ECSE 429 - A1 Report

Hamza Alfarrash

Assignment 1 Report : Exploratory Tesing of REST APIs

Student: Hamza Alfarrash (261017161)

1 Summary of Deliverables

My first task was to conduct exploratory testing of the REST API To-Do List Manager. This was carried out in a single session. During this session, I identified the system's key functionalities, noted possible weaknesses, and generated ideas for further testing. The session was executed using Postman, and detailed notes of the process are documented here.

The second task involved creating unit tests for the API. For this purpose, I selected JUnit as the testing framework. The test suite is structured into a dedicated file focusing on Todos. The file is available here.

Finally, I recorded a demonstration video showing the tests being executed in a randomized order. The video is accessible here.

2 Exploratory Testing Findings

I examined the following capability : todos. The following was the charter used for examination :

> **Exploratory Testing Charter**
>
> Identify capabilities and areas of potential instability of the "rest api todo list manager".
> Identify documented and undocumented "rest api todo list manager" capabilities.
> For each capability create a script or small program to demonstrate the capability.
> Exercise each capability identified with data typical to the intended use of the application.

2.1 Session Notes : Todos

The session notes can be found in this file.

2.1.1 Files

Session1Notes.txt

2.1.2 Session Findings

- **OPTIONS:**
  - Successfully validated documented options for both `/todos` and `/todos/:id`.
  - The provided options appear to be exhaustive.
- **GET:**
  - Todos were successfully retrieved using ID, title, description, and doneStatus.
  - Filtering by description only works with an exact match; partial keywords are not supported (suggesting inclusion-based filtering may be missing).
  - Data retrieval worked correctly in both JSON and XML formats.
- **HEAD:**
  - Headers were returned as expected in both JSON and XML for all tested requests.
- **POST:**
  - The API handled valid input properly, and rejected invalid input such as missing the mandatory title field (returned a 400 error).
  - Creation of new todos worked when valid fields were supplied.
  - Attempts to post with an existing ID did not succeed, as expected.
  - Requests with nonexistent IDs correctly produced 404 errors.
- **PUT:**
  - Sending a partial payload replaced the entire todo, with unspecified fields reset to their default values. This reset behavior is not mentioned in the documentation.
  - Attempts to update with nonexistent IDs failed appropriately.
- **DELETE:**
  - Deletion of todos by ID worked correctly.
  - Follow-up requests to retrieve deleted todos failed as expected.

2.1.3 Summary of concerns

- **Description Filtering:**
  - Partial matches in the description field are not supported. Adding substring or inclusion-based search would make the API more user-friendly.
- **PUT/POST Handling:**

  - While existing and nonexistent IDs are handled correctly, the undocumented behavior of PUT (resetting unspecified fields to defaults) could cause unintended data loss. This should either be documented clearly or modified.

- **Empty Todo Creation:**
  - Attempting to post an empty todo returns the error "title: field mandatory". If the title were not required, empty todos would be created, which would likely be a bug.

- **ID Incrementation Bug:**
  - Even when a POST fails, the internal ID counter still increments by two instead of one. This can lead to gaps in IDs and potential integrity issues.

- **Security Risk:**
  - Since the API lacks authentication, critical actions (like shutting down the app) could be triggered by anyone using a simple GET request.

2.1.4 Future Testing Ideas

- Try more edge cases for creating and deleting todos
- Check response headers in detail
- Investigate handling of non-existent todo IDs
- Look deeper into linking and unlinking tasks with todos
- Evaluate OPTIONS behavior across endpoints

3 Unit Test Suite Structure

Unit tests code source can be found here.

TodoUnitTest.java

This file is dedicated to verifying the CRUD operations (Create, Read, Update, Delete) for Todo objects. The tests confirm that the API responds with the correct status codes and that the lifecycle of created, updated, and deleted items behaves as intended.

- **Setup:** The `@BeforeAll` annotation is used to start the API server, ensuring it is running before any test cases are executed. The base URI for all HTTP requests is then configured.

- **Test Execution:**
  - With `@BeforeEach`, a fresh Todo object is generated before every test, checking that the fields (title, description, and done status) are properly initialized.

  - A series of tests verify the CRUD flow:
    * Creation with valid input confirms that correct status codes are returned and that the data in the response is accurate.
    * Update operations validate that changes to fields such as title and description are processed correctly.
    * Deletion checks confirm that existing records are removed as expected. Additional tests validate that deleting already removed items or non-existent IDs produces the appropriate error codes.

- **Error Handling:** Edge cases such as invalid input formats or attempts to remove missing entries are tested to ensure that the API returns consistent error messages and codes.

The test design ensures that the Todo API has a dedicated suite covering both functional correctness and handling of exceptional cases.

4 Source Code Repository Structure

The project's source code is hosted on Github (link) to support version control and development workflows.

4.1 Repository Structure

The repository is organized for clear development, testing, and documentation:

- **Session Notes/** – Documentation and notes from the exploratory test session (.txt file containing the session notes).
- **src/test/** – Unit test suite. Key subpaths include:
  - **src/test/java/unitTest/**
    * **TodoUnitTest.java** – Tests covering To-do functionality.
- **.gitignore** – Excludes build artifacts and environment-specific files.
- **gradle/wrapper** – Gradle configuration files
- **README.md** – Provides a general project overview, setup, and usage.

4.2 Commit Messages

Commits use concise, meaningful messages that summarize the change and its purpose to aid future maintainers.

5 Unit Test Suite Findings

During execution of the unit tests, several issues were detected across the different API endpoints. Each defect is summarized below with an explanation, its potential impact, and reproducible steps.

5.1 API Issues

Bug #1: Invalid XML in API Documentation Response

- **Summary:** The XML returned by the documentation endpoint is malformed due to a missing closing tag.
- **Details:** The payload from the `/docs` endpoint includes a `<link>` element that is not properly closed, preventing the response from being parsed as valid XML.
- **Impact:**
  - Automated tools or systems relying on well-formed XML fail when processing the response.
  - Tests and integrations that validate XML cannot execute successfully.
  - Clients depending on the API documentation in XML format may be unable to consume it.
- **Steps:**
  1. Send a GET request to `http://localhost:4567/docs`.
  2. Inspect the XML payload containing the `<link>` element.
  3. Run the response through an XML validator or parser.
  4. Confirm that an error is raised because of the missing closing tag.

5.2 Todo Issues

Bug #2: Incorrect Behavior of PUT on Todos

- **Summary:** A PUT request replaces the full todo entry rather than updating fields.
- **Details:** Although the specification states that PUT should allow updating of an existing todo, the current implementation discards unspecified fields and resets them to defaults.
- **Impact:** This behavior can overwrite useful data, producing inconsistencies and data loss.
- **Steps:**
  1. Perform a PUT request with only selected fields.

2. Observe that the entire todo is overwritten instead of being partially updated.

Bug #3: Use of POST Instead of PATCH for Updates

- **Summary:** POST is incorrectly used to modify existing todos.
- **Details:** The API uses POST requests for amendments, while PATCH would be the correct method for partial modifications.
- **Impact:** This practice introduces ambiguity and violates REST conventions, leading to potential confusion for developers.
- **Steps:**
  1. Send a POST request to update an existing todo.
  2. Notice that POST is accepted for changes even though PATCH should be used.