Student Management System Documentation..

1. PROJECT OVER VIEW

The Student Management System (SMS) is designed to streamline student data management in educational institutions. Built with Laravel, this system allows administrators to:

- Manage student information
- View student records
- Manage courses
- Generate reports

This system is divided into multiple phases, each with specific deliverables to ensure a systematic approach to project completion.

2. Project Pharses

Phase 1: Requirements Gathering…

- Objective: Define the project's requirements with input from stakeholders (e.g. school administrators).
- Deliverables: Requirements document listing functionalities like student registration, course management, reporting.

Phase 2: System set up

- Objective: We set up the development environment for Laravel and any supporting tools.
- Deliverables:
- Install Laravel and necessary packages.
- Set up a MySQL database.
- Configure environment variables.

Phase 3: Functional Development

•Objective: We developed the main functionalities based on the requirements.

•Modules that we developed:

1. Student Management: Add, update, delete, and view student records.

2. Subject Management: Add, update, delete, and assign courses.

3. Reporting: Generate reports on student performance

Phase 4: Testing and Debugging

•Objective: Perform unit testing and fix bugs in the system.

•Deliverables: Document listing any bugs found and fixes applied.

Phase 5: Final Deployment

•Objective: We deployed the system on a server.

•Deliverables: We did the deployment instructions and accessed the link to the live system.

3. System Setup Instructions

Step 1: We first Installed Laravel,

Before running the project we had to first be sure of the php and composer.

•Make sure PHP and Composer are installed.

•Run the following command in the terminal:

Step 2:Data base set up: The following was done under the database setup…

•Created a MySQL database for the project.

•Configured the .env file with our database details:

DB_CONNECTION=my sql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=database name

DB_USERNAME=_username

DB_PASSWORD=_password

Step 3: Migrations and Models

•We Created migrations for students, subjects, and other necessary tables:

php artisan make:migration create_students_table

php artisan make:migration create_courses_table

•After we had to run migrations to set up the database tables.

(Using the php artisan migrate)

4.Controller and route set up•We created controllers for each module (e.g., studentController, subjectController).

•Define routes in routes/web.php for each functionality:

Route::resource('students', StudentController::class);

Route::resource('courses', CourseController::class);

Step 5: Developed Frontend

•Use Blade templates to create views for each feature:

•Student registration form

•Report generation page

Step 6: Testing

•We used Laravel's built-in testing suite to write unit tests.

•Run tests:

php artisan test

ENTITY RELATIONSHIPS

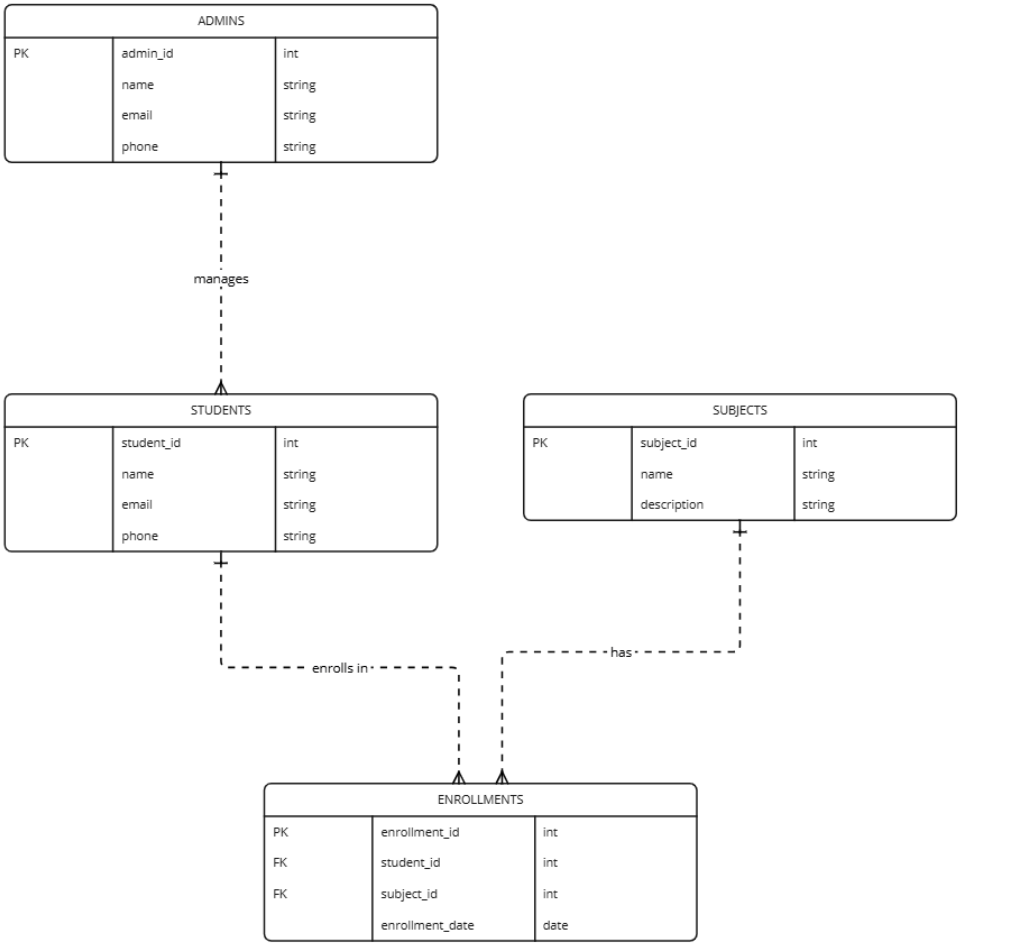
Here's how the entities `Students`, `Subjects`, `Enrollments`, and `Admins` could be related in a database for a student management system:

1. **Students**
    - **Attributes**: `student_id` (Primary Key), `name`, `email`, `phone`, etc.
    - Relationships:
        - **Enrollments**: A `Student` can have multiple `Enrollments`.
        - **Admins**: An `Admin` oversees or manages multiple `Students`.
2. **Subjects**
    - **Attributes**: `subject_id` (Primary Key), `name`, `description`, etc.
    - **Relationships**:
        - **Enrollments**: A `Subject` can have multiple `Enrollments`.
3. **Enrollments** (Join Table)
    - **Attributes**: `enrollment_id` (Primary Key), `student_id` (Foreign Key), `subject_id` (Foreign Key), `enrollment_date`, etc.
    - **Relationships**:
        - Connects **Students** and **Subjects** in a many-to-many relationship.
        - Each `Enrollment` links a `Student` to a `Subject` to represent the subjects a student is enrolled in.
4. **Admins**
    - **Attributes**: `admin_id` (Primary Key), `name`, `email`, `phone`, etc.
    - **Relationships**:
        - **Students**: An `Admin` manages or oversees multiple `Students`.

ER Diagram Description:

- **Students ↔ Enrollments ↔ Subjects**: Many-to-many relationship between `Students` and `Subjects` through the `Enrollments` table.
- **Admins ↔ Students**: One-to-many relationship, where each `Admin` manages multiple `Students`.

This structure allows you to efficiently manage and track student enrollment across subjects with admin oversight. Let me know if you'd like a visual representation



4. Key Functionalities

Student Management

- Create: Form to register new students.
- Read: View list of students, with options to filter and search.
- Update: Edit student information.

- Delete: Remove student records.

Subject Management

- **Add Subject: Interface to create and assign courses.**
- **Update Courses: Modify course details.**
- **Delete Subject: Option to remove courses no longer offered.**

Reporting

- **Performance Reports: Create reports based on student grades.**

5. Deployment Instructions

1. **Prepared the Server: Ensured the server had PHP, MySQL, and Composer installed.**

server.

3. **Set Up Environment:**

- **Copy .env file and configure it with server-specific database details.**

6.

Final Thoughts…..

This Student Management System provides a robust platform for handling student data efficiently. Through careful planning and phased development, the system will ensure reliable data management and a user-friendly experience for educational administrators.

This documentation should serve as a guide through the project, from setup to deployment, helping maintain clarity and consistency across all phases.