Assignment 1 Report : Exploratory Tesing of REST APIs

Students: Lounes Azzoun (261181741), Ahmed Mossa (261001739)

1 Summary of Deliverables

Our first task was to conduct exploratory testing of the REST API To-Do List Manager. This was carried out in two separate sessions by different testers. During these sessions, we identified the system's key functionalities, noted possible weaknesses, and generated ideas for further testing. All sessions were 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, we selected JUnit as the testing framework. The test suite is structured into two files — one focusing on Projects and the other on Todos. Both files are available here.

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

2 Exploratory Testing Findings

We examined the following capabilities : `todos` and `projects`. 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, part 1 : 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

2.2 Session Notes, part 2 : Projects

The session notes can be found in this file.

2.2.1 Files

Session2Notes.txt

2.2.2 Session Findings

- **OPTIONS:**
  - Both `/projects` and `/projects/:id` returned the expected set of supported methods.
  - The options list appeared complete.
- **GET:**
  - Successfully retrieved projects by ID, title, active status, and completed status.
  - Filtering by description required an exact match; partial keyword filtering was unsupported.
  - Project data was retrieved successfully in both JSON and XML formats.
- **HEAD:**
  - Header responses matched expectations in both JSON and XML for all tested endpoints.
- **POST:**
  - Submitting an empty project succeeded, raising questions about whether this behavior is appropriate.
  - Attempts to create a project while specifying an ID failed, as expected.
  - Projects with duplicate titles and descriptions could be created using both JSON and XML, suggesting no uniqueness validation.
  - The API allowed creating projects marked as both completed and active, which may represent a logical conflict.
- **PUT:**
  - Replacing a project with partial fields caused unspecified fields to reset to defaults, which was not documented.
  - Attempts to update nonexistent project IDs failed correctly with a 404.
- **DELETE:**
  - Deleting projects by ID worked as expected.
  - Attempts to retrieve deleted projects returned the correct 404 errors.

2.2.3 Summary of concerns

- **Description Filtering:**
  - Only exact description matches are supported. Inclusion-based filtering could provide a more user-friendly search experience.
- **Duplicate Projects:**
  - The system currently allows identical projects with the same title and description. Clarification is needed on whether duplicates should be restricted.
- **Conflicting Status Flags:**
  - The ability to create projects that are simultaneously active and completed may contradict business logic and requires further review.
- **PUT/POST Behavior:**
  - Although existing and nonexistent IDs are handled properly, the undocumented behavior of PUT (resetting unspecified fields to defaults) poses a risk of accidental data loss. This should be either clarified in documentation or adjusted in implementation.
- **Empty Project Creation:**
  - The system accepts empty projects. Whether this should be allowed is unclear, and likely problematic.

2.2.4 Future Testing Ideas

- Inclusion-Based Filtering: Create additional tests for partial description matching to confirm whether it could be supported in future.
- Duplicate Prevention: Develop test cases around project uniqueness, ensuring duplicates are either prevented or clearly documented.
- Status Logic: Investigate how status flags interact, including unusual states such as "completed and active" simultaneously.
- PUT/POST Validation: Extend testing of PUT and POST to document the reset behavior of missing fields, and evaluate PATCH as a more suitable option for partial updates.
- Empty Project Handling: Design test cases to decide whether empty projects are useful or should be disallowed entirely.

3 Unit Test Suite Structure

Unit tests code source can be found here.

1. **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.
2. **ProjectUnitTest.java**
   This file focuses on Project entities and mirrors the structure of the Todo tests. It covers setup, CRUD workflows, and error validation.
   - **Setup**: Similar to the Todo test class, the API is launched in advance and the base URI is set for all requests.
   - **Test Execution**:
     - Tests ensure that projects can be created, retrieved, updated, and deleted properly, with validations on status codes and field updates.
     - The suite also checks responses to invalid scenarios, such as attempting to create a project without mandatory fields or updating one that does not exist.
   - **Error Handling**: Includes tests for actions on non-existent projects (update or delete), confirming that the system returns the correct error messages.

In both files, the test design ensures that each API object (Todo and Project) 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, collaborative development, and code-review workflows.

4.1 Repository Structure

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

- `Session Notes/` – Documentation and notes from exploratory test sessions (.txt files containing the session notes).
- `src/test/` – Unit test suite. Key subpaths include:
  - `src/test/java/unitTest/`
    - `ProjectUnitTest.java` – Tests for `projects` functionality.
    - `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 Merging and Rebasing

- **Merging** is used to integrate feature branches into `main` via pull requests.
- **Rebasing** may be used during development to keep branches up-to-date with `main`, maintaining a cleaner history.

4.3 Commit Messages

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

4.4 Collaboration and Code Review

All changes targeting `main` undergo peer review. Pull requests must be approved before merging to ensure quality and maintain project standards.

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.

5.3 Project Issues

Bug #4: Allowing Empty Project Creation

- **Summary:** Projects can be created without a title or description.
- **Details:** The system does not validate input when a project is posted, accepting blank fields.
- **Impact:** Results in incomplete records, reducing clarity and traceability of projects.
- **Steps:**
  1. Submit a POST request with empty title and description values.
  2. Verify that the project is still created.

Bug #5: Duplicate Projects Allowed

- **Summary:** The API accepts multiple projects with identical fields.
- **Details:** No validation prevents creating projects with the same title and description.
- **Impact:** Leads to duplicate records, clutter, and difficulty in distinguishing projects.
- **Steps:**
  1. Send two POST requests with the same project data.
  2. Confirm that both requests succeed and create duplicates.

Bug #6: Creating Projects Already Marked as Completed

- **Summary:** Projects can be initialized with a “completed” status.
- **Details:** The workflow allows a project to be registered as completed at the moment of creation.
- **Impact:** Misrepresents the project lifecycle and distorts reporting and tracking.
- **Steps:**
  1. Issue a POST request creating a project with status set to completed.
  2. Observe that the project is successfully added.

Bug #7: Inconsistent Status Values on Projects

- **Summary:** A project can be posted with conflicting statuses (e.g., completed and active).
- **Details:** The API permits contradictory status flags, resulting in ambiguous state.
- **Impact:** Misleading project records that can cause errors in task tracking and workflow management.
- **Steps:**
  1. Submit a POST request creating a project marked as completed while also setting status to active.
  2. Confirm that the project is created with contradictory state.

Bug #8: PUT Replaces Projects Instead of Updating

- **Summary:** PUT on projects replaces the full record instead of modifying fields.
- **Details:** Similar to the todo bug, PUT discards any unspecified fields and overwrites the entire project object.
- **Impact:** Can unintentionally erase valid project data, creating inconsistency.
- **Steps:**
  1. Execute a PUT request with only a subset of project fields.
  2. Notice that the remaining fields are replaced with default values.

Bug #9: Misuse of POST for Updating Projects

- **Summary:** POST is used for project amendments instead of PATCH.
- **Details:** REST principles suggest PATCH for partial modifications, but the API accepts POST for this purpose.
- **Impact:** Creates inconsistency and confusion for developers consuming the API.
- **Steps:**
  1. Attempt to modify a project using POST.
  2. Confirm that the system allows this behavior.

Bug #10: Searching Projects by Description Requires Exact Match

- **Summary:** Search queries only work with exact description matches.
- **Details:** The system does not support partial description search, limiting usability.
- **Impact:** Reduces flexibility for users who want to filter projects by keywords or partial phrases.
- **Steps:**
  1. Perform a query with a partial description string.
  2. Observe that only exact matches are returned, not partial ones.