RESTest: Automated Black-Box Testing of RESTful Web APIs

Software Systems Architecture and Integration

January 2021

< Student 1's name > (<Email>)
< Student 2's name > (<Email>)
< Student 3's name > (<Email>)
< Student 4's name > (<Email>)

Tutor:
Group number:

Link to the project in GitHub: https://github.com/isa-group/RESTest

VERSION HISTORY

Date	Version	Description	Participants
07/01/2021	1.0	- Initial version.	Sergio Segura José A. Parejo

Index

1 Introduction

RESTest is a framework for the automated generation and execution of tests in RESTful web APIs. Given the specification of an API in OAS (Open API Specification) format [1], the framework allows generating tests for the API automatically through different types of generators (e.g. random). The generated tests can be instantiated into executables test cases using tools like REST Assured [2] and Postman [3]. Once generated, the tests can be executed automatically. Test results can be browsed in a dashboard generated using the library Allure [4] (see example in Figure 1).

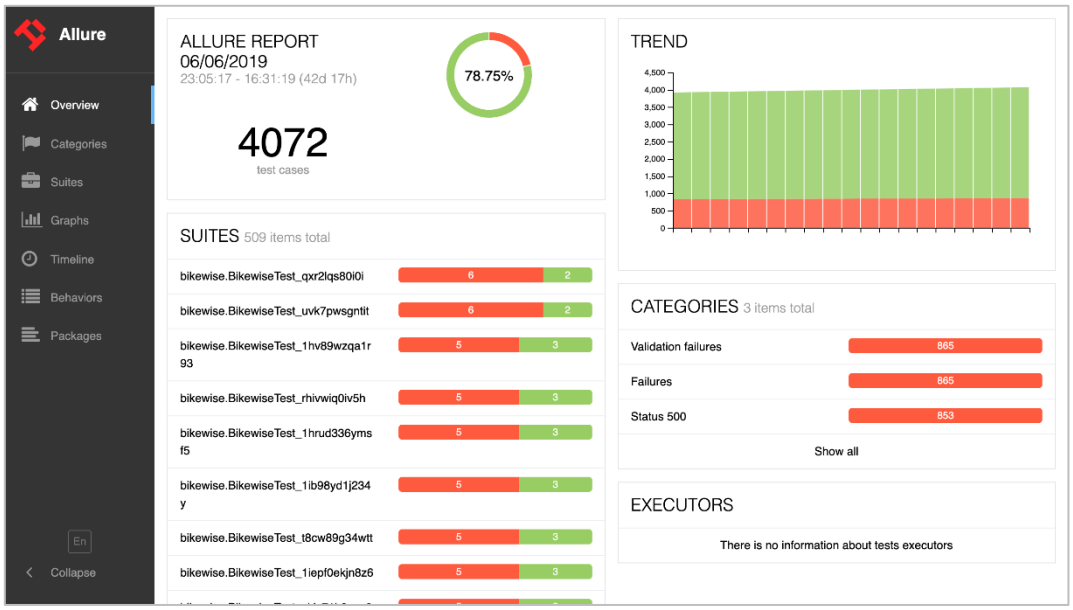

Figure 1. Allure test report

The tool is a research prototype developed by several researchers from the Applied Software Engineering group [5] of the University of Seville. Specifically, four researchers actively contribute to the project, being Alberto Martin, PhD student, the most active developer.

The project is developed in Java and uses Maven for dependency management. It has about 7K lines of code and 120 issues (about 30% of them open). The project is linked to the continuous integration platform CircleCI [6] and the source code analysis platform SonarCloud [7]. It is an active project released under the LGPL-3.0 license [8].

2 Overview

In RESTest, a test case represents a single call to an API operation and a set of assertions in the response. Next, we explain the basic workflow of RESTest, depicted in Figure 2.

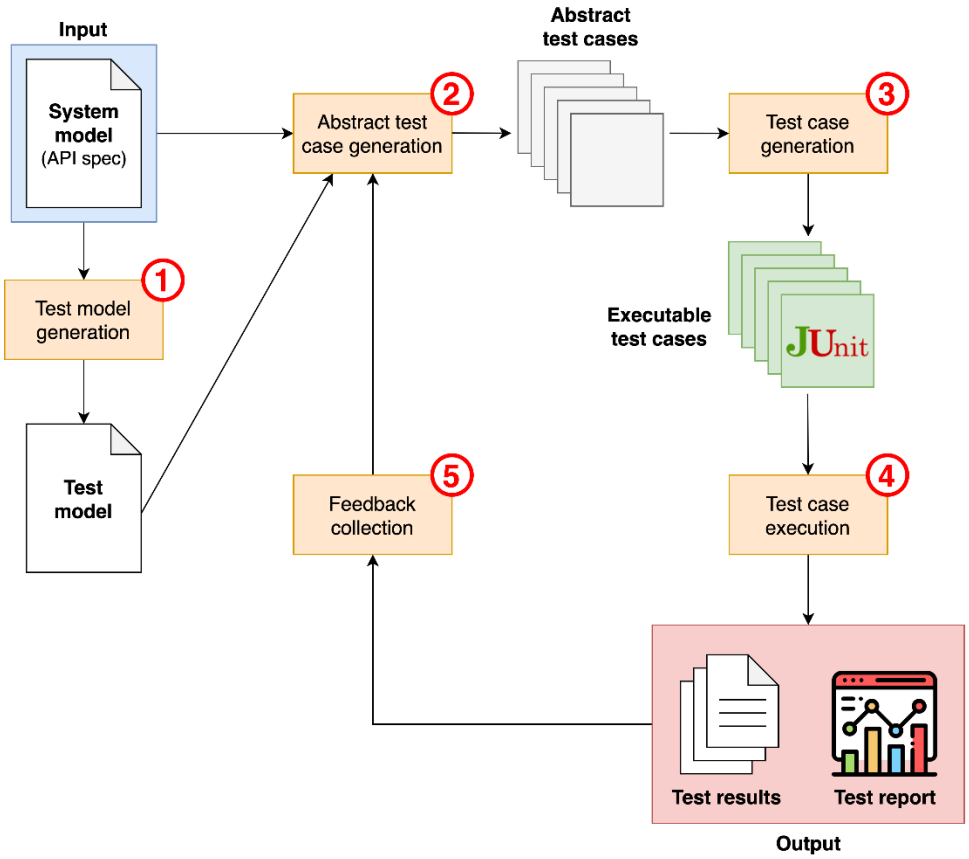

Figure 2. RESTest workflow

1. **Test model generation**. RESTest follows a model-based testing approach. Two models are used: the *system model* (i.e., the API specification), and the so-called *test configuration model*, consisting of a configuration file in YAML notation. The test model contains all test-related configuration settings for the API under test, and it may be manually augmented to tailor the testing process, for example, to specify authentication details (e.g., API keys). The test model also specifies the test data to be used for each parameter, which may include data dictionaries or test data generators (e.g., airport or currency codes).

2. **Abstract test case generation**. Test cases are derived from the system and test models using one or more testing techniques. These test cases are *abstract* or platform-independent, meaning that they can be later transformed into executable test cases for specific testing frameworks and programming languages.

3. **Test case generation**. Abstract test cases are *instantiated* into executable test cases using specific testing frameworks and libraries such as REST Assured. Figure 3 shows a test case for the API of Amadeus automatically generated by RESTest.

4. **Test case execution**. Test cases are optionally executed and the test results are exported to a machine-readable format and reported to the user, e.g., in a dashboard, using a test reporting framework like Allure (Figure 1).

5. **Feedback collection**. Test generators can react to the test outputs to create more sophisticated test cases, for example, applying search-based techniques in order to maximize the API coverage (e.g., status codes and response bodies).

```
@Test
public void test_qxtv3639pzkx_getMultiHotelOffers() {
    NominalOrFaultyTestCaseFilter nominalOrFaultyTestCaseFilter =
            new NominalOrFaultyTestCaseFilter(false, true, "none");

    try {
        Response response = RestAssured
        .given()
            .log().all()
            .queryParam("view", "FULL")
            .queryParam("boardType", "ROOM_ONLY")
            .queryParam("chains", "WW,HS,6C,BW")
            .queryParam("hotelIds", "HSORYABR")
            .queryParam("adults", "6")
            .queryParam("currency", "XAG")
            .queryParam("radius", "57")
            .queryParam("radiusUnit", "KM")
            .queryParam("priceRange", "26-267")
            .filter(statusCode5XXFilter)
            .filter(nominalOrFaultyTestCaseFilter)
            .filter(validationFilter)
        .when()
            .get("/shopping/hotel-offers");

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

Figure 3. Automatically generated test case (Amadeus API)

3 Stakeholders

Being a research prototype, the project's developers assume different roles. Specifically, according to Woods' stakeholder taxonomy [9], we identified the following types of participants in the project:

- **Developers**. Responsible for the development of the framework. According to the statistics of the project, there is clearly a main developer, Alberto Martín. The other three participants make sporadic contributions to the project.

- **Testers**. There is not a specific testing team. The developers themselves create issues when they detect a problem and create or modify the necessary test cases to solve them.

- **Support staff**. It is mainly carried out by two of the developers, Alberto Martín and José Ramón Fernández, to whom most issues are assigned. The support is done through the management of issues which includes tasks such as the triage, classification, labelling and monitoring of issues.

- **Communicators**. They are responsible for explaining the system to other potential participants. Once again, this work is done by the developers themselves through the project documentation (readme.md and wiki) and research papers.

- **Users**. They are the users of the framework, interested in using it to test their own APIs, or to compare it with similar tools.

4 Views

The architectural design of RESTest can be described using the following views.

4.1 Context view

This view defines the relationship and dependencies between the system and its environment. This includes how the system interacts with other systems, organizations, or people [9].

Figure 4 shows a context diagram of RESTest. The project is hosted in GitHub, currently used for version control, issue tracking, and hosting the documentation of the project. The GitHub project is linked to the cloud services of CircleCI, for continuous integration, and SonarCloud, for tracking code quality. The main developers are members of the ISA research group at the University of Seville. The tool is mostly used by researchers although it also targets practitioners.

As previously mentioned, RESTest is written in Java and use Maven for handling dependencies automatically. The main dependencies of the project are the RESTful API testing libraries REST Assured and Postman, the Swagger java parser, the test reporting library Allure, and the library IDLReasoner [10], developed by the same authors, for the automated analysis of IDL specifications describing the inter-parameter dependencies of the API.

There exist some related tools with the same goal as RESTest—automated generation of test cases—but that differ in their approach. RESTler [11], develop by Microsoft, also generates random test cases for RESTful APIs but, in contrast to RESTest, it supports the

generation of stateful test cases composed of sequences of calls to the API. EvoMaster [12] is white-box tool that requires access to the source code of the API under test.

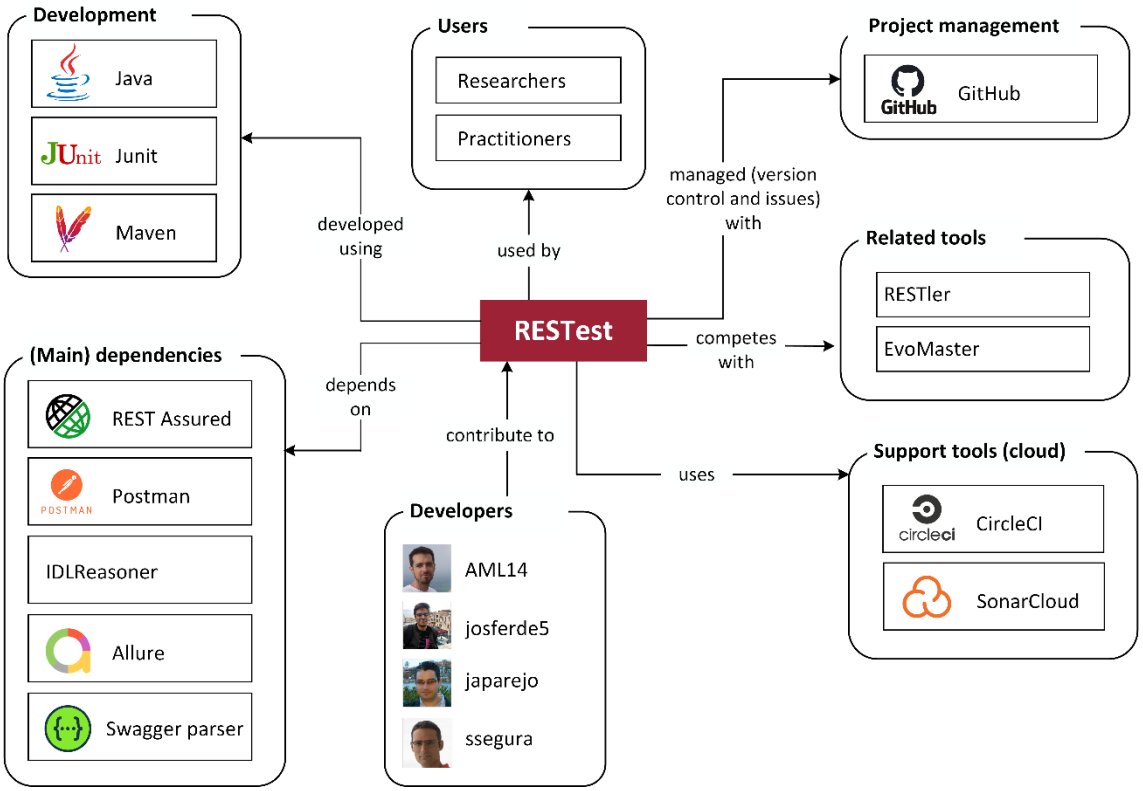

Figure 4. Context diagram

4.2 Scenarios view

We identify the following use case scenarios (depicted in Figure 5):

- **Generation of test cases.** This is the main functionality of the framework. Given the OAS specification [1] of an API, and a test configuration file, RESTest supports the generation of an unlimited number of abstract test cases. Once generated, Abstract test cases are instantiated into executable test cases using specific testing frameworks and libraries. RESTest currently supports the generation of executable test cases using REST Assured [2], a Java library for testing RESTful services developed as a JUnit extension, and Postman [3]. Some optional features can be optionally enabled during the generation such as the collection of statistics or measuring API coverage (saved in CSV format).

- **Execution of test cases.** Once generated and saved, test cases can be easily executed using the RESTAssured library. Since RESTAssured is an extension of the popular Junit library, test cases can be easily lunched from the GUI of standards IDE such as Eclipse and IntelliJ.

- **Online testing**. Test execution can be done either offline or online. In offline testing, test case generation and execution are independent tasks. For example, test cases can be generated once, and then be executed many times as a part of regression testing. Also, test generation and test execution can be performed on different machines and at different times. In online testing, test case generation and execution are interleaved. This enables, for example, fully autonomous testing of RESTful web APIs, e.g., generating and executing test cases 24/7 as a part of a Continuous Integration (CI) setup.

- **Generation of default test configuration models.** Test case generation in RESTest is driven by the specification of the service (OAS) plus a so-called *test configuration model* in Swagger format. RESTest supports the generation of a default test configuration file using the OAS specification of the API as an input. Once generated, the user can adjust the configuration settings by editing the file, for example, defining new test data dictionaries.

- **Test report generation.** Once test cases are executed and results collected a test case report can be generated using the Allure framework (see Figure 1). This can be done automatically in online testing.

Figure 5 depicts a graphical representation of the main use cases described above using an UML use case diagram.

4.3 Functional view

From a functional perspective, RESTest can be logically divided into the following components:

- **Test case generators**. These are the main components of the tool as they are the ones actually generating test cases. At the time of writing this report, RESTest integrates two types of test case generators: random and constraint-based. The random test case generator supports the generation of (pseudo) random test cases by generating random values (typically from a predefined set of words) for each parameter. The constraint-based test case generator is significantly more complex since it also makes sure that the inter-parameter dependencies among input parameters (if any) are satisfied. The constraint-based generator uses the external library IDLReasoner [] for the automated analysis of inter-parameter dependencies described using the language Inter-Parameter Dependency Language (IDL).

- **Test data generators**. These are the classes used for the generation of test input data: strings, dates, numbers, JSON objects, etc. There are different types of generators including random, boundary-value, and data dictionaries (i.e. predefined set of input values to choose from).

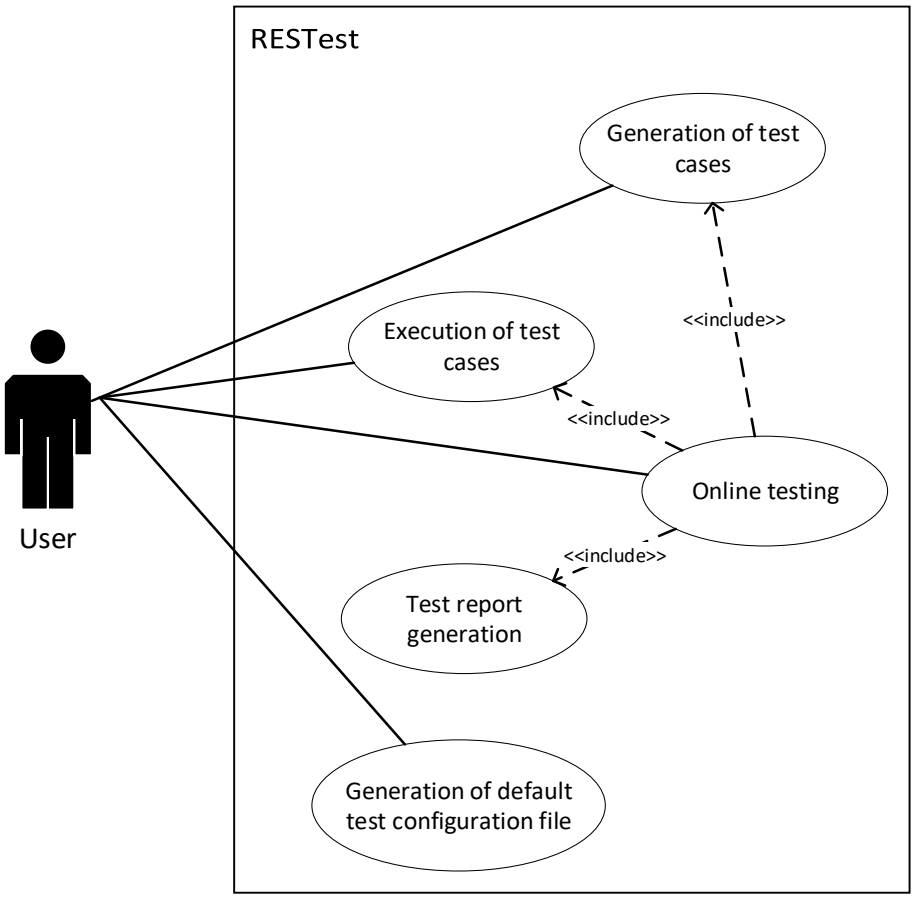

Figure 5. UML use case diagram

- **Test case mutator**. RESTest generates both nominal and faulty test cases. *Nominal test cases* aim to test the API with valid inputs, i.e., those conforming to the API specification. *Faulty test cases* check the ability of the API to handle invalid inputs, and therefore they expect a client error as a response (i.e., 4XX status codes). This component generates faulty test cases by creating faulty variants (i.e., mutants) of nominal test cases. For example, RESTest supports the automated generation of faulty test cases by excluding mandatory parameters, using out-of-range values (e.g., assigning a string to an integer parameter), and violating the JSON schema of the request body, among others.

- **Witters**. Abstract test cases are instantiated into executable test cases using different writters. Currently, RESTest supports the generation of executable test cases using the framework REST Assured and Postman.

- **Runners**. Runners manage the test case generation and test execution workflow. RESTest currently integrates a single runner that supports the generation and execution of test cases using a specific test case generator and one or more writers.

- **Result reporting**. Test results can be saved for later analysis. Currently, RESTest supports the generation of CSV statistics and Allure test reports (see Figure 1).

- **Utilities**. This includes several utilities used from the other components such as a CSV reader/writer, a timer, a class loader, etc.

Figure 6 shows an UML component diagram illustrating the main components of RESTest and their relationships.

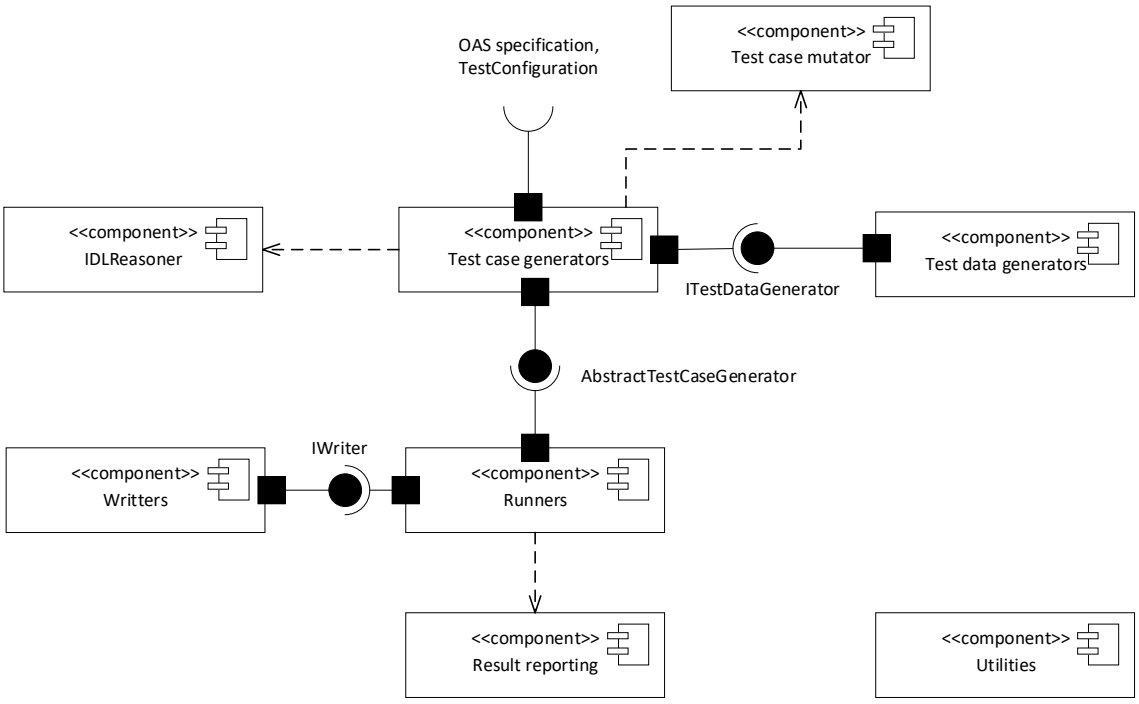

Figure 6. UML component diagram

4.4 Deployment view

Figure 7 depicts an UML deployment diagram of RESTest. As illustrated, deployment only requires a Java JDK installation and, in the case that the APIs under test are in remote servers, Internet connection. Allure reports are generated in HTML + JavaScript format and therefore can be browse without a specific web server. However, if the reports are expected to be available on the Internet, a standard web server should be installed.

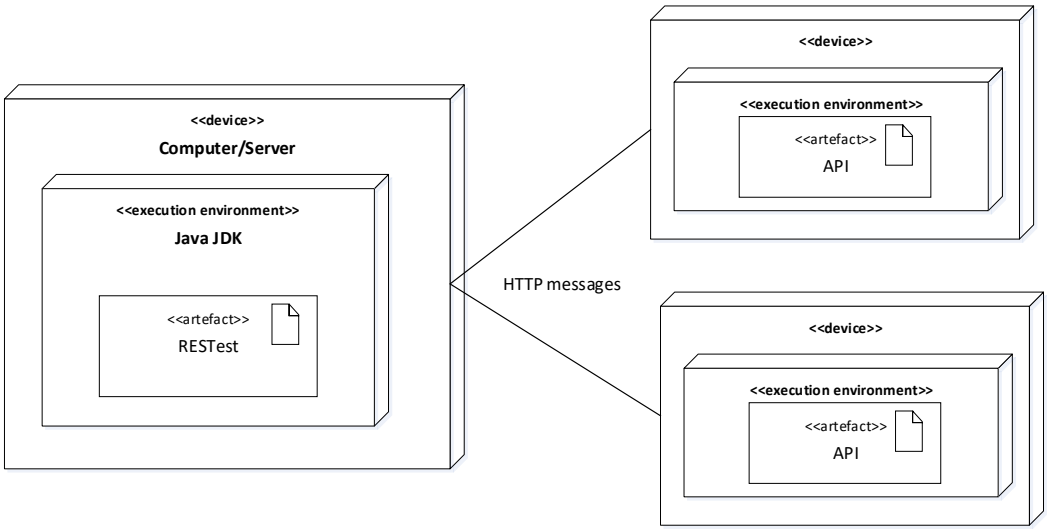

Figure 7. UML deployment diagram

4.5 Development view

Figure 8 shows an UML package diagram of RESTest automatically generated with the tool UML Lab [13]. The code is currently distributed along 25 packages, 88 Java classes (including interfaces) and about 7K lines of code.

Package name (es.us.isa.restest.*)	Description	Classes	LoC
configuration	Test configuration file management	12	823
coverage	Test coverage measurement	6	928
generators	Test case generators	3	512
inputs	Test data generators	15	1.278
main	Main files for running the application	2	273
mutation	Classes for mutating test cases	10	330
reporting	Test case reports generation	2	224
runners	Runners for online testing	1	91
specification	Swagger parser	2	170
testcases	Test cases management	20	1.580
util	Utilities	15	686
	Total	88	6.895

Table 1. Main packages

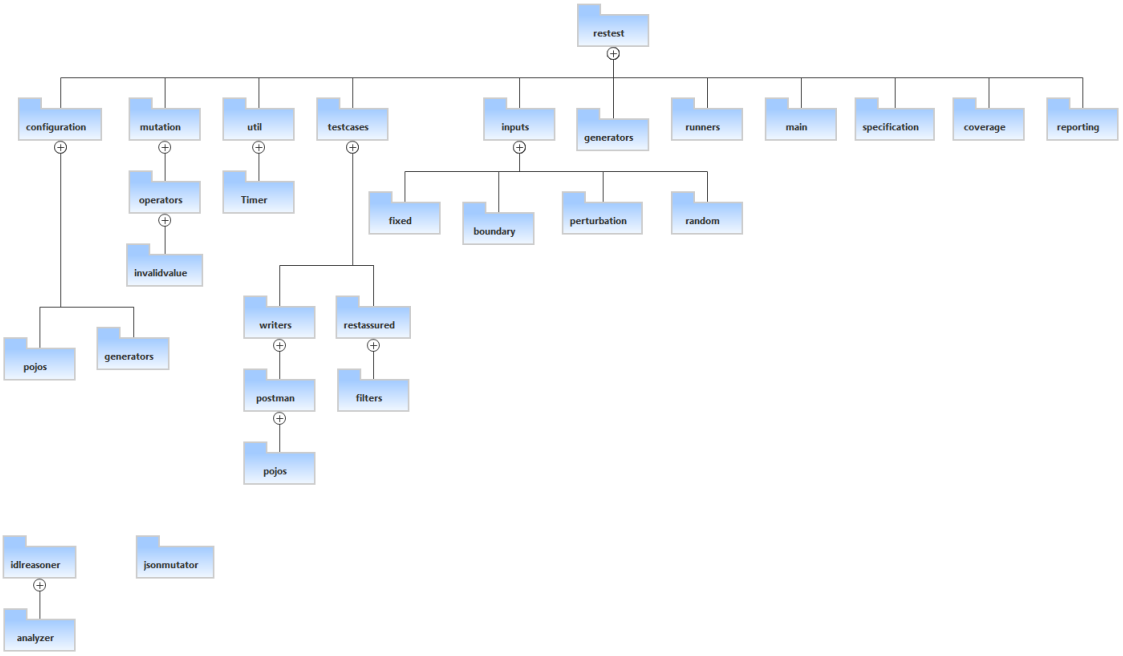

Figure 8. UML package diagram

The project is developed in Java and uses Maven for dependency management. Specifically, the project defines dependencies with the following main libraries:

- **Swagger**. Library for parsing and manipulating Swagger specifications in JSON format.
- **Rest Assured**. Testing library for REST APIs (based on Junit).
- **Junit.** Unit testing library.
- **Jackson**. Library for handling JSON and YAML files.
- **Allure**. Generation of graphical test reports.
- **Mockito**. Test mocking library.
- **Generex.** Generation of string from regular expressions.
- **Extjwnl.** Generation of random words from the WordNet English dictionary.
- **Apache Log4j**. Logging.
- **JSONMutator**. Library for mutating JSON objects.
- **IDLReasoner**. Library for the automated analysis of inter-parameter dependencies in Web APIs.

RESTest is linked to the continuous integration platform CircleCI [6], which automatically run the test cases of the project every time a commit is pushed. RESTest is also linked to the analysis tool SonarCloud [7], which provide developers and user with rich information about the project including vulnerabilities, code coverage, and code metrics.

5 Variability and extension points

Variability points define places where the system can adopt different alternatives (so-called variants). We identify the following variability points in RESTest:

- **Test data generators**. User can choose among a variety of test data generators mostly classified into two groups: random generators (integers, dates, strings, etc.) and data dictionaries: those selecting one or more values from a pre-defined set of values.

- **Test case generators**. The system can be configured to use different test case generators. The current options are random generation and constraint-based generation.

- **Test case writers**. Test cases can be serialized using ether the REST Assured framework or Postman.

- **Reporting.** Users can select whether to create test results reports using the Allure framework, CSV, or none of them.

Extension points are those part of the system where the system is expected to be extended in the future. In this case, we believe that the previous variation points are also extension points since the system is likely to be extended with new test data generators, test case generators, test writers and test report formats. Additionally, we identify the following extension point in RESTest:

- **API specification**. RESTest currently support API specification using the OAS format only (v2 and v3). However, we think that the system is likely to be extended in the future to support other languages like RAML or future versions of OAS.

6 Analysis of quality attributes

We reviewed the following quality attributes.

- **Performance.** This refers to the computer resources used by the system such as execution time, memory consumption, or energy usage. Performance does not seem to be a critical issue in RESTest. Hundreds of test cases can be generated in just a few seconds without identifying memory issues. The whole test suite, composed of 190 test cases, is executed in about 1 minute (Figure 9). There is room for improvement though. For example, it would be nice to have the possibility of executing the generated test cases directly, without writing them to disk first, which would reduce execution time drastically.

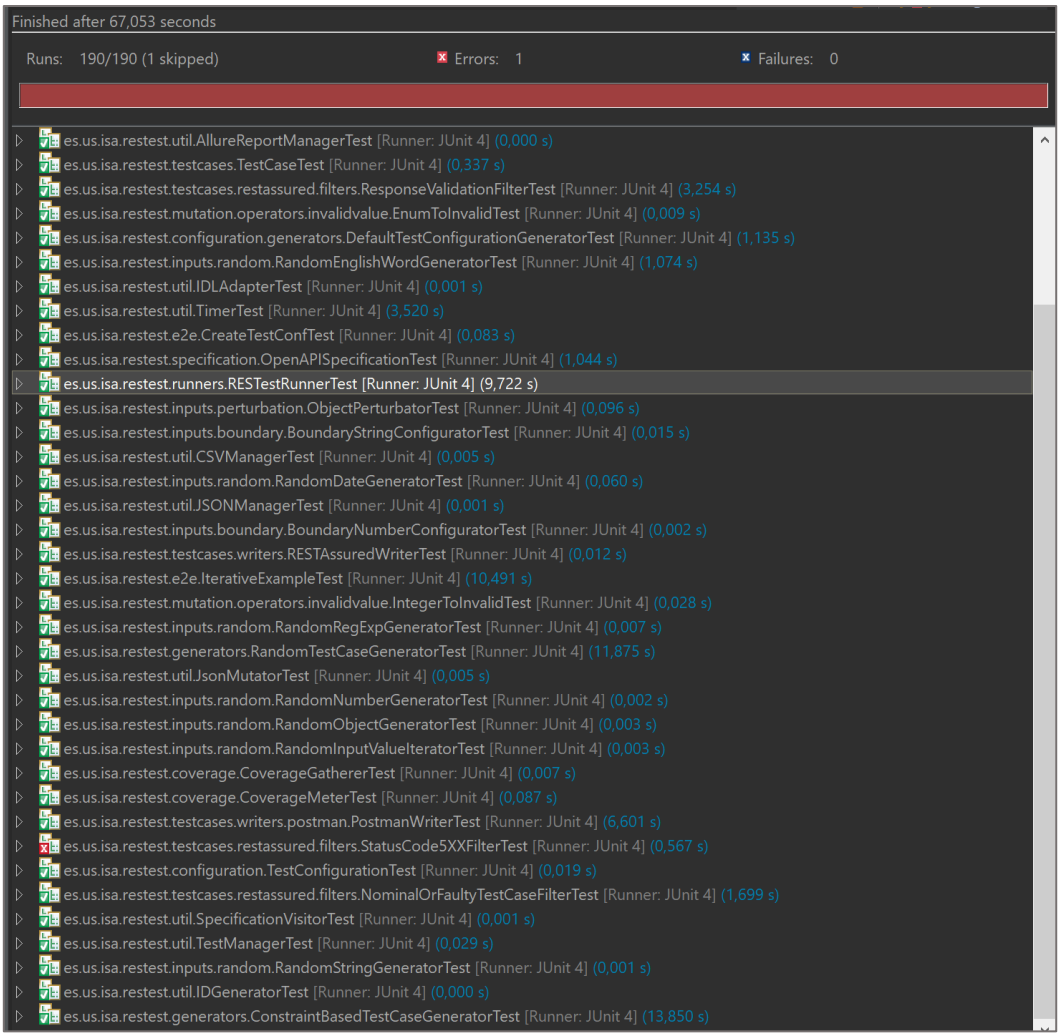

Figure 9. Unit test results

- **Usability**. This measures how easy is for users to use the system. Usability in RESTest is reasonably good due to the rich documentation of the project. Installation is straightforward as it simply requires downloading the necessary libraries using Maven. The test suite serves a good reference for users. We did not identify GitHub issues related to users' usability. From the developer perspective, however, the project lacks updated documentation for extending the tool.

- **Maintainability**. This refers to the ease with which the system can be changed, for example, to introduce improvements or fix bugs. We think that maintainability in RESTest is significantly good thanks to its modular design, good documentation and large test suite, which serves as a good entry point for newcomers. This is in line with the maintainability score in SonarCloud, rated with 'A'.

- **Reusability**. This refers to the ease with which parts of the system can be reused. We think that resuabilty in RESTest is fair due to is modular design and the use of design patterns (ex. Test data factory). Certain parts of the systems could be potentially used in third-party projects like, for example, test data generators.

- **Reliability**. This attribute refers to the ability of the system to work as expected. Reliability is affected by the defects found in the system and its the ability to handle failures. This attribute is rated with an 'A' in SonarCloud since there are no currently open bugs in the project. It is also noteworthy that the project has 190 unit test cases (all but one passes), with a line coverage of 78.6% and a condition coverage of 68.7%, which is significantly good considering the size of the application.

- **Security**. This refers to the ability of the system to avoid potential security risks like unauthorized access to restricted data. RESTest is not intended to be directly exposed on the web and therefore we think that this is a minor concern. However, we think that RESTest could be potentially used to attack servers by sending numerous API requests. Security in RESTest is rated in SonarCloud with an 'A'.

7 Suggestions for improvement

Based on our analysis of the project's architecture, we suggest the following improvements:

- **Unit tests**. Test cases should be reviewed to fix the failure identified in the test case "`StatusCode5XXFilterTest.java`". We also suggest arranging the test cases in different test suites to enable running tests more selectively. There are a couple of test classes that took significantly longer to execute than the others (between 9 and 14 seconds) and this should be revised too. Increasing code coverage, especially in the packages "reporting" and "specification" (Figure 10), would be desirable.

- **Documentation**. User documentation is reasonably good, but there is room for improvement. Step by step "hello world" examples would be helpful. Documentation for developer is insufficient and not updated.

- **Design**. The current implementation is highly coupled to OAS, which is a serious threat. We suggest decoupling the tool from the API specific specification format used.

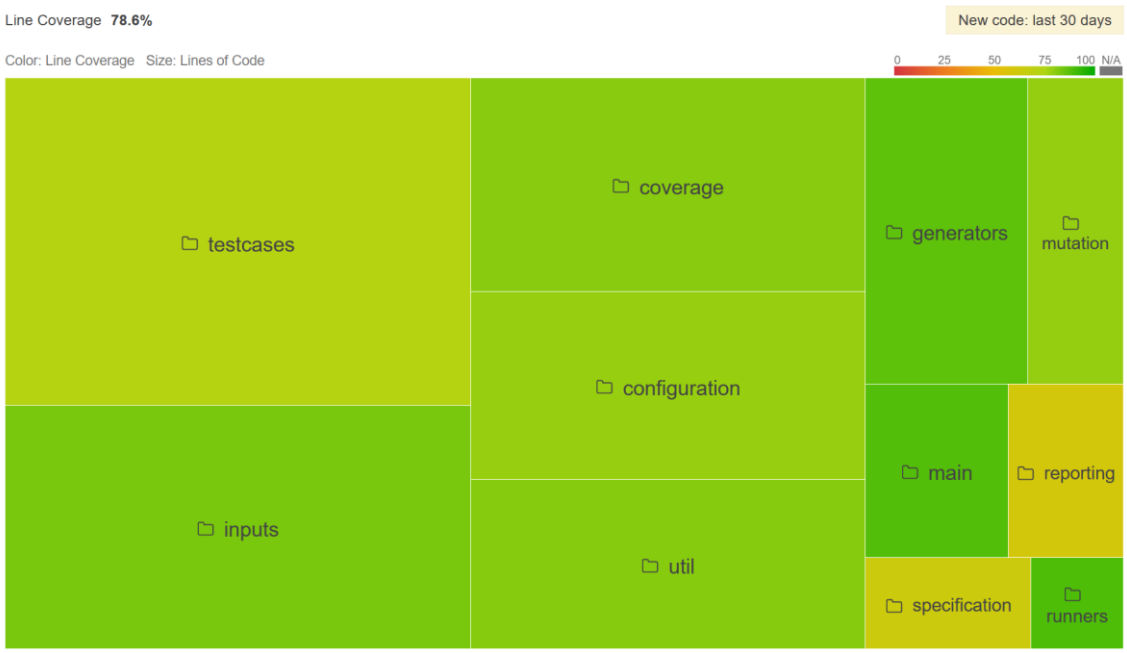

Figure 10. Line coverage

8 Contributions to the project

We have made the following contributions to RESTest:

1. **Feature request**: We have changed the names of the categories associated to the Allure test reporting framework, as requested in Issue #149 [1]by @AML14.

2. **Bug fix**: We have fixed a bug that was causing two unhandled exceptions to be thrown in some executions (issue #94[2]).

3. **Test coverage**: We have increased the overall test coverage of the project by creating new tests for the classes `TestCase` and `RESTestRunner`. Pull request #142[3]

9 Conclusions

In this project, we have documented the architectural design of RESTest, an open source framework for the automated generation of test cases for RESTful APIs developed by researchers at the University of Seville. The tool has a modular and extensible design showing a good performance in terms of standard quality metrics. However, there is room for improvement in aspects like documentation, test case arrangement, and

[1] https://github.com/isa-group/RESTest/issues/149
[2] https://github.com/isa-group/RESTest/issues/94
[3] https://github.com/isa-group/RESTest/pull/142

design. The main limitation is probably the coupling to Swagger (OAS), which could cause troubles in the future for adopting new API specification languages. We have contributed to RESTest in several ways including the improvement of the documentation, a feature request, several bug fixes and two pull request aimed to increase test coverage.

References

[1] "OpenAPI Specification - Version 3.0.3 | Swagger." https://swagger.io/specification/ (accessed Dec. 28, 2020).

[2] "REST Assured." https://rest-assured.io/ (accessed Dec. 28, 2020).

[3] "Postman | The Collaboration Platform for API Development." https://www.postman.com/ (accessed Dec. 28, 2020).

[4] "Allure | Test report and framework for writing self-documented tests." http://allure.qatools.ru/ (accessed Dec. 28, 2020).

[5] "ISA Group." https://www.isa.us.es/3.0/ (accessed Dec. 28, 2020).

[6] "Continuous Integration and Delivery - CircleCI." https://circleci.com/ (accessed Dec. 28, 2020).

[7] "Automatic Code Review, Testing, Inspection & Auditing | SonarCloud." https://sonarcloud.io/ (accessed Dec. 28, 2020).

[8] "isa-group/RESTest: RESTest: Automated Black-Box Testing of RESTful Web APIs." https://github.com/isa-group/RESTest (accessed Dec. 28, 2020).

[9] N. Rozanski and E. Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, 2nd ed. Addison-Wesley Professional, 2011.

[10] "isa-group/IDLReasoner: IDLReasoner: A MiniZinc-based Java library for analyzing IDL specifications." https://github.com/isa-group/IDLReasoner (accessed Dec. 28, 2020).

[11] "microsoft/restler-fuzzer: RESTler is the first stateful REST API fuzzing tool for automatically testing cloud services through their REST APIs and finding security and reliability bugs in these services." https://github.com/microsoft/restler-fuzzer (accessed Dec. 28, 2020).

[12] "EMResearch/EvoMaster: A tool for automatically generating system-level test cases. Currently targeting REST APIs." https://github.com/EMResearch/EvoMaster (accessed Dec. 28, 2020).

[13] "UML Lab - Yatta Solutions." https://www.uml-lab.com/en/uml-lab/ (accessed Dec. 28, 2020).