FINAL REPORT on

BOTest

A desktop tool for automated black box testing of web APIs

SE 801: Software Project Lab 3

Submitted By:
Ahmed Ryan
Roll-BSSE 1011
Exam Roll-100835
BSSE 10th Batch
Session: 2017-18

Supervised By:
Moumita Asad
Lecturer
Institute of Information Technology
University of Dhaka

Date: 7th December, 2022

Table Of Contents

Letter of Transmittal

December 7, 2022
BSSE 4th Year Exam Committee
Institute of Information Technology
University of Dhaka

Subject: Final report on “BOTest: A desktop tool for automated black box testing of Web APIs”.

Sir,

With due respect, I am pleased to submit the final report of Software project lab-III on “BOTest: A web based tool for automated black box testing of web APIs”. I have tried my best to deliver a good report. However, it might lack perfection. So, I, therefore, hope that you would be kind enough to accept my report and oblige thereby.

Sincerely yours,

Ahmed Ryan
Roll: BSSE 1011
BSSE 10th Batch
Institute of Information Technology
University of Dhaka.

Letter of Endorsement

7th December, 2022
BSSE 4th Year Exam Committee 2022
Institute of Information technology,
University of Dhaka

Subject: Approval of the report

This letter is to certify that Ahmed Ryan, BSSE 1011, student of Institute of Information Technology, University of Dhaka, has done "BOTest: A desktop tool for automated black box testing of web APIs" under my supervision. I have gone through the report. All the information mentioned in this document is true.

I wish him all the best and his bright future.

_____

Moumita Asad
Lecturer
Institute of Information Technology,
University of Dhaka

Acknowledgment

Firstly, I would like to thank the Almighty for helping me complete the final report. I am grateful to the Institute of Information Technology for giving me such a tremendous opportunity to work on “BOTest: A desktop tool for automated black box testing of Web APIs”. I would like to convey my tremendous gratitude to my supervisor, Moumita Asad, Lecturer, Institute of Information Technology, University of Dhaka, for providing me with guidelines about preparing this report. She helped me a lot by sharing her valuable knowledge with me and monitored my task throughout the whole time. Lastly, I would like to thank my classmates. They have always been helpful and provided valuable insights from time to time.

Abstract

This document contains the software requirements and specifications, architectural design and implementation details, testing plan of “BOTest: A desktop tool for automated black box testing of web APIs”. This tool can generate test cases for RESTful Web APIs automatically and perform a black box test on the API to detect faults and bugs in the API. Inputs will be provided through the desktop app and output will be demonstrated in the browser. The test results will be provided to the user so that s/he can easily determine the limitations of the API. This document can be followed to develop the web API testing tool mentioned above.

Table of Contents

Chapter 1: Introduction

90% software developers use APIs and spend 30% of their time coding APIs [1]. In addition, 93.4% of API developers use RESTful APIs [1]. For this reason, the testing of web API is very vital. It can save a lot of time for the developers and discover undesired bugs. However, source code of all applications are not readily available. Keeping that in mind, a blackbox testing tool for RESTful web api will be developed named "**BOTest**".

1.1 Purpose

The purposes of this document are:

- Identify the requirements that have to be carried out as a part of the project.
- Form the baseline for construction of the proposed system.
- Help to reduce the development effort and reveal misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.

1.2 Scope

The scope of this project is defined below:

- Open API Specification file will be taken as input.
- Outputs will be shown in html format.
- Array type of inputs is not handled due to limitations of the libraries being used in this project.

1.3 Assumptions

The assumptions of the project are:

- The system will contain java runtime libraries.
- The system will contain a browser to show HTML reports.
- The system will have an updated Windows OS.

1.4 Definitions

Software Testing

It is the act of examining the artifacts and the behavior of the software under test by validation and verification. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. By validation we check whether we are building the product right (software is not faulty). On the other hand, by validation we check whether we are building the right product (software requirements are met).

White Box Testing

It is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing. [3]

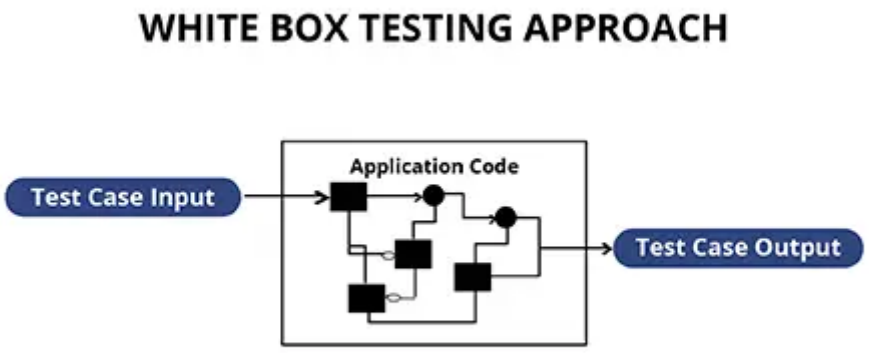


Figure 1: White Box Testing

Black Box Testing

It is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing. [4]

Figure 2: Black Box Testing

API

An application programming interface (API) is a way for two or more computer programs to communicate with each other. It allows heterogeneous devices to communicate with each other. It is a type of software interface, offering a service to other pieces of software. A document or standard that describes how to build or use such a connection or interface is called an API specification. [5]

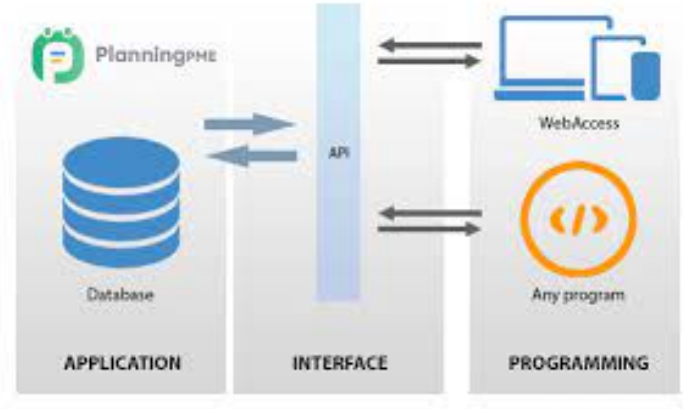


Figure 3: Application Programming Interface

RESTful API

Any web api that conforms to the 6 REST constraints can be called a RESTful API. [6] The 6 constraints are mentioned below -

1. Use of Uniform Interface
2. Client-Server
3. Stateless
4. Caching
5. Multiple Layers of Servers
6. Code on Demand

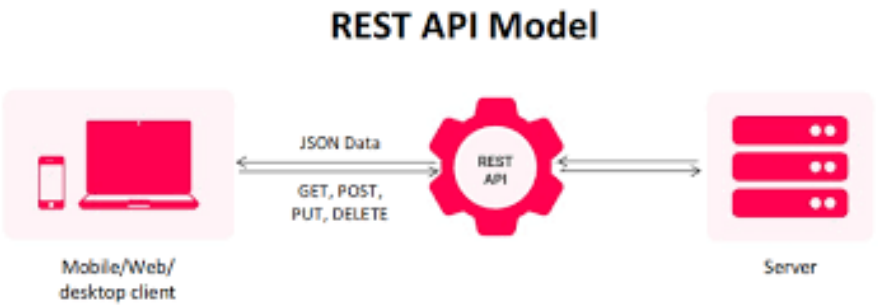


Figure 4: RESTful API

Open API Specification

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases. [7]

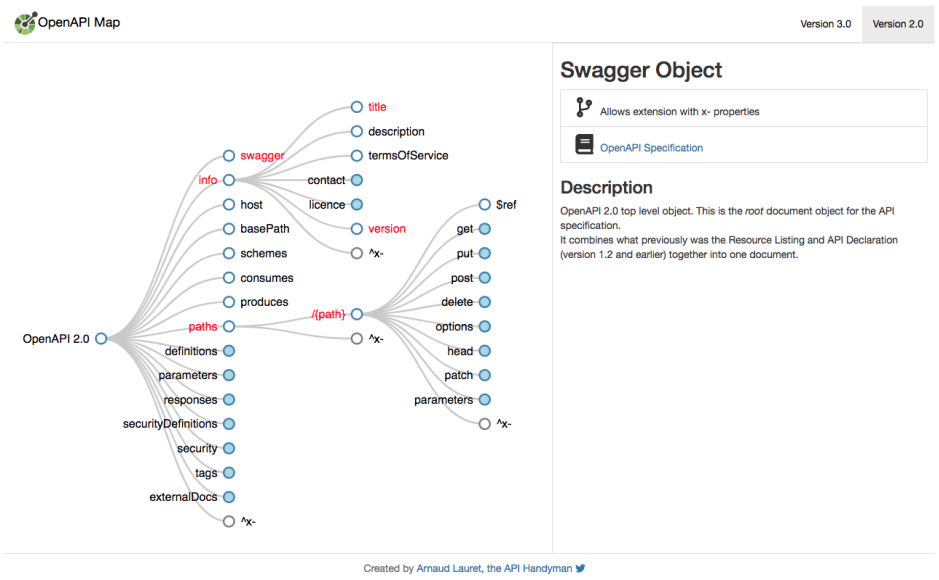


Figure 5: Open API Specification Skeleton

Chapter 2: Overall Description

This chapter presents the quality function deployment and usage scenario of the Blackbox Testing Tool of RESTful API named "**BOTest**".

2.1 Quality Function Deployment

Quality Function Deployment translates the needs of the customer into technical requirements for software. The following requirements for the system have been identified:

Normal Requirements

Normal requirements of RESTful Web API Testing Tool are:

- A user will be able to select an open api specification file as input.
- A user will be able to provide server related information.
- A user will be able to run a specification based testing.
- The tool will return a test report which will have insights about errors of the api under test.

Expected Requirements

Expected requirements of RESTful Web API Testing Tool are:

- The user will be able to provide input from a UI.
- A user friendly report will be generated which is easily understandable.

2.2 Usage Scenario

BOTest is a tool which will test RESTful Web APIs automatically with minimal human intervention. A user will let the tool know which API needs to be tested by means of an open API specification file. After that, the information of the server (base URI) needs to be provided by the user. That being done, if a user opts to run the bot, new tests will be generated.
After that, generated tests will be executed and test results will be obtained. Finally, a test report of HTML format will be generated from the test results and visualized in a graphical user interface for better user experience.
The main objective of this tool is to minimize human intervention in producing quality test cases and unveiling the anomalies in a web api. However, if a tester wants to make the tests better s/he can supervise the open api specification so that more relevant test cases are produced.

Input

The tool takes an open API specification file given by the user as input from the desktop application.
The user also needs to provide information about the server to the application.

Test Model Generation

Based on the openapi specification and the selected testing technique, a test model will be generated. Test models will vary depending on the testing techniques. A test model is the first building block of generating test cases.

Test Case Generation

Both valid and faulty test cases will be generated to test the API thoroughly.

Test Report Generation

The test cases generated using the test model will be executed and a test report will be generated based on the test results. It will show which For better user experience, a graphical user interface will be used to summarize the report.

Chapter 3: Scenario Based Modeling

This chapter describes scenario based modeling of the system.

3.1 Use Case Diagram

Use case diagrams of boTEST are given below:

Level 0: ***RESTful Web API Test Automation Tool System***
Primary actors: Authenticated User
Secondary actors:
Goal in context: The diagram shown in figure represents the whole RESTful Web API Testing Automation Tool System.

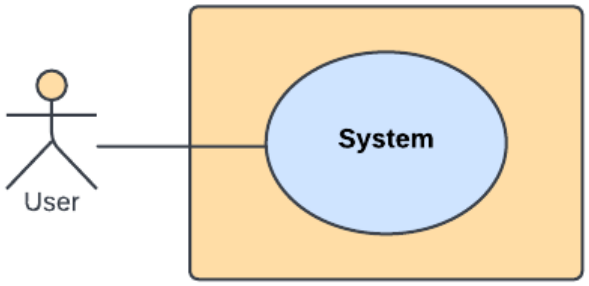


Figure 6: Use Case – Level 0 – Testing Automation Tool System

Level 1: ***Modules of Test Automation Tool***

Primary actors: Authenticated User

Secondary actors:

Goal in context: The diagram shown in figure shows all the modules of RESTful Web API Testing Automation Tool.
The tool consists of 4 modules. They are -
Level 1.1: Test Model Generation
Level 1.2: Test Case Generation
Level 1.3: Test Case Execution
Level 1.4: Test Report Generation

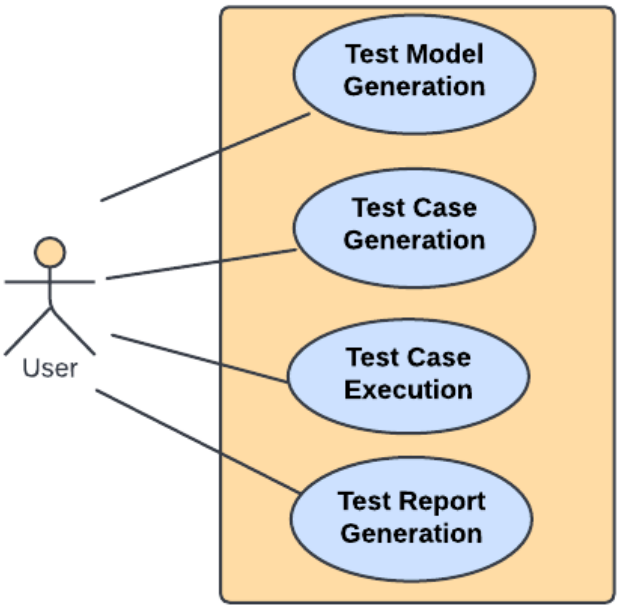


Figure 7: Use Case - Level 1 - Modules of API Testing Automation Tool System

Actions and Replies:

A1: User gives a file as input to test model generation module.
R1: Test model generation module generates a test model.
A2: User gives test model as input to test case generation module.
R2: Test case generation module generates suitable test cases.
A3: User wants to execute the generated test cases.
R3: Test case execution module executes the test cases and produces test results.
A4: User wants to see the test report.
R4: Test report generation module generates a test report.

Chapter 4: Class Based Modeling

This chapter describes class based modeling of the RESTful API Web Testing Tool.

4.1 Analysis Classes

After identifying nouns from the scenario, I have filtered nouns belonging to the solution domain using general classification (external entities, events, things, roles, organizational units, places and structures). Nouns selected as a potential class were filtered using selection criteria (retained information, needed services, multiple attributes, common attributes, common operations, and essential requirements).

After performing analysis on potential classes, I have found the following analysis classes:

1. TestCase
2. TestGenerator
3. TestWriter
4. TestRunner
5. TestResult
6. TestReport

4.2 Class Cards

TestCase	
Attributes	**Methods**
- id - faulty - faultyReason - operationId - method - path - inputFormat - outputFormat - headerParams - pathParams - queryParams - formParams - bodyParams	- getTestCase() - setTestCase()
Responsibilities	**Collaborators**
- Manages all operations related to test case	- TestGenerator

TestGenerator	
Attributes	**Methods**
- specification_file	- generateTest() - generateParameters() - getSpecificationFile() - setSpecificationFile()
Responsibilities	**Collaborators**
- Generates abstract test cases which are language agnostic	

TestWriter	
Attributes	**Methods**
- specification_file - test_cases	- writeTest() - getSpecificationFile() - setSpecificationFile() - getTestCases() - setTestCases()
Responsibilities	**Collaborators**
- Generates executable test cases which are language specific	- TestGenerator

TestRunner	
Attributes	**Methods**
- package_name - class_path - class_name	- runTest() - getPackageName() - setPackageName() - getClassPath() - setClassPath() - getClassName() - setClassName()
Responsibilities	**Collaborators**
- Runs the executable test cases to get test results	- TestWriter

TestResult	
Attributes	**Methods**
- result_id - status_code - response_body - output_format - passed - fail_reason	- getTestResult() - setTestResult()
Responsibilities	**Collaborators**
- A model class for test result	- TestRunner

TestReport	
Attributes	**Methods**
- result_id - status_code - response_body - output_format - passed - fail_reason	- getTestReport() - setTestReport()
Responsibilities	**Collaborators**
- Generates HTML test report from the test results	- TestResult

4.2 Class Responsibility Collaboration Diagram

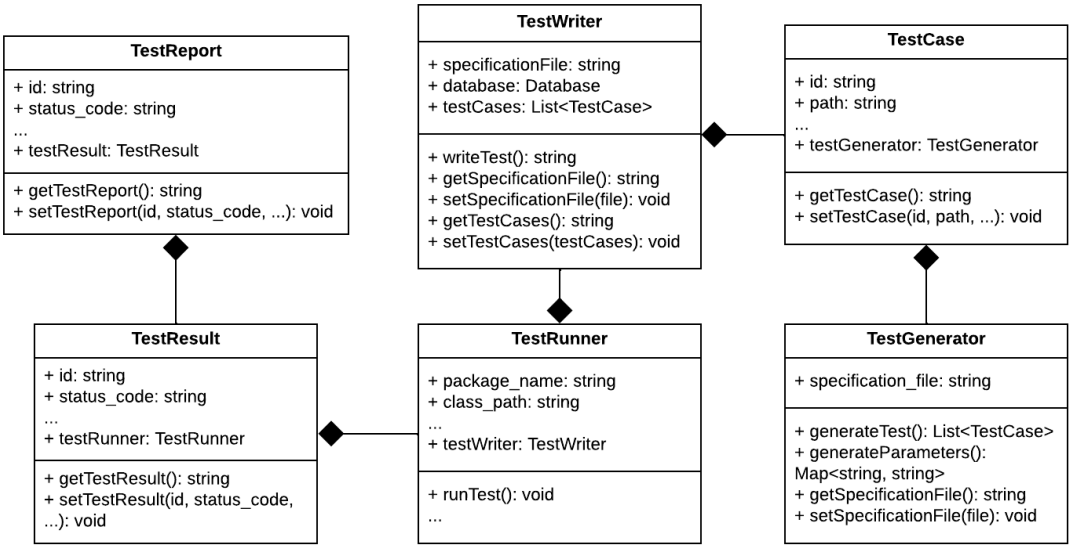


Figure 11: CRC diagram

Chapter 5: Software Design Architecture

5.1 Architectural Design

BOTest will be a desktop application without the support of a database. Hence, it won't need a 3 or 2 tier architecture. It will have a desktop User Interface to get the user Inputs. The output will be presented on a remote server.

5.2 Architecture Context Design

An architectural context diagram is used to model the manner in which software interacts with entities external to its boundaries. Systems that interpolate with the target system can be represented as -

a. Subordinate Systems: Those systems which are used by the target system and provide data or processing that are necessary to complete target system functionality.
b. Superordinate Systems: Those systems which are used by the target system and provide data or processing that are necessary to complete target system functionality.
c. Peer-Level Systems: Those systems that interact on a peer-to-peer basis (i.e. information is either produced or consumed by the peers and the target system.
d. Actors: Entities (people, devices) that interact with the target system by producing or consuming information that is necessary for requisite processing.

Each of these external entities communicates with the target system through an interface (the small shaded rectangles)

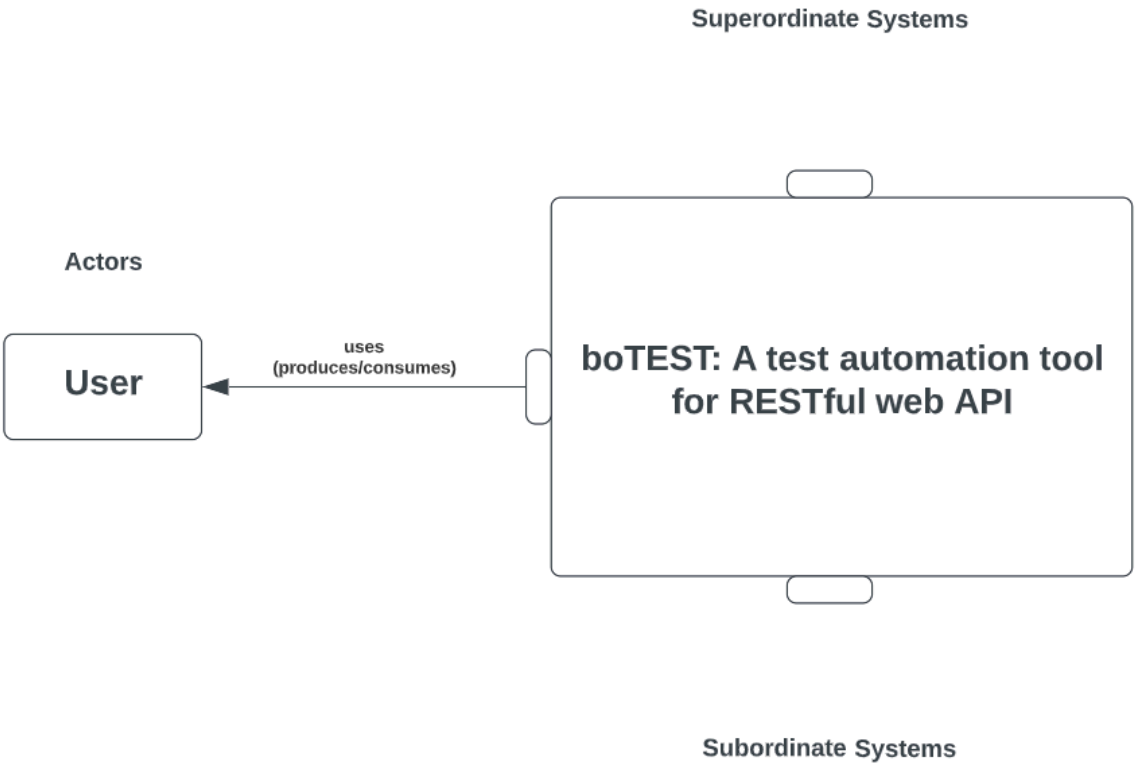


Figure 13: Architectural Context Diagram

5.3 Defining Archetypes

An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system. Archetypes are the abstract building blocks of an architectural design. In many cases, archetypes can be derived by examining the analysis classes defined as part of the requirements model. An archetype is a generic, idealized model of a person, object, or concept from which similar instances are derived, copied, patterned, or emulated.

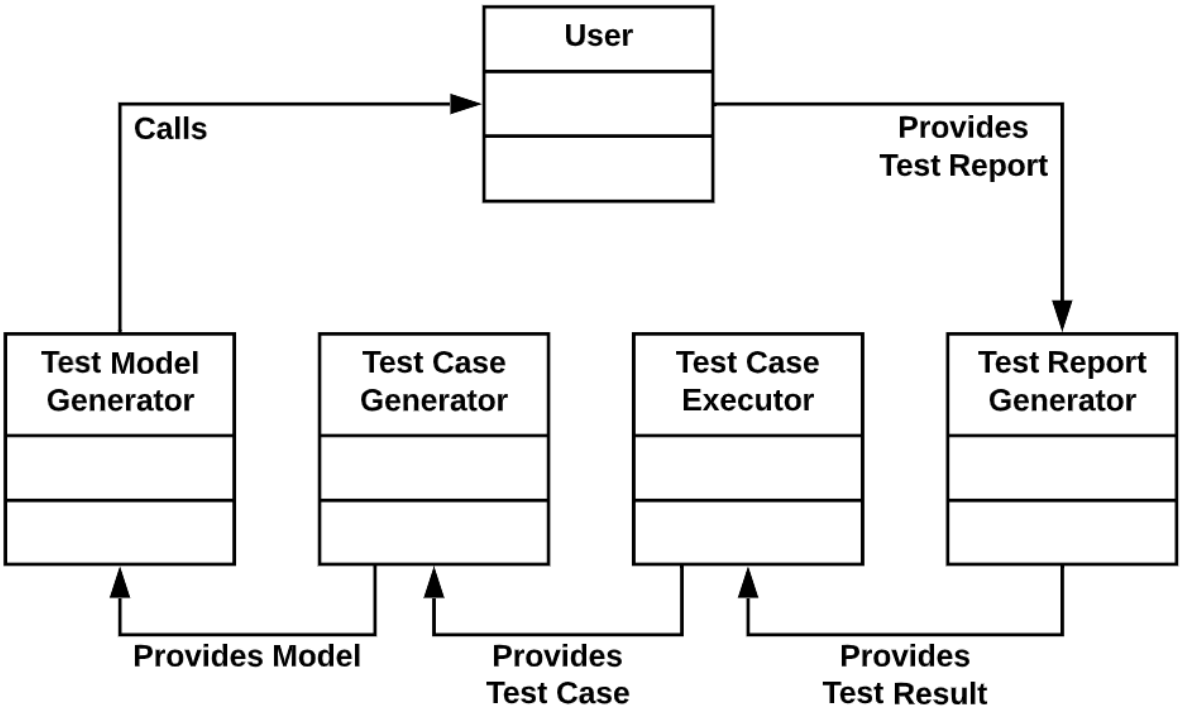


Figure 14: Defining Archetypes

Chapter 6: Methodology

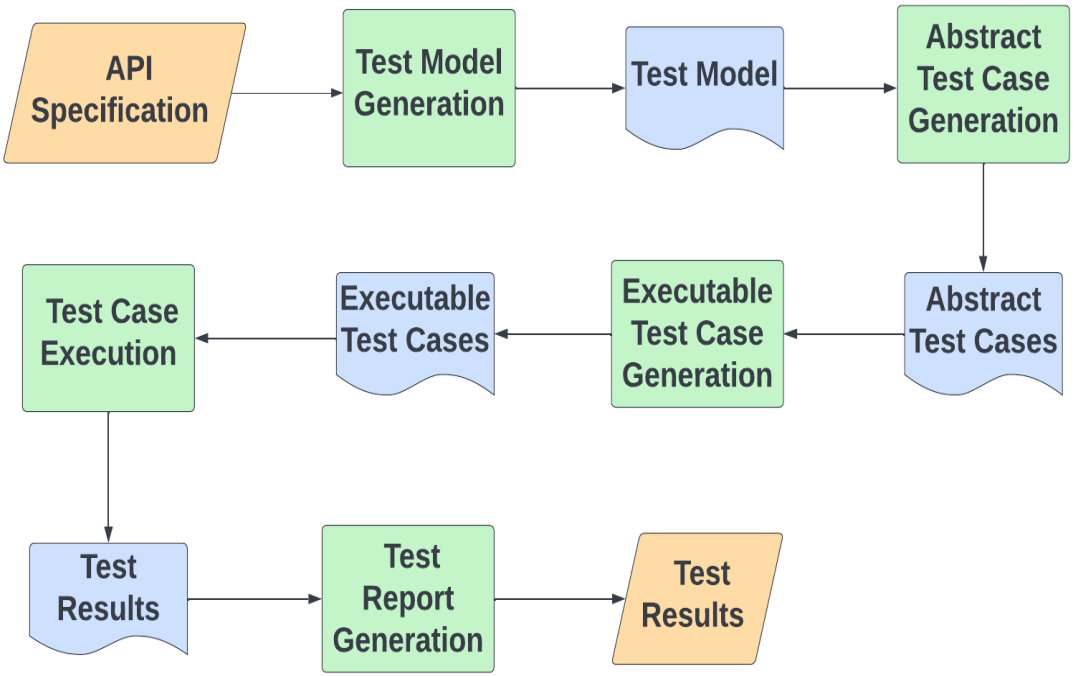


Figure 15: BOTest workflow

API SPECIFICATION:

An openAPI specification file will be provided as input to boTEST. Then the user will decide if s/he wants to generate new tests or run previously generated tests. Additionally, s/he will choose a testing technique.

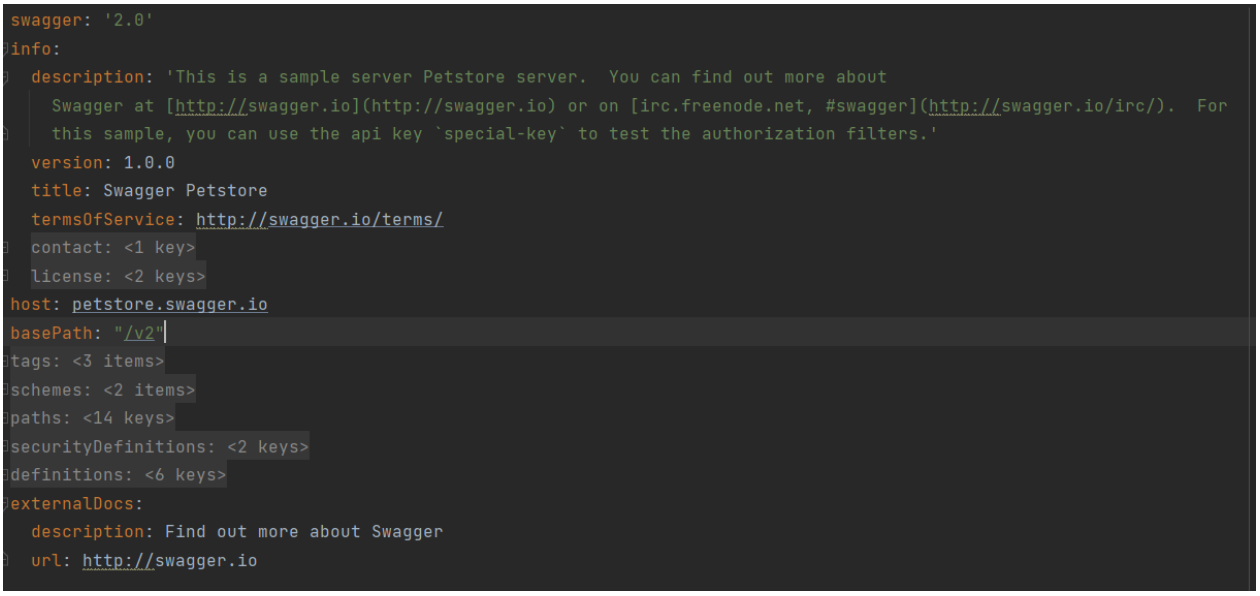


Figure 16: API Specification

TEST MODEL GENERATION:

Test model is a test configuration model based on which test cases will be generated. From the inputs taken, the test model will generate a test model based on the testing technique. Here, test models vary depending on testing technique. For example, if the technique is RT (Random Testing), the parameters generator will be random.

```
testConfiguration=TestConfiguration{
    operations=[Operation{
        testPath='/search',
        operationId='search',
        method='GET',
        testParameters=[TestParameter{
            name='query',
            in='query',
            weight=1.0,
            testcases.generators=[Generator{
                type='RandomStringGenerator',
                genParameters=[GenParameter{
                    name='minLength',
                    values=null,
                    objectValues=null
                }, GenParameter{
                    name='maxLength',
                    values=null,
                    objectValues=null
                }],
                valid=true
            }]
        }, TestParameter{
            name='type',
            in='query',
            weight=0.5,
            testcases.generators=[Generator{
```

Figure 17: Test Model

Users will be able to modify test models to generate more sophisticated test cases.

ABSTRACT TEST CASE GENERATION:

Abstract test cases are tests which are written in plain text, it is language agnostic and not executable yet. Based on abstract test cases, executable test cases (language dependent) will be generated in the future.

```
[TestCase{
    id='test_wH9cOlFqqaBk_search',
    faulty=false,
    faultyReason='none',
    operationId='search',
    method=GET,
    path='/search',
    inputFormat='application/json',
    outputFormat='null',
    headerParameters=null,
    pathParameters=null,
    queryParameters={market=StwQjUNxb, query=, type=wJ},
    formParameters=null,
    bodyParameter='null'
}, TestCase{
    id='test_ZBfvl9KORcfb_findArtistById',
    faulty=false,
    faultyReason='none',
    operationId='findArtistById',
    method=GET,
    path='/artists/{artistId}',
    inputFormat='application/json',
    outputFormat='null',
    headerParameters=null,
    pathParameters={artistId=1465432600},
    queryParameters=null,
    formParameters=null,
    bodyParameter='null'
},
```

Figure 18: Abstract Test Case

EXECUTABLE TEST CASE GENERATION:

Executable test cases are test classes which can be run by the machine. It is language dependent. I have used Junit and Rest Assured technologies for creating the executable test cases. Once the executable test cases are run, test results will be obtained.

```
public class SimpleAPI {

    ahmedryan *
    @Before
    public void setUp() { RestAssured.baseURI = "http://localhost:8800/api/v1"; }

    ahmedryan *
    @Test
    public void test_wH9cOlFqqaBk_search() {
        String testResultId = "test_wH9cOlFqqaBk_search";

        try {
            Response response = RestAssured
                    .given()
                        .log().all()
                        .queryParam("market", "StwQjUNxb")
                        .queryParam("query", "")
                        .queryParam("type", "wJ")
                    .when()
                        .get("/search");

            response.then().log().all();
            System.out.println("Test passed.");
        } catch (RuntimeException ex) {
            System.err.println(ex.getMessage());
            Assert.fail(ex.getMessage());
        }
    }
```

Figure 19: Executable Test Case

TEST CASE EXECUTION:

Executable test cases will be executed using test runners. Junit tests will be executed and we will get test results from here.

TEST REPORT GENERATION:

From the results obtained, a human readable test report will be generated using test report generation technologies like Allure report.

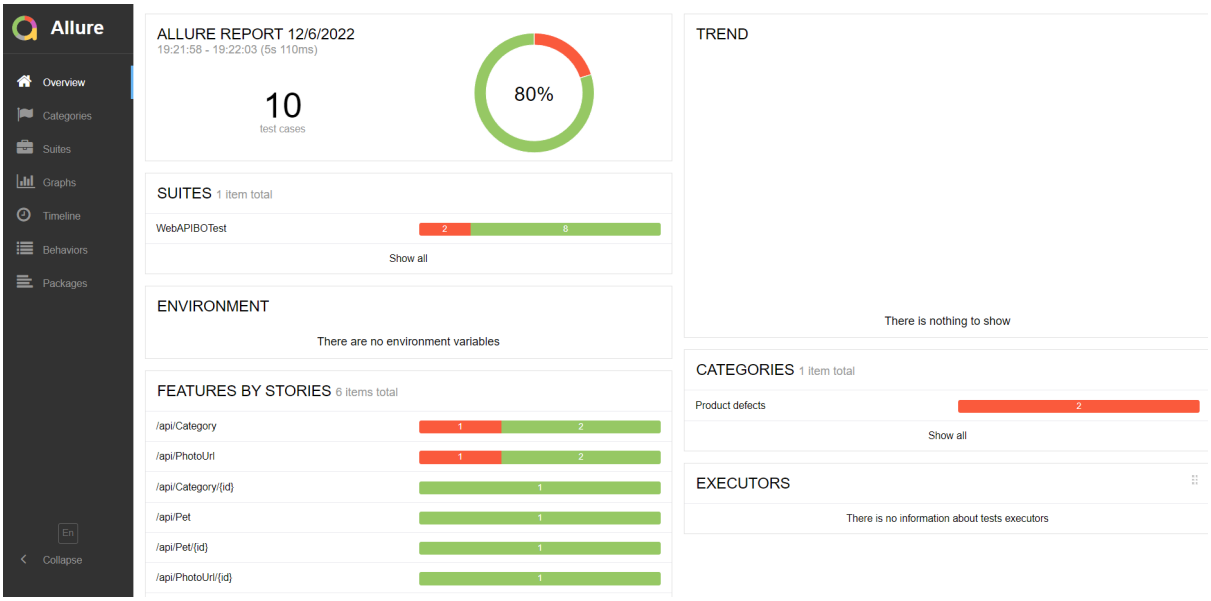


Figure 20: Test Report

Chapter 7: Component-Level Design

This chapter describes the component level design for the BOTest tool. Component level design occurs after the first iteration of architectural design has been completed.

7.1 Design Classes

The following design classes have been identified:

- TestGenerator
- TestWriter
- TestRunner

Test Generator

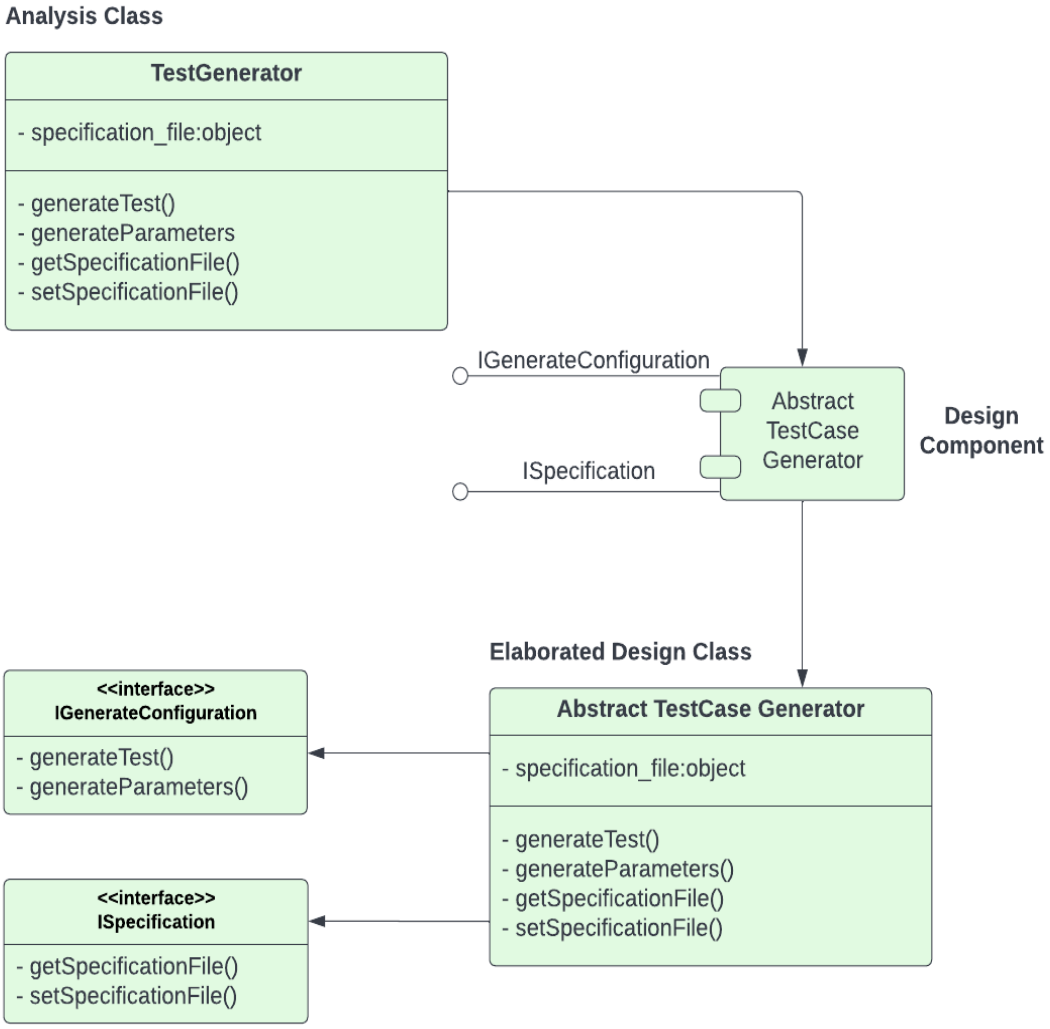


Figure 21: Test Generator

TestWriter

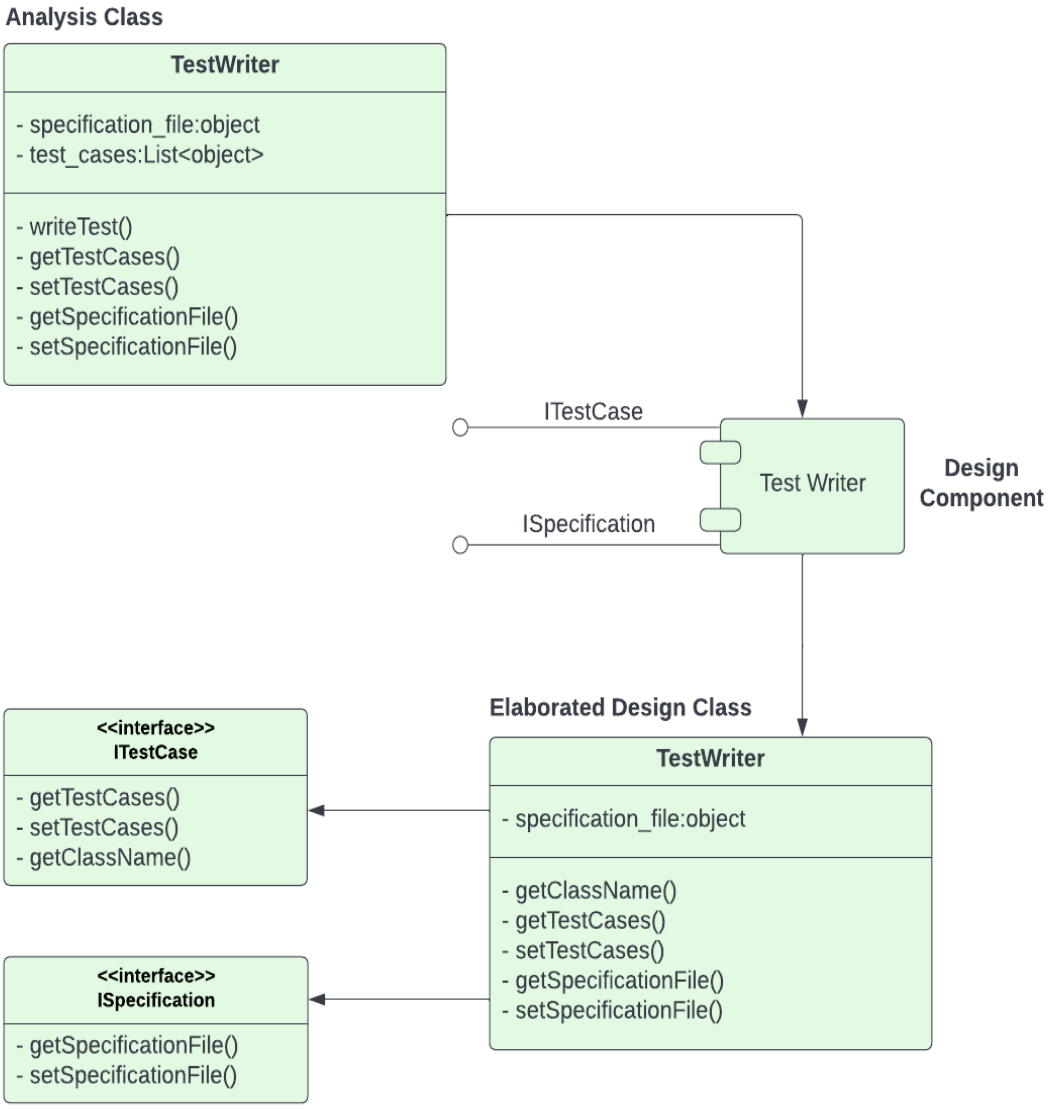


Figure 22: Test Writer

TestRunner

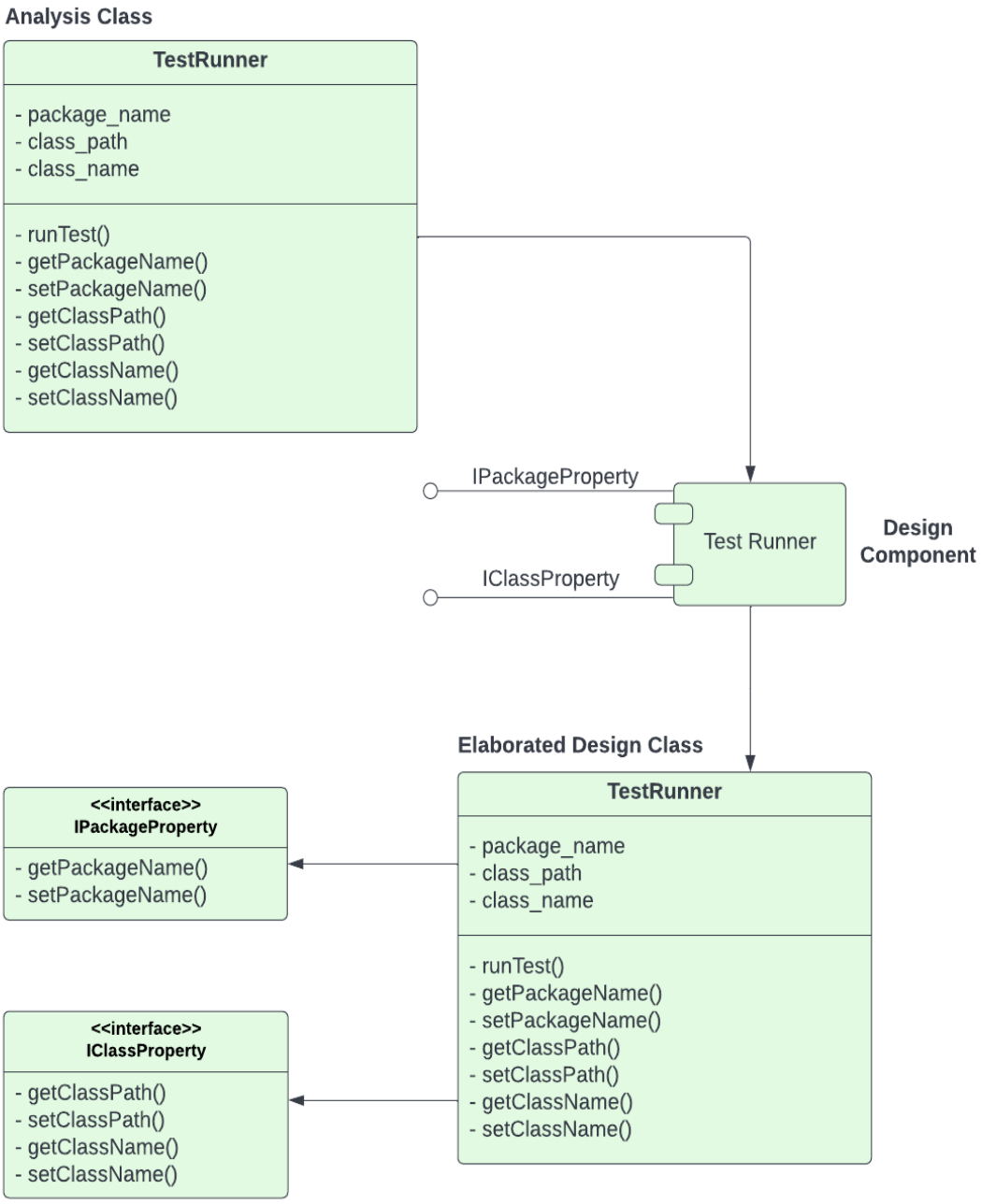


Figure 23: Test Runner

Chapter 8: User Interface Design

The below mentioned figure shows the homepage of the BOTest application.

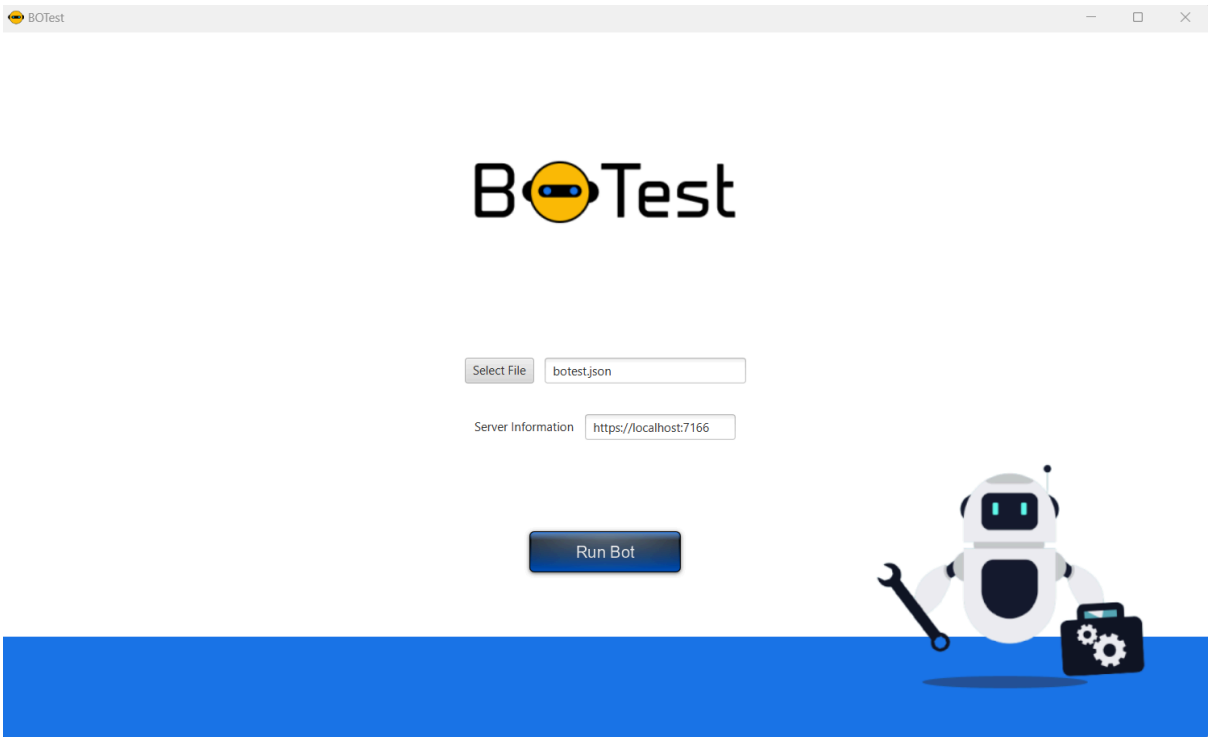


Figure 24: Homepage

The below figure shows the test report of the BOTest application which is served by allure.

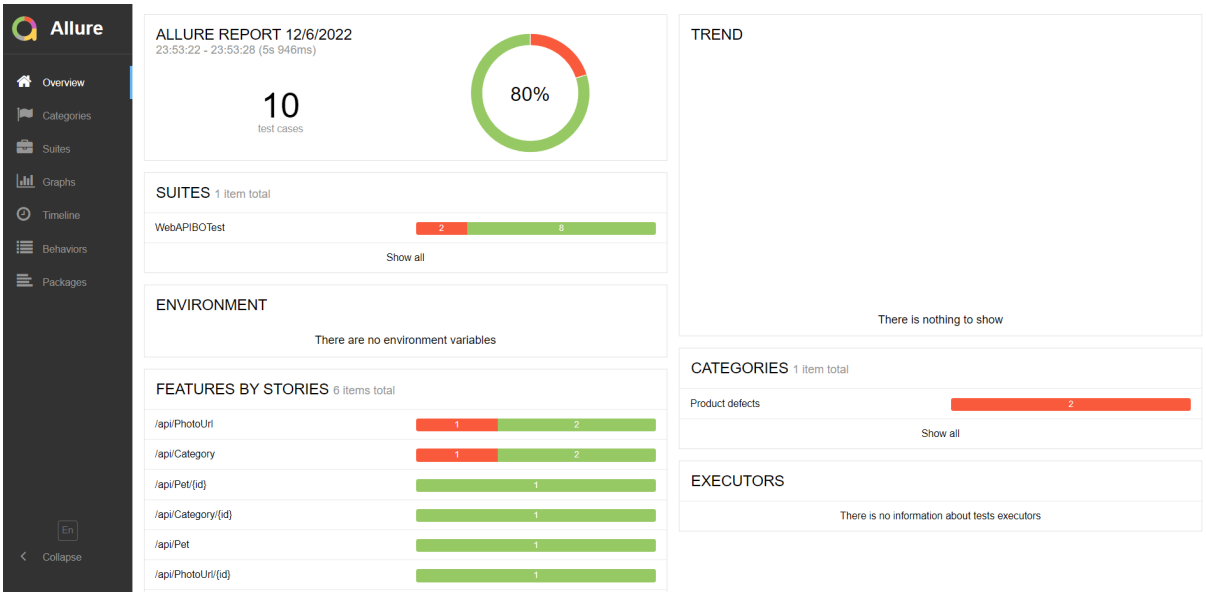


Figure 25: Test Report

Chapter 9: Testing Plan

9.1 High Level Testing Goals

boTEST will undergo high level testing which is popularly known as black-box testing technique. Black-box testing has already been defined in this document multiple times. In this phase, all the higher level components of the application will be tested to ensure that the application meets its requirements.

Goals of the higher level testing can be described as follows:

a. To demonstrate that boTEST meets the requirements.
b. To ensure that it produces valid and appropriate results/output.
c. To find any existing bugs or faults, design issues etc.
d. To ensure that the app runs smoothly without lags.
e. To ensure better user experience.

9.2 Test Cases

Test Id	Test Case	Input Test Data	Steps to be Executed	Expected Result
T1	Web API testing tool works for simple file	A simple API specification file	1. Give an open API specification file as input. 2. Choose Base URI.	A test report will be obtained.
T2	Web API testing tool works for GET request	A simple API specification file with GET request	1. Give an open API specification file with GET request. 2. Choose Base URI.	A test report will be obtained.
T3	Test model generation works correctly	A simple API specification file	1. Enter specification file. 2. Run the test generation model	Tool will generate valid test model
T4	Test cases generated	A simple API specification file	1. Enter specification file. 2. Run the test generation model	Tool will generate test cases.
T5	Test reports generated	A simple API specification	1. Enter specification file.	Tool will generate valid

	correctly	file	2. Run the test generation model	test reports according to data.
T6	Test case generation works correctly	A simple API specification file	1. Enter specification file. 2. Run the test generation model	Tool will generate valid test cases.
T7	UI remains responsive while testing	A simple API specification file	1. Enter specification file. 2. Run the bot.	Tool will generate valid test case for random testing
T8	Tool works for external apis	A simple API specification file with an external base URI	1. Enter specification file. 2. Run the test generation model	Tool will generate a test report.
T9	Test executes correctly	A simple API specification file	1. Enter specification file. 2. Run the test execution	Tool will generate valid test report
T10	Proper test report is generated	A simple API specification file	1. Enter specification file. 2. Run the test execution	Tool will generate test report with accurate info

Chapter 10: Conclusion

Learning new technology is always fascinating. Through this project, I am able to learn the concepts of web api testing automation. I had to read a number of research papers to get the grasp of the state of the art technologies and methodologies. I had to learn about APIs, RESTful APIs, Black Box Testing, Abstract Test Cases, Executable Test Cases, Junit, RestAssured, Allure Reports and many more. Besides, I will have to learn Java Spring to implement the web application.

I have tried my level best to analyze requirements correctly and make a design to implement the web application as practical as possible. From the SRS report on BOTest, the readers will get a clear and easy understanding of the overall system. This SRS document can be used effectively to maintain the software development life cycle. It will be very easy to conduct the whole project using SRS.

References

1. NordicAPIs https://nordicapis.com/20-impressive-api-economy-statistics/ [Last Accessed on 20th September, 2022 at 8:52 pm]
2. Martin-Lopez, A., Segura, S., & Ruiz-Cortés, A. (2021, July). RESTest: automated black-box testing of RESTful web APIs. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 682-685).
3. Software Testing https://www.guru99.com/software-testing-introduction-importance.html
   [Last Accessed on 20th September, 2022 at 8:52 pm]
4. White Box Testing https://www.guru99.com/white-box-testing.html
   [Last Accessed on 20th September, 2022 at 8:52 pm]
5. Black Box Testing https://www.guru99.com/black-box-testing.html
   [Last Accessed on 20th September, 2022 at 8:52 pm]
6. API https://www.mulesoft.com/resources/api/what-is-an-api
   [Last Accessed on 20th September, 2022 at 8:52 pm]
7. OpenAPI Specification https://swagger.io/specification/
   [Last Accessed on 20th September, 2022 at 8:52 pm]

GitHub Links

1. ahmedryanfaiyaz/_BOTest: A black box web API testing tool (github.com) This one has been created at a later time due to the architectural changes of the project structure.
2. ahmedryanfaiyaz/boTEST (github.com) This one was the primary URL. Before the architectural change, all the codes have been deposited here.

Appendix

User Manual

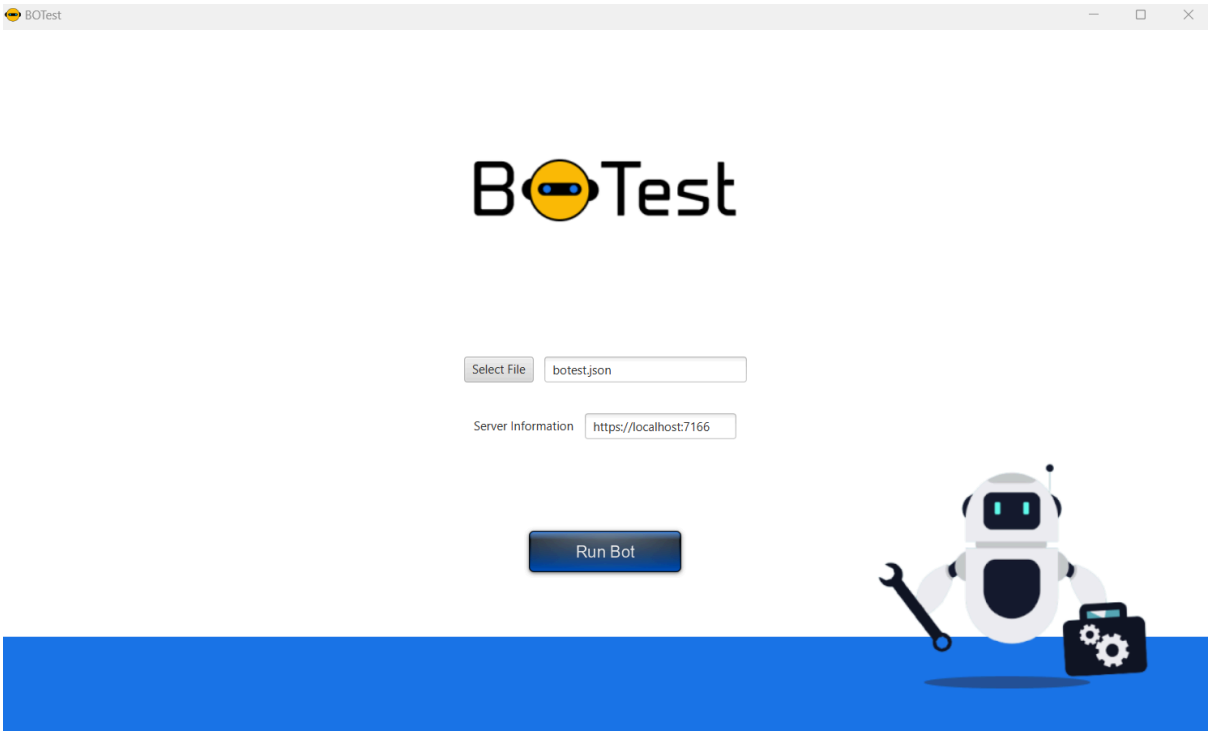
What does this BOTest tool do?

BOTest is a desktop tool for testing Web APIs automatically. This tool takes an Open API Specification file as input and also prompts for Base URI of the server. It then generates a test report through several steps like test configuration generation, executable test generation and test execution. Finally, a detailed report of the testing of the web API is obtained.

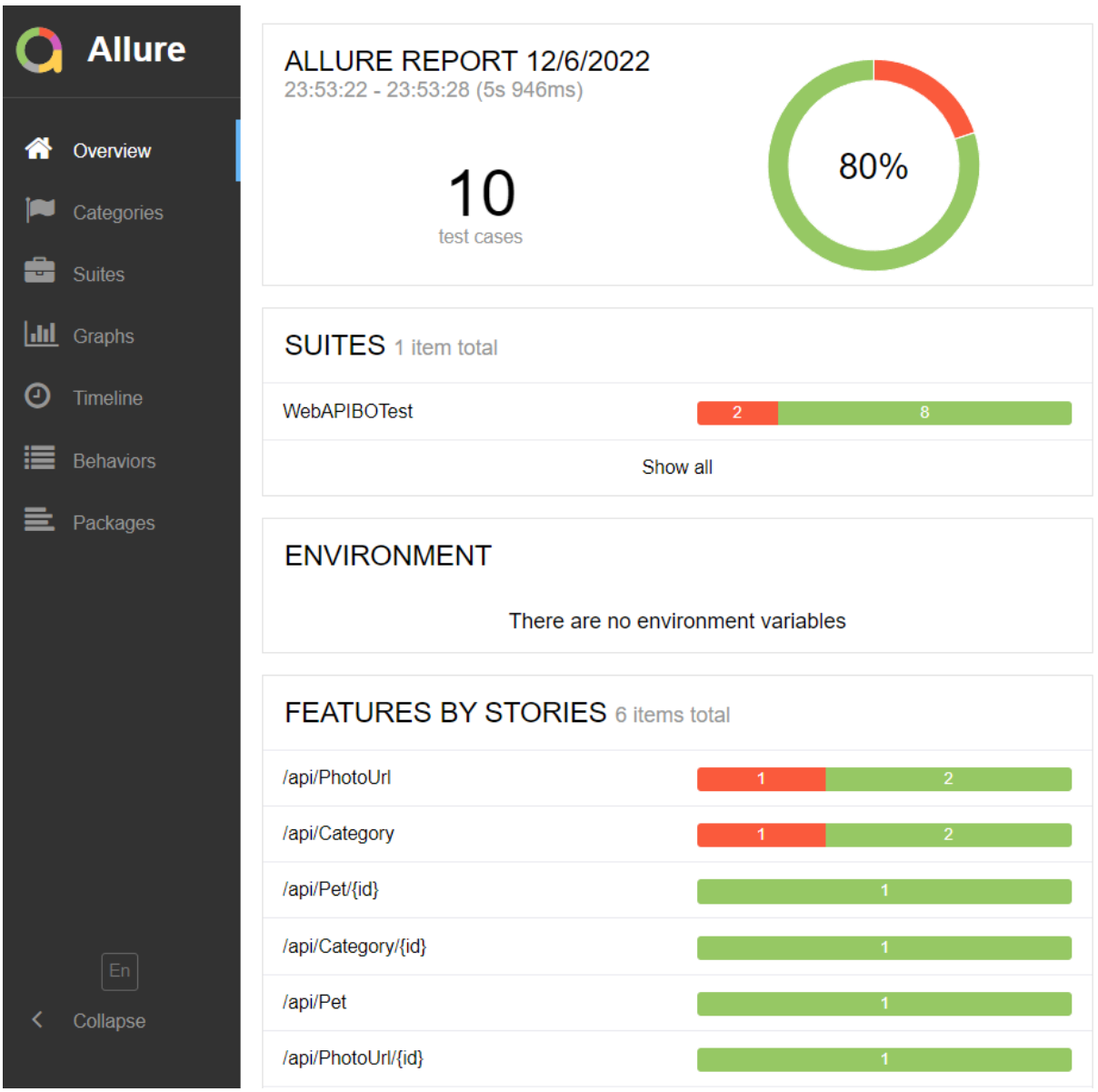
How to use this tool?

This tool is very easy to use. The user needs to give 2 things as input -

1. Open API Specification file of JSON format.
2. Base URI of the server on which the API resides.



Enter the above mentioned information to the BOT and click the run button. After that, enter the run bot button and you will get the test report!



This is a sample of the test report that will be generated by the BOTest tool.