DART

Final Report

Higher Diploma in Science in Computer Science

Matthew Hornby | 20058053

Table of Contents

Glossary

Term	Definition
Compiler	A program that takes a set of instructions in some form, parses out operations and translates them into a machine-level or a lower-level language.
DART	The name of the project, it stands for Document Automated Regression Testing.
IDE	Integrated Development Environment, this refers to the tool in which code is written. It normally allows for easy compilation of code or extra features in terms of debugging among other things.
Intellij	IntelliJ is the IDE created by Jetbrains and is usually the preferred development environment for any Java based products.
JSON	JSON is a human and machine readable data transfer standard. It groups objects into a collection of key, value pairs and is virtually supported by all languages. It is normally encountered when communicating with web applications.
Linked List	A data structure for storing references to elements of a list. Each element contains a reference to the next element. This allows the element list to be traversed forwards and reversed backwards.
Node	Node is an open source, cross-platform runtime for executing javascript.
NPM	Node Package Manager, is the tool used for managing the installation, updating and removal of external libraries for Node.
Sunlife	The company I work for, they supply insurance products to other companies – mainly for their employees. They are also involved in other financial products.
Syntax	The individual elements that make up a programming language. For example, the way an IF statement is written in python could be said to be a part of pythons syntax.
UI	User Interface, describes the mechanism through which a user can interact with a screen displaying an application.
UX	The users experience of using an application.
Unit Test	The process of testing small subsets of a programme in isolation to ensure features are working correctly
VScode	Visual Studio Code is a lightweight version of Microsofts main IDE Visual Studio. Visual Studio Code is more akin to a text editor, but is highly extensible with user created plugins. This makes it a very flexible IDE because functionality can be tailored toward the user.
XML	Markup language commonly used to define data. XML acts as both a human readable and machine readable conduit through data can be communicated between applications.
Xpression	A third-party proprietary set of tools licensed by Open Text. These tools are focused on allowing developers to create services relating to document output.

Stateful UI	The practice of have state changes in UI be reflected when they happen, instead of being reflected after a web page has refreshed.

Figures

Declaration

Introduction

Background
DART[1] is a set of tools that ultimately comprise a testing toolkit, related to the domain of my employment as an application developer. For context, I am currently employed on a team that manages and uses a low code tool to output pdf documents that feed into a larger application platform centred on insurance provision.

Focus & Scope
Whilst I have ended up creating many individual pieces that ultimately make up the entirety of the DART project, I have chosen to focus on a subset of what I have built, as to attempt to showcase and discuss all of the individual parts would be outside the scope of this assignment.

Some of the parts also contain more direct code and references to the company that I work for and attempting to perform redactions would leave me sidestepping important functionality of these pieces of technology, so I have chosen instead to leave them out altogether. Luckily, the two pieces of technology I am choosing to focus on for this project are the most interesting and the most impactful to the overall functionality of the programme.

Xpression
My project is linked to a third party application that has been licensed for internal use within my workplace. A brief background on this tool is helpful for providing context through which my project can be understood.

The tool is called Xpression[2]. It is a low code tool used to build various rulesets and conditions for text and image based content to play within a pdf document - similar to a templating engine. The main two things that should be understood about this tool are:

- The actual rules for each document type are housed in an XML format
- Customer Data, in the form of XML is fed to these rules – and this is what ultimately produces a variable pdf document.

Internal Tooling
The main justification for internal tooling is invariably the same across the tech industry – good internal tools increase productivity. However, as they are not customer facing applications – internal tools can suffer from some compromises when it comes to UI, functionality and support, meaning that in most software companies, there will always be room for expansion or development of internal tooling.

[1] Document Automated Regression Testing

[2] OpenText. (2024). xPression Enterprise Edition. Available from: https://www.opentext.co.uk/products-and-solutions/products/customer-experience-management/customer-communications-management/opentext-xpression/xpression-enterprise-edition [accessed 07 February 2024]

It can be difficult to evaluate the positive impact of an internal tool. In an online article I have read recently, the key points on what the benefits of an internal tool should be analysed against are summarised as follows[3]:

> *"The exact impacts of any software project are innumerable and complex, so it's important to distil it to a few key questions:*
>
> *What does this tool enable?*
>
> *How many people (internally) are impacted?*
>
> *What are the gains that result from delivering this tool?"*

We will keep these three key points in mind and use them as criteria for DART to have surmounted in our conclusion in order to justify it has having been a worthwhile project.

Goals

DART was aimed at creating a testing platform that allows developers and business stakeholders to interact with the document service provided by my team. The overall goal of this project was to "open up" the document service and allow individuals within my team, and outside my team to access the functionality of Xpression without using the actual application itself.

On a more technical level, the main goal in my implementation of DART was to be able to have an application understand the XML rules that were run via the rules engine in the Xpression application, this would provide a foundation upon which unit testing and other forms of automation functionality could be built.

Problem Definition

Whilst my project was looking to provide functionality for being able to generate documents and perform comparisons on documents, I believe these to fall more in the category of *extra value additions*, and that the main problem that I was trying to solve was related to an issue of Runtime.

I find it difficult to readily describe this problem, as it is not so much a *problem* but more so an absence of functionality. It's not something that was thought was possible prior to this project, but was something that I thought would add a lot of value if it could be implemented. In order to detail this problem in a tangible way I will describe a real scenario that pops up due to its absence.

Revision Units

As Part of the application, there exists the possibility to mark individual sections of content with an identifiable name. These names can Strings, but they can also be a combination of a String and some other variable. If you imagine a scenario where a section is looped an arbitrary number of times, you can see that you might end up with section names like:

- Section1

[3] Jimmy Shi. (2021). Internal Tools: A Cost Benefit Analysis. Available from: https://www.internal.io/blog/internal-tools-cost-benefit-analysis [accessed 07 February 2024]

- Section2
- Section3

A consequence of some internal workings means that if these Section names were to be duplicates, the document would crash on generation and fail to be delivered to the service requesting it. This has not been an uncommon problem and will usually arise just after a release as a result of a developer accidentally naming a section with the same name as another, or as a result of a loop where some variable part of the name is duplicated. These problems will get picked up and further code migrations are required to resolve the issue.

We are unable to run automated tests to pick up duplicate Section Units that are the result of loops, because we have no access to the runtime of the application. Obviously it is possible to take the data of an individual scenario and run it within the application to see if it produces a crash or not, but this is not an adequate solution to the problem as it cannot be replicated over a large number of documents.

In order to solve this and many more problems, I was looking to build a way to compile the rules of a document outside of the Xpression application.

Requirements
Given the above, I had settled on DART requiring a compiler for the rules engine of Xpression, in so much that given the same XML rules & the same XML data, the compiler could produce the same output as the Xpression application.

I also believed that DART was required to swallow any of the functionality of the other internal tools that existed within the company for the document service. My thinking behind this was that, if DART could provide the features which developers had come to expect from the other internal tools, then that this would encourage developers to use DART over these tools and help expose them to the more complicated elements that having the compiler would introduce.

Regardless of the developers, the document generation and comparison features were to be made available to business stakeholders, so I believed the UI had to be simple and easy-to-use. With these points in mind we can come to the following list of requirements:

Requirements
Generate PDF Documents
Compare PDF Documents
Create Difference Reports
Monitor Document Generation
Compiler / Parser for Rules Engine
Have a Nice Interface

Figure 1.0 - Requirements

Using these requirements as a base, I then moved to comparing how DART would stack up against the other tools on offer within the company.

Research and Analysis

Market Analysis

In preparing to undertake this project and to further refine the exact feature set of the project, I was able to analyse and use other internal work-based tools that performed much of the same functionality as DART. There are two main tools in circulation that provide some of the same functionality that DART does.

Company Tool 1

This tool is available on a company repository, but I can only view compiled exe's and .dll files. I originally thought the tool was a Java based application, but the presence of the .dll files has left me suspecting that it may be C based. All in all, I am not fully sure which language it was developed in and the source code is not available for me to view.

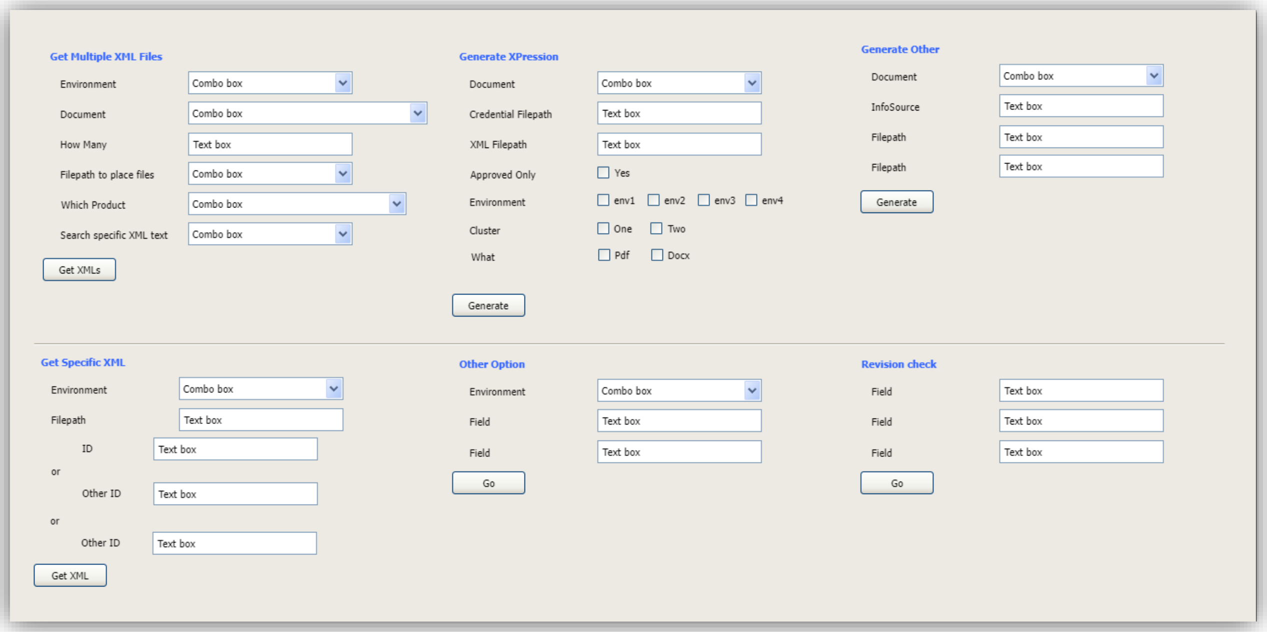


Figure 2.0 – Company Tool 1

This tool no longer seems to be in active development. It has received fixes recently to keep it operational, but there are no upcoming features being developed for it, that I am aware of.

I am not going to perform an analysis of this tools features in comparison to DART's, because, as we will see, the second tool is just a newer, more streamlined version of this tool. This makes the second tool a better candidate for analysis.

Company Tool 2

Company tool 2 is a desktop Java based application that uses the JavaFx framework. This tool is essentially a streamlined version of Company tool 1. It has slightly less options, however, in my day-to-day I have never needed the additional functionality of Company Tool 1 and I feel like these additional options were hold-overs from a time where they were required, but had since become obsolete.

As Company tool 2 is the closest application in existence to the DART application we will focus on it for the purposes of comparison.

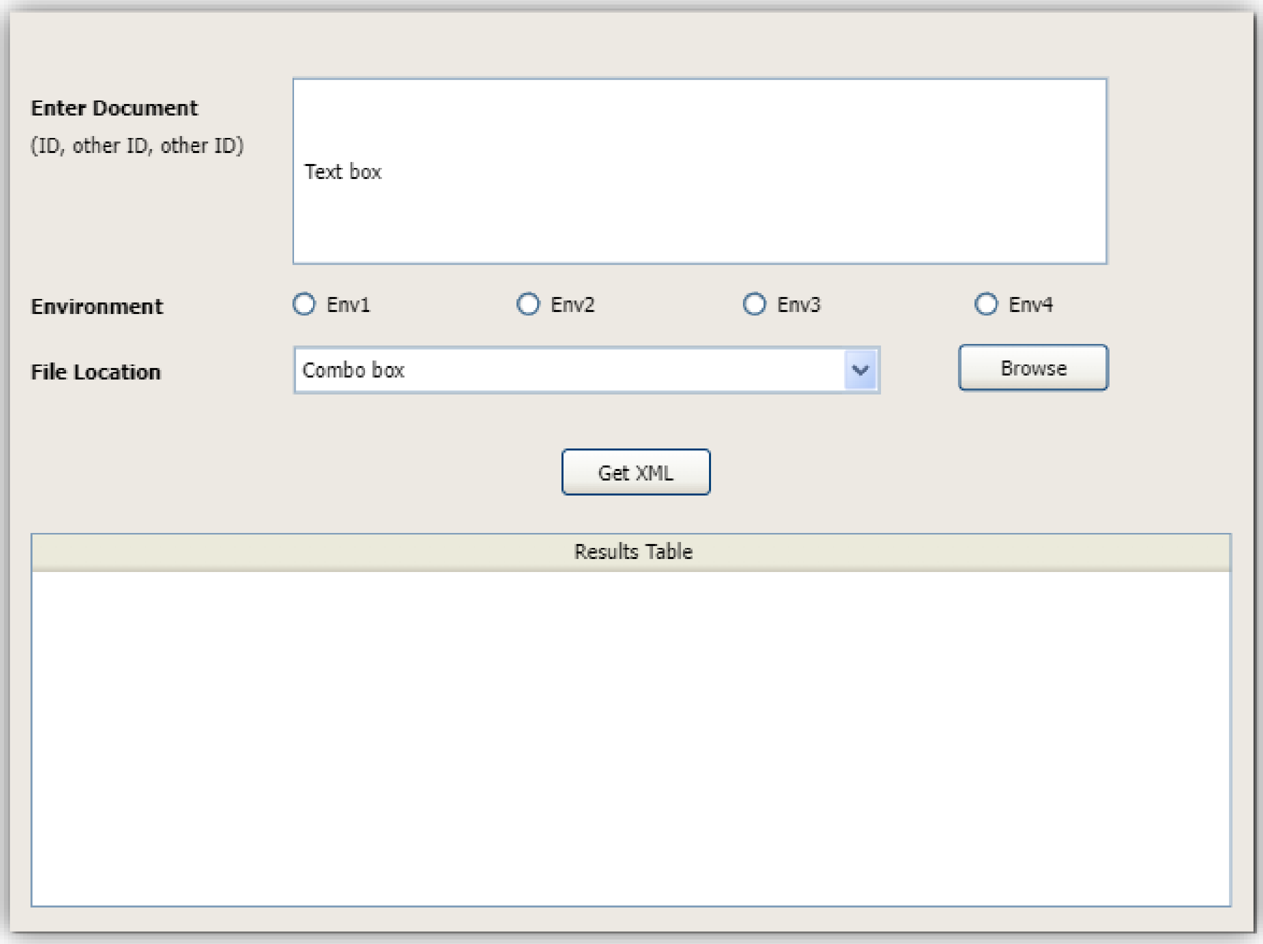


Figure 2.1 – Company Tool 2

Usage

In regards to my usage of this tool, I use it regularly in my day-to-day. I can make a mock XML data set for a document and then use the tool to generate the resulting pdf, amending and altering the XML to produce subsequent document generations. All of the developers, and myself, work in this manner when developing new additions to documents or fixing bugs with content display on existing documents.

Feature Set

Requirements	
Generate PDF Documents	✓
Compare PDF Documents	
Create Difference Reports	
Compiler for Rules Engine	
Monitor Document Generation	
Have a Nice Interface	✓

Figure 2.2 – Comparison Feature Set

Strengths

User Interface
In my opinion the user interface of the application is quite nice. It's easy to understand and provides the feedback one would expect when pushing buttons. Its performant and there are no framerate issues when loading in large amounts of data for generation.

Ease of use
The tool is generally easy to use. It was the first tool that I used within the company to assist in my role and I was able to do so with no tutorials or instruction from any other member of the team. Any errors within the application that may occur are well presented and usually able to be remedied easy enough as the tool is fairly stable and the only issues I have ever noted have been to do with internet connectivity or the service upon which the tool is calling to perform a generation of a document, i.e. things which are outside of the control of the tool.

Function
The actual function of the tool is a huge positive, as stated previously, all developers will use this tool when creating new features and fixing bugs. The ability to generate documents on the fly with custom XML datasets is very important to the overall function of a developer on the documents team.

Improvements

Document Comparison
Document comparisons are very important when it comes to Quality Assurance testers checking a developers work after it has been migrated to an upper region. DART relies on another, lower level, doc diffing tool, to provide comparison reports between a set of PDF's or Word documents. In my opinion, this is a very good improvement over the other Company Tools. This other doc diffing tool was originally licensed before I joined the team but had not seen a huge amount of use internally. DART makes this comparison feature very easy to use and suitable to be used by users of varying technical expertise.

Rules Compilation
The main benefit of DART over the other applications, was that it provided a sandbox within which the rules of a document could be run, with the resulting content assembled outside of the Xpression application. This allows documents to be tested in ways which were not possible before. This value add is a big improvement over any of the tools in circulation and they did not offer anything in comparison to this feature.

Portability & Extensibility
One of the main drawbacks of both internal tools is that they are standalone desktop applications. This has the general consequences of applications distributed in this manner in that compared to modern web applications, desktop applications are more difficult to distribute, keep up to date and install. Compounding this, the application is a Java Application – so it requires the correct Java version also be installed on the user's machine. Issues have arisen within my time at the company wherein a colleague has had errors due to having the wrong version of Java installed and attempting to run the application.

In comparison, the primary method of distribution for DART was via the web. The drawback of this is that, when compared with desktop, the performance may suffer. However, as DART and the other Company Tools actually rely on web calls to another service to produce the documents, this drawback is not seen here. It may become an issue in future should I seek to expand the functionality of DART but at this stage, even a small drawback in performance is a worthwhile trade-off for the ease of use that DART as a web application brings. Users no longer have any responsibility to keep the app up to date or to keep the Java installation correct. This is all managed for them. To use the app there is no start up time, they simply browse to the internal company address that it is hosted at.

Analysis
Company Tool 2 is an overall very good application that does the job it aims to do in a performant and easy to use manner. As it was not aimed at providing comparisons between documents, it is unable to be used for the exact same purpose that DART is. Similarly, as the compilation of the Xpression rules is fairly novel, the other tools don't provide this feature. At the time of the analysis, given the above, I came to the conclusion that there was a unique space that DART could occupy within the company. If DART could provide the same expected functionality of the other tools, along with the additional comparison and testing features - DART would become the go-to tool via its ease of use and functionality.

Technology

DART was required to be a web-based application and the compiler for the XML rules was to be a separate service that the backend could serve to the frontend.

Compiler

I had an inclination early on that the compiler would be required to be built in a language that supported strong typing and the other conveniences supported by languages that are based upon an object orientated paradigm like interfaces, classes, inheritance etc. Initially I had to decide between two languages that I wanted to try out for this project - Kotlin or Go.

Go

Go is a compiled, statically typed language. Go released version 1 in 2012[4] and is still in active development. Go has good industry adoption, it is utilised in many tech companies, such as Google, Meta, Netflix and more[5]. In terms of overall popularity, Go features high enough when discounting languages with respect to the requirements for this part of the project[6].

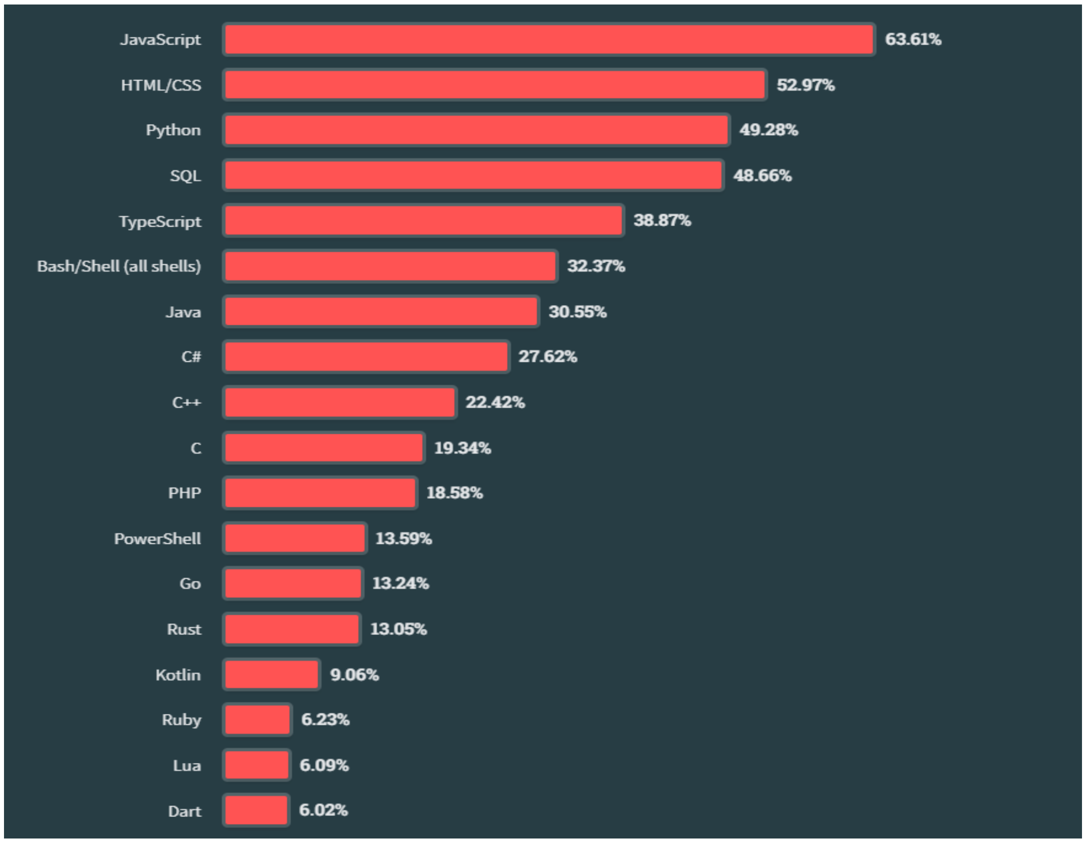


Figure 3.0 – Stack Overflow Developer Survey

[4] Google. (2024). Release History. Available from: https://go.dev/doc/devel/release#go1 [accessed 6 February 2024]

[5] Google. (2024). Case Studies. Available from: https://go.dev/solutions/case-studies [accessed 6 February 2024]

[6] Stack Overflow. (2024). Developer Survey 2023. Available from: https://survey.stackoverflow.co/2023/#most-popular-technologies-language [accessed 6 February 2024]

I believe one of Go's main strengths is that its authors have kept it relatively simple, opting to not add certain features that they felt would lead to language bloat, obviously others may feel that the absence of certain language features is restrictive, but I think this has been a positive development in the Go space. It gives Go the feel of a more dynamically typed and higher-level language like Python, but still retaining some of the power of a compiled lower-level language like C.

Kotlin

The Kotlin programming language was the other contender for the code of the lower-level language. Kotlin is a modern, statically typed, compiled language on the JVM platform. It initially began development in 2011[7] and has since become the main supported language for android mobile development. Whilst Kotlin is mainly known as the language for which to develop android applications in, the language does provide for ecosystems in which to develop more traditional applications, for example, Kotlin Native or Compose Multiplatform. One of Kotlin's main advantages is that it has full interoperability with Java code[8]. As seen in the stack overflow survey previously, Kotlin was considered as a slightly less popular language than Go in 2023, but it does still feature high enough overall when discounting the non-applicable languages.

Kotlin's syntax is modern and very expressive. One of its best features is probably null safety, i.e. nullability of a variable must be explicitly stated by the developer or there will be a compilation error, coding in this manner is slower to start, but much more stable when the foundations have been built.

Kotlin v Go

Both languages would have been well suited to the requirements for the project. Whilst I was initially attracted to Go, I ended up choosing to go with Kotlin.

The main point of reasoning for this decision came down to the fact that the main language used within the company for which the project was being developed is Java. By using Kotlin, I received all the advantages of Go but also the extra advantage of being able to interop with any other company services, or use any internal libraries that were available. IntelliJ can also directly convert Java code to Kotlin code within the editor so this would be beneficial if I ever had to read or adopt company code which I couldn't understand fully due to my lack of experience with Java.

Data Transfer Protocol

For data input to the compiler, I would stick with the already in place format of XML. The Xpression application communicated via XML and the rules for each document were stored in XML so there was not much of a decision to be made. In terms of output, the compiler was only going to respond to web requests for compilation, so it made sense for any of the compilation information to be communicated via JSON, the usual data-transfer language of the web[9].

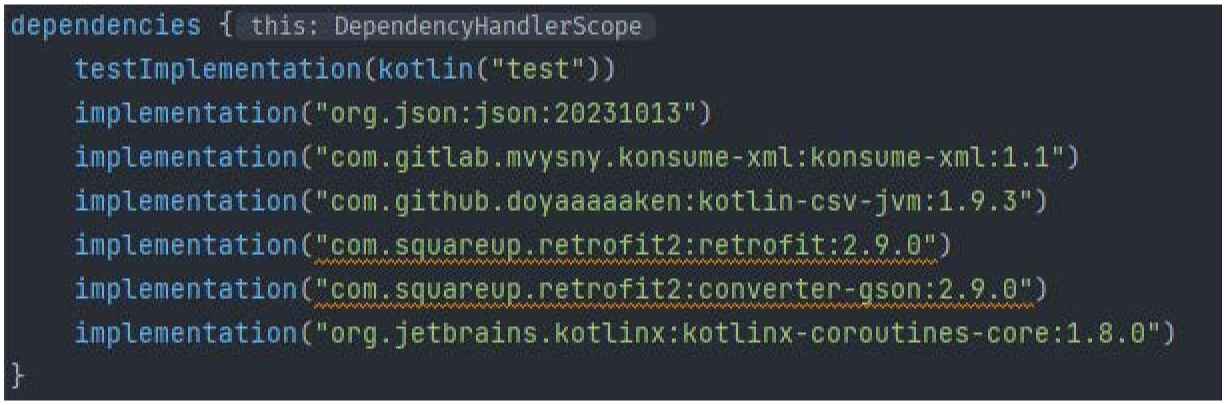
Libraries

There were not many dependencies for the compiler portion of the project.

[7] JetBrains. (2000). Hello World. Available from: https://blog.jetbrains.com/kotlin/2011/07/hello-world-2/ [accessed 6 February 2024]

[8] JetBrains. (2024). FAQ. Available from: https://kotlinlang.org/docs/faq.html [accessed 6 February 2024]

[9] Stack Overflow. (2022). A beginner's guide to JSON, the data format for the internet. Available from: https://stackoverflow.blog/2022/06/02/a-beginners-guide-to-json-the-data-format-for-the-internet/ [accessed 26th February 2024]



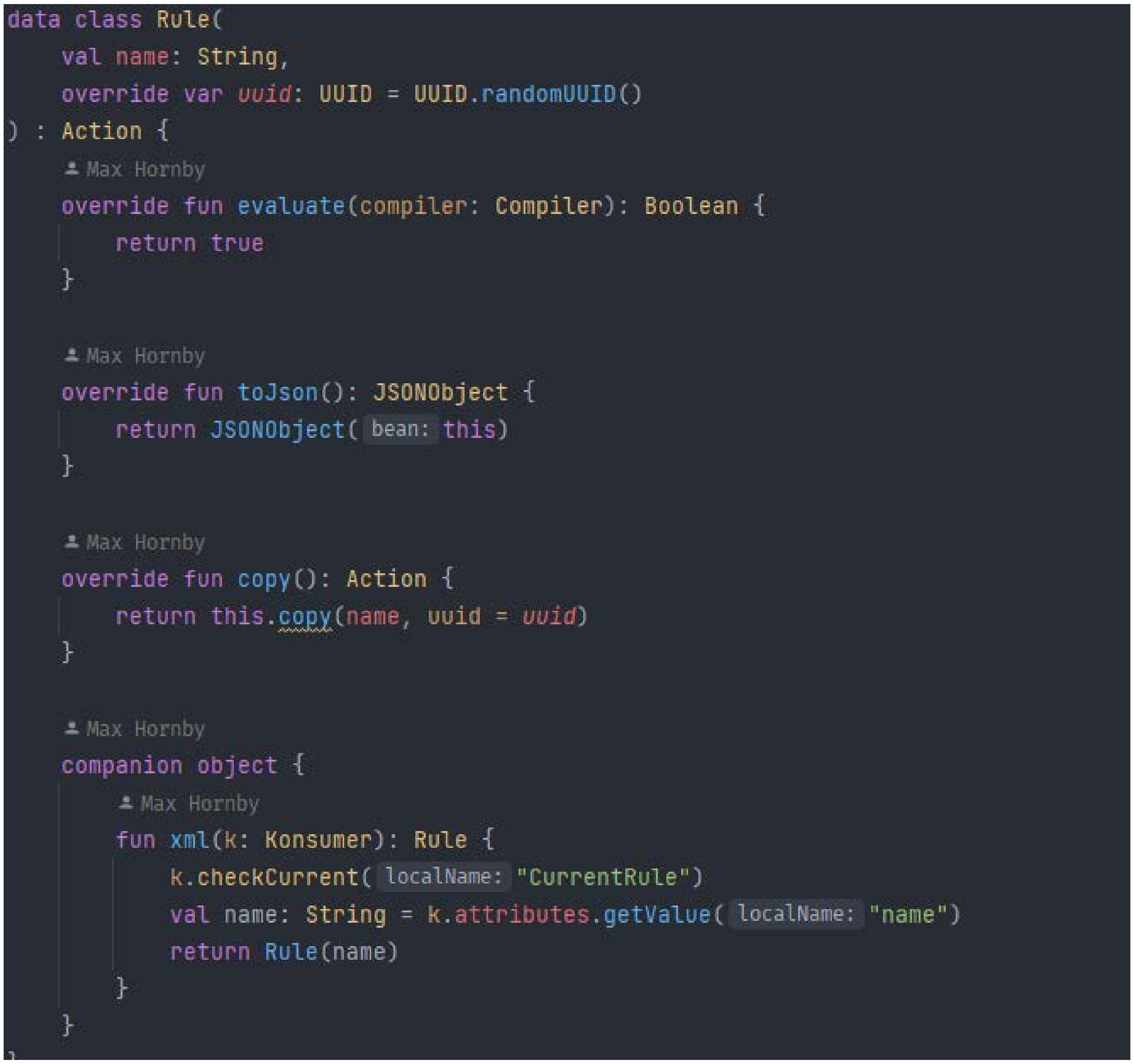
```
dependencies { this: DependencyHandlerScope
    testImplementation(kotlin("test"))
    implementation("org.json:json:20231013")
    implementation("com.gitlab.mvysny.konsume-xml:konsume-xml:1.1")
    implementation("com.github.doyaaaaaken:kotlin-csv-jvm:1.9.3")
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.8.0")
}
```

Figure 4.0 – Compiler Dependencies

Retrofit was used to communicate with the node server, serving the DART application – this was done in order to grab any assets the compiler required to assemble the document. I will detail this later in the modelling section.

The CSV package above ultimately went unused, it was originally used for demo implementations of the compiler at inception of the project.

The most impactful dependency was the XML library, because XML to object serialisation was such an important part of the process of parsing the XML rules. Thankfully, the Konsume library that I ended up with was a far superior option the common annotation based XML serialisation libraries found on the JVM. Konsume was also a Kotlin first library, so it fit perfectly into my program structure.



```
data class Rule(
    val name: String,
    override var uuid: UUID = UUID.randomUUID()
) : Action {
    Max Hornby
    override fun evaluate(compiler: Compiler): Boolean {
        return true
    }

    Max Hornby
    override fun toJson(): JSONObject {
        return JSONObject( bean: this)
    }

    Max Hornby
    override fun copy(): Action {
        return this.copy(name, uuid = uuid)
    }

    Max Hornby
    companion object {
        Max Hornby
        fun xml(k: Konsumer): Rule {
            k.checkCurrent( localName: "CurrentRule")
            val name: String = k.attributes.getValue( localName: "name")
            return Rule(name)
        }
    }
```

Figure 4.1 – Syntax Object Example [Rule]

Web Application

For the frontend of the web application I had already known I wanted to go with something that provided for a modern UI framework that updated as the state changes. My two option's here were between React and Svelte.

React

React is the most popular web framework for building UI's[10]. It was started by an employee at facebook in 2011 and is still in active development. React is the industry standard tool for building stateful UI's and has been adopted by leaders in the tech industry such as AirBnb, Instagram, Netflix and more. React allows for a developer to create components using html, these components can

[10] Stack Overflow. (2024). Developer Survey 2023. Available from: https://survey.stackoverflow.co/2023/#most-popular-technologies-language [accessed 6 February 2024]

then be combined to create large components and web pages. Working this way means a developer can create decoupled components that work anywhere and port them from project to project.

One of Reacts main features that cannot be understated is its rich ecosystem of libraries and packages. Nearly everything a developer could need is provided for in some way or another. There also exists very popular component libraries such as Material UI which can be used as a base from which to build components.

Svelte

Svelte is a newer framework basically seeking to do what React does but in a more developer friendly way. Svelte is developed by Rich Harris and is a major shift in the background workings of the framework when compared with it's contemporaries. Rich has many blog posts and videos in which he criticises the virtual dom reactivity of frameworks like React[11]. Instead of updating and maintaining a virtual dom in order to achieve stateful UI changes, Svelte is a full on compiler that takes code as input and outputs pure Javascript that is reactive.

React v Svelte

In practice, the actual results of the two frameworks cannot be said to be different. Svelte is supposed to be more performant than React, but nothing I am doing in this project would have pushed the boundaries of either framework as my UI requirements are quite light. I found the developer experience of Svelte to be very nice to work with. The way a svelte file is written is that it will contain more native <script> and <style> tags and I thought this was nicer to work within than React. For example, in React when styling a component, you actually create a JS object and apply it as a style. This is more convenient that writing css, but the syntax is changed for the css elements as a result – e.g. the "object-fit" property would be "objectFit" in a React component because a dash cannot be within a key of a JS object. In contrast, styles in Svelte are just written between two <style> tags in css syntax so no need for adjustment.

As I played around more with svelte I eventually came upon the issue that made me throw it away. Svelte cannot handle children components properly. It does have a system where one can create slots and pass components as children but it's not the same as Reacts and is not flexible enough for the purpose a system like this is supposed to solve[12]. It's difficult to discover issues like this unless you use the tool because the community pretends it does not exist in an effort to improve the perception of the framework among peers. Ultimately, this inflexibility led me to choose React as the frontend UI framework.

Libraries

For the libraries that I would rely upon for the frontend of the application, I stuck to very commonly used ones. This was so that if I ran into any issues, I could easily search for solutions online.

[11] Rich Harris (2018). The Virtual Dom is Pure Overhead. Available from: https://svelte.dev/blog/virtual-dom-is-pure-overhead [accessed 26th February 2024]

[12] Github: svelte/svelteJs (2020). Issue #4455. Available from: https://github.com/sveltejs/svelte/issues/4455 [accessed 3rd March 2024]

For data fetching, I went with the tried and tested React Query. This is a standard data fetching library for React. It makes it quite easy to setup and manage data-fetching and caching on the frontend. The benefit of using this library is that one can await the results of a data fetch but get transitory updates on state for example, when making a call with React Query an isLoading state is available to be used to trigger certain UI to show. This was used in DART to trigger loading transitions.

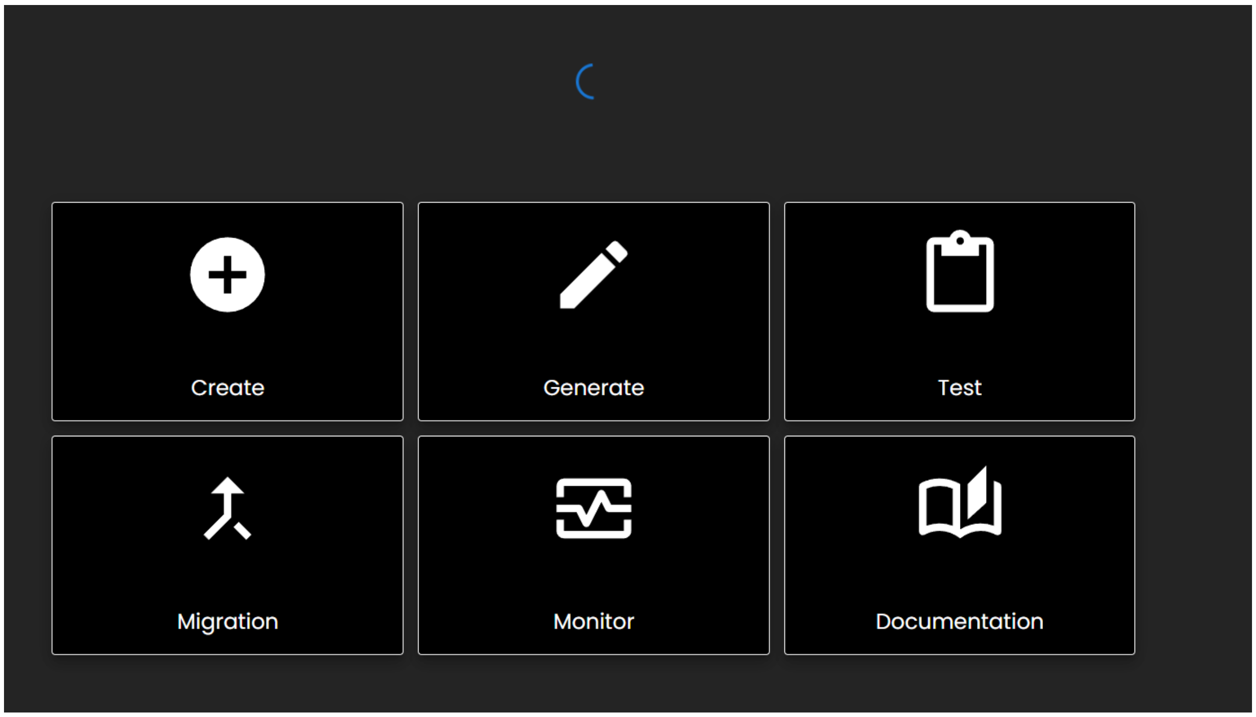


Figure 5.0 – Frontend Dependencies, React Query

I also opted to depend on some lower-level components that I would then build my own component on top of. This can be seen with React Dropzone, which I used as the basis for my dropbox like inputs.

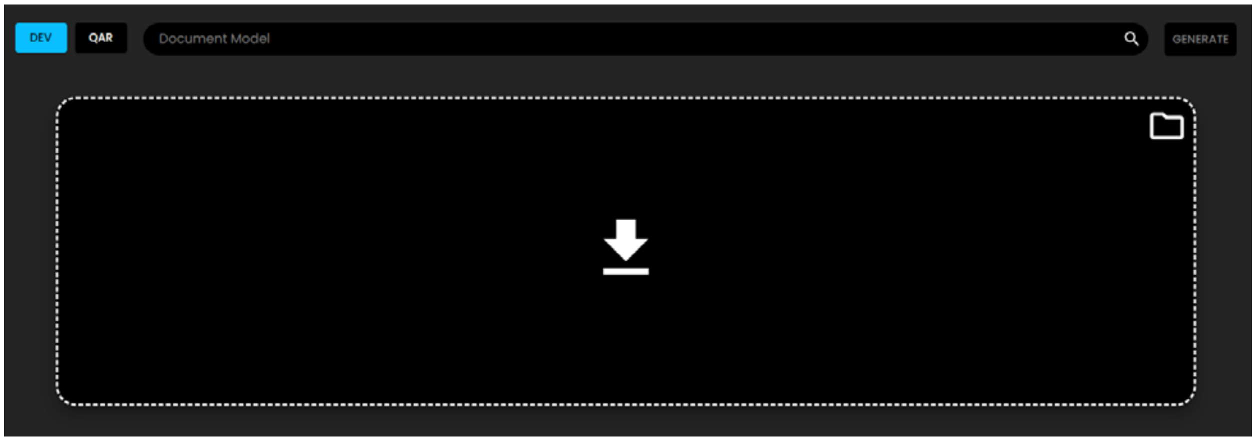
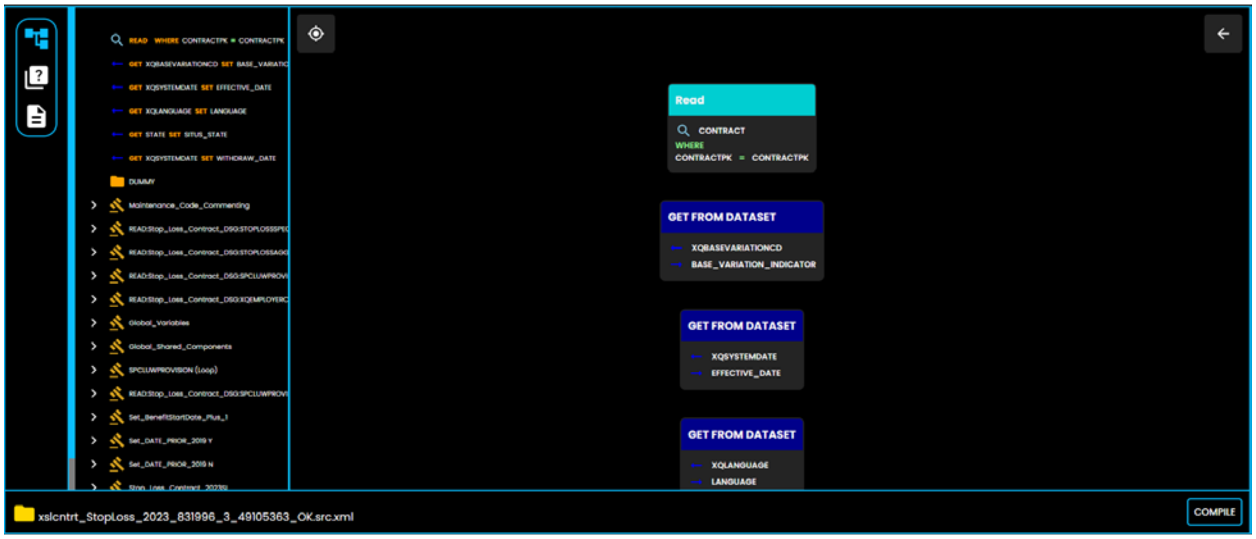


Figure 5.1 – Frontend Dependencies, React Dropzone

Another area that a similar development approach was taken was for the Rules Graph, whereby the graph like tree is a draggable element.



```
READ WHERE CONTRACTPK = CONTRACTPK
GET XQBASEVARIATIONCD SET BASE_VARIATIC
GET XQSYSTEMDATE SET EFFECTIVE_DATE
GET XQLANGUAGE SET LANGUAGE
GET STATE SET SITUS_STATE
GET XQSYSTEMDATE SET WITHDRAW_DATE
DUMMY
Maintenance_Code_Commenting
READ:Stop_Loss_Contract_DSG:STOPLOSSSPEC
READ:Stop_Loss_Contract_DSG:STOPLOSSAGG
READ:Stop_Loss_Contract_DSG:SPCLUWPROVI
READ:Stop_Loss_Contract_DSG:XQEMPLOYERC
Global_Variables
Global_Shared_Components
SPCLUWPROVISION (Loop)
READ:Stop_Loss_Contract_DSG:SPCLUWPROVI
Set_BenefitStartDate_Plus_1
Set_DATE_PRIOR_2019 Y
Set_DATE_PRIOR_2019 N
Stop_Loss_Contract_2023
xslcntrt_StopLoss_2023_831996_3_49105363_OK.src.xml
```

Figure 5.2 – Frontend Dependencies, React Draggable

Document Comparison

In regards to actually generating diffs for pdf documents, my choice of technology was to either use an open-source tool that produces diffs, or, use a licensed third-party tool created for this purpose. I had initially researched other third-party doc diffing tools, but none stood out as being a particularly better option than the tool that was already available internally.

Development Environment

For my development environment, there were some restrictions based on what was available to use within the company. This ultimately had no impact as the tools I would have chosen to develop with were available. For any Kotlin development, I chose IntelliJ IDEA Community edition, and, for all other development, I chose Visual Studio Code. The only alternative choice available was for the latter type of development, with Sublime Text as an available choice, but no consideration was given as Visual Studio code is what I use personally and all of my profile settings were easily able to be ported on to my work laptop for the development of DART.

Source Control

Source control for the project would be managed by Git, specifically BitBucket.

CI/CD

As DART is an internal work-based tool, the options for automated testing and deployment were limited. Instead of attempting to get access and clearance for tools for these purposes, I instead opted to develop powershell and bash scripts for whatever I needed. For example, one of the scripts I used most often was a bash script that build and packaged the frontend client of the project into the public folder of the backend, in this way I could kick off the build script and have a dev copy running in a few seconds. This helped to streamline development. Similar, I had a bash script to build and deploy the web application onto the remote server from which it was being served.

Modelling

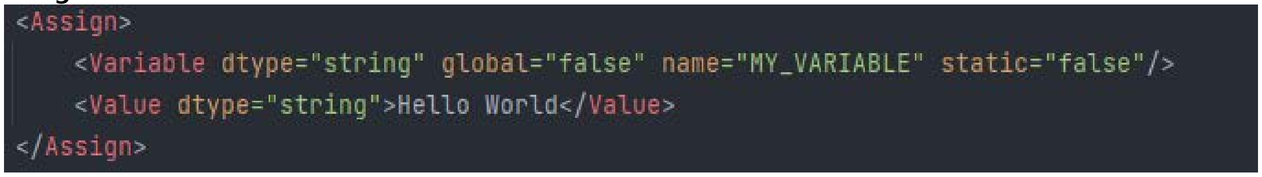
User Stories

User	Case	Acceptance Criteria
Developer / QA Developer / Business Stakeholder	Generate Document	• Easy to use form • Select Box's for document types • Quick Generation • Ability to Generate Multiple Documents at a time
QA Developer / Business Stakeholder	Compare Documents	• Easy to use form • Quick Compare • Difference report produced and available for download
Developer	Unit Test	• Able to test documents for the number of revision units they will produce at runtime
Developer	Migration Test	• Able to test a number of baseline documents against to see if they will produce duplicate revision units at runtime prior to regional move
Business Stakeholder	Region Comparison	• Able to click on a baseline document and see a different report between QA and Production regions.

Figure 6.0 – User Stories

Syntax Elements

Assignment

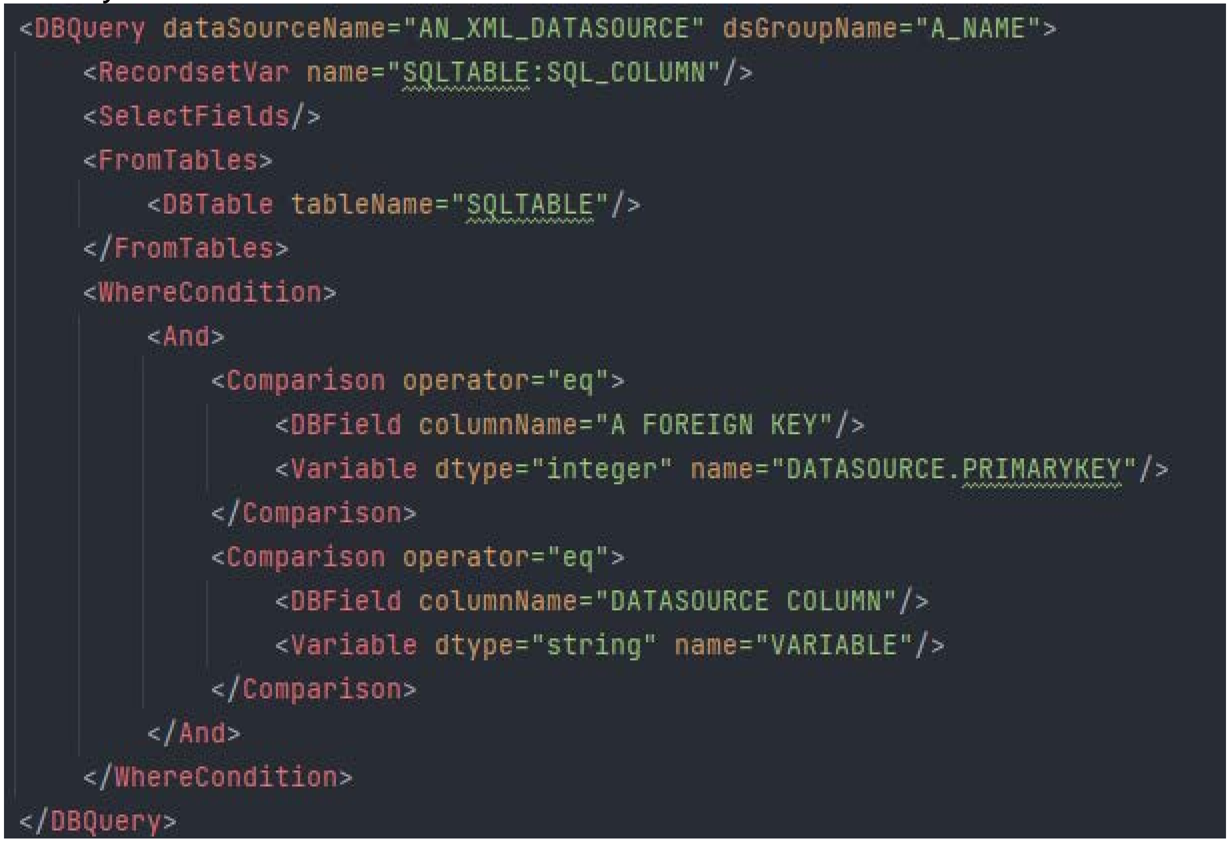


```
<Assign>
    <Variable dtype="string" global="false" name="MY_VARIABLE" static="false"/>
    <Value dtype="string">Hello World</Value>
</Assign>
```

Figure 7.0 – Assignment Syntax

Assigns the second elements value to the first element.

DbQuery

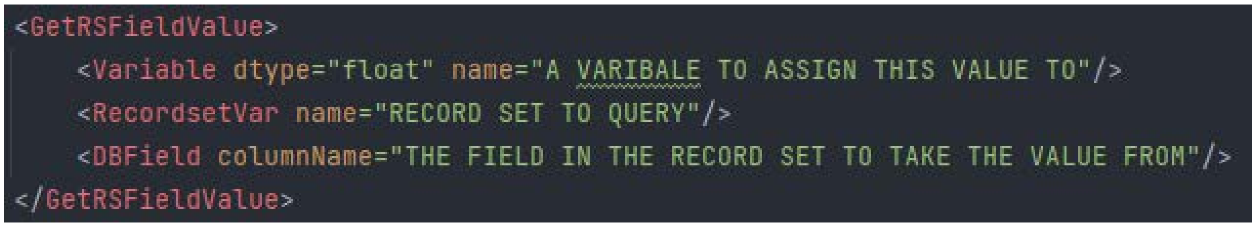


```
<DBQuery dataSourceName="AN_XML_DATASOURCE" dsGroupName="A_NAME">
    <RecordsetVar name="SQLTABLE:SQL_COLUMN"/>
    <SelectFields/>
    <FromTables>
        <DBTable tableName="SQLTABLE"/>
    </FromTables>
    <WhereCondition>
        <And>
            <Comparison operator="eq">
                <DBField columnName="A FOREIGN KEY"/>
                <Variable dtype="integer" name="DATASOURCE.PRIMARYKEY"/>
            </Comparison>
            <Comparison operator="eq">
                <DBField columnName="DATASOURCE COLUMN"/>
                <Variable dtype="string" name="VARIABLE"/>
            </Comparison>
        </And>
    </WhereCondition>
</DBQuery>
```

Figure 7.1 – DbQuery Syntax

Queries an xml datasource for data, the data is read into an SQL like data structure which is why the references to SQL above, the SQL data structure becomes the active record set, and any database queries made afterward are made against this data set until another is read in.

Get From Data

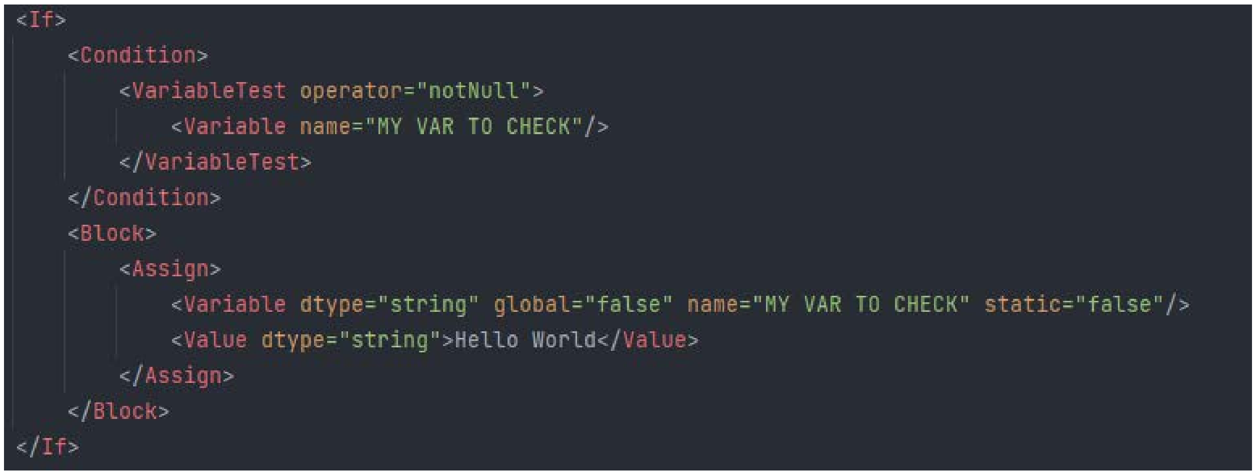


```
<GetRSFieldValue>
    <Variable dtype="float" name="A VARIBALE TO ASSIGN THIS VALUE TO"/>
    <RecordsetVar name="RECORD SET TO QUERY"/>
    <DBField columnName="THE FIELD IN THE RECORD SET TO TAKE THE VALUE FROM"/>
</GetRSFieldValue>
```

Figure 7.2 – DbField Syntax

Queries the active record set for a value from the datasource.

IF

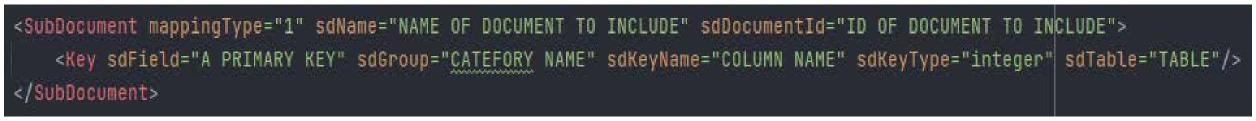


```
<If>
    <Condition>
        <VariableTest operator="notNull">
            <Variable name="MY VAR TO CHECK"/>
        </VariableTest>
    </Condition>
    <Block>
        <Assign>
            <Variable dtype="string" global="false" name="MY VAR TO CHECK" static="false"/>
            <Value dtype="string">Hello World</Value>
        </Assign>
    </Block>
</If>
```

Figure 7.3 – If Syntax

The syntax is an example of control flow, the variable contained within the <Condition> block, is checked to see if it is not equal to null. If this condition evaluates to true, the subsequent block is executed, in this case – this subsequent block simply assigns the value of "hello world" to the variable.

Subdocument

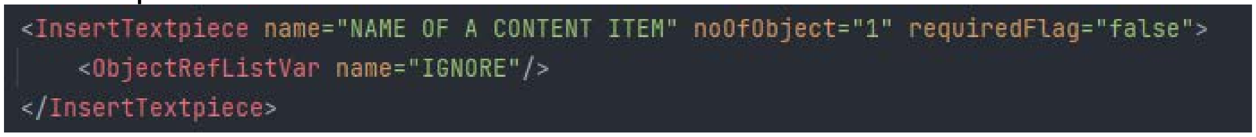


```
<SubDocument mappingType="1" sdName="NAME OF DOCUMENT TO INCLUDE" sdDocumentId="ID OF DOCUMENT TO INCLUDE">
    <Key sdField="A PRIMARY KEY" sdGroup="CATEFORY NAME" sdKeyName="COLUMN NAME" sdKeyType="integer" sdTable="TABLE"/>
</SubDocument>
```

Figure 7.4 – Subdocument Syntax

This syntax element allows for the inclusion of a different document, the compile will grab the rules for whatever document is to be included from the backend of the application. These rules will then be unrolled in place.

Insert Textpiece

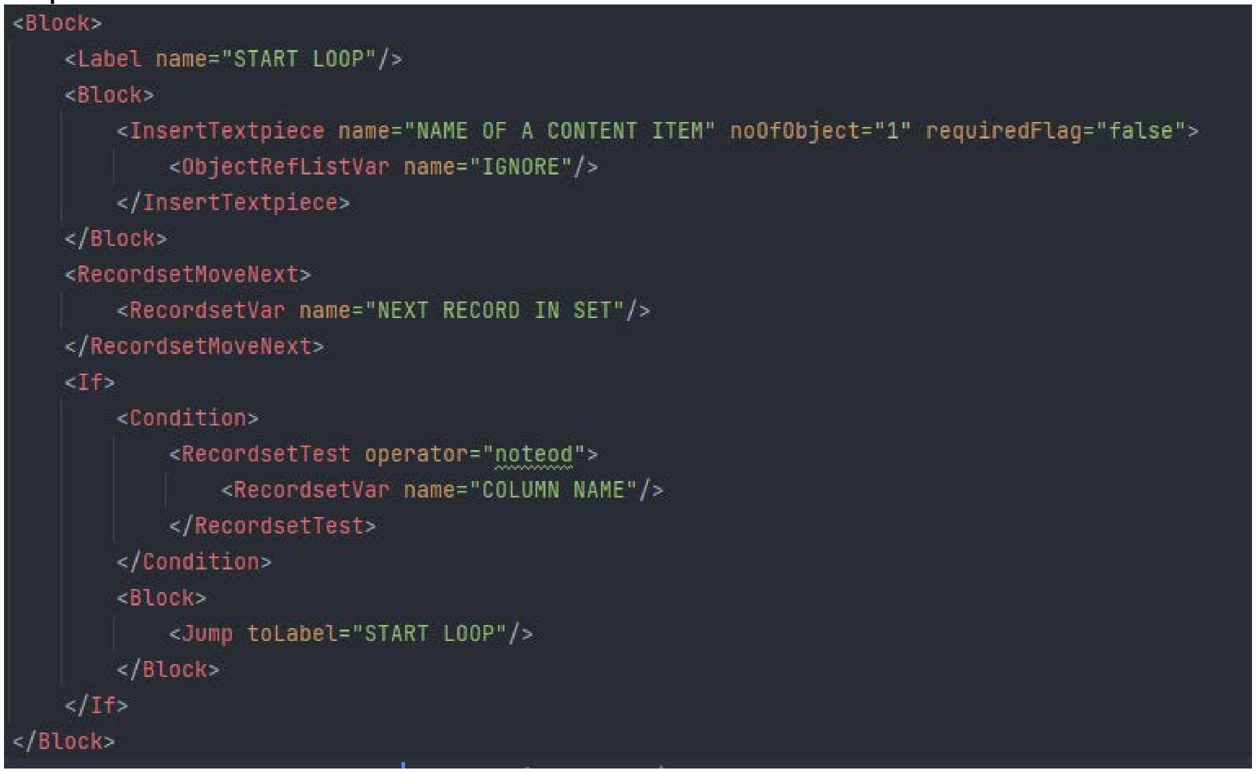


```
<InsertTextpiece name="NAME OF A CONTENT ITEM" noOfObject="1" requiredFlag="false">
    <ObjectRefListVar name="IGNORE"/>
</InsertTextpiece>
```

Figure 7.5 – Insert Textpiece Syntax

This is the actual element that causes to insertion of a content item into a document. There is a slightly more terse explanation to be given regarding how the content item ID's are obtained, but an understanding of this is not relevant for what the element is actually doing when its encountered by the compiler.

Loop



```
<Block>
    <Label name="START LOOP"/>
    <Block>
        <InsertTextpiece name="NAME OF A CONTENT ITEM" noOfObject="1" requiredFlag="false">
            <ObjectRefListVar name="IGNORE"/>
        </InsertTextpiece>
    </Block>
    <RecordsetMoveNext>
        <RecordsetVar name="NEXT RECORD IN SET"/>
    </RecordsetMoveNext>
    <If>
        <Condition>
            <RecordsetTest operator="noteod">
                <RecordsetVar name="COLUMN NAME"/>
            </RecordsetTest>
        </Condition>
        <Block>
            <Jump toLabel="START LOOP"/>
        </Block>
    </If>
</Block>
```

Figure 7.6 – Loop Syntax

This element is what allows for loops. Loops are more like GOTO statements in that a label is created at a certain position in the document. Then there is a Jump element which will direct the programme to start back from the Label position, normally this Jump is preceded by some sort of condition to avoid looping infinitely. In the case above, the current record is first moved to the next Record. The IF block checks to see if the Record Set is empty, if it is not empty we Jump backwards to the Loop Start Label.

Design

The overall design and theme of the application was just supposed to adhere to a few key points that I had created in my head:

- Dark Theme
- Big Buttons and Icons
- Feedback for Hover's and Clicks
- Simple

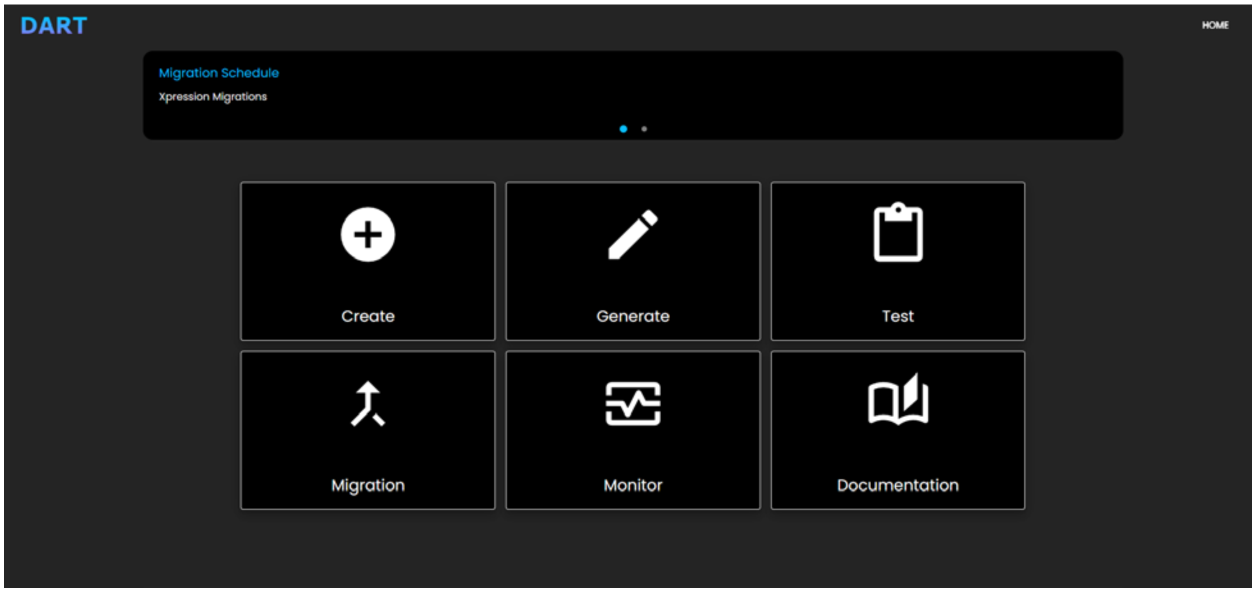


Figure 8.0 – UI Design, Home Screen

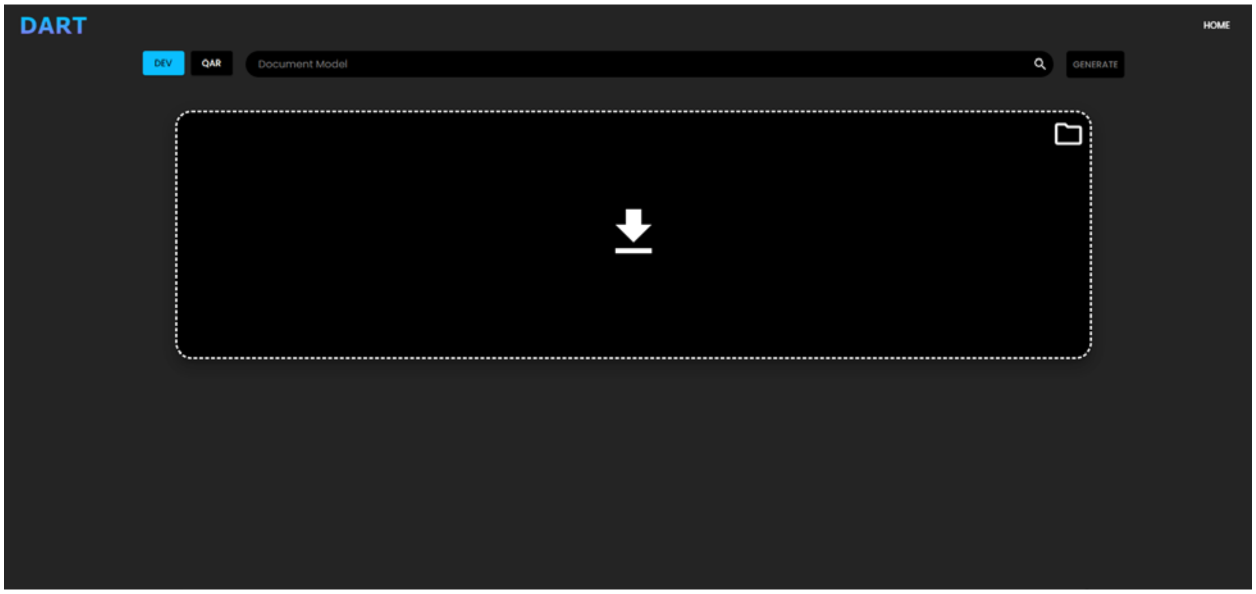


Figure 8.1 – UI Design, Generation Screen

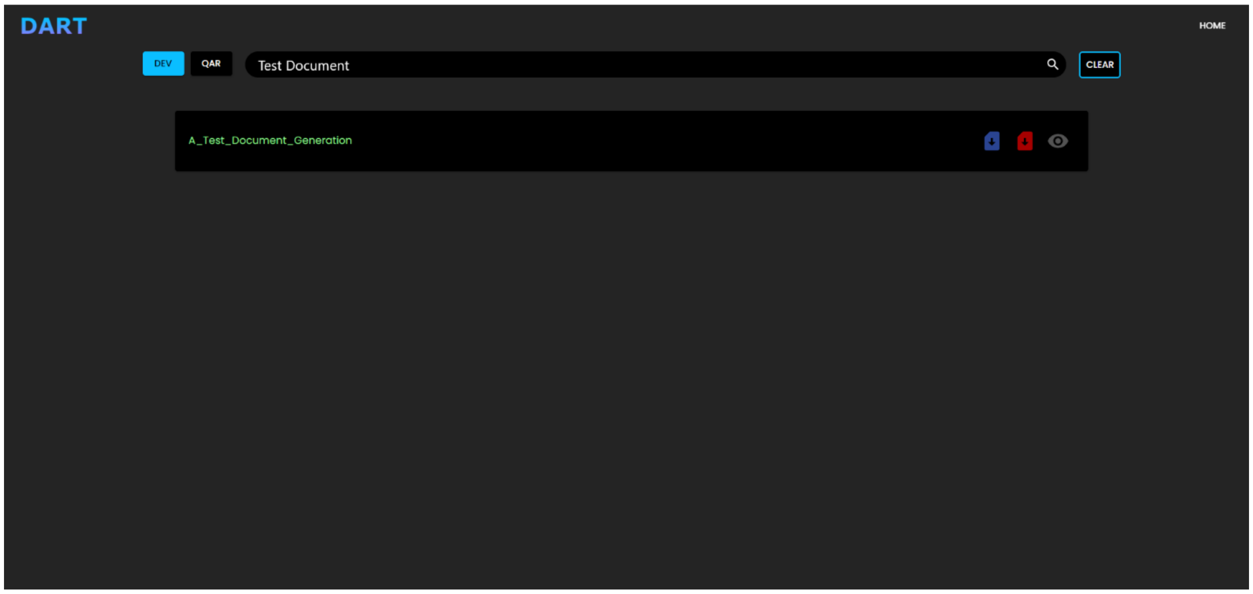
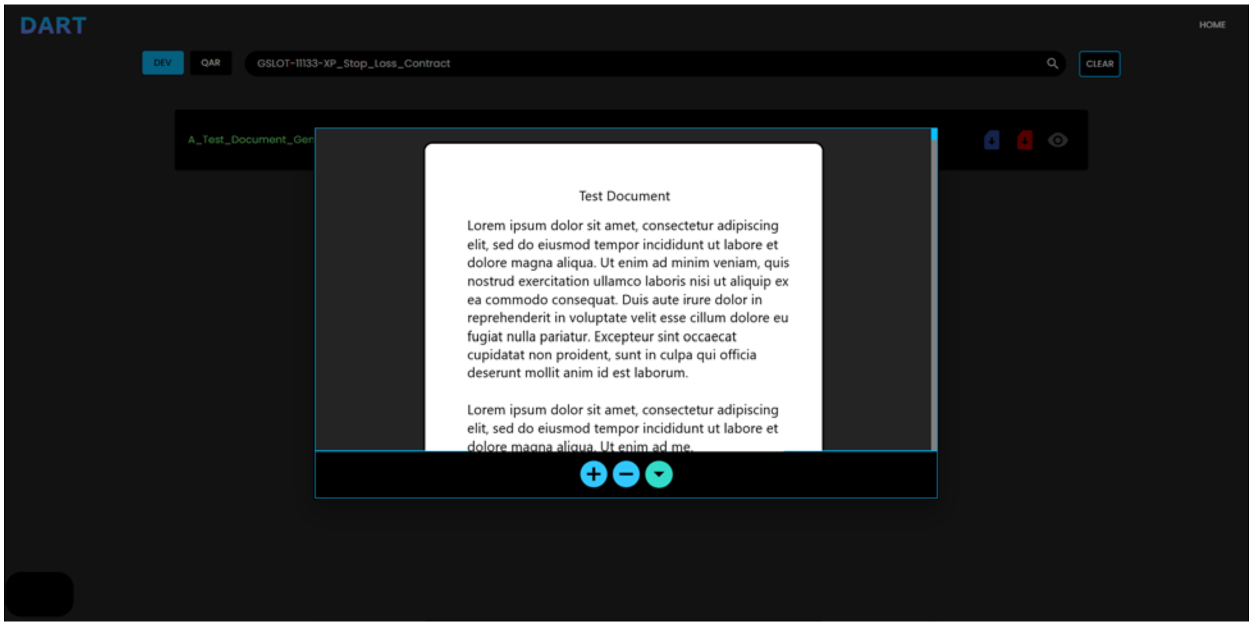


Figure 8.2 – UI Design, Generation Success



Test Document

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad me

Figure 8.3 – UI Design, PDF Viewer

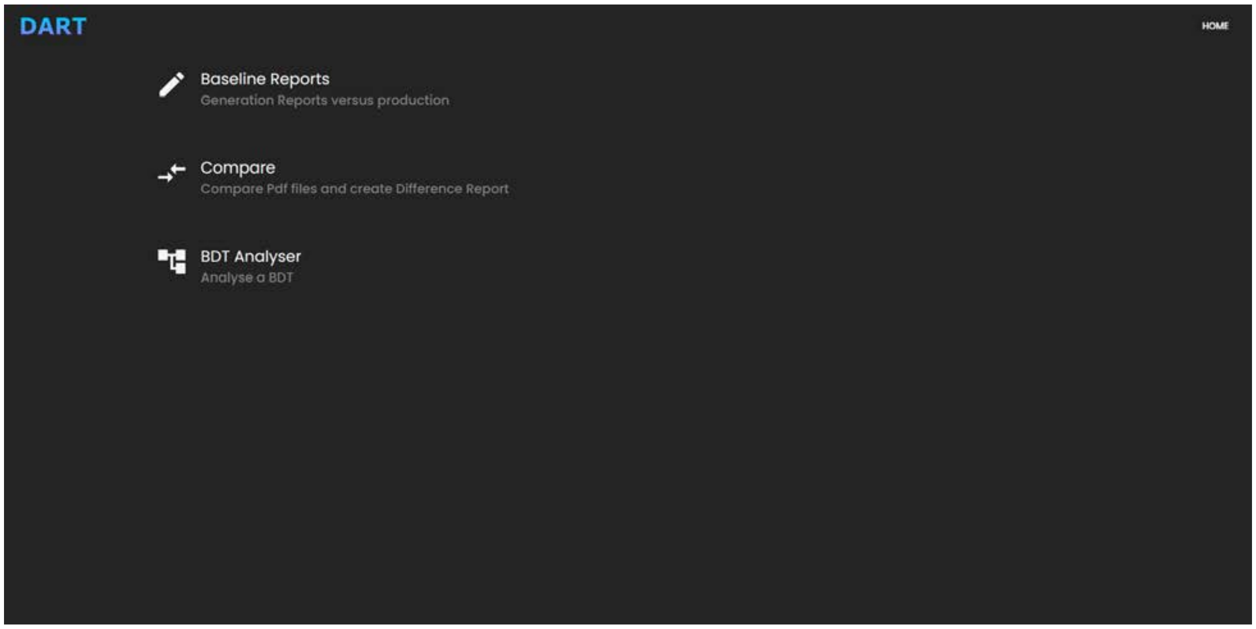

DART
HOME
Baseline Reports
Generation Reports versus production
Compare
Compare Pdf files and create Difference Report
BDT Analyser
Analyse a BDT

Figure 8.4 – UI Design, Testing Options

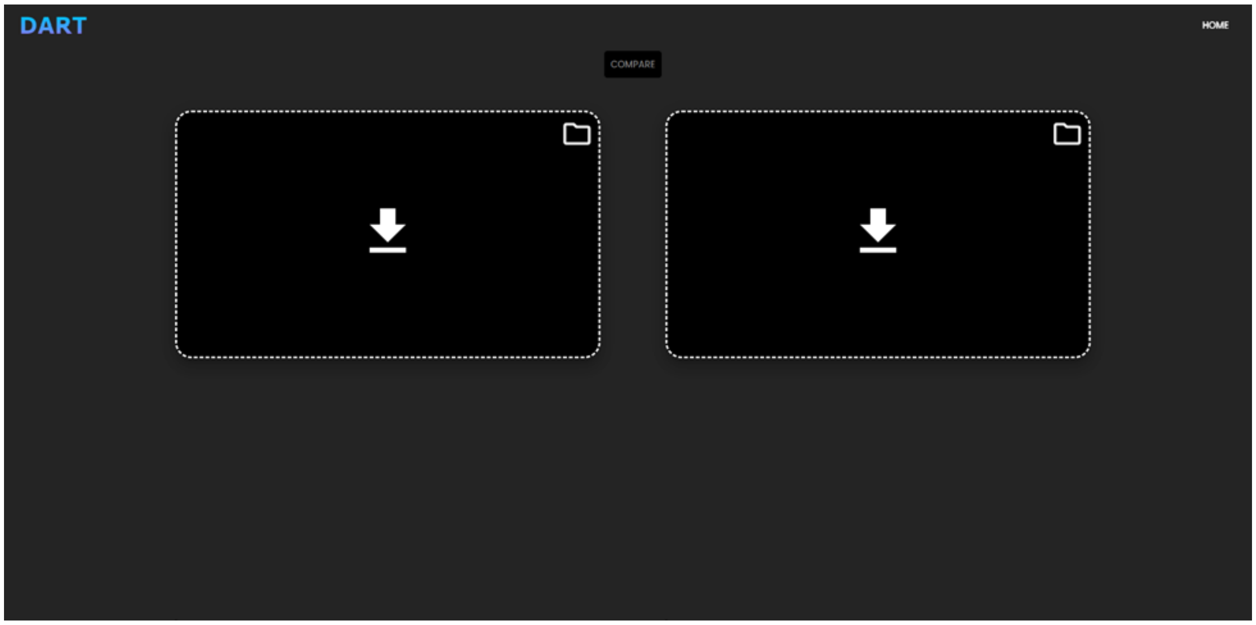
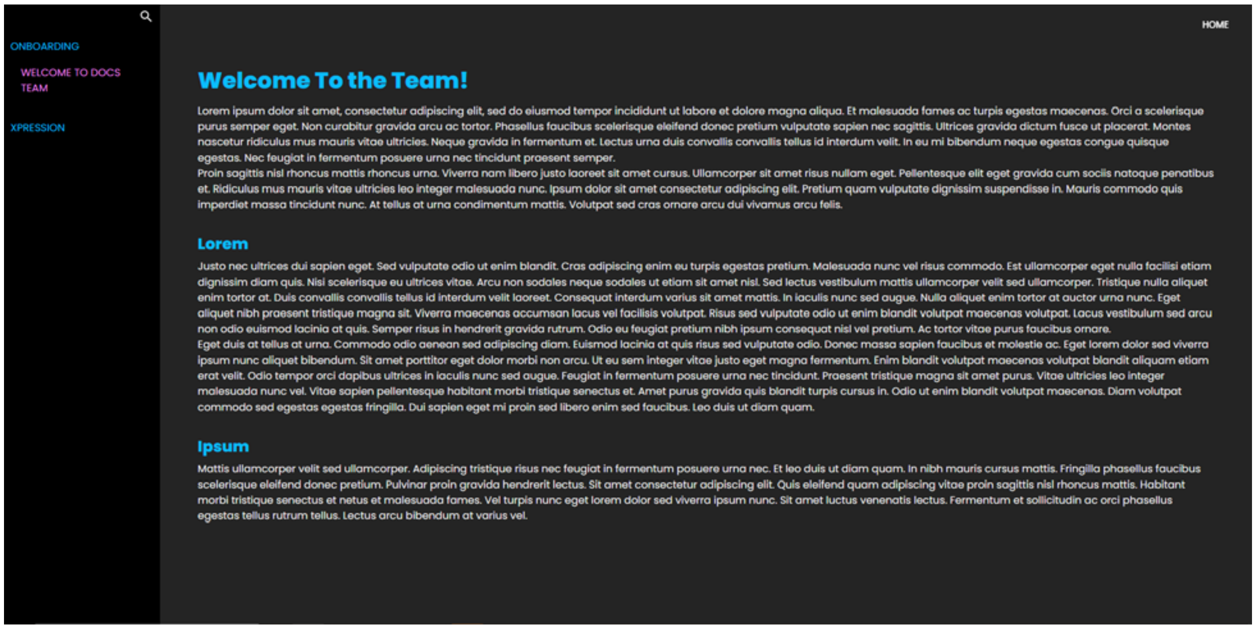


Figure 8.5 – UI Design, Comparison Screen



Welcome To the Team!

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Et malesuada fames ac turpis egestas maecenas. Orci a scelerisque purus semper eget. Non curabitur gravida arcu ac tortor. Phasellus faucibus scelerisque eleifend donec pretium vulputate sapien nec sagittis. Ultrices gravida dictum fusce ut placerat. Montes nascetur ridiculus mus mauris vitae ultricies. Neque gravida in fermentum et. Lectus urna duis convallis convallis tellus id interdum velit. In eu mi bibendum neque egestas congue quisque egestas. Nec feugiat in fermentum posuere urna nec tincidunt praesent semper.
Proin sagittis nisl rhoncus mattis rhoncus urna. Viverra nam libero justo laoreet sit amet cursus. Ullamcorper sit amet risus nullam eget. Pellentesque elit eget gravida cum sociis natoque penatibus et. Ridiculus mus mauris vitae ultricies leo integer malesuada nunc. Ipsum dolor sit amet consectetur adipiscing elit. Pretium quam vulputate dignissim suspendisse in. Mauris commodo quis imperdiet massa tincidunt nunc. At tellus at urna condimentum mattis. Volutpat sed cras ornare arcu dui vivamus arcu felis.

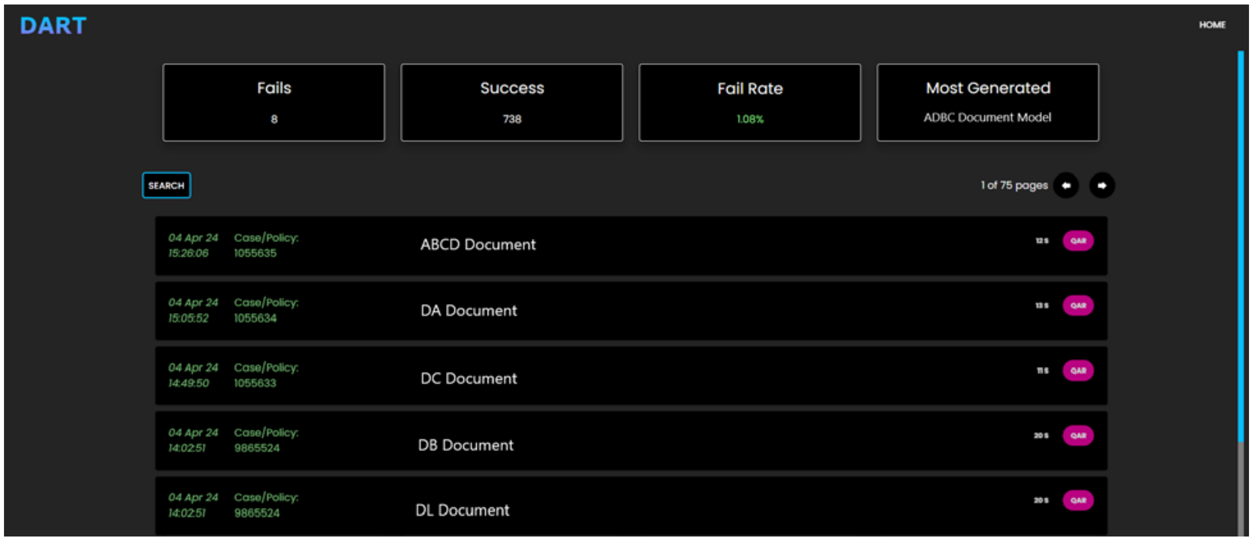
Lorem

Justo nec ultrices dui sapien eget. Sed vulputate odio ut enim blandit. Cras adipiscing enim eu turpis egestas pretium. Malesuada nunc vel risus commodo. Est ullamcorper eget nulla facilisi etiam dignissim diam quis. Nisi scelerisque eu ultrices vitae. Arcu non sodales neque sodales ut etiam sit amet nisl. Sed lectus vestibulum mattis ullamcorper velit sed ullamcorper. Tristique nulla aliquet enim tortor at. Duis convallis convallis tellus id interdum velit laoreet. Consequat interdum varius sit amet mattis. In iaculis nunc sed augue. Nulla aliquet enim tortor at auctor urna nunc. Eget aliquet nibh praesent tristique magna sit. Viverra maecenas accumsan lacus vel facilisis volutpat. Risus sed vulputate odio ut enim blandit volutpat maecenas volutpat. Lacus vestibulum sed arcu non odio euismod lacinia at quis. Semper risus in hendrerit gravida rutrum. Odio eu feugiat pretium nibh ipsum consequat nisl vel pretium. Ac tortor vitae purus faucibus ornare.
Eget duis at tellus at urna. Commodo odio aenean sed adipiscing diam. Euismod lacinia at quis risus sed vulputate odio. Donec massa sapien faucibus et molestie ac. Eget lorem dolor sed viverra ipsum nunc aliquet bibendum. Sit amet porttitor eget dolor morbi non arcu. Ut eu sem integer vitae justo eget magna fermentum. Enim blandit volutpat maecenas volutpat blandit aliquam etiam erat velit. Odio tempor orci dapibus ultrices in iaculis nunc sed augue. Feugiat in fermentum posuere urna nec tincidunt. Praesent tristique magna sit amet purus. Vitae ultricies leo integer malesuada nunc vel. Vitae sapien pellentesque habitant morbi tristique senectus et. Amet purus gravida quis blandit turpis cursus in. Odio ut enim blandit volutpat maecenas. Diam volutpat commodo sed egestas egestas fringilla. Dui sapien eget mi proin sed libero enim sed faucibus. Leo duis ut diam quam.

Ipsum

Mattis ullamcorper velit sed ullamcorper. Adipiscing tristique risus nec feugiat in fermentum posuere urna nec. Et leo duis ut diam quam. In nibh mauris cursus mattis. Fringilla phasellus faucibus scelerisque eleifend donec pretium. Pulvinar proin gravida hendrerit lectus. Sit amet consectetur adipiscing elit. Quis eleifend quam adipiscing vitae proin sagittis nisl rhoncus mattis. Habitant morbi tristique senectus et netus et malesuada fames. Vel turpis nunc eget lorem dolor sed viverra ipsum nunc. Sit amet luctus venenatis lectus. Fermentum et sollicitudin ac orci phasellus egestas tellus rutrum tellus. Lectus arcu bibendum at varius vel.

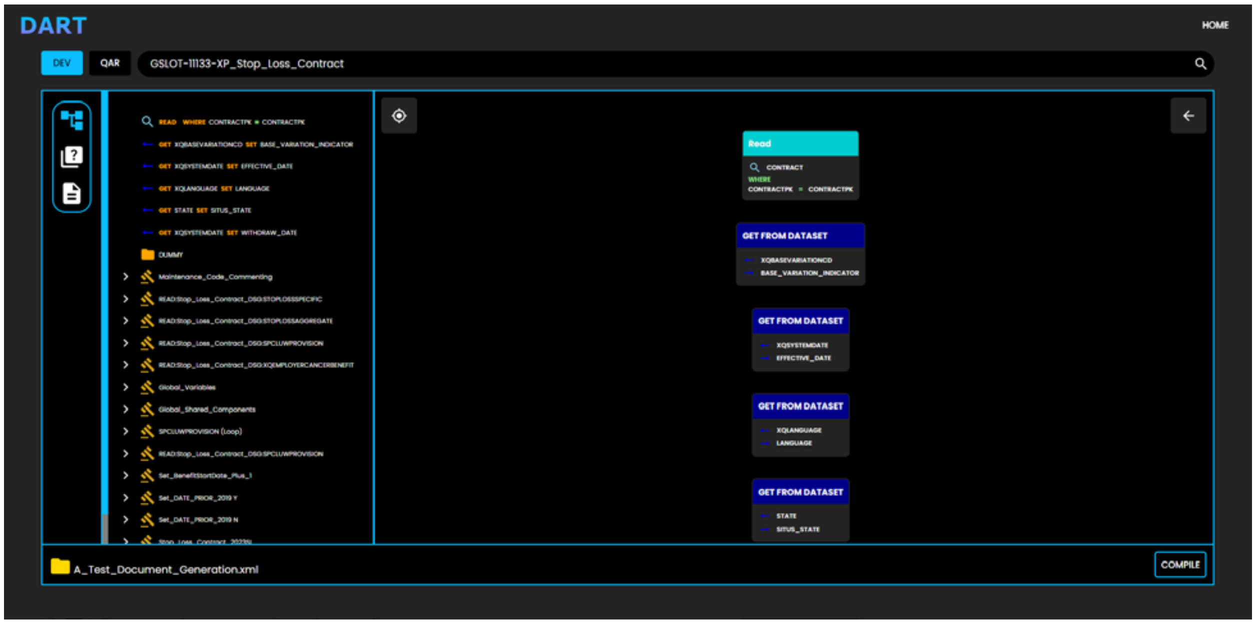
Figure 8.6 – UI Design, Documentation Screen



Fails	Success	Fail Rate	Most Generated
8	738	1.08%	ADBC Document Model

04 Apr 24 15:26:06	Case/Policy: 1055635	ABCD Document	12 s	QAR
04 Apr 24 15:05:52	Case/Policy: 1055634	DA Document	13 s	QAR
04 Apr 24 14:49:50	Case/Policy: 1055633	DC Document	11 s	QAR
04 Apr 24 14:02:51	Case/Policy: 9865524	DB Document	20 s	QAR
04 Apr 24 14:02:51	Case/Policy: 9865524	DL Document	20 s	QAR

Figure 8.7 – UI Design, Monitor Screen


DART
HOME
DEV
QAR
GSLOT-11133-XP_Stop_Loss_Contract
COMPILE
A_Test_Document_Generation.xml

READ WHERE CONTRACTPK = CONTRACTPK
GET XQBASEVARIATIONCD SET BASE_VARIATION_INDICATOR
GET XQSYSTEMDATE SET EFFECTIVE_DATE
GET XQLANGUAGE SET LANGUAGE
GET STATE SET SITUS_STATE
GET XQSYSTEMDATE SET WITHDRAW_DATE
DUMMY
Maintenance_Code_Commenting
READ:Stop_Loss_Contract_DSG:STOPLOSSSPECIFIC
READ:Stop_Loss_Contract_DSG:STOPLOSSAGGREGATE
READ:Stop_Loss_Contract_DSG:SPCLUWPROVISION
READ:Stop_Loss_Contract_DSG:XQEMPLOYERCANCERBENEFIT
Global_Variables
Global_Shared_Components
SPCLUWPROVISION (Loop)
READ:Stop_Loss_Contract_DSG:SPCLUWPROVISION
Set_BenefitStartDate_Plus_1
Set_DATE_PRIOR_2019 Y
Set_DATE_PRIOR_2019 N

Read
CONTRACT
WHERE
CONTRACTPK = CONTRACTPK

GET FROM DATASET
XQBASEVARIATIONCD
BASE_VARIATION_INDICATOR

GET FROM DATASET
XQSYSTEMDATE
EFFECTIVE_DATE

GET FROM DATASET
XQLANGUAGE
LANGUAGE

GET FROM DATASET
STATE
SITUS_STATE

Figure 8.8 – UI Design, Compilation Screen

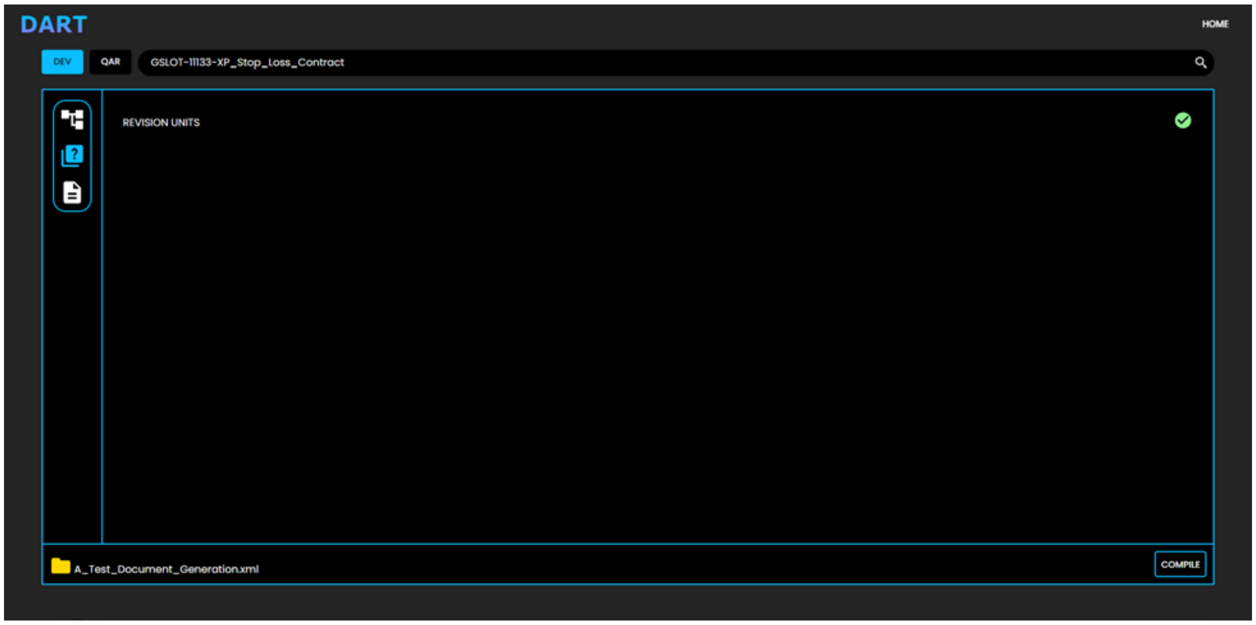

DART
HOME
DEV
QAR
GSLOT-11133-XP_Stop_Loss_Contract
REVISION UNITS
COMPILE
A_Test_Document_Generation.xml

Figure 8.9 – UI Design, Compilation Tests Screen

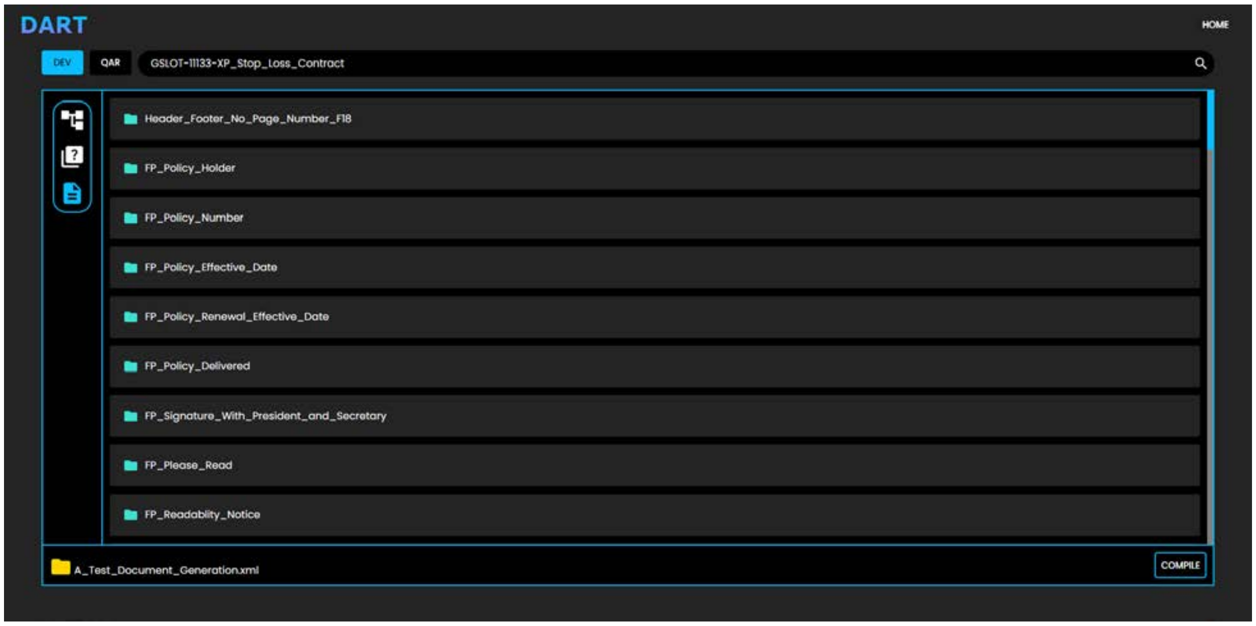

DART
HOME
DEV
QAR
GSLOT-11133-XP_Stop_Loss_Contract
Header_Footer_No_Page_Number_F18
FP_Policy_Holder
FP_Policy_Number
FP_Policy_Effective_Date
FP_Policy_Renewal_Effective_Date
FP_Policy_Delivered
FP_Signature_With_President_and_Secretary
FP_Please_Read
FP_Readability_Notice
A_Test_Document_Generation.xml
COMPILE

Figure 8.10 – UI Design, Compilation Content Screen

Planning

In terms of planning methodology, I opted to adopt Agile style project management.

Agile

Agile project management is an iterative approach to delivering a project, which focuses on continuous releases that incorporate customer feedback. The ability to adjust during each iteration promotes velocity and adaptability. This approach is different from a linear, waterfall project management approach, which follows a set path with limited deviation[13]. An Agile project is normally separated into smaller, more manageable incremental tasks that build toward a larger task. These incremental tasks can be separated into stories, epics and initiatives[14].

- Stories, also called "user stories," are short requirements or requests written from the perspective of an end user.

- Epics are large bodies of work that can be broken down into a number of smaller tasks (called stories).

- Initiatives are collections of epics that drive toward a common goal.

The reasoning behind adopting this style of project management was mainly because agile focuses upon continuous releases and the DART command line interface part of the project was due to be adopted much earlier than the Web Application portion. This meant it would be better to continuously develop, adding in functionality as I go, but always having a version of the command line interface available for use.

Tools

To manage project planning, I opted to go with Jira. Jira is a suite of software planning and management tools provided Atlassian[15]. Jira makes it easy to set up projects, create epics, stories and tasks and manage sprints. It is an industry standard tool and has many plugins available to help design and display software requirements.

Getting Started

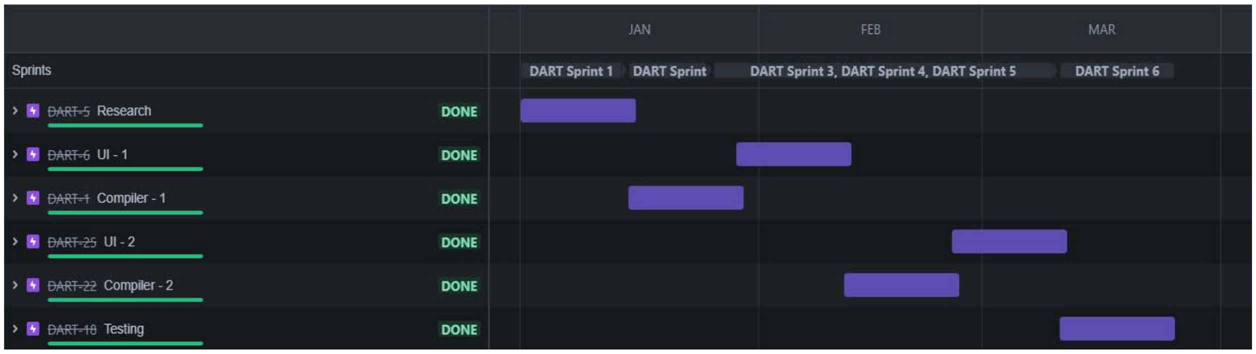
In aligning with the Agile project methodology, I separated the project into 6 distinct two-week sprints, that would take place between January and March. Each sprint would contain 1 – 2 epics, and each epic would contain a subset of stories that were due for that sprint.

[13] Atlassian. (2024). Agile vs Waterfall Project Management. Available from: https://www.atlassian.com/agile/project-management/project-management-intro [accessed 10 February 2024]

[14] Atlassian. Max Rehkopf. (2024). Stories, epics, and initiatives. Available from https://www.atlassian.com/agile/project-management/epics-stories-themes [accessed 10 February 2024]

[15] Atlassian. (2024). Introduction to Jira family. Available from https://www.atlassian.com/software/jira/guides/more/jira-family#what-is-the-jira-family [accessed 10 February 2024]

Implementation



~~DART-5~~	Research	DONE
~~DART-6~~	UI - 1	DONE
~~DART-1~~	Compiler - 1	DONE
~~DART-25~~	UI - 2	DONE
~~DART-22~~	Compiler - 2	DONE
~~DART-18~~	Testing	DONE

Figure 9.0 – Jira Sprint Plan

Sprint 1: Research

The first Sprint of the project was dedicated to research and deliberation on the tools and such that I would be using to create my application. I had been with the company for nearly one year at this stage but I did not have much exposure to inner workings of databases or tools that were available for development. I did not really know at this stage if it was actually feasible to create a web application.

Thankfully, just prior to this project I had completed a python automation project which was very first initial inspiration for DART. This python project was centred on running automation scripts to generate a number of documents in the quality assurance region, and compare them to the production region. This project basically got me started with the backend and the endpoints I would need for document generation. I had worked closely with the senior developers on the team when developing this initial python project, this led to me booking a series of meetings with the lead dev where we would have longer form conversations about the inner workings of Xpression, and any prior attempts to do something similar to what I was attempting with the compiler.

Along with the meetings, I used this time to set up my development environment, luckily IntelliJ community edition was already an approved IDE at the company and no steps were required to install it apart from a simple installation request, this made the setup for the Kotlin portion of the project very issue.

I was able to take some time in this phase to watch Udemy refreshers on Kotlin and React. I had only ever done Android development previously with Kotlin, so wanted to make sure I was up to speed to be able to use it in a more general sense, outside of the Android ecosystem.

Sprint 2: UI - 1

This sprint was centred around building the initial UI and functionality of the frontend application. The goal of this Sprint was to just have a basic working UI, allowing for working input elements on any forms required for document generation and comparison. The interface for the compiler was not considered at this stage as that was a bit more involved. I used this time to settle on the general theming palette of the application. I had already known from the beginning that I wanted to focus on having a dark theme but had not decided upon which accent colours to pick. I was able to use this part of the project to view components with different accents and such until I settled upon the light blue colour that appears in the final UI of the application.

I also used this time to build out demo versions of the components before I had settled on the actual background functionality of the component.

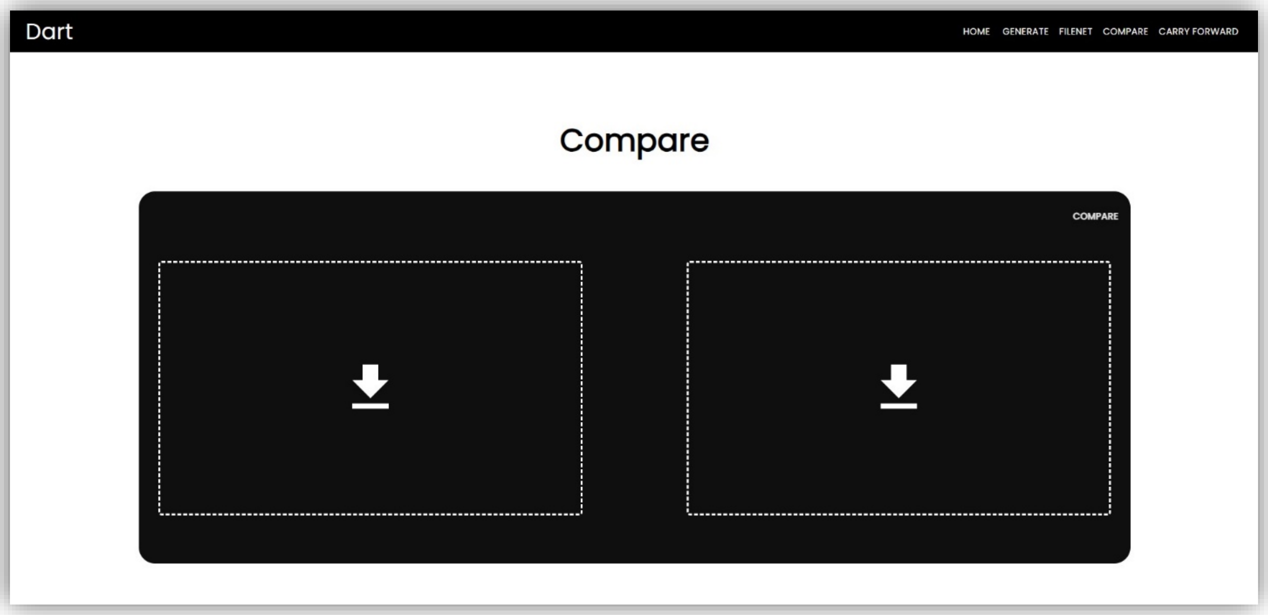


Figure 10.0 – Implementation, Early UI

Sprint 3: Compiler – 1

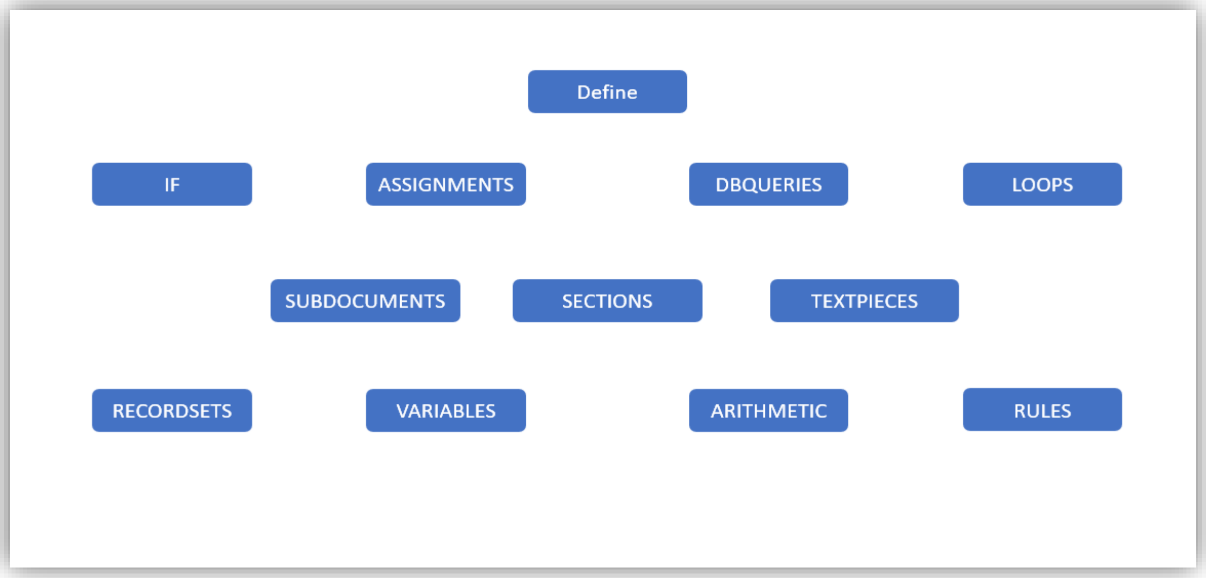


Figure 10.1 – Implementation, Syntax Map

Syntax Map

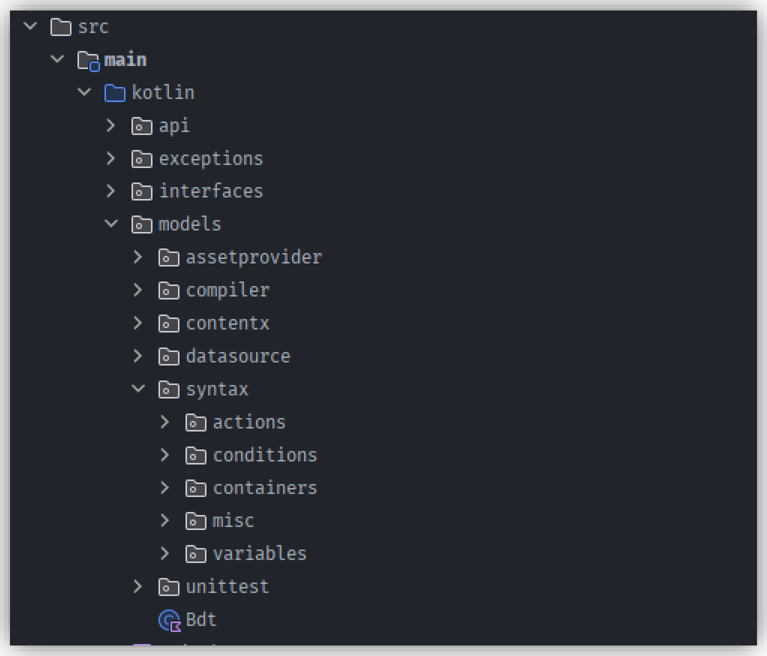
Sprint 3 was spend designing the mini compiler that would parse the XML based language the rules that make up an Xpression document. The image below highlights the different elements that can make up an individual document.

Designing this compiler was simply a case eyeballing the rule's XML to see if I could glean meaning from the element names, there was no documentation to reference but the elements are fairly self-explanatory. In times where I was unsure, I would take a guess and evaluate my programmes output against the output of the actual application to see if they matched.

One standout issue in this section was the actual execution model of the compiler. I will discuss this further in Compiler – 2 as it did not get fixed until them, but essentially the issue was centred around how I was traversing the event sequence of the compiler to roll back the program in order to perform looping operations. Originally, I relied upon an "event sequence", this event sequence was an array of every action that had been evaluated by the compiler. My strategy for loops was to check this event sequence for the Label that the GOTO of the Loop was looking to get too. Once the label was found in the event sequence, we I would rerun from there onward until I hit the loop again. This kind of worked, but the issue was that the event sequence was *every* action, so previous actions that had been spat into the event sequence would replay even if for that loop iteration they should not. At this stage in the project, I opted not to amend the execution model, instead I created a workaround whereby the event sequence elements that were to be included in the loop were rolled back up into unevaluated elements. This worked for now but was not an ideal solution.

Application Structure

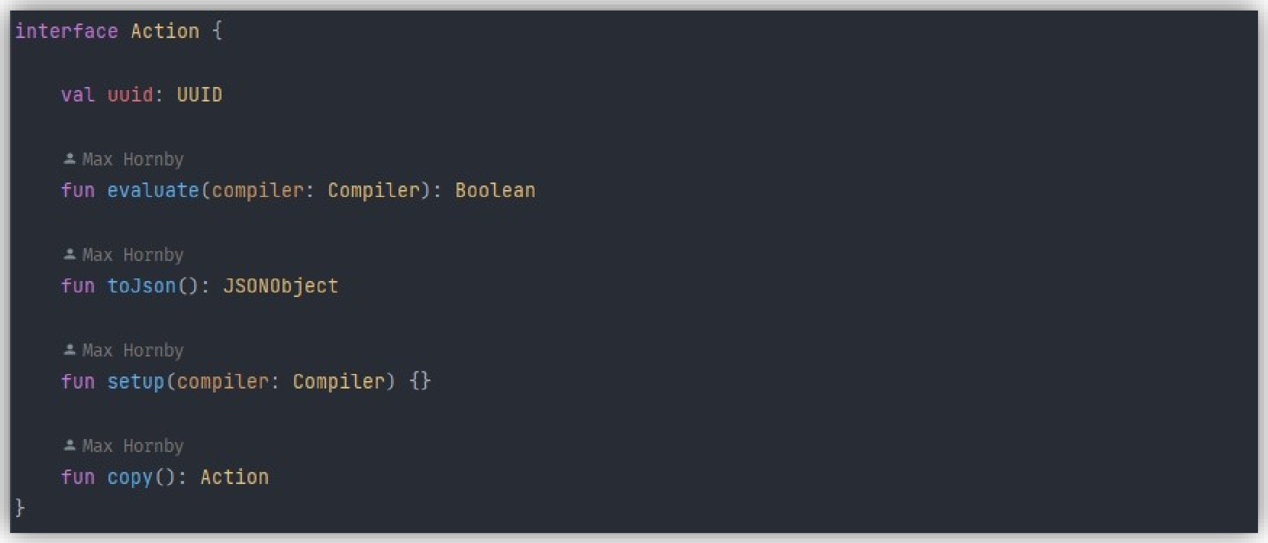
This image demonstrates the overall application structure of the compiler. As an application it mostly relies on the Konsume XML library to serialise the individual syntax elements into objects. Each object then has an evaluate() function. This function mostly just returns true if the object is evaluated, but will sometimes update some state that the compiler object is keeping track of.



```
src
  main
    kotlin
      api
      exceptions
      interfaces
      models
        assetprovider
        compiler
        contentx
        datasource
        syntax
          actions
          conditions
          containers
          misc
          variables
        unittest
        Bdt
```

Figure 10.2 – Implementation Compiler Structure

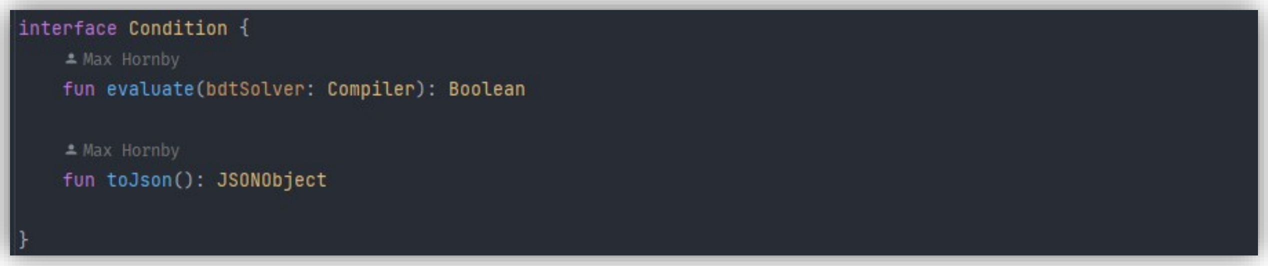
All of the syntax elements were made to implement and Actions interface, which keeps everything quite clean.



```
interface Action {

    val uuid: UUID

    Max Hornby
    fun evaluate(compiler: Compiler): Boolean

    Max Hornby
    fun toJson(): JSONObject

    Max Hornby
    fun setup(compiler: Compiler) {}

    Max Hornby
    fun copy(): Action
}
```

Figure 10.3 – Implementation, Action Interface

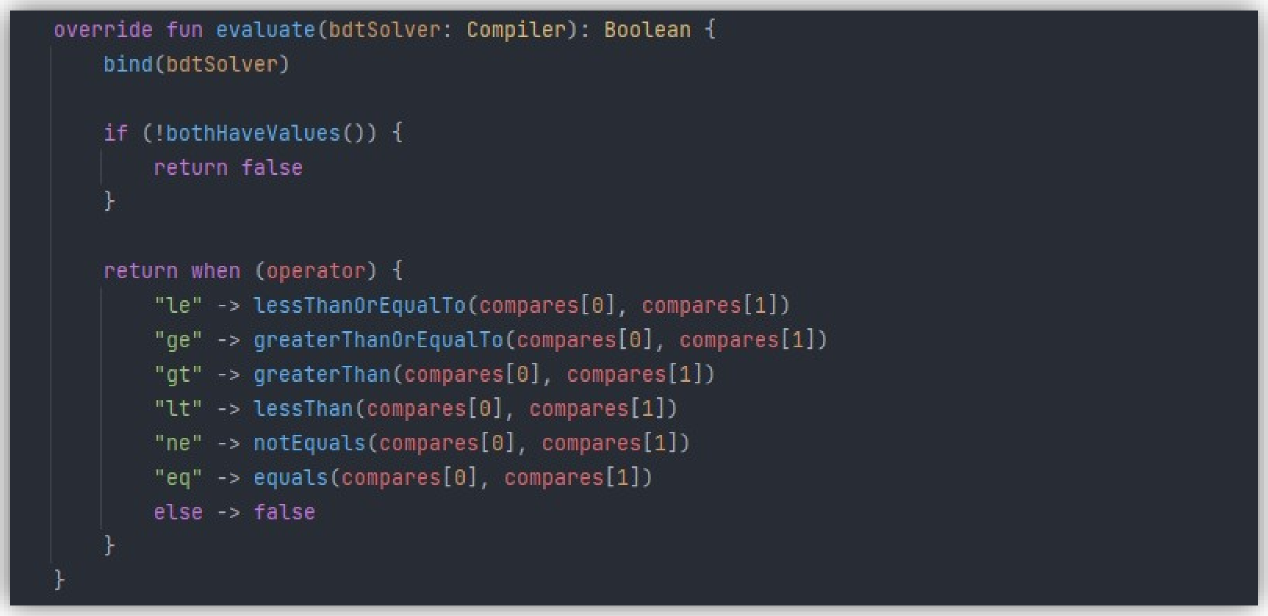
I utilised interfaces a lot to group functionality so as not to get bogged down by all of the different types of elements. Here is an interface for all the different types of conditions that could appear.



```
interface Condition {
    Max Hornby
    fun evaluate(bdtSolver: Compiler): Boolean

    Max Hornby
    fun toJson(): JSONObject

}
```

Figure 10.4 – Implementation, Condition Interface

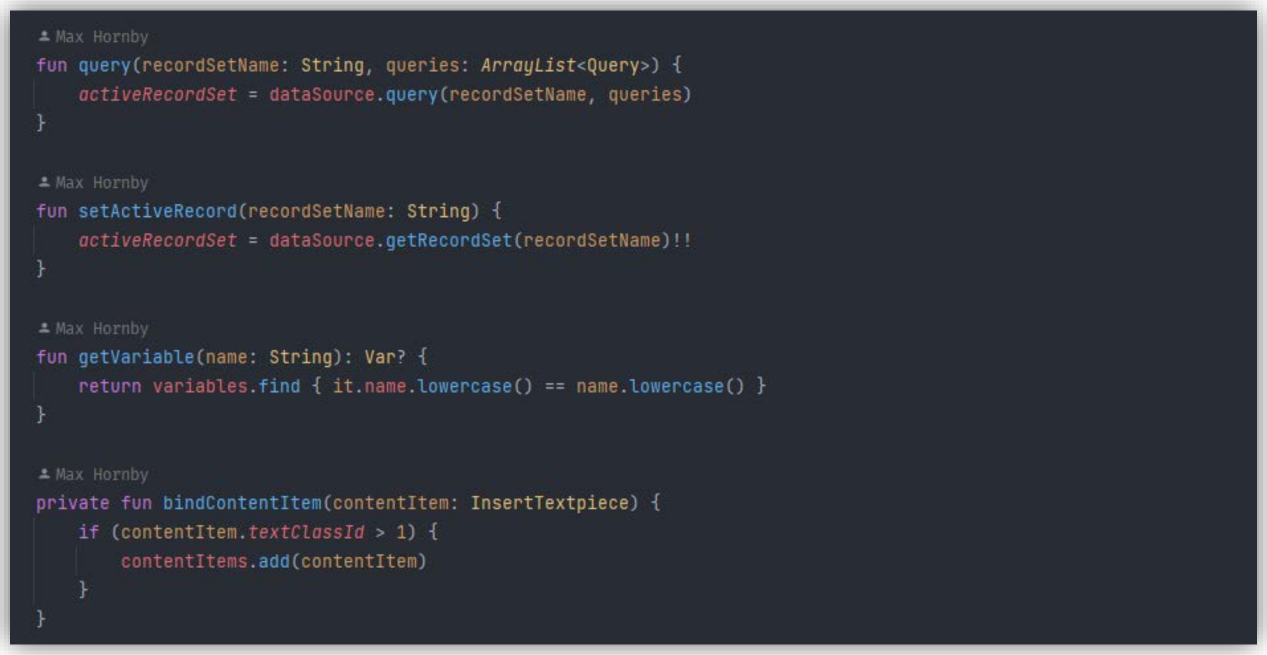
As can be seen, conditional checks simply return a Boolean value for when they end up passing or not. Here is an example of how a Comparison type condition evaluates itself. Just to reiterate a Comparison element is the XML syntax simply compares one variable or database fields value with another and is used for the conditional display of content.



```
override fun evaluate(bdtSolver: Compiler): Boolean {
    bind(bdtSolver)

    if (!bothHaveValues()) {
        return false
    }

    return when (operator) {
        "le" -> lessThanOrEqualTo(compares[0], compares[1])
        "ge" -> greaterThanOrEqualTo(compares[0], compares[1])
        "gt" -> greaterThan(compares[0], compares[1])
        "lt" -> lessThan(compares[0], compares[1])
        "ne" -> notEquals(compares[0], compares[1])
        "eq" -> equals(compares[0], compares[1])
        else -> false
    }
}
```

Figure 10.5 – Implementation, Conditional Evaluation

Compiler State
As mentioned the way in which the compiler works is that if first serialises all of the rules contained within the XML to an list of Actions and then traverses these actions evaluating them, the evaluate method of all of the Actions take the compiler as an argument and can manipulate state on the compiler if they need to. This is done in many cases, such as when the compiler encounters an element that contains a variable, the evaluate method ask the compiler what the value of the variable is supposed to be. Similarly, when the compiler encounters an element that needs to change or query the dataset, there are methods that the object representing the element can call to perform these functions.

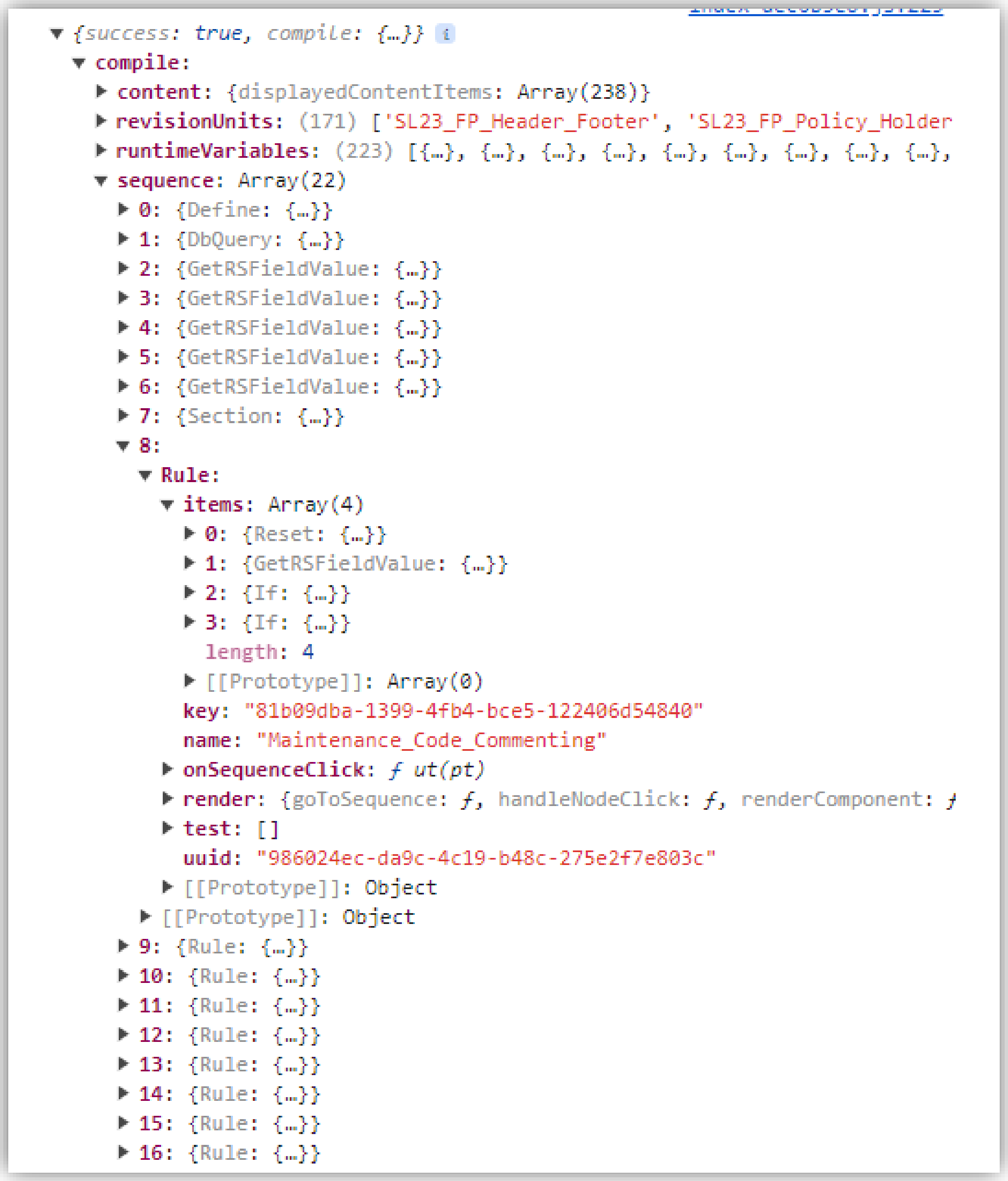


```
Max Hornby
fun query(recordSetName: String, queries: ArrayList<Query>) {
    activeRecordSet = dataSource.query(recordSetName, queries)
}

Max Hornby
fun setActiveRecord(recordSetName: String) {
    activeRecordSet = dataSource.getRecordSet(recordSetName)!!
}

Max Hornby
fun getVariable(name: String): Var? {
    return variables.find { it.name.lowercase() == name.lowercase() }
}

Max Hornby
private fun bindContentItem(contentItem: InsertTextpiece) {
    if (contentItem.textClassId > 1) {
        contentItems.add(contentItem)
    }
}
```

Figure 10.6 – Implementation, Compiler State

Sprint 4: UI – 2

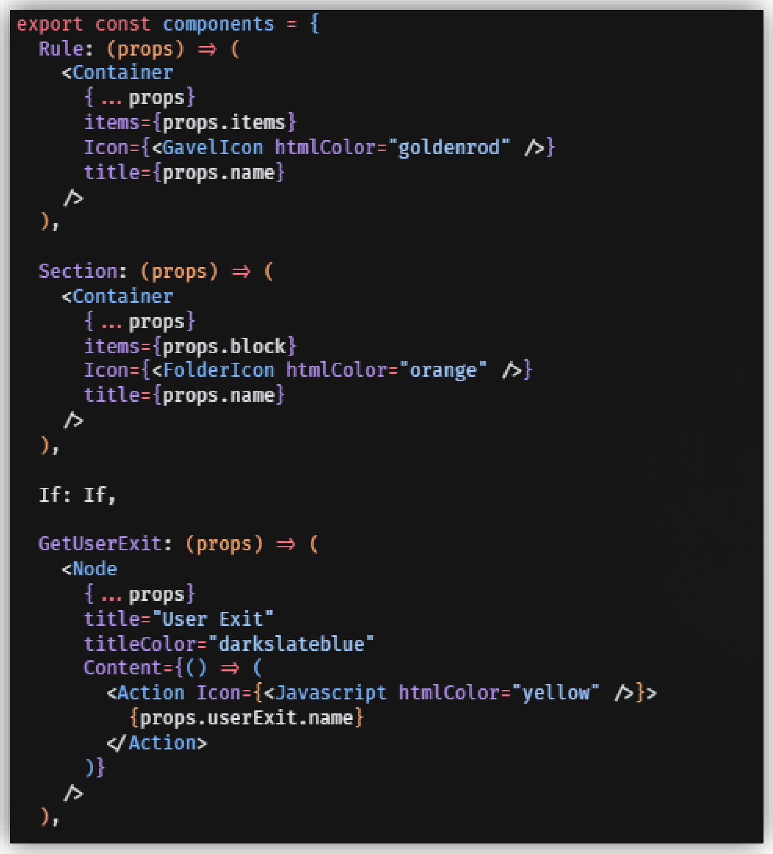
The majority of this portion of the project was spent creating the UI to represent the compiled rule that were being returned by the compiler. The evaluated sequence was being returned as an array of objects where the key was the object type and the value was the object properties.



```
▼ {success: true, compile: {…}} i
  ▼ compile:
    ▶ content: {displayedContentItems: Array(238)}
    ▶ revisionUnits: (171) ['SL23_FP_Header_Footer', 'SL23_FP_Policy_Holder
    ▶ runtimeVariables: (223) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
    ▼ sequence: Array(22)
      ▶ 0: {Define: {…}}
      ▶ 1: {DbQuery: {…}}
      ▶ 2: {GetRSFieldValue: {…}}
      ▶ 3: {GetRSFieldValue: {…}}
      ▶ 4: {GetRSFieldValue: {…}}
      ▶ 5: {GetRSFieldValue: {…}}
      ▶ 6: {GetRSFieldValue: {…}}
      ▶ 7: {Section: {…}}
      ▼ 8:
        ▼ Rule:
          ▼ items: Array(4)
            ▶ 0: {Reset: {…}}
            ▶ 1: {GetRSFieldValue: {…}}
            ▶ 2: {If: {…}}
            ▶ 3: {If: {…}}
              length: 4
            ▶ [[Prototype]]: Array(0)
            key: "81b09dba-1399-4fb4-bce5-122406d54840"
            name: "Maintenance_Code_Commenting"
          ▶ onSequenceClick: ƒ ut(pt)
          ▶ render: {goToSequence: ƒ, handleNodeClick: ƒ, renderComponent: ƒ
          ▶ test: []
            uuid: "986024ec-da9c-4c19-b48c-275e2f7e803c"
          ▶ [[Prototype]]: Object
        ▶ [[Prototype]]: Object
      ▶ 9: {Rule: {…}}
      ▶ 10: {Rule: {…}}
      ▶ 11: {Rule: {…}}
      ▶ 12: {Rule: {…}}
      ▶ 13: {Rule: {…}}
      ▶ 14: {Rule: {…}}
      ▶ 15: {Rule: {…}}
      ▶ 16: {Rule: {…}}
```

Figure 10.7 – Implementation, JSON Compilation Data

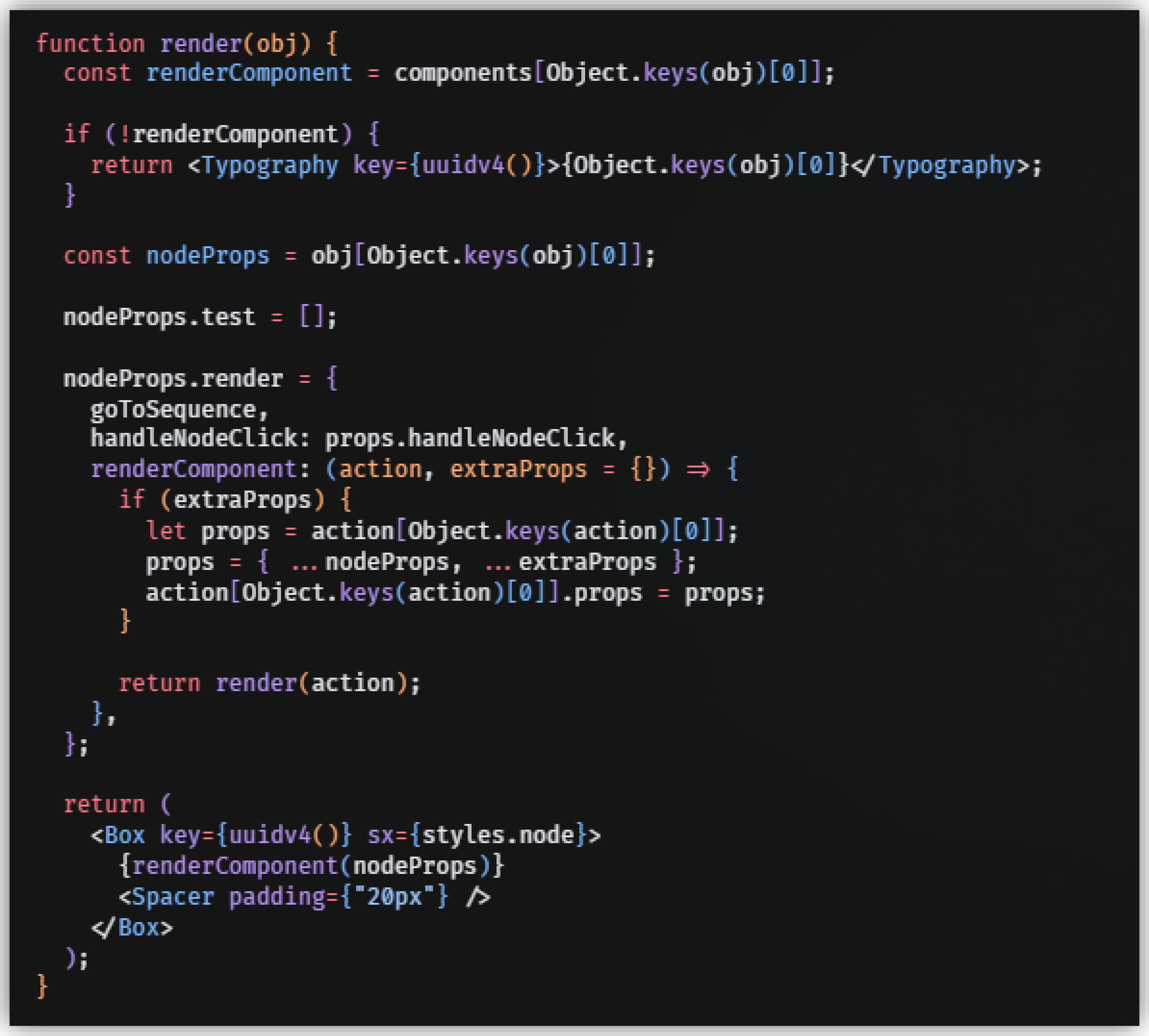
I had to come up with an unorthodox solution, whereby the key of the object dictated what function React would call the render it out. These components were mapped in the below manner.



```
export const components = {
  Rule: (props) => (
    <Container
      {...props}
      items={props.items}
      Icon={<GavelIcon htmlColor="goldenrod" />}
      title={props.name}
    />
  ),

  Section: (props) => (
    <Container
      {...props}
      items={props.block}
      Icon={<FolderIcon htmlColor="orange" />}
      title={props.name}
    />
  ),

  If: If,

  GetUserExit: (props) => (
    <Node
      {...props}
      title="User Exit"
      titleColor="darkslateblue"
      Content={() => (
        <Action Icon={<Javascript htmlColor="yellow" />}>
          {props.userExit.name}
        </Action>
      )}
    />
  ),
```

Figure 10.8 – Implementation, React Compilation Object Component

And were then rendered by the below method. The below method may be confusing because it is also adding a further render method into the props that are being passed to the component to be rendered. This duplicate render method is for any components that are actually containers, for example – the Section element is a container element that holds other actions as a list of items. Due the way I have chosen to render out the rules graph, I wanted it to be possible to click into a Section and only see the rules that are contained in that Section. To me, the easiest way to achieve this functionality was to give containers a render method they could call on their children.



```
function render(obj) {
  const renderComponent = components[Object.keys(obj)[0]];

  if (!renderComponent) {
    return <Typography key={uuidv4()}>{Object.keys(obj)[0]}</Typography>;
  }

  const nodeProps = obj[Object.keys(obj)[0]];

  nodeProps.test = [];

  nodeProps.render = {
    goToSequence,
    handleNodeClick: props.handleNodeClick,
    renderComponent: (action, extraProps = {}) => {
      if (extraProps) {
        let props = action[Object.keys(action)[0]];
        props = { ...nodeProps, ...extraProps };
        action[Object.keys(action)[0]].props = props;
      }

      return render(action);
    },
  };

  return (
    <Box key={uuidv4()} sx={styles.node}>
      {renderComponent(nodeProps)}
      <Spacer padding={"20px"} />
    </Box>
  );
}
```

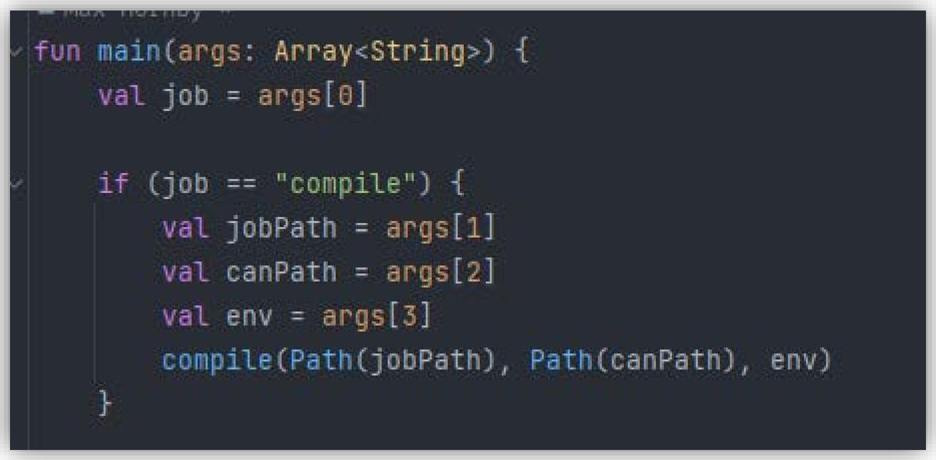
Figure 10.9 – Implementation, React Component Render Method

Sprint 5: Compiler – 2

This sprint was focused on more advanced parts of the compiler and also tidying up the structure of the program.

Kotlin – Node interop

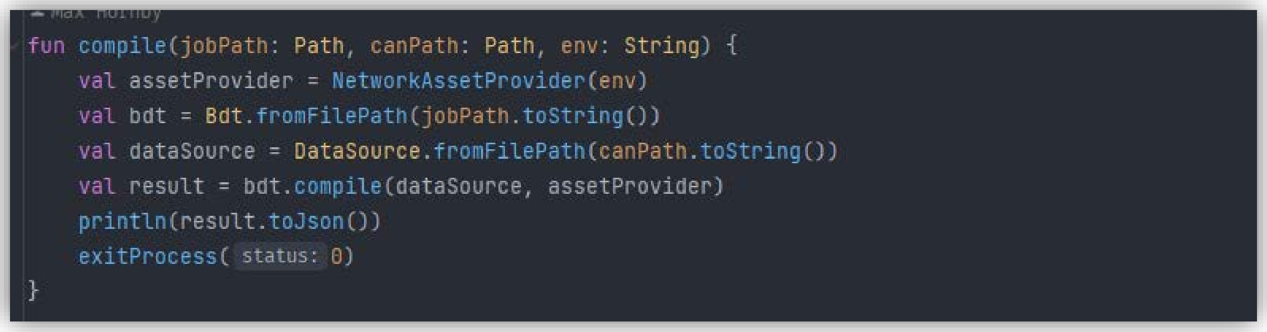
In order to allow this Kotlin compiler to be called by the Node backend in response to a request for compilation from the frontend, the Main portion had to be switched to allow the compiler to take some simple command line arguments



```kotlin
fun main(args: Array<String>) {
    val job = args[0]

    if (job == "compile") {
        val jobPath = args[1]
        val canPath = args[2]
        val env = args[3]
        compile(Path(jobPath), Path(canPath), env)
    }
```

Figure 10.10 – Implementation, Compiler Command Line Args

The way this worked in practice was that the Node application would write the requested document models XML rules to a file (jobPath), and also write the datasources XML data to a file (canPath) and finally pass in an argument for an environment variable, which would be used in the case of the compiler encountering elements that it needs to request assets for.



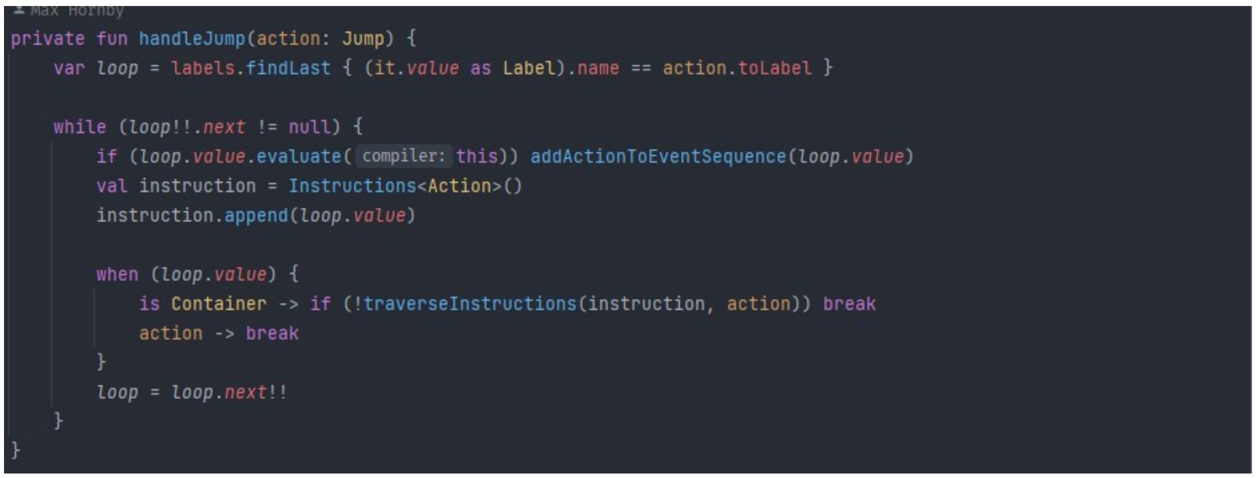
```kotlin
fun compile(jobPath: Path, canPath: Path, env: String) {
    val assetProvider = NetworkAssetProvider(env)
    val bdt = Bdt.fromFilePath(jobPath.toString())
    val dataSource = DataSource.fromFilePath(canPath.toString())
    val result = bdt.compile(dataSource, assetProvider)
    println(result.toJson())
    exitProcess( status: 0)
}
```

Figure 10.11 – Implementation, Compile Method

Compiler code execution model
I was able to take some time during this portion of the project to go back a iterate over how the compiler was actually executing the parse XML objects. Previously I spoke about how all elements were spat out into an event sequence and that this was used to recreate the code that was supposed to loop. This was an incorrect way to perform this operation. Up until this point I had not done much reading on how to actually implement what I was looking to make here – I had free handed on best estimations basis. After having tried many different ways to cut the loop sequence out of the main execution sequence of the code I began looking for guidance on the internet. I found the solution

here[16]. I had been using an array of objects up until this point, some objects could contain other objects and it had made traversing the sequence without starting from the beginning difficult.

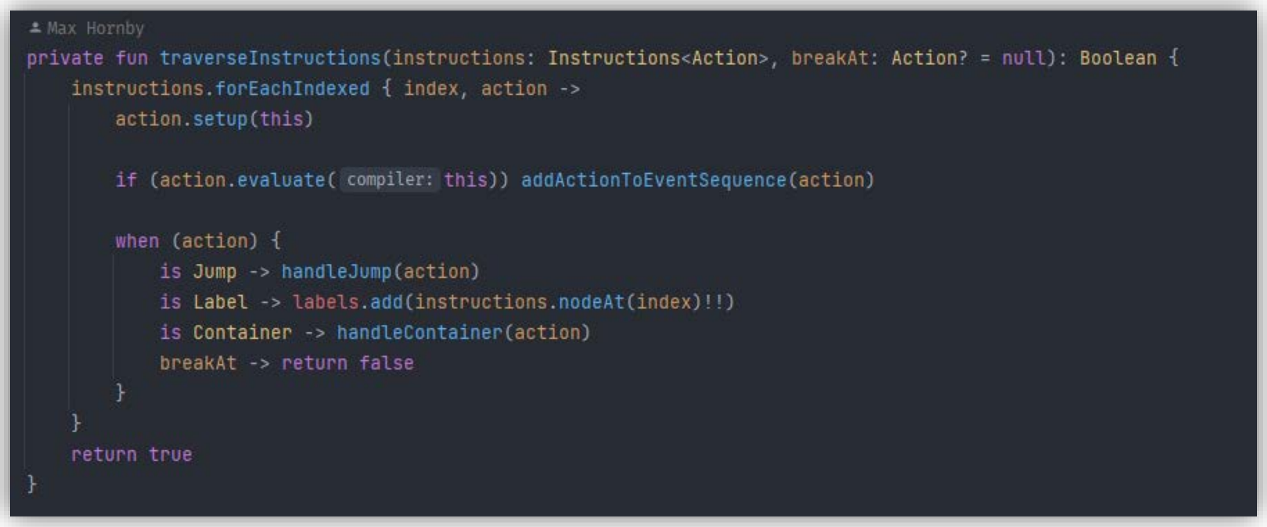
The linked list provided me with the right type of data structure that I needed to be able to pluck a node out a sequence and then just keep traversing to the next node until I found my exit point.



```
Max Hornby
private fun handleJump(action: Jump) {
    var loop = labels.findLast { (it.value as Label).name == action.toLabel }

    while (loop!!.next != null) {
        if (loop.value.evaluate( compiler: this)) addActionToEventSequence(loop.value)
        val instruction = Instructions<Action>()
        instruction.append(loop.value)

        when (loop.value) {
            is Container -> if (!traverseInstructions(instruction, action)) break
            action -> break
        }
        loop = loop.next!!
    }
}
```

Figure 10.12 – Implementation, Loop

As a result of reworking the data structure that the Action objects were contained in, I was able to rework the main program loop to be much cleaner.



```
Max Hornby
private fun traverseInstructions(instructions: Instructions<Action>, breakAt: Action? = null): Boolean {
    instructions.forEachIndexed { index, action ->
        action.setup(this)

        if (action.evaluate( compiler: this)) addActionToEventSequence(action)

        when (action) {
            is Jump -> handleJump(action)
            is Label -> labels.add(instructions.nodeAt(index)!!)
            is Container -> handleContainer(action)
            breakAt -> return false
        }
    }
    return true
}
```

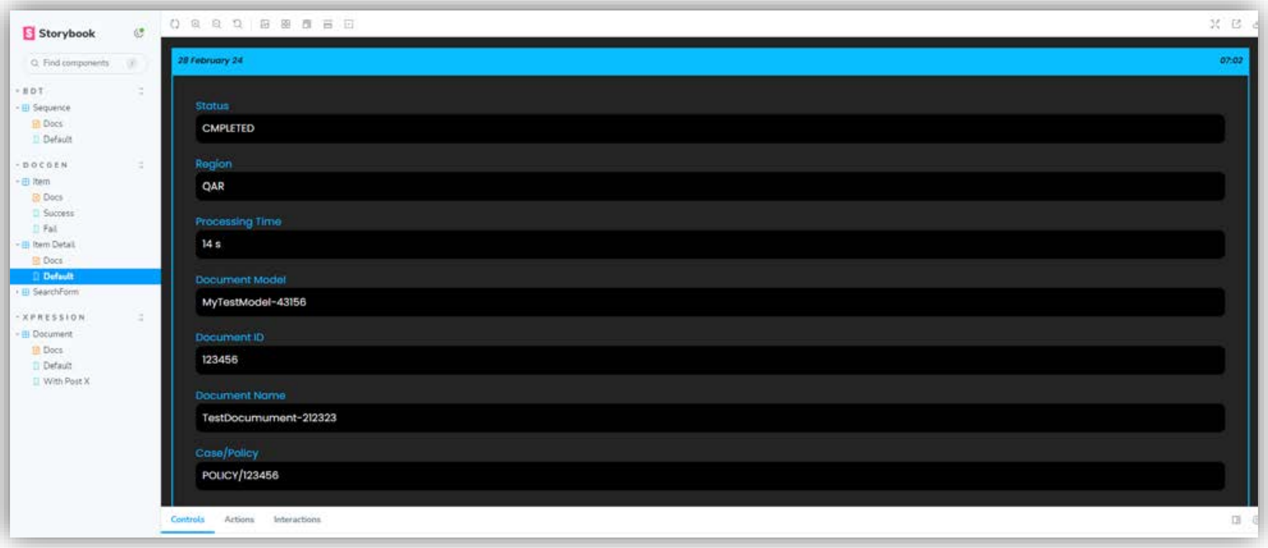
Figure 10.13 – Implementation, Traverse Instructions

[16] Kodeco. (2024). Data Structures & Algorithms in Kotlin. Available from: https://www.kodeco.com/books/data-structures-algorithms-in-kotlin/v1.0/chapters/3-linked-list [accessed March 1st 2024]

Sprint 6: Testing

Frontend

For testing the UI I just used storybook to view and develop some on the more intricate components in isolation. I also would run the program without a connection the the backend to ensure absence of data cases were signified correctly on the frontend.



28 February 24 07:02

Status
CMPLETED

Region
QAR

Processing Time
14 s

Document Model
MyTestModel-43156

Document ID
123456

Document Name
TestDocumument-212323

Case/Policy
POLICY/123456

Figure 10.14 – Implementation, Testing 1

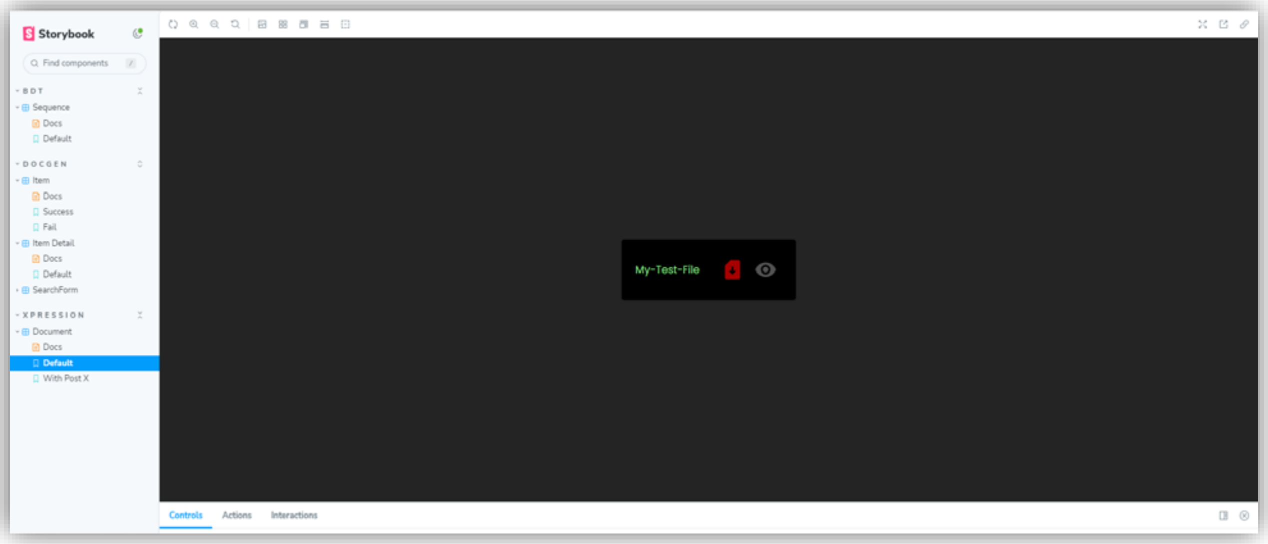
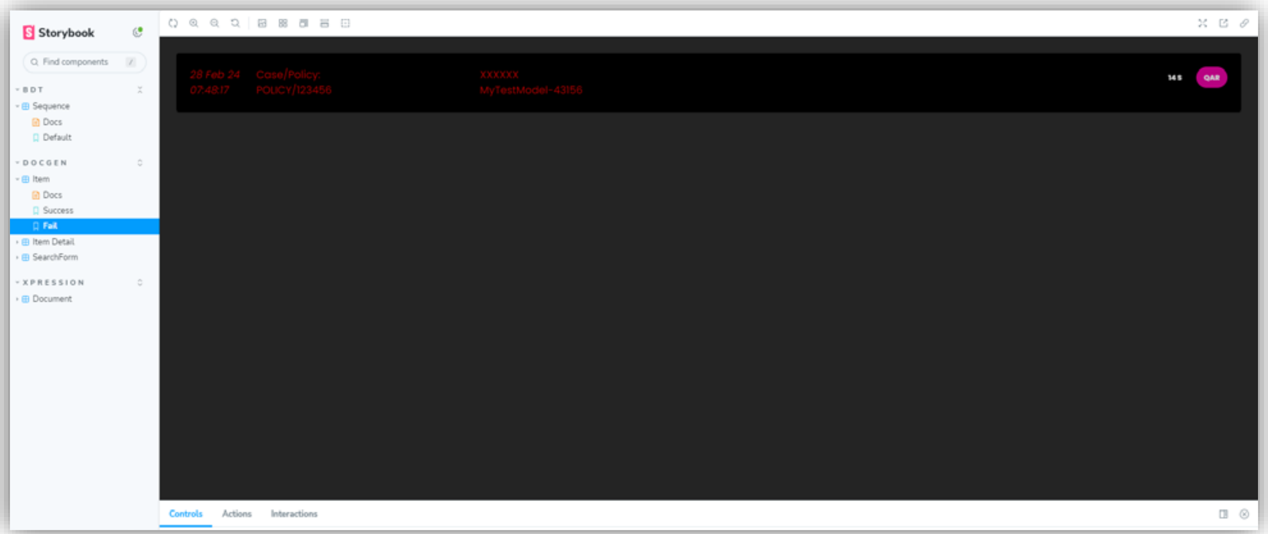


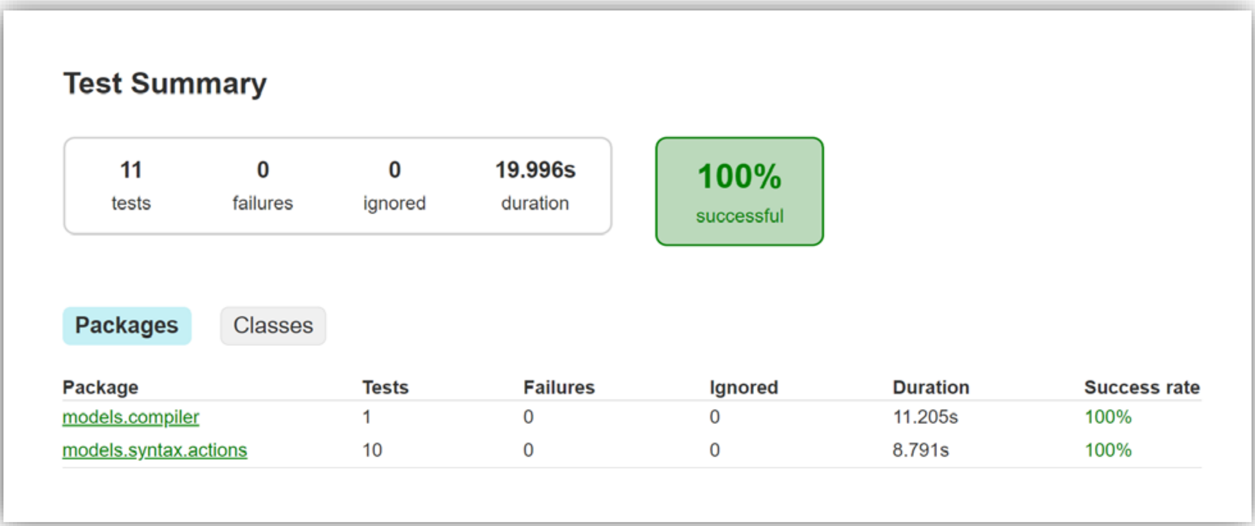
Figure 10.15 – Implementation, Testing 2



28 Feb 24 07:48:17 Case/Policy: POLICY/123456 XXXXXX MyTestModel-43156

Figure 10.16 – Implementation, Testing 3

Compiler
For the compiler, I used junit to write and run tests. The below test report is a sample of the tests involved.



Test Summary

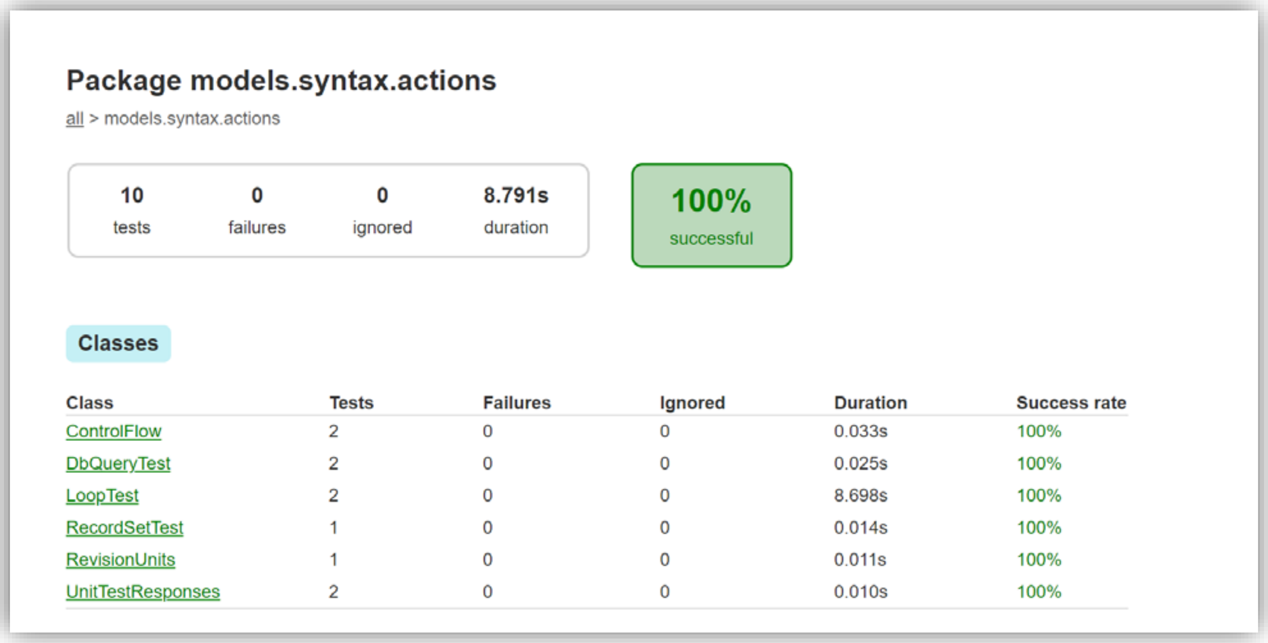
11 tests | 0 failures | 0 ignored | 19.996s duration | 100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
models.compiler	1	0	0	11.205s	100%
models.syntax.actions	10	0	0	8.791s	100%

Figure 10.17 – Implementation, Testing 4

The majority of tests were focused on the syntax elements of the compiler and whether the state of the compiler was updating correctly in response to the actions happening or whether the right content items were being output as a result of an evaluation.



Package models.syntax.actions

all > models.syntax.actions

10	0	0	8.791s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
ControlFlow	2	0	0	0.033s	100%
DbQueryTest	2	0	0	0.025s	100%
LoopTest	2	0	0	8.698s	100%
RecordSetTest	1	0	0	0.014s	100%
RevisionUnits	1	0	0	0.011s	100%
UnitTestResponses	2	0	0	0.010s	100%

Figure 10.17 – Implementation, Testing 5

I could and probably should of used more rigorous testing, but the above set seemed to cover the most involved scenarios. Testing in this manner helped isolate different portions of the expected behaviour in order to ensure they worked correctly.

Conclusion

Worthwhile endeavour
In my introduction I reference three key areas that a software project can be evaluated against when determining if it has had a positive impact, which we will analyse now.

"The exact impacts of any software project are innumerable and complex, so it's important to distil it to a few key questions:

What does this tool enable?

How many people (internally) are impacted?

What are the gains that result from delivering this tool?"

What does this tool enable?
DART enables the document generation and comparison features that developers require in order to carry out their functions. It also allows for business stakeholders to carry out the newly required document comparisons that are needed in order to ensure document output consistency across releases. DART will also enable a novel (to the company) feature of executing the rules from an XML outside of the Xpression application, this will allow for a foundation where unit tests can be built upon the compiler- which can be seen through the evaluation of the Revision Units of a document, minimising unexpected bugs in document output.

How many people are impacted
My whole team will be impacted by the project once it reaches a stable phase as they will move into using during their day-to-day. The comparison portions of the project are already in use by business teams as they compare document output from different regions to ensure any changes are expected and not the result of bugs.

What are the gains that result from delivering this tool
The gains are that it will help reduce the work placed on other senior members of my team, in that they will have to squash as many bugs in upper development regions and production that have come about as a result of developer error. As the compiler gets built out, more intricate unit tests will be provided for and as can already be seen, the issue of duplicate revision units crashing a document will no longer cease to be an issue.

In terms of general productivity gains, developers may be able to work slightly faster in that, the application is slightly faster to use than the other offerings due to its straightforward UI and the fact it's a web-based application. The benefits in this area will not be as much as the stability that can be gained, but one of the major benefits of the project is that I am the main developer and I actively work within the document output team. This means that as more needs are discovered over time, I will have the opportunity to add them to the DART application.

Learnings

Simple is good

Through the completion of this project I have learned that it is beneficial to seek feedback from as many people as possible when developing, especially when it comes to UI. To me, when I thought about UI, I would focus on the technicalities of it – ie. Does this button work? does it redirect me to where I am looking to go? I would not be too concerned with what it looks like or even so much with its position on the screen. Up until this project, any application I had built was ultimately built, used and aimed at – myself.

But I have learned that when developing for other users, a simple UI is key. If anything on the screen is not forthright in function, it can lead to a poor user experience. One of the things that was really beneficial in this project, was asking someone on the team to use the application, but not providing any commentary or direction on what they were looking to do. Early on, it was apparent that some of the functions of the application were a bit opaque or unclear and relied too much on text to explain function, so I tried to ensure that any type of functionality was accompanied by an animated button, or some form of simple icon, preferring these over actual text.

This had a good impact on the overall perceived simplicity of the application and in the final showcase of the application internally, the UI was specifically highlighted as being pleasant.

Scope Creep

Once I had a basic implementation of the compiler working, my mind was inundated with potential for other features to add into the application. However, this early on version of the compiler was not fully functional, and relied upon a poor execution model for looping conditions – so would not have been a good foundation for future features. After unsuccessfully attempting to implement an early form of version control for the rules, I scaled back and decided to not attempt to add any extra features to the overall application and just focus on the compiler implementation as is. This allowed me to have the time to go over the implementation of the execution model of the compiler close to end of the project build a much better foundation that I can extend when I am ready.

Time Management

During the course of the project I feel like I have improved upon my time management skills. Early on, I found it difficult to juggle my tasks in work, another python project I was working on and this project. I was context switching a lot during the day between fairly different problem sets, and progress on this project was slowing down. It helped me a lot to avoid regular context switching by making a day-to-day schedule, so instead of switching from one programming task to another during the day, I would have set days for different tasks. After making this change, progress on this project picked up again and the mental draw of the constant context switching was eliminated.

General Software Development Experience

As a consequence of my work on this project, I have become involved in many different things that I would have considered to be *"above my station"* in terms of my employment experience. I joined the company in a full time position as an Engineer Trainee. Once my work was showcased, I began to become involved in higher up discussions between seniors about other projects. I am also happy to have been included in a meeting about product offerings with an external vendor of a tool that I use quite often in work. My work on this project and others, using the skills I gained form the overall HDip lead to me being promoted to an Engineer at the beginning of this year.

Future Development

DART is still in active development. In regards to the compiler, I am working on extending its functionality to allow for more opportunities to test documents such as shadowing variables. I have also begun exploring conversion of the document rules to alternative formats – One very important issue facing our team is that ultimately, should we wish to move to another application like Xpression, we would be unable to take the documents with us as the rules and conditions have been set up in the Xpression application. However, because the compiler can understand and parse the rule, it should be able to help convert the rules to another application format.

Regarding the frontend of the application, this will also be updated as new features are added. One area that I did not get to explore during the project was the more intricate animations and screen transitions. I would very much like to make these additions to the application in the near future. I would also like to create tutorial overlays for some of the elements to further improve the legibility of individual features for users outside of my immediate team.

Bibliography

Website
Jimmy Shi. (2021). Internal Tools: A Cost Benefit Analysis. Available from: https://www.internal.io/blog/internal-tools-cost-benefit-analysis [accessed 07 February 2024]

Python Software Foundation. (2024). History and License. Available from: https://docs.python.org/3/license.html [accessed 6 February 2024]

Python Software Foundation. (2024). What is Python? Executive Summary. Available from: https://www.python.org/doc/essays/blurb/ [accessed 6 February 2024]

Google. (2024). Release History. Available from: https://go.dev/doc/devel/release#go1 [accessed 6 February 2024]

Google. (2024). Case Studies. Available from: https://go.dev/solutions/case-studies [accessed 6 February 2024]

JetBrains. (2024). FAQ. Available from: https://kotlinlang.org/docs/faq.html [accessed 6 February 2024]

Stack Overflow. (2024). Developer Survey 2023. Available from: https://survey.stackoverflow.co/2023/#most-popular-technologies-language [accessed 6 February 2024]

Atlassian. (2024). Agile vs Waterfall Project Management. Available from: https://www.atlassian.com/agile/project-management/project-management-intro [accessed 10 February 2024]

Max Rehkopf. (2024). Stories, epics, and initiatives. Available from https://www.atlassian.com/agile/project-management/epics-stories-themes [accessed 10 February 2024]

Atlassian. (2024). Introduction to Jira family. Available from https://www.atlassian.com/software/jira/guides/more/jira-family#what-is-the-jira-family [accessed 10 February 2024]

Kodeco. (2024). Data Structures & Algorithms in Kotlin. Available from: https://www.kodeco.com/books/data-structures-algorithms-in-kotlin/v1.0/chapters/3-linked-list [accessed March 1st 2024]

Rich Harris (2018). The Virtual Dom is Pure Overhead. Available from: https://svelte.dev/blog/virtual-dom-is-pure-overhead [accessed 26th February 2024]

Github: svelte/svelteJs (2020). Issue #4455. Available from: https://github.com/sveltejs/svelte/issues/4455 [accessed 3rd March 2024]