ECSE 429 – Software Validation Project

PART B – STORY TESTING OF `REST APIs`

Authors
Ahmed Mossa – 261001739 – ahmed.mossa@mail.mcgill.ca
Lounes Azzoun – 261181741 – lounes.azzoun@mail.mcgill.ca

---

1. Summary of Deliverables

In Part B, we developed and executed a Cucumber-based story testing suite to validate the functional behavior of the Todo Manager REST API. Ten distinct user stories were defined to cover the main CRUD operations for both Projects and Todos, ensuring comprehensive verification of expected and edge-case behaviors.

All story tests were written in Gherkin syntax and stored under `src/test/resources/features`, with one file per user story. The files are parsed and executed automatically using Cucumber for JUnit 5.
Test automation was implemented in two Java classes:

- `ProjectsStoryTest.java` for project-related features.
- `TodoStoryTest.java` for todo-related features.

The overall suite is coordinated by `CucumberTestRunner.java`, which collects, randomizes, and executes all feature files using a fixed seed to ensure deterministic reproducibility.

A demonstration video showing all tests executing in random order is provided separately.

1.1 User Stories

The following ten user stories were implemented as independent feature files:

1. **Create New Project**
   As a user, I want to create a project without specifying an ID so that I can quickly add new projects using minimal input.

2. **Create New Todo**
   As a user, I want to create a new todo task so that I can add it to my todo list, and save it in the system.
3. **Delete an Existing Project**
   As a user, I want to delete a project by its ID so that it no longer appears in listings.
4. **Delete an Existing Todo**
   As a user, I want to remove a todo so it no longer appears in the list.
5. **Get All Projects**
   As a user, I want to view all existing project entries so that I can review their current details.
6. **Get All Todos**
   As a user, I want to fetch all todo tasks so that I can view my entire todo list.
7. **Retrieve a Specific Project**
   As a user, I want to access a particular project by ID or title so that I can review its details.
8. **Retrieve a Specific Todo**
   As a user, I want to get a specific todo task so that I can retrieve its details.
9. **Modify an Existing Project**
   As a user, I want to update an existing project so that I can adjust its details.
10. **Modify an Existing Todo**
   As a user, I want to modify details of an existing todo task so that I can keep my todo list up to date.

Each feature file includes Normal, Alternate, and Error flows to validate success, optional cases, and failure responses.

---

2. Story Test Suite Structure

The suite ensures that all API endpoints are exercised under controlled conditions and is organized into three major components:

1. Service Initialization and Management.

2. Scenario Definitions and Step Implementations.

3. Assertions, Validations, and Cleanup Procedures.

2.1 Service Initialization and Management

In `TodoStoryTest.java`, the step “The service is running” automatically starts the REST API service if it is not active.

- The test first checks availability via a GET request to `http://localhost:4567`.
- If the service is already running, it is shut down gracefully before restart to avoid conflicts.
- The service is then launched through a ProcessBuilder that runs `runTodoManagerRestAPI-1.5.5.jar`.

After all tests, `@AfterAll` hooks (`killServer()` and `stopServer()`) ensure a clean shutdown so no background processes remain.

2.2 Common Background Setup

Several feature files use background steps to verify that specific resources exist before tests execute. Examples include:

- Ensuring todos with IDs 1 and 2 exist with the expected titles “scan paperwork” and “file paperwork”.
- Confirming that a project with ID 1 and title “Office Work” exists and is inactive.

These steps ensure a consistent starting state for all scenarios.

2.3 Scenario Steps for CRUD Operations

Each feature file implements the four core operations: Create (POST), Retrieve (GET), Update (PUT and POST), and Delete (DELETE).
Both Java classes define Cucumber steps mapped directly to these operations using RestAssured for HTTP calls. Examples include:

- Sending a POST to create a new project with a title and active flag.
- Sending a PUT to update a todo with new details.
- Sending a DELETE to remove an entity and verifying its absence.

This modular design allows shared steps and easy extension for future tests.

2.4 Flow Control (Normal, Alternate, Error Flows)

Every feature is divided into three logical flows:

- **Normal Flow:** Verifies successful operations, ensuring status codes (200 or 201) and valid content.
- **Alternate Flow:** Tests optional inputs or valid edge cases, such as creating a project with no fields or updating via POST instead of PUT.
- **Error Flow:** Checks error handling by using invalid IDs or malformed data and expects proper error messages and status codes (400 or 404).

2.5 Reusable Steps and Consistency

Common steps such as “The service is running”, “We get an HTTP response ”, and “The following todos exist in the system” are shared across feature files to ensure consistency. Clear naming and grouped methods keep the suite organized and readable.

2.6 Assertions and Validation

Assertions use JUnit methods to verify expected results:

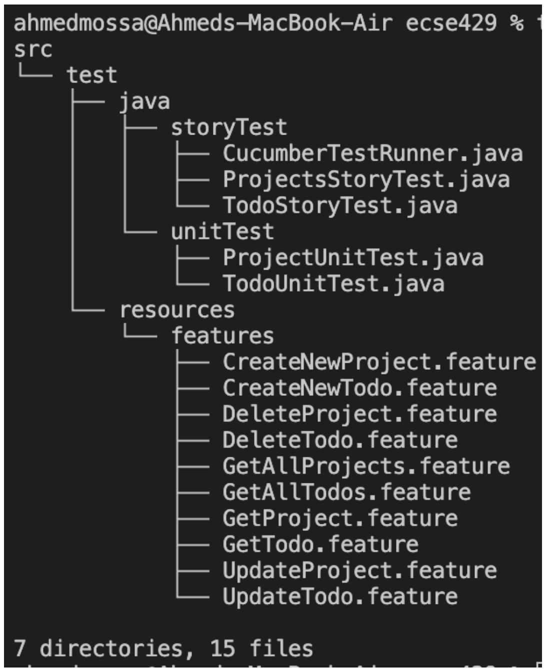
- Status codes are checked with `assertEquals`.
- Response fields such as title, active, or description are validated against expected values.
- Error messages are compared for exact text matching (e.g., “Failed Validation: active should be BOOLEAN”).
- Deletion is confirmed by subsequent GET requests returning 404.

This approach ensures precise validation and maintainable test logic.

---

3. Source Code Repository Description

Repository structure:

- 

```
ahmedmossa@Ahmeds-MacBook-Air ecse429 % 
src
└── test
    ├── java
    │   ├── storyTest
    │   │   ├── CucumberTestRunner.java
    │   │   ├── ProjectsStoryTest.java
    │   │   └── TodoStoryTest.java
    │   └── unitTest
    │       ├── ProjectUnitTest.java
    │       └── TodoUnitTest.java
    └── resources
        └── features
            ├── CreateNewProject.feature
            ├── CreateNewTodo.feature
            ├── DeleteProject.feature
            ├── DeleteTodo.feature
            ├── GetAllProjects.feature
            ├── GetAllTodos.feature
            ├── GetProject.feature
            ├── GetTodo.feature
            ├── UpdateProject.feature
            └── UpdateTodo.feature

7 directories, 15 files
```

Feature files store the user stories in Gherkin format. The runner shuffles and executes them with a fixed seed for consistent order. Both Java classes contain step definitions and assertions for each entity type.

This organization maintains traceability and clear separation of concerns while supporting reusability of common logic.

---

4. Story Test Suite Execution Findings

4.1 Test Coverage and Feature Verification

The suite achieves comprehensive coverage of the main functionalities of the Todo Manager REST API. It validates that projects and todos can be created, retrieved, updated, and deleted correctly. It also checks that the API returns appropriate error responses for invalid data or endpoints.

Coverage summary:

- Create operations verify POST behavior and field assignment.
- Retrieve operations confirm GET responses and filtering.
- Update operations validate both PUT and POST updates.

- Delete operations confirm successful removal and post-deletion status codes.

The suite demonstrates that the API is functionally sound and robust against common input errors.

4.2 Additional Bug Summary Considerations

Bug #1 – Filtering Projects with Invalid Parameters
A GET request to `/projects` with an unrecognized parameter (such as `?invalidParam=3`) returns unfiltered results instead of reporting an error. This can mislead users into thinking the filter was applied successfully.
Impact: Possible data misinterpretation and reduced API trustworthiness.
Steps to reproduce: Send GET `http://localhost:4567/projects?invalidParam=3` and observe that all projects are returned without any warning.

Bug #2 – Filtering Todos with Invalid Parameters
A GET request to `/todos` with an invalid parameter (such as `?booleanParam=false`) also returns unfiltered results without warnings.
Impact: Applications depending on filtered data may generate inaccurate views or analyses.
Steps to reproduce: Send GET `http://localhost:4567/todos?booleanParam=false` and observe that all todos are returned without error.

Both issues highlight the need for stricter validation of query parameters and clearer API error messages.

4.3 Overall Outcome

All core functional tests passed successfully across both Projects and Todos. The test suite provides reliable automated regression coverage and a foundation for continuous integration testing.
The identified bugs serve as important feedback for improving input validation and error reporting in future API versions.