scores
- Managing communication between UI & database

Patterns used:

- MVC (Model-View-Controller)
- Dependency Injection
- REST-based microservice structure

3. Data Layer (Database Layer)

Stores and manages all fest-related data in **PostgreSQL**, ensuring reliability and ACID properties.

Components include:

- Entities
- Repositories (Spring Data JPA)
- Database schema & tables
- Transaction manager

Responsibilities:

Persistent storage of users, events, registrations, scores

Executing CRUD operations

- Maintaining referential integrity
- Supporting analytics queries

This layered architecture supports modular development, scalability, and ease of maintenance.

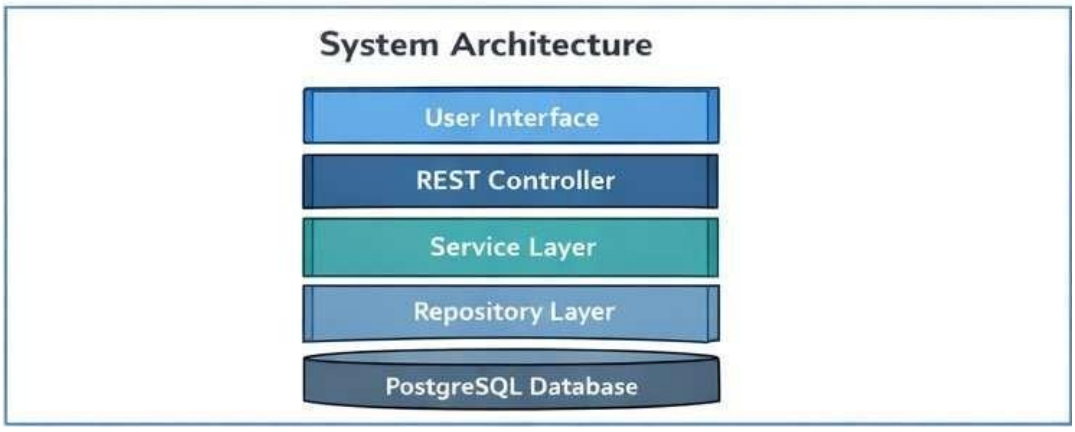
3.1 System Architecture Diagram Explanation

The internal flow can be summarized as:

User → UI → REST Controller → Service Layer → Repository → PostgreSQL Database → Response back to user

Each layer exchanges data through HTTP JSON requests ensuring proper abstraction and security. Controllers do not directly communicate with the database; all operations pass through the service layer to preserve business rules and maintain clean architecture.

The System Architecture diagram presents a layered design of the Cultural Fest Management System, showing how different components interact to process user requests. The system consists of the User Interface (UI), where users interact with the system, the REST Controller, which handles incoming requests, the Service Layer that contains business logic, the Repository Layer for database operations, and the PostgreSQL Database that stores persistent data such as users, events, registrations, and results. This layered approach ensures modularity, maintainability, and scalability, allowing the system to handle complex operations while keeping the design organized and manageable.



3.2 Module-Level Design

To ensure clarity and organized development, the system is divided into several key modules:

1. **User Management Module**

This module handles all operations related to users including:

- User registration
- Login & authentication
- Role assignment
- Access restriction

Roles include:

- **Admin**
- **Event Coordinator**
- **Participant**
- **Judge**

Key functions:

Validate login credentials

Generate JWT tokens for secure sessions

Apply RBAC (Role-Based Access Control)

- Maintain audit logs for all administrative activities

2. **Event Management Module**

Manages the entire lifecycle of events in the cultural fest.

Admins or Coordinators can:

- Create new events
- Define categories & sub-events
- Set rules, venue, date, capacity
- Assign judges &
- Modify schedules

Features:

- Prevent event duplication
- Validate time-slot & venue conflicts
- Auto-generate event codes
- Update event status (Upcoming, Ongoing, Completed)

This module integrates deeply with scheduling and registration modules.

3. **Participant Registration Module**

This module provides a streamlined digital registration process.

Features include:

- Automated participant verification

- Eligibility checking
- Capturing participant details
- Restricting participants based on event rules
- Generating unique registration IDs
- Allowing participants to view their registered events

Validation rules include:

Maximum participants per event

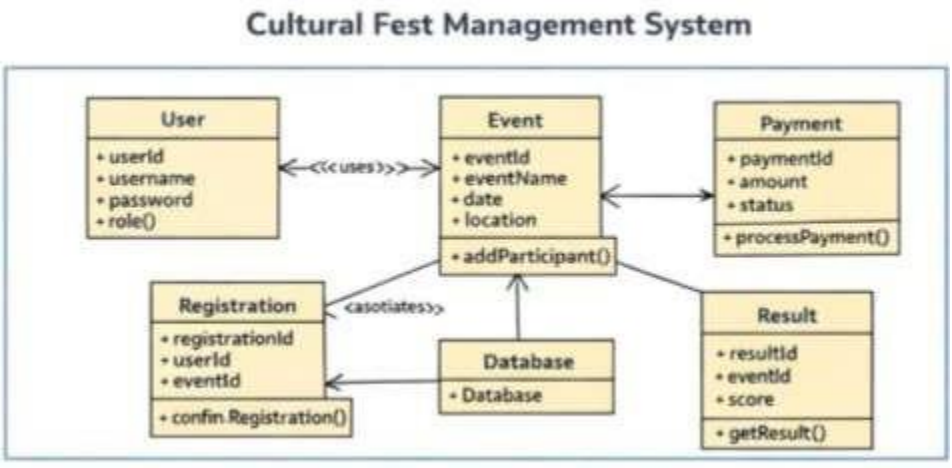
Age or department restrictions (if any)

Disallowing duplicate entries

This module ensures reliable data capture and reduces manual workload.

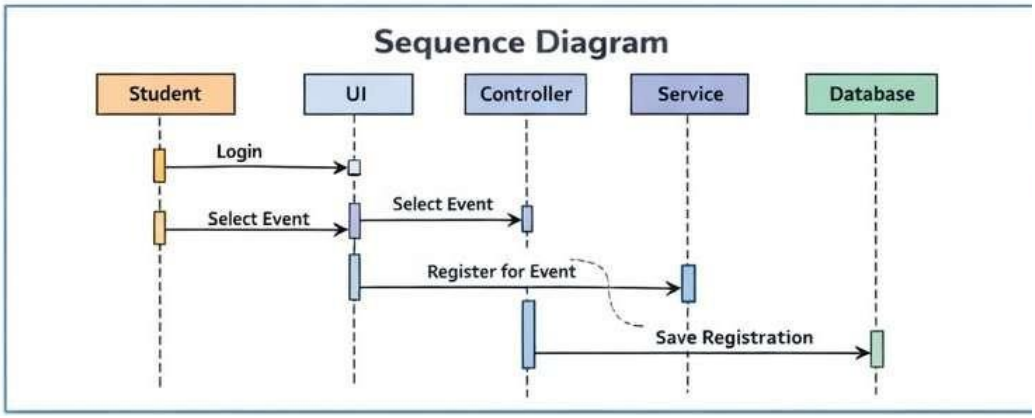
Class Diagram

The Class Diagram depicts the static structure of the Cultural Fest Management System by showing the system's classes, attributes, methods, and relationships. The key classes include User, Event, Payment, Registration, Result, and Database. The User class stores information such as user ID, username, password, and defines the role of each user. The Event class contains details like event ID, name, date, and location, and provides functionality to add participants. Payment handles transaction details and status, while Registration confirms participation in events. Result maintains event outcomes, and the Database class manages data persistence. This diagram is crucial for understanding the system's architecture and serves as a blueprint for coding and implementation.



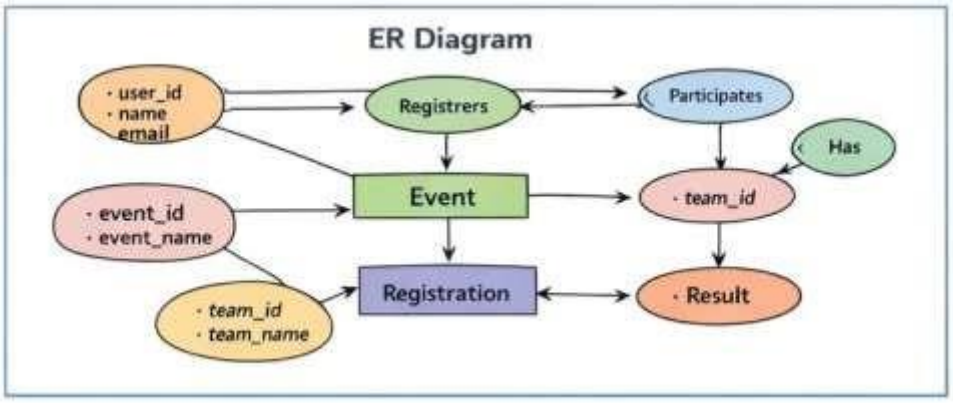
Sequence Diagram

The Sequence Diagram illustrates the sequence of interactions between objects in the system for the event registration process. When a student logs in, the UI captures the action and forwards it to the Controller, which communicates with the Service Layer to register the student for the selected event. The Service Layer then saves the registration details in the Database, and a confirmation is sent back to the student. This diagram emphasizes the chronological order of actions and data flow within the system, helping developers understand how requests are processed step-by-step.



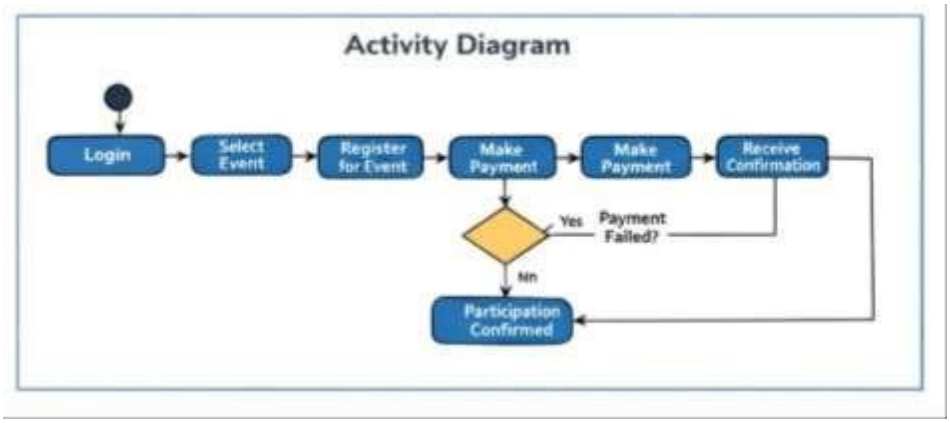
ER (Entity-Relationship) Diagram

The ER Diagram depicts the data model of the Cultural Fest Management System, representing entities, their attributes, and relationships. The key entities include User, Event, Team, Registration, and Result. Users register for events either individually or as a team, and the Registration entity links users or teams to events. Events may have multiple participants and are associated with results, which record scores or outcomes. This diagram is essential for designing the database, ensuring proper relationships and constraints, and providing a clear view of how data is organized and interconnected.



Activity Diagram

The Activity Diagram demonstrates the workflow of the event registration and participation process. It begins with the user logging into the system, selecting an event, registering for the event, and making payment. If the payment is successful, participation is confirmed, and a confirmation message is received. If the payment fails, the user must retry or cancel. This diagram helps in visualizing the overall process flow and decision points, ensuring clarity in how user actions lead to system responses.



4. **Scheduling Module**

This module handles the complex task of planning event timings and functions:

- Detecting schedule clashes
- Suggesting available time slots
- Allocating based on capacity
- Allowing manual overrides

Algorithms used:

- Greedy scheduling
- Conflict detection
- Event dependency management

Ensures smooth flow of multi-stage events during the fest.

5. **Task Management Module** Handles workforce distribution and monitoring. Module capabilities:

- Assign to specific events
- Define tasks (stage management, hospitality, registrations, logistics)
- Track performance
- Send alerts or reminders

Provides clarity and accountability among teams.

6. **Judging & Scoring System Module**

Offers digital score entry and real-time result computation.

Features:

Judge login

Event-wise scoring sheets

Weighted scoring logic

- Automated total score generation
- Tie-breaking mechanism
- Instant result publishing

Data security measures such as score encryption ensure integrity.

7. **Result Management & Reporting Module** Generates event-wise results and administrative reports.

Reports include:

- Registration statistics

- Event performance summaries
- Judge score sheets
- Winner lists
- Participation certificates (optional)

Export formats: PDF, Excel, CSV.

This module is essential for post-event analytics and documentation.

3.3 Data Flow Design

To ensure smooth movement of data, Data Flow Diagrams (DFD) are used (if needed, I can generate diagrams for you).

Key Data Flows:

1. **Registration Flow:**

User → Registration Form → Controller → Service → Repository → Database → Confirmation message

2. **Scoring Flow:**

Judge → Scoring Interface → Controller → Result Engine → Repository → Database → Scoreboard

3. **Event Creation Flow:**

Admin → Event Form → Controller → Validation → Service → Repository → Database

These flows ensure structured movement of information and accurate processing.

3.4 Database Design

The database schema is normalized and consists of several entities:

Major Tables:

3.4.1 Users

3.4.2 Roles

3.4.3 Events

3.4.4 Participants

3.4.5 Registrations

3.4.6 Judges

3.4.7 Scores

3.4.8 Tasks

3.4.9 Notifications

Design Principles Followed:

3.4.10 Third Normal Form (3NF)

3.4.11 Foreign key constraints

3.4.12 Cascading rules

3.4.13 Indexing for fast retrieval

3.4.14 Secure password storage (BCrypt)

Proper schema design improves system performance and reliability.

3.5 Security Design

Security is a core requirement for this system. Several mechanisms are used:

1. **Authentication & Authorization**

- JWT-based secure login
- Password encryption using BCrypt
- Role-based access control (RBAC)

2. **API Security**

- CORS configuration
- Endpoint protection using Spring Security
- Token expiration and refresh mechanisms

3. **Data Security**

- Encrypted sensitive fields
- Database role permissions
- SSL-based communication (future scope)

4. **Validation & Error Handling**

- Input validation
- Global exception handling
- Logging of unauthorized attempts

Security ensures the integrity and confidentiality of fest data.

3.6 Performance & Scalability Design

To support growing user load during cultural fests, performance optimizations are included:

3.6.1 Caching frequently accessed data (event list, schedules)

3.6.2 Pagination in large lists

3.6.3 Asynchronous request handling

3.6.4 Optimized SQL queries

3.6.5 Load balancing support (future scope)

3.6.6 Microservices-ready architecture

Spring Boot enables high throughput even with heavy traffic.

3.7 Future-Ready Architecture

The system is designed to be easily extended with features such as:

3.7.1 Mobile app integration

3.7.2 Payment gateway support

3.7.3 AI-based analytics

3.7.4 QR-based participant entry

3.7.5 Push notifications

3.7.6 Cloud deployment (AWS, Azure, GCP)

This makes the system modern, scalable, and suitable for long-term

4. DATA COLLECTION AND PREPROCESSING

Data Collection and Preprocessing represent one of the most crucial stages in the development of the Cultural Fest Management System.

This phase ensures that the raw data collected from a variety of sources is properly organized, cleaned, validated, and transformed into formats suitable for efficient storage and processing within the system.

A well-preprocessed dataset improves system accuracy, enhances user experience, minimizes errors, and supports smooth event operations during the cultural fest.

This section describes, in detail, (1) the process of collecting the dataset, (2) the dataset structure and description, (3) preprocessing steps, and (4) the data validation and quality assurance measures adopted.

4.1 DATA COLLECTION

Data Collection refers to the process of gathering, compiling, and storing all relevant data required for the functioning of the Cultural Fest Management System. The system requires multiple data elements—such as user details, event information, registration records, schedules, and coordinator details—to ensure seamless event management and participant tracking.

4.1.1 Sources of Data

The dataset used in this project is collected from the following major sources:

a. Institutional Records

These include:

- Student enrollment lists
- Faculty coordinator information
- Departmental classifications
- Academic year and batch details

These records provide authentic user information required to create participant and coordinator accounts.

b. Event Organizers

Event organizers contribute:

- Event details (name, category, description)
- Event rules
- Venue allocations
- Registration limits
- Event coordinators

These datasets form the backbone of the Events module.

c. Registration Forms

Both online and offline registration forms contribute:

- Participant names
- Selected events
- Team details (for group events)
- Registration fee payments
- Registration dates

This information is essential for generating registration statistics and reports.

d. System Logs (Automatic)

The system also collects:

- Login logs
- Session details
- Profile update logs
- Administrative actions

These logs support system security, auditing, and troubleshooting.

4.1.2 Nature of Data Collected

The collected data can be categorized into:

- **Structured Data:** Names, IDs, dates, categories, email, phone, event descriptions
- **Semi-Structured Data:** Feedback messages, user comments, free-text descriptions
- **Temporal Data:** Event timings, registration periods
- **Sensitive Data:** Passwords (encrypted), email IDs **4.1.3 Data Storage During Collection**

Initially, the data is stored in:

- Spreadsheets
- Google Forms
- Word/PDF documents
- Database exports

These raw data files are later imported into the system for cleaning and preprocessing.

4.2 DATASET DESCRIPTION

The dataset used in the Cultural Fest Management System is divided into multiple tables representing different entities. Each table is logically designed to reduce redundancy and maintain relational integrity.

4.2.1 Users Table

Contains details of all types of users:

- Students
- Faculty coordinators

- Admins

Attributes:

- User_ID
- Full_Name
- Email
- Mobile_Number
- Department
- Role (Student/Faculty/Admin)
- Password (encrypted)
- Date_Joined

4.2.2 Events Table

Stores data related to fest events.

Attributes:

- Event_ID
- Event_Name
- Category (Dance, Music, Drama, etc.)
- Event_Type (Solo/Group)
- Description
- Venue
- Event_Date
- Start_Time
- Max_Registrations

  Coordinator_Name
- Registration_Fee

4.2.3 Registrations Table

Tracks every participant who registers for an event.

Attributes:

- Registration_ID
- User_ID
- Event_ID
- Registration_Start
- Registration_End
- Payment_Status
- Team_Name (if applicable)

4.2.4 Schedules Table

Contains details of event timings and venue allocation.

Attributes:

- Schedule_ID
- Event_ID
- Event_Date
- Start_Time
- End_Time
- Venue

4.2.5 Notifications Table

Used to broadcast announcements and updates.

Attributes:

- Notification_ID
- Title

- Message
- Posted_By
- Timestamp

4.2.6 Dataset Size

Depending on the institution, dataset size may include:

- 500–3000 users
- 20–80 events
- 1000+ registrations

4.3 DATA PREPROCESSING

Data preprocessing involves transforming raw data into a clean and usable format. This ensures accuracy, consistency, and integrity once the data is stored in the database.

4.3.1 Data Cleaning

The first step is removing inconsistencies and errors.

a. Removal of Duplicate Records

Duplicate entries in:

- User lists
- Registration forms
- Event details are identified and removed to avoid redundancy.

b. Handling Missing Values

Missing values may occur in fields like:

- Phone numbers
- Event description
- Team names

Missing mandatory values are filled or flagged for correction.

c. Standardizing Formats

- Date format → *dd-mm-yyyy*
- Time format → *24-hour*
- Phone numbers standardized to 10 digits
- Email format validated using regex

4.3.2 Data Transformation

a. Text Normalization

Text values are standardized by:

Converting names to Title Case

- Ensuring category names follow a controlled vocabulary
- Removing extra spaces, symbols, or invalid characters

b. Converting Data Types

Incorrect data types are fixed:

- Numbers stored as text → converted to integers
- Dates stored as text → converted to date objects

c. ID Generation

Unique identifiers such as:

- User_ID
- Event_ID
- Registration_ID are auto-generated using database auto-increment sequences.

d. Password Encryption

Plain-text passwords are hashed using:

- SHA-256
- Bcrypt
- Or Argon2

This ensures secure authentication.

4.3.3 Data Normalization

Normalization ensures efficient storage and retrieval.

a. 1NF (First Normal Form)

- Each attribute holds atomic (single) values
- No repeating groups

b. 2NF

- Partial dependencies removed
- Composite keys separated

c. 3NF

- Transitive dependencies removed
- Separate tables for Users, Events, Registrations

4.3.4 Data Integration

Data from multiple sources is merged into the database.

a. Merging Offline & Online Registration Data

Offline forms are digitized and aligned with online datasets.

b. Linking Events to Coordinators

Coordinator IDs mapped correctly to event tables.

c. Integrating Departmental Data

Multiple department lists consolidated into a standardized structure.

4.4 DATA VALIDATION & QUALITY ASSURANCE

Validation ensures that all data entering the system is accurate and meets predefined rules.

4.4.1 Field-Level Validation

a. Mandatory Fields

Certain fields must not be empty:

- Event Name
- Category
- Event Date
- Coordinator Name
- Registration Start/End dates

b. Format Validation

- Email → must contain "@" & domain
- Date → must follow the system format
- Mobile → exactly 10 digits

c. Range Checks

- Registration fee ≥ 0
- Max registrations > 0
- Event date must be after current date

4.4.2 Referential Integrity Validation

Ensures connections between tables are correct:

- Every registration must map to a valid User_ID

- Event_ID must exist before schedule creation
- Coordinator must exist in Users table

4.4.3 Consistency Checks

a. Schedule Conflicts

- Same venue cannot host two events at the same time

b. Duplicate Registrations

- A student cannot register twice for the same event

c. Event Category Consistency

- "Dance" events must not be assigned an invalid subtype

4.4.4 Final Data Loading

Once cleaned, validated, and processed, the data is loaded into:

- MySQL/PostgreSQL database
- Backups stored for recovery
- Logs maintained for audit trail

This concludes the complete flow from data collection to fully validated system-ready data.

5. SYSTEM MODULE IMPLEMENTATION

5.1 Announcement Module

5.1.1. Introduction

The Announcement module is designed to manage and display important updates within the Cultural Fest Management System. It allows administrators to publish general notices, event-related updates, winner announcements, and urgent alerts. This module ensures that students, participants, and organizers receive timely and accurate information about various activities and changes happening during the fest.

5.1.2. Objectives

- **To provide a centralized platform** for publishing and managing announcements related to the cultural fest.
- **To ensure timely communication** of important updates such as event changes, results, and urgent notifications.
- **To categorize announcements effectively** using types like GENERAL, EVENT_UPDATE, WINNER_ANNOUNCEMENT, and URGENT.
- **To link announcements with specific events**, allowing users to view relevant updates contextually.
- **To maintain history and tracking** through automatic timestamp creation and updates.

5.1.3. Responsibilities

- **Administrator Responsibilities**
    - Create, update, and manage announcements.
    - Assign announcements to specific events when required.
    - Activate or deactivate announcements based on relevance.
    - Ensure accuracy and clarity of the posted information.

- **System Responsibilities**
    - Automatically record creation and update timestamps.
    - Display active announcements to users in proper order.
    - Categorize announcements based on type for better filtering.
    - Maintain data integrity through entity relationships.

5.2 Event module

5.2.1 Introduction

The Event class represents the core data model for managing events in the Cultural Fest Management System. It is implemented as a JPA Entity and mapped to the events table in the database. This model helps store and retrieve event-related information such as event name, category, schedule, venue, participant limits, registration details, and event status.

By using JPA annotations, the class supports automatic ORM (Object-Relational Mapping), making data persistence efficient and reducing manual query handling. The model also includes timestamp tracking and lifecycle callbacks to manage record creation and updates seamlessly.

5.2.2 Objectives

1. To model event data in a structured format

The class defines all essential attributes of an event (name, type, date, time, venue, fees, etc.), ensuring consistent event information across the system.

2. To enable seamless integration with the database using JPA

Through annotations like @Entity, @Table, @Id, and @GeneratedValue, the class supports automatic mapping between Java objects and database tables.

3. To track event lifecycle and status

Fields like status, createdAt, and updatedAt, combined with @PrePersist and @PreUpdate, help in monitoring the event's current state and maintaining audit history.

4. To support event registration control

Properties such as maxParticipants, maxRegistrations, and currentRegistrations allow the system to manage registration limits efficiently.

5. To improve modularity and maintainability

By separating the Event model into its own class, the system follows clean architecture practices, making the codebase easier to extend, test, and maintain.

5.2.3 Responsibilities of the Event Module

The Event module is responsible for handling all operations related to managing events within the Cultural Fest Management System. Its key responsibilities include:

1. Event Data Management

- Store, update, and retrieve all essential event details such as name, category, date, time, venue, type, fees, and descriptions.
- Maintain structured and accurate event information in the database.

2. Registration Handling & Limits

- Track the number of current registrations for each event.
- Enforce participation limits using fields like maxParticipants and maxRegistrations.
- Prevent over-registration by validating availability.

3. Event Status Tracking

- Manage the lifecycle of events through statuses such as **UPCOMING**, **ONGOING**, **COMPLETED**, and **CANCELLED**.
- Automatically update timestamps (createdAt, updatedAt) using lifecycle hooks.

4. Categorization and Classification

- Support different event categories (e.g., Cultural, Sports, Technical).
- Handle event types such as **Solo**, **Group**, or **Team** for better participant management.

5. Media & Presentation

- Store and manage event images for frontend display.
- Mark events as new (isNew) to highlight them on the website.

6. Data Validation & Integrity

- Enforce non-null fields, proper data types, and constraints using JPA annotations.
- Ensure only valid, meaningful event data is persisted.

7. Integration with Other Modules

- Provide consistent data to other modules like Registration, User Dashboard, and Admin Panel.

Act as a central reference for schedule and venue management across the system.

5.3 Judges Module

5.3.1 Introduction

The Judge module is responsible for managing the assignment of judges to events in the Cultural Fest Management System. Each judge is linked to a registered system user and is associated with a specific event. This module ensures proper allocation of judges, tracks their assigned specialization, and maintains record timestamps automatically. By using JPA annotations, it provides robust and maintainable data mapping between the application and database.

5.3.2 Objectives of the Judge Module

1. To map judges with users and events

Ensure proper linking between judges, users, and events using relational mappings (@ManyToOne).

2. To maintain judge specialization data

Store expertise or specialization areas to ensure proper judge allocation based on the type of event.

3. To automate assignment tracking

Use lifecycle methods (@PrePersist) to store the exact time a judge was assigned to an event.

4. To ensure structured and clean data storage

Leverage JPA for consistent and reliable persistence of judge-related information.

5. To support the event evaluation workflow

Enable smooth coordination between judges and event management activities.

5.3.3 Responsibilities of the Judge Module

1. Judge Assignment Management

- Assign judges to specific events.
- Ensure a judge (user) is linked correctly via user_id and event_id.

2. Specialization Handling

- Store and manage judge expertise such as *Dance Judge*, *Singing Judge*, *Technical Judge*, etc.
- Help event organizers choose the right judge for the right event.

3. Timestamp & Audit Tracking

- Automatically capture the time when a judge is assigned (assignedAt).
- Maintain reliable audit logs for event administration.

4. Data Integrity & Relationship Mapping

- Use @ManyToOne relationships to ensure proper mapping between Judge → User and Judge → Event.
- Prevent invalid data by enforcing non-null constraints (nullable = false).

5. Integration with Event Evaluation Workflow

- Provide judge details to scoring modules, event dashboards, and admin panels.

- Support functionalities like viewing assigned judges per event.

5.4 Notification Module

5.4.1 Introduction

The Notification module is responsible for delivering important updates and alerts to users within the Cultural Fest Management System. Each notification can be linked to a specific user and event, allowing the system to send targeted messages such as registration updates, event announcements, and winner notifications.
Using JPA annotations, the module ensures structured data storage, relationship mapping, and automatic timestamp creation. The module also supports additional communication fields like email and mobile, enabling multi-channel notification support.

5.4.2 Objectives of the Notification Module

1. To inform users about important event-related updates

Provide a reliable way to notify users about registrations, event changes, results, and other important announcements.

2. To maintain structured and relational notification data

Store notifications with proper links to users and events using @ManyToOne mappings.

3. To support multiple channels of communication

Handle message delivery through different mediums such as email or mobile (SMS/WhatsApp).

4. To track notification status

Record whether a notification has been read or not, helping improve user engagement and alert visibility.

5. To maintain accurate timestamps
Automatically store notification creation time for auditing and sorting purposes.

5.4.3 Responsibilities of the Notification Module

1. Notification Creation & Storage

- Generate notifications for various activities (registration success, updates, winner announcements).
- Persist notification details in the database with message content and type.

2. Linking Notifications to Users and Events

- Associate notifications with specific users (user_id).
- Connect notifications to related events (event_id) whenever applicable.

3. Multi-Channel Communication Support

- Store user email and mobile details to enable sending:
    - Email notifications
    - SMS/WhatsApp alerts
    - System-based in-app notifications

4. Status Tracking

- Track whether a user has read a notification using isRead.
- Help implement unread notification counts in user dashboards.

5. Notification Type Management

- Categorize notifications as:
    - REGISTRATION_SUCCESS
    - EVENT_UPDATE
    - WINNER_ANNOUNCEMENT
    - GENERAL
- Allow the system to trigger notifications based on these categories.

6. Timestamp & Audit Logging

- Automatically assign a creation time using @PrePersist.
- Maintain historical notification records for administrators.

7. Integration with Other Modules

- Work seamlessly with Registration, Event, Judge, and User modules.
- Provide structured data for frontend UI dashboards (Admin/User).

5.5 Registrations Module

5.5.1 Introduction

The Registration module manages the entire registration process for events within the Cultural Fest Management System. It records which user has registered for which event, along with participant details, registration status, and scores (if applicable).
Using JPA mappings, the module establishes strong relationships with the User and Event entities and ensures automatic tracking of registration timestamps. This module plays a central role in participant management and event workflow coordination.

5.5.2 Objectives of the Registration Module

1. To manage and store participant registrations

Maintain structured records of users registering for events, including participant names and event mapping.

2. To track registration status and participation

Monitor changes in participation status such as REGISTERED, PARTICIPATED, and CANCELLED.

3. To record timestamps automatically

Store the date and time when a registration is created or updated for auditing and sorting.

4. To support scoring and event evaluation

Allow judges or organizers to store participant scores after event participation.

5. To maintain relational integrity

Ensure proper linking between registration, user, and event entities using JPA @ManyToOne relationships.

5.5.3 Responsibilities of the Registration Module

1. Participant Registration Management

- Create new registrations for users enrolling in events.
- Store participant details like participantName.
- Prevent duplicate or invalid registrations (handled at service level).

2. Mapping Users to Events

- Link every registration to:
    - user_id → identifies who registered
    - event_id → identifies which event they registered for

3. Managing Registration Status

- Track and update registration life cycle:
    - **REGISTERED** → user has enrolled
    - **PARTICIPATED** → user attended the event
    - **CANCELLED** → registration withdrawn

4. Score Handling

- Store judging scores for participants (score).
- Support event result generation workflows.

5. Timestamp Tracking

- Automatically store:
    - registeredAt → when the registration was created
    - updatedAt → last modified time

6. Ensuring Data Integrity

- Enforce non-null constraints and relational consistency.
- Maintain clean and reliable registration records for admin and judges.

7. Integration with Other Modules

- Interacts with:
    - **Event module** (registration count, participation tracking)
    - **User module** (user-specific registrations)
    - **Notification module** (send alerts on successful registration, updates)
    - **Judge/Scoring module** (assign scores to participants)

5.6 User model Module

5.6.1. Introduction

The User module is a core component of the Cultural Fest Management System. It manages all user-related information, including students, administrators, and judges. This entity helps authenticate users, manage roles, track user activities, and ensure secure access across the platform. Every user registered in the system is stored with essential details such as name, email, password, role, and timestamps.

5.6.2. Objectives

- To store and manage user information securely, including login credentials and contact details.
- To differentiate system access using predefined roles such as STUDENT, ADMIN, and JUDGE.
- To provide a centralized user identity system for authentication and authorization across the platform.
- To ensure unique identification of users using a primary key and unique email constraint.
- To automatically track user creation and update timestamps using lifecycle callbacks.

5.6.3. Responsibilities

User Module Responsibilities

- Maintain accurate user details such as name, email, password, mobile, and college ID.
- Ensure email uniqueness to prevent duplicate accounts.
- Assign appropriate roles to users for controlled access to features.
- Provide secure handling of passwords and user identity information.
- Track creation and update times for audit purposes.

System Responsibilities

- Auto-generate unique user IDs and timestamps using JPA lifecycle annotations.
- Enforce database constraints like unique, nullable, and role validation.
- Support role-based access control (RBAC) throughout the application.
- Validate user data before saving or updating.

5.7 Winners Module

5.7.1 Introduction

The Winner module handles the management and announcement of winners for each event in the Cultural Fest Management System. It connects winning entries to their respective events and participant registrations while storing important details such as winning position, final score, prize, and announcement time.
Using JPA relationships, the module ensures proper linking between events and participant registrations and maintains reliable winner records for result display and reporting.

5.7.2 Objectives of the Winner Module

1. To record and maintain event winners

Store details of participants who secure first, second, or third positions in an event.

2. To link winners with event and registration details

Associate each winner with both the event and the exact registration record of the winning participant.

3. To support scoring and result evaluation

Store final scores computed by judges or event organizers, ensuring transparent result handling.

4. To automatically track announcement time

Capture when the winner information is officially announced.

5. To facilitate result display and notifications

Provide structured data for publishing results on dashboards and sending winner notifications.

5.7.3 Responsibilities of the Winner Module

1. Winner Result Recording

- Save winner details for each event, including the winning participant and their registration record.
- Ensure that only valid participants (from Registration module) can be marked as winners.

2. Position Assignment

- Store the winning position using the Position enum (FIRST, SECOND, THIRD).
- Prevent assigning duplicate positions for the same event (handled at service level).

3. Score & Prize Handling

- Save the final computed score (finalScore) for transparency and ranking.
- Store prize details associated with each winning position.

4. Timestamp Management

- Automatically record the announcement date and time using @PrePersist.
- Help administrators maintain audit logs of result announcements.

5. Relationship Mapping

- Establish strong relational links:
    - event_id → identifies the event
    - registration_id → identifies the winning participant
- Ensure data integrity in winner records.

6. Integration with Other Modules

- Interacts closely with:
    - Event module for linking results to events
    - Registration module to validate participants
    - Judge/Scoring module to compute final scores
    - Notification module to announce winners

7. Support for Result Publishing

- Provide winner details for:
    - Event results page
    - Certificates and prize reports
    - Admin and participant dashboards

CONCLUSION AND FUTURE SCOPE

The College Cultural Fest Management System is a complete, integrated software solution designed to digitally manage and automate the planning, execution, and monitoring of college-level cultural events. The system effectively replaces traditional manual processes with a structured, database-driven approach that ensures accuracy, efficiency, and transparency across all fest activities.

By implementing clearly defined modules such as User, Event, Registration, Announcement, Judge, Notification, and Winner, the system follows a modular and layered architecture, making it easy to understand, maintain, and extend. Each module has a specific responsibility and interacts seamlessly with other modules, ensuring smooth information flow and reliable system behavior.

The User module ensures secure authentication, role-based access control, and centralized user identity management. The Event module acts as the backbone of the system by maintaining all event-related details, schedules, and participation limits. The Registration module enables participants to enroll in events efficiently while maintaining participation status, scores, and timestamps.

The Announcement and Notification modules significantly improve communication by ensuring timely dissemination of important updates, event changes, and result declarations. The Judge module enables systematic judge allocation and specialization tracking, which enhances fairness and professionalism in event evaluation. Finally, the Winner module ensures accurate result management, transparent ranking, and reliable publishing of event outcomes.

The use of Spring Boot, JPA, and relational database concepts ensures data integrity, automatic timestamp handling, and efficient ORM-based persistence. Lifecycle callbacks reduce manual overhead, while entity relationships enforce consistency across modules. Overall, the system improves administrative efficiency, enhances participant experience, and provides a scalable solution suitable for real-world college cultural fests.

Thus, the College Cultural Fest Management System successfully meets its objectives by offering a reliable, secure, and user-friendly platform that simplifies event management while promoting transparency, accountability, and effective communication.

FUTURE SCOPE

Although the current system fulfills essential cultural fest management requirements, it has significant potential for enhancement and expansion. The following improvements can be considered in future versions:

1. **Online Payment & Financial Management**
    - Integration of secure payment gateways (UPI, Net Banking, Debit/Credit Cards).
    - Automated fee collection, refund processing, and payment status tracking.
    - Financial reports for organizers and college administration.
2. **Mobile Application Development**
    - Native Android and iOS applications for students, judges, and administrators.
    - Push notifications for instant alerts and reminders.
    - Offline access to event schedules and announcements.
3. **Real-Time Judge Scoring System**
    - Live score entry by judges during events.
    - Automatic score aggregation and ranking.
    - Real-time leaderboard display for audiences and participants.
4. **Certificate & Prize Automation**
    - Automatic generation of participation and winner certificates.
    - QR-code-based certificate verification.
    - Digital download and email delivery of certificates.
5. **Smart Attendance & Check-in**
    - QR-code or RFID-based participant check-in.
    - Prevention of proxy attendance.
    - Attendance reports for organizers.

6. Advanced Reporting & Analytics

- Visual dashboards with charts and graphs.
- Event popularity analysis.
- Participation trends across departments and years.
- Judge evaluation statistics.

7. AI-Based Event Recommendations

- Suggest events to students based on interests and past participation.
- Personalized dashboards for users.
- Improved engagement and participation rates.

8. Multi-Fest & Multi-College Support

- Support for organizing multiple cultural fests simultaneously.
- Centralized management for inter-college competitions.
- Role-based access for multiple institutions.

9. Enhanced Security & Performance

- Two-factor authentication (2FA).
- Password encryption and secure token-based login.
- Cloud deployment for scalability and high availability.

10. Social Media & Public Integration

- Automatic sharing of event highlights and results on social platforms.
- Public event pages for promotions and registrations.
- Live streaming integration for major events.

APPENDIX A
SOURCE CODE

Announcement.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "announcements")
public class Announcement {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(columnDefinition = "TEXT", nullable = false)
    private String content;

    @Enumerated(EnumType.STRING)
    private AnnouncementType type;

    @ManyToOne
    @JoinColumn(name = "event_id")
    private Event event;
```

```
private Boolean isActive = true;

@Column(updatable = false)
private LocalDateTime createdAt;

private LocalDateTime updatedAt;

// Constructors
public Announcement() {}

public Announcement(String title, String content, AnnouncementType type) {
    this.title = title;
    this.content = content;
    this.type = type;
}

// Getters and Setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getTitle() { return title; }
public void setTitle(String title) { this.title = title; }

public String getContent() { return content; }
public void setContent(String content) { this.content = content; }

public AnnouncementType getType() { return type; }
public void setType(AnnouncementType type) { this.type = type; }

public Event getEvent() { return event; }
```

```
    public void setEvent(Event event) { this.event = event; }

    public Boolean getIsActive() { return isActive; }
    public void setIsActive(Boolean isActive) { this.isActive = isActive; }

    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }

    public LocalDateTime getUpdatedAt() { return updatedAt; }
    public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }

    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
    }

    public enum AnnouncementType {
        GENERAL, EVENT_UPDATE, WINNER_ANNOUNCEMENT, URGENT
    }
}
```

Event.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;

@Entity
@Table(name = "events")
public class Event {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(nullable = false)
    private String category;

    private String type;  // ← ADDED (e.g., "Solo", "Group", "Team")

    private LocalDate eventDate;

    private LocalTime eventStartTime;
```

```
private String venue;

private Integer maxParticipants;  // ← ADDED

private Integer maxRegistrations;

private Integer currentRegistrations = 0;

private Double registrationFee;  // ← ADDED

@Enumerated(EnumType.STRING)
private EventStatus status = EventStatus.UPCOMING;

private String image;

private Boolean isNew = false;

@Column(updatable = false)
private LocalDateTime createdAt;

private LocalDateTime updatedAt;

// Constructors
public Event() {}

public Event(String name, String category, LocalDate eventDate, String venue) {
    this.name = name;
    this.category = category;
    this.eventDate = eventDate;
```

```
        this.venue = venue;
    }

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public String getCategory() { return category; }
    public void setCategory(String category) { this.category = category; }

    public String getType() { return type; }  // ← ADDED
    public void setType(String type) { this.type = type; }  // ← ADDED

    public LocalDate getEventDate() { return eventDate; }
    public void setEventDate(LocalDate eventDate) { this.eventDate = eventDate; }

    public LocalTime getEventStartTime() { return eventStartTime; }
    public void setEventStartTime(LocalTime eventStartTime) { this.eventStartTime =
eventStartTime; }

    public String getVenue() { return venue; }
    public void setVenue(String venue) { this.venue = venue; }

    public Integer getMaxParticipants() { return maxParticipants; }  // ← ADDED
```

```
    public void setMaxParticipants(Integer maxParticipants) { this.maxParticipants =
maxParticipants; }  // ← ADDED

    public Integer getMaxRegistrations() { return maxRegistrations; }
    public void setMaxRegistrations(Integer maxRegistrations) { this.maxRegistrations =
maxRegistrations; }

    public Integer getCurrentRegistrations() { return currentRegistrations; }
    public void setCurrentRegistrations(Integer currentRegistrations) {
this.currentRegistrations = currentRegistrations; }

    public Double getRegistrationFee() { return registrationFee; }  // ← ADDED
    public void setRegistrationFee(Double registrationFee) { this.registrationFee =
registrationFee; }  // ← ADDED

    public EventStatus getStatus() { return status; }
    public void setStatus(EventStatus status) { this.status = status; }

    public String getImage() { return image; }
    public void setImage(String image) { this.image = image; }

    public Boolean getIsNew() { return isNew; }
    public void setIsNew(Boolean isNew) { this.isNew = isNew; }

    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }

    public LocalDateTime getUpdatedAt() { return updatedAt; }
    public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }

    @PrePersist
```

```
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
    }

    public enum EventStatus {
        UPCOMING, ONGOING, COMPLETED, CANCELLED
    }
}
```

Judege.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "judges")

public class Judge {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
@ManyToOne
@JoinColumn(name = "user_id", nullable = false)
private User user;

@ManyToOne
@JoinColumn(name = "event_id", nullable = false)
private Event event;

private String specialization;  // ← ADDED

@Column(updatable = false)
private LocalDateTime assignedAt;

// Constructors
public Judge() {}

public Judge(User user, Event event) {
    this.user = user;
    this.event = event;
}

// Getters and Setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public User getUser() { return user; }
public void setUser(User user) { this.user = user; }

public Event getEvent() { return event; }
public void setEvent(Event event) { this.event = event; }
```

```
    public String getSpecialization() { return specialization; }  // ← ADDED
    public void setSpecialization(String specialization) { this.specialization = specialization;
}  // ← ADDED

    public LocalDateTime getAssignedAt() { return assignedAt; }
    public void setAssignedAt(LocalDateTime assignedAt) { this.assignedAt = assignedAt; }

    @PrePersist
    protected void onCreate() {
        assignedAt = LocalDateTime.now();
    }
}
```

Notification.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "notifications")
public class Notification {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
```

```
@ManyToOne
@JoinColumn(name = "event_id")
private Event event;

@Column(nullable = false)
private String message;

private String email;  // ← ADDED

private String mobile;  // ← ADDED

@Enumerated(EnumType.STRING)
private NotificationType type;

private Boolean isRead = false;

@Column(updatable = false)
private LocalDateTime createdAt;

// Constructors
public Notification() {}

public Notification(User user, String message, NotificationType type) {
    this.user = user;
    this.message = message;
    this.type = type;
}

// Getters and Setters
```

```
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public User getUser() { return user; }
public void setUser(User user) { this.user = user; }

public Event getEvent() { return event; }
public void setEvent(Event event) { this.event = event; }

public String getMessage() { return message; }
public void setMessage(String message) { this.message = message; }

public String getEmail() { return email; }  // ← ADDED
public void setEmail(String email) { this.email = email; }  // ← ADDED

public String getMobile() { return mobile; }  // ← ADDED
public void setMobile(String mobile) { this.mobile = mobile; }  // ← ADDED

public NotificationType getType() { return type; }
public void setType(NotificationType type) { this.type = type; }

public Boolean getIsRead() { return isRead; }
public void setIsRead(Boolean isRead) { this.isRead = isRead; }

public LocalDateTime getCreatedAt() { return createdAt; }
public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }

@PrePersist
protected void onCreate() {
    createdAt = LocalDateTime.now();
```

```
    }

    public enum NotificationType {
        REGISTRATION_SUCCESS, EVENT_UPDATE, WINNER_ANNOUNCEMENT,
GENERAL
    }
}
```

Registration.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "registrations")
public class Registration {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne
    @JoinColumn(name = "event_id", nullable = false)
    private Event event;

    private String participantName;  // ← ADDED
```

```
@Enumerated(EnumType.STRING)
private RegistrationStatus status = RegistrationStatus.REGISTERED;

private Double score;

@Column(updatable = false)
private LocalDateTime registeredAt;

private LocalDateTime updatedAt;

// Constructors
public Registration() {}

public Registration(User user, Event event) {
    this.user = user;
    this.event = event;
}

// Getters and Setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public User getUser() { return user; }
public void setUser(User user) { this.user = user; }

public Event getEvent() { return event; }
public void setEvent(Event event) { this.event = event; }

public String getParticipantName() { return participantName; }  // ← ADDED
```

```
    public void setParticipantName(String participantName) { this.participantName =
participantName; }  // ← ADDED

    public RegistrationStatus getStatus() { return status; }
    public void setStatus(RegistrationStatus status) { this.status = status; }

    public Double getScore() { return score; }
    public void setScore(Double score) { this.score = score; }

    public LocalDateTime getRegisteredAt() { return registeredAt; }
    public void setRegisteredAt(LocalDateTime registeredAt) { this.registeredAt =
registeredAt; }

    public LocalDateTime getUpdatedAt() { return updatedAt; }
    public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }

    @PrePersist
    protected void onCreate() {
        registeredAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
    }

    public enum RegistrationStatus {
        REGISTERED, PARTICIPATED, CANCELLED
    }
}
```

User.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;

import java.time.LocalDateTime;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;
```

```
 private String mobile;

private String collegeId;


 @Enumerated(EnumType.STRING)

 @Column(name = "role", nullable = false)

 private Role role;


 @Column(updatable = false)

 private LocalDateTime createdAt;


 private LocalDateTime updatedAt;


 // Constructors

 public User() {}


 public User(String name, String email, String password, Role role) {

    this.name = name;

    this.email = email;

    this.password = password;

    this.role = role;

 }


 // Getters and Setters
```

```
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getName() { return name; }
public void setName(String name) { this.name = name; }

public String getEmail() { return email; }
public void setEmail(String email) { this.email = email; }

public String getPassword() { return password; }
public void setPassword(String password) { this.password = password; }
public String getMobile() { return mobile; }
public void setMobile(String mobile) { this.mobile = mobile; }
public String getCollegeId() { return collegeId; }
public void setCollegeId(String collegeId) { this.collegeId = collegeId; }
public Role getRole() { return role; }
public void setRole(Role role) { this.role = role; }
public LocalDateTime getCreatedAt() { return createdAt; }
public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }

public LocalDateTime getUpdatedAt() { return updatedAt; }
public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }

@PrePersist
```

```
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
    }

    public enum Role {
        STUDENT, ADMIN, JUDGE
    }
}
```

Winner.java code:

```
package com.culturalfest.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "winners")
public class Winner {
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@ManyToOne
@JoinColumn(name = "event_id", nullable = false)
private Event event;

@ManyToOne
@JoinColumn(name = "registration_id", nullable = false)
private Registration registration;

@Enumerated(EnumType.STRING)
@Column(nullable = false)
private Position position;

private Double finalScore;  // ← ADDED

private String prize;

@Column(updatable = false)
private LocalDateTime announcedAt;

// Constructors
```

```
    public Winner() {}

    public Winner(Event event, Registration registration, Position position) {
        this.event = event;
        this.registration = registration;
        this.position = position;
    }

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public Event getEvent() { return event; }
    public void setEvent(Event event) { this.event = event; }

    public Registration getRegistration() { return registration; }
    public void setRegistration(Registration registration) { this.registration = registration; }

    public Position getPosition() { return position; }
    public void setPosition(Position position) { this.position = position; }

    public Double getFinalScore() { return finalScore; }  // ← ADDED
    public void setFinalScore(Double finalScore) { this.finalScore = finalScore; }  // ← ADDED
```

```
    public String getPrize() { return prize; }
    public void setPrize(String prize) { this.prize = prize; }

    public LocalDateTime getAnnouncedAt() { return announcedAt; }
    public void setAnnouncedAt(LocalDateTime announcedAt) { this.announcedAt =
announcedAt; }

    @PrePersist
    protected void onCreate() {
        announcedAt = LocalDateTime.now();
    }

    public enum Position {
        FIRST, SECOND, THIRD
    }
}
```

APPENDIX B

SCREENSHOTS

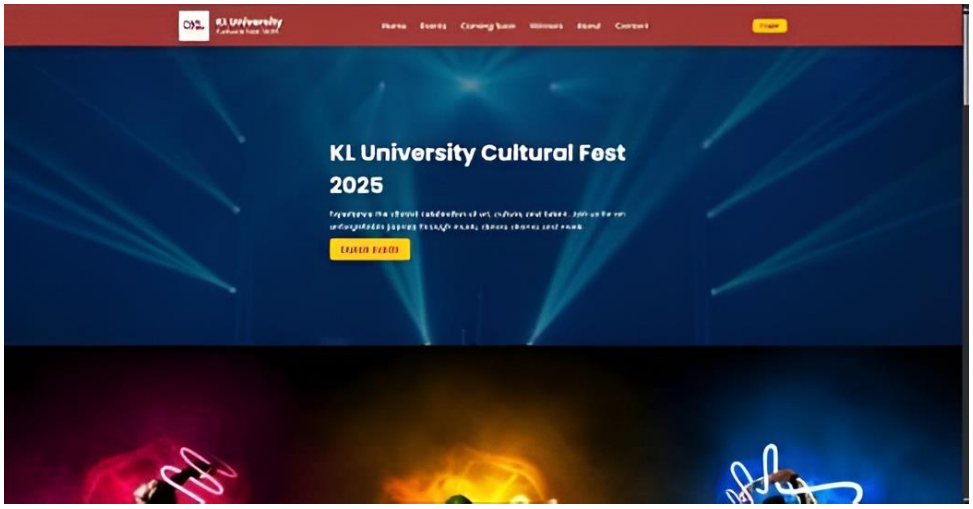


Fig B.1Website home page

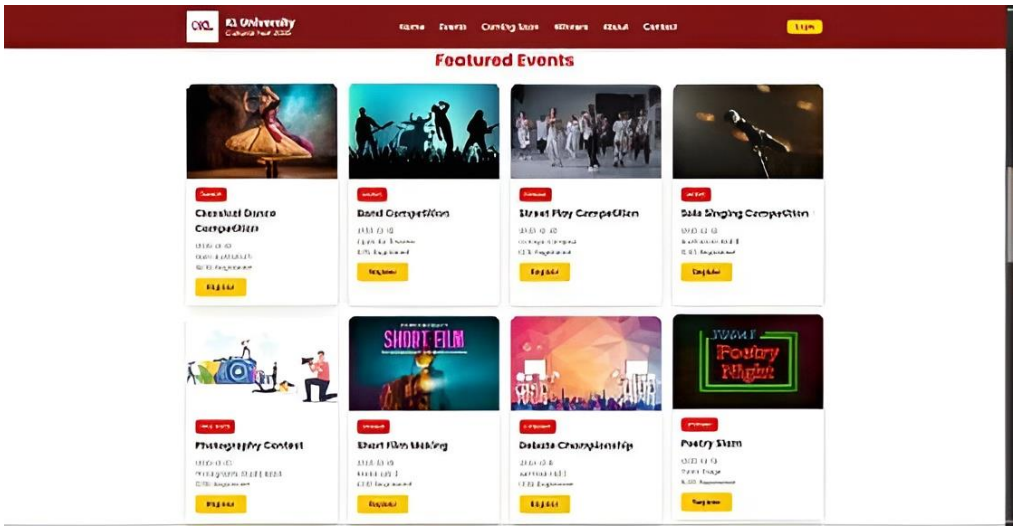


Fig B.2 Featured Events

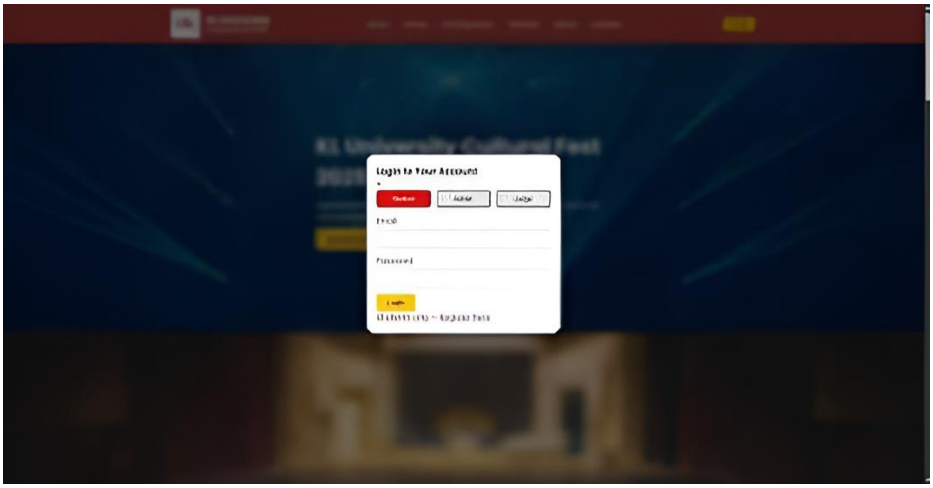


Fig B.3 Login page

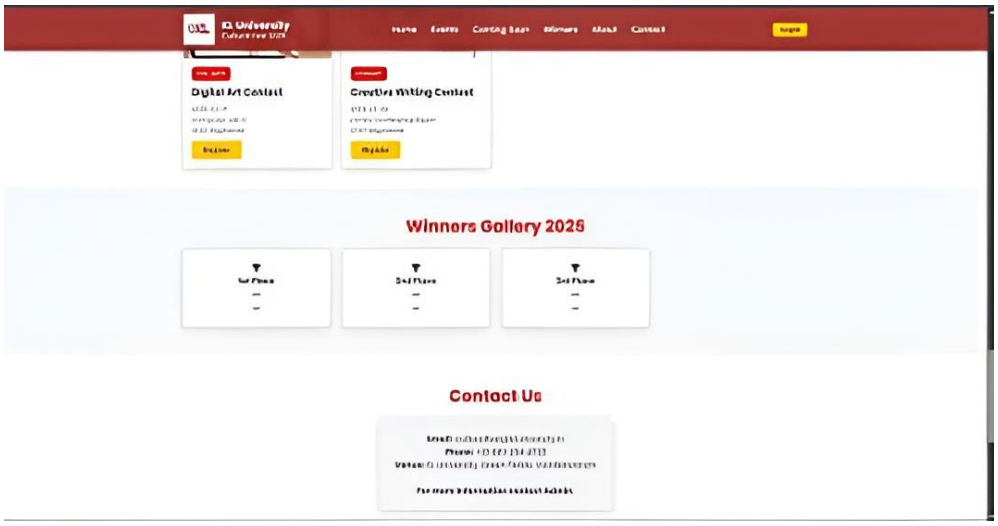


Fig B.4 Winners Dashboard

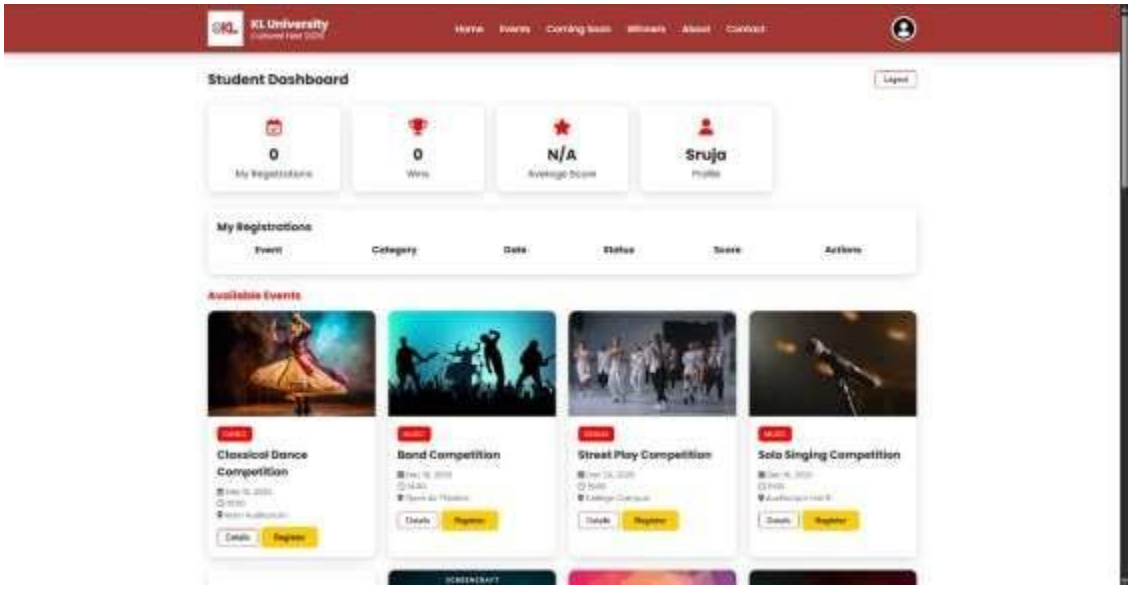


Fig B.5 Available Events

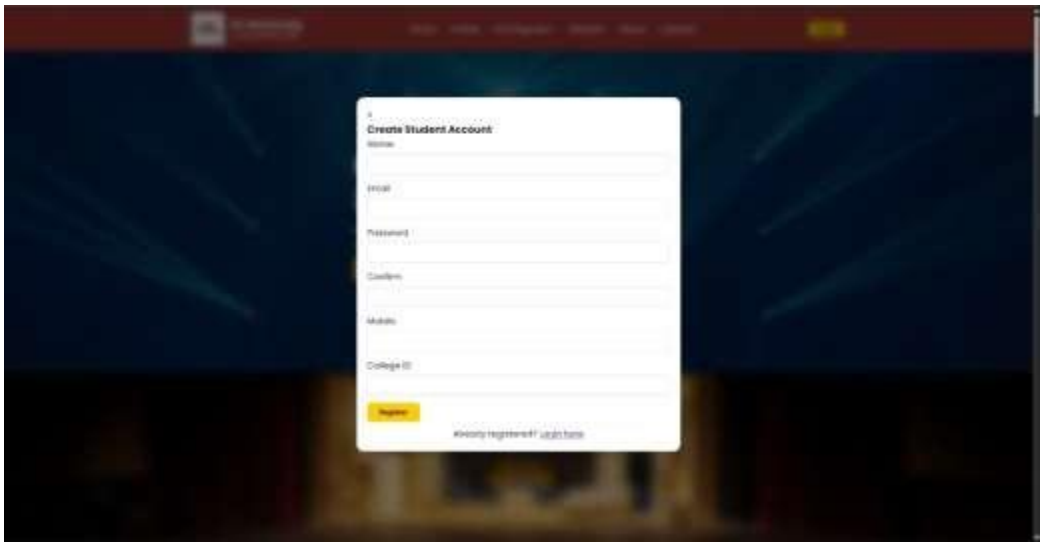


Fig B.6 student dashboard

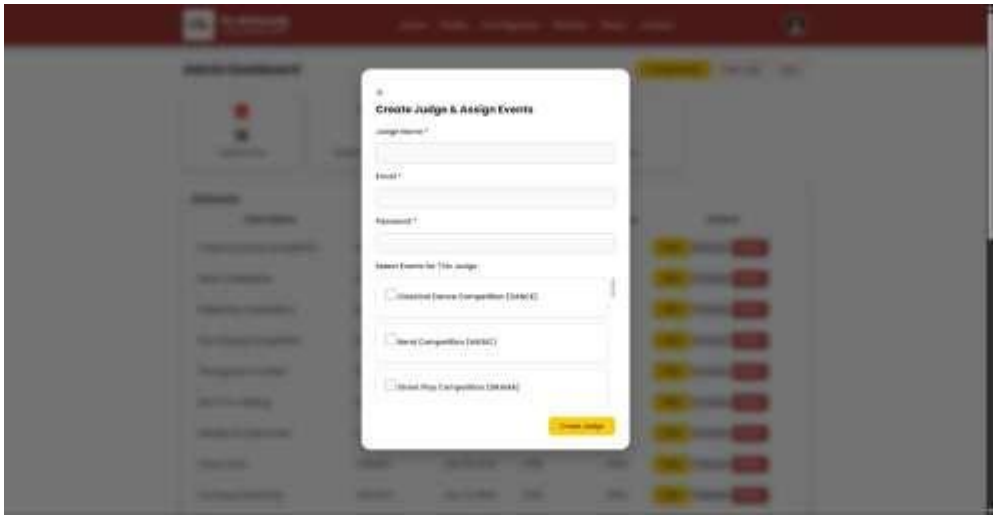


Fig B.7 Judges Dashboard

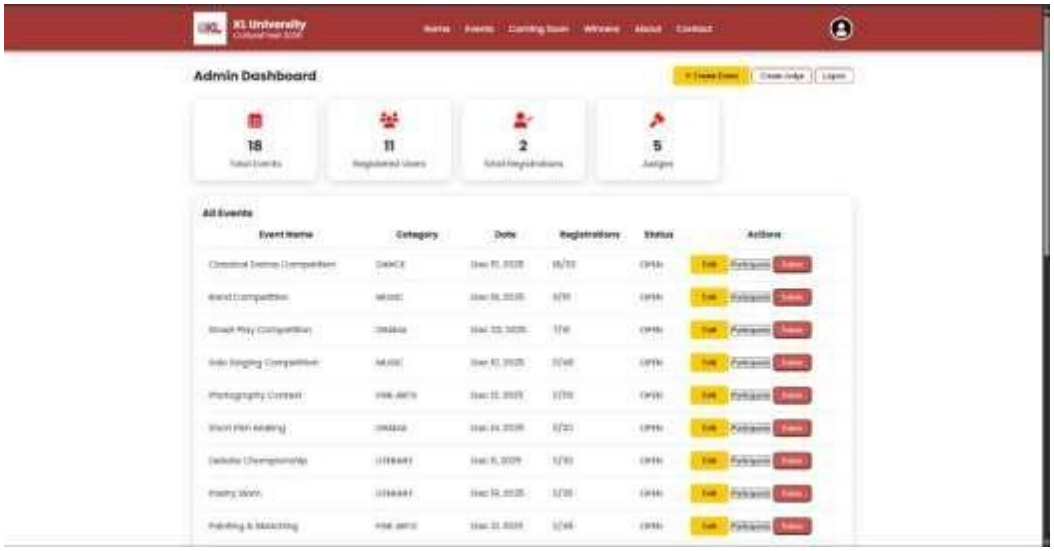


Fig B.8 Admin Dashboard

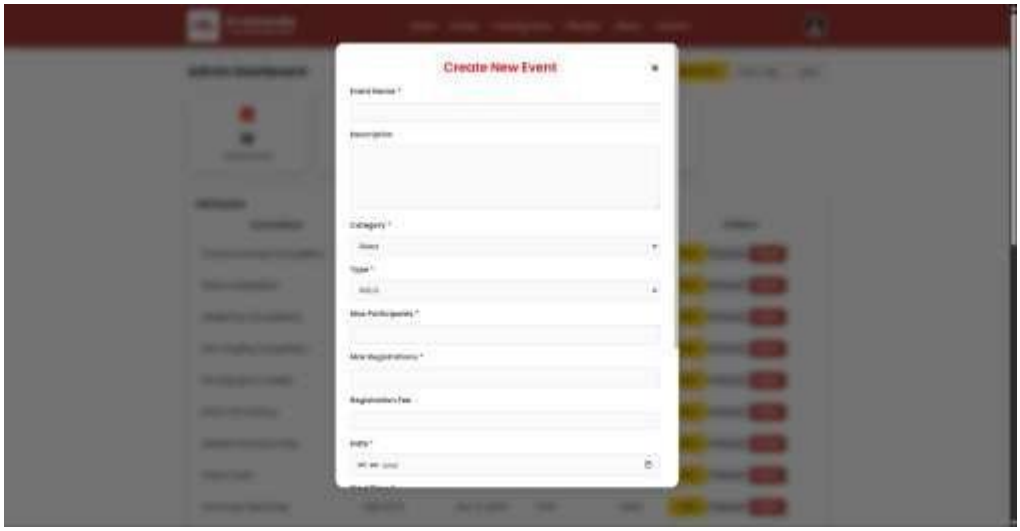


Fig B.9 Creating an event

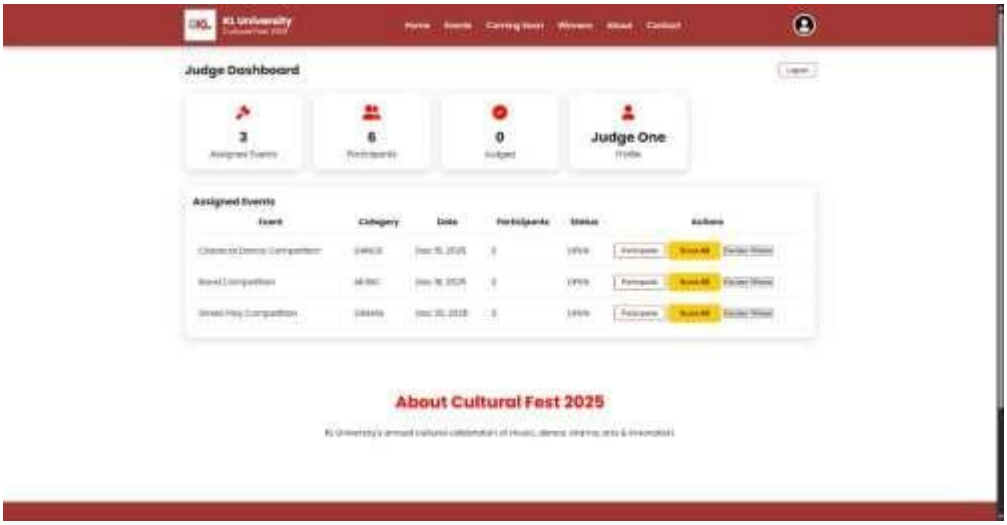


Fig B.10 Judges Dashboard

USER (PARTICIPANT) FUNCTIONALITY TEST CASES					
S.No	Feature Name	Test Case Description	Expected Result	Steps to Execute	Pre-requisites
1	User_Registration	To verify user can register successfully with valid details	User registration should be successful and confirmation message should be displayed	1. Open application 2. Enter valid participant name name 4. Select event 5. Enter valid contact number 6. Enter valid email 7. Click Submit	1. Application accessible
2	User_Registration	To verify registration fails when mandatory fields are empty (Negative)	Validation error should be displayed and registration should not proceed	1. Open application 2. Leave mandatory fields empty 3. Click Submit	1. Application accessible
3	User_Registration	To verify registration fails with invalid email format (Negative)	Error message should be shown for invalid email	1. Enter invalid email format 2. Click Submit	1. Application accessible
4	User_Registration	To verify registration fails with invalid contact number (Negative)	Error message should be displayed	1. Enter non-numeric or short contact number 2. Click Submit	1. Application accessible
5	User_Registration	To verify duplicate submission behavior (Negative)	System should prevent duplicate submission or overwrite safely	1. Submit same details twic	1. Valid user data
6	User_Reset	To verify user can reset registration form	All entered values should be cleared	1. Enter data in all fields 2. Click Reset	1. Application accessible
7	User_Event_Selection	To verify user can select only one event	Only one event should be selected at a time	1. Open event dropdown 2. Select an event	1. Application accessible
8	User_Event_Selection	To verify form submission fails without event selection (Negative)	Validation error should be displayed	1. Do not select event 2. Click Submit	1. Application accessible
9	User_Access_Control	To verify user cannot access judge or admin functionalities (Negative)	Access should be restricted	1. Attempt to access judge/	1. Logged in as user

Fig B.11 User Test case

JUDGE FUNCTIONALITY TEST CASES					
S.No	Feature Name	Test Case Description	Expected Result	Steps to Execute	Pre-requisites
10	Judge_Login	To verify judge can access judge panel successfully	Judge dashboard should load	1. Login as judge	1. Judge credentials available
11	Judge_Login	To verify invalid judge login is rejected (Negative)	Error message should be displayed	1. Enter invalid judge creder	1. Judge login screen
12	Judge_View_Participants	To verify judge can view assigned participants	Participant list should be displayed	1. Login as judge 2. Navigate to participants list	1. Judge logged in
13	Judge_View_Participants	To verify judge cannot see participants of other events (Negative)	Access should be restricted	1. Attempt to view other eve	1. Judge logged in
14	Judge_Scoring	To verify judge can assign valid scores	Score should be saved successfully	1. Select participant 2. Enter valid score 3. Submit score	1. Judge logged in
15	Judge_Scoring	To verify judge cannot submit invalid scores (Negative)	Validation error should be displayed	1. Enter score outside allow	1. Judge logged in
16	Judge_Scoring	To verify judge cannot submit score without selecting participant (Negative)	Submission should be blocked	1. Attempt scoring without p	1. Judge logged in
17	Judge_Scoring	To verify judge cannot modify score after submission (Negative)	Score should be locked	1. Submit score 2. Attempt edit	1. Score submitted
18	Judge_Access_Control	To verify judge cannot access admin functions (Negative)	Access denied	1. Attempt admin actions	1. Judge logged in

Fig B.12 Judges Test Case

ADMIN FUNCTIONALITY TEST CASES					
S.No	Feature Name	Test Case Description	Expected Result	Steps to Execute	Pre-requisites
19	Admin_Login	To verify admin can login successfully	Admin dashboard should load	1. Login using admin creden	1. Admin credentials available
20	Admin_Login	To verify admin login fails with invalid credentials (Negative)	Error message should be displayed	1. Enter invalid credentials	1. Admin login screen
21	Admin_Event_Management	To verify admin can create new event	Event should be created successfully	1. Login as admin 2. Create event	1. Admin logged in
22	Admin_Event_Management	To verify admin can edit event details	Event details should be updated	1. Select event 2. Edit details 3. Save	1. Admin logged in
23	Admin_Event_Management	To verify admin can delete event	Event should be removed	1. Select event 2. Delete	1. Admin logged in
24	Admin_Event_Management	To verify admin cannot delete event with participants (Negative)	Error message should be displayed	1. Attempt delete with activ	1. Admin logged in
25	Admin_Participant_Management	To verify admin can view all participants	Complete participant list should be displayed	1. Navigate to participants l	1. Admin logged in
26	Admin_Participant_Management	To verify admin can remove participant	Participant should be removed	1. Select participant 2. Remove	1. Admin logged in
27	Admin_Judge_Assignment	To verify admin can assign judges to events	Judge should be assigned successfully	1. Select event 2. Assign judge	1. Admin logged in
28	Admin_Judge_Assignment	To verify admin cannot assign same judge twice (Negative)	Error should be displayed	1. Assign same judge again	1. Admin logged in
29	Admin_Results	To verify admin can view final scores	Results should be displayed correctly	1. Navigate to results sectio	1. Admin logged in
30	Admin_Results	To verify admin can publish results	Results should be published	1. Click Publish Results	1. Scores finalized

Fig B.13 Admin test case

SYSTEM & SECURITY TEST CASES					
S.No	Feature Name	Test Case Description	Expected Result	Steps to Execute	Pre-requisites
31	Role_Access_Control	To verify users cannot access judge/admin pages (Negative)	Access denied	1. Attempt restricted URL	1. User logged in
32	Role_Access_Control	To verify judges cannot access admin pages (Negative)	Access denied	1. Attempt admin URL	1. Judge logged in
33	Session_Handling	To verify session expires correctly	User should be logged out	1. Stay idle 2. Perform action	1. Logged in
34	Data_Integrity	To verify submitted data is stored correctly	Data should persist accurately	1. Submit data 2. Verify stored values	1. Admin access
35	Error_Handling	To verify system does not crash on invalid actions	System should handle errors gracefully	1. Perform invalid operation	1. Application running

Fig B.14 System and security test case

END-TO-END FLOW					
S.No	Feature Name	Test Case Description	Expected Result	Steps to Execute	Pre-requisites
36	End_To_End	To verify complete fest lifecycle from registration to result declaration	System should function correctly end-to-end	1. User registers 2. Admin assigns judges 3. Judge scores 4. Admin publishes results	1. All roles available

Fig B.15 End-To-End flow

REFERENCES

[1] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach, 9th ed., McGraw-Hill Education, 2019.

[2] I. Sommerville, Software Engineering, 10th ed., Pearson Education, 2016.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, 2nd ed., Addison-Wesley, 2005.

[4] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.

[5] Oracle Corporation, Java Platform, Standard Edition Documentation, Oracle, Available online.

[6] PostgreSQL Global Development Group, PostgreSQL Database Documentation, PostgreSQL, Available online.

[7] World Wide Web Consortium (W3C), HTML5 Specification, W3C, Available online.

[8] World Wide Web Consortium (W3C), CSS3 Specification, W3C, Available online.

[9] Mozilla Developer Network, JavaScript Documentation, Mozilla, Available online.

[10] Spring Framework, Spring Boot and REST API Documentation, Available online.

[11] ISO/IEC 25010, Systems and Software Quality Models, ISO, 2011.

[12] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 6th ed., McGraw-Hill Education, 2011.

[13] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2002.

[14] GitHub, “Open-source event and fest management system projects,” GitHub Repository, Available online.

[15] OWASP Foundation, OWASP Top 10 Web Application Security Risks, Available online.