Overall Evaluation Report – QA Assignment (PetClinic)

🔍 Found Issues

API Issues

Endpoint: PUT /owners/{ownerId}/pets/{petId}
Issue Description: Service is returning a 500 error.
Severity: Medium

Endpoint: POST /owners/{ownerId}/pets
Issue Description: Service is returning a 500 error.
Severity: High

Endpoint: GET /owners
Issue Description: Looks like the service is not found
Severity: Low

Endpoint: GET /owners/{ownerId}
Issue Description: Service is returning a 500 error.
Severity: Low

Endpoint: PUT /owners/
Issue Description: Service is returning a 405 error.
Severity: Low

UI Issues

Feature: Searching for an existing owner
Issue Description: Looks like the service is not found
Severity: Low

Feature: Attempting to add a pet with a future birth date
Issue Description: It is possible to create a pet with future date of birth
Severity: Medium

Scenarios marked with *@BUG* tags in the test suite represent these known issues and are expected to fail to highlight system weaknesses.

☑ Test Cases Summary (Automated)

🧪 UI Test Cases

- Add a new pet to an existing owner
- Add a new visit to a pet
- Validate pet form input fields
- Navigate through pet and owner pages

🧪 API Test Cases

- Retrieve owner details by ID
- Update existing pet with valid data
- Try updating a pet with invalid ID (@BUG scenario)
- Add a pet with valid data
- Add a pet with missing data (@BUG scenario)

These tests were selected based on **critical user flows**, **common edge cases**, and **backend validations**.

💡 Recommendations for Improvement

1. **Backend Input Validation:**
    - Ensure APIs return proper status codes and error messages for invalid input.
    - Apply stricter schema validation for required fields like name, birthDate, etc.
2. **Frontend Form Controls:**
    - Implement input field validations (e.g., required fields, proper date format).
    - Disable form submission if data is missing or malformed.
    - In the UI there should be form validation for every field to align with the API requirements.
3. **Consistent Error Handling:**
    - Align UI and API error responses.
    - Show user-friendly error messages in the UI.
4. **Improved HTML Markup:**
    - The HTML elements should contain element IDs (for better debugging and locating).
5. **Missing Functionality - Deleting:**
    - There is no delete functionality for pets, visits, or owners through the UI or API.
6. **Missing Functionality - Visit Service:**
    - Visit creation, update, and deletion operations via API seem to be incomplete or missing.

⚠️ Identified Risks

- **Lack of Options to Delete Test Data**: Once test data is introduced through the UI or API, there's no way to clean it up, which may lead to database clutter or test conflicts.
- **Lack of Authentication**: APIs are publicly accessible without authentication, which is risky for a real-world app.
- **Unvalidated Input**: Both UI and API accept invalid or incomplete data, posing risks of bad data injection.

🧪 Testing Approach

☑ Automated Testing Strategy

- **Framework**: Cucumber with Playwright (UI) and Playwright's APIRequestContext (API)
- **Language**: TypeScript
- **Execution**: GitHub Actions on push/pull requests
- **Headless**: Default mode with screenshots and traces on failure
- **Dockerized Backend**: Ensures consistent local testing environment
- **Report Generation**: HTML reports generated using cucumber-html-reporter

📝 Manual Testing Strategy

- Edge cases and exploratory tests are documented under `manual-tests-cases/`

🔍 Risks of the Current Testing Approach and Recommendations

While the current testing setup provides strong coverage and automation, there are some risks and improvement opportunities:

Risks

- **Limited Test Data Control**: The use of static data and the lack of cleanup routines can lead to unpredictable results or test conflicts.
- **No Cross-Browser Validation**: The setup is currently focused on Chromium; other browsers (Firefox, WebKit) are not tested.
- **Incomplete Coverage for Negative Paths**: Some edge cases (e.g., failed validations, missing fields) are manually documented but not fully automated.
- **UI and API Decoupling Risks**: UI validations are not strictly aligned with backend API rules, potentially causing inconsistent behaviours.

Recommendations

- Implement data seeding and cleanup scripts to manage test state consistently.
- Extend Playwright config to include cross-browser testing for broader compatibility.
- Automate more negative and edge-case scenarios, especially those marked as manual.
- Use shared validation contracts or API specs to ensure UI and API rules are aligned (e.g., via contract testing or schema validation).

🏁 Conclusion

This QA project implements a well-structured automated framework covering both UI and API. It highlights multiple issues in validation and usability that would affect real users. Combining automation with targeted manual exploration helped uncover edge case problems. With the provided recommendations and CI pipeline, this setup provides a solid foundation for ongoing test scalability and quality assurance.