Faculty Kit

The faculty kit contains the evaluation strategy for the different milestones of the project and any other documents/links that may aid in the evaluation process (like sample quizzes on technologies etc)

Evaluation Strategy/Tips for the different milestones of the project

Objective:-

The objective of this faculty kit is to provide comprehensive documentation and guidance for the **Leave Management System (LMS)** project. LMS is a web-based platform designed to automate the process of leave application, approval, and tracking for employees in an organization. This kit outlines the project's goals, technical specifications, design structure, implementation strategy, and testing plans to aid faculty in overseeing and evaluating the project effectively.

Requirements Specification:-

The **Leave Management System (LMS)** includes the following core functionalities:

- **User Authentication and Role Management**: Secure login for Admins, Managers, and Employees.
- **Leave Application and Approval**: Employees can submit leave requests which managers can approve or reject.
- **Leave Balance Tracking**: Real-time leave balance updates after approvals and deductions.
- **Notifications and Alerts**: Automated alerts for approvals, rejections, reminders, and policy updates.
- **Leave Policy Management**: Admins can define different leave types, eligibility rules, and accrual methods.
- **Job and Mentorship Module**: Employees can post job opportunities and access mentorship resources.
- **Reporting and Analytics**: Provides reports on absenteeism, leave trends, and employee availability.
- **Feedback and Support System**: Employees can raise queries or provide feedback through a ticketing system.
- **Role-Based Access**: Different features and views are available based on user roles (Employee/Admin).
- **System Configuration Tools**: Admin dashboard includes settings for managing users, policies, and system notifications.

Technology Familiarization:-

The **Leave Management System** is developed using modern web technologies and tools that ensure scalability, maintainability, and performance:

- **Frontend:**
  - **HTML5, CSS3** – for structuring and styling web pages.
  - **Laravel Blade Templates** – for rendering dynamic views and layouts.
- **Backend:**
  - **PHP (Laravel Framework)** – provides MVC structure, routing, authentication, and ORM for database handling.
- **Database:**
  - **MySQL** – a relational database used for storing employee data, leave records, and system configurations.
- **Development Tools:**
  - **Visual Studio Code** – as the code editor.
  - **XAMPP** – for running the local development server.
- **Version Control:**
  - **Git and GitHub** – for source code management and collaborative development.
- **Documentation Tools:**
  - **MS Word, LaTeX, and PowerPoint** – used for creating formal project reports and presentations.

Database Creation:-

The **Leave Management System** uses a **relational database (MySQL)** to store and manage structured organizational data. The key tables in the system include:

1. **Users Table**: Stores details of all users (employees, managers, admins) including credentials, roles, and contact information.
2. **Leave_Requests Table**: Records each leave request with attributes like leave type, duration, reason, status (pending/approved/rejected), and timestamps.
3. **Leave_Types Table**: Contains the types of leave (e.g., sick leave, casual leave, earned leave) and their respective rules.
4. **Leave_Balance Table**: Tracks leave balances for each employee, updated automatically after leave actions.
5. **Notifications Table**: Stores system-generated alerts like leave status changes, low balance warnings, and reminders.
6. **Feedback Table**: Captures employee queries, feedback, and ticket statuses.
7. **Jobs and Mentorship Tables**: For job postings and mentorship-related entries within the organization.

The database schema ensures normalization, relational integrity, and scalability for future enhancements such as payroll or calendar integrations.

High-Level and Detailed Design:-

System Overview:

The **Leave Management System (LMS)** follows a three-tier architecture consisting of:

1. **Frontend Layer:**
   - Built using **HTML, CSS, and Laravel Blade templates**.
   - Provides an interactive UI for employees, managers, and administrators.
2. **Backend Layer:**
   - Powered by the **Laravel PHP framework**.
   - Handles business logic, authentication, data validation, and communication with the database.
3. **Database Layer:**
   - Uses **MySQL** for persistent storage of all system data.
   - Ensures data integrity and supports efficient querying of leave records, user profiles, and analytics.

Detailed Design:

- **Routing System:** Laravel's routing mechanism directs HTTP requests to appropriate controllers.
- **Controller Logic:** Controllers manage functionalities like leave application, profile updates, approval workflows, and feedback handling.
- **Models & ORM:** Laravel's Eloquent ORM manages database interactions using models mapped to tables.
- **Middleware:** Ensures secure access control by checking roles and permissions during requests.
- **UI Elements:** Dashboards, leave forms, alerts, and reports are dynamically rendered based on user roles.

The system is modular and extensible, allowing future enhancements like chatbot integration, API-based payroll sync, or mobile responsiveness.

Front-end implementation:-

A demo of the front-end can be given by the team for evaluation. Features like mail notification, actual updation of the database etc come in the next stage. This stage is for the front-end only.

Evaluation can be based on parameters like:

1. The look-and-feel of the user interface, like whether there are too many boxes/links in one page, whether any sound-effects are there (for successful/unsuccessful logins, for example) etc
2. The quality of help available for the user, like whether the help instructions match the actual implementation, whether they are simple to understand etc
3. The intutive-ness of the design, like whether you are able to use the application without spending too much time in reading help screens etc
4. Appropriate, meaningful and non-confusing error messages where applicable etc

Integrating the front-end with the database:-

The integration between the frontend and backend in the **Leave Management System** is handled efficiently using Laravel's MVC architecture. Data exchange is seamless due to Laravel's built-in tools.

Integration Process:

- **Blade Templates ↔ Controllers:**
  - Laravel Blade templates send form data (e.g., leave requests) to controllers via HTTP POST methods.
  - Controllers validate and process the data before interacting with the database.
- **Eloquent ORM ↔ MySQL:**
  - Laravel's Eloquent ORM maps PHP classes (models) to MySQL tables.
  - CRUD operations (Create, Read, Update, Delete) are handled with simple and secure syntax.
- **Data Binding:**
  - Dynamic content like user profiles, leave balances, and request statuses is displayed in the frontend using Blade's data binding features.
- **Session and Authentication:**
  - Laravel's built-in authentication system manages user sessions and access control, ensuring role-based data access.
- **Notification System:**
  - Notification triggers are integrated in the backend and sent to users via the frontend (pop-ups, banners, or emails).

This tight integration ensures consistent data handling, real-time updates, and a smooth user experience across all modules.

Test-plan review:-

The test-plan document will be reviewed for its completeness, correctness, and clarity. Some points to look for are:

1. **Coverage of Requirements:**
   Whether all the requirements specified in the **Requirement Specification (RS)** document are being adequately tested for their functionality. Each feature

mentioned in the RS should have corresponding test cases ensuring that the implemented functionality meets the expectations.

2. **Clarity in Test Case Execution:**
   Whether all the test cases contain clear and detailed descriptions on how to execute them. The instructions should be easy to follow, including steps, expected outcomes, and any setup requirements needed to execute the test.
3. **Testing Error Scenarios:**
   Whether all potential error scenarios, such as **invalid inputs**, **incorrect credentials**, or **failed transactions**, are being tested. This will ensure that the system behaves appropriately when faced with faulty or unexpected user input or system failures.
4. **Testing Exception Scenarios:**
   Whether all possible exception scenarios are being accounted for. These could include situations like a **database shutdown**, **system crashes**, or **network interruptions** during operations (e.g., during an order placement process or payment gateway interaction). Testing these scenarios helps ensure the system can recover gracefully and provide appropriate feedback to the user.

Final review:-

The final evaluation will be based on the **final demo** and the **project report**. These will be assessed to determine if the project meets all the specified requirements and objectives. In addition to these, any **intermediate presentations**, **documents**, and **write-ups** submitted throughout the project can also be reviewed as part of the evaluation process to assess the development progress and consistency in the project.

Documents/References that may aid the process of evaluation:-

1. **W3Schools ASP Quiz**
   - W3Schools ASP Quiz
2. **Java Official Documentation**
   - Java - Sun Microsystems
3. **Flutter Documentation**
   - Flutter Official Docs
4. **Firebase Documentation**
   - Firebase Docs
5. **Node.js Documentation**
   - Node.js Docs
6. **MongoDB Documentation**

- MongoDB Docs

7. **OWASP Top Ten Security Risks**
   - OWASP Top Ten
8. **UX Design Principles**
   - Nielsen Norman Group - UX Design
9. **Agile Methodology Guide**
   - Agile Alliance
10. **API Design and Best Practices**
    - API Design Guide by Google