**Documentation and Reporting: MiseriorPulse Integration Process**

This document outlines the integration process of the MiseriorPulse system, detailing the methodologies employed, system configurations, and the challenges encountered, along with their resolutions.

**System Configurations**

**1. Development Environment:**

* **Unity Version:** 2022.3 LTS
* **Programming Language:** C#
* **Backend:** PHP (Laravel framework)
* **Database:** MySQL
* **Server Configuration:**
  + **Operating System:** Ubuntu 22.04 LTS
  + **Web Server:** Apache 2.4
  + **PHP Version:** 8.1
  + **MySQL Version:** 8.0

**2. Integration Setup:**

* **API Communication:** RESTful API using JSON for data exchange between the Unity client and PHP backend.
* **Encryption:** AES-256 for sensitive data and salted SHA-256 for passwords.
* **Middleware:** JWT authentication for secure session handling.

**Integration Methodologies**

**1. API Development and Connection:**

* Developed RESTful APIs in PHP to handle attendance data, fetching student details, and updating the database.
* Unity application connected to the API using HTTP requests with JSON payloads for seamless communication.

**2. Error Handling in PHP Backend:**

* **Problem:** Misconfigured PHP error reporting caused blank responses during testing.
* **Solution:** Adjusted php.ini settings to enable error logging and display during development.

**3. Secure Data Exchange:**

* Integrated AES-256 encryption to protect sensitive data during transmission.
* Implemented HTTPS to secure API endpoints.

**4. Unity Integration Issues:**

* **Problem:** WWW and UnityWebRequest calls in Unity were not handling JSON responses properly.
* **Solution:** Used UnityWebRequest.Post and UnityWebRequest.Get correctly with appropriate headers (Content-Type: application/json).
* Added error handling for network timeouts and API failures.

**5. MySQL Database Optimization:**

* **Problem:** Query performance issues with large datasets during attendance retrieval.
* **Solution:**
  + Added indexing to frequently queried fields.
  + Optimized SQL queries using JOIN and LIMIT clauses.
  + Conducted stress tests to ensure scalability.

**Challenges and Solutions**

1. **Unity and PHP Data Communication**
   * **Challenge:** JSON parsing errors in Unity due to mismatched data types.
   * **Solution:** Standardized data formats on both ends and tested payloads using Postman.
2. **Database Injection Risks**
   * **Challenge:** Susceptibility to SQL injection in early API testing.
   * **Solution:** Used prepared statements in PHP (PDO) to sanitize inputs and prevent malicious queries.
3. **Authentication Issues**
   * **Challenge:** JWT tokens were not properly validated on the server.
   * **Solution:** Debugged token validation logic and ensured secure key storage.
4. **Cross-Origin Resource Sharing (CORS) Errors**
   * **Challenge:** API requests from Unity were blocked due to CORS policy.
   * **Solution:** Configured the Apache server to allow requests from the Unity client’s domain using appropriate headers.

**Reporting and Recommendations**

* **Integration Success:** The MiseriorPulse system now handles secure data transmission, efficient database interaction, and real-time updates.
* **Documentation:** Comprehensive documentation ensures the process is replicable, covering configuration files, API routes, Unity scripts, and database schemas.
* **Future Recommendations:**
  + Automate API testing using tools like Postman and Newman.
  + Implement a CI/CD pipeline for smoother deployments.
  + Continuously monitor system performance using analytics tools.