Application of Microservice Architecture to B2B Processes

(IBM Watson Customer Engagement)

by

Arpit Jain

(2015047)

Supervisor(s):

External
Mr. Atul A. Gohad
(IBM ISL, Bangalore)

Internal
Dr. Aparajita Ojha
(PDPM IIITDM Jabalpur)

Computer Science and Engineering

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING JABALPUR

(21st May 2018 – 15th November 2018)

End Term Review

Acknowledgement

It is exceptional working here at IBM ISL, starting my professional career among such great colleagues and friends. I have taken efforts in this project. However, it would not have been possible without them. I would like to extend my sincere thanks to all of them.

It is an amazing learning as well as a valuable experience for me to work in the IBM Watson Customer Engagement department. I am grateful for all your support, help, encouragement, innovation, and dedication which will always be a lesson in my life. I take this opportunity to express my profound gratitude and deep regards to **Mr. Atul A. Gohad** who has guided me in all aspects. Also, I would extend my gratitude to **Ms. Rashmi Acharya** for her wonderful mentorship, continuous guidance and for providing necessary information regarding the project & also for support in completing the project. Further, I thank my project partner, **Mr. Vipin Dhonkaria** for his valuable support and team spirit.

I would also take advantage of this opportunity to express gratitude and thankfulness to **Dr. Aparajita Ojha (Internal Supervisor).** I perceive this opportunity as a huge milestone in my career development and overall growth as an individual.

Date: 15th November, 2018

Warmest Regards,

(Arpit Jain)

Table of Contents

1. Introduction

Chapter 1

The International Business Machines Corporation (IBM) is an American multinational technology company headquartered in Armonk, New York, United States, with operations in over 170 countries. IBM manufactures and markets computer hardware, middleware and software, and provides hosting and consulting services in areas ranging from mainframe computers to nanotechnology.

Until this End-term review, during the duration of my internship, I was involved in gaining knowledge of Microservice Architecture, REST API architecture and about the project structure & organization along with workflow of the IBM Product called Sterling B2B (Business to Business) Integrator. Using this knowledge during these six months, I developed a REST API Client service for the Sterling B2B Integrator. Plus, as an added functionality, I also created a data-transformation service called XML-JSON Transformer. Both of these services are going to be included in the next product release (Sterling Integrator v6.0.1) release, in May 2019.

Apart from this, I also got a chance to work on the existing IBM TenX based Business APIs. I understood the source code to determine how these APIs worked and how was their structure. I found out with the help of my team mates that these APIs were based on the slower TenX framework which placed multiple sequential calls to perform a single operation (for instance a single CRUD Read operation). We took upon the POC to use modern Java-based Spring-Boot framework in place of the TenX to benchmark the performance changes and improvements. We converted several Business APIs from TenX to Spring-Boot and carried out the metric. One Business API namely UserAccounts was fully converted to the Spring-Boot framework (All the CRUD calls – Create, Read, Update and Delete). Upon carrying out the benchmarking against the TenX, it was found that we achieved the performance improvement of **approximately 12 times.** That is a big deal when large Databases are taken into consideration.

Further, these new microservice spring-based APIs were lacking the UI front so I got myself acquainted to the Angular JS frontend framework. As a part of IBM internal P2-hackathon, I developed a combined UI and a Spring-Boot API Gateway (Secured using IBM w3id) in less than 24 hours.

I further took an active part in IBM Lab - Call for Code Hackathon and Developed a Web application called **"Dsense.AI"**. The Web App simplified the process of Human Resource tracking during the mass scale disasters. Our project was selected for the Global Call for Code challenge. We were one of the 128 teams selected from the entire Globe. However, we didn't win the competition, but it was a great learning experience for our team. We were Issued IBM Call for Code contributor digital Badge for this. (Ctrl + Click on the image to view the badge)

Apart from this, I also worked on the **Xlence program** for IBM interns. As a member of this program, me along with other IBM interns, aimed to develop an external project under some IBM mentor. This project, called **IBM Deployment Framework,** dealt with the smoothening of DevOps by providing a combined click-based UI for deploying the Docker Build Images through all the four deployment environments – Development, QA, Sandbox and Production. We were Issued IBM XLENCE Advocate Digital Badge for this.

We were the only XLENCE team out of 12 teams which were able to complete their XLENCE demo on time with an End-to-End working demo. This demo was given to our Director (Watson Supply Chain Development) – Mr. Manjeri Dharmarajan and Vice President (Watson Supply Chain Development) – Mrs. Kelly Ryan.

1.1 Brief Overview

During this period of 6 months, I have completed several tasks pertaining to the B2Bi Sterling Integrator. Proceeding in stages,

Firstly, I researched about the Microservice Architecture, REST APIs and Business-to-Business processes.

Secondly, I understood the basics of Business Process Modelling language (BPML). BPML is an XML standard to write business process. I gave a presentation regarding the application of microservice architecture to my team. To experiment further with the microservice software development architecture, I developed two Microservices – One on node.js (Backend Service) and other using Java JSP (Frontend Service). Then, I established a REST API connection channel between both services.

Thirdly, I got myself acquainted with the Sterling Integrator product and deployed a Sample WAR (Web Application Archive) file on this platform. Further, I deployed the B2B-API WAR file on this platform.

Fourthly, I developed and Installed a REST-API Client service and Business workflow for Sterling Integrator. This will allow the clients to place CRUD calls onto the APIs deployed in the previous step. This is a major step as a lot of Sterling Integrator clients required a service like this.

Later, I worked on yet another service for Sterling Integrator called XML-JSON-Transformer. This will allow the users to inter-convert the data in desired format.

After this, my worked shifted onto the Business APIs. I converted some of the Business APIs to Spring-Boot framework and demonstrated the performance improvements.

These APIs lacked the API-reference documentation and the UI. Hence, I worked on that and completed these missing features.

Lastly, I had to document all the stuff that I worked upon during the internship. This was done for the future reference and testing of the work after I leave. Also, in extension to this, I had to explain all the code structure and working to the team members in the meeting.

1.2 Literature Survey

IBM Sterling B2B Integrator

IBM B2B Integrator helps companies integrate complex B2B (Business to Business) / EDI (Electronic Data Exchange) processes with their partner communities. It provides a single, flexible B2B platform that supports most communication protocols, helps secure your B2B network and data and achieve high availability operations. The offering enables companies to reduce costs by consolidating on a single B2B platform and helps automate B2B processes across enterprises while providing governance and visibility over those processes.

It is a B2B integration software to help synchronize the business partner communities. Today's empowered customers expect more from the companies they do business with.

IBM Sterling B2B Integrator software helps companies execute a smarter commerce strategy by synchronizing virtually every part of the value chain. It addresses complex integration challenges, enabling you to connect your systems to those of your business partners.

- IBM Sterling B2B Integrator is a transaction engine and set of components designed to run processes you define and manage according to your business needs.
- It supports high-volume electronic message exchange, complex routing, translation, and flexible interaction with multiple internal systems and external business partners.
- Has robust security infrastructure, visual management tools for easy configuration of and visibility into work flows, system and trading partner activities.

Integrates applications, processes, data and people, both within and outside an organization.

About the Spring Framework

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans (EJB) model.

Spring Boot follows some of these basic annotations-

@RestController
This is a stereotype annotation that combines @ResponseBody and @Controller. More than that, it gives more meaning to your Controller and also may carry additional semantics in future releases of the framework. Using this annotation, we can make our class to implement a REST API, thus serving only JSON, XML or custom MediaType content.

@Service
Using this annotation, we can make our class as service class and can inject dependencies to other classes. We don't need to instantiate this class again in controller, we just only need to create an object and use its methods.

@Entity
It's an annotation of JPA (Java Persistence APIs) that can make class as table in the database.

@Id
It's an annotation of JPA (Java Persistence APIs) that can make variable as primary key in the database.

We can add multiple dependencies using maven and need not install the associated jars. Automatically all the jars get installed.

Databases
Spring boot supports all types of relational and non-relational databases. We just need to add the dependency of it and can use any database in our application

Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications. Apart from this, it can support other securities such as Apache Shiro.

For the proper understanding and gaining prerequisite knowledge, we referred to several IBM DeveloperWorks articles and Sterling Integrator Documentation. Links for the References are cited in the **Literature Citations Chapter (Chapter 6).**

1.3 Internship Plan

Briefly, stating holistically, the complete internship consisted of these major tasks-

- Getting acquainted with the Sterling Integrator, Business Processes and microservice architecture. I gave a presentation to my team on Microservice Architecture. Also, I researched a way of deploying B2Bi WAR file on the SI server.

- Secondly, I was assigned to a project in which I had to Develop a Rest-API client service for the Sterling Integrator. In addition to that, I also had to write a data transformation service that converted XML data to JSON and Vice-Versa.

- Thirdly, I worked on Importing few of the previously created B2B REST APIs onto the Spring-boot framework.

- Then, API usage documentation and UI were created.

- Finally, the complete documentation of whatever I did at IBM was done for future reference.

I completed most of the tasks in time and in accordance to the requirement. I learned various tools and techniques that are in practice in the industry at present time. I was regularly involved in some or the other task during my whole internship period which has helped me in my overall research and development abilities.

This report describes my tasks that I was involved in during my internship in appropriate detail. The development code could not be presented during this report because it a property of IBM India Software Labs.

2. Report on the Present Investigation
(Progress until the End term review)

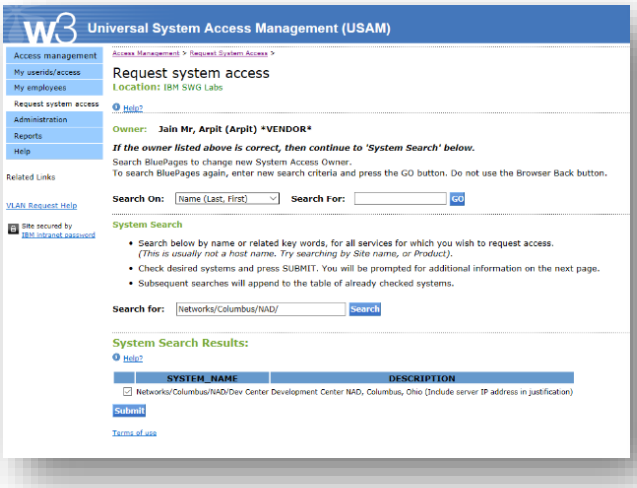
Chapter 2

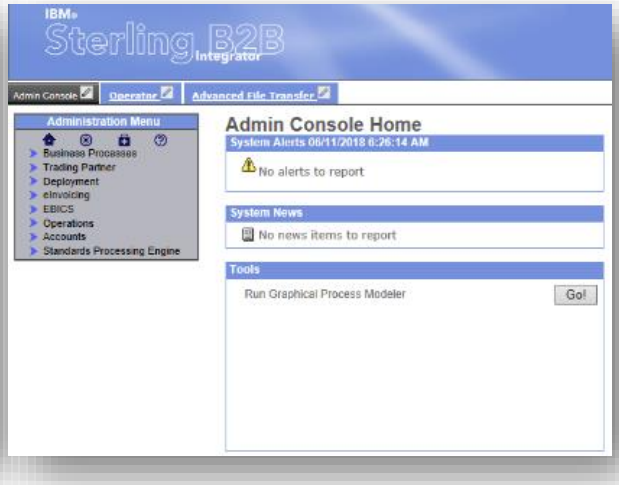
2.1 Researching about Microservice Architecture and B2B Processes

During this duration, I researched and studied about these topics and gave a presentation meeting to the entire B2B Team regarding my study and findings. Following is the link to the presentation I delivered.

2.2 Getting access to the B2B sterling Integrator server

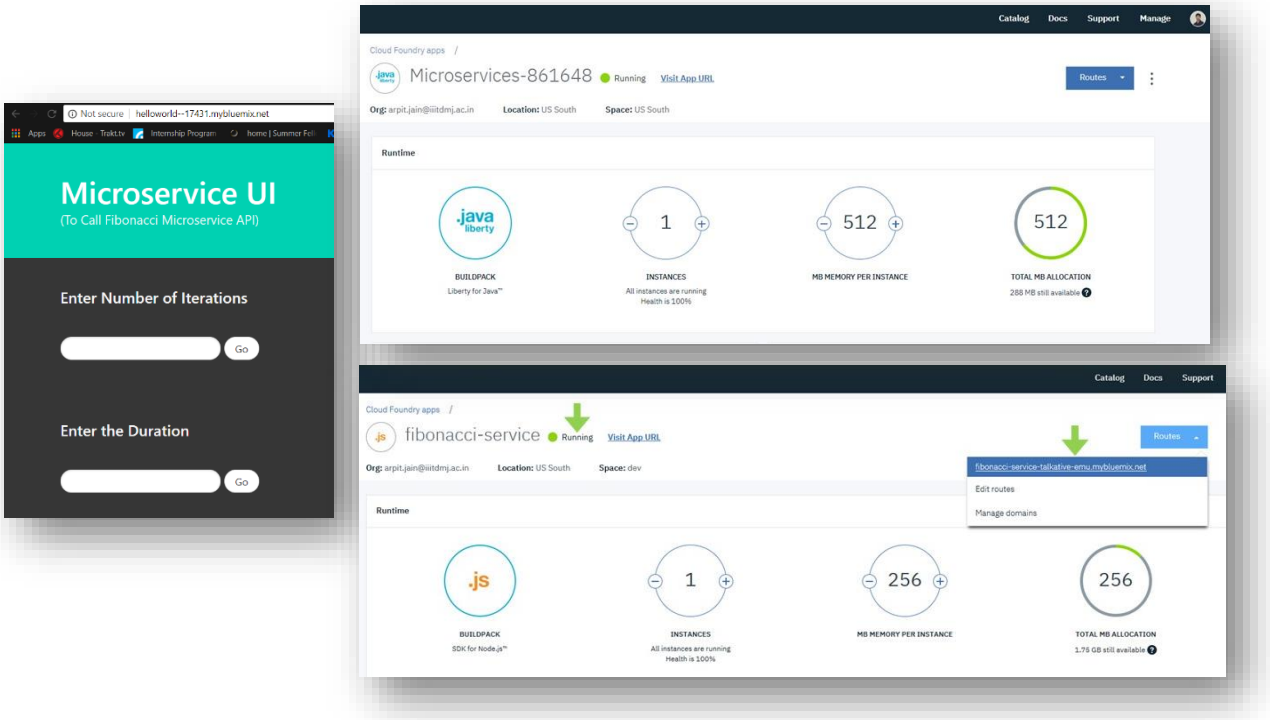
To work on the B2B Sterling Integrator product, access to the servers of IBM on which the product is currently deployed and is running live was to be acquired. The access to the servers and the B2B project team was requested and was granted within a week.





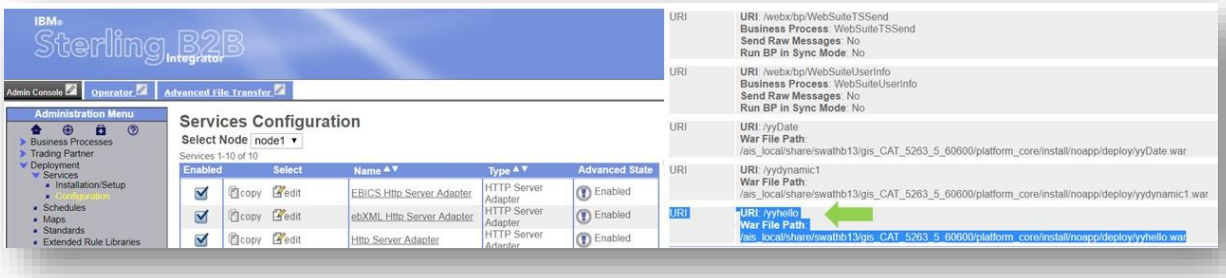
2.3 Microservice Demo application

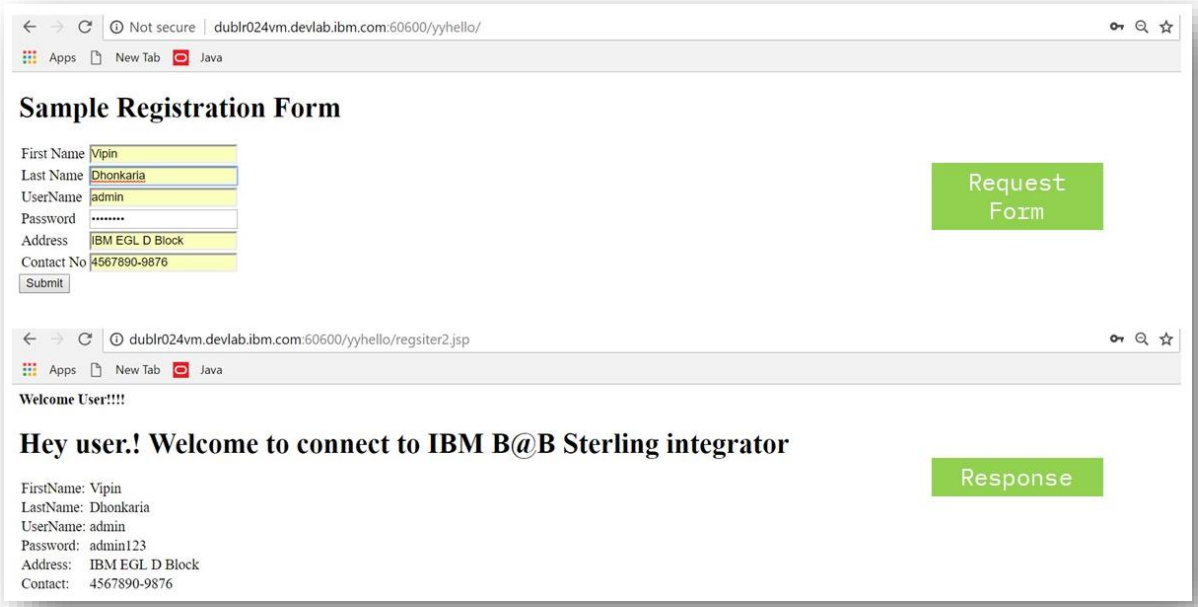
To demonstrate the usage of Microservices architecture for better application design, I implemented a two-microservice application. One microservice acted as a Backend API Fibonacci microservice which was called by another microservice which acted as an UI.



2.4 Deploying a sample WAR file on the B2B SI server

B2B Sterling Integrator works based on Business processes. These business processes are used to automate the operation of the services in the business environment. There is a basic set of base services which form the core of the SI product. These services are written in Java and are pre-packaged altogether in a WAR archive format. To execute and test these services, the B2Bi WAR file is to be deployed on the SI server. Before deploying the actual B2Bi WAR file, I deployed a sample WAR packaged JSP application to test out the dependencies.

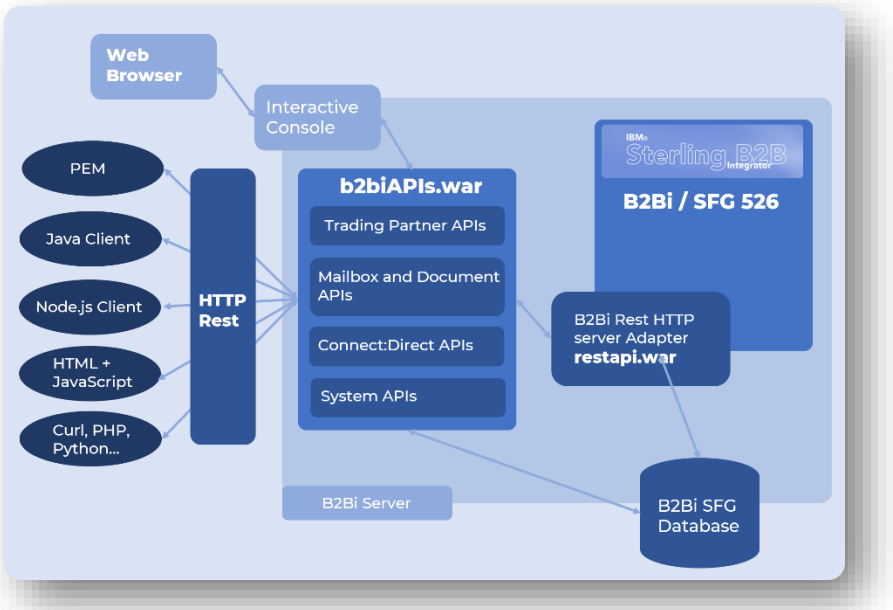




2.5 Deploying B2B-API WAR file on the B2B SI server

B2Bi REST implementation

It was introduced in B2Bi 526 and it allows user to programmatically Create, Read, Update and Delete resources in B2Bi. JSON and XML supported as input and output formats. It provides support for Partner Engagement Manager PEM. It is a more efficient mechanism for on boarding trading partners compared to Xapi (Prior to SFG 526 and REST API).

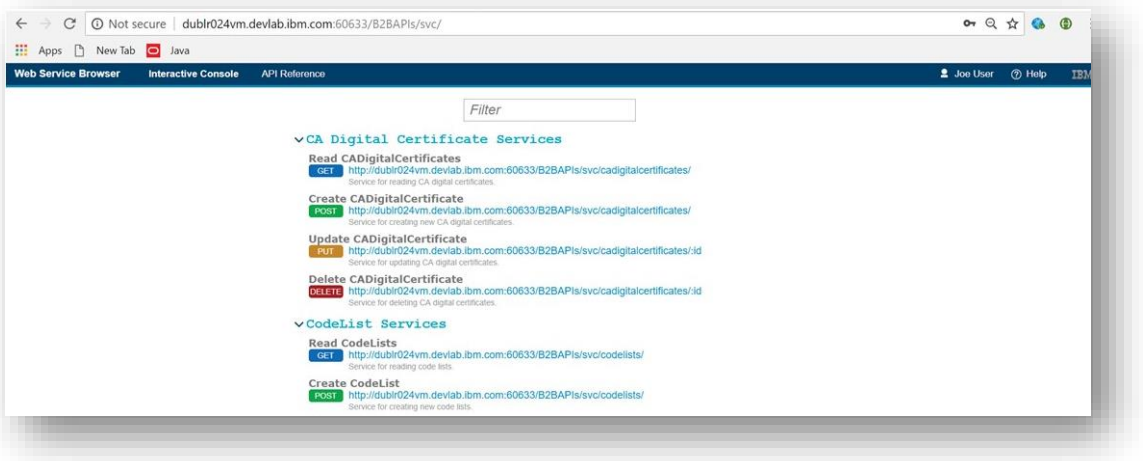


B2Bi REST API Resources

CA Digital Certificate Services
CodeList Services
CodeListCode Services
Community Services
CustomProtocol Services
Digital Certificate Duplicate Check Services
Document Services
External User Services
FgArrivedFile Services
FgRoute Services
Generated Password Services
JDBC Service Tracking Services
Mailbox Services
Mailbox Content Services
Mailbox Message Services
Message Batch Services
Partner Group Services
Permission Services
PGPKey Services
PGP Server Profile Services
Routing Channel Services
Routing Channel Duplicate Check Services
Routing Rule Services
Schedule Services
ServiceDefinition Services
ServiceInstance Services
SSH Authorized User Key Services
SSH Duplicate Check Services
SSH Known Host Key Services
SSH Remote Profile Services
SSH User Identity Key Services
Sterling Connect Direct Netmap Services
Sterling Connect Direct Netmap Xref Services
Sterling Connect Direct Node Services
Sterling Connect Direct Node Duplicate Check Services
Sterling Connect Direct XREF Duplicate Check Services
System Digital Certificate Services
TestSFGDeliveryStatus Services
Test Trading Partner Services
Trading Partner Services
Trusted Digital Certificate Services
User Account Services
User Group Services
UserVirtualRoot Services
Workflow Services
WorkFlowMonitor Services

B2B Sterling Integrator 5.2.6.1 introduced a new REST API interface to provide support for the recently released Partner Engagement Manager (formerly Multi-Enterprise Relationship Management (MRM)). The REST API provides a more efficient mechanism for onboarding trading partners. On the successful installation of B2B Sterling Integrator, B2Bi services WAR file is generated in the installation directory. For the deployment of this WAR file, a HTTP Server Adapter service is to be created for Binding the APIs to an URI (endpoint).

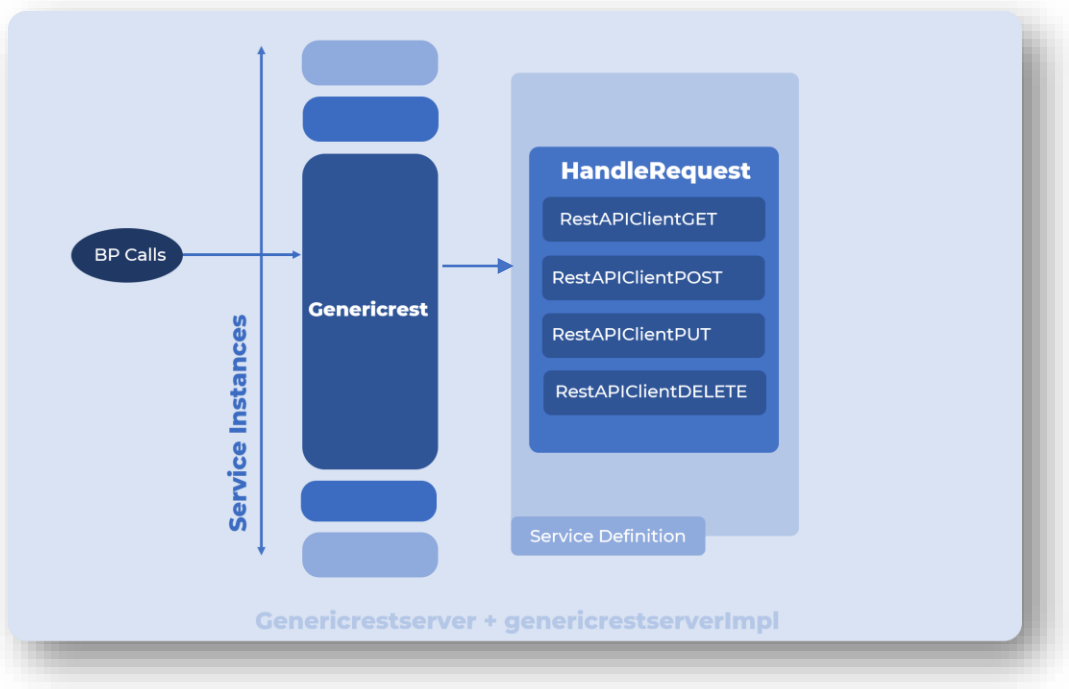
Finally, after the deployment of the B2Bi WAR file (nearly takes 2-3 Hours), the APIs can be accessed at the previously specified URI. The web interface thus opened, shows the list of all the APIs along with their Call methods.



The core of B2Bi is composed of multiple business processes which are used to automate the operation of B2B services (Mailbox, Invoice, Order; etc). Separate microservices can be deployed on the B2B SI platform server to enhance its functionality. By the deployment of B2Bi APIs on the sterling integrator platform, REST API calls can be placed to interact with the product. B2Bi APIs are composed of a set of multiple APIs like mailbox, certificates, codelists, useraccounts; etc. Services in the sterling Integrator are utilised using these APIs only.

2.6 Writing Rest API Client Java service

A service in Sterling Integrator is at least one Java class that implements an interface from Sterling Integrator. In addition to that you have a deployment descriptor that tells the installer during runtime which service you are trying to install. It also tells the system the name of the class that is the entry point into your custom service.



All the service Java files are packaged as following to provide the class definition. This packaged JAR file contains following three files (In the proper package hierarchy)–

```
com
   sterlingcommerce
      woodstock
            services
               genericrest
                  genericrest.java
                  RestRequestHandler.java
                  GenericrestGET.java
                  GenericrestPOST.java
                  GenericrestPUT.java
                  GenericrestDELETE.java
                  genericrestserver.java
                  genericrestserverImpl.java
```

genericrest.java – The service which actually acts as a gateway to handle the upcoming requests depending upon the input parameters from the process data.
RestRequestHandler.java – The java file which distributes the current request depending upon the request type.
GenericrestGET.java – The java file which executes the GET request.
GenericrestPOST.java - The java file which executes the POST request.
GenericrestPUT.java - The java file which executes the PUT request.
GenericrestDELETE.java - The java file which executes the DELETE request.
genericrestserver.java – The java file that is responsible for restarting, shutdown; etc of the genericrest service.

genericrestserverImpl.java – The java file that implements the genericrestserver class.

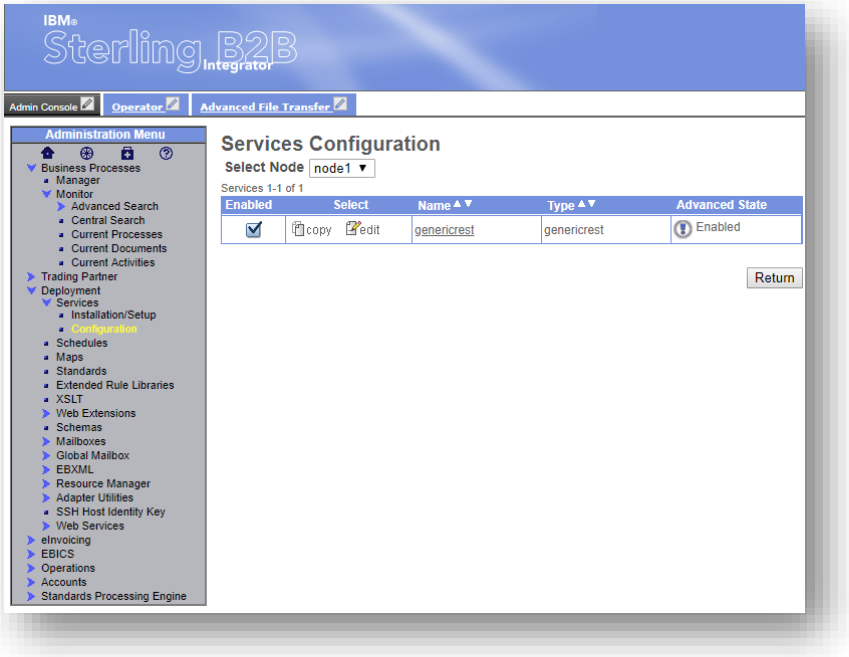
2.7 Writing Rest API Business Processes to test (run) the service

Following is a test workflow for testing the service. When it is executed, it sets the **requesttype** and **url** from the user interface into the process data, and then genericrest service is executed. Following is the Code for a sample Request from Business process. It represents the code for a single instance of the generic rest Java service. Within sequence section, Multiple operations can be written for invocation of multiple Service instances from the same Business process -

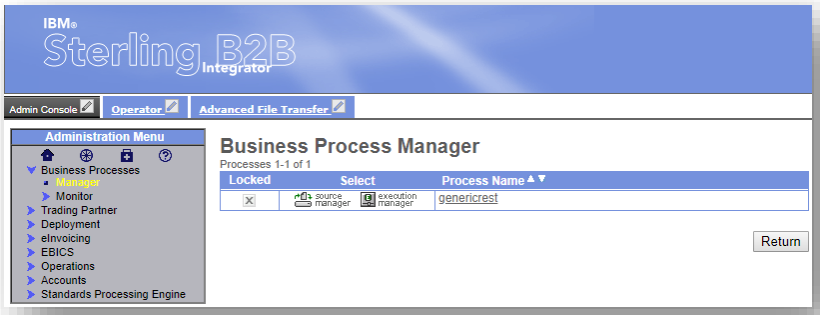
```
<process name="genericrest">
 <sequence>
     <operation name="Request">
      <participant name='genericrest'/>
      <output message='xout'>
       <assign to='url'>URL 1, URL 2, URL 3, ...</assign>
       <assign to='restoperation'>Request type 1, Request type 2, Request type 3, ...</assign>
       <assign to='.' from='*' />
      </output>
     <input message="xin">
      <assign to='.' from='*' />
     </input>
    </operation>
 </sequence>
</process>
```

2.8 Installing the Service to Sterling B2B Integrator

Installation is carried out by Pre-packaging the application (jar). Then, on the machine where the SI is running, use the InstallService.sh or Install Service.cmd file to install the packaged JAR file. A service called **genericrest Service** will be shown created in the SBI dashboard. This service can be seen by going to the Deployment>>Services>>Configuration from the left panel menu.



A workflow (business process) called **genericrest** will also be created. You can test the service by executing this business process. The response from the Service will be specified in the Process Data itself.



2.9 Running / testing the service

Once the Service is successfully installed, it can be tested by running the Business Process. Following is the sample Business process which executes two consecutive GET calls.

genericrest

Document Tracking:	False	**Version:**	208
Start Mode:	async	**Set onfault processing:**	False
Queue:	8	**Transaction:**	False
Recovery Level:	Manual	**Persistence Level:**	Full
Softstop Recovery Level:	Manual	**Document Storage Type:**	System Default
Life Span:	info	**Complete by - Deadline:**	None Available
Event Reporting Level:	None		

Description: BP Test
Business Process Definition:

```
<process name="genericrest">
  <sequence>
        <operation name="Request">
          <participant name='genericrest'/>
          <output message='xout'>
            <assign to='url'>http://9.55.53.11:51665/B2BAPIs/svc/cadigitalcertificates/b2bi_demo,http://9.55.53.11:51665/B2BAPIs/svc/useraccounts/2001</assign>
            <assign to='restoperation'>GET,GET</assign>
            <assign to='auth'>admin:password</assign>
            <assign to='.' from='*' />
          </output>
         <input message="xin">
           <assign to='.' from='*' />
         </input>
        </operation>
  </sequence>
</process>
```

Process Data

Process Name: genericrest **Instance ID:** 398423

Service Name: genericrest

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessData>
  <URL_0>http://9.55.53.11:51665/B2BAPIs/svc/cadigitalcertificates/b2bi_demo</URL_0>
  <REQUEST_TYPE0>GET</REQUEST_TYPE0>
  <Response_0>
    <Data>
      <_id>b2bi_demo</_id>
      <verifyValidity>{"display":"No","code":false}</verifyValidity>
      <certName>b2bi_demo</certName>
      <verifyAuthChain>{"display":"No","code":false}</verifyAuthChain>
      <certData>MIIDiTCCAnGgAwIBAgIEbSNAczANBgkqhkiG9w0BAQsFADB0MQswCQYDVQQGEwJVUzETMBEGA1UECBMKc29tZSBzdGF0ZTESMBAGA1UEBxMJc29tZSBjaXR5MREwDwYDVQQKEwhzb211IG9yZzEQMA4GA1UECxMHVW5rbm93bjEX
      <certGroups>[]</certGroups>
      <createdOrUpdatedBy>Joe User</createdOrUpdatedBy>
      <ref>http://9.55.53.11:51665/B2BAPIs/svc/cadigitalcertificates/b2bi_demo</ref>
      <creationOrUpdateTime>07/31/2018 02.12 AM</creationOrUpdateTime>
      <href>http://9.55.53.11:51665/B2BAPIs/svc/cadigitalcertificates/b2bi_demo</href>
      <_title>CADigitalCertificate(b2bi_demo)</_title>
    </Data>
  </Response_0>
  <URL_1>http://9.55.53.11:51665/B2BAPIs/svc/useraccounts/2001</URL_1>
  <REQUEST_TYPE1>GET</REQUEST_TYPE1>
  <Response_1>
    <Data>
      <_id>2001</_id>
      <userIdentity>DEFAULT Organization</userIdentity>
      <email>arjain08@in.ibm.com</email>
      <userId>2001</userId>
      <permissions>[{"name":"Admin Web App Permission"},{"name":"MyAccount"}]</permissions>
      <surname>jain</surname>
      <authorizedUserKeys>[]</authorizedUserKeys>
      <ref>http://9.55.53.11:51665/B2BAPIs/svc/useraccounts/2001</ref>
      <givenName>arpit</givenName>
      <authenticationType>{"display":"Local","code":"Local"}</authenticationType>
      <href>http://9.55.53.11:51665/B2BAPIs/svc/useraccounts/2001</href>
      <sessionTimeout>0</sessionTimeout>
      <preferredLanguage>{"display":"English","code":"en"}</preferredLanguage>
      <groups>[]</groups>
      <_title>2001</_title>
    </Data>
  </Response_1>
</ProcessData>
```

2.10 Writing XML-JSON Transformer service

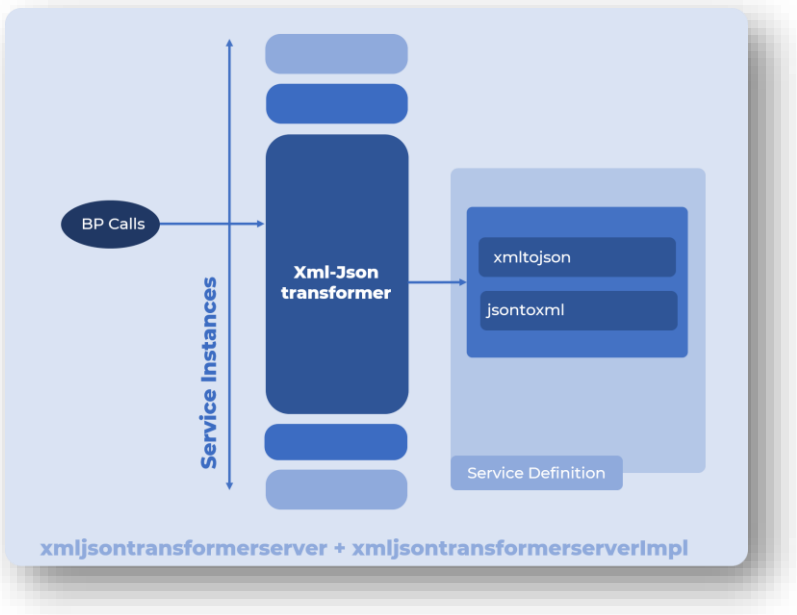
Structural Organization of the Service

The file xmljsontransformer.jar provides the class definition. This JAR file contains following three files (In the proper package hierarchy)–

```
com
   sterlingcommerce
      woodstock
            services
               xmljsontransformer
                  xmljsontransformer.java
                  xmljsontransformerserver.java
                  xmljsontransformerserverImpl.java
```

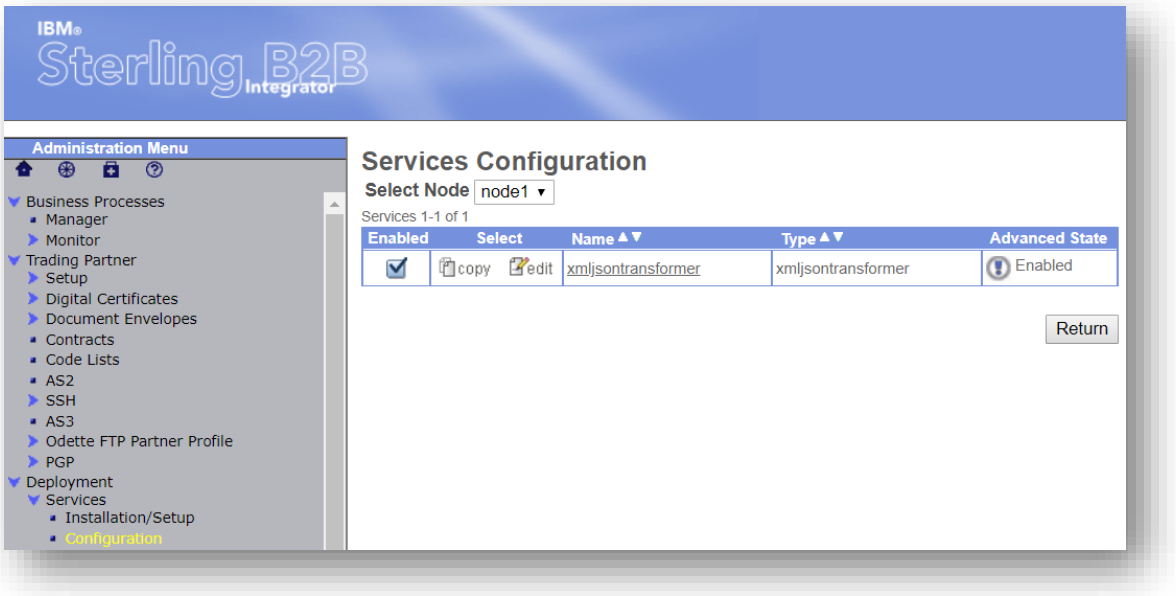
.

xmljsontransformer.java – The service which actually acts as a gateway to handle the upcoming converting requests depending upon the input parameters from the process data.
xmljsontransformerserver.java – The java file that is responsible for restarting, shutdown; etc of the xmljsontransformer service.
xmljsontransformerserverImpl.java – The java file that implements the xmljsontransformerserver class.



Running after Installing the Service to the SI Server.

After installation, on the UI, A service called **xmljsontransformer Service** will be shown created in the SI dashboard. This service can be seen by going to the **Deployment>>Services>>Configuration** from the left panel menu.



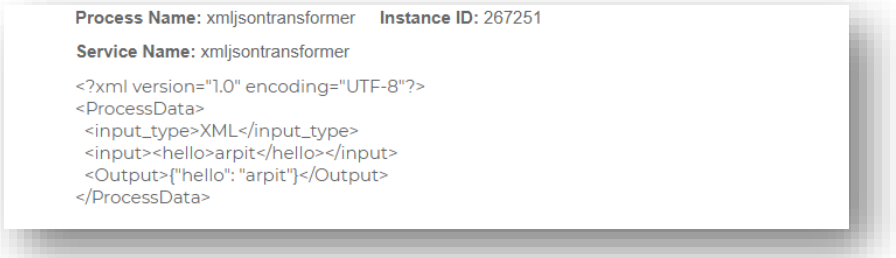
A workflow (business process) called **xmljsontransformer** will also be created. You can test the service by executing this business process. The response from the Service will be specified in the Process Data itself.

Variation in Workflow (BPs) to carry out diverse Operations

Depending upon the requirement, Business Processes can be modified to perform requests.
All the BPs must provide following 2 mandatory parameters:

- **Outputpath:** Path of some directory on the server
- **Processtype:** XMLtoJSON or JSONtoXML
- **Input:** Data to be converted

- **XML to JSON Request**



JSON to XML Request

2.11 Creating a sample REST Spring Application in STS (Spring tool Suite)

Spring Boot is a framework designed to simplify the creation of new services. For the simple use cases, the needed libraries are already bundled in the fitting combinations and versions in so-called spring starters. In any Spring Boot based REST API, following is the Separation of layers where each layer is an individual module/project.

REST API

- Packaged as war
- It has rest controllers that handle request/responses
- depends on Service Module below

Service

- Packaged as jar
- Business logic abstractions, this layer has no idea how to communicate with data source.
- It will be autowired in rest controllers
- Depends on DAO/Repository module below

DAO/Repository

- Packaged as jar
- Talks to data source directly, has operations commonly known as CRUD. It could be simple jdbc, JPA, or even file access.
- Depends on domain module below

Domain

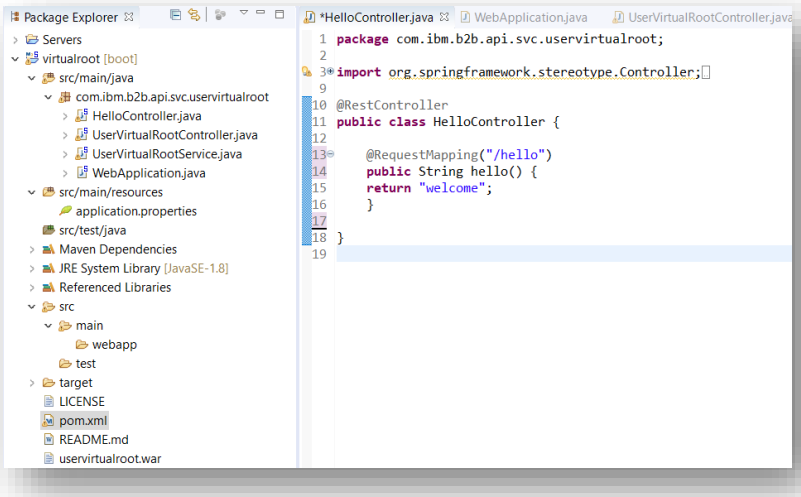
- Packaged as jar
- It has your domain models, normally POJO classes. If you are using ORM, they are ORM entities.
- It could also have DTO (Data Transfer Object), which are still under hot debates. Use it or not is your call.

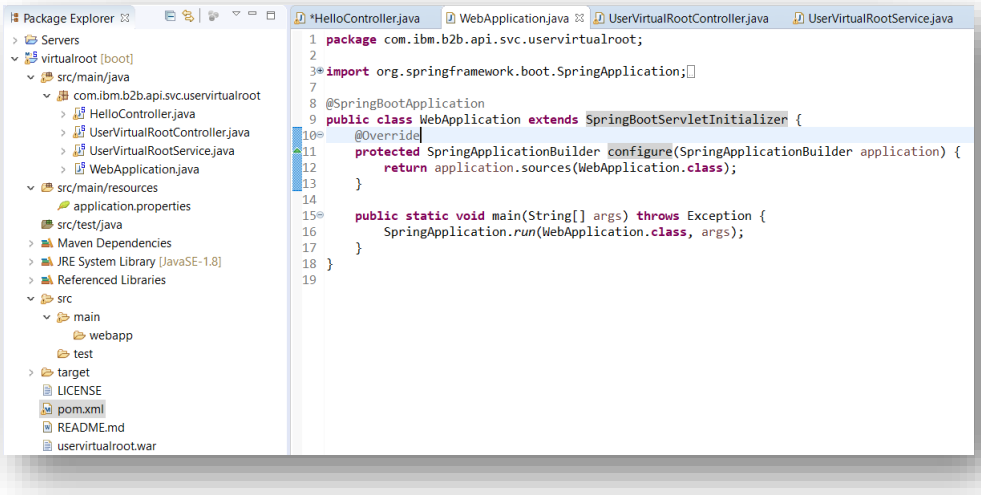
More modules such as utility, third party integration, etc. can be added. but the above are highly recommended to have.

Build/Dependency Management Tools - There are plenty of them like maven and gradle versions. We used **Maven with Spring**. It simply works for the above project structure.

A sample REST controller which returns Welcome message when a GET call to **/hello URL** is made. In this sample application, we have created a class HelloController and annotated it with @RestController which calls REST API and provides json response.

We can also provide individual mappings for http requests such as @PostMapping and can also set this in @RequestMapping(method="POST").

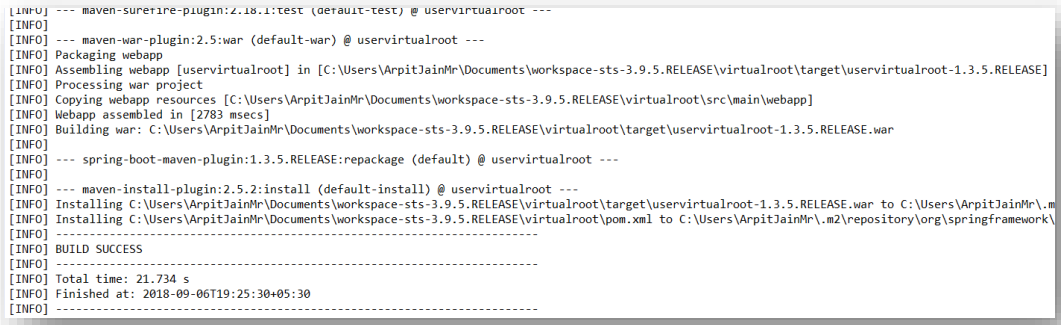




(Application class to start the spring boot)

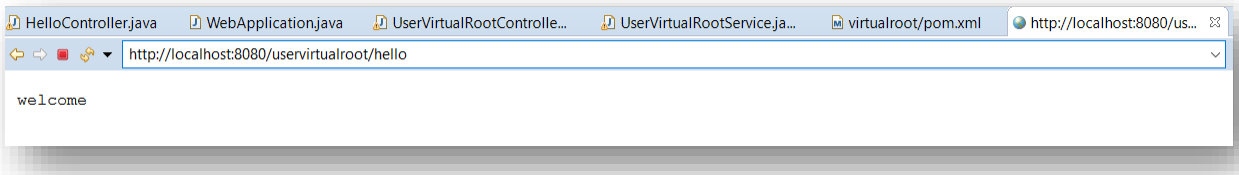
Packaging the Service as a WAR

Exported the application into a WAR file using Maven Install packaging.



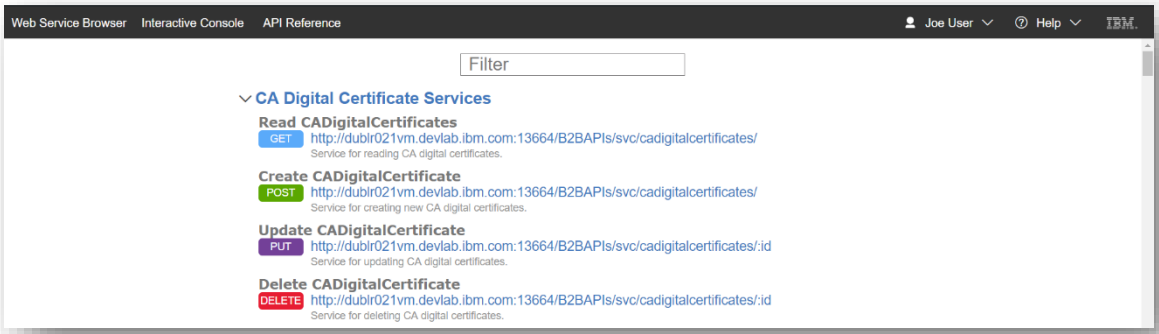
Installing/Deploying the WAR on the WebSphere Liberty Server

This WAR file is now copied to the Liberty server and the server.xml configuration file is modified to create an entry for the new app. The server is then restarted, and the liberty server is also run for reconfiguration.



2.12 Researching about Spring-Boot Architecture, IBM TenX Framework

During this duration, I researched about the Spring-Boot framework and how the Code structure of the TenX framework could be utilized for conversion into Spring-Boot.

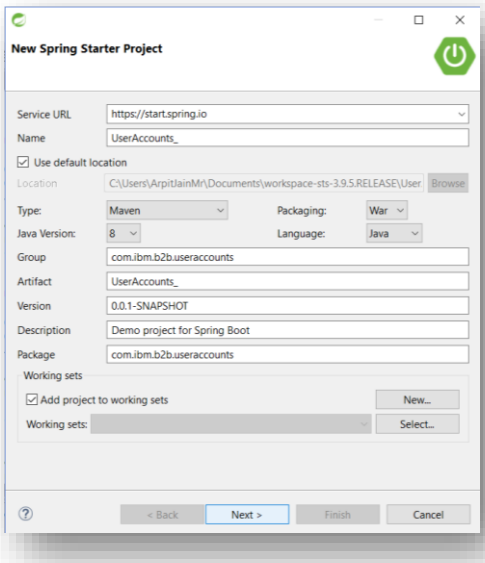


2.13 Converting UserAccounts and UserVirtualRoots APIs to Spring-Boot

Conversion of UserVirtualRoots API to Spring-Boot was explained in the last report. Here, I will cover the conversion of UserAccounts API.

This task required the following steps (demonstrated in detail in the last report).

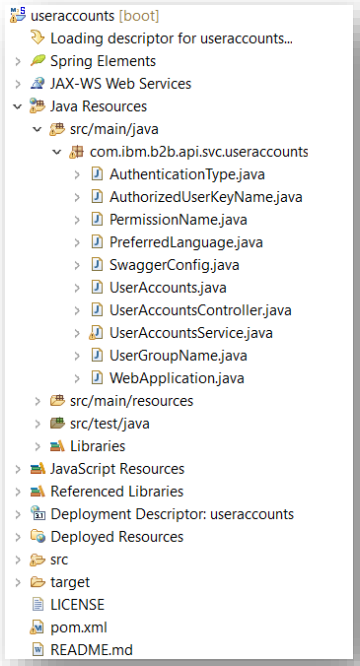
1. Creating the Spring Starter project as shown in the picture.
2. Change the POM.XML dependency file for adding the dependencies.
3. Changing the code structure of Java Source files. (Tenx to Spring-Boot format)



Here, there are following classes –
(Code Cannot be revealed due to confidentiality clauses)

- **POJO Classes:** Classes to represent the object entity structure.

UserAccounts.java
AuthenticationType.java
AuthorizedUserKeyName.java
PermissionName.java
PreferredLanguage.java
UserGroupName.java



- **Controller Class:** This class is used to map the URL and Request Type to the respective Method Calls.

UserAccountsController.java

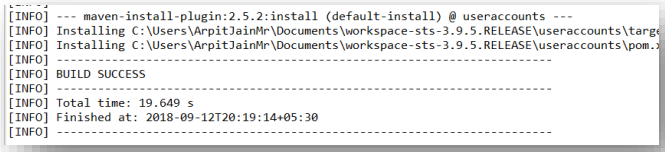
- **Service Class:** This class is used to implement the actual functions for CRUD operations.

UserAccountService.java

- **Web Application Class:** This class is used to start the Spring-Boot.

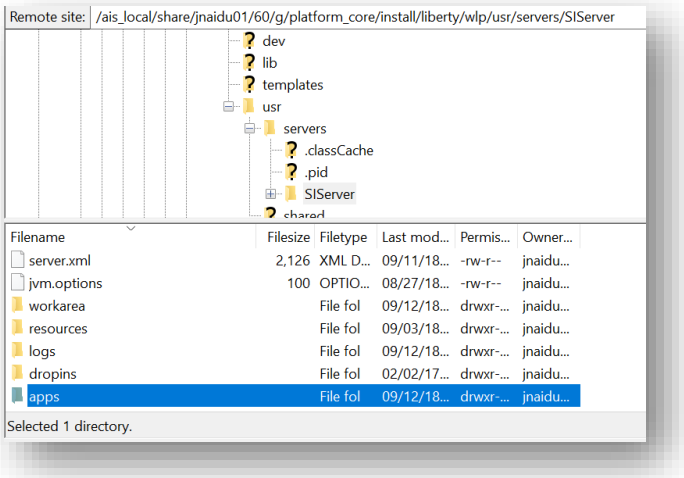
WebApplication.java

4. Package the Application as a WAR file using Maven Install



5. Deploy the File on the Liberty Server

- Copy the generated WAR file to the Liberty Server's App directory.

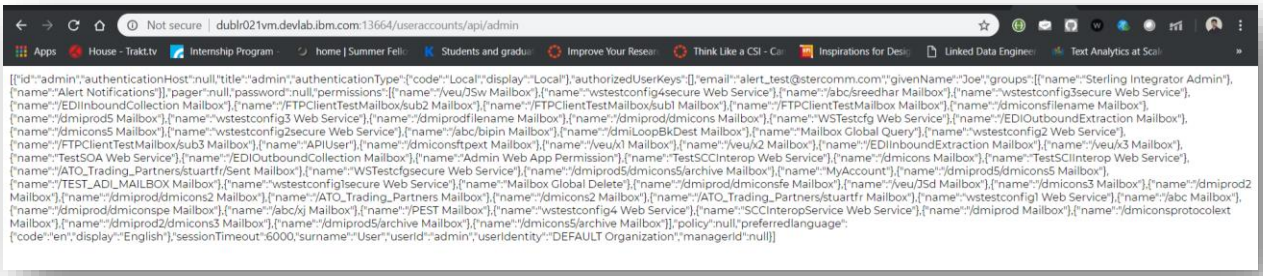


- Edit the server.xml file to add the External entry for Liberty to recognize your app WAR as an endpoint.

6. Restart the Server
   - Run these Commands from the `install/bin/` directory.

     `./hardstop.sh` -> `./run.sh` -> `./startLiberty.sh`

7. Access the API on the specified URL
   - Currently, it is just implemented for a Read Call.



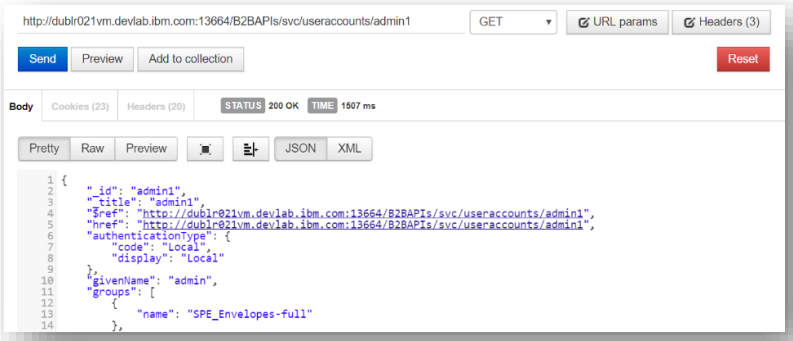
2.14 Performance benchmarking (TenX vs Spring-Boot)

Aim of this benchmark was to compare the performance of using Spring-Boot as a base B2B API framework against IBM TenX API framework. The API which we picked for comparison was UserAccounts.

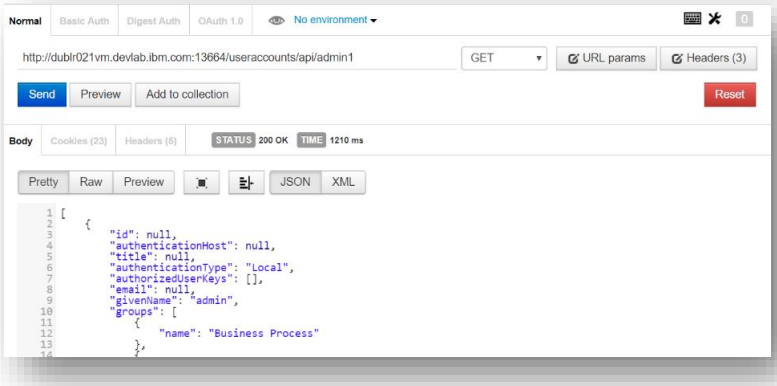
These tests have been done in two stages –

Case 1 : Sparsely populated DB of UserAccounts

Single-Read Request using TenX. (1507ms)



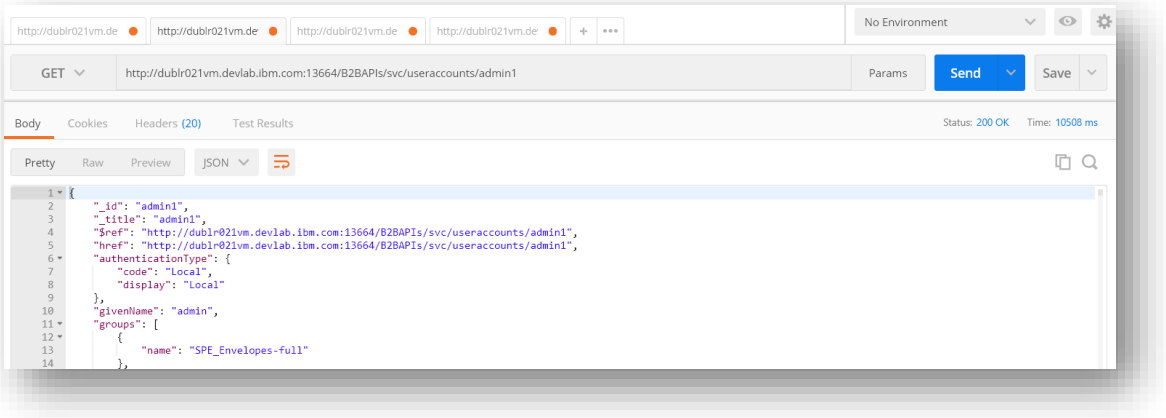
Single-Read Request using Spring-Boot. (1210ms)



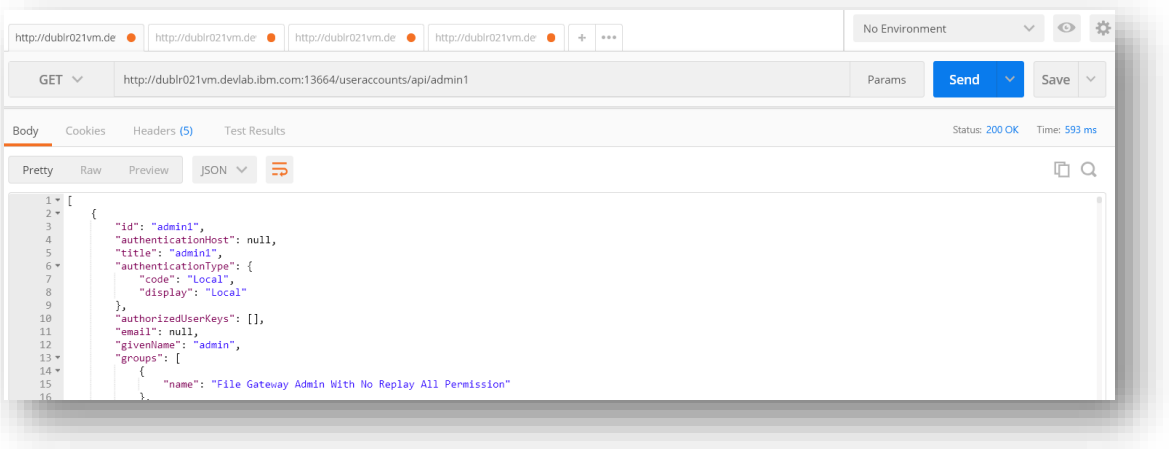
Case 2 : Densely populated DB of UserAccounts (10000 UserAccounts)

For performing extensive testing, we wrote a script that floods the UserAccounts Database by adding (sequentially posting) 10000 temporary User Accounts to the API endpoint.

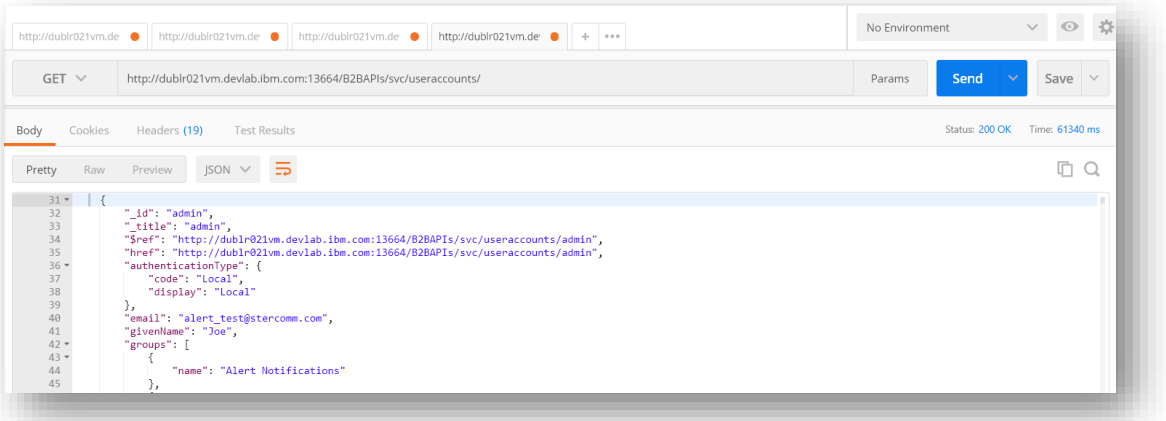
Single-Read Request using TenX. (10508ms)



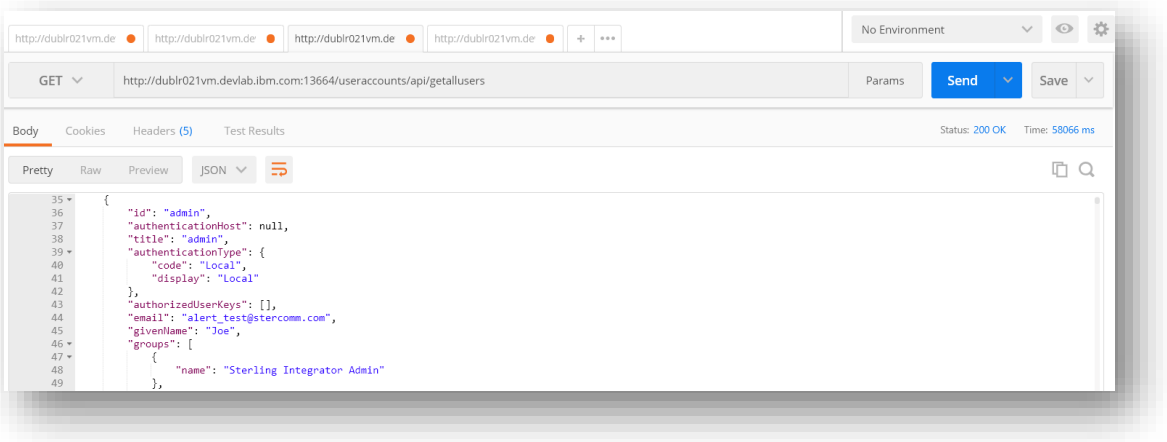
Single-Read Request using Spring-Boot. (593ms)



Read-All Request using TenX. (61340ms, This Request fetches nearly 1000 records)



Read-All Request using Spring-Boot. (58066ms, This Request fetches 10000 records)



Some Noteworthy Points

1. These tests have been performed on the same machine and on the same WebSphere Liberty server.

2. For Read-All requests, TenX only fetches maximum 1000 records.

Conclusions

1. In case of Sparsely populated DB, the difference between Spring-Boot and TenX is not very drastic. (Spring-Boot is nearly 200-250ms faster).

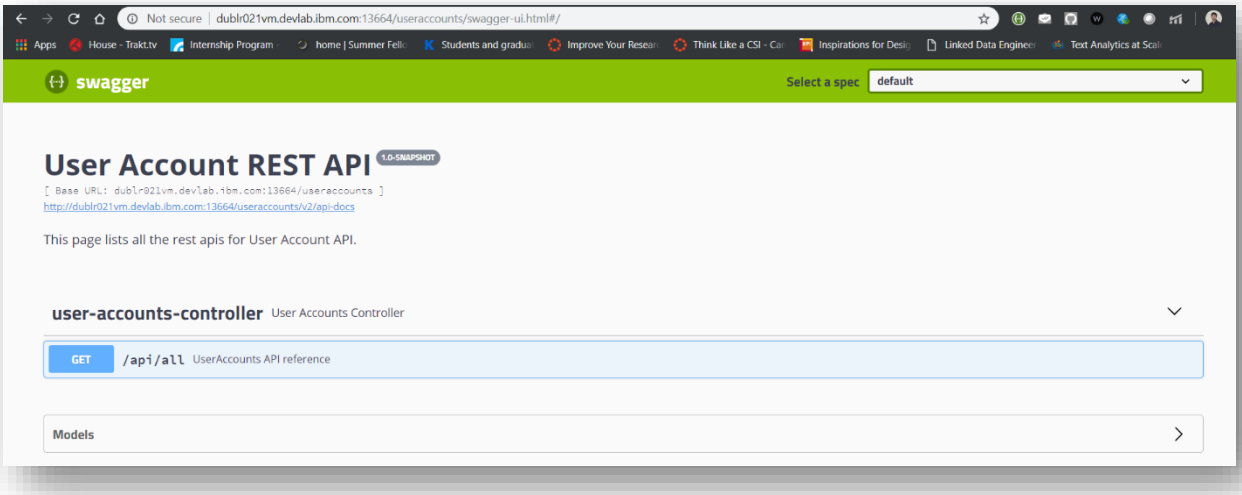
2. In the case of Densely populated DB, the difference is huge.
   a. Single-Read Call
      Effective time for TenX = 10000ms (Approx. Average)
      Effective time for Spring-Boot = 600ms (Approx. Average)
      Performance Ratio = 10000/600 = 16.66 times faster execution of Spring-Boot

   b. Read-All Call
      Effective time for TenX = 60000ms/1000 records = 60ms (Approx. Average)
      Effective time for Spring-Boot = 58000ms/10000 records = 5.8ms (Approx. Average)
      Performance Ratio = 60/5.8 = 10 times faster execution of Spring-Boot

3. Speaking holistically, on an average, Spring-Boot is nearly ~10-12 times faster than IBM TenX.

	TenX	**Spring-Boot**	**Performance Ratio**
Single GET Request	12 seconds	1 second	~12 times faster
Bulk GET Request	68 seconds	6 seconds	

2.15 Adding Swagger Hub Documentation Dependency



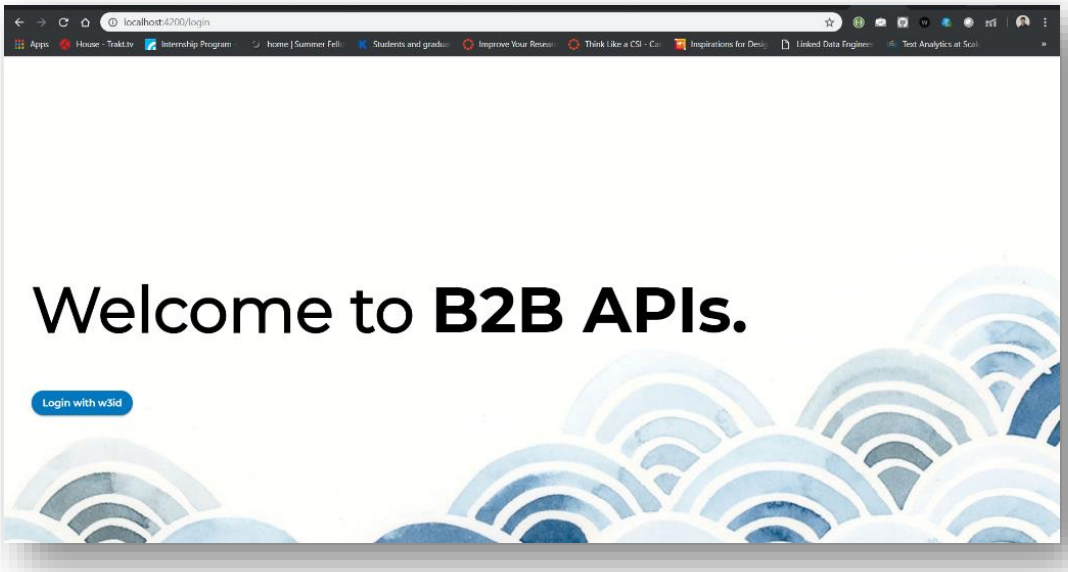
2.16 Adding UI using Angular JS to Spring-Boot APIs

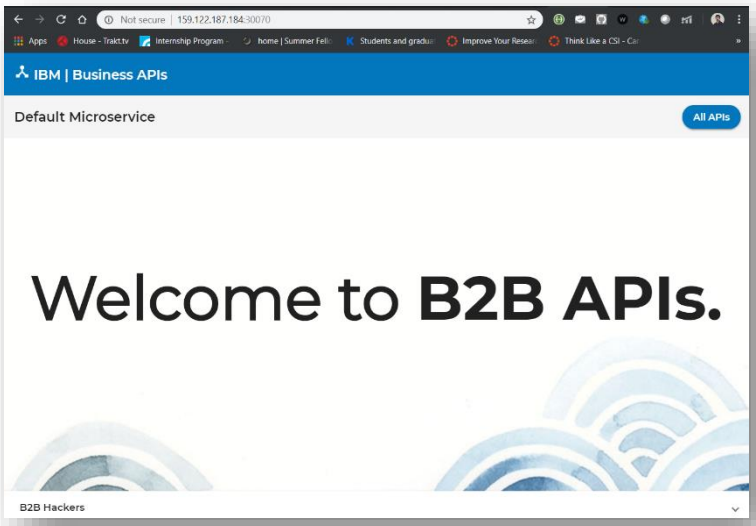
Now, the next task was to add the UI to the newly created APIs.

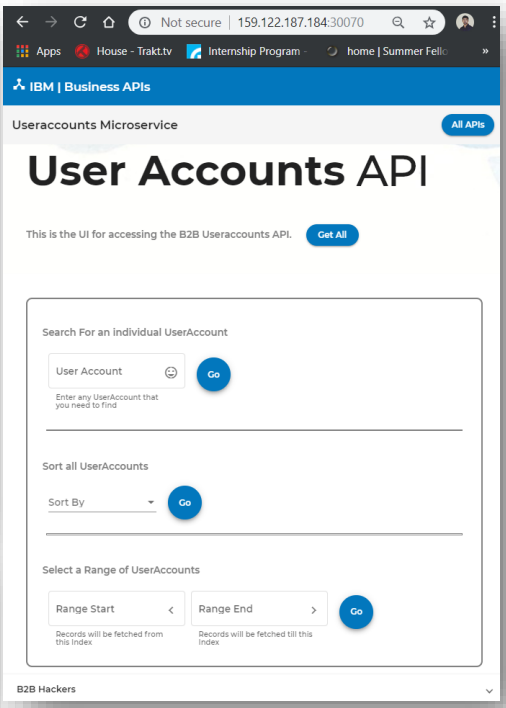
I chose Angular JS for this because it has large community support and is extremely modular. It uses a Language called Typescript (by Microsoft) for code.

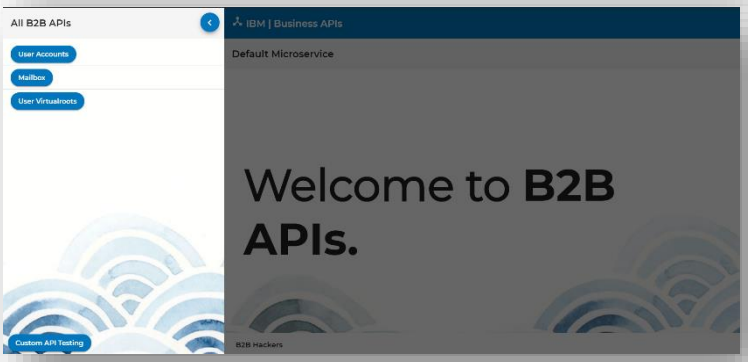
On the right, is the Angular Code Structure. Following are the UI screen snaps for the REST API client we created using Angular.

On the Login page, user needs to click on the **Login with w3id** Button to be authenticated with his/her IBM ID.



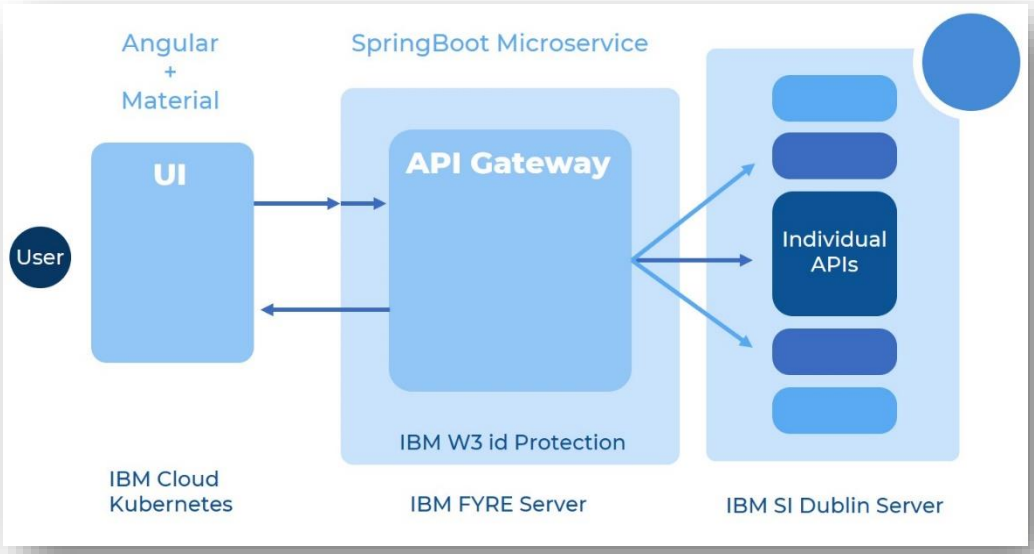






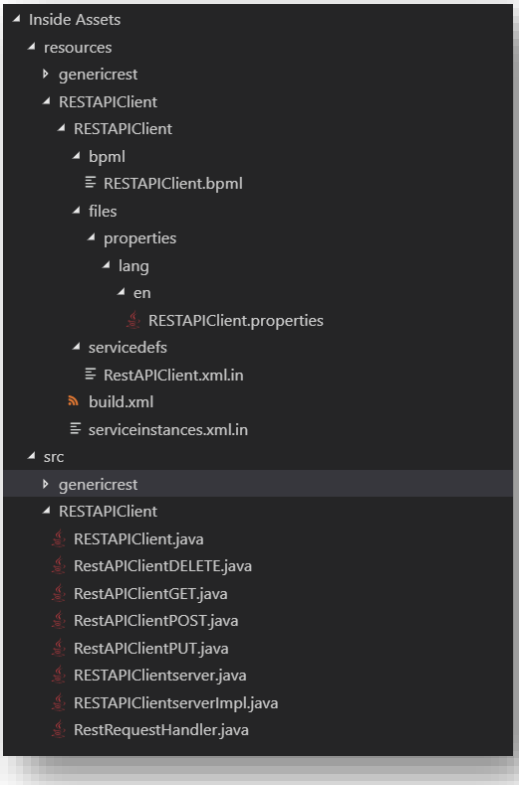
The above UI was deployed as a microservice separate than the Backend Spring based API Gateway on IBM Cloud Kubernetes Cluster by building it as a docker image.

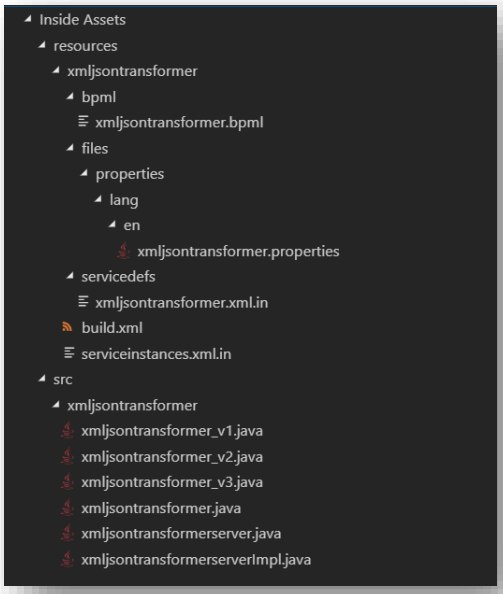
Following is the flow diagram of the Entire Work until now.



2.17 Refinement of the existing REST API Client service and XML JSON Transformer services

Existing services which were created had to be more production ready, so we organized the code structure to make it more understandable for the future maintenance. Also, we made improvements to the file structure organization for making it ready to be committed to the GitHub repository. After this, we had to explain this code to the future maintainers so that they can handle the bugs and issues in these product features during the QA cycles.





Adding Reference to Third Party Jars in Sterling Integrator

Initially, we were building the classes by adding references to the Third-party Jars in dynamic classpath for the product. That is a Hardcoded way of specifying the location of the Jars. One disadvantage of this methodology is that for this, we need to handle the maintenance of the future upgrades(versions) of jars as well. So, it becomes a bit complicated on our end. So as to avoid the complicacy, we added the Jars as a File-Set in the Base source clump. After this, we added the reference entries for these Jars in Build.xml file. So now the jars are being referenced and classes being compiled.

Following were the external third-party open-source JAR files required for the two services –

- Json-simple.jar
- Hamcrest-Core.jar
- Junit.jar
- Org.json.jar

2.18 SpringBoot UserAccounts Microservice improvements

In the past few days, some features were also added to UserAccounts Microservice. Now, it can take Sort and Range parameters simultaneously from the users and perform sorting on the fetched range of user accounts. On fetching all user accounts instances, it is being restricted to some 1000 useraccounts for performance benefits. Initially, it was fetching all the useraccounts irrespective of size. This became a bottleneck in the case of heavily populated Database.

```
@RequestMapping(method=RequestMethod.GET,value="/rangesort={sortId}/{first}/{last}")
public List<UserAccounts> rangeSort(@PathVariable String sortId,@PathVariable String first,@PathVariable String last) {
    return userAccountsService.rangeSort(sortId,first,last);
}

}
```

2.19 Linking XML-JSON-Transformer with REST-API-Client service

The following snippet of the Business Process shows the sequence of the two instances of separate services. The first service XML-JSON-Transformer takes a file input in XML Format and converts it into JSON format.

The second service (REST API Client) picks up this output from the first service and Posts it to form a UserAccount using the Tenx UserAccount API.

Description: BP POST using XML file and Xmljsontransformer service
Business Process Definition:

```
<process name="RESTAPIClient">
<sequence>
<operation name="Request">
<participant name='xmljsontransformer'/>
<output message='xout'>
<assign to='inputtype'>XML</assign>
<assign to='outputpath'>/home/sandbox_601_Nextrelease/gis_CATAMOUNT_BUILD_6001000_BRANCH_32000/platform_core/install/noapp/outpu
<assign to='.' from='PrimaryDocument/@SCIObjectID'></assign>
<assign to='.' from='*' />
</output>
<input message="xin">
<assign to='service1' from='*' />
</input>
</operation>
<operation name="Request1">
<participant name='RESTAPIClient'/>
<output message='xout'>
<assign to='url'>http://9.202.179.240:32163/B2BAPIs/svc/useraccounts/</assign>
<assign to='restoperation'>POST</assign>
<assign to='auth'>admin:password</assign>
<assign to="jsoninput1" from='service1/Output/text()'></assign>
</output>
<input message="xin">
<assign to='service2' from='*'/>
</input>
</operation>
</sequence>
</process>
```

Process Name: RESTAPIClient **Instance ID:** 144570

Service Name: RESTAPIClient

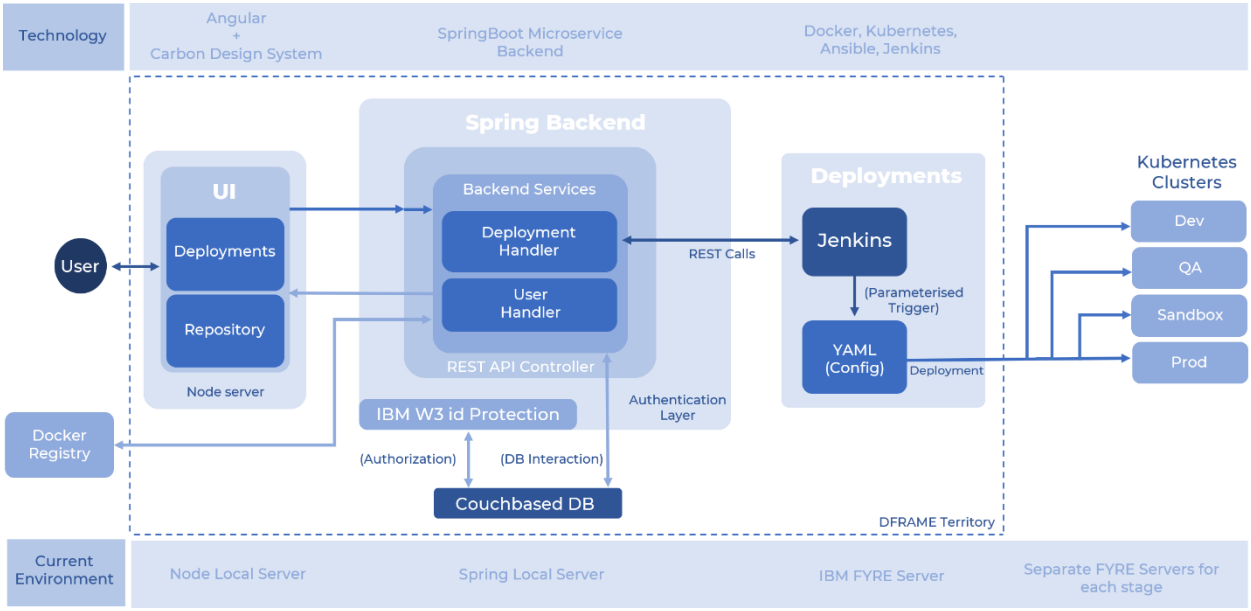
```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessData>
  <PrimaryDocument SCIObjectID="523105166d3b3fc7cnode1"/>
  <service1>
    <input_type>XML</input_type>
    <Output>{
 "password": "arpitdemo308",
 "permissions": {"element": {"name": "AS2"}},
 "surname": "demo3008",
 "givenName": "demo3008",
 "authenticationType": "Local",
 "userId": "demo3008"
}</Output>
  </service1>
  <service2>
    <URL_0>http://9.202.179.240:32163/B2BAPIs/svc/useraccounts/</URL_0>
    <REQUESTTYPE_0>POST</REQUESTTYPE_0>
    <Response_Code>400</Response_Code>
    <error>You have received an error!</error>
  </service2>
</ProcessData>
```

2.20 Committing changes to the Sterling Integrator V6.0.1 GitLab branch

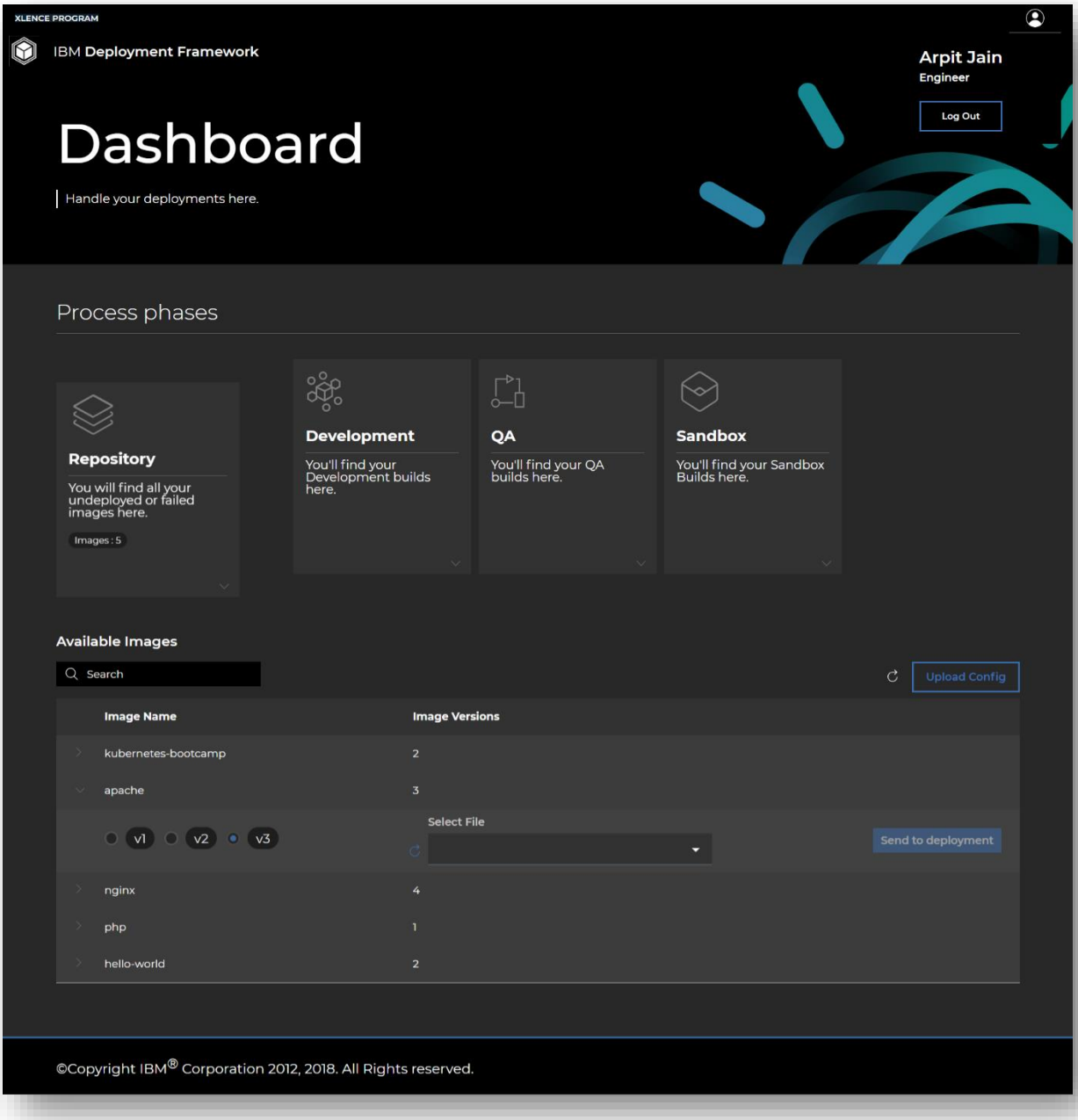
Once all the tasks pertaining to the Sterling Integrator were complete, the next task was to commit all the new feature changes to the Official Sterling Integrator branch. Due to confidentiality clauses, the details of this can't be revealed.

2.21 (Apart from the main work): DFRAME Docker deployment framework

As a part of the Xlence Program for the IBM interns, we developed a Docker image deployment framework that builds and deploys an Application Image in just a few clicks from the intuitive UI. Following figure will illustrate the exact functioning and architecture of this application.



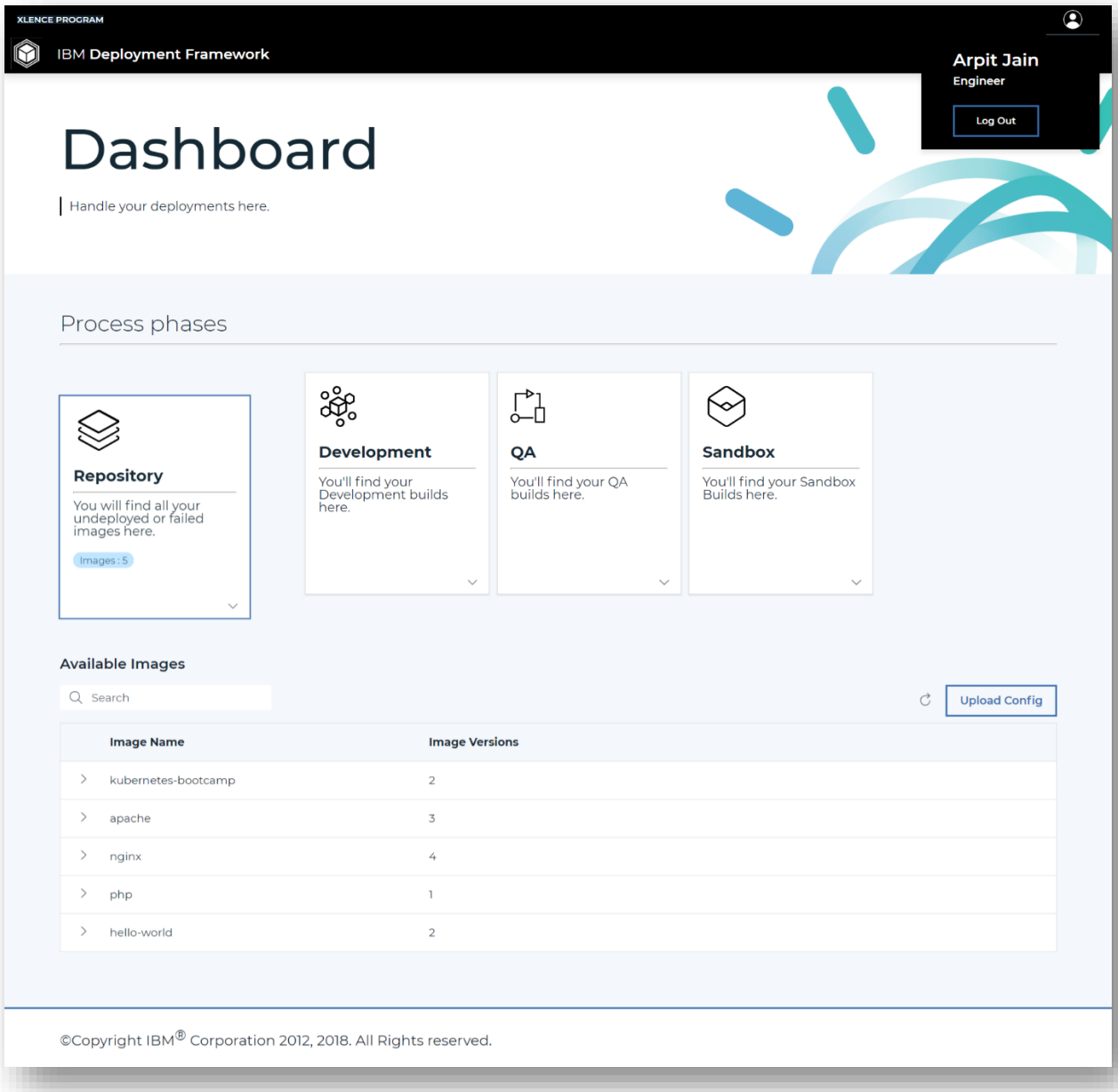
This is the main dashboard of the application.



When the IBM internal user successfully logs in using the IBM w3ID, He's being exposed to the Dashboard of the application. Depending upon his/her role in the team, for instance Engineer in the above example, he/she has access to the environments (Example – Development, QA, Sandbox, Production). Since, the above user is an engineer, he's not shown the Production environment.

So, upon the first glance, user is being shown the Docker images in his/her repository. Now, he/she can choose any of the available versions of any available image and then after choosing the Configuration YAML file from the drop-down, he/she can push that image version to Deployment. After this, that image is pushed to the Development environment and from there, it can gradually be pushed onto the next-level advanced environment (Example – Development Environment).

From the UI design point of view, we also worked on the Light theme mode (Primary) that is easy to the eyes. This is not any major functional change, but it still provides good intuitive User Experience.



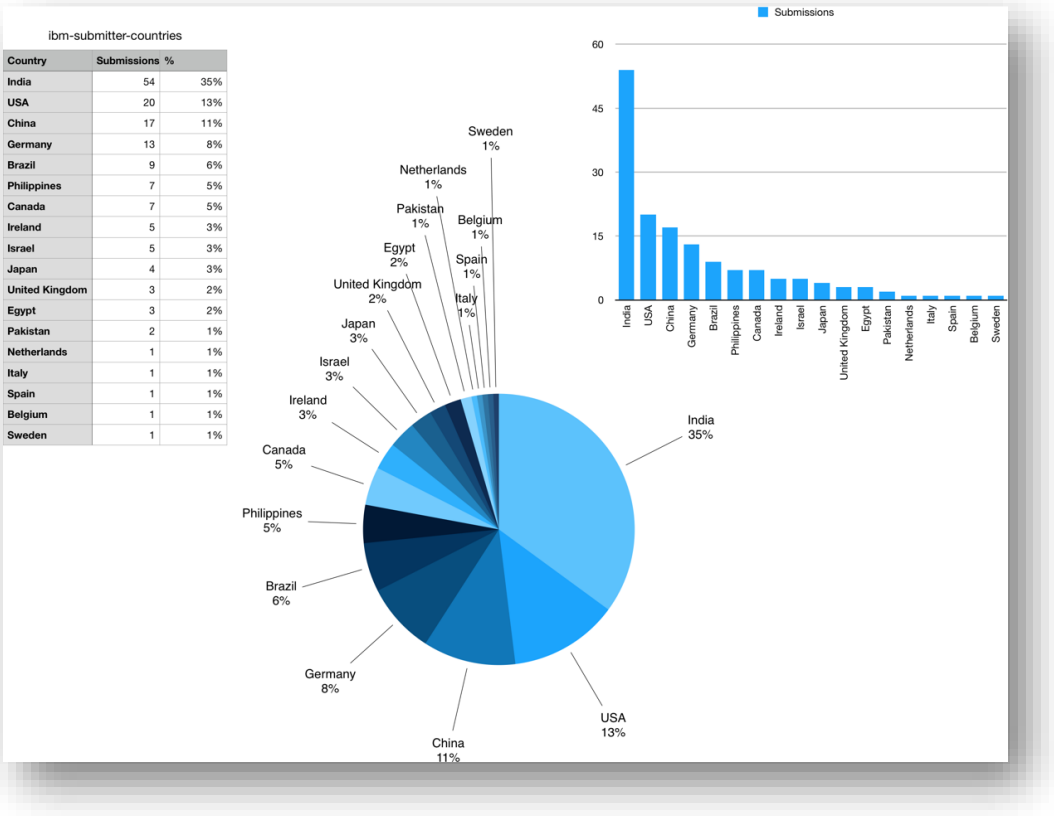
2.22 (Apart from the main work): DSENSE, IBM Call for Code

The Call for Code Global Initiative is a rallying cry to developers to use their skills and mastery of the latest technologies to drive positive and long-lasting change across the world with their code. For 2018, the IBM Call for Code Global Challenge asked developers to create solutions that significantly improve preparedness for natural disasters and relief when they hit in order to safeguard the health and well-being of communities. The Call for Code Challenge for 2018 is a competition that asks you to outthink natural disasters and build the solutions that will significantly improve the current state of disaster preparedness in your community and around the world.

As a part of IBM call for code initiative to smoothen disaster management and provide better post-disaster relief services, we developed a Strategic analytical piece of Web Application called **DSENSE.AI** which allowed the users (Most probably Government agencies) to be able to perform pre-disaster Human resource tracking.

We along with 128 teams from across the globe were selected for the global round of Call for Code, 2018. Although we didn't win the competition, but we were awarded digital badges by IBM for our Project submission and efforts.

You can verify the Badge **here.**



ibm-submitter-countries

Country	Submissions	%
India	54	35%
USA	20	13%
China	17	11%
Germany	13	8%
Brazil	9	6%
Philippines	7	5%
Canada	7	5%
Ireland	5	3%
Israel	5	3%
Japan	4	3%
United Kingdom	3	2%
Egypt	3	2%
Pakistan	2	1%
Netherlands	1	1%
Italy	1	1%
Spain	1	1%
Belgium	1	1%
Sweden	1	1%

Refer to the following Slides for further Information about the project. For further description and preview, refer **Appendix 5.1**

Dsense.AI

(Pre-disaster tracking of Human Resource)

Background.

PRESENT SITUATION

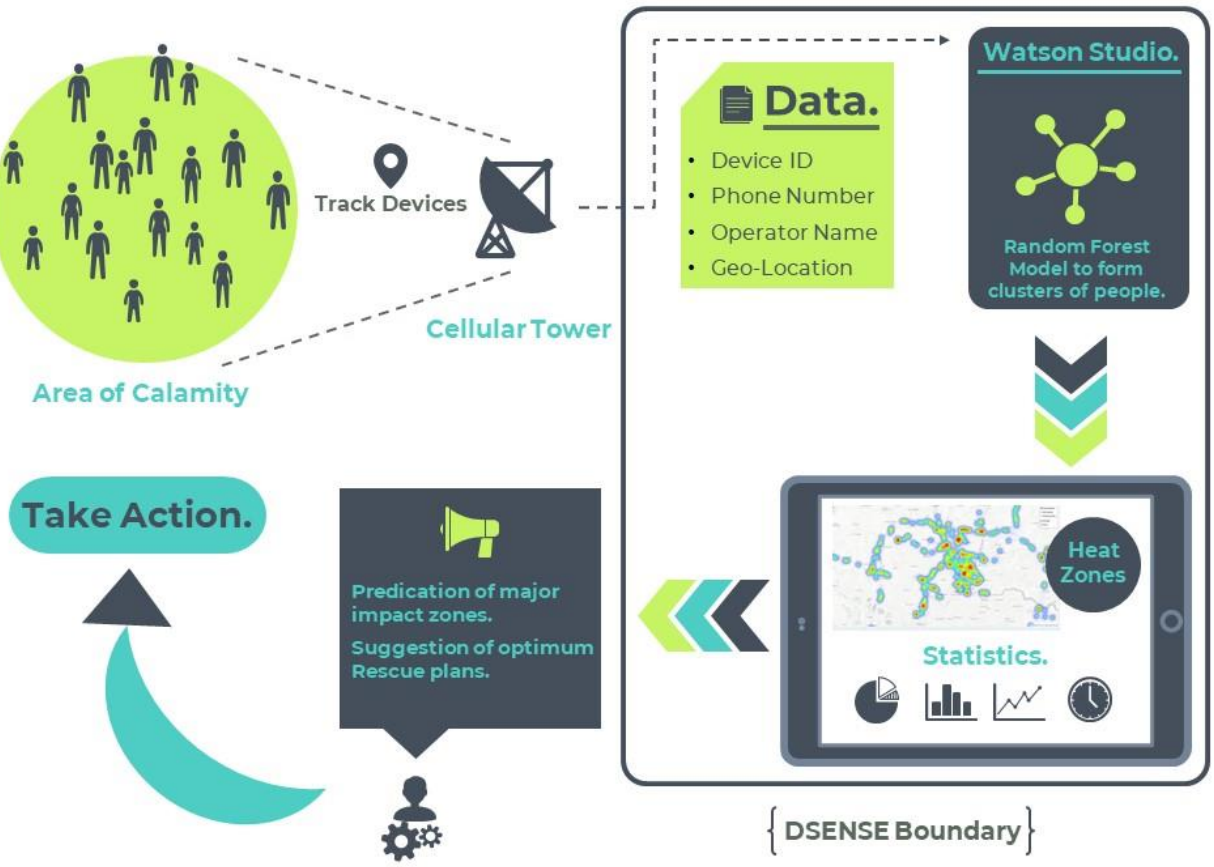
Before and during the occurrence of any Mass-scale disaster, there is a huge problem of Management and Estimation for an effective rescue process due to extreme hustle. There is no basis for early preparation as such. This is because there is no real time-detection and tracking of the scale of impact of any calamity.

We aim to predict the Scale of Impact of any calamity affected region by generating a Heat Map of the concerned region. With the availability of the estimated human entrapment density on the heat-map, medical and rescue agencies can prepare accordingly in prior. With the prediction of the regions with maximum impact we can then map them to give rescue agencies a heads up to where to start from with the rescue job or how much resources do we need to commit to any particular place.

Also, with the availability of this dynamically evolving data, one can take better actions for the future disasters with smarter insights.

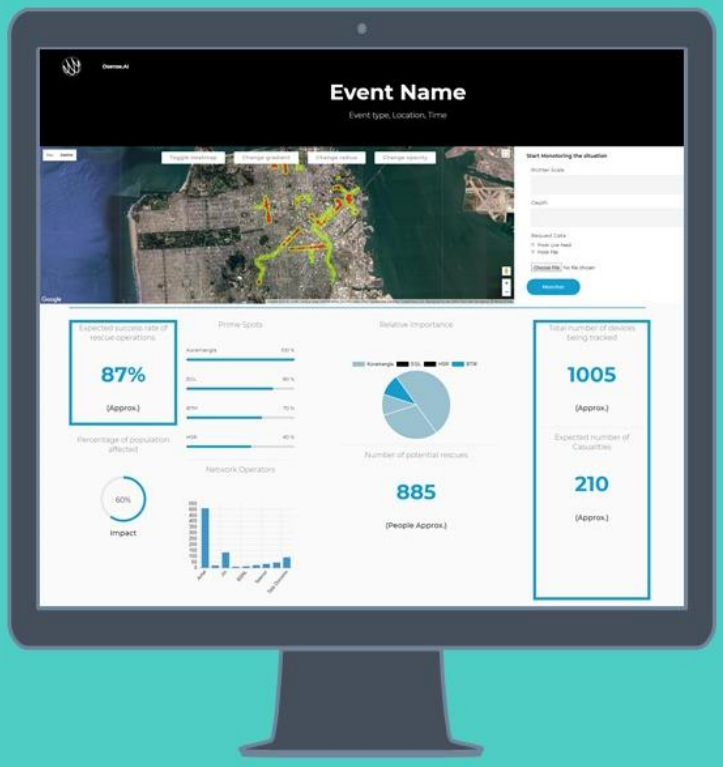
The Idea

- Whenever there's a high probability of an occurrence of any disaster, the cellular/mobile devices in that region, would be triggered to accept the location based tracking. The usage of this location based tracking would be strictly restricted to the duration of disaster. The Cellular towers in the area will start tracking the locations of all the cellular devices in that region. The obtained location data would then be utilized by the application to estimate the density of people trapped in a particular region. Also, the application would suggest the viable rescue plans keeping into record the statistics of that region. These plans would help the concerned rescue and disaster relief agencies to allocate the resources (food, medicines; etc.) accordingly.



Web App Snapshot

Event Monitoring dashboard



3. Results and Discussions

Chapter 3

1. Speaking on high-level, I got to know and learn about important Software-Industry level concepts. Some of these are – Microservice Architecture, Cloud deployment/development, Software as a Service (SAAS), QA Testing phases, Release Cycles, Code Management and Version Control.

2. The Tasks given to me were mostly stepwise in order. Hence, I needed to report at regular intervals with the current status of my job, and future work to my Mentor.

3. I got acquainted with the Business-Driven Methodology while working on Sterling Integrator Product. Also, I learned about Business Process Modelling using XML and XPATH standard.

4. Most of the initial days were mainly focused on familiarising me with the company's work methodology, performing initial Laptop Setup and getting Server permissions.

5. My knowledge of Java and OOP concepts in development improved drastically by working on the project in Eclipse IDE both locally and on remote Server.

6. My project dealt mainly with REST APIs hence my knowledge in that domain also improved.

7. I got to work with the implementation and reconstruction of the Business APIs in a modern framework – SpringBoot.

8. Also, for the first time, I was introduced to the concept of performance benchmarking of the APIs responses. I was able to pull off an entire test to check the performance of SpringBoot against the IBM TenX framework.

9. Apart from my Main project, working on the side Hackathon projects allowed me to manage time more judiciously.

10. Finally, I got to understand how things work in the real production cycle environments along with the business considerations and permissions.

Errors and Difficulties

1. Failure of POST and PUT operations just before the Demo.

On the day when the Demo meeting of our Service implementation was scheduled, nearly an Hour before the demo of the service to the team, POST and PUT service logics failed. We skipped the demonstration of the POST and PUT. Later we found out that it was merely a fault of a wrong statement order. Even before the connection was established (before opening the connection), we were requesting for the Response code and message. The program caught the exception and threw the error. Such merely small error caused such large trouble. That day I realized the importance of Pre-Demo testing.

2. Information passing from One service instance to Another.

In the last week of our first project completion, we had to implement the transfer of information (mainly input parameters) from one service to another within the same Business Process. The syntax of XPATH requires the Relative input paths (directory format) to be specified inside the Double quotes. The problem here is that the input received into these input fields themselves have

embedded Double quotes which renders the Whole input line invalid. As only possible solution, we had to substitute the double quotes by Escape Characters of XML (Like instead of **“** using **&quot**).

3. Proxy Connection support not added.

In the last few weeks of the internship, it was being planned that Proxy support must be added for the API Http Connections. Although, it was implemented on the local system, we were unable to implement on the Server for testing purposes. This was mainly due to limited availability of time.

4. Failing of the DFRAME – Deployment Framework

During the demo of the IBM deployment framework, when the image build was pushed from the Sandbox environment to the Production environment, the old version of the image already present in the production environment wasn’t replaced with the new version and it displayed an error notification. Of course, it was embarrassing for our team as the demo was being shown before the Director of the IBM Watson Supply Chain. Later, it was identified that the error was due to the poor handling of the Sessions between simultaneously logged in Users.

5. Failing of code builder after committing the final code changes

In the end when we committed all the changes to the GitLab Branch, the Builder which is used to compile all the source files and make a Jar out of it, failed due to non-availability of the external jars. Then these jars were instead of hardcoded path, were placed in the B2B base source directory itself for automatic access during code compilation.

4. Summary and Conclusions

Chapter 4

During this period of 6 Full Months, I came across various technologies that are being used in the industry and especially in IBM Watson Customer Engagement department. I got to learn and work on a lot of new things which will be very helpful for me to grow as a cognitive software developer. I came to know about the work culture of a company and how things go around in the professional business and technical world. I laid my hands on a lot of new technologies and learnt and implemented them successfully. During this period, I also learnt to work independently and as a team. I got to know the importance of collaboration and teamwork.

I have learnt quite a few new technologies and tools like Microservices, Service Oriented Architecture, Maven, Java Spring, DevOps, IBM Cloud cognitive service offerings; etc. I got a hands-on experience of JAVA and SpringBoot. I got acquainted to the Microservice architecture of software development, researched about the B2B processes and learned to code them. For implementing these ideas into practical uses, I understood the Code and working of IBM Sterling B2B Integrator.

Microservices are an effective and fault-tolerant manner of designing and developing an application. The heart of B2Bi is composed of several business processes which are used to automate the operation of B2B services (Mailbox, Invoice, Order; etc). By using REST APIs, you can perform certain B2B functions using Sterling B2B Integrator. B2B REST APIs were released starting with Sterling B2B Integrator V5.2.6.1. These Rest APIs were built using core Java and IBM TenX framework.

IBM and Sterling team is exploring about some alternative frameworks for converting these APIs which would make the execution of calls faster. Separate microservices can be deployed on the B2B SI platform server to enhance its functionality. During this duration, I transformed few of the B2B APIs to the SpringBoot framework to demonstrate to the team the potential performance improvement. All B2B APIs currently come packaged as a single WAR file. So, I took the goal to convert all these bundled APIs into separate Spring microservices and deploy them distinctively too.

Above all, I learnt from my mentor how to plan. I learnt small things like how to code efficiently and cleanly. I learnt how to think through things, how your requirements should be clear and then work on them maintaining the positivity of the Team even in the Hard-times.

5. Appendix

Chapter 5

5.1 Dsense.AI

OBJECTIVES

Our main objective is to make an estimate of number of people trapped in a calamity struck place and give a brief information to agencies in a intuitive way so that the agencies can take actions accordingly.

Our application provides an easy way to get to know about the demographics of any calamity before it actually occurs and that too in real time so that the immediate action is adopted.

Heat zones on maps will provide an convenient and more effective way of visualizing the actual condition of the disaster activity.

4. Resource allocation & Better Preperation

The Location and movement data obtained will be useful in training a predictive model for the better future preparations. Example - Building of Safe zones; etc.

5. Location preferences

Our model is smart in a way that it also takes into consideration the existence of several locations such as Schools, Markets (High Priority Zones) or Forests (Low Priority Zones) etc.

6. Post-disaster relocation

With the availability of the last traced location of any victim, it would be easier to find the person post-disaster.

6. Literature Citations

Chapter 6

6.1 Documentation of Sterling B2B Integrator.
https://www.ibm.com/support/knowledgecenter/en/SS3JSW_5.2.0/welcome/product_landing.html

6.2 Developing Business Processes – Best Practices.
https://www-01.ibm.com/support/docview.wss?uid=swg27043594&aid=1

6.3 Business Process Modelling Book.
ftp://public.dhe.ibm.com/software/commerce/doc/sb2bi/v5r2/SI52_BP_Modeling_book.pdf

6.4 Writing Custom services in IBM Sterling B2B Integrator.
https://www.ibm.com/developerworks/community/blogs/SterlingB2B/entry/Writing_Custom_Services_in_IBM_Sterling_B2B_Integrator?lang=en

6.5 Creating custom services and adapters.
https://www.ibm.com/support/knowledgecenter/en/SS3JSW_5.2.0/com.ibm.help.manage_svcs_adpts.doc/CS_AboutAddingNewSvcToSI.html

6.6 XPath and Process Data.
https://www.ibm.com/support/knowledgecenter/en/SS3JSW_5.2.0/com.ibm.help.bpml.doc/SI_XPathAndProcessData.html

6.7 Update, Compile and Redeploy WAR files in ISBI.
https://www.ibm.com/support/knowledgecenter/en/SS3JSW_5.2.0/com.ibm.help.web_extns.doc/WebExtns_UpdtCmplRdplyWAR.html

6.8 Getting started with SpringBoot.
https://spring.io/guides/gs/spring-boot/

6.9 SpringBoot Learning Practices.
https://www.baeldung.com/spring-boot

6.10 An analysis of XML and JSON.
https://www.cs.tufts.edu/comp/150IDS/final_papers/lizzied.3/FinalReport.html

6.11 World of Integration: XPath usage in SI.
http://www.worldofintegration.com/content/xpath-functions-all

6.12 Getting Started with Angular JS.
https://angular.io/guide/quickstart

6.13 Microservices with SpringBoot.
https://www.edureka.co/blog/microservices-with-spring-boot

6.14 Adding W3ID SSO authentication to your Apps.
https://console.bluemix.net/docs/customer-portal/cpmanacctconfsso.html#customerportal_confssoserv

7. Publications

Chapter 7

7.1 “REST client adapter- B2B Integrator”.

We wrote an IBM DeveloperWorks Article. Our Article encompasses the research work that we went through. Using our proposed service, Clients would be able place CRUD calls to B2B APIs using Business Processes.

Link
https://www.ibm.com/developerworks/community/forums/html/topic?id=13052465-b633-4480-8515-31b02333a3a3&ps=25

7.2 “XML – JSON Transformer”.

We wrote an IBM DeveloperWorks Article. Our Article encompasses the research work that we went through. Using our proposed service, Clients would be able to convert data in their required format using Business Processes.

Link
https://www.ibm.com/developerworks/community/forums/html/topic?id=13052465-b633-4480-8515-31b02333a3a3&ps=25

7.3 “B2B APIs to SpringBoot”.

We wrote an IBM DeveloperWorks Article. Our Article encompasses the research work that we went through. This article demonstrates how previously built TenX based APIs can be converted to SpringBoot framework.

Link
https://www.ibm.com/developerworks/community/forums/html/topic?id=13052465-b633-4480-8515-31b02333a3a3&ps=25