Trinity College Dublin

The University of Dublin

School of Computer Science and Statistics

Internship Technical Report

Efeosa Eguavoen
Master in Integrated Computer Science
Internship Technical Report
Supervisor: Tim Fernando

July 15, 2021

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this report is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed:Efeosa Eguavoen Date:15/07/2021

Contents

1 Goals

1.1 Technological Goals

Goal: Learn 1 new technology in asynchronous programming and successfully implement it in a task.

Goal Achievement: In terms of the accomplishment of this goal, I feel I completed it and might have even somewhat overachieved. I built a consumer that uses the Kafka asynchronous programming technology and also used HubSpot's in house queueing system TQ2 in a subsequent task to bring it to completion.

Goal Evaluation: This goal was somewhat difficult to accomplish as using Kafka isn't something that is necessary in every task. To drive this goal to completion, I specifically asked my team lead if I could work on a task that would allow me to gain some exposure to async technologies. One of the initial problems was how complicated Kafka was as a system and how in-depth the documentation was. I experienced some paralysis with getting started initially as there was so much to consider and I was over-thinking the task at hand. To overcome this, I sought some guidance from other members of my team that had previously worked on Kafka and asked them for some guidance in how to start. This enabled me to be able to get a better idea of what the task I was doing actually was and how to use Kafka to accomplish the task.

Success Factors: The biggest thing that helped me to get this goal to completion was the help of my team mates. By making it clear with my team lead that I had an interest in async technology and wanted to work with it, my team lead actively assigned me tasks that allowed me to learn more about it. Using my team mates expertise and experience as a resource helped me drive this goal to completion as they gave me the blueprint to using Kafka and what pitfalls to watch out for which made the process much smoother.

Goal Re-evaluation: If I could change this goal, I would've tried to get more exposure to TQ2, which was the in house software used. I used it only a few times during my time at Hubspot but I feel like I didn't get as deep an understanding into how it works like I did for Kafka.

Goal 2: Develop my coding style to be compatible with the world of work to reduce time how long my pull requests are in review by understanding the coding practices at Hubspot

Goal Achievement: In terms of accomplishment, I don't feel I accomplished this goal to the fullest extent but I definitely made a lot of progress on this goal. I learnt how to write clean, maintainable code and about how to write code with others in mind.

Goal Difficulty Evaluation: Overall, I think this goal might have been slightly difficult as changing your coding style drastically, quickly is rather difficult and takes a long time to do than I had at the internship. One of the biggest barriers to achievement in this goal was I had to rethink about how I write code and instead use patterns and such that were widely used as the code would need to be maintained by someone else who I may never meet months down the line, so the code would need to be able to explain itself. To solve this, I actively took notes on my pull requests to see what I was getting pulled up on multiple times and made the active choice to prevent that when I was completing tasks. By the end of my intership, I realised the average time a task spent in pr had reduced by roughly 25% and the comments in relation to coding style were much less frequent. Whilst I couldn't incorporate everything into my coding style, I could see I made serious progress and achieved my goal overall.

Success Factors: The biggest driver of success for me was the comments on my pull requests and reading 'Clean Code' by Robert Cecil. Using the core concepts in the book gave me the foundation to build off as well as alerting me to some of the mistakes I had been making up to this point. The pull-request comments acted as a sort of positive reinforcement to give me a gentle remind me about certain things I had been slacking on.

Goal Re-evaluation: One thing I would change here is I would expand this out to testing also. I learnt a lot about testing while at HubSpot but I feel I didn't learn as much about how to write good tests and such as much as I would've liked and this was sort of my weak point.

1.2 Soft Skill Goals

Goal 1 : Be able to receive feedback and also give back constructive feedback in order to increase my growth as an engineer by giving me instruction in areas I may be lagging behind, whilst being able to give my team lead direction in how best I learn

Goal Achievement: I think I achieved this goal completely. By the end of my internship, I had created a positive feedback loop between myself and my team lead and other members on my team, thus creating an space that was open and judgement free that enabled us all to grow as engineers.

Goal Difficulty Evaluation: I think this goal was absolutely necessary and even perpetuated into my other goals as being able to give and receive feedback is important as an engineer in terms of growth as you need to be able to change your style and learn new things constantly due to the nature of the industry. The goal wasn't difficult to achieve as at HubSpot there's a culture of growth and positivity around giving each other feedback and especially on my team as my team lead felt responsible for the growth of each member of the team, making it easier for me to achieve this goal. One challenge I faced was accepting that the comments on pull-requests aren't there to attack your ability to code but are instead there to help you to write better code. By being able to receive this type of feedback better and re-orientating how I thought about it, it enabled me to realise faults in my own work and work on fixing them. In terms of giving feedback, I had weekly personal meetings with my team lead to discuss things in general. In these meetings I focused on discussing how I work as a person and how my team lead can work alongside this to give me feedback and help me grow. These meetings accelerated the achievement of this goal as I could continually practice giving good feedback as I could continually tune what I should say as I could see weekly what was and wasn't working.

Success Factors: The biggest drivers for success on this task were the weekly meetings with my team lead. Having this pre-allocated time every week to give each other feedback made it that bit easier to discuss things as I knew that this space was made specifically to talk which created a mental break for myself

Goal Re-evaluation: If I could go back and change anything about this goal, I would expand it so that I could also be able to give feedback on other engineers pr's as I felt this was something I didn't do enough and could improve on. This would also have the added side-effect of being able to learn from the senior engineers in terms of how they write code, which I could incorporate into my own style.

Goal 2: Increase my networking skills by having at least 10 1:1's with people on and outside my team to widen my network and learn more about other peoples progression in Hubspot

Goal Achievement: I overachieved in terms of this goal. I managed to meet over 15 people in the time I was there and grew my Linkedin connections by over 50 which I felt made this goal an overwhelming success.

Goal Difficulty Evaluation: I felt like this was an easier goal to accomplish in relation to my other goals as setting up meetings with people at HubSpot wasn't difficult as there was already numerous things in place at HubSpot for meeting new people and as an intern, the intern team did coffee meets with interns based on different teams around the world. I didn't have any obstacles in achieving this goal.

Success Factors: The coffee chats with the other interns were important in driving this goal to fruition as the platform to meet new people and such was already in place, I just had to use it.

Goal Re-evaluation: If I could make an improvement on this goal if I could go into the past, I would ask to talk with people that maybe aren't in the engineering teams so I could get a better idea of HubSpot as a company overall.

1.3 Major Takeaways

From setting goals to achieve during my internship, they helped me to guide my focus and also encourage new behaviours. By actively trying to change how I code, I found myself thinking more about how other people would perceive my code and what pitfalls they would point out. Goal setting also helped me manage my tasks and prioritize them in-terms of what would drive my goals forward and what would take away from them. I also found a sort of motivation from setting goals as I had something bigger I was working towards and felt a sense of pride when I took major steps towards fulfilling these goals.

2 Weekly Reports

2.1 Week 9

This week, I re-orientated back to working on the Kafka Consumer. I had shifted focus from it for a few weeks as my team was focused on leaving Code Red which required us all to focus on doing the Hublet based work. As it had been a few weeks since I had last looked at the consumer, I spent some time re-familiarizing myself with it and figuring out what was the next step from where I had previously left it.

This week I had a meeting with my team lead to discuss the crit-sit that had occurred. I got some reassurance from my team lead not to take it personally or whatever and to just focus on continue working to my capacity as these things tend to happen.

2.1.1 Reflection

On reflection, I should have not left the kafka consumer so suddenly and written myself some notes on how to get started back on it when I returned to work on it as I spent a few days just re-reading my own code that could have been spent instead making some more progress on the consumer. Also, I realised how important it is to have a good team lead on your team that can provide you with reassurance and such when things don't go according to plan and can help you get back into the swing of things.

2.2 Week 10

The major task this week was to split up the task of making the whole consumer into smaller, actionable parts. After coming up with a plan of all the parts that needed to be done, I started on adding the filtering logic to the consumer to check what sort of message is being sent for processing. This wasn't too difficult and was completed and sent to pr by the end of the week.

This week I had a 1-1 meeting with an intern on the security team in the states. It was interesting getting to see what other sort of projects interns get to work on at HubSpot and also advanced my goal of meeting new people and expanding my network.

2.2.1 Reflection

In hindsight, splitting up the tasks for the kafka consumer definitely helped me to speed up the process of getting it done as I had more focus on what I was doing each week and helped me to narrow in on what was important and not to get caught up trying to get it all to work all in one go.

2.3 Week 12

This week, the primary focus was to create the function that would create the request body that would then be placed as part of the request to be sent to one of the endpoints. For this, I used a builder pattern as other classes have similar methods so I reused some of the code in those classes and just tweaked it for use here. Testing this change wasn't particularly difficult as it was just a matter of doing a series of asserts on the different fields in the class.

This week, a new intern joined the team. I was instructed to kind of show him the different products our team is responsible for and what our team mission is.

2.3.1 Reflection

The new intern joining my team really gave me some perspective on how far I had come from when I started at HubSpot. A lot of the things that I had struggled with initially like getting to grips with the different tools and the code base I no longer had issues with but could even show another person tips and tricks I had learnt from using them. This really helped me to put my growth into perspective.

2.4 Week 13

This week the major thing I worked on was writing the endpoints that the kafka consumer would hit to send data to the database. There were some examples of endpoints in the database already so I could use those as a sort of framework for the 2 new endpoints I was introducing. I initially had some issues with verification but I switched the endpoints to become internal and that solved a lot of the issues.
This week there was a talk from the CTO of HubSpot, Dharmesh Shah. All the interns were invited to hear him speak and it was really interesting hearing about how HubSpot came about as he was one of the co-founders of the company.

2.4.1 Reflection

I think the way I wrote the endpoints could have been more robust. While it's secure and will work, I thought the code was slightly messy as I rushed through it slightly as I wanted to move on to something else. The testing for this also wasn't as thorough as I'd have liked, I only wrote a series of unit tests and didn't perform an integration test with the other components.

2.5 Week 14

The final part of the consumer I wrote this week, the wrappers for the endpoints. The wrappers were there for re-usability so other classes could reuse the endpoints as necessary. The testing for these wrappers was kept minimal as they were just code covering the endpoints that were previously tested.

2.5.1 Reflection

This week I learnt about why having wrappers and such is so important. Having proprietary methods to access frequently used endpoints and such can really cut down on code duplication and also it makes it a lot easier to integrate into other projects as you can just use a builder pattern to create your request.

2.6 Week 15

The main focus of this week was to do some thorough testing of the consumer from end to end. While each of the individual parts were working as intended, a full end to end test was necessary to make sure it worked as intended for users. Some bugs were encountered while writing the consumer which took a while to fix.
This week our team also rebranded from Flagship Integrations to ERP-Strategic Integrations as the focus of our team and our vision was shifting to be more of a toolbox for having the core processes needed to keep companies running.

2.6.1 Reflection

This week, I learnt about how important the different types of testing are. If I hadn't done the end to end testing, I wouldn't have found some of the bugs in the code that I had encountered. Unit testing is fine when you're only working on smaller changes and not on larger systems, but using a mixture of unit testing, integration testing and end to end testing is the best way to make sure a system is tested exhaustively.

2.7 Week 16

For my final week, I spent the majority of it doing clean up on the consumer and handing off my remaining tasks to my team mates. Whilst I had completed the consumer, there was still some other work remaining for making line items editable I couldn't get to so I transferred this on to the team to see it to completion.

2.8 Final Reflective Diary

From working at HubSpot, I've learnt a lot about what it's like to work and be part of a team. Some of the major takeaways from the internship are as follows:

2.8.1 Ask Questions

One of the most important things I've learnt over the course of the internship is how important it is to ask questions. When you start working at a new company with a new set of tools, a whole new code base it can be quite overwhelming. My first instinct was to try tough it out but this was a mistake, I should have asked for help sooner which would have helped me on ramp sooner. I learnt that taking a small amount of time like 30 minutes or so to attempt things myself and then asking for help is the best way to get things done and moving as I would know enough to be know what questions to ask that would unblock me.

2.8.2 Get Familiar With The Tools

This was something I found incredibly useful and helped me to increase my productivity. Being comfortable with my development tools and learning the different IDE shortcuts saved me a few minutes constantly which added up over the course of the internship.

2.8.3 Taking Responsibility For Your Work

A major thing I found to be important was taking ownership of your work and being the one responsible for it. Being an intern it's easy to fall into the misconception that what you do doesn't really matter and you can make mistakes and nothing will happen, but this is a big misconception. I found that from breaking things that the work I do does have a real impact on our customers and that it goes beyond me with whatever I do. It's important to own up when you make mistakes and participate in conversations at work as you're part of the team and aren't treated differently just because you're an intern. Taking pride in your work and executing to the full extent of your ability is a good way to show ownership.

2.8.4 Ask For Feedback

At HubSpot, I had weekly meetings to give and receive feedback from my manager that was invaluable to my growth over my time there. Feedback is imperative to your growth as you often don't know in what sectors you may be lacking and in what sectors you may be strong in. Feedback is a great way to figure this out. While not all feedback is positive, not taking it personally and understanding you're being told this as your manager wants you to grow and be the best engineer you can be is important.

2.8.5 Setting Goals

The course of an internship is over quite a long time so it can be quite difficult to stay on track and you can drift somewhat and stagnate. Having goals and things you're working towards can keep you on track and keep you focused. Taking active steps towards your goals can help you grow faster and will keep you motivated as you work.

3 Executive Summary

3.1 Executive Summary

During my time at HubSpot I was assigned 2 major tasks that I worked on over the course of the few months of my placement. To get me started and introduced to the tech stack at HubSpot, I first started with an onboarding project that enabled me to become somewhat familiar with the stack by using many of the major tools and languages that are commonly used in HubSpot. This introductory project introduced us to the major programming languages in HubSpot, some of the predominant patterns used at HubSpot such as the Builder pattern and how to use things like annotations and dependency injection. There was also an aspect of the onboarding that required us to build a react app and create a front end that plugged into a simple backend we had created also. This wasn't a part I spent too much on as I was on a predominantly back-end team.

Once I had completed this, I was introduced to my first major task, building a multicurrency consumer to make currency fields editable in the UI. This was part of a wider set of changes to multiple repositories my team owned to standardise fields so we could make the prices of items and such editable in the UI, which was a major pain point for our customers. This change was quite large in size and required multiple pull requests and a lot of code to be written to get it going. For this change to be made, some endpoints had to be written alongside a brand new kafka consumer. This task was what I spent the majority of my time at the internship working on.

The other task I worked on during my time at HubSpot was a change to users using Shopify as part of a company wide initiative to make us GDPR compliant. In order to do this, a new EU Hublet was made as previously all data was routed to the US and handled there but under GDPR, users have to be allowed to both chose where their data is sent and where it's stored. To make us GDPR compliant, I had to change the webhooks for all users using Shopify to point to a load balancer that would then determine where the data should be routed. To do this I had to write a backfill job that performed some major operations on the database and changed the data in there. This change was fairly straight forward but there was a bug that made it to production that caused all users webhooks to be broken. This bug was fixed and the task was re-run but there was a period of down time that took a period of remidiation to fix and retrieve all missed data that couldn't be synced while the webhooks weren't active.

4 Project 1: Multi-Currency Consumer

4.1 Introduction

During my time at HubSpot I was assigned 2 major tasks that I worked on over the course of the few months of my placement. To get me started and introduced to the tech stack at HubSpot, I first started with an onboarding project that enabled me to become somewhat familiar with the stack by using many of the major tools and languages that are commonly used in HubSpot.

Once I had completed this, I was introduced to my first major task, building a multicurrency consumer to make currency fields editable in the UI. This was part of a wider set of changes to multiple repositories my team owned to standardise fields so we could make the prices of items and such editable in the UI, which was a major pain point for our customers.

When I was first presented with this task, I thought it would be fairly simple as on the surface, it seemed like simply getting what type of multicurrency change is being made and then just adding the field to the database. Once I had gotten started, it became much more complex and required multiple steps and code refactors whilst the project was being built.

For example, when a user makes a change to a multicurrency to their HubSpot portal, the consumer would then process what type of change was made (addition/deletion being the only two the conusmer was concerned with) and from there, the consume would create a a HubSpot standardized field and place it into the database for the user alongside other fields such as the label to be displayed in the UI in the case of an addition and in the case of a deletion, the consumer would remove the corresponding fields from the database.

This change seemed pretty simple but there's multiple hidden requirements that aren't immediately obvious. For instance, what should be done for all the multicurrencies that are already in a user's portal? What should be in the case where a user doesn't yet have any multicurrencies in their portal and decides to add one? How are calls to/from the database to be handled? Questions like these and many others needed to be answered and dealt with before and during the coding process.

Luckily for me, my team had started some of the necessary work making it easier for me to get to coding. For instance, my team already had created the Kafka producer and set up much of the infrastructure so I just had to work with it to do complete this task.

4.2 Design Approach and Architecture

To complete this task, I split the overall task into multiple subtasks to be completed:

- Standardized Multicurrency labels and fields Generator
- Build Kafka Consumer
    - Filter Kafka message into Add/Delete
    - Pre-build request body for processed message
    - Send request
- Create Internal Endpoints for adding/deleting multicurrencies
- Build Wrappers for hitting endpoints to be used by Kafka Consumer

I went with a top down approach initially when defining each of the individual components and fleshing out components as I went along. I decided to use this as I was sure there may be some hidden requirements that may arise during the coding process. Also, the top down approach to software engineering is much more requirements driven and was more in-line with clean code practices as I could avoid writing code I may not need. For some of the individual components, I did go with more of a bottom up approach like the kafka consumer, as this consumer could be reused for something elsewhere with some changes to the functionality.

For the overall architecture, I went with one of the more common design patterns, the **Model-View-Controller (MVC)** system. I went with MVC due to the scalability it provides alongside the support for asynchronous techniques like Apache Kafka which is a major framework in use at HubSpot.

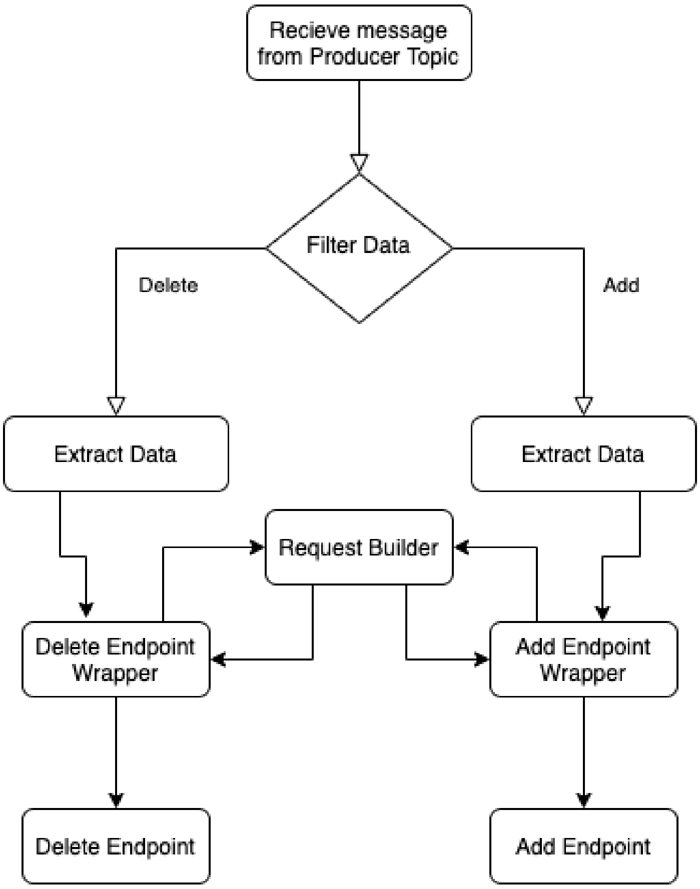

Figure 4.1: Flow Diagram of Architecture

4.2.1 Model-View-Controller (MVC)

MVC is a common architecture pattern used in web based applications. MVC separates out the layers of business logic and data interactions(model) from the presentation(view) and intermediary between the two layers(controller). This pattern is so popular as it decouples each of the major aspects of the design from each other and is also highly scalable due to how modular the design is.

Model

The data and business logic of the application are wrapped in the model layer. It's frequently made up of a number of interconnected domain objects. The models represent the nouns or

entities in the system, such as user or order, and interact with any of the other layers of the system.
The model layer, on the other hand, includes not just models but also validators, methodologies, and any other element that directly affects or distorts the underlying data of the application.

Each model has a user interface that reflects the logical functions that can be performed on the data it represents. An illustration of the capabilities contained in an application's model layer: A *FullName* method on a User class returns a merger of the first and last names associated with the user it depicts. The same object also has an *UpdateName* function that updates the corresponding name in the database. The same object could then use a Validator instance to make sure none of the fields are empty before being placed into the database.

The model Layer often utilises some form of **Object Relational Mapping**, which is a layer of abstraction which is designed to transfer data from the the scalar values used in database storage to the objects used in object orientated programming languages and vice versa. While a traditonal SQL database is used in many instances, noSQL databases and/or web services that perform some sort of data persistence could be used as the database layer instead.
In many versions of the model class, a set of static methods which utilise the ORM to search for and instantiate instances of the model.

View

The 'View' layer is responsible for the visual layer of the model. This represents the graphical user interface a user may interact with. Views perform as the output of the system, whereby data from the model is represented. In many MVC architectures, the view is sensitive to changes in the data held in the model and hence self updates to reflect any new changes in the data. The view itself does not transformations or alterations of the data itself, it just acts as thin layer between the model and the controller. Data from the user may be inputed in the view layer but is ultimately sent to the controller to be dealt with.

Controller

The controller layer is primarily responsible for processing user input. A controller normally has a 1-to-1 relationship with a view or a 1-to-many relationship with a series of views. When a user inputs data via an interface in the view layer, this triggers an event to occur which is dealt with by the controller layer. The controller layer could for instance instruct the view layer to resize itself if a user resizes the size of the input window. Another example is to update the data in the model and hence update the view layer if a user inputs new data into the view layer initially. Controllers have permissions to frequently access the model layer and re-construct and transform the data in that layer.

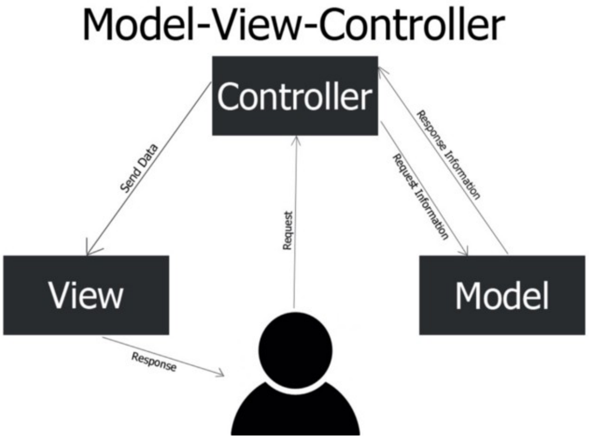

Figure 4.2: Model-View-Controller overview (1)

4.2.2 Technologies Used

Apache Kafka (2) Kafka is an event streaming software used to capture real time events from sources like cloud services in the form of streams which are then sent to other services as necessary. Kafka provides 3 main services:

- Publishing(write) and subscribe to (read) streams of events,
- Store streams of events for as long as necessary
- Process streams of events as they occur

Kafka works as a series of servers and clients communicating over TCP. An **event** records something that happened. A **Producer** writes events to Kafka while **Consumers** subscribe to these events.

Events are organised into **Topics** which any producer can write to and any consumer can subscribe to. Topics are partitioned i.e spread over a number of buckets located on different kafka brokers. This enables a high degree of scalability as producers and consumers can read from many brokers at once making sure there's no congestion on the network during high traffic times. Kafka also guarantees any consumer will read any event in a topic in the same order as they were written to the consumer.

Singularity (3) Singularity is a platform that enables deploying and running services and scheduled jobs in the cloud or data centres. Singularity combines long-running tasks and job scheduling functionality all in one package to support many of the processes necessary to deploy every day modern web applications and services.

4.3 Implementation

For the implementation of this task, it was split into multiple sub-tasks with each part being tested individually before the next part was created.

4.3.1 Standardized Labels and Fields Generator

The first task I worked on was creating a function that would take in some multicurrency and then convert it to a HubSpot standardized field.
For example, the function would take in a currency like *USD* and then return it in the form *hs_ external_price_ USD.*
Initially this change was simple to write as it required only the name of the currency that is being used to create the field according to the ISO 4217 standard.

As work progressed on this function, it was refactored slightly and given some more utility to be able to generate more fields that had to do with multicurrencies. A function to create the internal version of the fields was also created alongside the labels that would be displayed in the UI on the page in a users portal where multicurrencies would be managed.
This task took some time to get to production as there was a number of comments on the pull requests that needed to be implemented as it was my first major pr. This change showed me

how to write code according to the level that's expected at HubSpot.

Problems

The major problem that arose was that for some 3rd party developers didn't use the ISO 4217 standard for the currencies so matching them in the syncing engine wasn't possible.
To get around this issue, we first of all had to check if the inputted multicurrency was ISO 4217 compliant and then from there would generate the field. Any input data that was not compliant would then return an error.

Testing

In terms of testing for this change, I wrote a series of unit tests to test the different expected behaviours of this change. The tests were written in advance of the code being written as at HubSpot test-driven development is a big thing and is greatly encouraged.

4.3.2 Kafka Consumer

The next task I worked on was creating kafka consumer to consume messages from the kafka producer queue. This was a much larger task than the previous task so I split it up into smaller pieces to make the task more manageable. This aspect of the task would fall into the ”Controller” layer of the MVC framework as all data operations are performed here before being sent to the database or 'Model'

Building The Base Consumer

The base kafka consumer was constructed from an interface for building kafka consumers that was already pre-created. The consumer was subscribed to a topic that was responsible to publish messages concerning multicurrency updates from a producer. When a message gets published to the queue, the kafka consumer would then grab the message from the queue and then process it accordingly.

Filtering Kafka Messages

The next major part of the consumer I wrote was to grab the type of request in the message and from there, process the message accordingly. To process the message, I used a switch statement to filter the data based off the type in the message body. The currency being updated in question is then also extracted from the message body.

4.3.3 Pre-building The Request Body

Once the major aspects of the request has been collected from the message body, the request body is then created. The label's, standardized fields and such are all created here.

Sending the Request

Once all the previous steps have been completed, the request is then sent to the corresponding internal endpoint, using the previously built request body.

Problems

One of the major problems in creating this consumer was figuring out how to prevent messages being processed twice. Luckily, kafka has numerous guarantees that ensure that each message is processed exactly once making this a none issue. Another issue was figuring out how to scale this to work on a large scale. To solve this issue, kafka enables multiple kafka workers to be subscribed to a topic so more workers can be spun up as necessary or killed as traffic changes.

Testing

To test this change, a number of integration tests were written to make sure each of the core components interacted with each other as necessary. As each of the individual components do different things, unit testing the overall system wasn't possible. Each of the individual components were unit tested as necessary. One bug that was encountered in testing was getting the correct multicurrency field from the topic. To fix this issue, I had to re-parse the data given from the topic into ISO 4217 compliant fields.

4.3.4 Endpoints

The next major component that needed to be written was the internal endpoints that would hit the database and update the data in there. This would compromise part of the 'Model' aspect of the MVC framework as it's responsible for framework operations. The endpoints needed to be internal as we didn't want to expose them to external developers and also it allowed us to use the '@internal' annotation to allow it to be secure by using Oauth.

To create the endpoints, a path was first defined for both the add and delete endpoints in seperate functions. From here, we could then hit the database directly using some ORM methods. This component didn't take much time to create as it was a relatively easy aspect to code as it was basically a partial implementation of a REST api.

The endpoints that were created were

- An 'Add' endpoint to create a multicurrency field mapping in the database
- A 'Delete' endpoint to remove multicurrency field mappings in the database

Problems

One issue that I faced writing this was dealing with verification as I had initially planned on creating an open endpoint and hitting it as necessary. But I kept getting errors due to verification due to the fact I was trying to hit internal database's via external calls. To solve this, I made the endpoints internal which enabled stopped these errors from appearing as Oauth was used for verification automatically so I didn't need to worry about security.

Testing

To test this change, I did some integration testing instead of End-to-End testing as external users wouldn't have access to this endpoint so all that mattered was that it performed according to the parameters we expected. I did some asserts on the expected response from the endpoints, and mocked out the calls to the database as this was beyond the scope of the test.

4.3.5 Wrappers for Endpoints

The final part to complete this task was to write some wrappers around the endpoints defined above. Whilst the endpoints could be accessed without wrappers, to make the code cleaner, more maintainable and also more reusable, creating api wrappers was the best choice.

This code addition was also rather small, all that was necessary was to create functions that would take in the request from the kafka consumer and then send them to endpoint where they'd be dealt with. These functions acted as part of the 'Controller' aspect of the MVC architecture.

For testing and problems encountered, since this wasn't a huge change, no real problems were encountered whilst writing these wrappers.

4.4 Evaluation and Testing

Once all the parts of the consumer had been created, testing the entire flow of the system became top priority. Whilst each of the individual parts were unit tested and integration tested, an end-to-end test of the entire system could only be done when the entire thing was assembled.

To test the entire system, using Orion, a branch deploy was done of the code base including this consumer to the staging environment. Next, all active Kafka consumers subscribed to this topic in staging were temporarily paused to test this change to prevent a flood of errors being sent in case the change had some bugs. Finally, testing was done using a test portal. To begin testing, I started with testing adding new multicurrencies. To evaluate if the change worked, I searched the logs for https requests and checked if they were going to the correct services. Finally, I checked the database to see if the new fields were added to the test portal.

4.4.1 Deployment

According to the above flowchart, deploying new features is a fairly straightforward process. There's a few steps involved before code can be pushed to master:

- The code is first pushed to GitHub
- All tests are ran on the code in Blazar, the in-house testing automation software that runs all the tests and checks the code for code-style errors and such.

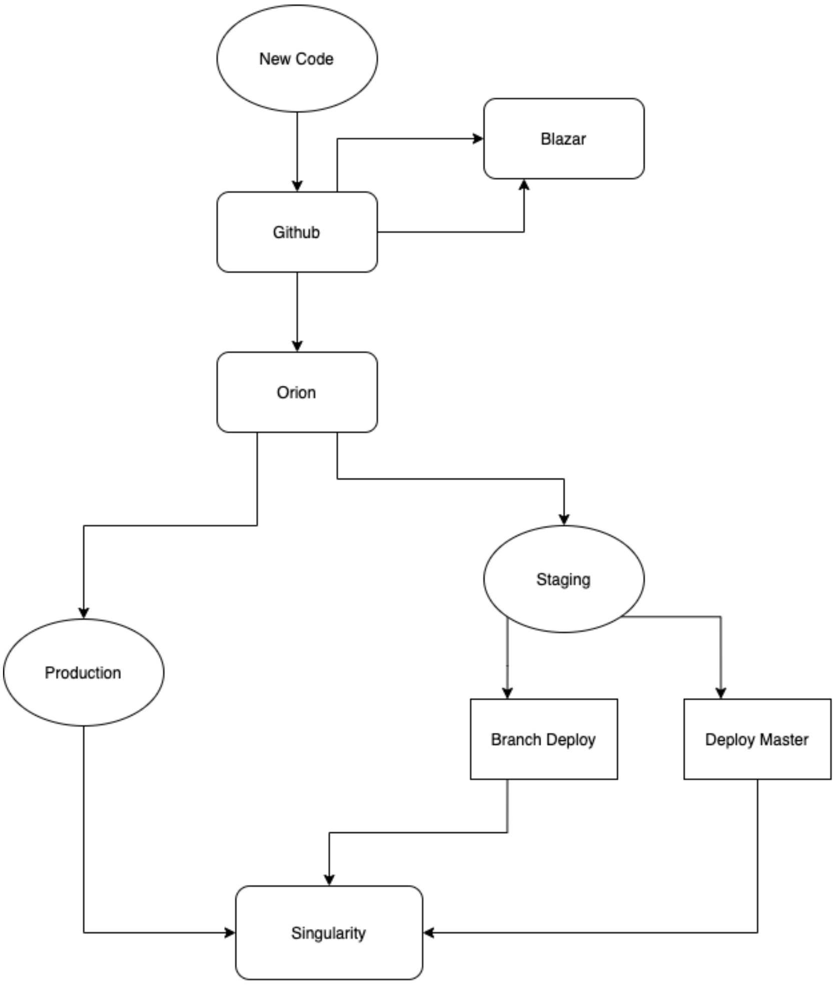

Figure 4.3: Flow Diagram of Architecture

- Following this, once all the test pass, the code can then be deployed via Orion, the deployment software we use to manage different deployments. Before code can be deployed to Production, it's usually first deployed to Staging to be monitored over the course of a few days to make sure no unknown behaviours occur. Specific branches of code can also be deployed to Staging, to be tested and such.

- Once everything looks good in Staging, the code is then pushed to Production. Depending on the change, we may gate how many people receive the change initially and monitor those portals and then as we feel more confident, roll-out the change to more and more people.

- Finally, Singularity is the software we use for running and scheduling tasks like the kafka consumer. From here we can create and kill tasks as necessary and there's also auto-scaling software in place. When any sort of code is deployed in Orion, Singularity is what creates the micro-services and actually runs the code and manages it over its lifetime.

4.4.2 Problems encountered

One issue I found while testing was I saw requests were not going to the correct services but were ending up somewhere completely different. I initally checked my code to debug it but I couldn't seem to figure out what was going wrong. Next I ran the kafka worker locally to find out what was wrong as I could debug it step by step as new requests came in, but again I couldn't figure out what was wrong. I finally asked my Team Lead for assistance, it turned out that because I was testing in Staging, the route requests take is slightly different as there are 2 streams in Production to deal with all the traffic while in Staging there's only 1 as it's just for testing. I was expecting the route of traffic to match Production but it turned out the testing was working as intended, my assumptions were just wrong.

4.5 Analysis and Possible Changes

Using the MVC framework was very useful in structuring how the code should be written and how to create tasks and assigning their importance. Using this framework also allowed for a high degree of scalability as each of the parts of the consumer work independently of each other which has the added benefit of easier maintenance as if any part breaks, there's no effect on the other parts of the code.

In terms of changes I would make if I could go back now, I would restructure how I created the endpoints. Whilst they worked as intended, the testing for them was somewhat fragile due to how I wrote the code for the endpoints. Malformed inputs could cause some errors or if the fields haven't been properly verified before being inputted errors could occur. While this isn't an issue at the present time as it's an internal API, if it were to ever become open to 3rd parties, it would need to be re-written to be more robust and do more error checking.

5 2: Hublets

5.1 Introduction

The second major task I worked on during my time at HubSpot was to write a task that would change the endpoints for all users who had Shopify connected to their portal. This change was part of a larger initiative at HubSpot to become GDPR compliant. To achieve this goal, an EU data centre was built alongside a whole new EU hublet being created (think of a hublet as a center of operations for all the products HubSpot offers). My team were responsible for getting our code base up to scratch in preparation for this so we were in 'Code Red' or maintenance mode. As a new member of the team, initially I was given the kafka consumer to focus on but I was asked to pivot to the hublet based work so we could move out of Code Red faster and move onto creating new features.

This change wasn't very difficult to write but there was somewhat of a learning curve as I had to learn how jobs are written in HubSpot and how to test them before they go out to production.

5.2 Design and Architecture

This job was classified as a 'Backfill Job'. What this meant is there was a standard set of operations that the job needed to perform . As this wasn't the first backfill job that my team had written, I could copy from another backfill job that was written and copy the overall architecture but edit the actual functionality.

A backfill job is composed of 2 main components:

- Re-writing all old data to suit the new format
- Changing the code so all future data is written according to the new format

The task was split accordingly to the 2 sections outlined above.

5.3 Implementation

5.3.1 Job Framework

To create the actual job framework, there's a standardized set of methods necessary. The main functions are

- Dry Run Option
- Argument Captors
- Run Option

When the job is first run, depending on the flags passed to the job, it either runs in dry run mode or a straight run through the job. A dry run is essentially a test of the job. The argument captors are there for any additional arguments that could be passed such as if there's a subset of portals you wish to test on.

5.3.2 Technologies Used

Blazar (4)

Blazar is a continuous integration system used at HubSpot to build recently pushed code from Github. It automatically picks up new builds and runs all the tests it finds in the code base alongside checking the code for styling and linting also. One of the great things about Blazar is it just runs whatever new code was pushed, it doesn't re-build an entire repository if it's not necessary.

5.3.3 Rewriting Old Data

To rewrite the old data, I wrote a filter to filter through all portals that had Shopify installed. From there I grabbed all the users webhooks by doing a get request to the database. Once I had grabbed a users webhooks, I replaced them with a new url that would hit a load balancer that would then in turn direct the traffic to the right webhook.

Testing and Problems

To test this change, I wrote a unit test with some dummy portals and tested to see if the webhooks on the dummy portals with Shopify installed had their webhooks updated or not. It worked as intended so I moved on to the next section

No real problems were encountered in this section as it was fairly straightforward to implement as I had an example job that did something similar in terms of database access so I had a good place to start from.

5.3.4 Refactoring Code for new format

The other section of this job was to refactor where webhooks are written so all new webhooks that are created have their webhooks pointing towards the load balancer. This change was very small to implement, all it required was an override in Orion for the webhooks base url as it was a config value that was configured when the service was first set up.

Testing and Changes

Testing this was fairly straightforward, all that was required was to create a new HubSpot user and install the Shopify app to their account. From there an assert on the database was done to see if the webhooks urls matched the new urls.

5.4 Evaluation and Testing

Once the job was completed, some test of the job were done locally using the dry run option. No weird behaviour was observed here so the code was then moved into Staging for further testing. Here we ran the job in it's entirety against 1 test portal to see if request from Shopify were getting through successfully. This seemed to work, then from there we shipped the change to Staging entirely.We monitored all portals in Staging but no real problems seemed to be occuring.

We then shipped the change to Production and ran the job against all portals in HubSpot. This took about 12 hours to complete. The day after the change had gone out, we noticed an up-tick in the number of 400's being returned from Shopify. Then a flood of tickets from customers came in complaining about Shopify data not syncing correctly. After spending about 30 minutes debugging with my team, we found out the change to the webhooks was causing the errors. There was a trailing space in the webhook URL that was causing them to fail somehow that the load balancer couldn't deal with. To quickly fix the webhooks for all the customers, another job was run that reset all the webhooks for customers to their previous state.

The reason this bug wasn't found in testing was because it seemed that we were getting some sort of false positive while testing which made it look like things were working fine but actually weren't. It took us about a week after this to recover all the customers lost data and change the webhooks to point at the load balancer again following this bug.

5.5 Analysis and Possible Changes

If I could change anything about how I approached this task, I would test the code much more vigorously. The trailing space was something that would be rather difficult to spot and even harder to diagnose if not for the expertise of my team mates. I would test on a larger subset of portals in Staging to avoid getting a false positive like this again.
I would also gate the roll-out of this change so that only a small subset of portals would receive this change to reduce the size of the impact in-case there was a bug in the code that made it to production. That way if something goes wrong only a few people are affected and this the clean up after the job is much easier and requires much less time to fix.

6 Conclusion

6.1 Final Thoughts

Overall, I had a very positive experience during my time at HubSpot. This internship was really useful and it taught me a bunch of skills and given me opportunities I would never have gotten during a college semester or in a college semester.

I was lucky to be placed alongside a group of really intelligent and experienced engineers that I could learn a lot from. The new tools and technology that I encountered during my time at HubSpot, whilst some are in-house, are invaluable and I think they'll be very useful to have when anywhere in the industry. The internship allowed me to gain plenty of experience in numerous aspects of the software engineering process, from the different areas of agile methodology to using continuous integration software.

In terms of soft skills, I learnt a lot about how to work in a professional capacity from working alongside my team mates and how to collaborate and interact with people. My team didn't treat me as 'just an intern' but instead treated me as a full time member of the team and encouraged me to engage in conversations about how the team should move forward. The weekly meetings with my team lead really helped me to grow and develop as an engineer and helped me grow my confidence overall.

The actual project work that I worked on during my time at HubSpot was very interesting and I learnt a lot about how code is written in a professional environment and how important it is to write clean, maintainable code. While the entire project couldn't be completed during my time there, I'm happy to know my addition to the team is something that will affect customers positively and will have a lasting impact long after I have left.

Bibliography

[1] Model-view-controller image. https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66, July 2021.

[2] Kafka documentation. https://kafka.apache.org/documentation/#intro_guarantees, July 2021.

[3] Singularity documentation. https://github.com/HubSpot/Singularity/blob/master/Docs/about/how-it-works.md, July 2021.

[4] Blazar documentation. https://github.com/HubSpot/Blazar-Archive, July 2021.