Dynamic Order Generation for Automated Testing

CS4991 Capstone, 2021

Qasim Ali
Computer Science
University of Virginia
Charlottesville, Virginia
qha4bh@virginia.edu

ABSTRACT

A multinational conglomerate e-commerce business (Company X) spent an excessive amount of time on the tedious and repetitive task of creating sample orders for testing. I decreased the time needed to test systems by creating several REST APIs that automated test order generation. I also created APIs to publish test orders to the proper environment. For other teams that only needed to generate sample orders, I created a frontend interface.

These improvements significantly reduced pre-deployment testing time and allowed other teams to easily generate sample orders. A possible expansion of this project would be to add the ability to not only generate and publish sample orders but also track published orders and ensure they are successfully routed to the correct team.

1 Introduction/Background

Company X is a multinational conglomerate which offers online shopping services to its customers. Customers can place orders online and either have the items delivered to their homes or pick them up in-store. The quality engineering team in Company X spent a significant portion of their time creating sample online orders for use in pre-deployment testing. Online orders are sent through XML (Extensible Markup Language) files. Each XML file contains all of the details associated with an order. Creating each sample order involves creating several layers of XML and populating their respective fields, requiring a substantial time commitment from engineers. Furthermore, once an order is created, it needs to be published to the development environment. Engineers also performed this task manually, and needed to publish each sample order separately. Not only did these inefficient tasks waste the time and talent of engineers, time spent creating these orders increased testing time and therefore increased overall cost for Company X.

My approach to solve this issue was to automate the time-consuming aspects of order generation. I created an application that could take as input the parameters for a sample order and send back the fully completed order file. This application allowed Company X engineers to programmatically create orders and eliminated the need to manually write up the order file. Similarly, the application had the ability to take in a generated sample order as input and publish the order to the appropriate development environment. This also allowed engineers to perform order publishing programmatically rather than manually.

2 Review of Research

There are three main ways to create a sample order XML file: manually creating the file or manually writing fields into a template, programmatically inserting fields into a template file, or dynamically creating an order file. Manually writing fields into a template is how Company X previously created sample order files. Engineers manually change the order field values in an XML template depending on the test being run. An advantage of this approach is that developers have full control over what the test order should look like, and the process of creating the order does not require any computing resources. However, a disadvantage of this approach (and the reason this project was taken on) is that it is slow and requires a significant amount of time from engineers that could otherwise be spent on more productive tasks.

Another method to create sample orders is to have a program that takes as input all of the fields that are needed in an order and automatically insert those values into an order template. This approach would save engineers a

considerable amount of time, but would still require them to supply every single field value in an order. This would still require a time commitment from engineering since there are a considerable number of fields in an order.

Dynamically creating test order files would solves many of the problems associated with the first two methods of sample order generation. In this approach the engineer would supply only the fields that are important for a particular test, and then an application would build the XML file and automatically give all other fields a default value. This approach has most of the flexibility of the manual approach while also having most of the time savings from the template approach[1]. For these reasons, I use this approach for sample order creation when building my application.

3 Project Design

The main goal of this project was to build an application that could programmatically generate sample orders quickly and without much manual input from engineers. The main functional requirement for this application was to allow engineers to programmatically generate and publish sample orders. Order generation and order publishing needed to be performed separately so that orders could be created but not published and vice versa. The order generation tool needed to take as input as many or as few order specifications as the user provided and would in return give the user a fully completed sample order. The order publishing tool needed to take a completed sample order as input along with an environment and publish the order to the given environment.

Another functional requirement was to have a front-end website where other users could generate orders. This website needed to be easy to use for users unfamiliar with the online order system and clearly show the structure of an order. Like the order generation tool, users should be able to supply as many or as few parameters as desired and should be able to view the generated order. Users should also have the ability to publish orders and receive feedback whether or not the order was successfully published.

In terms of non-functional requirements, high availability of the system was crucial. Requests would be made to this application to create and publish orders every time tests are run. Therefore, it is important to have maximum uptime as tests would be unable to run without this application. Security was also an important requirement. This application was intended for internal use only at Company X and therefore should not be accessible from the internet. Since this application would receive thousands of requests during stress and stability testing, it was important to have low latency to ensure tests were run in a timely manner.

3.1 Overview of System Architecture

The application has three core components: API servers, a NoSQL database, and an asynchronous background task.

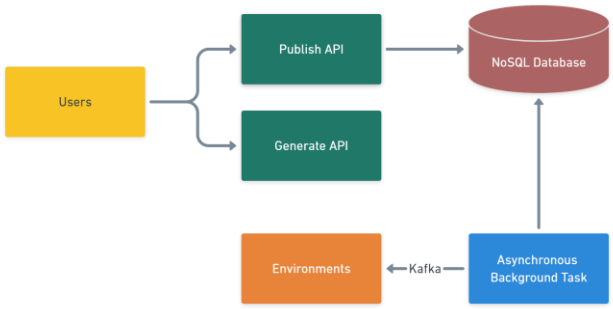

Figure 1: Core System Architecture

Users will interface with the application APIs either through direct API requests or through the front end. The publish API will store any valid publish requests in the database. The asynchronous background task sequentially processes database entries and publishes entries to the correct environment.

3.2 API Servers

The API servers are built using Spring Boot. This framework was chosen because of its vast library support and high compatibility with other Company X software. The API servers serve the front end to users and support two main requests: generate and publish.

The generate API requires a JSON (JavaScript Object Notation) body. JSON is similar to XML in that it is a notation to represent information in a computer readable way. This JSON body will contain all of the order fields the user would like set in the sample order. The API takes whichever fields were given in the JSON body and creates an order object using those fields. Any fields that are not included are either randomly generated or given a default value. The API sends back a response containing the sample order as XML and HTTP status code 200 OK. If any fields are malformed, a sample order is not returned and HTTP status code 400 Bad Request is sent.

The published API also requires a JSON body containing a fully complete sample order, Kafka topic, and an environment to publish to. If the sample order is malformed, or the Kafka topic or environment does not

exist, the API responds with HTTP status code 400 Bad Request. Otherwise, the sample order, Kafka topic, and environment are stored in the database and HTTP status code 201 Success is returned.

The API Servers are hosted on Azure which is Microsoft's cloud platform. Cloud services were chosen due to their high stability and reliability. Azure was also chosen based on its ability to scale resources based on server load. At times of high traffic, the system can add more servers and scale up existing servers, and at times of low traffic the system can scale down and decrease cost. These benefits of Azure ensured that the application met the requirement of high availability and low latency. Using Azure also helped to achieve the security requirement. Microsoft spends a significant amount of time and money to ensure that Azure has complete security, far more than Company X could reasonably spend using custom servers.

3.3 Database Design

Since the data being stored is not relational and queries will be trivial, a NoSQL database was used to store publish requests.

Figure 2: Database Diagram

As seen in Figure 2, the Publish Request table is the only table in the database and simply contains the JSON body received by the publish API. The sample order XML is stored in the JSON body as a string.

3.4 Asynchronous Background Task

Whenever the publish API is called, a new publish request entry is added to the database. The purpose of the asynchronous background task (ABT) is to process these publish request entries as they are added to the database. Each entry is processed in the order it was added to the database. For each entry, the ABT uses Apache Kafka, a data streaming platform, to send the sample order to the appropriate Kafka topic in the given environment[2].

4 Results

The application developed for this project is currently being used by Company X engineers. When quality engineers write tests, they make requests to the generate and publish APIs. For example, if an engineer wants to test how the system reacts when a customer orders three different items, they can use the application to generate and publish a sample order matching that description at the beginning of their test. The engineer can then write the rest of the test knowing this new order has been published to the development environment.

Based on the observations of quality engineers at Company X along with my personal observations, this application decreased the amount of time needed to complete pre-deployment testing by twenty to twenty-five percent. This decrease in testing time, along with the increase in availability of quality engineers to work on other projects, is likely saving Company X a significant amount of money.

5 Conclusion

The goal of this project was to decrease the amount of time spend by Company X engineers creating sample orders. The application developed to solve this issue successfully achieved this goal and therefore provided significant value to Company X. The application also created a front-end environment for less technically inclined employees at Company X to interface with order creation and learn more about online order structure.

6 Future Work

A possible expansion of this project would be to add the ability to track where a sample order is sent after being published. Once orders are published to an environment, they are routed based on details of the order. For example, an order for in-store pickup will need to be routed to the store that the customer would like to pickup from. Similarly, an order for home delivery will need to be sent to the closest fulfilment center that has the ordered item based on customer address.

Unfortunately, the application in the current state can only generate a sample order and publish it to an environment. Any testing related to where the order is sent must be performed by quality engineers writing the tests. To add

this functionality, the publisher API would need the ability to determine where an order should be sent. This calculation could be performed by checking the type of the object being sent. If the order is of type pickup, ensure it is sent to the store that the customer selected. If the order is of type delivery, ensure it is sent to the correct fulfilment center based on customer address.

This additional functionality would have the potential to fully automate the process at Company X for certain types of tests. Integration tests that simply check that an order was routed correctly without errors could be entirely automated. Scripts could be written that would execute before each deployment that randomly generate sample orders and then ensure they are routed correctly. This would have a substantial impact on Company X because these forms of tests would no longer need to be written by quality engineers.

7 UVA Computer Science Program Evaluation

Overall, the computer science department at University of Virginia (UVA) prepared me for several aspects of this project. Specifically, CS 3240: Advanced Software Development was crucial to my completion of this project. This course provided experience developing software in an agile scrum team. This project was developed in the same way. The course also provided introduction to topics such as REST APIs and web development. Having background in these topics increased development speed substantially.

However, engineers working on this project were expected to have an introductory understanding of databases. I did not have this understanding because I had not taken CS 4750: Database Systems. Students may benefit from making CS 4750 a required course, and would likely benefit from opportunities to take the course earlier in their course of study. This would entail offering additional sections of the course with additional capacity so that underclassmen have the opportunity to enroll.

References

[1] Lance Ashdown, Jack Melnick, Steve Muench, Mark Scardina, Jinyu Wang. 2005. Oracle XML Developer's Kit: Programmer’s Guide 10g Release 2 (10.2). https://docs.oracle.com/cd/B19306_01/appdev.102/b14252/toc.htm

[2] Gwen Shapira, Todd Palino, Rajini Sivaram and Krit Petty, 2017. Kafka – The Definitive Guide. [S.l.]: O'Reilly Media, Inc, USA.