**IBL Student Report – Saral Gautam 4C**

**In Depth: The Sustainability Data Capture (SDC)**

My legacy project at Origin was the Sustainability Data Capture (SDC) which was a web application that allowed users to directly input and review results for business metrics. The application had numerous metrics (roughly five-hundred). to account for which would then be drafted and used in the Origin Sustainability Report (OSR).

The OSR is a document published annually by Origin that gives an overview of all the initiatives taking place at Origin. It also details Origin’s performance in different business areas, which in the documents are categorised as ‘Customers’, ‘Communities’, ‘Planet’, ‘People and Culture’ and ‘Our reporting’. Though the OSR is not a legally required document, its publication serves two main purposes:

1. Anyone can use it to find out more about Origin as a business. This can include any potential investors or any government investigation, and
2. It acts as a certificate of operation, which shows that Origin is aware of it social responsibilities by reporting on its carbon footprint and contribution to local communities.

With the OSR being of such high importance the correctness of its reported values is crucial. Prior to the SDC the way these results were managed were a single spreadsheet which was sent back and forth to different data contributors. This had numerous issues some of which included:

1. Anyone could adjust the spreadsheet, so there were virtually no security measures.
2. There was no traceability- you could not see who inputted the results for metrics.
3. Each result would manually be entered into the spreadsheet.
4. Data contributors could leave Origin leaving empty results (this was actually a big issue).

The main objective of the SDC was to fix all of these issues and yet have a dynamic web application that could be easily adjusted should new features require implementation.

**Technical Summary: How it Works**

Like Scorecard the SDC uses a python module called Flask. Flask is a web development framework that works by combining python code with HTML to create web applications. Though Flask may sound quite technical, once its key concepts are understood it becomes much easier to work with. When an application is made it can be hosted on a URL. We did this with one of Origin’s in-house urls, so that when a user comes to the URL an instance of the application is launched. Within Flask you can specify specific url paths or ‘routes’:



Figure 1: The url route ‘/’. The route could be customised for different URL paths.

So that when a user comes to this ‘route’, the code within that route is launched where normally a HTML page is displayed at the end. That’s it. How does the user come to this route? On the HTML pages there will normally be forms that allow the user to interact with the web page, which redirect the application to different routes. In HTML forms begin with tags such as:



Figure 2: A HTML form tag used in the SDC.

The most important part is the ‘action’. This is saying “if the user submits a form, then go to the url ‘/index’”. Once we go to ‘/index’ whatever code is within this route will be launched. This process is repeated for any route, thus creating a web application that acts over different url paths.

**My role in the SDC**

In the Sustainability build I was the sole developer. I created the whole front-end of the application, with feedback mostly from Kai and Mary. With the back end, I built upon what Issac had previously built though it required significant adjustment to implement the search functionality, the forms and the routes. The only pieces of his build I kept were the user validation and the models for the databases. Issac also helped me significantly when it came to the JavaScript code for the search functionality and some of the other front-end features that required jQuery (a module of JavaScript) to make the application more interactive.

**Application Overview**

*Home Page*

Graphical user interface, text, application, email

Description automatically generatedThe home page had some key administrative information displayed. There were links to requesting access to Tableau, sending emails to the Operational reporting team and a guide to the csv file upload format.

Figure 3: The home page of the SDC.

*Add/Update Result*

This page was the main screen of the application. Here a user could search for a metric or upload a csv file for a bulk upload of metric results which removed the need for manually inputting multiple results (Figure 4).The search here was an important feature. Depending on what the user inputted in any field the rest of the fields would filter. So, if the user inputted ‘Person A’ for the data steward, then the metrics that had ‘Person A’ as a data steward would be the options in ‘Metric ID’. Once the user searched for a metric the page would show the metric’s metadata (on the left), history for that metric and the user could also input a new result (Figure 5). If the user did input a new result, it’s ‘Status’ would display as ‘Pending’ as it had not yet been approved and this record would be inserted into a Jindabyne table. Thus, we were able to keep an audit trail of any inputted results along with which user input the result, which was previously unknown in the spreadsheet method.

Graphical user interface, application

Description automatically generatedFigure 4: The add and update page of the SDC.

Graphical user interface

Description automatically generated

Figure 5: The add and update page of the SDC after a user searched a metric. The user could input a new result and see the result history for a metric

*Approve Metric*

Like the Add/Update result page the ‘Approve Metric’ page also had the same search functionality. Depending on what the user inputted into the search the relevant metric results would be displayed along with their status. The user could also filter the search on the ‘Status’ which could be either ‘Pending’, ‘Approved’ or ‘Review’. Once searched different metric results would display and the user could either click the tick for ‘approve’ or the question mark for ‘review’ (Figure 7). This workflow acted to simplify to approval of metric results. There was also the option to ‘Approve All’, should the user want to bulk approve results.

Graphical user interface, application

Description automatically generatedFigure 6: The approve metric landing page.

Graphical user interface, table

Description automatically generated

Figure 7: The approve metric page after the user searched.

**Stakeholder Analysis**

From creating the search functionality in the SDC, I could see some of the names that could be used to search for metrics. The names I saw were truly Origin-wide including executive managers, group managers, team managers and analysts. The SDC makes it easier for these stakeholders to input results and verify their correctness. The metric results are initially input by data contributors, then approved by group managers before their final sign of by executives and general managers, following a progressive path through the company. The application also makes it easier for the Sustainability team who actually create the Sustainability report, since they can begin drafting their report once the results have been entered. Furthermore, the results from the report can be put into a spreadsheet, into any required format so that the Sustainability team can send it to Tangelo (an annual report producing software company) who further support the drafting of the Sustainability report. The results themselves can also be audited by consultancies, usually one of the big four and the connectivity of the input tables in the SDC the Jindabyne further support the auditability as the inputted date and time along with the approved date and time are also recorded within Jindabyne.

**Current Status**

At the time of writing the SDC is in production, and it is planned for use in early January 2022. The main point of improvement for the build is to add functionality for calculated metrics. Many of the metrics used in the OSR are calculated from other metric results – the main aim is to be able to input the most basic metric results and the calculated ones could automatically be adjusted for their status and result values. This can only be done once the metric metadata has been verified. Since there are many metrics and only a handful of people know this metadata, this manual process of transferring the metadata to Jindabyne may take a month or so.

Along with the data adjustments, the application may also have to undergo a security review by the Origin security team to ensure its complying with their regulatory standards. If such a review is conducted, then the application may require additional security features such as single sign on (SSO) or login credentials.

**University Learning**

In the computer science degree Monash offers, all of the core problem-solving based units are taught entirely in python (FIT1045, FIT1008, FIT2004 and FIT3155). Since each unit goes for 3 months, I had virtually a year’s worth of python coding experience before my placement. Additionally, classes are taught in depth in FIT1008 which are extremely useful, in industry-based settings. Thus, in terms of just coding experience alone, I think university learning is quite comprehensive. Furthermore, with such a deep knowledge of python it becomes easier to learn other programming languages – more a matter of learning syntax and commands rather than learning key programming concepts such as classes and functions.

Aside from the core python units, I also did a unit called FIT2102 which taught JavaScript and functional programming. This unit was incredibly useful because it teaches how to write clean code, which was especially important in the SDC build. When I applied the programming principles, taught here in the SDC, I was able to drastically reduce the amount of code I wrote which overall simplified the application.

From a database perspective, FIT3171 was very important in my placement. FIT3171 covers database concepts and SQL which were the foundation of my team’s work. Furthermore, the unit also covers database concepts like entity relationship (ER) diagrams, relationship types and sequences which were all topics Origin actually applied in its Reporting and Data Operations teams. It would not be uncommon to see an ER diagram in a meeting for example. If hypothetically, I had not covered this unit, I think I would not have been much use to my team even over the 5-month IBL period - it was that important.

In terms of what Monash could improve on, I think in all of the introductory first year IT units there should be at least two to three weeks covering Git alone and in later units maybe virtual environments. Git and the usage of virtual environments are industry standard tools, especially Git which in itself is like a programming language. Its commands could be easily interwoven into the course content to make it’s teaching seamless (for example, there should be at least one git revert command in your repository before submission).

**Final Words**

All through my IBL placement, there were many different experiences as both a reporting analyst role and a software developer. The key experiences that I had, taught me practices that I could use to become a better IT professional in both technically and interpersonally. My legacy project will hopefully be used far beyond my placement and the link that Scorecard and the SDC have created between the user and Jindabyne will hopefully support the development of many more applications at Origin. Additionally, the units I had enrolled in at university greatly supported my transition to work at Origin and the skills I have developed, I will be sure to carry far into my career.