PHP Developer Assessment Assignment:
Complaint Ticket Management System

This assessment is designed to evaluate the applicant's knowledge of PHP (specifically with Laravel), API development, database design, SQL proficiency, and front-end skills. The applicant will build a *Complaint Ticket Management System* where users can submit and track complaints, and administrators can manage them.

Assignment Objective:

Build a basic *Complaint Ticket Management System* using Laravel with API-driven architecture. The system should have the following key features:

Functional Requirements:

1. **User Registration & Authentication**
   - Users must be able to register and log in to the system.
   - Use Laravel's built-in authentication or a custom solution.

2. **Roles and Permissions**
   - There should be two types of users: *Users* (who can submit complaints) and *Administrators* (who can manage complaints).
   - Only administrators can close complaints or update their statuses.

3. **Complaint Submission**
   Authenticated users should be able to submit a complaint with fields such as:
   - Title
   - Description
   - Category (e.g., Billing, Service Issue, Product Issue)
   - Priority (Low, Medium, High)
   - Attachment (optional, allow file upload)
   - Store complaint submission date and user ID.

4. **Complaint Management (Admin Only)**
   Administrators should be able to:
   - View all complaints.
   - Update the status of complaints (e.g., Open, In Progress, Resolved, Closed).
   - Add comments to complaints.
   - Filter complaints by status, priority, and category.

5. **API Development**
   The system should expose a REST API that allows users to:
   - Create a complaint

    - View all their complaints
    - View a specific complaint's details
    - Administrators should be able to update complaint status via the API.
  - Proper authentication should be implemented for API access.

6. **Database Design**
  - The applicant should design the database schema for managing users, complaints, and the relationships between them.
  - The design should support efficient querying, and the applicant should demonstrate their ability to handle complex SQL queries (e.g., retrieving complaints based on filters or status counts).

7. **Frontend Implementation (Bonus for React)**
  - Basic implementation of a frontend (admin panel for complaint management).
  - If possible, use *React* to build a simple interface where:
    - Users can submit complaints and track the status.
    - Administrators can manage complaints.

Reports Management:

The following reports must be implemented for admin and any additional report will carry extra points.

1. **Complaints by Status Report:**
  - This report provides a simple count of complaints grouped by their current status (e.g., Open, In Progress, Resolved, Closed).
    - Example output:
    - Open: 10
    - In Progress: 5
    - Resolved: 20
    - Closed: 15

2. **Complaints by Priority Report:**
  - This report lists the number of complaints grouped by their priority level (Low, Medium, High).
  - Example output:
    - Low: 12
    - Medium: 8
    - High: 5

3. **Average Resolution Time Report:**
  - This report calculates the average time taken to resolve complaints, grouped by complaint category (e.g., Billing, Service Issue, Product Issue).
    - Example output:
    - Billing: 3 days

  - Service Issue: 1.5 days
  - Product Issue: 5 days
- To compute this, the system needs to track the submission date and resolution date of each complaint and then perform SQL calculations to get the averages.

4. **Complaints Trend Over Time:**
- This report provides a trend analysis of complaints over time, showing how many complaints were submitted, resolved, and closed over a selected period (e.g., last month, quarter, year). It could show daily/weekly/monthly breakdowns and could be visualized as a line chart.
- Example output:
  - March 2024: 50 complaints submitted, 40 resolved, 35 closed
  - April 2024: 60 complaints submitted, 55 resolved, 50 closed
  - May 2024: 45 complaints submitted, 40 resolved, 38 closed

These reports will give the admin insight into both the current state of complaints and more detailed performance and trend metrics over time.

** Submission of test data will carry some points.*

Bonus Feature: Notification System

Implement a notification system to keep both users and administrators informed of important events in the Complaint Ticket Management System.

User Notifications:
- **Email Notifications:** Users should receive an email notification when:
  - A complaint is successfully submitted.
  - The status of their complaint is updated (e.g., "In Progress," "Resolved," "Closed").
- **In-App Notification:** Logged Users get a count of unread notifications. Clicking on the count list of notifications should show details of each notification mentioning its time of occurrence.

Admin Notifications:
- **Email Notifications:** Administrators should receive an email notification when:
  - A new complaint is submitted.
  - A complaint's priority is marked as "High."
- **In-App Notification:** The admin gets a count of unread notifications real-time. Clicking on the count list of notifications should show details of each notification mentioning its time of occurrence.

This feature would enhance the user experience by keeping both parties informed about the state of the complaints in real-time.

** Use Laravel's built-in *Notification* system (via Mail, or if needed, push notifications using services like Pusher or Firebase).

Technical Requirements:

- Use *Laravel* (v9 or higher) for backend development.
- Database: Use *MySQL* or *PostgreSQL*.
- API endpoints should follow RESTful principles and include proper error handling and validation.
- Use *Eloquent ORM* for database interactions but also demonstrate the ability to write *custom SQL queries* when-
    - Admin dynamic report on monthly performance
    - Report on average ticket resolution time
    - categorized complaint and resolution stats

are generated.
- The project must be well-structured and follow best practices for Laravel development (MVC architecture, service layers, etc.).
- Use *Laravel Passport* or *Sanctum* for API authentication.
- Unit testing or feature testing of critical parts is encouraged. (Written test cases will qualify for additional points)

Evaluation Criteria:

1. **Code Quality:** Clean, maintainable, and well-documented code.
2. **Laravel Proficiency:** Demonstrating good use of Laravel's features such as Eloquent, Middleware, Request Validation, and Authentication.
3. **API Design:** Well-structured API with appropriate endpoints, authentication, and response formats.
4. **Database Design:** Efficient schema design with correct relationships and indexes. Use of optimized view for specific scenarios (e.g., report generation).
5. **Frontend Skills:** Bonus points for building a React interface for user and admin functionalities.
6. **Completion and Depth:** The ability to complete the required features in the given time while demonstrating deep understanding of Laravel and API development.

Submission Guidelines:

- Applicants should submit the project on GitHub or any other repository hosting platform.
- Include instructions for setting up and running the application.
- The project should include a `README` file explaining the features implemented, any special configurations, and any additional points the applicant wants to highlight.

Timeframe:

The assignment should take no longer than **6 (six) days** to complete. [Feel free to communicate beforehand if you have a different opinion]

Sample API Endpoints (for reference):

- `POST /api/complaint/create` (Create a new complaint)
- `GET /api/complaints` (List all complaints for a user)
- `GET /api/complaints/{id}` (View a specific complaint)
- `PATCH /api/complaints/{id}` (Update complaint status - admin only)

This assessment is expected to give the hiring team a clear view of the applicant's experience and skills with Laravel, API development, database management, and capabilities of interpreting business requirements.