EvoMaster: A Tool for Automatically Generating System-Level Test Cases - Analysis

doi: <https://ieeexplore.ieee.org/document/8367066>

Bledea Mihaela Alexandra Bretan Cezar Alexandru Boanca Tudor

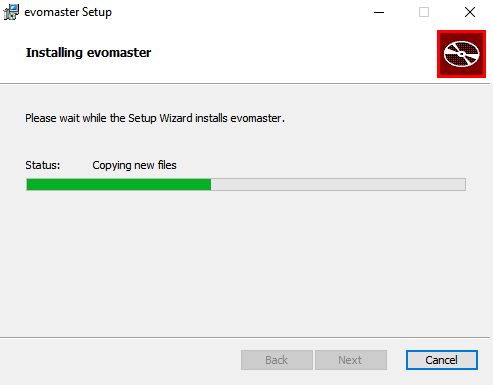
1. **Installation**

The EvoMaster white-box testing system consists of three components: the Core (server running the evolutionary algorithm), the SUT (i.e., a SpringBoot REST API) and the Driver (an additional module that needs to be attached to the SUT for opening a TCP socket for communication with the Core).

The Core can be installed using one of the installers provided by the developers (<https://github.com/EMResearch/EvoMaster/releases/tag/v1.6.1>), and the addition of the installation folder to the operating system’s PATH environment variable is recommended for more convenient usage of the CLI.

1. **Setting up the tool for usage**

The first step is to install the core tool, which as specified can be done using one of the installers provided by the developers.



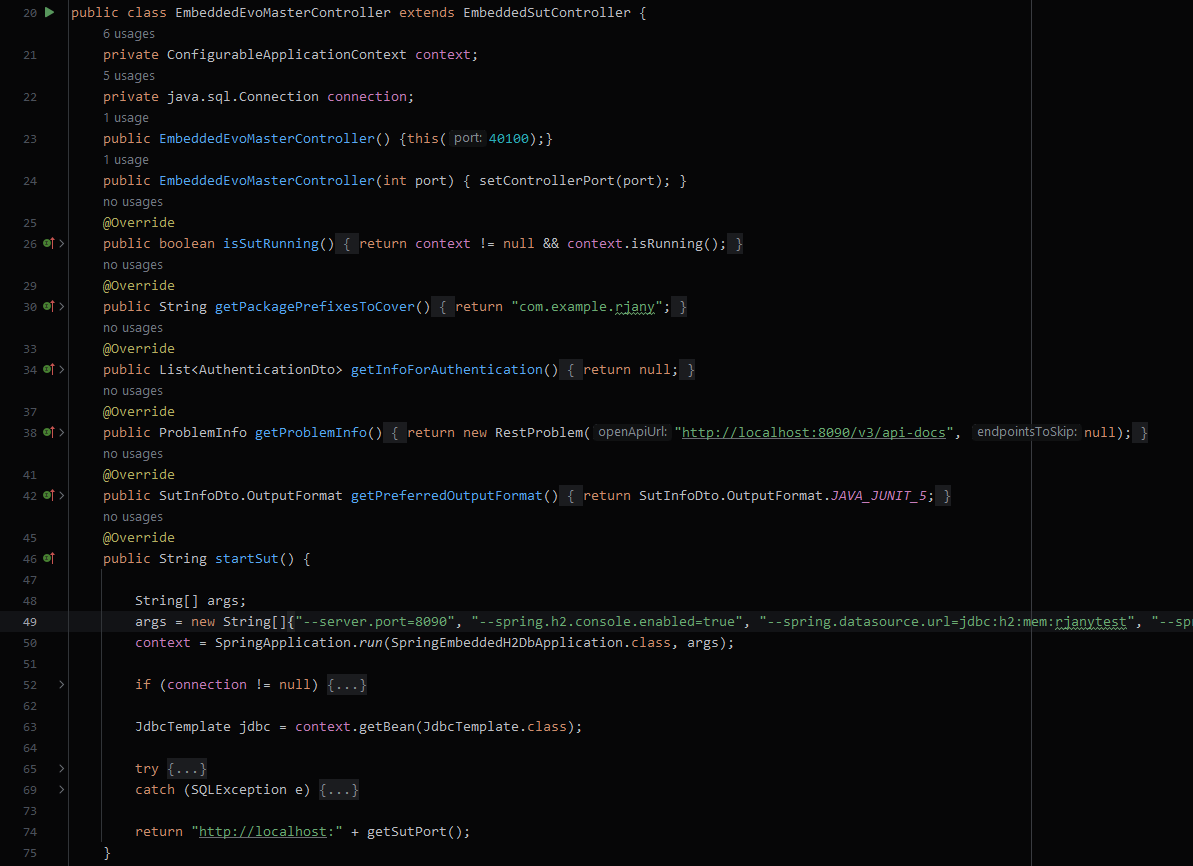
The tool can be used both locally and externally (in a different JVM), however the steps are different. Moving forward, we are going to describe how to use the tool for a local project. Details on how it works if you wish for an external setup can be found on the tool’s GitHub page (<https://github.com/EMResearch/EvoMaster>).

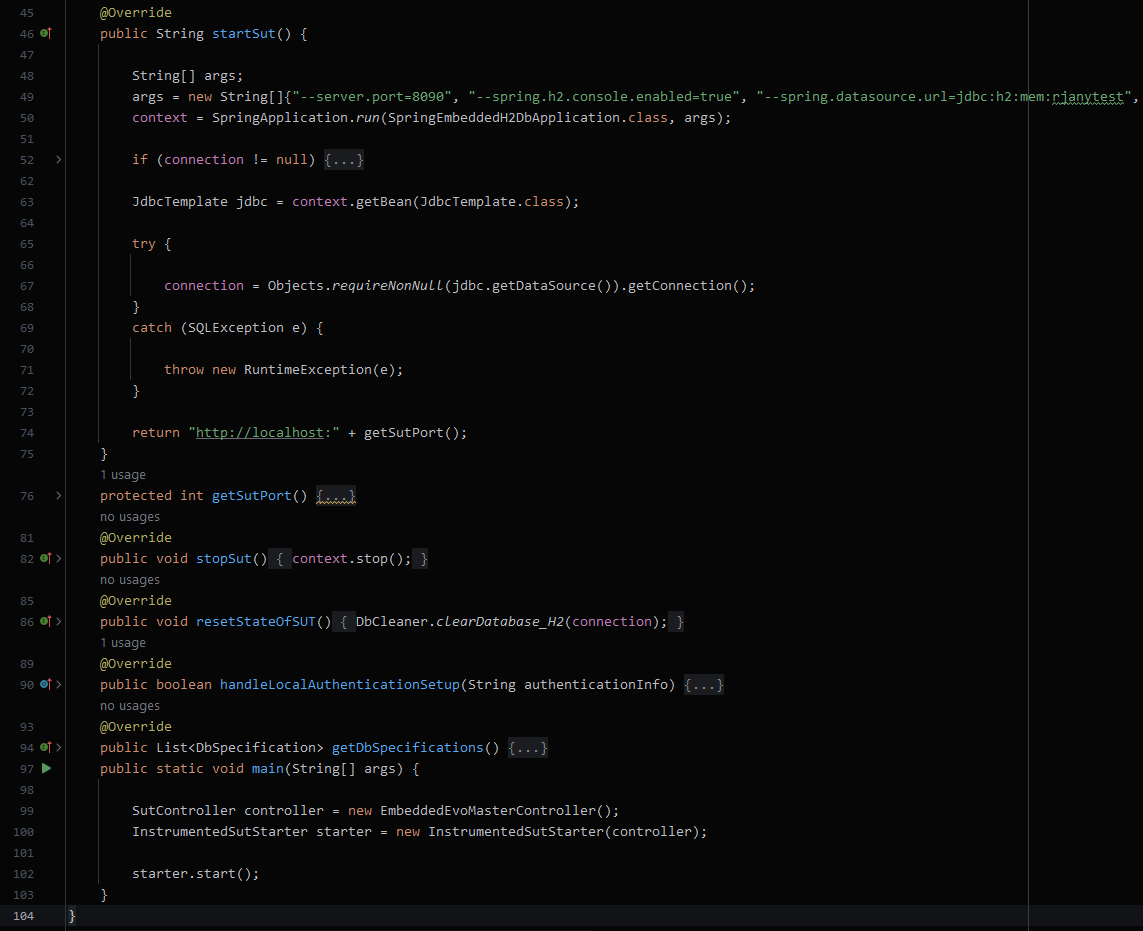
In order to get the tool working locally, the first step is to have a working REST API and implement a driver for the tool, which is a subclass of the EmbeddedSutController, class that is found in the tool’s jar files.

The noteworthy methods that need to be implemented are:

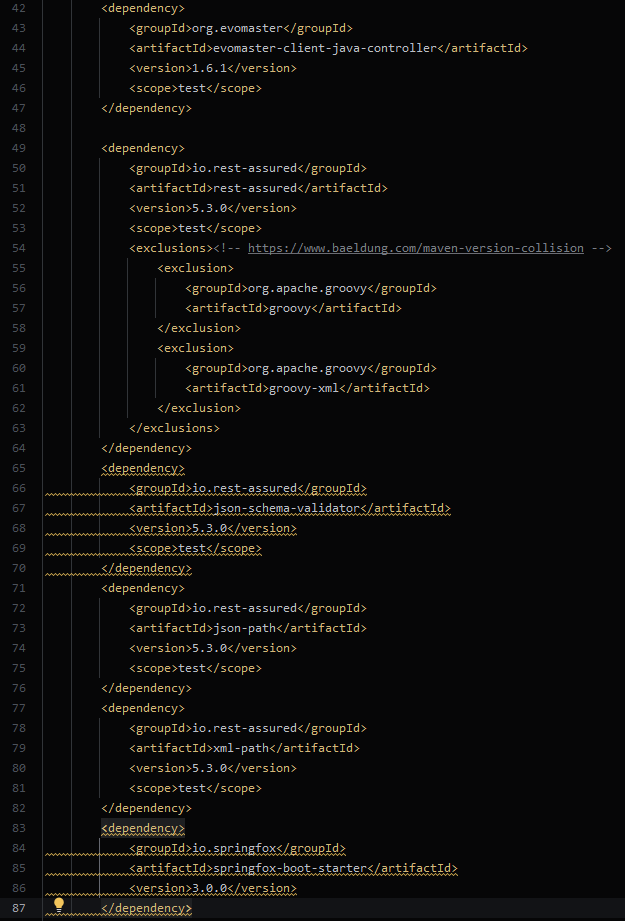
* getPackagePrefixesToCover() -> must return the package name where the entire REST API is located
* getInfoForAuthentication() -> in case you have guards for certain endpoints which require authorization, here you must specify the credentials
* getProblemInfo() -> must specify a url containing the REST API documentation as well as endpoints to skip. The easiest way to implement such a documentation is using Swagger
* getPreferredOutputFormat() -> must specify the required test output format (Junit4, Junit5, etc)
* getDbSpecifications() -> must specify the type of the SQL Database that the API is using

Other methods that need to be implemented can be found on the [EvoMaster GitHub](https://github.com/EMResearch/EvoMaster/blob/master/docs/write_driver.md). Here is our whole implementation of the driver:

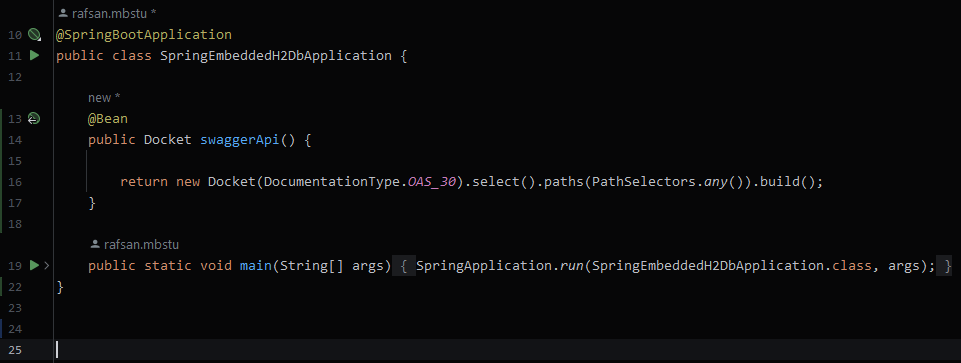




The required Maven dependencies can be found below. The exclusion of groovy from the first rest-assured was necessary because of a conflict of groovy versions between Spring Boot and rest-assured, while the last two dependencies were necessary for running the generated tests.



For setting up the REST API documentation, we used springfox and swagger, and in order to set up the required endpoint containing the documentation, the following code snippet was necessary.



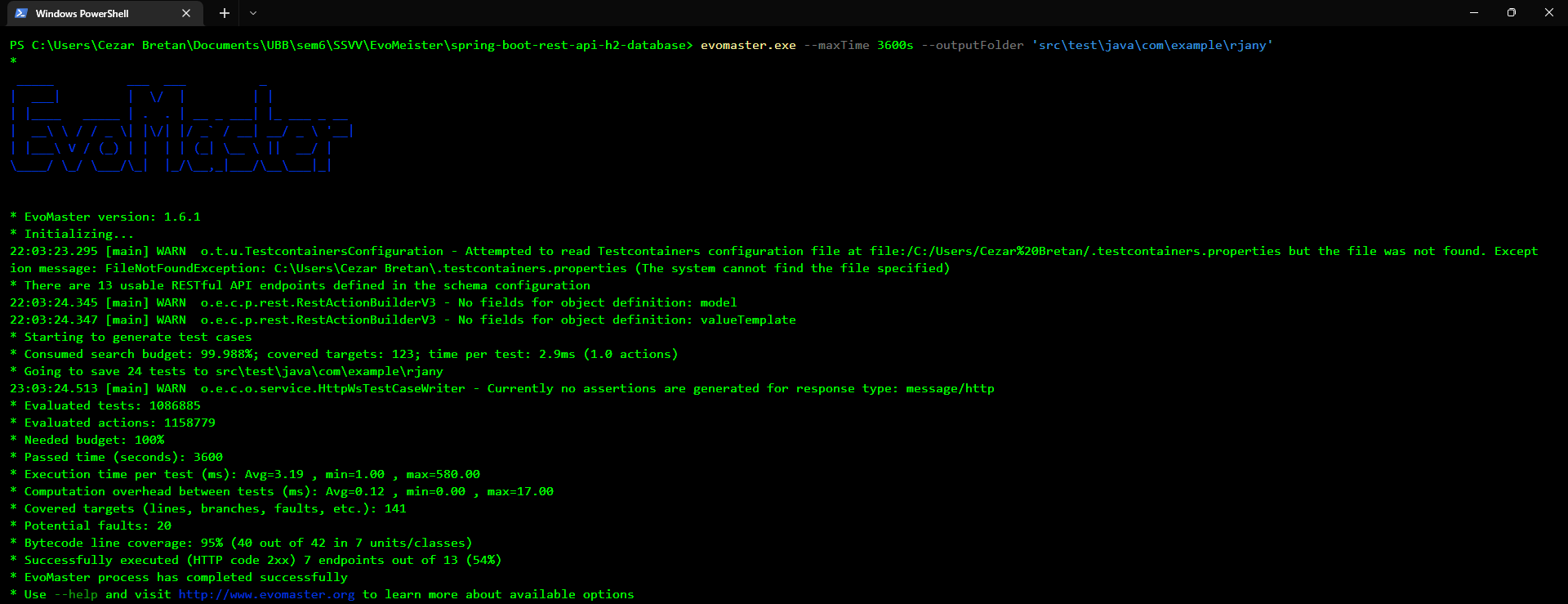
1. **Test project**

The SUT we are going to consider is a SpringBoot REST API with 5 endpoints for basic CRUD operations on Student entities persisted on an in-memory H2 database. It is compiled to Java 8, since EvoMaster specifies that the tool performs best for projects compiled to this version.

The Driver class was added to the project’s test module, and the Swagger API documentation that is required for the Core is provided in JSON form in the /v3/api-docs endpoint.

1. **Results**

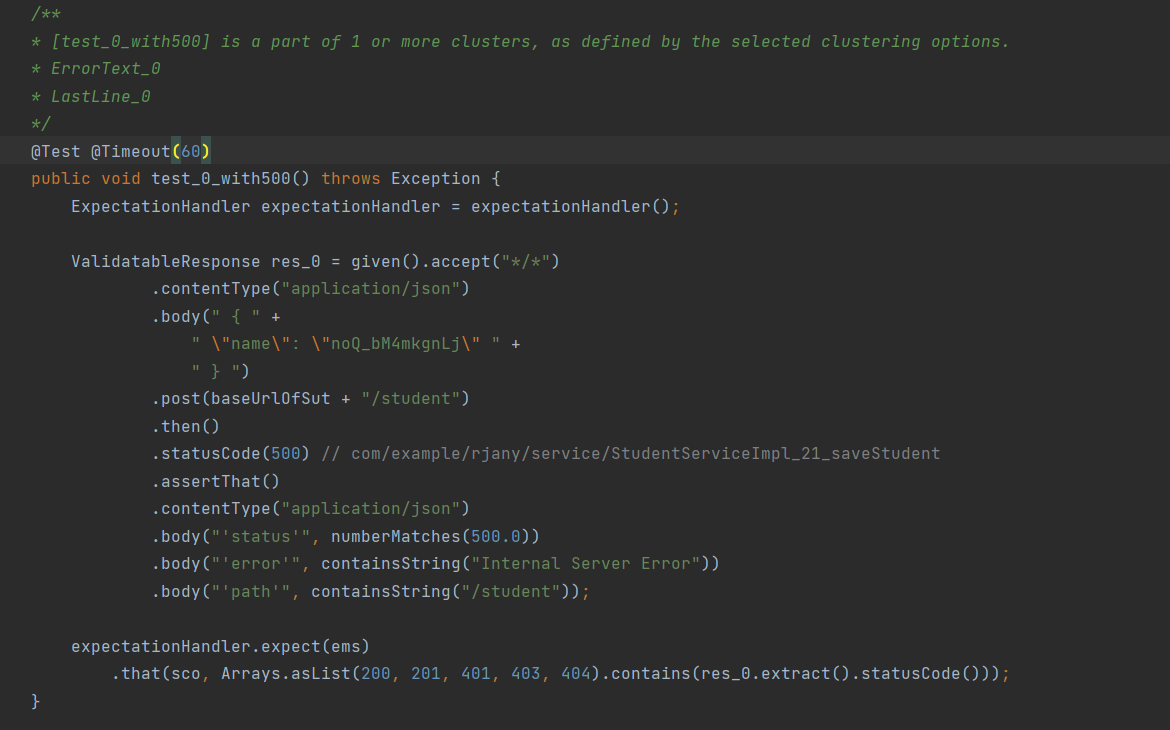
After running the tool for about one hour, the results can be seen below:



The tool generated four test classes with multiple methods each, called:

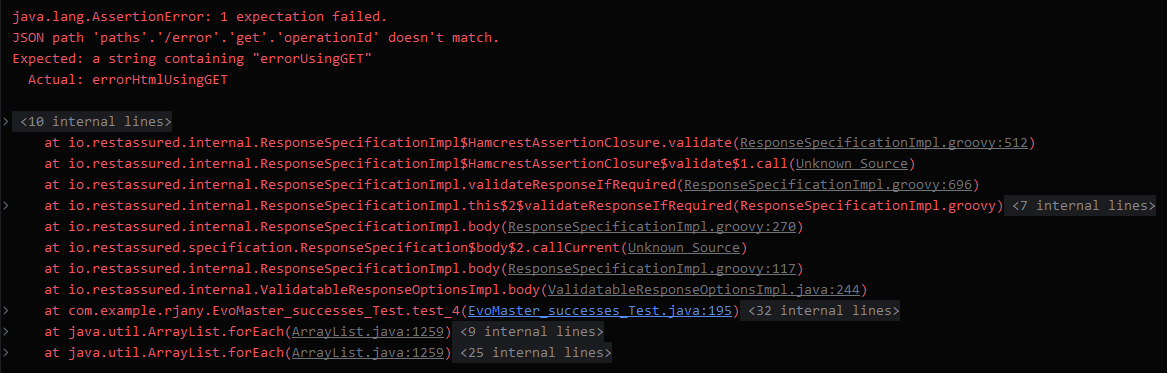
* EvoMaster\_successes\_Test
* EvoMaster\_fault\_representatives\_Test
* EvoMaster\_faults\_Test
* EvoMaster\_other\_Test

Below you can see an example of a generated test method from the fault\_representatives class.



Out of the generated tests, we have the following observations:

* Fault representatives : 4/4 tests passed
* Faults : 10/10 tests passed
* Others : 3/3 tests passed
* Successes: 11/11 passed, but sometimes test\_4() fails with the following stack trace:



As specified by the tool in the command line, it covered 95% of the Java bytecode lines and 7 of the 13 endpoints found.