Инж. Джеркс Салли
Студент аспирантской подготовки, Кафедра программной инженерии и информационных систем Факультет информатики, Университет Аль-Баас, Сирия, г. Хомс
Др. Эспер Алида
Доцент Аль-Баас Университета и Сирийского университета SPU
Университет Аль-Баас - Колледж инженерной информатики
Департамент программного обеспечения и информационной инженерии.
Др. Абу Салех Насер
Профессор Университета Аль-Баас - Колледж информационной инженерии - Департамент программной инженерии и информационных систем

ДИЗАЙН И ОСУЩЕСТВЛЯТЬ РАБОЧУЮ СРЕДУ ЧТОБЫ ТЕСТИРОВАТЬ ИНТЕРФЕЙСЫ ПРИЛОЖЕНИЙ REST – API

Аннотация: *Сделано в этом исследовании, Дизайн и осуществлять рабочую среду чтобы тестировать интерфейсы приложений Rest – API Как и положено изменяющейся рабочая среда Agile (Аджайл), В котором был предложен шаблон для создания тестовых примеров и для создания тестовых форматов файлов чтобы проверить ответы, полученные заказчиком, что они получены в соответствующем формате и с соответствующей функцией , Эта рабочая среда была реализована в системе управления контентом сайта.*

Результаты показали важность предложенных моделей для поддержки тестовых сред от этапа проектирования до этапа внедрения для эффективного и быстрого получения результатов.

Ключевые слова: *Автоматизированное тестирование, тестирование интерфейса приложения, сервисы, тестирование функциональный функция.*

Eng. Sally Jarkas
Ph.D, student Department of Software Engineering
Faculty of Informatics Engineering Al-Baath University Homs, Syria
Prof. Alida Esber
Department of Software Engineering and Information Systems
Faculty of Informatics Engineering - Al-Baath University Homs, Syria
Faculty of Computer and Informatics Engineering – Syrian Private University
Naser Abu Saleh
Department of Software Engineering
Faculty of Informatics Engineering Al-Baath University Homs, Syria

DESIGN AND IMPLEMENT AN AUTOMATED RESTFUL API TESTING FRAMEWORK

Annotation: *In our research, we focused on designing and implementing a Rest-API framework that follows the agile concepts; we suggested approaches to design test cases and test schema for validating the response data and executing the approaches in a real system. We applied this approach on a CMS. The approach helps to establish a good framework to start with to establish a Rest API framework from the design level to execution in high performance and efficiency.*

Keywords: *automated testing, API testing, services, functional testing, schema.*

1. Introduction

API stands for application programming interface; it is a contract that represents an agreement between parties. Also, we define the API as a set of definitions and protocols for building and integrating application software [1]. In our

research, we are going to focus on Web APIs, which are classified, depending upon the data exchange format, to SOAP [2], which uses XML exchange format and REST, which uses JSON Format [3]; which will be our research focus.

The JSON response contains data that are tested to ensure that it matches the desired schema; response code, contract data, DateTime, etc. The manual API testing method is time and effort consuming. On the other side, most existing automated testing techniques focus on validating the HTTP status code to test a collection, which is not enough but it requires analyzing the API response; so tools appear to check some response body schema validation. Emerging approaches do a full JSON Schema validation [4] to verify it matches the business-desired schema; others do a functional validation according to a specific approach [5].

Our proposed approach implements a suggested API design to check each restful API based on criteria like the ones mentioned in 4,5. We also will apply the functional API testing using the positive and negative test cases.

2. API Framework Design

2.1 Test Execution Script Design Approach

Our focus is to apply functional testing which will focus on the correctness of REST API responses, the correctness tested according to validators, like status code, and entity format. We suggest ignoring the header and entity format validators because all APIs responses should follow the JSON format according to the REST communication protocol, so these validators are implicitly checked. Other validators we are going to ignore are response size, and time because these validators are related to non-functional testing which is not in our scope.

Each Test Case would have many assertions that should follow general validation requirements, each of them should pass in all of these levels. We suggest implementing them as one transaction; ie if one of them fails then the whole test case would fail. In our research, we suggest assertion levels to be included in the API testing process (the assertion level stack) which are:

1- check the response could that should be in 2xx family.

2- check the response body contains error codes.

3- check the response is empty or not.

4- schema validation

These steps applied sequentially so failing in the first step will stop other steps from executing, so the test execution time would be smaller.

2.2 Test Schema Assertion Design

In section 2.1, we mentioned that the last step of test execution script design is to validate the schema. To create the expected schema the tester should write a JSON response format that matches the required business. We suggest a schema design approach; the goal of this schema is to maximize the test coverage. Our approach is to generate test cases from a GUI designed to accept the request input field. The generation algorithm depends on creating several test cases derived from the field required status, field type, name, and min and max values (taken from the business). The suggested maximum number of *generated test cases (m)* calculated using the following formula: $m = ((r * 2) + r + (r * 2))| + (n * 4)$

The equation comes from dividing the test case space into required *(r)* and none-required *(n)* fields. The first part refers that the test data of required fields should contain data for checking the existence of a field in the request (r*2); by taking into account that the positive cases should contain valid data (This would reduce the number of generated test cases). Then test data should contain a check on the field value type, which equals the required fields. After that, we add two test cases for each required field that is used to validate the under-min and over-max range values data. The second equation part refers to the number of test data generated for *none-required fields* multiplied by 4 which refers to a valid, none valid type value, under range, out of range values. We can simplify the equation to be:

$\boldsymbol{m = (5 * r) + (4 * n)}$; *r = #required fields and n = # none required fields*

3. Proposed API Framework implementation

We implement our API test approach as complementary for an e2e framework [6] we worked on previously. As a start point, we applied the suggested test script

stack suggested in 2.1 on several test suites from the mentioned system. We noticed that the first 3 steps are shared between all of the APIs, so we do a simple code refactor by creating a shared script that would be executed before each test case execution.

The next step is creating test data files that follow the suggested strategies mentioned in section 2.2 so we created a script with a simple GUI. This GUI is designed so the user enters each expected response field type, name, specifies if it is required or not, and finally the exact test case name; so he can link the generated test case with the generated test data file easily. After confirming, a test data file was created with the name "<testcasename>_data.json", so this way we are on our way to apply a data-driven model on the desired test case. Then at this level, we execute the test cases by applying API functional testing on Test Suites derived from the business logic. Finally, we run the tests using a suitable tool.

4. Results

To check the correctness of our approaches, we executed a set of test cases designed previously to compare it with test cases designed by our model. The following graph Fig1. shows a comparison between the 2 approaches on 4 cases; each case includes several test suites. We use execution time as a metric.

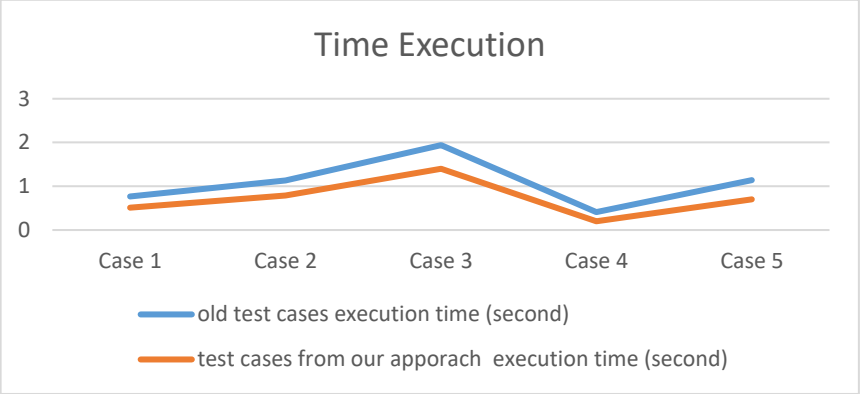

Fig 1. Comparison between random test cases and designed test cases execution time

The results show that the research suggested design approaches are always giving powerful results because of the constraints made in the first approach on executing the test cases; in case of incorrect response entity; the snippet will not

continue executing the following test steps. This equation can be generalized to any test data design in any testing level such as UI testing.

Our system is a good option to design Rest-API test cases and scripts; which reduces the development, execution time. On the other side, our schema design approach is a way to achieve high coverage with a minimum number of test cases; which is calculated from the attached mentioned equation.

References:

1. Ian Sommerville (2015), Service-oriented Engineering. In Software Engineering (10th Edition, 521-531). London, United Kingdom: Pearson.
2. M.Gudgin, M.Hadley, N.Mendelsohn, J.Morea, H.F.Nielsen, and Y. Lafon, "Soap version 1.2" W3C recommendation, vol. 24,2003.
3. S. Azzam, M. Al-Kabi, and I. Alsamdi, "Web services testing challenges and approaches", in Proceedings of the 1st Taibah University International Conference on Computing and Information Technology (ICCIT 2012), 2012, pp. 291-296.
4. Isha, A. Sharma, M.Revathi, "Automated API Testing" In proceedings of the International Conference on Inventive Computation Technologies (ICICT-2018)
5. H. Wenhui, Haung Yu, L. Xueyang, Xu Chen, "Study on Rest API Test Model Supporting Web Service Integration" in IEEE 3rd International Conference on Big Data and Security on Cloud 2017.
6. Sunil L. Bangare, Seema Borse, Pallavi S. "automated API testing approach." In International Journal of Engineering Science and Technology (IJEST). Vol. 4 No.02 February 2012.
7. S. Jarkas, E. Esper, N. Abu Saleh. "Applying Automated Testing of Event-Driven Software Systems" In Al-Baath university journal vol. 42 No. 21.