Jurnal Teknik Informatika (JUTIF)
Vol. 5, No. 2, April 2024, pp. 551-560

DOI: https://doi.org/10.52436/1.jutif.2024.5.2.1673
p-ISSN: 2723-3863
e-ISSN: 2723-3871

LOW CODE INTEGRATION TESTING IN OUTSYSTEMS PERSONAL ENVIRONMENT

I Dewa Ayu Indira Wulandari Chrisna[1], Dana Sulistiyo Kusumo[*2], Rosa Reska Riskiana[3]

[1,2,3]Informatics, School of Computing, Telkom University, Indonesia
Email: [1]indira.chrisna@gmail.com, [2]danakusumo@telkomuniversity.ac.id, [3]rosareskaa@telkomuniversity.ac.id

(Article received: January 04, 2024; Revision: January 11, 2024; published: April 15, 2024)

Abstract

As implied by its name, low code platforms enable software development with minimal or no coding involved. Consequently, ensuring the correctness of the software becomes crucial as developers are unable to directly scrutinize the logic. Furthermore, discussions about the various testing approaches applicable to such applications are relatively scarce. This study aims to conduct integration testing through both white box and black box methods, as well as exploring the types of testing that can be carried out on low code based applications. This research involves several stages, including creating a basic e-shop application and API using OutSystems, test preparation, and test execution. API testing utilizes OutSystems' BDDFramework and Postman automation testing tools, while web page integration is carried out using Katalon Studio. The test results indicate only one of the total 23 test cases was considered failed because the result did not match the expected result. Apart from that, of the four existing levels of testing, component testing can also be carried out on the OutSystems application. However, only with the black box testing method because testing is carried out without accessing the program source code. The comparative execution of API testing (white box) using two distinct testing tools reveals the superior effectiveness of Postman over BDDFramework, offering more comprehensive test outcomes and enhanced test case coverage. In the realm of UI integration testing, Katalon Studio emerges as a fitting tool, benefiting from its record and replay feature that facilitates the definition of test steps.

Keywords: *API, low-code, OutSystems, testing.*

1. INTRODUCTION

The Low Code Development Platform (LCDP) is a cloud-based software development platform. It enables users to create fully functional software through interaction with a dynamic graphical user interface, visual diagrams, and declarative languages [1]. LCDP has emerged as a promising solution for companies aiming to enable professionals without coding experience to construct applications [2]. Facilitated by pre-built modules and an intuitive interface that typically integrates drag-and-drop functionality to configure process models and the app's framework, it streamlines the app development process, making it more efficient and easily scalable [3]. Referring to The state of Application Development [4], 63% of organizations say that they will develop the majority of their applications using low-code development platform by the end of 2024.

In 2020, Faezeh Khorram, et al. Conduct research to determine the challenges and opportunities that exist in testing low-code based applications. The research is aimed at presenting the progress of low-code testing from a business perspective [5]. The results indicate that challenges in low-code testing encompass three issues: the role of the Citizen Developer in testing, the need for high-level test automation, and cloud testing [5]. In Low-Code-based application development, test automation is particularly crucial, especially for API testing. This is due to the reliance of Low-Code applications on APIs for integration with other services. Regular testing of these integrations is essential to prevent application failures [5].

The OutSystems development platform boasts a notable advantage with its self-correction feature, capable of automatically rectifying certain potential errors and promptly providing developers with relevant information about the necessary modifications [6]. This ensures that nothing is broken during the implementation stage.

However, despite the support offered by the platform, there is no assurance that errors will be entirely absent, and the evolving software will be entirely free of bugs. Consequently, various testing activities must be conducted at different stages throughout the life cycle of an OutSystems application [6].

Software testing is a verification and validation process to ensure that the system under development has fulfilled the business needs and technical requirements that underpin its design, ensuring that the developed system operates in accordance with the established expectations [7]. Failure of assuring software quality can leads to a serious bug that might cost more than one year's salary of a programmer [8].

Therefore, testing serves as a proactive safeguard, akin to an insurance mechanism. In software testing, there are three techniques, each with a more specific testing strategy, aimed at enhancing the effectiveness of the testing process. These techniques are white-box testing, black-box testing, and gray-box testing [9].

In a research entitled "Characteristics and Challenges of Low-Code Development: Practitioners' Perspective" (2021) by Luo et al. The research results show that one of the limitations of the Low Code development platform is the lack of access to the program's Source Code. The results of this research are directly correlate with a limitation in the OutSystems Personal Environment. Specifically, developers are unable to execute the detach process, preventing them from obtaining the Source Code of their applications. The ability to perform this Detach process is restricted to Enterprise Environment owners who decide to terminate their contract with OutSystems [10]. Consequently, White Box testing, which necessitates access to the program's Source Code, is not feasible under these circumstances.

On the other hand, OutSystems has a Framework component called BDDFramework which allows Citizen Developers to conduct testing from the server side and enables BDD / TDD Testing for applications developed with this Low-Code based development platform [11]. The focus of this component is to test the logic in the application module by practicing critical actions that can support Test Case design [11]. Component Testing in OutSystems with BDDFramework includes testing open actions and services that form the logic of the application being developed [12]. In this way, it can be concluded that Component Testing which can be carried out in the Low-Code OutSystems application is still not possible using the White Box method.

This research aims to explore the types of tests applicable to Low-Code based applications using the OutSystems platform, given its prominence as a low-code development platform widely adopted by major companies such as AXA, Honda Motor Co., Ltd., Intel Corporation, and numerous others across 22 industries [13].

Apart from that, this research will focus on integration testing (API Testing). The rationale for conducting only integration testing in this study is grounded in the research by Khorram et al., which specifically emphasizes the significance of integration testing, especially in the context of low-code-based applications [5]. Beside, integration testing is among the testing types applicable in low-code development. Integration testing will be conducted using 2 methods: White Box Testing (API Testing) and Black Box Testing (integration between website pages). API testing will be carried out using two testing tools. One is OutSystems' BDDFramework, and the other one is Postman which is a platform for building, using, and also testing APIs [14]. While UI Testing will be carried out using Katalon Studio which is an automation testing tool for conducting UI testing on Web and/or mobile applications [15]. It is anticipated that the results of this research will provide guidance on the types of testing applicable to Low-Code based applications developed with the OutSystem platform, particularly in the Personal Environment.

2. RESEARCH METHODOLOGY

This research involves three crucial phases, namely the preparation of the Application Under Test (AUT), testing preparation encompassing Test Planning and Test Case Creation, and the ultimate stage, Test Execution. The visual representation of the entire sequence of research stages is depicted in Figure 1.

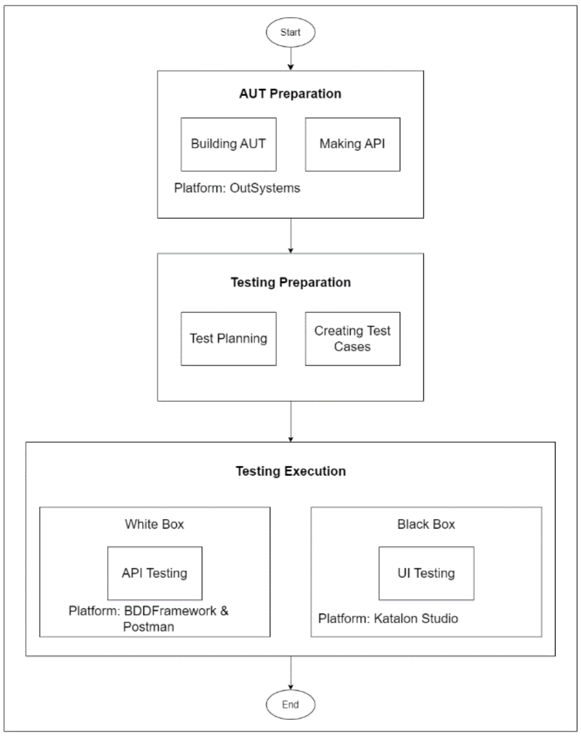


Figure 1. Research Flowchart

2.1. AUT Preparation

The first phase that will be taken in this research is to develop the application and API that will be tested. Application and API development is conducted utilizing OutSystems, a platform for low code development. The environment used is OutSystems' Personal Environment.

The Application Under Test (AUT) being developed is a basic e-shop application that has Create, Read, Update, and Delete (CRUD) features and can send data to an API called productsAPI. The Add To Cart feature was also added to this application to support integration testing with Black Box Testing.

The ProductsAPI has been developed with multiple methods, including POST, PUT, DELETE, and GET (such as GetAllProducts, GetProductById, GetProductByCategory). This API operates as a

Public API, meaning it does not necessitate an authentication process for data access. Furthermore, the required data requests are expected to be in JSON format.

2.2. Testing Preparation

In the second phase, Testing Preparation, two key activities will be undertaken: Test Planning and Test Case Generation. During the Test Planning stage, an evaluation of the application will be conducted to identify the types of tests applicable to Low-Code-based applications. Then, Test Cases for White Box and Black Box integration testing will be designed at the Test Case Making phase.

This research will focus on Integration Testing with automation, especially API Testing. Because based on research conducted by Khorram et al. In 2020, test automation is very important in application development with Low-Code platforms. Especially API Testing automation is important in Low Code Development Platform (LCDP) because Low-Code based applications use a lot of integration to other services using APIs [5]. According to [16], in every testing levels, there are several objects that can be tested. In AUT preparation and Test Preparation step, the application will be evaluated to know which test objects can be tested in OutSystems applications. A table of those testing objects per testing levels and OutSystems application's testability to those objects will be presented in subsection 3.2. which explain the results of test planning step.

The methods that will be tested in API testing are all methods that have been created at the AUT preparation stage, namely POST, PUT, DELETE, and 3 types of GET (Get All, By Id, and By CategoryId). This testing will be carried out using 2 different automation testing tools, namely the BDDFramework component from OutSystems and Postman. Integration Testing has a focus on testing Request/Response accuracy, service availability, and performance [17]. Therefore, in this research, the test parameters that will be analyzed in API Testing are the resulting Status Code and Response Time (ms).

In Black Box testing, the focus will be on assessing the integration between pages, specifically the Home Page and the Cart Page. In essence, this test aims to determine whether the Cart Page accurately displays identical information as observed before and/or after modifying the product information on the Home Page. Katalon Studio, an automation test tool designed for UI testing, will be utilized to conduct this testing.

2.3. Testing Execution

At this stage, all test cases that have been created in the previous stage will be executed. Testing will be carried out using 2 methods: black box testing and white box Testing. In the Black Box testing method, the assessment will involve testing the integration between pages, specifically focusing on the Home Page and Cart Page. It is expected that both pages will display the same data before and after data changes are made via the Home Page. In Black Box testing, the Katalon Studio automation test tool will be utilized. Meanwhile, API Testing (White Box) will be conducted using two testing tools: BDDFramework which is a component in OutSystems Forge and Postman which is an API testing tool to compare experiences in conducting Low Code API Testing.

OutSystems has a component called BDDFramework that can be installed in Forge in OutSystems Service Studio. As usual, OutSystems provides a web template that has implemented the BDDFramework, so Citizen Developers only need to use the template, then write scenarios and Gherkin scripts, namely Given, When, and Then. Apart from that, developers can also setup and teardown the actions that have been carried out during testing to reset the data so that it is not mixed with the testing results data. Apart from that, testing applications using the BDDFramework is recommended to create a special testing application that is separate from the existing application. under development. This can be considered as a form of Version Management.

As is its characteristic as a Low-Code platform, namely software development using the Drag-and-Drop method, creating logic for the Given, When, and Then syntax is also done by pulling the API Method node, Client Action, and other activities such as Assign Value, Exception Handling, etc. to the center of the screen. Designing application logic in general is also greatly facilitated by AI which can provide fairly accurate recommendations when developers assign values.

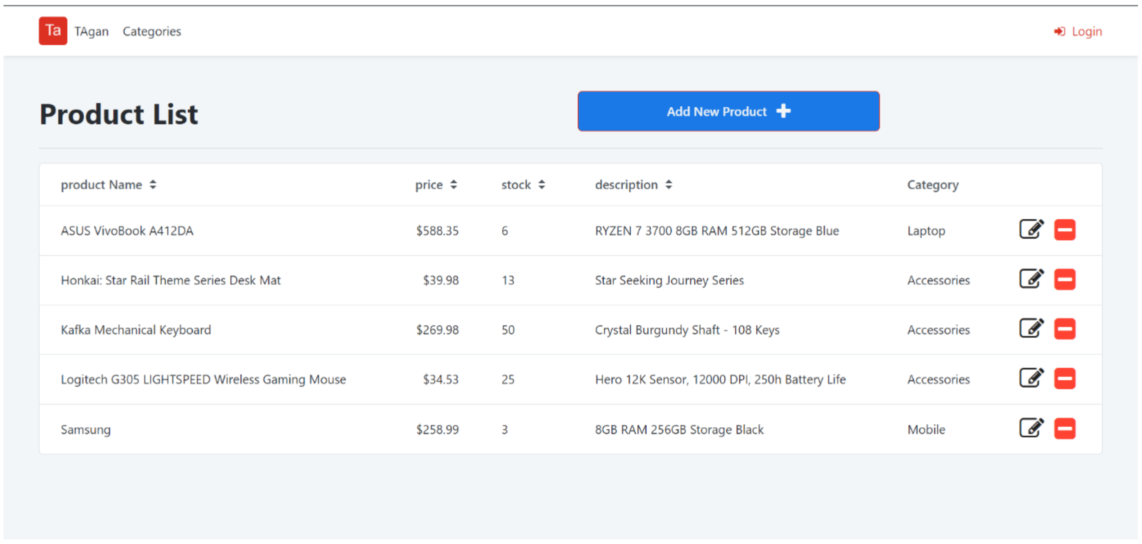
In the context of API testing, the first thing that must be done is to consume the REST API (or SOAP depending on the testing being carried out). After carrying out Consuming and all Methods have been saved in the Testing application, the next step is to start designing the logic that must be carried out behind each Given, When, and Then scenario.

3. RESULT

In this section will be delivered the result of the research from each of the research methodology steps. The third step of this research, namely "Test Execution" will be divided by three sections: API Testing in OutSystems, API Testing in Postman, and UI Testing in Katalon Studio.

3.1. AUT Preparation

At the AUT preparation stage, a simple e-shop application is created with CRUD features, and in addition, there is a function to add products to the cart. Meanwhile, at the API development stage, a REST API for product data has been created and can be accessed publicly. The request methods that have been created are POST, PUT, DELETE, and 3 types of GET (get all products, get product by id, and get product by category id). The homepage UI of the simple e-shop application that has been created can be seen in Figure 2, while the API documentation page can be seen in Figure 3.

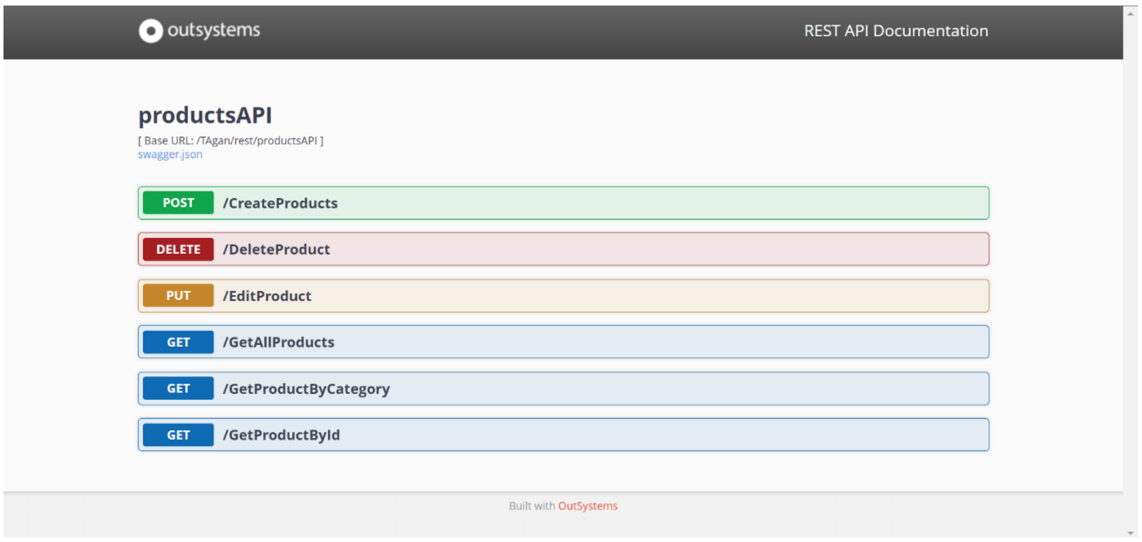

Ta TAgan Categories Login

Product List

Add New Product

product Name	price	stock	description	Category
ASUS VivoBook A412DA	$588.35	6	RYZEN 7 3700 8GB RAM 512GB Storage Blue	Laptop
Honkai: Star Rail Theme Series Desk Mat	$39.98	13	Star Seeking Journey Series	Accessories
Kafka Mechanical Keyboard	$269.98	50	Crystal Burgundy Shaft - 108 Keys	Accessories
Logitech G305 LIGHTSPEED Wireless Gaming Mouse	$34.53	25	Hero 12K Sensor, 12000 DPI, 250h Battery Life	Accessories
Samsung	$258.99	3	8GB RAM 256GB Storage Black	Mobile

Figure 2. AUT Home Page


outsystems REST API Documentation

productsAPI

[Base URL: /TAgan/rest/productsAPI]
swagger.json

POST /CreateProducts
DELETE /DeleteProduct
PUT /EditProduct
GET /GetAllProducts
GET /GetProductByCategory
GET /GetProductById

Built with OutSystems

Figure 3. API Documentation Page

3.2. Test Preparation

In this study, the feasibility of conducting testing has been examined based on the four established testing levels outlined in the Certified Tester Foundation Level (CTFL) Syllabus, which include Component Testing, Integration Testing, System Testing, and Acceptance Testing [16].

After evaluating the application through development and reviewing relevant documentation, it has been identified that certain test objects for each test level cannot be executed in OutSystems' applications due to limitations in accessing the source code. Lists of test objects that can be tested in OutSystems' application is presented in Table 1.

In test preparation phase, a total of 14 API Testing Test Cases has been created based on the 6 methods mentioned in the previous subchapter. Test Cases are built based on 2 main scenarios: valid and invalid input parameters. API test cases is described in subsection 3.5.

The aspect that will be examined in UI testing is integration between application pages (Home Page & Cart Page). Test Cases are built by paying attention to the business rules for adding products to Cart, namely:

1. The product will be displayed on the Cart Page if the product's addedToCart attribute is > 0.
2. Products can only be added to Cart as many times as product.Stock.

That way, the range of the number of times a product is added to the Cart is [1 … product.Stock], so that by using the Boundary Value Analysis (BVA) method, there are at least 6 Test Cases that can be built. The sample test case that has been built is described in subsection 3.6.

Table 1. Possible testing on Low-Code Applications

Testing Levels	**Component Testing**				**Integration Testing**						**System Testing**					**Acceptance Testing**					
Test Objects	Components, units, or modules	Code and Data Structure	Classes	Database Modules	Subsystems	Databases	Infrastructure	Interfaces	APIs	Microservices	Applications	Hardware/Software Systems	Operating Systems	System Under Test (SUT)	System configuration and configuration data	Business processes	Recovery Systems	Operational and Maintenance	Forms	Reports	Existing and Converted production data
Testability	×	×	×	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

3.3. API Testing in OutSystems

In one application, several Test Suites can be created by adding new Screens. The screen can then be filled with several Web Blocks with a BDD template (Figure 4).

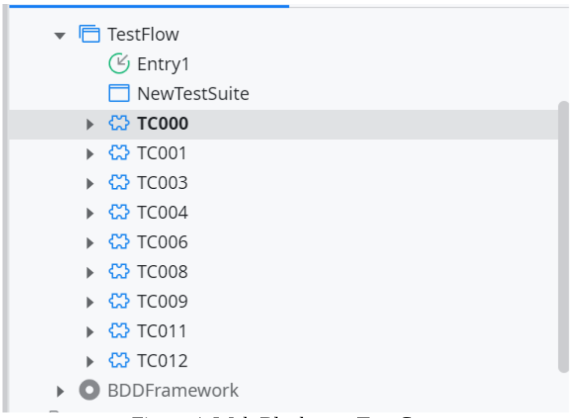
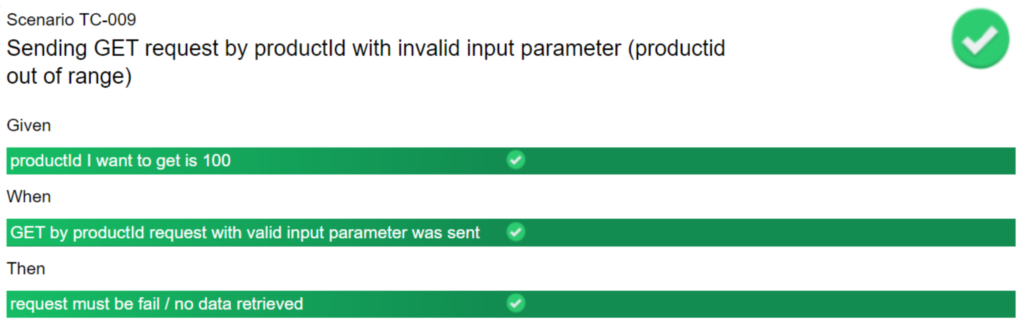

TestFlow
Entry1
NewTestSuite
TC000
TC001
TC003
TC004
TC006
TC008
TC009
TC011
TC012
BDDFramework

Figure 4. Web Blocks per Test Case

The Test Case that will be used for API testing with BDDFramework is the same as the Test Case that was designed based on the test scenario as described in Table 2. However, there are several disadvantages when carrying out API Testing with the BDDFramework. OutSystems has been designed to have a system that can perform syntax checking automatically to avoid errors due to syntax or data type errors, and does not allow empty parameter input. However, this advantage can become a disadvantage when carrying out testing which requires all possibilities including negative scenarios such as errors in writing data types in input parameters and cases of sending requests without parameters. These two types of negative scenarios cannot be carried out because OutSystems will give an error warning and the Testing application cannot be published (run). Therefore, as many as 4 out of 14 Test Cases cannot be carried out using the BDDFramework. Test cases that can be run with the BDDFramework are described in Table 2.

Table 2. Testing Result Using BDDFramework

TEST CASE ID	Scenario	Result
TC-000	Get All Product	Test Passed (all products retrieved successfully)
TC-001	Create New Product (valid input)	Test Passed (new product added successfully)
TC-003	Edit Product (valid input)	Test Passed (product data edited successfully)
TC-004	Edit product (invalid productId)	Exception (product with given Id is not found)
TC-006	Delete product (valid productId)	Test Passed (product deleted successfully)
TC-007	Delete product (invalid input Id)	Test Passed (no product to be deleted)
TC-008	Get product by Id (valid input)	Test Passed (product retrieved successfully)
TC-009	Get product by Id (invalid Id)	Test Passed (no data in response)
TC-011	Get product by categoryId (valid Id)	Test Passed (product retrieved successfully)
TC-012	Get product by categoryId (invalid Id Out of range)	Test Passed (no data retrieved)


Scenario TC-009

Sending GET request by productId with invalid input parameter (productid out of range)

Given

productId I want to get is 100

When

GET by productId request with valid input parameter was sent

Then

request must be fail / no data retrieved

Figure 5. Web Block TC-009 when test application published

In addition, when the Test Suite is run, the Response Time of each Request cannot be known and the Status Code requires additional instructions to be obtained.

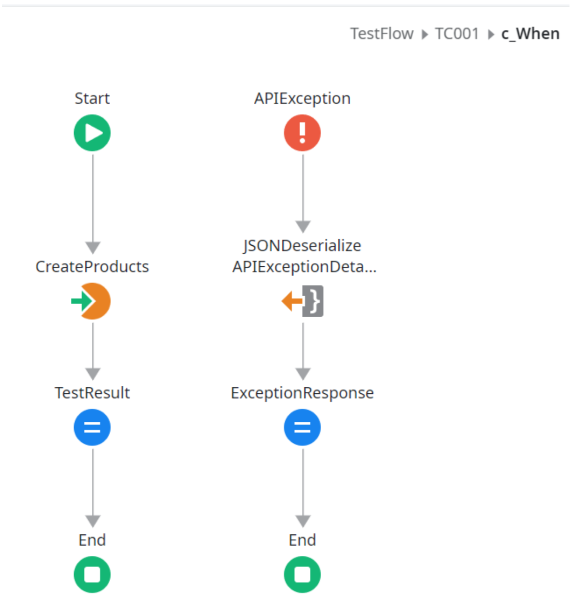
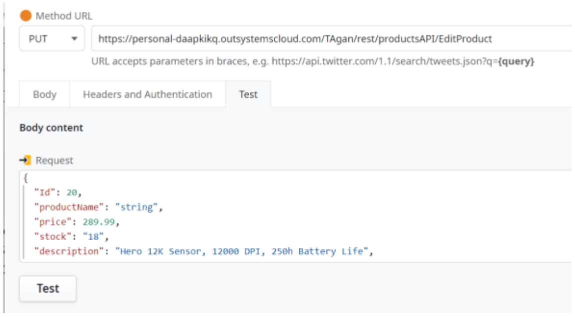


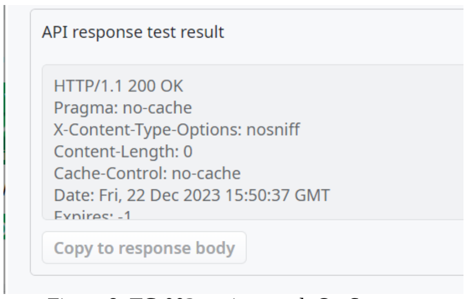
Figure 6. When logic creation

Apart from using the BDDFramework, OutSystems also allows users to perform API testing per method by double clicking on the method that has been saved in the Logic section. A pop up will appear and in the last tab, namely in the "Test" section. Testing can be done by having a Method in the Dropdown at the top left, then entering the Request Body or input parameters as needed (see Figure 7 & Figure 9). Then, the results can be seen in the same tab after clicking the Test button (see Figure 8 & Figure 10). 4 Test Cases that fail to run with the BDDFramework will be tested using this feature. The result was that 3 of the 4 Test Cases that could not be carried out previously, namely TC-005, TC-010, and TC-013, received test results that matched the expected results, namely 400 Bad Request. Meanwhile, TC-002 (Edit product with stock written as String) received a response of 200 Ok which was not in line with expectations.


Method URL
PUT
https://personal-daapkikq.outsystemscloud.com/TAgan/rest/productsAPI/EditProduct
URL accepts parameters in braces, e.g. https://api.twitter.com/1.1/search/tweets.json?q={query}
Body | Headers and Authentication | Test
Body content
Request

```
{
  "Id": 20,
  "productName": "string",
  "price": 289.99,
  "stock": "18",
  "description": "Hero 12K Sensor, 12000 DPI, 250h Battery Life",
```

Figure 7. Setup TC-002 in OutSystems


API response test result

```
HTTP/1.1 200 OK
Pragma: no-cache
X-Content-Type-Options: nosniff
Content-Length: 0
Cache-Control: no-cache
Date: Fri, 22 Dec 2023 15:50:37 GMT
Expires: -1
```

Figure 8. TC-002 testing result OutSystems

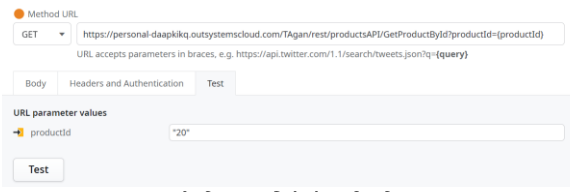
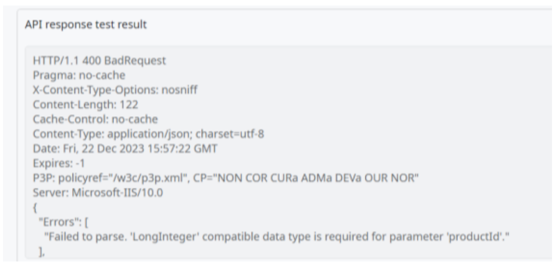

Method URL
GET
https://personal-daapkikq.outsystemscloud.com/TAgan/rest/productsAPI/GetProductById?productId={productId}
URL accepts parameters in braces, e.g. https://api.twitter.com/1.1/search/tweets.json?q={query}
Body | Headers and Authentication | Test
URL parameter values
productId: "20"

Figure 9. Setup TC-010 in OutSystems


API response test result

```
HTTP/1.1 400 BadRequest
Pragma: no-cache
X-Content-Type-Options: nosniff
Content-Length: 122
Cache-Control: no-cache
Content-Type: application/json; charset=utf-8
Date: Fri, 22 Dec 2023 15:57:22 GMT
Expires: -1
P3P: policyref="/w3c/p3p.xml", CP="NON COR CURa ADMa DEVa OUR NOR"
Server: Microsoft-IIS/10.0
{
 "Errors": [
  "Failed to parse. 'LongInteger' compatible data type is required for parameter 'productId'."
 ],
```

Figure 10. TC-010 testing result in OutSystems

3.4. API Testing in Postman

API Testing has been carried out in the Postman automation test tool by executing 14 Test Cases with a combination of Positive and Negative Cases. Test Cases are separated into 4 different folders according to the Method to be tested. Each Test Case in the same folder is then executed 1x.

5 of the 13 Test Cases executed had test results that were different from the expected results. The five Test Cases that had final results that were different from the expected results were then re-executed to validate the results obtained.

Table 3. Test Scenario and API Testing result in Postman

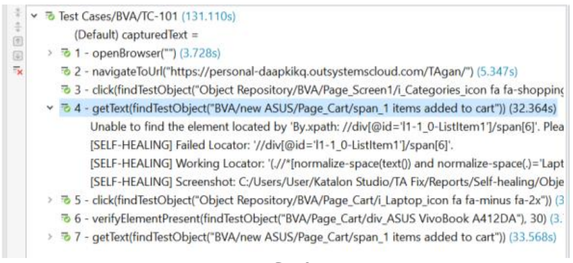
TC ID	Test Scenario	Expected Result (Status Code)	Actual Result (Status Code)
TC-000	Get All Product	200 Ok	200 Ok
TC-001	Create New Product (valid input)	200 Ok	200 Ok
TC-002	Create New Product (Invalid Input)	400 Bad Request	500 Internal Server Error
TC-003	Edit Product (valid input)	200 Ok	200 Ok
TC-004	Edit product (invalid productId)	404 Not found	500 Internal Server Error
TC-005	Edit product (invalid data type)	400 Bad Request	400 Bad Request
TC-006	Delete product (valid productId)	200 Ok	200 Ok

TC-007	Delete product (invalid input Id)	404 Not Found	200 Ok
TC-008	Get product by Id (valid input)	200 Ok	200 Ok
TC-009	Get product by Id (invalid Id)	404 Not Found	200 Ok
TC-010	Get product by Id (invalid Id data type)	400 Bad Request	400 Bad Request
TC-011	Get product by categoryId (valid Id)	200 Ok	200 Ok
TC-012	Get product by categoryId (invalid Id Out of range)	404 Not Found	200 Ok
TC-013	Get product by categoryId (invalid Id data type)	400 Bad Request	400 Bad Request

3.5. UI Testing in Katalon Studio

Black Box Testing is carried out by implementing two testing techniques, namely Boundary Value Analysis (BVA) and based on application requirements. The BVA testing technique works by testing data that is close to the boundaries of the data to be tested. These data include minimum, maximum values, just outside the data range, and just inside the data range [18]. 6 BVA Test Cases have been executed and the Actual Result obtained is in accordance with the Expected Result even though some test results were declared Failed by Katalon Studio due to a mismatch in the object to be verified. The following is a report on the results of the Test Case execution that was carried out at Katalon Studio using the Record-and-Replay testing technique.

The scenario of TC-101 is to test the application response when a product called ASUS VivoBook is added to Cart 0x. Meanwhile, on the TC-102, the application response was tested when the same product (ASUS VivoBook) was added to the Cart once.



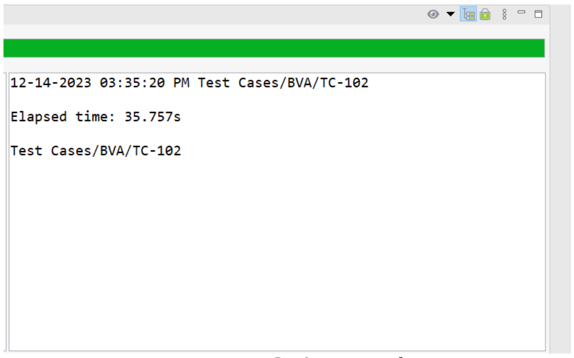
```
Test Cases/BVA/TC-101 (131.110s)
    (Default) capturedText =
  1 - openBrowser("") (3.728s)
  2 - navigateToUrl("https://personal-daapkikq.outsystemscloud.com/TAgan/") (5.347s)
  3 - click(findTestObject("Object Repository/BVA/Page_Screen1/i_Categories_icon fa fa-shopping
  4 - getText(findTestObject("BVA/new ASUS/Page_Cart/span_1 items added to cart")) (32.364s)
      Unable to find the element located by 'By.xpath: //div[@id='l1-1_0-ListItem1']/span[6]'. Plea
      [SELF-HEALING] Failed Locator: '//div[@id='l1-1_0-ListItem1']/span[6]'.
      [SELF-HEALING] Working Locator: '(.//*[normalize-space(text()) and normalize-space(.)='Lapt
      [SELF-HEALING] Screenshot: C:/Users/User/Katalon Studio/TA Fix/Reports/Self-healing/Obje
  5 - click(findTestObject("Object Repository/BVA/Page_Cart/i_Laptop_icon fa fa-minus fa-2x")) (3
  6 - verifyElementPresent(findTestObject("BVA/Page_Cart/div_ASUS VivoBook A412DA"), 30) (3.
  7 - getText(findTestObject("BVA/new ASUS/Page_Cart/span_1 items added to cart")) (33.568s)
```

Figure 11. TC-101 test steps

Table 4 Response Time API Testing

Method	POST		PUT			DELETE		GetAll	GET					
TC ID	TC-001	TC-002	TC-003	TC-004	TC-005	TC-006	TC-007	TC-000	TC-008	TC-009	TC-010	TC-011	TC-012	TC-013
Response Time (s)	2021	1886	1792	1810	2004	1909	1992	2033	1767	1757	2035	1792	1998	1296
Avg. Response Time	**1953.5**		**1868.66**			**1950.5**		**2033**	**1774.16**					

In the UI Testing execution process, TC-102 is carried out first to add the product to the Cart once. After that, TC-101 is executed to test the application when the product is added to the Cart 0x. As per Expected Result, once TC-101 is executed, the product disappears from the Cart Page.



```
12-14-2023 03:35:20 PM Test Cases/BVA/TC-102

Elapsed time: 35.757s

Test Cases/BVA/TC-102
```

Figure 12. TC-102 test result

Test Case TC-103 was declared failed by Katalon Studio because in the last step (Verify Element Present), the objects used were still the same as the objects used in TC-102 and TC-101. After executing TC-101, the ASUS VivoBook product has disappeared from the Cart Page. Therefore, after being added to the Cart on the TC-103, the XPath belonging to the ASUS VivoBook product object changed and its existence could not be verified. However, when viewed from the applications tested, ASUS VivoBook products have been successfully added to Cart 2x in accordance with the TC-103 testing objectives. So, TC-103 is considered successful and meets the expected results.

TC-104 has a scenario to test the application response when a product called ASUS VivoBook is added to the Cart as much as the stock of that product is reduced by one. In this case, the stock amount of the ASUS VivoBook is 6. So, the Test Data used is 5 (entering the product into the Cart 5x). Continuing the results of the previous test which had added the same product 2x, in this Test Case the product was added 3x more. The result of this test is the Actual Result in accordance with the Expected Result.

TC-107 and TC-108 are designed based on the requirements of the application being developed, namely when the user changes the data of a product in the Cart via the Home Page, the product data displayed on the Cart Page must also change (TC-107). Meanwhile, on TC-108, the user will delete a product that has been added to the Cart Page via the Home Page. Expected Result from TC-108 is that the product is missing from the Home Page or Cart Page.

Table 5. Test Case & UI Testing Result

TC ID	Scenario	Expected Result	Actual Result
TC-101	addedToCart count = 0	Product not present in Cart page	Product not present in Cart page
TC-102	Adding product to cart (addedToCart count = 1)	Product shown in Cart page	Product shown in Cart page
TC-103	Adding product to cart (addedToCart count = 2)	Product shown in Cart page	Product shown in Cart Page
TC-104	Adding product to cart (addedToCart count = 5)	Product shown in Cart page	Product shown in Cart Page
TC-105	Adding product to cart (addedToCart count = 6)	Product appears in Cart page	Product shown in Cart Page
TC-106	Adding product to cart (addedToCart count = 7)	addedToCart count not added & a notification appears	addedToCart count not added & a notification appears
TC-107	An in-Cart product is edited in Home Page and the same edited data must be shown in Cart Page	The data shown in Cart page is the same with Home Page	The data shown in Cart page is the same with Home Page
TC-108	An in-Cart product is deleted in Home Page and that product must dissapear from Cart Page	Product disappear from Cart page	Product disappear from Cart Page

4. DISCUSSION

As indicated in [5] and [6], BDDFramework is a viable tool for conducting tests on OutSystems applications, including API Testing. In the research conducted by [6], the author also performed API testing using BDDFramework. However, the primary objective was to illustrate the impacts of certain best practices in OutSystems development as facilitators in the test automation process. In contrast, this study obtained comparative results when conducting API testing using BDD Framework and Postman. Furthermore, [6] did not elaborate on the strengths and weaknesses of BDDFramework as described in this current research.

From the findings of this research, it is evident that API testing with Postman yields more comprehensive results. Postman provides detailed test results, including the Status Code, Response Time, and Size of the Response.

Setting up tests using BDDFramework is relatively uncomplicated, involving the drag-and-drop of methods or other activities to the central area of the Service Studio screen (refer to Figure 5). Nevertheless, this procedure is time-consuming when compared to the preparation of test cases using Postman. Importantly, it is acknowledged that this framework may not encompass all potential scenarios, such as errors in formulating the Request body. An example includes the inclusion of a stock attribute with a string data type instead of the requisite integer data type, and situations where input parameters are inadvertently left empty. This investigation provides insights into the evaluative aspects of testing objects at each testing level based on the categorizations outlined in [15].

Table 6. Postman & BDDFramework Comparison

Postman	BDDFramework
All test case can be executed	Some negative test case can’t be carried out
Visible testing results are status code, response time, and data size	Visible testing results are just wether the test case is passed or not
Simple, to the point testing. But needs more attention to request body format.	To build the test case, tester only needs to set up the client actions to be executed by drag and dropping every needed component.
Doesn’t need much time to set up test cases	Need more time to set up the test case

As illustrated in the comparison table above, Postman offers several advantages, particularly in terms of simplicity and efficiency in setting up test cases, requiring minimal time investment, provide complete test result data, and have wide negative and positive case coverage. The drawbacks of using Postman include the need for testers to be more careful in writing the request body as errors may lead to varied results. Postman proves to be a favorable option for conducting API testing, emphasizing the examination of the API's behavior and response across diverse scenarios. Conversely, BDDFramework serves as a suitable tool for observing the application's behavior when the API Method request is integrated into its logic.

From the UI aspect, testing using Katalon has several advantages, such as the record-replay feature which can make it easier for testers to define test steps. However, in UI Testing on Outsystems applications using Katalon Studio, the tester must pay attention to the XPath of the captured object, especially in the add and delete test cases. The reason is that the resulting XPath may be different and adjustments must be made for each test case.

5. CONCLUSION

From the research that has been carried out, it is known that the Outsystems development platform can be used to create website applications and APIs. OutSystems also has a component called BDDFramework which can help developers in the testing process.

Low code based applications, especially OutSystems, can be tested based on the four existing testing levels, namely component, integration, system, and acceptance. However, from these four levels, there are test objects that cannot be tested, such as program source code due to limited access. From previously developed websites and APIs, there are a total of 23 test cases that can be built to carry out integration testing.

Following the execution of API testing with BDDFramework and Postman, it is evident that the use of Postman proves to be more effective. This is attributed to the comprehensive testing results provided and the efficient setup of test cases, which does not consume much time. Conversely, BDDFramework is more suitable when the testing objective is to assess the application's behavior in various cases related to API usage. However, it is noteworthy that BDDFramework tends to require more time in preparing each test case. In UI testing, employing Katalon Studio is a prudent choice, given its record-replay feature that significantly aids in defining test steps.

Future research endeavors could explore testing low-code-based applications using various testing methodologies and monitor the evolution of testing tools and frameworks specifically designed for low-code applications, particularly those utilizing the OutSystems platform.

REFERENCES

[1] M. Tisi et al., 'Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms', *in STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop co-located with Software Technologies: Applications and Foundations (STAF 2019*), 2019.

[2] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, "Characteristics and Challenges of Low-Code Development," *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Oct. 2021, doi: https://doi.org/10.1145/3475716.3475782.

[3] S. Shridhar, "Analysis of Low Code-No Code Development Platforms in comparison with Traditional Development Methodologies," *International Journal for Research in Applied Science and Engineering Technology*, vol. 9, no. 12, pp. 508–513, Dec. 2021, doi: https://doi.org/10.22214/ijraset.2021.39328.

[4] OutSystems, "The State of Application Development," *OutSystems*, 2023. Accessed: Jan. 04, 2024. [Online]. Available: https://www.outsystems.com/1/state-app-development-trends/

[5] F. Khorram, J.-M. Mottu, and G. Sunyé, "Challenges & opportunities in low-code testing," *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Oct. 2020, doi: https://doi.org/10.1145/3417990.3420204.

[6] J. Salgueiro, F. Ribeiro, and José Metrôlho, "Best Practices for OutSystems Development and Its Influence on Test Automation," *Springer eBooks*, pp. 85–95, Jan. 2021, doi: https://doi.org/10.1007/978-3-030-72654-6_9.

[7] J. E. BENTLEY, W. Bank, & NC. Charlotte. (2005). Software Testing Fundamentals—Concepts, Roles, and Terminology Paper 141-30. *SUGI 30 Proceedings*. Philadelphia, Pennsylvania: SAS Institute Inc.

[8] A. Dennis, Roberta Marie Roth, and Barbara Haley Wixom, *System Analysis and Design, Fifth Edition.* John Wiley & Sons, 2012.

[9] M. A. Umar, 'Comprehensive study of software testing: Categories, levels, techniques, and types', vol. 5, pp. 32–40, 11 2019.

[10] "About the detach process," OutSystems Community, 2020. https://www.outsystems.com/forums/discussion/66302/about-the-detach-process/ (accessed Dec. 27, 2023).

[11] "BDDFramework - Overview | OutSystems," www.outsystems.com, 2016. https://www.outsystems.com/forge/component-overview/1201/bddframework#:~:text=The%20BDD%20Framework%20provides%20a (accessed Dec. 19, 2023).

[12] OutSystems, "Component Testing with BDDFramework Tools," Outsystems.com, 2023. https://success.outsystems.com/documentation/11/developing_an_application/testing_your_application/component_testing_with_bddframework_tools/ (accessed Dec. 19, 2023).

[13] D. Golovin, "OutSystems as a Rapid Application Development Platform for Mobile and Web Applications," Thesis, LAHTI UNIVERSITY OF APPLIED SCIENCES, 2017. Accessed: Dec. 27, 2023. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/132267/Golovin_Dmitry.pdf?sequence=2

[14] "Postman API Platform," Postman, 2023. https://www.postman.com/product/what-is-postman/

[15] "Katalon Platform Overview | Platform Software Testing Tools," katalon.com, 2023. https://katalon.com/katalon-platform

[16] D. Friedenberg, M. Hamburg, J. McKay, M. Posthuma, H. Schaefer, and R. Smilgin, *Certified Tester Foundation Level Syllabus*,

Version 2018 v3.1.1. International Software Testing Qualifications Board (ISTQB), 2021.

[17] Outsystems.com, 2023. https://success.outsystems.com/documentation/best_practices/outsystems_testing_guidelines/integration/api_testing/ (accessed Dec. 21, 2023).

[18] M. Araújo Cabeda, “Automated Test Generation Based on an Applicational Model,” Faculty of Sciences and Technology, NOVA University Lisbon, 2018..