External Cognizant Project

Elizabeth Coady – Software Developer Level 4

Table of Contents

Introduction

I have been involved in a second project, a financial client of Cognizant's. I was brought on to this project when I first started working for Cognizant in April, and I stayed on this project until moving to Connect in July. The 'Connect' project is talked about in the document 'Connect Portfolio Submission'. As mentioned there, I was in a pod of 9, with 1 more on shore pod and 4 off shore pods, totalling around 60 people. My pod consisted of myself, 4 other full stack developers, one BA, two SDETs and one scrum master. This was a first experience of working on the job straight off the 3 month boot camp, so it was exciting and also nerve wracking to begin on a real project for one of Cognizant's clients.

The 'Connect' project came to an end in March 2020 and therefore I re-joined back to this project in the same month. Because of the pandemic and restrictions arising in this time, we have all had to work from home. This has been a challenge for working on a large team again, not even having met most of my colleagues in person. The convenience of being able to contact someone with ease if there are any problems was more challenging now, making everyone having to adapt and become more pro-active quickly, as the client expectations of deadline remained the same.

When adapting to these new conditions, I had to make sure I considered the health and safety aspects of my mental wellbeing so I knew it was important to separate our work life from our personal life. I made sure I would stop working at a reasonable time and take regular breaks away from my work laptop to relax and not be too overwhelmed, making sure to create a clear disconnect between the two. This has worked extremely well as my mental health is doing great and my performance at work has been able to be maintained at a good level.

Project Overview

This project is mainly a financial project. It will be used my employees of the project owner and external companies as well. It has normal admin systems such as logging in / signing out, email notifications and data submission and retrieval.

Stakeholders

The team consisted of 9 main pods; 3 onshore and 6 offshore. Additional we also had a shaping team and infra team. Each main pod had 1 scrum master, 4 full stack developers, 1 full stack develop / architect, 1 tester and 1 UI designer. Other team members outside the main pods had main system architects, business analysts, deployment managers / infra team.

Concept, Design and User Story creation

The user stories were created by the business analysts and refined into tasks by the scrum masters.

Tools used

The tech stack and tools chosen by Cognizant and the client mainly consisted of: Java 8 (Spring), Drools, Scala, Angular, AWS (ECS, S3, Parameter Store, Lambda), Jenkins, Gitlab and more.

We used AWS services Elastic Container Service (ECS) for deploying services, S3 bucket for storing data, documentation and deploying the frontend, Parameter Store for storing environment variables and Lambda to trigger different services implemented, such as automatic email notifications. We used Jenkins for deployment automation. As we had different development and testing environments, we could control when we deployed each service to each environment, as well as checking whether the builds were successful or not. We used Gitlab to manage all repositories for version control and for reviewing merge requests from team members.

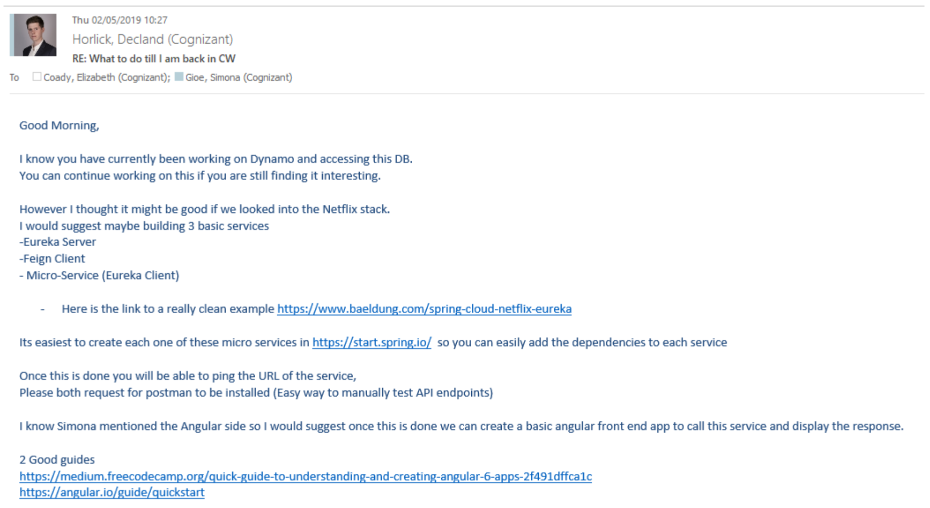
As the project is very big and has over 50 developers, microservices were chosen to be used. This allowed only a few developers at a time to work on each service so development was a lot easier and clearer.

To manage the tasks, we used the Azure DevOps service to manage tickets across the team. We were then able to see exactly which team member had which task and the progress of these.

I used the IDE IntelliJ to code the Java + Spring backend and Visual Studio Code to code the Angular frontend, the same as I have used in the 'Connect' project.

Beginning to Develop

As I was a new starter and inexperienced developer, I was given the tech stack and before being assigned a task on the project, to do some research on the tools and programmes used. Shown in Figure 1 is an email from my manager and fellow pod member on what to look at to prepare for the project, as I had no experience in the AWS service DynamoDB and cloud services in general, I looked into these first. I later looked into the Angular framework as although I was familiar with vanilla JavaScript, I had not used a JS framework before and knew I would have some frontend tasks down the line.



Thu 02/05/2019 10:27
Horlick, Decland (Cognizant)
RE: What to do till I am back in CW
To Coady, Elizabeth (Cognizant); Gioe, Simona (Cognizant)

Good Morning,

I know you have currently been working on Dynamo and accessing this DB.
You can continue working on this if you are still finding it interesting.

However I thought it might be good if we looked into the Netflix stack.
I would suggest maybe building 3 basic services
-Eureka Server
-Feign Client
- Micro-Service (Eureka Client)

- Here is the link to a really clean example https://www.baeldung.com/spring-cloud-netflix-eureka

Its easiest to create each one of these micro services in https://start.spring.io/ so you can easily add the dependencies to each service

Once this is done you will be able to ping the URL of the service,
Please both request for postman to be installed (Easy way to manually test API endpoints)

I know Simona mentioned the Angular side so I would suggest once this is done we can create a basic angular front end app to call this service and display the response.

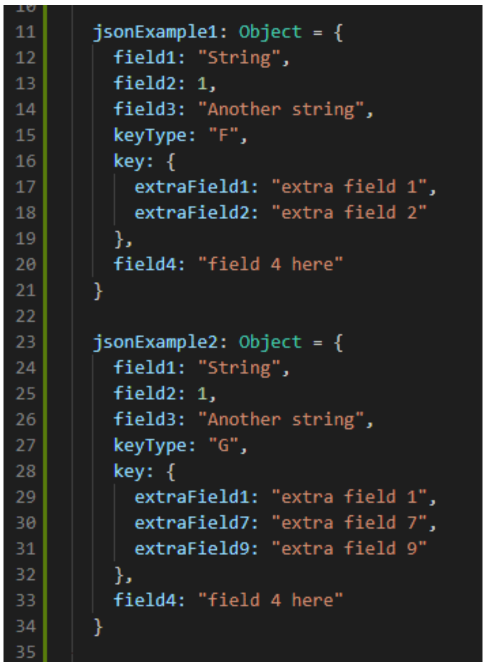
2 Good guides
https://medium.freecodecamp.org/quick-guide-to-understanding-and-creating-angular-6-apps-2f491dffca1c
https://angular.io/guide/quickstart

Figure 1 - Email sent from my manager at the time of what to look at the beginning of the project start

Developing

As this project is for an external client and we have an NDA, I cannot share screenshots of any code or tickets from the project itself. I have replicated the code I have written in a local project, changing names where appropriate.

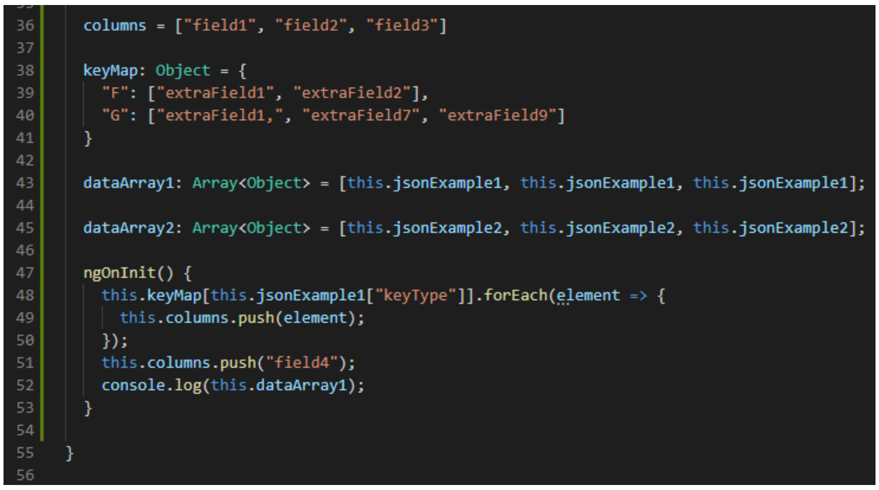
Frontend



```
  jsonExample1: Object = {
    field1: "String",
    field2: 1,
    field3: "Another string",
    keyType: "F",
    key: {
      extraField1: "extra field 1",
      extraField2: "extra field 2"
    },
    field4: "field 4 here"
  }

  jsonExample2: Object = {
    field1: "String",
    field2: 1,
    field3: "Another string",
    keyType: "G",
    key: {
      extraField1: "extra field 1",
      extraField7: "extra field 7",
      extraField9: "extra field 9"
    },
    field4: "field 4 here"
  }

```

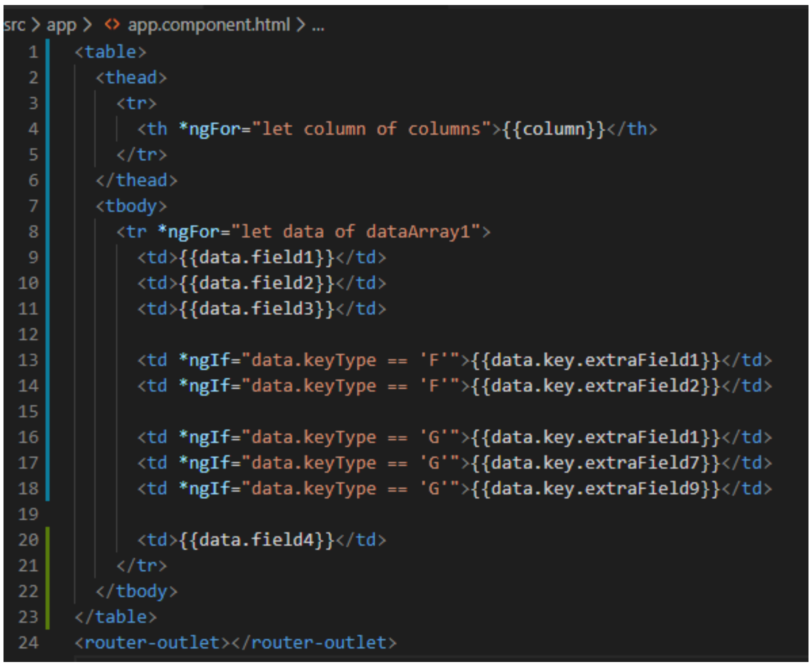
Figure 2 - JSON object examples of data from backend

For my first ticket in the project, I was assigned a frontend task. This task consisted mainly of being able to dynamically sort a table depending on the object that was sent. In Figure 2, we have 2 examples of the JSON object I have written which were mocked from the backend. Looking at lines 15 & 27, the requirements would be depending on the value of this property. If the field 'keyType' is equal to 'F', the 'key' field on line 16 will have different extra fields. We can see a contrast on line 28 with different fields. The table headings we required were the kay values from the object so 'field1', 'field2', 'field3', the extra key fields and 'field4' and the rows would be the corresponding values.



```
  columns = ["field1", "field2", "field3"]

  keyMap: Object = {
    "F": ["extraField1", "extraField2"],
    "G": ["extraField1,", "extraField7", "extraField9"]
  }

  dataArray1: Array<Object> = [this.jsonExample1, this.jsonExample1, this.jsonExample1];

  dataArray2: Array<Object> = [this.jsonExample2, this.jsonExample2, this.jsonExample2];

  ngOnInit() {
    this.keyMap[this.jsonExample1["keyType"]].forEach(element => {
      this.columns.push(element);
    });
    this.columns.push("field4");
    console.log(this.dataArray1);
  }

}
```

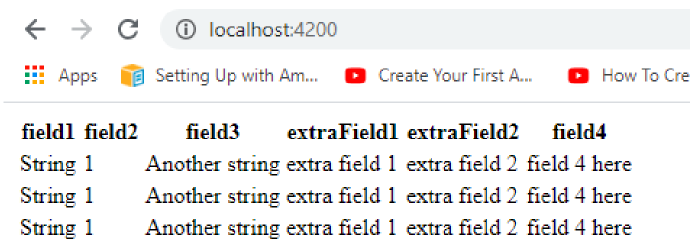
Figure 3 - Typescript code showing method to dynamically load column names



src > app > app.component.html > ...

```
<table>
  <thead>
    <tr>
      <th *ngFor="let column of columns">{{column}}</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let data of dataArray1">
      <td>{{data.field1}}</td>
      <td>{{data.field2}}</td>
      <td>{{data.field3}}</td>

      <td *ngIf="data.keyType == 'F'">{{data.key.extraField1}}</td>
      <td *ngIf="data.keyType == 'F'">{{data.key.extraField2}}</td>

      <td *ngIf="data.keyType == 'G'">{{data.key.extraField1}}</td>
      <td *ngIf="data.keyType == 'G'">{{data.key.extraField7}}</td>
      <td *ngIf="data.keyType == 'G'">{{data.key.extraField9}}</td>

      <td>{{data.field4}}</td>
    </tr>
  </tbody>
</table>
<router-outlet></router-outlet>
```

Figure 4 - HTML code showing the tables of data



field1	field2	field3	extraField1	extraField2	field4
String 1		Another string	extra field 1	extra field 2	field 4 here
String 1		Another string	extra field 1	extra field 2	field 4 here
String 1		Another string	extra field 1	extra field 2	field 4 here

Figure 5 - Rendering of tables code

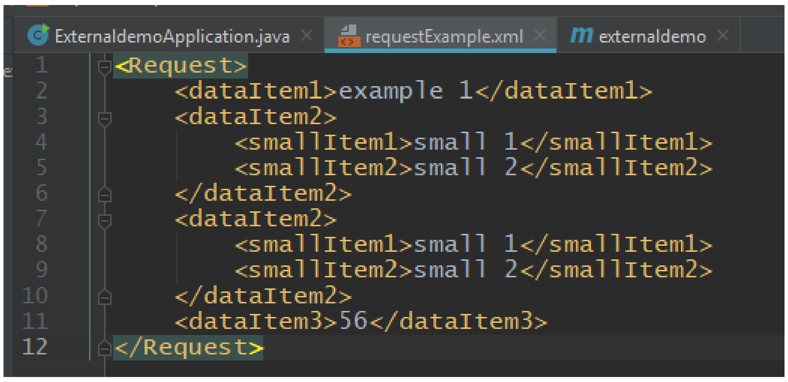
In Figure 3, I wrote the Angular Typescript code to create and array of the desired columns. In this case, we are using the 'jsonExample1' in Figure 2 to load the columns. On line 38 in Figure 3, I have created a JSON object to map the correct needed columns. On line 48, we take the 'keyType' field from the data and get the corresponding array of extra fields, then pushing them into our 'columns' array on line 49. Figure 4 shows the HTML code I have written corresponding to the typescript, looping through our 'columns' array on line 4 and looping through our 'dataArray1' on line 8 to get the correct data. On lines 13-14 and 16-18, we have conditional statements to only show the correct fields depending on our 'keyType'. Figure 5 is the output UI when it's compiled, showing that all the fields and columns are in their expected places.

For this being my first task, I thought methodically about how to approach and develop a solution and believe I wrote clear and concise code, demonstrating what I had learnt in the boot camp course a few months prior. I was then able to demonstrate the code I had written to the client myself, which went down very well with the client and my fellow colleagues.

Backend

Another feature we wanted to achieve in the project was a feature to be able to convert an XML request to a JSON file and to a Java Object. We wanted to do this as we had a user story about clients wanting to have an option of submitting data in an XML format rather than online, which I was tasked with undertaking.

After a lot of research and consulting with other members of the team, we decided using the JAXB and Jackson libraries for conversion would be the best approach.



```xml
<Request>
    <dataItem1>example 1</dataItem1>
    <dataItem2>
        <smallItem1>small 1</smallItem1>
        <smallItem2>small 2</smallItem2>
    </dataItem2>
    <dataItem2>
        <smallItem1>small 1</smallItem1>
        <smallItem2>small 2</smallItem2>
    </dataItem2>
    <dataItem3>56</dataItem3>
</Request>
```

Figure 6 - Example XML file request

Shown in Figure 6 is an example of one of our requests we would receiving in the service. We will need to convert this into an XSD schema. IntelliJ has an in built utility to generate an xsd file from an xml file (Figure 7), so I used this. This would make the process quicker and the most logical option to choose.

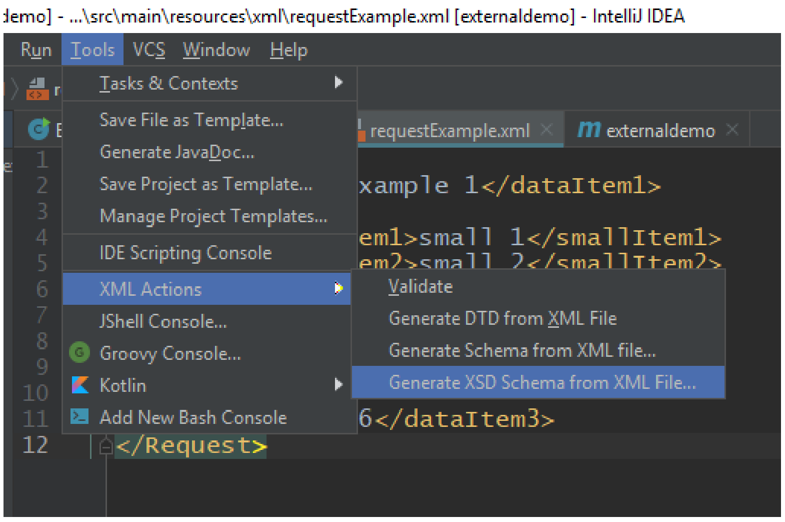
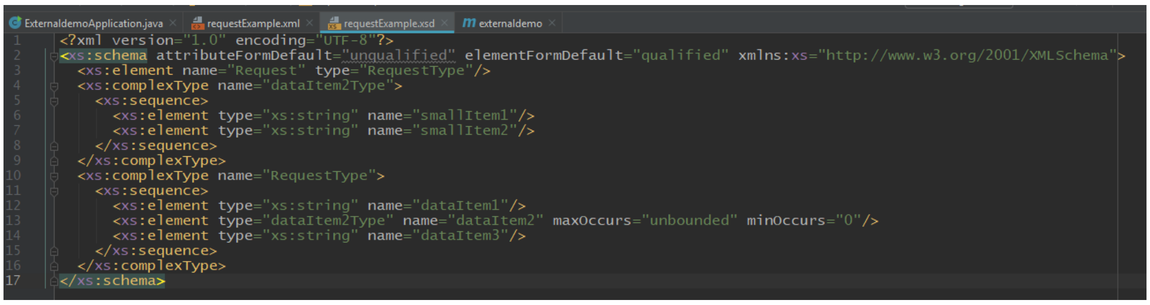

demo] - ...\src\main\resources\xml\requestExample.xml [externaldemo] - IntelliJ IDEA
Run
Tools
VCS
Window
Help
Tasks & Contexts
Save File as Template...
Generate JavaDoc...
Save Project as Template...
Manage Project Templates...
IDE Scripting Console
XML Actions
JShell Console...
Groovy Console...
Kotlin
Add New Bash Console
Validate
Generate DTD from XML File
Generate Schema from XML file...
Generate XSD Schema from XML File...
requestExample.xml
externaldemo

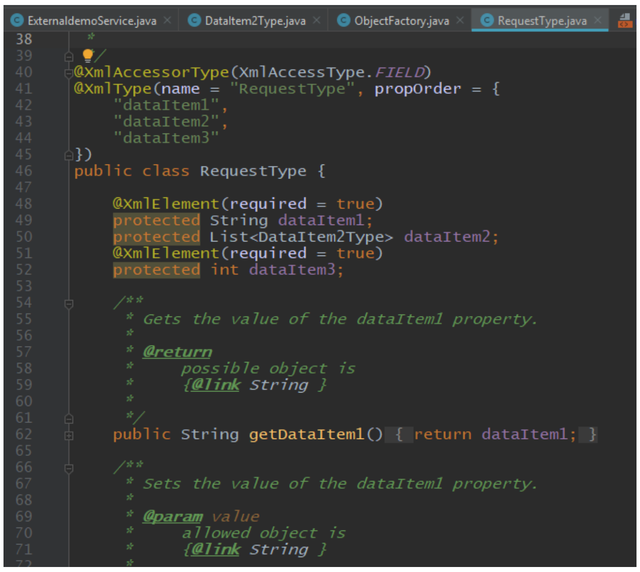
Figure 7 - IntelliJ's XSD Schema generation option



```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Request" type="RequestType"/>
  <xs:complexType name="dataItem2Type">
    <xs:sequence>
      <xs:element type="xs:string" name="smallItem1"/>
      <xs:element type="xs:string" name="smallItem2"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="RequestType">
    <xs:sequence>
      <xs:element type="xs:string" name="dataItem1"/>
      <xs:element type="dataItem2Type" name="dataItem2" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element type="xs:string" name="dataItem3"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 8 - XSD Schema generated from example XML

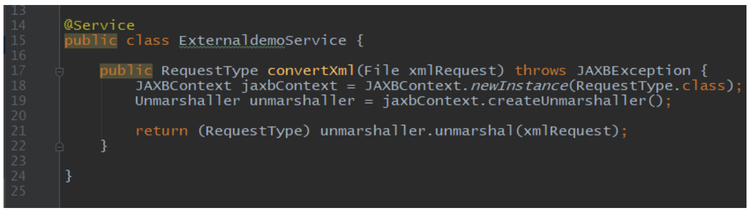
We can see in Figure 8 the generated Schema. Even though this saves a lot of time, I needed to make sure all fields were correct so when we use this to generate Java objects, we won't have any type mismatches. I found on line 14, the schema has not recognised that the field 'dataItem3' should be of 'int' type, so I corrected this.



```java
     *
     */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "RequestType", propOrder = {
    "dataItem1",
    "dataItem2",
    "dataItem3"
})
public class RequestType {

    @XmlElement(required = true)
    protected String dataItem1;
    protected List<DataItem2Type> dataItem2;
    @XmlElement(required = true)
    protected int dataItem3;

    /**
     * Gets the value of the dataItem1 property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getDataItem1() { return dataItem1; }

    /**
     * Sets the value of the dataItem1 property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
```

Figure 9 - Java Object of RequestType generated

Using another one of IntelliJ's plugins, I was generated the Java classes. We now had the POJOs to be able to map the XML, shown in in Figure 9.



```java
@Service
public class ExternaldemoService {

    public RequestType convertXml(File xmlRequest) throws JAXBException {
        JAXBContext jaxbContext = JAXBContext.newInstance(RequestType.class);
        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();

        return (RequestType) unmarshaller.unmarshal(xmlRequest);
    }

}
```

Figure 10 - Method to convert XML to RequestType Java object

In Figure 10, I have written a method to complete the flow of conversion. The method takes a file parameter to pass the XML file, then it uses the JAXB library to 'unmarshall' (unwrap, in a sense) the contents and map it to 'RequestType' java object we had generated.

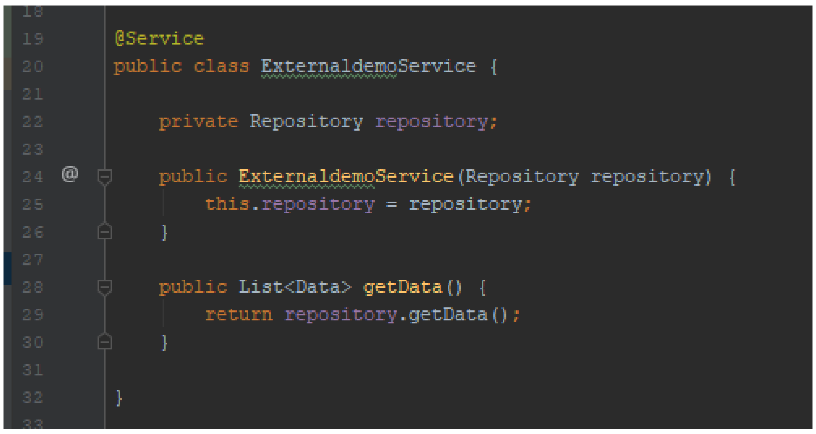
Even though I haven't written myself that much code for this feature, I have completed the task in the most decidedly efficient and cleanest way, as this is one of the most important parts about being a developer.

Another feature I developed is an email template production.

The email feature would be a daily automatic email sent to a selected list of people detailing the system processes that occurred in the previous 24 hours.

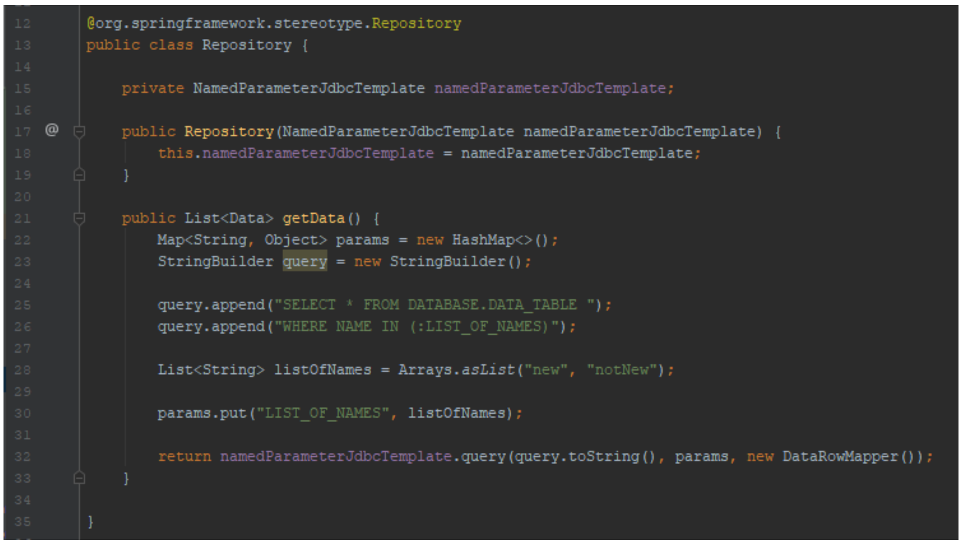
This was going to be a small micro service, so I was tasked with developing the whole service by myself, with support and review from my team if needed. It felt really good knowing that I had the trust of my scrum master and colleagues to be able to take ownership of this feature and be able to have the ability to choose how I wanted to develop it, but also adhering to the usual standards of the project and company.

The processes we want to get will be from the table called 'DATA_TABLE' in our database. We can see in the properties file (Figure 20) the configurations to connect to, on line 4 showing the port and name of the database: 'TEST_DB'. We can also see this in our 'DBConfig' class (Figure 19). We are using the @Value annotation to take the values from the properties file. Doing it this way will allow us to use different database configurations for switching to different environments, like dev, test and production.



```java
@Service
public class ExternaldemoService {

    private Repository repository;

    public ExternaldemoService(Repository repository) {
        this.repository = repository;
    }

    public List<Data> getData() {
        return repository.getData();
    }

}
```

Figure 11 - Service layer

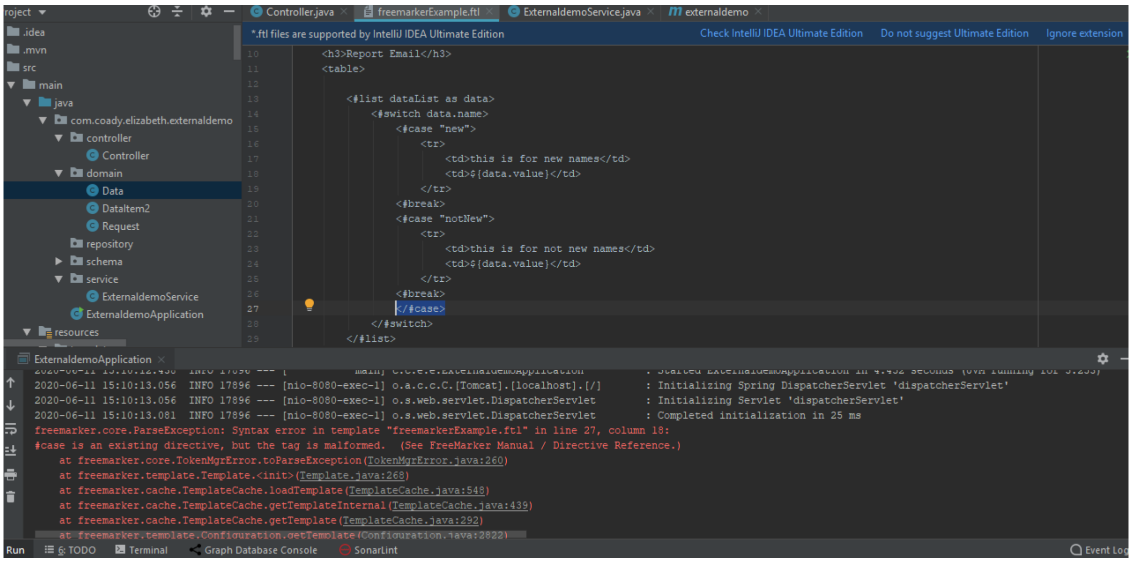


```java
@org.springframework.stereotype.Repository
public class Repository {

    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    public Repository(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {
        this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;
    }

    public List<Data> getData() {
        Map<String, Object> params = new HashMap<>();
        StringBuilder query = new StringBuilder();

        query.append("SELECT * FROM DATABASE.DATA_TABLE ");
        query.append("WHERE NAME IN (:LIST_OF_NAMES)");

        List<String> listOfNames = Arrays.asList("new", "notNew");

        params.put("LIST_OF_NAMES", listOfNames);

        return namedParameterJdbcTemplate.query(query.toString(), params, new DataRowMapper());
    }

}
```

Figure 12 - Repository layer

For the repository layer, I used the NamedParameterJdbcTemplate class because it was required by the company so therefore I am adhering to company standards. This class allows us to construct SQL queries to retrieve what we need, as shown in Figure 12 - Repository layerFigure 12. In Figure 17 we are implementing the RowMapper to map our data class to the ‘DATA_TABLE’ table in our database (screenshot of database from command line shown in Figure 21). This row mapper implements our domain object (Figure 18) so we can map our ‘NAME’ and ‘VALUE’ columns in our database to the corresponding fields in our object.

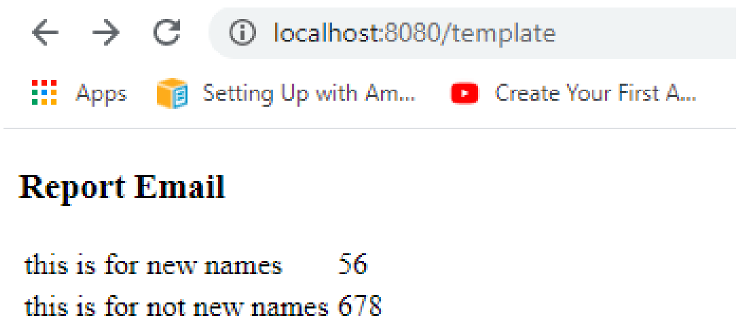
I am then linking to the service (Figure 11) and then the controller (Figure 16). We are mapping the email html endpoint to /template, which will be picked up by the lambda service created by my other POD members to trigger the emails We can see the rendered html output in a browser in Figure 14. I made sure I communicated with my team who were developing the lambda trigger to see if we were on the same page so our services would be compatible and work together, which they successfully did.



```
    <h3>Report Email</h3>
    <table>

        <#list dataList as data>
            <#switch data.name>
                <#case "new">
                    <tr>
                        <td>this is for new names</td>
                        <td>${data.value}</td>
                    </tr>
                <#break>
                <#case "notNew">
                    <tr>
                        <td>this is for not new names</td>
                        <td>${data.value}</td>
                    </tr>
                <#break>
                </#case>
            </#switch>
        </#list>
```

```
2020-06-11 15:10:12.458  INFO 17896 --- [           main] c.c.e.e.ExternaldemoApplication          : Started ExternaldemoApplication in 4.452 seconds (JVM running for 5.233)
2020-06-11 15:10:13.056  INFO 17896 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-06-11 15:10:13.056  INFO 17896 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-06-11 15:10:13.081  INFO 17896 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 25 ms
freemarker.core.ParseException: Syntax error in template "freemarkerExample.ftl" in line 27, column 18:
#case is an existing directive, but the tag is malformed.  (See FreeMarker Manual / Directive Reference.)
	at freemarker.core.TokenMgrError.toParseException(TokenMgrError.java:260)
	at freemarker.template.Template.<init>(Template.java:268)
	at freemarker.cache.TemplateCache.loadTemplate(TemplateCache.java:548)
	at freemarker.cache.TemplateCache.getTemplateInternal(TemplateCache.java:439)
	at freemarker.cache.TemplateCache.getTemplate(TemplateCache.java:292)
	at freemarker.template.Configuration.getTemplate(Configuration.java:2822)
```

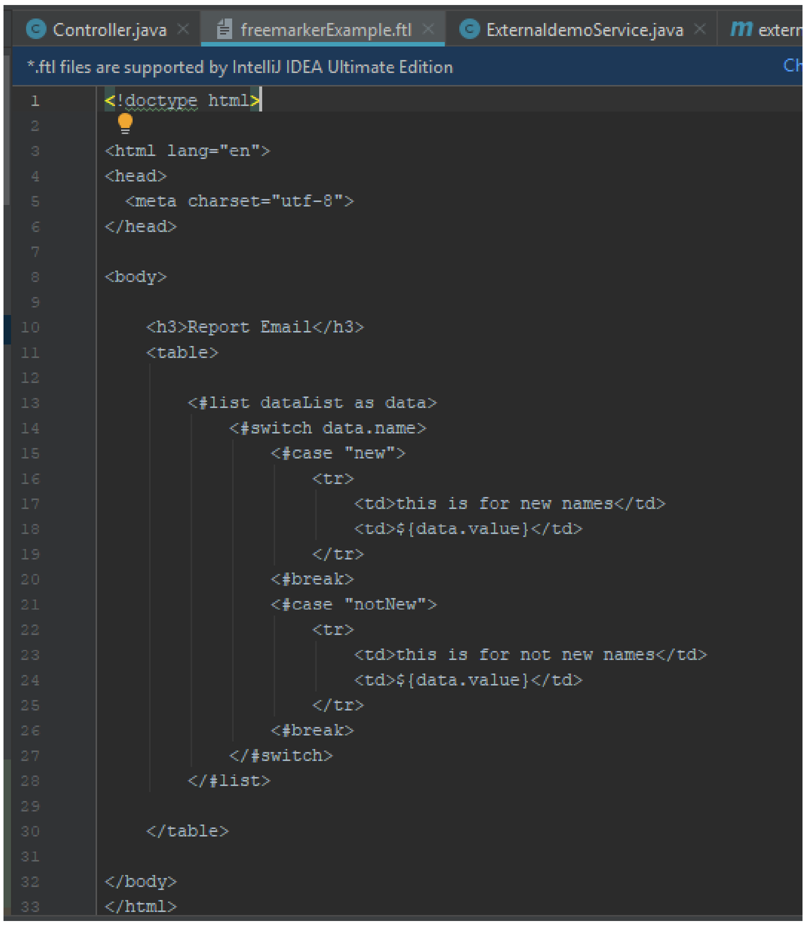
Figure 13 - Stack trace error in ftl template



Report Email

this is for new names	56
this is for not new names	678

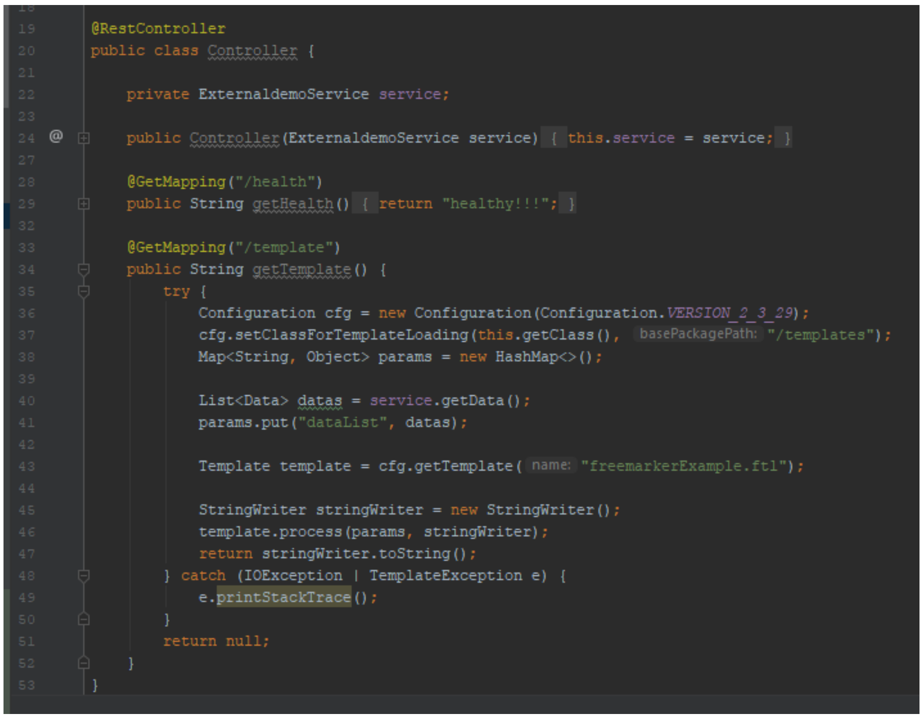
Figure 14 - Template output



```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">
</head>

<body>

    <h3>Report Email</h3>
    <table>

        <#list dataList as data>
            <#switch data.name>
                <#case "new">
                    <tr>
                        <td>this is for new names</td>
                        <td>${data.value}</td>
                    </tr>
                <#break>
                <#case "notNew">
                    <tr>
                        <td>this is for not new names</td>
                        <td>${data.value}</td>
                    </tr>
                <#break>
            </#switch>
        </#list>

    </table>

</body>
</html>
```

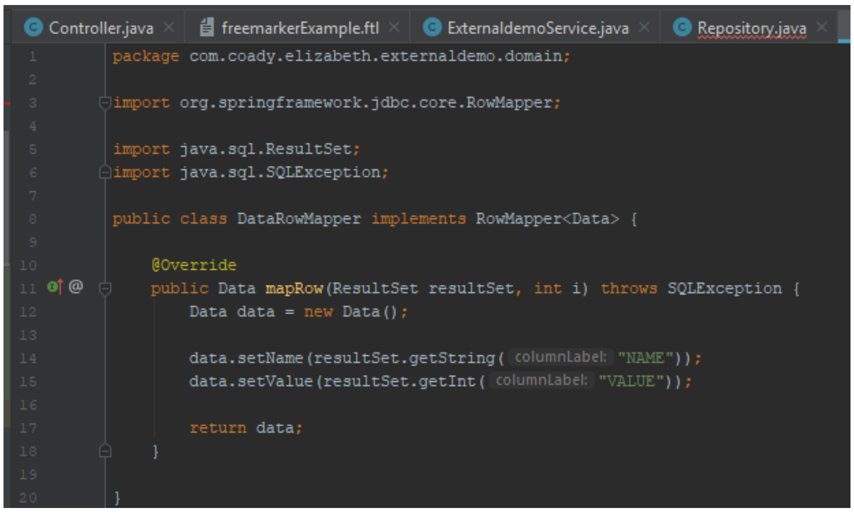
Figure 15 - Template file with corrected syntax

To create the email template, I decided to use the FreeMarker library as the email layout desired was extremely basic, requiring no CSS or JavaScript, so we can integrate it with one file in our backend codebase, putting the template in our resources folder. I had seen the email design and identified that we would not need any complex framework for it like Angular we had been using for the front end, so I used my initiative and decided FreeMarker would be the simplest and most efficient way, which got my senior colleagues approval.



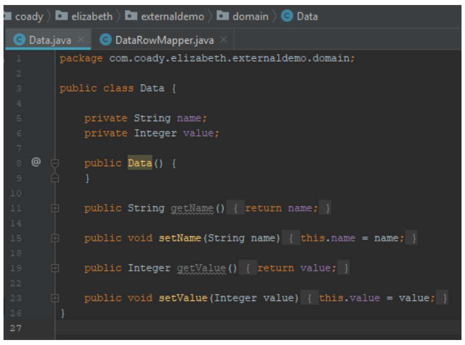
```java
@RestController
public class Controller {

    private ExternaldemoService service;

    public Controller(ExternaldemoService service) { this.service = service; }

    @GetMapping("/health")
    public String getHealth() { return "healthy!!!"; }

    @GetMapping("/template")
    public String getTemplate() {
        try {
            Configuration cfg = new Configuration(Configuration.VERSION_2_3_29);
            cfg.setClassForTemplateLoading(this.getClass(), basePackagePath: "/templates");
            Map<String, Object> params = new HashMap<>();

            List<Data> datas = service.getData();
            params.put("dataList", datas);

            Template template = cfg.getTemplate( name: "freemarkerExample.ftl");

            StringWriter stringWriter = new StringWriter();
            template.process(params, stringWriter);
            return stringWriter.toString();
        } catch (IOException | TemplateException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Figure 16 - Controller layer



```java
package com.coady.elizabeth.externaldemo.domain;

import org.springframework.jdbc.core.RowMapper;

import java.sql.ResultSet;
import java.sql.SQLException;

public class DataRowMapper implements RowMapper<Data> {

    @Override
    public Data mapRow(ResultSet resultSet, int i) throws SQLException {
        Data data = new Data();

        data.setName(resultSet.getString( columnLabel: "NAME"));
        data.setValue(resultSet.getInt( columnLabel: "VALUE"));

        return data;
    }

}
```

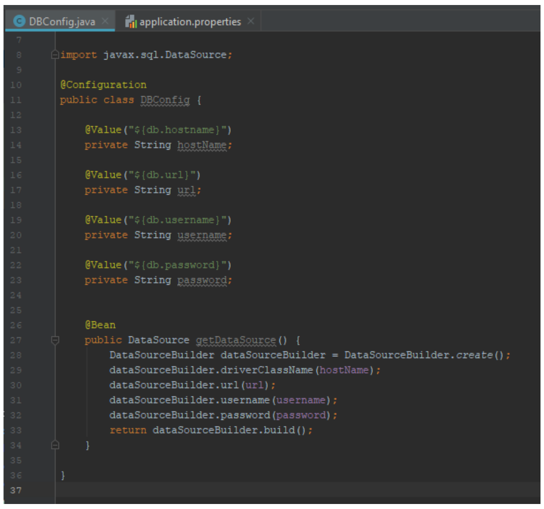
Figure 17 - RowMapper implementation for database mapping



```java
package com.coady.elizabeth.externaldemo.domain;

public class Data {

    private String name;
    private Integer value;

    public Data() {
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public Integer getValue() { return value; }

    public void setValue(Integer value) { this.value = value; }
}
```

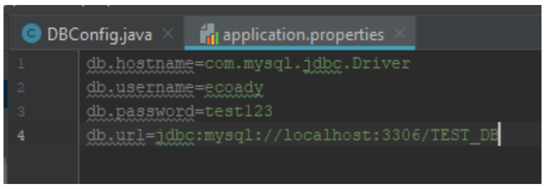
Figure 18 - 'Data' domain object

In Figure 15 we can see the html template of the email. On line 13, we are using the FreeMarker syntax to iterate through our list of Data, which is what we are returning in our service. We want to display information depending on a condition, so I added a switch case. In production we would have 6 different cases, I decided to use switch case instead of 'if' statements as it would be clearer and more logical. We use the ${} syntax for interpolation to display the values from our 'Data' object. Our list 'dataList' is from our controller in Figure 16 on line 41 which we are then passing as a parameter on line 46 to the FreeMarker template object that we are processing. We are then using the StringWriter class to convert this template to a string (line 46) and returning this, as this is the desired output. I was able to use the skills I had learned about string interpolation from my time using Angular and basic JavaScript, so I was able to translate this over to this feature.



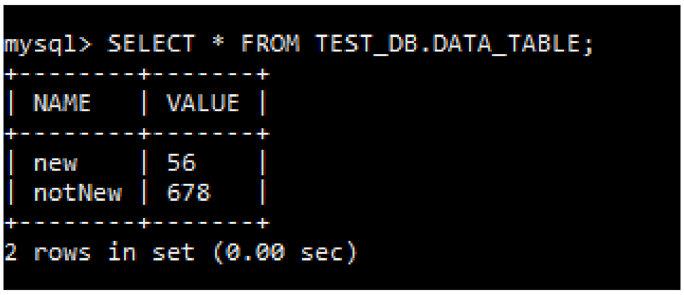
```java

import javax.sql.DataSource;

@Configuration
public class DBConfig {

    @Value("${db.hostname}")
    private String hostName;

    @Value("${db.url}")
    private String url;

    @Value("${db.username}")
    private String username;

    @Value("${db.password}")
    private String password;


    @Bean
    public DataSource getDataSource() {
        DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
        dataSourceBuilder.driverClassName(hostName);
        dataSourceBuilder.url(url);
        dataSourceBuilder.username(username);
        dataSourceBuilder.password(password);
        return dataSourceBuilder.build();
    }

}
```

Figure 19 - Database configuration class



```
db.hostname=com.mysql.jdbc.Driver
db.username=ecoady
db.password=test123
db.url=jdbc:mysql://localhost:3306/TEST_DB
```

Figure 20 - Application properties file



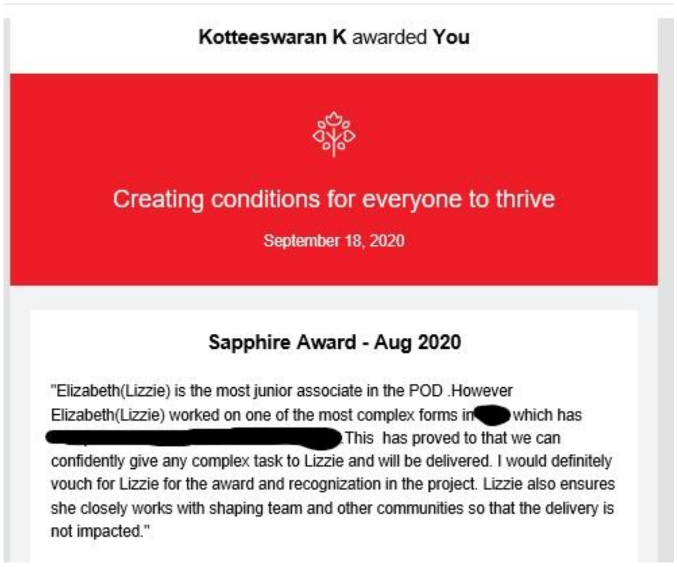
```
mysql> SELECT * FROM TEST_DB.DATA_TABLE;
+--------+-------+
| NAME   | VALUE |
+--------+-------+
| new    | 56    |
| notNew | 678   |
+--------+-------+
2 rows in set (0.00 sec)
```

Figure 21 - Database output from command line

As FreeMarker was a brand new engine to me and the template was technically a new language, using the .ftl extension, I had to learn the syntax that the template used. Although the documentation helped through most of it, I was also able to debug using the built in stack trace in IntelliJ. In Figure 13, I made a syntax error in the template on line 27 by adding an unnecessary closing tag, which thanks to the error message in the stack trace I could identify the issue and fix immediately.

Overall, having ownership on this feature was a really great opportunity for myself and boosted my confidence in my ability significantly, as it was a great success and gained more trust from my team that I was able to lead and have the confidence to take my own initiative when developing.

I was also very pleased to earn an official recognition on our company's appraisal system about my achievements, seen in Figure 22 (note: some words are blanked out as they mention client information).



Kotteeswaran K awarded You

Creating conditions for everyone to thrive

September 18, 2020

Sapphire Award - Aug 2020

"Elizabeth(Lizzie) is the most junior associate in the POD .However Elizabeth(Lizzie) worked on one of the most complex forms in [redacted] which has [redacted] This has proved to that we can confidently give any complex task to Lizzie and will be delivered. I would definitely vouch for Lizzie for the award and recognization in the project. Lizzie also ensures she closely works with shaping team and other communities so that the delivery is not impacted."

Figure 22 - Recognition award for work on this project

Review

This project was a challenging but rewarding one. I experienced what it was like to work in a real dev team and making actual decisions that would impact the team and the journey of the development. I was surrounded by many talented and hardworking team mates, who I always felt I could go to if I needed guidance, and later they would also being doing the same to me. I quickly become confident in my work and also for client presentations as I presented our demos on 3 occasions and answered client questions. I learned so much in such a short time I was on the project, but I believe made an impact on the team as I was then selected to undertake a solo project (Connect project) after 2 months. This new project would allow me to use my new skills and explore more areas of user story creation, UI frame working and cloud services which I didn't get too much look in on this one.

Re-joining the project after 10 months away working on 'Connect' was a great experience as I missed working on a bigger project with a team of other developers and was interested in seeing how far the project had progressed since I had last worked there. It was great to use all the new skills I had learned and the confidence I had acquired to use with my new team, leading to many successfully sprints where I feel like I have made a positive impact in the project and my colleagues, especially feeling like this when I got promoted. Overall, I am very proud of what I have achieved and can't wait to tackle more challenges as my career progresses.