Distributed Systems (SE3020)

3rd Year, 1st Semester

Assignment 2

Fire Alarm System Report

Submitted By

IT17182256 - Albert A. K

IT17163750 - Yasathilaka W. S. L

IT17163132 – Wimaladasa G. W. P. N

IT17169554 – Dissanayake A. G. I. U

01.05.2020

1. Introduction

This report describes a system designed for the fire alarm system which consist, a Rest API, RMI server and a desktop client application, a web application to monitor the sensors and a web application to simulate the behaviour of a fire alarm sensor.

2. Architectural diagram

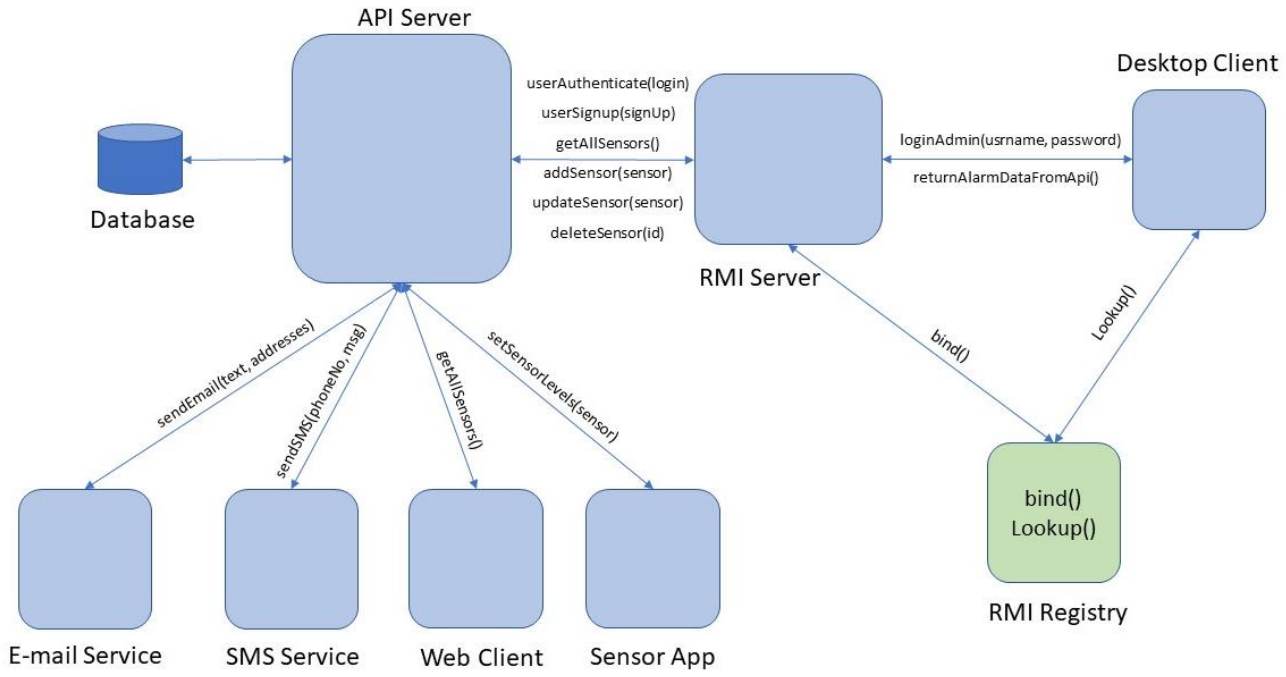

Figure 1 : Architecture Diagram

3. Sequence Diagram

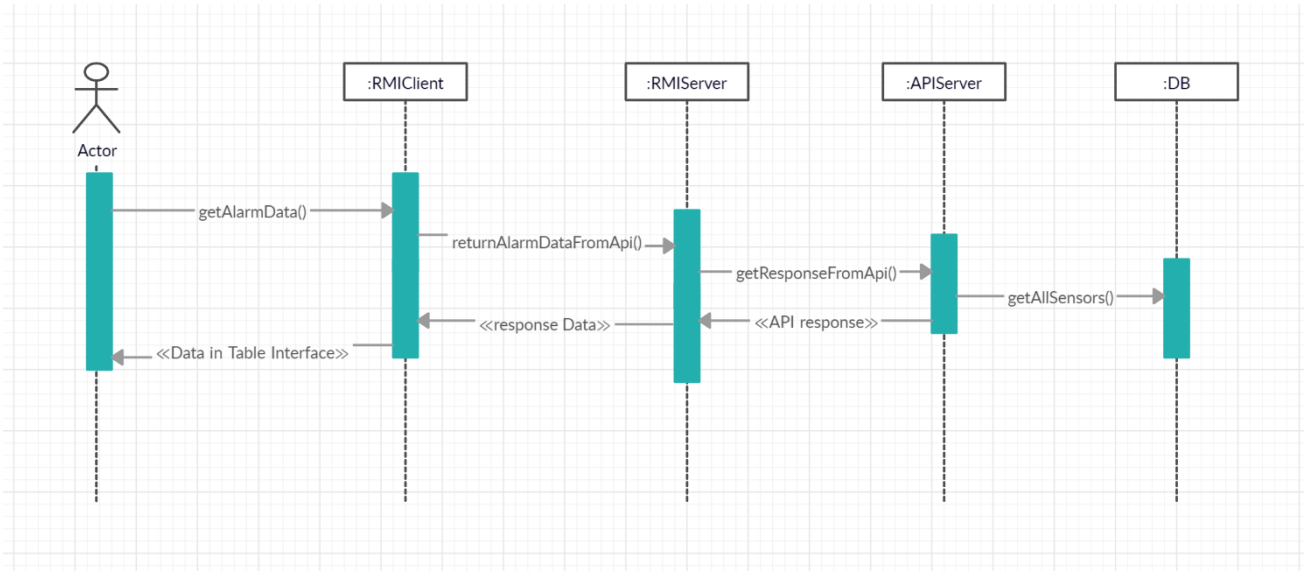

Figure 2 : Sequence Diagram

4. Components

4.1 Rest API

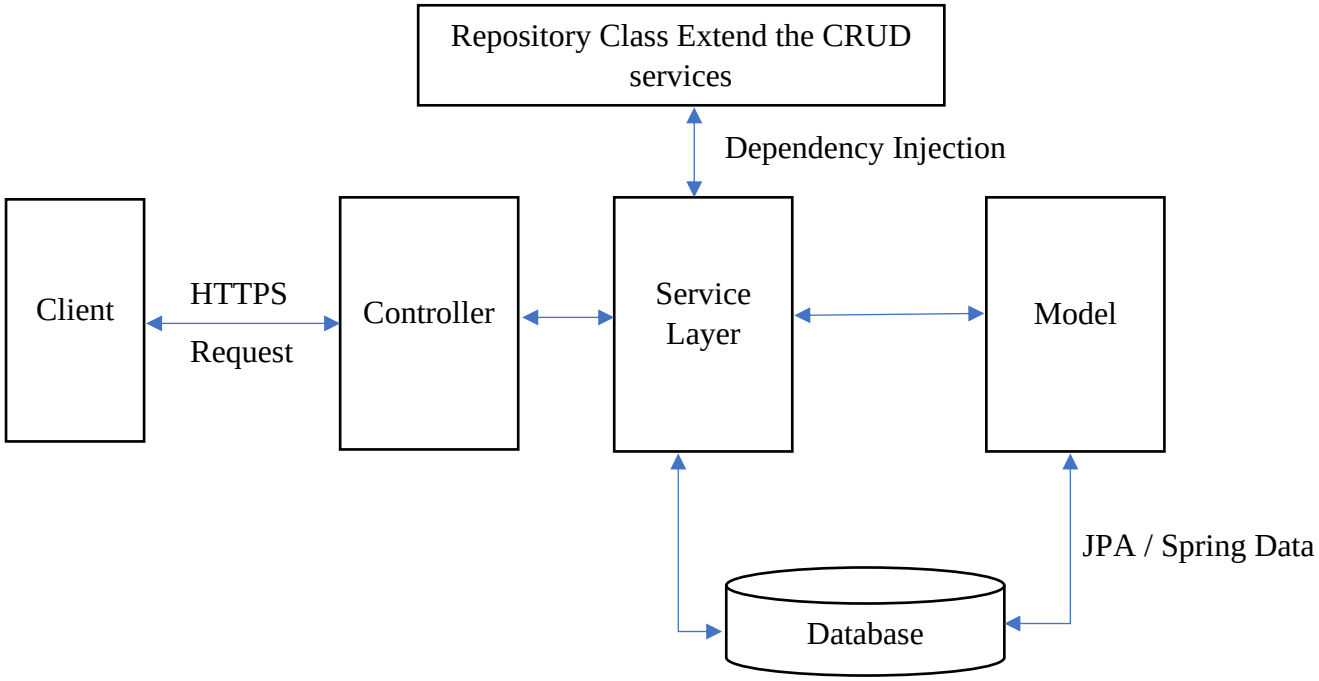

Figure 3: Spring Boot Rest API Architecture Diagram

Rest API for the fire alarm sensor is created using Spring boot framework. And MySQL is used as the database service. Rest API is connected to this MySQL database. There are two Entity classes (Models) created to map with the database tables. “User” model class for the “user” table and “Sensor” model class for the “alarm_sensor” table in the database.

User class is defined with the following private variables to map with the database table columns in user table. (id, username, password, role, phone, email, active)

Sensor class is defined with the following private variables to map with the database table columns in alarm_sensor table. (sensorId, floorNo, roomno, smoke, co2, active)

There are two interfaces as “UserRepository” and “SensorRepository” which extends the JpaRepository interface in Spring. These are used to do the insert, update, delete and retrieve from the database. UserRepository has the following methods,

```
Optional<User> findByUsername(String username);
Boolean existsByUsername(String username);
Boolean existsById(int id);
```

SensorRepository has the following additional method which is used to update only the sensor levels (CO_2 and Smoke).

```
@Modifying
@Transactional
@Query("UPDATE Sensor s SET s.co2 = ?1 , s.smoke = ?2 WHERE s.sensorId = ?3")
int setSensorLevels(int co2, int smoke, int id);
```

There are two service classes implemented to use methods from the repository classes. Controller classes access these methods from the service class.

There are two controller classes as “UserController and “SensorController” which has the url mapping to access the methods in the service classes.

Example method from SensorController

```
@PostMapping("/api/admin/addSensor")
public Sensor addSensor(@RequestBody Sensor sensor) {
   return sensorService.saveSensor(sensor);
}
```

It uses the sensorService object created using the SensorService class

```
@Autowired
private SensorService sensorService;
```

There is a controller “AuthController” which map the sign in and sign up URLs

```
@PostMapping("/signin")
public ResponseEntity<?> userAuthenticate(@Valid @RequestBody Login login) {
    Authentication auth = authManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                  login.getUsername(),
                  login.getPassword()
            )
    );
    SecurityContextHolder.getContext().setAuthentication(auth);
    String token = jwt.generateJwtToken(auth);
    return ResponseEntity.ok(new JwtResponse(token));
}
@PostMapping("/signup")
public ResponseEntity<String> userSignup(@Valid @RequestBody Register reg) {
    if(userRepo.existsByUsername(reg.getUsername())) {
        return new ResponseEntity<String>("Failed -> Username Exist!",
                HttpStatus.BAD_REQUEST);
    }

    User u = new User(reg.getUsername(),
           encoder.encode(reg.getPassword()), reg.getRole(), reg.getPhone(),
           reg.getEmail(), reg.isActive());
    userRepo.save(u);
    return ResponseEntity.ok().body("Successfully Registered The User!");
}
```

Security

JWT authentication is used in this project for the security implementation. When a user sign in, a Json Web Token is returned from the system. This token is used for authenticating the user across all functions that require a registered user to access. This token should be sent in the header as a Bearer token.

In the "WebSecurityConfig" class, http url patterns are authenticated as follows,

```
      .antMatchers("/api/auth/signin").permitAll()
      .antMatchers("/api/auth/signup").authenticated()
      .antMatchers("/api/admin/**").authenticated()
      .anyRequest().permitAll()
```

An Error response is returned if someone tries to access the url without the token. All the url patterns with "/api/admin/" and "/api/auth/signup" are authenticated. Url patterns "/api/auth/signin" and any other url patterns are not authenticated. Ex: "api/getSensors" is not authenticated because all the sensor details should be able to view by any user without login into the system

Email Service

```
public void sendEmail(String text, String addresses) {

       SimpleMailMessage email = new SimpleMailMessage();
       email.setTo(addresses);
       email.setSubject("Alarm Alert Message (Critical)");
       email.setText(text);
       javaMailSender.send(email);

   }
```

This is the method used to send alert email to the responsible people. Here set the mail addresses using the setTo method. Write the subject using the setSubject. After email address, message, subject and text. After that send the email using the send method in javaMailSender.

Then create an email controller and import userService and EmailService classes. We import the UserService because we need to get the email address of the user. And the EmailService class to send email to the recipients.

```
@Autowired

private UserService userService;

@Autowired

private EmailService emailService;
```

Then create the objects for UserService and EmailService.

```
public void sendEmail(@RequestBody String message) {
   List<User> userList = userService.getAllUsers();
   emails = "";
   try {
      userList.forEach((u) -> {
         if (u.getEmail() != null) {
            if (!u.getEmail().isEmpty()) {
               emailService.sendEmail(message, u.getEmail().toString());
            }
         }
      });
}
}
```

Then using the User and get the user list that is responsible for the fire sensors. And send email to all the users in the list.

SMS Service

A Twilio account is needed for SMS service to work. (A trial version of twilio is used in this project and it's only able to send sms for a verified phone no in the twilio account. Paid account is needed to send sms to any number)

```
public interface SmsService {

    public Message sendMessage(String phoneNo, String text);
}
```

Here SMS service interface is created with unimplemented method called SendSMS.

```
public class SmsServiceTwilio implements SmsService{

   @Value("${TWILIO_SID}")
    public String SID;
   @Value("${TWILIO_AUTH_TOKEN}")
    public String TOKEN;
   @Value("${PHONE_NO}")
   public String twilioPhoneNo;

    public Message sendMessage(String phoneNo, String msg) {
        Twilio.init(SID, TOKEN);
        Message text = Message.creator(
                new PhoneNumber(phoneNo),//The phone number you are sending text to
                new PhoneNumber(twilioPhoneNo),//The Twilio phone number
                msg)
                .create();
        return text;
    }
}
```

Then created a class called SmsServiceTwilio that implements the SmsService Interface. Phone number and the message must be passed to the sendSMS method.

A controller class called “SmsController” is created to map the post method url to sendSMS method.

```
@PostMapping("api/sendSMS")
public Message sendSMS(@RequestBody String message) {

   List<User> userList = userService.getAllUsers();
   try {
      userList.forEach((u) -> {
         if (u.getPhone() != null) {
            if (!u.getPhone().isEmpty()) {
               msg = smsService.sendSMS(u.getPhone(), message);
            }
         }
      });
   } catch (Exception e) {
      e.printStackTrace();
   }
   return msg;
}
```

Here objects for smsService and UserService classes are created to access the mothods in them. This method retrieves all the users from the database and send the message to each user’s phone number.

4.2 Web Client Application

A react web application is implemented to view the behaviour of the sensors. Following is the code snippet to get all the sensor details in the dashboard.

```
const getSensorDetails = () => {
    return fetch(`${API}/getSensors`, {
        method: "GET"
    })
        .then(response => {
            return response.json();
        })
        .catch(err => console.log(err));
};
```

The Main page UI shows each alarm’s floor no., room no., sensorId, status, smoke level and CO2 level. If either CO2 level or smoke level is greater than 5 it will indicate in red and as well as floor no., and room no. will indicate in red. If the alarm sensor is inactive, it is shown as greyed colour as it is disabled.

Below is the code snippet for updating sensor statuses every 40 seconds. Which call the loadAlarms() function every 40 seconds.

```
    useEffect(() => {
        const interval = setInterval(function(){
            loadAlarms();
        }, 40000);

        return () => clearInterval(interval);
    });
```

loadAlarms() function will call getSensorDetails() function which is mentioned in the beginning.

```
    const loadAlarms = () => {
        getSensorDetails().then(data => {
            if (data.error) {
                setError(data.error)
            } else {
                setAlarms(data)
            }
        });
    }
```

4.3 Desktop Client Application

Service.java

Here defined the Service Interface which is implemented later with Server.java.

```
public StringBuffer returnAlarmDataFromApi() throws RemoteException;
public String[] loginAdmin(String userName, String password) throws RemoteException;
public String[] addAlarmSensor(String jsonDetails, String token) throws RemoteException;
public String[] updateAlarmSensor(String jsonDetails, String token) throws RemoteException;
public String[] deleteAlarmSensor(String jsonDetails, String token) throws RemoteException;
public void getResponseFromApi()throws RemoteException;
```

Server.java

The RMI Client updates every 30 seconds meanwhile the RMI Server updates every 15 seconds and that process is handling by this code segment.

```
Timer timer = new Timer(); // Timer initialized in order to keep the program updated for every 15 seconds.
timer.schedule(new TimerTask() {
  @Override
  public void run() {
    try {   }      } catch (Exception ex) {         ex.printStackTrace();      }   }}, 0, 15000); // Timer set to 15 seconds
```

The new Server object creation and binding the server to the RMI Registry. (Main method)

```
Server svr = new Server(); // Creating the server.
Registry registry = LocateRegistry.getRegistry();
registry.bind("AlarmService", svr); // Bind service to the Registry.
```

Below code segment was used to acquire the API Server's response according to a request made by the RMI Server.

```
URL obj = new URL(url); // URL initialization.
HttpURLConnection con = (HttpURLConnection) obj.openConnection();
con.setRequestMethod("GET"); // sets the Method
con.setRequestProperty("User-Agent", "Mozilla/5.0");
BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
```

The POST request of the server.java to get the user logged in lays here.

```
Unirest.setTimeouts(0, 0);
HttpResponse<String> response = Unirest.post("http://localhost:8080/api/auth/signin") /
      .header("Content-Type", "application/json").body(jsonString) // Passing the JSON data to the API
      .asString();
```

Takes the result of the POST request and direct the user where he belongs.

```
if (response.getStatus() == 200) { // Investigate the response status.
   Server.TOKEN_BEARER = myResponse.getString("accessToken"); // Retreiving the Admin Token.
   String[] values = new String[2]; // Creation of String array to be sent to the RMI Client.
   values[0] = "Authorized!";
   values[1] = myResponse.getString("accessToken");
   return values;
} else {    String[] values = new String[2];
   values[0] = "Unauthorized!";
   return values; // Returns "Unauthorized!".}
```

This process also provides us with the Admin token if he is authorized. The token here will be used to access the register and update function later on.

This code segment below was used to register a new Alarm Sensor.

```
Unirest.setTimeouts(0, 0); // Time out from UNIREST.
HttpResponse<String> responseGet = Unirest.post("http://localhost:8080/api/admin/addSensor")
.header("Authorization", "Bearer " + token).header("Content-Type", "application/json")
      .body(jsonDetails)       .asString();
```

As well as this was used to update a particular Alarm Sensor.

```
Unirest.setTimeouts(0, 0);
HttpResponse<String> response = Unirest.put("http://localhost:8080/api/admin/updateSensor") // Perform the
 // PUT Action.
 .header("Authorization", "Bearer " + token) // Passing the Admin Token to get the permission to the update
process.  .header("Content-Type", "application/json").body(jsonDetails) // Pass the created JSON Data to the
      // PUT request.       .asString();
```

Here we created an ArrayList to get the detailed Alarm list and use a JTable to show the details.

```
ArrayList<Alarm> arrayList = new ArrayList<>();
   JSONArray jsonArray = new JSONArray(json); // Initialized JsonArray to get the JSONArray response.
   for (int count = 0; count < jsonArray.length(); count++) {
      Alarm alarmObject = new Alarm(); // Alarm Object initialzation.
      JSONObject jsonObject = jsonArray.getJSONObject(count);
      alarmObject.setAlarmId(jsonObject.getInt("sensorId")); // Storing the data in to an object.
      alarmObject.setFloorNumber(jsonObject.getInt("floorNo"));
```

```
    alarmObject.setRoomNumber(jsonObject.getInt("roomNo"));
    alarmObject.setSmokeLevel(jsonObject.getInt("smoke"));
    alarmObject.setCo2Level(jsonObject.getInt("co2"));
    alarmObject.setStatus(jsonObject.getInt("active"));
    // Taking data from particular Key.
    arrayList.add(alarmObject); // Object added to the ArrayList
```

Client.java

First, client is looking for the Server. And the client is updating every 30 seconds with the RMIServer.

```
service = (Service) Naming.lookup("//localhost/AlarmService");
```

This is how the RMIClient calls RMIServer to acquire the Alarm Sensor Details.

```
StringBuffer response = service.returnAlarmDataFromApi();
```

The RMIClient using the Login function to gather the Admin functions, through RMIServer.

```
response = service.loginAdmin(userName, password);
        if (response[0].equals("Authorized!")) {
           this.setVisible(false);
           timer.cancel();
           String srgs[] = new String[2];
           srgs[0] = response[1];
```

If the user is Authorized the Admin console will be popped up.

```
Home.main(srgs);
```

Security.

```
System.setProperty("java.security.policy", "file:allowall.policy");
```

And there's an Alarm class in order to create a JSONObject to pass to the POST request.

4.4 Sensor Simulator Web Application

Implemented client application to simulate the behaviour of a fire sensor. First it will load onto a dashboard and then user will be able to open each sensor in different tabs. When a user open a sensor it will automatically send the fire alarm sensor status to the REST API every 10 seconds with the values user will be given. User can manually update the sensors without waiting other he can change the sensor details and the system will do the rest. Below are some code snippets used to develop this system.

```
    useEffect(() => {
        const interval = setInterval(() => {

            setValues({...values, error: "", success: ""});

            if (smoke > 10 || smoke < 0) {
                setValues({...values, error: "Smoke level must be between 0 and 10", success: ""})
;
            } else if (co2 > 10 || co2 < 0) {
                setValues({...values, error: "CO2 level must be between 0 and 10", success: ""});
            } else {
                updateSensor(values).then(data => {
                    if (data.error) {
                        setValues({...values, error: data.error, success: false});
                    } else {
                        setValues({
                            ...values,
                            sensorId: values.sensorId,
                            smoke: values.smoke,
                            co2: values.co2,
                            error: false,
                        });
                        setValues({...values, error: false, success: "Sensor Values Automatically
Updated!"});

                    }
                });
            }
        }, 10000);
        return () => clearInterval(interval);
    });
    const updateSensor = () => {
        console.log(values);
        return fetch(`${API}/setSensorLevels`, {
            method: "PUT",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({
                sensorId: sensorId,
                smoke: smoke,
                co2: co2
            })
        }).then(response => {
            return response.json();
        })
            .catch(err => console.log(err));  };
```

5.0 Appendices

Appendix A

```
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/auth")
public class AuthController{

    @Autowired
    AuthenticationManager authManager;

    @Autowired
    UserRepository userRepo;

    @Autowired
    PasswordEncoder encoder;

    @Autowired
    JwtProvider jwt;

    @PostMapping("/signin")
    public ResponseEntity<?> userAuthenticate(@Valid @RequestBody Login login) {
        Authentication auth = authManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        login.getUsername(),
                        login.getPassword()
                )
        );
        SecurityContextHolder.getContext().setAuthentication(auth);
        String token = jwt.generateJwtToken(auth);
        return ResponseEntity.ok(new JwtResponse(token));
    }

    @PostMapping("/signup")
    public ResponseEntity<String> userSignup(@Valid @RequestBody Register reg) {
        if(userRepo.existsByUsername(reg.getUsername())) {
            return new ResponseEntity<String>("Failed -> Username Exist!",
                    HttpStatus.BAD_REQUEST);
        }

        User u = new User(reg.getUsername(),
                encoder.encode(reg.getPassword()), reg.getRole(), reg.getPhone(),
                reg.getEmail(), reg.isActive());
        userRepo.save(u);
        return ResponseEntity.ok().body("Successfully Registered The User!");
    }
}
```

Appendix B

```
public class SensorController {

	@Autowired
	private SensorService sensorService;

	@PostMapping("/api/admin/addSensor")
	public Sensor addSensor(@RequestBody Sensor sensor) {
		return sensorService.saveSensor(sensor);
	}

	@GetMapping("/api/getSensors")
	public List<Sensor> getAllSensors() {

		return sensorService.getAllSensors();
	}

	@GetMapping("/api/admin/getSensor/{id}")
	public Sensor getSensorById(@PathVariable int id) {

		return sensorService.getSensorById(id);
	}

	@PutMapping("/api/admin/updateSensor")
	public Sensor updateSensor(@RequestBody Sensor sensor) {

		return sensorService.updateSensor(sensor);
	}

	@DeleteMapping("/api/admin/deleteSensor/{id}")
	public String deleteSensor(@PathVariable int id) {

		return sensorService.deleteSensor(id);
	}

	@PutMapping("/api/setSensorLevels")
	public int setSensorLevels(@RequestBody Sensor sensor) {

		return sensorService.setSensorLevels(sensor);
	}
}
```

Appendix C

```
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
public class UserController {

	@Autowired
	private UserService userService;

	@GetMapping("/api/admin/getAllUsers")
	public List<User> getAllUsers() {

		return userService.getAllUsers();
	}

	@GetMapping("/api/admin/getUser/{id}")
	public User getUserById(@PathVariable int id) {

		return userService.getUserById(id);
	}

	@PutMapping("/api/admin/updateUser")
	public User updateUser(@RequestBody User user) {

		return userService.updateUser(user);
	}

	@DeleteMapping("/api/admin/deleteUser/{id}")
	public String deleteUser(@PathVariable int id) {

		return userService.deleteUser(id);
	}
}
```

Appendix D

```
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
public class SmsController {

	@Autowired
	private SmsService smsService;

	@Autowired
	private UserService userService;

	private Message msg;

	@PostMapping("api/sendSMS")
	public Message sendSMS(@RequestBody String message) {

		List<User> userList = userService.getAllUsers();

		try {
			userList.forEach((u) -> {

				if (u.getPhone() != null) {
					if (!u.getPhone().isEmpty()) {
						msg = smsService.sendSMS(u.getPhone(), message);
					}
				}
			});

		} catch (Exception e) {
			e.printStackTrace();
		}

		return msg;
	}
}
```

Appendix E

```
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
public class EmailController {

	@Autowired
	private UserService userService;

	@Autowired
	private EmailService emailService;

	private String emails;

	@PostMapping("api/sendEmail")
	public void sendEmail(@RequestBody String message) {
		List<User> userList = userService.getAllUsers();
		emails = "";
		try {
			userList.forEach((u) -> {

				if (u.getEmail() != null) {
					if (!u.getEmail().isEmpty()) {
						emailService.sendEmail(message,
						u.getEmail().toString());
					}
				}

			});
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Appendix F

```
public class SensorService {

	@Autowired
	private SensorRepository sensorRepository;

	public Sensor saveSensor(Sensor sensor) {

		return sensorRepository.save(sensor);
	}

	public List<Sensor> getAllSensors(){

		return sensorRepository.findAll();
	}

	public Sensor getSensorById(int id) {

		return sensorRepository.findById(id).orElse(null);
	}

	public String deleteSensor(int id) {

		sensorRepository.deleteById(id);

		return "sensor removed "+id;
	}

	public Sensor updateSensor(Sensor sensor) {

		Sensor currentSensor =
sensorRepository.findById(sensor.getSensorId()).orElse(null);
		currentSensor.setRoomNo(sensor.getRoomNo());
		currentSensor.setFloorNo(sensor.getFloorNo());
		currentSensor.setSmoke(sensor.getSmoke());
		currentSensor.setCo2(sensor.getCo2());
		currentSensor.setActive(sensor.getActive());

		return sensorRepository.save(currentSensor);

	}

	public int setSensorLevels(Sensor sensor) {

		return sensorRepository.setSensorLevels(sensor.getCo2(), sensor.getSmoke(),
sensor.getSensorId());
	}

}
```

Appendix G

```
@Service
public class UserService {

	@Autowired
	private UserRepository userRepo;

	@Autowired
	PasswordEncoder encoder;

	public List<User> getAllUsers() {

		return userRepo.findAll();
	}

	public User getUserById(int id) {

		return userRepo.findById(id).orElse(null);
	}

	public String deleteUser(int id) {

		userRepo.deleteById(id);

		return "user removed " + id;
	}

	public User updateUser(User user) {

		User currentUser = userRepo.findById(user.getId()).orElse(null);

		currentUser.setUsername(user.getUsername());
		currentUser.setPassword(encoder.encode(user.getPassword()));
		currentUser.setPhone(user.getPhone());
		currentUser.setEmail(user.getEmail());
		currentUser.setRole(user.getRole());
		currentUser.setActive(user.isActive());

		return userRepo.save(currentUser);

	}
}
```

Appendix H

```
@Override
protected void configure(HttpSecurity http) throws Exception {

	http.cors().and().csrf().disable().
	authorizeRequests()
	.antMatchers("/api/auth/signin").permitAll()
	.antMatchers("/api/auth/signup").authenticated()
	.antMatchers("/api/admin/**").authenticated()
	.anyRequest().permitAll()
	.and()
	.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
	.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

	http.addFilterBefore(authenticationJwtTokenFilter(),
	UsernamePasswordAuthenticationFilter.class);
}
```

Appendix I

```
@Repository
public interface SensorRepository extends JpaRepository<Sensor, Integer> {

	@Modifying
	@Transactional
	@Query("UPDATE Sensor s SET s.co2 = ?1 , s.smoke = ?2 WHERE s.sensorId = ?3")
	int setSensorLevels(int co2, int smoke, int id);
}
```

Appendix J

```
public interface UserRepository extends JpaRepository<User, Integer> {

    Optional<User> findByUsername(String username);
    Boolean existsByUsername(String username);
    Boolean existsById(int id);

}
```

Appendix K

```
public class EmailService {

	@Autowired
    private JavaMailSender javaMailSender;

	public void sendEmail(String text, String addresses) {

	  SimpleMailMessage email = new SimpleMailMessage();
	  email.setTo(addresses);

	  email.setSubject("Alarm Alert Message (Critical)");
	  email.setText(text);

	  javaMailSender.send(email);

    }
}
```

Appendix L

```
public class SmsServiceTwilio implements SmsService{

	@Value("${TWILIO_SID}")
    public String SID;

	@Value("${TWILIO_AUTH_TOKEN}")
    public String TOKEN;

	@Value("${PHONE_NO}")
	public String twilioPhoneNo;

    public Message sendMessage(String phoneNo, String msg) {
        Twilio.init(SID, TOKEN);
        Message text = Message.creator(
                new PhoneNumber(phoneNo),//The phone number you are sending text to
                new PhoneNumber(twilioPhoneNo),//The Twilio phone number
                msg)
                .create();

        return text;
    }

}
```

Appendix M

```
@Entity
@Table(name = "alarm_sensor")
public class Sensor {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "sensor_id")
	private int sensorId;
	@Column(name = "floor_no")
	private int floorNo;
	@Column(name = "room_no")
	private int roomNo;
	@Column(name = "smoke", columnDefinition = "integer default 0")
	private int smoke;
	@Column(name = "co2", columnDefinition = "integer default 0")
	private int co2;
	@Column(name = "active")
	private int active;

}
```

Appendix N

```
@Entity
@Table(name = "user")
public class User {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private int id;
	@NotBlank
	@Column(name = "username")
	private String username;
	@NotBlank
	@Column(name = "password")
	private String password;
	@NotBlank
	@Column(name = "role")
	private String role;
	@Column(name = "phone")
	private String phone;
	@Column(name = "email")
	private String email;
	@Column(name = "active")
	private boolean active;

	public User(String username, String password, String role, String phone, String
	email, boolean active) {
		this.username = username;
		this.password = password;
		this.role = role;
		this.phone = phone;
		this.email = email;
		this.active = active;
	}
}
```

Appendix O

AlarmDashboard.js

```
const getSensorDetails = () => {
    return fetch(`${API}/getSensors`, {
        method: "GET"
    })
        .then(response => {
            return response.json();
        })
        .catch(err => console.log(err));
};
const AlarmDashboard = () => {
    const [alarms, setAlarms] = useState([]);
    const [error, setError] = useState(false);

    const loadAlarms = () => {
        getSensorDetails().then(data => {
            if (data.error) {
                setError(data.error)
            } else {
                setAlarms(data)
            }
        });
    }
    useEffect(() => {
        const interval = setInterval(function(){
            loadAlarms();
        }, 40000);

        return () => clearInterval(interval);
    });

    useEffect(() => {
        loadAlarms()
    }, [])
    const showError = () => (
        <div className="alert alert-danger" style={{ display: error ? '' : 'none' }}>
            {error}
        </div>
    )
    return (
        <div className="container-fluid">
            <div className="jumbotron">
                <h2>Alarm Dashboard</h2>
                <p className="lead">Monitor all alarm statuses.</p>
            </div>
            {showError()}
            <div className="row">
                {alarms.map((alarm, i) => (
                    <Card key={i} alarm={alarm} />
                ))}
            </div>

        </div>
    );
};
```

Appendix P

Card.js

```
const isDanger = (val) => {
    if (val > 5) {
        return { color: "#ff0000" }
    }
};

const isStatus = (val) => {
    if (val) {
        return "Active"
    }
    else {
        return "Inactive"
    }
};

const isInactive = (val) => {
    if (!val) {
        return { color: "#C6C6C6" }
    }
};

const Card = ({ alarm }) => {
    return (
        <div className="col-4 mb-3">
            <div className="card" style={isInactive(alarm.active)}>
                <div className="card-
header" style={isDanger(alarm.co2) || isDanger(alarm.smoke)}>Floor No: {alarm.floorNo} -
 Room No: {alarm.roomNo}</div>
                <div className="card-body">
                    <p>Sensor Id: {alarm.sensorId}</p>
                    <p>Status: {isStatus(alarm.active)}</p>
                    <p style={isDanger(alarm.smoke)}>Smoke Level: {alarm.smoke}</p>
                    <p style={isDanger(alarm.co2)}>CO2 Level: {alarm.co2}</p>
                </div>
            </div>
        </div>
    )
};
```

Appendix Q

AlarmDashboard.js

```
const getSensorDetails = () => {
    return fetch(`${API}/getSensors`, {
        method: "GET"
    })
        .then(response => {
            return response.json();
        })
        .catch(err => console.log(err));
};
const AlarmDashboard = () => {

    const [alarms, setAlarms] = useState([]);
    const [error, setError] = useState(false);
    const loadAlarms = () => {
        getSensorDetails().then(data => {
            if (data.error) {
                setError(data.error)
            } else {
                setAlarms(data);
            }
        });
    }

    useEffect(() => {
        const interval = setInterval(function(){
            loadAlarms();
        }, 40000);

        return () => clearInterval(interval);
    });

    useEffect(() => {
        loadAlarms();
    }, []);
    const showError = () => (
        <div className="alert alert-danger" style={{ display: error ? '' : 'none' }}>
            {error}
        </div>
    );
    return (
        <div className="container-fluid">
            <div className="jumbotron">
                <h2>Alarm Dashboard</h2>
                <p className="lead">Monitor all alarm statuses.</p>
            </div>
            {showError()}
            <div className="row">
                {alarms.map((alarm, i) => (
                    <Card key={i} alarm={alarm} />
                ))}
            </div>
        </div>
    );
};
```

Appendix R

Card.js

```
const isDanger = (val) => {
    if (val > 5) {
        return { color: "#ff0000" }
    }
};

const isStatus = (val) => {
    if (val) {
        return "Active"
    }
    else {
        return "Inactive"
    }
};

const isInactive = (val) => {
    if (!val) {
        return { color: "#C6C6C6" }
    }
};

const Card = ({ alarm }) => {
    return (
        <div className="col-4 mb-3">
            <div className="card" style={isInactive(alarm.active)}>
                <div className="card-
header" style={isDanger(alarm.co2) || isDanger(alarm.smoke)}>Floor No: {alarm.floorNo} -
 Room No: {alarm.roomNo}</div>
                <div className="card-body">
                    <p>Sensor Id: {alarm.sensorId}</p>
                    <p>Status: {isStatus(alarm.active)}</p>
                    <p style={isDanger(alarm.smoke)}>Smoke Level: {alarm.smoke}</p>
                    <p style={isDanger(alarm.co2)}>CO2 Level: {alarm.co2}</p>
                    <Link to={`/admin/updateSensor/${alarm.sensorId}/${alarm.smoke}/${al
arm.co2}`} target="_blank">
                        <button className="btn btn-outline-warning mt-2 mb-
2" disabled={!alarm.active}>
                            Update Alarm Status
                        </button>
                    </Link>
                </div>
            </div>
        </div>
    )
};
```

Appendix S

UpdateAlarm.js

```
const UpdateAlarm = ({match}) => {
    const [values, setValues] = useState({
        sensorId: "",
        smoke: "",
        co2: "",
        error: false,
        success: false
    });

    const {
        sensorId,
        smoke,
        co2,
        error,
        success
    } = values;


    useEffect(() => {
        const interval = setInterval(() => {

            setValues({...values, error: "", success: ""});

            if (smoke > 10 || smoke < 0) {
                setValues({...values, error: "Smoke level must be between 0 and 10", suc
cess: ""});
            } else if (co2 > 10 || co2 < 0) {
                setValues({...values, error: "CO2 level must be between 0 and 10", succe
ss: ""});
            } else {
                updateSensor(values).then(data => {
                    if (data.error) {
                        setValues({...values, error: data.error, success: false});
                    } else {
                        setValues({
                            ...values,
                            sensorId: values.sensorId,
                            smoke: values.smoke,
                            co2: values.co2,
                            error: false,
                        });
                        setValues({...values, error: false, success: "Sensor Values Auto
matically Updated!"});

                    }
                });
            }
        }, 10000);
        return () => clearInterval(interval);
    });

    const updateSensor = () => {
        console.log(values);
        return fetch(`${API}/setSensorLevels`, {
            method: "PUT",
```

```
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({
                sensorId: sensorId,
                smoke: smoke,
                co2: co2
            })
        }).then(response => {
            return response.json();
        })
            .catch(err => console.log(err));
    };

    const init = (sensorId, smoke, co2) => {
        setValues({
            ...values,
            sensorId: sensorId,
            smoke: smoke,
            co2: co2,
        })
    };

    useEffect(() => {
        console.log("here");
        init(match.params.sensorId, match.params.smoke, match.params.co2);
    }, []);

    const handleChange = (event) => {
        const value = event.target.value;
        const name = event.target.name;
        setValues({...values, [name]: value});
        console.log(values);
    };


    const clickSubmit = event => {
        event.preventDefault();
        setValues({...values, error: ""});
        setValues({...values, error: "", success: ""});

        if (smoke > 10 || smoke < 0) {
            setValues({...values, error: "Smoke level must be between 0 and 10"});
        } else if (co2 > 10 || co2 < 0) {
            setValues({...values, error: "CO2 level must be between 0 and 10"});
        } else {
            updateSensor(values).then(data => {
                if (data.error) {
                    setValues({...values, error: data.error, success: false});
                } else {
                    setValues({
                        ...values,
                        sensorId: values.sensorId,
                        smoke: values.smoke,
                        co2: values.co2,
                        error: false,
                    });
                    setValues({...values, error: false, success: "Sensor Values Updated!
"});
```

```
            }
        });
    }
};


const newPostForm = (values) => (
    <form className="mb-3" onSubmit={clickSubmit}>

        <div className="form-group">
            <label className="text-muted">Sensor ID</label>
            <input
                type="number"
                className="form-control"
                value={values.sensorId}
                name="sensorId"
                disabled={true}/>
        </div>

        <div className="form-group">
            <label className="text-muted">Smoke Status</label>
            <input
                onChange={handleChange}
                type="number"
                className="form-control"
                min="0" max="10"
                name="smoke"
                value={values.smoke}/>
        </div>

        <div className="form-group">
            <label className="text-muted">CO2 Status</label>
            <input
                onChange={handleChange}
                type="number"
                className="form-control"
                min="0" max="10"
                name="co2"
                value={values.co2}/>
        </div>

        <button className="btn btn-dark">Update Sensor</button>
    </form>
);

const showError = () => (
    <div>{(error != '') ? <div className="alert alert-danger alert-dismissible">
        {error}
    </div> : ''}

    </div>

);

const showSuccess = () => (
    <div>{(success != '') ? <div className="alert alert-success">
        {success}
    </div> : ''}
```

```
        </div>

    );

    return (
        <div className="row">

                <div className="container">

                    <div className="jumbotron my-5 mx-2">
                        <h2>Update Sensor Status</h2>
                        <br/>
                        {showSuccess()}
                        {showError()}
                        {newPostForm(values)}
                        <div>
                            <span>Note: Sensor status will be updated automatically ever
y 10 seconds. You can manually update by clicking Update Sensor button</span>
                        </div>
                    </div>
                </div>



        </div>
    );
}
```

Appendix T

```
public class Alarm {
   int alarmId;
   int status;
   int floorNumber;
   int roomNumber;
   int smokeLevel;
   int co2Level;
   public int getAlarmId() {
      return alarmId;
   }

   public void setAlarmId(int alarmId) {
      this.alarmId = alarmId;
   }

   public int getStatus() {
      return status;
   }

   public void setStatus(int status) {
      this.status = status;
   }

   public int getFloorNumber() {
      return floorNumber;
   }

   public void setFloorNumber(int floorNumber) {
      this.floorNumber = floorNumber;
   }

   public int getRoomNumber() {
      return roomNumber;
   }

   public void setRoomNumber(int roomNumber) {
      this.roomNumber = roomNumber;
   }

   public int getSmokeLevel() {
      return smokeLevel;
   }

   public void setSmokeLevel(int smokeLevel) {
      this.smokeLevel = smokeLevel;
   }

   public int getCo2Level() {
      return co2Level;
   }

   public void setCo2Level(int co2Level) {
      this.co2Level = co2Level;
   }    }
```

Appendix U

```
public class Server extends UnicastRemoteObject implements Service {

   public static String TOKEN_BEARER; // Stores the admin token for future purposes.
   StringBuffer response = new StringBuffer(); // StringBuffer initialization. Stores the response from API Server.
   StringBuffer responseUsers = new StringBuffer();
   public static ArrayList<String> criticalAlarmIds = new ArrayList<String>();
   public static ArrayList<String> criticalAlarmIdsTemp = new ArrayList<String>();

   public Server() throws RemoteException {
      super();
      Timer timer = new Timer(); // Timer initialized in order to keep the program updated for every 15 seconds.
      timer.schedule(new TimerTask() {
         @Override
         public void run() {
            try {
               boolean critical = false;

               SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
               Date date = new Date();
               System.out.println("from server main");
               getResponseFromApi(); // This method is called every 15 seconds to take response from API server.

               ArrayList<Alarm> alarm = new ArrayList<>(); // ArrayList initialization.
               alarm = getList(response.toString()); // Get all the Alarm details by calling getList method in to
               // an ArrayList.

               for (int i = 0; i < alarm.size(); i++) {

                  if (alarm.get(i).smokeLevel > 5 || alarm.get(i).co2Level > 5) {
                     if (!criticalAlarmIds.contains(String.valueOf(alarm.get(i).alarmId))) {
                        criticalAlarmIds.add(String.valueOf(alarm.get(i).alarmId));
                        criticalAlarmIdsTemp = criticalAlarmIds;
                        critical = true;
                     }

                  } // Checks if there're alarms which are in critical situations and add those to

               }

               //send sms and email if the alarms are in critical condition
               if ((critical == true) && (criticalAlarmIds == criticalAlarmIdsTemp)) {
                  System.out.println("SMS Sent");
                  System.out.println("Email Sent");
                  sendSMS("Alert! Alarm ID " + criticalAlarmIds.toString() + " is/are in Critical Condition!");
                  sendEmail("Alert! Alarm ID " + criticalAlarmIds.toString() + " is/are in Critical Condition!");
               }
            } catch (Exception ex) {
               ex.printStackTrace();
            }
         }
      }, 0, 15000); // Timer set to 15 seconds

      Timer timer2 = new Timer();
      TimerTask hourlyJob = new TimerTask() {
```

```
            @Override
            public void run() {
                criticalAlarmIdsTemp.clear();
                criticalAlarmIds.clear();
            }
        };

        // run the resetting critical ids every hour
        timer2.schedule(hourlyJob, 0l, 1000 * 60 * 60);
    }

    public static void main(String[] args) {

        System.setProperty("java.security.policy", "file:allowall.policy"); // Security Permissions.

        try {

            Server svr = new Server(); // Creating the server.

            Registry registry = LocateRegistry.getRegistry();
            registry.bind("AlarmService", svr); // Bind service to the Registry.

            System.out.println("Service started....");
        } catch (RemoteException re) { // Catches exceptions.
            System.err.println(re.getMessage());
        } catch (AlreadyBoundException abe) {
            System.err.println(abe.getMessage());
        }

    }

    // This method is to get Alarm data from the API Server. This is called every 15
    // seconds.
    @Override
    public void getResponseFromApi() {
        String url = "http://localhost:8080/api/getSensors"; // The GET request URL.

        try {
            URL obj = new URL(url); // URL initialization.
            HttpURLConnection con = (HttpURLConnection) obj.openConnection(); // HttpURLConnection opens a
connection.

            con.setRequestMethod("GET"); // sets the Method
            con.setRequestProperty("User-Agent", "Mozilla/5.0");

            int responseCode = con.getResponseCode(); // Retrieves the status CODE such as, 200 or 404.

            BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); //
Initialized the
            // BufferReader.
            String inputLine;

            while ((inputLine = in.readLine()) != null) {
                this.response = null;
                this.response = new StringBuffer();
                this.response.append(inputLine); // Append the string to the response.
            }
            in.close(); // Buffer is closed.
```

```
        } catch (Exception e) {
            System.out.println("Make sure you've run the API Server!");
        }
    }

    // This method does return gathered response from API Server, to the RMI Client.
    @Override
    public StringBuffer returnAlarmDataFromApi() throws RemoteException {

        return response; // Return data.
    }

    // This method handle the Login of an Admin in a Secure Manner.
    @Override
    public String[] loginAdmin(String userName, String password) throws RemoteException {

        String jsonString = ""; // Creating the JSON Object to pass in the future.
        try {
            jsonString = new JSONObject().put("username", userName).put("password", password).toString();
        } catch (JSONException e) {
            e.printStackTrace();
        }

        try {
            Unirest.setTimeouts(0, 0);
            HttpResponse<String> response = Unirest.post("http://localhost:8080/api/auth/signin") // Http POST
request
                    // to the API
                    // Server.
                    .header("Content-Type", "application/json").body(jsonString) // Passing the JSON data to the API
                    // Server.
                    .asString();

            JSONObject myResponse = new JSONObject(response.getBody().toString()); // Creating an
JAONObject to collect
            // response.

            if (response.getStatus() == 200) { // Investigate the response status.
                Server.TOKEN_BEARER = myResponse.getString("accessToken"); // Retreiving the Admin Token.

                String[] values = new String[2]; // Creation of String array to be sent to the RMI Client.
                values[0] = "Authorized!";
                values[1] = myResponse.getString("accessToken");
                return values;
            } else {
                String[] values = new String[2];
                values[0] = "Unauthorized!";
                return values; // Returns "Unauthorized!".
            }
        } catch (UnirestException ex) {
            Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
        } catch (JSONException ex) {
            Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        return null;

    }
```

```
    // This method is to Register a new alarm.
    @Override
    public String[] addAlarmSensor(String jsonDetails, String token) throws RemoteException {
        String[] response = new String[1]; // Creation of String array to be sent to the RMI Client.
        try {

            Unirest.setTimeouts(0, 0); // Time out from UNIREST.
            HttpResponse<String> responseGet = Unirest.post("http://localhost:8080/api/admin/addSensor") //
POST
                    // Request.
                    .header("Authorization", "Bearer " + token).header("Content-Type", "application/json")
                    .body(jsonDetails) // Passing the JSON data to the API Server.
                    .asString();

            if (responseGet.getStatus() == 200) { // Investigate the response status.
                response[0] = String.valueOf(responseGet.getStatus());
                getResponseFromApi();

            } else {
                response[0] = String.valueOf(responseGet.getStatus());
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return response; // Return the response.
    }

    // This method habdle thi Update process of an Alarm.
    @Override
    public String[] updateAlarmSensor(String jsonDetails, String token) throws RemoteException {

        String[] values = new String[2];
        try {
            Unirest.setTimeouts(0, 0);
            HttpResponse<String> response = Unirest.put("http://localhost:8080/api/admin/updateSensor") //
Perform the
                    // PUT Action.
                    .header("Authorization", "Bearer " + token) // Passing the Admin Token to get the permission to
the
                    // update process.
                    .header("Content-Type", "application/json").body(jsonDetails) // Pass the created JSON Data to
the
                    // PUT request.
                    .asString();

            if (response.getStatus() == 200) {
                values[0] = String.valueOf(response.getStatus());
            } else {
                values[0] = String.valueOf(response.getStatus());
            }

        } catch (Exception ex) {
            ex.printStackTrace();
        }

        return values;
    }
```

```
public ArrayList<Alarm> getList(String json) {

    ArrayList<Alarm> arrayList = new ArrayList<>();
    try {
        JSONArray jsonArray = new JSONArray(json); // Initialized JsonArray to get the JSONArray response.
        for (int count = 0; count < jsonArray.length(); count++) {
            Alarm alarmObject = new Alarm(); // Alarm Object initialzation.
            JSONObject jsonObject = jsonArray.getJSONObject(count);
            alarmObject.setAlarmId(jsonObject.getInt("sensorId")); // Storing the data in to an object.
            alarmObject.setFloorNumber(jsonObject.getInt("floorNo"));
            alarmObject.setRoomNumber(jsonObject.getInt("roomNo"));
            alarmObject.setSmokeLevel(jsonObject.getInt("smoke"));
            alarmObject.setCo2Level(jsonObject.getInt("co2"));
            alarmObject.setStatus(jsonObject.getInt("active"));
            // Taking data from particular Key.

            arrayList.add(alarmObject); // Object added to the ArrayList
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return arrayList;
}
```

Appendix V

```
public class Login extends javax.swing.JFrame {

   Timer timer = new Timer(); //Explained in Home.java, Code segments are the same.
   public static Service service = null;

   public Login() throws Exception {
      initComponents();
//       getAlarmData();

      timer.schedule(new TimerTask() {
         @Override
         public void run() {
            try {
               SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
               Date date = new Date();
               getAlarmData();
               System.out.println("called");
               labelLogin.setText("Refreshed! " + formatter.format(date));
            } catch (Exception ex) {
               Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);
            }
         }
      }, 0, 30000);

      jTableLogin.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {
         @Override
         public Component getTableCellRendererComponent(JTable table,
               Object value, boolean isSelected, boolean hasFocus, int row, int col) {

            Component c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, col);

            int co2 = (int) table.getModel().getValueAt(row, 5);
            int smoke = (int) table.getModel().getValueAt(row, 4);

            String status = (String) table.getModel().getValueAt(row, 1);

            if (smoke > 5 || co2 > 5) {

               setBackground(new Color(255, 191, 191));
               setForeground(Color.BLACK);
            } else {
               setBackground(new Color(191, 255, 191));
               setForeground(table.getForeground());
            }

            if(status == "Inactive"){
               setBackground(new Color(226, 226, 226));
               setForeground(Color.BLACK);
            }

            return c;
         }
      });
   }
```

```
public void getAlarmData() throws Exception {//Explained in Home.java. Code segments are the same.

   boolean critical = false;

   StringBuffer response = service.returnAlarmDataFromApi();

   ArrayList<Alarm> alarm = new ArrayList<>();
   alarm = getList(response.toString());

   DefaultTableModel model = (DefaultTableModel) jTableLogin.getModel();
   model.setRowCount(0);
   Object rowData[] = new Object[6];

   ArrayList<String> criticalAlarmIds = new ArrayList<String>();

   for (int i = 0; i < alarm.size(); i++) {
      rowData[0] = alarm.get(i).alarmId;
      if(alarm.get(i).status == 1){
         rowData[1] = "Active";
      }else{
         rowData[1] = "Inactive";
      }
      rowData[2] = alarm.get(i).floorNumber;
      rowData[3] = alarm.get(i).roomNumber;
      rowData[4] = alarm.get(i).smokeLevel;
      rowData[5] = alarm.get(i).co2Level;

      if (alarm.get(i).smokeLevel > 5 || alarm.get(i).co2Level > 5) {
         criticalAlarmIds.add(String.valueOf(alarm.get(i).alarmId));
         critical = true;
      }

      if (critical == true) {
         alertDisplayLogin.setText("Alert! Alarm ID " + criticalAlarmIds + " is/are in Critical Condition!");
      } else {

         alertDisplayLogin.setText("");
      }

      model.addRow(rowData);
   }

}

public ArrayList<Alarm> getList(String json) {

   ArrayList<Alarm> arrayList = new ArrayList<>();
   try {
      JSONArray jsonArray = new JSONArray(json);
      for (int count = 0; count < jsonArray.length(); count++) {
         Alarm alarmObject = new Alarm();
         JSONObject jsonObject = jsonArray.getJSONObject(count);
         alarmObject.setAlarmId(jsonObject.getInt("sensorId"));
         alarmObject.setFloorNumber(jsonObject.getInt("floorNo"));
         alarmObject.setRoomNumber(jsonObject.getInt("roomNo"));
         alarmObject.setSmokeLevel(jsonObject.getInt("smoke"));
         alarmObject.setCo2Level(jsonObject.getInt("co2"));
         alarmObject.setStatus(jsonObject.getInt("active"));
```

```
            arrayList.add(alarmObject);
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return arrayList;
}
private void submitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_submitActionPerformed
    String userName = usernameInput.getText().trim();
    String password = passwordInput.getText().trim();
    String response[] = null;

    statusDisplayLogin.setText("");

    try {
        response = service.loginAdmin(userName, password);

        if (response[0].equals("Authorized!")) {
            this.setVisible(false);
            timer.cancel();
            String srgs[] = new String[2];
            srgs[0] = response[1];

//              new Home().setVisible(true);
            Home.main(srgs);
        } else if (response[0].equals("Unauthorized!")){
            statusDisplayLogin.setText("Unauthorized!");
        }

    } catch (RemoteException ex) {
        ex.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }


}//GEN-LAST:event_submitActionPerformed

public static void main(String args[]) {

    System.setProperty("java.security.policy", "file:allowall.policy");

    try {
        service = (Service) Naming.lookup("//localhost/AlarmService");


        service.returnAlarmDataFromApi();

    } catch (NotBoundException ex) {
        System.err.println(ex.getMessage());
    } catch (MalformedURLException ex) {
        System.err.println(ex.getMessage());
    } catch (RemoteException ex) {
        System.err.println(ex.getMessage());
    }
```

Appendix W

```
public class Home extends javax.swing.JFrame {

   Timer timer = new Timer(); //Timer initialized in order to keep the program updated for every 5 seconds.
   public static Service service = null; //Reference to the RMI Server.
   public static String AdminToken = null; //AdminToken declaration.

   public Home() {
      initComponents();
      updateError.setVisible(false);
      errorInsert.setVisible(false);
      errorDelete.setVisible(false);
      this.setTitle("Admin Dashboard");
      try {
         getAlarmData();
      } catch (Exception ex) {
         Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
      }

      timer.schedule(new TimerTask() {
         @Override
         public void run() {
            try {
               SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
               Date date = new Date(); //This is to inform the user the application is updated in this particular
time.
               getAlarmData(); //This is called to get data from the RMI Server.
               statusDisplayHome.setText("Refreshed! " + formatter.format(date));
            } catch (Exception ex) {
               ex.printStackTrace();
            }
         }
      }, 0, 30000);

      jTable8.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {
         @Override
         public Component getTableCellRendererComponent(JTable table,
               Object value, boolean isSelected, boolean hasFocus, int row, int col) {

            Component c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, col);

            int co2 = (int) table.getModel().getValueAt(row, 5);
            int smoke = (int) table.getModel().getValueAt(row, 4);
            String status = (String) table.getModel().getValueAt(row, 1);

            if (smoke > 5 || co2 > 5) {

               setBackground(new Color(255, 191, 191));
               setForeground(Color.BLACK);
            } else {
               setBackground(new Color(191, 255, 191));
               setForeground(table.getForeground());
            }

            if (status == "Inactive") {
               setBackground(new Color(226, 226, 226));
               setForeground(Color.BLACK);
            }
```

```
            return c;
        }
    });
}

public void getAlarmData() throws Exception {

    boolean critical = false; //Used to take the decision whether to display the critical data information on display.

    StringBuffer response = service.returnAlarmDataFromApi();

    ArrayList<Alarm> alarm = new ArrayList<>();  //ArrayList initialization.
    alarm = getList(response.toString()); //Get all the Alarm details by calling getList method in to an ArrayList.

    DefaultTableModel model = (DefaultTableModel) jTable8.getModel(); //Accessing the Table in the UI.
    model.setRowCount(0);
    Object rowData[] = new Object[6]; //Initial step to add rows to the Table.

    ArrayList<String> criticalAlarmIds = new ArrayList<String>(); //This stores AlarmIDs which has a critical situation.

    //Here the JSON Array data will be added to the JTable within a for loop.
    for (int i = 0; i < alarm.size(); i++) {
        rowData[0] = alarm.get(i).alarmId;
        if (alarm.get(i).status == 1) {
            rowData[1] = "Active";
        } else {
            rowData[1] = "Inactive";
        }

        rowData[2] = alarm.get(i).floorNumber;
        rowData[3] = alarm.get(i).roomNumber;
        rowData[4] = alarm.get(i).smokeLevel;
        rowData[5] = alarm.get(i).co2Level;

        if (alarm.get(i).smokeLevel > 5 || alarm.get(i).co2Level > 5) {
            criticalAlarmIds.add(String.valueOf(alarm.get(i).alarmId));
            critical = true;
        } //Checks if there're alarms which are in critical situations and add those to the criticalAlarmIds ArrayList.

        if (critical == true) {
            alertDisplayHome.setText("Alert! Alarm ID " + criticalAlarmIds + " is/are in Critical Condition!");
        } else {

            alertDisplayHome.setText("");  //Set the alert to none if there's not alarm in critical condition.
        }

        model.addRow(rowData); //Add rows to the JTable.
    }

}

//This is used to get the JSON response and derive the JSON array in to separated pieces and add them to the ArrayList.
public ArrayList<Alarm> getList(String json) {

    ArrayList<Alarm> arrayList = new ArrayList<>();
    try {
```

```
            JSONArray jsonArray = new JSONArray(json); //Initialized JsonArray to get the JSONArray response.
            for (int count = 0; count < jsonArray.length(); count++) {
                Alarm alarmObject = new Alarm(); //Alarm Object initialzation.
                JSONObject jsonObject = jsonArray.getJSONObject(count);
                alarmObject.setAlarmId(jsonObject.getInt("sensorId")); //Storing the data in to an object.
                alarmObject.setFloorNumber(jsonObject.getInt("floorNo"));
                alarmObject.setRoomNumber(jsonObject.getInt("roomNo"));
                alarmObject.setSmokeLevel(jsonObject.getInt("smoke"));
                alarmObject.setCo2Level(jsonObject.getInt("co2"));
                alarmObject.setStatus(jsonObject.getInt("active"));
                //Taking data from particular Key.

                arrayList.add(alarmObject); //Object added to the ArrayList
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return arrayList;
    }
private void submitBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_submitBtnActionPerformed
        if (validateInsertInputs()) {
            errorInsert.setVisible(false);

            String roomNumber = roomNumberRegister.getText(); //Getting the roomNumber from  the admin.
            String floorNumber = floorNumberRegister.getText(); //Getting the floorNumber from  the admin.
            int status = cmbInsertStatus.getSelectedIndex(); //Getting the status from  the admin.

            String jsonString = ""; //Creating a method to make a JSONObject.
            try {
                jsonString = new JSONObject()
                        .put("floorNo", floorNumber)
                        .put("roomNo", roomNumber)
                        .put("active", status).toString();
            } catch (JSONException e) {
                e.printStackTrace();
            }
            String[] response = new String[3];

            try {
                response = service.addAlarmSensor(jsonString, AdminToken);
            } catch (RemoteException ex) {
                Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
            }

            SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
            Date date = new Date();//This is used to identify Admin that a particular action happened on this time.

            if (response[0].equals("200")) {
                try {
                    service.getResponseFromApi();
                    getAlarmData();
                } catch (RemoteException ex) {
                    Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
                } catch (Exception ex) {
                    Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
                }
                statusDisplayHome.setText("Sensor Successfully Registered! " + formatter.format(date));  //Getting
the POST response and display success.
                roomNumberRegister.setText(""); //Setting blanks to the inputs.
```

```
            floorNumberRegister.setText(""); //Setting blanks to the inputs.
            cmbInsertStatus.setSelectedIndex(1); //set the default status

            try {
                this.getAlarmData();
            } catch (Exception ex) {
                Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
            }

        } else if (!response[0].equals("200")) {
            statusDisplayHome.setText("Error in Sensor Register! " + formatter.format(date)); //Show this if
JSON responded an error.
        }
      } else {
        errorInsert.setVisible(true);
      }
   }//GEN-LAST:event_submitBtnActionPerformed

   //This method is to perform the Alarm update task.
   private void updateBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_updateBtnActionPerformed
      if (validateUpdateInputs()) {
        updateError.setVisible(false);

        String id = idUpdate.getText().trim(); //Getting the id from  the admin.
        String roomNumber = roomNumberUpdate.getText().trim(); //Getting the roomNumber from  the
admin.
        String floorNumber = floorNumberUpdate.getText().trim(); //Getting the floorNumber from  the
admin.
        //String status = statusEdit.getText().trim(); //Getting the status from  the admin.
        int status = cmbUpdateStatus.getSelectedIndex();

        String jsonString = ""; //As mentioned earlier this is used to create a JSON Object.
        try {
            jsonString = new JSONObject()
                    .put("sensorId", id)
                    .put("floorNo", floorNumber)
                    .put("roomNo", roomNumber)
                    .put("active", status).toString();
        } catch (JSONException e) {
            e.printStackTrace();
        }

        SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        Date date = new Date();

        String[] response = new String[3];

        try {
            response = service.updateAlarmSensor(jsonString, AdminToken);
        } catch (RemoteException ex) {
            Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
        }

        if (response[0].equals("200")) {
            statusDisplayHome.setText("Sensor Successfully Updated! " + formatter.format(date)); //Display
Success Message.
            idUpdate.setText(""); //Clean the inputs
            roomNumberUpdate.setText(""); //Clean the inputs
            floorNumberUpdate.setText(""); //Clean the inputs
            cmbUpdateStatus.setSelectedIndex(1); //Clean the inputs
```

```
            try {
                service.getResponseFromApi();
                getAlarmData();
            } catch (Exception ex) {
                Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);
            }

        } else if (!response[0].equals("200")) {
            statusDisplayHome.setText("Error in Sensor Update! " + formatter.format(date));
        }
    } else {
        updateError.setVisible(true);
    }


}//GEN-LAST:event_updateBtnActionPerformed
```