**A close-up of a logo

Description automatically generated**

**Software Final Document for Lau Event Management System (LEMS)**

**Imad Mahmoud**

**ID:202301483**

**Ali Yassine**

**ID:202305136**

**Adam Shehadeh**

**ID:202302440**

Table of Contents

[Introduction 2](#_Toc197896480)

[Purpose: 2](#_Toc197896481)

[High Level Features: 2](#_Toc197896482)

[System Implementation: 3](#_Toc197896483)

[Overview: 3](#_Toc197896484)

[Technology Stack 3](#_Toc197896485)

[Architecture: 4](#_Toc197896486)

[Modular Breakdown 4](#_Toc197896487)

[Coding Standards and Best Practices 6](#_Toc197896488)

[Code Quality 6](#_Toc197896489)

[Security Considerations 6](#_Toc197896490)

[Error Handling 6](#_Toc197896491)

[Testing 7](#_Toc197896492)

[Testing Approach 7](#_Toc197896493)

[Unit Testing 7](#_Toc197896494)

[Integration Testing 9](#_Toc197896495)

[System Testing 10](#_Toc197896496)

[User Acceptance Testing (UAT) 11](#_Toc197896497)

[Challenges and Improvements 11](#_Toc197896498)

[Improvements if time were available: 12](#_Toc197896499)

# Introduction

## Purpose:

LEMS is a web application made to consolidate all events at LAU. It facilitates the discovery, registration, and attendance of events by students. Organizers and admins are also given the appropriate capabilities to manage, track, and oversee their events. Support for comment summarization and event recommendation, along with AI-powered tools, further enhances the user experience, while live streaming features allow for real-time viewing of events.

## High Level Features:

* AI-Generated Summarization of Feedback for fast feedback.
* Secure authentication via SSOutlook Single Sign-On (SSO).
* Real-time live streaming of events set up using NGINX with RTMP and HLS streaming protocols.
* Workflow for creation, management, and approval of events within set permissions per designated role.
* User-friendly interface, following the Glassmorphism and Skeuomorphism design trends.
* Collection and Tracking of Feedback for improving subsequent events.
* Notification system for events, reminders, and critical announcement emergency broadcasts.
* RBAC to secure confidential user information and event details.

# System Implementation:

## Overview:

An overview of the LEMS system with its primary components includes:

- Authentication & Authorization (auth/, login.php, oauth-callback.php, generate-auth-link.php): manages user registration/log in, maintains files, and oversees OAuth2 processes interfacing with Gmail.

- Domain Model & Data Access (classes.php, db\_connection.php): create PHP classes (for instance; EventManager, User, Review) and also make a secure database connection (PDO/mysqli).

- Event CRUD & Business Logic (CreateEvent.php + CreateEventLogic.php, EditEvent.php, DeleteEvent.php, UpdateEventState.php, cancel\_event\_logic.php): Each document takes charge of either creating, reading, updating, or deleting an event.

- User Interface (home.php, browse.php, recommended.php, event.php, event\_details.php, organizer\_dashboard.php, AdminDashboard.php): static parts of webpages are HTML/CSS and dynamic parts are data.

- File Uploads & Media (FileUpload.php, uploads/): manages files submitted by users, controls MIME type and size yolk, and places files into a specific subdirectory.

- Livestreaming (livestream.html, HLS.js): static HTML page Features a streamable <video> element and uses HLS.js to serve streams from NGINX-RTMP/HLS.

- Reviews & Feedback (submit\_review.php, ViewReviews.php, ViewFeedback.php): enable event rating by users as well as aggregated feedback viewing.

- Notifications (SendNotifications.php): used stored Gmail OAuth tokens.

## Technology Stack

- Backend: PHP 8.x, My SQL (approached through PDO or mysqli), Composer (dependency manager)

-Frontend: HTML5, CSS3 (on per-page stylesheets:home.css, browse.css, …), JavaScript (plain + script.js), HLS.js (through CDN)

- Server & Deployment: LEMP/LAMP stack with NGINX or Apache, NGINX RTMP module for livestreaming, Vercel static site preview (lems-full.vercel.app)

- Auth & API: Google OAuth2 PHP client, gmail-refresh-token.json for refresh tokens

- Version Control & Build: Git, GitHub repository, Composer for PHP packages

## Architecture:

The original design (Assignment 2) followed layered MVC-lite pattern with:

1. The data layer comprises a relational schema (ERD) where users events clubs reviews feedback notifications and uploaded files are stored in tables.

2. The logic layer is composed of business rules and validation within PHP classes and scripts.

3. The presentation layer consists of PHP-generated HTML views with per-page CSS and shared JavaScript.

In the implementation:

1. Each Controller was implemented to correspond one-to-one with the CRUD operations described in the design document.

2. Other than a few minor schema modifications: adding a club\_id column in events (for multi-club event support) and adding file\_path field in uploads, the ERD was largely maintained.

3. The livestream functionality was separated into its own HTML static page (livestream.html) and merged HLS.js on the client side.

4. The class diagram was also changed, and the modifications can be found in the source code in classes.php.

## Modular Breakdown

1.Database Design

-Table relations decrease redundancy and aids data integrity.

-Foreign keys reinforce referential integrity constraints.

-Event\_id and user\_id indexes accelerate lookups.

2.Front‑end

-Static PHP templates list events and clubs while rendering user profiles injects user filters.

-Styles & Layout: Per page CSS; navbar and footer as components extracted to includes for responsive adapatation.

-Live streaming: <video id="liveStream"> is initialized in livestream.html by HLS.js pulling an HLS manifest from the NGINX‑RTMP server.

3.Back‑end

•Business Logic Modules:

-Event Logic: CreateEventLogic.php validates inputs and checks for conflicts, it then calls EventManager->create().

-Club Management: ManagerClub.php manages club creation, deletion and their association with events.

-Review & Feedback: Reviews are submitted via submit\_review.php and aggregate ratings through ViewReviews.php

•APIs & Utilities:

- SendNotifications.php uses Gmail API wrappers to send requests utilizing gmail-refresh-token.json and Google’s PHP SDK.

-Db\_connection.php handles user credentials settings with PDO/mysqli, caches failed error connections and provides DB instance as singleton.

•Interaction Flow:

-Logging into the system as a user → A session is created.

-A user submits the form → Controller has script validation & calls Model.

-Model updates DB → returns status.

-Controller sets flash message and error logs, then redirect to view.

All modules interact through well-defined interfaces which can be function calls or class methods that maintain the separation of concerns in the system and enhance the maintainability of the codebase.

# Coding Standards and Best Practices

## Code Quality

-Adopted the PSR-12 PHP file structure, indentation, and brace placement standards within coding windows, streams of paralell work.

-Implemented EventManager as class name in classes.php and used CamelCase for class names zapernith → and generalized to Everything Zapper Case variable as renting. Therefore, all modules will be consistent.

-Modified naming convention for CRUD to Create-Update-Activate-View so each module will contain relevant logic. Each CRUD action resides in unique files, a singular responsibility per file a unique action thus there will not be overlap.

-Developed logic for claimed complete circuits to enhance public method documentation, public class Interface circuit claim methods properties undergo docblock PHPDoc so evident logic can emit noise to be traversed.

## Security Considerations

-Every interaction with the database uses a parameterized query (for example, PDO or mysqli prepared statements) that avoids SQL injection, such as in the logic classes under CreateEventLogic.php and DeleteEvent.php.

-I developed login-based authentication integrated with OAuth2 in login.php / oauth-callback.php, with secure token storage (gmail-refresh-token.json) and employed session\_regenerate\_id() on login to prevent session fixation.

- All user inputs were validated and sanitized using filter\_input() cleanup methods along with regular expressions before program logic or database insert steps. Additionally, files uploaded to ‘uploads/’ directories are validated against set MIME types and maximum file size.

## Error Handling

-Ensured that critical operations (for example, database connection in db\_connection.php) were wrapped with try–catch blocks to prevent uncaught fatal errors caused by thrown exceptions. Analyze performance impacts.

-Errors presented to users are surfaced as user-friendly messages (for example, “Unable to load events, please try again later”) while system logs provide full stack traces and error information for post-debugging analysis.

-Implemented a centralized error-logging system (for example in SendNotifications.php) that captures and timestamps errors, simplifying the process of tracking issues and performing post-analysis.

# Testing

## Testing Approach

The testing strategy for Lau Event Management System (LEMS) emphasized verifying the system’s features and functions relative to the interface (UX), along with other non-functional parameters like the response time of the system and security. The following testing methodologies were employed:

* Unit testing: Concentrated on a few selected classes and methods in the files of PHP to be as specific as possible like event generation, user role assignment, and feedback submission in classes.php.
* Integration testing: Checked for proper integration of frontend forms with corresponding backend processes, like CreateEvent.php with CreateEventLogic.php.
* System testing:Performed end-to-end testing on core user journeys, for example logging into the system via Outlook SSO, AIl generating recommendations, and event streaming.
* User Acceptance Testing (UAT): Carried out the process to validate whether the system meets the requirements from a usability perspective as per designed workflows and smooth interactive transactions.

## Unit Testing

* Approach

Each function and component were validated using unit testing. Special attention was directed towards critical backend classes like User, Event, and Feedback, data authentication in AuthService, as well as business feedback mechanisms. Business and data validation, including edge case processes, were verified for testing purposes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Description | Expected Result | Actual Result | Status |
| UTC-01 | Create Event with Valid Data | Event created successfully | Event created successfully | ✅ Pass |
| UTC-02 | Create Event with Missing Fields | Validation error returned | No validation triggered | ❌ Fail |
| UTC-03 | User Role Verification (Admin) | Admin access granted | Admin access granted | ✅ Pass |
| UTC-04 | User Role Verification (Student Access Admin Function) | Access Denied Error | Access granted (security flaw) | ❌ Fail |
| UTC-05 | Feedback Submission (Valid Data) | Feedback saved successfully | Feedback saved successfully | ✅ Pass |
| UTC-06 | Feedback Submission (Empty Data) | Validation error message | Validation error shown | ✅ Pass |
| UTC-07 | AI Recommendation API Integration | Recommendations returned | Recommendations returned | ✅ Pass |
| UTC-08 | AI Recommendation with Invalid API Key | API Error Handled Gracefully | Unhandled exception occurred | ❌ Fail |
| UTC-09 | Logout Functionality | User session terminated | User session terminated | ✅ Pass |
| UTC-10 | Event Capacity Limit | Registration denied after capacity reached | Registration still allowed | ❌ Fail |
| UTC-11 | SSO Login with Valid Token | User successfully logged in | User successfully logged in | ✅ Pass |
| UTC-12 | SSO Login with Invalid Token | Authentication failed gracefully | Unhandled error on redirect | ❌ Fail |
| UTC-13 | Password Reset (Valid Email) | Reset link sent to email | Reset link sent successfully | ✅ Pass |
| UTC-14 | Password Reset (Non-Registered Email) | Show user not found message | Error displayed properly | ✅ Pass |
| UTC-15 | Notification Sending to Attendees | Notification delivered | Notification delivered | ✅ Pass |

## Integration Testing

* Approach

Integration testing was done to validate the cooperation of the single modules. This involves the interaction of the frontend and backend with the database, and the integration of external applications like OpenAI API and Outlook SSO.

* + - * Integration Points Tested
      * Submission review from frontend forms to the backend controllers CreateEventLogic.php is working as intended.
      * The capability of the database through db\_connection.php to perform CRUD operations.
      * Recommended.php can fetch along with AI OpenAI API Recommendation Module.
      * Authentication works through Outlook SSO in generate-auth-link.php and oauth-callback.php.
      * Embedding live streaming from NGINX HLS streams in livestream.html.

Example Integration Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Integration Point | Input | Expected Output | Result |
| ITC-01 | Create Event (Frontend ↔ Backend) | Valid Event Form | Event stored in DB | ✅ Passed |
| ITC-02 | Submit Feedback | Feedback Data | Stored and AI summary generated | ✅ Passed |
| ITC-03 | AI Recommendation API | Transcript Upload | Recommendations displayed | ✅ Passed |
| ITC-04 | Outlook SSO Authentication | Valid Token | User authenticated and redirected | ✅ Passed |
| ITC-05 | Live Streaming Integration | Stream URL Access | Video player shows live stream | ✅ Passed |

## System Testing

* Approach

System testing aimed at verifying full functionality of the lems system by its actual users. All system components were integrated and tested to ascertain if all processes were completed successfully.

* Test Scenarios
  + Scenario 1: A Student Event Registration

Steps to reproduce:

Use SSO Outlook to Log in.

Go to ViewEvents.php.

Choose an event and click on Register.

Expected Outcomes: Message indicating successful registration along with events user posted to.

Actual Outcome: Achieved.

* + Scenario 2: Creating An Event By Event Organizer

Steps to Reproduce:

Organizing user login.

Access CreateEvent.php.

Event particulars are being completed and submitted.

Expected Outcome: Created and shown in the list of created events.

Actual outcome: Achieved.

* + Scenario 3: AI Personalized Recommendations from Uploaded Transcripts

Steps to Reproduce:

Log in as students.

Go to Recommended.php.

Upload a copy of a transcript and request for event recommendations.

Expected Outcomes: defined personalized Towards Event Recommendations outcome for per users provided

Actual outcome: Achieved.

* + Scenario 4: User Admin Cancels Event

Steps to Reproduce:

Login as Admin.

Go to AdminDashboard.php.

Choose an Event, Click ‘Cancel Event’.

Expected Outcomes: status of the event will change to “canceled” and messages sent.

Actual outcome: Achieved.

## User Acceptance Testing (UAT)

* Approach

To evaluate the system's usability and feature accessibility, a brief UAT session was held with actual LAU users. Practical changes were made in response to their input.

* Participants

2 Students

1 Organizer

Feedback Summary

|  |  |
| --- | --- |
| Feedback | Action Taken |
| Students struggled to find their own registered events | Added a “Manage Events” For easy access. |
| Organizers found it difficult to manage events. | Implemented a dedicated dashboard for the Organizers to do whatever they please. |

# Challenges and Improvements

Integrating the Outlook SSO system was one of the major challenges to overcome during LEMS development. This task demanded proper understanding of the OAuth 2.0 protocol and meticulous token validation due processes. The system as is, did not cater for adequate error handling for expired and invalid tokens, which burnt login attempts. This was bypassed by implementing full token validation and error handling in the authentication flow of the system and redirecting incorrectly routed expired and invalid tokens back to the authentication flow.

A further equally important challenge was incorporating the AI Recommendation API with OpenAI. The team had issues with balancing the rate limits of the API and error handling for the unexpected turn of events with the API. In addition to this, slow response times further worsened user experience. To mitigate these issues, API requests were streamlined from prior, add in additional logic that eliminated unnecessary response calls, and added loading indicator to lessen user control waiting during the delay.

From the security angle, Role-Based Access Control (RBAC) had security gaps where users were allowed to perform certain sensitive actions without role validation. This was fixed by implementing strict checks of the user role prior to executing any sensitive operations. The user role is validated at controller level which mitigates this gap.

# **Improvements if time were available:**

Paid Events Integration with Payment Processing:

-Integrate secure online payment processing systems to facilitate ticket sales and registrations within the platform for paid events

AI-Powered Chatbot Support:

-Employ OpenAI APIs to create a chatbot that will answer frequently asked questions, help users search for events, and guide them around the site.

Event Recommendation Algorithm Improvement:

-Augment the system’s recommendation accuracy through transcript analysis merged with collaborative filtering, which improves predictions based on user behavior and preferences.

-UI/UX Design Improvement:

Adopt new trends in user interfaces to improve platform aesthetics, adding animation and responsiveness to further enhance the site.

# Requirements Traceability Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement ID | Requirement Description | Implemented Feature (Code Reference) | Test Case IDs |
| REQ-001 | Event creation/edit/delete | CreateEventLogic.php, UpdateEventState.php, DeleteEvent.php | UTC-01, UTC-02 |
| REQ-002 | Review submission | submit\_review.php, Feedback class | UTC-05, UTC-06 |
| REQ-003 | AI-based recommendations | Recommended.php, Transcript class | UTC-07, UTC-08 |
| REQ-004 | AI-generated review summaries | ViewReviews.php | UTC-07, UTC-08 |
| REQ-005 | Registration, waitlist, notifications | Registration class, SendNotifications.php | UTC-10, UTC-15 |
| REQ-006 | Live streaming integration | livestream.html | Manual Validation |
| REQ-007 | Real-time attendance tracking | Event class (Capacity Logic) | UTC-10 |
| REQ-008 | Admin event approval | AdminDashboard.php, RBAC | UTC-03, UTC-04 |
| REQ-009 | Role-based access control | User class with RBAC | UTC-03, UTC-04 |
| REQ-010 | Advanced search & filtering | ViewEvents.php | Manual Validation |
| REQ-011 | Emergency notifications | SendNotifications.php | UTC-15 |
| REQ-012 | Event analytics dashboard | AdminDashboard.php | Manual Validation |
| REQ-013 | Report generation | ViewReviews.php | Manual Validation |
| REQ-014 | Intuitive UX | Bootstrap UI Improvements | UAT Feedback |
| REQ-015 | Performance (Load Time) | Caching, Query Optimization | Manual Validation |
| REQ-016 | Scalability | Modular Client-Server Architecture | - |
| REQ-017 | Security (AES-256, RBAC) | AES-256 Encryption, SSO, Role Checks | UTC-11, UTC-12 |
| REQ-018 | Usability | Added "Manage My Events" | UAT Feedback |
| REQ-019 | Reliability | NGINX Streaming with Error Handling | Manual Validation |
| REQ-020 | Maintainability | MVC Pattern, Modular Code | - |