DDM Assignment — Galaxy Course Platform

Project: Web-based Course Registration System

1. Introduction

Web-based applications are widely used to collect and manage user data. This project provides a minimal but complete example of a typical CRUD (Create, Read, Update, Delete) system for course registrations.

2. Abstract

This project implements a web-based course registration platform named **Galaxy Course Platform**. The system allows prospective students to register for courses, stores their registration details in a relational database, and provides an admin dashboard for viewing, editing, and deleting student records. The system demonstrates practical concepts in web application design, front-end and back-end implementation, database design, and basic security and usability considerations.

3. Characteristics of a Web-based System (in this project)

-
- **Persistence:** Data stored in a relational database (MySQL).
- **Platform Independence:** Works on any modern browser and OS with PHP+MySQL support.
- **Scalability (basic):** Can be scaled horizontally using a load balancer and a central DB.
- **Security Considerations:** Uses prepared statements to prevent SQL injection; future improvements include CSRF protection and input validation on server-side.

4. Technology Used

- **Frontend:** HTML5, CSS3, JavaScript (vanilla) — for form UI, client-side validation, and AJAX.

- **Backend:** PHP (procedural) — processes form submissions and serves the admin dashboard.
- **Database:** MySQL / MariaDB — stores registrations.
- **Server:** LAMP stack (Linux, Apache, MySQL, PHP) — recommended for deployment.

5. Front-end Design

Pages

- `index.php` — Registration page with form and client-side JS to submit via `fetch()`

- `admin.php` — Admin dashboard listing all students; supports edit and delete operations.

6. Design Considerations

Security

- **Prepared statements** (parameterized queries) to prevent SQL injection.
- **Escape output** using `htmlspecialchars()` to prevent XSS in rendered pages.
- **Server-side validation** for all input data (type checks, length limits, allowed values).
- **CSRF protection:** Add CSRF tokens to forms as a next step.
- **Rate limiting & logging** for abuse prevention (future improvement).

Performance & Scalability

- Use indexes on frequent query columns (e.g., `email`, `course_type`) if table grows large.
- For many concurrent writes, consider read-replicas, caching (Redis), or connection pooling.

Maintainability

- Separate concerns: move database logic to a dedicated file (e.g., `db.php`) and use functions.
- Consider using a lightweight framework (Slim, Laravel) for larger systems.

Usability

- Provide clear success/error messages and preserve user input on failure.
- Use form validation feedback inline.

7. Backend — Database Design Structure

Entity: `registrations`

Description: Stores student registrations.

Fields:

- `id` INT PRIMARY KEY AUTO_INCREMENT
- `full_name` VARCHAR(255) NOT NULL
- `age` INT NOT NULL
- `gender` ENUM('male','female','other') NOT NULL
- `email` VARCHAR(255) NOT NULL UNIQUE
- `phone_number` VARCHAR(30) NOT NULL
- `course_type` VARCHAR(50) NOT NULL
- `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP

SQL: Create Table

```
CREATE TABLE registrations (
  id INT AUTO_INCREMENT PRIMARY KEY,
  full_name VARCHAR(255) NOT NULL,
  age INT NOT NULL,
  gender ENUM('male', 'female', 'other') NOT NULL,
  email VARCHAR(255) NOT NULL,
  phone_number VARCHAR(30) NOT NULL,
  course_type VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(email)
);
```

Example Records

```
INSERT INTO registrations (full_name, age, gender, email, phone_number, course_type)
VALUES
  ('Anita Kumar', 22, 'female', 'anita@example.com', '9876543210', 'web'),
  ('Ravi Singh', 28, 'male', 'ravi@example.com', '9123456780', 'ai');
```

Typical Queries

- Get all students: `SELECT * FROM registrations ORDER BY full_name ASC;`
- Get count: `SELECT COUNT(*) FROM registrations;`
- Update record: `UPDATE registrations SET full_name=?, age=?, ... WHERE id=?;`
- Delete record: `DELETE FROM registrations WHERE id=?;`

8. Example Code (Simplified & Improved)

> **Note:** This is a cleaned and slightly improved version of the PHP you provided. It focuses on structure, security, and comments to help with the assignment.

`db.php` — central database connection (recommended)

```
<?php
// db.php
$DB_HOST = 'localhost';
$DB_USER = 'root';
$DB_PASS = '';
$DB_NAME = 'newRegiz';

mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
try {
    $conn = new mysqli($DB_HOST, $DB_USER, $DB_PASS, $DB_NAME);
    $conn->set_charset('utf8mb4');
} catch (Exception $e) {
    // Log error in real application
    http_response_code(500);
    echo json_encode(['status' => 'error', 'message' => 'Database connection failed.']);
    exit;
}
```

`admin.php` — insert handler (AJAX) and admin dashboard (simplified)

```
<?php
require_once 'db.php';
header('Content-Type: application/json');

// If AJAX insert
if ($_SERVER['REQUEST_METHOD'] === 'POST' && !isset($_POST['action'])) {
    // Basic server-side validation
    $full_name = trim($_POST['fullName'] ?? '');
    $age = intval($_POST['age'] ?? 0);
    $gender = $_POST['gender'] ?? '';
    $email = trim($_POST['email'] ?? '');
    $phone = trim($_POST['phoneNumber'] ?? '');
    $course = trim($_POST['courseType'] ?? '');

    // Validate required fields
    if ($full_name === '' || $age <= 0 || $gender === '' || $email === '' || $phone === '' ||
$course === '') {
        echo json_encode(['status' => 'error', 'message' => 'Please fill all required fields.']);
        exit;
    }

    $sql = "INSERT INTO registrations (full_name, age, gender, email, phone_number,
course_type)
            VALUES (?, ?, ?, ?, ?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param('sissss', $full_name, $age, $gender, $email, $phone, $course);

    try {
        $stmt->execute();
        echo json_encode(['status' => 'success']);
    } catch (Exception $e) {
        // Duplicate email example handling
        echo json_encode(['status' => 'error', 'message' => $e->getMessage()]);
    }
    exit;
}

// For admin dashboard (GET request), return HTML (not shown here in snippet)
```

`index.php` — front-end (shortened)

```
<!-- index.php (form) -->
<form id="registrationForm">
  <input name="fullName" required />
  <input name="age" type="number" required />
```

```
  <select name="gender" required></select>
  <input name="email" type="email" required />
  <input name="phoneNumber" required />
  <select name="courseType" required></select>
  <button type="submit">Register</button>
</form>
<script>
  document.getElementById('registrationForm').addEventListener('submit', function(e) {
    e.preventDefault();
    const formData = new FormData(this);
    fetch('admin.php', {method: 'POST', body: formData})
      .then(r => r.json())
      .then(data => { console.log(data); /* show messages */ })
  });
</script>
```

9. Example ER Diagram (textual)

```
[registrations]
  id (PK)
  full_name
  age
  gender
  email
  phone_number
  course_type
  created_at
```

For a small application a single table suffices; for more complex systems split `courses` into its own table and use foreign keys.

10. Conclusion

This assignment demonstrates the building blocks of a web-based registration system: a usable front end, a secure back end, and a normalized database design.