Step-by-Step Guide for Building a User Management System

1. Define Your Goal

Understand the core functionalities of the application:

- Add new users.
- Edit existing user details.
- Delete users with role-based restrictions.
- Validate user inputs.
- Dynamically render the user list.

---

2. Plan the Structure

Break the application into smaller components:

1. **Data Storage**: Use an array to store user details and a few additional properties for editing state.
2. **Methods**: Define functions for handling user forms, validating inputs, adding, editing, deleting users, and rendering the user list.
3. **Event Handling**: Handle interactions like form submissions, edit actions, and delete confirmations.

---

3. Start Writing Code

Begin with the basics and build incrementally:

A. Set Up a User Management Object

Create a JavaScript object (`userManagement`) to manage users and their associated actions:

- Properties:
    - `users`: Array to store user objects.
    - `isEditing`: Boolean to track editing state.
    - `editingEmail`: String to identify the user being edited.
- Methods:
    - `handleUserForm()`
    - `addUser()`
    - `updateUser()`
    - `deleteUser()`

    - `editUser()`
    - `validateEmail()`
    - `resetForm()`
    - `renderUsers()`

B. Implement Core Methods

Write the methods in a modular way:

- `handleUserForm`: Determine whether to add or update a user based on the editing state.
- `addUser`: Validate email and add a new user with details like name, email, role, and preferences.
- `updateUser`: Update the details of an existing user.
- `deleteUser`: Remove a user from the list with role-based validation.
- `editUser`: Populate the form with existing user details for editing.
- `validateEmail`: Check if the email format is valid using a regex.
- `resetForm`: Clear the form and reset editing state.
- `renderUsers`: Dynamically update the DOM to display the user list.

C. Attach Event Listeners

Handle the interactions:

- **Form Submission**: Use `addEventListener` on the form to trigger `handleUserForm` for adding or updating users.
- **Edit and Delete Buttons**: Dynamically generate buttons and attach handlers for editing and deleting users.

D. Test and Debug

- Test each method individually in the browser console.
- Validate inputs for edge cases (e.g., invalid email, empty fields).

---

4. Order of Implementation

Follow this sequence:

Initialize the User Management Object:

const userManagement = { users: [], isEditing: false, editingEmail: null };

1.

2. **Add Core Methods to the Object**:
   - `handleUserForm()`
   - `addUser()`
   - `updateUser()`
   - `deleteUser()`
   - `editUser()`
   - `validateEmail()`
   - `resetForm()`
   - `renderUsers()`
3. **Create Event Handlers**:
   - Write the logic for form submission and attach it to the "Submit" button.
   - Write the logic for edit and delete actions and attach them to dynamically generated buttons.
4. **DOM Manipulation**:
   - Use `document.createElement` and `appendChild` to render users.
   - Dynamically update the DOM for user list and form actions.
5. **Test the Flow**:
   - Add users.
   - Edit and update users.
   - Delete users with role validation.
   - Ensure the DOM updates correctly.

---

5. Add Features Incrementally

Once the basics work, enhance the application:

- **Role-Based Permissions**: Restrict actions like delete based on user roles.
- **User Preferences**: Allow users to set preferences and store them dynamically.
- **Styling**: Apply CSS classes for better UI and user experience.
- **Validation**: Ensure fields are filled correctly before submission.

---

6. Checklist for Completion

- Users are added correctly with unique IDs and valid email addresses.
- User details can be edited and updated.
- Users can be deleted with role-based restrictions.

- Form validation ensures proper input.
- The user list renders dynamically and updates after actions.

By following this roadmap, you will systematically build the User Management System with clarity and focus.