FINAL GLOBAL SYSTEM

A Comprehensive Platform Integration Solution

Project Report

AYUB MUSSA

August 15, 2025

Abstract

This report presents a comprehensive analysis of the Final Global System, a web-based platform integration solution designed to provide unified access to multiple educational platforms. The system implements a sophisticated architecture combining frontend technologies (HTML5, CSS3, JavaScript) with backend services (PHP, MySQL) to create a seamless user experience. The report covers the system overview, technical methodology, data structure design, and demonstrating the system's capability to handle complex authentication, role-based access control, and content management requirements in an educational environment.

Contents

1 Overview

1.1 System Purpose

The Final Global System is a comprehensive web-based platform integration solution designed to serve as a central hub for university students and staff. The system provides unified access to multiple educational platforms including Learning Management Systems (LMS), Student Information Systems (SIS), Leave Management Systems, and other institutional platforms through a single, authenticated interface.

1.2 Key Objectives

Unified Access: Provide single sign-on access to multiple educational platforms

Content Management: Centralized management of announcements and institutional information

User Management: Comprehensive user and administrative account management

Security: Implement robust authentication and authorization mechanisms

Scalability: Design for future expansion and additional platform integration

1.3 Target Users

Students	Primary users accessing educational platforms and viewing institutional announcements
Faculty	Academic staff requiring access to teaching and administrative platforms
Administrators	System managers responsible for content creation and user management
Super Administrators	System administrators with full control over all system features

2 Introduction

2.1 Background

Modern educational institutions operate multiple digital platforms to support various aspects of academic and administrative operations. Students and staff often need to access Learning Management Systems, Student Information Systems, Leave Management Systems, and other institutional platforms. Managing multiple login credentials and navigating between different systems creates significant user experience challenges and administrative overhead.

2.2 Problem Statement

Multiple login credentials for users

Inconsistent user interfaces across platforms

Difficulty in sharing information between systems

Administrative overhead in managing multiple user databases

Security risks associated with multiple authentication points

2.3 Solution Approach

Unified Authentication: Single login system for all platforms

Proxy Integration: Seamless navigation between platforms while maintaining session state

Content Aggregation: Centralized management of announcements and institutional information

Role-Based Access Control: Granular permissions based on user roles

Multi-language Support: Internationalization for diverse user base

3 Methodology

3.1 Development Approach

The project follows an iterative development methodology with the following phases:

3.1.1 Phase 1: Requirements Analysis

Requirements gathering

Platform integration requirements analysis

Security and authentication requirements definition

User interface and experience requirements specification

3.1.2 Phase 2: System Design

Database schema design

API architecture planning

User interface wireframing

Security architecture design

3.1.3 Phase 3: Implementation

Backend API development

Frontend interface implementation

Database implementation and optimization

Integration testing

3.1.4 Phase 4: Testing

Unit and integration testing

User acceptance testing

Performance optimization

3.2 Technology Stack

3.2.1 Frontend Technologies

HTML5	Semantic markup for structure and accessibility
CSS3	Responsive design and modern styling with Flexbox and Grid
JavaScript ES6+	Client-side functionality and asynchronous operations
Fetch API	HTTP communication with backend services
Local Storage	Client-side data persistence

3.2.2 Backend Technologies

PHP 7.4+	Server-side processing and API endpoints
MySQL 8.0	Relational database management
cURL	HTTP requests for platform integration
Session Management	User session handling and security

3.2.3 Development Tools

Git	Version control and collaboration
XAMPP	Local development environment
VS Code	Integrated development environment
Browser DevTools	Frontend debugging and testing

3.3 Architecture Design

3.3.1 System Architecture

The system follows a three-tier architecture:

1. **Presentation Tier**: HTML/CSS/JavaScript frontend
2. **Application Tier**: PHP backend with RESTful APIs
3. **Data Tier**: MySQL database for persistent storage

3.3.2 API Design

The system implements a RESTful API architecture with:

Standard HTTP methods (GET, POST, PUT, DELETE)

JSON data format for requests and responses

Consistent error handling and status codes

Endpoint-based routing system

4 Data Structure

4.1 Database Schema

4.1.1 Core Tables

Users Table

```sql
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    role ENUM('user', 'admin', 'super_admin') DEFAULT 'user',
    status ENUM('active', 'inactive') DEFAULT 'active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
        CURRENT_TIMESTAMP
);
```

Admins Table

```
CREATE TABLE admins (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    role ENUM('admin', 'super_admin') DEFAULT 'admin',
    status ENUM('active', 'inactive') DEFAULT 'active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
       CURRENT_TIMESTAMP
);
```

Announcements Table

```
CREATE TABLE announcements (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    content TEXT NOT NULL,
    author_id INT NOT NULL,
    status ENUM('active', 'inactive') DEFAULT 'active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
       CURRENT_TIMESTAMP,
    FOREIGN KEY (author_id) REFERENCES admins(id) ON DELETE CASCADE
);
```

Dining Menu Table

```
CREATE TABLE dining_menu (
    id INT PRIMARY KEY AUTO_INCREMENT,
    date DATE NOT NULL,
    breakfast_menu TEXT,
    breakfast_start_time TIME,
    breakfast_end_time TIME,
    lunch_menu TEXT,
    lunch_start_time TIME,
    lunch_end_time TIME,
    created_by INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
       CURRENT_TIMESTAMP,
    FOREIGN KEY (created_by) REFERENCES admins(id) ON DELETE CASCADE
);
```

Platforms Table

```
CREATE TABLE platforms (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    url VARCHAR(255) NOT NULL,
    description TEXT,
    category VARCHAR(50),
    status ENUM('active', 'inactive') DEFAULT 'active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Platform Credentials Table

```
CREATE TABLE platform_credentials (
    id INT PRIMARY KEY AUTO_INCREMENT,
    platform_id INT NOT NULL,
    username VARCHAR(100) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (platform_id) REFERENCES platforms(id) ON DELETE
       CASCADE
);
```

4.2 Data Relationships

4.2.1 Entity Relationship Diagram

The database design follows a normalized structure with the following key relationships:

1. **One-to-Many**: Admin to Announcements (one admin can create multiple announcements)
2. **One-to-Many**: Admin to Dining Menu (one admin can create multiple dining menus)
3. **One-to-Many**: Platform to Platform Credentials (one platform can have multiple credential sets)
4. **Many-to-One**: Announcements to Admin (many announcements belong to one admin)
5. **Many-to-One**: Dining Menu to Admin (many dining menus belong to one admin)

4.3 Data Access Patterns

4.3.1 Authentication Flow

1. User submits credentials via login form
2. System validates against users table
3. If valid, creates session and redirects to appropriate dashboard
4. If admin credentials, also validates against admins table

4.3.2 Content Retrieval

1. System queries announcements table for active announcements
2. System queries dining_menu table for today's menu

3. Content is formatted and displayed to users

4.3.3 Platform Integration

1. System retrieves platform credentials from platform_credentials table
2. Establishes proxy connection to external platform
3. Maintains session state across platform boundaries

5 Conclusion

The Final Global System project represents a comprehensive solution to the challenges of managing multiple educational platforms in a unified environment. Through careful analysis of requirements, implementation of robust architecture, and adherence to best practices in software development, the system successfully addresses the identified problems while providing a scalable foundation for future enhancements.

5.1 Key Achievements

Unified Authentication: Single sign-on across all integrated platforms

Role-Based Access Control: Granular permissions for different user types

Content Management: Centralized announcement and dining menu management

Multi-language Support: Internationalization for diverse user base

Security Implementation: Robust authentication and authorization mechanisms

Scalable Architecture: Modular design supporting future expansion

A API Endpoints Reference

A.1 User API Endpoints(api.php?endpoint=)

Method	Endpoint	Description	Parameters
POST	`/login`	User authentication	`username`, `password`
GET	`/platforms`	Get available platforms	None

(Continued on next page)

GET	`/announcements`	Get active announcements	None
GET	`/dining-menu-today`	Get today's dining menu	None
POST	`/proxy`	Platform proxy access	`platform_id`, `url`

A.2 Admin API Endpoints(admin_api.php?endpoint=)

Method	Endpoint	Description	Parameters
POST	`/admin-login`	Admin authentication	`username`, `password`
GET	`dashboard-stats`	Get dashboard statistics	None
POST	`/announcement-create`	Create announcement	`title`, `content`
PUT	`/announcement-update`	Update announcement	`id`, `title`, `content`
DELETE	`/announcement-delete`	Delete announcement	`id`
POST	`/dining-menu-create`	Create dining menu	`date`, `breakfast_menu`, `lunch_menu`
PUT	`/dining-menu-update`	Update dining menu	`id`, `menu_data`
DELETE	`/dining-menu-delete`	Delete dining menu	`id`
POST	`/user-promote`	Promote user to admin	`user_id`
POST	`/admin-create`	Create admin account	`username`, `email`, `password`, `role`
PUT	`/admin-update`	Update admin account	`id`, `username`, `email`, `role`
DELETE	`/admin-delete`	Delete admin account	`id`

B Database Schema Details

B.1 Indexes and Constraints

1. **Primary Keys**: All tables have auto-incrementing primary keys
2. **Foreign Keys**: Proper foreign key constraints for referential integrity
3. **Unique Constraints**: Username and email fields have unique constraints

4. **Indexes**: Indexes on frequently queried fields (username, email, date)

B.2 Data Types and Sizes

1. **VARCHAR**: Used for variable-length strings with appropriate size limits
2. **TEXT**: Used for longer content fields (announcements, menus)
3. **ENUM**: Used for status and role fields to ensure data integrity
4. **TIMESTAMP**: Used for created_at and updated_at fields
5. **DATE/TIME**: Used for date-specific fields (dining menu dates, meal times)

C Installation Guide

C.1 Prerequisites

1. PHP 7.4 or higher
2. MySQL 8.0 or higher
3. cURL extension enabled
4. Session support enabled

C.2 Installation Steps

1. Clone or download the project files
2. Configure database connection in `database/db_connection.php`
3. Create database and run the setup script
4. Configure web server to point to the project directory
5. Set proper file permissions
6. Create initial admin account
7. Test the installation

D Configuration

D.1 Environment Variables

The system uses the following configuration parameters:

1. Database connection settings
2. Platform URLs and endpoints
3. Admin account credentials
4. Logging configuration
5. Session timeout settings

D.2 Platform Configuration

Each integrated platform requires:

1. Platform URL and authentication endpoints
2. Credential management
3. Session handling configuration
4. Error handling and fallback mechanisms