IBL Student Report – Saral Gautam 4B

The Individual Role

At Origin my main role was to support my team in their reporting-based tasks, though I broadened my scope of work over the course of the placement. The technical name of my role was a 'Reporting Analyst' and my main reporting-based duties involved two major reports at Origin - the Net Position Report and the Origin Active Number (OAN) Report.

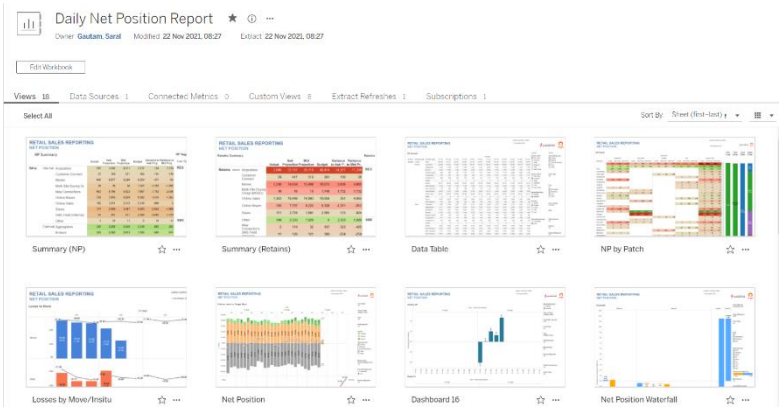

Figure 1: The Net Position report. One of the main reports I worked with during the placement. At its most fundamental level it showed the number of customers won subtracted from the amount lost.

These two reports were perhaps the most direct measures of Origins business performance, the Net Position report showing the number of customers won and lost over the financial year and the OAN showing the total number of accounts Origin is providing a service for. Within the Net Position report, the total sales of any of Origins products such as electricity and gas were visible. Additionally, in OAN one could see the overall status of any Origin account. This included their billing status, the consumption status (some premises were consuming electricity from Origin but did not have an Origin account) and scale of operations.

Since these reports could be used to assess Origin's performance in retail and in external audits from other companies (usually consultancies) their correctness was critical. Furthermore, OAN and Net Position acted to verify the success of different business plans- if your team was doing well then it should show in one of the two reports.

As such my main task was to refresh these reports daily and investigate the numbers within them. If there were any sharp increases or decreases, I would report this to the team, investigate further with SQL queries to Jindabyne data tables or look at the report itself with Tableau. Some days were almost entirely spent running SQL queries to analyse OAN movement or in explaining any drastic changes in Net position. The classification of customers within these two reports depended on 'business rules' which were essentially case statements in SQL. When the energy market changed in any form like with the 5MS (five minute settlement) implementation on 1st October 2021, the team would have to change the SQL code for the tables accordingly or the tableau reports themselves to ensure the reports correctness. With the 5MS implementation for example, this affected when losses were recorded in Net Position which affected the end-of-month projections, so we had to change the way we calculated the projections to ensure accuracy. Moreover, Origin could also change its internal workings which would also lead to code changes in OAN and Net Position. This was highlighted in the current change from the SAP billing system to Kraken, a company based in England. All the underlying Jindabyne tables are being remade so they could again be used in reporting. In the process of migrating customers to Kraken from SAP, there could be some 'missing' customers which we could frequently have to investigate for OAN, as its business rules did not factor for migrating customers. All these changes would need to be explained on a weekly basis in key meetings that my team lead, Eva, had so that the information could support business decisions such as directing more resources to an underperforming sector.

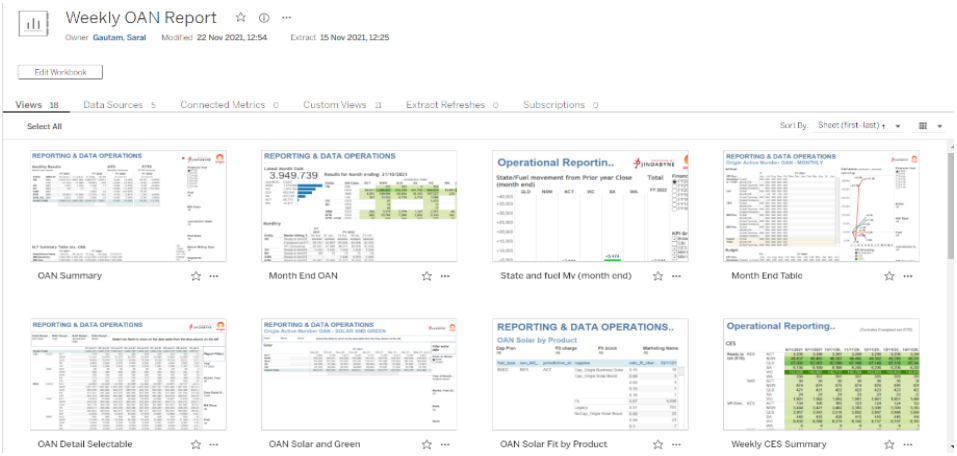

Figure 2: The OAN (Origin Active Number) report. One could see the details of any registered Origin account in this report.

Thus, there was an aspect of constant maintenance to the OAN and Net Position reports. Furthermore, I also worked on contributing to some Tableau reports creating views for stakeholders or adjusting their frequently used reports so they could find specific information.

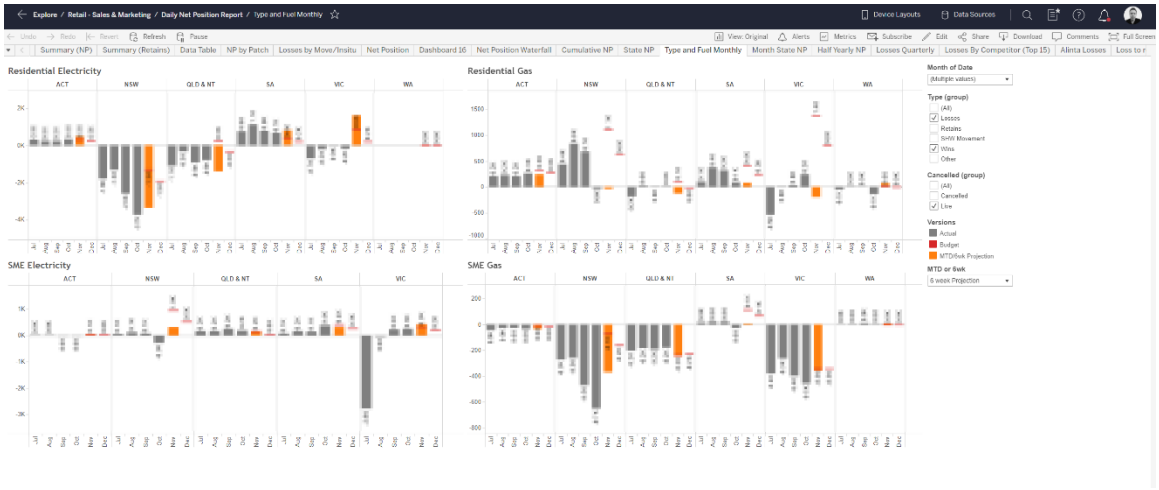

Figure 3: A view I made in Tableau, showing how residential and SME (small to medium sized enterprise) customers were performing against budget.

Though my role involved reporting based tasks, most of my placement was spent in software development. This happened because one of my colleagues, Kai, was looking to create an application that could directly input data into a Jindabyne table for Origin Scorecard. The Origin Scorecard was an important report because it was used in one of Origin's board reports, but it was managed with a spreadsheet. The whole workflow for gathering data for the metrics in the report was highly inefficient. Kai would generally have to email executives or managers for the results of metrics that Origin Scorecard contained, and their gathering really depended on whether they would respond to his emails. Furthermore, these people would have deadlines for inputting this data and could sometimes need to input data after work hours, where Kai would be unavailable. Thus, he envisaged an application where the executive and managers could directly come to input a result and it would then update into the Origin Scorecard Report instantaneously. When he posited this idea, he noted that another Origin employee, Issac Ekbote, had already created a similar application, but it was seldom used because of its grid-like user interface. Taking Issac's work as a proof of concept, I began working on developing such an application and within a month's time, Kai and I put a production version of the application into his Origin Scorecard Report. We called the application Scorecard for short.

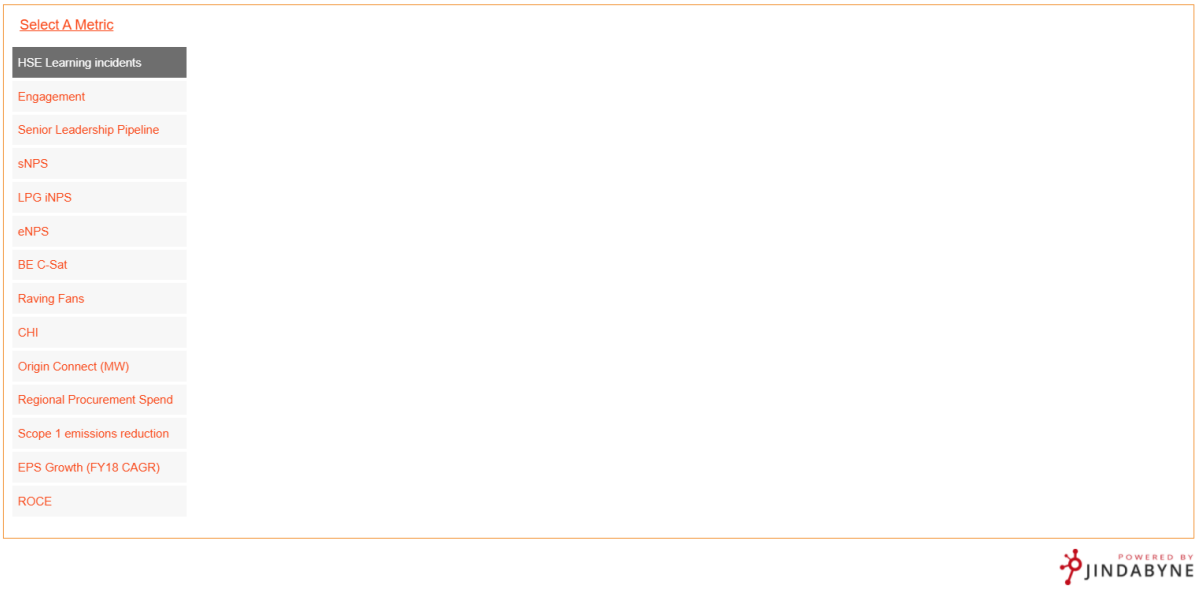

Figure 4: The landing page of the Scorecard application. A user could select a metric and then input a result.

With Scorecard we were able to create a direct link between any user and Jindabyne, which once their data was in Jindabyne, meant that it could be used for all the reporting capabilities that Origin had to offer. Previously such a link only came through an intermediary such as an analyst like Kai or myself or from an overnight batch job. Establishing this link to Jindabyne allowed us to broaden the spectrum of data capture with Origin-wide reports, which was pivotal in my legacy project.

Legacy Project Outline

Building on from Scorecard came the Sustainability Data Capture (SDC) which was my legacy project at Origin. I will describe this project in further detail in later sections, but for the purposes of reflecting upon the tasks I performed, most of the key lessons I learnt came from or were emphasised within the SDC build. In brief, the SDC was a web application that allowed users to input and review results for business metrics that would go on to be used in the Origin Sustainability Report. In creating the SDC, my team were given only a ‘requirements’ document which contained all the functionality the SDC was expected to have; other than that, we were on our own.

The Three Key Lessons

Through the reporting and software development roles at Origin, there were many skills I built upon. These skills are best encapsulated within three main lessons that cover the most memorable learning experiences I had at Origin, beginning with reading the documentation.

Read the documentation

In the SDC build, one expected requirement was that it could validate a user without them having to fill in any kind of login prompt, so I had to be able to retrieve the Origin username of anyone entering the application. Though I had already done this in Scorecard, SDC was different because it was expected work outside of Tableau (Tableau allowed us to send the username without any kind of code implementations). When confronted with a specific coding problem such as this, my workflow at this point was to look for commands on google or stack overflow posts and then go through a process of trial and failure until I found a command that worked. This was inefficient for two main reasons:

1) It was time-consuming.
2) There is no progression to a solution.

This trial and failure process was exactly what I did when confronted with the username retrieval problem. I had set a day to implement this functionality; four hours had passed, and I was not any closer to a solution then when I began. I eventually asked Issac, a colleague well versed in such problems, for support and what he said resonated with me quite deeply cause of its simplicity:

I asked, “Issac do you know any way I can retrieve the username?”.
He replied, “It is already there.”

He elaborated saying that the solution was in the documentation - the one place I had not looked. I then implemented the command he showed, and I was able to retrieve the username. This experience was important because a couple days later, I reached a similar point in the SDC build where I required a command that could send data to and from the front-end of the SDC. This part of the application used Ajax, a JavaScript library, and this time the documentation was the first place I looked for the command. Shortly after I found the ‘data’ method of Ajax, which worked instantaneously in sending the data to and from the front-end. I had taken the

time to read the documentation line by line to progressively arrive at the ‘data’ method. Since documentation for modules are written in this style where methods are first general and then become more specific, each method I read brought me closer to the solution- the complete opposite of the trial and failure I was following previously.

Targeted Questions

While working at Origin, particularly in reporting-based tasks it was very common that a stakeholder would ask for a view or some kind of functionality but did not how it should be delivered. Many times, I would hear questions in meetings from users similar to:

“Could you make me a view that gives the number of wins?”
“Can we have a pipeline view for OAN?”
“Could we add a flag for Kraken customers?”
“Can we filter by date?”

In none of these questions do the users explain any kind of delivery process. This leaves a developer or analyst in a position where they have to meet the user’s expectations of a product without a clear method of doing so. Thus, how should one direct their work efforts? One could produce a body of work, and have it be different to what the user expected though it may still meet their requirements. If we call the user’s expectation ‘Point B’ and the starting position prior to working, ‘Point A’, this workflow can be represented by the figure below.

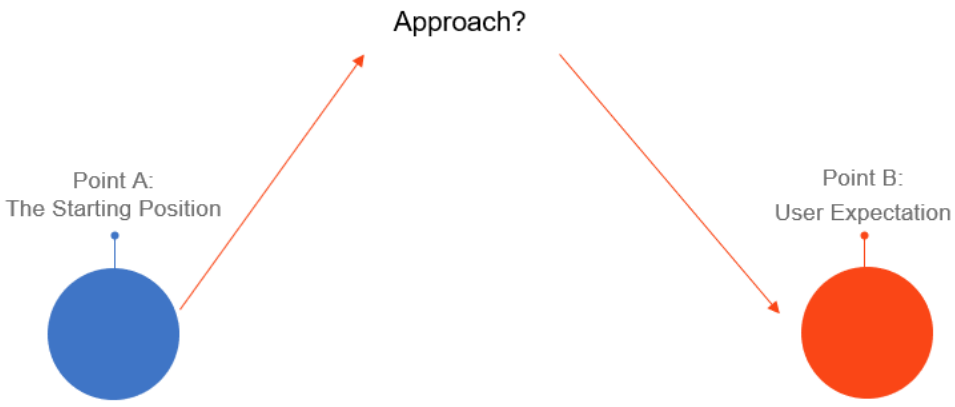

Figure 4: Following an approach that may not meet the users expectation. Then one may have to make significant adjustments to move back to Point B.

This style of work has a huge risk associated with it; the analyst or developer may produce something that is completely against user expectation. Instead, what I found my team did was slightly different. They would use *targeted questions* to guide their development process. By *targeted questions* I mean very specific questions on functionality, which allowed one to gauge the user's expectation. I noticed this initially in their reporting tasks. When they completed a feature, they would contact the end-user and ask if the feature fit their requirements. Then they would continue their development and continuously maintain contact with the end-user as they developed any major body of work. This created a workflow more like the one shown below:

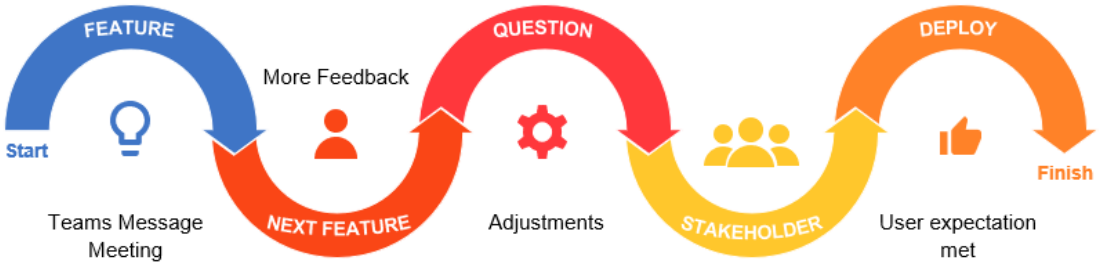

Figure 5: The reporting teams' workflow for development. Targeted questions guided the path to deployment.

The workflow shown in Figure 5 was what guided the SDC development process. Any time I developed a key feature on the build I would inform Kai and he would suggest any changes should they be required. I also had weekly meetings with Mary, the project manager, where she would also contribute with her own feedback. Additionally, Mary staged meetings with key stakeholders of the SDC - another opportunity to steer ourselves towards user expectation and even beyond. Thus, what I found was that using targeted questions allowed one to navigate to Point B (user expectation) from Point A (the starting position) without being given a path from any external source.

Structured code

Once feedback is received on a developed feature, one then implements the suggested adjustments, but how can this be done quickly? Prior to my placement, I had heard statements similar to:

"Structure your code."
"Create a function so you can reuse the code."
"Structuring helps you debug."

All statements I ignored with the Scorecard build. In Scorecard the HTML code was poorly structured. There were many repeated components and inconsistently styled elements leading to unnecessarily lengthy code (scrolling up and down on an editor can be time-consuming and even confusing). Making any changes to the HTML code required manually adjusting all the files relevant to a suggested change.

When I worked on the SDC though I changed this practice. Adopting coding practices suggested to me by a colleague, I began putting repeated code into functions and the repeated HTML styling elements into separate files. This may seem like a small change, but it had a drastic impact on my work output for sustainability. Now when a user suggested a change, I would only have to change that function that was relevant. I reached a point where sometimes I could adjust features during a meeting as the feedback was given and I myself, despite being the developer, was amazed at how dynamic the application became with good code structuring.

Soft Skills Inherent

Though the above experiences were mostly from a technical background there was a soft skill embedded within all of them that I came to develop over the course of placement. A key stage in *reading the documentation* was being able to reach out to a colleague for assistance as I did with Issac. This was important because I was able to build a network of colleagues that could support me when troubleshooting and similarly, I could also help them. *Targeted questions* involved including the end-user through the development process using empathy to target the questions and gauge the end-user's product expectation. This was particularly important – putting yourself in the user's shoes made it much easier to direct feature development within the SDC. Simple front-end features such as adding an 'Approve All' button on the approval

page and the file upload functionality came from imagining ourselves as the end-user. Furthermore, *code structuring* was about being able to receive and act upon feedback, though the feedback could be given in a stern and forthright manner. Being able to acknowledge its importance in the context of the project and then implementing the feedback was a key soft skill I learnt through the medium of structuring my code. Through these experiences what I ultimately found was that soft skills allowed me to direct my technical focus. Targeted questions, acting upon feedback and being able to reach out to colleagues were all initiatives that gave insight on what I should be working on at any given point in time. This level of awareness in work I found, made me a much more complete IT professional - one who knew where technical capabilities should be directed within the project's context.

Key Acknowledgment: Never stay still

Through the SDC build, I faced many technical challenges. The most difficult dealt with linking the front and back ends of the application with sound coding logic and familiarising myself with new coding language modules (like Ajax as I had previously mentioned). Despite all this, going from one technical challenge to the next, I felt a sense of momentum as I neared completion on the SDC. Then I encountered a challenge, previously unknown to me.

At one point in the SDC build, my Origin account did not have the necessary permissions to query a Jindabyne table, so I had to wait for access to a general service account. I was at a point where I could not continue working on the application which was highly frustrating because:

1) My momentum had come to a complete stop.
2) I had no control over when I would get the service account.

Since I could not work on the application, I did other related work. I knew that using Git and Bitbucket (online coding repositories) were skills I lacked so I worked on them over the week. It took a couple days for me to first get access to BitBucket and then familiarise myself with the commands. A week passed before I got the service account, and I then began working normally on the SDC; this time using Bitbucket to push code changes to a repository I made. Now all this might seem unrelated, but in the later stages of the SDC build, Mary was pushing for completion much sooner than I had expected. One day in late November she asked for three changes in a single meeting all regarding front end styling. One of the changes in particular

was adding the tableau logo to the home page - something which I had previously done and removed from my current version of the SDC.

However, because I was using Git and Bitbucket I had a commit (version of the SDC) on Bitbucket that had the tableau logo on the home page. Thus, I was simply able to go to that commit and copy-paste the code relevant to the tableau logo. Implement this feature may have taken another day or so but because I had spent the week where I was waiting for the service account learning Git and had pushed this feature onto Bitbucket, I was able implement Mary's suggested change instantaneously. Thus, the otherwise idle week became one of the primary reasons I was able to complete the sustainability build. I had learned that in a project setting, there was always relevant work to do which to me is surmised in the title of this paragraph- *never stay still*.