Simple User Management System — Advanced Test

Estimated time: ~60–120 minutes (No time constraints, this is just a estimated time of completion)
Stack: Vanilla **HTML/CSS/JS** + **PHP** (no frameworks)
Database: **MySQL** via PDO prepared statements

Goal

Build a small user management system implementing the four CRUD operations (Create, Read, Update, Delete) for a basic `users` table. We'll provide a `users_table.sql` file for you to import. Use a few sample records of your choice. When you're done, push your project to **GitHub** and share the link with **development@702pros.com**.

> **Environment**: Feel free to use **XAMPP**, **MAMP/WAMP**, **Docker**, or PHP's built-in server—whatever you prefer. All that matters is that we can run your files locally and it works as described.
>
> **Tools**: AI assistance is allowed for research and boilerplate. Avoid overreliance—the architecture, decisions, and code quality should be your own.

Pages to Build (3)

1) View (List) — `users.php`

- Display all users in a table: **First Name, Last Name, Email, Phone, Created At**.
- Include a **text search** that matches first name, last name, or email (case-insensitive).
- Each row links to the **Edit** page.
- Provide a button/link to the **Create** page.

2) Create — `user_create.php`

- Form fields: `first_name`, *`last_name`*, `email`*, `phone` (optional).
- **Server-side validation** (see rules below).
- On success, **redirect** back to the View page with a success message.

3) Edit (and Delete) — `user_edit.php?id={id}`

- Pre-fill the form with the existing user data.
- **Update** on submit (with server-side validation).
- Include a **Delete** button (submit via **POST**) and a simple `confirm()` prompt.
- On successful update or delete, redirect to the View page with a flash message.

Use the **POST/Redirect/GET** pattern to avoid accidental resubmissions on refresh.

Validation Rules (server-side)

- **first_name, last_name:** required, 1–100 chars; letters, spaces, and hyphens allowed.
- **email:** required, valid format, **unique** (case-insensitive).
- **phone:** optional; if provided, allow digits, spaces, `-`, `(`, `)`, `+`.
- Return clear, field-level error messages and preserve entered values on validation errors.

Tech Requirements

- **PDO** with prepared statements (no SQL string interpolation).
- **CSRF protection**: hidden token in forms and server-side check.

- **Output escaping**: HTML-escape any user content when rendering.
- **Sessions for flash messages** (success/error after redirects).
- **Clean file layout**, use as an example, we encourage you to try a more scalable file structure of your choosing:

- `/users.php` (list + search)
- `/user_create.php` (create)
- `/user_edit.php` (edit + delete)
- `/db.php` (PDO connection + session_start)
- `/users_table.sql` (provided)
- `/styles.css`
- `/README.md`

UX Expectations

- Clear, labeled, keyboard-friendly forms.
- Inline validation errors plus a top-level “flash” message after redirects (JS Alerts are fine to use).
- Confirm before delete (simple JS `confirm()` is fine).
- After searching, the query remains visible in the input.

Acceptance Checklist

- Full CRUD: create, list, edit, and delete users.
- **Email uniqueness** enforced with a friendly error state.
- Server-side validation errors shown next to fields.
- **CSRF** token verified on all mutating requests.

- Flash messages after create/update/delete.
- Case-insensitive search by first/last/email works.
- All displayed user data is **HTML-escaped** (no XSS).

Evaluation Rubric (100 pts)

- **Correctness (30):** CRUD works; unique email; proper redirects; flash messages.
- **PHP Quality (25):** PDO prepared statements, CSRF handling, escaping, structure.
- **Validation & Errors (15):** robust server-side checks; helpful messages.
- **UI/UX (15):** readable, accessible forms; usable search.
- **Code Organization (10):** clear structure; minimal duplication.
- **Polish/Stretch (5):** thoughtful extras (pagination, sorting, CSV export, etc.