

1. Task: Build a Pagination System with SQL Query Optimization

Objective: Implement a dynamic pagination system for a large dataset (e.g., 100,000+ records).

- **Details:**
 - Assume a `products` table with columns `id`, `name`, `price`, `category_id`.
 - Retrieve and display products with pagination, showing 20 products per page.
 - Optimize the query to ensure the page loads fast even with a large number of records.
 - Implement a search filter to narrow down products based on category and price range.
 - Include navigation links (Previous, Next, Page numbers) for easy page navigation.

Skills tested:

- Complex SQL queries with `LIMIT` and `OFFSET` for pagination.
 - Query optimization techniques (e.g., indexing, efficient pagination).
 - Handling large datasets and performance tuning.
 - Dynamic search and filtering.
-

2. Task: Create a Multi-Step Form with Data Persistence

Objective: Build a multi-step form where data is saved and carried over across multiple steps.

- **Details:**
 - Create a form that has multiple steps (e.g., Step 1: Personal Details, Step 2: Address, Step 3: Payment Details).
 - Use PHP to store the form data across steps using **sessions** or **database**.
 - Ensure that the form is not submitted until all steps are completed.
 - Validate form data at each step and show appropriate error messages for missing or incorrect data.
 - After all steps are completed, display the summary of the data entered before final submission.

Skills tested:

- Managing session or temporary data storage.
 - Complex form validation and persistence.
 - Multi-step form creation.
 - Session handling or temporary database usage.
-

3. Task: Create a Secure File Upload System

Objective: Build a file upload system that ensures secure file handling.

- **Details:**
 - Allow users to upload files (images, documents, etc.) via a form.
 - Implement validation to ensure the uploaded file is of a safe type (e.g., JPEG, PNG, PDF).
 - Ensure the file name is sanitized and renamed to prevent filename conflicts or malicious file names.
 - Limit the file size to 5MB.
 - Store the file in a directory, ensuring that it is not accessible directly (e.g., use `.htaccess` to block direct access).
 - Implement error handling for common issues like invalid file type, size, and permission issues.

Skills tested:

- Advanced file handling in PHP (`$_FILES`).
 - File security and validation.
 - Preventing malicious file uploads and overwriting issues.
 - Using `.htaccess` or other methods to restrict file access.
-

4. Task: Implement a Simple Content Management System (CMS)

Objective: Build a basic CMS that allows users to add, edit, and delete blog posts.

- **Details:**
 - Implement a `posts` table with columns `id`, `title`, `content`, `created_at`.
 - Create a simple admin interface where an authenticated user can:
 - Add new blog posts.
 - Edit existing posts.
 - Delete posts.
 - Use PHP sessions to manage user authentication (with login functionality).
 - Implement basic text sanitization (e.g., `strip_tags()`) to prevent XSS attacks.
 - Use `prepared statements` to prevent SQL injection.

Skills tested:

- CRUD operations (Create, Read, Update, Delete).
 - User authentication and session management.
 - Security (SQL injection prevention, XSS prevention).
 - Designing a basic CMS-like structure.
-

5. Task: Build a Simple RESTful API for Managing Users

Objective: Create a RESTful API that allows clients to interact with user data.

- **Details:**

- Create a `users` table with fields `id`, `name`, `email`, `password`.
- Implement the following API endpoints:
 - `GET /api/users`: Fetch all users.
 - `GET /api/users/{id}`: Fetch a single user by ID.
 - `POST /api/users`: Create a new user (with validation for name, email format, and password strength).
 - `PUT /api/users/{id}`: Update a user by ID.
 - `DELETE /api/users/{id}`: Delete a user by ID.
- Return responses in JSON format.
- Implement basic error handling and validation (e.g., 404 Not Found, 400 Bad Request).
- Implement `Authorization` using a simple API token (e.g., check for a valid token before allowing access).

Skills tested:

- RESTful API design and implementation.
 - Working with JSON responses and HTTP methods (GET, POST, PUT, DELETE).
 - User authentication and authorization using tokens.
 - Error handling in API responses.
 - Using `prepared statements` to prevent SQL injection.
-

6. Task: Implement a Simple Chat System with PHP and Sessions

Objective: Build a basic chat system where users can send and receive messages in real-time.

- **Details:**

- Create a `messages` table with columns `id`, `user_id`, `message`, `created_at`.
- Implement a form where users can enter their name (to identify themselves in the chat).
- Use PHP sessions to store the user's name.
- Allow users to send messages which are stored in the database.
- Implement a system to display the latest messages from the database in real-time.
- Use AJAX (JavaScript) to fetch and display new messages without refreshing the page.

Skills tested:

- Working with sessions for user identification.
 - Real-time data display using AJAX.
 - Database interaction (CRUD operations).
 - Basic chat system implementation.
-

7. Task: Implement an Authentication System with Password Reset

Objective: Implement a user authentication system with the ability to reset passwords.

- **Details:**
 - Create a `users` table with columns `id`, `email`, `password`, `reset_token`, `reset_token_expiry`.
 - Allow users to register with their email and password (use `password_hash()` for secure password storage).
 - Allow users to log in using their email and password (use `password_verify()`).
 - Implement a password reset feature:
 - The user submits their email.
 - Send a reset token to their email with a link to reset their password.
 - The token should expire after 1 hour.
 - Allow users to reset their password if the token is valid and not expired.

Skills tested:

- Secure password handling (`password_hash()` and `password_verify()`).
 - Implementing a password reset flow.
 - Sending emails via PHP (without using external libraries).
 - Token-based authentication and expiration handling.
-

8. Task: Build a Pagination System with Dynamic Filtering

Objective: Implement a complex pagination system that supports filtering and sorting.

- **Details:**
 - Create a `products` table with `id`, `name`, `category`, `price`, `created_at`.
 - Retrieve and display products with pagination, showing 10 products per page.
 - Allow users to filter products by `category` and `price range`.
 - Allow sorting by price (ascending or descending).
 - Implement SQL query logic to handle pagination, filtering, and sorting in a single query.

Skills tested:

- Complex SQL queries (pagination, filtering, and sorting).
 - Efficient data retrieval and presentation.
 - User interaction and filtering with form inputs.
 - Handling dynamic query parameters in PHP.
-

9. Task: Build a Custom Error Logging System

Objective: Create an error logging system that captures errors and saves them in a log file.

- **Details:**

- Write a custom error handler function that logs errors to a file (`error_log.txt`).
- Include the error message, timestamp, and file/line where the error occurred.
- Implement error handling for common PHP errors like `E_NOTICE`, `E_WARNING`, and `E_ERROR`.
- Make sure the log file is rotated after it reaches a certain size (e.g., 1MB).

Skills tested:

- Custom error handling in PHP (`set_error_handler()`).
 - Writing and appending to log files.
 - File size management and log rotation.
 - Handling different types of PHP errors.
-

10. Task: Build a Session-Based Shopping Cart System

Objective: Create a shopping cart that allows users to add, update, and remove items using PHP sessions.

- **Details:**

- Create a `products` table with `id`, `name`, `price`, `quantity`.
- Allow users to add products to a shopping cart (stored in `$_SESSION`).
- Allow users to update the quantity or remove products from the cart.
- Display the cart contents, including the total price.
- Implement a checkout system (simple, no payment gateway, just simulating a successful order).

Skills tested:

- Session management for storing cart data.
- Handling form submissions for cart updates.
- Displaying dynamic data and calculating totals.
- Simple checkout flow.