WEB APPLICATION

TEST PLAN

- DEMO -

Version 1.0

Author : Bogdan-Adrian RADA

Table of Contents

1. Introduction

Purpose

The purpose of this test plan is to provide a proof of concept documentation than highlights the approach, strategy, objectives and resources for a basic automation framework used for testing different API calls to a local web server.

Project Overview

The project is split into two parts : on one hand we have the source code for a web server based on Swagger for API endpoint testing and on the other a test script where we define the test conditions, test data and the test cases to be executed. The web server can be initiated from either debug mode (using Visual Studio) or from the build itself (worth mentioning that we need to modify the appsettings.json to have basic functionality working). The test script is built in the xUnit framework.

2. Scope

In-Scope

Test API endpoints using GET, POST, PUT, DELETE.

Validation of correct response status codes.

Testing of negative scenarios based on application's constraints.

Manual testing of the endpoints using web interface APIs.

Delivery of a test report with results.

Out-of-Scope

End-to-end automation testing in a CI/CD environment due to time constraints (could be considered for implementation when product reaches a higher level of maturity).

3. Testing Strategy

Test Objectives

Test the backend of an API driven web application using an automated test framework.

Test Assumptions

At this current stage of implementation, testing assumes that the web server is being run on the same environment as the test framework and that all dependencies have been covered.

Data Approach

*Currently the test data is hard-coded in the test script itself in the form of **InlineData** and covers both positive and negative scenarios, according to the requirements specified in the project's Readme file.*

Level of Testing

Test Type	Description	Responsible Parties
Functional	*Ensure the correct functionality of the APIs under test according to the requirements (query task list and their status, add/delete tasks, update tasks etc.)*	**Automation Testing Team**
Non-Functional	*Validate various performance and security related features.*	
User Acceptance	*Confirm correct functionality of the API endpoints from the server's web page.*	**Manual Testing Team**

4. Test Case List

Test ID	Test Name	Test Steps	Expected Output
100	*Check Server Response*	*Send GET request to server uri.*	*Response code is 200 or OK and content format is application/json*
101	*Check For Empty Response Content*	*1. Delete all existing tasks.* *2. Send GET request to server uri.*	*1. Response code is 200 or OK* *2. Server returns an empty list*
200	*Create New Task*	*1. Create a new task with a Boolean value for the completion state attribute* *2. Create a new task with wrong completion state type (e.g. string)* *3. Create e new task with an empty name.* *4. Test 2-value BVA by creating one task with name having 100 characters and another with 101 characters*	*1. Response code is 200 or OK* *2-3. Response code is 400 or Bad Request* *4. Response OK (for 100 character name), Bad Request (for 101)*
201	*Modify Existing Task*	*1. Use GET to retrieve existing tasks as list of objects* *1.1 In case there is no task present, create a new task and update list* *2. Retrieve the name and id of the first task from the list* *3. Use PUT and id retrieved from step 2 to change the completion state of the task* *4. Retrieve task list and save completion state of the first task (the one modified at step 3)*	*1-1.1. Obtain a list of tasks and confirm it's not empty* *2. Name and ID should not be empty strings* *3. Response code is 200 or OK* *4. Updated completion state of the modified task should be the same as*

			the one sent at step 3
202	*Modify Non-Existing Task*	*1. Retrieve task list from web server using GET and save all task ids in a separate list* *2. Create a new task and assign it an id, a name and a completion state* *2.1 If task id is present in task list, generate a different id* *3. Use PUT to update the task id from step 2*	*1. List count should be >=0* *3. Response code is 404 or Not Found*
300	*Check HTTP Redirection*	*Send GET request to http://localhost:5248*	*Response code is 307 or Temporary Redirect and location should be https://localhost:7246/*
301	*Check Basic Server Response Time*	*Send GET request to server and capture response time* *HINT :* ***curl -o NUL -s -w "Total: %{time_total}s\n" https://localhost:7246/tasks***	*Response time should be < 100 ms.*
302	*Check Task Limit*	*Create tasks continuously until reaching the limit supported by the server*	*No prior information about this limit so we'll go with the assumption based on the default value of the* ***JavaScriptSerializer.MaxJsonLength*** *Property (2097152 characters, which is equivalent to 4 MB of Unicode string data). If limit is reached, server should return a negative response.*
303	*Check Request Rate Limit*	*Send requests with a high rate that should reach and/or exceed the configured limit of the server.*	*Response code is 429 or Too Many Requests (No prior information about this limit)*
401	*Check Web Functionality*	*Access the URI through a web browser and perform task query, creation, update and deletion via the exposed interfaces.*	*Proper responses should be received based on the actions taken.*

Test IDs 302 and 303 can be considered edge cases. We could expend this list with other validations (e.g. use GNS3 or other network simulators to reproduce a request from a remote host over a service provider network and measure response times).

5. Execution Strategy

Entry Criteria

Entry Criteria	Test Team	Notes
Test environment(s) is available	✓	
Test data is available	✓	
Code has been merged successfully	✓	
Development has completed unit testing	✗	Pending UT
Test scripts are completed, reviewed and approved by the Test Manager and Customer	✓	

Exit criteria

Exit Criteria	Test Team	Notes
100% Test Scripts executed	✓	
90% pass rate of Test Scripts	✗	Non-Functional testing failed
No open Critical and High severity defects	✗	Issues documented in internal ticketing system
All remaining defects are either canceled or documented as Change Requests for a future release	✓	
All expected and actual results are captured and documented with the test script	✓	
All defects logged in -Defect Tracker/Spreadsheet	✓	
Test environment cleanup completed	✓	

Validation and Defect Management

Validation of test output should be done against the requirements, by analyzing the test output and the execution logs.

Defects found during the Testing should be categorized as below:

Severity	Impact
1 (Critical)	*Functionality is blocked and no testing can proceed Application/program/feature is unusable in the current state*
2 (High)	*Functionality is not usable and there is no workaround but testing can proceed*
3 (Medium)	*Functionality issues but there is workaround for achieving the desired functionality*
4 (Low)	*Unclear error message or cosmetic error which has minimum impact on product use.*

6. Reporting

As part of the exit criteria, the automated test execution activity should include a testing report that's to be shared to the stakeholders. This report must highlight the total number of executed test cases, the list of tests, the passed/failed/skipped summary, pass percentage, execution time and execution logs in case of failures (as seen below).

Test run details

Total tests	Passed : 13 Failed : 1 Skipped : 0	Pass percentage	Run duration
14		92 %	1s 707ms

Failed Results

▼ C:\Users\borada\Documents\PROJECTS\api_test_project\ApiTest\bin\Debug\net8.0\ApiTest.dll

✗ ApiTest.UnitTest1.Test1GetResponseOk 15ms

Error:

```
Assert.Equal() Failure: Values differ
Expected: OK
Actual:   NotFound
```

Stack trace:

```
   at ApiTest.UnitTest1.Test1GetResponseOk() in C:\Users\borada\Documents\PROJECTS\api_test_project\ApiTest\UnitTest.cs:line 75
--- End of stack trace from previous location ---
```

All Results

▼ C:\Users\borada\Documents\PROJECTS\api_test_project\ApiTest\bin\Debug\net8.0\ApiTest.dll

✓ ApiTest.UnitTest1.Test5UpdateExistingTask(update: "true", expected: True)	376ms
✓ ApiTest.UnitTest1.Test5UpdateExistingTask(update: "false", expected: False)	16ms
✓ ApiTest.UnitTest1.Test7DeleteExistingTask	16ms
✓ ApiTest.UnitTest1.Test4CreateTask(testScope: "New task, wrong completion state", task: "{\"name\":\"task43\", \"isCompleted\": \"false\"}", expected: BadRequest)	10ms
✓ ApiTest.UnitTest1.Test4CreateTask(testScope: "New task, wrong task name", task: "{\"name\":\"\", \"isCompleted\": false}", expected: BadRequest)	9ms
✓ ApiTest.UnitTest1.Test4CreateTask(testScope: "New task, completion true", task: "{\"name\":\"task42\", \"isCompleted\": true}", expected: OK)	9ms
✓ ApiTest.UnitTest1.Test4CreateTask(testScope: "New task, name has 101 characters", task: "{\"name\":\"tasktasktasktasktasktasktasktaskta"···, expected: BadRequest)	9ms
✓ ApiTest.UnitTest1.Test4CreateTask(testScope: "New task, name has 100 characters", task: "{\"name\":\"tasktasktasktasktasktasktasktaskta"···, expected: OK)	9ms
✓ ApiTest.UnitTest1.Test4CreateTask(testScope: "New task, completion false", task: "{\"name\":\"task41\", \"isCompleted\": false}", expected: OK)	9ms
✓ ApiTest.UnitTest1.Test6UpdateNonExistingTask	13ms
✓ ApiTest.UnitTest1.Test8DeleteNonExistingTask	14ms
✓ ApiTest.UnitTest1.Test2GetResponseJsonFormat	12ms
✓ ApiTest.UnitTest1.Test3GetEmptyListAfterCleanup	20ms
✗ ApiTest.UnitTest1.Test1GetResponseOk	15ms

Error:

```
Assert.Equal() Failure: Values differ
Expected: OK
Actual:   NotFound
```