Configuration Manual

MSc Research Project
Cyber Security

Shiva Ramasamy
Student ID: X20135530

School of Computing
National College of Ireland

Supervisor: Ross Spelman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shiva Ramasamy
Student ID:	X20135530
Programme:	Cyber Security
Year:	2022
Module:	MSc Research Project
Supervisor:	Ross Spelman
Submission Due Date:	15/08/2022
Project Title:	Hybrid Security For Securing A Combination Of Physical And Virtual Information Assets
Word Count:	3284
Page Count:	18

Signature:	Shiva Ramasamy
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shiva Ramasamy
X20135530

1 Introduction

This document presents the configuration manual for the primary research settings and runtime pertaining to the topic "Hybrid security for securing a combination of physical and virtual information assets". The security design is created for an organisation having distributed critical IT assets in a large campus, such as an industrial setting. One may visualise several servers installed in small scale data centres running monitoring and control applications of the production lines manageable through Augmented Reality (Gomes et al.; 2020) (Palmarini et al.; 2018). Such campuses may be as large as 100 square kilometres (such as a primary metal manufacturing plant). For security administrators, it will be very challenging to administer hundreds of servers running the production lines. This research investigates a "hybrid security monitoring system" that can detect physical as well as virtual threats and raise alarms to the security administrators.

The design of the hybrid security monitoring system is presented in Figure 1:

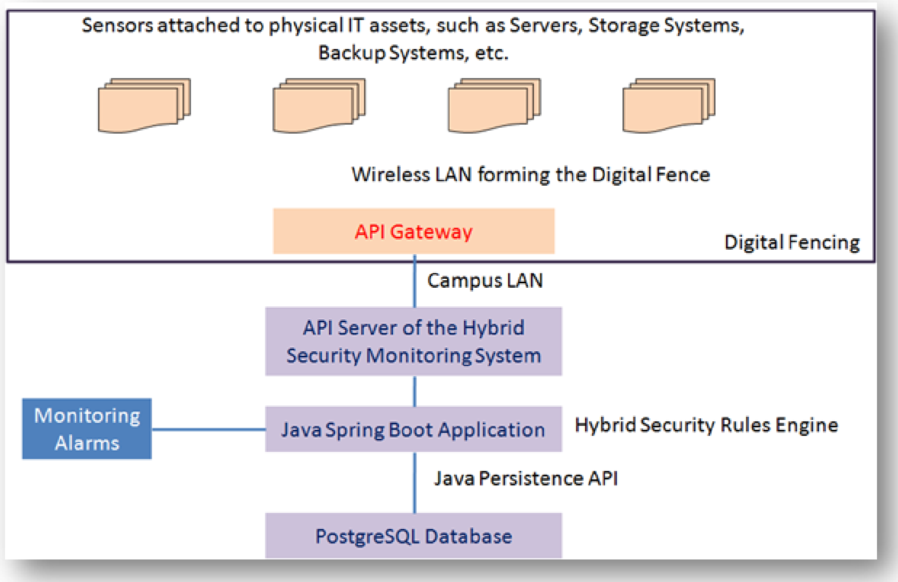

Sensors attached to physical IT assets, such as Servers, Storage Systems, Backup Systems, etc.
Wireless LAN forming the Digital Fence
API Gateway
Digital Fencing
Campus LAN
API Server of the Hybrid Security Monitoring System
Monitoring Alarms
Java Spring Boot Application
Hybrid Security Rules Engine
Java Persistence API
PostgreSQL Database

Figure 1: Hybrid Security Monitoring System

Figure 1 shows a digital fencing around physical IT assets. A digital fence can be configured by deploying at least three Wi-Fi access points such that the local X, Y, and Z coordinates within the digital fence can be calculated using several metrics of localisation in a network of Wi-Fi access points and Internet of Things deployment (Basri and Elkhadimi; 2020) (Braggaar; 2018).

A large campus may have hundreds of such small fences. The IT assets within the digital fences will have Internet of Things sensors attached with them communicating with their respective API gateways. Each fence will have an API gateway consolidating the data collected from the sensors in a JSON file that shall be sent to a centralised API server over the campus LAN. In this research, the API gateway and the API server have been programmed and tested. The API gateway is configured using Insomnia and the API server is programmed in Java 13 using the Spring Boot framework. The database used is PostgreSQL. The Spring Boot application comprises a rules engine written in Java 13 that receives the JSON files from the API gateways, matches with the stated rules and the generates the alarms for the security administrators.

The variables entered in the JSON file of the API gateways are: X-location, Y-location, Z-location, CPU fan speed, CPU temperature, GPU fan speed, GPU temperature, whether three logins failed, whether critical folder access denied occurred three times, whether critical software installation failed, and whether critical security patch installation failed. In practice, the sensors will send the data to the API gateway, which in turn will enter them into a JSON file and send to the API server. In this research, the JSON data entry is done manually in the same format as the sensor data will be entered in it. The subsequent section presents the installation instructions and the runtime reports.

2 Preparing the Ubuntu 20 virtual box inside Windows 10

The steps for preparing the Ubuntu 20 environment are the following:

(a) Download and install VMware Workstation 16 Pro (an evaluation copy is sufficient) for Windows 10 (a simple Windows installation).

(b) Download the ISO file of Ubuntu 20.0.4 available on the Ubuntu.com website.

(c) Launch VMware Workstation 16 Pro and click on the icon allowing creation of a new virtual machine.

(d) It will open a wizard for creating a guest operating system on the top of Windows 10.

(e) In the wizard, give the path to the ISO file of Ubuntu 20.0.4nd press Next.

(f) Setup the machine name, username and password and press next.

(g) Give the virtual machine a unique name and also a path, and press next.

(h) Assign a size of disk to Ubuntu (preferred is 100GB); prefer splitting of the virtual disk into multiple files and press next.

(i) This screen will show all the settings done for the virtual machine. Confirm them and press finish.

(j) Now the virtual machine hardware settings will be shown. Verify the configurations and press OK.

(k) The virtual machine will be installed. After installation, it will have two tabs: home and virtual machine name allocated.

(l) Press on a link "power on this VM"; Ubuntu installation will begin.

(m) It will give two options: Try Ubuntu and Install Ubuntu; press the button “Install Ubuntu.

(n) Now follow the instructions on the subsequent screens. It is a simple “Windows-type” installation.

(o) The Ubuntu 20.0.4 desktop will be visible after installation. Just login and the desktop screen appears.

The next steps are to install the development and runtime environment. To prepare for it, the following steps should be taken:

Launch the Terminal and run the following command:

```
sudo su
```

Input the administrator password. The main command prompt will appear. Now run the following command:

```
sudo apt-get update
```

This command should be run at the beginning and after every installation.

3 Installing the Java environment

The next step is to install Java JDK 13. In the terminal, run the following command:

```
sudo apt install openjdk-13-jdk
```

Press y and then ”Enter” to start installation. Ubuntu will begin installation by downloading Java JDK 13 from the Ubuntu servers.

To check installation, run the following command:

```
java –version
```

Finally, run the command sudo apt-get update again.

4 Installing Maven for running Spring Boot Framework

Maven is needed to run the Java Spring Boot framework. Install it by running the following command:

```
sudo apt install maven
```

To verify installation, run the following command:

```
mvn -version
```

Finally, run the command sudo apt-get update again.

5 Installing Insomnia API Gateway

Insomnia is needed to prepare and send the JSON files to the API server. In real world implementation, it will require a code layer for collecting the data from the Internet of Things sensors attached to the IT assets and compile the JSON file. For the testing purpose in this research, the JSON file is created manually.

To install Insomnia, run the following command in the terminal to create an etc file in Ubuntu for Insomnia:

```
echo ”deb [trusted=yes arch=amd64] https://download.konghq.com/insomnia-ubuntu/ default all” — sudo tee -a /etc/apt/sources.list.d/insomnia.list
```

Now, run the following command:

```
sudo apt-get install insomnia
```

Press y and then ”Enter” to start installation.

Finally, run the command sudo apt-get update again

6 Installing PostgreSQL and PGAdmin4

PostgreSQL is the database server in which, the data sent by the API gateway in JSON format is parsed and stored. The API server parses the data received and stores in the PostgreSQL database. There are several steps to be followed to install PostgreSQL and its user interface client called PGAdmin4 (Drake; 2020) . PGAdmin4 can be used to conduct all database tasks from the graphic user interface that is a windows-like interface.

(a) To create file repository configuration, run the following command:

```
sudo sh -c 'echo ”deb http://apt.postgresql.org/pub/repos/apt $(lsb_release − cs) − pgdgmain” > /etc/apt/sources.list.d/pgdg.list′
```

(b) Import signing key:

```
wget –quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc — sudo apt-key add -
```

(c) Run the main command for installing postgresql:

```
sudo apt-get -y install postgresql
```

Press y and then ”Enter” to start installation.

(d) UI Client for PostgreSQL:

Next, install User Interface Client for Postgresql, known as PGAdmin.

Before starting run: sudo apt-get update

(a) Install public key for repository:

sudo curl https://www.pgadmin.org/static/packages$_{p}gadmin_{o}rg.pub|sudoapt-keyadd$

(b) Create Repository configuration file:

sudo sh -c 'echo "deb https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/\$(lsb$_{r}elease-cs)pgadmin4main" > /etc/apt/sources.list.d/pgadmin4.listaptupdate'$

(c)) Install PGAdmin:

sudo apt install pgadmin4

Press y and then "Enter" to start installation.

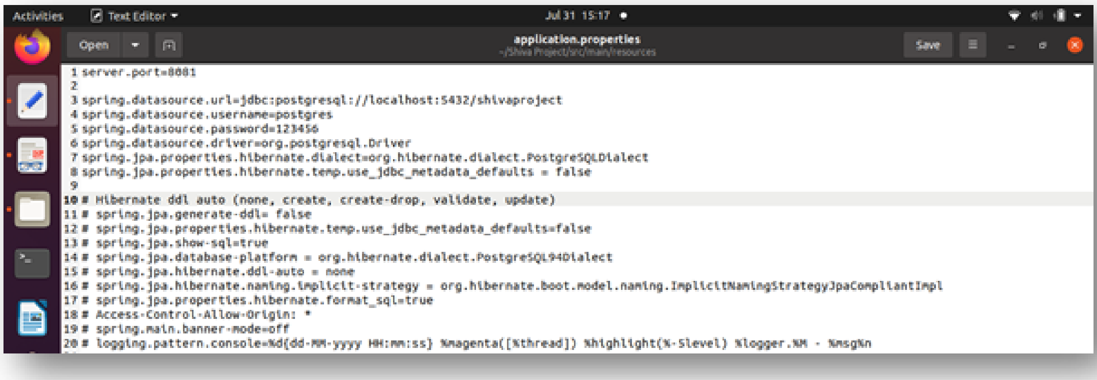
Finally, run the command sudo apt-get update again.

7 Explanation of the codes

The codes written for the API server in Spring Boot are explained in this section. Spring Boot is a Java framework that is built upon Java JDK 13 and Maven. The framework works as a system comprising an internal API server (called embedded Apache Tomcat) and routing of API calls managed through an application properties file (shown in Figure 2) (Webb et al.; 2020). It is a standard file that comes with hashed commands. The hashes need to be removed as per the project requirements. The application Properties file defines the database connection, database name, database access credentials, and the port number to launch the API server. The database connector used is Java Data Base Connector (JDBC) and the data access layer is created using Java Persistent API (JPA) Hibernate running the PostgreSQL driver.

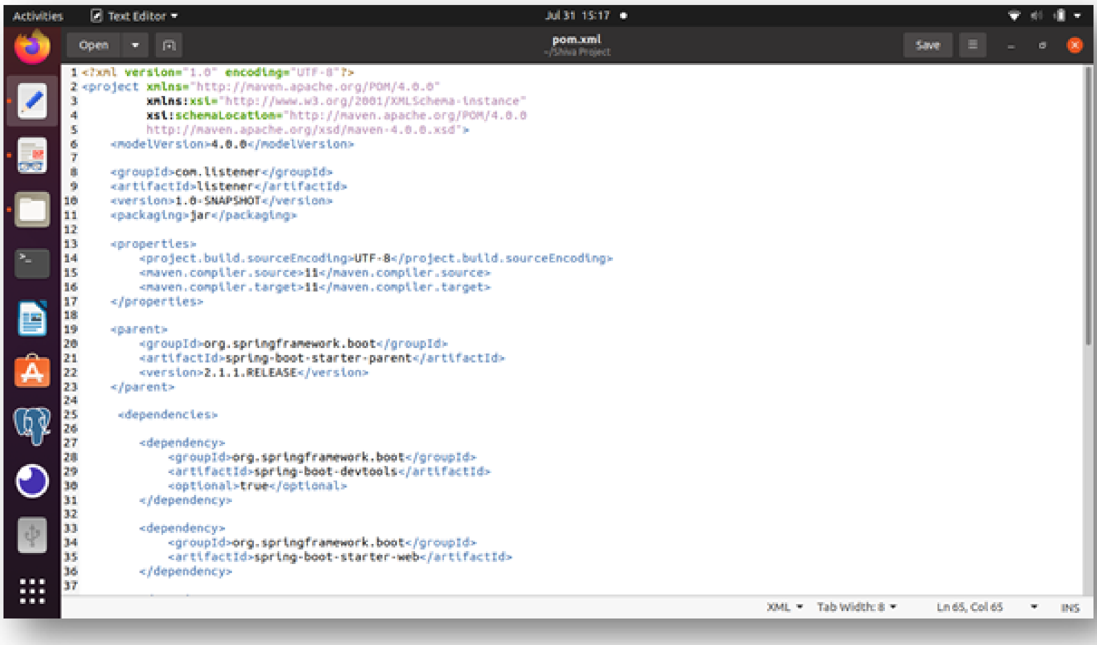
From Figure 2, it may be observed that the TCP port of API server, the path of database for JDBC connection, and the username and password of database connection are hard coded in this file. The PostgreSQL driver and the Java Persistent API (uses JDBC to create a data access layer called boiler plate) are also defined in the application properties file (Oliver Gierke; 2022)l (Webb et al.; 2020). This is a critical file stored deep inside the runtime folder structure. It is accessible only to the application administrators. If deployed on the cloud computing, either in Kubernetes orchestration engine or in a virtual machine, this file is accessible to the virtual private cloud administrator.

The framework installation is controlled through a file called pom.xml (Webb et al.; 2020). Figure 3 shows the screenshot of pom.xml. This file comprises all the dependencies that need to be downloaded and installed in Sprint Boot framework pertaining to a project. Figure 3 presents the Pom.xml configuration of Spring Boot. Pom.xml is an XML application file connecting with the schema at http://maven.apache.org/POM/4.0.0. It is a configuration to request for packages from Spring Boot framework. The application name is defined in this file along with the name of the final JAR file to be compiled. In this research, the application name is “listener” and the JAR file composition is named as: listener-1.0-SNAPSHOT.jar. It comprises all the essential classes of Spring Boot to run the class files created by the Java coding done for the “listener” application. Its composition is explained later in this section.



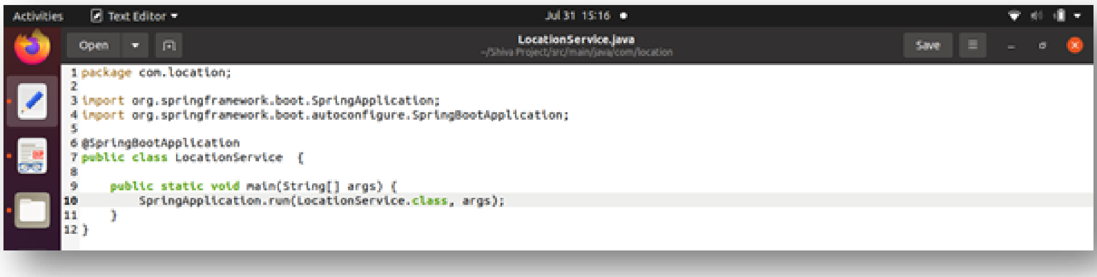
```
server.port=8081

spring.datasource.url=jdbc:postgresql://localhost:5432/shivaproject
spring.datasource.username=postgres
spring.datasource.password=123456
spring.datasource.driver=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false

# Hibernate ddl auto (none, create, create-drop, validate, update)
# spring.jpa.generate-ddl= false
# spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false
# spring.jpa.show-sql=true
# spring.jpa.database-platform = org.hibernate.dialect.PostgreSQL94Dialect
# spring.jpa.hibernate.ddl-auto = none
# spring.jpa.hibernate.naming.implicit-strategy = org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl
# spring.jpa.properties.hibernate.format_sql=true
# Access-Control-Allow-Origin: *
# spring.main.banner-mode=off
# logging.pattern.console=%d{dd-MM-yyyy HH:mm:ss} %magenta([%thread]) %highlight(%-5level) %logger.%M - %msg%n
```

Figure 2: Java Spring Boot Application properties



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.listener</groupId>
    <artifactId>listener</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
    </parent>

     <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
        </dependency>

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

```

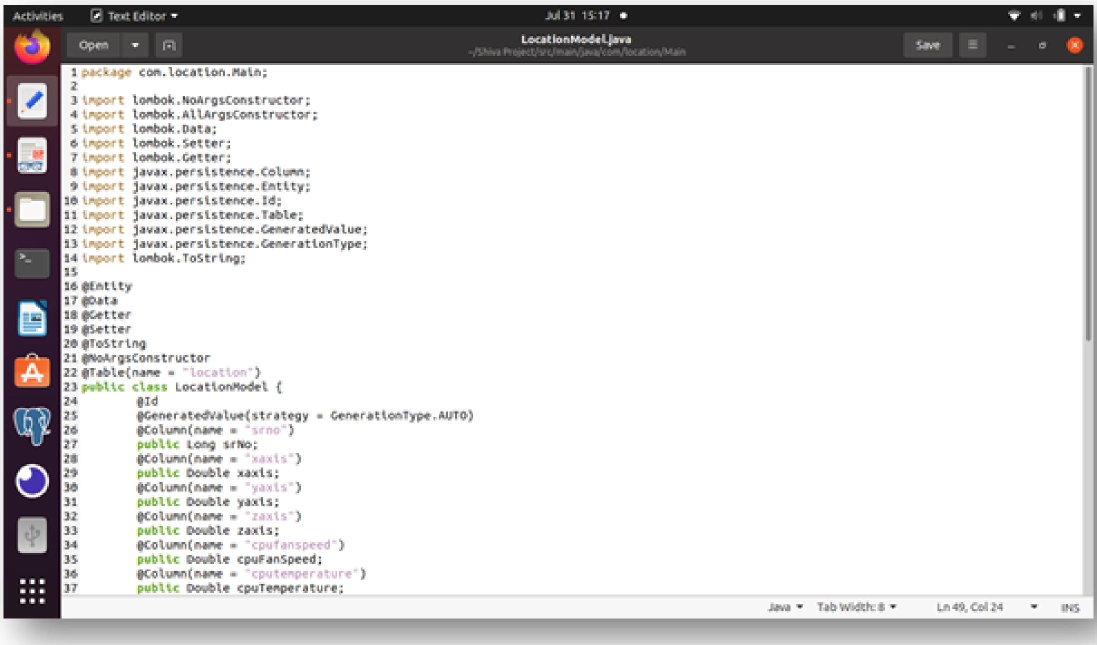
Figure 3: Java Spring Boot pom.xml code snippet



```java
package com.location;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LocationService  {

    public static void main(String[] args) {
        SpringApplication.run(LocationService.class, args);
    }
}
```

Figure 4: Java Main Class for the Spring Boot API server

Figure 4 shows the main class called "LocationService". The package name defined is "location"

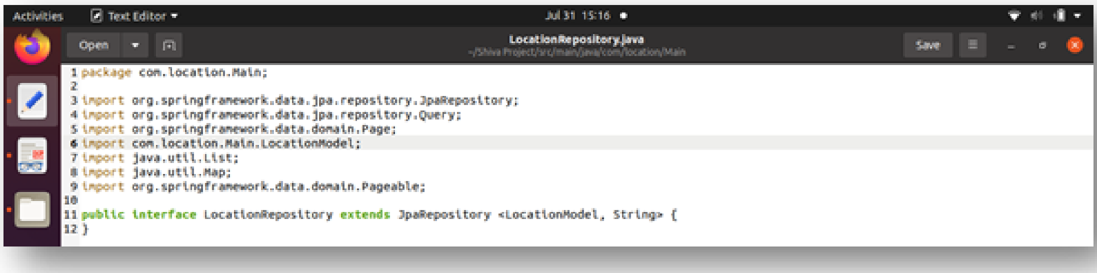


```java
package com.location.Main;

import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Setter;
import lombok.Getter;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import lombok.ToString;

@Entity
@Data
@Getter
@Setter
@ToString
@NoArgsConstructor
@Table(name = "location")
public class LocationModel {
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        @Column(name = "srno")
        public Long srNo;
        @Column(name = "xaxis")
        public Double xaxis;
        @Column(name = "yaxis")
        public Double yaxis;
        @Column(name = "zaxis")
        public Double zaxis;
        @Column(name = "cpufanspeed")
        public Double cpuFanSpeed;
        @Column(name = "cputemperature")
        public Double cpuTemperature;
```

Figure 5: Java Model File for defining the database structure for the Spring Boot API server

Figure 5 shows the class called "LocationModel". It is stored under the "Main" folder of the package "Location". A model file is used to define the structure of data in the database and the corresponding variable name of each data unit. It uses the Lombok and javac.persistence packages along with several dependency injections. When the controller code receives the data, this file helps in saving the parsed data in appropriate columns in the database.

Figure 6 shows the repository code. Its class is "LocationRepository". This code is used to initiate the JPA queries that can be run on the database to fetch data for analysis. The “Map” and “Pageable” packages can organise data fit for creating a data presentation page.



```java
package com.location.Main;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.domain.Page;
import com.location.Main.LocationModel;
import java.util.List;
import java.util.Map;
import org.springframework.data.domain.Pageable;

public interface LocationRepository extends JpaRepository <LocationModel, String> {
}
```

Figure 6: Java Repository File for creating a JPA instance for database connectivity for the Spring Boot API server

Figures 7 and 8 show the main controller code in the Spring Boot API server. Its class is “LocationController”. This code receives the data from the API gateway, sends the data to the repository that uses the model code to save the data in the database, and runs the main rules for hybrid security to generate alarms. There are four risk levels programmed in this code:

(a) Low - when no parameter is breached.
(b) Medium - when one parameter is breahed.
(c) High - when two parameters are breached.
(d) Critical - when three or more parameters are breached.

When the controller receives the data from the API gateway, it runs them through the rules stated as the following:

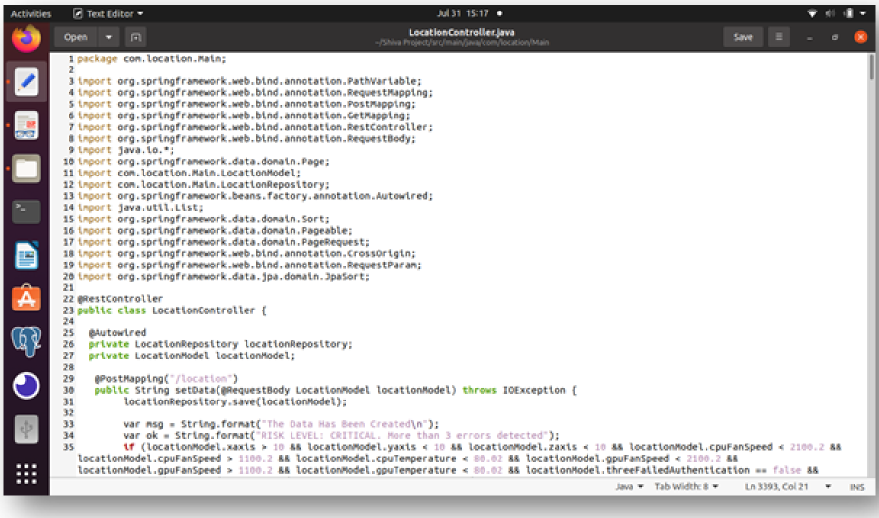
(a) X-location is changed by greater than 10 metres.
(b) Y-location is changed by greater than 10 metres.
(c) Z-location is changed by greater than 10 metres.
(d) CPU fan speed has increased beyond 1100 RPM.
(e) CPU temperature has increased beyond 80 degrees Celsius.
(f) GPU fan speed has increased beyond 1100 RPM.
(g) GPU temperature has increased beyond 80 degrees Celsius.
(h) Whether three logins have failed (Boolean – Yes or No).
(i) Whether critical folder access denied occurred three times (Boolean – Yes or No).
(j) Whether critical software installation failed (Boolean – Yes or No), and/or
(k) Whether critical security patch installation failed (Boolean – Yes or No).

If none of these rules have occurred, the API server will send a response: ”RISK LEVEL: RISK LEVEL: LOW. No errors detected”

If one of these rules has occurred, the API server will send a response: ”RISK LEVEL: MEDIUM. Description of the specific breach.”

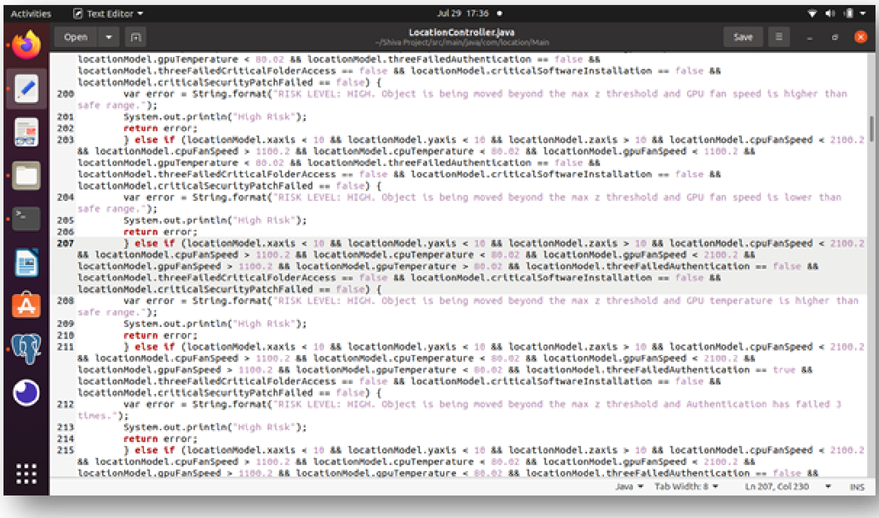
If two of these rules have occurred, the API server will send a response: ”RISK LEVEL: HIGH. Description of the specific breaches.”

If three or more of these rules have occurred, the API server will send a response: ”RISK LEVEL: CRITICAL. More than 3 errors detected”



```java
package com.location.Main;

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestBody;
import java.io.*;
import org.springframework.data.domain.Page;
import com.location.Main.LocationModel;
import com.location.Main.LocationRepository;
import org.springframework.beans.factory.annotation.Autowired;
import java.util.List;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.PageRequest;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.data.jpa.domain.JpaSort;

@RestController
public class LocationController {

  @Autowired
  private LocationRepository locationRepository;
  private LocationModel locationModel;

   @PostMapping("/location")
   public String setData(@RequestBody LocationModel locationModel) throws IOException {
        locationRepository.save(locationModel);

        var msg = String.format("The Data Has Been Created\n");
        var ok = String.format("RISK LEVEL: CRITICAL. More than 3 errors detected");
        if (locationModel.xaxis > 10 && locationModel.yaxis < 10 && locationModel.zaxis < 10 && locationModel.cpuFanSpeed < 2100.2 &&
locationModel.cpuFanSpeed > 1100.2 && locationModel.cpuTemperature < 80.02 && locationModel.gpuFanSpeed < 2100.2 &&
locationModel.gpuFanSpeed > 1100.2 && locationModel.gpuTemperature < 80.02 && locationModel.threeFailedAuthentication == false &&
```

Figure 7: Java Controller File acting as the rules engine for the hybrid security threats detection and raising the alarms for the Spring Boot API server



```java
locationModel.gpuTemperature < 80.02 && locationModel.threeFailedAuthentication == false &&
locationModel.threeFailedCriticalFolderAccess == false && locationModel.criticalSoftwareInstallation == false &&
locationModel.criticalSecurityPatchFailed == false) {
        var error = String.format("RISK LEVEL: HIGH. Object is being moved beyond the max z threshold and GPU fan speed is higher than
safe range.");
        System.out.println("High Risk");
        return error;
        } else if (locationModel.xaxis < 10 && locationModel.yaxis < 10 && locationModel.zaxis > 10 && locationModel.cpuFanSpeed < 2100.2
&& locationModel.cpuFanSpeed > 1100.2 && locationModel.cpuTemperature < 80.02 && locationModel.gpuFanSpeed < 1100.2 &&
locationModel.gpuTemperature < 80.02 && locationModel.threeFailedAuthentication == false &&
locationModel.threeFailedCriticalFolderAccess == false && locationModel.criticalSoftwareInstallation == false &&
locationModel.criticalSecurityPatchFailed == false) {
        var error = String.format("RISK LEVEL: HIGH. Object is being moved beyond the max z threshold and GPU fan speed is lower than
safe range.");
        System.out.println("High Risk");
        return error;
        } else if (locationModel.xaxis < 10 && locationModel.yaxis < 10 && locationModel.zaxis > 10 && locationModel.cpuFanSpeed < 2100.2
&& locationModel.cpuFanSpeed > 1100.2 && locationModel.cpuTemperature < 80.02 && locationModel.gpuFanSpeed < 2100.2 &&
locationModel.gpuFanSpeed > 1100.2 && locationModel.gpuTemperature > 80.02 && locationModel.threeFailedAuthentication == false &&
locationModel.threeFailedCriticalFolderAccess == false && locationModel.criticalSoftwareInstallation == false &&
locationModel.criticalSecurityPatchFailed == false) {
        var error = String.format("RISK LEVEL: HIGH. Object is being moved beyond the max z threshold and GPU temperature is higher than
safe range.");
        System.out.println("High Risk");
        return error;
        } else if (locationModel.xaxis < 10 && locationModel.yaxis < 10 && locationModel.zaxis > 10 && locationModel.cpuFanSpeed < 2100.2
&& locationModel.cpuFanSpeed > 1100.2 && locationModel.cpuTemperature < 80.02 && locationModel.gpuFanSpeed < 2100.2 &&
locationModel.gpuFanSpeed > 1100.2 && locationModel.gpuTemperature < 80.02 && locationModel.threeFailedAuthentication == true &&
locationModel.threeFailedCriticalFolderAccess == false && locationModel.criticalSoftwareInstallation == false &&
locationModel.criticalSecurityPatchFailed == false) {
        var error = String.format("RISK LEVEL: HIGH. Object is being moved beyond the max z threshold and Authentication has failed 3
times.");
        System.out.println("High Risk");
        return error;
        } else if (locationModel.xaxis < 10 && locationModel.yaxis < 10 && locationModel.zaxis > 10 && locationModel.cpuFanSpeed < 2100.2
&& locationModel.cpuFanSpeed > 1100.2 && locationModel.cpuTemperature < 80.02 && locationModel.gpuFanSpeed < 2100.2 &&
locationModel.gpuFanSpeed > 1100.2 && locationModel.gpuTemperature < 80.02 && locationModel.threeFailedAuthentication == false &&
```

Figure 8: Java Controller File showing a snippet of the rules engine coding for the hybrid security threats detection and raising the alarms for the Spring Boot API server

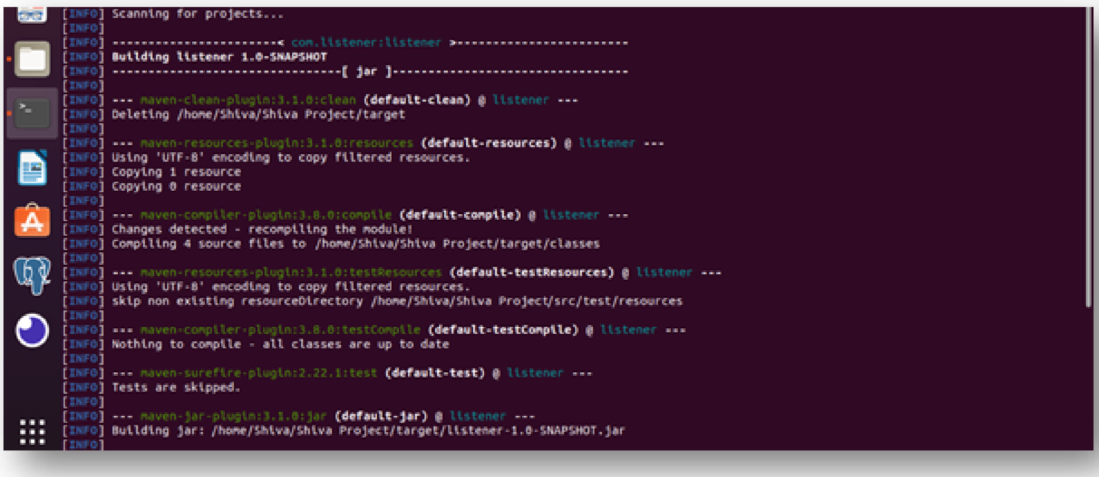
In order to return the right risk level after reading the data set arrived, all possible scenarios of comparison of parameters have been written in the code. Given that there are eleven parameters, writing all possible combination of rules caused the Location Controller Java code to be written in almost hundred pages. This has been done to create a complete model of collaborative security as explained by (Happa et al.; 2019; Plósz et al.; 2017).

The next section presents screenshots and explanations of the tests executed through the runtimes.

8 Explanation of the runtimes

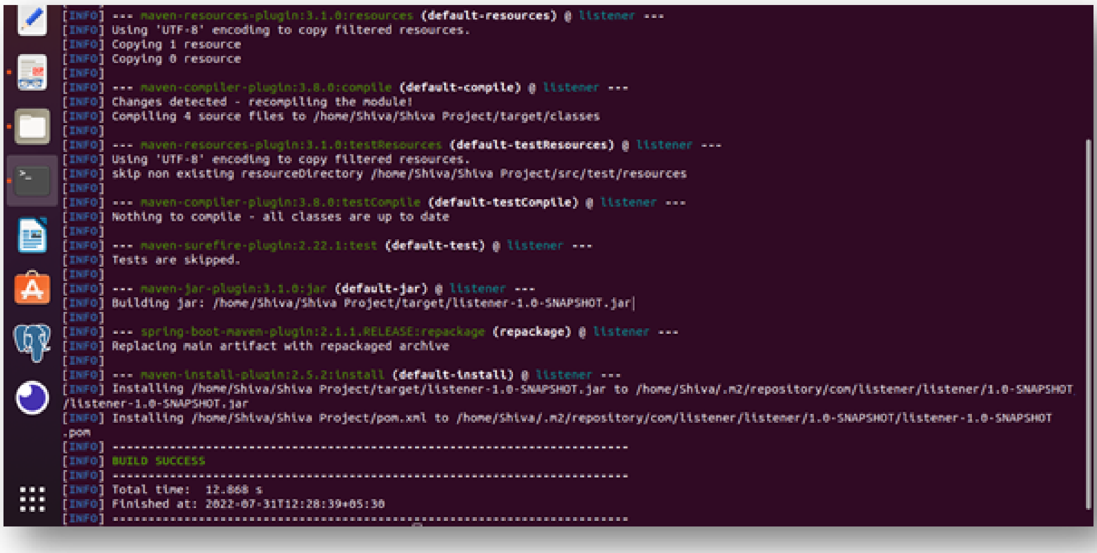
The Spring Boot API server runtime is invoked by the following command at the path /Shiva Project:

mvn clean install -DskipTests



```
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.listener:listener >-----------------------
[INFO] Building listener 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ listener ---
[INFO] Deleting /home/Shiva/Shiva Project/target
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ listener ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ listener ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 4 source files to /home/Shiva/Shiva Project/target/classes
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ listener ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/Shiva/Shiva Project/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ listener ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ listener ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:3.1.0:jar (default-jar) @ listener ---
[INFO] Building jar: /home/Shiva/Shiva Project/target/listener-1.0-SNAPSHOT.jar
[INFO]
```

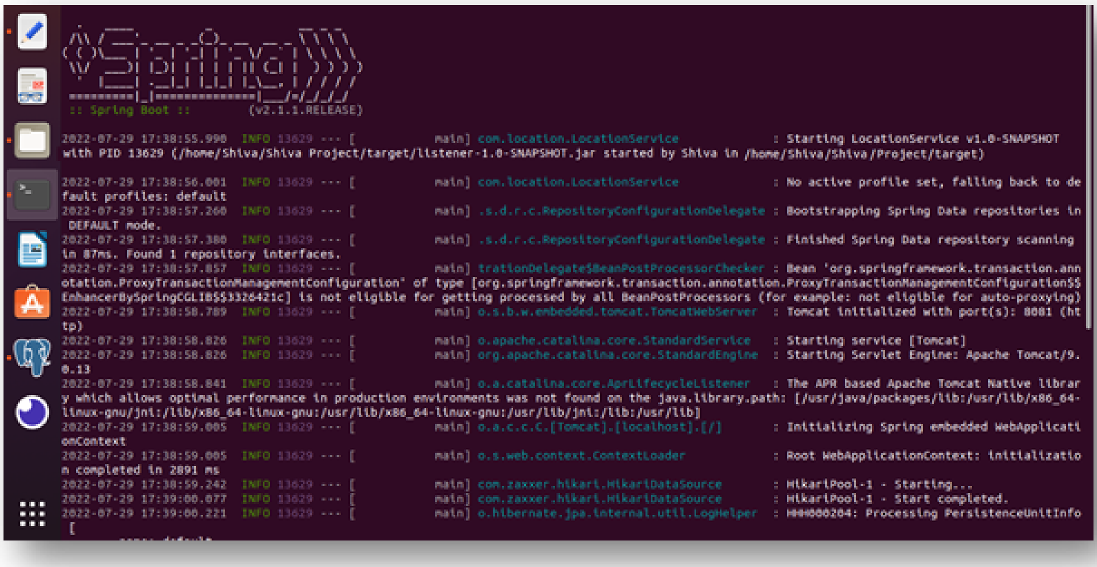
Figure 9: Starting to build the Sprint Boot application for the API server



```
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ listener ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ listener ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 4 source files to /home/Shiva/Shiva Project/target/classes
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ listener ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/Shiva/Shiva Project/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ listener ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ listener ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:3.1.0:jar (default-jar) @ listener ---
[INFO] Building jar: /home/Shiva/Shiva Project/target/listener-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.1.RELEASE:repackage (repackage) @ listener ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ listener ---
[INFO] Installing /home/Shiva/Shiva Project/target/listener-1.0-SNAPSHOT.jar to /home/Shiva/.m2/repository/com/listener/listener/1.0-SNAPSHOT
/listener-1.0-SNAPSHOT.jar
[INFO] Installing /home/Shiva/Shiva Project/pom.xml to /home/Shiva/.m2/repository/com/listener/listener/1.0-SNAPSHOT/listener-1.0-SNAPSHOT
.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  12.868 s
[INFO] Finished at: 2022-07-31T12:28:39+05:30
[INFO] ------------------------------------------------------------------------
```

Figure 10: Ending to build the Sprint Boot application for the API server

The command "mvn" invokes Maven, which is essential for installing a spring boot project as a runtime. This occurs as a .JAR file of the entire project is created in the path: /Shiva Project/target. The screenshot in Figure 9 shows the beginning of this process and the compilation of the four source files created. The compiled classes are stored at /Shiva Project/target/classes. Figure 10 shows completion of the build process and creation of the JAR file as: /Shiva Project/target/listener-1.0-SNAPSHOT.jar.



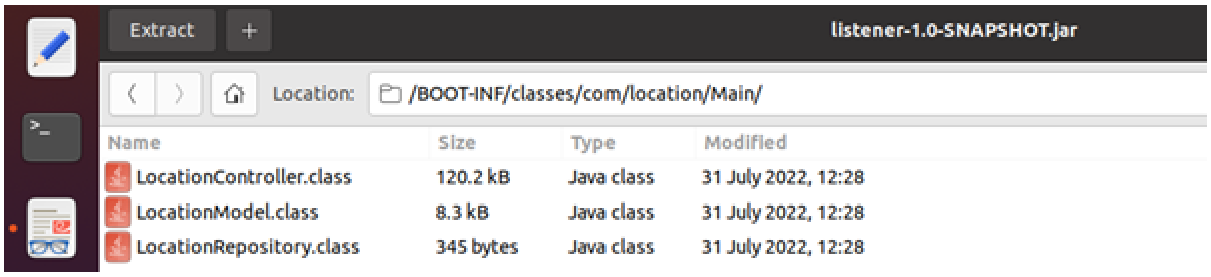
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.1.RELEASE)

2022-07-29 17:38:55.998  INFO 13629 --- [           main] com.location.LocationService             : Starting LocationService v1.0-SNAPSHOT
with PID 13629 (/home/Shiva/Shiva Project/target/listener-1.0-SNAPSHOT.jar started by Shiva in /home/Shiva/Shiva/Project/target)

2022-07-29 17:38:56.001  INFO 13629 --- [           main] com.location.LocationService             : No active profile set, falling back to de
fault profiles: default
2022-07-29 17:38:57.260  INFO 13629 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repositories in
 DEFAULT mode.
2022-07-29 17:38:57.380  INFO 13629 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning
in 87ms. Found 1 repository interfaces.
2022-07-29 17:38:57.857  INFO 13629 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.transaction.ann
otation.ProxyTransactionManagementConfiguration' of type [org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration$$
EnhancerBySpringCGLIB$$3326421c] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2022-07-29 17:38:58.789  INFO 13629 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8081 (ht
tp)
2022-07-29 17:38:58.826  INFO 13629 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-07-29 17:38:58.826  INFO 13629 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/9.
0.13
2022-07-29 17:38:58.841  INFO 13629 --- [           main] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native librar
y which allows optimal performance in production environments was not found on the java.library.path: [/usr/java/packages/lib:/usr/lib/x86_64-
linux-gnu/jni:/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib]
2022-07-29 17:38:59.005  INFO 13629 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicati
onContext
2022-07-29 17:38:59.005  INFO 13629 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initializatio
n completed in 2891 ms
2022-07-29 17:38:59.242  INFO 13629 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-07-29 17:39:00.077  INFO 13629 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-07-29 17:39:00.221  INFO 13629 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo
 [
```

Figure 11: Starting the main JAR file of Sprint Boot application for the API server

Finally, the Spring Boot API Server application is launched through the following command at the path: /Shiva Project/target/

java –jar listener-1.0-SNAPSHOT.jar

Launching of Spring Boot is shown in Figure 11. This command launches all the spring boot services needed to run the application, such as JPA Persistence, Hibernate, Embedded Apache Tomcat, API listener, and Spring Data Repository. As per the "application properties" file (discussed earlier), the API service launches at http://localhost:8081.

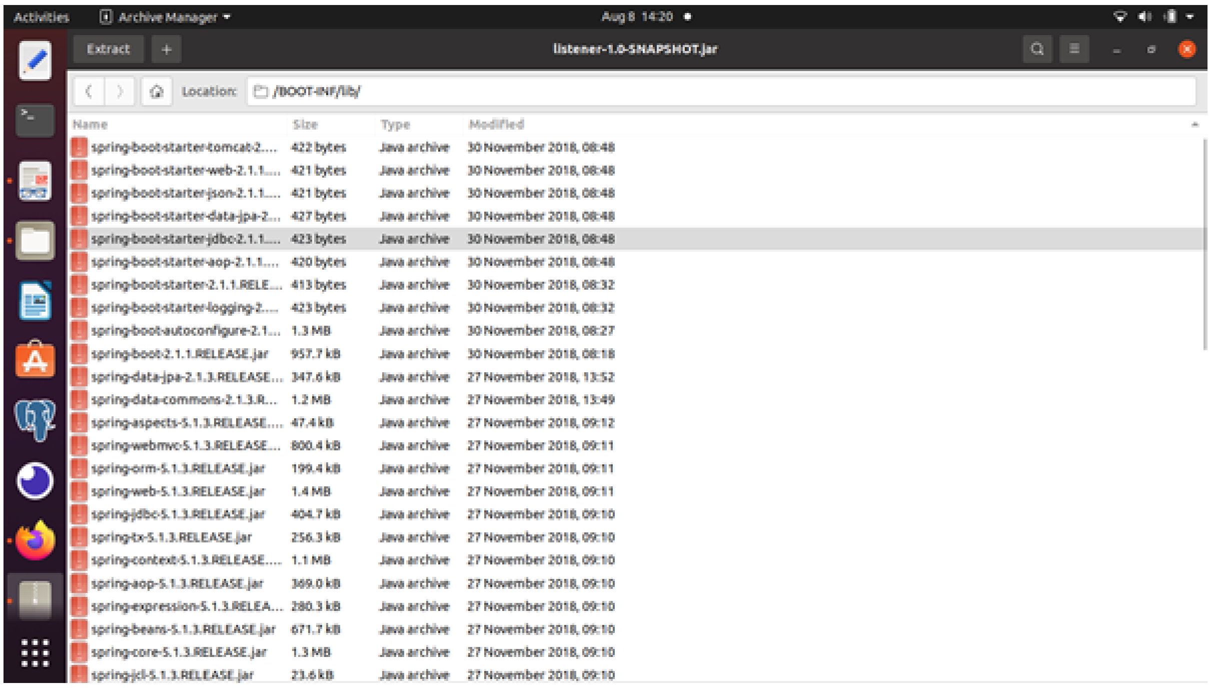


/BOOT-INF/classes/com/location/Main/

Name	Size	Type	Modified
LocationController.class	120.2 kB	Java class	31 July 2022, 12:28
LocationModel.class	8.3 kB	Java class	31 July 2022, 12:28
LocationRepository.class	345 bytes	Java class	31 July 2022, 12:28

Figure 12: The three classes created in the "listener" application shown inside the listener-1.0-SNAPSHOT.jar file

Figures 12 and 13 are presented specifically to show the classes inside the "listener-1.0-SNAPSHOT.jar" JAR file. Figure 7 shows the classes created for this experimentation and Figure 12 shows the Spring Boot classes downloaded and installed by Maven. These classes are essential to run the fully packaged API server with network, transport, session, and application layers using Spring Boot framework.



/BOOT-INF/lib/

Name	Size	Type	Modified
spring-boot-starter-tomcat-2....	422 bytes	Java archive	30 November 2018, 08:48
spring-boot-starter-web-2.1.1....	421 bytes	Java archive	30 November 2018, 08:48
spring-boot-starter-json-2.1.1....	421 bytes	Java archive	30 November 2018, 08:48
spring-boot-starter-data-jpa-2....	427 bytes	Java archive	30 November 2018, 08:48
spring-boot-starter-jdbc-2.1.1....	423 bytes	Java archive	30 November 2018, 08:48
spring-boot-starter-aop-2.1.1....	420 bytes	Java archive	30 November 2018, 08:48
spring-boot-starter-2.1.1.RELE...	413 bytes	Java archive	30 November 2018, 08:32
spring-boot-starter-logging-2....	423 bytes	Java archive	30 November 2018, 08:32
spring-boot-autoconfigure-2.1....	1.3 MB	Java archive	30 November 2018, 08:27
spring-boot-2.1.1.RELEASE.jar	957.7 kB	Java archive	30 November 2018, 08:18
spring-data-jpa-2.1.3.RELEASE...	347.6 kB	Java archive	27 November 2018, 13:52
spring-data-commons-2.1.3.R...	1.2 MB	Java archive	27 November 2018, 13:49
spring-aspects-5.1.3.RELEASE....	47.4 kB	Java archive	27 November 2018, 09:12
spring-webmvc-5.1.3.RELEASE...	800.4 kB	Java archive	27 November 2018, 09:11
spring-orm-5.1.3.RELEASE.jar	199.4 kB	Java archive	27 November 2018, 09:11
spring-web-5.1.3.RELEASE.jar	1.4 MB	Java archive	27 November 2018, 09:11
spring-jdbc-5.1.3.RELEASE.jar	404.7 kB	Java archive	27 November 2018, 09:10
spring-tx-5.1.3.RELEASE.jar	256.3 kB	Java archive	27 November 2018, 09:10
spring-context-5.1.3.RELEASE....	1.1 MB	Java archive	27 November 2018, 09:10
spring-aop-5.1.3.RELEASE.jar	369.0 kB	Java archive	27 November 2018, 09:10
spring-expression-5.1.3.RELEA...	280.3 kB	Java archive	27 November 2018, 09:10
spring-beans-5.1.3.RELEASE.jar	671.7 kB	Java archive	27 November 2018, 09:10
spring-core-5.1.3.RELEASE.jar	1.3 MB	Java archive	27 November 2018, 09:10
spring-jcl-5.1.3.RELEASE.jar	23.6 kB	Java archive	27 November 2018, 09:10

Figure 13: The Spring Boot classes downloaded by Maven and created in the "listener" application shown inside the listener-1.0-SNAPSHOT.jar file

After completing the Spring Boot coding and configuration tasks, the API gateway was installed and configured to prepare it to send the JSON file data to the API server (Figure 14). In this screen, the JSON file can be inserted as a "Body" and the POST REST call can be made by entering the URL of the API server.

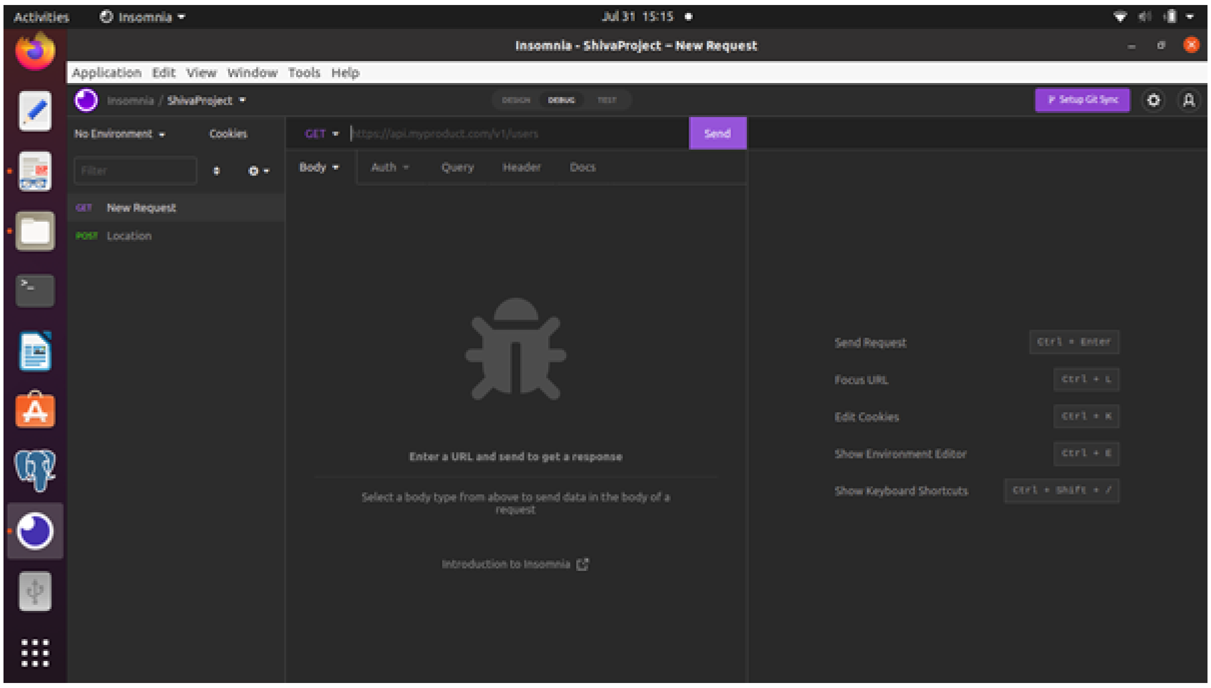
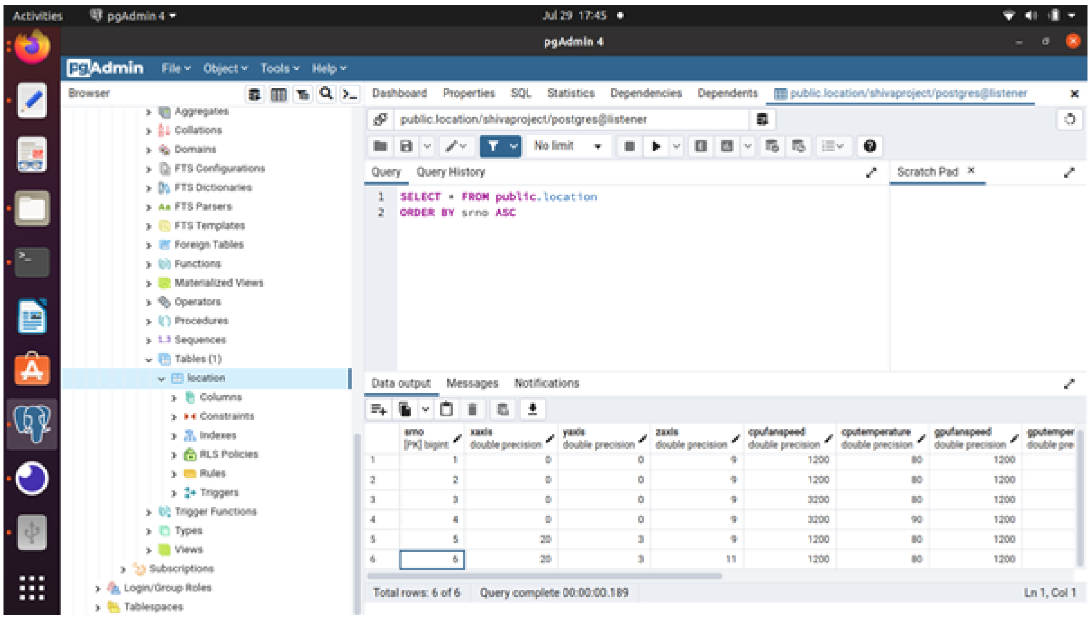

Activities
Insomnia
Jul 31 15:15
Insomnia - ShivaProject – New Request
Application Edit View Window Tools Help
Insomnia / ShivaProject
Setup Git Sync
No Environment
Cookies
GET
https://api.myproduct.com/v1/users
Send
Body
Auth
Query
Header
Docs
GET New Request
POST Location
Enter a URL and send to get a response
Select a body type from above to send data in the body of a request
Introduction to Insomnia
Send Request Ctrl + Enter
Focus URL Ctrl + L
Edit Cookies Ctrl + K
Show Environment Editor Ctrl + E
Show Keyboard Shortcuts Ctrl + Shift + /

Figure 14: Configuring Insomnia to prepare it for JSON object transmission to the API server



```
SELECT * FROM public.location
ORDER BY srno ASC
```

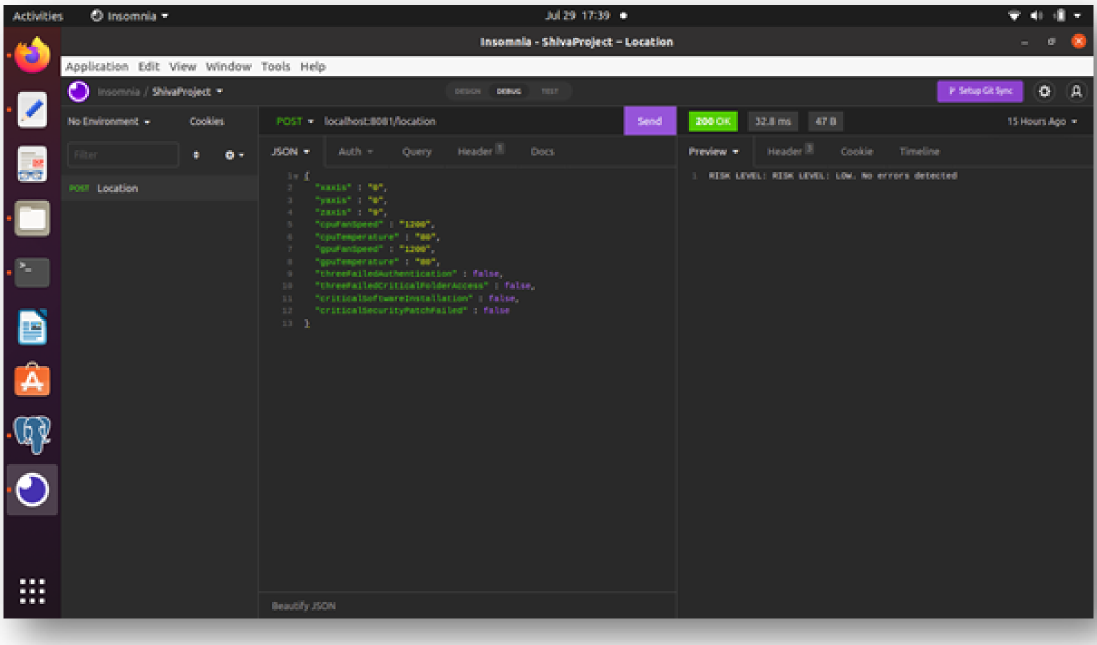
	srno [PK] bigint	xaxis double precision	yaxis double precision	zaxis double precision	cpufanspeed double precision	cputemperature double precision	gpufanspeed double precision	gputemper double pre
1	1	0	0	9	1200	80	1200	
2	2	0	0	9	1200	80	1200	
3	3	0	0	9	3200	80	1200	
4	4	0	0	9	3200	90	1200	
5	5	20	3	9	1200	80	1200	
6	6	20	3	11	1200	80	1200	

Total rows: 6 of 6 Query complete 00:00:00.189 Ln 1, Col 1

Figure 15: PostgreSQL database creation

Finally, the database was created in PostgreSQL as shown in Figure 15 . The database name created was "shivaproject" and the table created was postgres@listener (listener schema accessed by the username postgres). The columns for the eleven parameters were created in the listener table. The JDBC, JPA, and Hibernate settings were tested by sending test data from the API gateway through example JSON files.

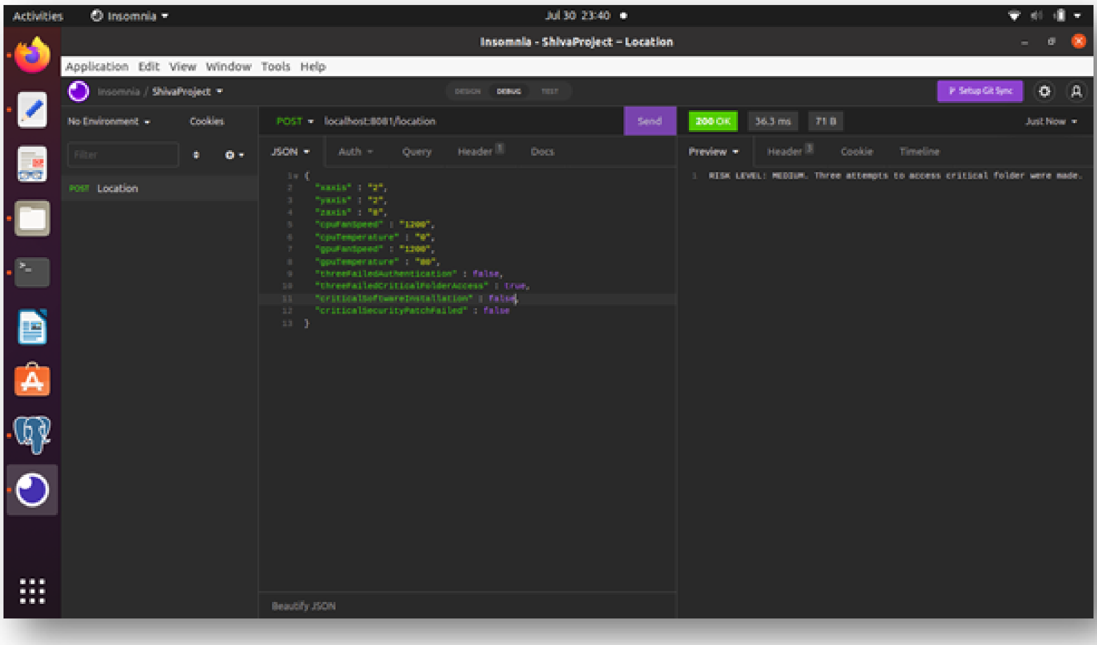
With the API server in operation, the next step is to launch the API gateway, prepare the JSON file, and send to the API service address. Figure 16 shows this process. An insomnia instance is first created within the Insomnia application at the path /Insomnia/ShivaProject. The full API server path is at: http://localhost:8081/location. Here, "location" is the name of the package. Figure 16 also shows the JSON file configuration with the eleven parameters related to hybrid security. The specific JSON file in Figure 16 has no parameter breached. Hence, the response is:'RISK LEVEL: LOW. No errors detected"'. The Figure 17 however shows a breach. The response for the specific breach is: 'RISK LEVEL: MEDIUM. Three attempts to access critical folder was made'.



```
{
    "xaxis" : "0",
    "yaxis" : "0",
    "zaxis" : "0",
    "cpuFanSpeed" : "1200",
    "cpuTemperature" : "80",
    "gpuFanSpeed" : "1200",
    "gpuTemperature" : "80",
    "threeFailedAuthentication" : false,
    "threeFailedCriticalFolderAccess" : false,
    "criticalSoftwareInstallation" : false,
    "criticalSecurityPatchFailed" : false
}
```

```
RISK LEVEL: RISK LEVEL: LOW. No errors detected
```

Figure 16: The API gateway showing JSON file feed to the API server and the response; this screenshot shows a Low Risk Level response



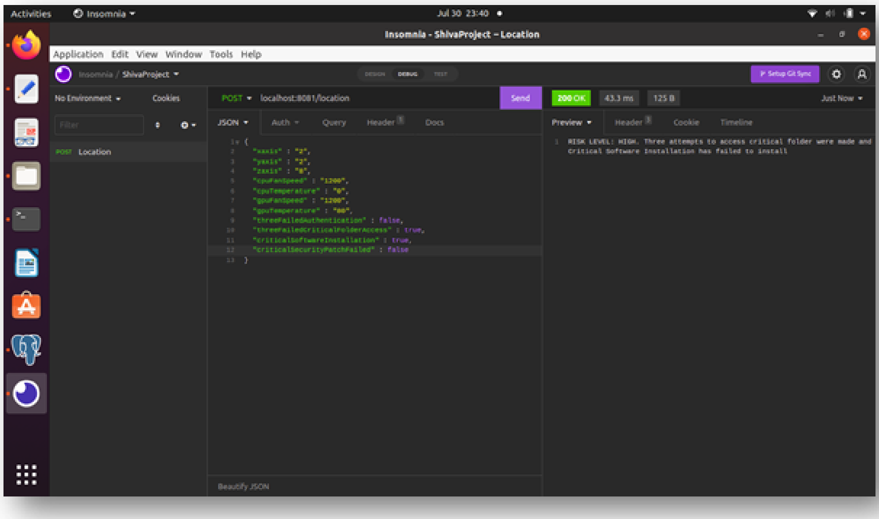
localhost:8081/location

```
{
  "xaxis" : "2",
  "yaxis" : "2",
  "zaxis" : "8",
  "cpuFanSpeed" : "1200",
  "cpuTemperature" : "0",
  "gpuFanSpeed" : "1200",
  "gpuTemperature" : "80",
  "threeFailedAuthentication" : false,
  "threeFailedCriticalFolderAccess" : true,
  "criticalSoftwareInstallation" : false,
  "criticalSecurityPatchFailed" : false
}
```

```
RISK LEVEL: MEDIUM. Three attempts to access critical folder were made.
```

Figure 17: The API gateway showing JSON file feed to the API server and the response; this screenshot shows a Medium Risk Level response

Figure 18 shows the JSON file with two breaches. Hence the response is:'RISK LEVEL: HIGH. Three attempts to access critical folder was made and critical software installation has failed'.

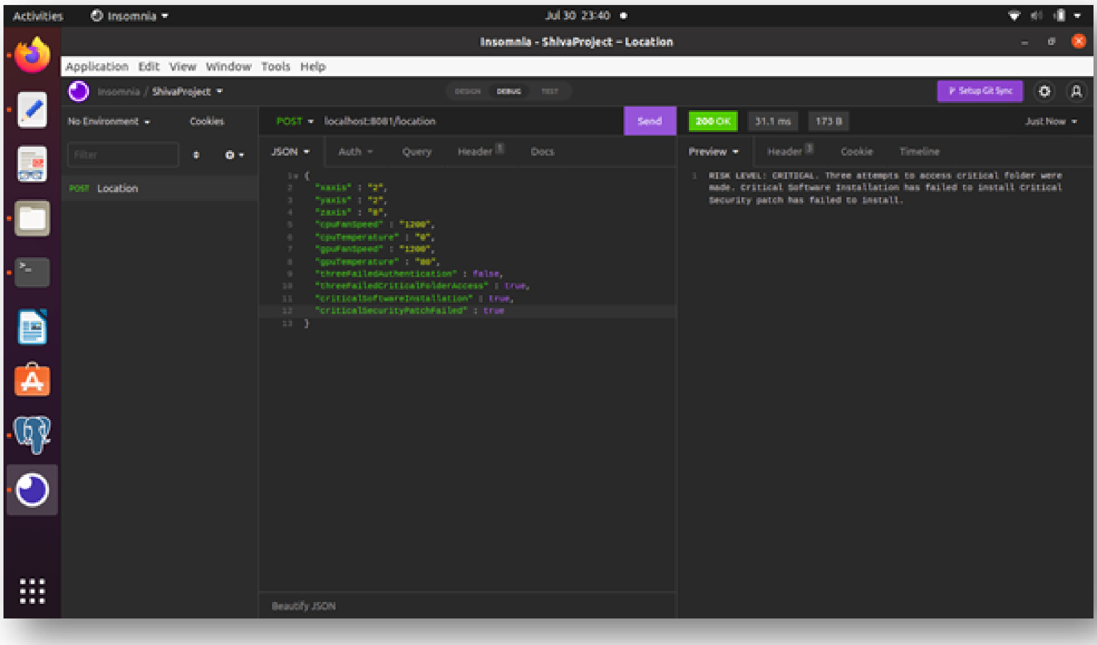


localhost:8081/location

```
{
  "xaxis" : "2",
  "yaxis" : "2",
  "zaxis" : "8",
  "cpuFanSpeed" : "1200",
  "cpuTemperature" : "0",
  "gpuFanSpeed" : "1200",
  "gpuTemperature" : "80",
  "threeFailedAuthentication" : false,
  "threeFailedCriticalFolderAccess" : true,
  "criticalSoftwareInstallation" : true,
  "criticalSecurityPatchFailed" : false
}
```

```
RISK LEVEL: HIGH. Three attempts to access critical folder were made and
critical Software Installation has failed to install
```

Figure 18: The API gateway showing JSON file feed to the API server and the response; this screenshot shows a High Risk Level response

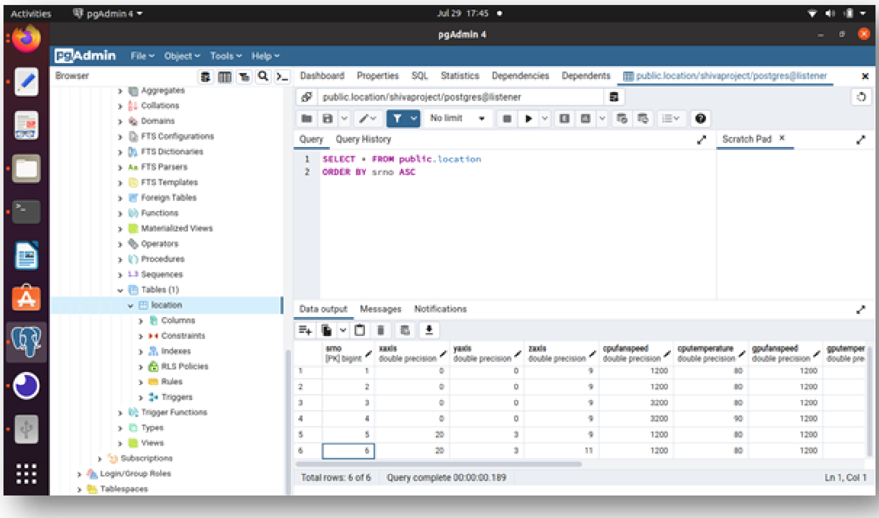


```
POST localhost:8081/location

{
  "xaxis" : "2",
  "yaxis" : "2",
  "zaxis" : "8",
  "cpuFanSpeed" : "1200",
  "cpuTemperature" : "0",
  "gpuFanSpeed" : "1200",
  "gpuTemperature" : "80",
  "threeFailedAuthentication" : false,
  "threeFailedCriticalFolderAccess" : true,
  "criticalSoftwareInstallation" : true,
  "criticalSecurityPatchFailed" : true
}
```

```
200 OK  31.1 ms  173 B

RISK LEVEL: CRITICAL. Three attempts to access critical folder were
made. critical Software Installation has failed to install critical
Security patch has failed to install.
```

Figure 19: The API gateway showing JSON file feed to the API server and the response; this screenshot shows a Critical Risk Level response

Figure 19 shows the JSON file with three breaches. Hence the response is:'RISK LEVEL: CRITICAL. Three attempts to access critical folder was made. Critical software installation has failed. Critical security patch has failed to install'.



public.location/shivaproject/postgres@listener

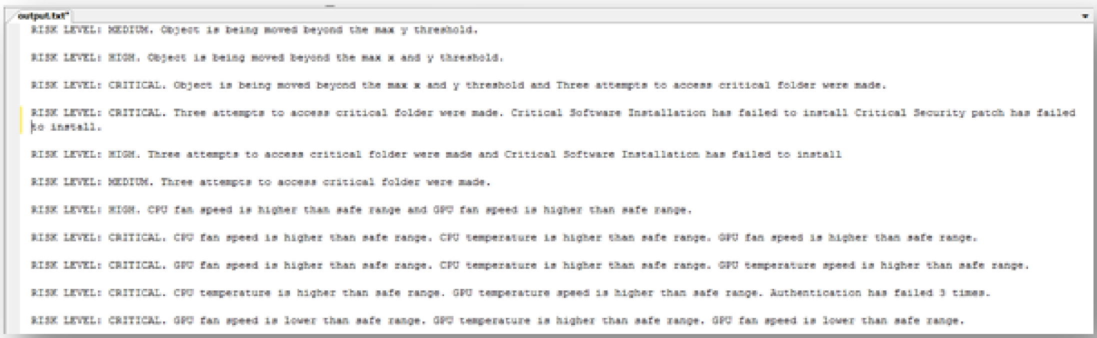
```
SELECT * FROM public.location
ORDER BY srno ASC
```

	srno [PK] bigint	xaxis double precision	yaxis double precision	zaxis double precision	cpufanspeed double precision	cputemperature double precision	gpufanspeed double precision	gputemper double pre
1	1	0	0	9	1200	80	1200	
2	2	0	0	9	1200	80	1200	
3	3	0	0	9	3200	80	1200	
4	4	0	0	9	3200	90	1200	
5	5	20	3	9	1200	80	1200	
6	6	20	3	11	1200	80	1200	

Total rows: 6 of 6   Query complete 00:00:00.189   Ln 1, Col 1

Figure 20: The PGAdmin4 interface connected to the PostgreSQL server showing the data coming from the API gateway getting stored

In this way, multiple cases were tested. All cases are recorded in the database, as shown in Figure 20. Finally, Figure 21 shows that all alert logs are recorded in a text file, which can be used for further analysis. The database can be accessed by going to the path Tables – Location – Columns. A SQL query needs to be executed as shown in the Figure 20.



```
RISK LEVEL: MEDIUM. Object is being moved beyond the max y threshold.

RISK LEVEL: HIGH. Object is being moved beyond the max x and y threshold.

RISK LEVEL: CRITICAL. Object is being moved beyond the max x and y threshold and Three attempts to access critical folder were made.

RISK LEVEL: CRITICAL. Three attempts to access critical folder were made. Critical Software Installation has failed to install Critical Security patch has failed
to install.

RISK LEVEL: HIGH. Three attempts to access critical folder were made and Critical Software Installation has failed to install

RISK LEVEL: MEDIUM. Three attempts to access critical folder were made.

RISK LEVEL: HIGH. CPU fan speed is higher than safe range and GPU fan speed is higher than safe range.

RISK LEVEL: CRITICAL. CPU fan speed is higher than safe range. CPU temperature is higher than safe range. GPU fan speed is higher than safe range.

RISK LEVEL: CRITICAL. GPU fan speed is higher than safe range. CPU temperature is higher than safe range. GPU temperature speed is higher than safe range.

RISK LEVEL: CRITICAL. CPU temperature is higher than safe range. GPU temperature speed is higher than safe range. Authentication has failed 3 times.

RISK LEVEL: CRITICAL. GPU fan speed is lower than safe range. GPU temperature is higher than safe range. GPU fan speed is lower than safe range.
```

Figure 21: The Risk logs generated by the rules engine in the Spring Boot API Server

Multiple test cases were tested for all the four risk levels and multiple parameter breaches. The system worked satisfactorily for all the cases. The data was saved in the database in appropriate columns; the risk level responses made along with description of the breaches were accurate. Further, the alert messages were logged into a text file.

The text file is used in this research to log Alerts. In real world systems, the logs will be stored in a separate activity logging database, which will have its own analytics layer. By collaborating the information in multiple parameters, the exact threat scenario be judged. For example, if the GPU and CPU temperatures of a server have breached the limits and the server itself has been shifted by about 20 metres, the security administrator can judge that the server has been shifted from its air conditioning environment and is now facing heat. This may have happened because the air conditioning malfunctioned and a technician is on site to repair it. For plant continuity, the supervisor may have decided to shift the server outside the air conditioning cubical while the technician is carrying out the repairs but keeps the server running. There can be numerous such scenarios in a plant campus when this hybrid security system can be very useful.

9 Conclusion

The research was conducted for designing a test lab implementation of a hybrid security system for detecting both physical and virtual threats in the same framework. This framework shall be useful for campus deployments of distributed IT assets in several air-conditioned cubicles for running several modules of manufacturing. This system is designed based on the concept of digital fencing and localisation of coordinates within the fence using Wi-Fi access points. The application was developed in Java Spring Boot framework serving as an API server and a hybrid security rules engine. API gateways were configured with a conception that sensory data from the IT assets will be consolidated in JSON files and transmitted to the API server. This document presented detailed instructions on installation of the environment. In addition, the document presented a description of the coding done (although the codes are not included in this document because of their length; they are available in a different document and can be provided on request). Finally, the runtime test reports were also presented. All evidences have been supported by screenshots where sufficient details are justified to be shown in them.

References

Basri, C. and Elkhadimi, A. (2020). A review on indoor localization with internet of things, *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XLIV-4/W3-2020**: 121–128.

Braggaar, R. (2018). Wi-fi network-based indoor localisation: The case of the tu delft campus.

Drake, M. (2020). How to install and use postgresql on ubuntu 20.04, `https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-20-04`.

Gomes, P., Magaia, N. and Neves, N. (2020). Industrial and artificial internet of things with augmented reality, *Convergence of Artificial Intelligence and the Internet of Things*, Springer, pp. 323–346.

Happa, J., Glencross, M. and Steed, A. (2019). Cyber security threats and challenges in collaborative mixed-reality, *Frontiers in ICT* **6**.

Oliver Gierke, J. B. (2022). Spring data jpa - reference documentation, `https://docs.spring.io/spring-data/jpa/docs/current/reference/html/`.

Palmarini, R., Erkoyuncu, J. A., Roy, R. and Torabmostaedi, H. (2018). A systematic review of augmented reality applications in maintenance, *Robotics and Computer-Integrated Manufacturing* **49**: 215–228.

Plósz, S., Schmittner, C. and Varga, P. (2017). Combining safety and security analysis for industrial collaborative automation systems, *International Conference on Computer Safety, Reliability, and Security*, Springer, pp. 187–198.

Webb, P., Syer, D., Long, J., Nicoll, S., Winch, R., Wilkinson, A., Overdijk, M., Dupuis, C., Deleuze, S., Simons, M. et al. (2020). Spring boot reference documentation, *Retrieved June* **22**.