# User Management Application (Assessment Solution)

This project implements a comprehensive user management system following the requirements of a technical assessment. It is built using the **Flask** framework in Python, utilizing MySQL for persistence, and implementing essential features like role-based authentication, user registration with full validation, and APIs for listing, searching, and filtering users.

## 🛠️ Technology Stack

| Component | Technology | Purpose |
|---|---|---|
| **Backend Framework** | Python (Flask) | Primary framework for routing and business logic. |
| **Database** | MySQL | Data persistence for user records. |
| **Database Connector** | `flask_mysqldb` | Connects Flask to the MySQL database. |
| **Forms/Validation** | `Flask-WTF` | Handles form data and server-side input validation. |
| **Security** | `bcrypt` | Secure hashing and verification of passwords. |
| **Frontend** | Jinja2 Templates, Bootstrap 4 | Templating and responsive user interface styling. |

## 🚀 Features Implemented

The application fulfills all four core requirements of the assessment, with secure session management replacing JWT for authentication in this Flask implementation.

### 1. Login API ( `/login` )

- **Functionality:** Allows existing users to sign in.
- **Fields:** Email and Password.
- **Security:** Passwords are verified against stored hashes using `bcrypt` .
- **Session Management:** Upon successful login, the `user_id` and `user_role` are stored in the Flask session to manage authentication and authorization across the application.

### 2. Registration API ( `/register` )

- **Functionality:** Allows new users to create an account.
- **Fields:** Name, Email, Password, **Role (Admin/Staff), Phone, City, Country.**
- **Validation:** Includes server-side validation for:
    - All fields are required ( `DataRequired` ).

- Email format validity and uniqueness check.
- Role validation (ensures input is strictly 'Admin' or 'Staff').

- **Security:** Passwords are hashed using `bcrypt` before storage.

### 3. List Users API ( `/users` )

This is a protected route with advanced querying capabilities.

- **Authentication & Authorization:**
    - Requires a logged-in session.
    - **Access Restricted:** Only users with the `Admin` role can access this list. Non-Admin users are redirected.
- **Querying:** The endpoint dynamically handles two query parameters:
    - **Search Users:** Allows searching by **Name** or **Email** (using SQL `LIKE` for partial matches).
    - **Filter Users:** Allows filtering the list by the **Country** field.
- **Implementation:** Dynamic SQL is constructed based on the presence of `search` and `country` parameters.

### 4. User Details API ( `/users/<user_id>` )

Retrieves and displays the full registration details for a specific user ID.

- **Role-Based Access Control (RBAC):**
    - **Admin Role:** Can view the details of **any** user ID.
    - **Staff/Other Role:** Can **only** view their own registration details (i.e., `user_id` in URL must match the `user_id` in the session).
- **Implementation:** The route checks the `user_id` in the URL against the session's `user_id` and `user_role` before executing the database query.

## 💾 Database Schema

The core structure relies on a single `users` table in MySQL. The table must include the following required columns to satisfy the registration and filtering requirements.

**Table:** `users`

| Column | Type | Attributes | Purpose |
|---|---|---|---|
| `id` | `INT` | `PK`, `AUTO_INCREMENT` | Unique identifier. |
| `name` | `VARCHAR(75)` | | User's full name. |
| `email` | `VARCHAR(75)` | | Login credential, required to be unique. |

| | | | |
|---|---|---|---|
| `password` | `VARCHAR(150)` | | Stored password hash ( `bcrypt` ). |
| `role` | `VARCHAR(50)` | `NOT NULL` | **Critical for authorization (Admin/Staff).** |
| `phone` | `VARCHAR(20)` | | User's contact number. |
| `city` | `VARCHAR(100)` | | User's city. |
| `country` | `VARCHAR(100)` | | **Used for the List Users API filter.** |

### SQL Command to Create/Alter Table

If your table is missing the required columns, run this SQL command:

```
-- Use this command to add the missing columns to an existing table
ALTER TABLE users
ADD role VARCHAR(50) NOT NULL,
ADD phone VARCHAR(20),
ADD city VARCHAR(100),
ADD country VARCHAR(100);

-- Or use this full CREATE TABLE statement if starting from scratch
/*
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(75) NOT NULL,
    email VARCHAR(75) UNIQUE NOT NULL,
    password VARCHAR(150) NOT NULL,
    role VARCHAR(50) NOT NULL,
    phone VARCHAR(20),
    city VARCHAR(100),
    country VARCHAR(100)
);
*/
```

## ⚙️ Setup and Installation

### Prerequisites

1. Python 3.x
2. MySQL Server
3. A virtual environment (recommended)

### Installation Steps

1. **Clone the Repository (Hypothetical):**

```
git clone [repository-url]
cd user-management-app
```

2. **Install Dependencies:**

```
pip install Flask Flask-WTF Flask-Bcrypt Flask-MySQLdb
```

3. **Database Configuration:**
   - Ensure your MySQL server is running.
   - Update the configuration in `app.py` with your credentials:

   ```
   app.config['MYSQL_HOST'] = 'localhost'
   app.config['MYSQL_USER'] = 'app_user'
   app.config['MYSQL_PASSWORD'] = 'strong_password'
   app.config['MYSQL_DB'] = 'mydatabase'
   ```

   - Execute the necessary SQL command (provided above) to ensure the `users` table has all eight columns.
4. **Run the Application:**

```
python app.py
```

5. **Access the Application:** Open your web browser and navigate to `http://127.0.0.1:5000/` .

## 🧪 Testing Scenarios

To fully test the role-based security features:

1. **Register a User with Role:** `Admin`
   - Log in as this user.
   - Verify you can see the **"View All Users (Admin)"** link on the Dashboard.
   - Navigate to `/users` and verify you can see the list, search, and filter.
   - Verify you can click any user's name to view their details.
2. **Register a Second User with Role:** `Staff`
   - Log in as this Staff user.
   - Verify you **cannot** see the **"View All Users (Admin)"** link.
   - Attempt to navigate directly to `/users` and verify you are blocked/redirected.

- Click your own name on the Dashboard and verify you can see your own details ( `/users/<staff_id>` ).
- Attempt to access the Admin user's details ( `/users/<admin_id>` ) and verify you are blocked.