St. Cloud State University
theRepository at St. Cloud State

Culminating Projects in Information Assurance | Department of Information Systems

3-2018

Using Shared Memory as a Means to Provide Data Concurrency Across Vm's in a Cloud Architecture

Shravani Meneni
St. Cloud State University, smeneni@stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/msia_etds

Recommended Citation
Meneni, Shravani, "Using Shared Memory as a Means to Provide Data Concurrency Across Vm's in a Cloud Architecture" (2018). *Culminating Projects in Information Assurance*. 63.
https://repository.stcloudstate.edu/msia_etds/63

This Starred Paper is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

Using Shared Memory as a Means to Provide Data Concurrency Across Vm's in a Cloud Architecture

by

Shravani Meneni

A Starred Paper

Submitted to the Graduate Faculty

of St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in

Information Assurance

March, 2018

Starred Paper Committee:
Dennis Guster, Chairperson
Susantha Herath
Balasubramanian Kasi

Abstract

As the world is progressing towards full adoption of the - WWW (World Wide Web), communication of data becomes more vital, especially when it must be achieved by making use of Restful web services. Restful web services enable the ease to transfer data between virtual machines on the cloud or the virtual machines hosted on-premises. The main idea of the project is to showcase that data replication is possible between Primary HOSTS and Secondary HOSTS, with the primary host responsible for sharing the data with other virtual machines, by making use of a REST API. Further, any changes made to the data resided in the primary host must be reflected in the secondary hosts. Concepts of virtualization, cloud computing and Rest API are used here in this paper to achieve the goal of this project.

Table of Contents

Page

List of Tables 6

List of Figures 7

Chapter

I. Introduction 10

Definition of Terms 13

II. Literature Review and Background 14

Advantages of Virtualization 14

Need for the REST API 16

Advantages of the REST API 17

Shared Memory 18

III. Methodology 19

Scope 19

Design Approach 19

Primary Host 19

Secondary Hosts 19

Design of Study 20

Tools and Techniques 22

Java 1.8 22

Spring Boot 22

Tomcat 7.0.81 23

Chapter Page

JPA 23
IntelliJ IDEA 23
MySQL5.5 Server 24
Gradle 24
Technology Stack 24
UML Diagrams 25
IV. Implementation 31
Gradle Setup 31
MySQL Serve Setup 31
Install MySQL Server 32
Configure Environment Variables 35
Install Java 39
Install Tomcat 40
Configure Tomcat 41
Technical Design Explained 43
Primary Host 43
MySQL Data Base Schema 45
Secondary Host 54
Setting up an IDE for the Development Environment 55
Build and Deployment 61
V. Conclusion 73

Page

References.....74

Appendix.....76

List of Tables

Table Page

1. Glossary of Different Terms Used in This Paper.....13

2. Classification of API's.....17

3. Student Table Schema.....45

4. Log Table Schema 45

List of Figures

Figure Page

1. Database Server Using Shared Memory to Enable Virtual Processors 18
2. Architecture.....20
3. High Level Design 20
4. Data Model in MySQL 21
5. Class Diagram.....21
6. Get Request Sequence Diagram.....25
7. POST Request Sequence Diagram.....25
8. PUT Request Sequence Diagram.....26
9. DELETE Request Sequence Diagram 26
10. Secondary Host Get Sequence Diagram 27
11. Project Structure.....28
12. Primary Host Technical Design 43
13. Secondary Host Technical Design 43
14. Create Student Request 46
15. Update Student.....48
16. Database with Updated Record.....48
17. DELETE Request on Postman.....49
18. Deleted Record in the Database 50
19. Secondary Host Changes 55
20. Development Environment in IntelliJ IDEA 56

Figure Page

21. Gradle Settings 57
22. Run /Debug Configuration 58
23. Window that Appears to Create a Configuration 58
24. Configuration is Created Enabling us to do a Debug or a Run 59
25. Client Cron Setup Instructions 60
26. Window that Appears to Create a Cron Configuration 60
27. Setup Instructions for Cron 61
28. Application BootRun Option 62
29. Build Successful Message on Command Prompt 63
30. War File Generated 63
31. Tomcat Configuration 64
32. Tomcat Deployment 65
33. War File Generated in Tomcat Folder 66
34. Student Details Displayed 66
35. Jar File Being Generated 67
36. Jar File Generated Inside the Build Folder 67
37. Execution of Cron Job 68
38. POST Operation 70
39. Database Before Modification 70
40. PUT Operation 71
41. Modified Date in Log Table 71

Figure Page

42. Student Table After Modification 71

43. Updated Data is Printed as a Response in the Cron Job Fetch 72

44. Log Table Cron Job 72

Chapter I: Introduction

This chapter gives a brief introduction about Data replication between virtual machines through the use of a REST API.

The world is progressing towards the complete adoption of the internet, and hence it is necessary to explore the possibilities for sharing the pool of data between terminals, virtual machines, and hosts, making it easier for the Database Administrators (DBAs). Anyone who can understand the REST API can easily access the data. More importantly, a monitoring tool can be built to check the status of back-end systems, and better log management is made possible.

REST stands for Representational State Transfer, which is an architectural style for designing networked applications. Earlier, communication between machines was achieved using complex methods like SOAP, CORBA, and RPC. Now, with the REST protocol in place, they can communicate directly through the HTTP protocol. RestFul applications make use of HTTP requests to read, create, update and delete data through CRUD operations.

In the existing approach, sharing or updating data between a Master-Slave database (DB) is possible through database administration, SOAP, and RPC. However, DBAs can only understand the replication procedure as well as the related issues involved with the process. Also, it is time-consuming to analyze the log files to resolve any issues with them, and there are no monitoring tools available.

Remote machines and the use of virtualization are considered as the essential inside handling development techniques in current use. Besides these, the use of VM (Virtual Machine) communication is a step beyond, and it is also recognized as one of the primary roots of data

heightened structures and applications in most of the server-based systems and disseminated processing circumstances (Lombardi & Di Pietro, 2010). One way to improve the VM communication capability is to reinforce the local VM communication by making use of information exchange methods and fall back on routine TCP/IP for exchange between remote systems that are on different physical machines.

Recently, another approach is concentrated on upgrading communication capability between local VMs using shared information documents, and the change fluctuates with each different way the standard memory channel is set-up. However, this paper gives a clear outline of the setup of choices and systems for execution of local VM communication (Pearce, Zeadally, & Hunt, 2013).

This project is configured with on premise (local machines), but this could also be deployed in a cloud environment. In the same way, it can also be implemented on Virtual Machines and made accessible to other users. When trying to get access to the code on VM's for security purposes, it is available only by making use of a VPN (Virtual Private Network). When it comes to large organizations, users are provided with tokens (which generate a random number) to get access to the VM'-s, and this is the most secure way as these tokens are used every time to gain access to the VM'-s. When it comes to the cloud, it is a similar process. The advantage of having it in the cloud is that, based on the traffic, the number of users, and CPU'-s the number of systems could be increased. At the same time, it acts more drastically, and thus it incurs an enormous cost (Gurav & Shaikh, 2010).

In the existing approach, if data has to be replicated, the database administrators must handle the data on their systems and every other system. DBAs will set up one primary MySQL database server and two secondary MySQL database servers. Using MySQL scripting they will have enabled data replication in place. Following this approach might cause a delay in updating the data, and it might generate errors with frequent changes. Also, if the number of slave databases are increased, it becomes riskier for the DBA. Though an automated process can accomplish this, there is no mechanism to monitor for errors. Any issues encountered can only be understood and then solved by the DBA. For this reason, it is not considered as a reliable approach (Tutorials Point, n.d.-a).

Making use of REST API will help us to handle these issues. REST has few rules that must be incorporated into the application, which will make it REST specific and easy for us to understand. Using the CRUD operations, we can communicate with the backend efficiently to handle HTTP requests, and support post, put, delete and get commands. The response format of REST is in JSON (Javascript Object Notation) and JSON is preferred over XML, as it is more lightweight (Tutorials Point, n.d.-b).

This approach does not need a DBA to administer the data because REST APIs will handle the replication and other related tasks. The only thing to be considered with this approach is to implement the REST API with all of the necessary infrastructure considerations taken into account, like the application server. In this method, Tomcat is used to host the application, which will communicate to the backend using the REST API. The advantage of using the REST API is that, if the same data must be replicated in multiple systems, it is not required to copy the same code manually each time. Instead, we can have a build system like GRADLE or MAVEN, which

will create a WAR file during the build process and then this WAR file can be directly deployed into Tomcat.

Definition of Terms

Table 1

Glossary of Different Terms Used in This Paper

TERMS	ABBREVIATION
API	Application Programming Interface
JSON	Java Script Object Notation
REST	Representational State Transfer
CORBA	Common Object Request Broker Architecture
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
VM	Virtual Machine
XML	Extensible Markup Language
DBMS	Database Management System
PK	Primary Key
FK	Foreign Key
HTTP	Hypertext Transfer Protocol
Tomcat	Apache Tomcat
WAR	Web Application Resource
MySQL	Structured Query Language

Chapter II: Literature Review and Background

According to Wood et al. (2009), various data center virtualization courses of action, for instance, VMware ESX, use content-based page sharing to have the advantage of using multiple servers. The concept of page sharing involves the technique of comparing virtual machine memory pages to an uncertain substance and then coalesce them into a single shared page. This type of system, when executed at the host level, is applied just between the VMs that are put together on a given physical host. In a multi-server environment, the chances of sharing may be greater in the light of the fact that the VMs holding undefined pages are localized on multiple servers. To make use of content-based page sharing it is critical to put virtual machines to such a degree as possible, to the point that VMs with practically identical memory substance are arranged on similar hosts.

With the concept of virtualization, multiple VMs can be executed in parallel on a single processor. Also, various operating systems can co-exist on the same physical platform. Traditionally when it was first introduced, it was mainly used in server environments with a motive to increase the usage and availability of resources whenever required. Now-a-days, it is used in embedded systems as well. More recently, considering all the factors like performance, power, security and safety of the embedded domain, researchers have put their efforts into developing more effective solutions to the embedded virtualization environment (Heiser, 2008; Varanasi & Heiser, 2011).

Advantages of Virtualization

Following are advantages of virtualization:

- Dynamic Load-balancing

- Disaster Recovery
- Server Consolidation
- Testing and Development
- Improved System Reliability
- Security

Cloud computing depends on the concept of virtualization to share its resources with the end users over the web. It is composed of both a distributed and VM computing infrastructure. There are three different ways in which the cloud provides its services to the end user (Albeshri & Caelli, 2010). They are (a) infrastructure as a service, platform as a service, and (c) software as a service (Albeshri & Caelli, 2010).

According to Ren et al. (2016), virtual machines (VMs) and virtualization are one of the core computing technologies today. When VM communication is considered, it is the most basic and one of the primary roots for data concentrated structures and applications in most server-based environments and cloud computing. Making use of local VM communication is considered as an essential step in improving the intercommunication between VMs more efficient, which indirectly makes use of REST API for communicating between different VMs located on different machines.

Also, when the sender VM and the receiver VM are co-resident on the same physical host, the data can also be transmitted to the shared host and bypass the long method for the TCP/IP system stack (Wang, 2009).

When the sender VM and the receiver VM are on different hosts, the data will be sent from sender to receiver through standard TCP/IP protocol stack. To develop such a typical

information exchange between VM communication channels, it is required to recognize the following limitations:

- Get every possible data request, dissect it, and recognize whether the recipient VM is a co-resident with the sender VM on the similar host.
- Maintain both adjacent and remote midway VM communication techniques, and an unending supply of neighborhood inter-VM communication, which enables switching and redirects the dynamic data to the shared memory-based station.
- Twist the simple memory-based inter-VM communication into the current virtualized system in a compelling and distinct route over existing programming layers, which can be complex (Burtsev et al., 2009).

In earlier days, communication between different hosts was achieved by making use of client/server architecture. In this kind of approach, each time a client wants to collect the data from the server, it has to send it a request first. The request must be accepted and acknowledged by the server, and then the communication is established between them (Muthunagai, Karthic, & Sujatha, 2012).

Need for the REST API

REST API will make communication between machines through the HTTP protocol. RESTFUL applications use HTTP requests to read, create, update and delete data through CRUD operations which are easy to understand. APIs can also act as a network between the software application and the operating system. It also serves as a guide between different applications by directing them in each step.

Table 2

Classification of API's

Web Service APIs	SOAP XML-RPC and JSON-RPC REST
Library-based APIs	JavaScript TWAIN
Class-based APIs	Java API Android API
OS Functions and routines	Access to file system Access to user interface
Object remoting APIs	COBRA .NET Remoting
Hardware APIs	Video acceleration Hard disk drives PCI buses

Advantages of the REST API

Among all of the available APIs, REST is best used for service application development. REST is the simpler data processing solution. The four most-important operations: GET, PUT, POST, and DELETE will also make it easier to obtain a uniform interface and simplifies the data transmission.

It acts as an interface between any systems using the HTTP call, which will enable the data collection process. It will also allow us to perform all sorts of operations on the data and generate output formats as required. It is lightweight and more flexible in comparison to any other API currently available. Following are a few advantages of REST for development.

- Separation between the client and server.
- Visibility, reliability and scalability.
- It is both platform and language independent.

Shared Memory

Memory that can be shared and accessed by a different number of programs is called shared memory. Multiple applications can easily communicate with each other by making use of shared memory. It can be considered the easiest and fastest means of communicating with different applications.

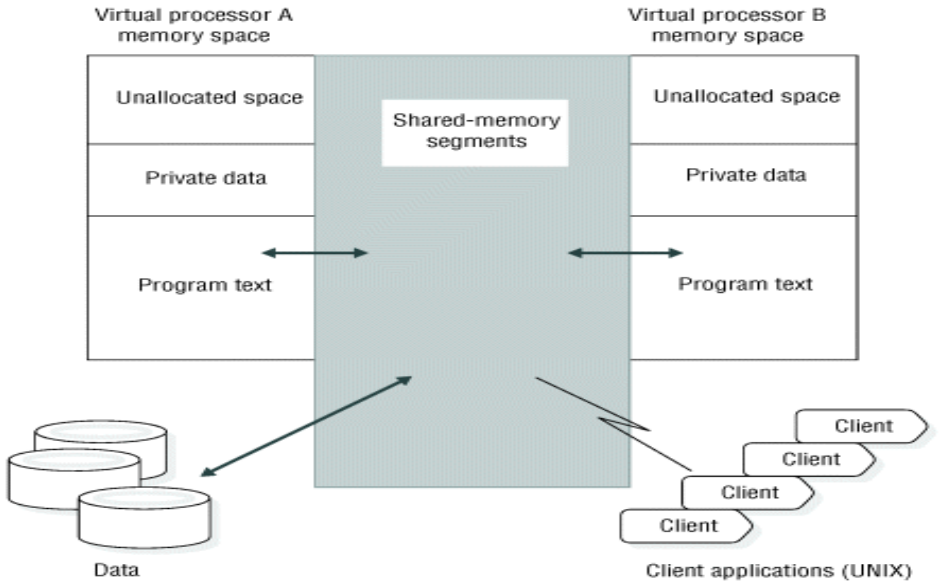


Figure 1. Database Server Using Shared Memory to Enable Virtual Processors

Chapter III: Methodology

Scope

The main scope of this project is to showcase the ability to share the data between master and slave nodes using the Rest API. Any changes in master data should be replicated in the slaves. Any changes made in slaves will be overridden by server changes as server changes require the highest priority.

Design Approach

A primary host and two secondary hosts are considered for the design and implementation. The data resides in the primary host and all updates occur on the primary host. Data is replicated in the secondary hosts periodically using REST API exposed by the primary host. A standard application is built to serve for both the primary and the secondary hosts.

Primary Host

- Server application is built to serve CRUD operations through the REST API.
- When the data is created, **createdDate** is updated.
- When the data is updated, **lastModifiedDate** is updated.
- Log Table is available to manage tables and **lastFetchedDate** in the application.

Secondary Hosts

- Secondary hosts use the same application except that, the client uses a pull approach to update its DB.
- Cron job is triggered on a specified time-frame to see if there are any updates on the Server Data.

- If anything is updated, the delta which is greater than "**lastFetchDate**" is returned in a JSON format.
- Once the response is fetched, the POST call is made to the secondary host with the delta.
- All of these are handled in a separate thread for performance optimization reasons.

Design of Study

The following diagram explains the architecture used for this project. Secondary virtual hosts check for updated Delta (i.e., data) and make a GET request. Each time when the data is updated in the primary host, the database gets updated, from their it sends the data to the secondary virtual hosts.

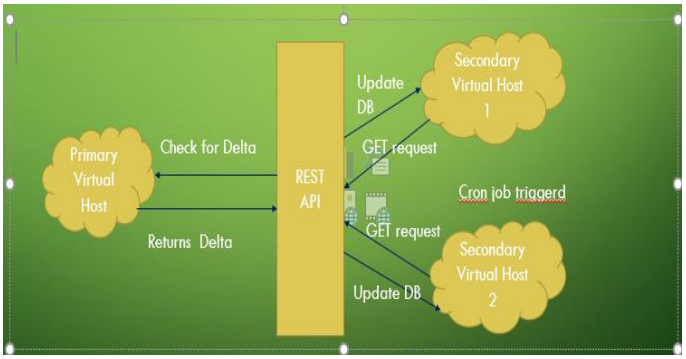


Figure 2. Architecture

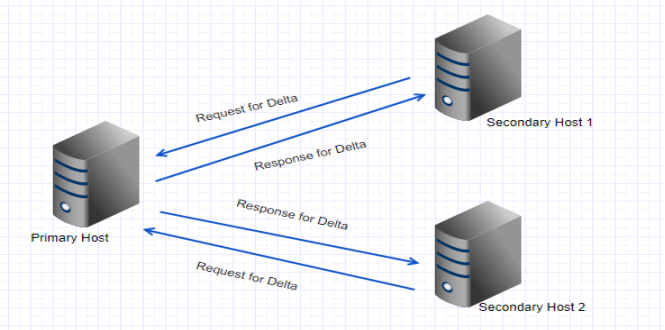


Figure 3. High Level Design

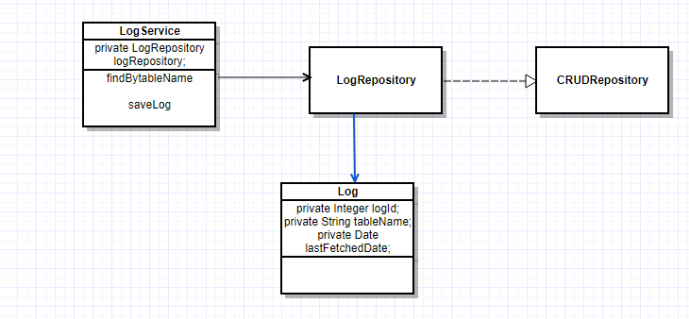


Figure 4. Data Model in MySQL

The Student Controller uses log service through autowiring and updates the last fetched date if the GET request is triggered from a Cron job.

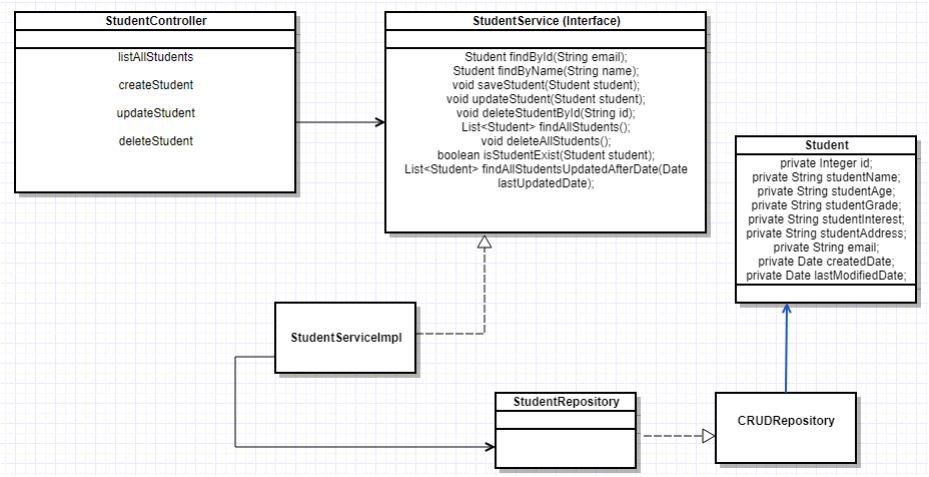


Figure 5. Class Diagram

- Student Controller: The class is mapped with @RestController annotation, which is readily used by Spring MVC to handle web requests. The controller also has @RequestMapping annotation, which will assign the API call to the corresponding method implementation. The controller is auto-wired to the service layer using annotation @autowired. The controller acts as a serving point for any CRUD operations.

- Student Service (Interface): This interface exposes all necessary operations that need to be performed on the Student entity.
- StudentSerivceImpl: This class implements all the interface methods exposed by student service, which depends on the repository through @autowiring.
- StudentRepository: The Interface implements a CRUDRepository interface, which will have a default behavior for all CRUD related operations to be performed on the entities.
- CRUDRepository: The interface provides most of the CRUD operation methods needed. Any additional methods are written to the custom repository.

Tools and Techniques

Java 1.8. Java is a high-level programming language. It was developed by Sun Microsystems, and was released in 1995. It is simple, robust, dynamic, platform independent and a flexible programming language compared to many other languages. With each version that has been released, there are new features that have been added. After Java 1.5 was released, Java 1.7 called Dolphin has been the most significant update. The version 1.8 release is the latest and has more available features compared to earlier versions, Java 1.8 is what is used in this project.

Spring boot. The project uses Spring boot which makes the creation of stand-alone applications easy. Spring boot has embedded Tomcat and Jetty enabling easy deployment especially if there is no specific need to generate the .war file. It also allows for less configuration in XML for the Spring framework and provides a starter configuration for Maven files which, in turn, makes it easy to kick-start any web application development. It is also easy to integrate with Gradle, as it is the most used build management system.

Tomcat 7.0.81. It implements almost all the Java EE specifications like Java Servlets, JSP, Java Expression language and Web socket. With all of these specs, Tomcat makes it easy to run the Java code in an HTTP server environment. It is light-weight, open-source, highly flexible and fully secured. It is also a stable platform that makes the Java applications run more smoothly.

JPA. Java Persistence API (JPA) provides a better way for the developers to easily access and manage data between Java applications and its associated database, by making use of Object Relational Mapping. It is easy to understand and easy to implement.

Overall the application uses Spring Boot, with Java 1.8 and Gradle as a dependency and build deployment system (Java Code Geeks, n.d.).

IntelliJ IDEA. IntelliJ Idea is used for development purposes. It is a Java IDE (Integrated Development Environment) which was developed by JetBrains. Choosing an IDE for code development is the most critical factor to be considered by any programmer. IntelliJ IDEA is the best IDE in comparison to others such as Eclipse, NetBeans, BlueJ and many others, the following describes why it is the best option (JetBrains, n.d.).

A few of the advantages to are:

- Autocomplete option makes it faster for the IDE to understand the keystrokes and auto-populate the suggestions based on the keywords typed. This functionality will reduce the time taken to write the code.
- IDEA refactoring is intelligent in that they provide various options based on the situation.
- Debugging is the best part of IDEA, it easily understands all the variables and makes it is easy as possible for the developer to track and fix bugs.

MySQL5.5 server. MySQL is used for the back-end database application. MySQL is an open-source database management system, which is a traditional way of storing the data in the back-end. MySQL Workbench, which is an integrated environment for MySQL is used to write different tables that are required for this project. It is very user-friendly and easily understood.

Gradle. Gradle is an open-source build automation system. It is built on the concepts of ANT and MAVEN. ANT and MAVEN are implemented using XML, whereas Gradle makes use of Groovy. The Project is developed with Gradle to generate the .war and .jar file, which can eventually be run on multiple systems. The .war file is deployed in the Tomcat application server to serve the REST endpoints.

Technology Stack

- Java 1.8
- Spring Boot
- Tomcat 7.*
- JPA
- MySQL 5.5
- Gradle 2.14.1
- IntelliJ Idea

UML Diagrams

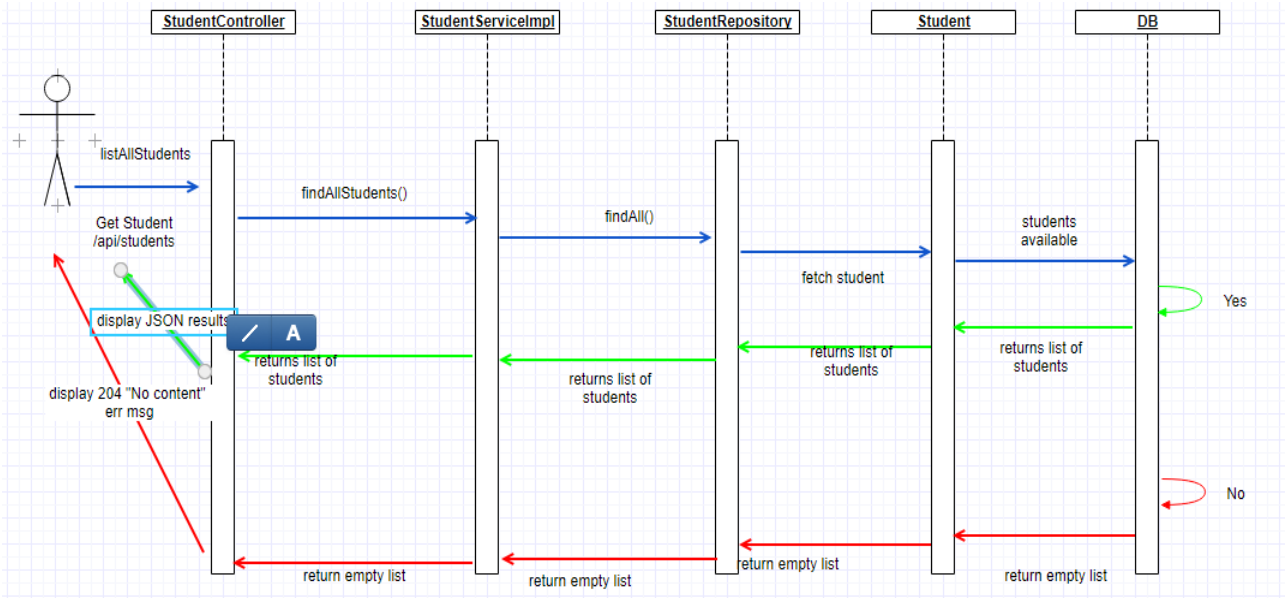


Figure 6. Get Request Sequence Diagram

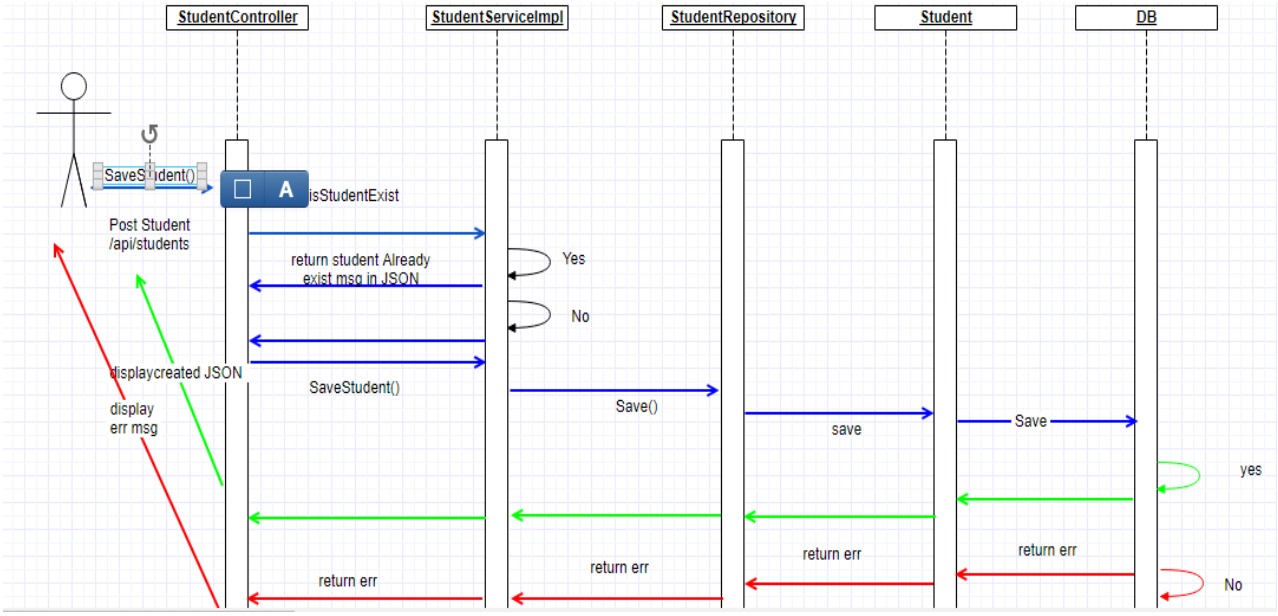


Figure 7. POST Request Sequence Diagram

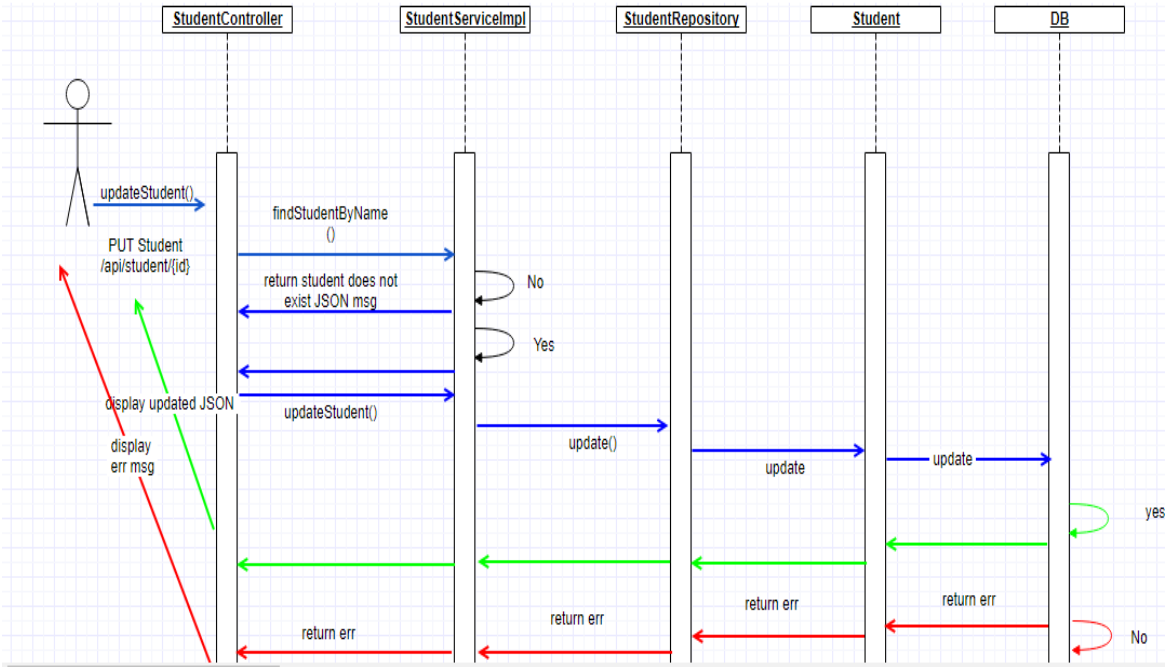


Figure 8. PUT Request Sequence Diagram

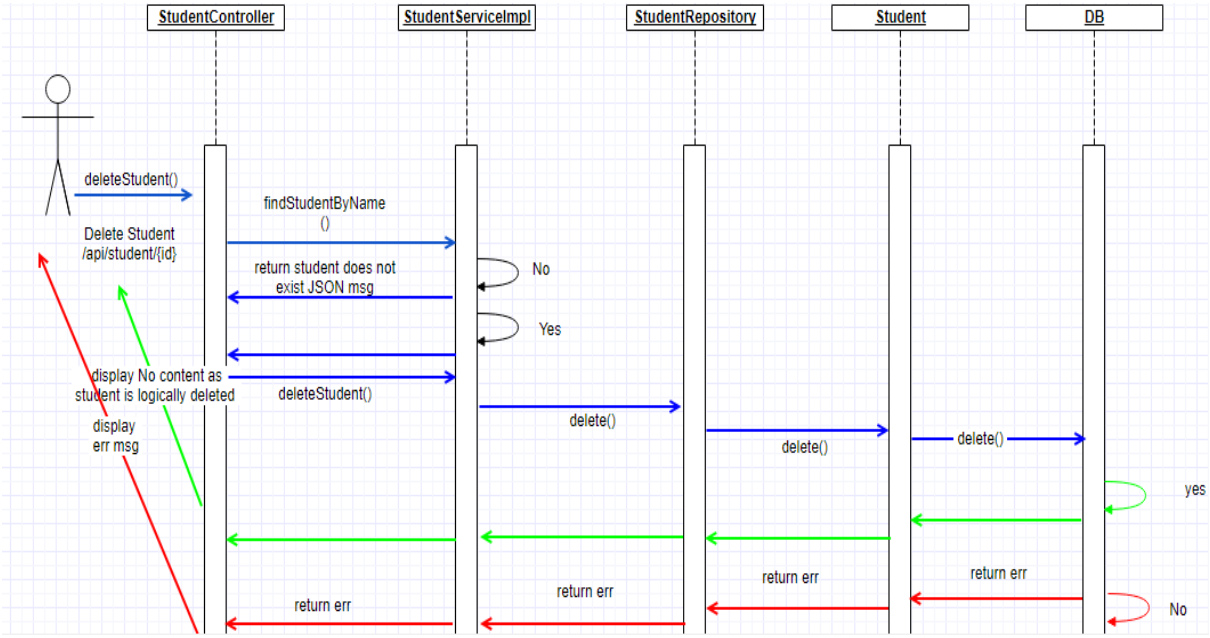


Figure 9. DELETE Request Sequence Diagram

When the data is created, the last modified date is set equal to the created date. And when the subsequent data is modified, the last modified data is then updated. In this way, the created data is not lost and sent back to the client to do an UPSERT.

When the data is deleted, a column with the name isDeleted is set to true, which is the logical deletion of data rather than a physical deletion, which will also be updated in the slave machines.

Thus, the records created, updated or deleted are synced up to date.

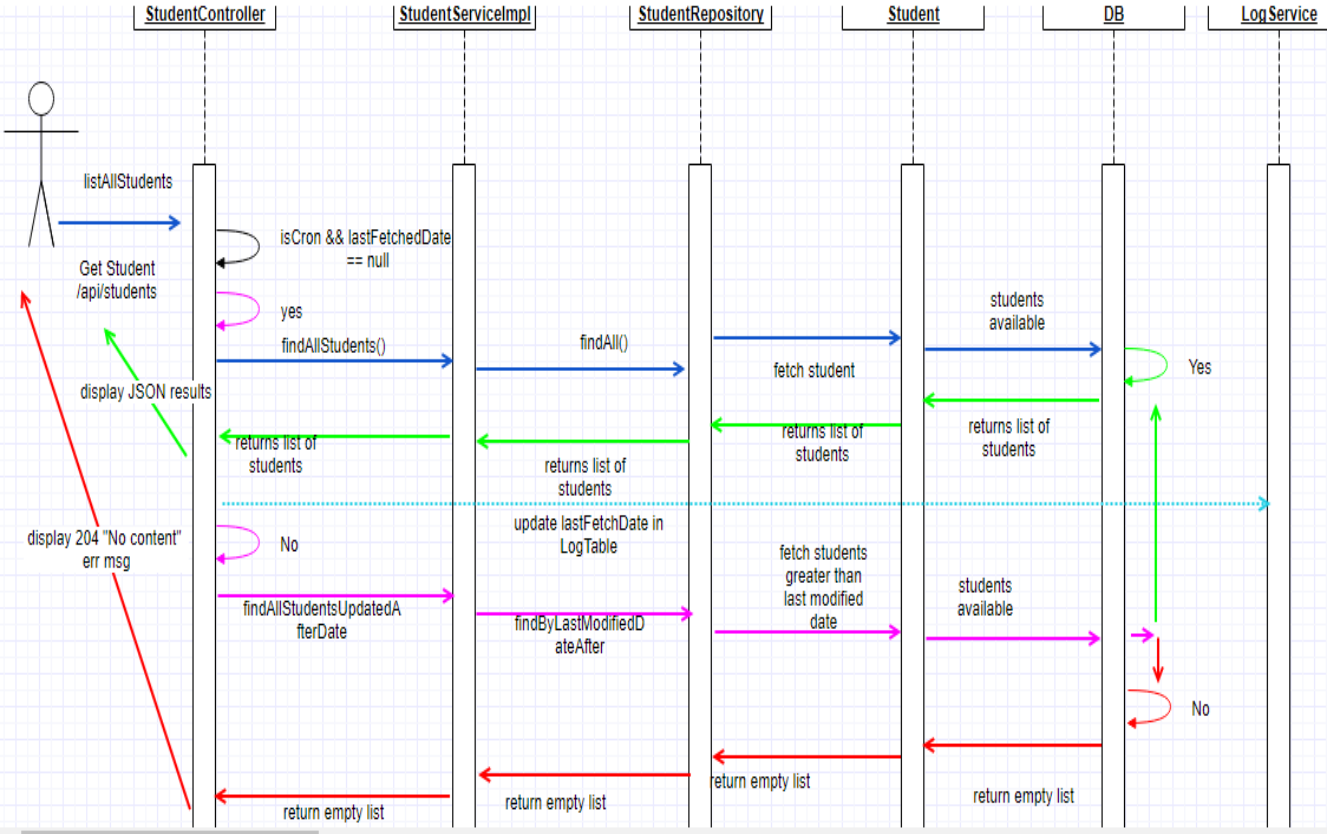


Figure 10. Secondary Host Get Sequence Diagram

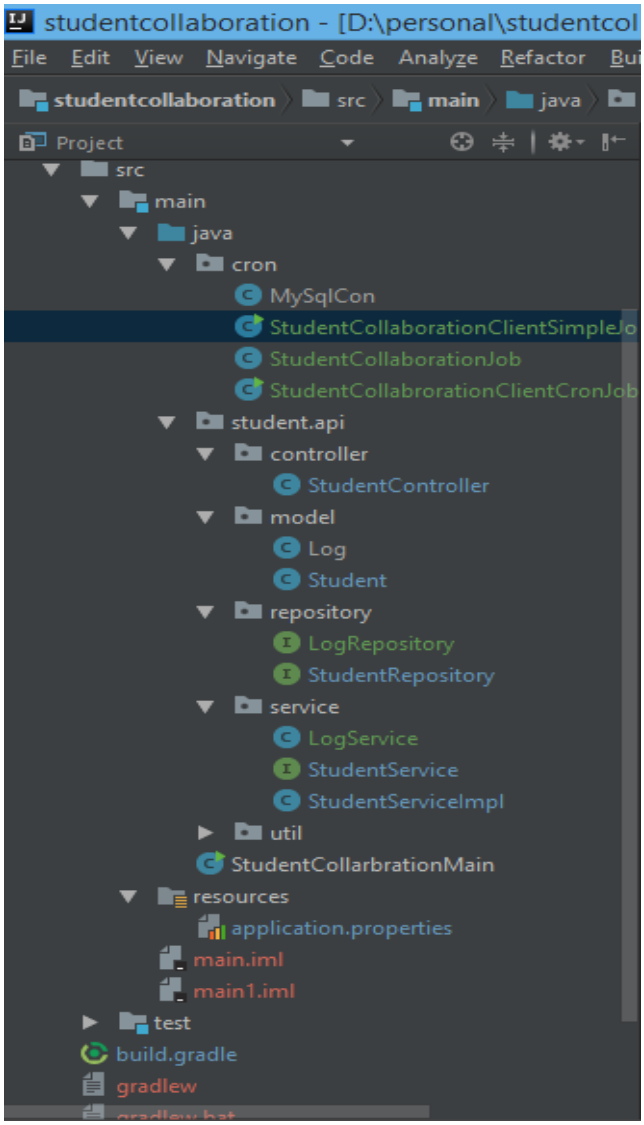


Figure 11. Project Structure

- Main file. **StudentCollaborationMain.java** serves as the main file. The Java compiler looks for this main file and is configured for the Spring boot and all the necessary JPA parameters like the package to lookup for Service, Repository and Model are configured through the respective annotations.

- Cron Package. The Cron Folder has the **StudentCollaborationClientSimpleJob.java** and **StudentCollaborationJob.java** files which are mainly used by slave machines to trigger and fetch modified data from the server. A separate MySQL connection is used to connect to the server and bring the latest data which then returns a JSON response, and which again triggers a CRUD request on its virtual machines to save the data back to its DB.
- Student API Package. This package contains the Controller, Model and Service.
- Controller. The Controller is used to process CRUD user requests, build an appropriate model and pass over it to view. The Controller's dependency is specified through an autowiring concept through annotation @autowired. The necessary dependencies a controller can have is the service.
- Model. Model is a Plain Old Java Object that encapsulates the application specific data and is mainly used by the controller and Repository to take further actions on entities.
- Service. The Service is autowired with the controller for CRUD related operations to be performed through an Interface without having to create an object. The custom implementations can be handled here.
- Repository. The Repository is mainly used to reduce the boilerplate code required to access data layers for various persistence stores. Spring provides a CRUDRepository interface, which has almost all the CRUD related operations performed without having to re-create them. The custom implementations can be implemented by

extending the CRUDRepository interface and providing any additional methods needed.

- Util. Util contains any utility methods needed for the Project. One example is the Custom error class which is required to simplify error messages to be returned to the user.
- Application.properties. The properties file is used to specify any application related properties such as MySQL connection, username, password, connectivity type, environment-related settings, and log levels. These features are used by Spring and tweaking this file during run-time is easier, enabling to change the properties based on the environment.
- Build.gradle. The build dependencies in the project are specified in the build.gradle file which uses a Groovy-based DSL (Domain Specific Language), which supports automatic download of build dependencies. The file has tasks and dependencies which can be specified for compile time as well as run time.
- Settings.gradle. Settings related to Gradle is specified as the project name and all the subprojects to be included for the build and can be added here.

Chapter IV: Implementation

Gradle Setup

- In your browser, go to https://gradle.org/releases/ and choose Version 2.x preferably 2.14.1.
- Create a folder in c: drive as c:/gradle, unzip the downloaded gradle file and copy it to the new directory created.
- Set **PATH** variable for gradle under system variable section to "C:\Gradle\gradle-2.14.1\bin".

MySQL Server Setup

To build, run and deploy the research platform database, you will need the following:

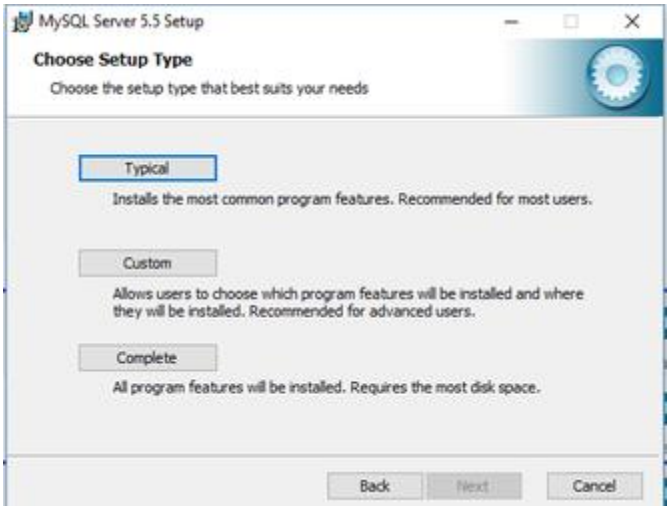
- MySQL Server 5.5 or above

You may also want to consider these optional tools for your deployment.

- MySQL Client (e.g., mysql-workbench or similar)

Install MySQL Server

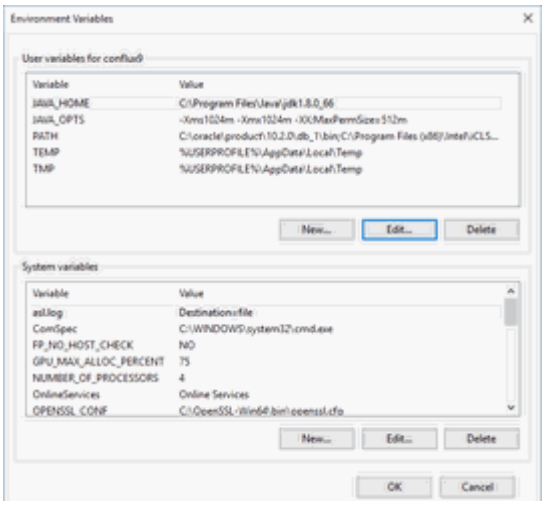
1	In your web browser, go to http://dev.mysql.com/downloads/mysql/5.5.html#downloads.To download MYSQL server 5.5v. Click on the Download button with respect to your system architecture i.e., Windows (x86, 32-bit) or Windows (x64, 64-bit).

2	You can skip this option by clicking the link at the bottom of the page that says, “No thanks, just start my download.” and Click on the “Save file to your system.”
3	Once the file is downloaded, Run the MySQL installation file. Click on Typical button and then click on the Next button. 

4	During MySQL installation, use Advanced Configuration > Show Advanced Options and set the MySQL ROOT Password. Then click on the Save button. Now your user name will be root and password will be the password provided by you.

Configure Environment Variables

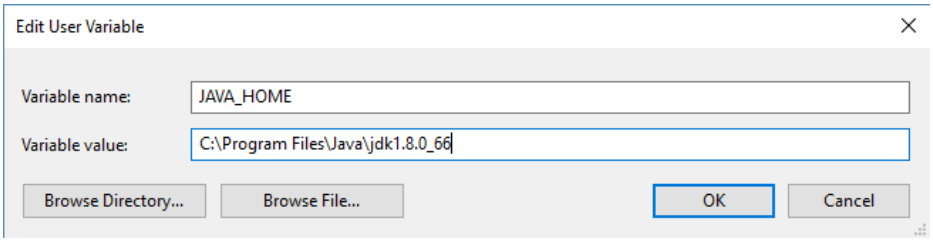
This section assumes a Windows environment, the content is the same for other environments, but the methods for accessing and setting the environment variables are different.

1	Open your Environment Variables: Start Menu > Control Panel > System and Security > System > Advanced System Settings > Environment Variables 

2

Create **JAVA_HOME** variable:

- Click on the New button, under the User Variables section.
- For Variable Name, enter **JAVA_HOME**
- For Variable Value, provide the Java path for, example: **C:\Program Files\Java\jdk1.8.0_144**

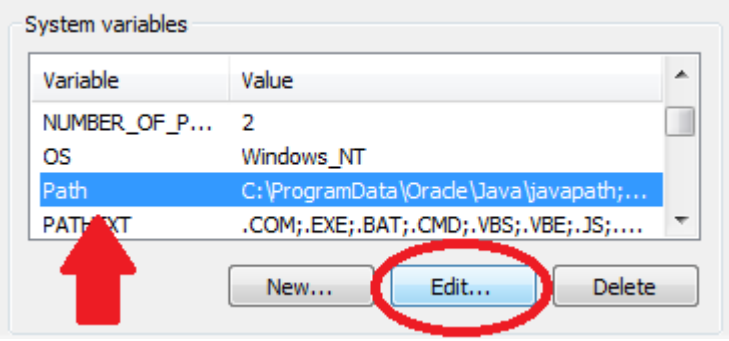


- Click the **OK** button.

1. If you are running 32-bit Windows, then the path would be ex:- C:\Program Files (x86)\Java\jdk1.8.0_144
2. Java jdk path "jdk1.8.0_144" will be with respect to the installed version and it may not be the same as mentioned in the above example.

Edit **PATH** variable:

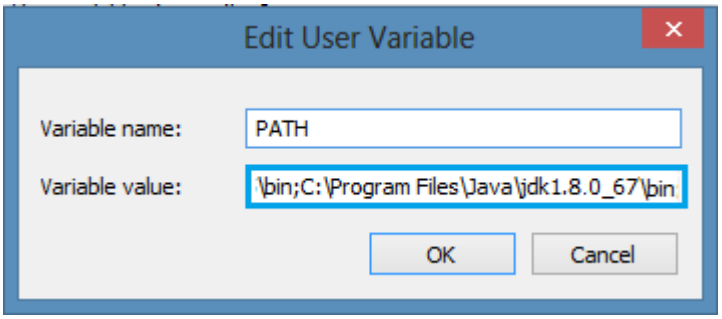
- In the System Variables section, scroll down to and select the **Path** variable.
- Click the Edit button just below.

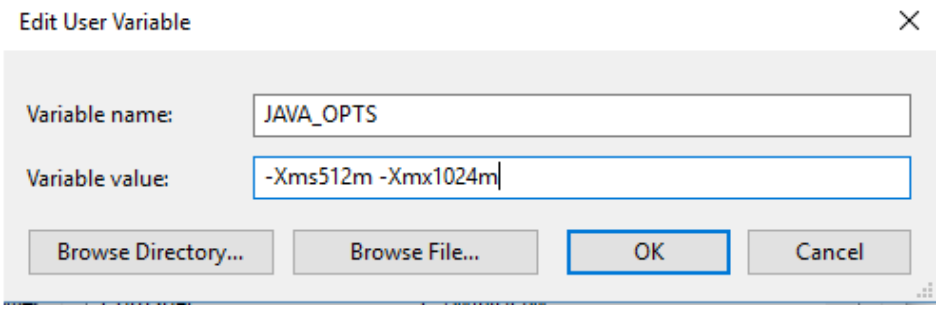


- Click your mouse cursor onto Variable Value box.

  Using your keyboard's right-arrow or End key, move all the way to the far right of the current text in Variable Value.
- Add the following text onto the end of the existing Variable Value text:
  - For **Java path** example:- **;C:\Program Files\Java\jdk1.8.0_144\bin**
  - For **MySQL server path** example:- ;**C:\Program Files\MySQL\MySQL Server 5.5\bin**
  - For **Gradle Path** example: **C:\Gradle\gradle-2.14.1\bin**

1. There must be exactly one semi-colon between whatever is already in your PATH variable, and the new C:\Program Files\Java\jdk1.8.0_66\bin – don't add a semi-colon if your previous PATH already ended with a semi-colon.
2. Java jdk path "jdk1.8.0_66" will be with respect to the installed version and it may not be the same as mentioned in the example.



3	Create **JAVA_OPTS** variable: • Click the New button again, under the User Variables section. • For Variable Name, enter **JAVA_OPTS** • For Variable Value, enter: **-Xms512m -Xmx1024m** Note: Variable value example can be changed with respect to the system memory size for example: -Xms512m -Xmx1024m 
4	Add **CLASSPATH** variable: In the System variables section, create **JAVA_HOME** (jdk path), **JRE HOME** (jdk path) and **CATALINA_HOME** (tomcat path) • point classpath variable to %JAVA_HOME%\bin;%JRE_HOME%\bin;%CATALINA_HOME%\lib; Then click **OK** button

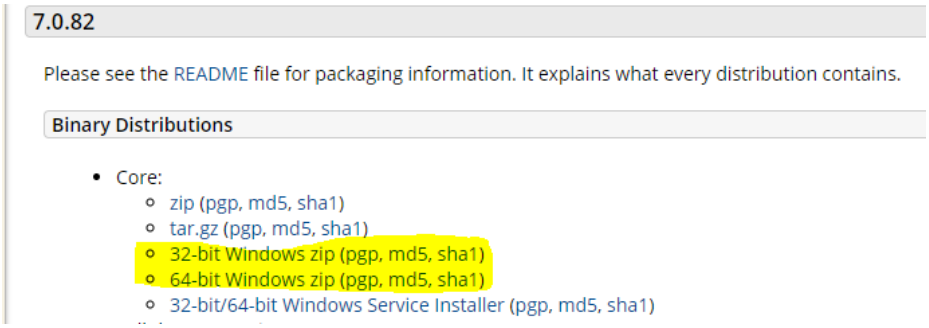
Install Java

1	Before beginning, uninstall any existing versions of Java currently on your computer.
2	In your web browser, go to http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, to download Java 1.8.
3	Oracle requires that you read the license agreement, then click the circle labeled Accept License Agreement. **Java SE Development Kit 8u144** You must accept the Oracle Binary Code License Agreement for Java SE to download this software. ○ Accept License Agreement ○ Decline License Agreement 	Product / File Description	File Size	Download
---	---	---
Linux ARM 32 Hard Float ABI	77.89 MB	jdk-8u144-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u144-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.65 MB	jdk-8u144-linux-i586.rpm
Linux x86	179.44 MB	jdk-8u144-linux-i586.tar.gz
Linux x64	162.1 MB	jdk-8u144-linux-x64.rpm
Linux x64	176.92 MB	jdk-8u144-linux-x64.tar.gz
Mac OS X	226.6 MB	jdk-8u144-macosx-x64.dmg
Solaris SPARC 64-bit	139.87 MB	jdk-8u144-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.18 MB	jdk-8u144-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u144-solaris-x64.tar.Z
Solaris x64	96.99 MB	jdk-8u144-solaris-x64.tar.gz
Windows x86	190.94 MB	jdk-8u144-windows-i586.exe
Windows x64	197.78 MB	jdk-8u144-windows-x64.exe | **Java SE Development Kit 8u144 Demos and Samples Downloads** You must accept the Oracle BSD License. to download this software. ○ Accept License Agreement ○ Decline License Agreement | Product / File Description	File Size	Download
---	---	---
Linux ARM 32 Hard Float ABI	9.95 MB	jdk-8u144-linux-arm32-vfp-hflt-demos.tar.gz
Linux ARM 64 Hard Float ABI	9.94 MB	jdk-8u144-linux-arm64-vfp-hflt-demos.tar.gz
Linux x86	52.66 MB	jdk-8u144-linux-i586-demos.rpm
Linux x86	52.52 MB	jdk-8u144-linux-i586-demos.tar.gz
Linux x64	52.72 MB	jdk-8u144-linux-x64-demos.rpm
Linux x64	52.54 MB	jdk-8u144-linux-x64-demos.tar.gz
Mac OS X	53.09 MB	jdk-8u144-macosx-x86_64-demos.zip
Solaris x64	13.52 MB	jdk-8u144-solaris-x64-demos.tar.Z
Solaris x64	9.31 MB	jdk-8u144-solaris-x64-demos.tar.gz
Solaris SPARC 64-bit	13.58 MB	jdk-8u144-solaris-sparcv9-demos.tar.Z
Solaris SPARC 64-bit	9.34 MB	jdk-8u144-solaris-sparcv9-demos.tar.gz
Windows x86	53.8 MB	jdk-8u144-windows-i586-demos.zip
Windows x64	53.82 MB	jdk-8u144-windows-x64-demos.zip	 4 Click the link to download with respect to your system architecture i.e., x86 for 32-bit or x64 for 64-bit.

5	Save the Java installation file to your computer
6	Run the Java Installation

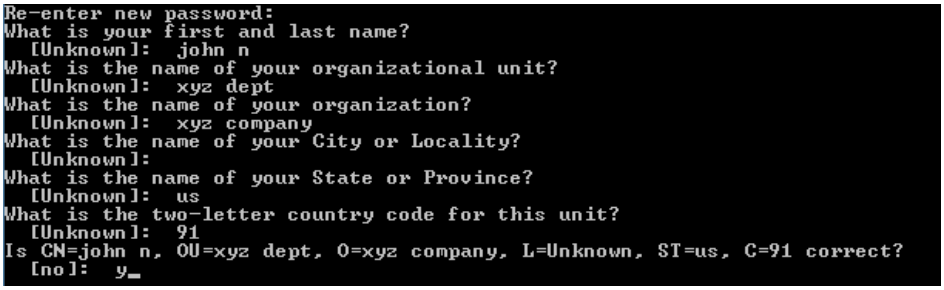
7	Open a command line and execute java -version to verify that the install was successful.
	
	`bbudd@gorilla ~$ java -version`
	`java version "1.8.0_144"`
	`Java(TM) SE Runtime Environment (build 1.8.0_144-b01)`
	`Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)`

Install Tomcat

1	In your browser, go to http://tomcat.apache.org/download-70.cgi to download Tomcat 7. • Under the section Binary Distributions Core click on the link to 32-bit or 64-bit zip (that corresponds to your system architecture) to download. 
2	Extract apache-tomcat-7.0.82.zip
3	Folder: apache-tomcat-7.0.82, gets extracted
4	Move apache-tomcat-7.0.82 folder to /usr/local/tomcat7
5	Open Command prompt and enter:- cd /usr/local/tomcat7
6	Enter: ./bin/startup.sh

Configure Tomcat

1	**Setting up Lib folder** 1. Download drizzle jdbc jar file from https://mvnrepository.com/artifact/org.drizzle.jdbc/drizzle-jdbc/1.3 2. Once downloaded**,** copy and paste to lib folder → Go to /usr/local/tomcat7/lib and put the jar file there. **Note**: If tomcat-jdbc.jar file is not available in the /usr/local/tomcat7/lib folder then you will need to download and add it to the tomcat/lib folder from: http://www.java2s.com/Code/JarDownload/tomcat-jdbc/tomcat-jdbc.jar.zip

2	**Generate keystore** Generate a new keystore using java keytool • Open the command prompt • Go to the $JAVA_HOME/bin folder. • Run this command to generate the key: **keytool -genkey -keyalg RSA -alias tomcat -keystore /usr/share/tomcat.keystore** • Provide the password. • Provide input and enter: y (for ”yes”) for the last question to confirm. 

3

Updating server.xml configuration file

- Open **c:/tomcat7/conf/server.xml** in your text editor
- Copy and paste the following contents into server.xml

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container" type="org.apache.catalina.UserDatabase"
      description="User database that can be updated and saved"
      factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
      pathname="conf/tomcat-users.xml" />
    <Resource type="javax.sql.DataSource" name="jdbc/mifosplatform-tenants"
      factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
      driverClassName="org.drizzle.jdbc.DrizzleDriver"
      url="jdbc:mysql:thin://#MYSQL_DB_ADDRESS#:3306/mifosplatform-tenants"
      username="#MYSQL_USER#"
      password="#MYSQL_PASSWORD#"
      initialSize="3"
      maxActive="10"
      maxIdle="6"
      minIdle="3"
      validationQuery="SELECT 1"
      testOnBorrow="true"
      testOnReturn="true"
      testWhileIdle="true"
      timeBetweenEvictionRunsMillis="30000"
      minEvictableIdleTimeMillis="60000"
      logAbandoned="true"
      suspectTimeout="60"
    />
  </GlobalNamingResources>
  <Service name="Catalina">
    <Connector protocol="org.apache.coyote.http11.Http11Protocol"
      port="443" maxThreads="200" scheme="https" secure="true" SSLEnabled="true"
      keystoreFile="/usr/share/tomcat.keystore" keystorePass="#KEYSTORE_PASSWORD#"
      clientAuth="false" sslProtocol="TLS" URIEncoding="UTF-8" compression="force"
      compressableMimeType="text/html,text/xml,text/plain,text/javascript,text/css"/>
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase"/>
      </Realm>
      <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
          prefix="localhost_access_log." suffix=".txt"
          pattern="%h %l %u %t "%r" %s %b" />
      </Host>
    </Engine>
  </Service>
</Server>
```

You'll need to replace the following placeholders with appropriate values for your environment.

- #MYSQL_DB_ADDRESS# = server name or IP address
- #MYSQL_USER#
- #MYSQL_PASSWORD#
- #KEYSTORE_PASSWORD#

Save the file

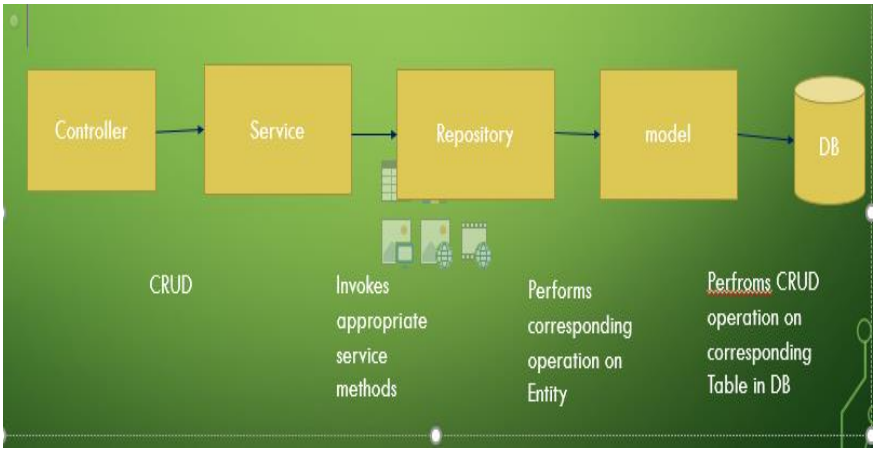


Figure 12. Primary Host Technical Design

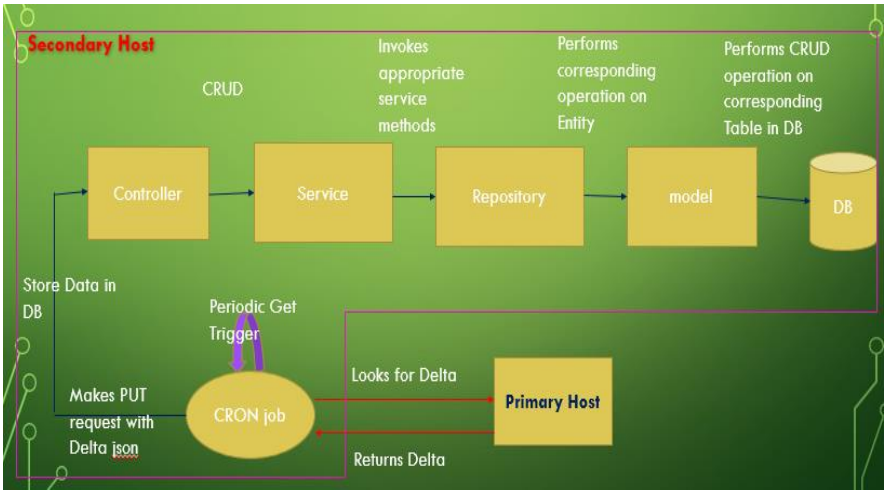


Figure 13. Secondary Host Technical Design

Technical Design Explained

Primary host.

- The StudentCollaborationController has the trigger point for Create/Get/Update/Delete requests.
- The Controller interacts with the service layer when the corresponding requests are triggered.

- The StudentService looks up corresponding CRUD operation and performs the CRUD on the Student entity
- Spring framework with JPA will help to do manipulation on the entities.
- When a new student is created, createdDate and lastModifiedDate is set to the same time.
- When the same student is modified, lastModifiedDate is updated.
- This way, the last operations performed on the student entity are not lost, but it is created or updated.
- The response is returned in a JSON format.
- The custom error message can be returned as the JSON response to any business logic failure such as, “Student Already exists or No student with that name to update or delete.”
- The data is deleted logically instead of a physical deletion to sync the same in the client DB.

MySQL database schema.

Table 3

Student Table Schema

#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	id	INT	11	☐	☐	☐	AUTO_INCREMENT
2	created_date	DATETIME		☐	☑	☐	NULL
3	deleted	TINYINT	1	☐	☐	☐	0
4	email	VARCHAR	255	☐	☑	☐	NULL
5	last_modified_date	DATETIME		☐	☑	☐	NULL
6	student_address	VARCHAR	255	☐	☑	☐	NULL
7	student_age	VARCHAR	255	☐	☑	☐	NULL
8	student_grade	VARCHAR	255	☐	☑	☐	NULL
9	student_interest	VARCHAR	255	☐	☑	☐	NULL
10	student_name	VARCHAR	255	☐	☑	☐	NULL
11	log_id	INT	11	☐	☑	☐	NULL
12	last_fetched_date	DATETIME		☐	☑	☐	NULL
13	log	TINYBLOB		☐	☑	☐	No default

Table 4

Log Table Schema

#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	log_id	INT	11	☐	☐	☐	AUTO_INCREMENT
2	last_fetched_date	DATETIME		☐	☑	☐	NULL
3	table_name	VARCHAR	255	☐	☑	☐	NULL

- CreateStudent Request: This request shows how to create a student entry. In POSTMAN set the path as – http://localhost:8080/api/student/Shravani and select POST option. Then put the following code under body and select the radio button "raw" and content-type as application/JSON, then click send. This request will create a new entry with the following details.

{

"studentName":"Shravani",

"studentAge":"20",

"studentGrade":"A",

"studentInterest":"Technology, freelancing, music",

"studentAddress":"Thompson7 st",

"email":"shravanimenneni@gmail.com"

}

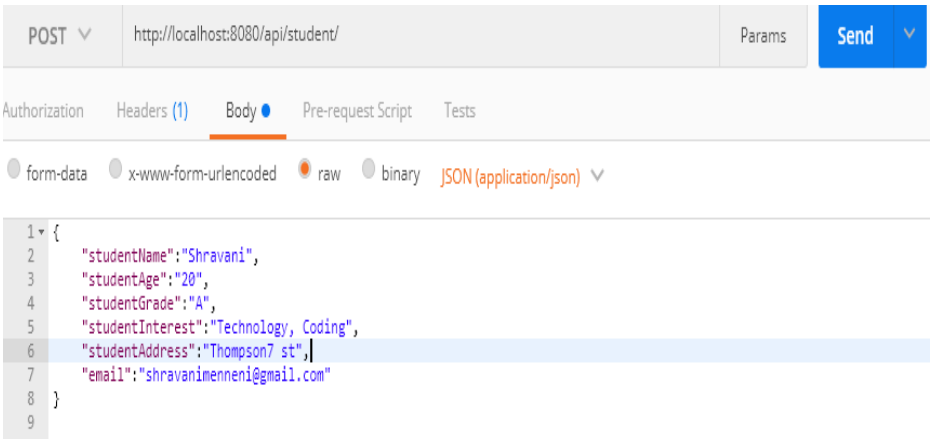


Figure 14. Create Student Request

- UpdateStudent Request: This request shows how to update a student entry. In POSTMAN set the path as - http://localhost:8080/api/student/Shravani1 and select PUT option. Then put the following code under Body and select the radio button “raw” and content-type as application/JSON, then click send. This request will update the changes for the entry with the username specified.

{

"studentName":"Shravani1",

"studentAge":"21",

"studentGrade":"B",

"studentInterest":"Technology",

"studentAddress":"Thompson43 st",

"email":"shravanimenneni1@gmail.com"

}

- Response:

{

"id": 3,

"studentName": "Shravani1",

"studentAge": "21",

"studentGrade": "B",

"studentInterest": "Technology",

"studentAddress": "Thompson43 st",

"email": "shravanimenneni1@gmail.com",

"createdDate": 1507656613000,

"lastModifiedDate": 1507656790174,

"deleted": false,

"studentLog": {

"logId": 1,

```
        "tableName": "Student",
        "lastFetchedDate": 1507615646000
    }
}
```

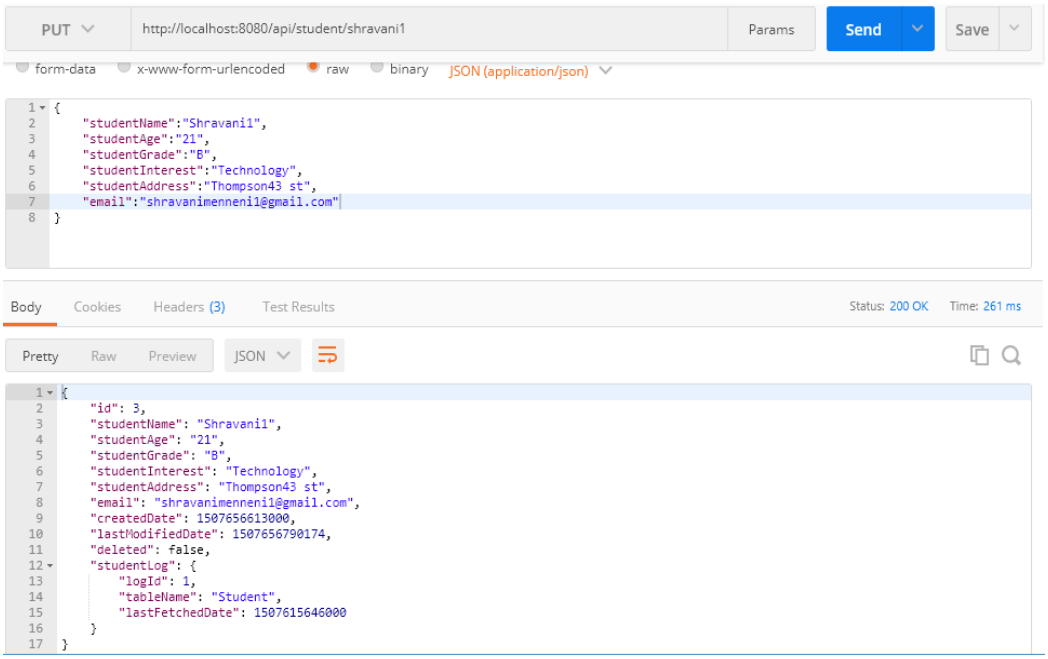



Figure 15. Update Student

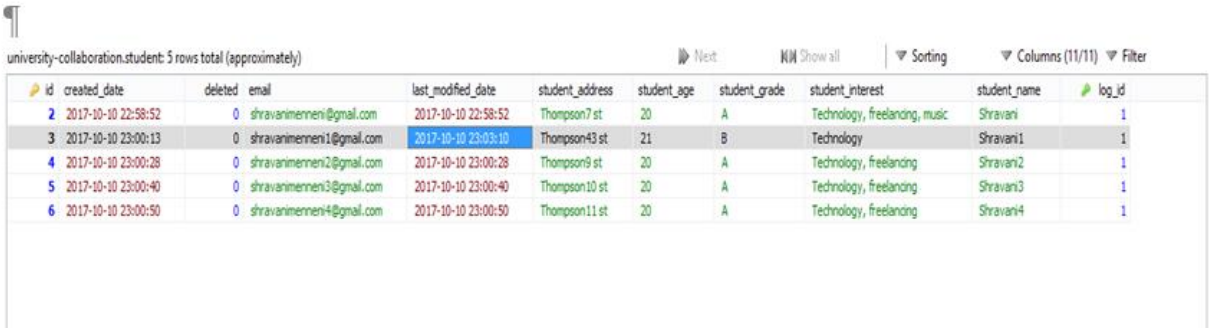


Figure 16. Database with Updated Record

- DeleteStudent Request: This request shows how to delete a student entry. In POSTMAN set the path as – http://localhost:8080/api/student/Shravani2 and select Delete option. Then put the following code under body and select the radio button “raw” and content-type as application/JSON, then click send. This request will delete the entry with the studentName set to Shravani2.

  {

  "studentName":"Shravani2",

  "studentAge":"20",

  "studentGrade":"A",

  "studentInterest":"Technology, freelancing",

  "studentAddress":"Thompson9 st",

  "email":"shravanimenneni2@gmail.com"}

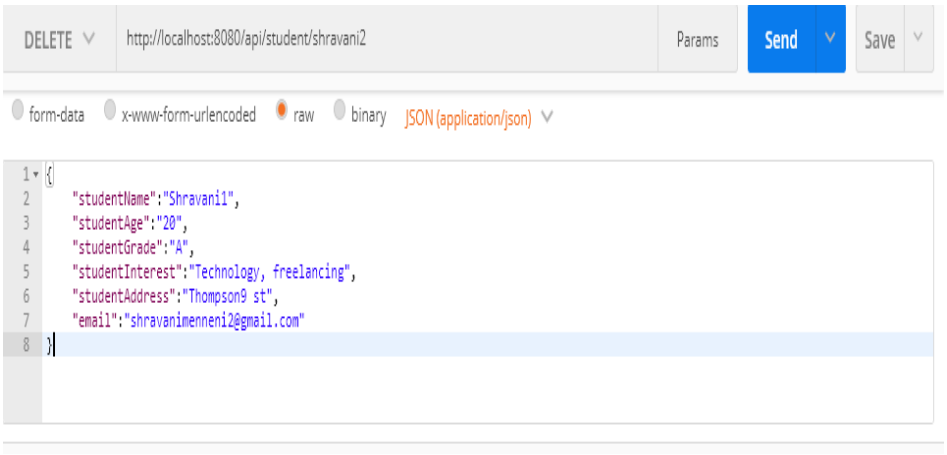


Figure 17. DELETE Request on Postman

university-collaboration.student: 5 rows total (approximately)

id	created_date	deleted	email	last_modified_date	student_address	student_age	student_grade	student_interest	student_name	log_id
2	2017-10-10 22:58:52	0	shravanimenneni@gmail.com	2017-10-10 22:58:52	Thompson7 st	20	A	Technology, freelancing, music	Shravani	1
3	2017-10-10 23:00:13	0	shravanimenneni1@gmail.com	2017-10-10 23:03:10	Thompson43 st	21	B	Technology	Shravani1	1
4	2017-10-10 23:00:28	1	shravanimenneni2@gmail.com	2017-10-10 23:08:38	Thompson9 st	20	A	Technology, freelancing	Shravani2	1
5	2017-10-10 23:00:40	0	shravanimenneni3@gmail.com	2017-10-10 23:00:40	Thompson10 st	20	A	Technology, freelancing	Shravani3	1
6	2017-10-10 23:00:50	0	shravanimenneni4@gmail.com	2017-10-10 23:00:50	Thompson11 st	20	A	Technology, freelancing	Shravani4	1

Figure 18. Deleted Record in the Database

- GetStudents Request: This request shows how to get all the entries. In POSTMAN set the path as – http://localhost:8080/api/students then click send. It returns the following output.
- Response:

[{

"id": 2,

"studentName": "Shravani",

"studentAge": "20",

"studentGrade": "A",

"studentInterest": "Technology, freelancing, music",

"studentAddress": "Thompson7 st",

"email": "shravanimenneni@gmail.com",

"createdDate": 1507656532000,

"lastModifiedDate": 1507656532000,

"deleted": false,

"studentLog": {

"logId": 1,

"tableName": "Student",

```
        "lastFetchedDate": 1507615646000
    }
},
{
    "id": 3,
    "studentName": "Shravani1",
    "studentAge": "21",
    "studentGrade": "B",
    "studentInterest": "Technology",
    "studentAddress": "Thompson43 st",
    "email": "shravanimenneni1@gmail.com",
    "createdDate": 1507656613000,
    "lastModifiedDate": 1507656790000,
    "deleted": false,
    "studentLog": {
        "logId": 1,
        "tableName": "Student",
        "lastFetchedDate": 1507615646000
    }
},
{
    "id": 4,
```

```
    "studentName": "Shravani2",
    "studentAge": "20",
    "studentGrade": "A",
    "studentInterest": "Technology, freelancing",
    "studentAddress": "Thompson9 st",
    "email": "shravanimenneni2@gmail.com",
    "createdDate": 1507656628000,
    "lastModifiedDate": 1507657118000,
    "deleted": true,
    "studentLog": {
        "logId": 1,
        "tableName": "Student",
        "lastFetchedDate": 1507615646000
    }
},
{
    "id": 5,
    "studentName": "Shravani3",
    "studentAge": "20",
    "studentGrade": "A",
    "studentInterest": "Technology, freelancing",
    "studentAddress": "Thompson10 st",
```

```
    "email": "shravanimenneni3@gmail.com",
    "createdDate": 1507656640000,
    "lastModifiedDate": 1507656640000,
    "deleted": false,
    "studentLog": {
        "logId": 1,
        "tableName": "Student",
        "lastFetchedDate": 1507615646000
    }
},
{
    "id": 6,
    "studentName": "Shravani4",
    "studentAge": "20",
    "studentGrade": "A",
    "studentInterest": "Technology, freelancing",
    "studentAddress": "Thompson11 st",
    "email": "shravanimenneni4@gmail.com",
    "createdDate": 1507656650000,
    "lastModifiedDate": 1507656650000,
    "deleted": false,
    "studentLog": {
```

```
            "logId": 1,
            "tableName": "Student",
            "lastFetchedDate": 1507615646000
        }
    }
]
```


Secondary host.

- Secondary Host performs the same operations as the primary host.
- Secondary Host always pulls the data from the server.
- The main difference is that secondary host triggers Get Request using CRON on a periodic time frame.
- When there is any delta available on the server, the server responds in a JSON format.
- The client again makes a PUT request, which UPSERTS the data in the DB.
- If the server contains data that is deleted, lastModifiedDate is updated along with isDeleted field set to true. The same changes are updated on the client.

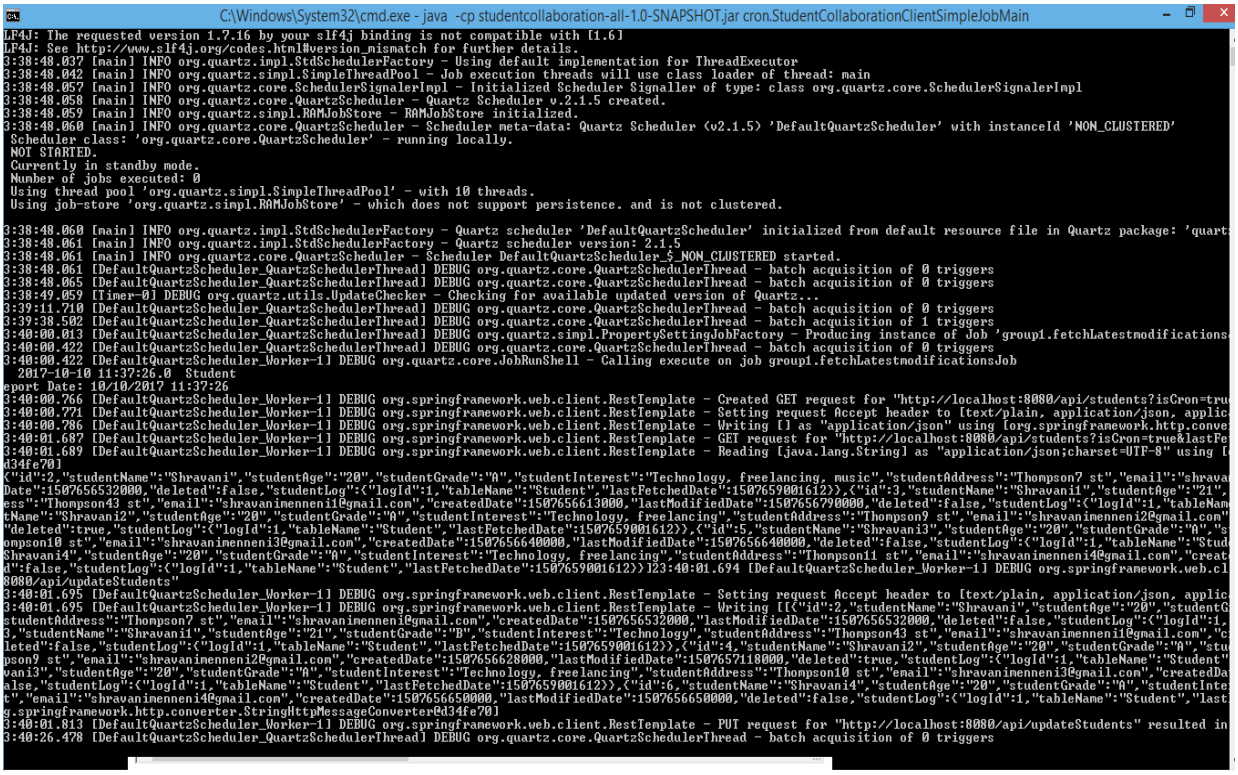


Figure 19. Secondary Host Changes

Setting up an IDE for the Development Environment

IntelliJ Idea is used for the development purpose with the latest download community edition from https://www.jetbrains.com/idea/download/#section=windows.

Next it is necessary to create a project with the proper java structure in place along with the build.gradle specifying the necessary dependencies.

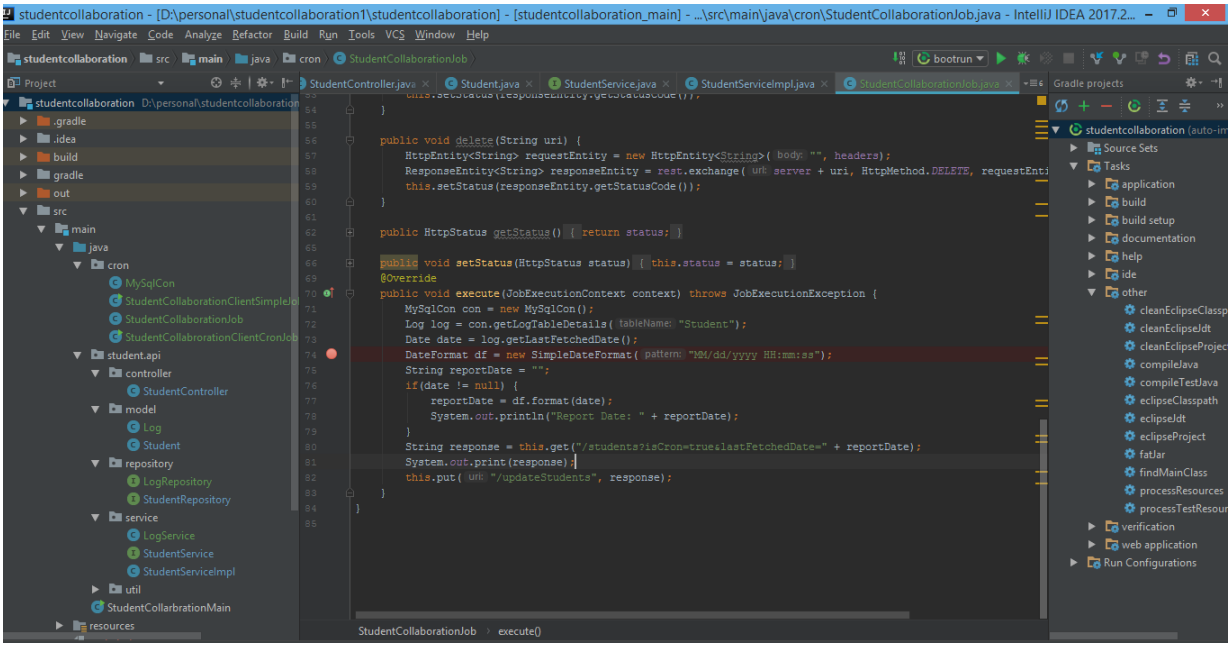


Figure 20. Development Environment in IntelliJ IDEA

Once done, Gradle has a few built-in tasks available enabling us to create war, jar files. Since Spring boot is used for development, which also has the embedded tomcat, the project can be executed by creating RUN/DEBUG configuration as Gradle application and setting a boot run task.

In IntelliJ, Go to View -> Tools window -> Gradle, below the side panel appears a displaying of tasks and run configurations to be set for Gradle.

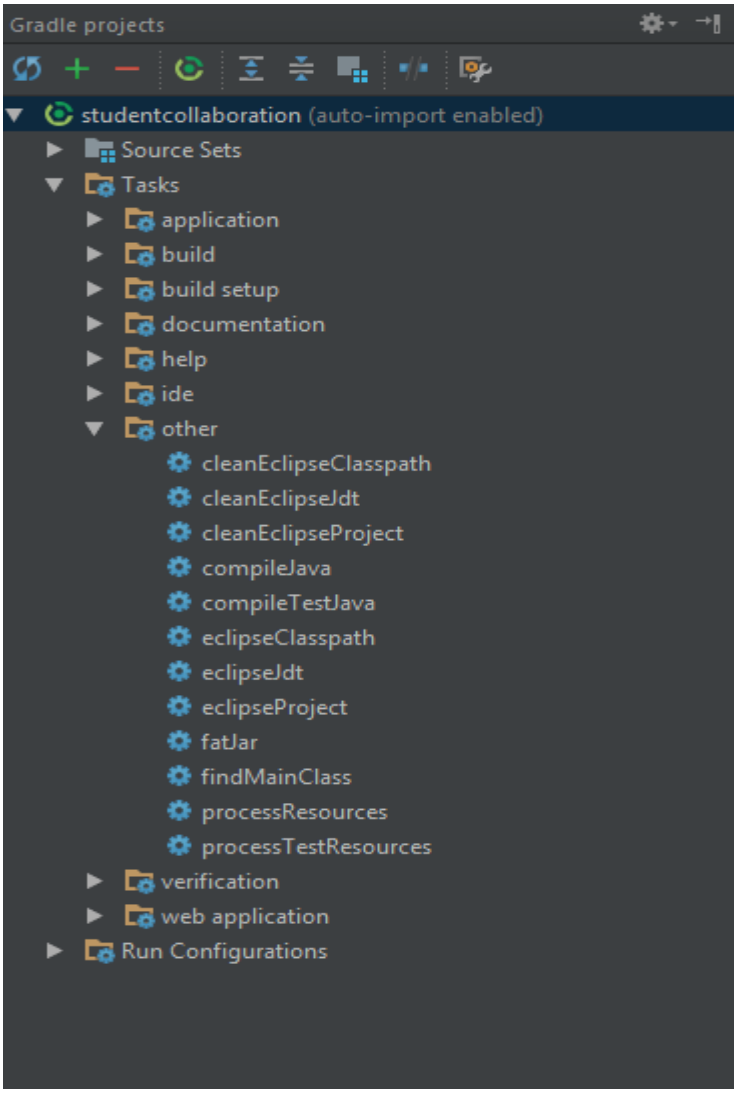


Figure 21. Gradle Settings

- Enable RUN/DEBUG configuration

  Go to Run -> Edit Configuration -> Click on the "+" on the top left corner of the popup window. Select Gradle from the list of applications as shown below.

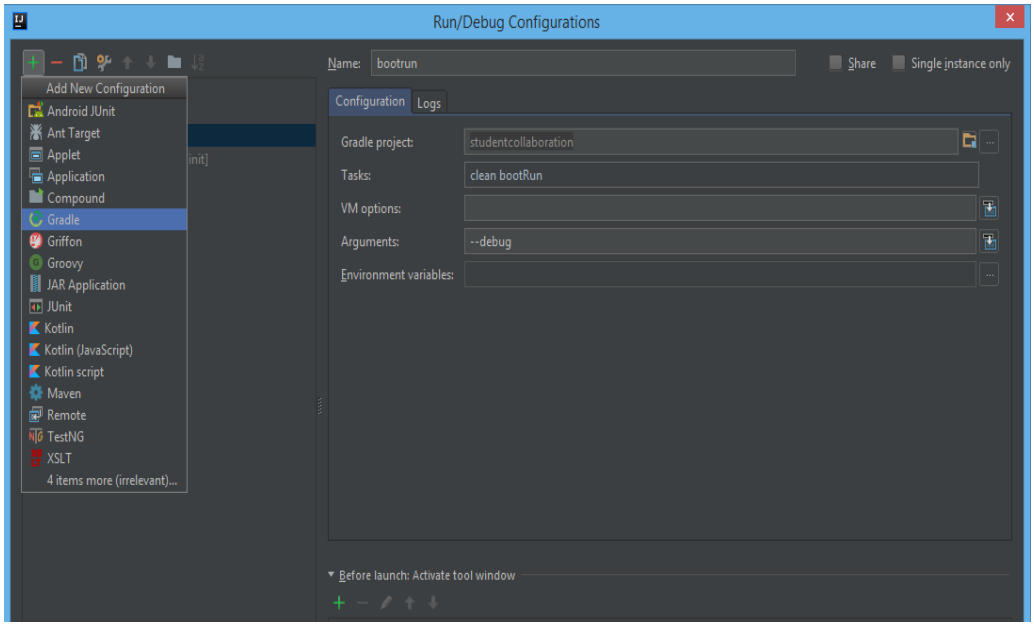


Figure 22. Run/Debug Configuration

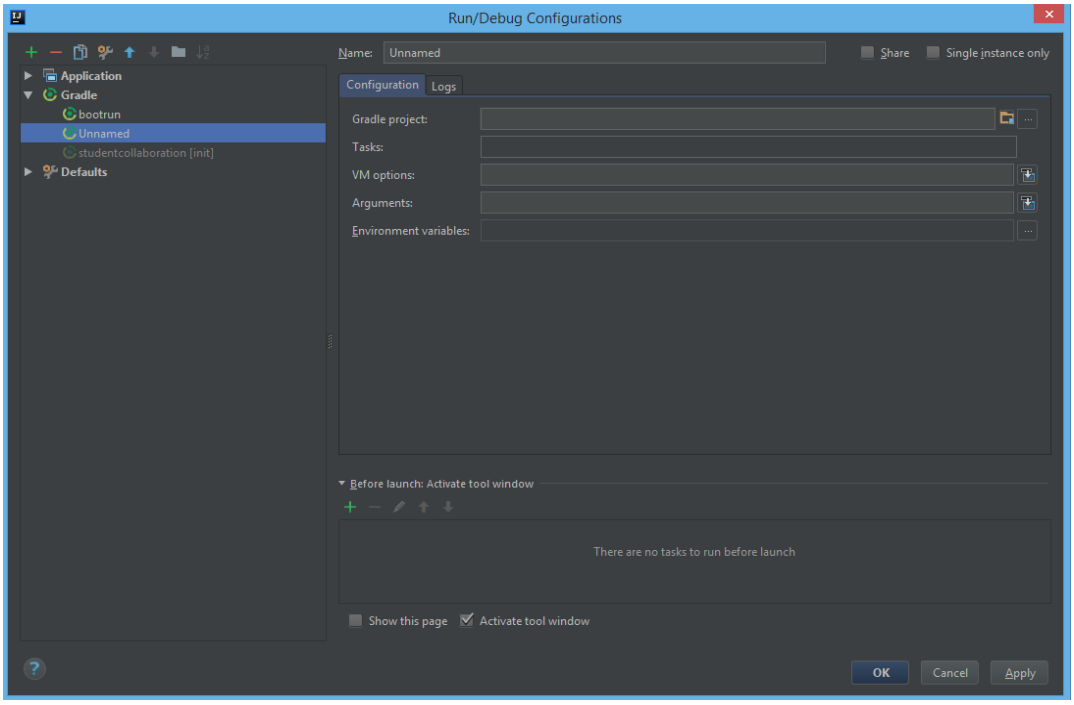


Figure 23. Window that Appears to Create a Configuration

In the **Name** field, specify the name of config as “Bootrun” (the name can be anything). Under the **Configurations** tab, set the below parameters:

Gradle Project: Root folder of the Project where build.gradle is visible

Tasks: clean bootrun

Arguments: --debug

Click on Apply -> OK

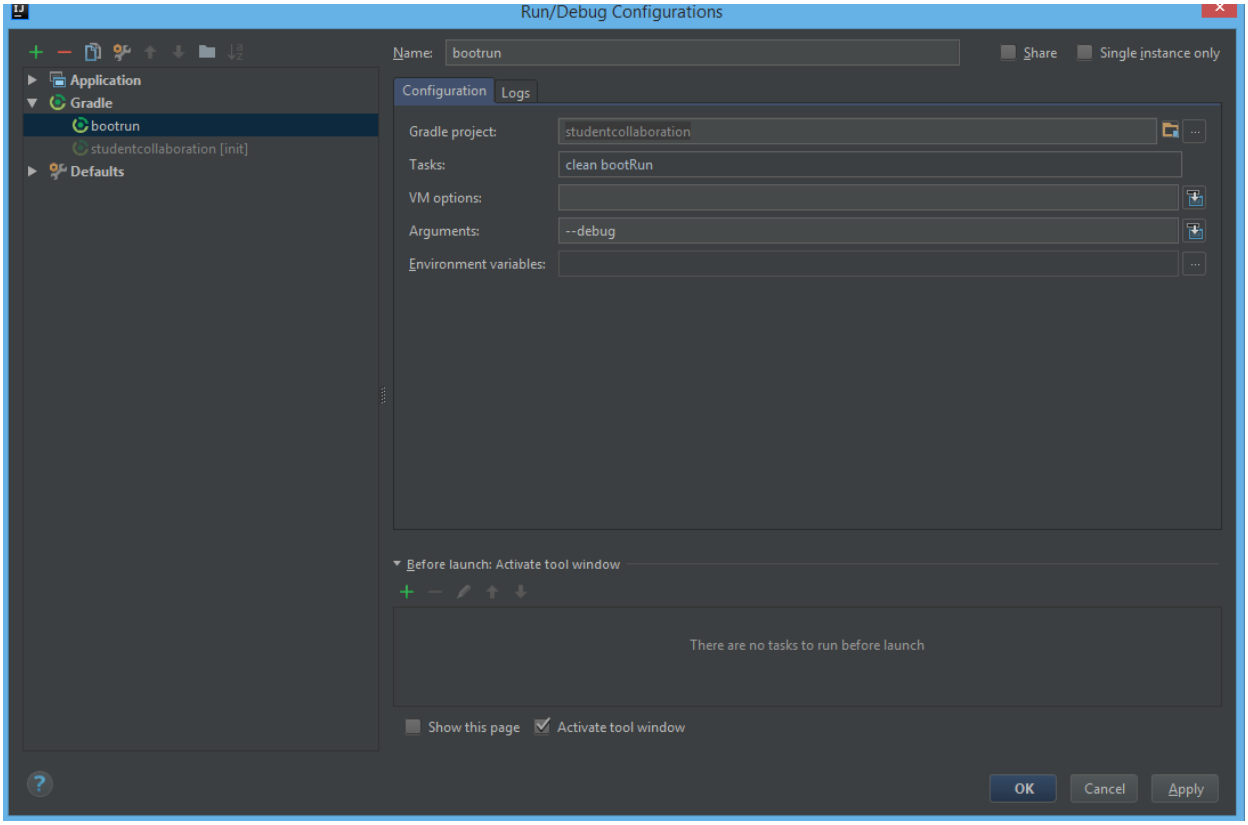


Figure 24. Configuration is Created Enabling us to do a Debug or a Run

- Setting up Client cron

  Go to Run-> click on + -> select **Application** from the list

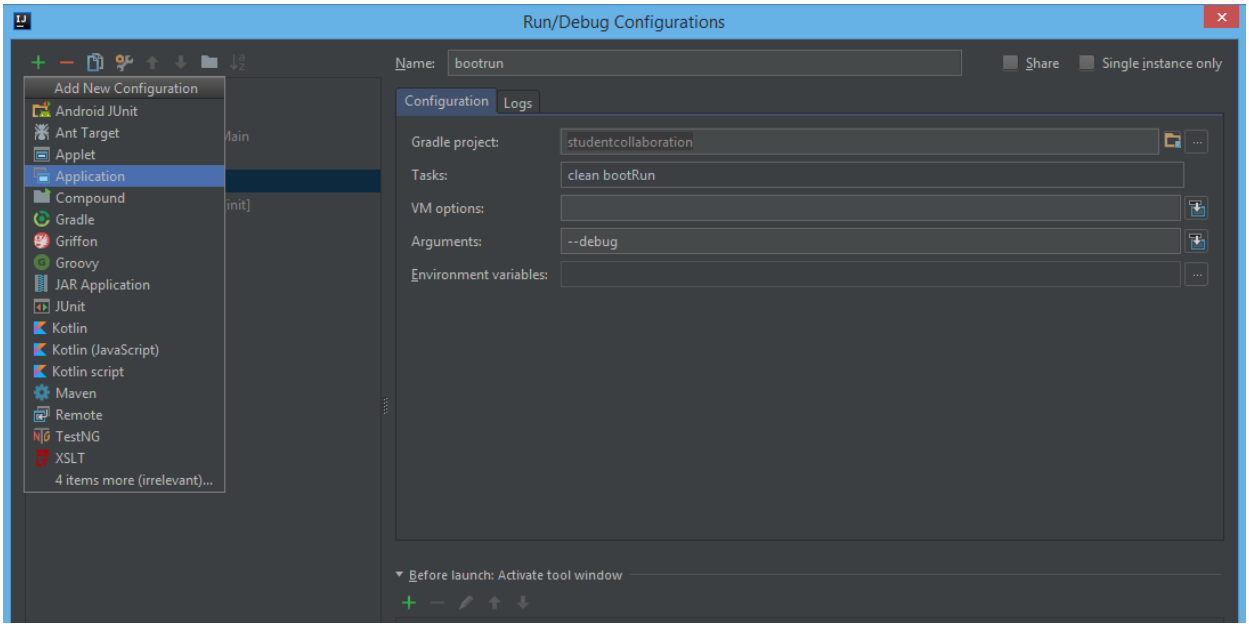


Figure 25. Client Cron Setup Instructions

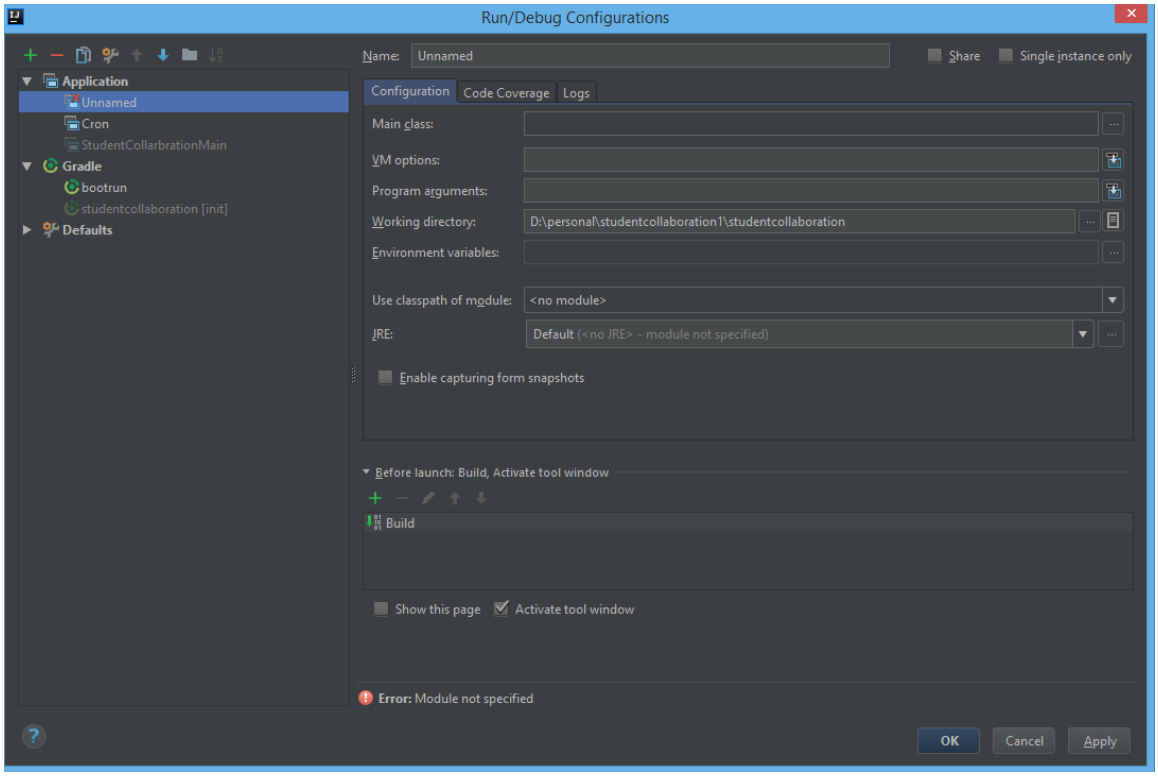


Figure 26. Window that Appears to Create a Cron Configuration

In the Name field, enter “Cron” (name can be anything)

Under the Configurations Tab,

Enter the following:

Main Class: Select the StudentCollaborationClientSimpleCronJob

Working directory: by default, it is populated.

Use the class path of Module: Select StudentCollaboration_main

Click on Apply -> OK

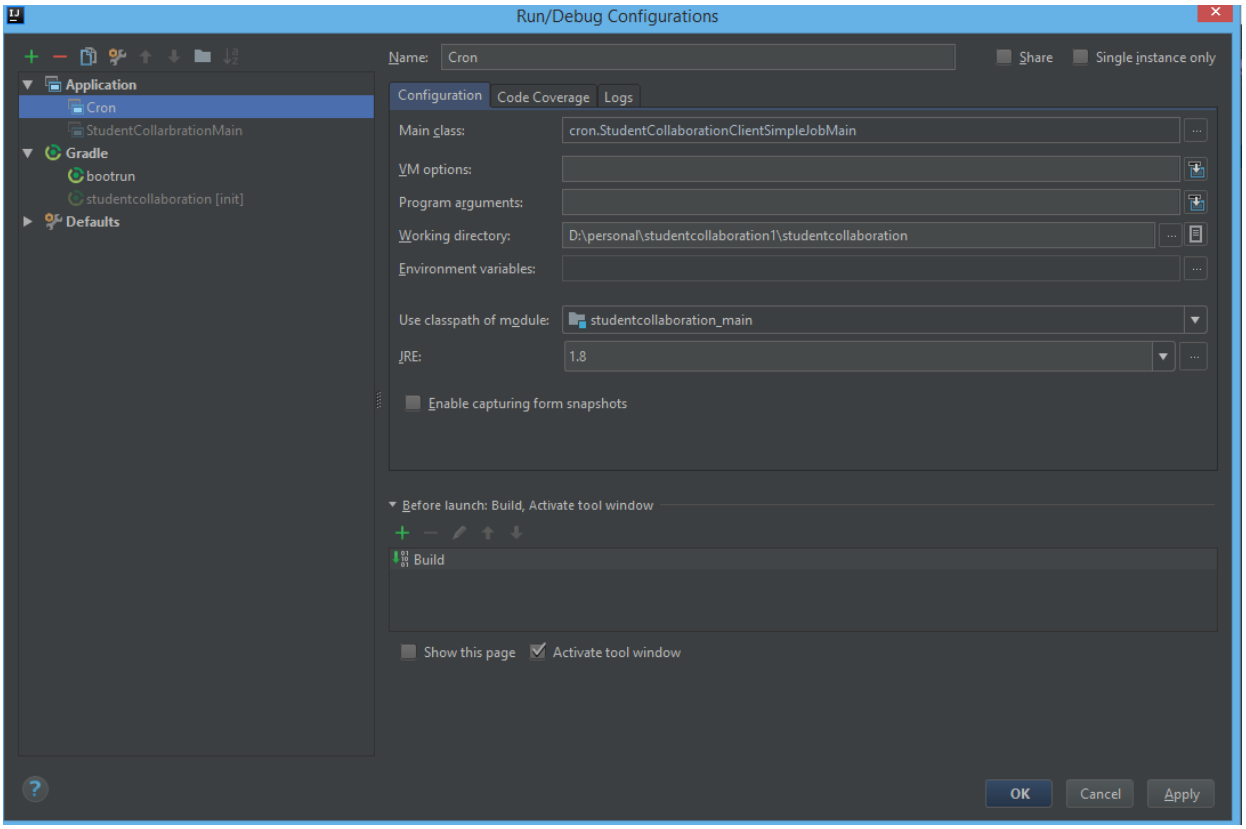


Figure 27. Setup Instructions for Cron

Build and Deployment

The application can be built both for development and a production environment, the development environment can be setup on an IDE to enable debugging and finding of issues.

- Development Environment

  On the right side of the IDE or from the Run menu, choose the configuration "BootRun" created above as debug or run configuration. The application starts to debug, and can be viewed in the console.

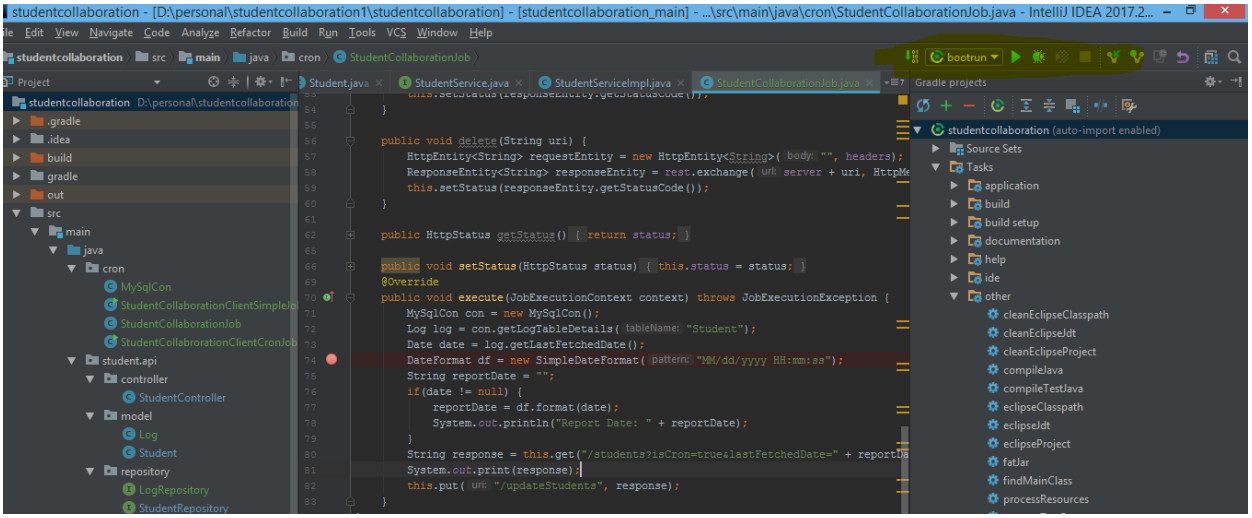


Figure 28. Application BootRun Option

Once done, check the application by clicking on the browser or postman.

POSTMAN is an HTTP client that is used for testing web services. It is useful in interfacing with the REST API's.

- Production Environment

Primary Host: Go to the root folder of the project in the terminal or command prompt and enter the command "**gradle clean war**".

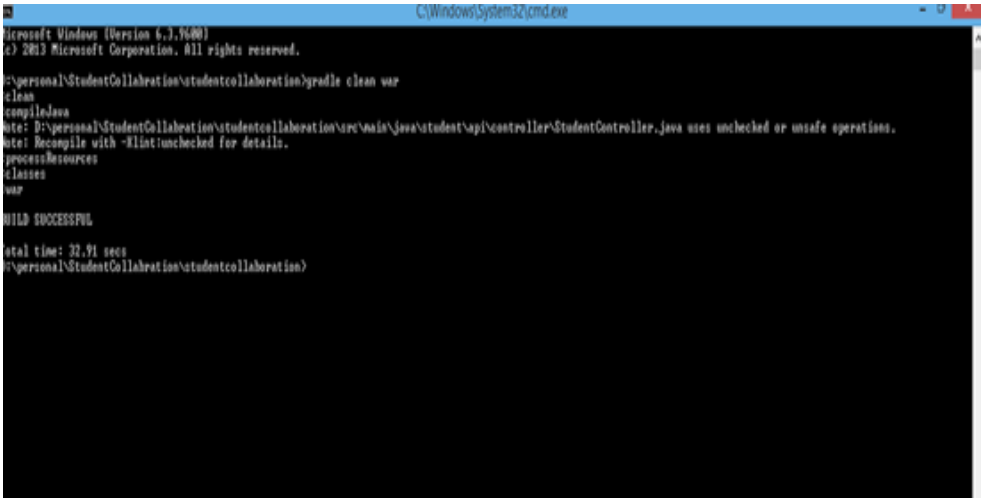


Figure 29. Build Successful Message on Command Prompt

The .war file is generated successfully in /StudentCollaboration/build/libs/1-1.0-SNAPSHOT.war.

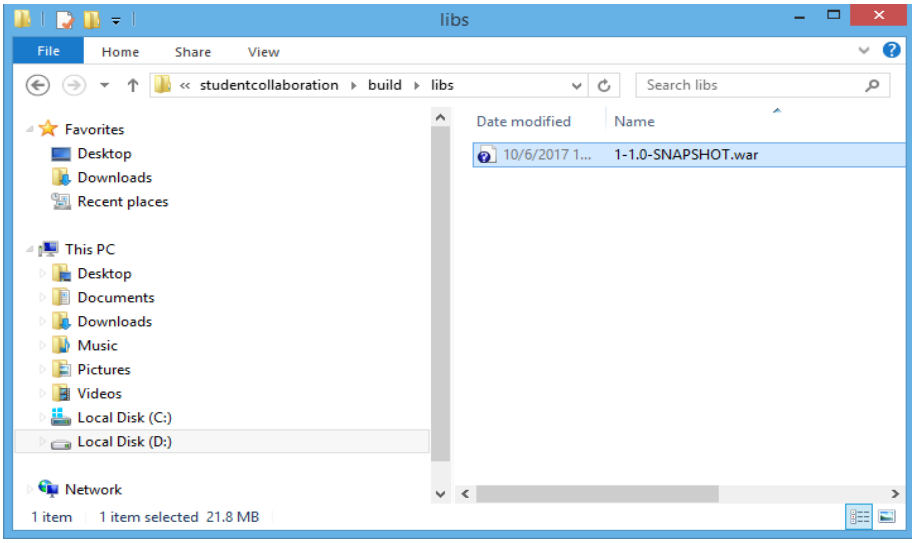


Figure 30. War File Generated

- Copy the generated .war file.
- Paste the .war file in c:\Tomcat\webapps.
- Start the tomcat -> Go to c:\tomcat\bin > startup.bat.

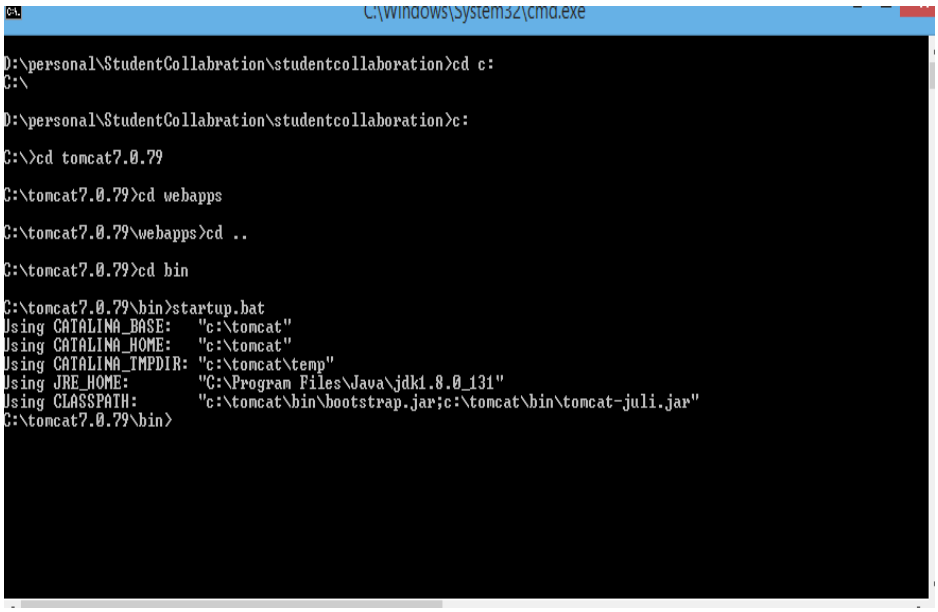


Figure 31. Tomcat Configuration

- Tomcat will start to deploy the .war file and the entire logs can then be found in the console.

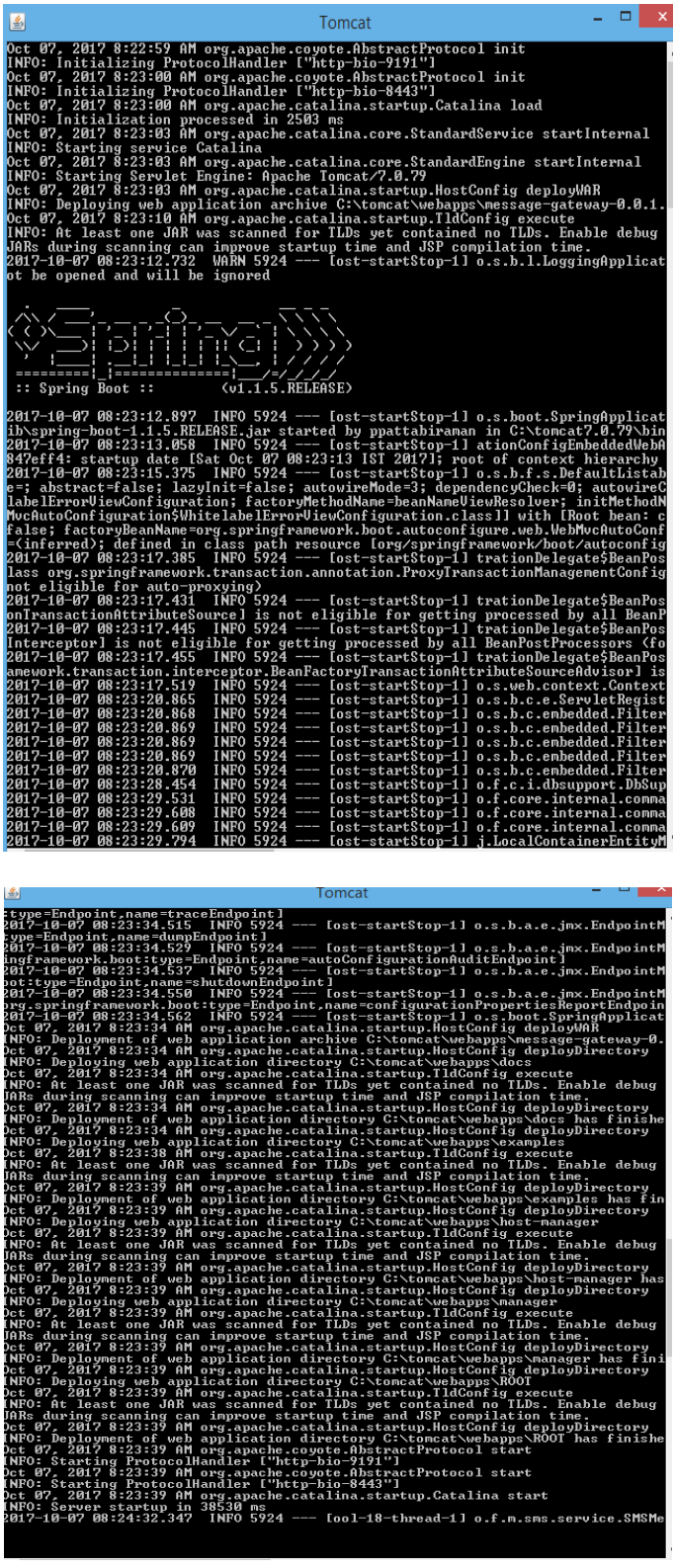


Figure 32. Tomcat Deployment

The above console shows that the tomcat is deployed successfully. Note that the .war file will be extracted in the name of 1-1.0-SNAPSHOT and can be accessible by the context as http://localhost:8080/ 1-1.0-SNAPSHOT/api/student.

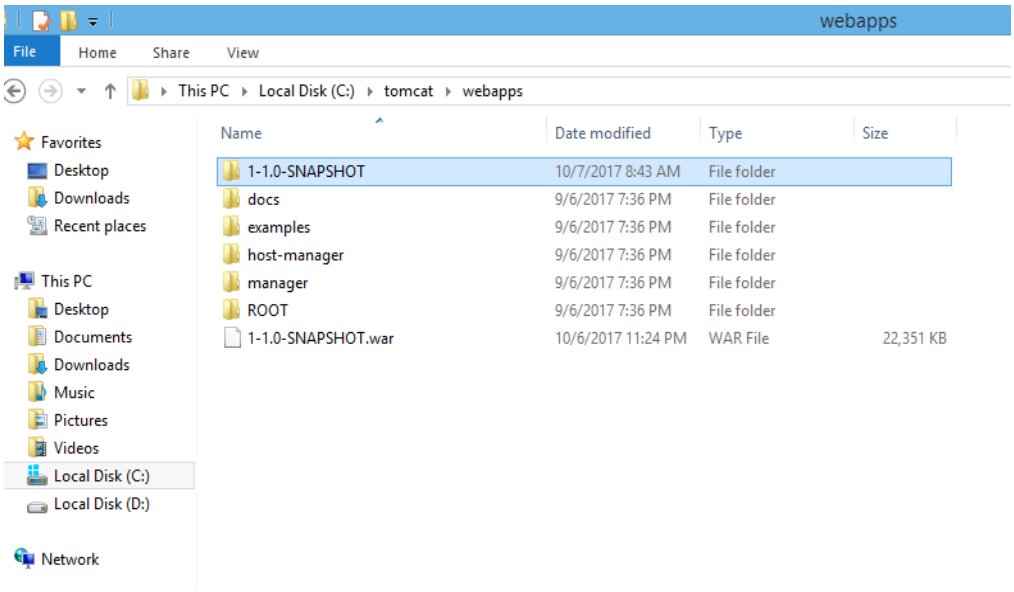


Figure 33. War File Generated in Tomcat Folder

- Go to a browser and enter: http://localhost:8080/api/students
- The server is now up and running, see Figure 34.

Figure 34. Student Details Displayed

Secondary Host:

- Follow the same steps as with the primary host build deployment.

- Additional steps required are to generate a .jar file to be able to run on an independent application, see Figure 35.

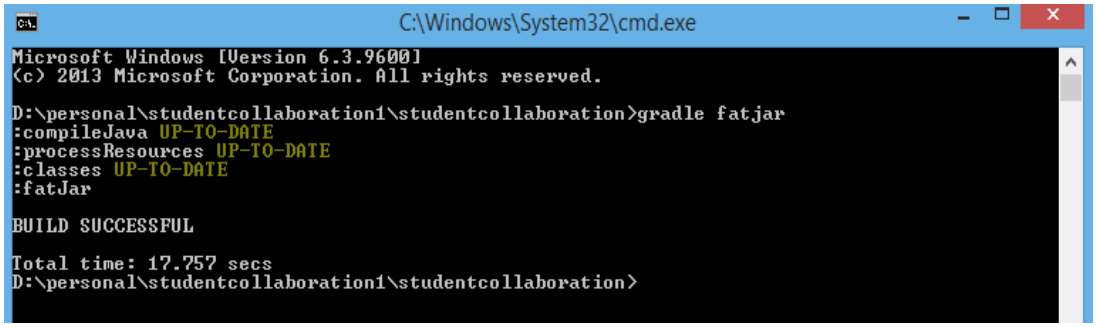


Figure 35. Jar File Being Generated

- .jar is generated in build\libs folder

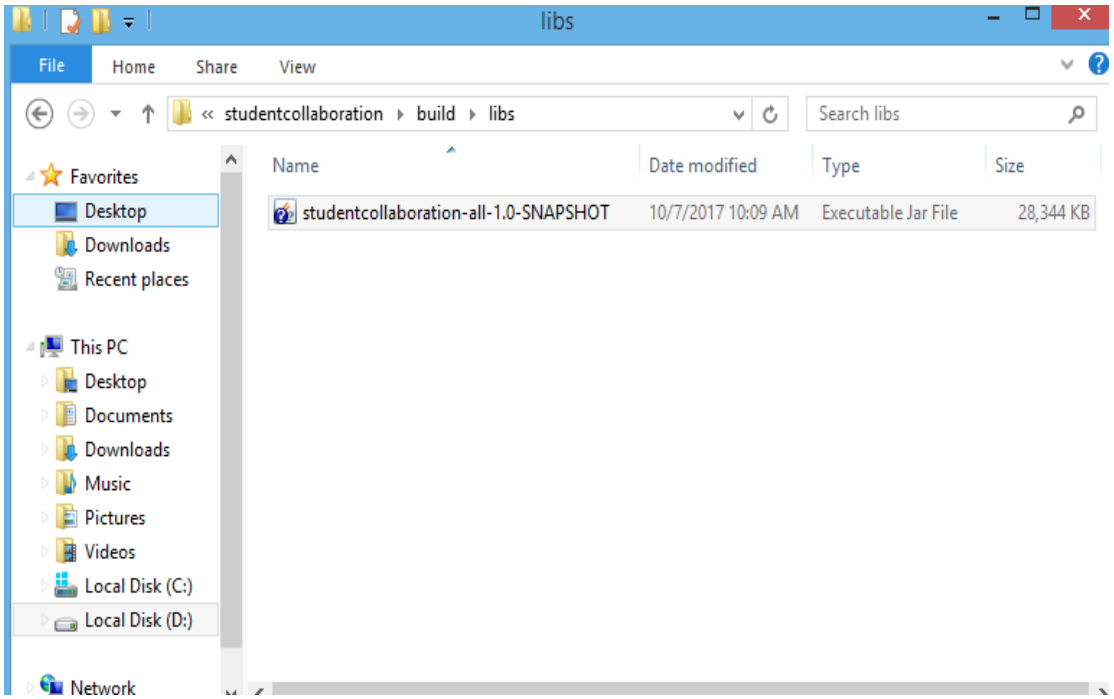


Figure 36. Jar File Generated Inside the Build Folder

- Once the .jar is generated, start the client application cron job to see if there are any modifications to the data from the last fetched date.

- For the very first time, the last fetched date is null and hence all the data is fetched from the server and inserted in the DB. Also, the last fetch date is now updated to the current date and time.
- To start the cron job from jar → go to build/libs folder where .jar is generated and execute the command “java -cp studentcollaboration-all-1.0-SNAPSHOT.jar cron.StudentCollaborationClientSimpleJobMain”

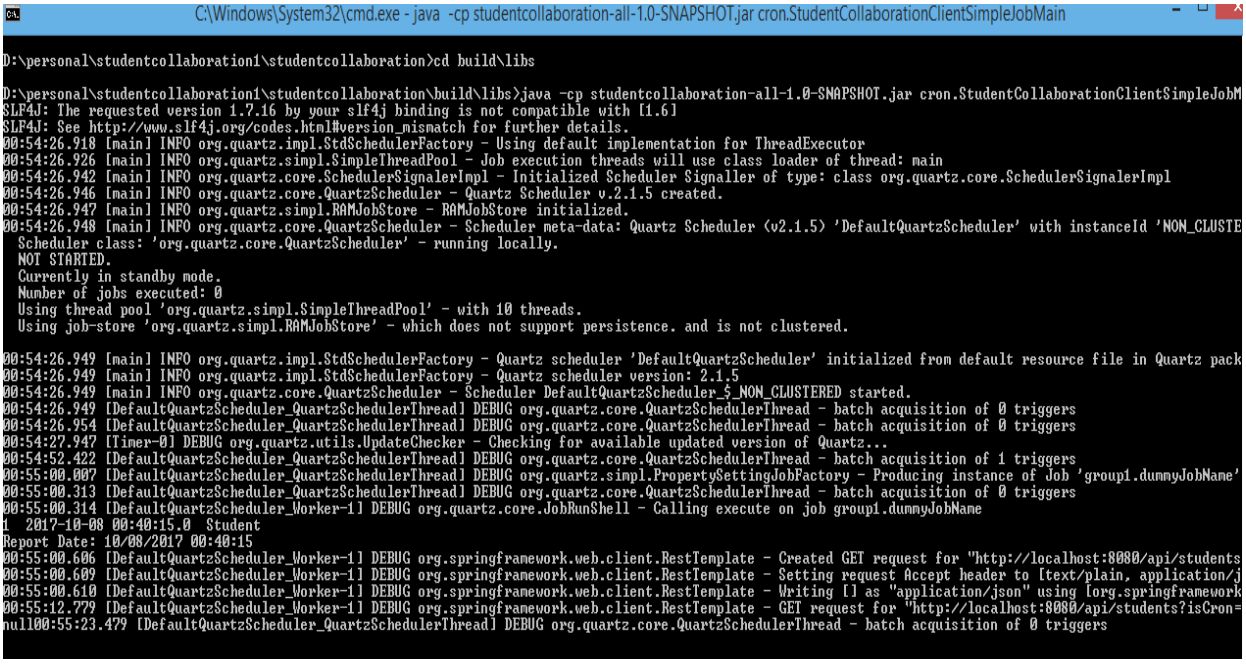


Figure 37. Execution of Cron Job

From the above console screen in Figure 37, you can see that the cron job has started successfully and makes the request to the server to fetch any modified data. The cron time can be tweaked to minutes, hours, seconds and the frequency of updates can also be set.

Sample application explained:

- To prove the capability of data transfer between the primary and secondary hosts, a StudentCollaboartion application is created.

- Considered only student Create/Update/Get/Delete scenarios.
- Maintained a Log table to update the last fetched time for the corresponding table.
- The idea is to have the same application to be hosted in all three hosts, which will have their own application server(tomcat) running along with the MySQL database.
- The only difference between the secondary and primary hosts is that the secondary host will have a cron job running to look for any modifications on the server.
- If there are any modifications, the delta is fetched greater than the last fetched time.
- Secondary hosts cron job makes a Get request to the student API with the lastfetchdate parameter.
- It will check for the last modified data after the given fetch date.
- The client performs UPSERT (update or insert) of data based on the current state of DB.
- Any updates in the client will be overwritten by the server update.
- Insert data in the Student table through create request and start the cron job as specified before.

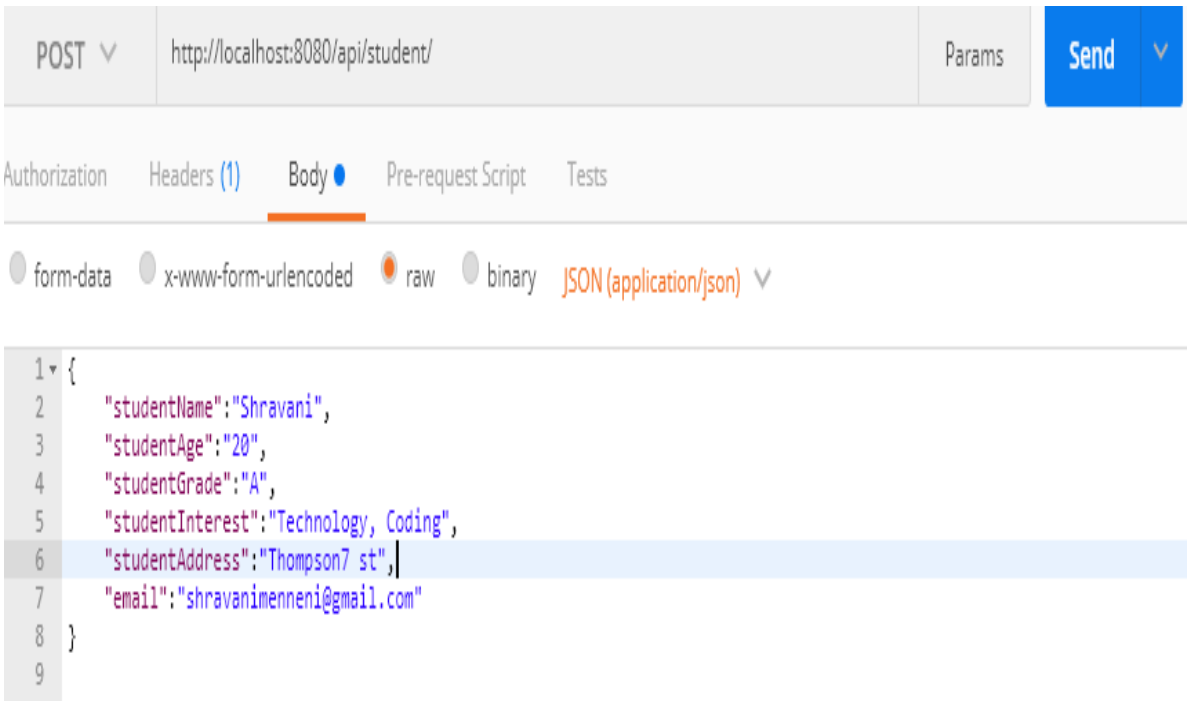


Figure 38. POST Operation

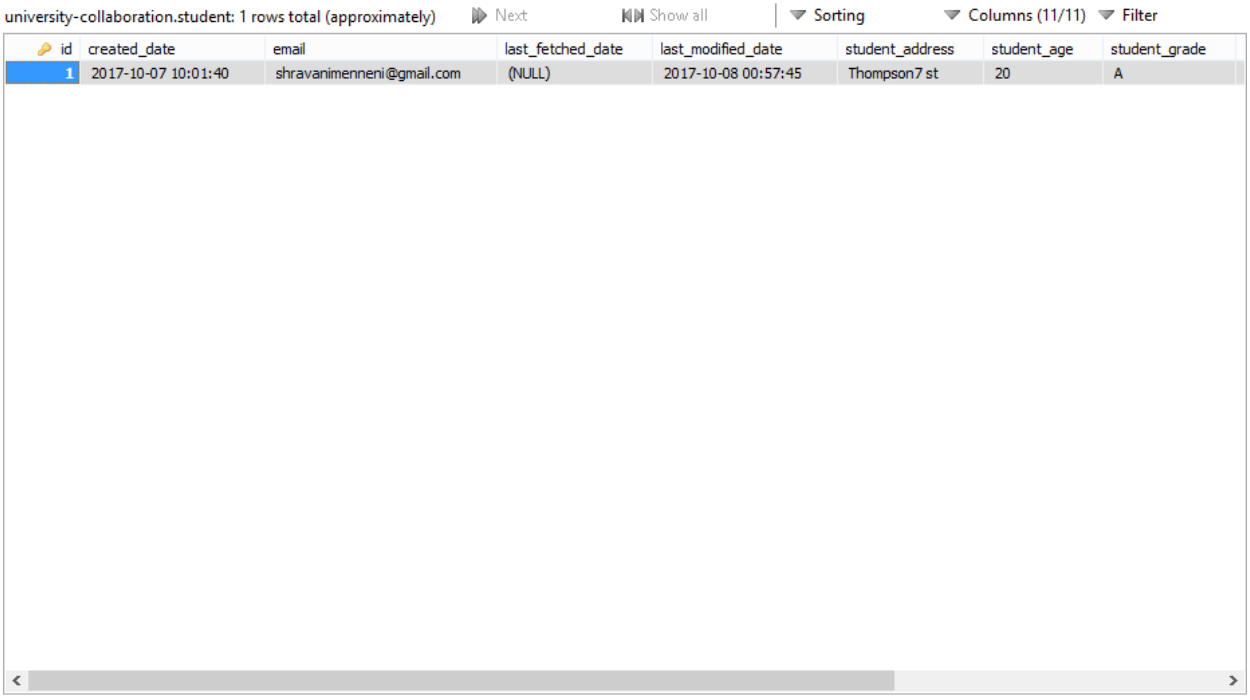


Figure 39. Database Before Modification

- Meanwhile, go to postman and update the student as seen below in Figure 40.

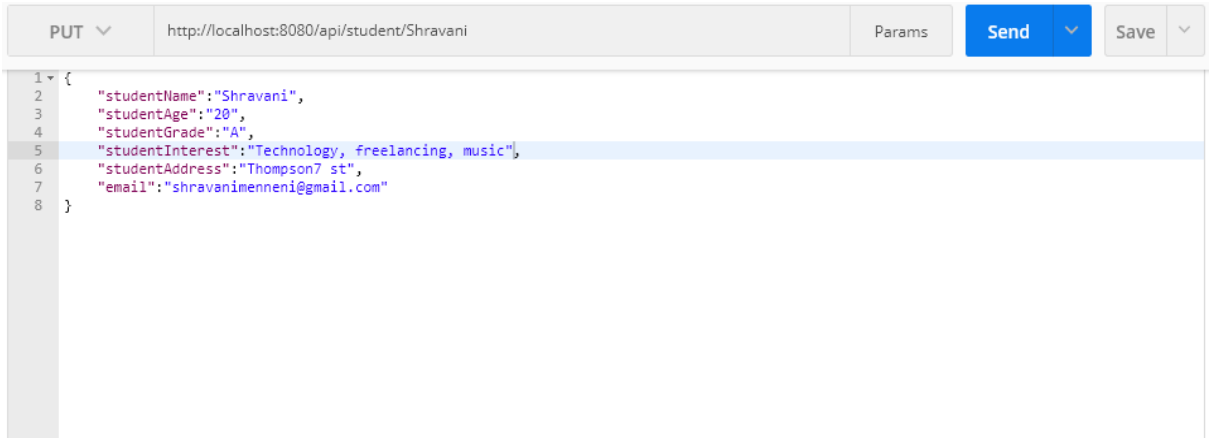


Figure 40. PUT Operation

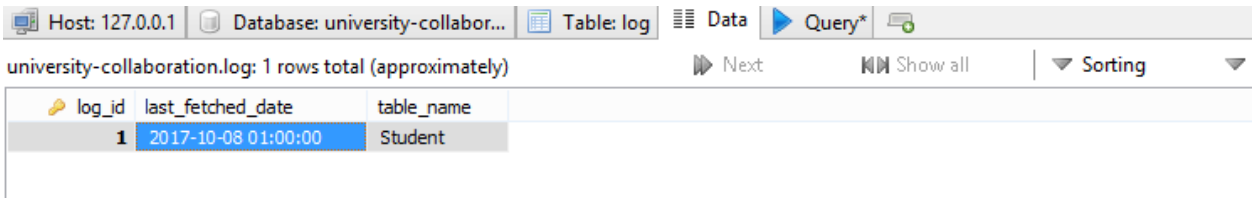


Figure 41. Modified Date in Log Table

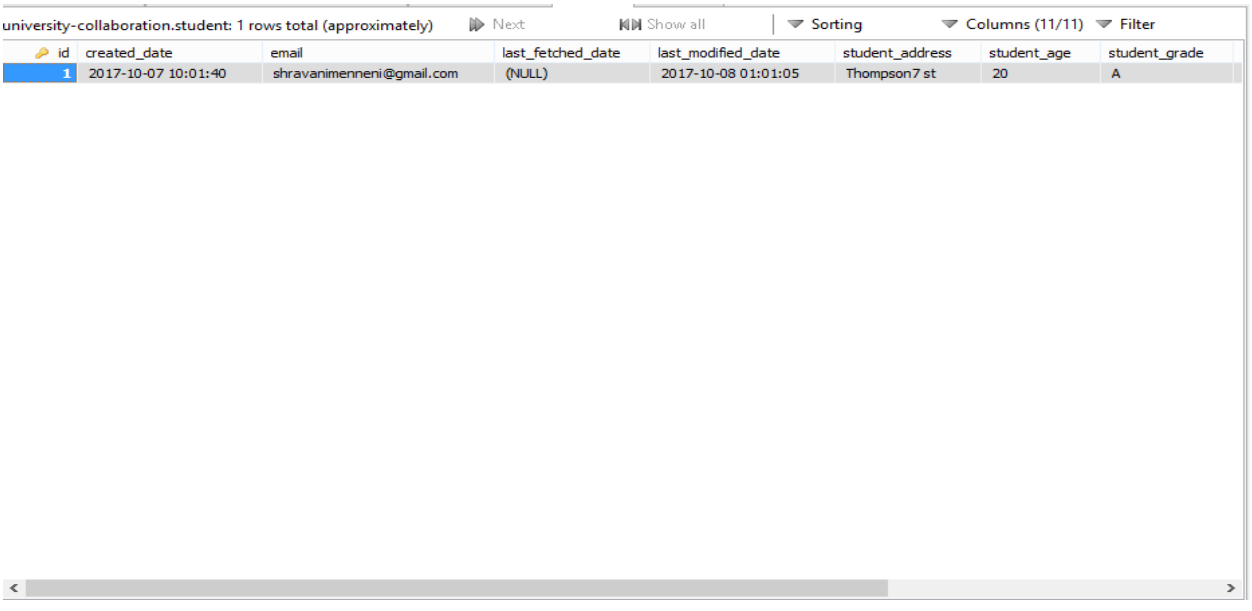


Figure 42. Student Table After Modification

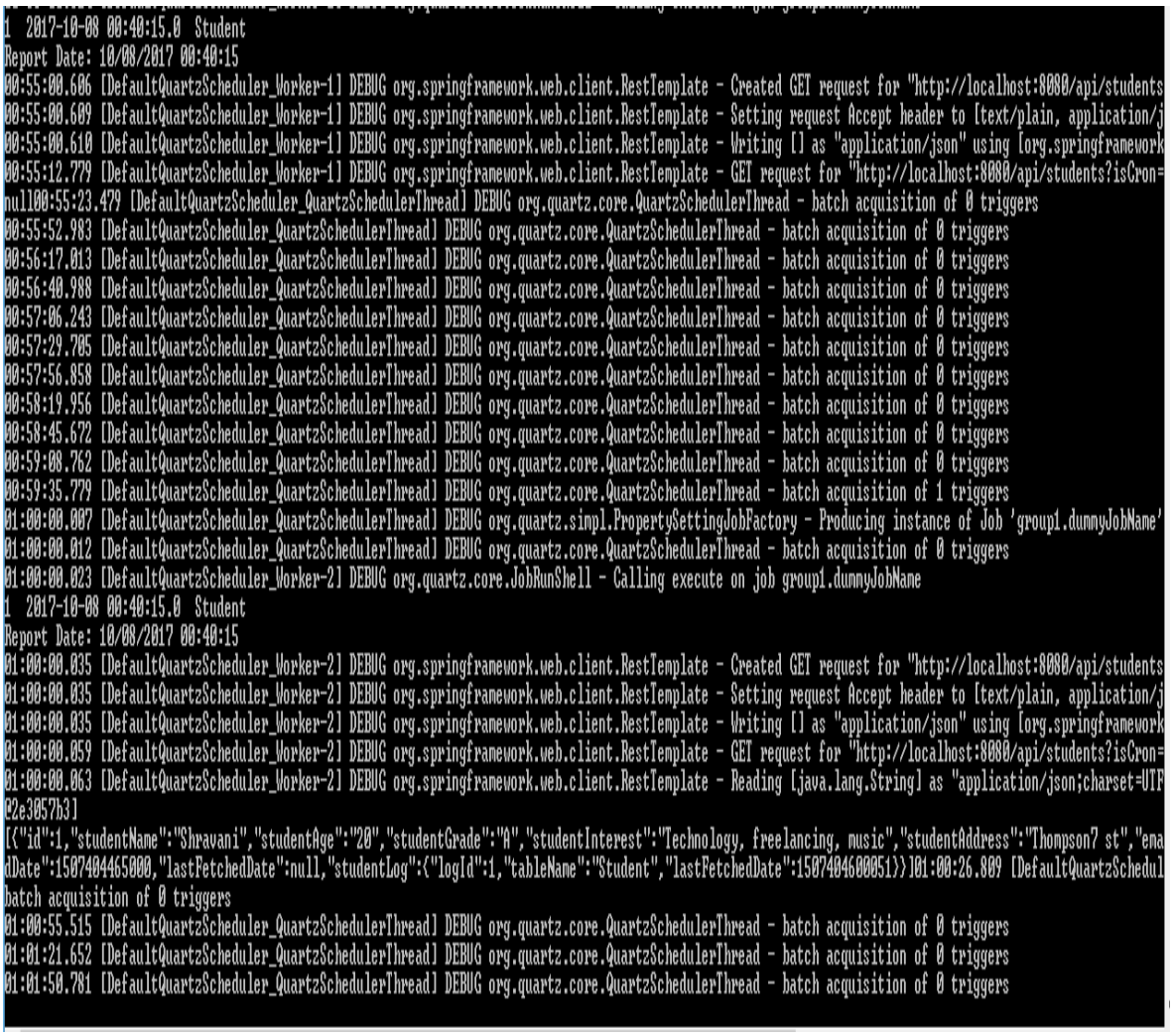


Figure 43. Updated Data is Printed as a Response in the Cron Job Fetch

Figure 44. Log Table Cron Job

Chapter V: Conclusion

In this paper, a Java application was developed using Java server pages to replicate the data from the primary host to secondary hosts by making use of the REST API. Even though this process can be achieved by a DBA or through the use of SOAP and RPC, it is time-consuming and solely dependent on the DBA, which makes it complicated to find the issues and resolve them on time. Whereas, with REST API it is easy to replicate the data without any limitations and it can be replicated on as many servers as we need. Also, REST acts as a secure platform for transforming the data and makes it easy to track performance issues.

Virtualization, cloud computing and REST API concepts are implemented here to meet the objective of the paper and achieve the successful implementation of the project. Data shared with REST API in place has more advantages and is more secure when compared to the method involved with DBA handling. This shows better handling of data within the emerging changes in the IT industry.

References

Albeshri, A. A., & Caelli, W. (2010). Mutual protection in a cloud computing environment. In *IEEE 12th International Conference on High Performance Computing and Communications* (pp. 1-3). Melbourne.

Burtsev, A., Srinivasan, K., Radhakrishnan, P., Voruganti, K., & Goodson, G. R. (2009, June). *Fido: Fast inter-virtual-machine communication for enterprise appliances*. Retrieved from https://www.usenix.org/legacy/events/usenix09/tech/full_papers/Burtsev/Burtsev_html/index.html

Gurav, U., & Shaikh, R. (2010). Virtualization: A key feature of cloud computing. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology* (ICWET'10) (pp. 227-229).

Heiser, G. (2008). The role of virtualization in embedded systems. In *IIES 2008 Proceedings of the First Workshop on Isolation and Integration in Embedded Systems* (pp. 11-16). Glasgow, Scotland.

Java Code Geeks. (n.d.). *The gradle build automation handbook*. Retrieved from https://www.javacodegeeks.com/wp-content/uploads/2016/09/Gradle-Build-Automation-Handbook.pdf

JetBrains. (n.d.). *The IntelliJ IDEA help*. Retrieved from https://www.jetbrains.com/help/idea/2016.1/intellij-idea-help.pdf

Lombardi, F., & Di Pietro, R. (2010). Secure virtualization for cloud computing. *Journal of Network and Computer Applications, 34*(4), 1113-1122.

Muthunagai, S., Karthic, C., & Sujatha, S. (2012). Efficient access of cloud resources through virtualization techniques. In *Proceedings of International Conference on Recent Trends in Information Technology* (pp. 174-178).

Pearce, M., Zeadally, S., & Hunt, R. (2013). Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys, 45*(2), 17.

Ren, Y., Liu, L., Zhang, Q., Wu, Q., Guan, J., Kong, J., . . . & Shao, L. (2016). Shared-data optimizations for inter-virtual-machine communication. *ACM Computing Surveys, 48*(4), 49.

Tutorials Point. (n.d.-a). *The MySQL tutorial*. Retrieved from https://www.tutorialspoint.com/mysql/mysql_tutorial.pdf

Tutorials Point. (n.d.-b). *The RESTful web services tutorial*. Retrieved https://www.tutorialspoint.com/restful/

Varanasi, P., & Heiser, G. (2011). Hardware-supported virtualization on arm. In *Proceedings of the Second Asia-Pacific Workshop on Systems* (p. 11).

Wang, J. (2009). *Survey of state-of-the-art in inter-vm communication mechanisms*. Retrieved from https://pdfs.semanticscholar.org/6555/64255d8c6bb4df3f93f128a955101b6e13ed.pdf

Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E., & Corner, M. D. (2009, March). Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (pp. 31-40).

Appendix

Student controller

```
package student.api.controller;


import com.google.gson.*;
import com.google.gson.reflect.TypeToken;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import student.api.model.Log;
import student.api.model.Student;
import org.springframework.web.util.UriComponentsBuilder;
import student.api.service.LogService;
import student.api.service.StudentService;
import student.api.util.CustomErrorType;


import java.lang.reflect.Type;
```

```
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import static org.springframework.util.MimeTypeUtils.APPLICATION_JSON_VALUE;
@RestController
@RequestMapping("/api")
public class StudentController {
    public static final Logger = LoggerFactory.getLogger(StudentController.class);
    @Autowired
    StudentService studentService;

    @Autowired
    LogService logService;

    @RequestMapping(value = "/students", method = RequestMethod.GET)
    public ResponseEntity<List<Student>> listAllStudents(@RequestParam(required = false,
value = "lastFetchedDate") String lastFetchedDate, @RequestParam(required = false, value =
"isCron") boolean isCron) throws ParseException {
```

```
List<Student> students = new ArrayList<>();
if(isCron) {
    if (lastFetchedDate.isEmpty() || lastFetchedDate == "null"){
        students = studentService.findAllStudents();
    }
    else {
        DateFormat df = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
        Date lastFetchDate = null;
        try {
            lastFetchDate = df.parse(lastFetchedDate);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        students = studentService.findAllStudentsUpdatedAfterDate(lastFetchDate);
    }
    if(!students.isEmpty()) {
        Log log= logService.findBytableName("Student");
        log.setLastFetchedDate(new Date());
        logService.saveLog(log);
    }
} else {
    students = studentService.findAllStudents();
```

```
        }


        if (students.isEmpty()) {
            return new ResponseEntity(HttpStatus.NO_CONTENT);
            // You many decide to return HttpStatus.NOT_FOUND
        }
        return new ResponseEntity<List<Student>>(students, HttpStatus.OK);
    }
    @RequestMapping(value = "/student/", method = RequestMethod.POST)
    public ResponseEntity<?> createStudent(@RequestBody Student, UriComponentsBuilder
ucBuilder) {
        logger.info("Creating student : {}", student);


        if (studentService.isStudentExist(student) && student.getDeleted() == true) {
            logger.error("Unable to create new. A User with name {} already exist",
student.getStudentName());
            student.setDeleted(false);
            student.setLastModifiedDate(new Date());
            studentService.saveStudent(student);
            return new ResponseEntity<String>(HttpStatus.CREATED);
//            //return new ResponseEntity(new CustomErrorType("Unable to create. A User with
name " +
```

```
//                  student.getStudentName() + " already exist."),HttpStatus.CONFLICT);
        }
         Log log= logService.findBytableName("Student");
         log.getLogId();
         student.setStudentLog(log);
         student.setCreatedDate(new Date());
         student.setLastModifiedDate(new Date());
         student.setDeleted(false);
         studentService.saveStudent(student);

        HttpHeaders headers = new HttpHeaders();

headers.setLocation(ucBuilder.path("/api/student/{email}").buildAndExpand(student.getEmail())
.toUri());
        return new ResponseEntity<String>(headers, HttpStatus.CREATED);
    }

    @RequestMapping(value = "/updateStudents", method = RequestMethod.PUT, consumes =
APPLICATION_JSON_VALUE)
    @ResponseBody
    public ResponseEntity<?> updateStudents(HttpEntity<String> httpEntity) {
        logger.info("Updating bulk Students " );
```

```
        // Creates the json object which will manage the information received
        GsonBuilder builder = new GsonBuilder();

// Register an adapter to manage the date types as long values
        builder.registerTypeAdapter(Date.class, new JsonDeserializer<Date>() {
            public Date deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext
context) throws JsonParseException {
                return new Date(json.getAsJsonPrimitive().getAsLong());
            }
        });
        Gson = builder.create();
        String body = httpEntity.getBody();
        List<Student> jsonList = gson.fromJson(body, new
TypeToken<ArrayList<Student>>(){}.getType());

        for (Student s: jsonList) {
            studentService.updateStudent(s);
        }
        return new ResponseEntity<Student>(HttpStatus.OK);
    }
    @RequestMapping(value = "/student/{id}", method = RequestMethod.PUT)
    public ResponseEntity<?> updateStudent(@PathVariable("id") String id, @RequestBody
```

```
Student student) {
        logger.info("Updating Student with id {}", id);
        Student currentStudent = studentService.findByName(id);
        if (currentStudent == null) {
            logger.error("Unable to update. Student with id {} not found.", id);
            return new ResponseEntity(new CustomErrorType("Unable to update. Student with id "
+ id + " not found."),
                    HttpStatus.NOT_FOUND);
        }
        currentStudent.setStudentName(student.getStudentName());
        currentStudent.setStudentAddress(student.getStudentAddress());
        currentStudent.setStudentAge(student.getStudentAge());
        currentStudent.setStudentGrade(student.getStudentGrade());
        currentStudent.setStudentInterest(student.getStudentInterest());
        currentStudent.setLastModifiedDate(new Date());
        studentService.updateStudent(currentStudent);
        return new ResponseEntity<Student>(currentStudent, HttpStatus.OK);
    }
    @RequestMapping(value = "/student/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<?> deleteStudent(@PathVariable("id") String name) {
        logger.info("Fetching & Deleting Student with id {}", name);
        Student = studentService.findByName(name);
```

```
        if (student == null) {
            logger.error("Unable to delete. User with id {} not found.", name);
            return new ResponseEntity(new CustomErrorType("Unable to delete. Student with id " +
name + " not found."),
                    HttpStatus.NOT_FOUND);
        }
        student.setDeleted(true);
        student.setLastModifiedDate(new Date());
        studentService.updateStudent(student);
        return new ResponseEntity<Student>(HttpStatus.NO_CONTENT);
    }
    @RequestMapping(value = "/students/", method = RequestMethod.DELETE)
    public ResponseEntity<Student> deleteAllUsers() {
        logger.info("Deleting All Students");

        studentService.deleteAllStudents();
        return new ResponseEntity<Student>(HttpStatus.NO_CONTENT);
    }
}
```

Model

Log

```
package student.api.model;

import javax.persistence.*;

import java.io.Serializable;

import java.util.Date;


@Entity

public class Log implements Serializable{


    @Id

    @GeneratedValue(strategy= GenerationType.AUTO)

    @Column(name = "log_id")

    private Integer logId;

    private String tableName;

    private Date lastFetchedDate;

    public String getTableName() {

        return tableName;

    }

    public void setTableName(String tableName) {

        this.tableName = tableName;

    }

    public Date getLastFetchedDate() {

        return lastFetchedDate;
```

```
    }
    public void setLastFetchedDate(Date lastFetchedDate) {
        this.lastFetchedDate = lastFetchedDate;
    }
    public Integer getLogId() {
        return logId;
    }
    public void setLogId(Integer logId) {
        this.logId = logId;
    }
}tudent
package student.api.model;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;


@Entity
public class Student implements Serializable{
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
```

```
private Integer id;
private String studentName;
private String studentAge;
private String studentGrade;
private String studentInterest;
private String studentAddress;
private String email;

@CreatedDate
private Date createdDate;
@LastModifiedDate
private Date lastModifiedDate;

@Column(name = "deleted", columnDefinition = "boolean default false", nullable = false)
private Boolean deleted;
@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name="logId", referencedColumnName="log_Id")
 public Log studentLog;
public Log getStudentLog() {
    return studentLog;
}
public void setStudentLog(Log studentLog) {
```

```
        this.studentLog = studentLog;
    }
    public Student(String email, String studentName, String studentAge, String studentGrade,
String studentInterest, String studentAddress) {
        this.studentName = studentName;
        this.studentAge = studentAge;
        this.studentGrade = studentGrade;
        this.studentInterest = studentInterest;
        this.studentAddress = studentAddress;
        this.email = email;
    }
    public Student() {
    }

    public String getStudentName() {
        return studentName;
    }
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    public String getStudentAge() {
        return studentAge;
```

```
}
public void setStudentAge(String studentAge) {
    this.studentAge = studentAge;
}
public String getStudentGrade() {
    return studentGrade;
}
public void setStudentGrade(String studentGrade) {
    this.studentGrade = studentGrade;
}
public String getStudentInterest() {
    return studentInterest;
}

public void setStudentInterest(String studentInterest) {
    this.studentInterest = studentInterest;
}
public String getStudentAddress() {
    return studentAddress;
}
public void setStudentAddress(String studentAddress) {
    this.studentAddress = studentAddress;
```

```
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public Integer getId() {
    return id;
}
public void setId(Integer id) {
    this.id = id;
}
public Date getCreatedDate() {
    return createdDate;
}
public void setCreatedDate(Date createdDate) {
    this.createdDate = createdDate;
}
public Date getLastModifiedDate() {
    return lastModifiedDate;
}
```

```
    public void setLastModifiedDate(Date lastModifiedDate) {
        this.lastModifiedDate = lastModifiedDate;
    }
    public Boolean getDeleted() {
        return deleted;
    }
    public void setDeleted(Boolean deleted) {
        this.deleted = deleted;
    }
}
```

Repository

LogRepository

```
package student.api.repository;
import org.springframework.data.repository.CrudRepository;
import student.api.model.Log;
public interface LogRepository extends CrudRepository<Log,String> {
    public Log findBytableName(String tableName);
}
```

StudentRepository

```
package student.api.repository;
import org.springframework.data.repository.CrudRepository;
```

```
import student.api.model.Student;
import java.util.Date;
public interface StudentRepository extends CrudRepository<Student, Long>{
    public  Iterable<Student> findByLastModifiedDateAfter(Date lastModifiedDate);
}
```

Service

StudentService

```
package student.api.service;
import student.api.model.Student;
import java.util.Date;
import java.util.List;


public interface StudentService {
    Student findById(String email);
    Student findByName(String name);
    void saveStudent(Student student);
    void updateStudent(Student student);
    void deleteStudentById(String id);
    List<Student> findAllStudents();
    void deleteAllStudents();
    boolean isStudentExist(Student student);
```

```
    List<Student> findAllStudentsUpdatedAfterDate(Date lastUpdatedDate);
    }
```

StudentServiceImpl

```
package student.api.service;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import student.api.model.Student;
import student.api.repository.StudentRepository;


import java.util.*;
import java.util.concurrent.atomic.AtomicLong;
@Service("studentService")
public class StudentServiceImpl implements StudentService{
    private static final AtomicLong counter = new AtomicLong();
    private static List<Student> students;
    @Autowired
    public StudentRepository;


    static  {
        students = populateDummyStudents();
    }
```

```
@Override
public Student findById(String email) {
    List<Student> students_ = (List<Student>) studentRepository.findAll();
    for(Student : students_){
        if(student.getEmail().equalsIgnoreCase(email)){
            return student;
        }
    }
    return null;
}
@Override
public Student findByName(String name) {
    List<Student> students_ = (List<Student>) studentRepository.findAll();
    for(Student : students_){
        if(student.getStudentName().equalsIgnoreCase(name)){
            return student;
        }
    }
    return null;
}

@Override
```

```
public void saveStudent(Student student) {
    studentRepository.save(student);
}

@Override
public void updateStudent(Student student) {
    studentRepository.save(student);
}
@Override
public void deleteStudentById(String id) {
    for(Student : students){
        if(student.getEmail().equalsIgnoreCase(id)){
            studentRepository.delete(student);
        }
    }
}

@Override
public List<Student> findAllStudents() {
    List<Student> students = (List<Student>) studentRepository.findAll();

    /*for (Student: students) {
```

```
            student.getLog().setLastFetchedDate(new Date());
            studentRepository.save(student);
        }*/
        return students;
    }

    public List<Student> findAllStudentsUpdatedAfterDate(Date lastUpdatedDate){
        return (List<Student>) studentRepository.findByLastModifiedDateAfter(lastUpdatedDate);
    }
    @Override
    public void deleteAllStudents() {
        studentRepository.deleteAll();
    }
    @Override
    public boolean isStudentExist(Student student) {
        return findByName(student.getStudentName())!=null;
    }
    private static List<Student> populateDummyStudents(){
        List<Student> students = new ArrayList<Student>();
        students.add(new Student("student1@email.com", "student1", "20", "A", "music,
gardening", "address1"));
        students.add(new Student("student2@email.com", "student2", "19", "B", "music,
```

```
gardening", "address2"));
        students.add(new Student("student3@email.com", "student3", "21", "O", "music,
gardening", "address3"));
        students.add(new Student("student4@email.com", "student4", "18", "C", "music,
gardening", "address4"));
        return students;
    }
}
```

Util

```
package student.api.util;
public class CustomErrorType {
    private String errorMessage;
    public CustomErrorType(String errorMessage){
        this.errorMessage = errorMessage;
    }
    public String getErrorMessage() {
        return errorMessage;
    }
}
```

StudentCollaborationMain

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.SpringBootServletInitializer;
 org.springframework.data.jpa.repository.config.EnableJpaRepositories;


@SpringBootApplication(scanBasePackages={"student.api", "student.api.repository"})
@EnableJpaRepositories("student.api.repository")
@EntityScan("student.api.model")
public class StudentCollarbrationMain  extends SpringBootServletInitializer {


    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(StudentCollarbrationMain.class);
    }
    public static void main(String[] args) {
        SpringApplication.run(StudentCollarbrationMain.class, args);
    }
}
```

Cron

MySqlCon

```
package cron;

import student.api.model.Log;

import java.sql.*;

import java.util.List;

public class MySqlCon {

    public Log getLogTableDetails(String tableName) {

        Log  = new Log();

        try {

            Connection con =

DriverManager.getConnection("jdbc:mysql://localhost:3306/university-collaboration", "root",

"mysql");

            Statement stmt = con.createStatement();

            ResultSet rs = stmt.executeQuery("select * from log where table_name='" + tableName +

"'");

            while (rs.next()) {

                System.out.println(rs.getInt(1) + "  " + rs.getString(2) + "  " + rs.getString(3));

                log.setLogId(rs.getInt(1));

                log.setTableName(rs.getString(3));

                log.setLastFetchedDate(rs.getTimestamp(2));

            }

                con.close();
```

```
        } catch (Exception e) {
            System.out.println(e);
        }
        return log;
    }
}
```

StudentCollaborationSimpleCron

```
package cron;
import org.quartz.*;
import org.quartz.impl.StdSchedulerFactory;
public class StudentCollaborationClientSimpleJobMain {
public static void main(String args[]) throws Exception{
        JobDetail job = JobBuilder.newJob(StudentCollaborationJob.class)
                .withIdentity("fetchLatestmodificationsJob", "group1").build();
        Trigger = TriggerBuilder
                .newTrigger()
                .withIdentity("latestModificationTrigger", "group1")
                .withSchedule(
                        CronScheduleBuilder.cronSchedule("0 0/5 * 1/1 * ? *"))
                .build();
```

```
        //schedule it
        Scheduler = new StdSchedulerFactory().getScheduler();
        scheduler.start();
        scheduler.scheduleJob(job, trigger);
    }
}
```

StudentCollaborationJob

```
package cron;
import com.google.gson.Gson;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.reflect.TypeToken;
import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
import org.springframework.http.*;
import org.springframework.web.client.RestTemplate;
import student.api.model.Log;
import student.api.model.Student;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
public class StudentCollaborationJob implements Job {
    private String server = "http://localhost:8080/api";
    private RestTemplate rest;
    private HttpHeaders headers;
    private HttpStatus status;

    public StudentCollaborationJob() {
        this.rest = new RestTemplate();
        this.headers = new HttpHeaders();
        headers.add("Content-Type", "application/json");
        headers.add("Accept", "*/*");
    }
    public String get(String uri) {
        HttpEntity<String> requestEntity = new HttpEntity<String>("", headers);
        ResponseEntity<String> responseEntity = rest.exchange(server + uri, HttpMethod.GET,
requestEntity, String.class);
        this.setStatus(responseEntity.getStatusCode());
        return responseEntity.getBody();
    }
```

```
    public String post(String uri, String json) {
        HttpEntity<String> requestEntity = new HttpEntity<String>(json, headers);
        ResponseEntity<String> responseEntity = rest.exchange(server + uri, HttpMethod.POST,
requestEntity, String.class);
        this.setStatus(responseEntity.getStatusCode());
        return responseEntity.getBody();
    }
    public void put(String uri, String json) {
        HttpEntity<String> requestEntity = new HttpEntity<String>(json, headers);
        ResponseEntity<String> responseEntity = rest.exchange(server + uri, HttpMethod.PUT,
requestEntity, String.class);
        this.setStatus(responseEntity.getStatusCode());
    }

    public void delete(String uri) {
        HttpEntity<String> requestEntity = new HttpEntity<String>("", headers);
        ResponseEntity<String> responseEntity = rest.exchange(server + uri,
HttpMethod.DELETE, requestEntity, String.class);
        this.setStatus(responseEntity.getStatusCode());
    }
    public HttpStatus getStatus() {
```

```
    return status;
}


public void setStatus(HttpStatus status) {
    this.status = status;
}
@Override
public void execute(JobExecutionContext context) throws JobExecutionException {
    MySqlCon con = new MySqlCon();
    Log = con.getLogTableDetails("Student");
    Date = log.getLastFetchedDate();


    DateFormat df = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
    String reportDate = "";
    if(date != null) {
        reportDate = df.format(date);
        System.out.println("Report Date: " + reportDate);
    }
    String response = this.get("/students?isCron=true&lastFetchedDate=" + reportDate);
    System.out.print(response);
    if(response != null) {
        this.put("/updateStudents", response);
```

```
        }
    }
}
```

build.gradle

```
group '1'
version '1.0-SNAPSHOT'
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'org.springframework.boot:spring-boot-gradle-plugin:1.5.6.RELEASE'
        'com.bmuschko:gradle-tomcat-plugin:2.3'
    }
}
apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'org.springframework.boot'
dependencies {
```

```
    def tomcatVersion = '7.0.79'
    tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
            "org.apache.tomcat.embed:tomcat-embed-logging-juli:${tomcatVersion}"
    tomcat("org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}") {
        exclude group: 'org.eclipse.jdt.core.compiler', module: 'ecj'
    }
}
tomcatRun.contextPath='/'
tomcatRunWar.contextPath='/'
jar {
    baseName = 'student-collaboration-rest-service'
    version =  '0.1.0'
}
task fatJar(type: Jar) {
    manifest {
        attributes 'Main-Class': 'cron.StudentCollaborationClientSimpleJobMain'
    }
    baseName = project.name + '-all'
    from { configurations.compile.collect { it.isDirectory() ? it : zipTree(it) } }
    with jar
}
sourceCompatibility = 1.8
```

```
repositories {
    mavenCentral()
}
dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    providedRuntime("org.springframework.boot:spring-boot-starter-tomcat")
    compile 'org.springframework.boot:spring-boot-starter-data-jpa'
    compile 'com.google.code.gson:gson:2.3.1'
    compile 'mysql:mysql-connector-java'
    compile 'org.quartz-scheduler:quartz:2.1.5'
    compile 'org.slf4j:slf4j-api:1.6.1'
    testCompile group: 'junit', name: 'junit', version: '4.12'
    providedCompile "javax.servlet:javax.servlet-api:3.1.0"
}
```

Settings.gradle

```
/*
 * This settings file was auto generated by the Gradle buildInit task
 * by 'mpilli' at '5/10/17 11:45 AM' with Gradle 2.14.1
 *
 * The settings file is used to specify which projects to include in your build.
 * In a single project build this file can be empty or even removed.
```

```
 *
 * Detailed information about configuring a multi-project build in Gradle can be found
 * in the user guide at https://docs.gradle.org/2.14.1/userguide/multi_project_builds.html
 */
/*
// To declare projects as part of a multi-project build use the 'include' method
include 'shared'
include 'api'
include 'services:webservice'
*/
rootProject.name = 'studentcollaboration'
```