Back-end Software Development: A Look into the Importance of Data Verification

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

Jonathan Hail

Spring, 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

Rosanne Vrugtman, Department of Computer Science

Briana Morrison, Department of Computer Science

Back-end Software Development: A Look into the Importance of Data Verification

CS4991 Capstone Report, 2022
Jonathan Hail
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
jjh3wqt@virginia.edu

Abstract
Walmart was sued for having insufficient prevention for fraud in their financial services department. This was in part due the claim that associates were not properly trained for the financial service they were providing. To address this, I created a back-end service to determine whether an associate is adequately trained. I implemented this remedy in Java Spring Boot as a RESTful API. When the API was finished, it returned a list of the required courses an associate has not taken for each action. In the future, this service could be expanded to departments of Walmart other than Financial Services—the Pharmacy department, for example.

1. Introduction

With the switch to a Java back-end for Financial Services, there needed to be the presence of an API to confirm the ULearn status of Financial Services associates. To do this, I worked with one other intern to add the ULearn integration with the Financial Services back-end. This involved a change from the previous batch process to a live check with ULearn to confirm associate eligibility when they go to perform an action for a customer. If associates have not completed proper coursework through ULearn, they cannot utilize the specific action for which their training is incomplete. It is possible for the associate to complete the courses they still have not done for the action by going through it on the ULearn website. After this, the API would update to unlock the action in real time.

2. Related Works

We decided to use Spring Boot for this project since the rest of our back-end was already using Java Spring Boot. However, this framework was probably originally selected for a number of reasons. According to IBM (2020), Spring Boot is built on top of Spring Framework and is popular because it offers Dependency Injection and other features that provide built-in support for many functions developers commonly use. In addition, as opposed to only using the Spring Framework, Spring Boot adds configuration that makes deployment easier on top of the Spring Framework.

This was built on top of Java, utilizing Spring Boot to add endpoints to the code. According to Mercer (2017), Java is a common choice among companies, because it has a lot of practical uses, is platform independent, and object-oriented. Also, it is an older language so users have a lot of support and libraries at their disposal. For all of these reasons, there are a lot of Java developers, and so Java is a common choice among companies.

3. Process Design

Before starting to write the code for our endpoint, we had to coordinate with the

ULearn team to figure out how to make proper requests to their API. We scheduled a meeting with them to get an idea of how their API worked. From that, we could structure a GraphQL request that returned all of the information we needed to handle on a given associate.

When starting the process of creating the ULearn API, another intern and I had to review the existing Java code as we were adding the endpoint to an existing back-end environment. We looked through the other endpoints and took note of the architecture and general format that the rest of the code has been written in. From there, we outlined a sample Spring Boot API in a new class to give basic test functionality. This endpoint took in the parameters of store ID and customer ID.

After that, we researched how to make a successful API request in Java. In total, we had to make two different API calls to get the information necessary for proper return information. These two API calls were to the operator maintenance API, then the ULearn API. The operator maintenance API was necessary to convert the customer ID and store ID taken in as parameters in our endpoint and convert them to just a Walmart ID. The ULearn API was needed to take the Walmart ID and convert that with our input to provide necessary course information.

We then created a new class to handle calling each API and returning the data to each endpoint. Calling each API was an involved process that required proper information such as http headers and the request body. For the operator maintenance API, this request body was just the store and customer ID, but the ULearn one was a longer GraphQL request, as we figured out earlier. With these requests, we needed to implement proper error handling. This ranged from improper return from the API (the API could be down, as well) in addition to improper user input for our API. To add this, we needed to extend the existing error handling method used elsewhere in the back-end.

After we wrote the code for a basic integration with the two APIs, we had to integrate with two different Walmart cloud services. One service was to provide custom configuration to our API, where we needed to put information such as required courses for each action, which is necessary to expand this to different actions in the future; and endpoint URLs, in case these endpoints change. This service took a lot of research to properly integrate, as it required the addition of more classes in a particular format, as well as changing existing configuration files in the back-end. To test this functionality locally, we needed to create some local files that had dummy information, since the cloud information could not be received locally.

The other service was used to store secrets, such as passwords to prevent hard coding them into our code. This was needed in our code to put in our API keys, for example. Integration also required configuration of the config and the addition of a new class and dummy test files for local testing. Configuration of these new classes for integration required some coordination with co-workers who had experience with implementing these cloud services.

Between all of the above steps, we created testing classes to go along with each class that we created. These testing classes were designed to test all functionality of the current class while mocking external resources. These tests were required to pass maven build, which needed to execute successfully to deploy to the staging environment.

When we finished all of the functionality for our project, we submitted a GitHub pull request to run a few checks and, if all of them passed, deploy our code to the staging environment. From there, we requested the leaders of this back-end to review our code, and if successful, merge into the main upstream branch. We had to make several iterations of changing our code to meet requirements until it finally was acceptable, and then we merged our branch into the main branch.

After our code was approved, we added documentation to our endpoints via the existing Swagger page. This page described our endpoints and how to use them, and provided an area to test querying our API endpoint in real time.

4. Outcomes

The ULearn API was finished and presented to other interns and Walmart Global Tech management. The API successfully returned a JSON list of prohibited actions and the courses still needed to perform these actions on ULearn. To execute this, it is necessary to pass the Walmart associate's ID and store number. This API should reach production sometime in the future to update the existing ULearn implementation.

5. Conclusion

This API provides Financial Services with an updated way to access associate training information and ensure that they have taken the proper courses for the service they are providing. This is legally required and can help prevent scams from taking place. Over the course of my internship, this feature was successfully deployed to the staging environment.

6. Future Work

This work could also be applied to other fields. Currently, it only supports checking requirements for Financial Service associates, but this could be expanded to work for other departments, as well. For example, this could work with the Pharmacy department as a way to ensure associates have taken their proper Ulearn courses. This could be implemented as a modification to the cloud configuration.

References

Mercer, J. (2017, October 25). *Why is Java so popular for developers and programmers?* FRG Technology Consulting. Retrieved September, 20, 2022 from https://www.frgconsulting.com/insights/why-is-java-so-popular-developers

IBM Cloud Education. (2020). *What is Java Spring Boot?* (2020, March 25). Retrieved September, 28, 2022 from https://www.ibm.com/cloud/learn/java-spring-boot