⚠️ Automation Considerations and Limitations

During the development of the automated test suite, the test cases were aligned with the manual scenarios to ensure consistency. However, due to specific platform constraints, some adjustments were made:

🔒 Email Uniqueness

- The system enforces **strict uniqueness on email addresses** across all households.
- To prevent conflicts and test failures due to repeated executions, email fields in test cases should be **dynamically generated** (e.g., using timestamps or random strings).
- For this reason, in the automated scripts, **only one static email per scenario was used**, and tests are **not designed to run multiple times without modification**.

🗑️ Soft Delete Behavior

- Deleting a household member does not remove the user permanently from the database (**soft delete** is applied).
- As a result, re-adding the same member with the same email after deletion is not possible, since the email remains reserved.
- Therefore, automated tests for deletion were designed to **validate UI behavior** (e.g., modal confirmation and navigation) rather than verifying permanent data removal.

✅ Suggested Improvements for Future Iterations

To make the test suite more robust and repeatable:

- Implement a **random email generator** for test users (e.g., `user+{timestamp}@domain.com`).
- Consider using **API-level cleanup** or database resets in a test environment.
- Use **mocked environments** or **isolated test accounts** when possible to ensure independence and data reset between executions.

Part 2: Technical Knowledge Evaluation

Write an SQL query to find all the members of a specific household.

```
SELECT
  hu.user_id,
  hu.role,
  hu.created_at
FROM
  Household_Users hu
JOIN
  Households h ON hu.household_id = h.id
WHERE
  h.identifier = 'HOUSEHOLD_IDENTIFIER';
```

Write an SQL query to show the name and role within a household.

```
SELECT
  h.name AS household_name,
  hu.user_id,
  hu.role
FROM
  Household_Users hu
JOIN
  Households h ON hu.household_id = h.id;
```

Part 3: Practical Questions

✅ Test Case 1: Enable “Add Member” button with valid data

Test Case ID: TC_HH_001
Title: Verify that the “Add Member” button becomes enabled with valid input
Objective: Ensure the "Add Member" button is enabled when all fields contain valid data
Preconditions:

- User is logged in as Advisor
- “Add Member” modal is open

Test Steps:

1. Enter a valid email: `nuevo.miembro+test01@gmail.com`
2. Enter first name: `Juan`
3. Enter last name: `Perez`
4. Enter phone number: `3204031589`
5. Check the status of the “Add Member” button

Expected Result:
The “Add Member” button should be enabled when all fields are correctly filled.

✅ Test Case 2: “Add Member” button remains disabled with empty fields

Test Case ID: TC_HH_002
Title: Verify that the “Add Member” button stays disabled with empty inputs
Objective: Ensure the form cannot be submitted when required fields are empty
Preconditions:

- User is logged in as Advisor
- “Add Member” modal is open

Test Steps:

1. Leave all form fields empty
2. Check the status of the “Add Member” button

Expected Result:
The button must remain disabled if the required fields are not filled.

✅ Test Case 3: Show validation errors for invalid name or last name

Test Case ID: TC_HH_003
Title: Validate that errors are displayed when first name or last name contains invalid characters
Objective: Ensure that the system blocks names with numbers or special characters
Preconditions:

- User is logged in as Advisor
- “Add Member” modal is open

Test Steps:

1. Enter first name: `Juan123`
2. Enter last name: `Perez@`
3. Optionally enter valid email and phone number

Expected Result:
Error messages should be displayed for invalid first name and last name inputs.

✅ Test Case 4: Email field should be disabled when editing a member

Test Case ID: TC_HH_004
Title: Verify that the email field is disabled in the edit member modal
Objective: Ensure that the email field cannot be modified when editing an existing member
Preconditions:

- User is logged in as Advisor

- User has navigated to the household and clicked the edit icon on a member

Test Steps:

1. Click the "Edit" button for any existing household member
2. Wait for the "Edit Member" modal to appear
3. Check the state of the email field

Expected Result:
The email field should be visible but disabled (read-only) in the edit form.

✅ Test Case 5: Save Changes button should be disabled initially

Test Case ID: TC_HH_005
Title: Verify that the "Save Changes" button is initially disabled
Objective: Ensure the system prevents updates if no changes are made in the form
Preconditions:

- User is logged in as Advisor
- Edit Member modal is open

Test Steps:

1. Click the "Edit" button on a household member
2. Wait for the "Edit Member" modal to open
3. Verify the state of the "Save Changes" button without modifying any field

Expected Result:
The "Save Changes" button should remain disabled until a valid change is made in the form.

✅ Test Case 6: Save Changes button becomes enabled after a valid update

Test Case ID: TC_HH_006
Title: Verify that the "Save Changes" button becomes enabled after updating a field

Objective: Ensure the system allows saving changes only when a valid modification is made

Preconditions:

- User is logged in as Advisor
- Edit Member modal is open

Test Steps:

1. Click the "Edit" button on a household member
2. Wait for the "Edit Member" modal to appear
3. Modify the first name by adding a single character
4. Observe the state of the "Save Changes" button

Expected Result:
The "Save Changes" button should become enabled after modifying a field in the form.

✅ Test Case 7: Display confirmation modal when clicking delete

Test Case ID: TC_HH_007
Title: Verify that a confirmation modal is displayed when clicking the delete button
Objective: Ensure the system prompts the user before permanently deleting a household member
Preconditions:

- User is logged in as Advisor
- User has navigated to the household page

Test Steps:

1. Click the "Delete" icon on a household member
2. Wait for the confirmation modal to appear

Expected Result:
A confirmation modal should be displayed asking the user to confirm the deletion of the member.

✅ Test Case 8: Canceling the delete action returns the user to the household page

Test Case ID: TC_HH_008
Title: Verify that clicking "Cancel" on the confirmation modal closes the modal
Objective: Ensure the user can cancel the delete action without affecting household data
Preconditions:

- User is logged in as Advisor
- Confirmation modal is visible after clicking "Delete" on a member

Test Steps:

1. Click the "Delete" icon on a household member
2. Wait for the confirmation modal to appear
3. Click the "Cancel" button on the modal

Expected Result:
The confirmation modal should close, and the user should return to the household page without changes made.

✅ Test Case 9: User with permissions should see Add, Edit, and Delete buttons

Test Case ID: TC_HH_009
Title: Verify that a user with proper permissions sees Add, Edit, and Delete member buttons
Objective: Ensure users with the appropriate role (e.g., Advisor) can manage household members
Preconditions:

- User is logged in as Advisor
- User is on the Household page

Test Steps:

1. Log in with the Advisor account: `zoefin.advisor+qaroleadv01@gmail.com`

2. Navigate to the Household page
3. Check for visibility of the following buttons:
    - “Add Member”
    - “Edit” (on any member)
    - “Delete” (on any member)

Expected Result:
All three buttons (“Add Member”, “Edit”, and “Delete”) should be visible and accessible to the Advisor user.

✅ Test Case 10: Client user should not see Pipeline or household management options

Test Case ID: TC_HH_010
Title: Verify that a client user without permissions cannot access household management features
Objective: Ensure restricted users (Client role) cannot see or use privileged features
Preconditions:

- User is logged in as a Client
- User is on the dashboard

Test Steps:

1. Log in with a Client user account:
   `zoefin.client+qaroleclient02@gmail.com`
2. Observe the visible navigation menu items
3. Check for presence of “Dashboard” menu
4. Confirm absence of “Pipeline” menu and any household-related actions

Expected Result:
The user should only see the “Dashboard” menu. No access should be available to “Pipeline” or to household member management features (add/edit/delete).

Part 4: Case Studies

📌 Case Study 1

Scenario:
Advisors, Owners, and PMs report that they cannot add new members to households.

Steps to investigate and resolve the issue:

1. **Reproduce the problem:**
   Log in as each affected role (Advisor, Owner, PM) and attempt to add a new member to validate the issue.

2. **UI behavior check:**
   Confirm that the “Add Member” button is visible and clickable. If it’s not rendered, inspect role-based rendering conditions in the front-end code.

3. **API-level validation:**
   Use tools like Postman or automated scripts to send POST `/households/{id}/users` requests with valid bearer tokens for each role and observe the response codes or error messages.

4. **Permissions verification:**
   Ensure that these roles are correctly mapped to the action of adding members in the backend. Confirm that they are not mistakenly restricted due to misconfigured permissions.

5. **Email uniqueness constraint:**
   Check if the attempted email already exists in another household, which would violate the system constraint on unique emails.

6. **Error logs review:**
   Inspect browser console logs and backend logs for permission-related errors or validation failures.

7. **Fix and validate:**
   Based on the root cause (e.g., missing role permission, UI bug, or email conflict), apply the necessary fix and validate that the issue is resolved for all authorized roles.

📌 Case Study 2

Scenario:
Editing household members is intermittently allowed for users who already have completed investment accounts.

Steps to investigate and prevent recurrence:

1. **Clarify business rules:**
   Confirm that editing members with completed investment accounts is not allowed under any circumstance.

2. **Reproduce under multiple scenarios:**
   Test with multiple user roles and various investment statuses to determine when the restriction fails.

3. **Inspect frontend validation logic:**
   Check if the UI is correctly disabling or hiding the edit button for ineligible members. Look for timing issues or missing conditions in rendering logic.

4. **Validate backend protection:**
   Ensure that the `PUT /households/{householdId}/users/{userId}` endpoint includes strict checks to reject edit requests for members with investment accounts, regardless of UI restrictions.

5. **Review API responses:**
   If editing is incorrectly allowed, verify whether the server fails to enforce the rule or if the member's investment status is outdated or incorrectly reported.

6. **Implement improved validation and tests:**
   Enhance both client-side and server-side validation. Add regression tests to cover edge cases and prevent similar bugs in future releases.

7. **Monitor logs and usage:**
   Enable better logging around member edit attempts and track role and investment status to detect future violations early.

📌 Debugging Scenario

Problem:
Edit and Delete buttons are shown for household members who already have completed investment accounts.

Debugging process:

1. **Understand the UI logic:**
   Review the component or page code responsible for rendering the Edit and Delete buttons. These buttons should only be shown for members **without** investment accounts.

2. **Confirm data availability:**
   Ensure that the household member object includes a clear property (e.g., `hasInvestmentAccount`) that the frontend can use to conditionally render or hide these actions.

3. **Simulate multiple edge cases:**
   Log in with different roles and navigate to households with a variety of member states (active investment, no investment, deleted accounts, etc.).

4. **Verify backend enforcement:**
   Even if the frontend fails to hide the buttons, the backend should block any `PUT` or `DELETE` requests on members with completed investment accounts. Validate this protection.

5. **Test UI reactivity:**
   If investment accounts are created shortly after member creation, ensure that the UI properly refreshes the member's status in real-time or on reload.

6. **Fix UI logic and add automated tests:**
   Update the rendering logic to hide the Edit/Delete buttons based on the correct flags. Add UI tests that validate their absence for investment-linked members.

7. **Deploy, monitor, and document:**
   Deploy the fix and include monitoring to alert developers if restricted actions appear again. Update internal documentation for future reference.