SOCCSYNC: A WEB-BASED STUDENT DIGITAL HUB FOR THE COLLEGE OF COMPUTER STUDIES OF LAGUNA STATE POLYTECHNIC UNIVERSITY

A Final Project

Presented to the

Faculty of College of Computer Studies

Laguna State Polytechnic University

Santa Cruz Campus

In partial Fulfilment of the Requirements for the course

CLIENT SERVER TECHNOLOGIES

Submitted by:

Symon Kiel G. Beato
Justine Mark R. Gahi
Jake Elizer M. Jaqueca
Zaldy B. Ybardolaza II

Submitted to:

Harlene Gabrielle E. Origines
Course Instructor

December 2025

CHAPTER I

INTRODUCTION

The proliferation of digital technology has fundamentally reshaped higher education, demanding more integrated and accessible platforms for student services. Universities worldwide are adopting comprehensive systems to manage academic information, communication, and student engagement *(Al-Samarraie & Saeed, 2021).* These systems serve as a central nervous system, connecting students, faculty, and administration through a single, cohesive interface. The shift from traditional, paper-based processes to digital hubs streamlines operations, enhances information dissemination, and fosters a more connected campus community. This evolution is no longer a luxury but a necessity for modern educational institutions aiming to provide a responsive and efficient learning environment. Consequently, the development of specialized, in-house systems tailored to specific college needs is becoming a critical area of focus.

The College of Computer Studies (CCS) at Laguna State Polytechnic University, while technologically proficient, currently operates with a fragmented information ecosystem. Announcements are disseminated through multiple, social media channels, leading to confusion and missed updates. Students face difficulties in accessing learning materials, tracking their academic requirements, and processing college-specific transactions, which often require physical presence and manual paperwork. This disjointed approach creates inefficiencies, hinders student participation, and places an administrative burden on both the Student Organization (SOCCS) and the college faculty. The

lack of a centralized repository for academic resources, payment processing, and official forms results in wasted time and a disconnected student experience.

In the international landscape, learning management systems (LMS) and student portals like Blackboard and Canvas are standard, providing robust infrastructures for academic and administrative tasks (*García-Peñalvo et al., 2021*). Locally, many Philippine universities are developing their own integrated portals to address similar challenges. This study builds on that trend by addressing the specific operational gaps within the CCS at LSPU. The rationale for this project is the urgent need to consolidate disparate functions into a single, reliable, and accessible web application. This research aims to create a centralized digital hub, "SOCCSync," to improve efficiency, accessibility, and participation for the CCS community.

Background of the Study

This project is anchored in client-server technology, a computing model where a central server hosts, delivers, and manages most of the resources and services consumed by multiple clients (i.e., student and administrator web browsers). The SOCCSync system utilizes this architecture: a web server (the server) will host the application logic, database, and all student records, while students and faculty (the clients) will access this information via their devices. The importance of this model lies in its centralization, security, and scalability. It allows for student data to be securely stored and managed in one location, ensures that all users receive the same consistent information, and provides a

platform that can be updated and maintained efficiently without requiring changes to end-user devices.

Research Problem

The general problem is the lack of a centralized, efficient, and accessible digital platform for the College of Computer Studies (CCS), leading to fragmented communication, inefficient manual processes, and a disconnected student experience. The reliance on disparate non-academic platforms creates information silos, administrative bottlenecks in tracking payments and documents, and a reactive student governance model. Specifically, this research seeks to answer the following questions:

1. How can a centralized web application streamline the broadcasting of organizational announcements, event details, and real-time inquiries to replace the current fragmented multi-platform approach?
2. What system features are necessary to eliminate inefficiencies in manual payment processing, receipt verification, and paper-based student elections?
3. How can a digital repository system improve student access to learning materials (e-Library), facilitate peer-to-peer resource sharing (EduShare), and digitize the submission of required student documents?
4. How can the system provide a secure and structured channel for student complaints, suggestions, and direct administrative communication to ensure student concerns are addressed proactively?

5. What are the necessary design standards, specifically regarding mobile responsiveness and UI/UX branding (e.g., violet/purple palette), to ensure high adoption rates and system usability among students and officers?

The primary cause of these problems is the reliance on disparate, non-academic platforms and manual, paper-based systems for official college operations. The effects include information silos, administrative bottlenecks, difficulty in tracking student payments, and a reactive rather than proactive student governance model.

The proposed solution is the development and implementation of SOCCSync, a comprehensive web application serving as a singular digital hub for all academic and administrative needs of CCS students.

Project Objectives

The general objective of this project is to design, develop, and implement SOCCSync, a centralized web-based student digital hub for the College of Computer Studies, to streamline communication, transactions, and access to resources.

Specifically, it aims to:

1. To develop the Announcement, Event Management, and Chat Modules to centralize the dissemination of general and urgent updates, manage event calendars with Google Calendar-style UI, and facilitate real-time inquiries through an AI-powered chatbot and admin messaging system.
2. To develop the Payment Processing and Election & Voting Modules to automate the collection of fees (membership, events) via GCash with receipt

verification, generate financial reports, and conduct secure, automated student elections with real-time tallying.

3. To develop the Digital Library, EduShare, and Document Management Modules to create a centralized repository for academic PDFs and links, enable students to share projects and code, and facilitate the secure uploading and verification of files such as medical certificates and consent forms.
4. To develop the Complaint and Feedback Module to provide a structured, anonymous or attributed channel for student grievances and suggestions, allowing administrators to track and resolve issues efficiently.
5. To evaluate and ensure the system's Usability and Performance by implementing a fully responsive mobile-first design using Tailwind CSS, adhering to the institutional branding (purple/white palette), and enforcing role-based access control (RBAC) security for Students, Admins, and Super Admins.

Scope and Limitations of the Research

This research focuses on the design, development, and implementation of the SOCCSync web application. The scope is strictly limited to the students and faculty (specifically SOCCS officers and a Super Admin) within the College of Computer Studies at the Laguna State Polytechnic University, Santa Cruz Campus. The system will manage functionalities directly related to the SOCCS and college administration, including announcements, payments for college-specific fees, event management, resource sharing (e-Library and EduShare), student profiling, and a voting system for SOCCS elections. The duration of the project covers the development lifecycle from requirements gathering to

deployment and testing, estimated to be completed by December 2025. Furthermore, the study has several limitations. The system will not be integrated with the university's main registrar or finance office, as it is designed to manage internal college records and payments only. The online payment feature is restricted to GCASH integration, and other payment gateways such as Maya or credit card services are beyond the current scope. Additionally, the project focuses solely on development and initial deployment; long-term maintenance and server hosting costs after implementation are not included. Lastly, the system is web-based and supports mobile access only through a responsive browser interface, rather than a native mobile application.

Significance of the Study

The successful implementation of SOCCSync will provide significant benefits to several stakeholders:

***Students**:* They will benefit from a single, reliable source for all announcements, academic materials, and forms. The system will provide convenience through online payments, remote access to resources, and easier tracking of their requirements, fostering a more engaged and informed student experience.

SOCCS Officers (Student Organization): This system will serve as a powerful administrative tool, automating tasks such as disseminating information, collecting payments, generating reports, managing events, and conducting elections. This will reduce their administrative workload, improve accuracy, and allow them to focus more on student governance and initiatives.

College of Computer Studies (Faculty and Admin): The college will benefit from a streamlined communication channel, a centralized database of student information and requirements, and an organized platform for academic resources. The complaints and suggestion box also provides a formal channel for feedback, aiding in continuous improvement.

Future Researchers: This project will serve as a model and baseline for developing similar integrated student systems in other colleges within LSPU or at other state universities, providing a practical case study on client-server application development for academic administration.

CHAPTER II
SYSTEM ANALYSIS AND DESIGN

The SOCCSync (Student Organization Management System for CCS LSPU) is built upon a traditional three-tier client-server architecture utilizing the Laravel framework. This architectural model is fundamental to the system's design, ensuring the clear separation of the presentation layer, business logic layer, and data access layer to promote maintainability, scalability, and security.

The architecture defines distinct roles for the client and the server. The Client acts as the interface for the end-user and functions as a "thin client". It runs on modern web browsers such as Chrome, Firefox, or Safari. The client's primary responsibilities include rendering the user interface using HTML, CSS, and JavaScript, capturing user inputs through interactive elements, and providing real-time feedback via dynamic updates. Crucially, the client does not store business logic or sensitive data; it relies on the server for processing.

The Server is the central processing unit of the application, hosted on a web server like Apache or Nginx. It handles the authentication and authorization of users (students, admins, deans), processes business logic for payments and complaints, and manages data persistence by interacting with the MySQL database. The server also manages file uploads, generates reports, and ensures data integrity through rigorous validation.

The interaction between these two components follows a standard HTTP request-response cycle. A user action triggers an HTTP/HTTPS request containing methods (GET, POST), target URLs, headers, and payloads. The Laravel application receives this request, routes it to the appropriate controller, executes the necessary business logic, and returns an HTTP response

containing the status code and the requested content, such as an HTML view or JSON data.

This model offers significant benefits to the system. Security is enhanced because sensitive logic and database credentials remain on the server, inaccessible to the client. Scalability is achieved as the server can handle multiple concurrent connections, and the system supports load balancing. Furthermore, the Separation of Concerns allows frontend and backend developers to work independently, ensuring that changes to the user interface do not disrupt the underlying business logic.

System Requirements

To ensure the efficient operation of SOCCSync, specific hardware, software, and network prerequisites must be met. These requirements cover the client workstations, the server environment, and the networking infrastructure.

Hardware Requirements

The system requires specific hardware configurations for both the client and server machines to function optimally.

- ***Client Machine****:* Student and administrator workstations require a minimum of a Dual-core CPU (1.5 GHz or higher) and at least 2 GB of RAM. The display should support a minimum resolution of 1024x768. For mobile access, a camera is optional but useful for capturing receipt photos. Recommended specifications for a smoother experience include a Quad-core CPU and 4 GB of RAM.

- ***Server Machine:*** The server hosting the application requires a minimum of a Quad-core CPU (2.5 GHz) and 4 GB of RAM. It must have at least 20 GB of SSD storage for the application and database, along with a 100 Mbps network interface. For production environments, an Octa-core CPU, 8 GB of RAM, and NVMe SSD storage are recommended to handle higher loads.

Software Requirements

The software environment is critical for the compatibility and execution of the system's code and features.

- ***Client Software:*** Users must access the system via a web browser with JavaScript and cookies enabled. Supported browsers include Google Chrome (v90+), Mozilla Firefox (v88+), Microsoft Edge (v90+), or Safari (v14+). The operating system may be Windows 10+, macOS 10.14+, Linux (Ubuntu 20.04), or mobile OS versions Android 8.0+ and iOS 13+.
- ***Server Software:*** The backend runs on PHP version 8.2 or higher. It requires a web server such as Apache HTTP Server (2.4+) with mod_rewrite enabled or Nginx (1.18+) configured with PHP-FPM.
- ***Database:*** The system uses MySQL 8.0 or higher (or MariaDB 10.5+) utilizing the InnoDB storage engine for transactional support.
- ***Dependencies and Libraries:*** The application utilizes the Laravel Framework 11.0. Essential PHP extensions include BCMath, Ctype, cURL, DOM, Fileinfo, and OpenSSL. Additional libraries installed via Composer (v2.0+) include Dompdf 3.1 for generating reports and PHPSpreadsheet 2.0

for Excel exports. Node.js (v16+) and NPM (v8+) are required for compiling assets.

Network Requirements

Reliable connectivity is necessary for the API calls and data exchange between the client and server.

- ***Connectivity:*** The server requires a stable internet connection with a minimum 10 Mbps symmetric speed (100 Mbps recommended) and a 99.9% uptime target. Clients require at least 1 Mbps download speed.
- ***Protocols:*** The system primarily uses HTTP/HTTPS over ports 80 and 443 for client-server communication, with TLS 1.2 or higher required for production security. Database connections use the MySQL protocol on port 3306, restricted to internal networks for security. SMTP (ports 587/465) is used for outbound email notifications.

System Design

The structural design of SOCCSync is grounded in a modular approach that ensures data integrity and logical separation of duties. The system's architecture relies on a robust relational database schema and a component-based application structure, allowing for complex interactions between users, financial records, and organizational events.

The system utilizes a relational database consisting of twenty-six interconnected tables to manage the organization's data. The core of this

design focuses on the users entity, which serves as the primary key for role-based interactions, linking to student data, administrative logs, and dean-level oversight.

Key structural relationships include the interaction between the events and users modules. An event entity maintains a one-to-many relationship with event registrations, while simultaneously linking to financial reports for budget tracking. A specialized design consideration was applied to the Election module, which handles complex circular dependencies between elections and candidates (where an election requires a secretary candidate, but the candidate must belong to an election). This is resolved through a phased data insertion strategy to maintain referential integrity.

Furthermore, the document management structure utilizes a hierarchical design for the EduShare module, employing self-referencing relationships to allow nested folder organizations. This schema ensures that all data points—from simple user profiles to complex nested file structures—are normalized and optimized for retrieval.

Module 1: User Authentication & Registration

User Registration Flow: The registration process initiates when a student accesses the registration interface. The system presents a form requiring personal data, academic details (course, year level, section), and account credentials. Upon submission, the system executes a rigorous, sequential validation protocol. First, input formats are verified for compliance, followed by database queries to ensure the uniqueness of the provided email address and Student ID.

Subsequently, the system validates the email address through three specific checks: domain whitelisting (e.g., Gmail, Yahoo), pattern analysis to

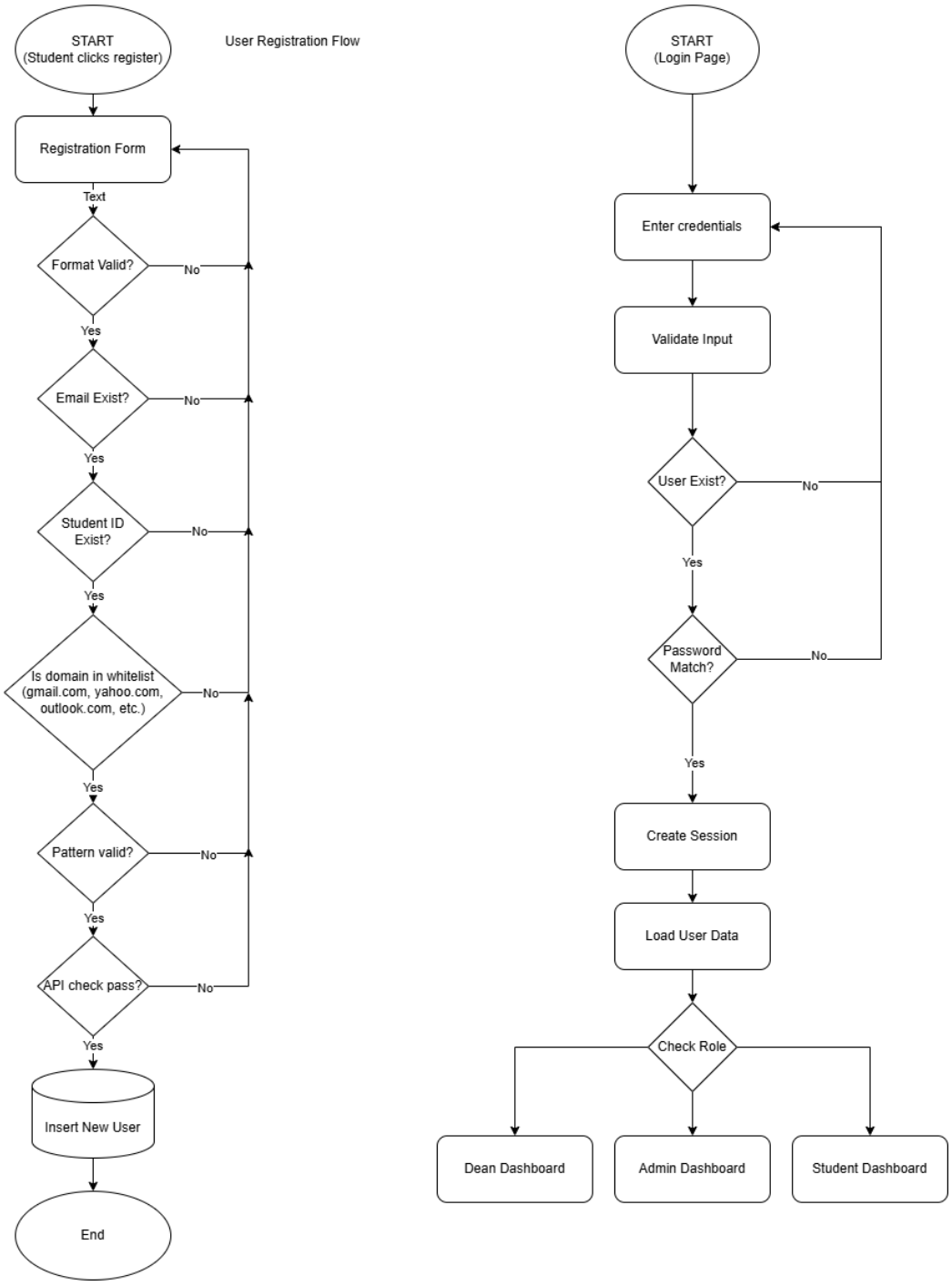


Figure 1. *User Authentication and Registration Logic Flowchart*

reject fake or keyboard-patterned local parts, and verification via external APIs (Kickbox, Debounce, EmailRep, EVA) to identify disposable or suspicious

addresses. Failure at any stage returns specific error messages to the user. Once all validations pass, the system inserts the new user record into the database with the 'student' role and redirects the user to the login page with a success notification

User Login Flow: The login process begins when a user submits their Student ID and password. The system first validates that both fields are populated. It then queries the database to locate the corresponding user record. If the user exists, the system compares the submitted password against the stored plain-text credentials.

Upon successful authentication, the system creates a session and regenerates the session ID to enhance security. Finally, the system evaluates the user's assigned role to determine the redirection logic: users with the 'super_admin' role are routed to the Dean Dashboard, while those with 'admin' or 'student' roles are directed to the standard Dashboard.

Module 2: Election & Voting System

Student Voting Flow*:* The voting process begins when a student navigates to the elections page, where the system queries and displays all posted elections. Upon selecting a specific election, the system retrieves the candidate list and simultaneously checks the database to determine if the student has already cast a vote. If a vote record exists, the system restricts access to the ballot, displaying the current results instead.

If the student has not yet voted, the system presents the candidates grouped by position. The student selects one candidate per category and submits the ballot. The system then performs a multi-stage validation: ensuring

one candidate is selected for every position, confirming the election is currently active, and performing a secondary check for existing votes to prevent race

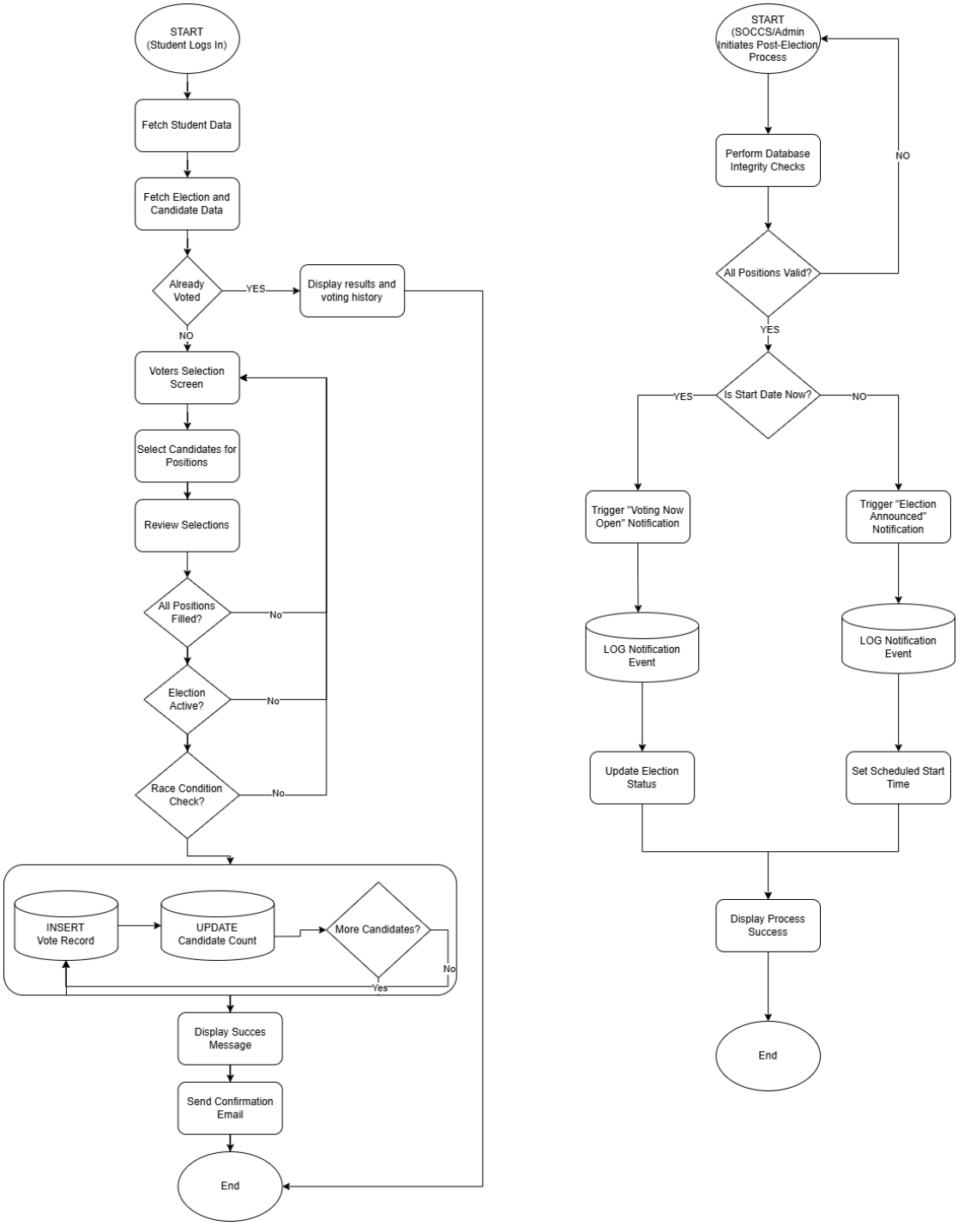


***Figure 2**. Election and Voting System Logic Flowchart*

conditions. Once validated, the system executes an atomic database transaction. This process iterates through the selected candidates, inserting

individual vote records and incrementing the respective candidate's vote count. Finally, the transaction is committed, and the student is redirected to the index page with a success confirmation.

***Admin Post Election Flow**:* The election posting process allows an administrator to activate an election for student participation. Upon selecting an election to post, the system first verifies that every required position (e.g., President, Year Representatives) has at least one registered candidate. If this validation fails, the system returns an error detailing the missing positions.

If the lineup is complete, the system updates the election status to "posted" and "active." It then retrieves the student list to manage notifications based on the election's start schedule. If the start date matches the current time, the system generates "Voting Now Open" notifications and flags the election as notified. Conversely, if the start date is in the future, it generates "Election Announced" notifications. These notifications are inserted into the database, and the administrator receives a confirmation that the election has been successfully posted.

Module 3: Payment Processing System

Student GCash Payment Submission Flow: The payment submission process commences when a student accesses the payments interface, where the system retrieves and displays active payment announcements. Upon selecting the GCash option for a specific item, the user is prompted to input a transaction reference number and upload a corresponding receipt image. The system subjects these inputs to strict validation regarding file type and size (maximum 5MB), while simultaneously querying the database to ensure the

user has no existing pending or approved payments for the selected announcement.

Once the input is validated, the system generates a unique transaction identifier prefixed with "SOCCS-", archives the receipt image in the designated

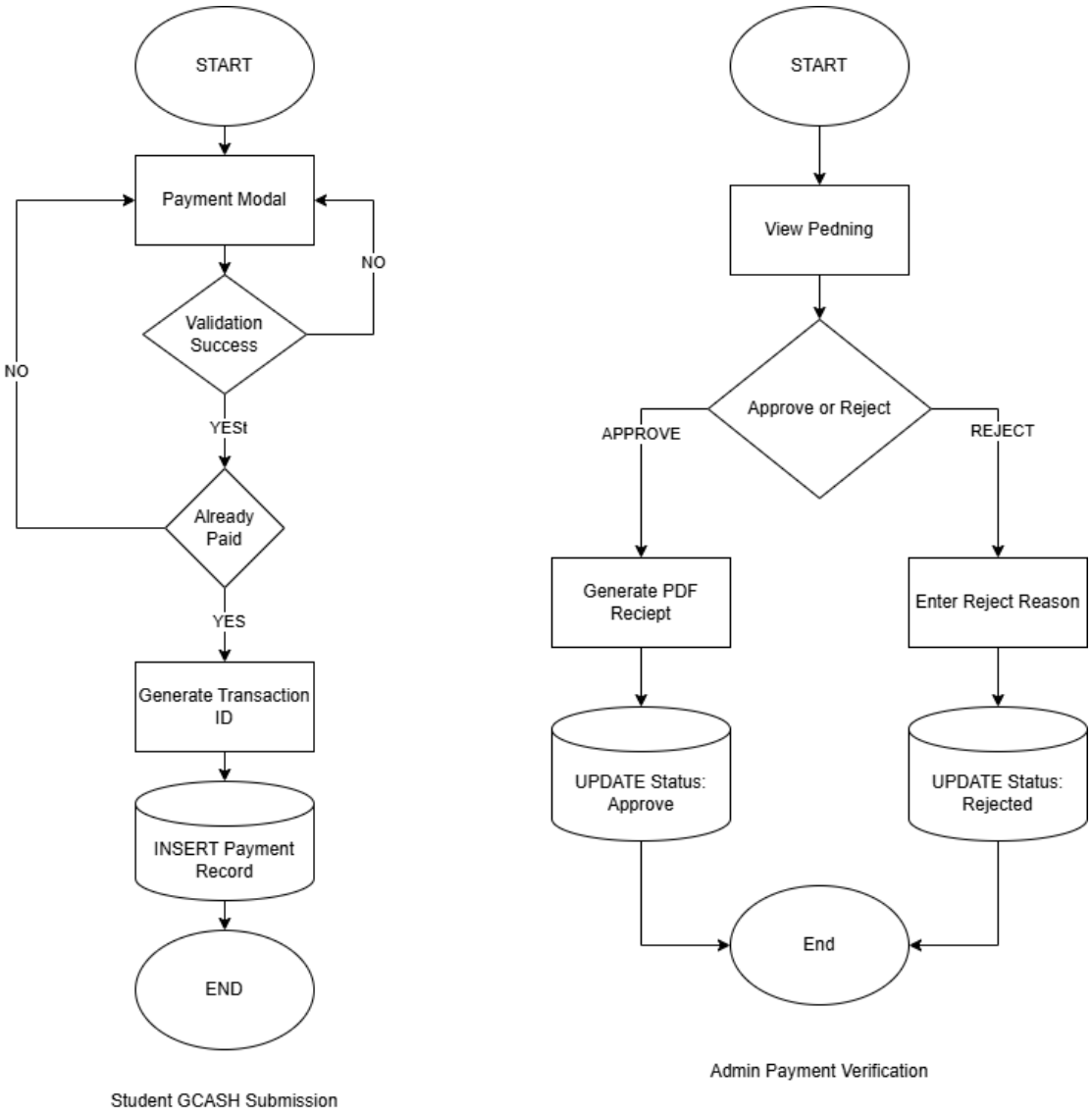


Figure 3. *Payment Processing System Logic Flowchart*

storage directory, and inserts a new payment record with a 'pending' verification status. The process concludes by displaying a confirmation notification to the student.

Admin Payment Verification Flow: The verification workflow enables administrators to manage and review financial transactions. The system initiates by retrieving all records with a 'pending' status, allowing the administrator to inspect the associated student details and the attached receipt preview. The administrator then adjudicates the payment as either approved or rejected.

In the event of approval, the system updates the payment status to 'completed', generates a formal PDF receipt, and issues a notification to the student via both the platform and email. Conversely, if the payment is rejected, the administrator is required to input a valid justification. The system then updates the record status to 'rejected,' logs the specific reason, and triggers a notification to inform the student of the unsuccessful transaction.

Module 4: Document Management System

Student Document Upload Flow: The document management process initiates when a student accesses the "My Documents" interface, which displays existing records and an upload utility. The user selects a specific document type—Certificate of Registration (COR), Medical Certificate, or Parental Consent—prompting the system to dynamically display relevant input fields. Specifically, COR uploads require academic term details (Year Level and Semester), while other documents require an issue date.

Upon submission, the system performs a validation check on the file format (PDF, JPG, PNG), size (maximum 5MB), and mandatory metadata. Validated files are archived in the storage directory, and the system automatically calculates the document's expiration date: one year for parental

consents, six months for medical certificates, and no expiration for CORs. The record is subsequently stored in the database with an initial "unverified" status, and the document list is refreshed.

Document Sharing with Admin Flow: The sharing protocol allows

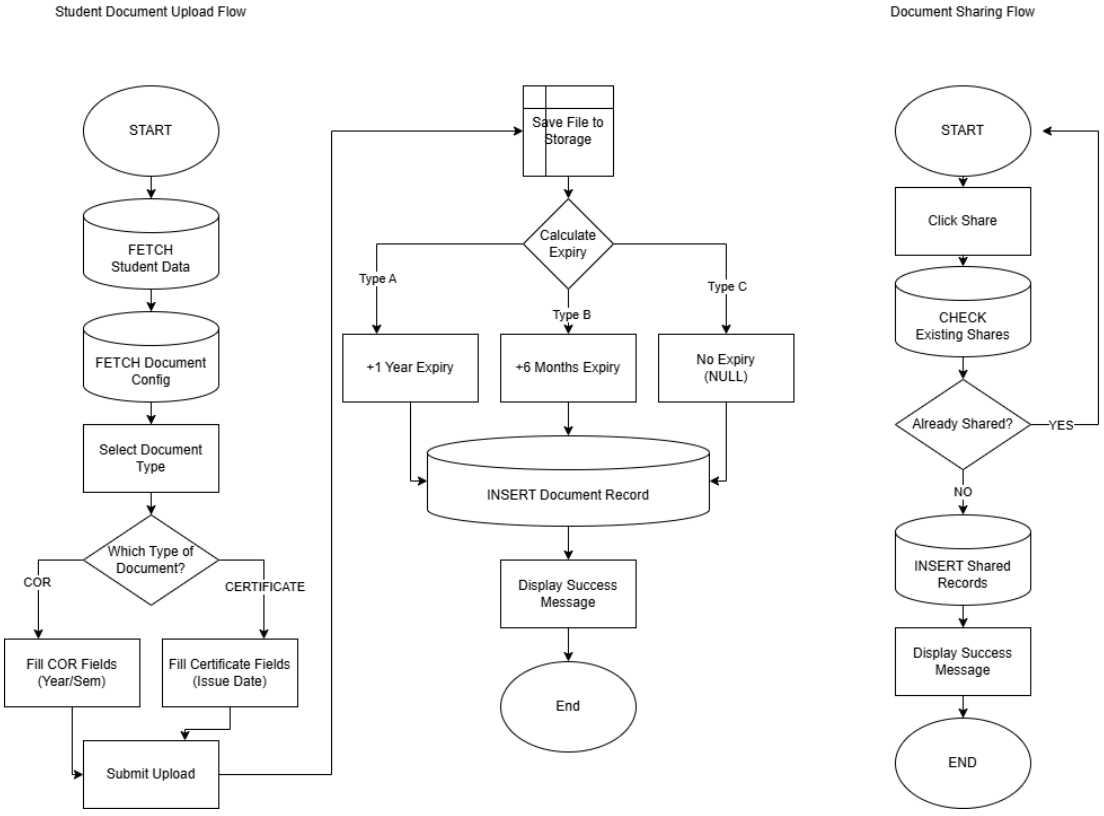


Figure 4*. Document Management System Logic Flowchart*

students to submit specific documents for administrative review. When a student initiates the sharing action, the system first queries the database to ensure the document has not previously been shared. If the document is unique to the sharing log, the system inserts a new record into the shared_documents table with an "unread" status and confirms the successful transmission to the user.

Module 5: Complaint & Feedback System

Student Complaint Submission Flow: The submission process begins when a student navigates to the complaints interface. The system presents a

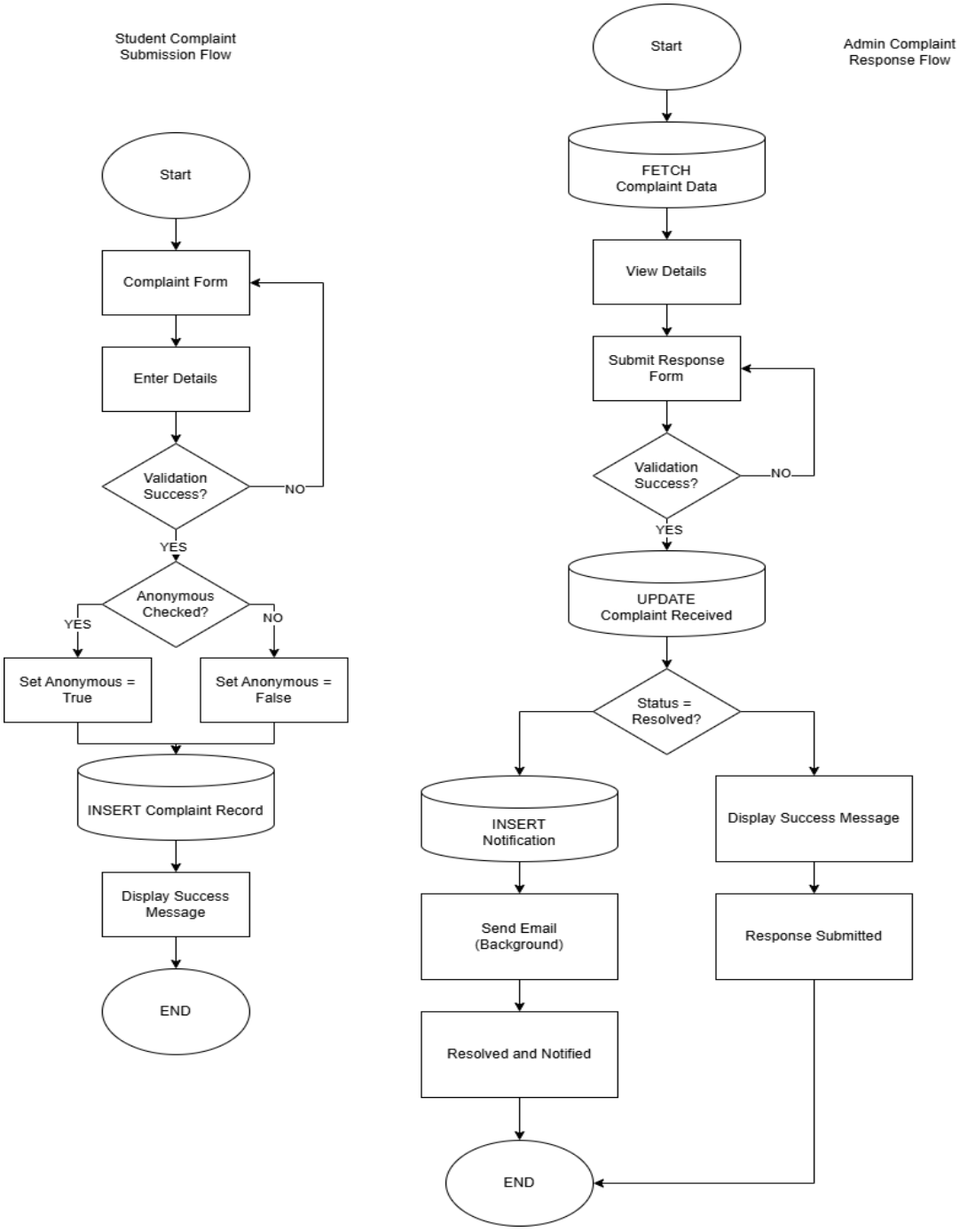


Figure 5. *Complaint and Feedback System Logic Flowchart*

form requiring the user to categorize their input (complaint, suggestion, inquiry, or feedback) and provide a subject and detailed message. Users effectively

control privacy by opting to check the "Submit Anonymously" validation box. Upon submission, the system verifies that all mandatory fields meet length and content constraints. Validated entries are then inserted into the database with a 'pending' status and the appropriate anonymity flag, followed by a redirection to the complaints list with a success confirmation.

Admin Complaint Response Flow: The response workflow allows administrators to address submitted feedback. The process starts with the system retrieving and displaying a filtered list of complaints sorted by date. When an administrator selects a specific entry, the system displays the submission details, revealing the student's identity only if the "anonymous" flag was not selected during submission.

The administrator drafts a response and assigns an updated status (pending, reviewed, or resolved). Upon validation of these inputs, the system updates the record in the database. If the status is designated as 'resolved,' the system automatically triggers a dual-notification process, alerting the student via both the platform and email; otherwise, the system simply finalizes the update and displays a standard success message.

Module 6: Event Management System

Admin Event Creation and Approval Flow: The event management lifecycle initiates when an Administrator accesses the creation interface to propose a new event. The system requires comprehensive details—including title, schedule, venue, and planned budget—alongside mandatory image uploads. Upon submission, the system executes a validation protocol to ensure chronological consistency (confirming the end date follows the start date) and

file compliance (verifying image count and size limits). Successful validation triggers the secure storage of media assets and the insertion of a new database

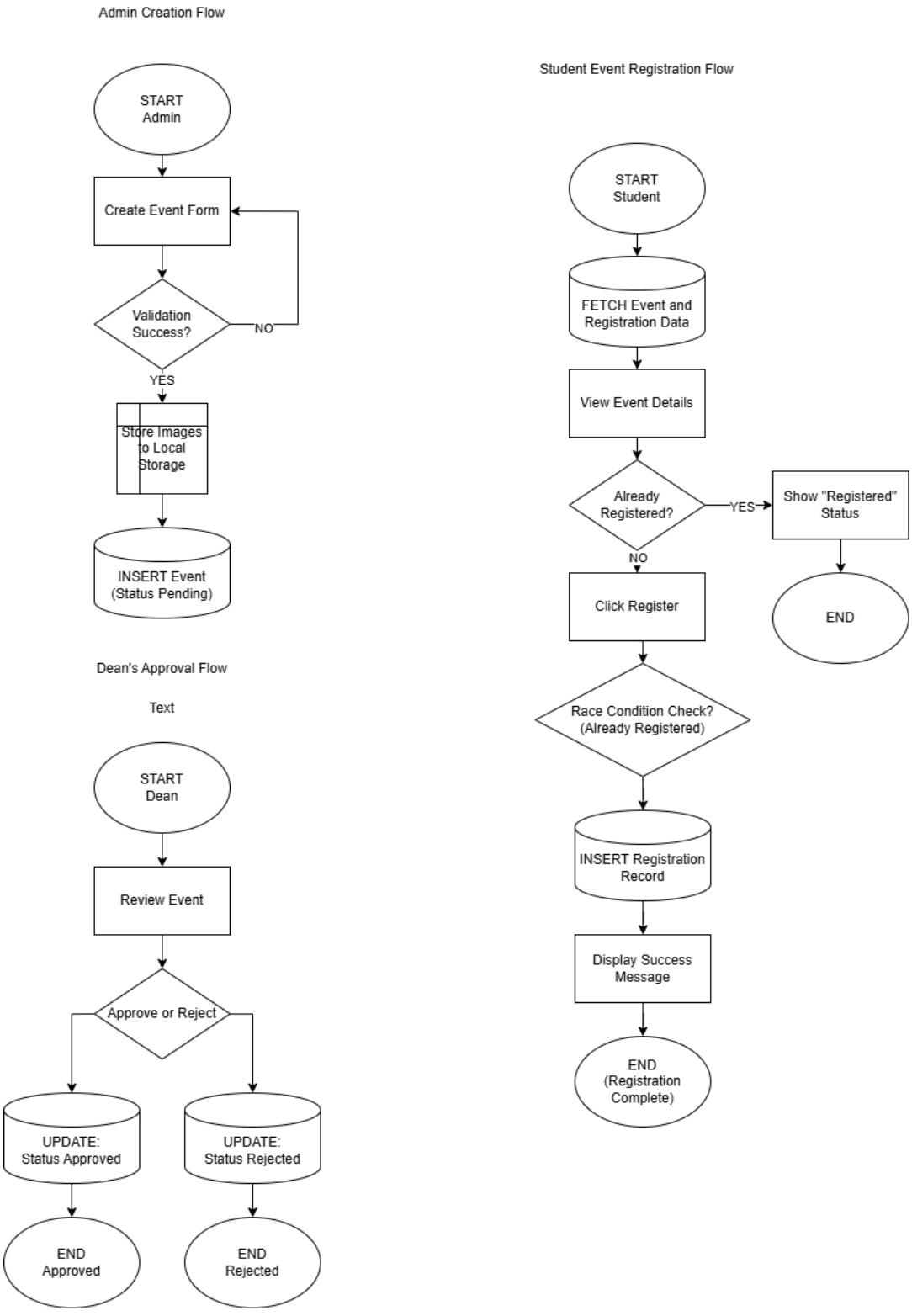


Figure 6. *Event Management System Logic Flowchart*

record marked with a 'pending' approval status.

Subsequently, the workflow transitions to the Dean, who accesses the approvals dashboard to review pending proposals. The Dean adjudicates the request by either approving or rejecting the event. The system immediately updates the event's status in the database based on this decision, records the timestamp and approver identity, and confirms the action via a system notification.

Student Event Registration Flow: The registration process begins when a student navigates to the events module, where the system retrieves and displays a list of active, published events. Upon selecting a specific event, the system performs an initial database query to verify the student's registration status. If a prior registration is detected, the interface restricts further action by displaying an "Already Registered" indicator.

For unregistered users, the system enables the registration interface. When the student initiates the registration command, the system performs a secondary, real-time validation to prevent race conditions. Once the user's non-registered status is confirmed, the system inserts a new registration record into the database and refreshes the interface to reflect the successful enrollment.

Entity Relationship Analysis

The database architecture for the SOCCSync system is designed around a centralized relational model, ensuring data integrity and efficient query performance across all functional modules. The schema is anchored by a primary entity that interacts with five distinct sub-systems: Elections, Payments, Events, Document Management, and Communication.

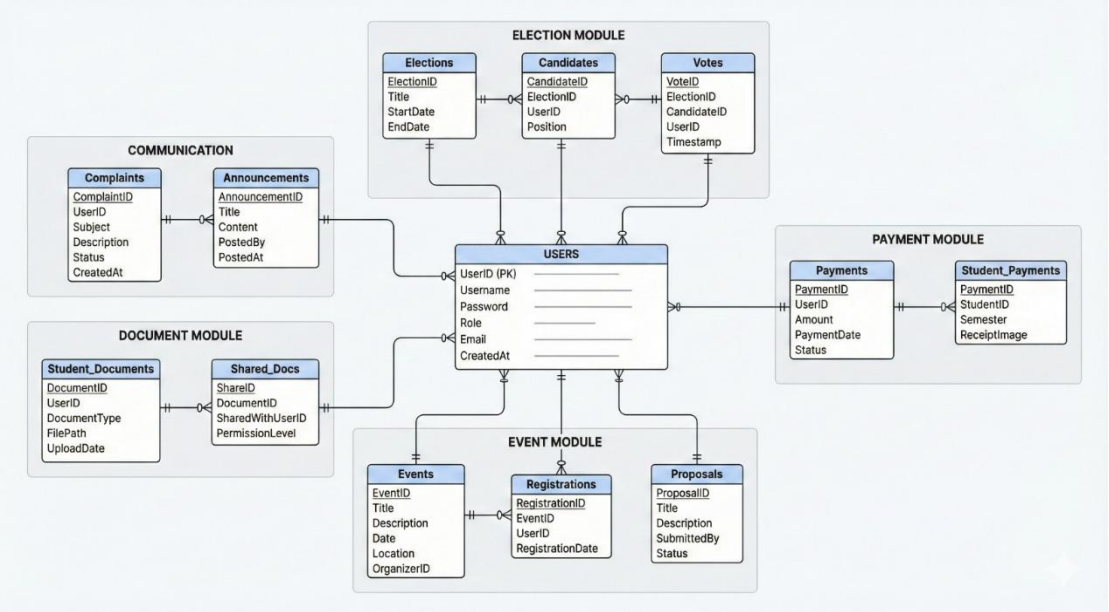


Figure 7. *Entity Relationship Diagram (ERD) of SOCCSync*

Central Entity: *User Management*

At the core of the database lies the users table, which serves as the primary reference point for the entire system. This entity stores essential authentication and profile data, including student_id, email, role, and academic details (course, year_level, section). The unique id primary key of this table acts as a foreign key across almost all other modules, establishing the identity of actors within the system—whether they are students, administrators, or deans.

Election and Voting Module: The election subsystem is structured around three key entities: elections, candidates, and votes. The elections table defines the voting period and metadata, while the candidates table links specific individuals to an election. A critical relational feature here is the votes table, which functions as an associative entity. It establishes a many-to-one relationship with users, elections, and candidates, creating a verifiable record of which user voted for which candidate in a specific election. Additionally, the secretary_id field in the elections table creates a specific relational link to the candidates table to identify the election secretary.

Payment Processing Module*:* The financial architecture distinguishes between payment categories and individual transactions. The payments entity represents the receivable items or fees created by administrators. The student_payments entity records the actual transactions made by students. This table maintains a complex relationship structure: it links to the users table to identify the payer, the payments table to identify the fee being paid, and recursively back to the users table via the verified_by field to track the administrator responsible for validating the transaction.

Event Management Module: The event management schema is centered on the events table, which stores logistical details such as venue, budget, and schedules. This entity maintains dual relationships with the users table to track both the event creator and the approving authority (approved_by). The module expands through several child tables: event_registrations manages student participation, while event_proposals and event_financial_reports ensure comprehensive documentation of the event lifecycle.

Document and Resource Management: To handle academic records, the student_documents table stores file paths and metadata for uploaded files. This entity includes a verification workflow, linking to the users table to identify the administrator who verified the document. The sharing logic is handled by the shared_documents table, which acts as a bridge between a specific document and the administrative user it is shared with. Parallel to this, the edushare_items and edushare_folders tables manage the repository of shared educational resources.

Communication and Feedback: The system's interaction layer is supported by the announcements, complaints, and admin_chats tables. All three entities maintain direct foreign key constraints referencing the users table. The complaints table is particularly notable for its self-contained workflow tracking, linking the complainant (user) with the responder (admin) to maintain a clear audit trail of issue resolution.

API Integration

The SOCCSync system utilizes a RESTful API architecture to facilitate communication between the frontend and backend, as well as to integrate with internal services for file management and reporting. The integration follows standard REST principles to ensure consistency and ease of data exchange.

APIs serve several critical functions within the system. They manage Payment Processing by handling transaction records and validating payment claims. File Storage APIs handle the secure upload and retrieval of receipts and documents. Data Exchange APIs allow the frontend to update dynamically

without full page reloads, and Notification Services utilize APIs to send automated emails.

While the system is designed to support external payment gateways (like PayMongo or Xendit) for automated processing, the current implementation focuses on internal API endpoints that manage a manual verification workflow. This allows students to initiate payments and upload receipts, which administrators then verify via the system's backend.

The system exposes protected endpoints for specific operations. These requests are authenticated and require CSRF tokens.

1. **Student Payment Creation**

   Endpoint: POST /payments/gcash/create-pending

   Purpose: This endpoint creates a pending payment record when a student initiates a transaction.

   Data Exchanged: The client sends the amount, title, and payment_id via form data. The server returns a JSON response containing a success boolean, a status message, and the student_payment_id.

   Process: The system validates the request, checks for duplicates, creates a record with a "pending" status, and generates a unique transaction ID.

2. **Receipt Upload**

   Endpoint: POST /payments/gcash/upload-receipt

   Purpose: Uploads the proof of payment (screenshot) for verification.

   Data Exchanged: The request is a multipart/form-data POST containing the payment_id and the receipt file object.

***Process**:* The server validates that the file is an image under 5MB and that the user owns the payment record. The file is stored securely, and the payment record is updated with the file path.

3. **Admin Payment Management**

   Endpoint: POST /admin/payments/gcash/{payment}/approve

   Purpose: Allows administrators to approve a verified payment.

   Process: Upon calling this endpoint, the system updates the payment status to "completed" and records the timestamp and the administrator's ID.

API security is enforced through Laravel's session-based authentication and Role-Based Access Control (RBAC), ensuring students and admins only access appropriate endpoints. All state-changing requests (POST, PUT, DELETE) require a valid CSRF token to prevent attacks. HTTPS encryption (TLS 1.2+) is mandatory for all data in transit.

Responses follow a standardized JSON format. Successful requests return a status of 200 OK or 201 Created, with a body containing success: true and a data object. Error responses (e.g., 400, 401, 422) return success: false with detailed error messages and validation specifics to aid debugging.

Data Flow and User Interface

The flow of data within SOCCSync follows a strictly defined lifecycle, moving from user input at the client interface, through secure API transport, to server-side processing, and finally returning as a structured response.

Data Flow Process

The Data Flow Diagram *(Figure 8)* delineates the trajectory of information within the SOCCSync system, illustrating how inputs from external entities—Students, Administrators, and Deans—are processed, transformed, and stored across the six core functional modules.

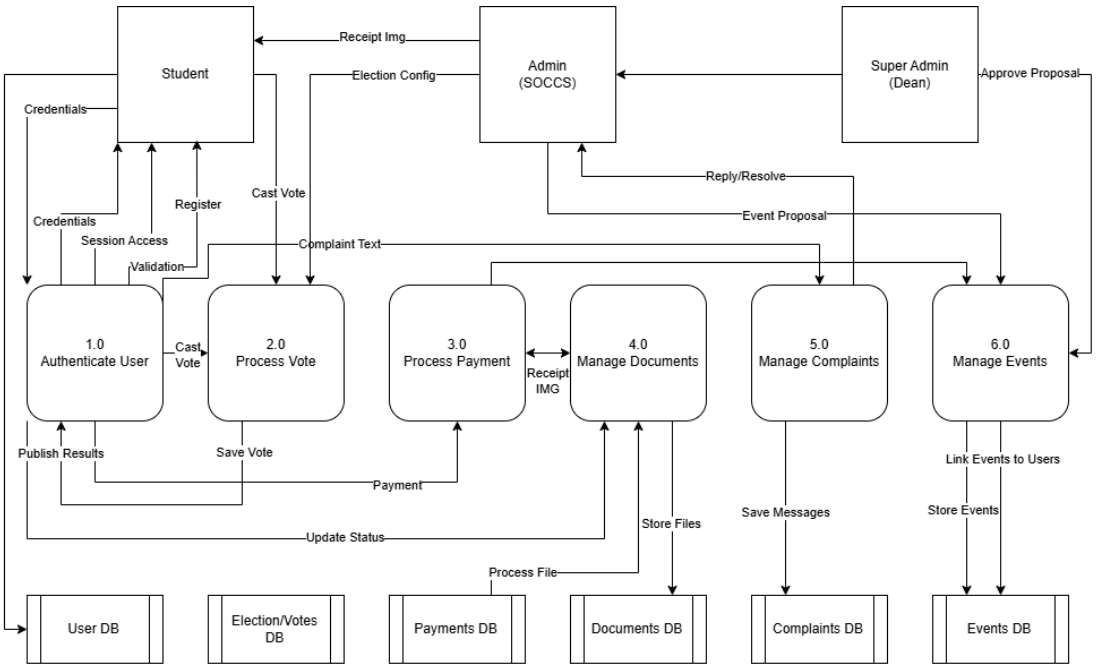


Figure 8. *Data Flow Diagram (DFD) of SOCCSync*

Module 1: User Authentication & Registration. The data flow initiates when a user submits credentials. The system hashes passwords using the bcrypt algorithm and validates the request against the User Database. Upon successful authentication, the system generates a session token and redirects the user to their designated role-based dashboard, effectively transitioning them into the active system environment.

Module 2: Election & Voting System. In the election subsystem, the flow begins with the Administrator configuring parameters in the Election Database. During the voting period, student submissions are processed

through a validation layer to enforce the "one vote per position" constraint. Valid votes are tallied in real-time but remain sequestered until the Administrator triggers the publication protocol, releasing the final results to the student interface.

Module 3: Payment Processing System. Financial data flow commences when a student initiates a transaction and uploads a receipt image. The image file is routed to file storage, while transaction metadata is recorded in the Payments Database with a "Pending" status. Administrators retrieve these records for verification; approval triggers the Dompdf library to generate a PDF receipt, simultaneously updating the payment status to "Completed."

Module 4: Document Management System. The document management flow involves the upload, validation, and archival of student files. Metadata is indexed in the Documents Database, awaiting administrative review. Once an Administrator marks a document as "Verified," the system updates the record, enabling the student to utilize the sharing functionality.

Module 5: Complaint & Feedback System. Communication data flows from student submissions, which are sanitized and stored in the Complaints Database with a "Pending" status. The workflow transitions to the Administrator, who formulates a response and updates the status to "Resolved." This action triggers an automated notification alerting the student that their concern has been addressed.

Module 6: Event Management System. Event data flow originates with an Administrator submitting a proposal to the Events Database, initiating a review workflow for the Dean. Upon approval, the event becomes accessible

for student registration, linking user IDs to the event record. Post-event data processing involves the input of financial figures, which the system compiles into comprehensive attendance and expenditure reports.

Level 1 Data Flow Diagram (DFD) Components The following components constitute the Level 1 DFD structure:

External Entities: The primary sources and destinations of data are the **Student, Administrator (SOCCS),** and **Super Admin (Dean).**

Core Processes: The system logic is encapsulated in six main processes:

1. Authenticate User
2. Process Vote
3. Process Payment
4. Manage Documents
5. Manage Complaints
6. Manage Events

Data Stores: Information is persisted in six dedicated repositories:

1. User DB
2. Election/Votes DB
3. Payments DB
4. Documents DB
5. Complaints DB
6. Events DB

User Interface Design

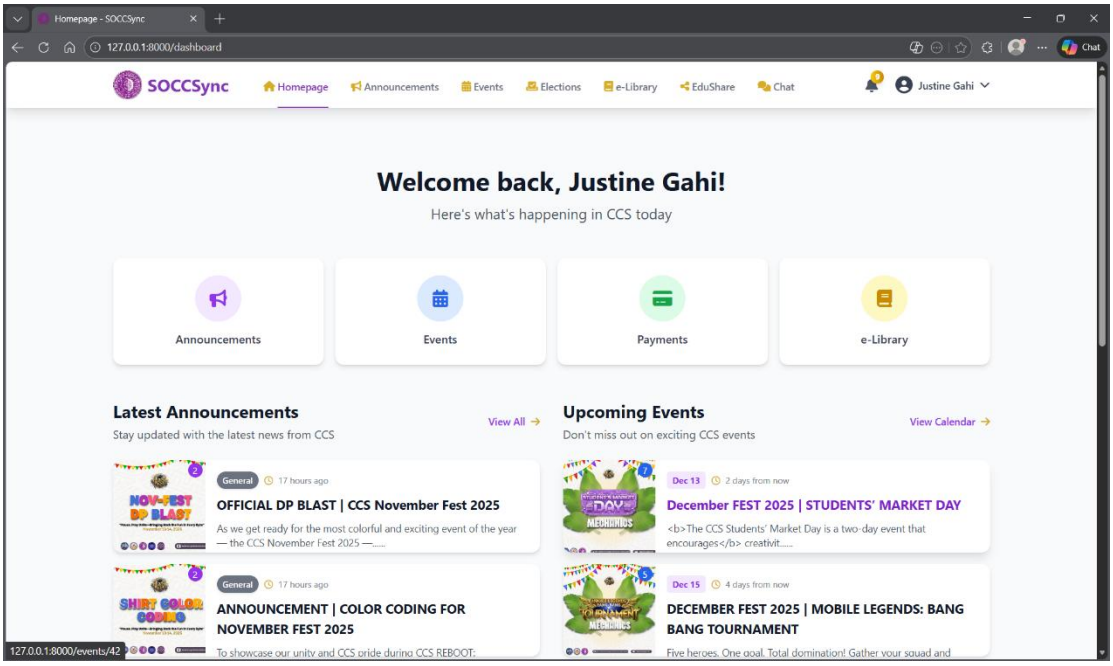


***Figure 9**. Student Dashboard*

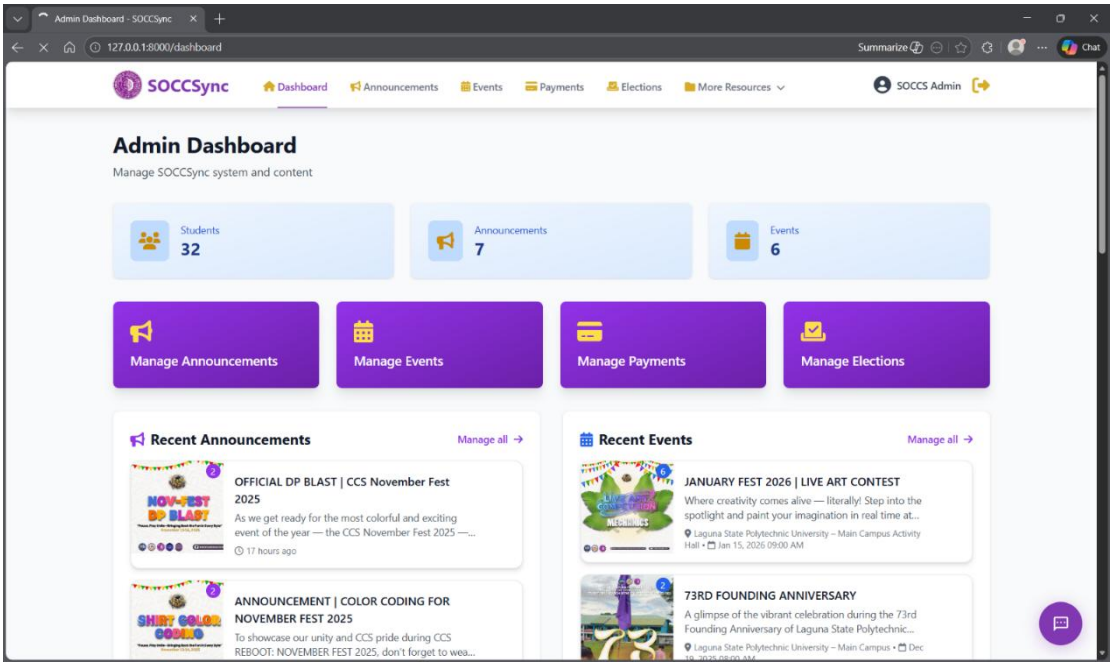


***Figure 10**. Admin (SOCCS) Dashboard*

The user interface is designed with a "mobile-first" approach using the Tailwind CSS framework, ensuring usability across desktops, tablets, and smartphones. The visual identity incorporates the institution's branding, utilizing specific purple variants for primary and secondary elements to foster

organizational identity. The interface is segmented by user role. The **Student Dashboard** prioritizes actionable items, displaying chronological feeds of announcements, upcoming event cards, and payment obligation statuses (e.g., "Paid" or "Pending"). Conversely, the **Admin Dashboard** focuses on oversight, presenting data tables for verification queues, analytics widgets for system usage, and management tools for content creation.

Use Case Analysis

The Use Case Diagram illustrates the functional requirements of the SOCCSync system by depicting the interactions between external agents (Actors) and the system's functional processes (Use Cases). The system identifies three primary actors: the Student, the Administrator, and the Dean/Authority.

The **Student** actor represents the end-user base. Their primary interactions involve initiating financial transactions, participating in democratic elections, and accessing organizational resources. Specifically, students utilize the system to view announcements, register for events, upload GCash receipts for payment verification, and submit organizational documents.

The **Administrator** actor possesses elevated privileges responsible for content management and operational oversight. Their use cases include creating announcements, verifying pending payments, managing event proposals, and configuring election parameters.

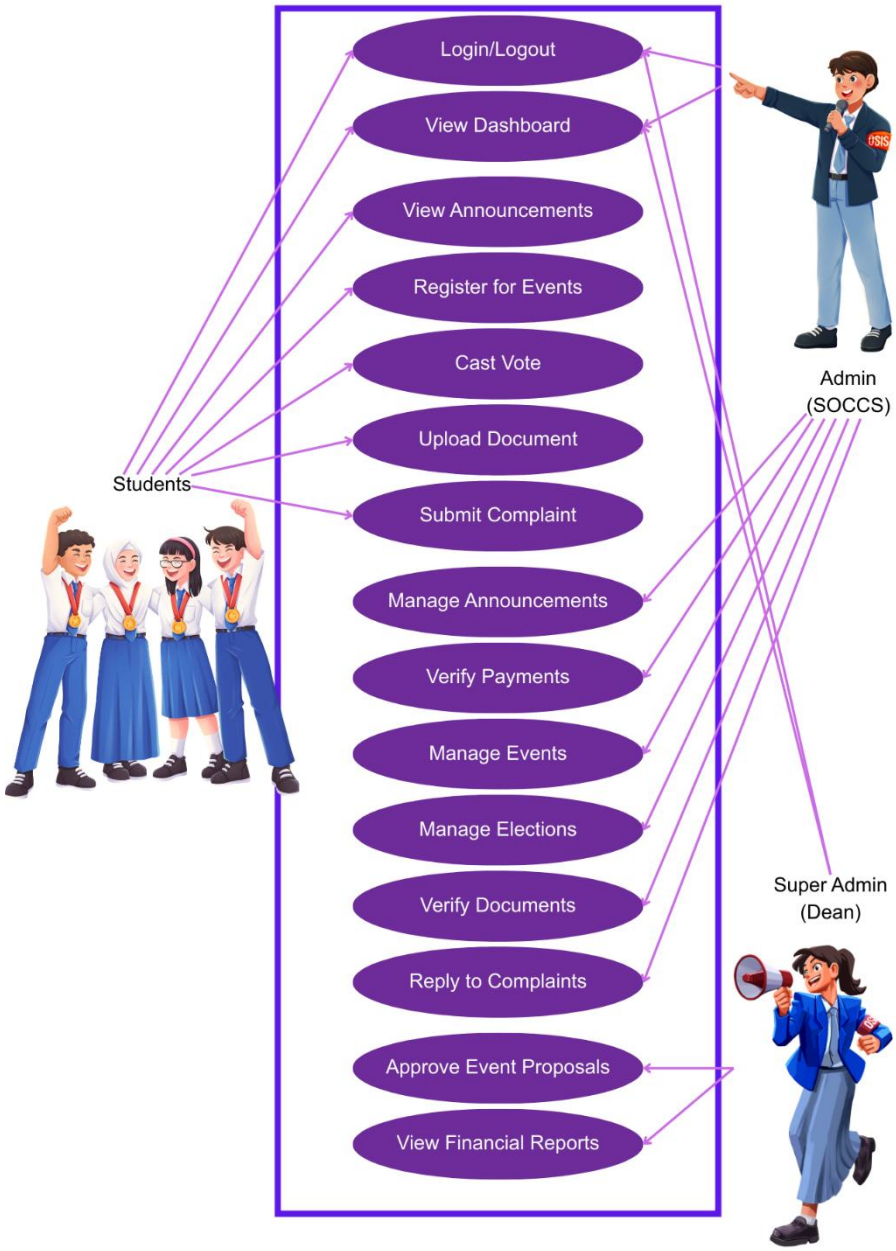


Figure 11. *Use Case Diagram of SOCCSync*

The **Dean** (or Approving Authority) actor plays a supervisory role, primarily interacting with high-level modules such as Event Management for the approval of activity proposals and financial reports.

All actors share common use cases regarding authentication, such as "Login" and "Logout," ensuring secure access to their respective dashboards.

CHAPTER III

IMPLEMENTATION

Technology Stack

The SOCCSync system was developed using a modern full-stack web architecture, carefully selected to ensure scalability, maintainability, and robust performance within the client-server model established in the system design. The technology stack is organized into the following categories:

Backend Framework and Language: The system backend functions on the Laravel Framework version 11.0, a mature PHP framework chosen for its elegant syntax, streamlined application structure, and comprehensive security features. The framework operates on PHP version 8.2, which was selected to provide enhanced performance, type safety, and modern language features such as named arguments and attributes.

Database Management System: To handle data persistence, the system utilizes MySQL version 8.0 as its primary relational database management system. MySQL was selected for its reliability, ACID compliance, and performance capabilities regarding complex relational queries. The database supports UTF-8 MB4 character encoding to ensure proper handling of special characters and multilingual content.

Frontend Technologies: The user interface, functioning as the "thin client," was constructed using Tailwind CSS version 3.x. This utility-first framework enables the rapid development of responsive, mobile-first interfaces. For interactive components and state management without the complexity of heavy frontend frameworks, the system employs Alpine.js version

3.x. Visual consistency is maintained through Font Awesome version 6.4.0 for icon representation.

Server-Side Rendering Engine: The presentation layer utilizes the Blade Templating Engine. This native Laravel system allows for the embedding of PHP logic within HTML while maintaining a separation of concerns, supporting consistent layout structures across student and administrator views.

Document Processing Libraries: The system integrates specific libraries for file generation. Dompdf version 3.1 is used to convert HTML and CSS into PDF documents for receipts and reports. PHPOffice PhpSpreadsheet version 2.0 is utilized for importing and exporting spreadsheet data, facilitating bulk information processing.

Development and Testing Tools: To ensure code quality, the development environment incorporates PHPUnit version 11.0 for testing, Laravel Pint version 1.13 for code styling, Mockery version 1.6 for unit testing objects, and FakerPHP version 1.23 for generating realistic test data.

System Modules

The SOCCSync system is architecturally decomposed into eleven distinct functional modules. Each module addresses specific organizational needs while maintaining integration through shared data models and authentication layers.

User Authentication and Authorization Module: This foundational module manages the user lifecycle, including registration, login, and session management. It implements role-based access control (RBAC) for super administrators, administrators, and students, ensuring that permissions govern

access to specific features. Security is enforced via bcrypt password hashing and CSRF token protection.

Announcement Management Module: Serving as the primary communication channel, this module allows administrators to broadcast information categorized as general, urgent, event-related, or academic. It supports rich text content, image attachments, and scheduled publishing, ensuring students receive timely updates via their dashboards.

Event Management and Registration Module: This module handles the full event lifecycle, from proposal submission to financial reporting. Administrators create event records which undergo an approval workflow involving the review of activity plans and budgets. Once approved, students can register for events, and post-event reporting allows organizers to document expenditures and attendance analytics.

Payment Processing and Verification Module: This module facilitates financial transactions by allowing administrators to create payment announcements for fees. Students submit payments via GCash (uploading receipts) or manual processing. The system generates unique transaction IDs, and administrators verify payments by reviewing receipts. Upon approval, the system generates PDF receipts and updates financial reports.

Election and Voting Module: To facilitate student leadership selection, this module offers a secure voting platform. Administrators configure campaigns and register candidates. The system enforces unique voting constraints to ensure one vote per position per student. Results remain concealed until officially published by administrators, at which point real-time tallies become visible.

Document Management Module: This module provides centralized storage for student documents such as Certificates of Registration and medical certificates. Administrators verify uploaded documents, which can then be selectively shared by students for collaborative requirements.

Digital Library Module: The e-Library provides centralized access to educational resources like PDFs and presentation files. Resources are tagged with metadata for easy discovery, and the system tracks usage analytics to ensure students have equal access to study materials.

EduShare Collaborative Platform Module: EduShare facilitates peer-to-peer file sharing with a hierarchical folder structure. Students can upload academic materials, organize them into folders, and control permissions to share items publicly or with specific individuals.

Complaint and Feedback Module: This module enables students to submit complaints or inquiries, either anonymously or with attribution. Administrators track these submissions through pending, reviewed, and resolved states, providing written responses to ensure student concerns are addressed.

Administrative and AI Chat Modules: The system includes a real-time messaging module for direct student-admin communication and an AI-powered chatbot for handling routine queries about membership and procedures. The chatbot reduces administrative workload, while complex queries are redirected to human administrators.

Dashboard and Analytics Module: Acting as the central navigation hub, this module presents role-specific information. Student dashboards show announcements and payment obligations, while admin and dean dashboards display system-wide statistics, approval queues, and financial analytics.

Testing and Debugging

The development process incorporated systematic testing methodologies to ensure system reliability and data integrity, combining automated frameworks with manual validation.

Testing Methodology

Manual User Acceptance Testing (UAT): This was the primary strategy, conducted iteratively. Test accounts representing all roles were used to exercise every functional pathway, validating both successful operations and error handling.

Unit Testing: Implemented using PHPUnit version 11.0, these tests focused on critical business logic, data validation rules, and financial calculations.

Integration Testing**:** Validated the interaction between modules, such as ensuring payment verification triggered notifications or that vote submissions correctly updated candidate tallies. It also verified the handling of circular dependencies in the election system.

Database Testing: Addressed the complex relational structure of twenty-six tables, verifying cascade deletes, foreign key constraints, and composite unique constraints.

Debugging Tools

Browser Developer Tools: Chrome DevTools was used for frontend debugging, monitoring network requests, and inspecting DOM manipulation.

Laravel Debug Bar: Integrated to provide real-time insights into application performance, exposing executed database queries to identify optimization opportunities and N+1 query problems.

Log Analysis: The system's logging configuration captured application errors and stack traces. Custom log statements were used to trace execution flow in complex logic.

Challenges and Solutions

The development process encountered technical challenges which were resolved through specific architectural solutions.

Secure Payment Verification with Image Upload

Challenge: The system needed to securely handle GCash receipt uploads, preventing malicious files while allowing administrators to view receipts for verification.

Solution: A comprehensive security strategy was applied involving client-side and server-side validation of MIME types and extensions. Files were stored outside the public web root with randomized filenames. Access was mediated through a controller ensuring only authorized admins could view the financial documents.

Optimizing Database Query Performance

Challenge: As the database grew, views displaying lists of records (e.g., events with creators) generated excessive queries due to the N+1 query problem, impacting performance.

Solution: Laravel’s Eloquent ORM eager loading was applied to load related records in single optimized queries. Database indexes were added to foreign keys and frequently queried columns, and result caching was implemented for static data.

Ensuring Data Consistency During Concurrent Operations

Challenge: Multiple users performing simultaneous operations, such as voting or verifying payments, could lead to race conditions and data inconsistency.

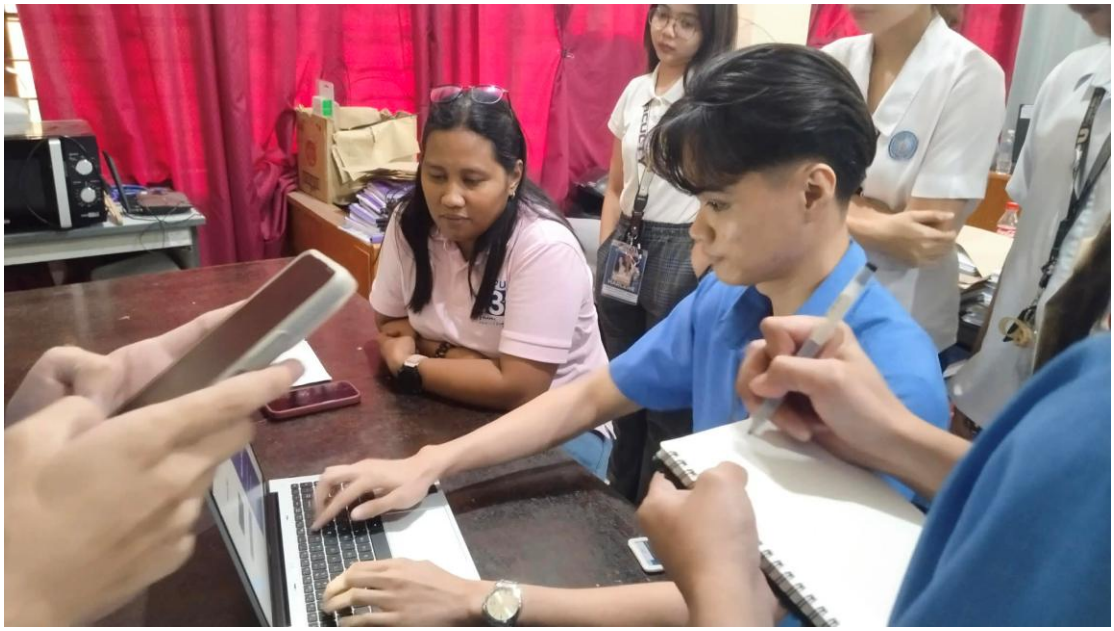
Solution: Database transactions were implemented to wrap related operations, ensuring atomicity. Unique constraints at the database level prevented duplicate entries (e.g., one vote per student). Optimistic locking was employed to prevent administrators from overwriting each other's updates.

APPENDICES

Appendix A
Approved Letter

Appendix B
Photo Documentation

Action Photo



Formal Photo

REFERENCES

Al-Samarraie, H., & Saeed, N. (2021). *A systematic review of cloud computing in education: A research synthesis. Journal of Information Technology Education: Research, 17, 307-336. https://doi.org/10.28945/4060*

García-Peñalvo, F. J., Corell, A., Abella-García, V., & Grande, M. (2021). *Learning management systems in higher education: A systematic review. Education in the Knowledge Society, 22, e23663. https://doi.org/10.14201/eks. 23663*