WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

SIET CSE EE Project Portal: A role-based web application for academic project management and evaluation

Mrs. S.Nandhini[1], M.Vijay Rathinam[2], R.Siva Darshan[3], Tarunika Rajagopal[4], N.Vishwanath[5]

Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamilnadu, India

A B S T R A C T:

Managing final-year and experiential-engagement (EE) projects in an engineering department commonly depends on spreadsheets, messaging groups, e-mail threads and loose documents, so that team records, guide allocations, title approvals, weekly reports and evaluation marks become scattered and difficult to trace. This paper presents the SIET CSE EE Project Portal, a centralized web application that manages the academic project lifecycle for the Computer Science and Engineering department of Sri Shakthi Institute of Engineering and Technology. The portal serves five roles — student, project guide, class advisor, head of department (HOD) and administrator — and implements team formation with a four-member limit, guide allocation, project-title approval and rejection with mandatory reasons, weekly report submission, faculty feedback, submission history and manually entered evaluation marks. The client is a React, TypeScript, Vite and Tailwind CSS single-page application; the server is a Python ASGI application served by Uvicorn and organised into routers, services and repositories, with a relational database whose schema is versioned with Alembic. The client is hosted on Vercel and the server on Render. Role-based access control is enforced in the backend so that permissions hold even when the interface is bypassed. The paper documents the design and implementation, defines the acceptance tests for every business rule and records the limitations found in the report, notably demonstration login shortcuts, an unformatted empty-score display and a pending deployment verification. No user study or load test has yet been conducted.

Keywords: Academic project management; Role-based access control; React; TypeScript; Python ASGI; REST API; Team allocation; Project evaluation; Web application

Introduction

Project work is a central part of an engineering curriculum, and it demands coordination among many people. Students form teams, advisors allocate guides, guides approve titles and review progress, the head of department monitors the whole department, and administrators maintain faculty records. When these activities are carried out through spreadsheets, messaging groups and e-mail, information is duplicated, decisions are difficult to trace, and evaluation records are kept apart from the work they describe. Institutional learning-management and ERP systems usually cover courses, attendance and examinations, but rarely the project-specific rules of a department (Sandhu et al., 1996; Fielding, 2000).

Two practical needs shape such a system. First, different people must see and change different data: a student should see only their own team, a guide only the teams assigned to them, and only an advisor should enter marks. Second, the rules of the process — team size, one team per student, a reason for every rejected title — must be enforced by the system and not left to manual checking. This paper describes the SIET CSE EE Project Portal, a web application developed for the Computer Science and Engineering department of Sri Shakthi Institute of Engineering and Technology to meet these needs. The central idea is one application, centralised project records and different permissions for each user role. The contributions of the paper are:

- a full-stack reference implementation that separates a role-aware React client, a layered Python API and a relational database with versioned migrations;
- a specification of the department's business rules (Section 3.7) written so that each rule can be tested at the backend and not only displayed as an interface warning;
- a deployment design that hosts the client and the API separately and lists the configuration dependencies this creates; and
- an honest account of what has and has not been verified, including the corrections required before an empirical evaluation.

Because the portal is a student project, the paper deliberately reports what the design and report state, and does not claim measurements that were not made. Section 2 reviews related work, Section 3 describes the design and methodology, Section 4 the implementation and Section 5 the verification. Section 6 discusses the findings, and Sections 7–9 present limitations, future scope and conclusions.

1.1. ***Problem statement***

Given the students, faculty and project guides of one department and one academic cycle, the system should (i) let an advisor form teams of at most four students and allocate a guide to each; (ii) let a team propose a title that its guide can approve or reject with a recorded reason; (iii) collect weekly reports and feedback in an auditable history; and (iv) let advisors record evaluation marks — while ensuring that every user can perform only the operations that belong to their role.

1.2. ***Objectives***

The objectives of the project were to:

- digitise academic project administration for the department;
- form teams of at most four students and allocate project guides;
- support title submission, approval and rejection with recorded reasons;
- collect weekly reports and project documents and retain submission and feedback history;
- allow advisors to enter and edit evaluation marks manually; and
- provide separate interfaces and enforced permissions for the five user roles.

Literature review

1.3. Academic project management systems

Learning-management and ERP platforms generally manage courses, attendance and examinations. Project-specific workflows — limits on team size, guide allocation, title approval with reasons and weekly review cycles — are typically handled outside them, in spreadsheets and messaging groups that offer no role-specific views and no audit trail. The portal targets this gap with a workflow modelled on the department's own process.

1.4. Role-based access control

Role-based access control (RBAC) assigns permissions to roles and gives users permissions by role membership, which reduces the administrative effort of managing many individual users (Sandhu et al., 1996). The model suits an academic department whose responsibilities are clearly divided. The portal applies RBAC at two levels: route guards in the client decide which pages a user sees, and authorization checks in the backend decide which operations are accepted.

1.5. Single-page applications and typed components

Component-based libraries such as React divide an interface into reusable units, so the same table, filter or status badge can serve many pages (Meta Open Source, n.d.). TypeScript adds static types to components and API data, allowing some errors to be found before runtime. Vite provides fast development builds and optimised production bundles (Vite Team, n.d.), and Tailwind CSS supplies utility classes for consistent, responsive styling (Tailwind Labs, n.d.).

1.6. REST APIs and layered backends

Representational State Transfer (REST) describes an architectural style in which resources are manipulated through a uniform set of HTTP operations (Fielding, 2000). Layered backends separate request handling, business rules and data access, which improves testability and maintainability. The ASGI specification allows Python applications to run on asynchronous servers such as Uvicorn (ASGI documentation, n.d.; Uvicorn, n.d.).

1.7. Authentication and security

Token-based authentication represents a user's claims in a compact, signed token that the server verifies on every request (Jones et al., 2015), and bearer tokens must be protected in transit (Jones & Hardt, 2012). Passwords should be stored with salted hashing, secrets kept out of the client, and authorization decided on the server, because any check performed only in the browser can be bypassed.

1.8. Schema migration and deployment

Migration tools such as Alembic version a database schema alongside the code so that environments can be created and upgraded reproducibly (Alembic, n.d.). Hosting the client and the API on separate platforms, here Vercel and Render (Vercel, n.d.; Render, n.d.), lets each be updated independently, at the price of extra configuration such as the API base URL and cross-origin resource sharing (CORS) rules.

1.9. Research gap

Published work on RBAC, REST design and web deployment is general; little describes a complete, small-scale system built around the project-allocation and evaluation rules of an engineering department, together with an account of its own limitations. The portal is a first step in this direction, and this paper treats its verification as incomplete.

System design and methodology

1.10. Design principles

The design follows five principles, which are stated in the project report:

- *Single source of truth.* Teams, guides, titles, submissions, feedback and marks are held in one database.
- *Least privilege.* Each role sees and changes only the data its responsibilities require.
- *Backend enforcement.* Permissions and business rules are enforced by the API, never by the interface alone.
- *Traceability.* Decisions, feedback, submissions and supported actions are recorded with their history.
- *Separation of concerns.* Presentation, business logic and persistence are separate layers that can be changed independently.

1.11. Users and responsibilities

The portal defines five roles, summarised in Table 1. The report's abstract emphasises the student, advisor and guide workflows; the HOD and administrator roles complete the permission model.

Table 1 - User roles and their principal responsibilities.

Role	Principal responsibilities
Student	View team and assigned guide; submit project titles; upload weekly reports and documents; read feedback; view previous submissions
Project guide	View assigned teams; approve or reject titles with reasons; review weekly submissions; give feedback; view history
Class advisor	Create and manage teams; assign guides and manage allocations; browse student and guide directories; enter and edit marks
HOD	Review department-wide status; filter student and team records; monitor weekly submissions and review status; access action history
Administrator	Register faculty; maintain names, institutional e-mails and designations; assign roles such as advisor or guide

1.12. System architecture

Fig. 1 shows the three-tier architecture. The browser client is a React single-page application with a separate layout and set of pages for each role. Every user action becomes a REST request to the Python API, which authenticates the caller, authorises the action, validates the data and applies the business rules. Persistent records are stored in a relational database and its schema is versioned with Alembic migrations. The browser never accesses the database directly and never holds database credentials. Table 2 summarises the technology stack.

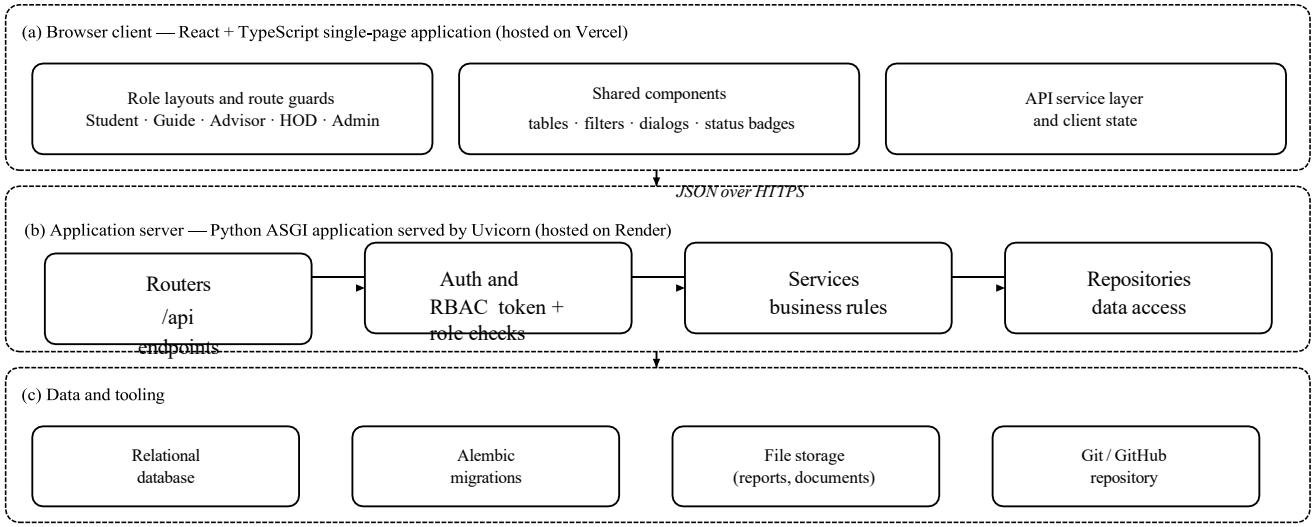


Fig. 1 - System architecture of the SIET CSE EE Project Portal.

Table 2 - Technology stack of the portal.

Layer	Technology	Role in the portal
Client interface	React.js, TypeScript (TSX), Tailwind CSS	Single-page application with role-specific portals
Build tooling	Vite	Development server and production build
Application server	Python ASGI application; Uvicorn	REST API, authentication, business rules
Persistence	Relational database; Alembic	Users, teams, titles, submissions, feedback, evaluations; schema migrations
Version control	Git; GitHub	Source tracking and shared repository
Hosting	Vercel (client); Render (API); Docker	Independent deployment of each tier; containerised backend

1.13. Frontend design

The client is organised in layers: an application entry point; routing; route guards that restrict pages by authentication and role; role-specific layouts with their own navigation; shared components (tables, filters, dialogs, status badges); an API service layer that wraps all HTTP calls; and state management for session information, fetched records and local interface state. Table 3 lists the pages of each portal.

Table 3 - Main pages of each role portal.

Portal	Main pages
Student	My Team, Submission, My Submission
Guide	Approve Submissions, My Teams, History
Advisor	Students, team management and guide-allocation controls
HOD	Advisors, Students, Weekly Submissions, Action History
Administrator	Home, Advisors, Guides, Students

1.14. Backend design and request processing

The API entry point is app.main:app. The code base is divided into routers (endpoints for authentication, teams, submissions and marks), services (business rules), repositories (database reads and writes), an auth package (password hashing, token handling and role checks), database configuration, Alembic migrations, background-job and worker modules, data models, request schemas and configuration, together with tests, a requirements file and a Dockerfile. A request follows the sequence in Fig. 2: the commit step is reached only if every earlier step succeeds, so a failed operation leaves no partial change.

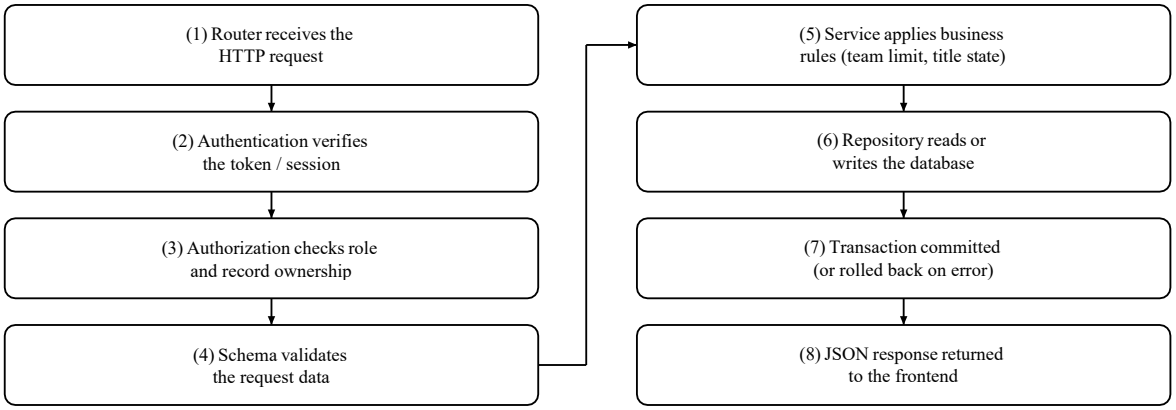


Fig. 2 - Request-processing pipeline of the backend API.

1.15. Data model

The logical data model contains the entities in Table 4. The entities are derived from the described workflows; the physical table and column names must be taken from the database models.

Table 4 - Logical entities of the data model.

Entity	Purpose
Users	Student and faculty identities and account details
Teams / team members	Team identifiers and the relationship between students
and teams Guide allocations	Relationship between teams and faculty guides
Project titles	Proposed titles and their approval state
Submissions	Weekly reports and documentation metadata
Feedback	Guide comments and review outcomes
Evaluations	Marks, rubric values and remarks
History records	Records of supported changes and actions

1.16. Business rules

The portal enforces the following rules in the backend:

- A team has at most four students, and a student belongs to at most one team in an academic cycle. Team identifiers are sequential and are never reused.

- Guides can access only their assigned teams; advisors manage teams, allocations and marks; students submit titles and reports.
- A title is approved or rejected by the assigned guide. A rejection requires a reason, and a rejected title can be revised and resubmitted. Approved titles are locked against the allocation changes defined in the workflow.
- Marks are entered and edited manually by advisors and are not calculated automatically from progress percentages.
- Student accounts use the institutional domain @srishakthi.ac.in and teacher accounts use @siet.ac.in; teacher roles are distinguished and not treated as a single faculty role.

Implementation

1.17. Client application

The client is written in TypeScript with React and bundled by Vite; Tailwind CSS provides layout, spacing and responsive behaviour. Each role has its own layout and navigation, and route guards redirect a user who is not signed in or lacks the required role. Reusable components render student directories, team records and submission lists. The API service layer builds every request, so pages do not construct HTTP calls themselves.

1.18. Server application and API

The server is a Python ASGI application started with Uvicorn on the port supplied by the hosting platform:

```
uvicorn app.main:app --host 0.0.0.0 --port $PORT
```

The client and server exchange JSON through REST-style endpoints (Table 5). The table shows the representative operations described in the report and not a verified list of endpoint paths.

Table 5 - Representative REST operations.

HTTP method	Purpose	Example in the portal
GET	Retrieve data	Load the teams assigned to a guide
POST	Create or submit data	Create a team; submit a title or weekly report
PUT / PATCH	Update a resource or selected fields	Edit marks; change an allocation
DELETE	Remove a record	Remove a student from an unlocked team

1.19. Data persistence and migrations

A relational database stores all persistent records, so that team assignments, approved titles, reports and marks survive an application restart. Alembic migration scripts create and upgrade the schema consistently in development and deployment. The exact database engine and object-relational mapper are not stated in the report and must be confirmed from the project configuration before submission.

1.20. Authentication and security

Users sign in with institutional accounts. The backend issues tokens, verifies them on every protected request and then checks the caller's role and ownership of the requested record. Passwords are stored with salted hashing; secrets and database credentials are kept in environment variables and never reach the client. CORS is configured to allow only the intended client origin, and multi-step operations run inside database transactions. Because authorisation is enforced on the server, a user who bypasses the interface still cannot perform actions outside their role. This describes the intended design; it cannot be taken as proof that every measure is implemented correctly in the running application.

1.21. Example workflow: weekly report submission

A student signs in; the client checks that required fields are filled; the report is sent to the API; the backend verifies the session, role, team ownership and submission rules; valid data is saved in the database and file storage; and the client displays the submission status. The assigned guide then reviews the report and records feedback, which the student can read together with the submission history.

1.22. Deployment

The source code is kept in a Git repository on GitHub. The client is built and hosted on Vercel; the API runs on Render, optionally as a Docker container, with its environment variables set in the platform. Deployment proceeds by pushing the code, configuring the backend variables, applying migrations, confirming that Uvicorn starts, setting the API URL in the client, configuring CORS for the client domain, deploying the client and then testing login, team management, title approval, submissions and evaluation in the deployed environment. Separate hosting allows independent updates, but an incorrect API URL, CORS setting or database connection can break the application even when the client loads.

1.23. Implementation status

Table 6 summarises the status of each feature as stated in the project report.

Table 6 - Feature implementation status.

Feature	Status	Notes
Institutional login and role-based portals	Implemented	Five roles; separate layouts and route guards
Team creation and guide allocation	Implemented	Four-member limit; one team per student per cycle
Title submission, approval and rejection	Implemented	Rejection requires a reason; resubmission allowed

Weekly reports and documents	Implemented	Submission history retained
Guide feedback	Implemented	Visible to the student with history
Manual evaluation by advisor	Implemented	Marks entered and edited manually
HOD monitoring and action history	Implemented	Department-wide filters and history
Administrator faculty registration	Implemented	Names, e-mails, designations, roles
Demonstration login buttons	Open issue	To be removed or disabled for production
Empty evaluation display	Open issue	Shows NaN; should show Pending or a dash
Deployment verification screenshot	Pending	Not yet captured

Testing and verification

This section reports verification honestly. The report defines functional tests but states that results from the final build must still be recorded; no pass or fail result is therefore claimed here.

1.24. Acceptance test plan

Table 7 lists the acceptance tests derived from the documented requirements. They should be run on the deployed build for all five roles, and the actual result and status recorded before submission.

Table 7 - Acceptance tests (results to be recorded on the final build).

Test	Expected result	Actual	Status
Valid institutional login	User reaches the correct portal	[ ]	[ ]
Invalid credentials	Access denied with an error	[ ]	[ ]
Create a team with four students	Team creation succeeds	[ ]	[ ]
Add a fifth student	Request rejected	[ ]	[ ]
Allocate one student to two teams	Conflicting allocation rejected	[ ]	[ ]
Guide opens an unassigned team	Access denied	[ ]	[ ]
Reject a title without a reason	Request rejected	[ ]	[ ]
Reject a title with a reason	Decision and reason recorded	[ ]	[ ]
Submit a weekly report	Submission appears in relevant records	[ ]	[ ]
Guide gives feedback	Feedback visible to the student	[ ]	[ ]
Advisor edits marks	Updated marks persisted	[ ]	[ ]
Page with no rubric scores	Pending or a dash shown, not NaN	[ ]	[ ]

1.25. Aspects not yet evaluated

The following have not been carried out, and no results for them are claimed:

- user studies, usability tests or surveys of students, guides and advisors;
- measurement of time saved against the manual process;
- end-to-end tests of the deployed client, API and database, latency measurements and load tests;
- automated tests of the backend authorisation rules; and
- formal accessibility, security and privacy assessments.

Discussion

1.26. What the portal demonstrates

The portal shows that the project lifecycle of a department can be managed in one role-based application built from widely used open tools. Its main design decision is to enforce authorisation and academic rules in the backend, so that they hold regardless of how a request is made. The router–service–repository structure, typed client and versioned migrations make the system easier to test and extend than a set of loosely connected documents.

1.27. Comparison with the manual process

Table 8 - Existing manual process compared with the portal.

Existing manual process	SIET Project Portal
Team records in separate documents	Centralised team records
Guide assignments tracked manually	Digital guide allocation
Title approval through informal messages	Recorded approval and rejection workflow
Reports spread across platforms	Organised submission history
Feedback difficult to trace	Feedback linked to submissions
Evaluation kept separately	Manual marks management in the portal
Access based on shared files and messages	Role-based access and permissions

The expected benefit is better organisation, traceability and coordination. The amount of time saved has not been measured, and a timing study or user survey would be a useful addition.

1.28. Security considerations

Authentication answers who the user is; authorisation answers what the user may do. The portal's design addresses both, but three items need attention before production use. First, the quick-fill demonstration login buttons expose sample-account access. Second, empty evaluation values appear as NaN, which misreports missing marks. Third, confidential student information must not appear in logs or API responses, and rate limiting and stronger faculty authentication are not yet in place.

1.29. Deployment dependencies

Separating the tiers creates configuration dependencies. The client must know the correct API URL, the API must allow the client origin through CORS, and the API must reach its database. A fault in any of these can make the application unusable even though the client page loads, which is why the report calls for a screenshot and a test of the deployed system.

Limitations

The principal limitations of the current version are:

- acceptance-test results on the final build are not yet recorded;
- the database engine, ORM and exact Python framework are not stated in the report;
- demonstration login shortcuts and the NaN display of empty evaluations remain open;
- no notifications, deadline management, reporting or analytics;
- no similarity checking, version history or strict validation of uploaded documents; and
- no integration with institutional student-record or examination systems.

Future scope

- *Notifications and reminders* for title decisions, feedback, deadlines and reviews, by e-mail or in-app alerts.
- *Report generation* of team lists, project summaries, evaluation reports and department-level reports.
- *Analytics for the HOD* on submission status, guide allocation and review completion.
- *Stronger security and audit logs*: improved session management, login rate limiting, stronger faculty authentication and traceable records of important actions.
- *Document management*: file validation, version history and project-report similarity checking.
- *Accessibility and integration*: better mobile accessibility and integration with institutional student records and examination systems; deadline management and advanced search.

Research extensions include a usability study with students and faculty under institutional approval, a timing comparison with the manual process, and automated tests of every authorisation rule at the API level.

Conclusion

The SIET CSE EE Project Portal shows that the academic project lifecycle of a department — team formation, guide allocation, title approval, weekly submissions, feedback and manual evaluation — can be managed in a single role-based web application. A React and TypeScript client, a layered Python API served by Uvicorn, a relational database with versioned migrations and separate Vercel and Render deployments replace scattered documents and messages with one traceable system of record. Its strengths are enforced backend authorisation, an explicit set of business rules and a maintainable architecture. Its open items are equally clear: demonstration logins, the NaN display, unrecorded test results and an unconfirmed deployment. With these corrected, the portal would provide a sound basis for evaluation with real users.

Acknowledgements

[Placeholder — acknowledge the project guide, the department, the institution and any funding.]

REFERENCES

- Alembic. (n.d.). *Alembic documentation*. https://alembic.sqlalchemy.org
- ASGI documentation. (n.d.). *ASGI specification*. https://asgi.readthedocs.io
- Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* [Doctoral dissertation, University of California, Irvine]. Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc7519 Jones, M., & Hardt, D. (2012). *The OAuth 2.0 authorization framework: Bearer token usage* (RFC 6750). Internet Engineering Task Force.
- https://www.rfc-editor.org/rfc/rfc6750
- Meta Open Source. (n.d.). *React documentation*. https://react.dev Render. (n.d.). *Render documentation*. https://render.com/docs
- Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *IEEE Computer, 29*(2), 38–47. Tailwind Labs. (n.d.). *Tailwind CSS documentation*. https://tailwindcss.com
- Uvicorn. (n.d.). *Uvicorn documentation*. https://www.uvicorn.org Vercel. (n.d.). *Vercel documentation*. https://vercel.com/docs Vite Team. (n.d.). *Vite documentation*. https://vite.dev