Students Voice And Support System: A Web-Based Grievance Management Platform For Academic Institutions

S. Shankar[1], Monika M[2], Nihitha B[3], Santhiya R[4], Shanjini R S[5]
[1]Assist prof, Dept of Computer Science and Engineering
[2, 3, 4, 5] Dept of Computer Science and Engineering
[1, 2, 3, 4, 5] Vivekanandha College of Technology for Women, Namakkal, Tamil Nadu, India

Abstract- *Grievance handling in most academic institutions still depends on paper-based, informal processes that offer students no transparency or follow-up. The Students Voice and Support System (SVSS) is a web-based platform built using Python, Django, MySQL, and JavaScript that digitizes the entire complaint lifecycle. Students submit grievances online and instantly receive a unique Ticket ID for tracking. They can monitor the status of their complaints at any stage, attach supporting files, receive email updates, and contribute to a community suggestion board through anonymousupvoting. A dedicated administrator interface provides real-time statistics, category-wise complaint breakdowns, and tools to update complaint status and post official responses. The system supports anonymous filing, is fully responsive on both mobile and desktop devices, and incorporates security measures, including CSRF protection and password hashing. Testing confirmed that SVSS fulfils all functional requirements reliably and outperforms six comparable systems in the literature by being the only solution to simultaneously offer full lifecycle management, low infrastructure cost, mobile accessibility, and anonymous submission support.*

Keywords: complaint management system, Django framework, student grievance portal, role-based access control, MySQL, web application, ticket tracking

I. INTRODUCTION

Students in engineering colleges across Tamil Nadu and the broader Indian higher education system deal with daily service failures, such as erratic bus schedules, neglected hostel facilities, and classroom equipment that remains broken for weeks. These issues rarely appear inthe formal records. Most students stay silent, not because they are indifferent, but because the channels available to them — speaking to a faculty member, writing a letter, or dropping a note in a suggestion box — offer no acknowledgement, no tracking, and no visible outcome. When raising concerns produces nothing, students stop doing so. This silence carries a real institutional cost: without a documented grievance record, administrators cannot identify recurring failures, justify targeted maintenance spending or demonstrate student welfare monitoring to accreditation authorities.

Consider a hostel water problem reported informally by ten different students on ten separate occasions. Because it was neverlogged, it never existed from an administrative perspective. The Students Voice and Support System (SVSS) was developed to bridge this gap. It is a web application that allows students to file grievances directly and equips administrators with the tools requiredfor systematic responses. When a complaint is submitted, the SVSS assigns a Ticket ID and saves the record in the database. The administrator can view, update its status, assign it to the relevant department, and post a formal response. Throughout this process, students can check their progress from any browser at any time without having to visit a single office.

The remainder of this paper is organized as follows.

Section II reviews the related work and identifies the limitations that motivated this study. Section III describes the proposed system, its architecture, and key design decisions. Section IV discusses the hardware and software requirements of the proposed system. Section V presents the UML-based system design of the proposed model. Section VI explains each modulealong with the system interface screenshots. Section VII presents the verification and validation results. Section VIII presents a comparative analysis.Finally, Section IX concludes with a discussion of future directions.

II. LITERATURE SURVEY

Research on digital grievance and feedback management for academic institutions has grown considerably over the past five years, with proposals spanning deep-learning classifiers, blockchain ledgers, and mobile-first applications. Each contribution advances the field in some way, yet a closer look at the literature reveals a recurring disconnect: technically sophisticated systems that most real-

world institutions cannot afford or maintain, and simpler systems that cover only part of the grievance workflow.

Kumar and Reddy (2026) introduced the most technically ambitious solution examined here — a platform using LSTM and CNN models to automatically classify complaints and assign priority scores through sentiment analysis. The concept is compelling in theory, but classification accuracy depends heavily on the volume and quality of historical training data. A college deploying this platform for the first time would start with almost no historical complaint data, meaning that the models would perform poorly at the exact moment when institutional buy-in was most critical.

Joseph and Shah (2025) built a real-time monitoring platform with automated SMS and email notifications and multi-tier administrator access. The analytics functionality is genuinely useful, but the reported infrastructure cost is substantial, and the system was not designed for mobile browsers — a notable limitation given that smartphones are the dominant means of Internet access for most students in Indian colleges.

Prakash and Nair (2024) pursued tamper-proof record-keeping through blockchain technology and, employed smart contracts to enforce complaint workflows and prevent record alterations. While the security argument holds in principle, the operational overhead of running a distributed ledger alongside conventional institutional databases, combined with the rarity of blockchain expertise in academic IT teams, creates adoption barriers that outweigh security gains in everyday campus grievance handling.

Sharma and Mehta (2023) drew on e-governance frameworks to design a hierarchical complaint system with automated escalation timers and complete audit trails. The architecture shows a clear understanding of large bureaucratic institutions, but it translates poorly to smaller private colleges with flatter administrative structures, where rigid workflow assumptions and complex interfaces add friction without adding value.

Gupta and Rao (2022) developed a feedback collection platform for higher education with well-structured forms and a thoughtfully implemented anonymous submission option. The critical gap is the absence of any complaint lifecycle: there is no Ticket ID, no status tracking, and no way for a student to find out what action, if any, was taken. Collecting feedback without closing the loop addresses only half of what institutions need.

Patel and Iyer (2023) shifted the interface to the smartphone, building native applications for both Android and iOS with push notification support. Maintaining two separate codebases increases both the development effort and the ongoing maintenance costs. Students using older or low-specification devices may miss notifications entirely, and exclusive cloud-based storage raises data residency concerns for institutions with local data governance obligations.

Together, these studies point to a consistent and unresolved gap in the literature. Every existing solution either demands a costly infrastructure, trades practical deployability for technical sophistication, or covers only a portion of the grievance lifecycle. The SVSS was designed to occupy precisely the space that none of them fills: a complete, low-cost, mobile-compatible, and institutionally practical platform that handles every stage of a complaint, from the first submission to the final resolution.

III. PROPOSED SYSTEM

A. Architecture Overview

The SVSS follows a three-tier architecture, which is naturally supported by the Model-View-Templatepattern of the Django. The outermost layer is the Presentation Tier, assembled from HTML5, CSS3, Bootstrap 5, and vanilla JavaScript, which is responsible for everything that user sees and interacts with. Below this is the Application Tier, which runs on Django 4.2 andhandles URL routing, form validation, session management, ORM-based database queries, role-based access decisions, and outbound emails via Gmail SMTP. At the foundation is the Data Tier — MySQL 8.x accessed exclusively through Django's ORM, so raw SQL never appears in application code, and every database interaction benefits from the framework's built-in security protections.

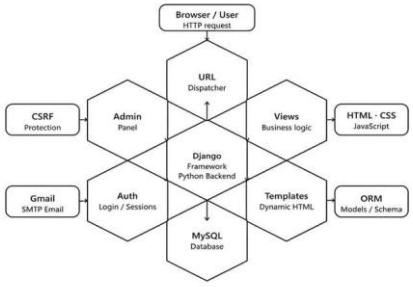

Fig. 1. Web Application and Django Framework Architecture

Fig. 2 illustrates the three-tier deployment model of SVSS, showing how the Presentation Tier (student and administrator browsers), Application Tier (Django with URL dispatcher, ORM, authentication engine, and CSRF

middleware), and Data Tier (MySQL database and Gmail SMTP) exchange data through a well-defined HTTP request-response cycle. Every page renders correctly on desktop, tablet, and mobile browsers through Bootstrap's responsive grid and targeted media queries.

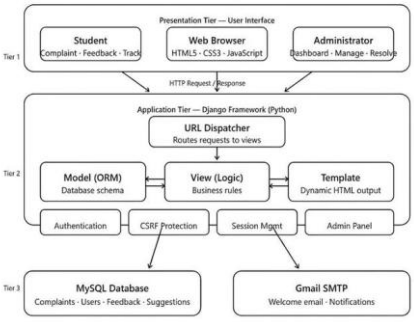

Fig. 2. Three-Tier Architecture of the Proposed System

B. System Block Diagram

The block diagram in Fig. 3 traces the full data and process flow across the three main actors — Student, Web Server, and Administrator — and the four core database tables: Complaints, Users, Feedback, and Suggestions. It shows how information moves from the moment a student registers and files a complaint through Django's URL routing layer into MySQL and back to both the student's status tracking view and the administrator's management dashboard.

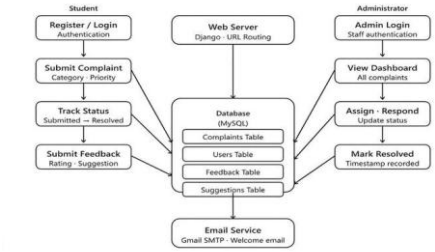

Fig. 3. Block Diagram of the Proposed System

C. Key Design Decisions

A few deliberate design choices give the SVSS its unique characteristics. Ticket ID generation calls Python's uuid.uuid4() to produce a permanent eight-character alphanumeric identifier in the format SV-X when a complaint record is first saved. This identifier never changes and can be shared freely; a student can pass it to a parent, classmate, or staff member, and that person can check the current resolution status on the public tracking page without needing to log in to the system.

The anonymous submission toggle responds to a well-known reality in small residential colleges: students with genuine grievances about a staff member or department frequently stay silent rather than risk being identified as the complainant. The toggle lets a student hide their name from every non-administrative view, while the system keeps the submitting account linked internally for auditing purposes. Administrators can see who submitted the complaint, but the student's name never appears on any shared or student-facing page.

AJAX-based upvoting lets students express their shared concerns about complaints without a full page reload. The upvote complaint view returns a JSON object with the updated vote count and current toggle state for that user, and a JavaScript callback immediately updates the displayed number. Gmail SMTP is integrated through Django's built-in email backend, configured against smtp.gmail.com on port 587 with TLS. Setting fail silently true means that a temporary SMTP outage does not disrupt or block the registration process.

IV. SYSTEM REQUIREMENTS

A. Hardware Specification

The departmental deployment of SVSS runs adequately on a server with an Intel Core i3 processor at 2.0 GHz or higher. Four gigabytes of RAM is the minimum; eight gigabytes is recommended when the concurrent user load is expected to be high. A 250 GB storage allocation comfortably covers the

application codebase, database records, and user-uploaded media files to be used. A screen resolution of 1366 × 768 or higher ensured that the administrator dashboard was displayed without truncation. A reliable Internet connection is necessary for the Gmail SMTP email delivery to function correctly.

B. Software Specification

On the server side, the platform requires Python 3.8 (64-bit) and Django 4.2 or a later version to run. MySQL 8.x serves as the relational database and is managed through phpMyAdmin via XAMPP. The front end was built using HTML5, CSS3, Bootstrap 5, and JavaScript. The development was performed using Visual Studio Code and PyCharm on a Windows 10 (64-bit) machine. Gmail SMTP

runs on port 587 with TLS. Browser compatibility was verified using Google Chrome, Firefox, and Microsoft Edge.

V. SYSTEM IMPLEMENTATION

A. UML Design

The five UML diagram types document the entire system design. The Use Case diagram in Fig. 4 identifies two actors, namely, the Student and Administrator, and maps their respective interactions. Students are associated with eight use cases: Register/Login, Submit Complaint, Track Complaint, Submit Feedback, Upvote Suggestion, View Dashboard, Update Complaint Status, and Assign & Respond. Administrators sharedashboards, status updates, and assignment use cases, reflecting the role-based separation enforced at the view level.

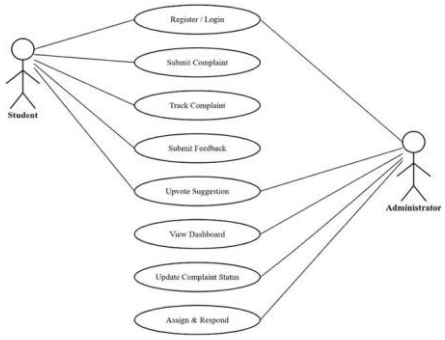

Fig. 4. Use Case Diagram of the Proposed System

The Class diagram defines four core data classes. The user stores the username, email address, PBKDF2-hashed password, and is staff Boolean, which controls all routing and access decisions. A complaint carries a ticket_id, title, category, priority, status, description text, an anonymous flag, a created at timestamp, and a resolved at timestamp, along with a ManyToMany upvote relationship to user. Feedback holds subject, message, feedback type, an optional numeric star rating, an anonymous flag, and a moderation status. The suggestion contains a title, description, an optional category tag, a public visibility flag, and an implemented flag, with its own ManyToMany upvote relationship.

The Sequence diagram traces the complaint submission flow step by step — from the HTTP POST request, through the URL dispatcher and view function, through the ORM save and database confirmation, to the final redirect that delivers the Ticket ID to the student. The Activity diagram maps the complete grievance lifecycle across three swim lanes — Student, System/Django, and Administrator — from the initial login to the final resolution. The Data Flow diagram links the Authentication, Complaint Management, Feedback Management, and Admin Dashboard processes to the MySQL database and Gmail SMTP service.

B. Database Design

The MySQL database, named student voice, runs on 127.0.0.1 at the port 3306. All schema changes are handled through Django's migration framework, which converts Python model definitions into SQL statements and applies them in a numbered, reversible sequence. The Users table anchors the schema using foreign-key relationships. The Complaints table holds a unique index on ticket_id and a foreign key back to users. The rating column of the feedback table accepts integers from one to five and allows null where a student chooses not to provide a star rating. Two junction tables manage the ManyToMany upvote relationships, each with a composite unique constraint that prevents any user from voting more than once on a given complaint or suggestion.

VI. MODULE DESCRIPTION AND SYSTEM INTERFACE

A. Accounts Module

All users enter the system through the Accounts Modules. The Login page shown in Fig. 5 keeps things simple — a username field, a password field, and a submit button — with credential validation handled by Django's built-in Authentication Form. When credentials fail, the error message is intentionally generic and does not reveal which specific field is incorrect, guarding against username enumeration. A successful login creates an encrypted server-side session and routes the user to the appropriate dashboard, based on their role.

Fig. 5. Login Page of the Students Voice and Support System

The Registration page, shown in Fig. 6, collects seven fields: first name, last name, student ID, institutional email address, username, password, and password confirmation. All validation is performed server-side — the

username and email are checked for uniqueness against existing

records, the password must be at least eight characters long and the confirmation field must match exactly. A successful registration sends a welcome email through Gmail SMTP and redirects the new user to the login page.

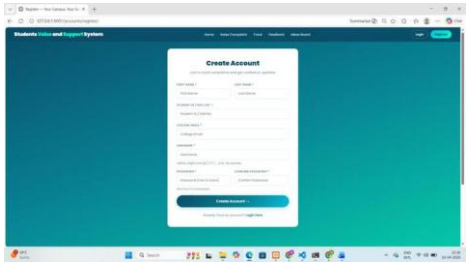

Fig. 6. New User Registration Page

After authentication, Django's session framework creates an encrypted session record for the user. The routing logic checks is staff and is superuser on the authenticated user profile, directing staff accounts to the Admin Dashboard and regular students to the Student Dashboard (Fig.7). Critically, this separation is enforced within the view functions themselves, not just at the template rendering level; therefore, it cannot be bypassed by direct URL manipulation.

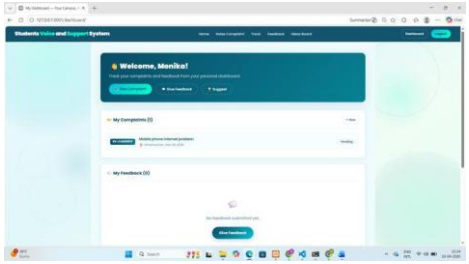

Fig. 7. Student Dashboard Showing Complaints and Status

B. Complaints Module

The Complaints Module handles the end-to-end complaint lifecycle. The Complaint Submission form in Fig. 8 collects a title, a category chosen from ten options (transport, cleanliness, infrastructure, canteen, hostel, library, academics, administration, sports, and other), a priority level from Low to Urgent, a full description, an optional location, an optional file attachment, and an anonymous submission toggle. When the form is submitted, Python's uuid.uuid4() generates the SV-X Ticket ID, which is permanently saved in the complaint record.

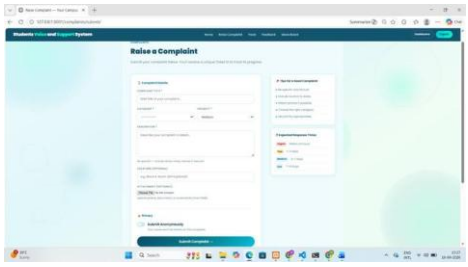

Fig. 8. Complaint Submission Form with Category and Priority Selection

The Complaint Tracking page shown in Fig. 9 allows any authenticated user to look up a complaint by typing its Ticket ID into a straightforward search field. This is especially useful when a student needs to show a staff member the current resolution status without granting them full access to the platform. The my complaints view provides every student with a personal list of all their submitted complaints, ordered chronologically, and marked with color-coded status badges.

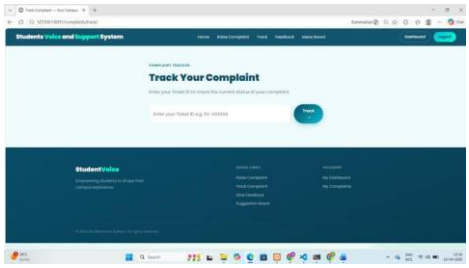

Fig. 9. Complaint Status Tracking Page

C. Admin Dashboard Module

The Admin Dashboard shown in Fig. 10 is accessible only to is staff and is superuser accounts. A Statistics Summary Panel at the top displays five live counters: Total, Pending, Under Review, In Progress, and Resolved, each calculated through Django ORM filter and count calls at the request time. Below this, a Complaints Management Table lists the 20 most recently submitted complaints in reverse chronological order, showing the Ticket ID, student name, category, priority, a color-coded status badge, and an action button that opens an inline update form.

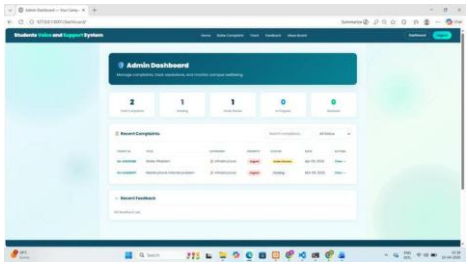

Fig. 10. Admin Dashboard with Complaint Statistics and Management Table

When an administrator selects one of the five status values — Pending, Under Review, In Progress, Resolved, or Closed — and submits an official response, the update view first verifies the requesting user's staff status before writing any changes to the database. When the status is advanced to resolved, the system automatically records the resolved at timestamp. A Category Distribution Panel groups complaints by category using Django's values and annotate query functions, making it straightforward to identify which service areas generate the most student dissatisfaction.

D. Feedback Module

The Feedback Module broadens the platform's purpose beyond reactive complaint handling to support ongoing dialogue between students and the institution. The Feedback Submission form shown in Fig. 11 invites students to choose a feedback type from six categories, write a subject line and message, select an optional star rating between one and five, and decide whether to submit anonymously. Django's form validation enforces all required fields before any database write takes place.

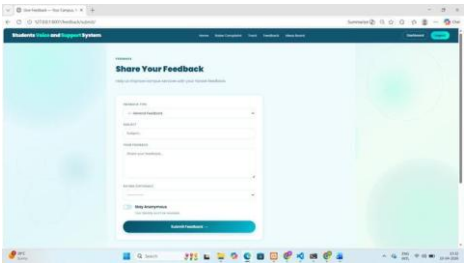

Fig. 11. Feedback Submission Page with Rating and Anonymous Toggle

The Suggestion Board shown in Fig. 12 operates on a different principle than the complaint workflow. Any student can post a public improvement idea with a title, description, and optional category tag, and any other authenticated student can vote on it. The suggestion board view sorts all visible suggestions in descending order of upvote count using Django's annotate function; thus, the most widely supported proposals naturally rise to the top without requiring any editorial intervention.

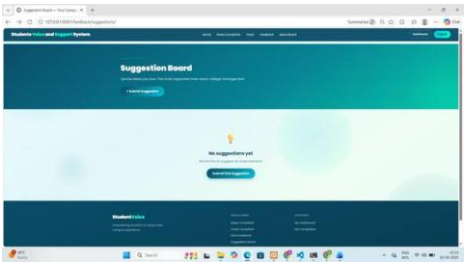

Fig. 12. Community Suggestion Board Sorted by Upvote Count

The Suggestion Submission form shown in Fig. 13 collects a title, description, optional category tag, anonymity preference, and public visibility toggle. Administrators can mark any suggestion as implemented, which triggers a visual indicator on the Suggestion Board — a visible signal to the student community that their collective input resulted in a concrete institutional change.

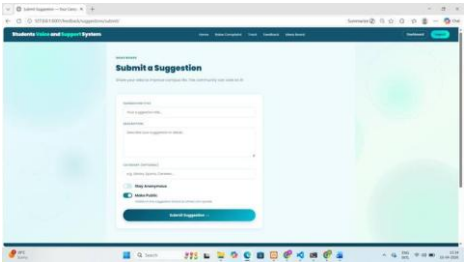

Fig. 13. Suggestion Submission Form on the Ideas Board

VII. VERIFICATION AND VALIDATION

A. Verification

Verification asks a straightforward question: Does the system perform as its specifications state? For the SVSS, this was answered through four layers of testing, beginning with manual code inspection and progressing to increasingly integrated checks.

The code review went through all Python source files across the three application modules — models.py, views.py, forms.py, and urls.py — verifying consistent naming conventions, alignment of validation rules with the design specification, and correct mapping between URL routing patterns and their intended view functions. Unit tests were then performed to check the individual components of the model in isolation. The save() method of the complaint model was confirmed to produce a unique SV-X ticket_id on the first save and leave it unchanged on all subsequent saves. The upvote count property was tested across multiple voting scenarios. The RegisterForm was tested against a duplicate username, duplicate email, password shorter than eight characters, and mismatched confirmation, and returned the expected validation errors in each case.

Integration testing examines the behavior of components when they work together as a system. The complete complaint submission chain was traced from form entry through validation, database writing, Ticket ID generation, and redirection to the complaint detail page. The administrator status update flow confirmed that resolved_at

was recorded correctly when a complaint was advanced to resolved and that requests from non-staff accounts were rejected before any database operation was attempted. The AJAX upvote cycle confirmed that the JSON response carried the correct vote count and toggle state for both adding and removing the votes.

Security verification confirmed the presence of four protective mechanisms in the proposed model. Every POST form carries a CSRF token validated automatically by Django's CSRF middleware.

The @login_required decorator on each protected view correctly intercepts the unauthenticated requests and redirects them to the login page. The is_staff check in the status update view rejects modification attempts from non-staff accounts before any database operation is performed. All passwords are stored as PBKDF2 SHA-256 hashes and never appear in plain text in thedatabase or any application logs.The X-Frame-Options response header is set to deny, which prevents the application from loading inside an external frames.

B. Validation

Functional validation confirmed all 12 primary use cases against their respective specifications. Complaint submission producing a unique Ticket ID, Ticket ID lookup returning the current resolution status, personal complaint listings displaying accurate status badges, AJAX upvoting updating the count without a page reload, administrator status updates appearing in the student's view immediately, automatic resolved at timestamp recording, and suggestion board rendering in upvote-count-descending order were all confirmed to operate exactly as designed.

User interface validation confirmed correct rendering across Google Chrome, Firefox, and Microsoft Edge browsers. Responsive design validation confirmed that the complaint form, tracking page, student dashboard, and admin dashboard all adapted their layouts correctly to desktop, tablet, and mobile viewports. Data integrity validation confirmed that anonymously submitted complaints were displayed as 'Anonymous' in all non-administrative views, while the actual submitting account remained visible to the staff in the administrative interface.

VII. CONCLUSION

Student grievances are frequently the earliest indicators that something is amiss in an institution. Their value as administrative intelligence is highest when they are captured systematically and tracked to resolution and lowest when they are dissolve into informal conversations without a record. SVSS was built around this premise: the institutions that most need a grievance management tool are ordinary colleges, and an ordinary college needs a solution that works in practice, not one that looks impressive on paper but demands resources that the institution does not have.

The platform covers the complete grievance workflow — from the moment a student submits a concern and receives a Ticket ID, through every administrator response and status change, to the final resolution timestamp — without relying on machine learning models, blockchain infrastructure, or specialized hardware. It is built on Python and Django, which are among the most widely understood technologies in academic computing; therefore, staff with basic programming knowledge can maintain and extend it as needed. The open-source stack incurs no licensing fees and receives continuous security maintenance from a large, global developer community.

Testing confirmed that each designed feature operated correctly under normal and adversarial conditions. Security mechanisms, such as CSRF protection, PBKDF2

password hashing, session-based access enforcement, and clickjacking prevention through the X-Frame-Options header, function as specified. The comparative analysis showed that no existing published system covers the same combination of full lifecyclemanagement, low infrastructure cost, mobile responsiveness, and anonymous submission support as the SVSS does.

Looking ahead, the most valuable near-term addition would be automated email notifications triggered whenever the complaint status changes to keep customers updated. A REST API layer would make it straightforward to build native mobile clients on the existing backend. Over a longer horizon, an NLP-based classification module can automate category and priority assignment as the system builds sufficient historical data to train a dependable model. None of these enhancements would require rebuilding or substantially rearchitecting what is already in place, and the ease of extension is a reflection of how thoughtfully the foundation was designed.

REFERENCES

[1] T. Kumar and V. Reddy, “Intelligent Student Issue Resolution Platform Using Deep Learning,” IEEE

Transactions on Learning Technologies, vol. 19, no. 2, pp. 145–158, 2026.

[2] L. Joseph and N. Shah, "Smart Campus Support System with Real-Time Monitoring and Data Analytics," IEEE Access, vol. 13, pp. 22456–22471, 2025.

[3] M. Prakash and R. Nair, "Secure Web-Based Student Feedback and Grievance System Using Blockchain," IEEE Internet of Things Journal, vol. 11, no. 4, pp. 6781–6795, 2024.

[4] R. Sharma and P. Mehta, "E-Governance Based Student Grievance Redressal System for Academic Institutions," International Journal of Computer Applications, vol. 185, no. 12, pp. 34–41, 2023.

[5] A. Gupta and S. Rao, "Web-Based Feedback Management System for Higher Education Institutions," Journal of Engineering Education Transformations, vol. 36, no. 1, pp. 112–121, 2022.

[6] D. Patel and K. Iyer, "Mobile-Based Campus Grievance System with Push Notifications," in Proc. International Conference on Emerging Technologies, pp. 210–217, 2023.

[7] A. Holovaty and J. Kaplan-Moss, The Definitive Guide to Django: Web Development Done Right, Apress, New York, 2009.

[8] W. S. Vincent, Django for Beginners: Build Websites with Python and Django, Independently Published, 2022.

[9] R. S. Pressman, Software Engineering: A Practitioner's Approach, 8th ed., McGraw-Hill Education, New York, 2014.

[10] I. Sommerville, Software Engineering, 10th ed., Pearson Education Limited, London, 2016.

[11] Django Software Foundation, Django Documentation – Version 4.2, 2023. [Online]. Available: https://docs.djangoproject.com/en/4.2/

[12] Mozilla Developer Network, XMLHttpRequest – AJAX, Mozilla Foundation, 2024. [Online]. Available: https://developer.mozilla.org/enUS/docs/Web/API/XMLHttpRequest