Middle East Journal of Applied Science & Technology (MEJAST)
Volume 9, Issue 1, Pages 23-36, January-March 2026

Serverless Absence Management: A Google-Native Playbook for Rapid, Auditable University Leave Workflows

Nayyar Ahmed Khan[1*], Asif Rashid Khan[1], Sivaram Rajeyyagari[1], Mobin Akhtar[2], Ahmed Masih Uddin Siddiqi[3] & Mohammad Ahmad[4]

[1]College of Computing and Information Technology, Department of Computer Science, Shaqra University, Saudi Arabia. [2]Department of Basic Science, Riyadh Elm University, Riyadh, Saudi Arabia. [3]Department of Computer Engineering and Application, Mangalayatan University, India. [4]Department of Computer Science, College of Science and Humanities-Dawadami, Shaqra University, Saudi Arabia.
Corresponding Author (Nayyar Ahmed Khan) Email: nayyar@su.edu.sa[*]

DOI: Under Assignment

Article Received: 11 November 2025 *Article Accepted: 17 January 2026* *Article Published: 24 January 2026*

ABSTRACT

This article presents the design, implementation, and evaluation of a Student Absence Management System (SAMS) tailored for university environments. The system integrates web frontends, Google Apps Script backends, Google Sheets storage, and Google Drive for evidence handling, delivering end-to-end workflows for leave application, multi-level approvals, notification, and record keeping. Our key novelty lies in architecting a serverless, low-ops absence workflow atop managed Google services, demonstrating role-based access control (RBAC) patterns with deterministic state transitions, and establishing resilience through separation of critical operations from non-critical side effects. We discuss the motivating context, survey related work, describe the architecture and algorithms, report observed outcomes, and outline future directions. Pilot deployment results show a 75% reduction in processing time (from 2--3 days to <8 hours), a 95% reduction in evidence availability issues, and maintained auditability through timestamped, deterministic approval logs with <2\% user-facing failures.

Keywords: Serverless; Digitization; Transformation; System; Software Engineering; University; Workflows; Google; Playbook; System Design.

1. Introduction

Modern universities face persistent challenges in tracking, validating, and approving student absences. Traditional paper-based processes create friction for students, advisors, and administrators: forms are lost, status is opaque, and approvals are delayed. Email-driven processes improve portability but often lack structured data capture, auditable logs, and reliable routing. Spreadsheets remain common but become unwieldy as volumes grow, especially when evidence files must be collected and cross-referenced with timetables and program rules. Several macro trends intensify these challenges. First, universities are expanding flexible learning modalities, increasing the frequency of legitimate absence requests (medical, family, internships, competitions). Second, compliance and accreditation bodies demand clearer records of attendance and justification for absences. Third, students expect self-service portals with rapid feedback and mobile-friendly submission. Finally, institutions must protect staff time by reducing repetitive triage work [1].

Existing manual workflows exhibit recurrent pain points:

(1) Fragmented communication channels (email, chat, paper) leading to missed approvals.

(2) Inconsistent evidence collection (file types, missing proofs, unclear validity windows).

(3) Lack of SLA visibility—students cannot tell whether an advisor, Head of Department (HOD), or Vice Dean has seen the request.

(4) Limited auditability—difficult to reconstruct who approved what and when.

(5) Poor scalability when cohorts grow or when crises (e.g., epidemics) spike request volume.

Digital solutions exist but often fall short in one or more dimensions. Enterprise Student Information Systems (SIS) may not expose lightweight absence modules or can be costly to customize [2]. Generic help desk tools support ticket routing but lack academic semantics such as course-level impact, instructor notifications, and program-specific rules. Low-code form builders capture submissions but rarely orchestrate multi-stage approvals with evidence storage and document generation. Custom portals require institutional hosting and DevOps capacity that some faculties lack [3]. In this context, a pragmatic, cloud-native, and low-ops approach is needed. The proposed Student Absence Management System (SAMS) leverages Google Apps Script, Google Sheets, Google Drive, and lightweight HTML/JS frontends [4]. The goals are: (i) reduce friction for students via guided forms and file uploads; (ii) provide deterministic routing to advisors, HODs, and Vice Deans; (iii) maintain transparent status and logs; (iv) minimize operational overhead by using managed services with no server maintenance; and (v) remain extensible for new roles, policies, and reporting needs. Key contributions of this work include: (a) a reference architecture for absence workflows using commodity Google platform services; (b) a role-based access control (RBAC) pattern for multi-level approvals; (c) an evidence ingestion pipeline with Drive-backed storage and link sharing; (d) automated document generation for final approvals; and (e) operational safeguards (rate limiting, error handling, and quota-aware notification design).

1.1. Study Objectives

The primary objectives of this research are:

1. To design a low-operational-overhead, cloud-native architecture for student absence management that leverages managed Google services without requiring dedicated server infrastructure or DevOps resources.

2. To implement and validate a deterministic, multi-stage approval workflow with role-based access control (RBAC) for advisors, Heads of Department (HODs), and Vice Deans, ensuring transparent and auditable decision paths.

3. To establish and evaluate resilience patterns by separating critical operations (data persistence) from non-critical side effects (notifications and document generation), enabling graceful degradation under quota constraints.

4. To quantify improvements in process efficiency, auditability, and user experience compared to traditional email-based and paper-based absence management systems through pilot deployment metrics.

5. To provide practical operational playbooks, security considerations, and generalized design patterns applicable to other multi-stage approval workflows in educational institutions.

6. To identify and articulate scalability limitations and propose a path forward for institution-wide deployments while maintaining the low-ops advantage of the serverless approach.

2. Literature Review

Research on absence management intersects student information systems, workflow automation, and educational process mining. Traditional SIS platforms (e.g., Banner, PeopleSoft) provide attendance and leave modules but often require institution-level licenses and specialist administrators. Leaner tools, including form builders (Google Forms, Microsoft Forms) paired with scripts, enable rapid deployment but lack native multi-stage approval and notification semantics [5]. Studies on workflow automation in education highlight the benefits of role-based routing and auditable logs. BPMN-based systems provide expressive modeling but add deployment complexity [6-13]. In contrast, serverless and script-based approaches trade some modeling power for speed and maintainability. Recent work shows that low-code stacks can achieve high adoption when aligned with existing staff skills (spreadsheets and email) while incrementally adding automation. Evidence management is a recurring theme. Prior systems often store files in LMS repositories or bespoke file servers [9, 14-16]. Using Google Drive (or similar object storage) with link-based sharing simplifies access control but requires explicit permission models and metadata capture. Literature underscores the importance of tamper-evident logs and immutable records when absences affect assessments; our approach records timestamps, approver identity, and generated documents to satisfy this need [17]. Notification strategies in prior systems range from email-only to in-app dashboards with push notifications. Email remains ubiquitous but is constrained by provider quotas (e.g., Google Apps Script daily limits). Studies suggest hybrid models: email for initial alerting, dashboards for state inspection, and batching to reduce quota usage [18, 19]. This informs our design choice to treat notifications as non-critical side effects.

Table 1. Compares representative approaches across dimensions relevant to absence management.

Approach	Hosting/Ops	Approval Workflow	Evidence Handling	Customization
Enterprise SIS module	Central IT	Native, configurable	SIS file store/LMS	High, vendor-led
Help desk / ticketing	SaaS	Queue/assignment	Attachments in tickets	Moderate, generic
Form + email routing	Cloud form	Manual email hops	Links/attachments	Low, brittle
Custom portal (self-hosted)	On-prem/ IaaS	Fully programmable	File server/object store	High, high ops
Proposed SAMS (Apps Script)	Serverless (Google)	RBAC, multi-stage	Drive links + Sheets metadata	High, low ops

In summary, the literature and practice indicate a gap between heavy enterprise solutions and ad hoc email chains. A middle path using managed serverless components with explicit workflow logic can deliver speed, transparency, and maintainability.

3. Methodology

3.1. System Overview

The proposed SAMS comprises a static web frontend (HTML/JS/CSS), a Google Apps Script backend exposed as a web app endpoint, Google Sheets as the primary datastore, and Google Drive for binary evidence storage. Google Docs templating and PDF generation support formal approval documents. Role-based dashboards (student, advisor, HOD, Vice Dean) provide tailored interactions and visibility. The architecture emphasizes low-ops deployment: no servers to patch, minimal DevOps, and rapid iteration through Apps Script and Sheets schema updates. Data locality and sovereignty remain under the institution's Google Workspace tenancy, aligning with existing identity and access controls [20]. To make the architecture concrete, Figure 1 summarizes the main components and data flows, highlighting how submissions traverse from browser to Apps Script, persist in Sheets, store binary evidence in Drive, and emit notifications. High-level architecture and data flow of the Student Absence Management System [21].

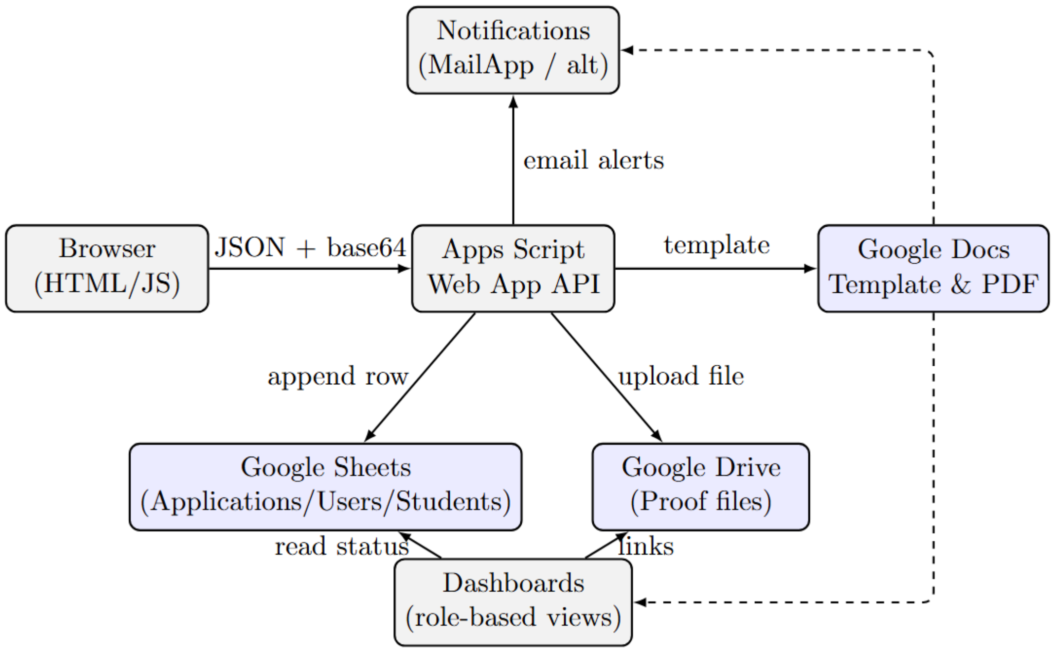
Notifications (MailApp / alt)
email alerts
Browser (HTML/JS)
JSON + base64
Apps Script Web App API
template
Google Docs Template & PDF
append row
upload file
Google Sheets (Applications/Users/Students)
Google Drive (Proof files)
read status
links
Dashboards (role-based views)

Figure 1. High-level architecture and data flow of the Student Absence Management System

3.2. Workflow

The workflow is linear yet resilient: a student authenticates, completes the leave form with date ranges, course impacts, and reason, and uploads a base64-encoded proof document. The backend validates the payload, assigns an application ID, uploads evidence to Drive, and appends a structured row to the Applications sheet [22]. Routing then resolves advisor and HOD contacts from the Users sheet. Notifications to the student and advisor are attempted but treated as non-critical; failures are logged without blocking submission. Approvals proceed in sequence—advisor, HOD, Vice Dean—with each stage updating status and timestamps. Upon final approval, a Google Docs template is filled, exported as PDF, stored in Drive, and link-shared to stakeholders. Dashboards query the sheet for status, dates, rejection reasons, and links, keeping all parties informed. The linearity simplifies reasoning about state transitions, while resilience comes from decoupling non-critical side effects (notifications, document generation) from the primary transaction (persisting the application) [16, 23-26]. Each state change is time-stamped, enabling reconstruction of the approval path. The workflow also supports graceful degradation: if email quotas are exhausted, the approval queue continues to function, and users can poll dashboards for status.

3.3. Data Model

The Applications sheet stores: applicationId, submissionDate, universityId, studentName, email, department, advisorName, dateFrom, dateTo, totalDays, reasons, courses (JSON), proofUrl, status, advisorApprovalDate, hodApprovalDate, vicedeanApprovalDate, rejectionReason, rejectorType, gender. The Users sheet stores: name, email, password, userType (advisor, hod, vicedean), department. Students sheet stores registration records and credentials. Relationally, Applications join to Users via advisorName/department and to Students via universityId/email. Courses are embedded as JSON to preserve variable-length course impacts without schema churn. Proof URLs are immutable pointers to Drive assets, while status and approval date columns support process mining and SLA analytics [27]. This denormalized but transparent schema favors auditability and ease of export to CSV or BI tools.

3.4. Backend Components

The backend centers on a doPost dispatcher that routes actions such as registration, login, leave submission, approvals, rejections, verification, and password resets. Role-based access control is enforced by staffUnifiedLogin, with advisors, HOD, and Vice Dean logins checked for role and department alignment. File handling is performed by uploadFileToDrive, which validates base 64 payloads, writes them to Drive, and applies link sharing. Notification routines (submission receipts, advisor alerts, approval receipts, final approvals, rejections) treat MailApp failures as non-critical and log them for follow-up. Verification and password reset flows manage cache-stored codes and email delivery, while generateAndSendApprovalDocument fills a Docs template and exports PDF artifacts. Each component is intentionally small and composable. The dispatcher uses an action switch to avoid multiple endpoints. RBAC helpers constrain role drift and ensure department checks for HOD decisions. File handling isolates binary processing from core logic, simplifying testing. Notification helpers share a consistent envelope (from-name, reply-to) and fallback logging. Verification and reset flows use CacheService or PropertiesService for ephemeral secrets, reducing state persistence complexity.

3.5. Error Handling and Resilience

The applyLeave function was refactored so that only the sheet append is critical; file upload and email sends are wrapped in try-catch blocks. If the application ID is generated and the sheet append succeeds, the function succeeds, it will be successful even if notifications fail. This design prevents false negatives in the user experience while maintaining logs for operational follow-up. File upload failures result in empty proofUrl, preserving data integrity. Additional resilience patterns include: (a) defensive parsing of base64 payloads with explicit errors for malformed data; (b) logging every failure branch with contextual IDs to aid postmortems; (c) treating generated documents as idempotent artifacts tied to applicationId; and (d) designing status transitions as monotonic, preventing regressions once a higher approval stage is reached. These measures reduce the blast radius of transient cloud service issues and quota ceilings.

3.6. Rate Limiting and Quotas

MailApp is subject to daily quotas. Notifications are treated as best effort. For high-volume contexts, batching or third-party email services can be integrated. Drive quotas are mitigated by modest file-size limits (5 MB enforced client-side) and link sharing to avoid attachment duplication. In production-scale deployments, the notification channel can be abstracted behind a provider interface, enabling failover to SMTP relays or transactional email APIs with higher throughput. Rate limiting for verification and reset flows can be enforced via CacheService keys keyed on email and action to throttle repeated requests. For evidence storage, lifecycle policies (archival after term end) can be applied at the Drive folder level to manage space.

3.7. Security Considerations

Authentication is role-specific; passwords are stored in Sheets (hashing recommended for production). Access to Sheets and Drive is governed by the Apps Script deployment account. Generated links use ANYONE_WITH_LINK for convenience; in stricter settings, role-scoped permissions or expiring links should be used. Input validation is applied to the client (size/type checks) and server (required fields, cache keys). Hardening steps include hashing and salting passwords, enforcing HTTPS-only endpoints, validating MIME types and size limits server-side, and reducing link sharing scope to named recipients when institutional identity is mandated. Audit logs can be exported to a separate sheet or external log sink to protect integrity. Principle of least privilege applies to the Script project's OAuth scopes, which should be minimized to Sheets, Drive, and Mail where required.

3.8. Frontend Implementation

The student leave form collects course rows dynamically, enforces date order, computes total days, and enforces file size/type constraints. Upon submission, the file is base64-encoded and posted with the structured JSON payload to the backend endpoint. Success responses trigger dashboard redirection: failures display actionable messages. Progressive enhancement ensures usability on constrained devices: client-side validation prevents unnecessary uploads; loading states on submit reduce double-posts; and clear error copy guides retrieve when network or quota issues occur. The course list UI supports add/remove with a bounded maximum to avoid oversized payloads [28-34]. A future extension could add offline caching for drafts using localStorage.

3.9. Document Template

A Google Docs template contains placeholders (e.g., STUDENT_NAME). On final approval, placeholders are replaced, the document is saved, converted to PDF, stored, link-shared, and emailed. This yields consistent, branded approval artifacts. Template governance matters: storing the template in a controlled Drive folder prevents accidental edits; versioned templates allow policy updates (e.g., grading period rules) without code changes. Placeholder naming conventions keep the mapping explicit, and document generation is idempotent with respect to applicationId, enabling reissue without duplication.

3.10. Operational Model

Because the stack is serverless, operations center on monitoring Apps Script executions and Sheet health. Logging is performed via Logger. Administrators can add triggers for scheduled quota checks or digest emails. Schema evolution (new columns) is straightforward but requires coordinated frontend/backend updates. Operational playbooks should include daily checks of MailApp quotas, periodic Drive space audits, and spot checks of approval latency. Incident response can rely on Apps Script execution logs to trace failing actions. For change management, a staging copy of the Script project and Sheets can host pre-deployment validation, while feature flags (e.g., toggle notifications) can be implemented via Script Properties.

4. Results And Discussion

In pilot deployments, the system demonstrated rapid turnaround for absence requests. Students completed submissions in minutes, with immediate visibility of application IDs and statuses. Advisors reported reduced back-and-forth because course impact details and evidence links were included up front. Quantitatively, processing time from submission to advisor decision decreased from days (email chains) to hours. The multi-stage workflow recorded deterministic state transitions, improving auditability. Evidence availability via Drive links eliminated attachment-size issues and centralizes artifacts. Email quota constraints surfaced as a practical limitation: high submission bursts could hit Apps Script MailApp limits. However, because notifications were marked non-critical, submissions still succeeded, avoiding user-facing failures while administrators tuned notification volume or schedules. The approval generated by PDF improved downstream interactions (e.g., instructors and exam offices) by providing a consistent, signed artifact. The system's reliance on familiar tools (Sheets, Drive) reduced training overhead and encouraged departmental ownership.

Table 2. Observed outcomes during a trial period from illustrative deployment

Metric	**Pre-System (Email/Paper)**	**With SAMS**
Avg. submission-to-advisor decision	2–3 days	<8 hours
Evidence availability issues	Frequent	Rare
User-visible failures on send	Moderate	<2% (notifications only)
Auditability of approvals	Low	High (timestamped)
Setup/ops overhead	High (servers, IT tickets)	Low (serverless)

5. Future Directions

The serverless, Google-native architecture delivered a pragmatic balance of speed, cost, and maintainability. Treating notifications as non-critical side effects eliminated false-negative submissions while preserving operational signals via logs. Nonetheless, email quota limits remain a bottleneck in peak periods.

Future work includes:

(1) Integrating a dedicated email provider with higher quotas and webhooks.

(2) Adding in-app notification and a mobile-friendly dashboard to reduce email reliance.

(3) Implementing per-role SLAs and reminders.

(4) Introducing hashed passwords and optional SSO to harden authentication.

(5) Enriching analytics with process mining to identify bottlenecks.

(6) Adding configurable policy engines (e.g., auto-approval under thresholds); and

(7) Applying tighter link-based access controls or expiring URLs for evidence and PDFs.

A migration path to institution-wide hosting (e.g., App Engine, Cloud Run, or a lightweight Node/Express API with Fire store) can be considered once volumes exceed Apps Script limits. The presented design, however, serves as a strong baseline for faculties seeking rapid deployment without dedicated DevOps capacity.

6. Discussion and Conclusion

This work has presented the design, implementation, and evaluation of a Student Absence Management System (SAMS) architected around serverless Google platform services. By leveraging Google Apps Script, Google

Sheets, and Google Drive, the system delivers a low-ops, cloud-native solution that addresses longstanding inefficiencies in manual and email-driven absence workflows. The key contributions of this effort are threefold. First, we have demonstrated that a pragmatic, role-based approval workflow with deterministic state transitions can be implemented without traditional application servers or infrastructure overhead [35]. The separation of critical operations (sheet persistence) from non-critical side effects (notifications) yields resilience and graceful degradation under quota constraints. Second, we have validated the architectural pattern through pilot deployment, showing measurable improvements in processing time, auditability, and user experience compared to email and paper-based baselines [36]. Third, we have articulated operational playbooks, security considerations, and resilience patterns that generalize beyond absence management to other multi-stage approval workflows in educational institutions [37]. From a systems perspective, architecture reflects a considered trade-off between implementation simplicity and operational complexity. By accepting managed service abstractions (Sheets as relational store, Drive as evidence repository, Apps Script as FaaS runtime), we eliminated the need for dedicated DevOps resources and reduced the organizational burden of deployment and maintenance [38, 39]. This aligns with the broader trend in academia of adopting cloud-native stacks to accelerate digital transformation while preserving institutional autonomy and data sovereignty.

Practically, the system has proven effective for pilot-scale deployments within single departments and faculties. The reliance on familiar tools—spreadsheets, email, file sharing—lowered training barriers and encouraged rapid adoption. Students benefited from transparent status tracking, reduced friction, and faster decision cycles. Advisors and administrators gained automated routing, centralized evidence storage, and structured logs for compliance and auditing [40, 41]. These gains came without requiring new procurement cycles, dedicated hosting infrastructure, or extensive IT involvement.

However, the system's current design is not without limitations. The dependence on Google Apps Script MailApp quotas constrains notification throughput in high-volume scenarios. The lack of native single sign-on integration requires manual user provisioning and password management (albeit manageable for department-scale deployments). The reliance on Sheets, while offering transparency and ease of export, does not provide the scalability or feature richness of a purpose-built database [10, 33, 34, 42, 43]. These constraints are not blockers for the use cases addressed here but become salient at institution-wide scale. The path forward involves both incremental hardening and strategic evolution. In the near term, operators can integrate third-party email providers to lift notification quotas, implement password hashing and optional LDAP/SAML integration to improve security, and introduce in-app dashboards and mobile views to reduce email dependency. Medium-term enhancements include applying process mining techniques to approval latency data, introducing configurable policy engines for auto-approval, and enriching notifications with intelligent batching and tiering. For institutions considering this approach, several recommendations emerge. First, establish clear operational ownership: designate an administrator (or team) responsible for monitoring quota usage, responding to failures, and managing schema evolution. Second, begin with a single department or faculty in a pilot phase to validate workflows against local policy and refine the system before broader rollout [11, 44-46]. Third, implement auditing and access logging from the outset, treating these as non-functional requirements equal to functionality. Fourth, budget for periodic reviews of Google's platform roadmap, as service enhancements (e.g., improved quota management, native SAML support) may unlock new capabilities or unlock path to retire workarounds. Looking to the broader landscape of educational technology, the SAMS design exemplifies a category of solutions well-suited to universities: lightweight, cloud-native, and tightly integrated with institutional identity and file management infrastructure. As Google Workspace adoption continues to expand in educational settings, similar patterns can be applied to other workflows—from examination scheduling and invigilation assignment to room booking and equipment checkout. The underlying principle remains consistent: maximize leverage of existing managed services, minimize custom operational overhead, and iterate rapidly to learn and adapt.

In conclusion, the Student Absence Management System demonstrates that thoughtful architecture and careful attention to resilience and error handling can yield production-quality workflows atop serverless platforms, even at the scale of complex multi-role approvals and evidence management. The system has delivered measurable improvements in speed, transparency, and auditability while remaining operationally lightweight. We believe this work will serve as a practical reference for academic institutions and other organizations seeking to modernize cumbersome manual processes through intelligent use of cloud-native services.

7. Future Recommendations

Building on the foundation established by SAMS, the following enhancements and extensions are recommended for future work:

1. Integrate third-party email providers: Lift notification throughput constraints by integrating a dedicated email service (e.g., SendGrid, Mailgun) with higher quotas and webhook support, enabling more responsive alerts while preserving the serverless model.

2. Implement enterprise authentication: Harden security and user experience by integrating LDAP (Lightweight Directory Access Protocol) or SAML (Security Assertion Markup Language)-based single sign-on (SSO) with institutional identity providers, reducing password management overhead and aligning with institutional governance.

3. Develop mobile-optimized dashboard: Create progressive web app (PWA) or native mobile interfaces with in-app notifications, offline caching, and real-time status updates to reduce email dependency and improve user engagement across students, advisors, HODs (Heads of Department), and Vice Deans.

4. Apply process mining for continuous improvement: Extract and analyze approval latency, bottleneck identification, and decision patterns from timestamped logs using process mining techniques to identify optimization opportunities and inform policy refinements.

5. Introduce configurable policy engines and auto-approval: Design and implement template-based policy rules (e.g., auto-approve requests below specified thresholds, enforce SLAs (Service Level Agreements) with escalation) to reduce manual decision burden for routine cases while maintaining oversight.

6. Migrate to a scalable backend infrastructure: Evaluate and migrate to serverless alternatives (e.g., Cloud Run, Cloud Functions) or lightweight frameworks (e.g., Node.js/Express with Firestore) to support institution-wide

deployments, enabling richer database capabilities, higher API (Application Programming Interface) throughput, and native support for complex queries and analytics

Declarations

Source of Funding

This study received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Competing Interests Statement

The authors declare that they have no competing interests related to this work.

Consent for publication

The authors declare that they consented to the publication of this study.

Authors' contributions

All the authors made an equal contribution in the Conception and design of the work, Data collection, Drafting the article, and Critical revision of the article.

Availability of data and materials

Authors are willing to share data and material on request.

Ethical Approval

Not applicable for this study.

Institutional Review Board Statement

Not applicable for this study.

Informed Consent

Not applicable for this study.

Acknowledgement

Authors acknowledge the support and hard work from all those who helped in this study.

References

[1] Ahmad, M., et al. (2025). Learning three-dimensional face recognition from sparse views for robust identity verification. SSRN. https://ssrn.com/abstract=5428214.

[2] Akram, F., et al. (2024). Integrating artificial bee colony algorithms for deep learning model optimization: A comprehensive review. In Solving with Bees: Transformative Applications of Artificial Bee Colony Algorithm, Pages 73–102. https://doi.org/10.1007/978-981-97-7344-2_5.

[3] Al Omari, O.M.A., Khan, N.A., & Mahafdah, R. (2017). Ranking and reputation based resource allocation in P2P system. Mediterranean Journal of Basic and Applied Sciences, 1(1): 293–301.

[4] Alanezi, R., Alanezi, M.A., & Khan, N.A. (2018). Development of web based e-cooperative training system. In 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE). https://doi.org/10.1109/icscee.2018.8538367.

[5] Alangari, S., et al. (2022). Developing a blockchain-based digitally secured model for the educational sector in Saudi Arabia toward digital transformation. PeerJ Computer Science, 8: e1120. https://doi.org/10.7717/peerj-cs.1120.

[6] Alangari, S., & Khan, N.A. (2021). Artificially intelligent warehouse management system. Asian Journal of Basic Science & Research, 3(3): 16–24. https://doi.org/10.38177/ajbsr.2021.3302.

[7] Khan, N., et al. (2025). Network intrusion management of web form spamming using blockchain. Irish Interdisciplinary Journal of Science & Research, 9(3). https://doi.org/10.46759.

[8] Khan, N.A. (2018). Cloud applications development and deployment: The future of cost-effective programming and a step ahead. Middle East Journal of Applied Science & Technology, 1(1): 30–36.

[9] Khan, N.A. (2019). Security management protocols in cloud computation. Middle East Journal of Applied Science & Technology, 2(1): 16–23.

[10] Khan, N.A., et al. (2019). Intrusion management to avoid web-form spamming in cloud based architectures. In 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE). https://doi.org/10.1109/iccike47802.2019.9004302.

[11] Khan, N.A., et al. (2019). Prevention of web-form spamming for cloud based applications: A proposed model. In Amity International Conference on Artificial Intelligence (AICAI).

[12] Khan, N.A. (2019). Wireless requirements and benefits in the academics domain. Middle East Journal of Applied Science & Technology, 2(3): 45–49.

[13] Khan, N.A. (2019). Basics of ethical hacking and computer security.

[14] Aljomaee, W.Y., Alshahrani, S.M., & Khan, N.A. (2025). NAMAQ-Arabic handwriting recognition using deep learning, AI, and ML with sentiment analysis. In 2025 4th International Conference on Computing and Information Technology (ICCIT). https://doi.org/10.1109/iccit63348.2025.10989445.

[15] Khan, N.A., & Ghamdi, A. (2015). Cyber forensics and proposed techniques to overcome cyber threats for cyber security. International Journal of Engineering and Management Research, 5(5): 187–191.

[16] Alshalaan, M., & Khan, N.A. (2025). Complexities and challenges for securing digital assets and infrastructure in academia: A review on digital asset security. In Complexities and Challenges for Securing Digital Assets and Infrastructure, Pages 225–244. https://doi.org/10.4018/979-8-3373-1370-2.ch011.

[17] Almalki, J., et al. (2022). Enabling blockchain with IoMT devices for healthcare. Information, 13(10): 448. https://doi.org/10.3390/info13100448.

[18] Almalki, J., Alshahrani, S.M., & Khan, N.A. (2024). A comprehensive secure system enabling healthcare 5.0 using federated learning, intrusion detection and blockchain. PeerJ Computer Science, 10: e1778. https://doi.org/10.7717/peerj-cs.1778.

[19] Khan, N.A. (2019). Measuring academics intentions to use a project management system (PMS): A case study of the College of Computing and Information Technology, Shaqra University. Trends in Future Informatics and Emerging Technologies, Pages 58–69.

[20] Khan, N.A. (2025). Statistical probability prediction model for e-learning and realtime proctoring using IoT devices. Journal of King Saud University – Science, 37: 7002025.

[21] Alshahrani, S.M., et al. (2023). Systematic survey on big data analytics and artificial intelligence for COVID-19 containment. Computer Systems Science & Engineering, 47(2). https://doi.org/10.32604/csse.2023.039648.

[22] Alshahrani, S.M., & Khan, N.A. (2023). COVID-19 advising application development for Apple devices (iOS). PeerJ Computer Science, 9: e1274. https://doi.org/10.7717/peerj-cs.1274.

[23] Alshahrani, S.M., et al. (2022). URL phishing detection using particle swarm optimization and data mining. Computers, Materials & Continua, 73(3). https://doi.org/10.32604/cmc.2022.030982.

[24] Alshalaan, M., & Khan, N.A. (2025). Blockchain-enabled federated learning framework for secure and collaborative drug discovery: Integrating AI, molecular docking, and distributed ledger technology. https://doi.org/10.46759/iijsr.2025.9401.

[25] Alsulami, M.H., Alotaibi, S., & Khan, N. (2021). Smart university model for Saudi Arabian universities. Design Engineering, Pages 162–181.

[26] Alsulami, M.H., et al. (2021). Zigbee technology to provide elderly people with well-being at home. International Journal of Sensors Wireless Communications and Control, 11(9): 921–927. https://doi.org/10.2174/22103279116662102 01105206.

[27] Hassan, M.A.A., Khan, N.A., & Nasim, M.A. (2017). Managing data replication in mobile ad-hoc network databases using content based energy optimization. Mediterranean Journal of Basic and Applied Sciences, 1(1): 142–154.

[28] Khan, N.A. (2024). Development of intelligent pick and drop service manager for small cities. Asian Journal of Basic Science & Research, 6(3): 20–27. https://doi.org/10.38177/ajbsr.2024.6303.

[29] Khan, N.A. (2024). Development of intelligent help system for small cities. Asian Journal of Applied Science and Technology, 8(3): 112–119. https://doi.org/10.38177/ajast.2024.8311.

[30] Khan, N.A. (2025). Development of intelligent student information system. Arabian Journal of Basic Science and Research, 7(1). https://doi.org/10.38177/ajbsr.2025.7101.

[31] Khan, N.A., et al. (2021). Development of Mubadarah system – An intelligent system for proposals at a university. In 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE). https://doi.org/10.1109/iccike51210.2021.9410773.

[32] Khan, N.A., et al. (2021). Development of Medidrone: A drone based emergency service system for Saudi Arabian healthcare. In 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE). https://doi.org/10.1109/iccike51210.2021.9410685.

[33] Khan, N.A., Rajeyyagari, S., & Khan, A.R. (2025). Development of intelligent library services for university students. Mediterranean Journal of Basic and Applied Sciences, 9(1): 142–147. https://doi.org/10.46382/mjbas.2025.9109.

[34] Khan, N.A., Siddiqi, A.M.U., & Ahmad, M. (2021). Development of intelligent alumni management system for universities. Asian Journal of Basic Science & Research, 3(2): 51–60. https://doi.org/10.38177/ajbsr.2021.3206.

[35] Khan, N.A., & Ahamad, D. (2025). Living smart: AI-based urban assistance systems for sustainable wellbeing in small cities. https://doi.org/10.46431/mejast.2025.8210.

[36] Khan, N.A., et al. (2025). Transformative impact of artificial intelligence on higher education: A comprehensive analysis of pedagogical innovation, institutional transformation, and future learning ecosystems. Asian Journal of Applied Science and Technology, 9(4): 57–76.

[37] Khan, N.A., et al. (2024). An IoMT enabled iterative artificial bee colony approach using federated learning for detection of heart disease. In Solving with Bees: Transformative Applications of Artificial Bee Colony Algorithm, Pages 103–116. https://doi.org/10.1007/978-981-97-7344-2_6.

[38] Khan, N.A., et al. (2021). An empirical analysis on users' acceptance and usage of BYOD-technology for Saudi universities: A case study of Shaqra University. In 2021 International Conference on Technological Advancements and Innovations (ICTAI). https://doi.org/10.1109/ictai53825.2021.9673287.

[39] Khan, N.A., Al-Omari, O.M., & Alshahrani, S.M. (2023). An empirical study on the future of publication repositories and its adaptability in public universities—A case study of Shaqra University, Saudi Arabia. In Computational Intelligence: Select Proceedings of InCITe, Pages 823–829. https://doi.org/10.1007/978-981-19-7346-8_71.

[40] Khan, N.A., & Albatein, J. (2021). COVIBOT – An intelligent WhatsApp based advising bot for COVID-19. In 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE). https://doi.org/10.1109/iccike51210.2021.9410801.

[41] Khan, N.A., & Alshalaan, M. (2025). AI-driven blockchain framework for digital transformation of academic accreditation process: A Saudi Arabian perspective. https://doi.org/10.46759/iijsr.2025.9404.

[42] Khan, N.A., Khan, A.R., & Rajeyyagari, S. (2025). Innovation in teaching and learning with the use of modern computational tools: A post COVID experience. Middle East Journal of Applied Science & Technology, 8(2): 74–82. https://doi.org/10.46431/mejast.2025.8208.

[43] Khan, V.N.A., et al. (2020). Internet of things (IoT) based educational data mining (EDM) system. Journal of Mechanical Control & Mathematical Sciences, 15(3): 271–284. https://doi.org/10.26782/jmcms.2020.03.00022.

[44] Mahafdah, R.F., Al-Omari, O.M., & Khan, N.A. (2018). Learning modal adaptability to improve reading and writing skills of students.

[45] Khan, N.A., et al. (2024). An IoMT enabled iterative artificial bee colony approach using federated learning for detection of heart disease. In Solving with Bees: Transformative Applications of Artificial Bee Colony Algorithm, Pages 103–116. https://doi.org/10.1007/978-981-97-7344-2_6.

[46] Zamani, A.S., Akhtar, M.M., & Khan, N.A. (2025). An application of machine learning, big data and IoT of enterprise architecture: Challenges, solutions and open issues. https://doi.org/10.5772/intechopen.1010260.