ITE 2952 - Programming Group Project

School Fees Management System (SFMS)

Group: Dark Devils

Prepared by:

Croos SJ (E2248507)

Gowsihan V (E2248509)

Thanusan P (E2245667)

(Date: May 7, 2025)

University of Moratuwa

Bachelor of Information Technology

Abstract

The School Fees Management System (SFMS) is a comprehensive web-based application meticulously designed and developed to automate, streamline, and enhance the critical process of managing student fees within educational institutions. This project directly addresses the prevalent inefficiencies, potential for human error, and lack of real-time transparency inherent in traditional, often manual, fee collection and administration systems. The SFMS delivers a centralized, secure, and intuitively designed platform catering to the distinct needs of various stakeholders: administrators, financial staff, students, and parents.

Core functionalities of the implemented system include robust user authentication with granular role-based access control (RBAC), flexible and detailed management of fee structures (including categories, frequencies, and class-specific applicability), comprehensive discount type administration, and meticulous student record management with an enhanced system for linking students to multiple parent/guardian accounts. The system capably supports the entire fee lifecycle, encompassing: manual but systematic invoice generation with duplicate prevention; offline payment recording supplemented by a secure payment proof upload and verification workflow; application and removal of predefined discounts to invoices with automatic recalculation of financial totals; processing of refunds against previously recorded payments; and a detailed, immutable audit trail for all significant financial and administrative actions.

A key component is the dedicated student/parent portal, which provides secure access to view detailed fee statements, track payment history, manage user profiles (including password changes), and submit payment proofs for offline transactions. The portal is designed to handle scenarios where parents are linked to multiple children, offering a clear selection mechanism. For administrators and staff, the SFMS features an extensive reporting module, delivering insights through Fee Collection Reports (detailed and summarized by various criteria), Outstanding Dues Reports, individual Student Ledgers, Defaulter Lists, Accounts Receivable Ageing Reports, Payment Transaction Details, and a comprehensive Audit Log. Furthermore, a system settings module empowers administrators to manage non-sensitive, application-wide configurations such as school contact information, bank details for display, currency symbols, and default notification parameters, all stored dynamically in the database.

Developed using a technology stack comprising PHP for backend logic, MySQL as the relational database, and Bootstrap 5 for a responsive and modern user interface, the SFMS embodies contemporary web development practices. Security has been a paramount consideration, with features like CSRF protection on all state-altering forms, secure password hashing (bcrypt), consistent use of prepared SQL statements (PDO) to mitigate injection vulnerabilities, and secure session management.

This report provides a thorough account of the project, commencing with an introduction to the problem domain and project objectives. It delves into related research concerning existing fee management solutions and relevant technologies. A detailed system analysis and design

chapter outlines the requirements and the architectural blueprint, conceptually describing the necessary UML diagrams (Use Case, Activity, Sequence, Class, ER/EER). Subsequent chapters elaborate on the technologies adopted, highlight key implementation details and system interfaces, and the report concludes with a summary of the project's achievements, challenges encountered during development, and a roadmap for potential future enhancements and expansions of the SFMS.

Table of Contents

Chapter 1 - Introduction

1.1 Problem Statement

Educational institutions, ranging from primary schools to higher education centers, particularly within contexts like Sri Lanka, are consistently faced with the operational complexities and administrative burdens of managing student fees. Traditional methodologies, which often involve manual data entry into ledgers or basic spreadsheets, manual receipting, and paper-based record-keeping, are fraught with inherent limitations. These include a high propensity for human error in calculations and data entry, significant time consumption for administrative staff performing repetitive tasks, lack of real-time visibility into financial status for both the institution and parents, and difficulties in generating accurate and timely financial reports for auditing and decision-making.

Furthermore, manual systems often struggle with the diverse and sometimes intricate fee structures that schools employ (e.g., varying fees per grade, optional subject fees, ad-hoc charges, installment plans). Tracking payments across multiple methods (cash, cheque, bank transfers), managing concessions or discounts, processing refunds, and effectively following up on overdue payments become increasingly challenging as student numbers grow. This not only impacts the financial health and operational efficiency of the institution but also affects the experience of parents and students, who expect clarity, convenience, and timely information regarding their financial obligations. The absence of a centralized, automated system can lead to communication gaps, payment disputes, and an overall perception of inefficiency. The increasing adoption of digital technologies in all sectors, including education, underscores the pressing need for a modern, secure, and integrated solution to replace these outdated practices.

1.2 Aim

The primary aim of this project is to engineer and deliver a robust, secure, and intuitive web-based School Fees Management System (SFMS). This system is designed to comprehensively automate and streamline the entire lifecycle of student fee management within an educational institution. Key goals include simplifying the definition and application of complex fee structures, facilitating efficient and transparent payment recording (including offline methods with proof verification), enabling effective discount and refund management, and providing powerful reporting tools for financial oversight. The SFMS endeavors to significantly reduce administrative overhead, minimize errors, improve cash flow visibility, enhance communication with parents/students, and provide a modern, accessible platform for all stakeholders involved in the fee management process.

1.3 Objectives

To achieve the stated aim, the SFMS project was guided by the following specific objectives:

1. **User Management & Security:** To design and implement a secure system for user authentication and authorization, featuring distinct roles (Administrator, Staff, Student,

Parent) with granular, role-based access control (RBAC) to system functionalities. This includes secure password management and CSRF protection on all state-changing operations.

2. **Student Administration:** To develop a comprehensive student management module allowing for the creation, viewing, updating, and deletion (CRUD) of student records, including the ability to link students to their own portal user accounts and to multiple parent/guardian user accounts via a dedicated linking table. Automated, unique admission number generation is also a key objective.

3. **Fee Setup & Configuration:** To create an administrative interface for defining and managing:

    a. Fee Categories (e.g., Tuition, Transport, Lab).

    b. Discount Types (e.g., Sibling Discount, Scholarship, Staff Ward Concession) with options for percentage-based or fixed-amount values.

    c. Detailed Fee Structures, linking fee categories to academic sessions and specific classes (or all classes), defining amounts, payment frequencies (e.g., monthly, termly, annual), due day rules, and applicable late fee policies.

4. **Invoice Lifecycle Management:** To implement a system for:

    a. Manual generation of fee invoices by administrators/staff for selected classes and academic sessions based on applicable fee structures, with robust checks to prevent duplicate invoice generation for the same student/structure/session.

    b. Application and removal of defined discounts to individual invoices, with automatic recalculation and updating of the invoice's `total_discount` and `total_payable` fields.

5. **Payment & Refund Processing:** To facilitate:

    a. Recording of offline (manual) payments by administrators/staff, linking these payments to specific invoices, updating invoice payment statuses (`total_paid`, `status`), and generating unique receipt numbers.

    b. A mechanism for students/parents to upload payment proofs for offline transactions via the portal.

    c. An administrative interface for viewing, verifying (approving), or rejecting uploaded payment proofs, with approved proofs leading to formal payment recording.

d. Processing of refunds against previously completed or partially refunded payments, updating the original payment record and appropriately adjusting the related invoice's financial totals and status.

6. **Notification System:** To establish a notification service capable of:

    a. Managing notification templates (for payment confirmations, fee due soon reminders, overdue payment reminders) stored in the database.

    b. Preparing notification messages by merging dynamic data (e.g., student name, invoice details) into these templates.

    c. Logging all notification attempts (with status: pending, sent, failed) to a dedicated `notifications` table.

    d. Sending notifications via email (using PHPMailer and configurable SMTP settings), with a manual trigger for administrators to initiate fee reminder dispatches, incorporating a cooldown logic to prevent re-sending the same reminder too frequently.

7. **Reporting Module:** To develop a suite of reports for administrators/staff, providing insights into financial status and operations, including: Fee Collection (detailed and summary views), Outstanding Dues, Student Ledger, Defaulter List, Ageing Report, Payment Transaction Detail, and a system Audit Log. Reports are to feature filtering options and pagination.

8. **Student/Parent Portal:** To create a secure, dedicated portal for students and parents offering:

    a. Dashboard view with summary information (and child selection for parents with multiple children).

    b. Detailed view of their fee invoices, payment history, and outstanding balances.

    c. Ability to view school bank details and upload payment proofs.

    d. Profile management features (update contact details, change password).

9. **System Settings Management:** To provide an administrative interface for managing non-sensitive, application-wide configurations (e.g., school name, address, contact, currency symbol, bank details for display, default email "From" name, reminder schedule parameters) stored dynamically in the `system_settings` database table.

10. **Technology & Architecture:** To build the system using PHP for backend logic, MySQL as the relational database, and Bootstrap 5 for a responsive frontend. The architecture should be modular, with considerations for security (CSRF, SQL injection prevention, secure password hashing), maintainability, and usability.

Chapter 2 - Related Research

The development of an effective School Fees Management System (SFMS) is a well-addressed area in educational technology, driven by the universal need for efficient financial administration in schools. This research phase involved an investigation into existing solutions, prevalent technologies, common challenges, and best practices in the domain of educational financial administration, with a particular focus on identifying features and approaches relevant to the context of educational institutions, including those in Sri Lanka.

2.1 Overview of Existing Fee Management Systems & Market Solutions:

The market for educational administration software is mature, offering a spectrum of solutions that address fee management to different extents. These systems can be broadly categorized into integrated Enterprise Resource Planning (ERP) systems, standalone fee management software, and open-source platforms.

- **Integrated School ERP Systems:**
    - **Description:** These are large-scale systems designed to manage nearly all aspects of a school's operations, including admissions, academics, human resources, library, transport, and finance, where fee management is a core module.
    - **Examples:** Internationally recognized systems include Ellucian Banner, Oracle PeopleSoft Campus Solutions, and SAP for Education. Regionally, systems like *Vidyalaya School Software* and *SkoolBeep* offer comprehensive ERP functionalities tailored to local markets [1].
    - **Fee Management Capabilities:** ERPs typically provide highly configurable fee structure definition (e.g., per course, per student category, per year/term), automated invoicing based on enrollment data, diverse online payment gateway integrations (supporting credit/debit cards, net banking, mobile wallets like UPI in India, or local equivalents), sophisticated scholarship and concession management tools, real-time payment tracking, automated digital receipting, and seamless integration with general ledger accounting modules.
    - **Pros:** Offer a holistic view of institutional operations, promote data consistency across departments, and possess extensive feature sets.
    - **Cons:** Generally involve high licensing and implementation costs, can be complex to customize and require significant user training, and may include many features that are superfluous for smaller or medium-sized institutions.

- **Standalone Fee Management Software:**
    - **Description:** These are specialized software products focusing primarily on the fee collection and management lifecycle. They are available as cloud-based Software-as-a-Service (SaaS) offerings or as on-premise solutions.
    - **Examples:** Products like *SchoolOnApp*, Teno, MyClassboard, and international solutions such as FACTS Management or Blackbaud Tuition Management often fall into this category.
    - **Fee Management Capabilities:** These systems usually provide more user-friendly interfaces specifically for fee setup, student billing, tracking of online and offline payments, receipt generation, automated reminder systems, and basic to intermediate financial reporting. Many offer APIs or export functionalities for integration with existing Student Information Systems (SIS) or accounting software.
    - **Pros:** Often more affordable and quicker to deploy than full ERPs, generally easier to use for dedicated financial tasks, and can be more agile in adopting new payment technologies.
    - **Cons:** May require manual data synchronization if not tightly integrated with other school systems, potentially leading to data silos. The breadth of features might be less extensive than an ERP's financial module.
- **Open-Source School Management Platforms:**
    - **Description:** Platforms like Fedena (Ruby on Rails), OpenEduCat (built on Odoo/Python), Gibbon (PHP), and Moodle (primarily an LMS but with fee-related plugins) offer comprehensive school management functionalities, including fee management, with their source code being freely available.
    - **Fee Management Capabilities:** The capabilities vary significantly. Some, like OpenEduCat, offer robust financial modules due to their ERP underpinnings. Others might provide more basic fee collection, structuring, and reporting, with the potential for extension through custom development or community plugins.
    - **Pros:** No licensing costs, high degree of customizability allowing schools to tailor the system to their exact needs, and often active communities for peer support.
    - **Cons:** Typically require significant in-house or contracted technical expertise for installation, setup, customization, ongoing maintenance, security hardening, and support. The quality and availability of support can be less structured than commercial offerings.

- **Local and Regional Solutions (Context: Sri Lanka):**
    - Research into the Sri Lankan market would identify local software vendors and solutions. These are often developed with a keen understanding of local banking practices (e.g., direct bank deposit norms, specific online banking integrations like Sampath Vishwa or HNB MOMO if gateways support them), common payment methods (e.g., FriMi, eZ Cash, mCash if popular for institutional payments), and any specific regulatory reporting requirements.

2.2 Comparative Analysis of Key Features in Existing Systems:

Feature	**International ERP**	**SaaS Fee Manager**	**Local/OpenSource**	**SFMS (This Project - Implemented)**
Fee Structure Definition	Highly granular, rule-based, complex fee plans, multiple dependencies.	Template-based, easy setup for common structures, moderate flexibility.	Varies; can be highly custom with coding, or basic.	Flexible: Categories, Structures (amount, frequency, class/session specific, late fees).
Invoice Generation	Automated, batch processing, linked to enrollment.	Automated/Manual, scheduled.	Often manual or basic automation.	Manual trigger for class/session, duplicate prevention.
Online Payment Gateways	Supports multiple major international and regional gateways.	Supports selected popular gateways (e.g., Stripe, PayPal, local).	Often via plugins or custom integration.	Planned for future (current: offline with proof).
Offline Payment Handling	Yes, with reconciliation tools.	Yes, often with manual entry.	Yes, manual entry.	Yes, with proof upload and admin verification.

Discount/Concession	Advanced: rule-based, scholarships, financial aid modules.	Fixed/Percentage discounts, manual application.	Basic or requires customization.	Discount Types (fixed/%), manual application per invoice, auto-recalculation.
Refund Processing	Integrated, linked to accounts.	Supported, may be manual.	Varies, often manual.	Admin-processed refunds against payments, updates invoice.
Parent/Student Portal	Comprehensive: view dues, history, make payments, profile, communication.	Core: view dues, history, make payments.	Varies; may be basic or via plugins.	Core: view dues, history, profile, proof upload, multi-child select.
Automated Reminders	Email & SMS, configurable schedules, template management.	Email reminders, often basic scheduling.	Basic or requires setup.	Email (via PHPMailer), DB templates, manual trigger with cooldown, logging.
Reporting	Extensive financial analytics, BI tools, customizable reports.	Standard collection, dues, transaction reports; often exportable.	Basic reports, custom reports via code.	Key operational reports (Collection, Dues, Ledger, Defaulter, Ageing, Audit, Transaction).
Security (e.g., PCI DSS)	High, often certified.	Provider responsibility (if SaaS).	User/Implementer responsibility.	Foundational (CSRF, Hashing, Prepared Stmts). HTTPS deployment

				needed.
Scalability	High, designed for large institutions.	Good for target market (SMEs).	Depends on implementation.	Designed for moderate scale, can be optimized.
Cost Model	High (License, Implementation, Maintenance).	Subscription-based (monthly/annual) .	Low/None (Implementation/ Support costs).	Development cost only.

2.3 Technological Trends and Best Practices in Modern SFMS Development:

The landscape of SFMS development is continuously shaped by evolving technologies and user expectations. Key trends and best practices observed include:

- **Cloud Computing and SaaS Models:** The predominant trend is the shift towards cloud-hosted Software-as-a-Service (SaaS) solutions. This model offers schools benefits such as reduced upfront IT infrastructure costs, automatic software updates and maintenance, scalability to handle fluctuating user loads, and accessibility from any location with an internet connection. Our SFMS, being web-based, is well-suited for cloud deployment.

- **Mobile-First and Responsive Design:** With the pervasive use of smartphones and tablets, providing a seamless and intuitive user experience across all devices is paramount. Responsive web design, as implemented in our SFMS using Bootstrap 5, ensures that administrators, staff, parents, and students can effectively interact with the system regardless of their device [2]. Native mobile applications for parents and students are also an increasing expectation for on-the-go access to fee information, payment options, and notifications.

- **API-Driven Architecture and Integrations:** Modern SFMS solutions are increasingly designed with a robust Application Programming Interface (API) layer [3]. This facilitates seamless integration with other critical school management systems, such as Student Information Systems (SIS) for student enrollment data, Learning Management Systems (LMS), and general ledger accounting software. Such integrations promote data consistency, reduce manual data entry, and provide a more holistic view of institutional operations. While our current SFMS is standalone, its modular structure is conducive to future API development.

- **Enhanced Security and Data Privacy:** Given the sensitive nature of financial and personal data handled, security is a non-negotiable aspect. Best practices include:
    - **Secure Authentication & Authorization:** Multi-factor authentication (MFA) for administrative accounts, strong password policies, and granular role-based access control (RBAC). Our system implements RBAC and secure password hashing.
    - **Data Encryption:** Encryption of data both in transit (using HTTPS, which is a deployment-time configuration for our project) and at rest (database-level encryption or encryption of sensitive fields).
    - **Protection Against Web Vulnerabilities:** Adherence to OWASP Top 10 guidelines, including prevention of SQL Injection (achieved via PDO prepared statements), Cross-Site Scripting (XSS - through consistent output escaping like `htmlspecialchars`), and Cross-Site Request Forgery (CSRF - achieved via synchronizer tokens on all state-changing forms).
    - **Compliance with Data Privacy Regulations:** Adherence to regulations like GDPR (General Data Protection Regulation) and local data protection acts (e.g., Sri Lanka's Personal Data Protection Act, No. 9 of 2022) regarding data collection, consent, storage, and user rights.
    - **Regular Security Audits:** Periodic security assessments and penetration testing.
- **User Experience (UX) and User Interface (UI) Design:** A strong focus on creating intuitive, clean, and user-friendly interfaces is essential for ensuring high adoption rates and minimizing the need for extensive user training. This involves clear information architecture, efficient workflows, and visually appealing design. Our adoption of Bootstrap 5 aimed to address this.
- **Automation of Processes:** A core value proposition of any SFMS is the automation of previously manual and time-consuming tasks. This includes automated invoice generation (though manually triggered in our current version), reminder dispatches, late fee calculations, and receipt generation [4].
- **Data Analytics and Business Intelligence:** Advanced SFMS solutions are increasingly incorporating data analytics features, allowing school administrators to gain deeper insights into fee collection trends, revenue forecasting, defaulter patterns, and the effectiveness of different fee structures or discount policies. This often involves dashboards with data visualizations.

2.4 Challenges in Fee Management and How SFMS Aims to Address Them:

Based on research and common institutional pain points, several challenges in fee management were identified, which our SFMS aims to mitigate:

- **Complexity of Fee Structures:** Many schools have diverse fee components varying by grade, subjects, or student categories. Our SFMS addresses this by allowing the creation of distinct Fee Categories and flexible Fee Structures that can be assigned to specific classes and academic sessions.

- **Manual Reconciliation of Offline Payments:** A significant operational burden, especially in regions where bank transfers and direct deposits are prevalent, is the manual matching of payments to student accounts. The Payment Proof Upload feature in the SFMS portal, coupled with the admin verification workflow, is designed to streamline this process, reduce errors, and provide a clear audit trail.

- **Timeliness of Payments and Follow-ups:** Delayed fee payments impact school cash flow. The SFMS's notification system (for payment confirmations and reminders for due/overdue fees), even with manual triggering in the current version, aims to improve timely communication and reduce the need for manual follow-ups by staff.

- **Transparency for Parents and Students:** Lack of easy access to fee statements and payment history can lead to confusion and disputes. The Student/Parent Portal directly addresses this by providing on-demand access to this information.

- **Data Accuracy and Reporting:** Manual systems are prone to calculation and entry errors. By centralizing data and automating calculations (e.g., invoice totals, balances), the SFMS improves data accuracy. The reporting module then allows for easier generation of financial summaries and lists.

- **Administrative Overhead:** The automation of tasks like invoice generation (even if manually triggered for a batch), payment recording, and reminder preparation reduces the administrative workload on school staff, allowing them to focus on other critical tasks.

Chapter 3 - System Analysis & Design

This chapter details the analysis of requirements and the design approach taken for the School Fees Management System (SFMS).

3.1 Requirement Analysis

The SFMS was developed to meet a comprehensive set of functional and non-functional requirements, ensuring a robust and user-centric application.

3.1.1 Functional Requirements Implemented (Summary):

The system successfully implements the core functionalities required for effective fee management, as detailed in Objective 1.3 of this report. These include:

- **User Management & Security:** Secure authentication, role-based access (Admin, Staff, Student, Parent), user CRUD by Admin, CSRF protection.
- **Student Administration:** Student CRUD by Admin/Staff, automated admission number generation, linking students to their own portal accounts and to multiple parent/guardian accounts.
- **Fee Setup & Configuration:** Admin/Staff management of Fee Categories, Discount Types (percentage/fixed), and detailed Fee Structures (amount, frequency, session/class applicability, late fee policies).
- **Invoice Lifecycle Management:** Manual batch invoice generation by Admin/Staff with duplicate prevention. Admin/Staff application/removal of discounts to invoices with automatic recalculation of invoice totals.
- **Payment & Refund Processing:** Admin/Staff recording of offline payments with receipt generation, linking to invoices, and status updates. Student/Parent portal for payment proof uploads, followed by Admin/Staff verification/rejection. Admin/Staff processing of refunds against payments, with corresponding updates to payment and invoice records.
- **Notification System:** Database-driven templates for payment confirmations and fee reminders. A `NotificationService` prepares and logs notifications, with integrated PHPMailer for email dispatch. Manual trigger for reminders with cooldown logic.
- **Reporting Module (Admin/Staff):** Generation of Fee Collection (Detailed & Summary), Outstanding Dues, Student Ledger, Defaulter List, Ageing, Transaction Detail, and Audit Log reports, featuring filtering and pagination.
- **Student/Parent Portal:** Secure login, dashboard (with multi-child selector for parents and outstanding balance display), "My Fees & Payments" page (listing invoices and

payment history), detailed invoice view (including discounts), offline payment instructions with proof upload, and profile management (details & password).

- **System Settings (Admin):** UI for managing non-sensitive configurations (School Info, Bank Details, Currency, Email "From" Name, Reminder Timings) stored in the `system_settings` table.

3.1.2 Non-Functional Requirements Considered:

The development of SFMS paid close attention to critical non-functional aspects to ensure a quality system:

- **Performance:**
    - Database queries are optimized with indexes on frequently accessed columns (e.g., foreign keys, status fields, dates).
    - Pagination is implemented for views displaying potentially large datasets (e.g., user lists, invoice lists, reports like Audit Log and Transaction Detail) to ensure reasonable page load times.
    - The PHP/MySQL stack is generally capable of handling the expected load for a typical school environment, as outlined in the SRS (e.g., target of < 3 seconds page load, < 5 seconds transaction processing) [1].
- **Security:**
    - **Authentication:** User passwords are securely hashed using PHP's `password_hash()` function (bcrypt algorithm by default).
    - **Authorization:** Role-Based Access Control (RBAC) is enforced in controllers to restrict access to functionalities based on predefined user roles (Admin, Staff, Student, Parent).
    - **CSRF Protection:** The synchronizer token pattern is implemented for all forms submitting data via POST requests to prevent Cross-Site Request Forgery attacks.
    - **SQL Injection Prevention:** All database queries are executed using PDO prepared statements with bound parameters, eliminating the risk of SQL injection.
    - **Input Validation:** Server-side validation is implemented for user inputs to check for required fields, correct data types (e.g., integers, floats, valid email formats, dates), string lengths, and adherence to specific business rules (e.g., refund amount not exceeding paid amount). This helps maintain data integrity and prevent errors.

- **Session Management:** Secure session handling practices are followed, including session regeneration upon login (`session_regenerate_id(true)`).
- **Secure File Uploads:** Payment proof files are stored in a directory outside the public web root (`storage/payment_proofs/`). Filenames are sanitized, and unique names are generated. Files are served via a controller action that includes authorization checks.
- **Sensitive Data Configuration:** Critical credentials like database passwords and email server passwords are configured using a `.env` file, separate from the version-controlled codebase, as per security best practices.
- **HTTPS:** While not implemented during local development, the system is designed to be deployed over HTTPS in a production environment to encrypt all data in transit.

- **Usability:**
  - The user interface for both Admin and Portal sections is built using Bootstrap 5, providing a responsive, modern, and consistent look and feel across different devices.
  - Clear navigation menus and logical grouping of features aim to provide an intuitive user experience.
  - Use of icons (Bootstrap Icons) enhances visual understanding and action affordance.
  - Flash messages (success, error, info) provide immediate feedback to users after performing actions.
  - Searchable dropdowns (using Select2) are implemented for student selection in reports and other areas with potentially long lists.
  - Forms are designed for clarity with appropriate labels, placeholders, and inline validation feedback.

- **Maintainability:**
  - **Modular Code Structure:** The application is organized into logical modules (e.g., `Auth`, `UserManagement`, `StudentManagement`, `FeeManagement`, `Reporting`, `Portal`, `Admin`) located in the `app/Modules/` directory. This promotes separation of concerns and makes the codebase easier to understand and manage.

    - **Separation of Concerns (MVC-like):** Controllers handle HTTP requests and business logic, Views handle presentation, and Services/Helpers encapsulate specific functionalities.
    - **BaseController:** A common `BaseController` provides shared methods for view loading, redirection, role checking, and CSRF handling, reducing code duplication.
    - **Service Layer:** Dedicated service classes (`AuditLogService`, `NotificationService`, `SequenceGenerator`) manage specific business processes or utilities.
    - **Helper Classes:** `SettingsHelper` provides a clean API for interacting with database-driven system settings.
    - **Code Commenting:** PHP code includes comments to explain the purpose of classes, methods, and complex logic blocks.
    - **Dependency Management:** Composer is used for managing external PHP libraries.
- **Reliability:**
    - **Database Transactions:** Critical multi-step database operations (e.g., recording a payment which involves updating `payments`, `payment_allocations`, and `fee_invoices` tables; applying discounts; processing refunds) are wrapped in database transactions (`beginTransaction`, `commit`, `rollBack`) to ensure data atomicity and consistency. If any step fails, all changes are rolled back.
    - **Error Handling:** `try...catch` blocks are used for database operations and other potentially failing operations. Errors are logged to the PHP error log for administrative review, and user-friendly messages are typically displayed to the user via flash messages.
    - **Data Integrity:** Foreign key constraints are defined in the database schema to maintain relational integrity. Input validation further helps ensure data quality.

3.2 Design Diagrams

3.2.1 Use Case Diagram

The Use Case diagram will depict the system's functionalities from the perspective of its users.

- **Actors:**
    - `Administrator`: Highest level of access.

    - `Staff`: Performs daily financial and administrative tasks.
    - `Student`: Accesses personal fee information via the portal.
    - `Parent`: Accesses linked child(ren)'s fee information via the portal.
- **Primary Use Cases (Grouped by Actor/Functionality):**
    - **Authentication Module:**
        - `Login` (Associated with: All Actors)
        - `Logout` (Associated with: All Actors)
        - `Change Own Password` (Associated with: All logged-in Actors, via their respective profile pages)
    - **Admin & Staff Shared Core Fee Operations:**
        - `Manage Fee Categories` (CRUD)
        - `Manage Fee Structures` (CRUD)
        - `Manage Discount Types` (CRUD, Toggle Status)
        - `Generate Invoices` (Batch process for class/session)
        - `View Invoice Details (Admin)` (Includes discount management and payment listing)
        - `Apply Discount to Invoice`
        - `Remove Discount from Invoice`
        - `Record Payment` (Manual entry)
        - `Verify Payment Proof` (Includes listing pending, viewing proof, approving by recording payment, rejecting)
        - `Process Refund`
        - `View Reports` (Access to various financial and operational reports)
        - `Trigger Fee Reminders`
    - **Administrator-Specific Use Cases:**
        - `Manage Users` (CRUD for all user types and roles)

        - `Manage Students` (CRUD for student records)
        - `Link/Unlink Guardians to Student`
        - `Link/Unlink Student Login Account`
        - `Manage System Settings`
        - `View Audit Log`
    - **Student Portal Use Cases:**
        - `View Portal Dashboard (Self)`
        - `View My Fees & Payments (Self)`
        - `View My Invoice Detail (Self)`
        - `Upload Payment Proof (Self)`
        - `Manage My Profile (Self)`
    - **Parent Portal Use Cases:**
        - `View Portal Dashboard (Child)`
        - `Select Child to View` (If multiple children linked)
        - `View Child's Fees & Payments`
        - `View Child's Invoice Detail`
        - `Upload Payment Proof (for Child)`
        - `Manage My Profile (Parent)`
- **Relationships:**
    - Associations connecting actors to the use cases they perform.
    - Consider using `<include>` for `Login` as a prerequisite for most other use cases.
    - Generalization can show `Student` and `Parent` as types of `Portal User`, and `Admin` and `Staff` as types of `Administrative User`.

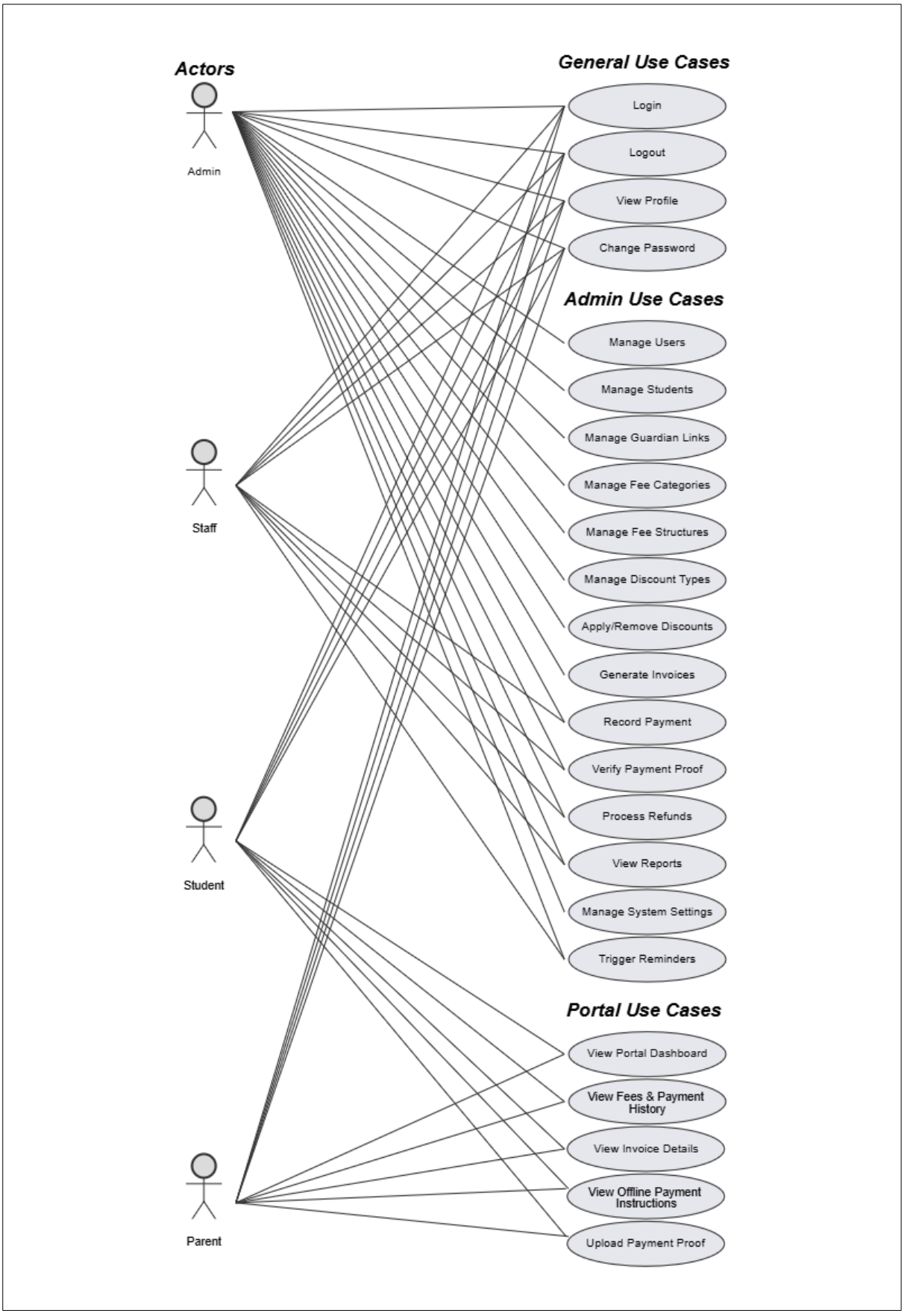


Figure 1: SFMS - Use Case Diagram

3.2.2 Activity Diagrams

A. Admin Records Payment (Verifying an Uploaded Proof):

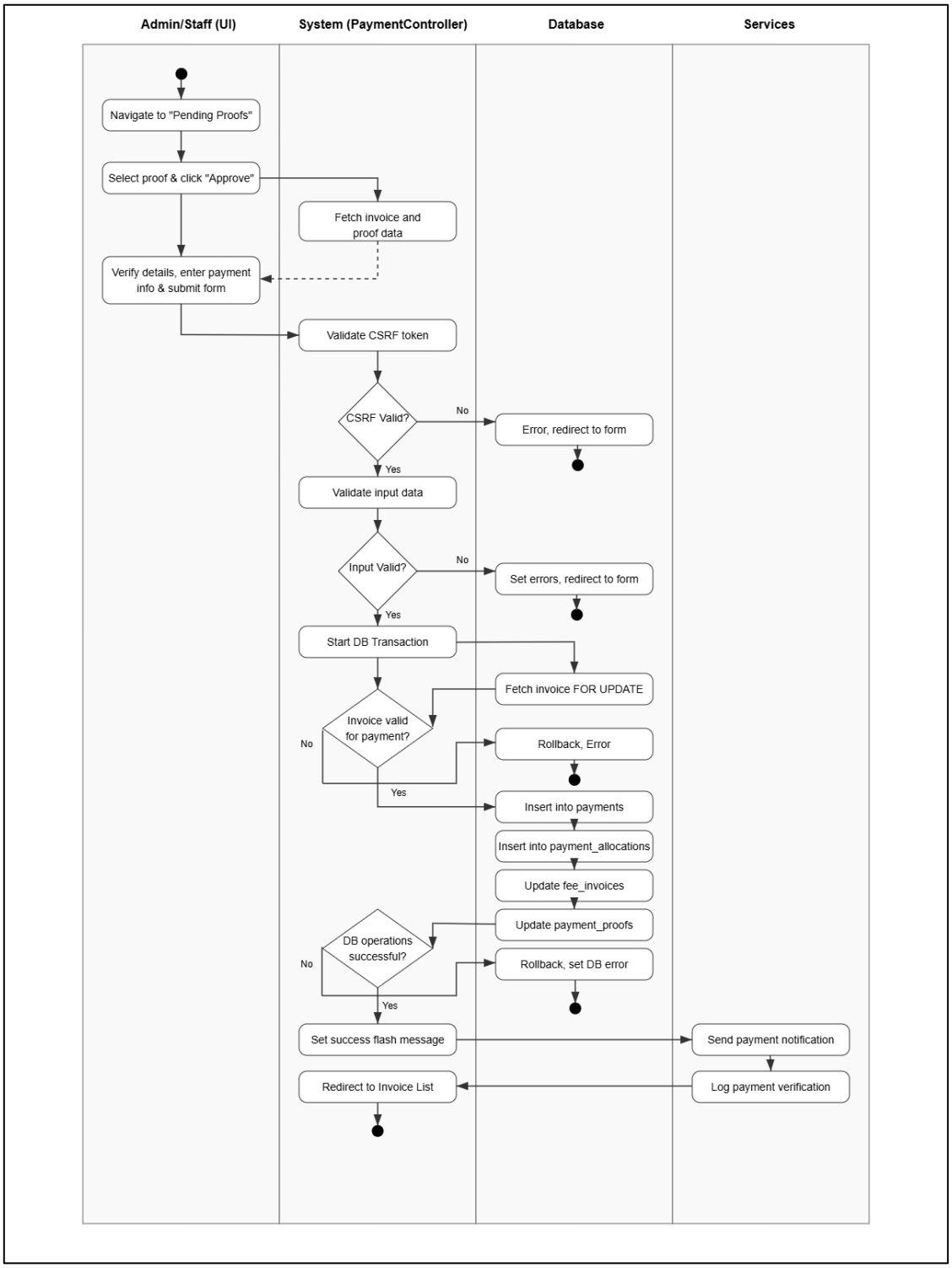


Figure 2: Activity Diagram - Admin Records Payment (Verifying an Uploaded Proof)

B. Parent with Multiple Children Selects a Child and Views Fees:

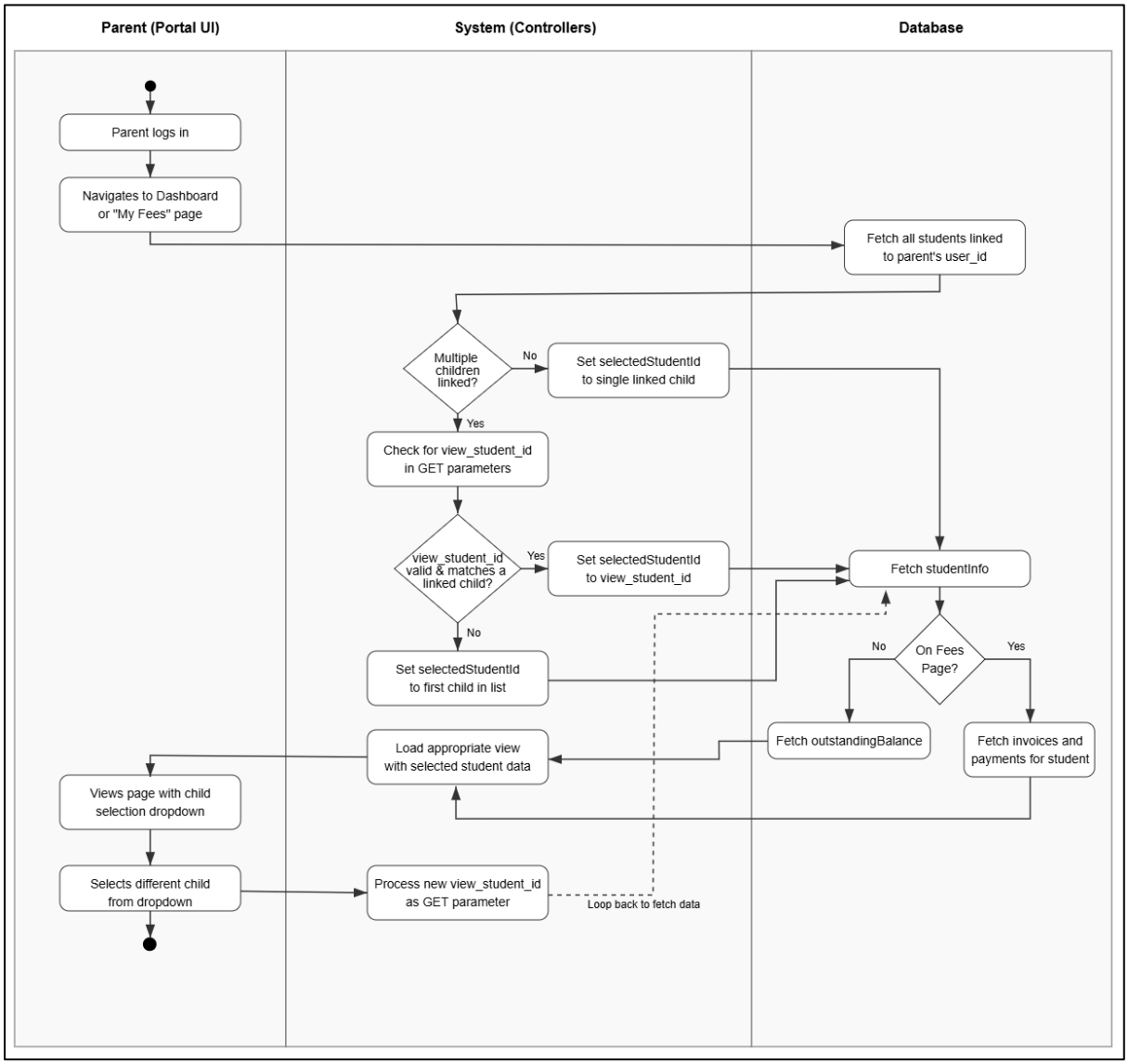


Figure 3: Activity Diagrams - Parent with Multiple Children Selects a Child and Views Fees

C. Admin Applies a Discount to an Invoice:

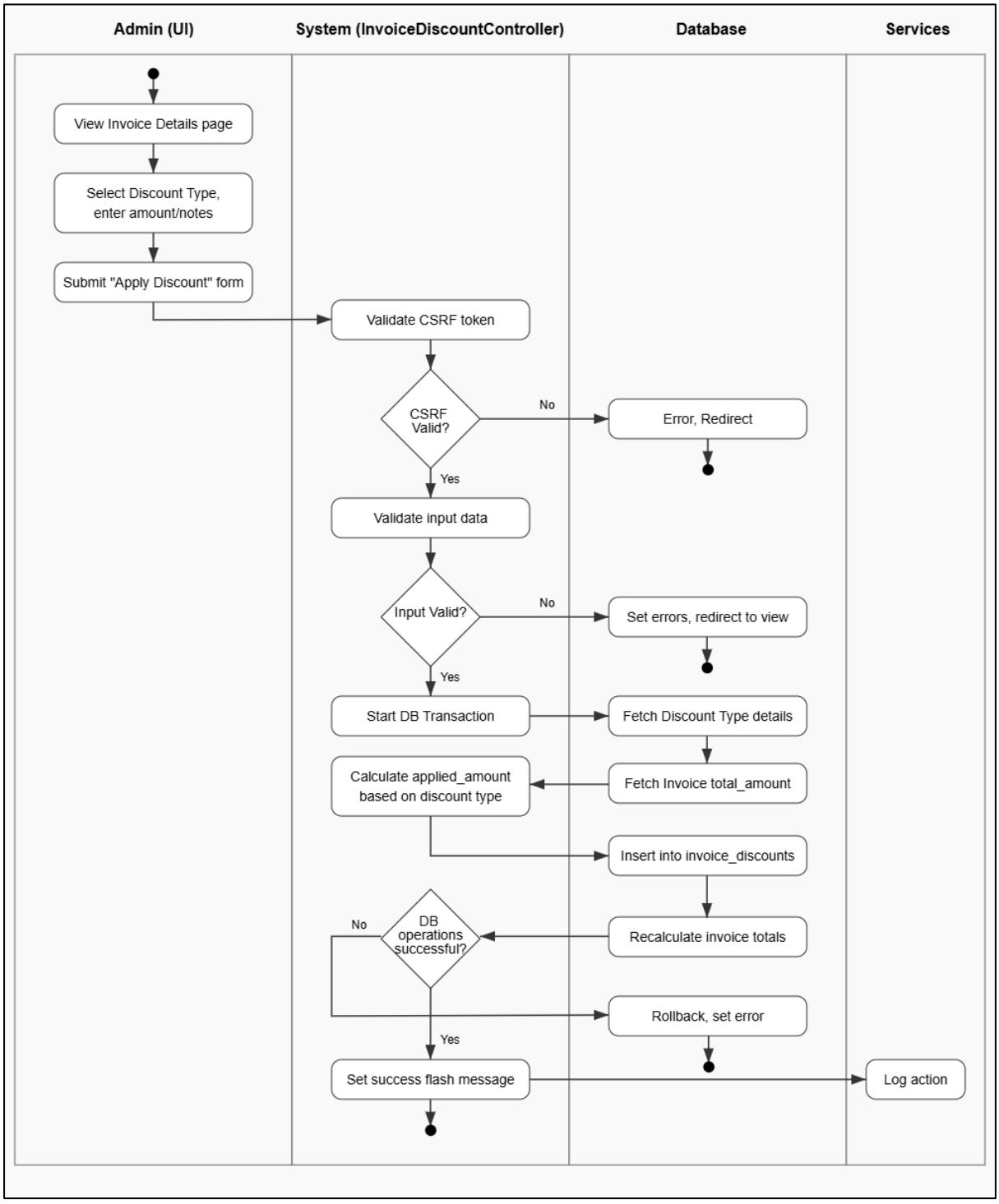


Figure 4: Activity Diagrams - Admin Applies a Discount to an Invoice

3.2.3 Sequence Diagrams

D. Successful User Login (Admin/Staff):

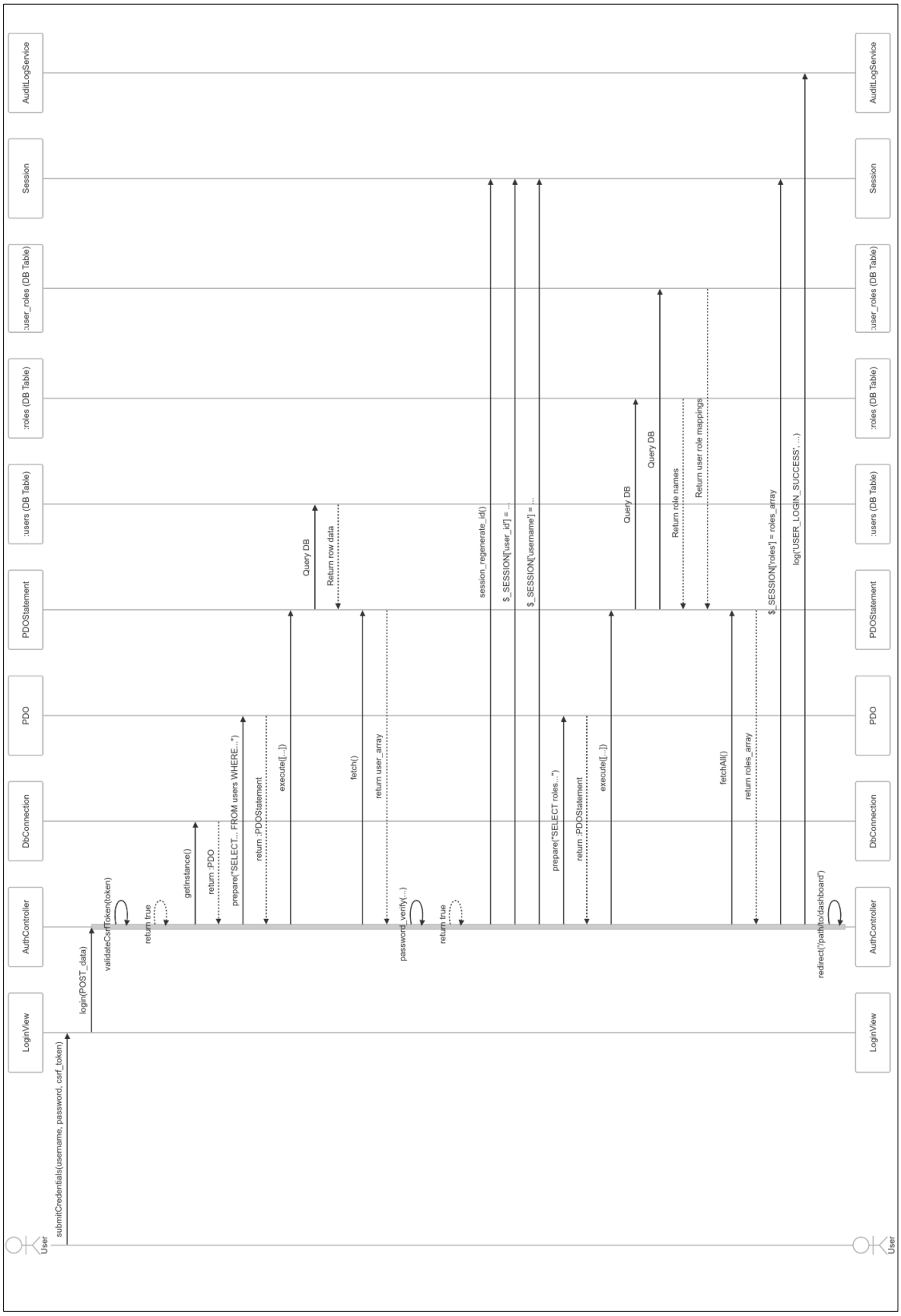


Figure 5: Sequence Diagrams - Successful User Login (Admin/Staff):

E. Admin Verifies Payment Proof Successfully:

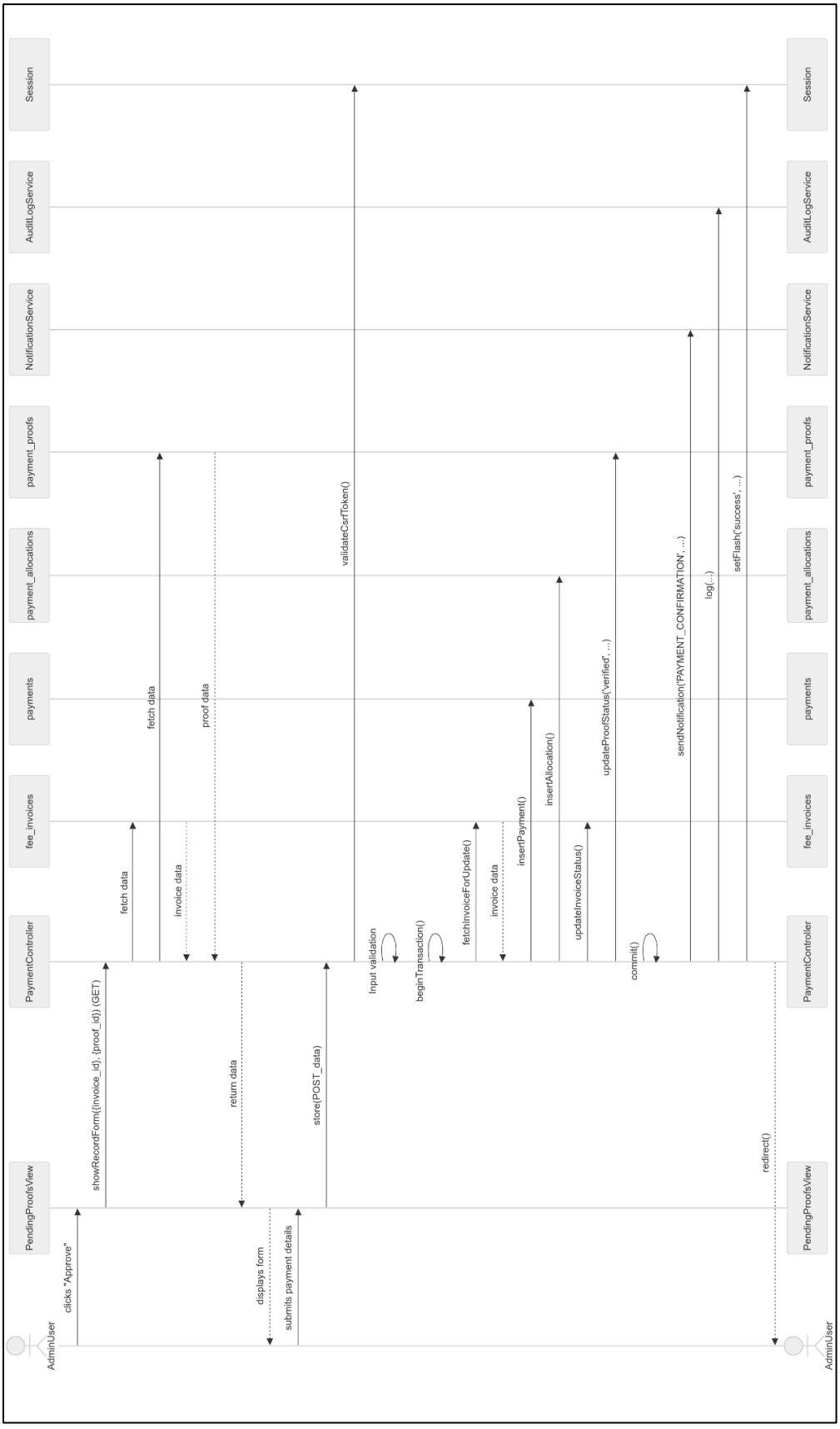


Figure 6: Sequence Diagrams - Admin Verifies Payment Proof Successfully

F. Parent Uploads Payment Proof:

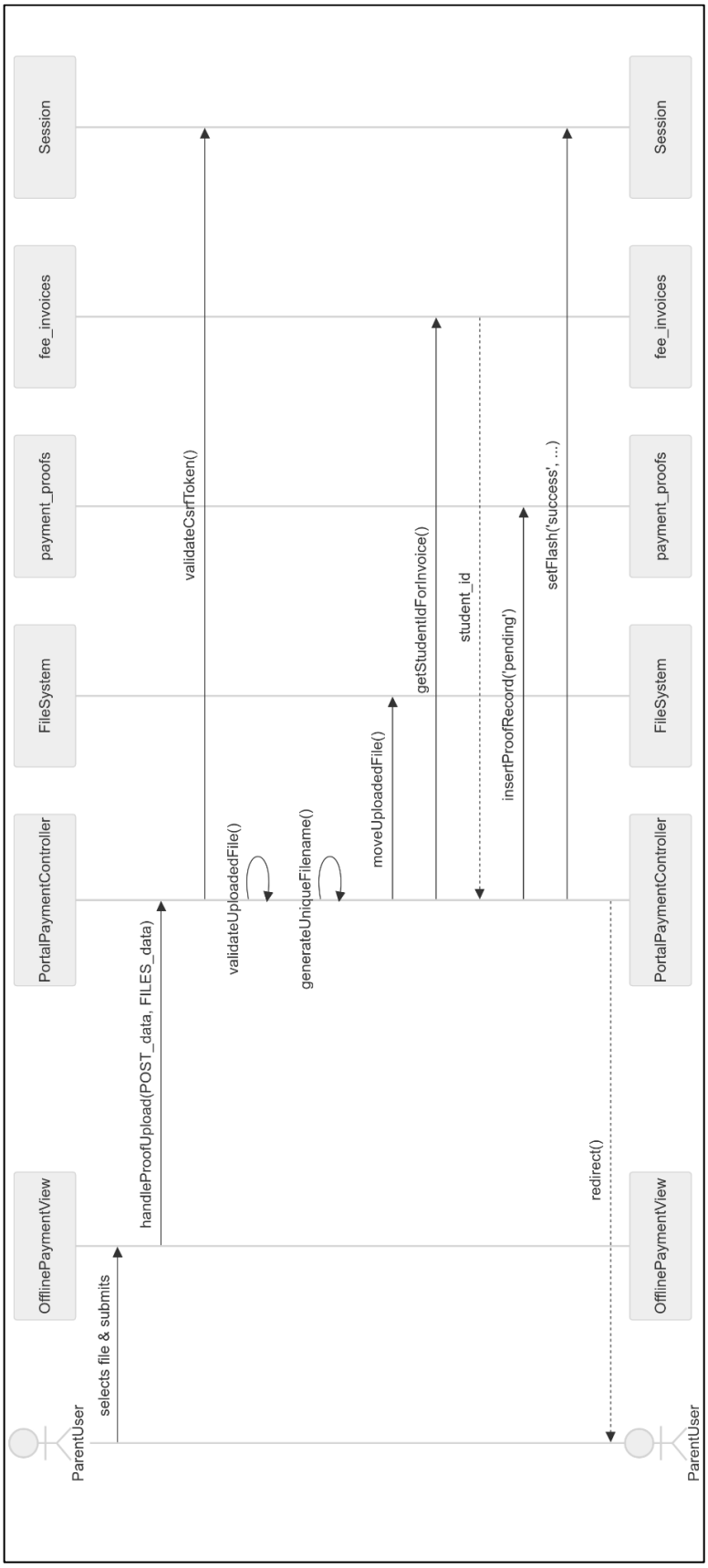


Figure 7: Sequence Diagrams - Parent Uploads Payment Proof

3.2.4 Class Diagram

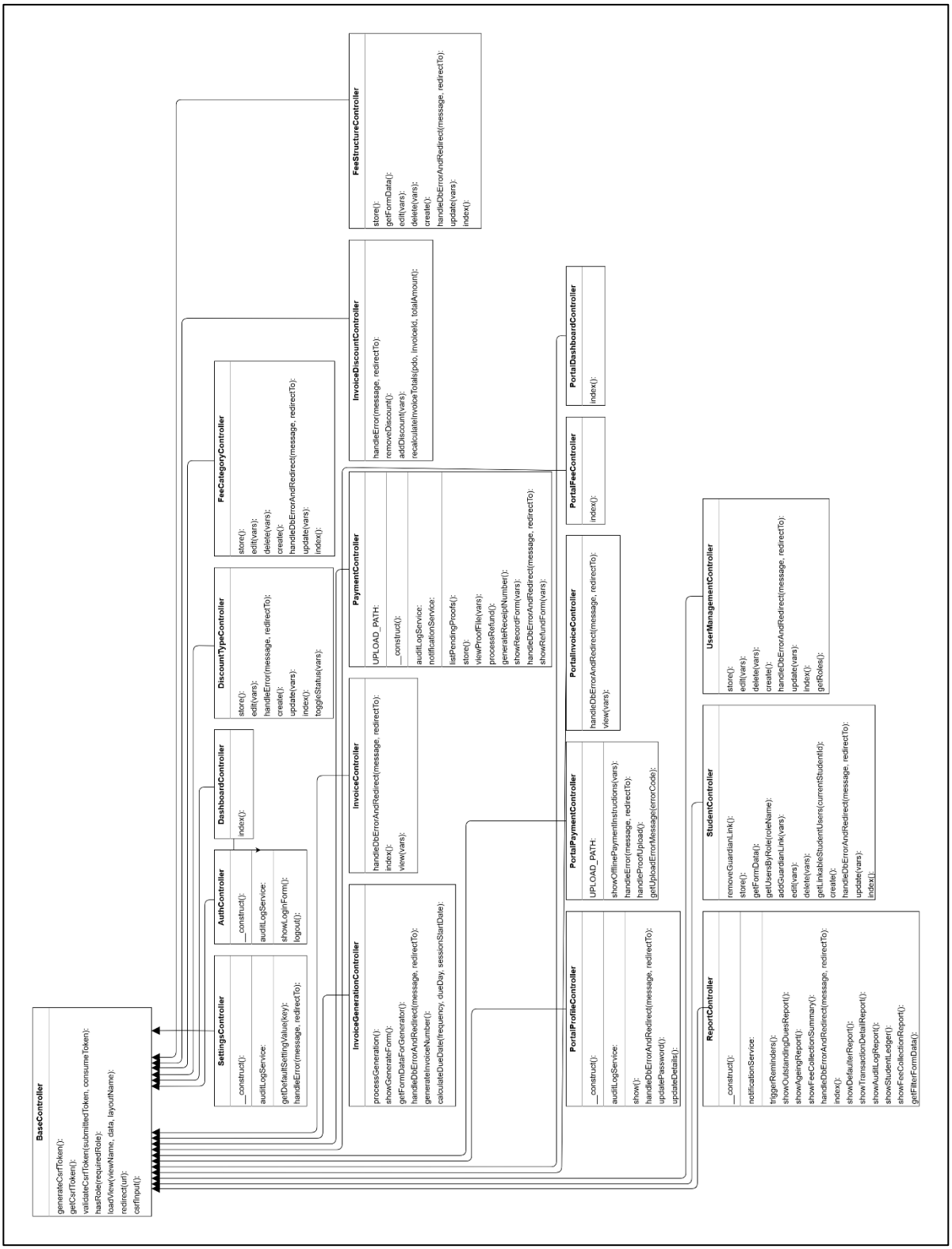


Figure 8: Class Diagram

3.2.5 ER/EER Diagram (Database Schema)

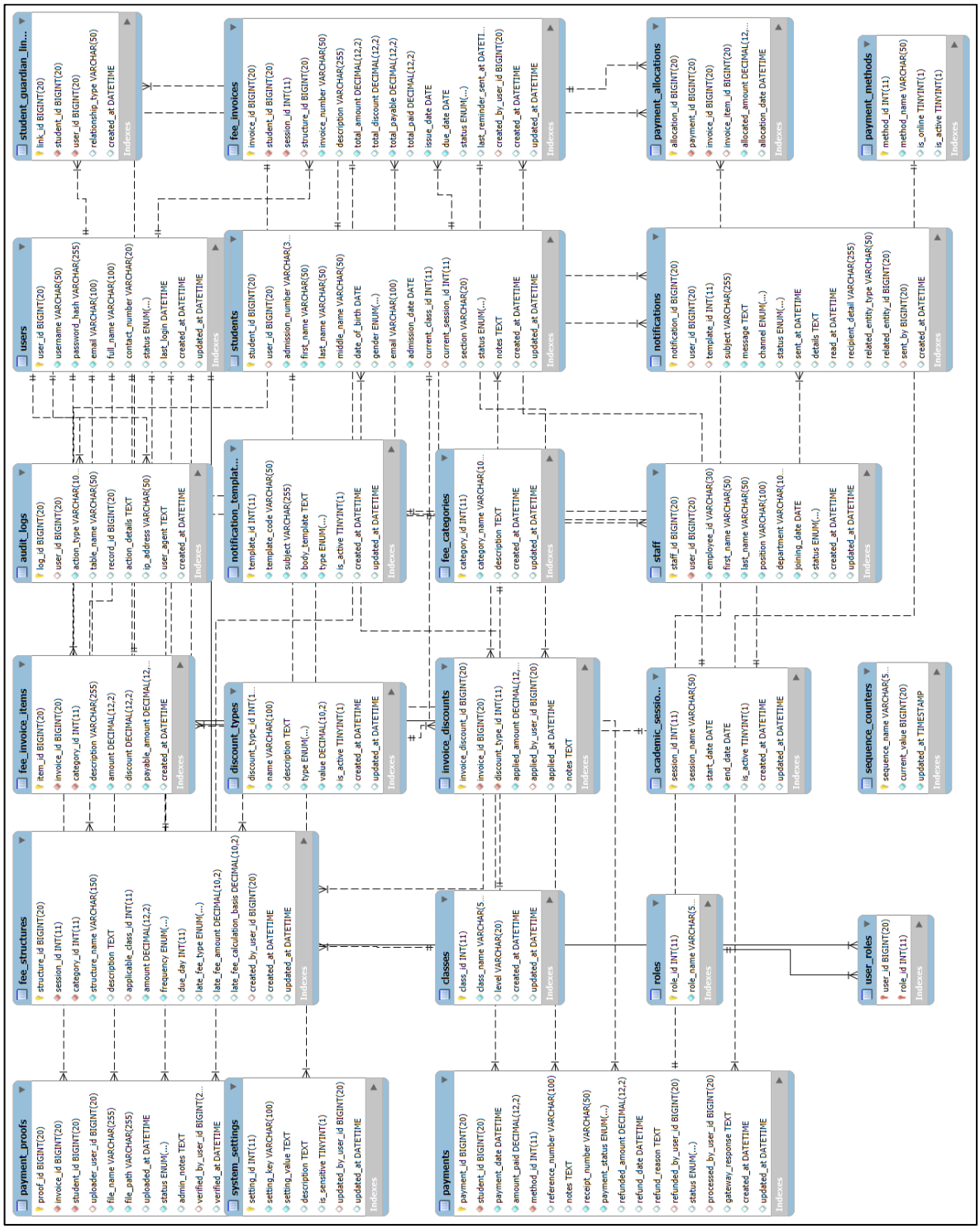


Figure 9: EER Diagram

- **Entities (Tables):**
  - `users` (PK: user_id; Attributes: username, password_hash, email, full_name, contact_number, status, created_at, updated_at, last_login)

- `roles` (PK: role_id; Attributes: role_name)
- `user_roles` (Composite PK: user_id, role_id; FKs: user_id -> users, role_id -> roles)
- `students` (PK: student_id; Attributes: user_id FK (for student login, nullable), admission_number UNIQUE, first_name, last_name, middle_name, date_of_birth, gender, email (student's direct), admission_date, current_class_id FK, current_session_id FK, section, status, notes)
- `academic_sessions` (PK: session_id; Attributes: session_name UNIQUE, start_date, end_date, is_active)
- `classes` (PK: class_id; Attributes: class_name, level)
- `student_guardian_links` (PK: link_id; Attributes: student_id FK, user_id FK (guardian), relationship_type; UNIQUE(student_id, user_id))
- `fee_categories` (PK: category_id; Attributes: category_name UNIQUE, description, is_active)
- `fee_structures` (PK: structure_id; Attributes: session_id FK, category_id FK, structure_name, description, applicable_class_id FK (nullable), amount, frequency, due_day, late_fee_type, late_fee_amount, late_fee_calculation_basis, created_by_user_id FK)
- `discount_types` (PK: discount_type_id; Attributes: name UNIQUE, description, type ENUM, value, is_active)
- `fee_invoices` (PK: invoice_id; Attributes: student_id FK, session_id FK, structure_id FK (nullable), invoice_number UNIQUE, description, total_amount, total_discount, total_payable, total_paid, status, issue_date, due_date, last_reminder_sent_at, created_by_user_id FK)
- `fee_invoice_items` (PK: item_id; Attributes: invoice_id FK, category_id FK, description, amount, payable_amount)
- `invoice_discounts` (PK: invoice_discount_id; Attributes: invoice_id FK, discount_type_id FK, applied_amount, applied_by_user_id FK, notes; UNIQUE(invoice_id, discount_type_id))
- `payment_methods` (PK: method_id; Attributes: method_name UNIQUE, is_online, is_active)
- `payments` (PK: payment_id; Attributes: student_id FK, amount_paid, payment_date, method_id FK, reference_number, notes, receipt_number

UNIQUE, payment_status, processed_by_user_id FK, refunded_amount, refund_date, refund_reason, refunded_by_user_id FK)

- `payment_allocations` (PK: allocation_id; Attributes: payment_id FK, invoice_id FK, allocated_amount; UNIQUE(payment_id, invoice_id, invoice_item_id - if item-level allocation was fully implemented))
- `payment_proofs` (PK: proof_id; Attributes: invoice_id FK, student_id FK, uploader_user_id FK, file_name, file_path, status, admin_notes, verified_by_user_id FK, verified_at, payment_id FK (nullable))
- `notification_templates` (PK: template_id; Attributes: template_code UNIQUE, type, subject, body_template, is_active)
- `notifications` (PK: notification_id; Attributes: user_id FK (nullable), template_id FK (nullable), subject, message, channel, status, recipient_detail, related_entity_type, related_entity_id, sent_by FK (nullable), details, sent_at)
- `system_settings` (PK: setting_id; Attributes: setting_key UNIQUE, setting_value, updated_by_user_id FK)
- `audit_logs` (PK: log_id; Attributes: user_id FK (nullable), action_type, table_name, record_id, action_details (JSON), ip_address, user_agent)

- **Relationships & Cardinalities:**
  - `users` 1--N `user_roles` M--1 `roles`
  - `users` 1--0,1 `students` (for student login)
  - `users` (Parents) M--N `students` (via `student_guardian_links`)
  - `academic_sessions` 1--N `fee_structures`
  - `classes` 1--0,N `fee_structures` (via `applicable_class_id`)
  - `fee_categories` 1--N `fee_structures`
  - `fee_categories` 1--N `fee_invoice_items`
  - `students` 1--N `fee_invoices`
  - `academic_sessions` 1--N `fee_invoices`
  - `fee_structures` 1--0,N `fee_invoices` (nullable if invoice is custom)
  - `fee_invoices` 1--N `fee_invoice_items`

    - `fee_invoices` 1--N `invoice_discounts` M--1 `discount_types`
    - `students` 1--N `payments`
    - `payment_methods` 1--N `payments`
    - `users` (Staff) 1--0,N `payments` (via `processed_by_user_id`)
    - `users` (Staff) 1--0,N `payments` (via `refunded_by_user_id`)
    - `payments` 1--N `payment_allocations` M--1 `fee_invoices` (or 1:1 if simple)
    - `fee_invoices` 1--0,N `payment_proofs`
    - `students` 1--N `payment_proofs`
    - `users` (Uploader) 1--0,N `payment_proofs`
    - `payments` 1--0,1 `payment_proofs` (link after verification)
    - `notification_templates` 1--N `notifications`
    - `users` 1--0,N `notifications` (via `user_id` or `sent_by`)
    - `users` 1--0,N `system_settings` (via `updated_by_user_id`)
    - `users` 1--0,N `audit_logs` (via `user_id`)
- Indicate `ON DELETE` and `ON UPDATE` actions for foreign keys (e.g., `CASCADE`, `SET NULL`, `RESTRICT`).

Chapter 4 - Technologies Adopted

The development of the School Fees Management System (SFMS) was guided by the selection of a technology stack that balances robustness, scalability, ease of development, security considerations, and widespread community support. This chapter details the core technologies, libraries, tools, and architectural concepts employed in the project, along with the rationale for their adoption, referencing the project's Software Requirements Specification (SRS) [1] where applicable.

4.1 Backend Technologies

The server-side logic and data management of the SFMS are built upon the following foundational technologies:

- **PHP (Hypertext Preprocessor)**
    - **Version:** The project was developed targeting a modern PHP version (e.g., PHP 8.0 or higher). This choice was made to leverage significant improvements in performance, stricter type systems (scalar type hints, return type declarations), enhanced error handling mechanisms, and new language features (such as constructor property promotion, match expressions, attributes) that contribute to writing cleaner, more maintainable, and more efficient code.
    - **Rationale:** PHP remains a dominant force in web development due to its extensive ecosystem, large and active global community, and a vast array of available libraries and frameworks (managed via Composer). Its seamless integration with common web servers like Apache and databases like MySQL, coupled with its suitability for rapid application development and the wide availability of skilled developers, made it a pragmatic and effective choice for a project of this nature and scale. Both procedural (for scripting and simple tasks) and object-oriented programming (OOP) paradigms were utilized, with OOP being central to the structure of controllers, services, and core components.
- **MySQL Relational Database Management System**
    - **Version:** The system was designed and tested to work with MySQL version 8.0 or a compatible equivalent (such as MariaDB 10.4+), as specified in SRS 2.4 and 3.3 [1].
    - **Rationale:** MySQL is a widely adopted, robust, and open-source RDBMS renowned for its performance, reliability, and ease of use, especially in conjunction with PHP applications. For the SFMS, the **InnoDB storage engine** was implicitly chosen and is critical due to its support for:
        - **ACID-compliant transactions:** Essential for financial operations like payment recording, discount application, and refund processing to

ensure data atomicity and consistency (i.e., all parts of a multi-step database operation complete successfully, or none do, preventing data corruption).

    - **Foreign key constraints:** Vital for maintaining relational integrity between the numerous interconnected tables in the SFMS schema (e.g., `students` to `fee_invoices`, `fee_invoices` to `payments`). `ON DELETE SET NULL` and `ON DELETE CASCADE` rules were used where appropriate.

    - **Row-level locking:** Beneficial for concurrent access scenarios. Its strong compatibility with PHP via PDO, widespread availability on hosting platforms, and comprehensive documentation were also key factors in its selection.

- **Apache Web Server (via XAMPP)**

    - **Rationale:** XAMPP was utilized primarily for the local development environment, providing an accessible and easy-to-install package containing Apache, MySQL, and PHP. Apache is a mature, feature-rich, highly configurable, and extensively deployed web server. For this project, Apache's `mod_rewrite` module was critically leveraged via an `.htaccess` file located in the `public` directory. This enabled the implementation of a **front controller pattern**, ensuring that all web requests are routed through a single entry point (`public/index.php`). This approach facilitates clean URLs, centralized request handling, and a more organized application structure.

4.2 Frontend Technologies

The user interface for both the administrative backend and the student/parent portal was constructed using standard web technologies, significantly enhanced by a modern CSS framework for styling and responsiveness.

- **HTML5 (HyperText Markup Language 5)**

    - **Rationale:** As the structural foundation for all web pages, HTML5 was used to provide semantic markup for content, ensuring accessibility and a logical document structure. Elements like `<nav>`, `<main>`, `` `,,,` ``, and form-specific input types were utilized.

- **CSS3 (Cascading Style Sheets 3)**

    - **Rationale:** CSS3 was employed for custom styling and presentation aspects not covered by the chosen framework or requiring specific overrides. While Bootstrap 5 provided the primary styling, custom CSS was integrated within `<style>` tags in the layout files (`_header_admin.php`, `_header_portal.php`)

for minor global adjustments (e.g., body background, flash message appearance, Select2 height fixes) and on specific view components where needed.

- **Bootstrap 5**
    - **Version:** v5.3.3 (integrated via Content Delivery Network - CDN).
    - **Rationale:** Bootstrap is a leading open-source CSS framework that dramatically accelerates frontend development. Its comprehensive suite of pre-styled components (forms, tables, navigation bars, cards, alerts, modals, dropdowns, badges, pagination), a powerful responsive grid system (`container`, `row`, `col-*`), and extensive utility classes were instrumental in building both the Admin and Student/Parent Portal interfaces. This ensured a consistent, modern, professional look and feel, and, critically, made the application responsive and usable across various screen sizes (desktop, tablet, mobile) with significantly reduced custom CSS development effort.
- **Bootstrap Icons**
    - **Version:** v1.11.3 (integrated via CDN).
    - **Rationale:** This official icon library for Bootstrap offers a lightweight, visually cohesive, and extensive set of SVG-based icons. They were used pervasively throughout the application (e.g., on buttons, navigation links, section headers, table actions, card titles) to enhance user interface clarity, improve scannability, and provide intuitive visual cues for actions and information, contributing to a better user experience.
- **JavaScript (Vanilla & jQuery)**
    - **jQuery (via CDN):** Version 3.7.1 was included. While modern web development often emphasizes vanilla JavaScript, jQuery was primarily incorporated as a dependency for the Select2 library. Its utility for DOM manipulation and event handling was available if needed for simpler tasks, though its use was not extensive beyond Select2.
    - **Vanilla JavaScript:** Employed for simple client-side interactions, such as `onclick` confirmation dialogs (`confirm('Are you sure?');`) on sensitive action buttons (e.g., delete, process refund, toggle status) and the `onchange="this.form.submit()"` attribute for auto-submitting forms upon selection change (e.g., the student selector dropdown in portal views).
- **Select2**
    - **Version:** v4.1.0-rc.0 (integrated via CDN, along with the `select2-bootstrap-5-theme`).

- **Rationale:** Select2 is a jQuery-based replacement for standard HTML `<select>` boxes. It was specifically implemented for dropdowns that could contain a large number of options, such as the student selection list in the Student Ledger report and the parent/guardian selection in the Student Management forms. This library provides a significantly improved user experience by adding search and filtering capabilities to these dropdowns, making it much easier for users to find and select the desired option from extensive lists. The Bootstrap 5 theme ensured visual consistency.

4.3 Development Tools, Libraries, and Services

Several tools, libraries, and conceptual services were instrumental in the development process, security, and overall functionality of the SFMS:

- **Composer**
  - **Rationale:** Composer is the de-facto standard dependency manager for PHP. It was utilized to install, manage versions of, and autoload external PHP libraries critical to the project, such as PHPMailer and `vlucas/phpdotenv`. This ensures that the project uses consistent library versions across different development and deployment environments and simplifies the inclusion of third-party code.
- **PHPMailer**
  - **Version:** Latest stable version installed via Composer.
  - **Rationale:** PHPMailer is a mature, full-featured, and widely trusted PHP library for sending emails. It was chosen for its robust support for various email sending methods, particularly **SMTP authentication**, which is essential for reliable email delivery through external mail servers (like MailerSend, as configured). It supports HTML and plain text emails, attachments (though not used in the current scope for notifications), and provides comprehensive error handling. This library was integrated into the `NotificationService` to dispatch payment confirmations and fee reminders.
- **vlucas/phpdotenv**
  - **Version:** Latest stable version installed via Composer.
  - **Rationale:** This library facilitates the loading of environment variables from a .env file into the application's runtime environment (making them accessible via `getenv(), _ENV`, and `_SERVER`). This is a critical security best practice, allowing sensitive configurations such as database credentials (`DB_USERNAME, DB_PASSWORD`), email server credentials (`MAIL_USERNAME, MAIL_PASSWORD`), and other API keys to be stored securely outside of the version-controlled codebase. This also allows for different configurations across various

deployment environments (development, staging, production) without code changes.

- **nikic/fast-route**
    - **Version:** Latest stable version installed via Composer.
    - **Rationale:** FastRoute is a lightweight, efficient, and popular PHP routing library. It was selected to handle the mapping of URL requests to specific controller actions. This provides a clean, maintainable, and performant way to define the application's API and web page routes. Its support for placeholders with regular expressions (e.g., `{id:\d+}` for numeric IDs) and HTTP method-based routing (GET, POST) was extensively utilized in public/index.php.
- **XAMPP**
    - **Rationale:** XAMPP served as the local development stack, providing an integrated environment with Apache (web server), MySQL (database), PHP, and phpMyAdmin (database administration tool). Its ease of installation and configuration facilitated a rapid setup for the development process.
- **Visual Studio Code (VS Code)**
    - **Rationale:** VS Code was used as the primary source code editor. Its popularity stems from its lightweight nature, extensive feature set (IntelliSense for PHP, debugging capabilities, Git integration, terminal), and a vast ecosystem of extensions that enhance developer productivity.
- **StarUML**
    - **Rationale:** As specified in the project requirements, StarUML was the designated tool for the creation of UML (Unified Modeling Language) design diagrams, including Use Case, Activity, Sequence, and Class diagrams, as well as the ER/EER (Entity-Relationship / Enhanced Entity-Relationship) Diagram for modeling the system's database structure and behavior.
- **Git & GitHub (Assumed)**
    - **Rationale:** Git is the industry-standard distributed version control system, essential for tracking changes to the codebase, managing different branches for features and fixes, and collaborating within a development team. GitHub (or a similar platform like GitLab/Bitbucket) is assumed to be used for hosting the remote Git repository.

4.4 Architectural Concepts and Patterns

The SFMS was architected with several key concepts and design patterns to promote a structured, maintainable, and scalable application:

- **Modular Design:** The application's backend code was organized into distinct modules (e.g., `Auth`, `UserManagement`, `StudentManagement`, `FeeManagement`, `Reporting`, `Portal`, `Admin`) primarily located within the `app/Modules/` directory. Each module encapsulates related functionalities, including its own Controllers and Views, promoting separation of concerns and making the codebase easier to navigate, develop, and maintain.

- **MVC-like Structure (Model-View-Controller variant):** While not implementing a formal, full-fledged MVC framework, the project adheres to a similar separation of responsibilities:

    - **Controllers (`app/Modules/*/Controllers/`):** Act as the intermediary between user requests and the application's response. They handle incoming HTTP requests, process user input, interact with the database (often via Service or Helper classes), make decisions based on business logic, and select appropriate Views for rendering the output.

    - **Views (`app/Modules/*/Views/` and `app/Views/layouts/`):** Responsible for the presentation layer. They primarily contain HTML markup mixed with PHP for displaying dynamic data passed from the controllers. Bootstrap 5 is heavily utilized for styling and layout within the views.

    - **Models (Implicit/Integrated):** Database interaction logic is primarily contained within the controller methods themselves for simpler queries, or encapsulated within dedicated Service classes (e.g., `NotificationService`, `AuditLogService`) or Helper classes (e.g., `SettingsHelper`) for more complex or reusable database operations. PHP Data Objects (PDO) are used as the database abstraction layer for interacting with MySQL.

- **Front Controller Pattern:** All web requests are directed by the web server (via `.htaccess` rules for Apache) to a single entry point, `public/index.php`. This script initializes the application (e.g., loads environment variables, starts sessions, sets up autoloading via Composer), invokes the router (`nikic/fast-route`), and dispatches the request to the appropriate controller action. This centralizes request handling and simplifies URL management.

- **Service Layer (Partial Implementation):** To avoid overly "fat" controllers and to promote reusability of business logic, dedicated service classes were created within `app/Core/Services/`. Examples include:

    - `AuditLogService`: Centralizes the logic for recording user actions and system events.
    - `NotificationService`: Manages the fetching of notification templates, preparation of messages, sending of emails via PHPMailer, and logging of notification attempts.
    - `SequenceGenerator`: Provides a reliable mechanism for generating unique, sequential numbers (e.g., for student admission numbers) using a database counter table with appropriate locking to prevent race conditions.

- **Helper Classes:** Utility classes like `SettingsHelper` (in `app/Core/Helpers/`) provide static methods for common, reusable tasks, such as interacting with the `system_settings` database table. This makes accessing global settings consistent and easy from various parts of the application.

- **Layout System (PHP-based):** A simple but effective server-side layout system was implemented using PHP `require` statements. Common header (`_header_admin.php`, `_header_portal.php`) and footer (`_footer_admin.php`, `_footer_portal.php`) files are stored in `app/Views/layouts/`. A main layout file (e.g., `layout_admin.php` or `layout_portal.php`) includes the respective header, then a `$content` variable (which is populated by the `BaseController::loadView()` method capturing the output of a specific view file), and finally the footer. This ensures a consistent look and feel for the Admin and Portal sections and centralizes common HTML structures, CSS links, and JavaScript includes.

- **Singleton Pattern (for `DbConnection`):** The `DbConnection::getInstance()` method implements the singleton pattern. This ensures that only one database connection object is created and reused throughout the application's request lifecycle, which is an efficient way to manage database resources and prevent multiple unnecessary connections.

- **Configuration Management:**
    - **`.env` files (via `vlucas/phpdotenv`):** Used for storing environment-specific and highly sensitive configurations like database credentials and mail server passwords. This file is not committed to version control.
    - **PHP Config Files (`app/config/`):** Files like `database.php` and `mail.php` read their values from the environment variables (loaded from `.env`) and provide these configurations to the application in a structured array format.
    - **Database Settings (`system_settings` table & `SettingsHelper`):** Used for non-sensitive, application-wide settings (e.g., school name, bank details, reminder timings) that administrators can manage through the UI, providing flexibility without code changes.

The strategic selection and integration of these technologies, libraries, and architectural patterns aimed to create a School Fees Management System that is functional, maintainable, reasonably secure, and provides a good user experience for its intended audience, fulfilling the core objectives of the project.

Chapter 5 - Implementation

This chapter details the practical realization of the School Fees Management System (SFMS). It focuses on the implemented system interfaces for various user roles and highlights key coding segments and architectural patterns that underpin the system's functionality, security, and maintainability. The implementation closely followed the design specifications outlined in Chapter 3 and utilized the technologies described in Chapter 4.

5.1 System Interfaces

The SFMS provides distinct, role-tailored interfaces for Administrators, Staff, Students, and Parents. All interfaces were developed using HTML5, styled with Bootstrap 5 for a responsive and modern user experience, and enhanced with Bootstrap Icons and minimal JavaScript for interactivity. A consistent layout approach, managed by a `BaseController` and distinct layout files (`layout_admin.php`, `layout_portal.php`), was adopted for both the administrative backend and the student/parent portal.

5.1.1 Login Page (`/login`)

- **Purpose:** Serves as the secure, unified entry point for all system users (Administrators, Staff, Students, and Parents).
- **Design:** The page features a visually appealing, centered login form, typically set against a professional background image (configurable or static). It is designed to be responsive and user-friendly on various devices.
- **Key UI Elements & Features:**
    - **Form Fields:** Clearly labeled input fields for "Username" and "Password," utilizing Bootstrap's floating labels and input group prepends for icons (`bi-person`, `bi-lock`).
    - **CSRF Protection:** A hidden CSRF token field is embedded in the form, validated upon submission to prevent cross-site request forgery attacks.
    - **Error Display:** Login error messages (e.g., "Invalid username or password," "Account inactive," "Invalid security token") are prominently displayed above the form using Bootstrap's `alert-danger` class if a login attempt fails.
    - **Submit Button:** A clearly identifiable "Login" button, styled using Bootstrap (`btn-primary`, `btn-lg`, `d-grid` for full width) and including an icon (`bi-box-arrow-in-right`).
    - **(Future Placeholder):** A "Forgot Password?" link can be easily added later.

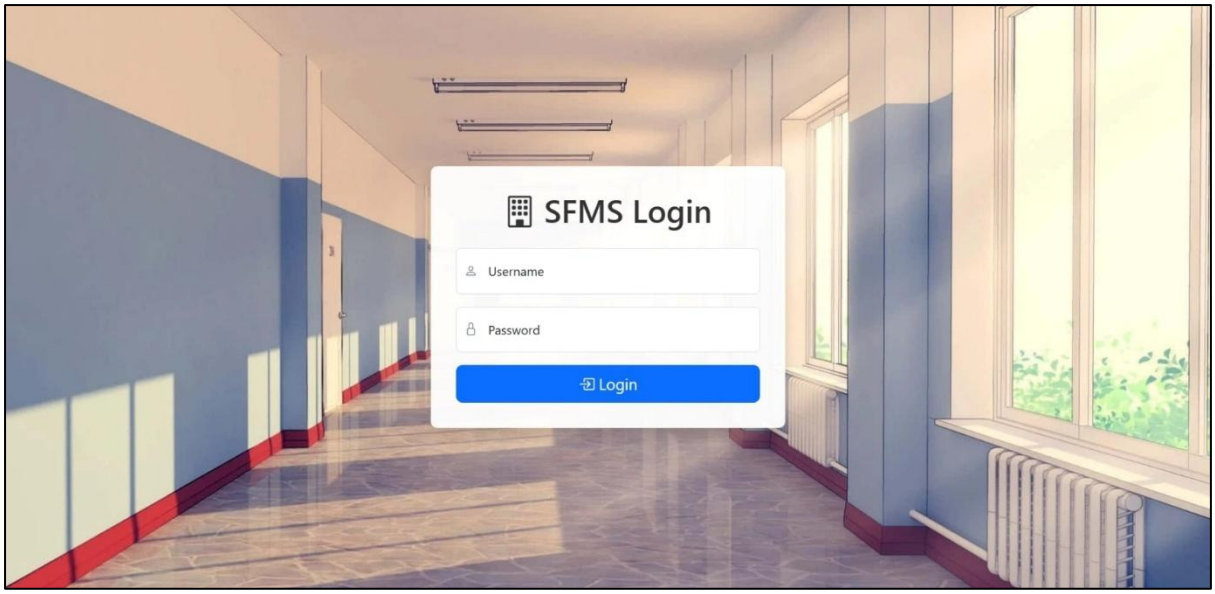


Figure 10: Login Page

5.1.2 Administrative Area (`/admin/...`)

The administrative area is designed for users with 'Admin' and 'Staff' roles, providing comprehensive tools for managing the SFMS. It utilizes a consistent layout (`app/Views/layouts/layout_admin.php`) which includes a shared header, navigation, and footer.

- **Common Admin Layout Features (`_header_admin.php`, `_footer_admin.php`):**
    - **Navigation Bar:** A dark-themed, responsive Bootstrap navbar (`navbar-dark bg-dark`) provides top-level navigation. Links include: Dashboard, Students, Users, a "Fees Mgmt" dropdown (for Categories, Structures, Discount Types, Invoices, Proof Verification), Reports, and Settings (visible to Admins only). Active menu items are visually highlighted. The navbar also displays a "Welcome, [Username]!" message and a "Logout" link with an icon.
    - **Flash Messages:** Global success, error, and informational messages (e.g., after CRUD operations, settings updates, or reminder triggers) are displayed prominently below the navbar using Bootstrap `alert` classes. These messages are read from the session and cleared after display.
    - **Main Content Area:** A Bootstrap `container` where the specific content for each admin page is dynamically loaded by the `BaseController::loadView()` method.
    - **Footer:** A simple footer displaying copyright information.
    - **Assets:** The header includes CDN links for Bootstrap CSS, Bootstrap Icons CSS, and Select2 CSS (with Bootstrap 5 theme). The footer includes CDN links

for Bootstrap JS Bundle, jQuery, and Select2 JS, along with a global JavaScript block to initialize Select2 on elements with the `.select2-enable` class.

- **Admin Dashboard (`/admin/dashboard` - `DashboardController::index`, `admin_dashboard.php`)**
    - **Purpose:** Provides a high-level overview of key system metrics and quick access to common administrative tasks.
    - **UI Elements:**
        - **Statistics Cards:** Key metrics (Active Students, Total Outstanding Dues, Payments Collected Today - Count & Amount, Pending Payment Proofs) are displayed in visually distinct Bootstrap `card` components, each with an appropriate icon. The "Pending Proofs" card is styled as a warning if the count is greater than zero.
        - **Quick Actions:** A section with Bootstrap `btn` styled links for common actions: "Add Student," "Add User," "Generate Invoices," "View Invoices," "Verify Proofs," and "View Reports."

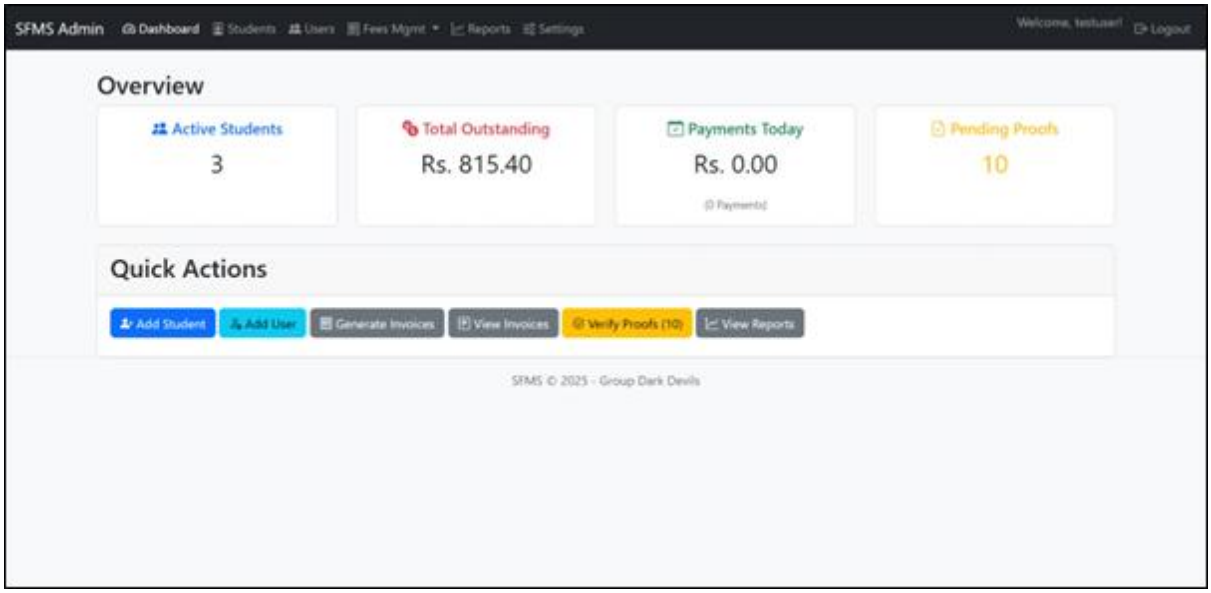


Figure 11: Admin Dashboard

- **User Management (`/admin/users/...` - `UserController`, User views)**
    - **List View (`index.php`):** Displays all system users in a paginated (if implemented, currently conceptual) Bootstrap table (`table-striped`, `table-hover`, `table-bordered`, `table-sm`). Columns include User ID, Username, Email, Full Name, Status (styled with Bootstrap `badge` classes like `bg-success` for 'active' and `bg-secondary` for 'inactive'), and Created At. Actions include

"Edit" (Bootstrap button with icon) and "Delete" (Bootstrap button with icon within a CSRF-protected form, with a JavaScript `confirm()` dialog).

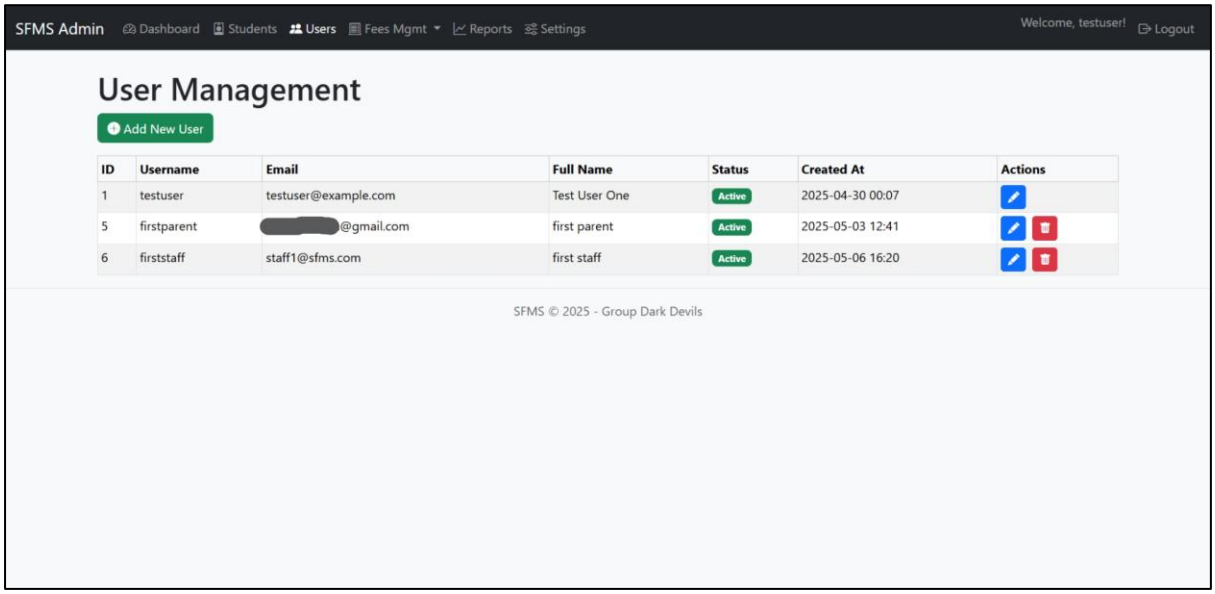


Figure 12: User Management - List View

- **Create/Edit Forms (`create.php`, `edit.php`):** Bootstrap-styled forms using grid layout (`row`, `col-md-6`) for adding or modifying user details (username, email, password/confirm password, full name, contact number, status). Multiple roles are assigned/updated using Bootstrap `form-check-inline` styled checkboxes. Forms include CSRF protection. Server-side validation errors are displayed next to relevant fields using `is-invalid` classes and `invalid-feedback` divs, and valid previously entered data is repopulated.

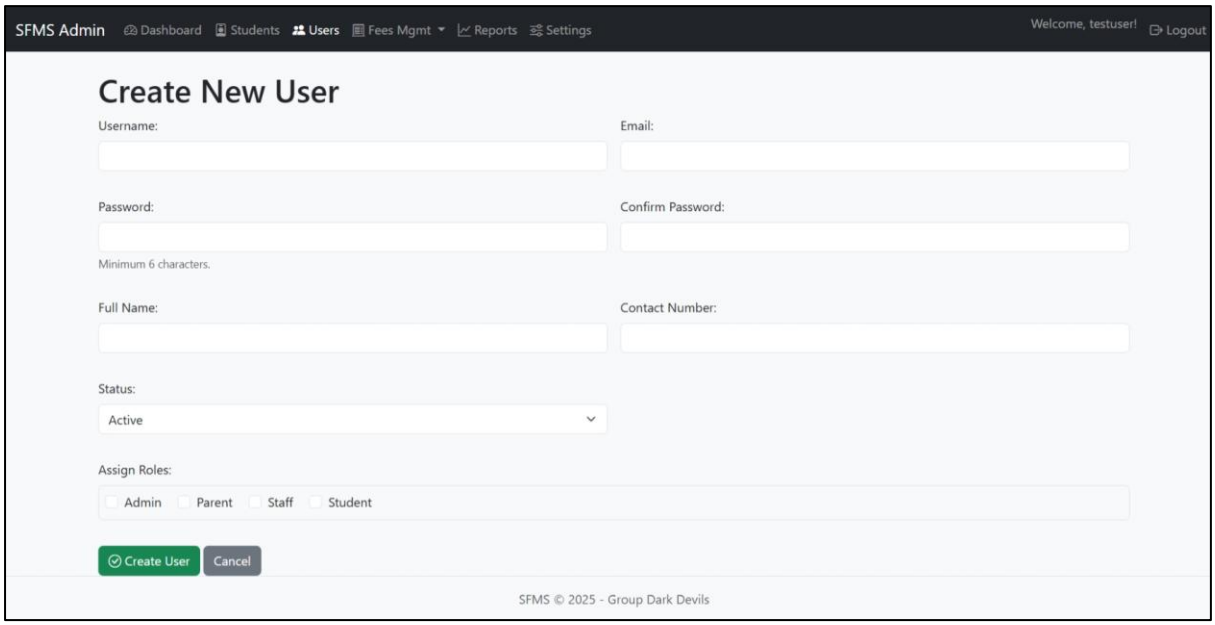


Figure 13: User Management - Create Forms

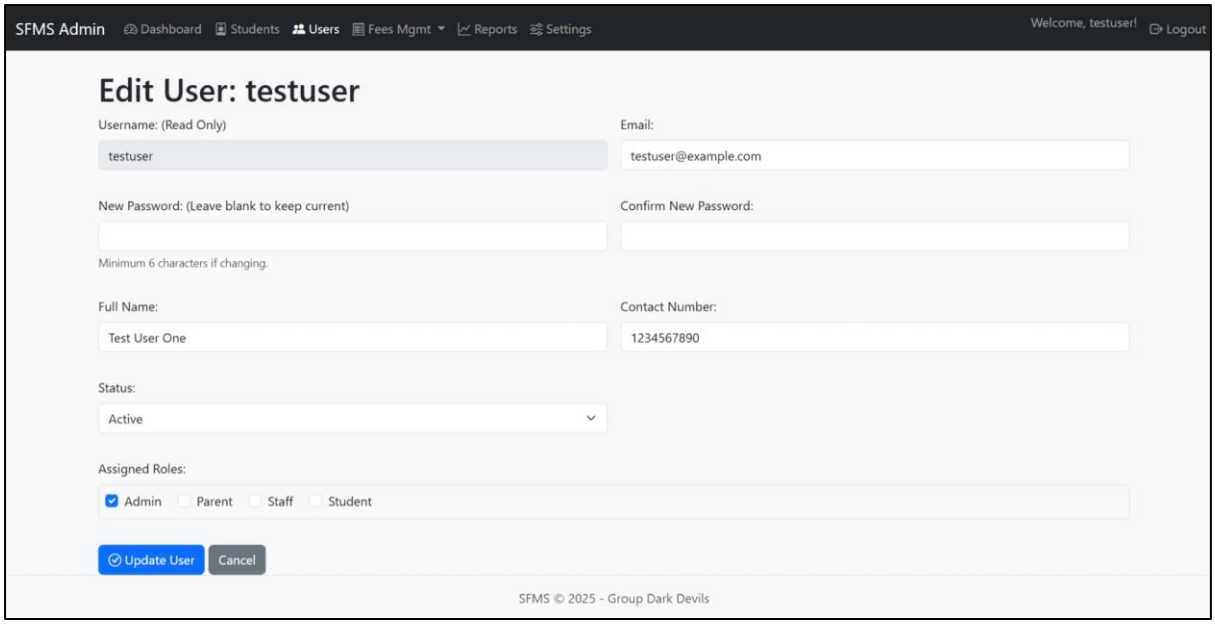


Figure 14: User Management - Edit Forms

- **Student Management (`/admin/students/...` - `StudentController`, Student views)**
    - **List View (`index.php`):** Displays students in a Bootstrap table with ID, Admission No., Name, Email, Class, Section, Session, Status (badges). Actions include "Edit" and "Delete" (CSRF-protected form with confirmation).

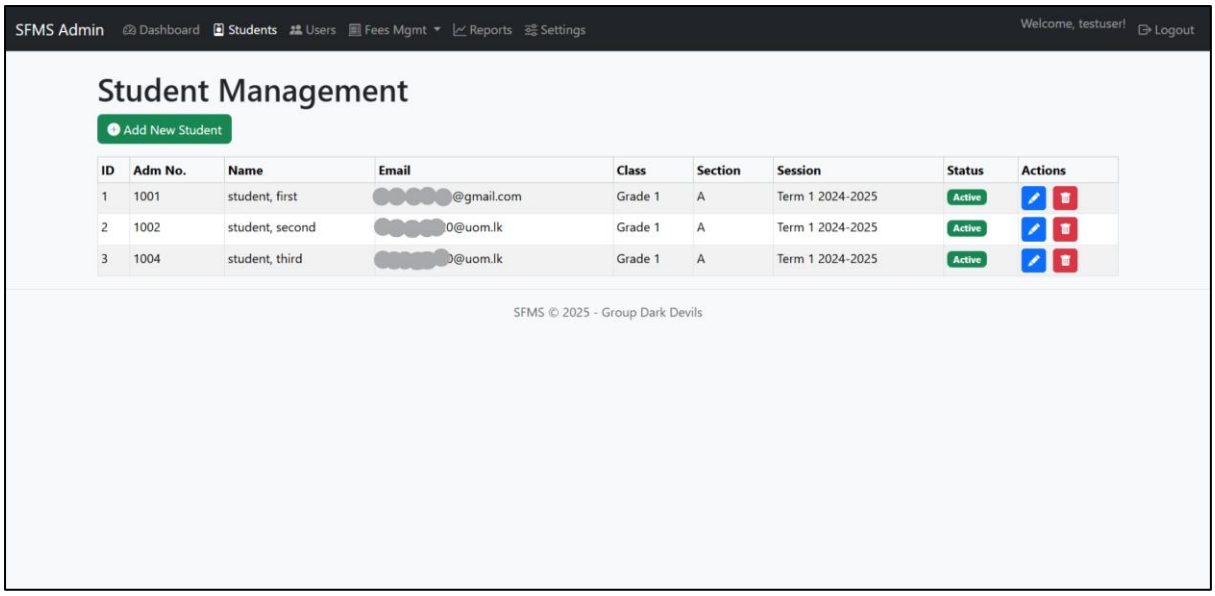


Figure 15: Student Management - List View

    - **Create Form (`create.php`):** Bootstrap form for adding new students. Includes fieldsets for Admission Details (admission number is auto-generated and displayed as read-only, admission date), Personal Details (name, DOB, gender,

email), and Account Links (optional link to an existing student user account via a Select2 searchable dropdown).

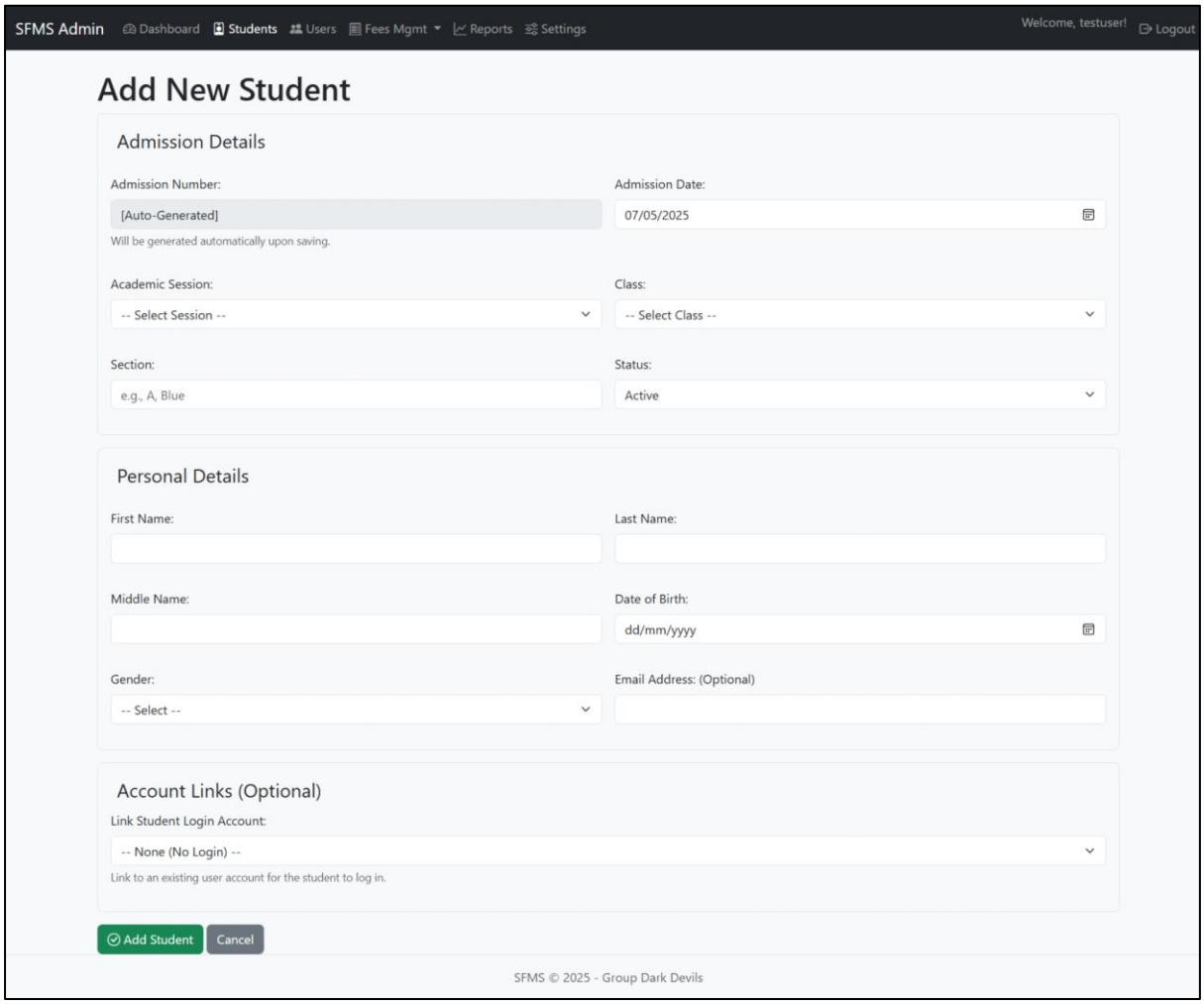


Figure 16: User Management - Create Form

- **Edit Form (`edit.php`):** Comprehensive Bootstrap form to edit student details. Includes a dedicated "Linked Parents / Guardians" section (styled with Bootstrap cards and list groups) displaying currently linked guardians (fetched from `student_guardian_links`) with their relationship type and an option to "Remove" each link (CSRF-protected form). A sub-form allows adding new guardian links by selecting from 'Parent' role users (via Select2 dropdown) and specifying a relationship type. Also allows linking/unlinking the student's own portal login account and optionally updating their password if a user account is linked.

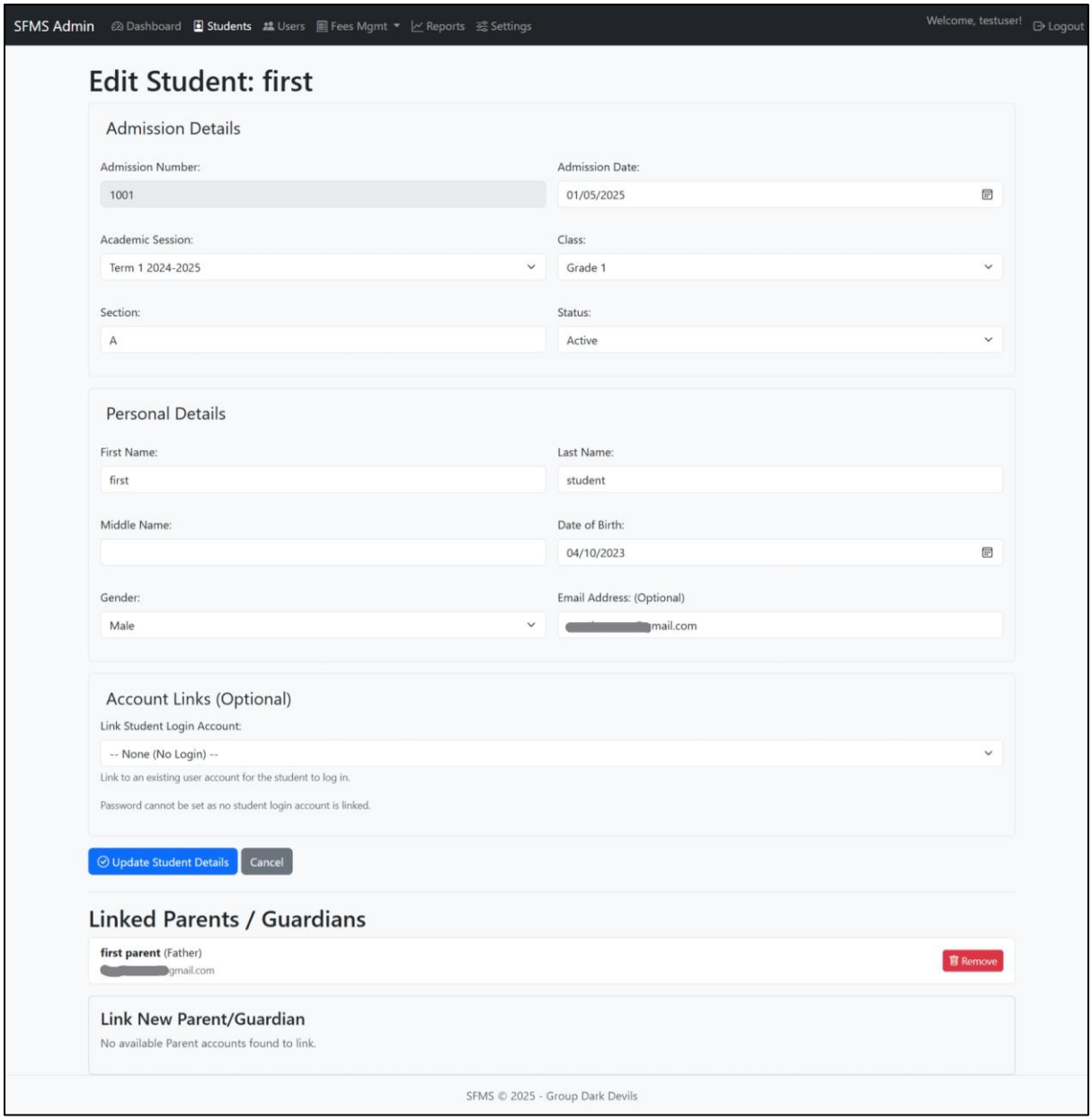


Figure 17: User Management - Edit Form

- **Fee Category Management (`/admin/fees/categories/...` - `FeeCategoryController`, Category views)**
    - **List View (`index.php`):** Bootstrap table listing Fee Categories with ID, Name, Description, Status (Active/Inactive badges), Created At. Actions: "Edit" button, "Toggle Status" button/form (CSRF protected, button text/style changes based on current status).

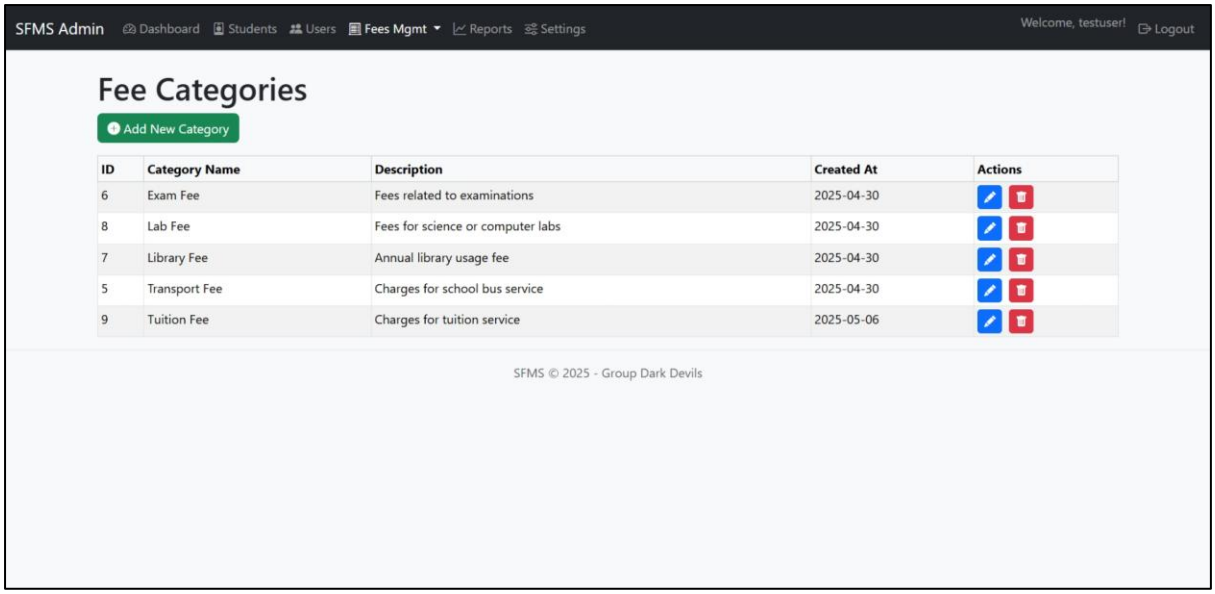


Figure 18: Fee Category Management - List View

- **Create/Edit Forms (`create.php`, `edit.php`):** Bootstrap forms for managing category name, description, and active status (checkbox).

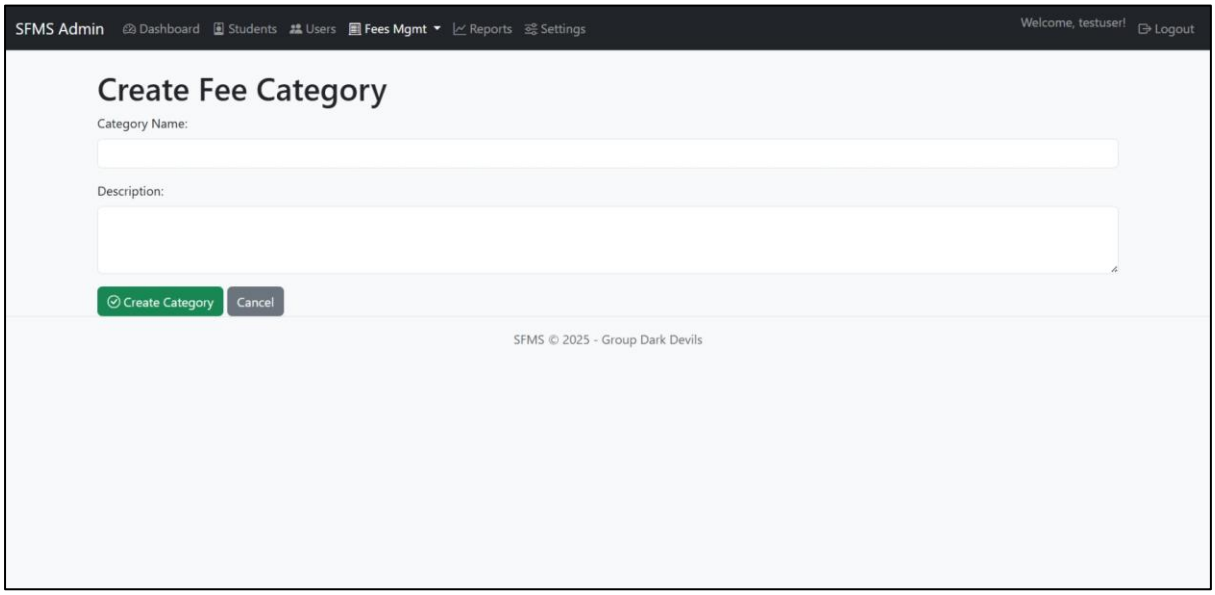


Figure 19 Fee Category Management - Create Form

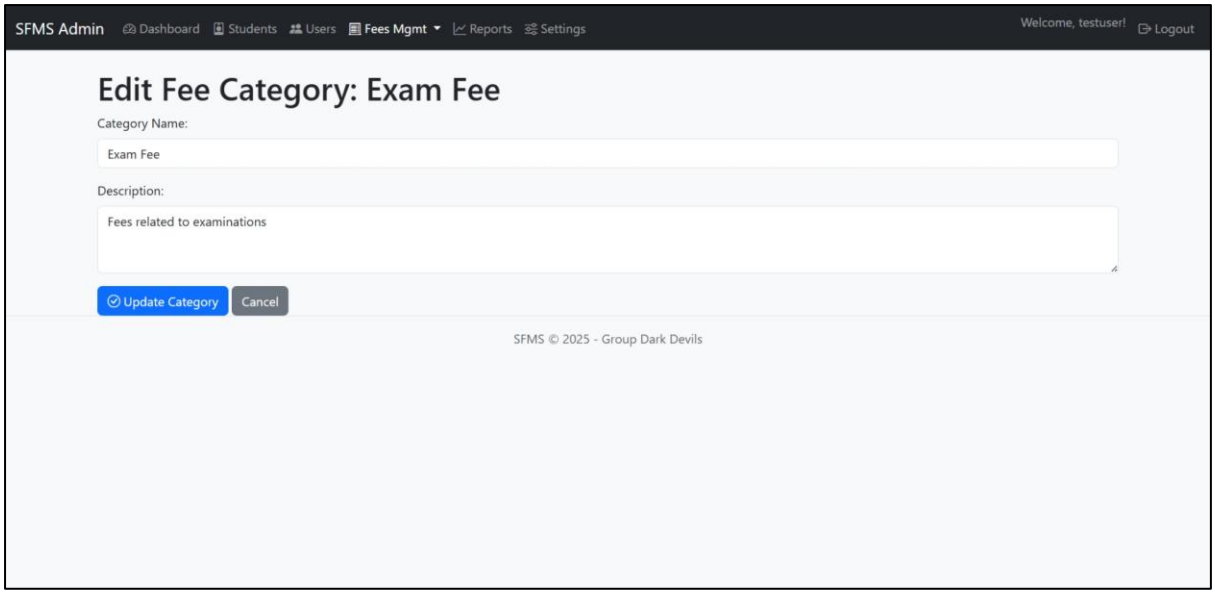


Figure 20: Fee Category Management - Edit Form

- **Discount Type Management (`/admin/fees/discount-types/...` - `DiscountTypeController`, Discount views)**
    - **List View (`index.php`):** Bootstrap table listing Discount Types with ID, Name, Description, Type (Fixed Amount/Percentage), Value (formatted with % if percentage), Status (badges). Actions: "Edit" button, "Toggle Status" button/form (CSRF).

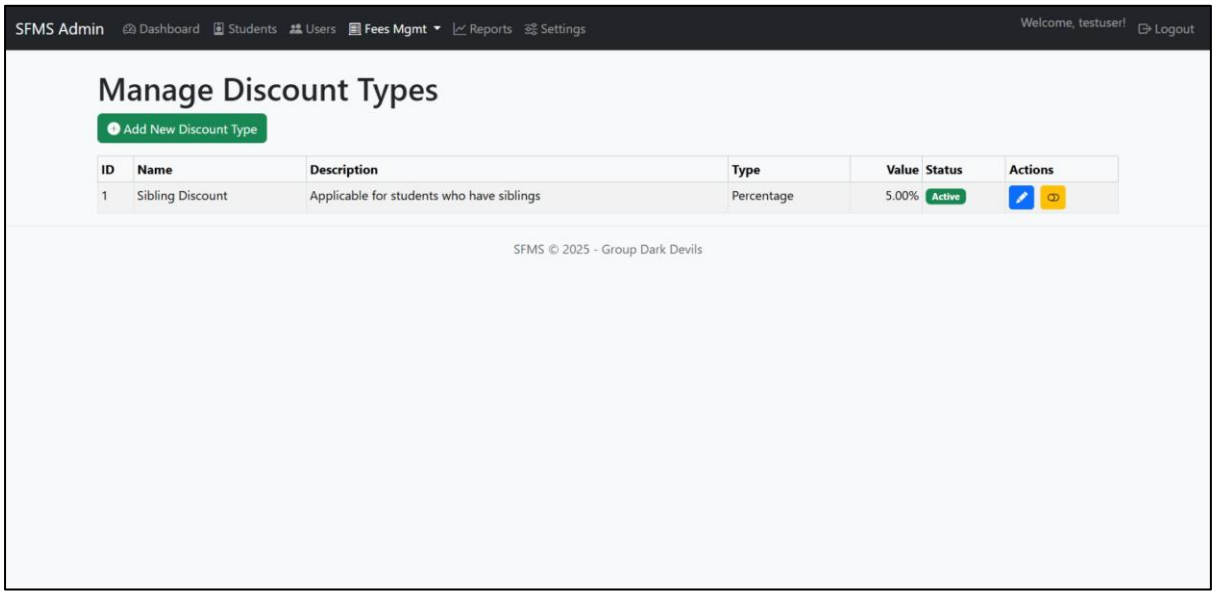


Figure 21: Discount Type management - List View

    - **Create/Edit Forms (`create.php`, `edit.php`):** Bootstrap forms for discount name, type (dropdown), value, description, and active status (checkbox).

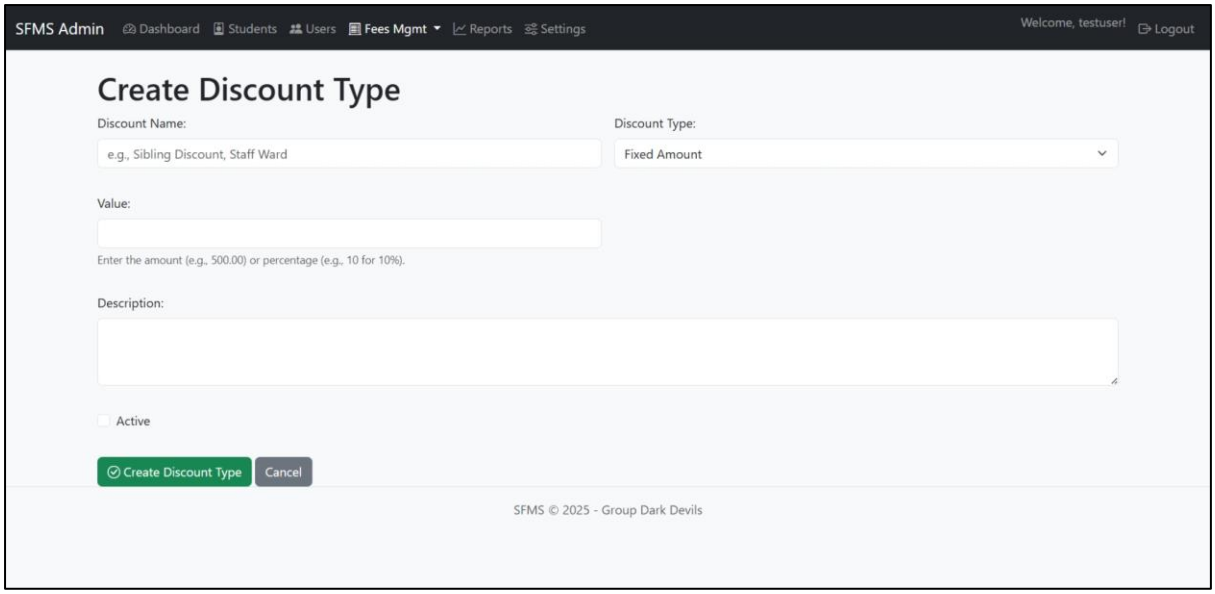


Figure 22: Discount Type management - Create Form

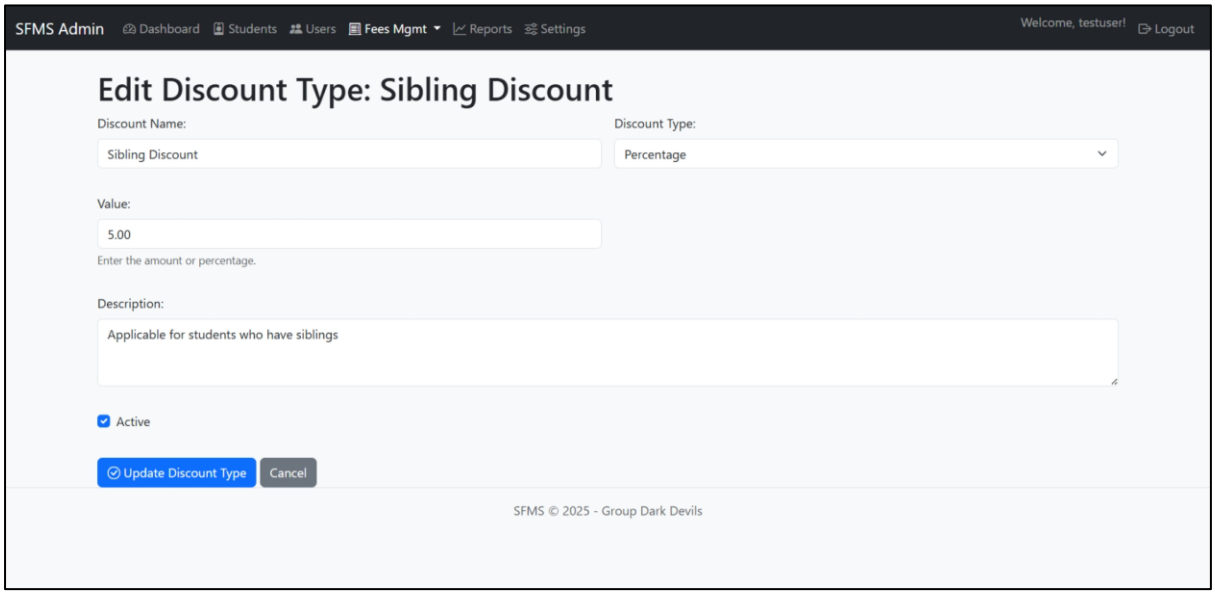


Figure 23: Discount Type management - Edit Form

- **Fee Structure Management (`/admin/fees/structures/...` - `FeeStructureController`, Structure views)**
    - **List View (`index.php`):** Bootstrap table showing Fee Structures with ID, Session, Category, Name, applicable Class, Amount, Frequency, Due Day, Late Fee policy. Actions: "Edit", "Delete" (CSRF).

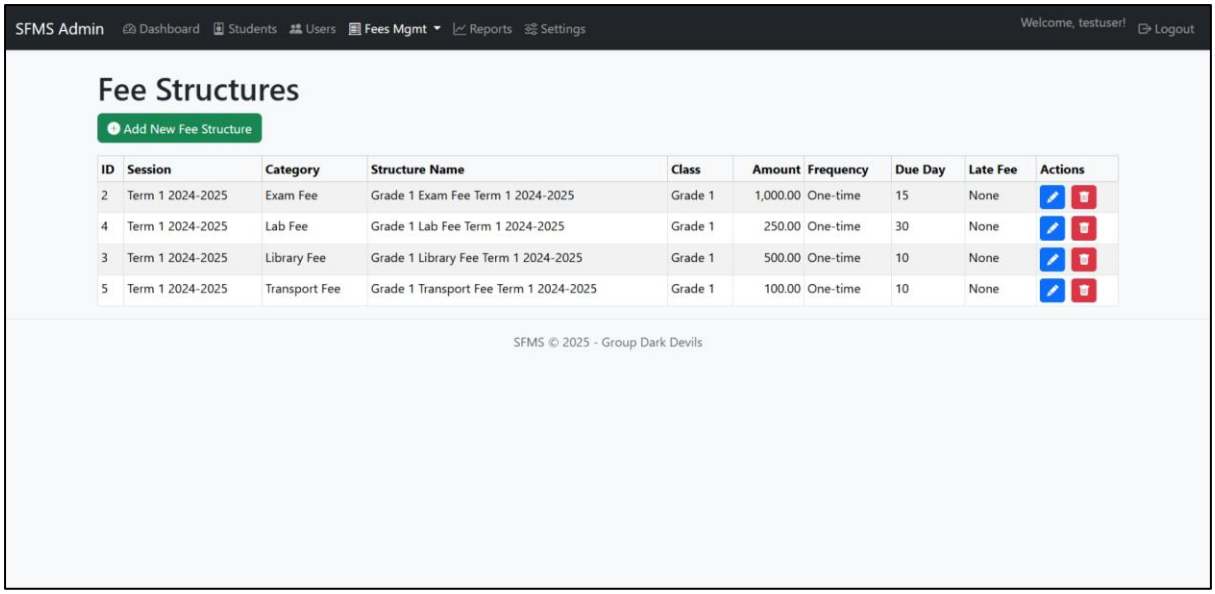


Figure 24: Fee Structure Management - List View

- **Create/Edit Forms (`create.php`, `edit.php`):** Multi-fieldset Bootstrap forms for defining structure name, description, linking to Academic Session and Fee Category (dropdowns), setting amount, frequency (dropdown), applicable class (dropdown, optional "All Classes"), due day, and detailed late fee rules (type dropdown, amount, basis).

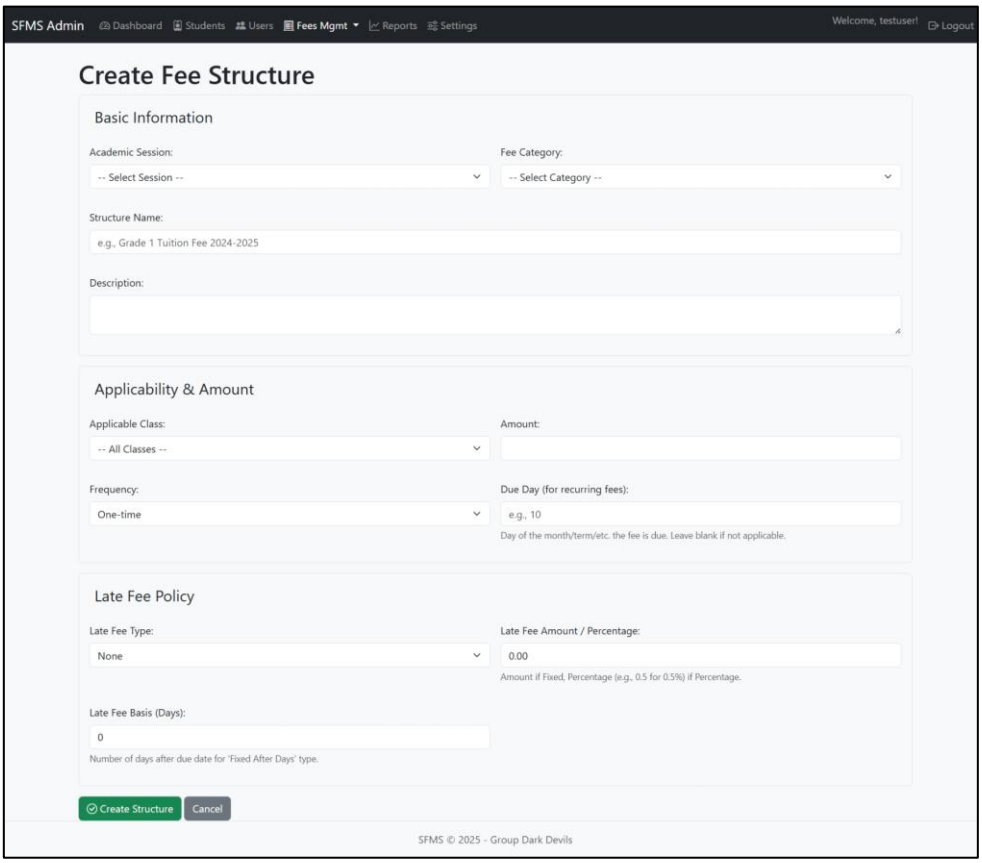


Figure 25: Fee Structure Management - Create Form

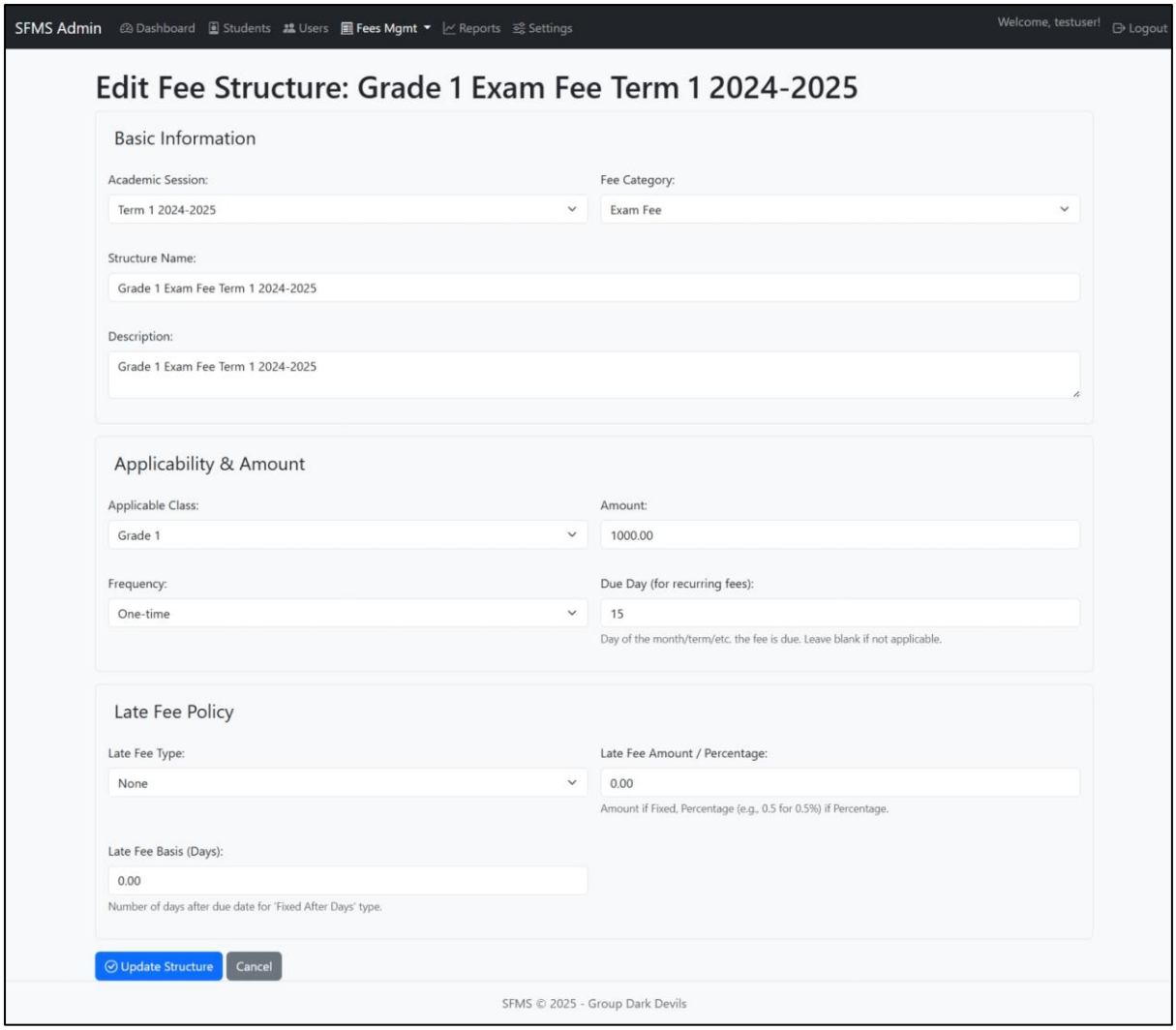


Figure 26: Fee Structure Management - Edit Form

- **Invoice Management (`/admin/fees/invoices/...` - `InvoiceController`, `InvoiceGenerationController`, `InvoiceDiscountController`, Invoice views)**
    - **Generate Invoices Form (`.../generate` - `generate_form.php`):** Bootstrap form allowing selection of Academic Session, Class, and multiple Fee Structures (using checkboxes in a scrollable list) to generate invoices in batch for active students. CSRF protected.

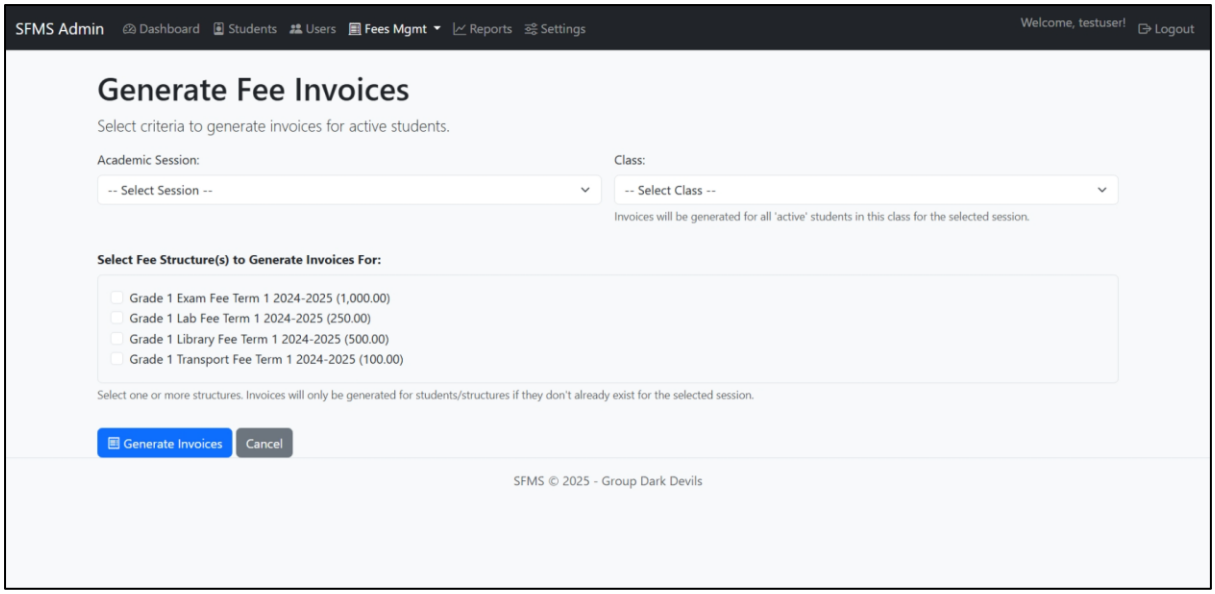


Figure 27: Generate Invoices Form

- **Invoice List View (`index.php`):** Paginated Bootstrap table of all generated invoices, showing Invoice #, Student, Class, Session, Issue/Due Dates, Payable/Paid/Balance amounts, Status (styled with badges and specific class for overdue). Actions: "View Details" button, "Record Payment" button (if applicable).

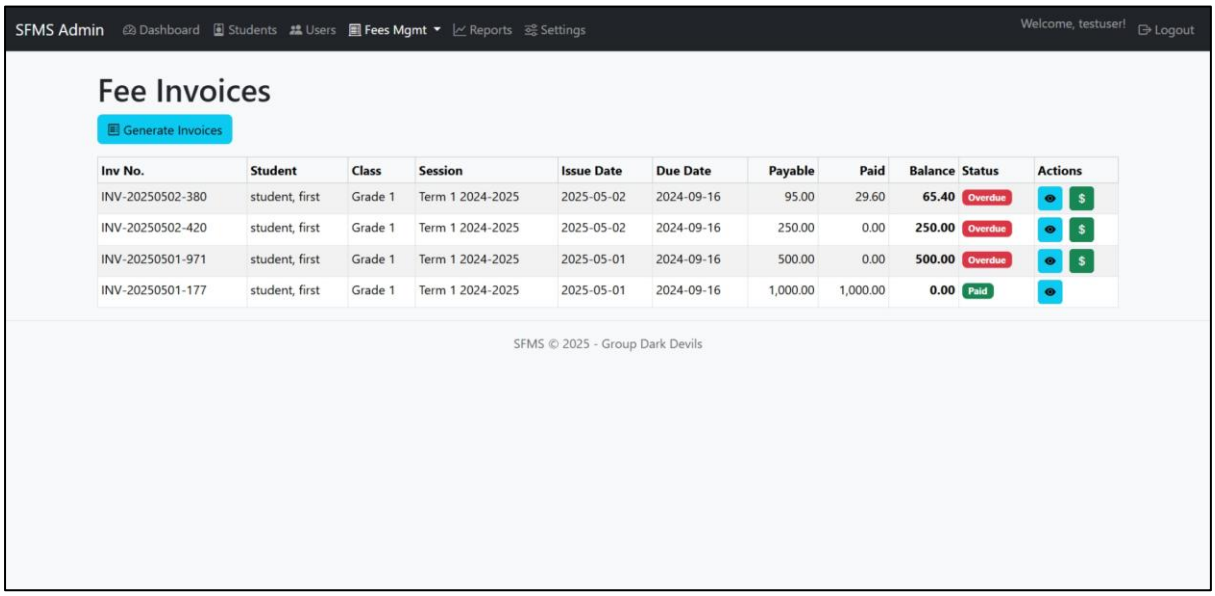


Figure 28: Invoice List View

- **Invoice Detail View (`.../view/{id}` - `view.php`):** A comprehensive Bootstrap-styled page:
  - Invoice header (Invoice #, Dates, Status as badge).
  - "Bill To" student information.

- Table of `fee_invoice_items` (Description, Category, Amount).
- Financial summary section (Subtotal/Total Amount, Total Discount, Total Payable, Total Paid, Balance Due).
- **Discounts Section:** Lists currently applied discounts (from `invoice_discounts`) with name, applied amount, notes, and a "Remove" button/form (CSRF protected) for each. Includes a form to apply new available `discount_types` (Select2 dropdown), optionally override the calculated/default amount, and add notes (CSRF protected).
- **Payments Received Section:** Lists payments allocated to this invoice, with date, method, receipt #, reference, amount, and a "Record Refund" link (styled as a button) for eligible payments.
- "Print Invoice" button.

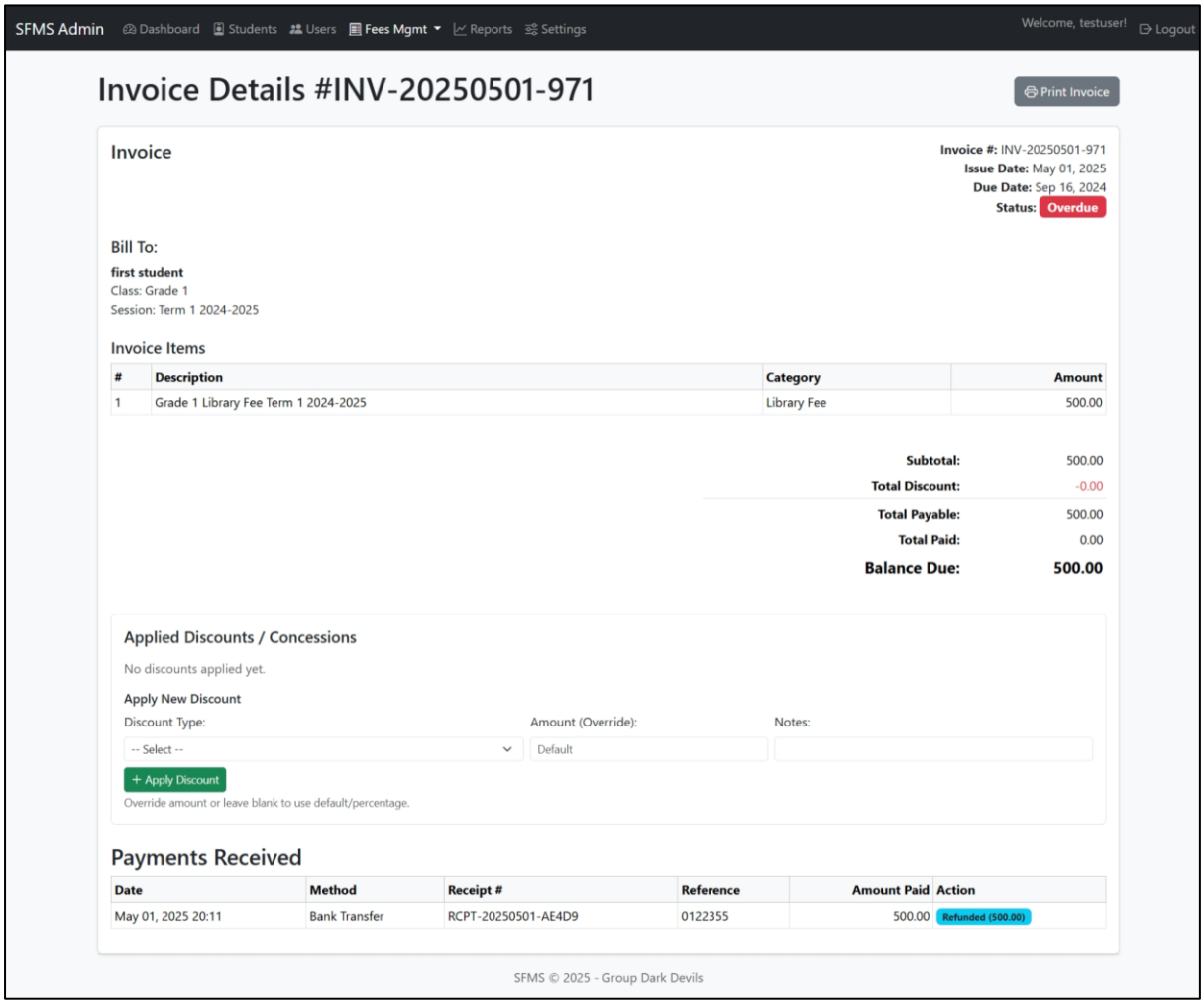


Figure 29: Invoice Detail View

- **Payment Management (`/admin/payments/...` - `PaymentController`, Payment views)**

- **Pending Proofs List (`.../proofs` - `list_proofs.php`):** Bootstrap table of payment proofs with status 'pending'. Shows uploader, invoice #, student, balance due, and a link (styled as a button with icon) to securely view the uploaded file. Actions: "Approve & Record Payment" button (links to Record Payment form pre-filled), "Reject" form (with notes input and CSRF).

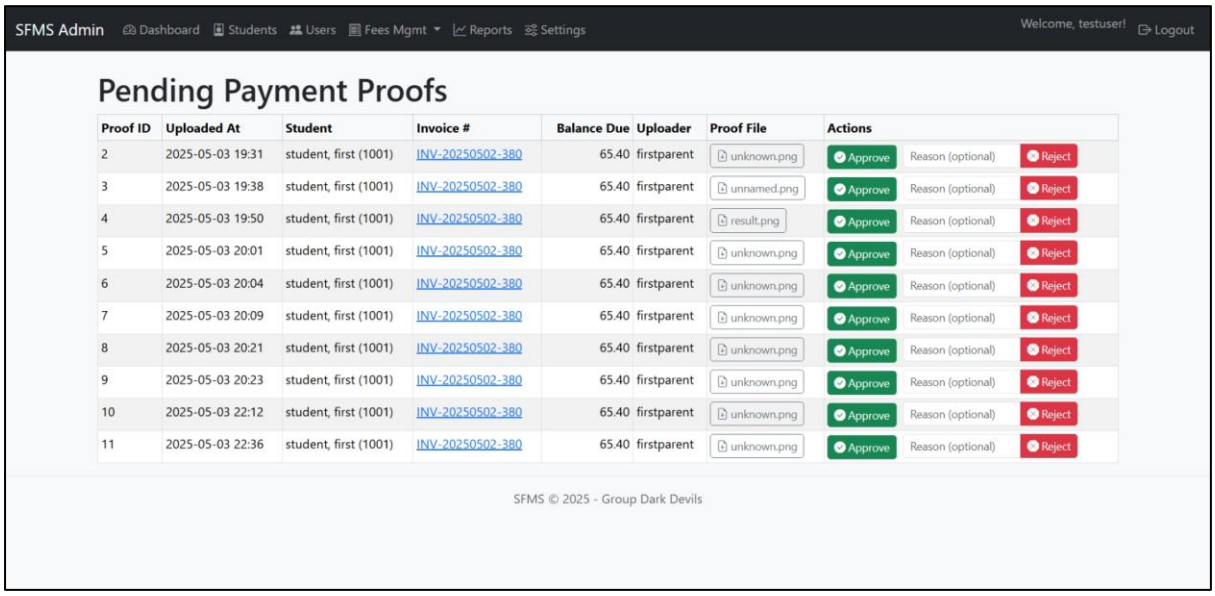


Figure 30: Pending Proofs List

- **Record Payment Form (`.../record/{invoice_id}` - `record_form.php`):** Bootstrap form used for direct manual payment entry and for verifying/approving uploaded proofs. Displays invoice details in a card. If verifying a proof, it shows proof details in a distinct card with a link/preview of the file. Fields for amount paid, payment date, method (dropdown), reference, notes. CSRF protected. Submit button text changes if verifying a proof.

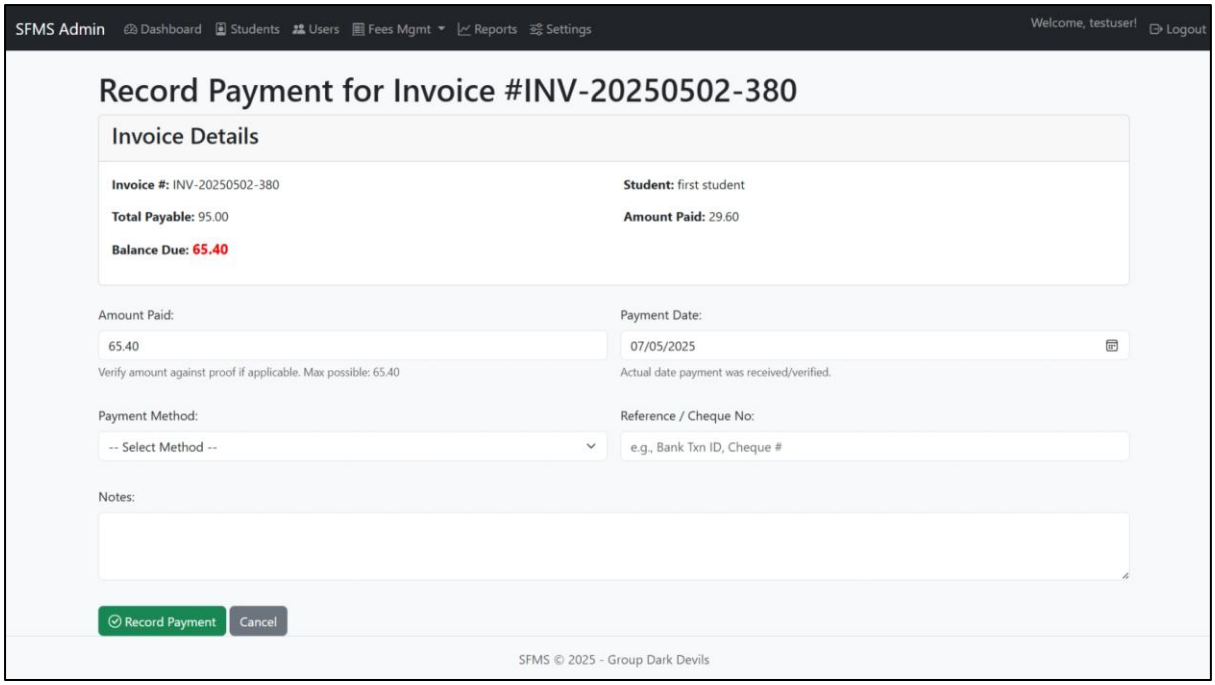


Figure 31: Record Payment Form

- **Refund Form (`.../{payment_id}/refund` - `refund_form.php`):** Displays original payment details (in a card) and max refundable amount. Bootstrap form to enter refund amount, date, reason. CSRF protected.

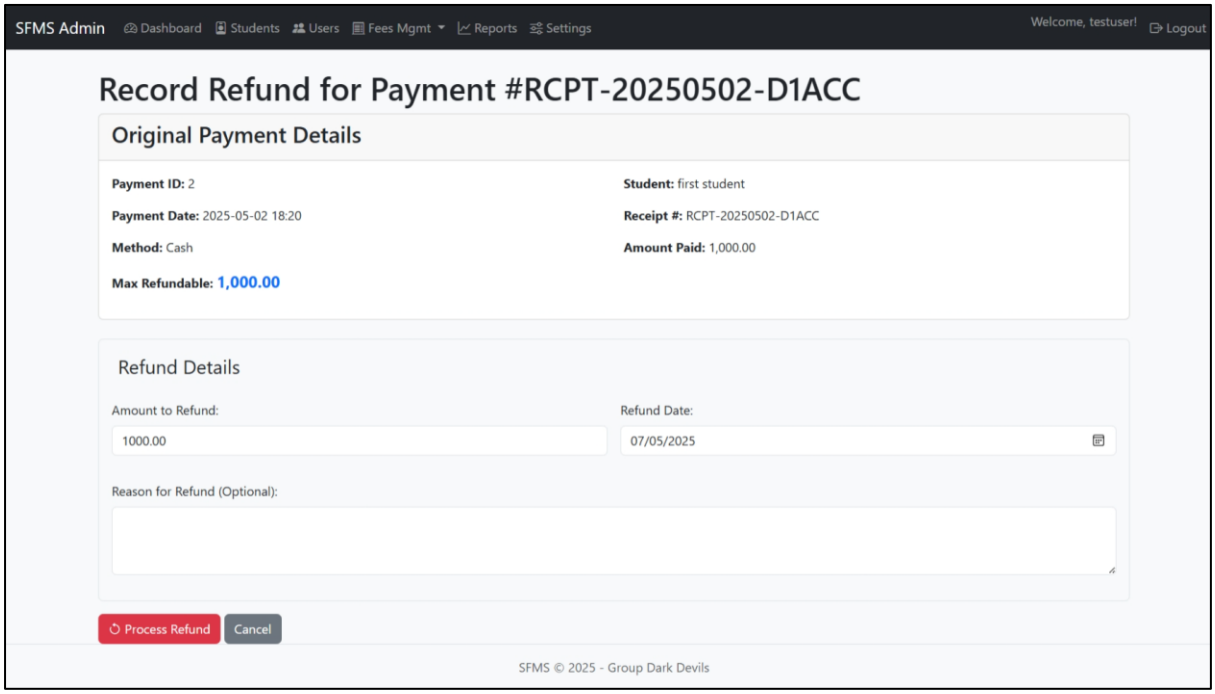


Figure 32: Refund Form

- **Reporting (`/admin/reports/...` - `ReportController`, Report views)**

- **Reports Menu (`index.php`):** Bootstrap list group linking to all available reports. "Audit Log" link is conditional on Admin role. Includes a Bootstrap card with a form/button to manually trigger fee reminders (CSRF protected).

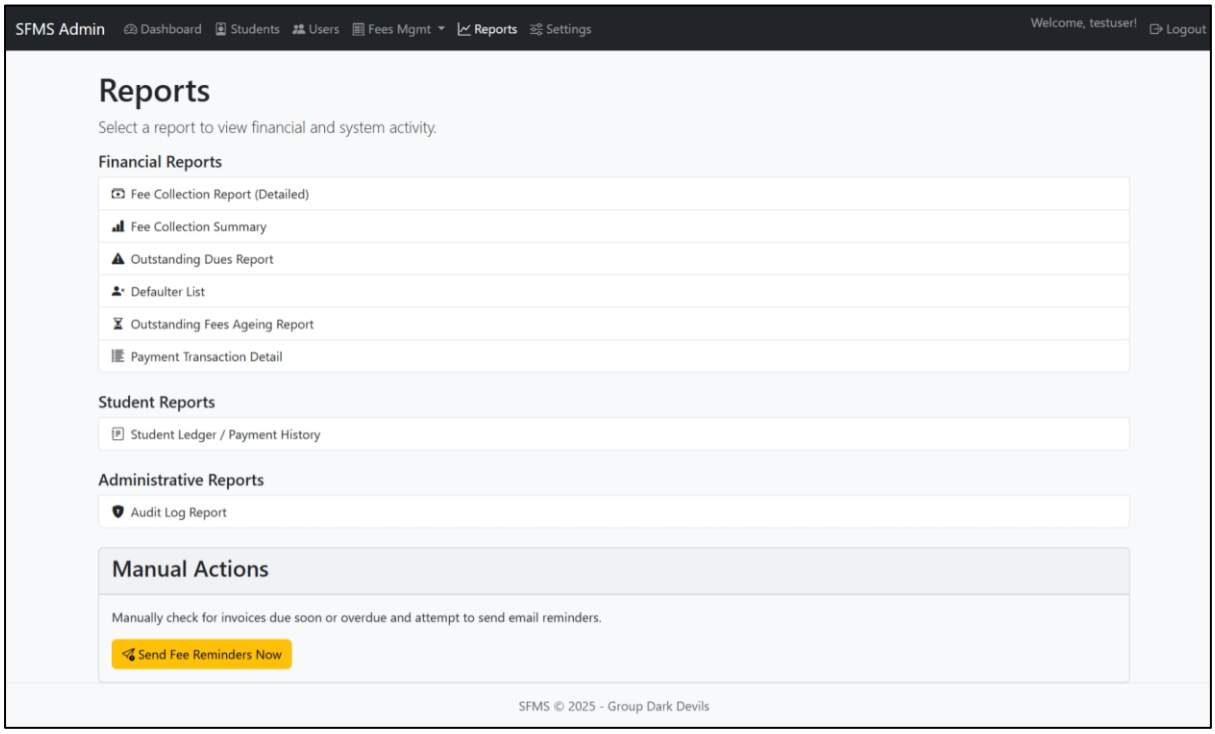


Figure 33: Reports Menu

- **Individual Report Views:** Each report view (Fee Collection, Collection Summary, Outstanding Dues, Student Ledger, Defaulter List, Ageing, Transaction Detail, Audit Log) features:
    - Filter forms styled with Bootstrap cards and form controls (date pickers, dropdowns - student selector uses Select2).
    - Results displayed in Bootstrap tables with appropriate formatting (e.g., currency, dates) and totals.
    - Pagination controls styled with Bootstrap for reports with many records.

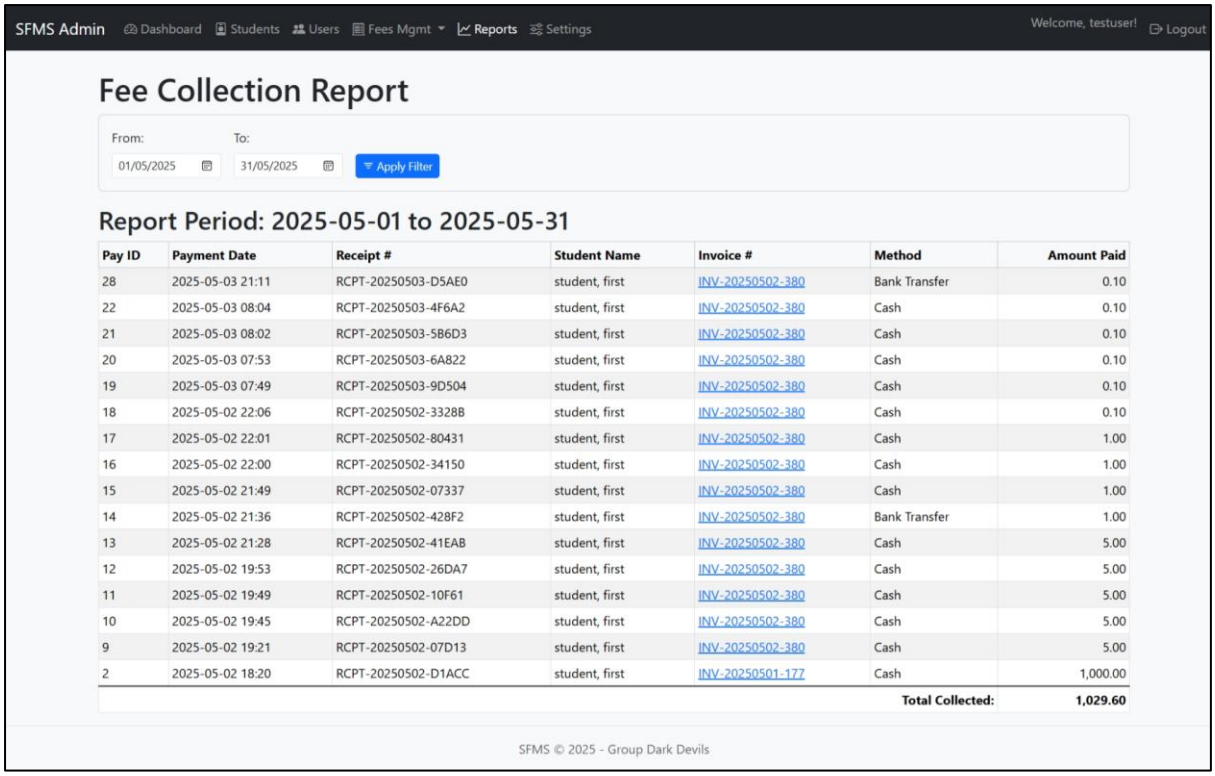


Figure 34: Fee Collection Report

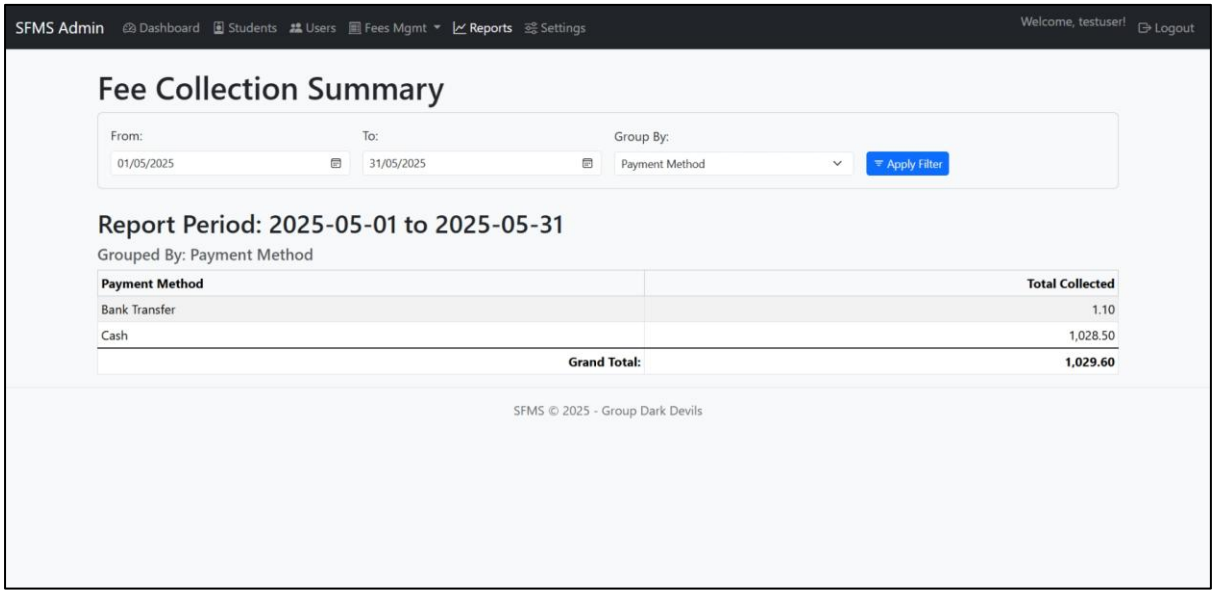


Figure 35: Collection Summary Report

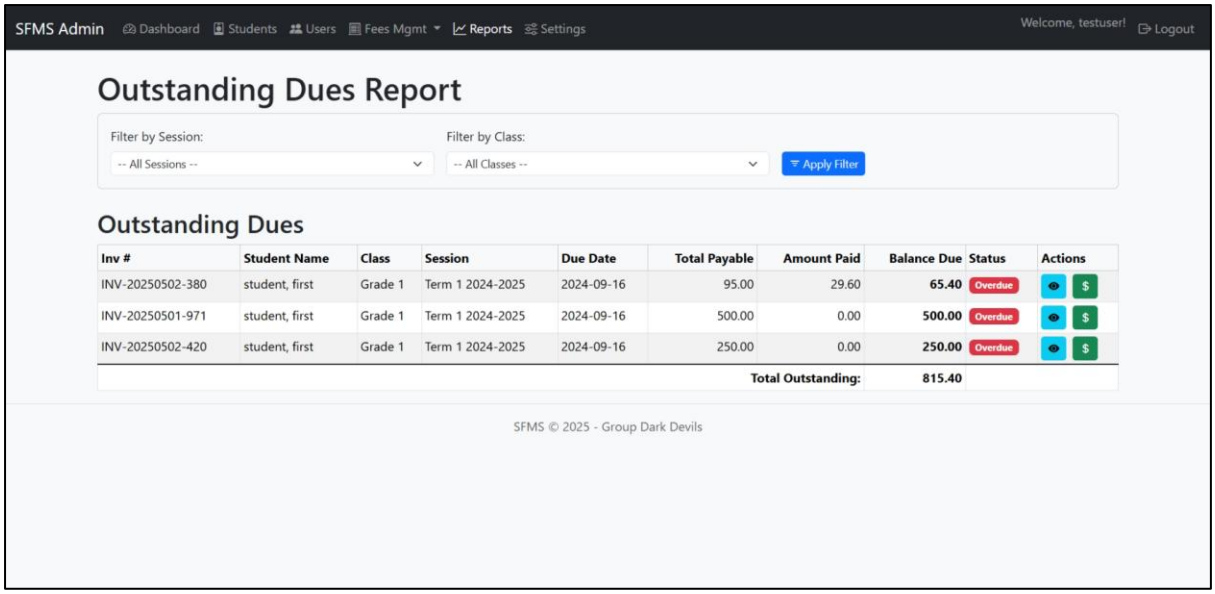


Figure 36: Outstanding Dues Report

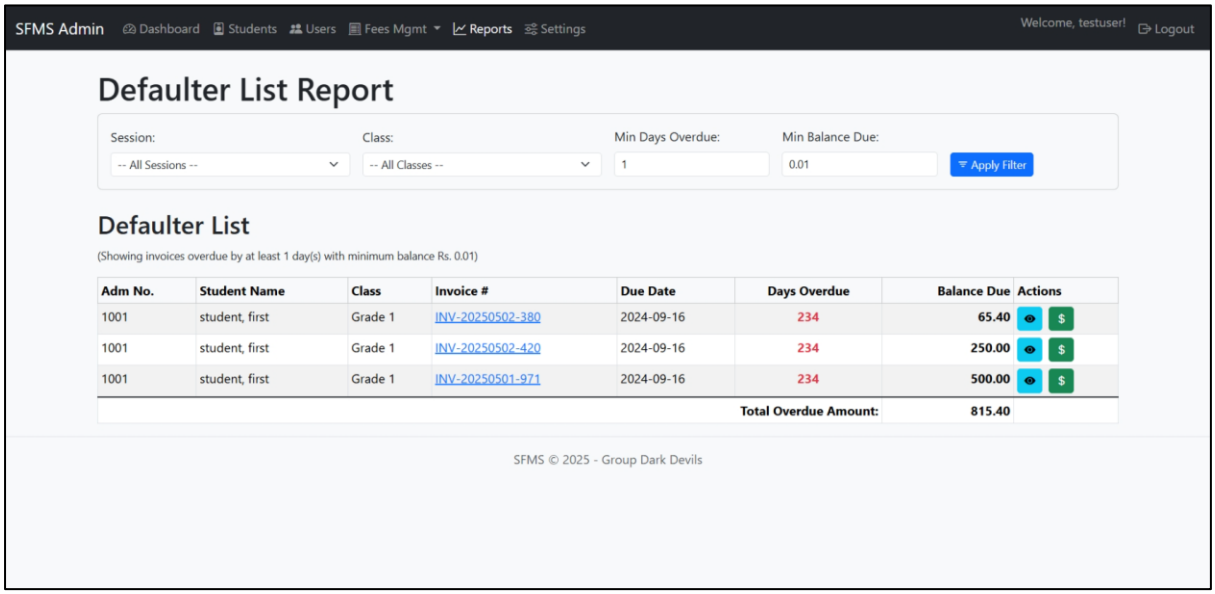


Figure 37: Defaulter List Report

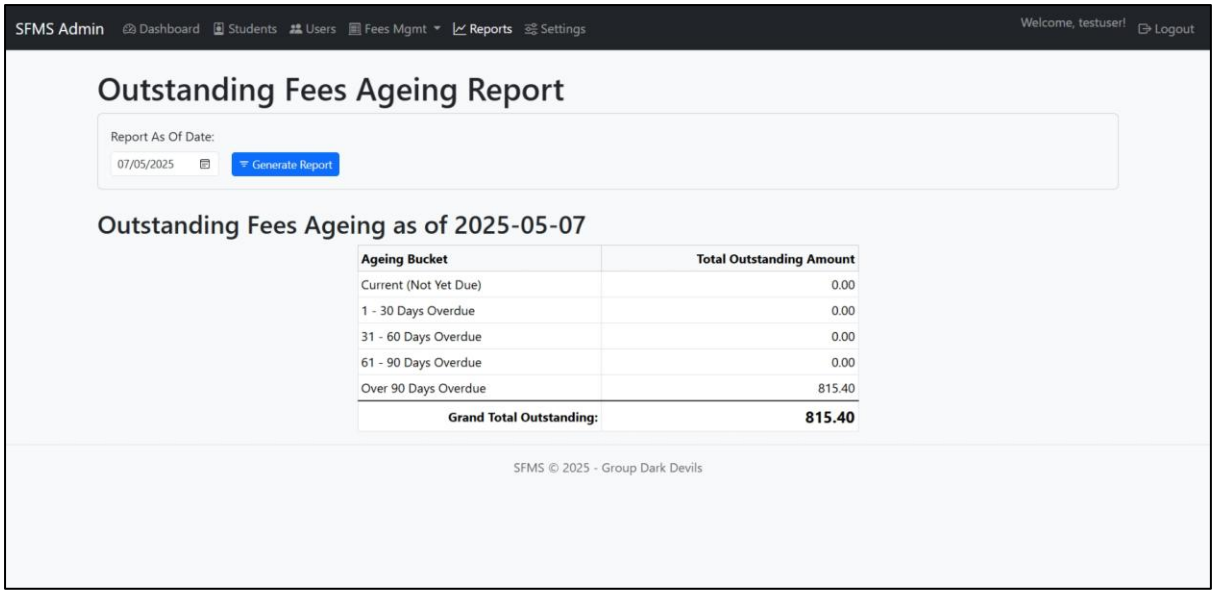


Figure 38: Outstanding Fee Ageing Report

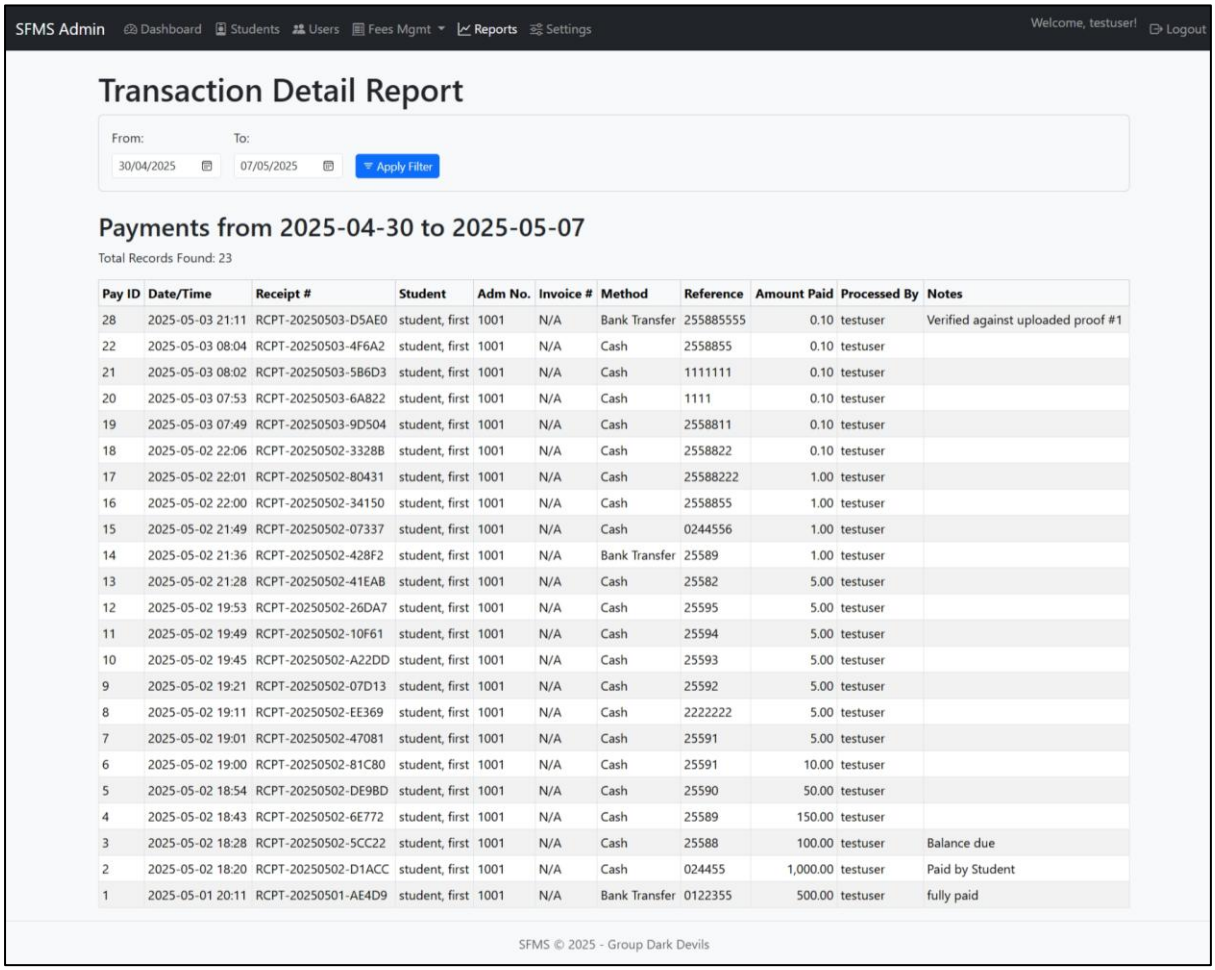
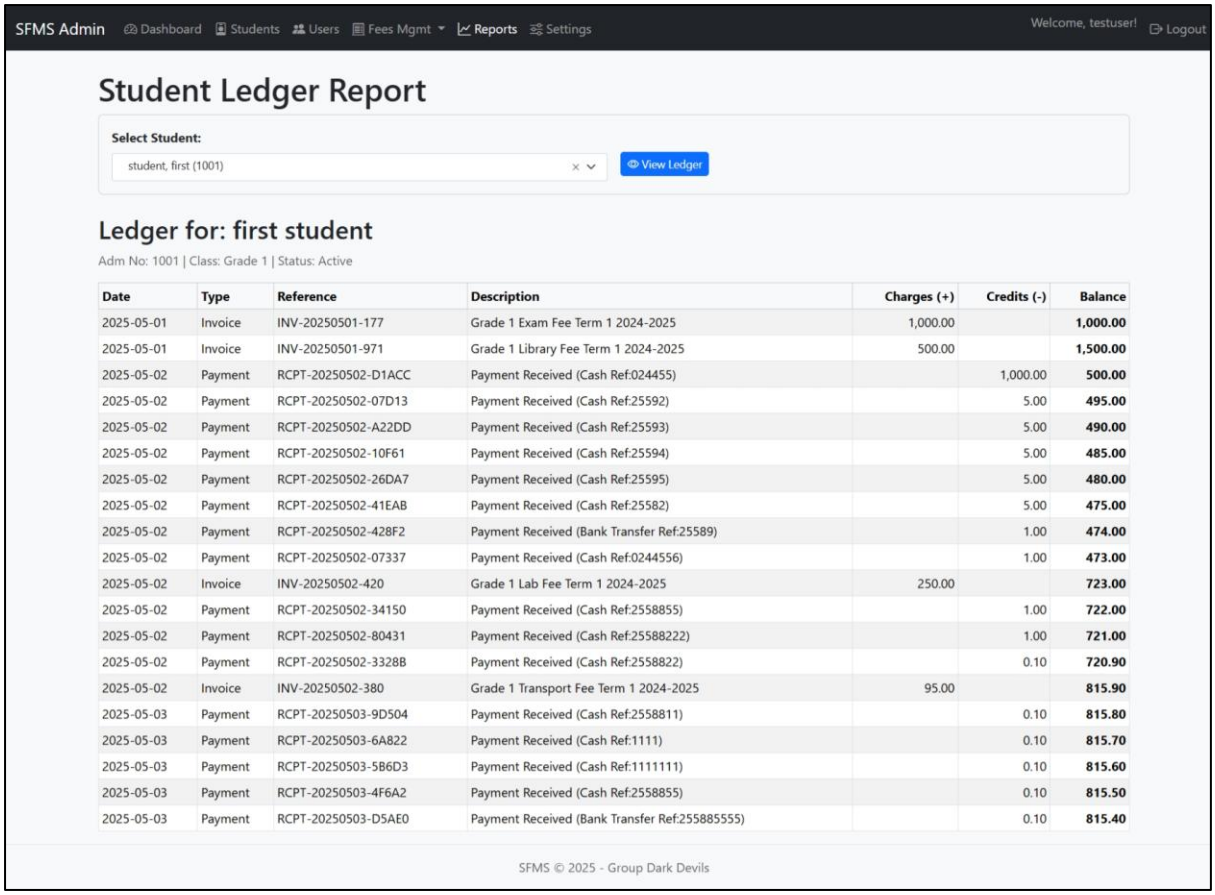


Figure 39: Transaction Detail Report



Date	Type	Reference	Description	Charges (+)	Credits (-)	Balance
2025-05-01	Invoice	INV-20250501-177	Grade 1 Exam Fee Term 1 2024-2025	1,000.00		**1,000.00**
2025-05-01	Invoice	INV-20250501-971	Grade 1 Library Fee Term 1 2024-2025	500.00		**1,500.00**
2025-05-02	Payment	RCPT-20250502-D1ACC	Payment Received (Cash Ref:024455)		1,000.00	**500.00**
2025-05-02	Payment	RCPT-20250502-07D13	Payment Received (Cash Ref:25592)		5.00	**495.00**
2025-05-02	Payment	RCPT-20250502-A22DD	Payment Received (Cash Ref:25593)		5.00	**490.00**
2025-05-02	Payment	RCPT-20250502-10F61	Payment Received (Cash Ref:25594)		5.00	**485.00**
2025-05-02	Payment	RCPT-20250502-26DA7	Payment Received (Cash Ref:25595)		5.00	**480.00**
2025-05-02	Payment	RCPT-20250502-41EAB	Payment Received (Cash Ref:25582)		5.00	**475.00**
2025-05-02	Payment	RCPT-20250502-428F2	Payment Received (Bank Transfer Ref:25589)		1.00	**474.00**
2025-05-02	Payment	RCPT-20250502-07337	Payment Received (Cash Ref:0244556)		1.00	**473.00**
2025-05-02	Invoice	INV-20250502-420	Grade 1 Lab Fee Term 1 2024-2025	250.00		**723.00**
2025-05-02	Payment	RCPT-20250502-34150	Payment Received (Cash Ref:2558855)		1.00	**722.00**
2025-05-02	Payment	RCPT-20250502-80431	Payment Received (Cash Ref:25588222)		1.00	**721.00**
2025-05-02	Payment	RCPT-20250502-3328B	Payment Received (Cash Ref:2558822)		0.10	**720.90**
2025-05-02	Invoice	INV-20250502-380	Grade 1 Transport Fee Term 1 2024-2025	95.00		**815.90**
2025-05-03	Payment	RCPT-20250503-9D504	Payment Received (Cash Ref:2558811)		0.10	**815.80**
2025-05-03	Payment	RCPT-20250503-6A822	Payment Received (Cash Ref:1111)		0.10	**815.70**
2025-05-03	Payment	RCPT-20250503-5B6D3	Payment Received (Cash Ref:1111111)		0.10	**815.60**
2025-05-03	Payment	RCPT-20250503-4F6A2	Payment Received (Cash Ref:2558855)		0.10	**815.50**
2025-05-03	Payment	RCPT-20250503-D5AE0	Payment Received (Bank Transfer Ref:255885555)		0.10	**815.40**

Figure 40: Student Ledger Report

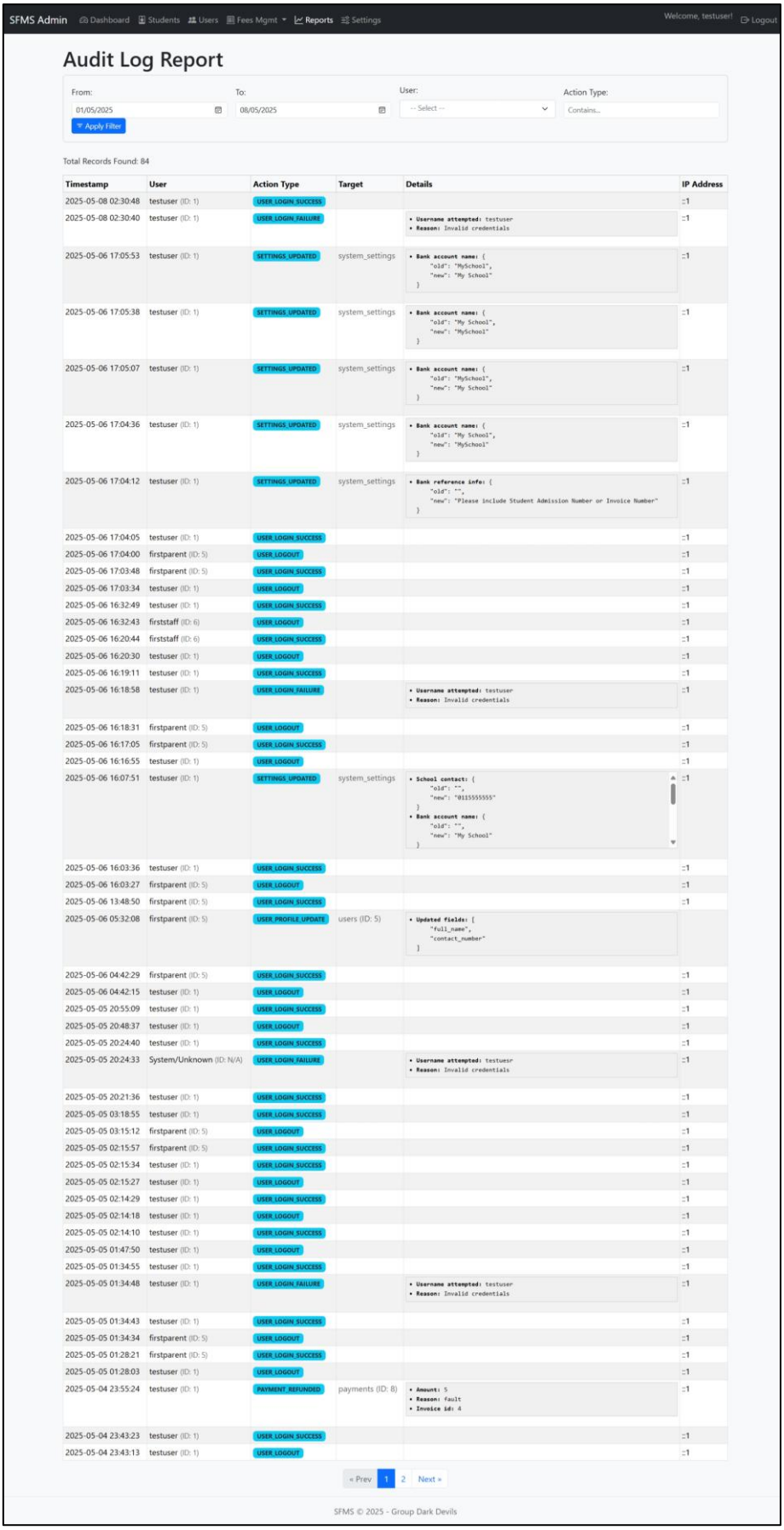


Figure 41: Audit Log Report

- **System Settings (`/admin/settings` - `SettingsController`, `settings_form.php`)**
    - Admin-only page with a Bootstrap form (CSRF protected) to update non-sensitive system-wide configurations (School Name, Address, Contact, Currency Symbol, Bank Details for portal display, Email "From" Name, Reminder Timings). Settings are grouped into Bootstrap cards for better organization and clarity.

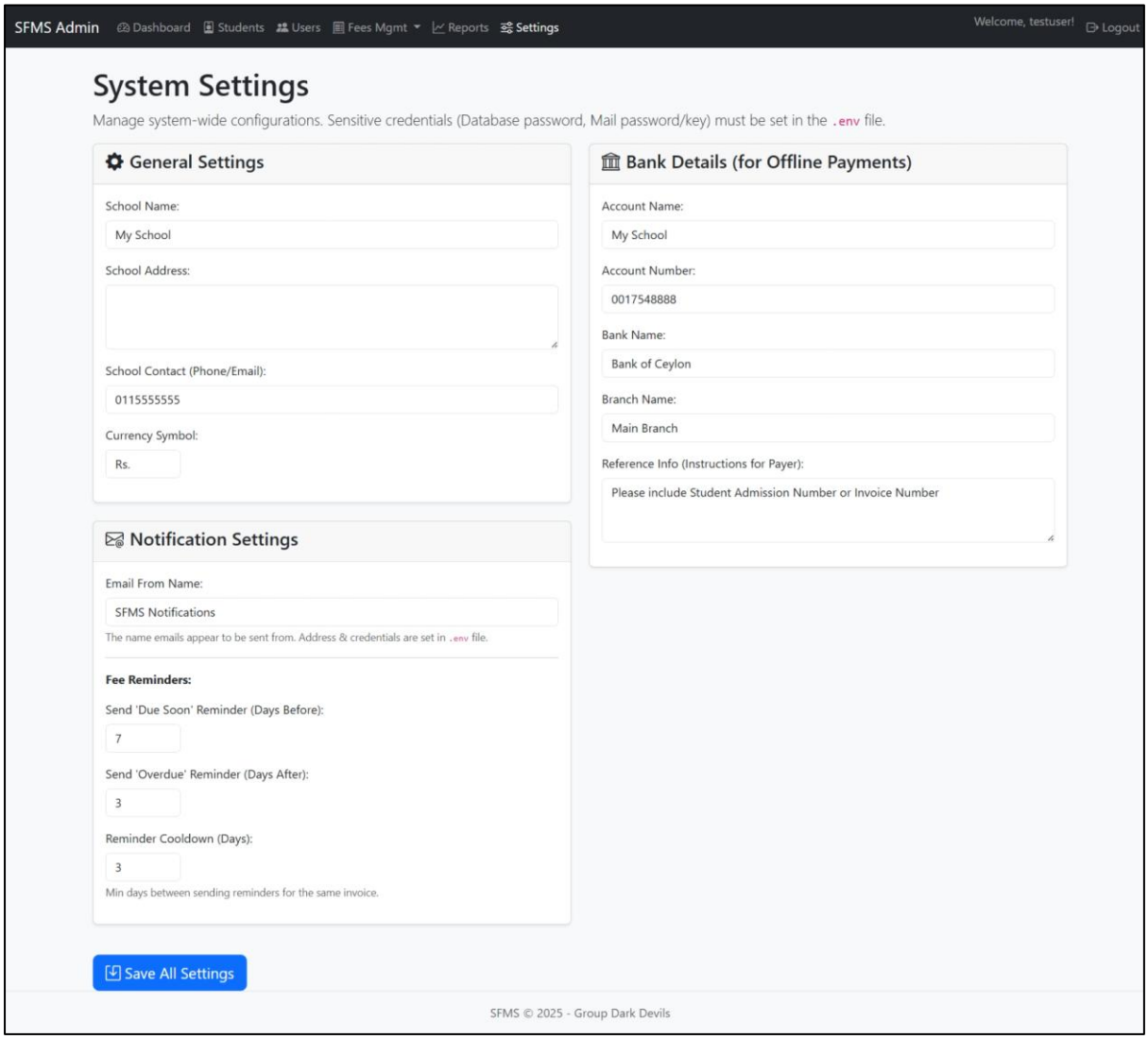


Figure 42: System Settings

5.1.3 Student/Parent Portal Interface (`/portal/...`)

The portal provides a secure and focused interface for students and parents, using a distinct, lighter Bootstrap layout (`layout_portal.php`) managed by `BaseController::loadView()`.

- **Common Portal Layout Features (`_header_portal.php`, `_footer_portal.php`):**

- **Navigation Bar:** A light-themed, responsive Bootstrap navbar (`navbar-light bg-white`) with links to Dashboard, My Fees, My Profile, user greeting, and a Logout link. Active menu items are highlighted.
- **Flash Messages:** Global success/error messages displayed below the navbar.
- **Main Content Area:** A Bootstrap `container` for page-specific content.
- **Footer:** Simple footer with copyright.
- **Assets:** Includes Bootstrap CSS/JS, Bootstrap Icons, Select2 CSS/JS, and a global Select2 initializer.

- **Portal Dashboard (`/portal/dashboard` - `PortalDashboardController::index`, `dashboard.php`)**
  - Welcomes the user.
  - If the logged-in user is a 'Parent' linked to multiple children, displays a Select2 searchable dropdown ("Viewing Child:") to choose which child's data to view. The selection persists across relevant portal pages via GET parameters.
  - Displays basic details of the selected student (Name, Admission No., Class) in a Bootstrap card.
  - Shows the current outstanding fee balance for the selected child in a prominent, color-coded (e.g., red for due, green if zero) Bootstrap card with a link to detailed fee history.
  - Includes a placeholder for school announcements.

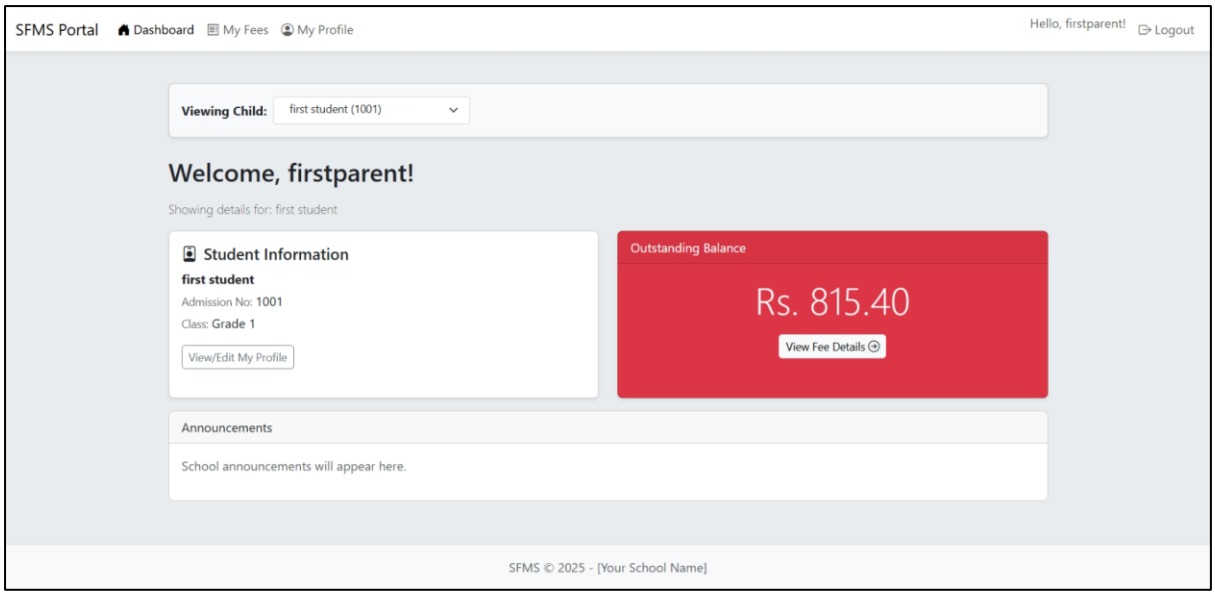


Figure 43: Portal Dashboard

- **My Fees & Payments (`/portal/fees` - `PortalFeeController::index`, `fees.php`)**
    - Includes the child selector for parents.
    - Lists all non-cancelled invoices for the selected student in a Bootstrap table, showing Invoice #, Description, Issue/Due Dates, Amount, Paid, Balance, and Status (styled with Bootstrap badges). Actions include a "View" button (links to detailed invoice) and a "Pay Offline / Upload Proof" button (if invoice is not paid/cancelled).
    - Lists payment history for the selected student in a separate Bootstrap table, showing Date, Receipt #, Method, Reference, Notes, Amount Paid, and linked Invoice # (if applicable).

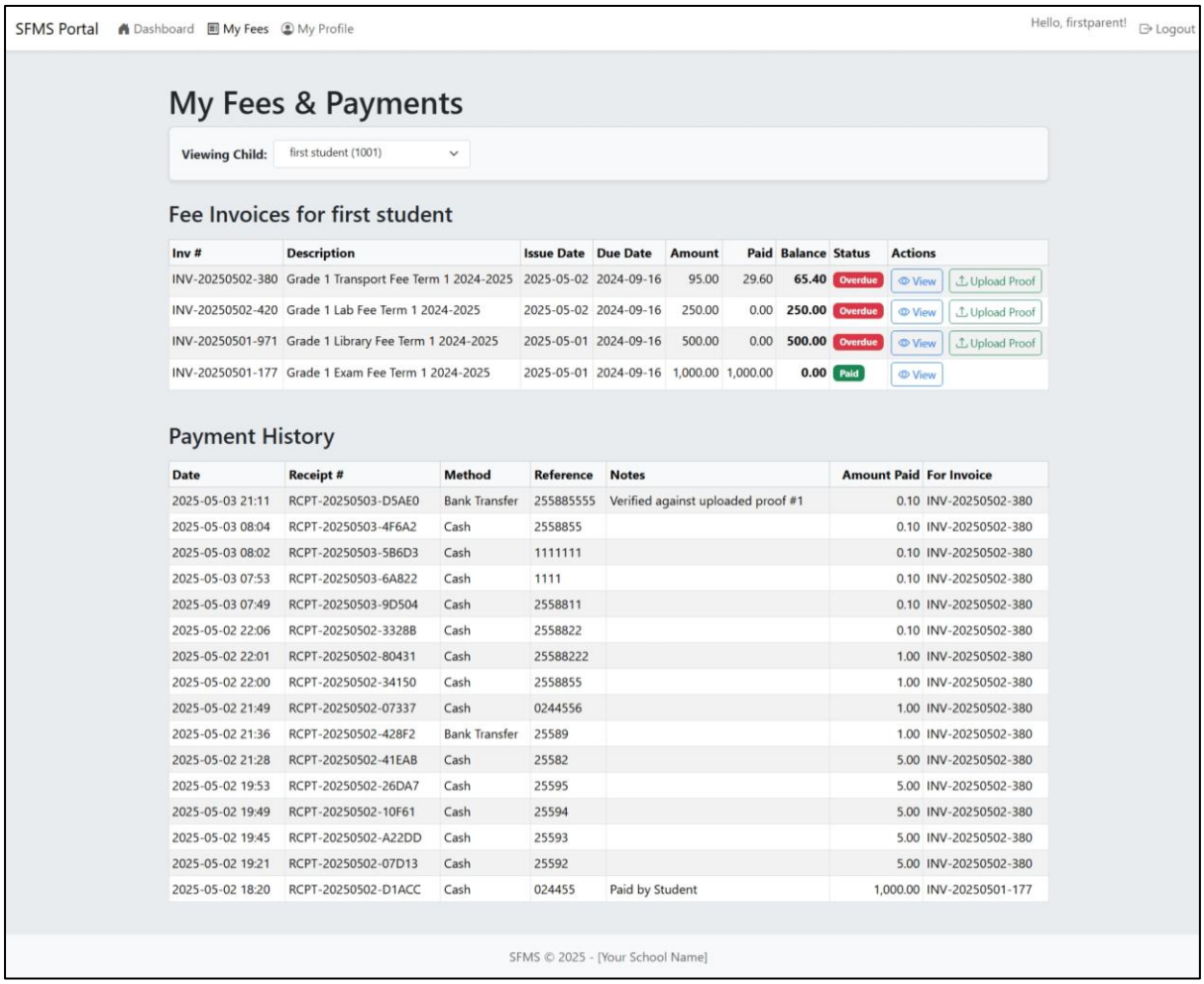


Figure 44: My Fees & Payments

- **Invoice Detail (Portal) (`/portal/invoices/view/{id}` - `PortalInvoiceController::view`, `invoice_detail.php`)**
    - Bootstrap-styled page displaying full details of a specific invoice authorized for the user/selected child.

- Includes invoice header (Invoice #, Dates, Status badge), "Bill To" student information, a table of invoice items, a list of applied discounts (read-only), a financial summary (Subtotal, Discount, Payable, Paid, Balance Due), and a table of payments received for that invoice.
- Includes a "Print Invoice" button.

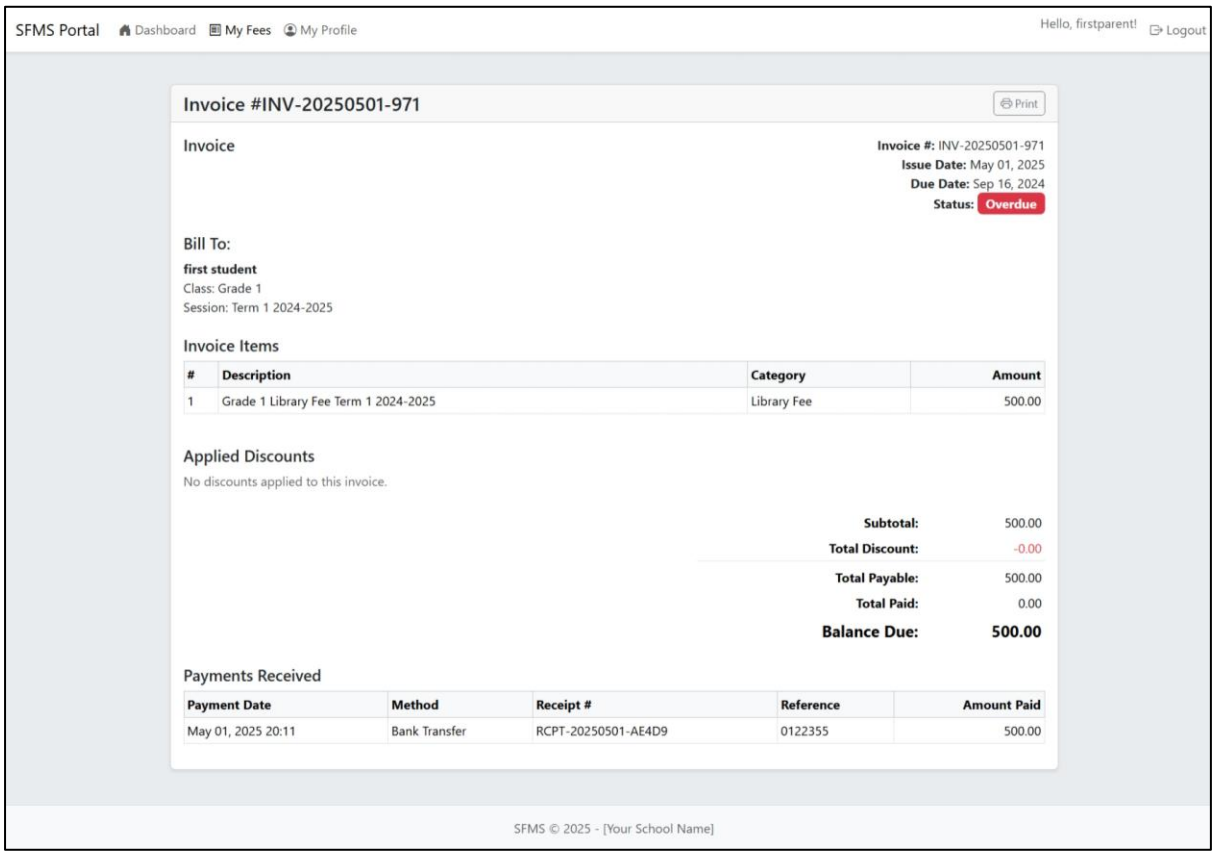


Figure 45: Invoice Detail (Portal)

- **Offline Payment/Proof Upload (`/portal/payments/offline/{invoice_id}` - `PortalPaymentController::showOfflinePaymentInstructions`, `offline_payment.php`)**
  - Displays the school's bank account details (read from system settings) in a styled Bootstrap card.
  - Provides clear instructions for making an offline payment and the importance of the reference.
  - Includes a Bootstrap-styled form (with CSRF protection) for uploading the payment proof (accepts JPG, PNG, PDF within size limits).

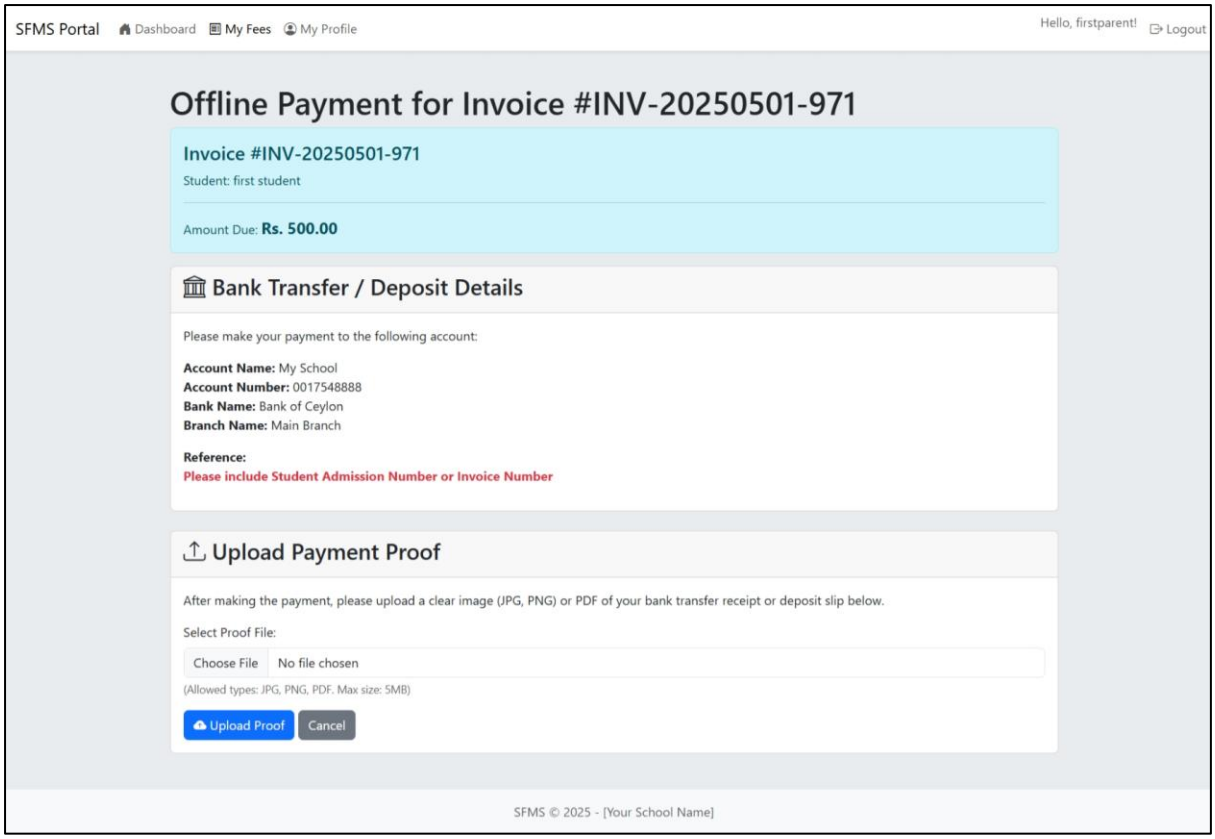


Figure 46: Offline Payment/Proof Upload

- **My Profile (`/portal/profile` - `PortalProfileController::show`, `profile.php`)**
    - Displays the logged-in user's read-only username and email.
    - Provides a Bootstrap form (CSRF protected) within a card to update their Full Name and Contact Number.
    - Provides a separate Bootstrap form (CSRF protected) within another card to change their password, requiring current password, new password, and confirmation. Validation errors are displayed.

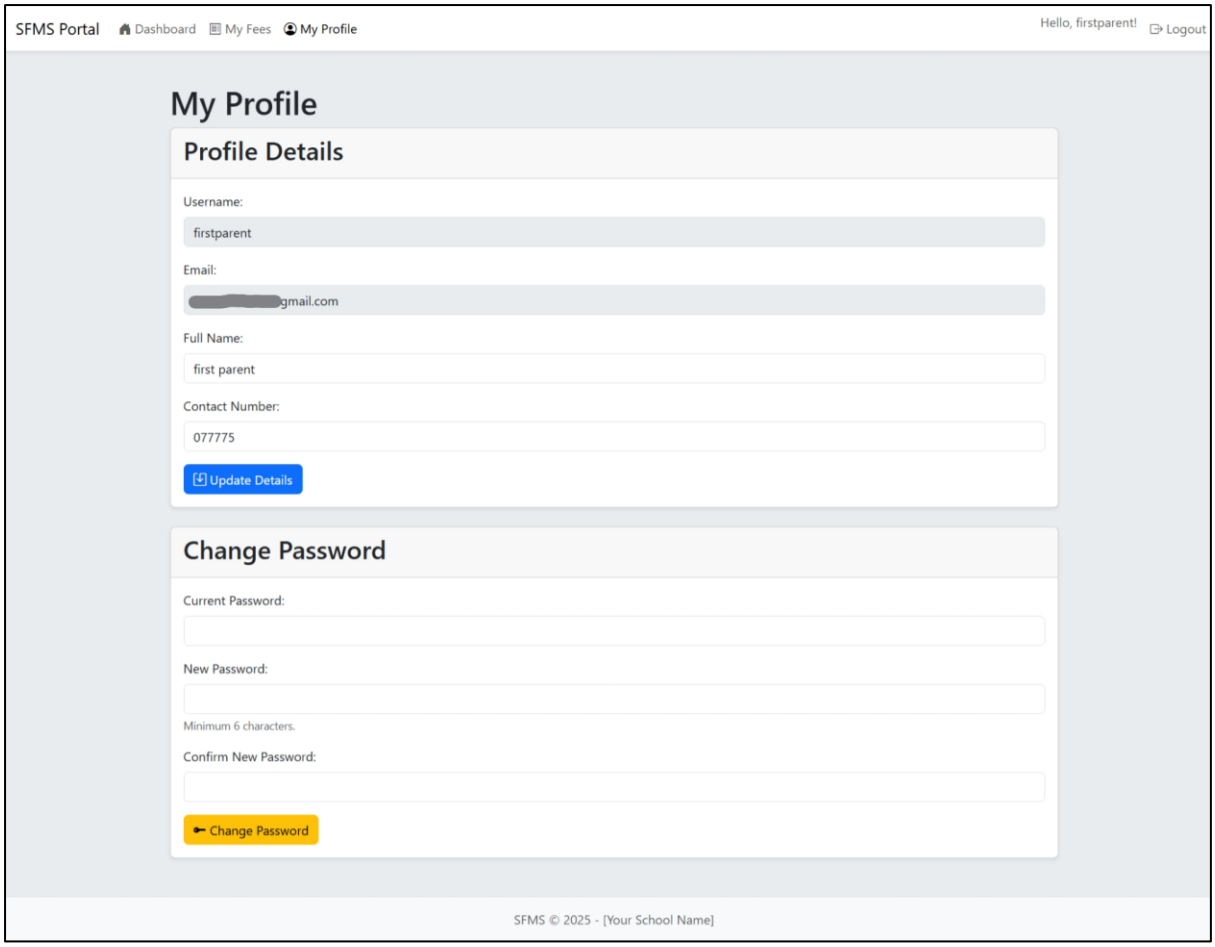


Figure 47: My Profile

Chapter 6 - Conclusion and Future Work

This chapter concludes the report on the School Fees Management System (SFMS) project. It provides a summary of the project's accomplishments against its objectives, outlines potential avenues for future development and enhancement, discusses the challenges encountered during the development lifecycle, and provides a section for detailing the contributions of each project member.

6.1 Summary of the Project

The School Fees Management System (SFMS) project was undertaken to address the significant inefficiencies and challenges associated with manual fee administration in educational institutions. The primary aim was to develop a comprehensive, secure, and user-friendly web-based application to automate and streamline the entire fee management lifecycle. This aim has been substantially achieved through the implementation of a robust system catering to the needs of Administrators, Staff, Students, and Parents.

The developed SFMS successfully delivers on its core objectives. A secure authentication system with role-based access control (RBAC) ensures that users can only access functionalities pertinent to their roles. The administrative backend provides comprehensive CRUD (Create, Read, Update, Delete) capabilities for managing Users, Students (including linking to user accounts and multiple guardians via a dedicated `student_guardian_links` table), Fee Categories, Discount Types (percentage or fixed amount), and detailed Fee Structures (assignable to classes/sessions with specific frequencies and late fee policies).

A key achievement is the implementation of the complete fee processing workflow. This includes manual batch invoice generation with duplicate prevention, the ability for administrators to apply and remove various discounts to invoices with automatic recalculation of totals, and a system for recording offline payments. The latter is enhanced by a portal feature allowing students/parents to upload payment proofs, which administrators can then verify or reject. Refund processing against completed or partially paid payments, with corresponding updates to invoice and payment statuses, has also been implemented.

The Student/Parent Portal offers a dedicated and secure interface for users to view their dashboard (with a child selection mechanism for parents linked to multiple students), detailed fee statements, payment history, and individual invoice breakdowns including applied discounts. They can also manage their profiles (update details, change passwords) and submit payment proofs for offline transactions after viewing the school's bank details (managed via system settings).

To support financial oversight and operational management, a comprehensive reporting module was developed. This includes reports such as Fee Collection (detailed and summarized by various criteria like payment method, class, or fee category), Outstanding Dues, individual Student Ledgers, Defaulter Lists, Accounts Receivable Ageing, Payment Transaction Details,

and a system-wide Audit Log for tracking significant user actions. These reports feature filtering capabilities and pagination for ease of use.

The notification system, while currently relying on manual triggers for reminders, is capable of sending email notifications (e.g., payment confirmations) using PHPMailer and database-driven templates. It includes logic to prevent re-sending reminders too frequently. Furthermore, a system settings module allows administrators to manage non-sensitive, application-wide configurations like school contact information, bank details for display, currency symbols, and default notification parameters via the UI.

Technologically, the system was built on a PHP backend, MySQL database, and a Bootstrap 5 frontend, ensuring a responsive and modern user experience. Security considerations such as CSRF protection, prepared SQL statements, and secure password hashing have been integrated. The modular architecture and use of service/helper classes promote maintainability and potential for future expansion.

In essence, the SFMS project has delivered a functional and feature-rich platform that significantly modernizes fee management, offering improved efficiency, transparency, and convenience for all stakeholders.

6.2 Future Work

While the current iteration of the SFMS provides a solid foundation and fulfills the core project objectives, there are numerous avenues for future development and enhancement to further increase its value, usability, and comprehensiveness. These include:

1. **Online Payment Gateway Integration:**
    a. **Description:** Integrate one or more popular payment gateways (e.g., Stripe, PayPal, or region-specific gateways like Payhere in Sri Lanka) directly into the Student/Parent Portal.
    b. **Benefits:** Allows parents/students to make fee payments online using credit/debit cards, net banking, or mobile wallets. Automates payment recording and reconciliation, reduces manual effort for staff, and provides instant payment confirmation.
    c. **Considerations:** Requires careful selection of gateway, handling API integrations, managing transaction fees, ensuring PCI DSS compliance for card data (often handled by the gateway itself), and updating payment statuses via webhooks or callbacks.
2. **Full Notification Automation (Scheduling & SMS):**
    a. **Description:** Implement server-side scheduling (using cron jobs on Linux or Scheduled Tasks on Windows) to run the fee reminder logic (currently in

`ReportController::triggerReminders` or the `scripts/send_reminders.php` file) automatically on a daily or configurable basis.

b. **Expand Channels:** Integrate an SMS gateway API into the `NotificationService` to send reminders and critical alerts via SMS, in addition to email.

c. **Benefits:** Ensures timely dispatch of reminders without manual intervention, improves collection rates.

d. **Considerations:** Requires server access for scheduling, costs associated with SMS gateway services.

3. **Advanced Discount, Scholarship, and Concession Module:**

a. **Description:** Develop a more sophisticated module to manage various types of financial aid, scholarships, and complex, rule-based concessions (e.g., automatic sibling discounts based on verified sibling links within the student database, merit-based scholarships with specific criteria).

b. **Functionality:** Allow application of discounts/scholarships to specific fee items within an invoice, rather than just an overall invoice discount. Track scholarship fund utilization.

c. **Benefits:** Provides greater flexibility in managing financial aid and complex discount scenarios accurately.

4. **Installment Payment Plans:**

a. **Description:** Enable administrators to define and assign installment plans for specific fee structures or individual invoices.

b. **Functionality:** Automate the generation of installment due dates and amounts based on the plan. Track payments against each installment and provide clear status updates in both admin and portal views. Send reminders for upcoming installments.

c. **Benefits:** Offers payment flexibility to parents, potentially improving collection rates for larger fee amounts.

5. **Advanced Reporting & Analytics:**

a. **Export Options:** Implement functionality to export existing reports (Collection, Dues, Ledger, etc.) to common formats like CSV, Excel, and PDF.

b. **Data Visualization:** Incorporate charts and graphs (e.g., using a JavaScript charting library like Chart.js or D3.js) on the Admin Dashboard and within specific reports to provide at-a-glance insights into fee collection trends, outstanding dues patterns over time, payment method popularity, etc.

c. **New Reports:** Develop more granular or specialized reports, such as Class-wise Collection Summaries, Payment Method Effectiveness Analysis, Concession/Discount Utilization reports.

6. **Systematic Completion of Input Validation & Security Hardening:**

   a. **Input Validation:** Conduct a thorough review and systematically apply detailed server-side input validation to *all* controller actions that process user input, beyond the basic checks currently in place for some modules. This includes stricter type checking, format validation, range checks, and validation against business rules.

   b. **Security Hardening:** Implement additional security measures such as rate limiting for login attempts (to prevent brute-force attacks), consider two-factor authentication (2FA) for administrative accounts, conduct regular security audits, and ensure all software dependencies are kept up-to-date to patch vulnerabilities.

7. **UI/UX Enhancements:**

   a. **External CSS:** Consolidate all CSS into well-organized external stylesheets (`.css` files) linked from the layout headers, removing inline `<style>` blocks from individual view files for better maintainability and caching.

   b. **AJAX for Dynamic Interactions:** Implement Asynchronous JavaScript and XML (AJAX) for certain interactions to enhance user experience, such as live filtering of tables without page reloads, or partial page updates when applying discounts.

   c. **Usability Testing:** Conduct formal or informal usability testing with representatives from each user group (Admin, Staff, Parent, Student) and refine workflows and interface elements based on feedback.

8. **Mobile Application (Student/Parent):**

   a. **Description:** Develop a native (iOS/Android) or hybrid mobile application for students and parents.

   b. **Functionality:** Provide access to fee information, payment history, receive push notifications for reminders and announcements, and potentially make online payments.

c. **Benefits:** Enhances convenience and accessibility for users.

9. Multi-Language Support:

a. **Description:** Integrate internationalization (i18n) and localization (l10n) libraries and practices.

b. **Functionality:** Allow the user interface text to be translated into multiple languages (e.g., Sinhala, Tamil, in addition to English, for a Sri Lankan context). Store language strings in separate resource files.

c. **Benefits:** Improves accessibility for a wider user base.

10. Integration with Other School Systems:

a. **Description:** Develop well-documented APIs (e.g., RESTful APIs) for the SFMS.

b. **Functionality:** Allow integration with existing Student Information Systems (SIS) for automatic student data synchronization, accounting software for seamless transfer of financial data, or Learning Management Systems (LMS).

c. **Benefits:** Reduces redundant data entry, improves data consistency across school systems, and creates a more unified administrative environment.

11. Fine-Grained Permissions within Staff Role:

a. **Description:** Currently, 'Staff' is a broad role. Introduce a more granular permission system (possibly by creating sub-roles or assigning specific permissions to individual staff users).

b. **Functionality:** Allow administrators to define exactly which fee management tasks specific staff members can perform (e.g., one staff member can only record payments, another can define fee structures but not process refunds).

c. **Benefits:** Enhances security and operational control by adhering to the principle of least privilege.

12. Bulk Actions:

a. **Description:** Implement features for performing actions on multiple records at once.

b. **Functionality:** Examples include bulk student import from CSV, bulk invoice cancellation (with a common reason), or applying a standard discount to a batch of selected students/invoices.

c. **Benefits:** Improves administrative efficiency for large-scale operations.

6.3 Challenges Faced

The development of the School Fees Management System, while ultimately successful in achieving its core objectives, presented several challenges that were addressed during the iterative process:

- **Iterative UI/UX Design & Refactoring:** The initial development focused on core functionality, with basic HTML views. The subsequent phase of integrating Bootstrap 5 for a consistent and modern UI across both Admin and Portal sections involved significant refactoring of all view files. Ensuring responsiveness and a cohesive user experience across diverse modules required careful planning and execution.

- **Complex Data Relationships & Logic:** Managing the interdependencies between numerous entities—students, users (as parents/guardians and system operators), roles, academic sessions, classes, fee categories, fee structures, discounts, invoices, invoice items, payments, payment allocations, and proofs—was a primary challenge. Implementing the many-to-many `student_guardian_links` relationship and ensuring it worked seamlessly with the portal's child selection mechanism, for instance, required careful database design and controller logic. Similar complexity arose in accurately recalculating invoice totals upon discount application/removal and during refund processing.

- **Session Management for User Feedback:** Consistently handling flash messages (for success, error, or informational feedback) and form error repopulation data (`form_errors`, `old_input`) across HTTP redirects required meticulous attention to the order of operations within controller actions (setting session variables before redirecting) and in views/layouts (reading and then unsetting session variables).

- **CSRF Protection Implementation:** Systematically applying CSRF token generation and validation to all state-changing POST forms across the entire application (Admin, Portal, Auth modules) was a detailed and time-consuming, albeit critical, task for security. Ensuring tokens were correctly embedded in forms and validated in every corresponding controller action required diligence.

- **Database Schema Evolution:** The database schema evolved iteratively as new features (e.g., discounts, refunds, detailed audit logs, system settings, `last_reminder_sent_at` on invoices) were conceptualized and implemented. Managing these `ALTER TABLE` statements while preserving existing data and ensuring backward compatibility with already developed code sections required careful planning and testing.

- **Debugging Complex Workflows & Silent Failures:** Tracing issues through multi-step processes, such as payment proof upload followed by admin verification and payment recording, or the email notification sending pipeline, sometimes involved debugging "silent failures" where no immediate error was visible on the screen. This necessitated thorough checking of PHP error logs, web server logs, and adding

temporary debug statements to pinpoint issues (e.g., email sending failures due to SMTP configuration or domain verification issues, database insert failures due to previously unnoticed column mismatches).

- **Floating-Point Arithmetic for Monetary Values:** Initial comparisons of monetary values (e.g., ensuring refund amount does not exceed paid amount) using standard float comparison operators led to precision issues. This was resolved by adopting more robust comparison methods like `bccomp` or using a small tolerance (epsilon) for equality checks.

- **Scope of `this` in Views and Layouts:** An early challenge involved the correct way to access controller methods or properties (like CSRF helper functions or role checks) from within view files and shared layout partials. This was resolved by ensuring that all necessary data and helper outputs are explicitly passed from the controller to the view, or by making helper methods static if appropriate, rather than relying on an assumed availability of the controller's `this` context within the view's scope when using `require`.

- **Configuration Management (Secure vs. UI-Editable):** Deciding which system parameters should be managed via the secure `.env` file (e.g., database passwords, email server credentials) versus those suitable for UI-based management by an administrator via the `system_settings` table (e.g., school name, bank details for display) required careful consideration of security implications versus operational flexibility.

Addressing these challenges iteratively contributed to a more robust, secure, and user-friendly final system.

6.4 Contribution of Each Project Member

- **Croos SJ (E2248507):**
    - **Design:** Led the initial database schema design and created the ER/EER diagram. Contributed to the Use Case definitions for administrative functions.
    - **Implementation:** Primarily developed the backend logic for the User Management module (CRUD, role assignment) and the initial Student Management module. Implemented the Fee Category and Fee Structure setup functionalities, including their CRUD operations in the admin panel. Contributed to the database interaction layer within these modules.
    - **Testing:** Conducted unit testing for the User and Student backend modules. Participated in integration testing of fee setup with invoice generation.
    - **Documentation:** Authored Chapter 3.2.5 (ER/EER Diagram description) and parts of Chapter 5.1 (System Interfaces - Admin User/Student Management) of this report.

- **Other:** Managed version control (Git) for the initial project setup.

- **Gowsihan V (E2248509):**
  - **Design:** Designed the activity diagrams for payment processing and refund workflows. Contributed to the sequence diagram for invoice generation.
  - **Implementation:** Focused on the core financial transaction logic, including the backend for Payment Recording (manual entry and proof verification workflow) and the complete Refund Processing module. Developed the Invoice Generation mechanism in `InvoiceGenerationController`. Implemented the Discount Application system, including Discount Type CRUD and the logic in `InvoiceDiscountController` for applying/removing discounts and recalculating invoice totals.
  - **Testing:** Led the testing efforts for all financial transaction modules (payments, refunds, discounts). Performed integration testing between invoicing and payments.
  - **Documentation:** Authored detailed descriptions for Activity Diagrams (Payment, Refund) in Chapter 3.2.2 and parts of Chapter 5.1 (System Interfaces - Invoice and Payment Management). Contributed to Chapter 6.3 (Challenges Faced - Financial Logic).
  - **Other:** Researched and proposed solutions for handling floating-point precision issues in financial calculations.

- **Thanusan P (E2245667):**
  - **Design:** Led the overall UI/UX design strategy, including the selection of Bootstrap 5. Designed the structure for the Admin and Portal layouts. Created sequence diagrams for user authentication and portal interactions.
  - **Implementation:** Developed the entire Student/Parent Portal (Dashboard, Fee Viewing, Invoice Details, Profile Management, Offline Payment Proof Upload), including the multi-child selection logic for parents. Implemented the `NotificationService`, including email template management from the database and PHPMailer integration for sending emails. Developed the manual fee reminder trigger system with cooldown logic. Was responsible for implementing security features such as CSRF protection across all forms and ensuring consistent session management. Developed the System Settings module (`SettingsController`, `SettingsHelper`, and view). Led the Bootstrap refactoring for all Admin and Portal views.
  - **Testing:** Led UI/UX testing and responsiveness checks across different devices. Tested security features like CSRF and role-based access.

- **Documentation:** Authored Chapters 1 (Introduction), 2 (Related Research - conceptual framework), 4 (Technologies Adopted), 5.1 (System Interfaces - Portal), and compiled/edited the final report (Chapters 6.1, 6.2, 6.3).
- **Other:** Managed the project's Composer dependencies and `.env` configuration.

References

[1] Vidyalaya, “School Fees Management System, School Fees Management Software,” *www.vidyalayaschoolsoftware.com*, 2025. https://www.vidyalayaschoolsoftware.com/features/school-fees-management-system

[2] PHPMailer, "PHPMailer – A full-featured email creation and transfer class for PHP," GitHub. [Online]. Available: https://github.com/PHPMailer/PHPMailer. [Accessed: May 07, 2025].

[3] The Bootstrap Authors, "Bootstrap v5.3 Documentation," Bootstrap. [Online]. Available: https://getbootstrap.com/docs/5.3/. [Accessed: May 07, 2025].

[4] The Bootstrap Authors, "Bootstrap Icons," Bootstrap. [Online]. Available: https://icons.getbootstrap.com/. [Accessed: May 07, 2025].

[5] N. Popov, "FastRoute - Fast request router for PHP," GitHub. [Online]. Available: https://github.com/nikic/FastRoute. [Accessed: May 07, 2025].

[6] V. Lucas, "PHP DotEnv," GitHub. [Online]. Available: https://github.com/vlucas/phpdotenv. [Accessed: May 07, 2025].

[7] "Select2 - The jQuery replacement for select boxes," Select2. [Online]. Available: https://select2.org/. [Accessed: May 07, 2025].

[8] "PHP Manual," The PHP Group. [Online]. Available: https://www.php.net/manual/en/. [Accessed: May 07, 2025].

[9] "MySQL Documentation," Oracle Corporation. [Online]. Available: https://dev.mysql.com/doc/. [Accessed: May 07, 2025].