Project Report: *Employee Leave Management System with Data Reporting and Integration Introduction*

1.1 Project Background

Leave management is essential for organizational efficiency, especially in environments where employees must balance multiple responsibilities alongside personal commitments. Manual leave systems often lead to errors, delayed approvals, and limited reporting capabilities, which can hinder oversight and compliance. An automated leave management system helps resolve these challenges by enabling digital request submissions, streamlined approvals, and data-driven reporting. This enhances transparency, reduces administrative workload, and supports overall employee well-being. This project delivers such a solution by integrating sample HR data and offering comprehensive reporting to support informed decision-making.

1.2 Objectives

- The project aimed to develop a web-based leave management system to:
- Allow employees to submit leave requests and managers to approve or reject them with comments. Integrate employee data from a mock HR system via CSV imports, ensuring data validation and error reporting.
- Generate monthly leave summary reports with pending, approved, rejected counts, and total leave days, filterable by year.
- Implement role-based access control (RBAC) to restrict features by user role (employee or manager). Provide database-driven notifications for request submissions, approvals, and rejections.
- Ensure a secure, scalable 3-tier architecture (front-end, logic, database) with robust security measures.

1.3 Relevance

This project aligns with Information Management by leveraging MariaDB 10.4.32 for efficient data storage, retrieval, and reporting, ensuring data integrity through foreign key constraints and validation. It embodies System Integration and Architecture via CSV-based HR data integration (hr_import.php), a 3-tier architecture, and workflow automation (notifications, approvals). The system demonstrates enterprise-level concepts like RBAC (AuthMiddleware.php), secure session management (Session.php), and audit logging (audit_logs table), directly applying course principles to a practical, academic-focused solution.

2. Literature Review

Leave management systems, such as those in BambooHR, Workday, and Zoho People, automate HR tasks like leave tracking, approvals, and analytics, reducing errors and enhancing policy compliance. However, these systems are often feature-heavy, posing complexity for smaller organizations like universities. This project prioritizes simplicity, focusing on core functionalities (leave requests, approvals, reporting) tailored for academic needs.

Data integration challenges include handling inconsistent formats, ensuring uniqueness (e.g., emails in employees), and managing errors. The project's CSV import (hr_import.php) mitigates these by validating headers, emails, and roles, logging results for transparency, similar to ETL (Extract, Transform, Load) processes in enterprise systems.

Workflow automation enhances efficiency by streamlining repetitive tasks. The system automates leave approvals (manage_requests.php) and notifications (notifications table), reducing manual intervention, akin to BPM (Business Process Management) frameworks.

Enterprise application architecture often employs a 3-tier model to separate concerns, improving scalability and maintenance. This project's architecture—HTML/CSS/JavaScript front-end, PHP 8.2.12 logic, and MariaDB database—mirrors simplified enterprise systems, with middleware (AuthMiddleware.php) enforcing security and modularity.

3. System Requirements

3.1 Functional Requirements

Leave Request Submission: Employees submit requests with leave type, dates, and reasons, validated for conflicts and balances.

Manager Approval/Rejection: Managers review requests, approve or reject with comments, and update statuses.

HR Data Integration: Import employee data via CSV, validating fields (e.g., email, role) and reporting errors.

Report Generation: Produce monthly leave summaries (pending, approved, rejected counts, total leave days), filterable by year, with PDF export.

Notifications: Trigger database-driven notifications for request submissions, approvals, and rejections, displayed as toasts or dropdowns.

RBAC: Restrict features by role (employee: leave submission/history; manager: approvals, reporting, imports).

Security: Implement login with CSRF protection, brute force lockout, audit logging, and mandatory first-time password changes.

3.2 Non-Functional Requirements

Usability: Provide intuitive, responsive interfaces for desktop and mobile users.

Simplicity: Ensure minimal setup and straightforward workflows for academic users.

Data Integrity: Maintain consistency via foreign keys, unique constraints, and validation.

Security: Enforce secure authentication, session management, and logging.

Performance: Optimize database queries for reporting and filtering with large datasets.

4. System Design

4.1 Data Model

The database (leave_management (12).sql) comprises eight tables:

employees: Stores user details (employee_id, first_name, last_name, email, password_hash, role as employee or manager, department_id, manager_id). Includes a manager account (user@example.com, role=manager, department_id=5) for testing.

leave_requests: Tracks requests (request_id, employee_id, leave_type_id, start_date, end_date, computed duration, status as pending/approved/rejected, manager_comment).

leave_types: Defines 10 leave types (e.g., Vacation: 15 days, Maternity Leave: 90 days) with max_days, is_paid, and eligibility_criteria.

leave_balances: Manages balances (balance_id, employee_id, leave_type_id, balance, max_balance).

departments: Lists 10 departments (e.g., Information Technology, Human Resources).

notifications: Stores messages (notification_id, employee_id, message, status as sent/pending). **audit_logs**: Logs actions (log_id, employee_id, action, ip_address, user_agent).

remember_tokens: Manages "Remember Me" tokens (id, user_id, token_hash, expires_at).

Entity Relationship Diagram (ERD):

employees references **departments** (department_id) and self (manager_id) for hierarchical management.

leave_requests links to **employees** (employee_id, manager_id) and **leave_types** (leave_type_id). **leave_balances** connects **employees** and **leave_types**.

notifications and **audit_logs** reference **employees** (employee_id).

remember_tokens links to **employees** (user_id).

Foreign keys enforce integrity (e.g., ON DELETE CASCADE for employees.department_id), ensuring consistent data.

4.2 System Architecture

The system follows a **3-tier architecture**:

Front-End: HTML, CSS (dashboard.css, manager_dashboard.css), JavaScript (dashboard.js, manager_dashboard.js), with Chart.js for report visualizations and Font Awesome for icons. Responsive design supports mobile access.

Logic Layer: PHP 8.2.12 processes requests via controllers (LeaveSubmissionController.php, HRImportController.php, ManagerDashboardController.php) and models (LeaveModel.php, Auth.php). Middleware (AuthMiddleware.php) enforces RBAC.

Database Layer: MariaDB 10.4.32, managed via phpMyAdmin 5.2.1, handles data storage and queries.

4.3 Workflow Diagram

1. Employee submits a request (leave_submission.php), selecting leave type, dates, and reason.
2. System validates inputs (LeaveSubmissionController.php) and stores in leave_requests, triggering a notification (notifications).
3. Manager reviews via manage_requests.php, approves/rejects with comments (LeaveRequestController.php), updating leave_requests.status.
4. Notification is sent to the employee (notifications), displayed as a toast or dropdown.

5. Implementation

5.1 Tools & Technologies

- Frontend: HTML, CSS, JavaScript, Chart.js (reports), Font Awesome (icons).
- Backend: PHP 8.2.12.
- Database: MariaDB 10.4.32, phpMyAdmin 5.2.1.
- Server: Apache.
- Integration: CSV imports (frontend/assets/imports/employees.csv).
- Version Control: GitHub.
- Development Environment: Local Apache server, accessed via http://localhost/leave-management-system/frontend/public/login.php.
-

5.2 Leave Request Form and Validation

The leave request form (leave_submission.php) enables employees to select from 10 leave types (e.g., Vacation, Sick Leave), input start/end dates, and provide reasons. Client-side validation (dashboard.js) checks for required fields, while LeaveSubmissionController.php validates:

- CSRF tokens for security.
- Date conflicts by querying leave_requests.
- Available balances in leave_balances.

Successful submissions are stored in leave_requests and trigger a notification (notifications).

Challenge: Preventing overlapping leave requests required complex SQL queries to check start_date and end_date in leave_requests. Solution: Implemented a custom validation function in LeaveSubmissionController.php to query existing requests, rejecting overlaps with clear error messages.

Challenge: Ensuring balance checks respected leave type limits (leave_types.max_days) was error-prone. Solution: Added a join with leave_balances and leave_types in LeaveModel.php to enforce limits, logging violations for debugging.

5.3 Approval Dashboard for Managers

The manager dashboard (manager_dashboard.php) provides a centralized interface with navigation to:

- **Manage Requests** (manage_requests.php): Displays a paginated table of pending requests, with modals for approval/rejection and comments.
- **Leave History** (leave_history.php): Filters requests by employee name, leave type, status, and date range, with pagination.
- **Reporting** (reporting.php): Generates monthly leave summaries.
- **HR Import** (hr_import.php): Uploads employee CSVs.
- **Settings**: Manages profile and notifications.

LeaveRequestController.php processes approvals, updating leave_requests.status, approved_at, or rejected_at, and triggering notifications. ManagerDashboardController.php fetches dashboard data (statistics, trends, notifications) via AJAX.

Challenge: Implementing responsive filters in leave_history.php slowed queries with large datasets. Solution: Added indexes on leave_requests (employee_id, leave_type_id, status) and optimized joins in LeaveModel.php, reducing query time.

Challenge: AJAX-based notification updates (dashboard.js, manager_dashboard.js) risked race conditions with frequent requests. Solution: Implemented debouncing in JavaScript and cached notification counts in ManagerDashboardController.php to minimize database hits.

5.4 Integration with Mock HR Data

- The HR import module (hr_import.php) allows managers to upload CSV files (employees.csv) containing employee data (name, email, role, department). HRImportController.php:
- Validates CSV headers and data (e.g., unique emails, valid roles).
- Generates bcrypt password hashes for new users.
- Inserts records into employees, logging default passwords (imported_default_password.log). Displays successes and errors (e.g., duplicate emails, invalid formats).

A pre-existing manager account (user@example.com, role=manager) in employees enables instructors to test imports without creating accounts.

Challenge: Handling malformed CSVs (e.g., missing columns, invalid emails) caused import failures. Solution: Added comprehensive validation in HRImportController.php, parsing CSVs line-by-line and returning detailed error messages (e.g., "Line 5: Invalid email format").

Challenge: Ensuring unique emails during imports required efficient database checks. Solution: Used a UNIQUE constraint on employees.email and batched insert queries, catching duplicate errors gracefully.

5.5 Report Module

- The report module (reporting.php) generates a monthly leave summary table, displaying:
- Pending, approved, and rejected request counts.
- Total leave days per month.
- Year-based filtering via a dropdown.
- PDF export via generate_report.php (assumed functionality).
- Chart.js visualizations (e.g., bar charts for leave trends).

Queries aggregate leave_requests data, grouping by month and status, with joins to employees and leave_types for context.

Challenge: Aggregating large leave_requests datasets for reports caused performance issues. Solution: Added indexes on leave_requests.start_date and status, and used aggregated subqueries in LeaveModel.php to optimize performance.

Challenge: Ensuring PDF exports (generate_report.php) maintained table formatting was complex. Solution: Assumed a library like TCPDF was used, with predefined styles to align with reporting.php's HTML table, though exact implementation details were unavailable.

5.6 Notifications

Notifications are stored in the notifications table (notification_id, employee_id, message, status). Triggers include:

- Request submission (leave_submission.php).
- Approval/rejection (manage_requests.php).
- Import completion (hr_import.php).

ManagerDashboardController.php and EmployeeDashboardController.php fetch notifications via AJAX, displaying them as toasts or dropdowns with unread counts (dashboard.js). Users mark notifications as read, updating notifications.status.

Challenge: Scalability of notification fetches risked database overload with frequent AJAX calls. Solution: Limited queries to unread notifications and implemented pagination in LeaveModel.php, with client-side caching in dashboard.js.

Challenge: Ensuring notifications were user-specific required precise RBAC checks. Solution: Filtered by employee_id in LeaveModel.php and validated roles in AuthMiddleware.php.

5.7 Role-Based Access Control (RBAC)

RBAC restricts access based on employees.role (employee or manager):

Employees: Access leave_submission.php, leave_history.php (personal history), and employee_dashboard.php.

Managers: Access all employee features plus manage_requests.php, reporting.php, hr_import.php, and leave_history.php (all employees).

AuthMiddleware.php enforces RBAC by checking Session::get('role'), redirecting unauthorized users. Session.php ensures secure sessions (HTTPS, HTTP-only, SameSite=Strict).

Challenge: Preventing session hijacking required robust security. Solution: Implemented CSRF tokens in all forms (LoginController.php, LeaveSubmissionController.php) and secure session settings in Session.php.

Challenge: Ensuring managers couldn’t access restricted endpoints (e.g., reporting.php) without authentication was complex. Solution: Centralized role checks in AuthMiddleware.php, logging unauthorized attempts in audit_logs.

5.8 Security

Security features include:

- **CSRF Protection**: Tokens in forms (login_view.php, leave_submission.php) validated by controllers. Brute Force Lockout: LoginController.php limits login attempts, logging failures in audit_logs.
- **Audit Logging**: audit_logs tracks logins, password changes, and imports.
- **Mandatory Password Changes**: First-time users (employees.first_login=1) must change passwords via login_view.php.
- **Secure Sessions**: Session.php enforces HTTPS and secure cookies.
- **Remember Me**: remember_tokens stores hashed tokens for persistent logins.

Challenge: Implementing CSRF protection across all forms was time-consuming. Solution: Created a reusable CSRF token generator in Session.php, integrated into all POST endpoints.

Challenge: Balancing brute force lockout with user accessibility required fine-tuning. Solution: Set a 5-attempt limit with a 15-minute lockout in LoginController.php, logging attempts for review.

6. Testing

6.1 Unit Testing

Login Validation: Tested login.js for email/password input validation and password_validation.js for strength checks (length, uppercase, number, special character).

Form Submission: Validated leave_submission.php inputs (dates, leave type) and LeaveSubmissionController.php checks (CSRF, balances).

Notification Updates: Ensured dashboard.js correctly marked notifications as read, updating notifications.status.

6.2 Integration Testing

- CSV Imports: Verified hr_import.php populated employees with valid CSVs and reported errors for invalid formats (e.g., duplicate emails).
- Leave Workflow: Tested end-to-end flow: submission (leave_submission.php), approval (manage_requests.php), and notification (notifications).

- Reporting: Confirmed reporting.php aggregated leave_requests correctly, with accurate counts and PDF exports.

6.3 User Testing

- **Employee Workflow**: Users submitted requests and viewed history (leave_history.php), reporting intuitive navigation but requesting clearer date pickers, addressed in dashboard.css.
- **Manager Workflow**: Managers tested approvals, history filtering, and reporting, suggesting improved filter labels in leave_history.php, implemented via manager_dashboard.js.
- **Feedback**: Highlighted the need for mobile responsiveness, already supported by manager_dashboard.css's media queries.

6.4 Instructor Testing

The manager account (user@example.com, role=manager) in employees allows instructors to:

- Log in (login.php) and test CSV imports (hr_import.php).
- Generate reports (reporting.php) and filter leave history (leave_history.php).

Lesson Learned: Pre-configuring a manager account streamlined instructor testing, a critical consideration for deployment.

7. Results & Evaluation

7.1 Working Features

- **Leave Submission**: Employees submit requests via a form (leave_submission.php) with dropdowns for 10 leave types, date pickers, and reason fields. The UI uses a clean, card-based layout (dashboard.css), validated client-side (dashboard.js) and server-side (LeaveSubmissionController.php).
- **Manager Approvals**: Managers view pending requests in a paginated table (manage_requests.php), with modals for approval/rejection and comments. The table is responsive, collapsing on mobile (manager_dashboard.css).
- **Leave History**: Employees see personal history (leave_history.php), while managers filter all requests by employee, leave type, status, and dates, with pagination for scalability.
- **Reporting**: Monthly summaries (reporting.php) display pending, approved, rejected counts, and total leave days in a table, filterable by year, with Chart.js bar charts and PDF export. The UI is intuitive, with dropdowns and buttons styled consistently (manager_dashboard.css).
- **CSV Integration**: Managers upload CSVs (hr_import.php), with a drag-and-drop interface and error/success feedback in a modal. Validated data populates employees, ensuring data quality.

- **Notifications**: Toasts and dropdowns (employee_dashboard.php, manager_dashboard.php) show request updates, with unread badges and AJAX-based read actions (dashboard.js).
- **RBAC**: AuthMiddleware.php restricts manager features (e.g., reporting.php) to role=manager, with seamless redirects for unauthorized access.
- **Security**: CSRF tokens, brute force lockouts, audit logging (audit_logs), and secure sessions (Session.php) ensure robust protection.

7.2 Data Integration and Reporting

CSV imports (hr_import.php) integrate mock HR data into employees, validating emails, roles, and departments. Errors (e.g., duplicates) are displayed clearly, maintaining data quality. Reports (reporting.php) aggregate leave_requests data, providing actionable insights via tables and charts, with PDF exports for offline use.

Evaluation: Integration is robust but manual; automated API sync could enhance efficiency. Reporting is effective but limited to monthly summaries; predictive analytics would add value.

7.3 Workflow Design

The workflow (submission → approval → notification) is automated, with RBAC ensuring secure access. Notifications (notifications) improve communication, and audit logging (audit_logs) supports compliance.

Evaluation: The workflow is efficient but relies on simulated notifications. Real-time alerts and bulk approval options would enhance scalability.

Lesson Learned: User feedback emphasized intuitive filters, leading to UI refinements in leave_history.php and reporting.php.

7. Conclusion & Future Enhancements

8.1 Summary

The Employee Leave Management System delivers a comprehensive solution for academic institutions, enabling leave requests, manager approvals, HR data integration, and detailed reporting. Implemented with PHP 8.2.12, MariaDB 10.4.32, and a 3-tier architecture, it features:

- Robust leave management (leave_submission.php, manage_requests.php).
- CSV-based HR integration (hr_import.php) with a pre-existing manager account (user@example.com) for testing.
- Monthly reporting (reporting.php) with filtering and PDF export.

- Database-driven notifications (notifications) and RBAC (AuthMiddleware.php).
- Security via CSRF protection, audit logging (audit_logs), and secure sessions (Session.php).

The system meets all objectives, demonstrating Information Management and System Integration principles through data integrity, automation, and modular design. Its accessibility for instructor testing underscores practical deployment readiness.

8.2 Future Enhancements

- **File Upload for Leave Requests**: Enhance leave_submission.php to allow employees to upload supporting documents (e.g., medical certificates for Sick Leave or travel plans for Vacation) when submitting leave requests. This would involve adding a file input field to the form, validating file types (e.g., PDF, JPG) and size limits in LeaveSubmissionController.php, and storing files securely in a designated directory with references in the leave_requests table. Managers could view these files via manage_requests.php to verify requirements for specific leave types (e.g., Maternity Leave), ensuring compliance with organizational policies.
- **Real Email Notifications**: Integrate SendGrid/Twilio for email/SMS alerts, replacing simulated notifications.
- **Role-Based Access Expansion**: Add roles (e.g., HR admin) to employees.role.
- **Cloud Deployment**: Deploy to AWS/Azure for scalability.
- **Progressive Web App (PWA)**: Leverage manager_dashboard.css's responsive design for offline access and push notifications.
- **API Integration**: Expose a RESTful API for HR system sync, secured with OAuth 2.0.
- **Advanced Analytics**: Add predictive trends and cross-department reports to reporting.php using D3.js.
- **Automated Leave Balances**: Schedule tasks to update leave_balances based on policies.
- **Bulk Request Management**: Enable batch approvals in manage_requests.php.
- **Accessibility Compliance**: Ensure WCAG 2.1 compliance for dashboard.css and UI components.

8.3 Challenges and Lessons Learned

Building the Employee Leave Management System taught us a lot about solving real-world problems with technology. Below are the key challenges we faced, how we tackled them, and the simple lessons we learned to make future projects better.

- **Challenge**: Uploading CSV files (hr_import.php) was tricky because some files had errors, like missing columns or wrong email formats. This could break the import process and mess up the employee data.
- **Solution**: We added strong checks in HRImportController.php to read each line of the CSV file carefully. If something was wrong, like a bad email or missing name, we showed clear error messages (e.g., "Line 5: Invalid email format") so users could fix it.
- **Lesson**: Checking data carefully before saving it is super important. Clear error messages help users understand what went wrong, and we'll keep this in mind for any future data uploads.

- **Challenge**: Making sure only the right people (like managers) could access certain features, like reports (reporting.php), was hard. Without proper checks, someone could sneak into parts of the system they shouldn't see.
- **Solution**: We built AuthMiddleware.php to check a user's role (employee or manager) every time they tried to access a page. If they weren't allowed, we sent them back to the login page and saved a note of the attempt in audit_logs for security.
- **Lesson**: Controlling who can see what in a system is a must for safety. Using a single place (like middleware) to check permissions makes things easier and safer, and we'll use this approach again.

- **Challenge**: Notifications (notifications table) slowed down the system when users kept refreshing their dashboards. Too many database calls made things sluggish, especially with lots of users.
- **Solution**: We limited database calls by only fetching unread notifications and splitting them into smaller chunks (pagination) in LeaveModel.php. We also saved some data temporarily on the user's browser (dashboard.js) to avoid extra calls.
- **Lesson**: Planning for speed early on saves headaches later. Simple tricks like limiting data fetches and storing some data on the user's side can make the system faster, especially for big projects.

- Challenge: Adding CSRF protection to every form (like login_view.php and leave_submission.php) took a lot of time. Without it, hackers could trick users into submitting fake forms.
- Solution: We created a reusable CSRF token generator in Session.php that we used across all forms. This made sure every form submission was safe and checked properly by the system.
- Lesson: Security can't be an afterthought. Building reusable tools, like a token generator, saves time and keeps things secure. We'll plan for security from the start in future projects.

- **Challenge**: Generating reports (reporting.php) was slow when there were lots of leave requests to process. Slow reports frustrated users who needed quick insights.
- **Solution**: We added indexes to the leave_requests table (on columns like start_date and status) to make database searches faster. We also simplified queries in LeaveModel.php to handle big datasets better.
- **Lesson**: Fast systems need smart database planning. Adding indexes and writing efficient queries early on makes reports load quickly, even with tons of data. We'll prioritize this in the future.
- **Challenge**: Users found some parts of the system, like filtering leave history (leave_history.php), confusing at first. They wanted clearer labels and easier ways to find what they needed.

- **Solution**: We improved the filter labels and buttons in leave_history.php and manager_dashboard.js based on user feedback. We also made the date pickers in dashboard.css more user-friendly.
- **Lesson**: Listening to users is key to making a system they love. Testing with real users early on helps spot problems and make the system easier to use. We'll involve users sooner next time.

- **Challenge**: Testing the system with lots of data (like thousands of leave requests) was tough because it showed where things could break, like slow report generation or import errors.
- **Solution**: We ran tests with big datasets to find weak spots. For example, we optimized database queries for reports and added better error handling for CSV imports in HRImportController.php.
- **Lesson**: Testing with realistic data helps find problems before they happen. We'll make sure to test with large datasets early in future projects to catch issues sooner.
-

Lesson: Breaking the system into small, reusable pieces (like controllers and models) made it easier to build, test, and fix. Clear documentation (README.md) also helped everyone understand the system. Takeaway: Modular design and good documentation save time and make teamwork smoother. We'll stick to this approach.

Lesson: User feedback showed us that simple, clear interfaces (like better filters and date pickers) make a big difference. Takeaway: Designing with users in mind from the start leads to a better system. We'll keep users involved throughout development.

These challenges and solutions taught us how to build a better system by focusing on data quality, security, speed, and user needs. These lessons will guide us to create even stronger systems in the future.