

Web-Based Quality Assurance Activity Monitoring Dashboard for LPM UIN Sumatera Utara Using RAD

Zaki Mahbub^{*1}, Diaz Alfaridzi², Arif Syafaruddin Gultom³

^{1,2,3}Computer Science, Universitas Islam Negeri Sumatera Utara, Indonesia

¹zakimahbub1208@gmail.com, ²diazalfaridzii@gmail.com, ³arifsgultom@gmail.com

Received: September 19, 2026 | Revised: September 22, 2026 | Accepted: October 30, 2026 | Published: October 3, 2026



COSIE is licensed a Creative Commons Attribution-Share Alike 4.0 International License

Abstract

The Quality Assurance Institute (LPM) at UIN Sumatera Utara manages diverse quality assurance activities, but records remain distributed across printed documents and spreadsheets, hindering progress monitoring and report compilation. This study aimed to design and implement a web-based quality assurance activity monitoring dashboard using Rapid Application Development (RAD) that centralizes activity recording, monitoring, and reporting, and to verify its functional behavior against specified requirements. Development followed four stages: requirements planning, user design, construction, and cutover. Use case, activity, and class diagrams guided implementation. The backend used Node.js, Express, and Prisma with PostgreSQL, while the frontend used React and Vite. Functional evaluation employed black box testing of nine scenarios, one for each of nine functional requirements; non-functional requirements were specified as design requirements and were not empirically evaluated. The resulting system comprised nine web pages covering authentication, profiles, a summary dashboard, activities, a calendar, categories, documentation, documentation details, and user administration. All nine executed test cases (TC-01–TC-09) passed, yielding a functional test-case success rate of 100% (9/9) within the tested scope. The findings indicate that the implemented dashboard satisfied the specified functional requirements in the executed scenarios and brought quality assurance activity monitoring into a single web application. Its applicability to similar institutions remains to be validated. Effects on reporting time, usability, security, and operational performance were not measured and require further evaluation.

Keywords: *black box testing; monitoring dashboard; quality assurance; Rapid Application Development; web-based information system*

1. INTRODUCTION

State Islamic higher education institutions perform quality assurance through their Quality Assurance Institutes (LPM), which manage the Internal Quality Assurance System (SPMI) cycle, including internal quality audits [1], monitoring and evaluation, training, dissemination of quality standards, and preparation for study program accreditation. These diverse activities require LPM UIN Sumatera Utara to record, manage, and report their implementation systematically so that university leaders and external assessors can monitor progress. In practice, activities at LPM UIN Sumatera Utara are still recorded manually in printed documents and electronic spreadsheets distributed among several staff members, making it difficult for leadership to obtain an up-to-date overview of progress. The absence of a centralized monitoring dashboard makes data compilation for accreditation and performance evaluation time-consuming. Similar problems occur in organizations without integrated monitoring dashboards, including previously decentralized municipal surveillance camera monitoring [2], manual sales performance monitoring [3], and manual inventory recording and reporting [4]. These conditions delay decision-making and limit access to current information. These examples show why centralized monitoring dashboards are generally relevant, but they come from municipal, commercial, and corporate settings and therefore do not, by themselves, establish a research gap for higher-education quality assurance, which is examined in the following paragraphs.

Previous studies review Rapid Application Development (RAD) among information system development methods [5]. RAD emphasizes active user involvement and rapid iteration across requirements planning, user design, construction, and cutover, enabling relatively short development cycles while maintaining alignment with user needs. RAD has been applied across domains, including a hospital complaint management application, where its use was reported to speed up application development [6], a tourism village information system [7], and higher education courses that teach rapid development on the Oracle Application Express platform [22]. These characteristics make RAD relevant to quality assurance

organizations that require timely development and continuous adaptation to user needs. In quality assurance, previous studies have developed information systems and dashboards to support SPMI. Examples include an online quality assurance information system for Muhammadiyah/Aisyiyah higher education institutions using Laravel [8], a higher education quality assurance dashboard designed using Design Thinking [9], and a web-based SPMI system developed using RAD with monitoring dashboard features [10]. These studies demonstrate the growing importance of web-based quality assurance monitoring, although their main focus is on specific modules such as SPMI document management or general quality administration. Business intelligence dashboards have also been proposed for monitoring quality assurance data in higher education institutions [23]; that work addresses big-data handling and compliance monitoring through BI dashboards, which differs from the activity-recording focus of this study. Web-based activity monitoring dashboards have also been studied in other contexts, including a campus event information system that accelerates the dissemination of student activity information [11] and a student achievement monitoring application for secondary schools [12]. Taken together, the reviewed quality assurance studies [8]–[10], [23] emphasize SPMI modules, general quality administration, and BI-based compliance monitoring, while the activity monitoring studies [11], [12] address non-quality-assurance settings. Among the reviewed studies, none was found to describe an activity-centered dashboard for an LPM in a state Islamic higher education institution that combines activity status, schedules, and supporting documentation under role-based access. This study addresses that gap. Its contributions are: (i) a set of functional requirements and non-functional design requirements for LPM activity monitoring; (ii) an activity-centered dashboard design that links activity status, a schedule calendar, summary indicators, and documentation evidence under ADMIN and STAF roles; and (iii) a documented RAD implementation with functional verification through black box testing. The findings are limited to a single institution and should be read as a reference design rather than a generalized solution. Accordingly, this study addresses three research questions: (1) what functional requirements and non-functional design requirements were specified for a quality assurance activity monitoring dashboard for LPM UIN Sumatera Utara; (2) how can the dashboard be designed and implemented using RAD; and (3) to what extent does the implemented system satisfy the specified functional requirements in black box testing? The non-functional requirements are treated as design specifications and are not empirically evaluated in this study.

The objective of this study is to design and implement a web-based quality assurance activity monitoring dashboard for LPM UIN Sumatera Utara using RAD, with the aim of centralizing activity recording, monitoring, and reporting in a single role-based application, and to verify its functional behavior against the specified functional requirements through black box testing.

2. MATERIALS AND METHODS

2.1 Research Design

This study employed RAD, which emphasizes active user involvement and rapid iteration between development stages, as applied in web-based monitoring of fund disbursement files [13] and periodic supervision systems [14]. The RAD stages used in this study were as follows.

- a. Requirements Planning : identifying the needs of LPM UIN Sumatera Utara through observation and a literature review, and defining the scope of the activity monitoring dashboard.
- b. User Design : iteratively developing use case, activity, and class diagrams and interface mockups based on feedback from LPM stakeholders until the design was approved.
- c. Construction : implementing the agreed design as a backend REST API and a frontend Single Page Application (SPA).
- d. Cutover : testing system functionality, correcting identified errors, and handing over the system for use by LPM.

These stages allow repeated refinement between user design and construction without restarting the entire process. Figure 1 presents the RAD workflow used in this study.

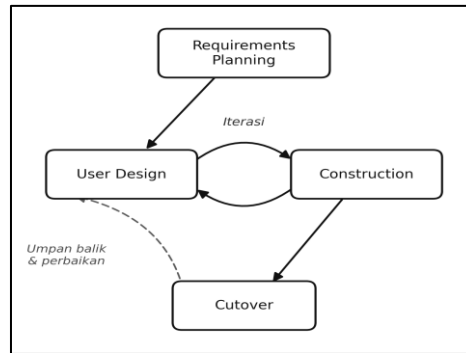


Figure 1. Stages of Rapid Application Development

Original diagram labels: Iterasi = iteration; Umpan balik & perbaikan = feedback and refinement.

RAD execution and user involvement. Feedback during user design is described as stakeholder review and consultation with LPM staff, not as a measured user study. The RAD cycle proceeded as follows: requirements planning fixed the scope of the dashboard; user design translated that scope into UML diagrams and interface mockups that were reviewed with LPM stakeholders and revised until accepted; construction implemented the accepted design; and cutover ended when identified errors had been corrected and the nine functional scenarios in Table 3 passed. Because feedback was obtained through informal consultation, the number of review cycles, the participants, and quantitative approval criteria were not formally recorded; this is acknowledged as a limitation in Section 3.4.

2.2 Data Sources and Participants

Research data were collected through two techniques. First, observation involved directly examining activity documents, archived reports, and existing reporting procedures at LPM UIN Sumatera Utara to identify activity types, workflows, and recording constraints.

The observation was exploratory and was carried out by the authors at LPM UIN Sumatera Utara; no fixed observation period, document sampling procedure, or selection criteria were applied, so the evidence base is qualitative and cannot be independently audited, which is acknowledged as a limitation. The recurring constraints identified were records dispersed across documents and spreadsheets, no consolidated view of activity status, and manual report compilation. These constraints were translated into requirements as follows: dispersed records led to centralized activity management (F-05) with search and filters (F-06); the missing overview led to the summary dashboard and calendar (F-08, F-09); supporting evidence kept in separate files led to documentation upload (F-07); and the involvement of several staff members and leadership led to authentication, profile, user, and category management with role-based access (F-01–F-04). The non-functional requirements were set from the same context: shared access to institutional records (NF-01, NF-05), non-technical users (NF-02), long activity lists (NF-03), and use on different devices (NF-04).

Second, the literature review examined previous research on RAD in web-based academic information systems [15] and monitoring dashboards informed by field observation [16] to guide feature and workflow design.

The study considered two user roles: ADMIN, representing LPM leadership responsible for categories, users, and overall activity monitoring; and STAF, representing staff members who record activities and upload documentation within their responsibilities. These identifiers are retained as implemented in the system. The role model follows the division of responsibilities observed at LPM and is presented as a design decision rather than a validated organizational model. Stakeholders other than these two groups, such as external assessors and report recipients, were not involved in validating the requirements or roles.

2.3 Requirements Analysis and Data Preparation

Observation and literature review findings were organized into functional and non-functional requirements, following the requirements classification approach used in a RAD-based correspondence management system [17]. Table 1 presents the functional requirements.

Table 1. System functional requirements

Code	Requirement	Description
F-01	User authentication	Users log in with a username and password and log out according to ADMIN/STAF access rights.
F-02	Profile management	Users view and update their own profile information.
F-03	User management	ADMIN creates, edits, and deletes accounts, with an assigned-activity check before deletion.
F-04	Category management	ADMIN creates, edits, and deletes activity categories; STAF has read-only access.
F-05	Activity management	Users create, edit, delete, and update the status of quality assurance activities.
F-06	Search and filters	The system provides search, filters, and pagination for the activity list.
F-07	Activity documentation	Users upload, view, download, and delete JPEG/PNG/PDF files up to 10 MiB per file.
F-08	Summary dashboard	The system displays annual summaries, monthly charts, and upcoming activities.
F-09	Activity calendar	The system displays activity schedules in a calendar view.

Table 2 presents the non-functional requirements. They are specified as design requirements (quality criteria) that guided the implementation; they were not empirically evaluated in this study, as summarized in Table 5 (Section 3.3).

Table 2. System non-functional requirements (design requirements)

Code	Aspect	Description
NF-01	Security	Passwords are stored as hashes, and API requests are designed to be validated using session tokens and role-based permissions.
NF-02	Usability	The interface is designed with navigation intended to be understandable to non-technical LPM users.
NF-03	Performance	Activity data are presented through filters and pagination to limit the number of records loaded at once; response time was not measured.
NF-04	Portability	The system is designed to be accessible through browsers on desktop and mobile devices.
NF-05	Data integrity	All user input is designed to be validated by the backend before being stored in the database.

Activity records were standardized into a consistent data structure comprising the activity name, category, responsible person, implementation date, and status to support dashboard visualization. The structure was implemented as a new database schema (Figure 4); no legacy LPM records were migrated, so no cleaning of missing, duplicate, or inconsistent historical values was required. The records shown in the screenshots were entered through the application.

2.4 Proposed System Architecture

The system uses a client–server architecture that separates the backend and frontend. The backend functions as a REST API that provides data through standard HTTP methods. The frontend is an SPA that loads a main page and updates its contents without reloading the entire page. This separation supports scalability and maintainability by allowing independent development of both components [18], consistent with service separation in RAD-based management systems [19]. The technologies used are described below.

- a. Backend — Node.js and TypeScript with Express, Prisma ORM, PostgreSQL hosted on Supabase, Zod for data validation, bcrypt for password hashing, Multer for file handling, Winston for logging, and Jest and Supertest for automated testing.

- b. Frontend — React with Vite as the build tool, React Router for navigation, Tailwind CSS for interface layout, Axios for REST API communication, and React Context and sessionStorage for authentication session management.
- c. Documentation storage — activity files are stored privately in Supabase Storage through a workflow consisting of a temporary upload URL request (upload-url), direct upload to Storage, and metadata validation (complete). This workflow was verified functionally (TC-07); security properties such as authorization robustness, file-content validation, and malicious-file handling were not tested.
- d. Versions and deployment — Exact versions of the runtime, frameworks, and database are recorded in the project's dependency manifests (package.json and lockfile), and the database and private file storage are hosted on Supabase. At the architecture level, the browser-based SPA sends requests through Axios to the REST API; the API applies session-token authentication, role-based permission checks, and Zod validation, and uses Prisma to read and write PostgreSQL, while documentation files are uploaded directly to Supabase Storage through signed upload URLs.

System design was modeled using Unified Modeling Language (UML), a visual modeling language for representing, designing, and documenting software analysis so that relationships between system elements are easier for developers and stakeholders to understand [20]. The following UML diagrams were used.

- a. Use case diagram : represents system behavior from the actors' perspective, showing how ADMIN and STAF interact with the activity monitoring dashboard functions.
- b. Activity diagram : represents process flows from initiation through conditional branches to completion, including activity recording and documentation upload.
- c. Class diagram : represents the static system structure, including the attributes and relationships of the main classes: User, Category, Activity, and Documentation.

Interface mockups were also prepared to represent feature layouts before implementation. Drawing on the dashboard design of a RAD-based web system [21], the dashboard presents annual activity summaries, monthly charts, upcoming activities, and an activity calendar adapted to the SPMI cycle at LPM. Specific interface decisions follow from the recording constraints observed at LPM: the summary dashboard and calendar provide the consolidated view that was missing when records were scattered across documents and spreadsheets; activity status and category fields give records a common structure; the documentation page keeps supporting files attached to the activity record; and role-based menus separate ADMIN management functions from STAF recording tasks. These decisions are design rationale; their effect on monitoring and reporting time was not measured.

2.5 Implementation Procedure

- a. Requirements Planning — document review and observation established the scope of authentication, activities, categories, users, documentation, the dashboard, and the calendar.
- b. User Design — UML diagrams and interface mockups were refined iteratively using feedback from LPM stakeholders until the design was approved.
- c. Construction — the backend REST API was implemented using Node.js, TypeScript, Express, and Prisma with PostgreSQL (Supabase). The frontend SPA was developed using React and Vite and communicated with the backend through Axios.
- d. Cutover — all features underwent black box testing, identified errors were corrected, and the system was prepared for use by LPM.
- e. 2.6 Evaluation Metrics

System evaluation focused on functional correctness using black box testing. Non-functional requirements (NF-01–NF-05) were not evaluated; the 100% result therefore describes functional test-case success only and not overall system quality.

Black box testing evaluates functional specifications without examining the internal code structure. Specified inputs are provided, and the resulting outputs are compared with the expected responses. This approach was used to assess the functional success of each feature across the defined test scenarios. Test cases were derived from the functional requirements F-01–F-09, with one test case per requirement; several cases combine multiple operations. Test cases were run against the completed

application before handover. A formal test log recording the browser, device, operating system, application version, and execution date was not kept, so these details are not reported; this limits reproducibility. The functional success rate was calculated as the number of passed test cases divided by the total number of executed test cases, multiplied by 100%. The evaluation checked whether the dashboard features met the specified functional requirements before handover to users. The result is interpreted as functional conformance for the executed scenarios only.

3. RESULTS AND DISCUSSION

This section presents the results according to the RAD stages: UML designs from user design, implemented features from construction, and functional testing outcomes from cutover, following the procedure described in Section 2.

3.1 User Design Results

The user design stage produced use case, activity, and class diagrams. These diagrams were refined iteratively based on the functional and non-functional requirements in Tables 1 and 2, documenting the intended actor behavior, process flows, and data structures before coding. The diagrams guided implementation during construction.

a. Use Case Diagram

The use case diagram shows the functions available to the two actors, STAF and ADMIN. Figure 2 presents the system use case design.

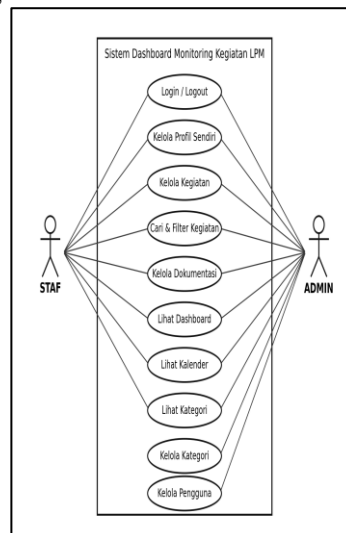


Figure 2. System use case diagram

Original diagram labels: Kelola = manage; Lihat = view; Profil Sendiri = own profile; Kegiatan = activities; Cari & Filter = search and filter; Dokumentasi = documentation; Kalender = calendar; Kategori = categories; Pengguna = users.

As shown in Figure 2, both actors can log in and out, manage their own profiles, manage activities, search and filter activities, manage documentation, view the dashboard and activity calendar, and read the category list. Category modification and user administration are restricted to ADMIN. Authorization is enforced by the backend, and unauthorized access to restricted management pages redirects STAF users to the dashboard.

b. Activity Diagram

The activity diagram represents the process flow within the system. One principal workflow is activity documentation upload, which involves the user, frontend, backend, and Storage, as shown in Figure 3.

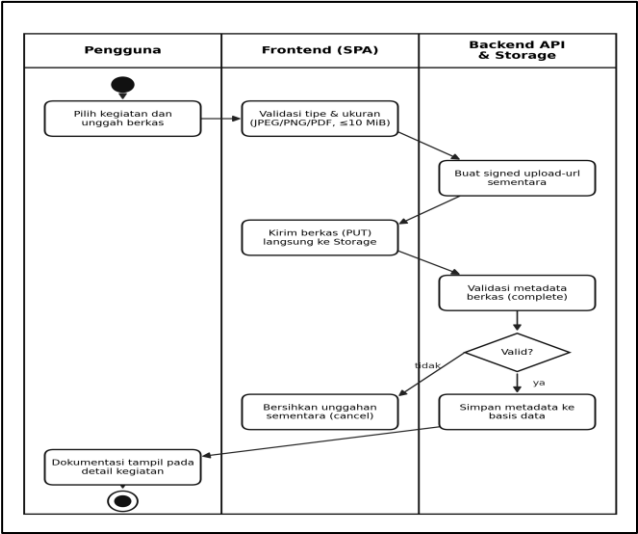


Figure 3. Activity diagram for documentation upload

The original diagram is retained; its upload workflow is described in English below. Pengguna denotes the user.

The workflow in Figure 3 begins when a user selects an activity and uploads a documentation file. The frontend first validates the file type and size, accepting JPEG, PNG, or PDF files up to 10 MiB. The backend then generates a temporary upload URL (upload-url), and the frontend uploads the file directly to Storage. During complete, the backend validates the file metadata. If valid, the metadata are stored in the database and the documentation appears on the activity detail page. Otherwise, cancel removes the temporary upload to prevent orphaned files in Storage.

c. Class Diagram

The class diagram represents the system data structure and relationships between classes. Figure 4 presents the design.

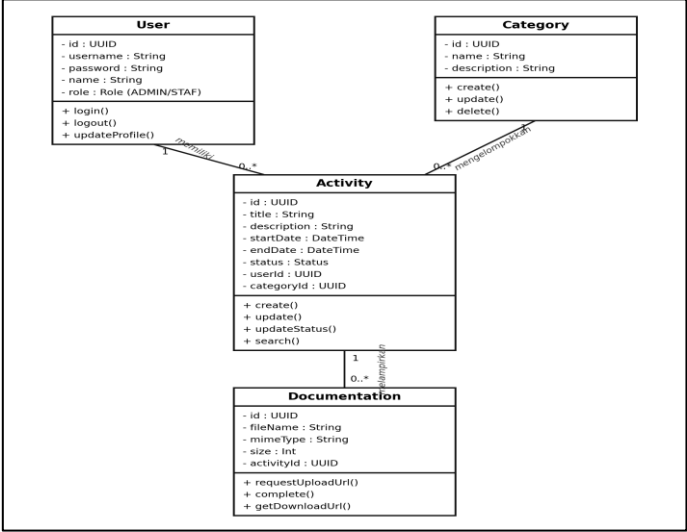


Figure 4. System class diagram

Figure 4 shows four main classes: User, Category, Activity, and Documentation. User stores account data and a role attribute that distinguishes ADMIN and STAF permissions. User and Activity have a one-to-many relationship: one user can be responsible for multiple activities, while each activity belongs to one user. Similarly, one Category can contain multiple activities, while each activity belongs to one category. Activity also has a one-to-many relationship with Documentation, allowing multiple files to be attached to one activity.

3.2 Construction Results

During construction, the use case, activity, and class diagrams were implemented as a backend REST API using Node.js, TypeScript, Express, and Prisma with PostgreSQL (Supabase), and a frontend

SPA using React, Vite, and Tailwind CSS, as described in Section 2.4. Functional requirements F-01–F-09 (Table 1) were implemented across the following nine web pages. Screenshots retain the original Indonesian interface used by LPM. The screenshots illustrate the implemented interface and are not evidence of usability, performance, or portability (NF-02–NF-04).

a. Login Page

This page provides access to the system. Users enter a username and password, which are validated through POST /users/login (F-01). Once the credentials are verified, the system issues a session token and redirects the user to the dashboard according to the ADMIN or STAF role.

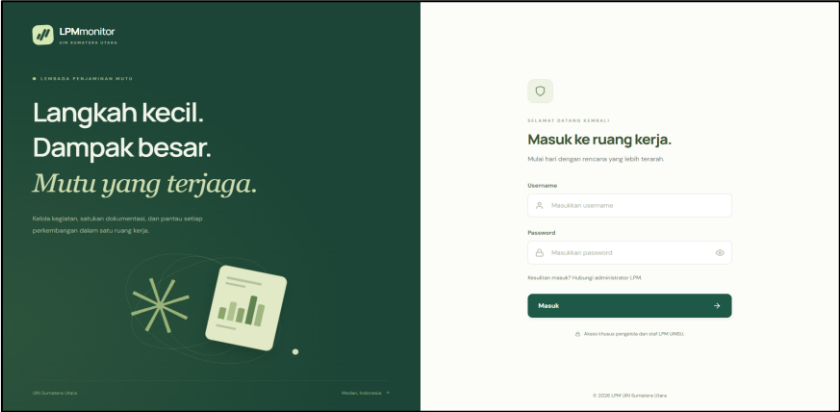


Figure 5. Login page

b. Profile Page

This page displays the account information of the signed-in user. Users can view and update their own profile and log out through GET, PATCH, and DELETE /users/current (F-02).

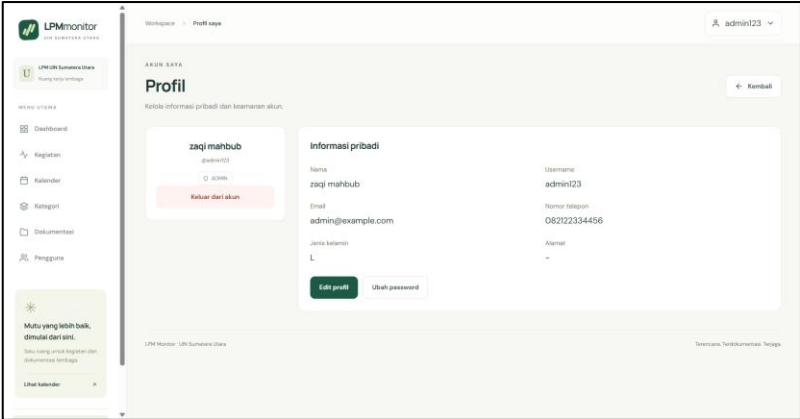


Figure 6. Profile page

c. Dashboard Page

The dashboard displays annual activity summaries, a chart of monthly activity counts, and upcoming activities. Data are retrieved through GET /dashboard (F-08), enabling leadership to review progress without opening individual activity records.

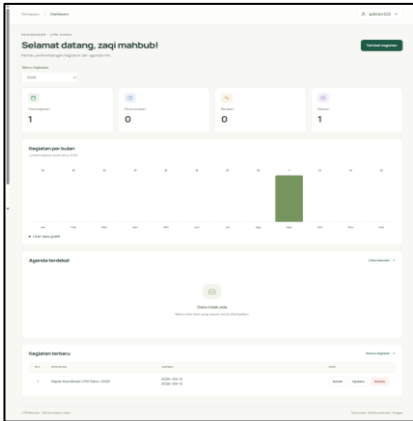


Figure 7. Dashboard page

d. Activities Page

This page displays quality assurance activities with search, filters, and pagination (F-06). Administrators and staff can create, edit, delete, and update activity status to planned, ongoing, or completed through the /activities endpoints (F-05).

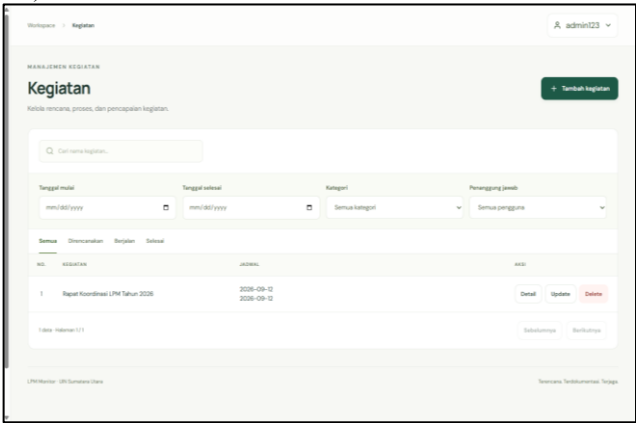


Figure 8. Activities page

e. Calendar Page

This page displays activity schedules in a monthly calendar, allowing users to identify ongoing and upcoming activities on specific dates. Calendar data are retrieved through GET /calendar (F-09).

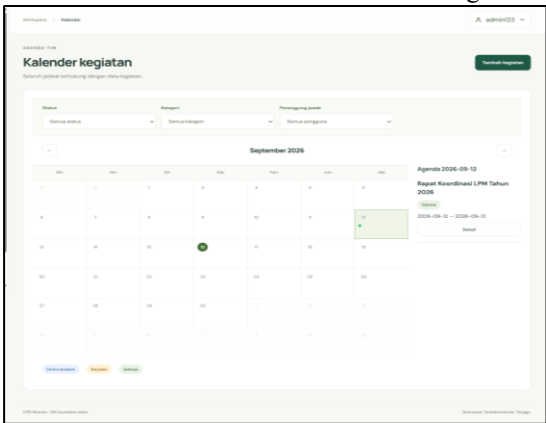


Figure 9. Calendar page

f. Categories Page

This page displays activity categories. ADMIN can create, edit, and delete categories through the category management endpoints, including POST and /categories/current/:id. STAF has read-only access through GET /categories/current (F-04).

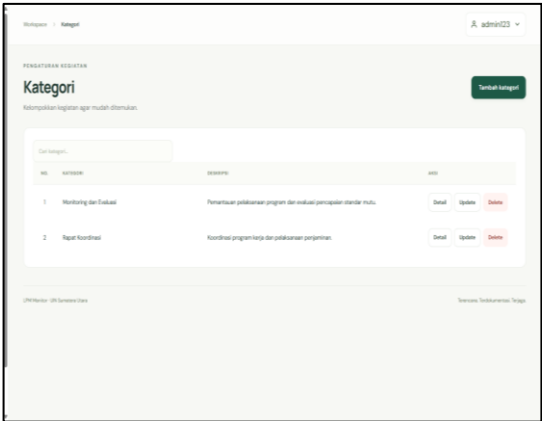


Figure 10. Categories page

g. Documentation Page

This page lists activities whose supporting documentation can be viewed or managed. Selecting an activity opens its documentation detail page.

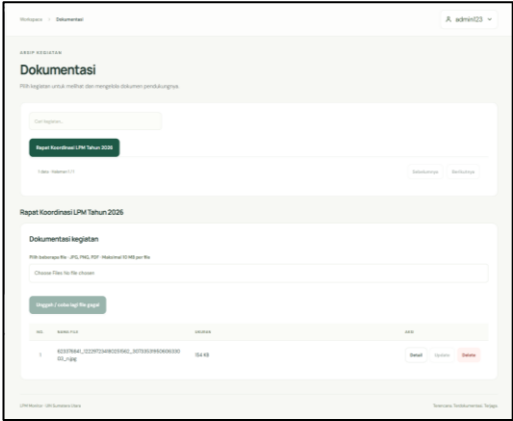


Figure 11. Documentation page

h. Documentation Detail Page

This page displays the documentation files associated with an activity. Users can upload JPEG, PNG, or PDF files up to 10 MiB, and view, download, or delete documentation (F-07). Uploads follow the upload-url, direct Storage upload, and complete workflow shown in Figure 3.

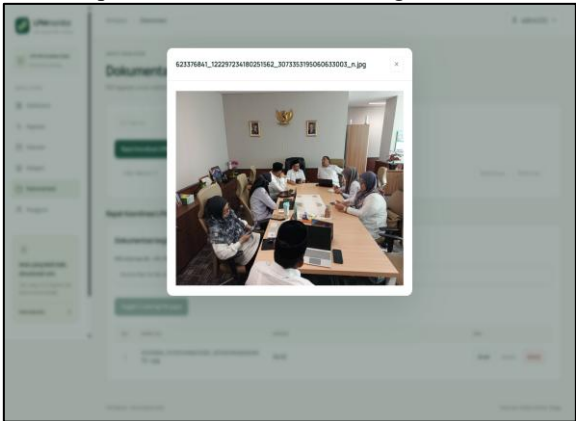


Figure 12. Documentation detail page

i. User Management Page

This page is accessible only to ADMIN for creating, editing, and deleting system accounts (F-03). Before deleting an account, the system checks its assigned activities through GET /users/:id/deletion-preview so that responsibility can be reassigned without losing activity data. STAF users attempting to access /users are redirected to the dashboard, consistent with the access restrictions in Figure 2.

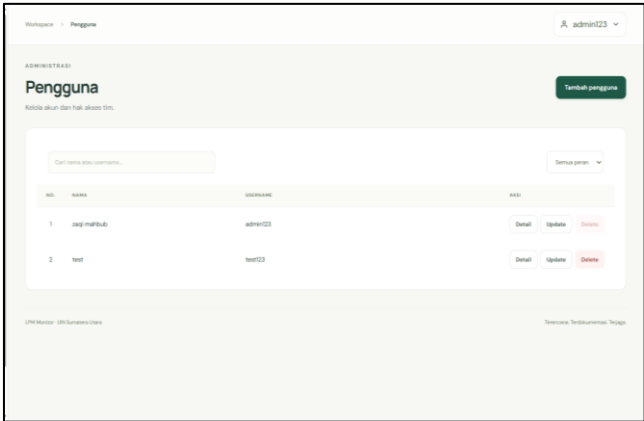


Figure 13. User management page

3.3 Cutover Results

During cutover, black box testing evaluated whether each feature met the functional requirements in Table 1. Each scenario was executed, and the system response was compared with the expected result. Table 3 presents the reported test outcomes.

Table 3. Black box testing results

Code	Requirement	Test scenario	Result
TC-01	User authentication	Log in with valid/invalid credentials, then log out.	Passed
TC-02	Profile management	View and update the signed-in user’s profile.	Passed
TC-03	User management	Create, edit, and delete accounts, including checking assigned activities before deletion.	Passed
TC-04	Category management	Create, edit, and delete categories as ADMIN; verify that STAF cannot modify categories.	Passed
TC-05	Activity management	Create, edit, delete, and update activity status.	Passed
TC-06	Search and filters	Search and filter activities by keyword, category, and status, including pagination.	Passed
TC-07	Activity documentation	Upload, view, download, and delete documentation, including file type and size validation.	Passed
TC-08	Summary dashboard	Display annual summaries and monthly charts consistent with stored records.	Passed
TC-09	Activity calendar	Display activities in the calendar according to their scheduled dates.	Passed

All nine test cases in Table 3 passed. Within these scenarios, authentication, profile management, user and category administration, activities, search and filtering, documentation, the dashboard, and the calendar behaved according to the functional requirements in Table 1.

Table 4 summarizes the test outcomes by functional group and reports the corresponding success rates.

Table 4. Summary of black box testing outcomes

Functional group	Test cases	Passed	Failed	Success rate
Authentication and profiles (TC-01–TC-02)	2	2	0	100%
Master data: users and categories (TC-03–TC-04)	2	2	0	100%
Activities and documentation (TC-05–TC-07)	3	3	0	100%
Monitoring: dashboard and calendar (TC-08–TC-09)	2	2	0	100%
Total	9	9	0	100%

All nine executed test cases passed, giving a functional success rate of 100% (9/9) and a failure rate of 0% (0/9). Each functional group achieved 100%: authentication and profiles (2/2), master data management

(2/2), activity and documentation management (3/3), and monitoring visualization (2/2). These percentages describe the reported test cases rather than exhaustive coverage of all possible system behavior.

Table 5 traces each requirement to its verification method and evidence. Functional requirements were verified through black box scenarios; non-functional requirements are design specifications that were not empirically validated.

Table 5. Requirements-to-evaluation traceability

Code	Requirement	Verification method	Evidence and status
F-01	User authentication	Black box testing	TC-01 passed
F-02	Profile management	Black box testing	TC-02 passed
F-03	User management	Black box testing	TC-03 passed
F-04	Category management	Black box testing	TC-04 passed
F-05	Activity management	Black box testing	TC-05 passed
F-06	Search and filters	Black box testing	TC-06 passed
F-07	Activity documentation	Black box testing	TC-07 passed
F-08	Summary dashboard	Black box testing	TC-08 passed
F-09	Activity calendar	Black box testing	TC-09 passed
NF-01	Security	Design specification; role restriction exercised only in TC-04	Not validated; no security testing performed
NF-02	Usability	Design specification	Not evaluated; screenshots are not evidence
NF-03	Performance	Design specification	Not evaluated; response time not measured
NF-04	Portability	Design specification	Not evaluated; no browser/device matrix
NF-05	Data integrity	Design specification; input validation partly exercised in TC-01 and TC-07	Not validated systematically

3.4 Discussion and Limitations

The 9/9 result shows that the implemented dashboard behaved as specified in the nine executed scenarios, which answers RQ3 at the level of functional verification. It should be read as successful verification of specified scenarios rather than as generalized validation of the system. Several limitations apply. First, test cases were derived from the functional requirements with one case per requirement, several cases combine multiple operations, and boundary and negative cases were limited to invalid credentials (TC-01), the STAF role restriction (TC-04), and file type and size validation (TC-07). A detailed test protocol listing preconditions, input data, expected results, observed results, and pass/fail criteria per operation was not produced, so Table 3 reports scenario-level outcomes only, and compound cases such as TC-03 and TC-07 may conceal partial failures. Second, the non-functional requirements were not evaluated: security (including authorization robustness and file-content validation), usability, performance, portability, and data integrity remain unverified. Third, no baseline workflow measurement was collected, so benefits such as faster or more accurate reporting cannot be claimed. Fourth, the study covers a single institution, so transferability to other LPM is a potential application and not an established generalization. Fifth, user involvement in the RAD process took the form of informal consultation and the observation was exploratory, so the requirements rest on qualitative evidence that was not formally documented. Future work should include a before-and-after comparison of reporting time and error rate, a usability evaluation with LPM users, performance and security testing, additional negative, boundary, and empty-state test cases, and a browser and device matrix for portability.

4. CONCLUSION

This study designed and implemented a web-based quality assurance activity monitoring dashboard for LPM UIN Sumatera Utara using four RAD stages: requirements planning, user design,

construction, and cutover. Use case, activity, and class diagrams guided development. The resulting system comprises nine web pages covering authentication, profiles, a summary dashboard, activities, a calendar, categories, documentation, documentation details, and user administration. All nine black box test cases (TC-01–TC-09) passed, indicating that the system met the specified functional requirements within the executed scenarios; this is a functional test-case success rate and does not validate overall system quality. The dashboard centralizes the recording, monitoring, and reporting of quality assurance activities in a single web application. Its design may be applicable as a reference for LPM at other state Islamic higher education institutions with similar recording problems, but this transferability has not been validated beyond the single LPM context. The non-functional requirements were not evaluated, and benefits such as faster reporting, improved usability, or better operational performance were not measured; these require further evaluation, for example through before-and-after task-time and error-rate comparison and usability and performance testing.

FUNDING

This research received no external funding

DATA AVAILABILITY STATEMENT

The data supporting the findings of this study are available from the corresponding author upon reasonable request

CONFLICT OF INTEREST

The author declares that there is no conflict of interest regarding the research, writing, or publication of this article

REFERENCES

- [1] A. W. Gafur, R. H. Dai, B. S. Rijal, I. R. Padiku, M. Polin, and A. Dwinanto, "Implementasi Sistem Informasi Audit Mutu Internal Berbasis Web di Universitas Negeri Gorontalo," *J. Autom. Comput. Inf. Syst.*, vol. 6, no. 1, 2026, doi: 10.47134/jacis.v6i1.147.
- [2] F. A. Ahmad and E. P. Silmina, "Implementasi Rapid Application Development (RAD) pada Pengembangan Dashboard Monitoring CCTV Kota Yogyakarta Menggunakan Framework Laravel," *JATI J. Mhs. Tek. Inform.*, vol. 9, no. 5, pp. 8141–8148, 2025, doi: 10.36040/jati.v9i5.14824.
- [3] Rasim et al., "Rancang Bangun Sistem Informasi Monitoring dan Analisis Kinerja Penjualan Menggunakan Metode Rapid Application Development (RAD)," *J. Inf. Syst. Informatics Comput.*, vol. 9, no. 2, pp. 266–274, 2025, doi: 10.52362/jisicom.v9i2.2078.
- [4] Y. M. Djaksana, "Analisa dan Perancangan Sistem Monitoring Inventory Berbasis Website dengan Metode Rapid Application Development," *SENTRI J. Ris. Ilm.*, vol. 4, no. 10, pp. 2341–2354, 2025, doi: 10.55681/sentri.v4i10.4772.
- [5] A. P. G. Cahyani and D. Mahdiana, "Systematic Literature Review Application of Methods in Information Systems Development," *J. Tek. Inform. (JUTIF)*, vol. 5, no. 4, 2024, doi: 10.52436/1.jutif.2024.5.4.1954.
- [6] D. Brinendo and A. M. Mayestino, "Rancang Bangun Aplikasi Pengaduan Pelanggan Menggunakan Metode RAD (Rapid Application Development)," *JATI J. Mhs. Tek. Inform.*, vol. 8, no. 2, pp. 1462–1469, 2024, doi: 10.36040/jati.v8i2.9017.
- [7] B. B. Aryandra and A. C. Wardhana, "Perancangan Sistem Informasi Desa Wisata Berbasis Website dengan Metode Rapid Application Development (RAD)," *JATI J. Mhs. Tek. Inform.*, vol. 8, no. 4, pp. 5936–5944, 2024, doi: 10.36040/jati.v8i4.9768.
- [8] R. F. Pahlevi, E. K. Permatasari, and M. S. Parikesit, "Pengembangan Sistem Informasi Penjaminan Mutu Online untuk Perguruan Tinggi Muhammadiyah/Aisyiyah Menggunakan Framework Laravel," *J. Tek. Inform. dan Sist. Inf.*, vol. 11, no. 4, pp. 208–219, 2024, doi: 10.35957/jatisi.v11i4.8338.
- [9] L. Rosyidi, E. Wibowo, and H. Tsabitah, "Perancangan Dashboard Sistem Informasi Penjaminan Mutu Perguruan Tinggi dengan Metode Design Thinking," *J. Riset Rumpun Ilmu Tek.*, vol. 4, no. 3, pp. 23–32, 2025, doi: 10.55606/jurritek.v4i3.5955.
- [10] J. Cevin, D. Gultom, and L. Hermawan, "Digitalisasi Sistem Penjaminan Mutu Internal (SPMI) untuk Institusi Pendidikan Berbasis Website," *MDP Student Conf.*, vol. 4, no. 1, pp. 313–320, 2025, doi: 10.35957/mdp-sc.v4i1.11183.

- [11] N. W. Rhomadhona, M. A. Muhammad, P. B. Wintoro, and Y. Mulyani, "Penerapan Metode Rapid Application Development untuk Sistem Informasi Event Berbasis Web pada Universitas Lampung," *J. Inform. dan Tek. Elektro Terap.*, vol. 13, no. 1, pp. 339–351, 2025, doi: 10.23960/jitet.v13i1.5597.
- [12] M. Al Hafidz, A. C. Puspitaningrum, M. S. Prasetya, and L. D. Fitriani, "Implementation of Web-Based Student Achievement Applications in Senior High Schools in Indonesia," *Sistemasi J. Sist. Inf.*, vol. 13, no. 4, pp. 1464–1475, 2024, doi: 10.32520/stmsi.v13i4.4143.
- [13] S. Supriyatna and S. Farizy, "Perancangan dan Implementasi Aplikasi Monitoring Berkas Pencairan Dana Berbasis Web Menggunakan Metode Rapid Application Development," *Sainstech*, vol. 34, no. 3, pp. 1–8, 2024, doi: 10.37277/stch.v34i3.2078.
- [14] A. P. Wibisono, L. Andrawina, and F. H. Hasanudin, "Rancangan Sistem Informasi Manajemen Pengawasan Berkala Menggunakan Metode Rapid Application Development," *Matrik J. Manaj. dan Tek. Ind. Produksi*, vol. 25, no. 2, pp. 131–142, 2025, doi: 10.30587/matrik.v25i2.8516.
- [15] E. Fitriani et al., "Penerapan Sistem Informasi Akademik Berbasis Web Menggunakan Metode Rapid Application Development," *J. Inf. Syst. Appl. Manag. Account. Res.*, vol. 8, no. 4, pp. 770–782, 2024, doi: 10.52362/jisamar.v8i4.1551.
- [16] A. P. Sari, M. M. Al Haromainy, and R. Purnomo, "Implementasi Metode Rapid Application Development Pada Aplikasi Sistem Informasi Monitoring Santri Berbasis Website," *Decode J. Pendidik. Teknol. Inf.*, vol. 4, no. 1, pp. 316–325, 2024, doi: 10.51454/decode.v4i1.348.
- [17] S. Dalis, S. Agustiani, S. Bahri, Wahyudin, and A. Prawikas, "Model Rapid Application Development (RAD) Untuk Perancangan Sistem Informasi Pengelolaan Surat (SIPERA) Pada Kelurahan," *IMTechno J. Ind. Manag. Technol.*, vol. 5, no. 1, pp. 31–37, 2024, doi: 10.31294/imtechno.v5i1.2448.
- [18] F. P. E. Putra, R. W. Efendi, A. B. Tamam, and W. A. Pramadi, "Tren dan Praktik Terbaik dalam Pengembangan Web Berbasis API: Kajian Literatur terhadap Framework Laravel dan React," *Infomatek*, vol. 27, no. 1, pp. 165–178, 2025, doi: 10.23969/infomatek.v27i1.25122.
- [19] N. P. Lestari and L. L. Utami, "Pengembangan Sistem Manajemen Bengkel Menggunakan Laravel Dengan Metode Rapid Application Development (RAD)," *J. Inform. dan Tek. Elektro Terap.*, vol. 13, no. 2, pp. 1132–1141, 2025, doi: 10.23960/jitet.v13i2.6413.
- [20] Y. Aryani, I. Aqil, and B. Paramita, "Penerapan Unified Modeling Language (UML) pada Digitalisasi Sistem Informasi Perpustakaan," *Digit. Transform. Technol.*, vol. 4, no. 2, pp. 1032–1040, 2025, doi: 10.47709/digitech.v4i2.5153.
- [21] M. A. Senubekti, M. A. Kartawidjaja, and G. B. W. Adyana, "Penerapan Metode Hybrid Waterfall-RAD pada Sistem Antrian Web untuk Optimalisasi Layanan di Pengadilan Negeri Bale Bandung," *JOINS J. Inf. Syst.*, vol. 10, no. 1, pp. 60–70, 2025, doi: 10.33633/joins.v10i1.12668.
- [22] U. Radvilaitė and D. Kalibaitienė, "Integrating rapid application development courses into higher education curricula," *Appl. Sci.*, vol. 15, no. 6, Art. no. 3323, 2025, doi: 10.3390/app15063323.
- [23] A. Sorour et al., "Big data challenge for monitoring quality in higher education institutions using business intelligence dashboards," *J. Electron. Sci. Technol.*, vol. 22, no. 1, Art. no. 100233, 2024, doi: 10.1016/j.jnlest.2024.100233.