Microservices: Consolidating Functionality Across Multiple Services

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

Jarod Johnson

Spring, 2022

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

Joshua Earle, Department of Engineering and Society

Abstract

Amount Small Business, an independent subsidiary of Amount Inc., has a platform which runs on microservices to take loan applications for smaller banks and return decisions for those applications. Legacy functionality in the PDF document microservice, named "PDX" generates and stores PDFs reports with useless information. The software team decided to replace this functionality with logic that only pulled PDFs from third party financial institutions such as Equifax, ThreatMatrix, and Paynet. I was tasked with imitating the PDF storage and retrieval functionality of the third party reports in a different microservice. In order to do this, I needed to create new database tables for the PDF objects in the alternate microservice, "UDX." Logic from PDF would be tweaked and transferred over to UDX. The main tools used to accomplish this were Orika object mappers, JOOQ database query library, Reactive Java (JavaRx). For testing, I sent local HTTP requests using postman for debugging and ensuring functionality worked as expected, and I used Mockito testing library for Java for unit tests. This new functionality was successfully migrated with a few small adjustments that will need to be made by my team. One of the major changes will need to be in the front end React.js service to ensure that it pulls PDFs from UDX instead of PDX.

1. Introduction

The microservice software architecture is an alternative to the more traditional monolithic architecture that many web applications are built on. To create its platform, Amount Small Business used Micronaut, a Java based application framework designed for microservices. The platform is made up of over 15 different microservices, each with unique functionalities. There were two main microservices that I worked on: PDX which was responsible for generating, storing, and retrieving PDF reports of applications; and UDX which is responsible for doing the same with non-PDF report objects.

A majority of the PDF objects generated in PDX were created with null attributes and/or little to no useful information. Due to this, the team I worked with aimed to consolidate these functionalities into a single service, deprecating the PDX service as a whole in the process. This was my main task during the summer of 2022. There were two main concepts and tools that I needed to grasp before starting to make my own changes in the UDX repository: Reactive programming and Microservice architecture.

2. Related Work

In my first couple of weeks, I dove into a multitude of articles and websites relating to reactive programming and microservices. The first resource I came across, *microservices.io,* was recommended by my appointed mentor. This website holds articles and pages explaining different aspects of microservices on both high and low levels for developers looking to create new applications of their own or switch to the microservice architecture. The most important differences between a monolith and microservice application is the fact that they are loosely coupled, and independently

deployable. Micronaut's documentation page, *Docs* (2022), was another useful resource that held any needed information on syntax, annotations, and the general makeup of a *Micronaut* application.
Last, *Baeldung* (2022) was crucial for my understanding of reactive programming. RxJava is a reactive programming library and Java Virtual Machine extension used almost everywhere in Amount Small Business' Java code base. This covered the basics of observables, flowables, nullables, and the concepts of publishers and subscribers. Grasping reactive programming as a concept was one of the largest obstacles I faced in my internship.

3. Process Design

The micro-services that I worked with mainly benefitted the front-end service that was used by the company's employees in order to manually check and process loan applications. The two previously mentioned services, UDX and PDX, are just two of the many services that interact with the API gateway and UI service. All services perform individual tasks that are ultimately leveraged by the API gateway to pass values to the React front end service. Originally, the API gateway would call endpoints in both PDX and UDX separately in order to retrieve the PDF and report objects which both end up being displayed in the same webpage. See figure 1 below.

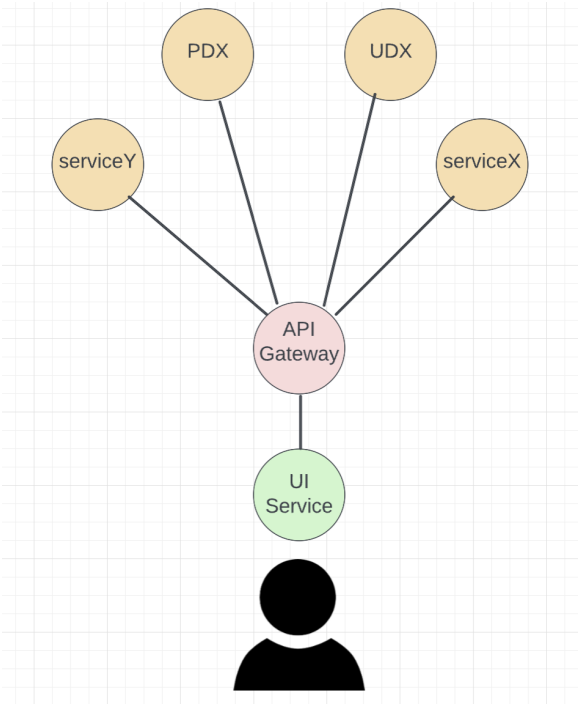

Figure 1: Simplified Conceptual Layout of System Architecture

The endpoints that were called by the UI service in PDX needed to be transferred to UDX so that PDX could be deprecated while all original functionality continued to run without issues. To start this process, a new SQL migration file would need to be created to add a table for storing PDF objects in UDX's database instances. The important attributes that each table entry should include are: ID, ReportID, ApplicationID, Data, and DataSourceID. The ID is a primary key, which uniquely identifies every single entry in the database. The ReportID is a foreign key which references which report it is referencing. A foreign key is used here because of its many-to-one relationship aspect. In order to prevent any data loss, all PDF reports are to be stored even if they are outdated. Therefore, a single report may have multiple

PDFs connected to it. The ApplicationID refers to the individual application that all of the reporting and information applies to. The Data attribute is a string value that holds the bytecode for the PDF document; this is converted to a viewable PDF at runtime when an instance of one of the new UDX endpoints is called. Last, the DataSourceID references which third party financial reporting agency originally created the document.

The next step was to write the actual business logic that would retrieve any PDF objects and return them to the front end. The most crucial endpoint used to do this involved a HTTP GET request endpoint that passed in the specified ApplicationID in order to retrieve all PDF objects by this path variable. The second endpoint I needed to implement involved retrieving a single PDF by passing in the primary key, ID, into a HTTP GET request. This endpoint was never actually used on the front end service, but the convention is to always have working endpoints that retrieve a database object by its primary key.

Once an endpoint was fully implemented in each layer of the micro-service, they needed to be tested. The layers of a Micronaut service are comprised of a controller, service, dao, and database. Each of these has its own responsibilities with the majority of any complicated logic lying in the service layer. I used Mockito unit tests in order to separately test each of these layers. Mockito tests allow for a forced response from a test instance of one of the layers. This allows a programmer to isolate which layer is being tested.

Once unit tests were written, I performed local testing with the Postman HTTP request tool. In order to being this, I created a local Postgresql database table with test data. Then the UDX service is run in my local environment. Followed by calling the endpoint in Postman. The path used in testing would require manual input of an ApplicationID, like: localhost:8080/PDF/appID123/. A similar endpoint was also called locally to test the secondary endpoint needing the primary key of the PDF table.

4. Results

After completing the implementation of the migrated functionality, all underlying bugs and issues were eventually resolved. This allowed me to deploy my pull request into the live testing environments that ASB uses. I was able to see my changes work properly by viewing PDFs of different test reports in the UI service. My internship before my branch was actually merged to the development branch. However, my mentor and coworkers confirmed that my code would be built upon as they continue to improve all of their services.

5. Conclusion

The time I spent as an intern at ASB was a great learning experience. I was able to hone my skills as a software engineer, as well as improve some of my soft skills. Version control, relational databases, HTTP requests, and API’s are all foundational concepts for an aspiring software engineer to understand. Completing an entire project involving the

aforementioned technologies was an important step in forwarding my career.

6. Future Work

In order to fully take advantage of the contributions that I made during my internship, my coworkers will have to update the front end React.js service to use UDX for PDFs as opposed to PDX. This should be a relatively easy task. Since PDX will now be deprecated, the only edits needed in the UI service will be to replace any use of PDX and its endpoints with UDX and the replacement endpoints that I implemented.

References

Richardson, C. (2014, March 18). *What are Microservices?* microservices.io. Retrieved May 25, 2022, from https://microservices.io/

Rocher, G. (2018, May) “Docs,” *Micronaut Framework*. [Online]. Available: https://docs.micronaut.io/index.html. [Accessed: 30-May-2021].

Baeldung. (2017, September) “Introduction to rxjava,” *Baeldung*, 05-Jul-2022. [Online]. Available: https://www.baeldung.com/rx-java. [Accessed: 23-May-2022].