Acceptance Testing at Capital One: Technologies and Processes to Simplify the Testing Process for Developers

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Micah William Cho

Fall 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-related Assignments

Advisor

Rosanne Vrugtman, Department of Computer Science

ABSTRACT
Capital One needed a singular and independent acceptance testing process for their enrollments application program interfaces (API's), as the existing process was inefficient. As a Technology Intern, I utilized a highly scalable Cucumber-based acceptance testing suite and the configuration of a Spring profile to eliminate the need for each developer to undergo an individual configuration process. Additionally, I constructed mocked API responses using tools such as WireMock and Postman, making the testing process independent of all downstream applications and endpoints. Finally, I developed a comprehensive set of acceptance tests in a Cucumber-based format for several of the API's. This new framework saved hours of labor on the acceptance testing process, boosting overall productivity and allowing for more efficient changes to the production environment. Future work involves implementing a similar acceptance testing suite for all of the enrollments API's, and potentially for API's across Capital One's technology department.

1. INTRODUCTION

Time is money. This is especially true for companies that pay employees hourly wages. Thus, the implementation of processes to reduce time spent on tasks that do not directly contribute to projects, such as configuration procedures, is imperative for efficiency.

Before this summer, the enrollments team at Capital One utilized an acceptance testing process that required each developer to go through a four-step configuration process just to run the acceptance tests. First, the developer would have to install an Integrated Development Environment (IDE), such as VSCode or IntelliJ along with the Spring framework. Next, a set of certificates needed to be downloaded on the developer's local machine in a specific file path for proper encryption and decryption. For the third step, a set of program arguments were needed in order to run the API's locally. Both the certificates and the program arguments could only be retrieved through other developers on the team. Finally, the developer would have to gain access to data from secure downstream applications which the enrollments API's needed access to. For a developer with substantial experience with Spring, this process could still take hours, including time waiting for certificates, program arguments, and data from other developers, during which no work could be performed.

In order to make this process more efficient and reduce the time spent on the configuration process, I proposed a Cucumber-based acceptance testing framework that utilized mocked data and a preconfigured Spring properties file that did not require individual setup for each developer. Essentially, the proposed solution would eliminate steps two through four in the existing system, and would reduce the time needed to setup the acceptance testing process from almost a full workday to about half an hour.

2. RELATED WORKS

A number of experts in quality assurance and software testing are recommending using a behavior driven development (BDD) testing methodology, because it is easy to assimilate the wants of the customer with the work of the developer. Eqbal (2022) specifically recommends a Cucumber-based framework because it offers dependency injection containers, which can increase efficiency and decrease state complexity. While other methodologies, such as test driven development (TDD), are recommended by other experts, my solution included a BDD approach due to the compatibility and simplicity of Cucumber.

Several industry professionals also recommend using WireMock when dealing with RESTful API's in Java. Baihaqi (2021) and Bolte (2023) cite the benefits of using mocked data when performing acceptance testing, as complications can arise from accessing endpoints in other code bases. Baihaqi also notes that using WireMock can make acceptance testing significantly less time consuming. Previously, the enrollments team utilized Mockito, a technology with similar function, to mock endpoint responses, but communication with other developers on the team revealed that Mockito is better suited for unit tests, while WireMock was designed specifically for integration and acceptance testing.

3. PROCESS DESIGN

The acceptance testing process for Capital One's enrollments team was improved through the incorporation of a generic, pre-configured Spring profile, a mock environment, a WireMock server, and a Cucumber-based testing suite.

3.1. REVIEW OF SYSTEM ARCHITECTURE

Figures 1 and 2 below show the acceptance testing configuration process before and after simplifications, respectively, were made.

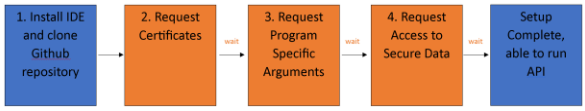

Figure 1 : Existing Acceptance Testing Process

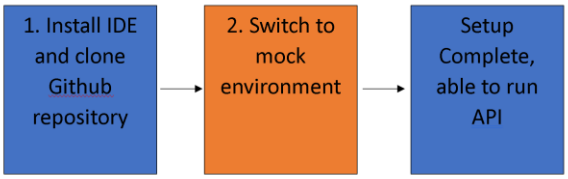

Figure 2 : New Acceptance Testing Process

Figure 1 shows the existing process, which required a four-step configuration process with multiple waiting periods. Figure 2 shows the configuration process after my solution was implemented, where there is only one step in the configuration process after an IDE is installed with no time wasted from waiting.

3.2. METHODOLOGY

In order to reduce the time spent on the configuration process, it was imperative to eliminate waiting periods where developers could not do any work. Thus, elimination of certificates, program-specific arguments, and secure data was a primary target in the process design.

3.2.1. KEY COMPONENTS

In order to eliminate these waiting periods, a new approach with new components was needed. The three major components utilized in the implementation of my solution were a Spring properties file, mocked data from WireMock, and a suite of tests in Cucumber format.

Previously, developers needed to configure Spring properties files for the production, non-production, and quality assurance environments before they could begin to contribute to code bases. In order to configure these files, a set of certificates was needed, which had to be retrieved from another developer and then downloaded onto the local machine. This process was dependent on the availability of developers with access to the certificates and could result in a waiting period in which no further work could be performed.

To eliminate this issue, I configured a Spring properties file along with a mock environment for acceptance testing that used data fabricated in collaboration with a WireMock server. With this new configuration, data accessed in the acceptance

testing environment did not need these certificates for encryption and decryption, eliminating the need for developers to obtain the certificates when not working in the production environment.

The second key component to my solution was WireMock stubs. These stubs contained mocked data in JSON format so that accessing secure data from downstream applications and other endpoints was not needed to perform acceptance testing. This was an issue with the existing process, as developers would be stuck in another waiting period when trying to gain access to this data. With these stubs, the Spring properties file in the mock environment would connect to a WireMock server and use the mocked data from that server instead of the secure data which required access. This allowed for an independent acceptance testing process.

Finally, I chose Cucumber to construct the suite of tests used for acceptance testing, as it was easy to incorporate the WireMock server, and because of its usage of behavior-driven development. In order to protect proprietary information, no other details about these tests can be revealed.

4. RESULTS

In order to analyze the effectiveness of the new system, other interns went through the new configuration process. The average time to complete setup was around half an hour. The existing process took me almost a full workday (eight hours) when I was first assigned to the enrollments team. This reduction to about 6% of the original time is a substantial decrease and will save hours of labor as new developers join the team. The enrollments team will get a substantial amount of paid hours in which new developers can actively contribute to the code base instead of configuring their machines.

5. CONCLUSION

Capital One needed a singular, independent acceptance testing process in order to increase efficiency and employee productivity. This was accomplished by the implementation and incorporation of a generic, pre-configured Spring profile, a mock environment, a WireMock server, and a Cucumber-based testing suite. This solution condensed a four-step configuration process that was dependent on other developers into an independent two-step process.

Through this project, I gained valuable real-world experience in software development and testing. I was able to see how a software engineering team operates while directly contributing to quality assurance processes. Additionally, I acquired many new skills working with technologies, such as WireMock and Spring, and methodologies, such as BDD, that I had not previously used.

6. FUTURE WORK

In addition to the enrollments team, many other software development teams within Capital One would benefit from this acceptance testing process, as Spring is commonly used throughout the company. Future work involves implementing a similar acceptance testing framework on the code bases of these other teams to increase productivity across the technology department. I recorded documentation on my method and processes in order to make a broader implementation of my solution feasible for these other teams.

REFERENCES

Baihaqi, A (2021, April 22). RESTful Integration Testing with WireMock in Java. Semaphore. https://semaphoreci.com/community/tutorials/restful-integration-testing-with-wiremock-in-java

Bolte, D. (2023, June 22). More Than Just Stubs. Medium. https://betterprogramming.pub/more-than-just-stubs-b2a7ad3742b8

Eqbal, H. (2022, April 13). Maximize Software Testing with Cucumber Tests in Spring Boot. Nisum. https://www.nisum.com/nisum-knows/maximize-software-testing-with-cucumber-tests-in-spring-boot