

# Application of Microservice Architecture on B2B Processes

(IBM Watson Customer Engagement)

*by*

Vipin Dhonkaria  
(Roll No. 2015274)

Supervisor(s):

External

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

Internal

Dr. Manish Kumar Bajpai  
(PDPM IIITDM Jabalpur)



Computer Science and Engineering

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND  
MANUFACTURING JABALPUR

(8<sup>th</sup> September 2018 – 21<sup>st</sup> September 2018)

# Introduction

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.

IBM aims to bring Businesses closer and smarter than ever with the help of their state of the art enterprise software product called B2B Sterling Integrator. IBM B2B Integrator helps companies integrate complex B2B (Business to Business) / EDI (Electronic Data Exchange) processes with their partner communities. IBM aims to transform the B2B Sterling product into Microservice architecture.

## Brief Overview

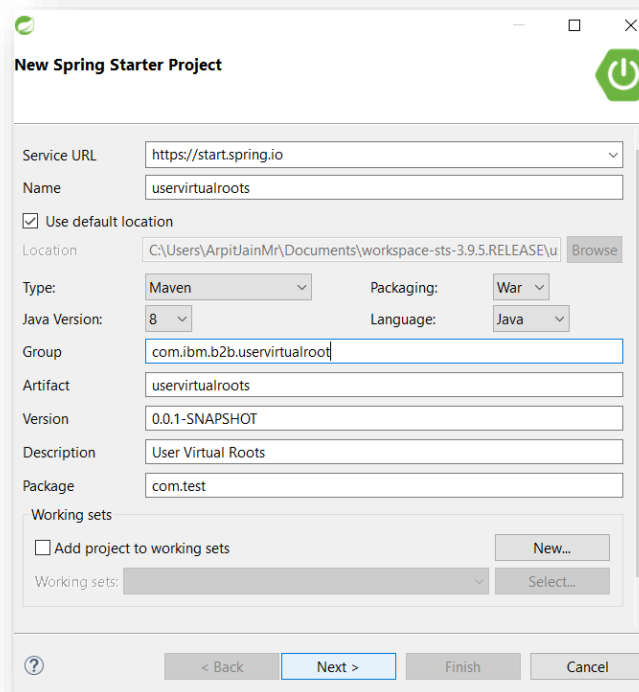
During the duration of my last report, I created a sample Spring Boot REST application and deployed it on the Sterling Integrator (SI) WebSphere Liberty server. Now, my task is to convert the existing B2B Rest APIs to the Spring Framework and then compare the execution call time of the of the modified Spring APIs with the previously created IBM TENX APIs.

In this Report, I converted a very simple B2B API – UserVirtualRoot to Spring framework and deployed it on SI-Liberty server along with Swagger Hub API documentation Integration.

## Report on the Present Investigation

(Progress during this 15-days period)

Create a basic Spring Starter Project



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

Now Update the pom.xml file in the root directory for dependencies .

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <artifactId>uservirtualroot</artifactId>
  <name>uservirtualroots</name>
  <description>uservirtualrootsapi</description>
  <packaging>war</packaging>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.5.RELEASE</version>
  </parent>

  <properties>
    <java.version>1.7</java.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <start-class>com.ibm.b2b.api.svc.uservirtualroot.WebApplication</start-class>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>io.springfox</groupId>
      <artifactId>springfox-swagger2</artifactId>
      <version>2.9.2</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>io.springfox</groupId>
      <artifactId>springfox-swagger-ui</artifactId>
      <version>2.9.2</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>javax.persistence</groupId>
      <artifactId>persistence-api</artifactId>
      <version>1.0.2</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <executable>true</executable>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

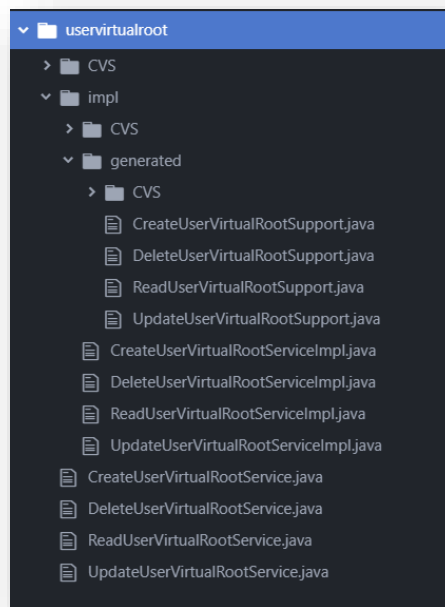
```

<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.6.1</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <showWarnings>true</showWarnings>
        <compilerArgs>
            <compilerArg>-Xlint:all</compilerArg>
            <compilerArg>-Werror</compilerArg>
        </compilerArgs>
    </configuration>
</plugin>
</plugins>
</build>
</project>

```

Change the Structure of Java Source files

- Firstly, In the Tenx implementation, the source file structure was as specified below –

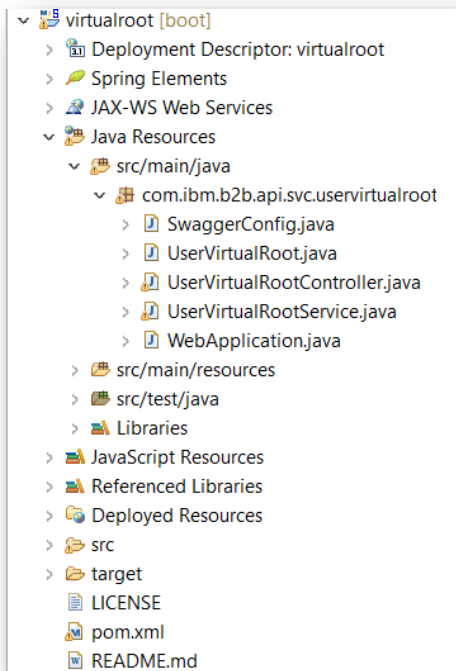


Here, there are three  
4 CRUD operations –

types of files for each of the

1. UserVirtualRootService class: An interface that acts as an entry point to the service (Create, Read, Update or Delete). This interface calls the functions specified in the UserVirtualRootSupport Class.
2. UserVirtualRootSupport class: An abstract class that contains the methods to call the UserVirtualRootServiceImpl functions. This class basically performs the transformation of the input parameters for calling the Impl functions.
3. UserVirtualRootServiceImpl: The class which performs the actual implementation of the CRUD operations.

- Secondly, with SpringBoot, the above source files are needed to be modified into the following structure.



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

1. POJO Classes: Classes to represent the object entity structure.

UserAccounts.java

UserGroupName.java

AuthorizedUserKeyname.java

PermissionName.java

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

UserAccountsController.java

```
package com.ibm.b2b.api.svc.uservirtualroot;
import java.io.Serializable;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;
import java.lang.String;
import java.util.List;
import java.util.HashMap;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
```

```

@RestController
@Api(value="/api",produces="application/json")
@RequestMapping("/api")
public class UserVirtualRootController implements Serializable {

    @Autowired
    UserVirtualRootService userVirtualRootService;
    @ApiOperation(value="UserVirtualRoot API reference",response=com.ibm.b2b.api.svc.uservirtualroot.UserVirtualRoot.class)
    @ApiResponses(value={
        @ApiResponse(code=200,message="API is up for function",response=com.ibm.b2b.api.svc.uservirtualroot.UserVirtualRoot.class),
        @ApiResponse(code=500,message="Internal Server Error"),
        @ApiResponse(code=404,message="Not found")
    })
    @RequestMapping(method=RequestMethod.GET,value="/{userName}")
    public List<UserVirtualRoot> read(@PathVariable String userName){
        System.out.println("Input url userName" + userName);
        System.out.println("Param passed to service class");
        return userVirtualRootService.read(userName);
    }
}

```

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

#### UserAccountsService.java

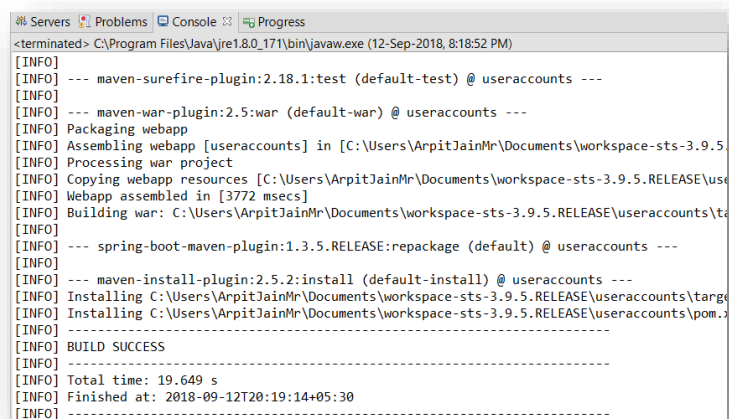
4. Web Application Class: This class is used to start the SpringBoot.

```

WebApplication.java
package com.ibm.b2b.api.svc.uservirtualroots;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.web.SpringBootServletInitializer;
import springfox.documentation.swagger2.annotations.EnableSwagger2;
@SpringBootApplication
@EnableSwagger2
public class WebApplication extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(WebApplication.class);
    }
    public static void main(String[] args) throws Exception {
        SpringApplication.run(WebApplication.class, args);
    }
}

```

Package the Application as a WAR file using Maven Install



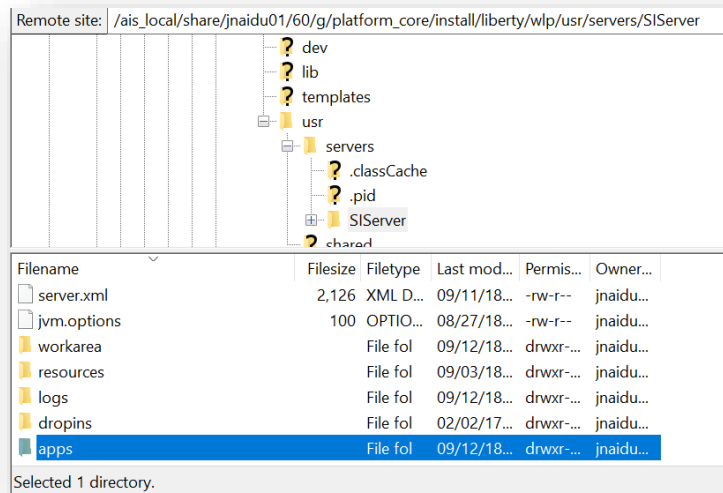
```

C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (12-Sep-2018, 8:18:52 PM)
[INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ useraccounts ---
[INFO] --- maven-war-plugin:2.5:war (default-war) @ useraccounts ---
[INFO] Packaging webapp
[INFO] Assembling webapp [useraccounts] in [C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\war]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\classes]
[INFO] Webapp assembled in [3772 msecs]
[INFO] Building war: C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\useraccounts.war
[INFO] --- spring-boot-maven-plugin:1.3.5.RELEASE:repackage (default) @ useraccounts ---
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ useraccounts ---
[INFO] Installing C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\useraccounts.war to C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\maven-repository\useraccounts\war\3.9.5
[INFO] Installing C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\pom.xml to C:\Users\ArpitJainMr\Documents\workspace-sts-3.9.5.RELEASE\useraccounts\target\maven-repository\useraccounts\pom\3.9.5
[INFO] BUILD SUCCESS
[INFO] Total time: 19.649 s
[INFO] Finished at: 2018-09-12T20:19:14+05:30
[INFO]

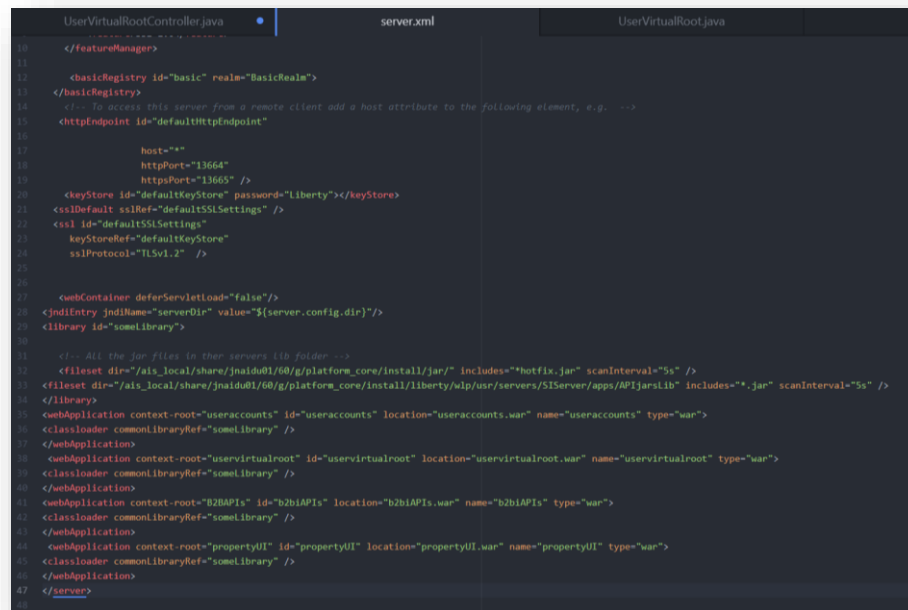
```

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.



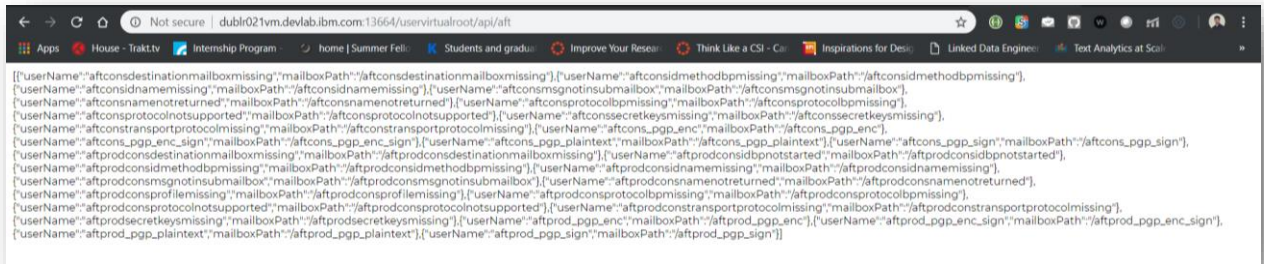
Restart the Server

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

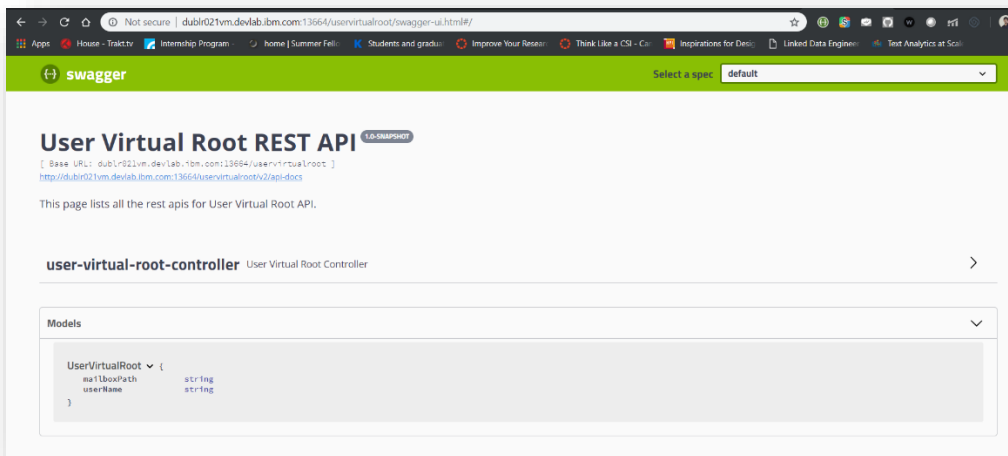
```
./hardstop.sh -> ./run.sh -> ./startLiberty.sh
```

Access the API on the specified URL

Currently, it is just implemented for a Read Call.



Access the Swagger Hub API Documentation as follows



## Results and Discussions

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.

## Conclusions

During past few weeks, I was involved in converting a small B2B API like UserVirtualRoot to Spring Boot Rest API Framework and deploy it in the form of a WAR file on the liberty server of the Sterling Integrator. All the IBM B2B APIs are currently built over the IBM TENX framework that is slow in function as it has useless layering of procedure calls. My team wants to test if converting these APIs to Spring Boot framework will make them faster in execution. For this purpose, currently I am assigned for POCs (Proof of Concept).

### Next Target

My target for the next 15 days is to transform a large API like UserAccounts to the Spring Boot framework and measure how fast is it when compared to the older IBM TenX framework