Human Resource Management System (HRMS) - FullStack Developer Assignment

Overview:

Design a Human Resource Management System (HRMS) with both a frontend UI and a backend service. The requirements are intentionally broad to allow candidates to make design decisions on their own.

Functional Requirements:

1. **Employee List**: Display a list of employees.
2. **Teams List**: Display a list of teams.
3. **Team Assignment**: Show which employees belong to which teams.
4. **Forms**: Provide forms for creating employees and teams.
5. **CRUD Operations**: Allow for the updating and deletion of employees and teams.
6. **Logging**: Maintain a log of all backend operations, including the following actions:
    - User login/logout
    - Employee creation
    - Team creation
    - Employee-team assignment changes
    - Employee/team deletion or updates

Example of log entries:

```
[TIMESTAMP] User '<user_id>' logged in.
[TIMESTAMP] User '<user_id>' added a new employee with ID <employee_id>.
[TIMESTAMP] User '<user_id>' updated employee <employee_id>.
[TIMESTAMP] User '<user_id>' assigned employee <employee_id> to team <team_id>.
[TIMESTAMP] User '<user_id>' deleted employee <employee_id>.
[TIMESTAMP] User '<user_id>' logged out.
```

- The log entries can be in a different format, but the goal is to record all relevant operations that affect the data.

Additional Notes:

- An **employee** can belong to **multiple teams**. This relationship should be reflected in the system design.
- All actions (adding, updating, deleting employees or teams) should be permitted **only after successful authentication**.
    - A user must first create an **organisation account** to manage employees and teams for that organisation.

Must-use Technologies:

- **Backend**: Node.js
- **Frontend**: React.js
- **Database**: Any SQL database (e.g., MySQL, PostgreSQL, etc.)

Judging Criteria:

1. **Code Architecture**: How well is the project structured? Does it follow best practices for scalability and maintainability?
2. **Code Optimisation**: Is the code efficient? Does it follow performance best practices?
3. **Security & Logging**: How well is user authentication implemented? How secure is the system? Is logging handled appropriately for audit trails?
4. **Database Schema Design**: How well is the database schema designed? Is it normalized? Does it support the application's needs efficiently?

Bonus Points:

- Design a simple but effective UI for managing employees and teams.
- Use libraries or frameworks that improve productivity (e.g., ORM for database interaction, authentication middleware, etc.).
- Provide clear documentation, including instructions for setting up and running the project.