International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

College Entrance Kiosk System: A Self-Service Digital Platform For College Information and Admission-Related Services

Author 1: Arefa Huned Khokhar

Author 2: Manya Balar

Author 3: Ifraa Shariff

Guide: Ms. Shrinkhala Kumari

Abstract - Students and visitors may need to consult several offices or information sources for course, fee, admission, and faculty-appointment information. This paper presents the design and preliminary implementation of a College Entrance Kiosk System intended to consolidate selected college services in a single self-service interface. The application is developed using Flutter and Dart, with Node.js and Express.js forming the server-side layer and MongoDB selected for document-oriented data storage. The planned service scope includes college information screens, course and admission details, fee information, faculty appointment scheduling, and WhatsApp/E-mail notifications. At the current development stage, the user interface and backend exist, but some client–API integrations and persistent appointment-booking behavior remain incomplete. Basic manual testing has been performed; however, a controlled user study and quantitative performance evaluation have not yet been completed. Consequently, this paper reports the system architecture, design decisions, implementation status, and an evaluation plan rather than claiming measured improvements in waiting time, error rate, or user satisfaction. The work provides a practical prototype and a structured basis for completing and evaluating a self-service college information system.

Keywords—College information kiosk, self-service system, admission information, Flutter, Dart, Node.js, Express.js, MongoDB, appointment scheduling.

I. INTRODUCTION

Colleges communicate programme, admission, fee, faculty, and schedule information through notices, websites, phone contacts, and offices. This fragmentation can make information harder to find and create repeated routine enquiries. A self-service kiosk can present frequently requested information through one consistent interface.

Kiosks are a form of self-service technology whose purpose and interaction depend on the service context. Earlier campus case studies used touch-screen kiosks for course catalogues, directories, advising, and campus information. These studies support the relevance of kiosk access, although their technical contexts differ from a cross-platform client with a web API.

This student project combines college, course, admission, and fee information with faculty appointment requests and planned messaging. Flutter/Dart form the client, Node.js/Express.js the backend, and

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

MongoDB the data store. It remains a prototype: some integration and persistent booking are pending, so planned capabilities are not treated as validated production features.

A. Aim and Objectives

The aim is to centralize selected college information and admission-related services. Objectives are to: (1) present course, admission, and fee information; (2) separate UI, service logic, and storage; (3) define a faculty appointment workflow; (4) plan WhatsApp/E-mail integration; and (5) specify pre-deployment tests.

B. Research Questions

RQ1: How can a cross-platform client, REST-style backend, and document database be organized to support a college information kiosk? RQ2: Which integration and data-persistence requirements must be satisfied for appointment scheduling and notification features to be dependable? RQ3: What preliminary functional checks and future measurements are necessary to assess the prototype without overstating its current maturity?

C. Contributions and Scope

The paper contributes a documented system architecture, a feature-oriented workflow, an illustrative data-domain model, an implementation-status assessment, and a test plan. It does not claim a novel algorithm, a completed production deployment, verified notification delivery, or statistically significant user benefits. Payment processing, official admission decision-making, identity verification, and institution-wide integration are outside the confirmed scope.

II. RELATED WORK

Kiosk systems are generally designed to make selected information or transactions available without requiring a staff member to mediate every interaction. Slack discusses kiosk-based self-service as a broad category rather than a single fixed technology. The implication for campus systems is that the interface should be designed around the user's task and the services the institution is prepared to maintain, rather than around hardware alone.

Jonas et al. reported campus information kiosk implementations at community colleges, including course catalogues, directories, advising information. The cases demonstrate that successful deployment involves more than software: institutional ownership, content maintenance, and coordination between administrative units are also relevant. These operational factors apply to a college entrance kiosk because outdated fee or admission information could mislead users even when the interface functions correctly.

Negandhi et al. presented an online college admission system intended to reduce manual effort and queues by moving selected admission activities online. This is closely related in domain, but an online admission portal and an entrance kiosk are not identical. A kiosk emphasizes a guided, potentially shared on-campus

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

point of access; an online portal is commonly accessed from a user's own device. The proposed system combines a kiosk-oriented information interface with a backend that can support selected requests.

The technology stack is supported by established documentation. Flutter provides a framework for building applications from a shared codebase, while Express.js defines HTTP routes that respond to methods such as GET and POST . MongoDB stores records as BSON documents and supports document-oriented data modelling. These technologies provide implementation building blocks, but their availability alone does not establish that an application is secure, usable, reliable, or correctly integrated.

The project addresses a practical need for a documented design linking campus information with appointment requests while distinguishing implemented components from pending integrations. This is a scoped project contribution, not a claim that no comparable system exists.

D. Comparative Analysis

The reviewed approaches show complementary strengths. Campus information kiosks emphasize convenient on-site access to institutional content, while online admission systems focus on form submission and administrative processing. The proposed kiosk is positioned as a guided access point for common information and appointment requests. The comparison below is qualitative and is based on the scope described in the cited sources; it is not a benchmark test of those systems.

Approach	Typical emphasis	Relevance to this work	Important distinction
Information kiosk taxonomy	Classifies kiosk services by location, user, task and technology.	Supports treating the kiosk as a service interface shaped around user tasks.	Conceptual taxonomy, not a specific college implementation.
Campus kiosk case studies	Touch-screen access to course catalogues, directories, advising and campus information.	Shows the importance of institutional content and cross-department coordination.	Historical deployments; their technology stack differs from this project.
Online admission system	Online registration and admission-related forms with server-side records.	Demonstrates the wider move from manual to digital admission workflows.	Online portal and shared on-campus kiosk have different access contexts.
Proposed College Entrance Kiosk	College/course/admission/fee information plus	Combines information retrieval	Prototype status; booking persistence

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

	appointment-request workflow.	with a planned service-request pathway.	and notification integration require verification.

III. PROBLEM STATEMENT AND REQUIREMENTS

The problem is fragmented access to college information and the absence of one guided interface for enquiries and appointment requests. Whether the kiosk reduces queues or staff workload requires future field evaluation.

A. Functional Requirements

ID	Requirement	Expected behaviour / status
FR-01	College information	Display approved college information; define content owner and update cycle.
FR-02	Course and admission details	Show course, eligibility, and admission guidance approved by the institution.
FR-03	Fee information	Show dated fee details with an update process; no payment processing is claimed.
FR-04	Faculty appointment scheduling	Collect request and preferred time; requires API integration, validation, conflict handling, and storage.
FR-05	WhatsApp/E-mail notifications	Send updates after saving booking; provider integration and delivery need verification.
FR-06	Data access	Use backend APIs for data access; test all routes and errors.

B. Non-Functional Requirements

Usability requires clear labels and navigation. Reliability requires clear handling of failed requests. Security requires validation, authorization, and secure transport. Information should be maintainable without unnecessary client rebuilds. Privacy requires data minimization and transparent use. Accessibility checks should cover text, contrast, touch targets, language, and disability access.

C. User Roles and Assumptions

Primary users are prospective students, students, and visitors. Faculty or administrative staff may review requests and maintain information, depending on the final role model. The institution must supply accurate information. Integration with an official student information system is not claimed.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

IV. PROPOSED SYSTEM DESIGN

A. Architecture

The architecture separates presentation, API, service, and data layers. Flutter/Dart provide the UI; HTTP requests connect the client to Express.js routes running on Node.js; MongoDB stores records as documents. Express maps routes and HTTP methods to application behaviour, while MongoDB supports document-oriented data modelling .

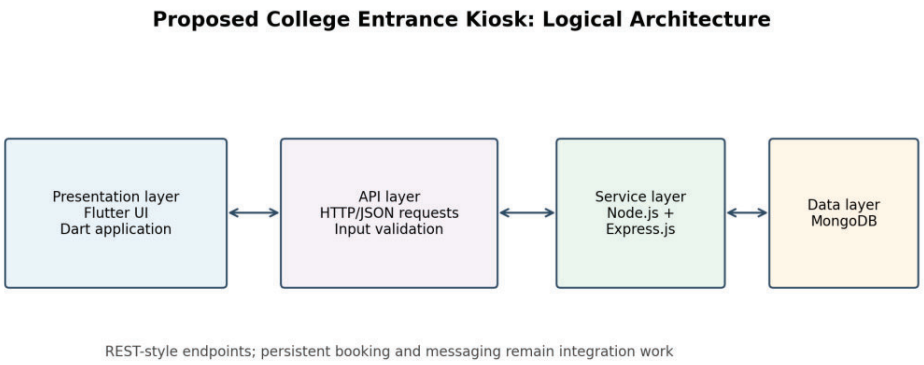


Fig. 1. Logical architecture of the proposed College Entrance Kiosk System.

B. Main Modules

Information module: displays college, course, admission, and fee content. Frequently changing information, especially fees and deadlines, should be date-stamped.

Appointment module: captures purpose, faculty/department, preferred time, and necessary contact details. The server should validate fields and availability. A booking must not be shown as confirmed until saved successfully. Persistent booking is pending, so this is currently a partial workflow.

Notification module: sends messages only after booking state is saved. Provider credentials must remain server-side, and messages should minimize personal data. Delivery depends on provider setup, consent, network, and provider status; it is not verified in this paper.

Administrative content: deployment requires controlled updates to course, admission, fee, and faculty information. A complete admin dashboard has not been confirmed and is treated as a requirement.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

C. Workflow

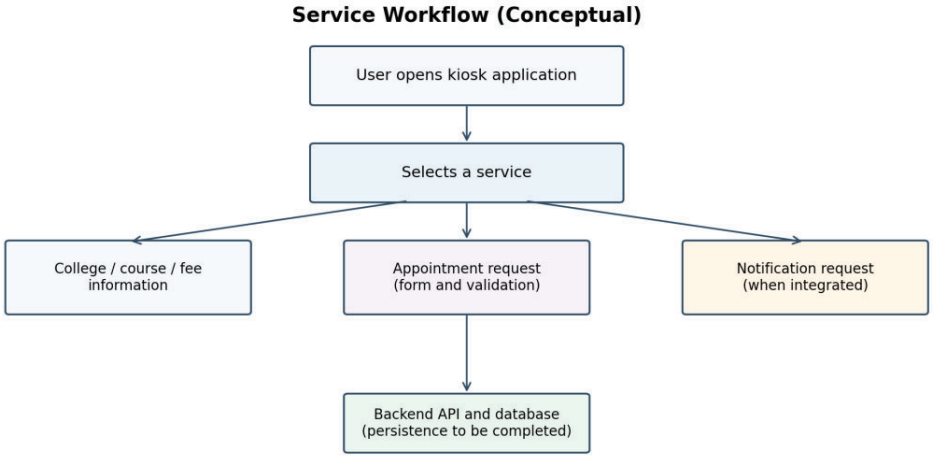


Fig. 2. Conceptual service workflow; the persistence and notification steps require completion and verification.

For information requests, users select a service and view content. For appointments, users enter details, the client checks basic fields, and the backend validates and checks scheduling rules before saving and returning status. Notifications should follow a successful state change. A failed save must produce an error, not a false confirmation.

D. Illustrative Data Domains

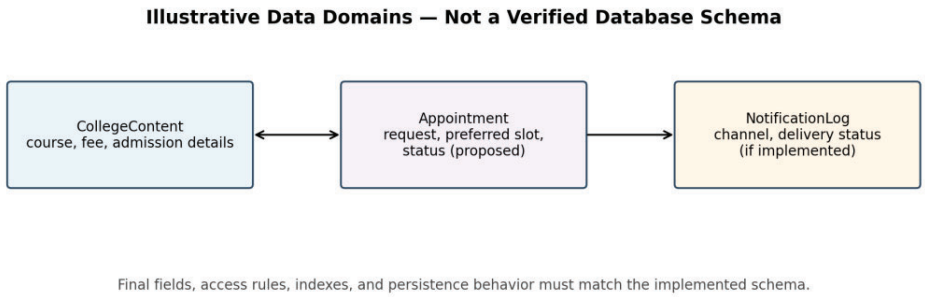


Fig. 3. Illustrative data domains for discussion; this is not a verified representation of the existing MongoDB schema.

Possible data domains include CollegeContent (approved information and update metadata), Appointment (request details, selected slot, and status), and NotificationLog (channel, related request identifier, attempt time, and delivery result). These are logical domains, not a statement that collections with these exact names

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

already exist. The implemented schema should be documented from the actual code and database before submission.

E. Data Validation and Error Handling

Input validation should occur at both the client and server layers. Client-side checks can improve immediate feedback, but they cannot replace server-side validation because API requests may be sent without using the application interface. The server should reject missing required fields, malformed contact information, invalid identifiers, unsupported appointment times, and values outside defined limits. Responses should distinguish validation errors, unavailable resources, authentication failures, and unexpected server errors. The client should preserve user-entered information where safe, display a clear recovery action, and avoid presenting an appointment as confirmed until the database write succeeds.

V. IMPLEMENTATION METHODOLOGY

B. Development Approach

The project follows an iterative prototype approach: define requirements, design screens, build backend routes, connect storage, and test workflows. The UI and backend exist, but some API integration and booking persistence remain incomplete. Booking persistence should be completed before notification delivery.

A. Research and Development Method

This paper uses a design-and-development approach. The problem context and feature scope are defined first; a layered architecture is then specified; the client interface and backend are developed as separate components; and functional checks are used to identify integration gaps. The present paper is a preliminary system report rather than an experimental comparison. It documents the intended workflow and known implementation status, then specifies a repeatable evaluation plan for the next development stage. The study does not include a controlled comparison group or a statistically sampled user population.

C. Technology Stack

Technology	Role in the system	Design rationale
Flutter	Cross-platform application UI	Cross-platform widget UI framework.
Dart	Client-side programming language	Client UI behaviour and request handling.
Node.js	Server runtime	Runs backend JavaScript services.
Express.js	HTTP API framework	Routes and middleware for HTTP requests.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

MongoDB	Data store	Stores structured records as BSON documents and supports document-oriented modelling.

D. Client–Server Communication

The client should use API endpoints, not access MongoDB directly. GET retrieves information; POST submits requests; PATCH/PUT may update status; DELETE must be restricted. Confirm all routes against the source code. Use consistent responses, suitable HTTP status codes, validation, and error handling.

E. Booking Persistence Requirements

Reliable booking requires server-side field validation, valid faculty/department selection, availability checks, MongoDB persistence, a stable request ID, duplicate handling, and authorized status updates. Return success only after saving. Concurrent requests require a strategy that prevents double booking.

F. Notification Integration

WhatsApp/E-mail is a separate integration. Use an approved provider, store credentials securely, log attempts, and handle errors. Distinguish a saved booking, provider acceptance, and actual delivery. Do not report notifications as complete before integration tests.

G. Security and Privacy Considerations

A shared kiosk must clear personal data and return to a neutral screen after inactivity. Server controls should include validation, authentication and authorization for privileged operations, rate limiting where appropriate, secure transport, restricted database credentials, and safe errors. Retain contact details only as needed. Security and privacy reviews are required before real data is used.

VI. PRELIMINARY TESTING AND RESULTS

Basic manual testing has been performed. No formal report, sample size, timing, satisfaction score, or benchmark data was supplied. The evaluation separates known status from proposed tests; no unreported test is claimed as passed.

Area	Current evidence/status	Next verification step
User interface	UI screens exist; individual screen completeness should be checked against the final build.	Verify navigation, text overflow, touch targets, and return-to-home behaviour.
Backend	Backend exists; full client–API integration is incomplete.	Exercise each implemented route using valid and invalid inputs.
Course/admission/fee information	Included in the feature scope; final data source and update process must be confirmed.	Check content accuracy, date/version, empty states, and unavailable API handling.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

Appointment scheduling	Feature is in scope; persistent booking is pending.	Submit a request, verify the database record, retrieve its status, and test duplicate/conflicting requests.
WhatsApp/E-mail	Notification integration is planned/in scope; successful delivery is not established.	Test provider configuration, consent, success/failure responses, and delivery status.
Security/session reset	Requires targeted validation before shared-kiosk deployment.	Test unauthorized access, session data clearing, inactivity reset, and credential handling.

A. Test Procedure to Complete

Test a documented build against a non-production database. Record ID, preconditions, steps, expected and actual results, outcome, date, and build. Repeat tests after API/database changes. Appointment tests should cover missing fields, invalid contacts, unavailable slots, duplicates, database failures, and timeouts. Notification tests should cover rejection and delay.

B. Evaluation Metrics

After integration, measure task completion, information-retrieval time, booking persistence success, duplicate booking, API response time, notification status, and user feedback. Report sample size, environment, method, and limitations. No measurements were supplied for this draft.

The following matrix is a proposed test plan, not a record of completed or passed tests. The actual-result column must be filled only after execution against a specific build.

Table V. Proposed functional test cases.

Test ID	Scenario	Expected result	Current status
TC-01	Open course/admission/fee screens.	Correct content appears; navigation remains responsive.	To execute and record.
TC-02	Request data endpoint while backend is unavailable.	User receives a clear error; app does not falsely show data loaded.	To execute and record.
TC-03	Submit appointment with required fields missing.	Server rejects request and returns a validation response.	Pending end-to-end integration.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

TC-04	Submit a valid appointment request.	Record persists and response includes a stable request identifier.	Pending persistent booking.
TC-05	Submit the same request twice or request an occupied slot.	Duplicate/conflicting booking is prevented or clearly handled.	Pending booking rules.
TC-06	Attempt notification after a failed database save.	No confirmation message is sent for an unpersisted booking.	Pending notification integration.
TC-07	Allow a kiosk session to remain idle after entering contact details.	Personal information is cleared and the application returns to a neutral screen.	Requires verification.
TC-08	Attempt privileged operation without authorization.	Request is denied without exposing private records.	Requires security testing.

C. Discussion

The prototype separates a cross-platform UI from backend services, but this alone does not prove reduced queues or improved satisfaction. Main risks are incomplete API integration, unverified booking persistence, and unverified messaging. Priorities are to stabilize the API, persist and validate bookings, add authorization and session reset, then integrate messaging and retest.

A public kiosk is shared and often used briefly by first-time users. Evaluation should include accessibility and task completion without staff help. Content governance matters because outdated fees or deadlines can mislead users.

VII. LIMITATIONS AND FUTURE WORK

This work describes a prototype and basic manual testing. Full Flutter–backend integration is not confirmed for every feature; persistent booking and WhatsApp/E-mail delivery remain unverified. No controlled user study, baseline comparison, performance benchmark, or security audit is available. Diagrams must be checked against the final source code.

Future work should complete booking persistence, conflict and duplicate handling, messaging, session reset, role-based access, and content maintenance. Evaluation should compare task completion and retrieval time with current processes, explain participant selection, and report all findings. Further options include

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

multilingual content, accessibility, audit logs, privacy-preserving analytics, and authorized institutional integration.

VIII. THREATS TO VALIDITY

Several limitations affect interpretation. First, the implementation is not fully integrated, so the architecture and proposed workflow should not be interpreted as proof that every feature operates end to end. Second, basic manual testing provides limited coverage and does not establish performance under concurrent use, security robustness, or usability across different user groups. Third, no quantitative baseline has been collected to compare the kiosk with the current information-seeking process. Finally, the content accuracy and operational suitability depend on institutional approval and ongoing maintenance. These limitations restrict the conclusions to the design and preliminary development status reported here.

IX. CONCLUSION

This paper presented a preliminary College Entrance Kiosk System for college, course, admission, and fee information, faculty appointment requests, and planned notifications. It uses Flutter/Dart, Node.js/Express.js, and MongoDB. Some integrations and persistent booking remain incomplete, and basic manual testing does not establish improved efficiency or usability. Next steps are to complete booking persistence, verify notifications, test shared-kiosk privacy, and conduct documented functional and user evaluation. The system is a prototype, not a validated production deployment.

REFERENCES

[1] J. Rowley and F. Slack, “Information kiosks: a taxonomy,” Journal of Documentation, vol. 63, no. 6, pp. 879–897, 2007. doi: 10.1108/00220410710836402. Available: https://shura.shu.ac.uk/51/

[2] S. Jonas, S. Megregian, J. Leslie, D. Burnett, and G. Wenger, “Implementing a Kiosk-based Campus Information System at Community Colleges: Three Case Studies,” CAUSE Conference Proceedings, 1994. Available: https://library.educause.edu/resources/1994/1/implementing-a-kioskbased-campus-information-system-at-community-colleges-three-case-studies

[3] D. Negandhi, A. Mishra, A. Doshi, S. Desai, and R. Thakare, “Online College Admission System,” International Journal of Engineering Research & Technology (IJERT), ICIATE, vol. 5, no. 1, 2017/2018. DOI: 10.17577/IJERTCONV5IS01013. Available: https://www.ijert.org/online-college-admission-system

[4] Flutter, “Flutter documentation.” Available: https://docs.flutter.dev/ (accessed 9 Oct. 2026).

[5] Express.js, “Routing.” Available: https://expressjs.com/en/guide/routing.html (accessed 9 Oct. 2026).

[6] MongoDB, “Documents — MongoDB Manual.” Available: https://www.mongodb.com/docs/manual/core/document/ (accessed 9 Oct. 2026).

(This work is licensed under a Creative Commons Attribution 4.0 International License.)