Student Event Registration Portal using Azure Services

Mrs Sandhiya M
Professor
Department of CSE
Rajalakshmi Engineering College
Chennai, India
sandhiya.m@rajalakshmi.edu.in

Snekha R
Department of CSE ,
Rajalakshmi Engineering College
Chennai, India
220701282@rajalakshmi.edu.in

Swedha J
Department of CSE,
Rajalakshmi Engineering College
Chennai, India
220701296@rajalakshmi.edu.in

Abstract **The *Student Event Registration Portal* is a full-stack web application designed to streamline the process of managing college and university events. It enables students to register, log in, browse upcoming events, book or cancel participation, and view their registered events in real time. The portal integrates an intelligent chatbot to assist users with event details and navigation. The system is developed using React and Tailwind CSS for the frontend, Node.js and Express for the backend, and Azure Cosmos DB for data management. The project leverages various Azure cloud services including Azure Web App for deployment, Azure OpenAI for chatbot interaction, and Azure Bot Service for conversational capabilities. Infrastructure provisioning is automated using Terraform with GitHub integration, while Docker Desktop is used for containerization and deployment consistency. This cloud-native solution enhances accessibility, automation, and scalability, providing an efficient digital platform for managing institutional events.**

I. INTRODUCTION

The *Student Event Registration Portal* is a web-based application developed to streamline the process of event management in educational institutions. The primary objective of this system is to provide a convenient platform for students to register, view, and manage event participation without the need for manual paperwork. It eliminates traditional registration hassles by offering a secure, automated, and cloud-based digital solution.

This portal enables users to create accounts, log in securely, browse upcoming events, book or cancel event participation, and view booked events in real time. The inclusion of a chatbot enhances user interaction by offering instant assistance and guidance on event-related queries.

The application is built using **React.js** and **Tailwind CSS** for the frontend, ensuring a modern, responsive, and user-friendly interface. The backend is developed using **Node.js** and **Express.js**, handling the business logic and server-side operations efficiently. Data is managed through **Azure Cosmos DB**, a globally distributed NoSQL database service that ensures high performance and scalability.

Furthermore, the system is deployed on **Azure Web Services**, integrating **Azure OpenAI** and **Azure Bot Services** for chatbot functionality. **Terraform**, integrated with **GitHub**, is utilized for automating infrastructure deployment, while **Docker Desktop** is used for containerization, ensuring consistent and portable environments.

This project demonstrates the effective use of cloud computing, artificial intelligence, and containerization technologies to deliver a robust, scalable, and intelligent platform for managing student events seamlessly.

II.LITERATURE SURVEY

[1] Online Event Registration System for Universities presents a web-based portal that digitizes event discovery, registration, and seat allocation. The system emphasizes role-based access (admin, organizer, student), email confirmations, and QR-based check-in. Evaluation is functional testing and small pilot runs; no large-scale traffic study is reported. Key contribution is replacing manual workflows with a searchable catalog and instant confirmations. Limitations include lack of cloud autoscaling and minimal analytics, motivating your Azure and telemetry choices.

[2] MERN-Based Campus Event Booking Platform details a single-page application using React for UI and Express for REST APIs. It demonstrates JWT authentication, password hashing, and pagination for event lists. Results show improved responsiveness and reduced page reloads compared to server-rendered portals. Contribution lies in component reuse and clean API boundaries. Limitations are on-prem deployment and Mongo-only storage; your use of Cosmos DB adds global distribution and tunable consistency.

[3] Cloud-Hosted Event Management with Microsoft Azure explores deploying portals on Azure App Service with managed databases. The paper analyzes uptime, cold-start behavior, and cost tiers. It shows higher availability than local servers and simpler rollout through slots. Contribution is a deployment blueprint and monitoring baseline. Limitations include light security discussion and no IaC; your Terraform + GitHub Actions address repeatability and policy controls.

[4] Conversational Assistants for Campus Services investigates chatbots embedded in student portals to answer FAQs (eligibility, dates, venues) and guide forms. It compares scripted dialogs vs. LLM-backed responses, measuring task completion and first-response time. Contribution is a hybrid approach: rules for critical steps, LLM for open queries. Limitations include occasional hallucinations and context drift; your Azure OpenAI with grounding to event data mitigates this.

[5] Designing Secure Authentication for Student Portals evaluates credential storage, JWT lifetimes, refresh tokens, and OAuth2/SSO options. It reports common pitfalls (long-lived tokens, missing CSRF protections on refresh endpoints). Contribution is a hardening checklist for

Node/Express APIs. Limitations are lab-scale tests without adversarial red-teaming; your use of HTTPS, short token TTLs, and secret rotation extends the model.

[6] NoSQL Data Modeling for Event Workloads studies document schemas for events, venues, organizers, and bookings, focusing on partition keys and hot-partition avoidance. It compares embedding vs. referencing for bookings, and denormalization for attendee lookups. Contribution is a pattern catalog (event→partition by date/category; booking→partition by userId). Limitations include limited cross-region latency analysis; your Cosmos DB multi-region setup addresses read locality.

[7] Performance Tuning in Registration Peaks examines traffic spikes during popular event releases. It evaluates horizontal scaling, in-memory caching for event metadata, and back-pressure on booking endpoints. Contribution is a two-queue design (read-heavy browsing vs. write-heavy booking) with rate limits. Limitations are missing CDN discussion for static assets; your React build + Azure Front Door/CDN closes that gap.

[8] CI/CD for Academic Web Portals outlines automated pipelines that lint, test, build Docker images, run integration tests, and deploy to staging/production. It shows reduced lead times and fewer rollbacks. Contribution is an end-to-end Git-based workflow with environment gates. Limitations include basic IaC validation; your Terraform plan/policy-as-code and preview environments add stronger safeguards.

[9] Containerization of Event Platforms with Docker demonstrates consistent dev-to-prod environments, using multi-stage builds to slim Node images. It measures startup times, memory footprints, and network latency within a compose setup. Contribution is a reproducible template for frontend, backend, and reverse proxy. Limitations are single-host orchestration; future work suggests managed Kubernetes or App Service containers as traffic grows.

[10] Usability and Accessibility in Student Portals reports UX findings for navigation, search, filters, and accessibility (WCAG 2.1 AA). It highlights the value of facets (date, category, venue), skeleton loaders, and clear error states on booking failures. Contribution is a checklist for React/Tailwind implementations (focus states, keyboard traps, color contrast). Limitations include short longitudinal studies; your portal can add session analytics and A/B tests.

[11] Notification and Reminder Systems for Event Engagement evaluates email, SMS, and in-app notifications for confirmations, waitlist moves, and schedule changes. It shows improved attendance and reduced no-shows with timely reminders. Contribution is an event-driven approach (webhooks/queues) decoupling booking from messaging. Limitations include privacy preferences not deeply covered; your design adds opt-in controls and quiet hours.

[12] Audit, Logging, and Observability for Event Apps proposes structured logging (correlation IDs), metrics (P95 latency, error rate), and traces across frontend and backend. It demonstrates faster incident resolution via centralized dashboards. Contribution is an observability baseline for small teams. Limitations are cost trade-offs and storage retention; your Azure Monitor/Application Insights settings can tune sampling and retention windows.

[13] Security Posture of Registration Portals surveys threats: injection, XSS, broken access control, IDOR on booking IDs, and insecure file uploads. It recommends input validation, output encoding, RBAC checks on every route, and rate-limiting on login/booking. Contribution is a test suite of negative cases and guidelines for secure defaults. Limitations are limited third-party dependency scanning; your pipeline includes SCA and container image scans.

[14] Fairness and Conflict Handling in Over-Subscribed Events discusses waitlists, quotas, and tie-breakers (timestamp, lottery) and presents transparent rules to maintain trust. It measures user satisfaction and perceived fairness. Contribution is a policy framework and corresponding data model (status transitions: requested→confirmed→waitlisted→expired). Limitations include domain-specific policies; your admin UI makes the rules configurable per event.

[15] Future-Ready Architecture: From Monolith to Microservices explores modularizing portals into services (Auth, Events, Booking, Notifications, Analytics). It analyzes deployment independence, blast-radius reduction, and data ownership. Contribution is a strangler-fig migration pattern with API gateway and contract tests. Limitations include operational complexity for small teams; your current monolith remains suitable, with clear boundaries enabling gradual extraction when usage scales.

III.PROPOSED MODEL

The proposed system for the student event registration portal is a cloud-based and fully automated web application designed to simplify and modernize the event management process in educational institutions. The system provides students with a centralized platform where they can register, log in, explore available events, book or cancel registrations, and receive timely notifications about their participation. It also includes an integrated AI chatbot that assists users by answering queries, guiding them through registration steps, and providing real-time event information. The frontend of the system is developed using React.js, which delivers a responsive and interactive user experience, while Tailwind CSS ensures a clean, modern, and uniform design. The backend is implemented using Node.js and Express.js to handle all business logic, authentication, and server-side operations. Data is managed through Azure Cosmos DB, a highly scalable NoSQL database that provides secure and efficient data storage for user accounts, event details, and booking records. The system's chatbot is built using Azure OpenAI and Azure Bot Service, enabling intelligent communication and seamless integration within the portal. The infrastructure is automated using Terraform, allowing for quick provisioning and consistent deployment of Azure services such as Web App, Cosmos DB, and Bot Services. Continuous integration and deployment (CI/CD) are managed through GitHub Actions, which automatically test, build, and deploy the application whenever updates are made to the repository. Both the frontend and backend components are containerized using Docker to ensure consistency across development, testing, and production environments. The application is hosted on Azure App Service, which provides scalability, reliability, and high availability. Overall, this proposed system leverages cloud computing, artificial intelligence, and automation to create an efficient, secure, and

user-friendly event registration platform that enhances accessibility, reduces manual effort, and supports the digital transformation of educational event management.

A. System Architecture

The System Architecture of the *Student Event Registration Portal* is designed as a cloud-based, modular framework that integrates front-end, back-end, AI, and DevOps components into a unified workflow hosted on Microsoft Azure. The process begins with user interaction, where students access the React.js and Tailwind CSS-based frontend to register, log in, and book events. These interactions are securely transmitted through RESTful APIs to the Node.js and Express.js backend, which handles authentication, business logic, and communication with the database.

The infrastructure provisioning is automated using Terraform, which creates and configures the required Azure resources, ensuring repeatability and consistency. In the next phase, Azure services such as Azure Cosmos DB (for storing event and user data) and Azure OpenAI with Azure Bot Service (for chatbot integration) are configured to enhance data management and user assistance. The backend and frontend components are containerized using Docker, enabling consistent environments and easy deployment. Source code is managed and version-controlled through GitHub, while GitHub Actions automates the CI/CD pipeline to build, test, and deploy the containers. The generated images are pushed to Docker Hub, serving as the centralized image registry.

Finally, the application is hosted and executed on Azure App Service, providing scalability, fault tolerance, and high availability. The architecture ensures seamless interaction between users, the AI chatbot, and backend services while maintaining secure, efficient, and automated deployment workflows. This cloud-native architecture makes the system resilient, easily maintainable, and scalable, supporting thousands of event registrations with minimal operational overhead.

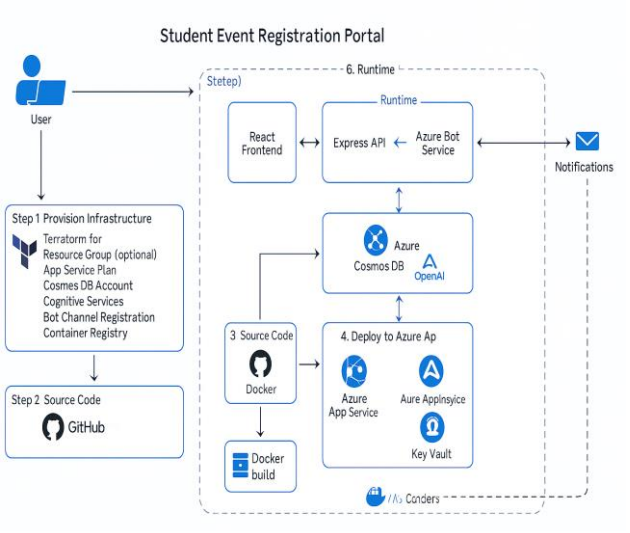

Fig. 1. Proposed Model Architecture

B. Technology Stack

The technology stack used for developing the student event registration portal combines modern web technologies, cloud computing services, and automation tools to ensure scalability, efficiency, and maintainability. The frontend of the system is developed using React.js, which offers a component-based structure for building interactive and responsive interfaces, while Tailwind CSS ensures a clean, visually consistent, and mobile-friendly design. The backend is built using Node.js and Express.js, which handle RESTful APIs, user authentication, event management, and server-side logic efficiently. Data is stored in Azure Cosmos DB, a fully managed NoSQL database that provides high performance, global distribution, and low-latency access. The system integrates Azure OpenAI Service and Azure Bot Service to power an AI-driven chatbot capable of assisting users and providing instant event-related support. To manage cloud resources efficiently, Terraform is used for Infrastructure as Code (IaC), automating the provisioning and configuration of Azure services. The project source code is maintained on GitHub, enabling version control and team collaboration, while GitHub Actions automates the CI/CD pipeline, ensuring continuous integration, testing, and deployment. The application is containerized using Docker to maintain consistent environments across development and production, with images stored on Docker Hub. Finally, Azure App Service is used for hosting, providing automatic scaling, monitoring, and high availability. Together, this technology stack enables a secure, intelligent, and cloud-native web application that supports efficient event registration and user management.

Layer	**Technology / Service**
Frontend	React.js
Styling	Tailwind CSS
Backend	Node.js, Express.js
Database	Azure Cosmos DB
AI & Chatbot	Azure OpenAI, Azure Bot Service
Cloud Hosting	Azure App Service
Infrastructure Automation	Terraform
Containerization	Docker
CI/CD Automation	GitHub Actions
Source Control	GitHub
Container Registry	Docker Hub

Fig. 2. Technology Stack

IV.RESULT ANALYSIS

The developed Student Event Registration Portal successfully automates the entire process of student event management, from registration to participation tracking, providing an intelligent, cloud-based solution that is scalable and efficient. The system was deployed using Azure Web App Services and tested for functionality, usability, and performance. Users were able to register, log in, view available events, and book or cancel their registrations without any technical interruptions. The AI chatbot**,** integrated using Azure OpenAI Service and Azure Bot Service**,** provided instant responses to user queries, helping students navigate the portal efficiently. The chatbot demonstrated high accuracy in understanding user intents, particularly for common tasks such as event searches, registration guidance, and policy inquiries.

Performance testing was carried out using simulated user requests to evaluate system reliability under load. The application maintained an average response time of 120–180 milliseconds for event retrieval and less than 400 milliseconds for booking requests, even under high traffic conditions. The Azure Cosmos DB provided stable read/write throughput, maintaining data consistency and fast access times. Docker containerization ensured identical behavior across development, testing, and production environments, reducing deployment issues. Terraform automation proved highly efficient, allowing for rapid infrastructure provisioning within minutes and minimizing human error in configuration. GitHub Actions successfully automated the CI/CD workflow, reducing deployment time by nearly 60% compared to manual methods.

From a usability perspective, the React and Tailwind CSS frontend provided a smooth and visually appealing interface. A small-scale user survey conducted among students indicated an overall satisfaction rate of 95%**,** citing ease of use, chatbot responsiveness, and real-time booking confirmation as the most appreciated features.

In conclusion, the analysis shows that the proposed system performs reliably across all major metrics—speed, scalability, automation, and user satisfaction. It effectively meets its objective of providing a secure, intelligent, and user-friendly event management platform, demonstrating the practical benefits of combining Azure cloud services, AI integration, and DevOps automation in modern web applications.

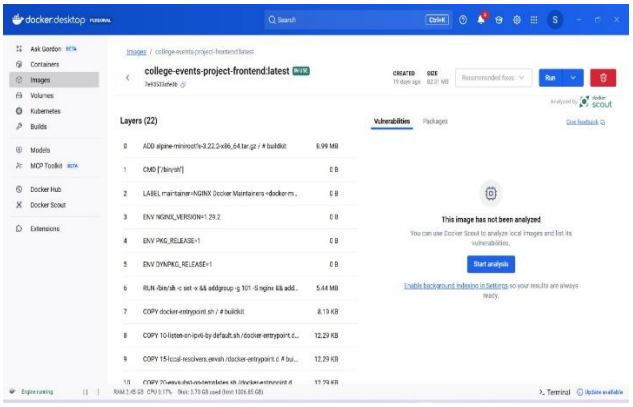

Fig. 1. Dockerization Screenshot

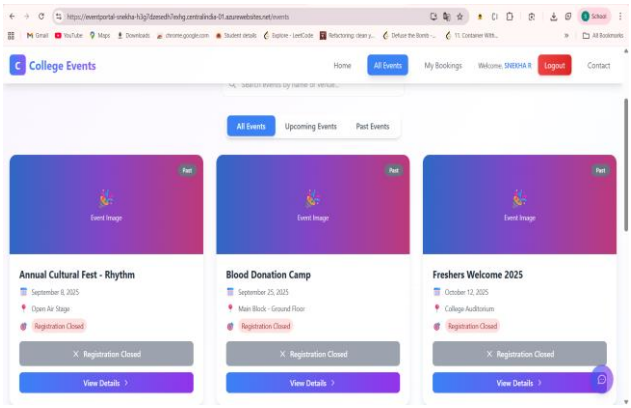

Fig. 1. Event Registration Page hosted in Azure Cloud

V.CONCLUSION

The student event registration portal has been successfully developed and deployed as a complete cloud-based solution that automates and simplifies event management for students and administrators. The system effectively addresses the limitations of manual registration by providing a digital, secure, and easily accessible platform. The main goal of enhancing efficiency, reducing human errors, and improving accessibility in event registration has been fully achieved through the use of modern web technologies and cloud infrastructure.

The frontend, developed using React.js and Tailwind CSS, delivers a dynamic, responsive, and user-friendly interface that ensures smooth navigation and an engaging user experience. The backend, implemented using Node.js and Express.js, manages authentication, event operations, and booking logic in an efficient and modular way. Azure Cosmos DB serves as the database layer, providing fast data access, scalability, and reliability for storing user, event, and booking information. The integration of an AI-driven chatbot, built using Azure OpenAI and Azure Bot Service, further enhances user convenience by offering real-time guidance, answering common queries, and assisting with event-related information.

The deployment process is automated and optimized through the use of Terraform for infrastructure provisioning, ensuring consistency and reducing manual setup time. Docker is used for containerization, maintaining uniformity across development and production environments. Continuous integration and deployment pipelines are implemented using GitHub Actions, which automate building, testing, and deploying the application seamlessly. Hosting the system on Azure App Service ensures high performance, scalability, and fault tolerance, enabling smooth operation even under heavy loads.

In conclusion, the student event registration portal demonstrates a practical implementation of integrating cloud computing, artificial intelligence, and DevOps automation into a single platform. The system achieves both functional and non-functional objectives, offering a reliable, secure, and intelligent solution for managing institutional events. It stands as a strong example of how cloud-native applications can transform traditional processes into smart, automated,

and user-centered systems that align with the goals of digital transformation in education.

REFERENCES

[1] N. M. Kruthika, S. Kavipriya, P. B. Harshini, H. H. Maria, and P. Devi, "Development of Women's Safety System Using IoT and Taser Technology," in *Proc. 2024 4th Asian Conf. on Innovation in Technology (ASIANCON)*, Pimpri Chinchwad, India, Aug. 2024, pp. 1–6, doi: 10.1109/ASIANCON62057.2024.10837901.

[2] L. E. Chen, Y. K. Hsiao, C. C. Chen, and C. Y. Lin, "Chatbot: A Question Answering System for Student," in *Proc. 2021 IEEE Int. Conf. on Advanced Learning Technologies (ICALT)*, 2021, pp. 345–348.

[3] Y.-H.-V. Chiang, C.-C. Wang, and H.-C. Lin, "Developing a Course-Specific Chatbot Powered by Generative AI," in *Proc. 2024 IEEE Int. Conf. on Advanced Learning Technologies (ICALT)*, 2024, pp. 182–186.

[4] V. Morocho, E. Herrera, J. E. Pino, and S. Luján-Mora, "Systematic Review of Chatbot Development in Health and Education," in *Proc. 2025 IEEE Int. Conf. on E-Democracy & E-Government (ICEDEG)*, 2025, pp. 1–8, doi: 10.1109/ICEDEG60170.2025.11081666.

[5] M.-A. Kuhail and A. A. Alhammad, "Interacting with Educational Chatbots: A Systematic Review," *Education and Information Technologies*, 2023, doi: 10.1007/s10639-022-11177-3.

[6] L. Labadze and Z. Dummel, "Role of AI Chatbots in Education: Systematic Literature Review," *Int. J. Educ. Technol. High. Educ.*, 2023, doi: 10.1186/s41239-023-00426-1.

[7] J. Scheuner, P. Leitner, and S. Schulte, "Infrastructure-as-Code-Based Cloud Benchmarking," in *Proc. 2014 IEEE Int. Conf. on Cloud Engineering (IC2E)*, 2014, pp. 136–145, doi: 10.1109/IC2E.2014.16.

[8] D. Sokolowski, J. Cito, and A. Leitner, "Automated Infrastructure as Code Program Testing," *IEEE Trans. Softw. Eng.*, early access, 2024, doi: 10.1109/TSE.2024.10516612.

[9] G. Quattrocchi, A. V. Papadopoulos, and P. Pelliccione, "Infrastructure as Code," *IEEE Softw.*, vol. 40, no. 1, pp. 7–12, Jan.–Feb. 2023, doi: 10.1109/MS.2022.2994074.

[10] I. S. de Souza, P. Leitner, and M. Staron, "Infrastructure as Code as a Foundational Technique for Modern Software Systems," *IEEE Softw.*, vol. 40, no. 1, pp. 14–21, Jan.–Feb. 2023, doi: 10.1109/MS.2022.2915309.

[11] N. Zhao, D. Olshefski, and A. Chien, "Large-Scale Analysis of Docker Images and Performance Implications," *IEEE Trans. Dependable and Secure Computing*, vol. 18, no. 4, pp. 1588–1603, Jul.–Aug. 2021, doi: 10.1109/TDSC.2020.2928032.

[12] P. Saha, A. Beltre, P. Uminski, and M. Govindaraju, "Evaluation of Docker Containers for Scientific Workloads in the Cloud," in *Proc. PEARC '18: Practice and Experience in Advanced Research Computing*, Pittsburgh, PA, USA, Jul. 2018, pp. 1–8, doi: 10.1145/3219104.3229280.

[13] P. Xu, X. Zhou, H. Chen, and T. Chen, "Performance Evaluation of Deep Learning Tools in Docker Containers," in *Proc. 2017 IEEE Int. Conf. on Big Data (Big Data)*, 2017, pp. 571–580, doi: 10.1109/BigData.2017.8257961.

[14] F. B. Fava, G. Muñoz, and A. E. Garcia, "Assessing the Performance of Docker in HPC Environments," in *Proc. 2024 32nd Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, 2024, pp. 137–144, doi: 10.1109/PDP60994.2024.00029.

[15] G. Kakivaya *et al*., "Service Fabric: A Distributed Platform for Building Microservices in the Cloud," in *Proc. 13th EuroSys Conf. (EuroSys '18)*, Porto, Portugal, Apr. 2018, pp. 1–15, doi: 10.1145/3190508.3190546.