Employee Management System Requirements specifications.

**Abstract.**

The Employee Management System is a web-based application that enables managers to manage employee records, leave schedules, and performance. The system is designed to be user-friendly, efficient, and secure, providing a comprehensive solution for managing employees. The system will be built using ReactJS for the front-end, and NodeJS/ExpressJS for the back-end. The database will be built using MySQL. The system will be hosted on a cloud platform called Linode server. The system will allow managers to add, edit, and delete employee records, track the number of leave days due and taken by each employee, approve or reject employee leave applications, track employee performance metrics such as number of hours worked and projects completed, and generate reports based on employee performance. The system will also have proper authentication and authorization mechanisms in place to prevent unauthorized access, and will be designed to handle a growing number of employees and transactions. Overall, the Employee Management System is a comprehensive solution for managing employee records, leave schedules, and performance, designed to help managers manage their employees more effectively.

* 1. **Introduction**

The Employee Management System is a web application that enables managers to manage employee records, leave schedules, and performance. The application is designed to be user-friendly and efficient, providing a comprehensive solution for employee management.

* 1. **Purpose Of This Document**

This document is primarily intended to be proposed to the interested stakeholders and users for approval and also as a reference for developing the first version of the employee management system. The SRS documents outlines the requirements and specifications for the employee management system. It helps to ensure that the software meets the needs of the business and its customers. It also ensures that all stakeholders understand the scope of the project. This document, to be specific, was prepared to describe the functionality, performance, attributes, and design constraints of the software to be developed.

* 1. **Objectives Of The Employee Management System.**
* Allows managers to create, read, update and delete employee records.
* Track the number of leave days due and taken by each employee.
* Approve or reject employee leave applications.
* Notify other employees of approved leave applications and relevant dates
* Remind all employees when leave dates are approaching.
* Track employee performance metrics such as number of hours worked and projects completed.
* Generate reports based on employee performance.
* Secure login for employees and managers.
* Different access levels for employees and managers.
* A dashboard for managers to view important metrics and summaries of employee records, leave schedules, and performance.
  1. **Software Overview.**
* The Employee Management System is a web-based application that enables managers to manage employee records, leave schedules, and performance.
* The system is designed to be user-friendly, efficient, and secure, providing a comprehensive solution for managing employees.
* The system consists of a front-end, back-end, and database, each serving a specific purpose.
* The front-end of the system is built using modern web frameworks such as ReactJS, providing a user-friendly interface for managers to interact with the system.
* The back-end of the system is built using ExpressJS programming language, handling the business logic of the system, processing requests from the front-end, and communicating with the database.
* The database of the system will be done using MySQL, storing all employee records, leave schedules, and performance metrics.
* The system allows managers to add, edit, and delete employee records, track employee leave schedules, track employee performance metrics, and generate reports based on employee performance.
* The system provides proper authentication and authorization mechanisms to prevent unauthorized access.
* The system is designed to handle a growing number of employees and transactions, ensuring scalability and reliability.
  1. **Business Requirements.**
* The system should be user-friendly and easy to use for managers and employees.
* The system should be scalable and able to handle a growing number of employees and transactions.
* The system should be secure and protect employee data from unauthorized access.
* The system should be designed to improve employee productivity and performance.
* The system should provide accurate and timely information to managers for decision-making.
* The system should be integrated with other HR systems or payroll systems as required.
* The system should allow for customization and configuration based on the needs of the organization.

**1.6) Market Requirements.**

* The system should be competitively priced and affordable for small and medium-sized businesses.
* The system should have a user-friendly interface that requires minimal training for managers and employees.
* The system should be reliable and available with minimal downtime.
* The system should be compatible with different devices and operating systems.
* The system should be compliant with relevant laws and regulations regarding employee data privacy and security.
* The system should provide features that are relevant and useful for different industries and types of businesses.
* The system should provide good customer support and maintenance services.
  1. **Functional Requirements.**
* **Employee Records Management:**

The system should allow authorized users to add, edit, and delete employee records. This functionality should be easy to use, with clear prompts and instructions to guide users through the process. The system should store and manage employee information such as personal details, job title, employment status, salary, and performance metrics. The system should also allow authorized users to view employee records and search for specific employees based on various criteria.

* **Leave management:**

The system should allow employees to submit leave requests and managers to approve or reject employee leave requests. This functionality should include a calendar view of leave schedules and the ability to view employee leave balances. The system should track the number of leave days due and taken by each employee and notify other employees of approved leave applications and relevant dates. The system should also remind all employees when leave dates are approaching.

* **Authentication and Authorization:**

The system should provide secure login for employees and managers, with strong password policies enforced. Different access levels should be provided for employees and managers, with appropriate permissions and restrictions. This functionality should also include password reset and recovery options for users who forget their login credentials.

* **Dashboard**

The system should provide a dashboard for managers to view important metrics and summaries of employee records, leave schedules, and performance. The dashboard should be customizable, allowing managers to choose which metrics and summaries are displayed and in what format. The dashboard should also include charts, graphs, and other visual aids to help managers quickly understand important information.

* **Reporting**

The system should allow authorized users to generate reports based on employee data such as attendance, leave, and performance. The reports should be customizable, allowing users to choose which data is included and in what format. The reports should also be exportable to different formats such as PDF or CSV, and should be easy to read and understand.

* **Integration**

The system should allow integration with other HR systems or payroll systems as required. This functionality should include the ability to import and export employee data and the ability to synchronize data between different systems. The system should also be compatible with different devices and operating systems to ensure maximum flexibility and accessibility.

* **Notifications.**

The system should provide notifications to employees and managers for important events such as leave approvals or performance reviews. The notifications should be customizable, allowing users to choose how they receive them (e.g., email, SMS, in-app notifications). The notifications should also be timely and relevant, providing users with the information they need to stay informed and make informed decisions.

* 1. **Non-Functional Requirements.**
* **Performance**

The system should respond to user requests quickly and efficiently, with minimal latency and delays. The system should be able to handle a large number of concurrent users without compromising performance. The response time for critical functions such as login and data retrieval should be within acceptable limits. The system should also be designed to handle peak loads and spikes in traffic without crashing or slowing down.

* **Security**

The system should be designed to protect employee data from unauthorized access, with appropriate encryption and hashing mechanisms in place. The system should be compliant with relevant laws and regulations regarding employee data privacy and security. The system should have proper access controls in place, with different levels of access for employees and managers. The system should also have proper backup and disaster recovery mechanisms in place to ensure data is not lost in case of system failures or disasters.

* **Usability**

The system should be easy to use and navigate, with a user-friendly interface that requires minimal training for users. The system should provide clear and concise error messages when errors occur. The system should be accessible to users with disabilities, with appropriate assistive technologies in place. The system should also be designed to be responsive and adaptable to different screen sizes and resolutions.

* **Availability**

The system should be highly available, with minimal downtime for maintenance and upgrades. The system should have proper monitoring and alerting mechanisms in place to detect and respond to system failures quickly. The system should also be designed to be fault-tolerant, with the ability to automatically recover from system failures or errors.

* **Compatibility**

The system should be compatible with different devices and operating systems, including desktop and mobile devices. The system should be compatible with different web browsers, including popular browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge. The system should also be designed to work with different hardware and software configurations.

* **Scalability**

The system should be designed to handle a growing number of employees and transactions without compromising performance. The system should be able to scale horizontally and vertically, with the ability to add more servers or resources as needed. The system should also be designed to handle different workloads and usage patterns, with the ability to automatically adjust resources as needed.

* **Maintainability**

The system should be easy to maintain and modify, with clear and well-documented code. The system should be designed with modularity and reusability in mind, with code that is easy to test and debug. The system should have proper version control mechanisms in place, with a clear and well-defined release process. The system should also be designed to minimize downtime during maintenance and upgrades.

* 1. **System Architecture.**

**Frontend:**

* React for building user interface components and rendering pages.
* Redux for state management.
* React Router for routing.
* Webpack for bundling.
* The frontend will need to communicate with the backend to retrieve data and process user requests using the axios library or redux toolkit query.

**Backend:**

* ExpressJS will be used for making the backend-APIs for the application.
* ExpressJS will be used for handling the HTTP requests,
* MySQL will be used as the relational database for storing information.

**Hosting:**

* Linode server for hosting and deploying the application.
  1. **System Design.**

In order to design the web application, the relational database should be designed first. Conceptual design can be divided into two parts: data model and the process model. The data model focuses on what data should be stored in the relational database while the process model deals with how the data is processed. To put this in context of relational databases, the data model is used to design relational tables. The process model is used to design the queries that will access and perform operations on those tables

**Tables in the database**

* users
* departments
* jobs
* employees
* leaves
* performances
* applications

**Users**

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Description |
| Id (PK) | Integer | Auto incrementing unique Id |
| username | Text | Name of user |
| email | Text | Email of user |
| password | Text (Hashed password) | Password of user |
| Is\_admin | Text | (Yes / No) |

**Jobs**

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Description |
| Id (PK) | Integer | Auto incrementing unique Id |
| Job\_title | text | Title of the job |
| Job\_descriptions | text | Description of the job |
| Job\_location | text | Location of the job |
| Job\_salary | text | Salary of the job |
| Job\_postedBy (FK Id to users.id | text | Admin who posts the job |
| Posted\_at | date | Time of the job posting |

**Applications**

|  |  |  |
| --- | --- | --- |
| Column name | Type | Description |
| id (PK) | Int | Auto incrementing unique Id |
| First\_name | text | Applicant\_name |
| Last\_name | text | Applicant\_last name |
| email | text | Applicant email |
| number | text | Applicant phone number |
| job\_id (FK to jobs.id) | Int | Foreign key from jobs table |
| user\_id (FK to users.id) | Int | Foreign key from users table |
| status (accepted, rejected, pending) | text | Status of the application |
| applied\_at | text | Time of applying the job |
| reviewed\_by (FK to users.id) | Int | Admin who reviewed the application |
| reviewed\_at | text | Time of review |

**Employees**

|  |  |  |
| --- | --- | --- |
| Column name | Type | Description |
| Id (PK) | Int | Auto increment id |
| First\_name | text | Employee name |
| Last\_name | text | Employee surname |
| email | text | Employee email |
| Phone\_number | text | Employee number |
| Hire\_date | text | Date of hiring |
| Job\_id (FK to jobs.id) | Int (FK) | Employee job id |
| Manager\_id (FK to employees.id) | Int (FK) | Employee manager Id |
| Department\_id (FK to departments.id) | Int (FK) | Department Id |

**Departments**

|  |  |  |
| --- | --- | --- |
| Column name | Type | Description |
| Id (PK) | Int | Auto increment id |
| Department\_name | text | Name of dept |

**Performances**

|  |  |  |
| --- | --- | --- |
| Column name | Type | Description |
| Id (PK) | Int | Auto increment id |
| Employee\_id (FK to employees.id) | Int | Employee id |
| date | text | Date of performance |
| rating | text (high, medium, low) | Employee rating |
| description |  | Description of the review |

**Leaves**

|  |  |  |
| --- | --- | --- |
| Column name | Type | Description |
| Id (PK) | Int | Auto increment id |
| Employee\_id (FK to employees.id) | Int | The owner of the leave |
| Start\_date | text | Start date of the leave |
| End\_date | text | End data of the leave |
| Approved\_by (FK to employees.id) | Int | The manager which approves the leave |
| Approval\_status | text | (Approved, pending, rejected) |