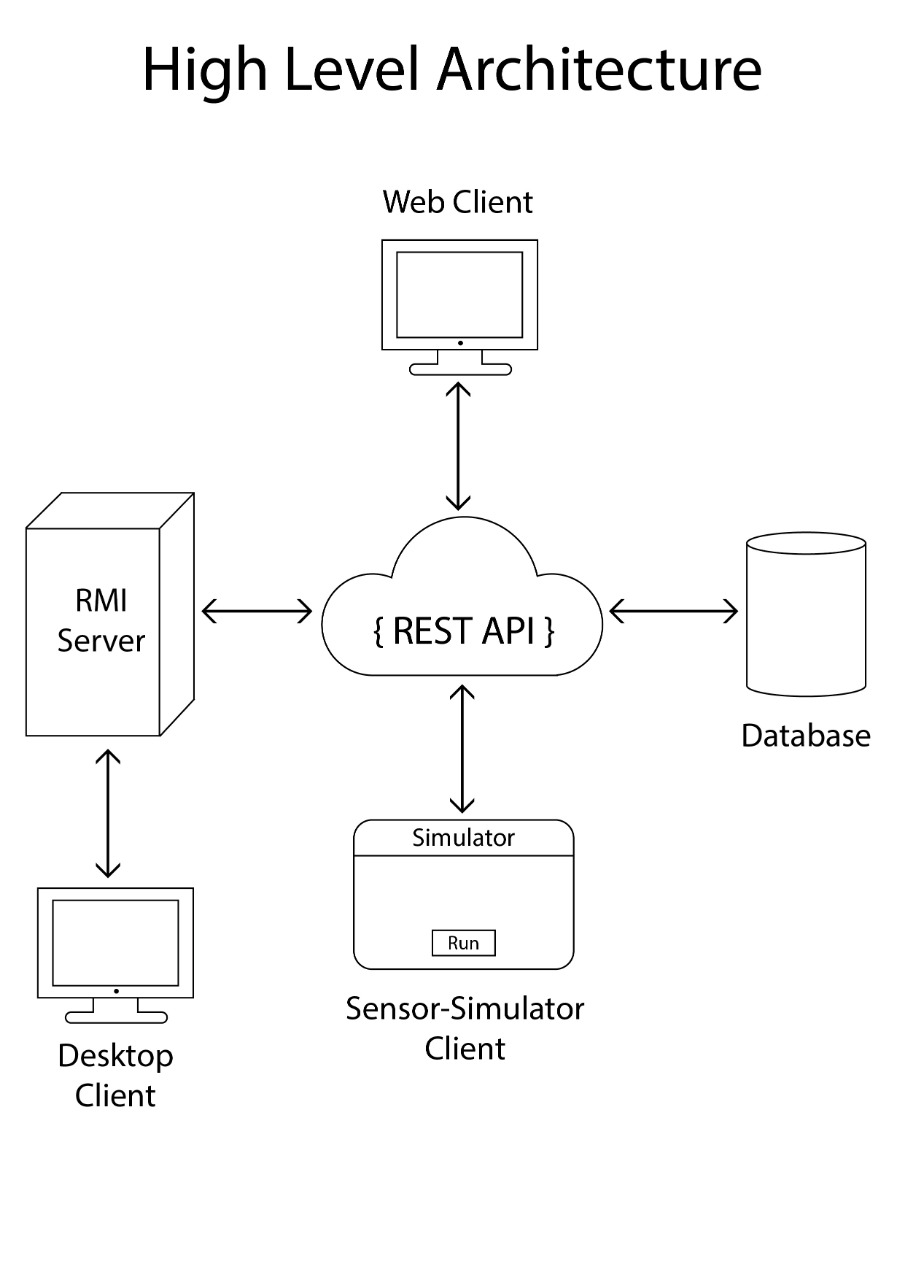
DS – SE3020

Assignment 2 – REST API

|  |  |
| --- | --- |
| **Group Details** | |
| **Student ID** | **Name** |
| IT18013610 | Ekanayake P.M.D.P |
| IT18004564 | Thennakoon T.M.K.H.B |
| IT18011012 | Ranasinghe R.M.A.K |
| IT18078992 | Marapana S.K.C.W.K.M.R.T.S.B |

High lever architecture diagram

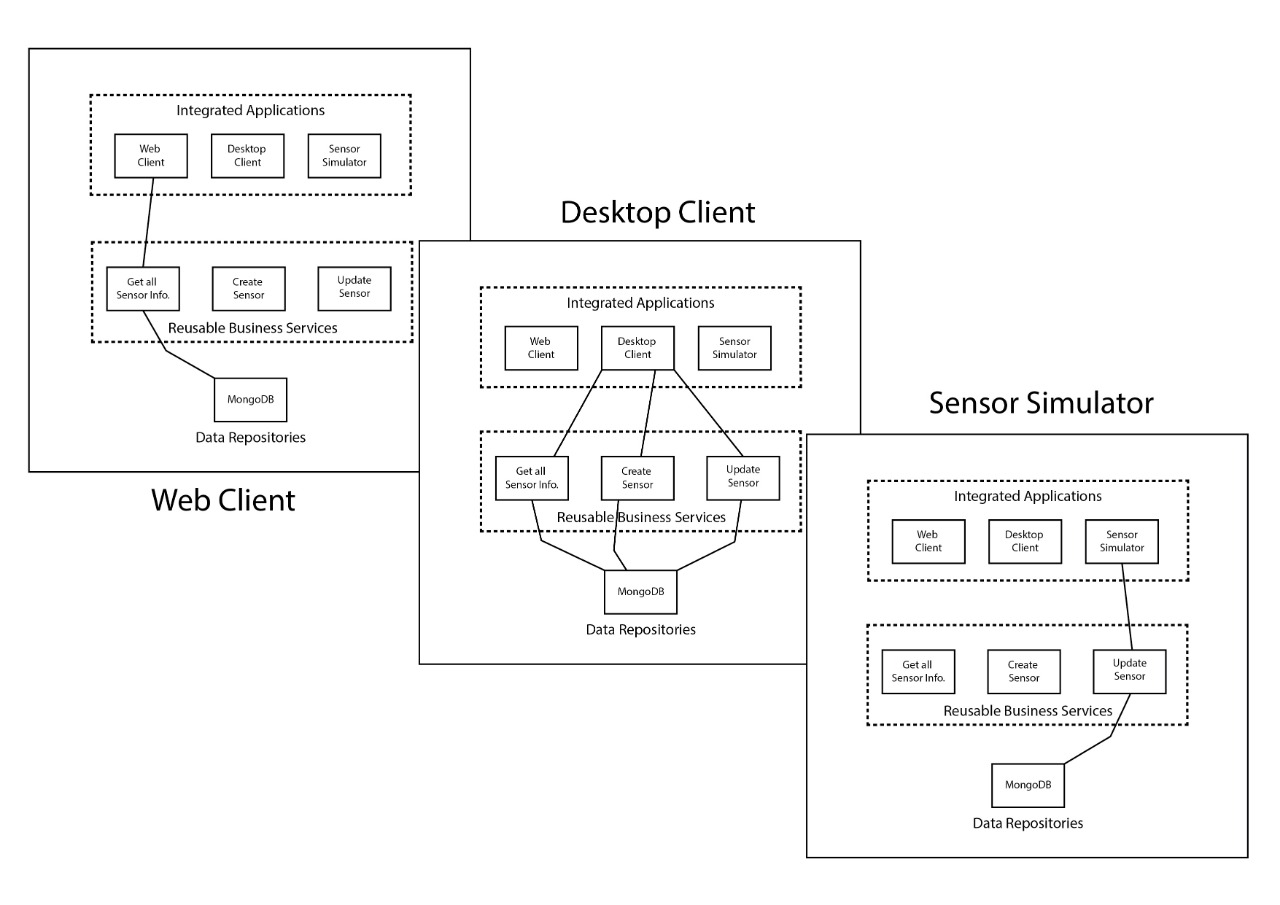
(showing the services and their interconnectivity.)



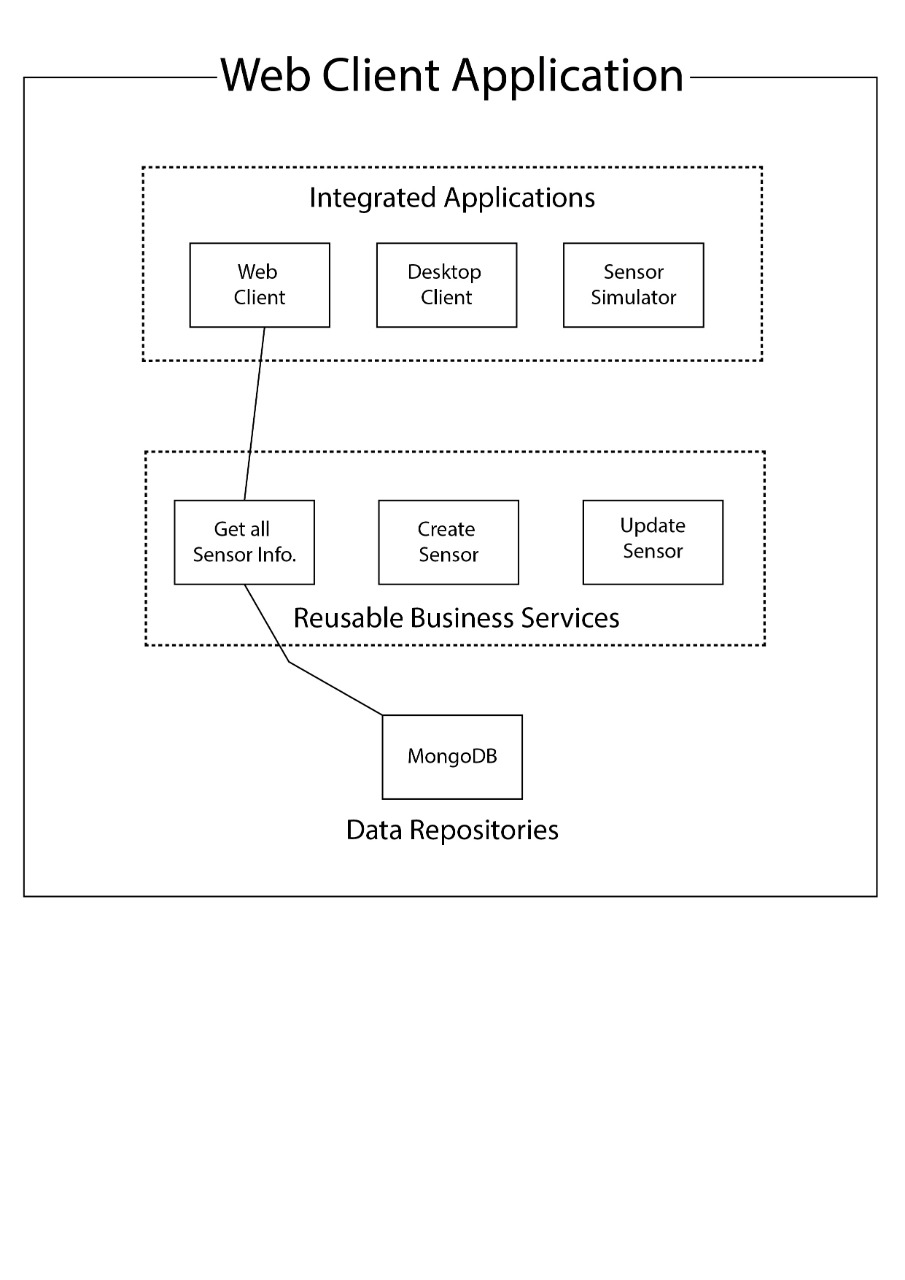
This is the structure of our system, Desktop client will connect to RMI server. RMI server, sensor simulator and web client will connect to REST API and api is connected to the Database.

SOA STYLE

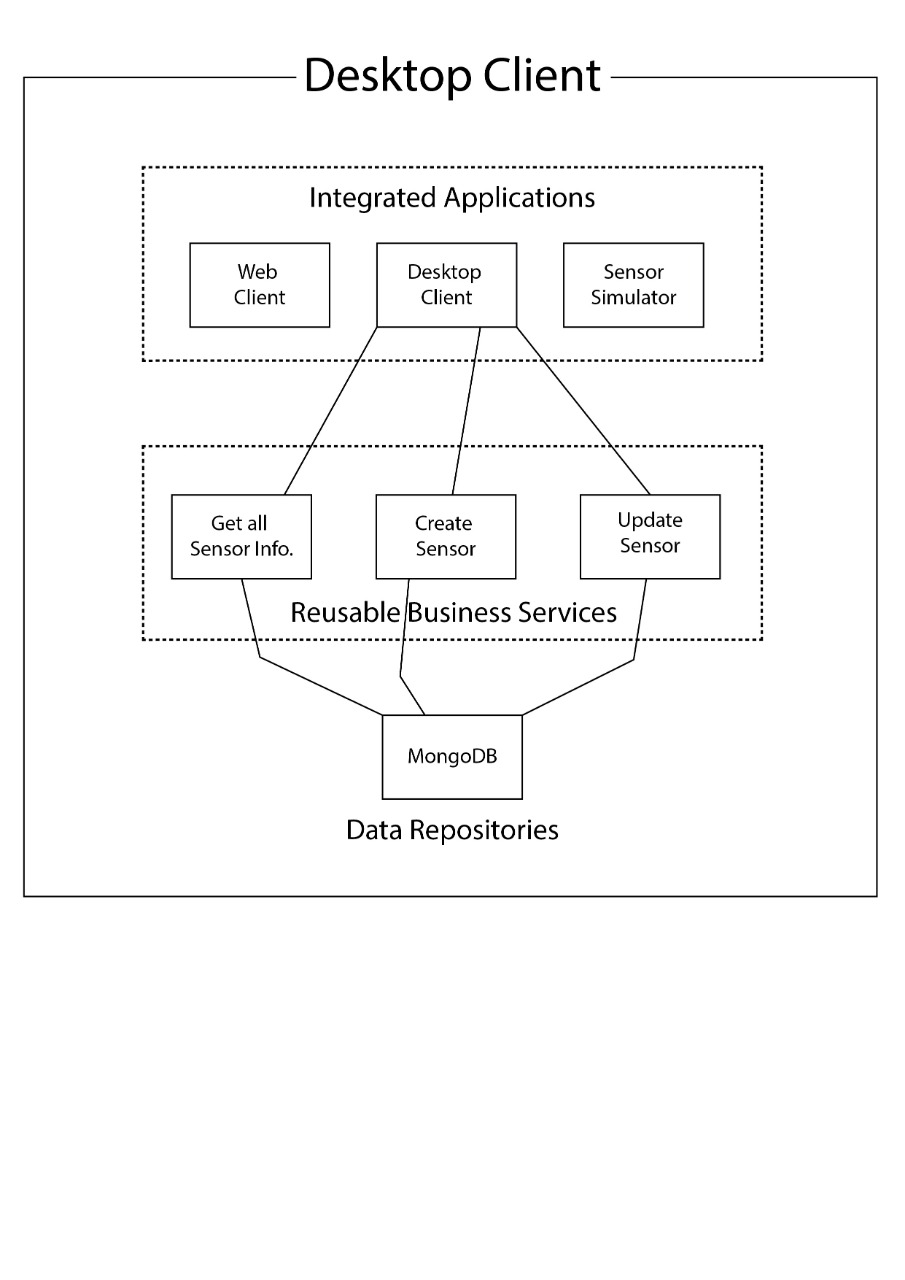
(Service oriented architecture of our system)



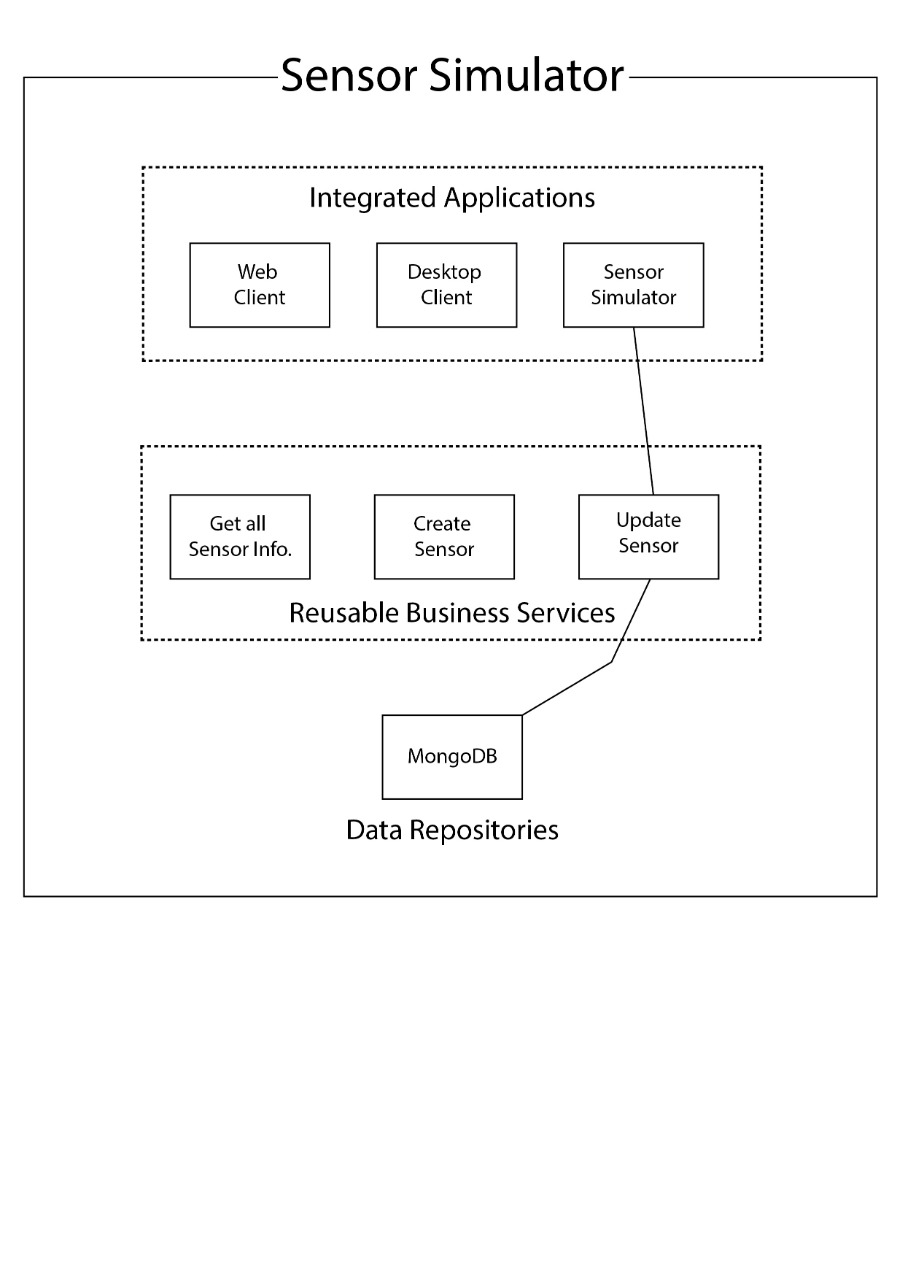
Web client of SOA:



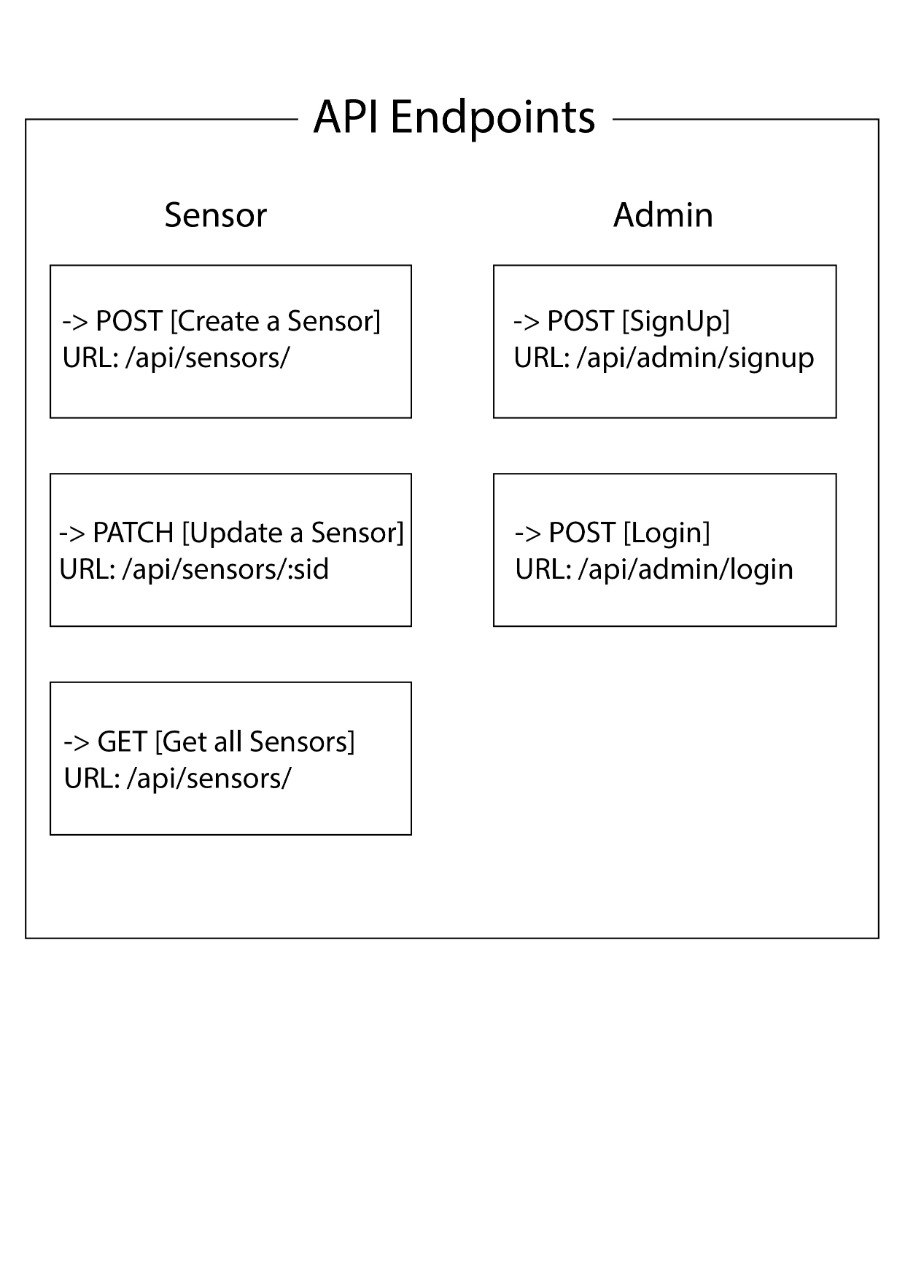
Desktop Client of SOA:



Sensor Simulator SOA:



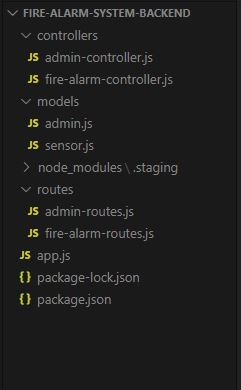
REST API



This REST API has routes for create, update and read the Sensors. We used a Model-Controller architecture to build this REST API.

We have used node.js and Express.js framework to create this API. Express js has released the workload of handling the routes, creating request end-points and the server. Also express has a middleware called body-parser, which is a 3rd party package. Using that package, we can easily extract the entire body portion of a any request easily.

This is the folder structure of the REST API.



Model folder contains the source code of the Schemas for Sensor and Admin.

Controllers folder contains the source code of all the CRUD operations related to the Sensor and admin. We have used the “express-validator” 3rd party package to validate all the client requests using its check() function. All the database query related errors are handled by using try catch blocks.

In app.js file we create two custom middlewares. First one handles all the invalid API requests throws a error with a message and a error code, and second one handles all the errors which are thrown by other middlewares. ExpressJs framework has used to

Database

MongoDB - Mongoose

We have use MongoDB as our database for this project because fire sensors are updating all the time and clients also requesting updated sensors details through the API in every 40 seconds. So there are many request to database in very short time period. MongoDB is really good at handling these kinds of operations.

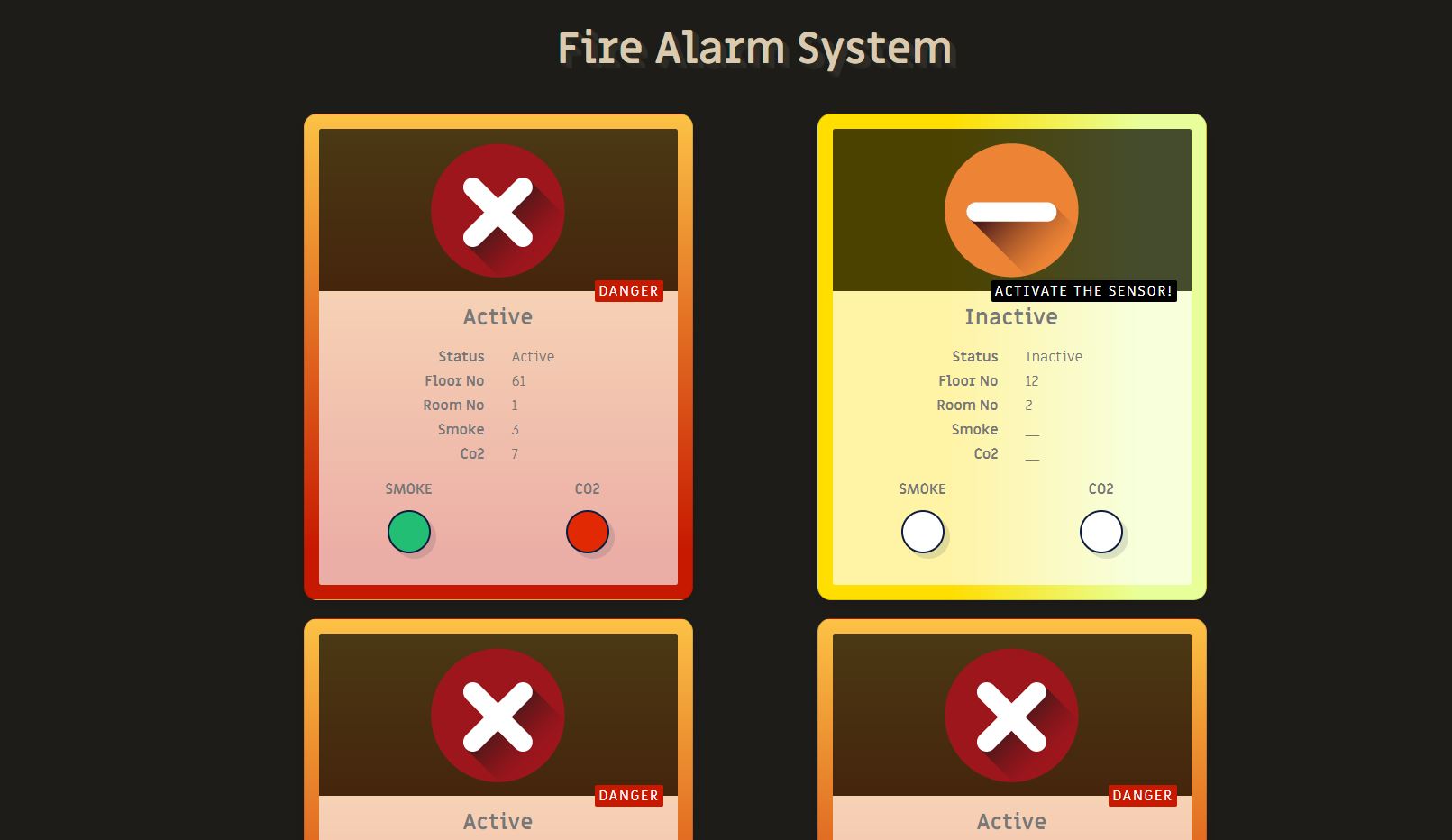
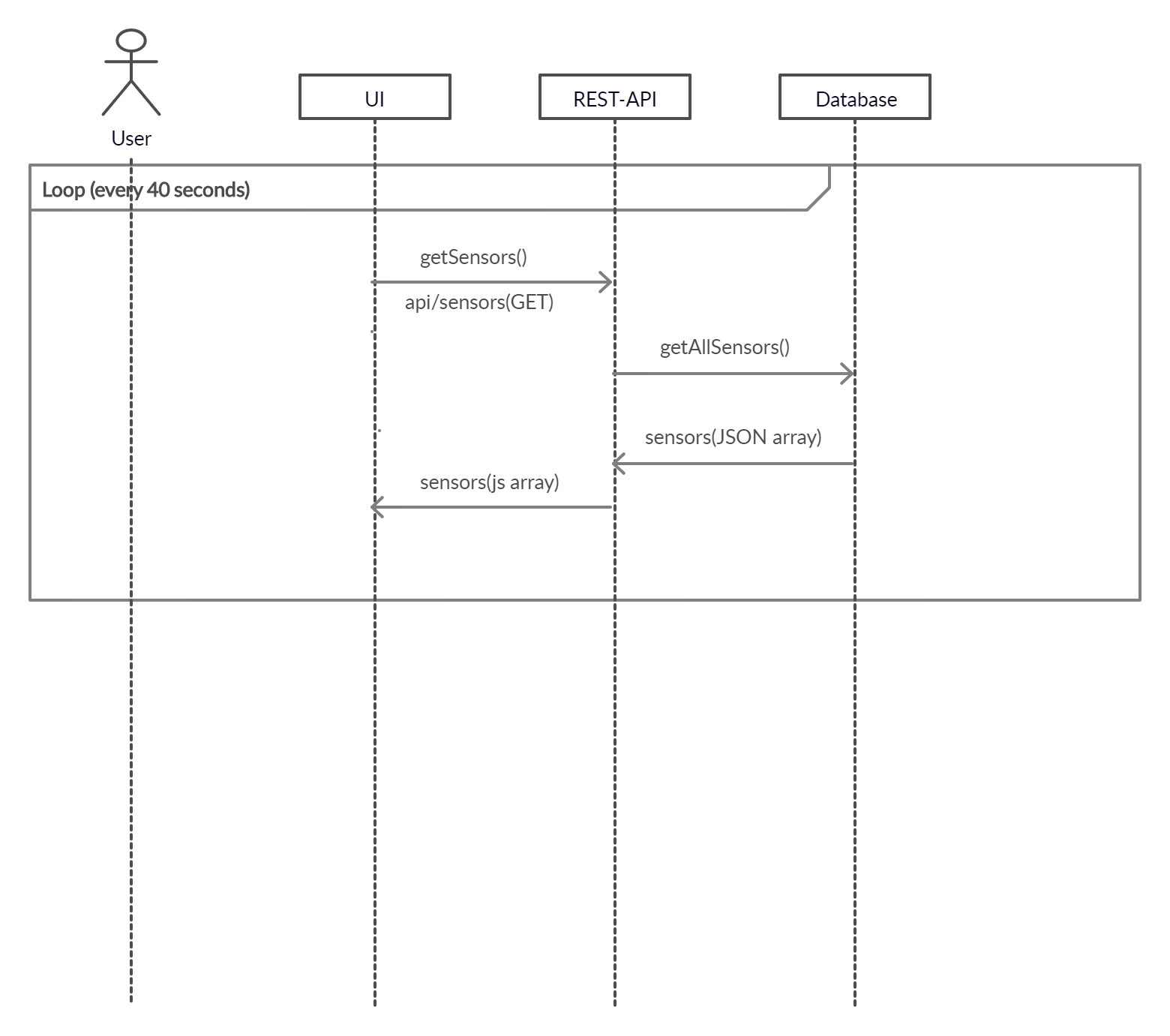
In this project we have used the mongoose, which is and Object data modeling (ODM) library, to model our data. Using the mongoose schema, we structure the data by adding required keyword to indicate a field is required and giving data types (String, Boolean, etc.) to sensor properties. Then we turned it to a model using mongoose.model() function. Mongoose model provides queries to find records in the database and to performs all the CRUD operations.

Service Interfaces

(of each service and **sequence diagram** for each to show the work flow)

**Web Client**

React



Used technologies: React js , fetch api, bootsrap

To create web client React js is used for front end. Since this is already a frontend part there is no backend codes. (fire-alarm-backend is the backend of this project)

Fetch api: to fetch data from the REST API and use them in application we used fetch api.

Bootstrap4 : we use Bootstrap 4 to style our front end

See: [backend\_app\_js](#backend_app_js) [adminController](#adminController) [sensor\_component\_react](#sensor_component_react)

[frontend\_app\_js](#frontend_app_js) [fire\_alarm\_controller](#fire_alarm_controller) [sensorlist\_component\_react](#sensorlist_component_react)

[Admin\_model\_js](#Admin_model_js) [Fire\_alarm\_routes](#Fire_alarm_routes)

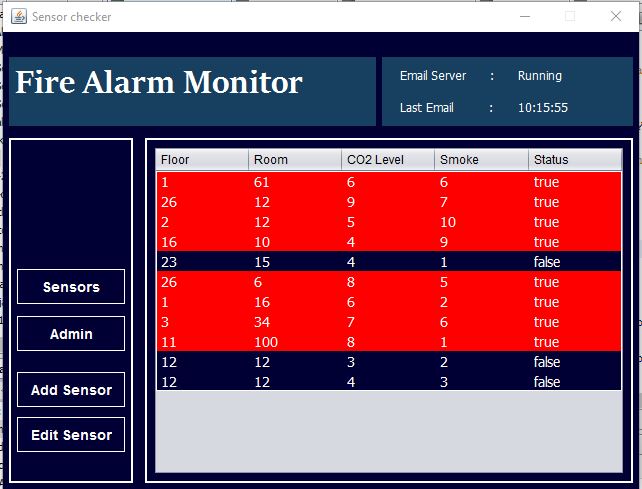
[sensor\_model\_js](#sensor_model_js) [admin\_routes](#admin_routes)

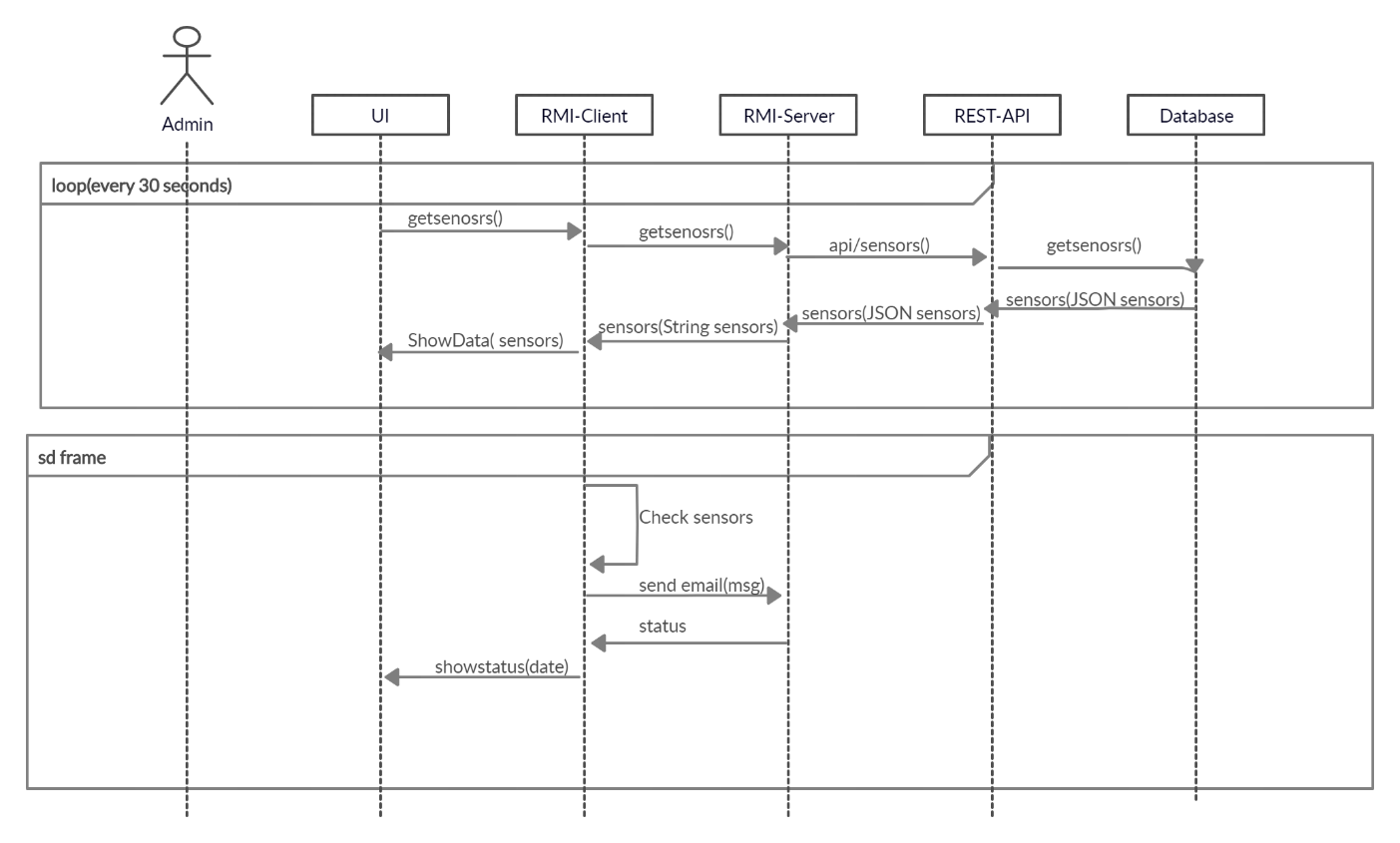
**RMI-Client –View Sensor data and send email**

Get Sensor details from database through api and set data to a Table.

If co2 level or smoke level is greater than 5 row colors will be changed to red

See : [getSensor\_client](#getSensor_client), [getSensor\_rmi\_server](#getSensor_rmi_server), [set\_jTable](#set_jTable), [parse](#parse), [send\_email\_client](#send_email_client), [send\_email\_server](#send_email_server)



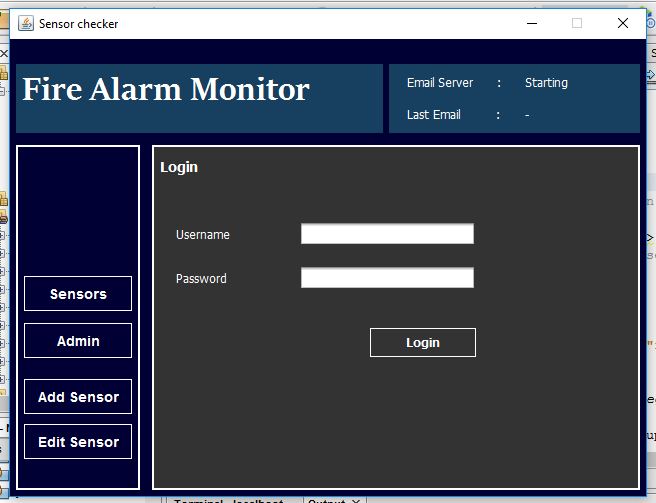


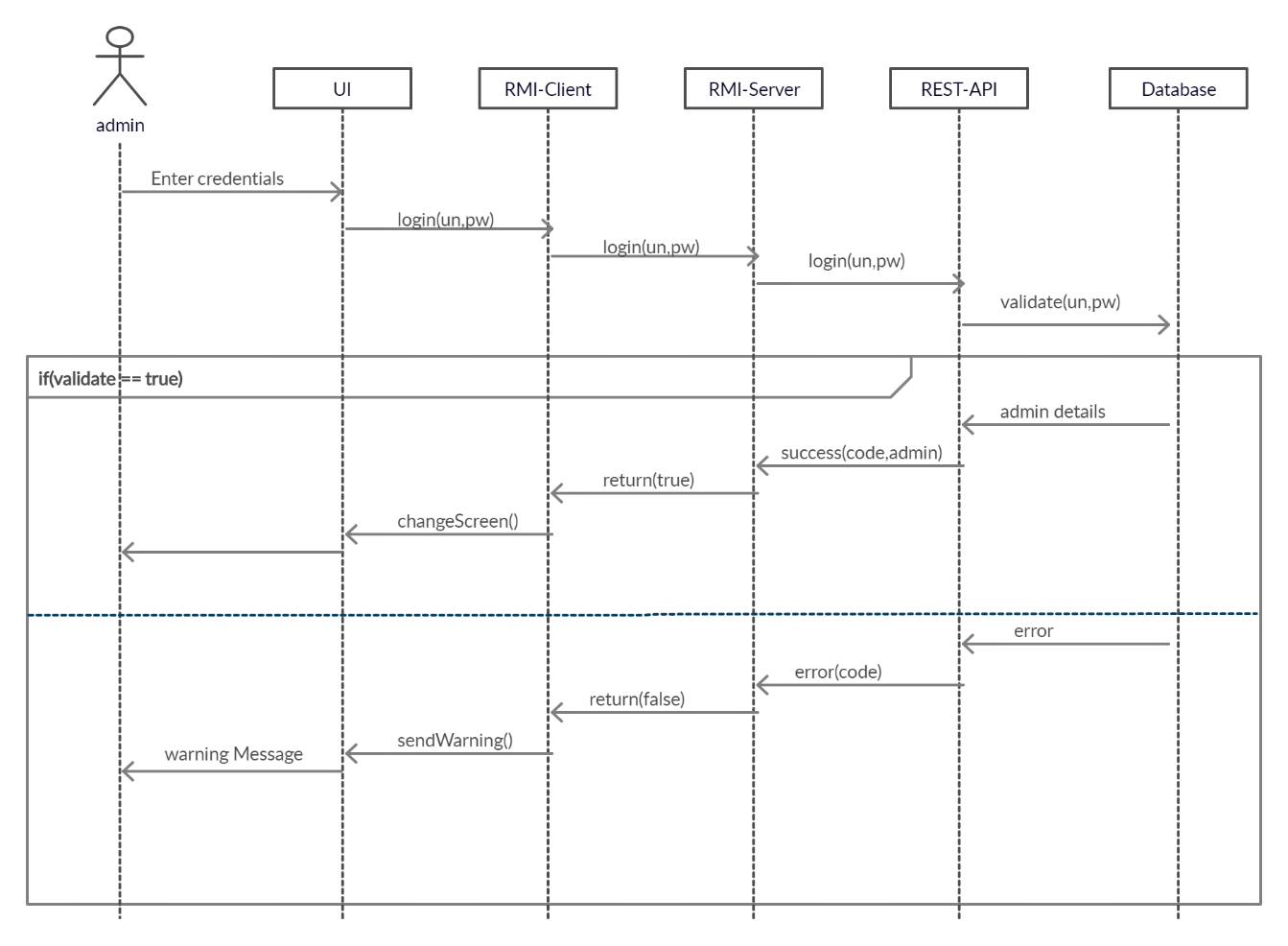
**Admin Login**

Admin need to log into client application in order to add or edit sensor.

Use : [logan@gmail.com](mailto:logan@gmail.com) and password : 1234

See : [admin\_login](#admin_login), [server\_login](#server_login)

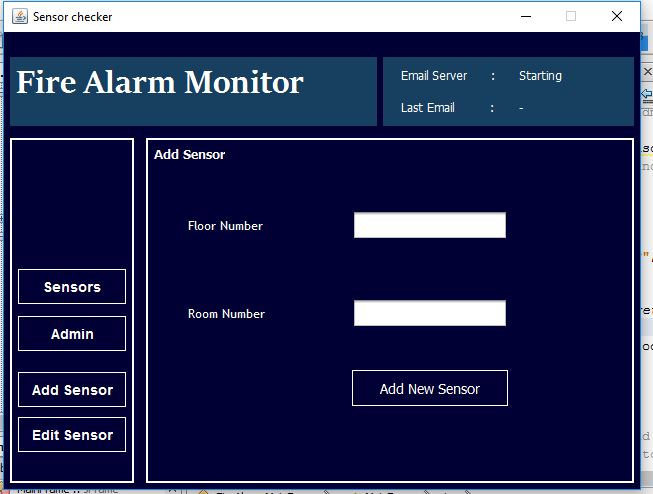
****

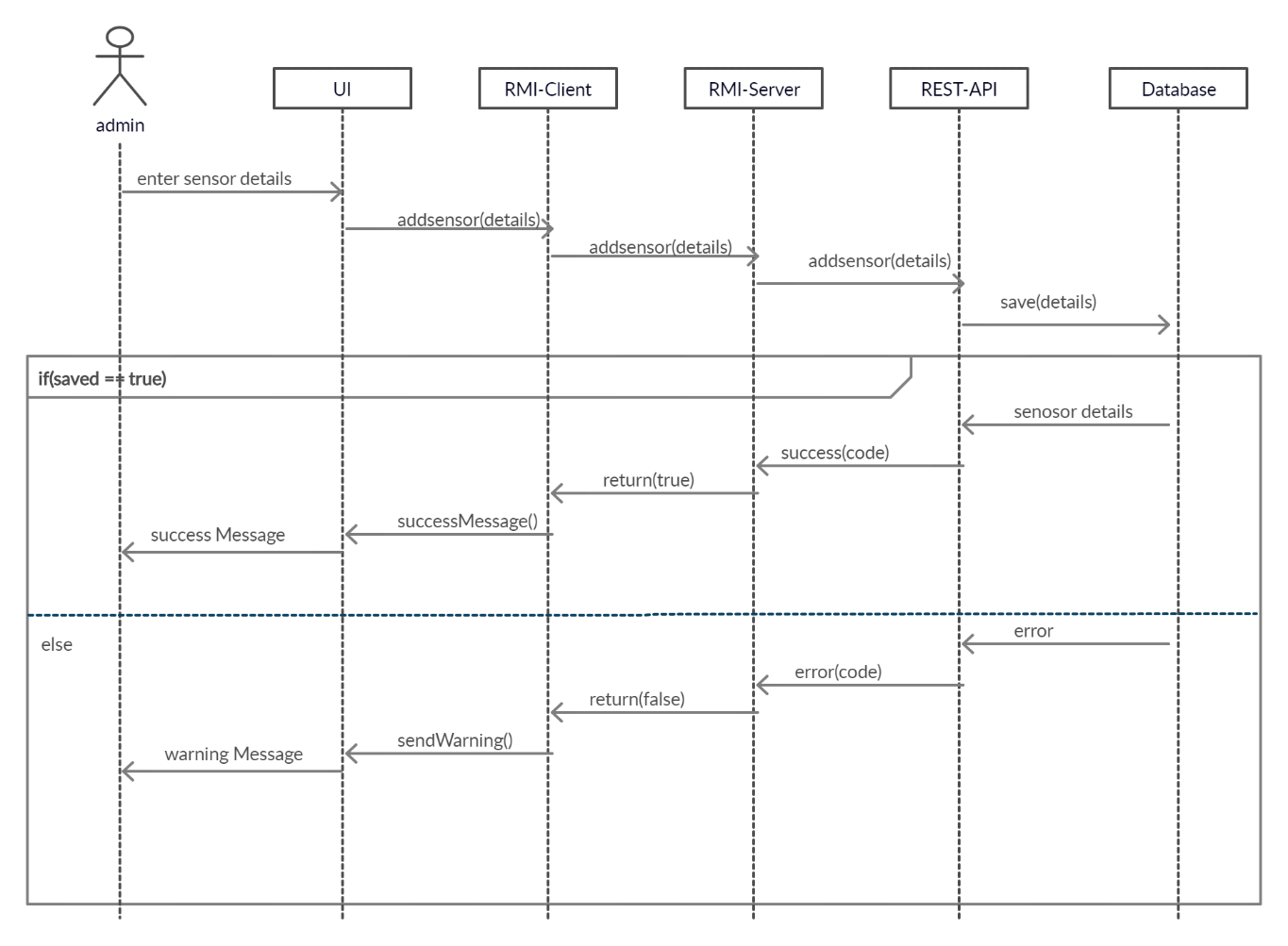


**Add Sensor**

Admin can add sensors after log in. Floor number and room number must be filled by admin

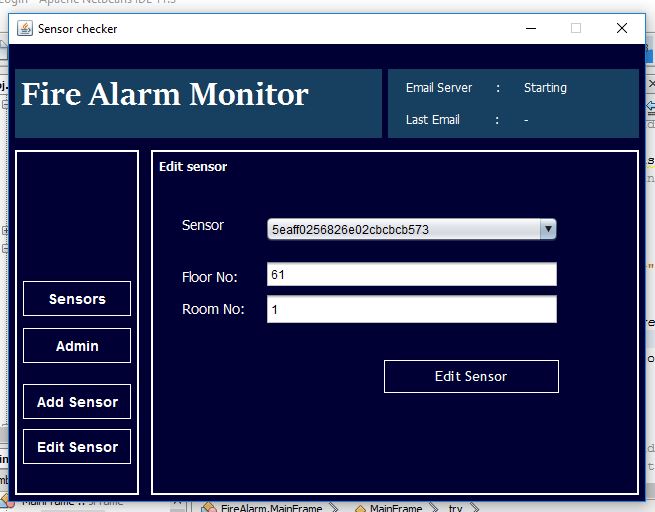
See: [add\_new\_sensor\_server](#add_new_sensor_server), [add\_sensor\_client](#add_sensor_client)

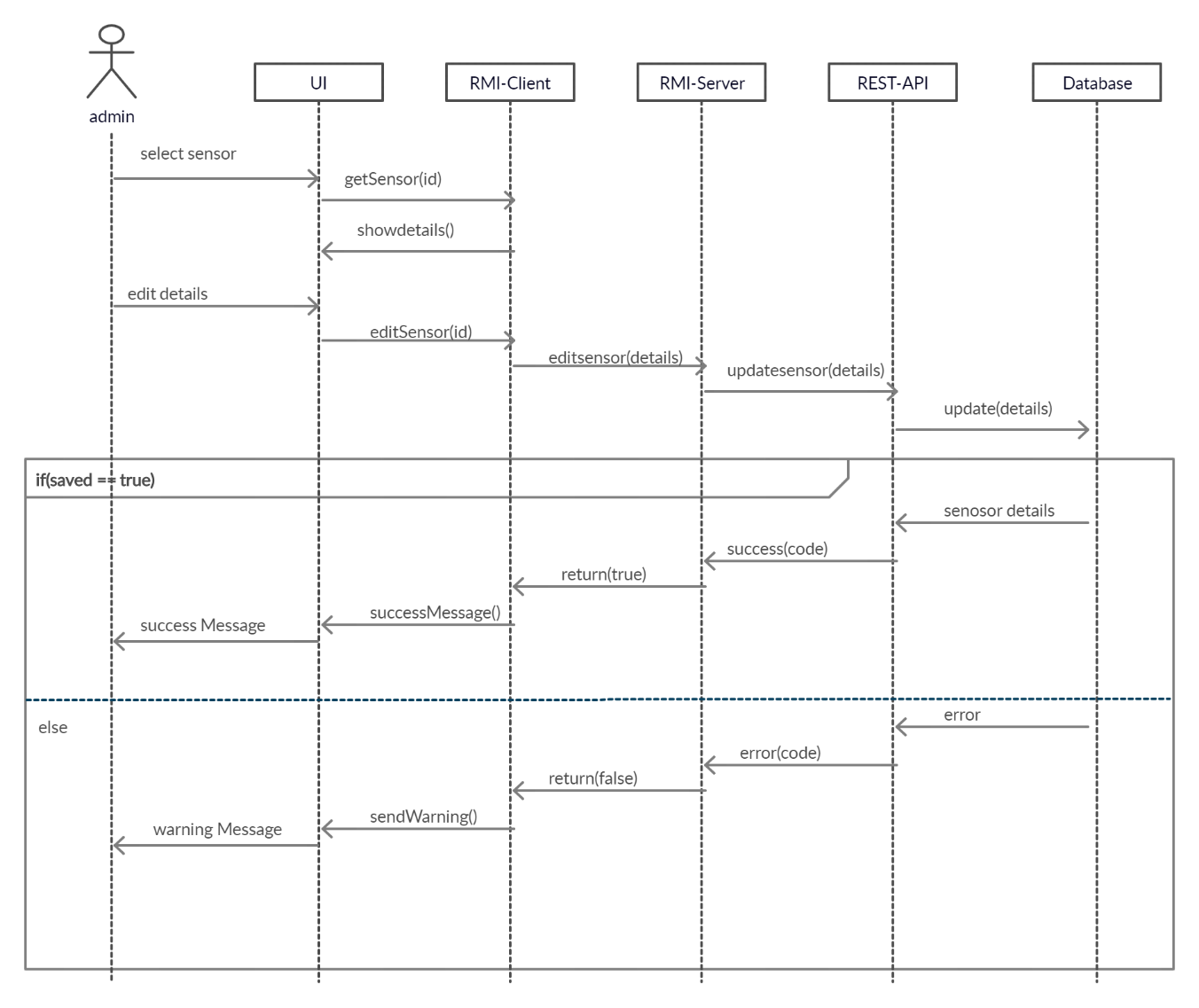




**Edit Sensor**

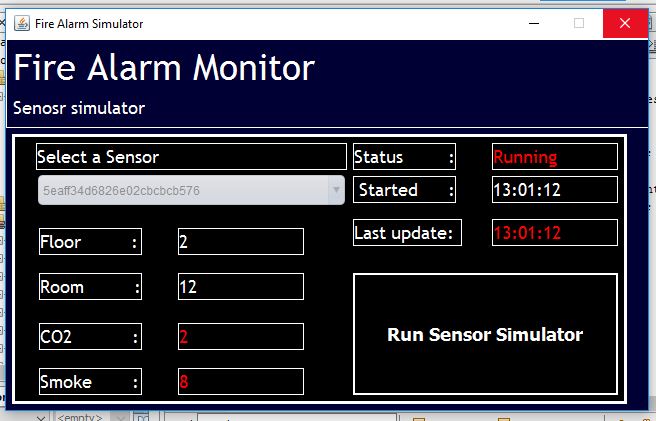
Admin can edit sensors by choosing sensor in the combobox and filling the necessary data.

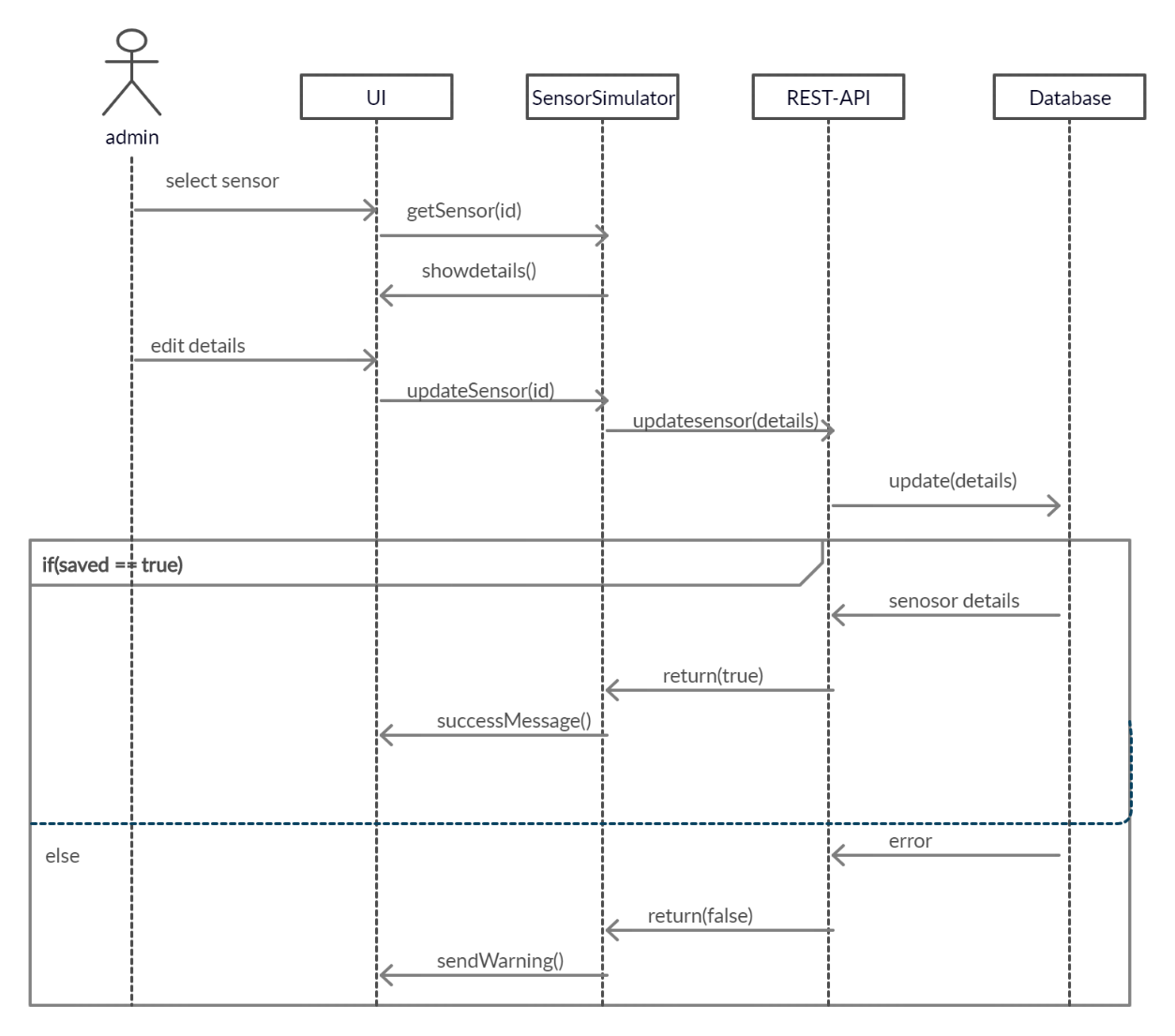
****See : [update\_sensor\_client](#update_sensor_client), [update\_sensor\_server](#update_sensor_server)



**Sensor Emulator**

Since we don’t have actual sensors we have to use this application to manipulate sensor data.

We can run one application for one sensor. Sensor data will updated in every 15 seconds. And UI will show the newly added data and last updated time.



Appendix

1. **Sensor service interface:**

|  |
| --- |
| package FireAlarm**;**  **import** java**.**rmi**.**Remote**;**  **import** java**.**rmi**.**RemoteException**;**  public interface SensorService **extends** Remote**{**  public boolean login**(**String name**,**String password**)** **throws** RemoteException**;**  public String getSensors**()** **throws** RemoteException**;**  public boolean updateSensor**(**String id**,**int room**,** int floor**)** **throws** RemoteException**;**  public boolean addSensor**(**int room**,** int floor**)** **throws** RemoteException**;**  public boolean sendEmail**(**String msg**)** **throws** RemoteException**;**  **}** |

1. **SensorServer -getsensors() – Get sensor data from database through Rest api**

|  |
| --- |
| @Override  public String getSensors**()** **throws** RemoteException **{**  String res **=** **null;**  **try** **{**  ArrayList**<**Sensor**>** sensors **=** **new** ArrayList**<**Sensor**>();**  CloseableHttpClient httpClient **=** HttpClients**.**createDefault**();**  HttpGet request **=** **new** HttpGet**(**"http://localhost:9000/api/sensors"**);**  CloseableHttpResponse response **=** httpClient**.**execute**(**request**);**  HttpEntity entity **=** response**.**getEntity**();**  res **=** EntityUtils**.**toString**(**entity**);**  System**.**out**.**println**(**res**);**  **return** res**;**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**ParseException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **return** res**;**  **}** |

1. **SensorServer –Main**

|  |
| --- |
| public static void main**(**String**[]** args**){**  System**.**setProperty**(**"java.security.policy"**,** "file:allowal.policy"**);**    **try{**  SensorServer svr **=** **new** SensorServer**();**  Registry registry **=** LocateRegistry**.**createRegistry**(**Registry**.**REGISTRY\_PORT**);**  registry**.**rebind**(**"SensorService"**,**svr**);**  System**.**out**.**println **(**"Service started...."**+** registry**);**    **}** **catch** **(**RemoteException ex**)** **{**  System**.**err**.**println**(**"ERROR1 "**+**ex**.**getMessage**());**  **}**  **}** |

1. **SensorServer – Add New Server**

|  |
| --- |
| //adding new sensor  @Override  public boolean addSensor**(**int room**,** int floor**)** **throws** RemoteException **{**  **try{**  //creating the api url  URL url **=** **new** URL **(**"http://localhost:9000/api/sensors"**);**  HttpURLConnection con **=** **(**HttpURLConnection**)**url**.**openConnection**();**  //setting methods and other details  con**.**setRequestMethod**(**"POST"**);**  con**.**setRequestProperty**(**"Content-Type"**,** "application/json; charset=utf-8"**);**  con**.**setRequestProperty**(**"Accept"**,** "application/json"**);**  con**.**setDoOutput**(true);**  //creating json body  String jsonInputString **=** "{\n" **+**  " \"status\":true,\n" **+**  " \"floorNo\":" **+**floor**+**",\n" **+**  " \"roomNo\" :" **+**room**+**",\n" **+**  " \"co2\":1,\n" **+**  " \"smoke\":1\n" **+**  "}"**;**    **try(**OutputStream os **=** con**.**getOutputStream**())** **{**  byte**[]** input **=** jsonInputString**.**getBytes**(**"utf-8"**);**  os**.**write**(**input**,** 0**,** input**.**length**);**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **try** **{**  **if(**con**.**getResponseCode**()** **==** 201**){**  **try(**BufferedReader br **=** **new** BufferedReader**(**  **new** InputStreamReader**(**con**.**getInputStream**(),** "utf-8"**)))** **{**  StringBuilder response **=** **new** StringBuilder**();**  String responseLine **=** **null;**  **while** **((**responseLine **=** br**.**readLine**())** **!=** **null)** **{**  response**.**append**(**responseLine**.**trim**());**  **}**  System**.**out**.**println**(**con**.**getResponseCode**());**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**    **return** **true;**  **}else{**  System**.**out**.**println**(**"Error can't add new sensor"**);**  **return** **false;**  **}**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**    **}** **catch** **(**MalformedURLException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**ProtocolException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **return** **false;**  **}** |

1. **SensorServer – updateSensor()**

|  |
| --- |
| //Updating sensor details  @Override  public boolean updateSensor**(**String id**,** int room**,** int floor**)** **throws** RemoteException **{**  **try** **{**  //Creating JSON Body  String data **=** "{\n" **+**  " \"floorNo\": "**+**floor**+**",\n" **+**  " \"roomNo\" :"**+**room**+**"\n" **+**  "}"**;**    HttpClient httpClient **=** **new** DefaultHttpClient**();**  // Prepare a request object  HttpUriRequest req **=** **new** HttpPatch**(**"http://localhost:9000/api/sensors/"**+**id**);**  req**.**setHeader**(**"Content-type"**,** "application/json"**);**  //Setting headers and content types  final StringEntity stringData **=** **new** StringEntity**(**data**.**toString**());**  **((**HttpPatch**)**req**).**setEntity**(**stringData**);**  HttpResponse execute **=** httpClient**.**execute**(**req**);**  //executing  **if(**execute**.**getStatusLine**().**getStatusCode**()** **==** 200**){**  **return** **true;**  //return true if status code is 200(200 is the code of success)  **}**  **}** **catch** **(**UnsupportedEncodingException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **return** **false;** //if code is not 200 or any other error happens, return false  **}** |

1. **SensorServer – Login()**

|  |
| --- |
| //Admin login  @Override  public boolean login**(**String name**,** String password**)** **throws** RemoteException **{**  **try{**  URL url **=** **new** URL **(**"http://localhost:9000/api/admin/login"**);**  HttpURLConnection con **=** **(**HttpURLConnection**)**url**.**openConnection**();**  con**.**setRequestMethod**(**"POST"**);**  con**.**setRequestProperty**(**"Content-Type"**,** "application/json; charset=utf-8"**);**  con**.**setRequestProperty**(**"Accept"**,** "application/json"**);**  con**.**setDoOutput**(true);**  String jsonInputString **=** "{\n" **+**  " \"email\":\""**+**name**+**"\",\n" **+**  " \"password\":\""**+**password**+**"\"\n" **+**  "}"**;**    **try(**OutputStream os **=** con**.**getOutputStream**())** **{**  byte**[]** input **=** jsonInputString**.**getBytes**(**"utf-8"**);**  os**.**write**(**input**,** 0**,** input**.**length**);**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **try** **{**  **if(**con**.**getResponseCode**()** **==** 200**){**  **try(**BufferedReader br **=** **new** BufferedReader**(**  **new** InputStreamReader**(**con**.**getInputStream**(),** "utf-8"**)))** **{**  StringBuilder response **=** **new** StringBuilder**();**  String responseLine **=** **null;**  **while** **((**responseLine **=** br**.**readLine**())** **!=** **null)** **{**  response**.**append**(**responseLine**.**trim**());**  **}**  System**.**out**.**println**(**con**.**getResponseCode**());**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**    **return** **true;**  //returning true if admin credentials valid  **}else{**  System**.**out**.**println**(**"Error can't log in"**);**  **return** **false;**  **}**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**    **}** **catch** **(**MalformedURLException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**ProtocolException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**IOException ex**)** **{**  Logger**.**getLogger**(**SensorServer**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **return** **false;**  **}** |

1. **SensorServer –SendEmail()**

|  |
| --- |
| @Override  public boolean sendEmail**(**String msg**)** **throws** RemoteException **{**  String user **=** "user@gmail.com"**;** // user email  String password **=** "\*\*\*\*\*\*\*\*\*"**;** // password for user    //Update above fields with gmail credentials and allow low security apps in gmail settings  //If you are using 2-step verification with your gmail, use gmail App password.    // Recipient's email ID needs to be mentioned.  String to **=** "97machan@gmail.com"**;**  // Sender's email ID needs to be mentioned  String from **=** "noreply.myzone@gmail.com"**;**  // Assuming you are sending email from through gmails smtp  String host **=** "smtp.gmail.com"**;**  // Get system properties  Properties properties **=** System**.**getProperties**();**  // Setup mail server  properties**.**put**(**"mail.smtp.host"**,** host**);**  properties**.**put**(**"mail.smtp.port"**,** "465"**);**  properties**.**put**(**"mail.smtp.ssl.enable"**,** "true"**);**  properties**.**put**(**"mail.smtp.auth"**,** "true"**);**  //authonticate mail and password  Session session **=** Session**.**getDefaultInstance**(**properties**,**  **new** javax**.**mail**.**Authenticator**()** **{**  protected PasswordAuthentication getPasswordAuthentication**()** **{**  **return** **new** PasswordAuthentication**(**user**,**password**);**  **}**  **});**  //Compose the message  **try** **{**  MimeMessage message **=** **new** MimeMessage**(**session**);**  message**.**setFrom**(new** InternetAddress**(**user**));**  message**.**addRecipient**(**Message**.**RecipientType**.**TO**,new** InternetAddress**(**to**));**  message**.**setSubject**(**"Fire Alarm Monitor"**);**    message**.**setText**(**"Hello \n These Sensors indicates high amount of Gas or CO2 levels \n" **+** msg**);**  //send the message  Transport**.**send**(**message**);**  System**.**out**.**println**(**"message sent successfully..."**);**  **}** **catch** **(**MessagingException e**)** **{**e**.**printStackTrace**();}**  **return** **false;**  **}** |

**RMI-Client (MainFrame.java)**

1. **Constructor()**

|  |
| --- |
| public static boolean logged **=** **false;**  //if admin logged in this will be true and admin have to login with every rmi-client  SensorService service **=** **null;**  ArrayList**<**Sensor**>** sensors**=** **new** ArrayList**<**Sensor**>();**  //Declaring and initializing SensorService and sensors array to use inside class  static String msg **=** ""**;**  //Message for Email  public MainFrame**()** **{**  initComponents**();**  System**.**setProperty**(**"java.security.policy"**,** "file:allowal.policy"**);**  //Initializing security policy  **try{**  Registry registry **=** LocateRegistry**.**getRegistry**(**"localhost"**);**  System**.**out**.**println**(**registry**);**  service **=** **(**SensorService**)** registry**.**lookup**(**"SensorService"**);**  //Getting server instance    sensorRetrieve**();**  //sendMail();  //Uncomment sendMail function to send emails  //Before uncomment Read Install.txt to config emailserver in SensorServer.java  //Retrieving sensor data  **}** **catch** **(**NotBoundException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}** **catch** **(**RemoteException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**    **}** |

1. **SendMail()**

|  |
| --- |
| public void sendMail**(){**  //change label to show that email server is running  jLabel16**.**setText**(**"Running"**);**    //timer function used for continues run with certain intervels  Timer timer **=** **new** Timer**();**  timer**.**scheduleAtFixedRate**(new** TimerTask**()** **{**  @Override  public void run**()** **{**  **try** **{**  java**.**util**.**Date date **=** **new** java**.**util**.**Date**();**  String**[]** d **=** date**.**toString**().**split**(**"\\s+"**);**  jLabel14**.**setText**(**d**[**3**]);**  //get latest time      **for(**Sensor s**:** sensors**){**  int co2 **=** s**.**co2**;**  int smoke **=** s**.**gas**;**  int room **=** s**.**roomNo**;**  int floor **=** s**.**floorNO**;**    **if(**co2 **>**5 **||** smoke**>**5**){**  //checking whether the sensor's smnoke or co2 is high  //add the sensor details to message  msg **=** msg **+** " floor : " **+** floor **+** "\tRoom:" **+** room **+** "\t\tco2 level: " **+** co2 **+** "\t\tSmoke level : " **+** smoke **+** " \n"**;**  **}**  **}**  service**.**sendEmail**(**msg**);** //sending email with msg data  msg **=** ""**;**  **}** **catch** **(**RemoteException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **}**  **},** 0**,** 15000**);** // Send email after every 15 seconds.  **}** |

1. **SensorRetrieve() – call Rmi server function every 30 seconds.**

|  |
| --- |
| //getting sensor data from rmi-server ->rest api  public ArrayList**<**Sensor**>** sensorRetrieve**(){**  Timer timer **=** **new** Timer**();**  timer**.**scheduleAtFixedRate**(new** TimerTask**()** **{**  @Override  public void run**()** **{**  **try** **{**  String ss **=** service**.**getSensors**();**  sensors **=** parse**(**ss**);**  //Patse string to Sensors array list  //Set data to the table  setData**();**  **}** **catch** **(**RemoteException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **}**  **},** 0**,** 30000**);** // updating sensor details after every 30 seconds      **return** sensors**;**  **}** |

1. **Button Navigation**

|  |
| --- |
| //jButton2 = Admin button  private void jButton2ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  **if(**logged**){**  //Check if admin is logged in  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    //if admin already logged in card view of jpanel 3 will show add sensor panel(jpanel6)  jPanel3**.**add**(**jPanel6**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}else{**  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    //if admin not logged in card view of jpanel3 will show the admin logging panel(jpanel5)  jPanel3**.**add**(**jPanel5**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}**    **}**    //jButton4 = edit event button  private void jButton4ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  **if(**logged**){**  //check admin is logged or not  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    DefaultComboBoxModel model **=** **(**DefaultComboBoxModel**)** jComboBox1**.**getModel**();**  //set sensor ids as combobox items  **for(**Sensor s **:** sensors**){**  model**.**addElement**(**s**.**id**);**  **}**    jPanel3**.**add**(**jPanel9**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}else{**  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    //if admin not logged in this will show login panel  jPanel3**.**add**(**jPanel5**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}**  **}**  private void jButton5ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  **if(**logged**){**  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    jPanel3**.**add**(**jPanel6**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}else{**  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    jPanel3**.**add**(**jPanel5**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}**  **}** |

1. **Admin login with button click**

|  |
| --- |
| //jButton3 = admin log in button  private void jButton3ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  **try** **{**  String un **=** jTextField1**.**getText**().**toString**();**  String pw **=** **new** String**(**jPasswordField1**.**getPassword**());**  //getting admin username and password from admin login panel    boolean x **=** service**.**login**(**un**,** pw**);**  //login admin with rest api  **if(**x**){**  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**    //if admin successfully logged in this will show add event panel  logged **=** **true;**  jPanel3**.**add**(**jPanel6**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}else{**  //if admin cannot log in successfully error message with show up  JOptionPane**.**showMessageDialog**(new** JFrame**(),**  "Username or password does not match"**,**  "Login Error"**,**  JOptionPane**.**WARNING\_MESSAGE**);**  //JOptionPane error message  **}**  **}** **catch** **(**RemoteException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **}** |

1. **Call** **addSensor by Button click action**

|  |
| --- |
| //jButton6 = add sensot button  private void jButton6ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  **try** **{**  int room**,** floor**;**  String r**,**f**;**  r **=** jTextField2**.**getText**().**toString**();**  f **=** jTextField3**.**getText**().**toString**();**  //getting sensor details to add new sensor    room **=** Integer**.**parseInt**(**r**);**  floor **=** Integer**.**parseInt**(**f**);**  //convert string to int for use in JSON  boolean add **=** service**.**addSensor**(**room**,** floor**);**  //addSensor is function in rmi server to add new sensor  **if(**add**){**//Check whether the new sensor is added or not  JOptionPane**.**showMessageDialog**(new** JFrame**(),**  "Successfully added"**,**  "Successfull"**,**  JOptionPane**.**WARNING\_MESSAGE**);**  //Successfull message  jTextField2**.**setText**(**""**);**  jTextField3**.**setText**(**""**);**    //if new sensor is successfully added, this will retrieve new sensors and set table data  sensors **=** parse**(**service**.**getSensors**());**  setData**();**  **}else{**  JOptionPane**.**showMessageDialog**(new** JFrame**(),**  "Cannot add new sensor Please try again"**,**  "Error"**,**  JOptionPane**.**WARNING\_MESSAGE**);**  //Error message - new sensor isn't added  **}**  **}** **catch** **(**RemoteException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**    **}** |

1. **Call** **updateSensor by button click action**

|  |
| --- |
| //Update sensor by admin  private void jButton7ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  String x **=** String**.**valueOf**(**jComboBox1**.**getSelectedItem**());**  //get selected sensor id from the combobox  int r **,** f**;**  f **=** Integer**.**parseInt**(**jTextField4**.**getText**());**  r **=** Integer**.**parseInt**(**jTextField5**.**getText**());**  //getting the new values for the sensor  **try** **{**  **if(**service**.**updateSensor**(**x**,** r**,** f**)){**  //updating sensor  JOptionPane**.**showMessageDialog**(new** JFrame**(),**  "Successfully Updated"**,**  "Successfull"**,**  JOptionPane**.**WARNING\_MESSAGE**);**  //if update is successfull this message will show up  jTextField4**.**setText**(**""**);**  jTextField5**.**setText**(**""**);**  //clearing text fields  sensors **=** parse**(**service**.**getSensors**());**  setData**();**  //retrieving and setting the table data with newly updated sensor details  **}else{**  JOptionPane**.**showMessageDialog**(new** JFrame**(),**  "Cannot update sensor Please try again"**,**  "Error: Update cannot be done"**,**  JOptionPane**.**WARNING\_MESSAGE**);**  //Error message will show up  **}**  **}** **catch** **(**RemoteException ex**)** **{**  Logger**.**getLogger**(**MainFrame**.**class**.**getName**()).**log**(**Level**.**SEVERE**,** **null,** ex**);**  **}**  **}** |

1. **Set Sensor details in jTable**

|  |
| --- |
| //set the sensor data into a table in order to easy understand  public void setData**(){**  jPanel3**.**removeAll**();**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  DefaultTableModel model **=** **(**DefaultTableModel**)** jTable1**.**getModel**();**  model**.**setRowCount**(**0**);**  ArrayList**<**Sensor**>** data **=** **new** ArrayList**<**Sensor**>();**  data **=** **this.**sensors**;**  //get sensors  Object rowData**[]** **=** **new** Object**[**5**];**  //declaring table data  **for(**int i **=** 0**;** i**<** data**.**size**();** i**++){**  rowData**[**0**]** **=** data**.**get**(**i**).**roomNo**;**  rowData**[**1**]** **=** data**.**get**(**i**).**floorNO**;**  rowData**[**2**]** **=** data**.**get**(**i**).**co2**;**  rowData**[**3**]** **=** data**.**get**(**i**).**gas**;**  rowData**[**4**]** **=** data**.**get**(**i**).**status**;**  model**.**addRow**(**rowData**);**  //initalizing and adding data to table  **}**  jTable1**.**setDefaultRenderer**(**String**.**class**,** **new** CustomTableRenderer**());**  jPanel3**.**add**(**jPanel4**);**  jPanel3**.**repaint**();**  jPanel3**.**revalidate**();**  **}** |

1. **Parse JSON type Api response to Sensor array**

|  |
| --- |
| **/**parsing json string sensor list  public static ArrayList**<**Sensor**>** parse**(**String response**){**  ArrayList **<**Sensor**>** snsrs **=** **new** ArrayList**<**Sensor**>();**  JSONObject sens **=** **new** JSONObject**(**response**);**  JSONArray sensors **=** sens**.**getJSONArray**(**"sensors"**);**  **for(**int i **=**0**;** i**<**sensors**.**length**();** i**++){**  JSONObject sensor **=** sensors**.**getJSONObject**(**i**);**  String id **=** sensor**.**getString**(**"id"**);**  int room **=** sensor**.**getInt**(**"roomNo"**);**  int floor **=** sensor**.**getInt**(**"floorNo"**);**  boolean status **=** sensor**.**getBoolean**(**"status"**);**  int co2 **=** sensor**.**getInt**(**"co2"**);**  int smoke **=** sensor**.**getInt**(**"smoke"**);**  //getting all items from response to make sensor object  Sensor s **=** **new** Sensor**(**id**,**status**,**room**,**floor**,**co2**,**smoke**);**  snsrs**.**add**(**s**);**  **}**  **return** snsrs**;**  **}** |

1. **Change Edit event details when combobox selected item change**

|  |
| --- |
| //Edit sensor combobox change action  private void jComboBox1ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  //get the selected item from combobox  String x **=** String**.**valueOf**(**jComboBox1**.**getSelectedItem**());**  int room **=** 0**,** floor **=** 0**;**    //find the room number and floor numebr of the selected sensor  ArrayList**<**Sensor**>** se **=** **this.**sensors**;**  **for(**Sensor s**:** se**){**  **if(**s**.**id **==** **null** **?** x **==** **null** **:** s**.**id**.**equals**(**x**)){**  room **=** s**.**roomNo**;**  floor **=** s**.**floorNO**;**  **}**  **}**  //set the room number and floor numebr of the selected sensor  jTextField4**.**setText**(**String**.**valueOf**(**floor**));**  jTextField5**.**setText**(**String**.**valueOf**(**room**));**  **}** |

1. **Sensor class**

|  |
| --- |
| package FireAlarm**;**  public class Sensor **{**  public String id**;**  public boolean status**;**  public int roomNo**;**  public int floorNO**;**  public int co2**;**  public int gas**;**    public Sensor **(**String id**,**boolean status**,**int r**,** int s**,** int d**,** int g**){**  **this.**id **=** id**;**  **this.**status **=** status**;**  **this.**roomNo **=** r**;**  **this.**floorNO **=** s**;**  **this.**co2 **=** d**;**  **this.**gas **=** g**;**  **}**  **}** |

**Sensor Emulator(Sensor demo)**

1. **Set combobox items**

|  |
| --- |
| public void setComboBox**(){**  DefaultComboBoxModel model **=** **(**DefaultComboBoxModel**)** jComboBox1**.**getModel**();**  //set sensor ids as combobox items    String response **=** getSensors**();**  //retrieving the sensor details  **if(**response **!=** **null){**  JSONObject sens **=** **new** JSONObject**(**response**);**  JSONArray sensors **=** sens**.**getJSONArray**(**"sensors"**);**    **for(**int i **=**0**;** i**<**sensors**.**length**();** i**++){**  JSONObject sensor **=** sensors**.**getJSONObject**(**i**);**  String id **=** sensor**.**getString**(**"id"**);**  model**.**addElement**(**id**);**  **}**  **}**  **}** |

1. **Update relevant data when combobox changed**

|  |
| --- |
| private void jComboBox1ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  String x **=** String**.**valueOf**(**jComboBox1**.**getSelectedItem**());**    String response **=** getSensors**();**  //retrieving the sensor details  **if(**response **!=** **null){**  JSONObject sens **=** **new** JSONObject**(**response**);**  JSONArray sensors **=** sens**.**getJSONArray**(**"sensors"**);**    **for(**int i **=**0**;** i**<**sensors**.**length**();** i**++){**  JSONObject sensor **=** sensors**.**getJSONObject**(**i**);**  String id **=** sensor**.**getString**(**"id"**);**  **if(**id **==** **null** **?** x **==** **null** **:** id**.**equals**(**x**)){**  int room **=** sensor**.**getInt**(**"roomNo"**);**  int floor **=** sensor**.**getInt**(**"floorNo"**);**  int co2 **=** sensor**.**getInt**(**"co2"**);**  int smoke **=** sensor**.**getInt**(**"smoke"**);**    jLabel18**.**setText**(**String**.**valueOf**(**room**));**  jLabel19**.**setText**(**String**.**valueOf**(**floor**));**  jLabel23**.**setText**(**String**.**valueOf**(**co2**));**  jLabel20**.**setText**(**String**.**valueOf**(**smoke**));**    **}**  **}**  **}**    **}** |

1. **Call update sensor in every 15 seconds**

|  |
| --- |
| private void jButton2ActionPerformed**(**java**.**awt**.**event**.**ActionEvent evt**)** **{**  //Disabling the combobox  jComboBox1**.**setEnabled**(false);**  //Disabling the run sensor button  jButton2**.**setText**(**"Running"**);**  jButton2**.**setEnabled**(false);**    //Indicate sensor simulator is running  jLabel4**.**setText**(**"Running"**);**  java**.**util**.**Date date **=** **new** java**.**util**.**Date**();**  String**[]** d **=** date**.**toString**().**split**(**"\\s+"**);**  jLabel6**.**setText**(**d**[**3**]);**  //indicate the started time.  String x **=** String**.**valueOf**(**jComboBox1**.**getSelectedItem**());**  Timer timer **=** **new** Timer**();**  timer**.**scheduleAtFixedRate**(new** TimerTask**()** **{**  @Override  public void run**()** **{**  java**.**util**.**Date date **=** **new** java**.**util**.**Date**();**  String**[]** d **=** date**.**toString**().**split**(**"\\s+"**);**  jLabel12**.**setText**(**d**[**3**]);**  //indicating the last updated time  updateSensor**(**x**);**    **}**  **},** 0**,** 15000**);** // Set time intervel in miliseconds.  **}** |

1. **Update sensor**

|  |
| --- |
| public boolean updateSensor**(**String id**){**  //Genarating random integers for co2 and smoke level simulation using java.util.Random  Random ran **=** **new** Random**();**  int x **=** ran**.**nextInt**(**10**)+**1**;** // (10)+1 will give the random number within 1 and 10  int y **=** ran**.**nextInt**(**10**)+**1**;**  boolean t **=** **true;**  **if(**x**<=**5 **&&** y**<=**5**){**  t **=** **false;**  **}**  jLabel23**.**setText**(**String**.**valueOf**(**x**));**  jLabel20**.**setText**(**String**.**valueOf**(**y**));**  //Displaying newest added co2 and smoke levels  **try** **{**  String data **=** "{\n" **+**  " \"status\":"**+**t**+**",\n"**+**  " \"co2\":"**+**x **+**",\n" **+**  " \"smoke\":"**+**y**+**"\n" **+**  "}"**;**  //JSON data for update sensor    HttpClient httpClient **=** **new** DefaultHttpClient**();**  // Prepare a request object  HttpUriRequest req **=** **new** HttpPatch**(**"http://localhost:9000/api/sensors/"**+**id**);**  // creating dynamic req with String id for update each sensor    req**.**setHeader**(**"Content-type"**,** "application/json"**);**  final StringEntity stringData **=** **new** StringEntity**(**data**);**  **((**HttpPatch**)**req**).**setEntity**(**stringData**);**  HttpResponse execute **=** httpClient**.**execute**(**req**);**    **if(**execute**.**getStatusLine**().**getStatusCode**()** **==** 200**){**  //status code 200 is return code for successful request  //returning true when statuscode is 200  **return** **true;**  **}**  **}** **catch** **(**UnsupportedEncodingException ex**)** **{**  System**.**out**.**println**(**ex**);**  **}** **catch** **(**IOException ex**)** **{**  System**.**out**.**println**(**ex**);**  **}**  **return** **false;**  //returning false because execution has been failed  **}**      //Get sensors from database Using REST api  public String getSensors**()** **{**  String res **=** **null;**  **try** **{**  CloseableHttpClient httpClient **=** HttpClients**.**createDefault**();**  HttpGet request **=** **new** HttpGet**(**"http://localhost:9000/api/sensors"**);**  CloseableHttpResponse response **=** httpClient**.**execute**(**request**);**  HttpEntity entity **=** response**.**getEntity**();**  res **=** EntityUtils**.**toString**(**entity**);**    **return** res**;**  **}** **catch** **(**IOException ex**)** **{**  System**.**out**.**println**(**"ERROR"**);**  **}**  **return** res**;**  **}** |

Web-client

1. **Sensor component**

|  |
| --- |
| *import* React *from* "react";  *import* alert1 *from* "../../assets/images/alert-1.png";  *import* alert2 *from* "../../assets/images/alert-2.png";  *import* alert3 *from* "../../assets/images/alert-3.png";  */\**  *\* sStatus - going through the below logic, it will assign a value*  *\* url - it will load the related image to the current status of sensor*  *\*/*  const Sensor = (props) => {    let sStatus;    let url;  *if* (!props.sensor.status) {      sStatus = "inactive";      url = alert3;    } *else* {  *if* (props.sensor.co2 > 5 || props.sensor.smoke > 5) {        sStatus = "danger";        url = alert1;      } *else* {        sStatus = "normal";        url = alert2;      }    }  *return* (  */\**  *\* Used the bacticks method to create all the css classNames according to the sensor status (sStatus)*  *\* Used Ternary Operator to handle all the logics*  *\* All other sensor details retrived using props.sensor*  *\*/*      <div className="col-lg-6 card-holder">        <div          className={`card ${            sStatus === "inactive"              ? "card--inactive"              : sStatus === "danger"              ? "card--fire"              : "card--normal"          }`}        >          <div className="card\_\_image-container">            <img src={url} alt="alert-sign" className="card\_\_image" />          </div>          <div className="card\_\_caption">            <h1 className="card\_\_name">              {props.sensor.status ? "Active" : "Inactive"}            </h1>              <h3 className="card\_\_type">{`${              sStatus === "inactive"                ? "Activate the sensor!"                : sStatus === "danger"                ? "Danger"                : "Normal"            }`}</h3>            <div className="card\_\_stats">              <div className="row">                <div className="col-6 card\_\_stats-title">Status</div>                <div className="col-6 card\_\_stats-value">                  {props.sensor.status ? "Active" : "Inactive"}                </div>              </div>              <div className="row">                <div className="col-6 card\_\_stats-title">Floor No</div>                <div className="col-6 card\_\_stats-value">                  {props.sensor.floorNo}                </div>              </div>              <div className="row">                <div className="col-6 card\_\_stats-title">Room No</div>                <div className="col-6 card\_\_stats-value">                  {props.sensor.roomNo}                </div>              </div>              <div className="row">                <div className="col-6 card\_\_stats-title">Smoke</div>                <div className="col-6 card\_\_stats-value">                  {`${sStatus==="inactive"?"\_\_": props.sensor.smoke}`}                </div>              </div>              <div className="row">                <div className="col-6 card\_\_stats-title">Co2</div>                <div className="col-6 card\_\_stats-value">{`${sStatus==="inactive"?"\_\_": props.sensor.co2}`}</div>              </div>            </div>            <div className="row">              <div className="col-6">                <div className="card\_\_ability ">                  <div className="row">                    <div className="col-12">                      <h4 className="card\_\_label">Smoke</h4>                    </div>                    <div className="col-12 circle-container">                      <div                        className={`card ${                          sStatus === "inactive"                            ? "yellow-circle"                            : props.sensor.smoke > 5                            ? "red-circle"                            : "green-circle"                        }`}                      ></div>                    </div>                  </div>                </div>              </div>              <div className="col-6">                <div className="card\_\_ability ">                  <div className="row">                    <div className="col-12">                      <h4 className="card\_\_label">Co2</h4>                    </div>                    <div className="col-12 circle-container">                    <div                        className={`card ${                          sStatus === "inactive"                            ? "yellow-circle"                            : props.sensor.co2 > 5                            ? "red-circle"                            : "green-circle"                        }`}                      ></div>                    </div>                  </div>                </div>              </div>            </div>          </div>        </div>      </div>    );  };  *export* *default* Sensor; |

1. **Sensorlist component**

|  |
| --- |
| *import* React *from* "react";  *import* Sensor *from* "../sensor/sensor.component";  *export* const SensorList = (props) => {  *return* (      <div className="row">        {*/\* Looping the array using map() function\*/*}        {props.array.map((mon) => (          <Sensor key={mon.id} sensor={mon}></Sensor> */\* Passing each array member to the Sensor component using props\*/*        ))}      </div>    );  }; |

1. **Client side appJs**

|  |
| --- |
| *import* React *from* "react";  *import* "./App.css";  *import* { SensorList } *from* "./components/sensor-list/sensor-list.component";  class App extends React.Component {    constructor() {      super();      this.state = {        sensors: [],      };    }      componentDidMount() {      this.getSensors();      this.timer=setInterval(() => this.getSensors(), 40000);*//this will call the function every 40 seconds(40000ms=40s)*      }    componentWillUnmount(){      clearInterval(this.timer);*//stop the calling, when component unmount*    }  */\**  *\* This getSensor() function will fetch data from the REST API*  *\* Then store the result in state->sensors[] array.*  *\*/*    getSensors(){      fetch("http://localhost:9000/api/sensors")        .then((res) => res.json())        .then((sen) => this.setState({ sensors: sen.sensors }));    }    render() {      const { sensors } = this.state;  *return* (        <div className="container">          <h1 className="app-title">Fire Alarm System</h1>          <SensorList array={sensors}></SensorList> {*/\* Pass the whole array to the SensorList component using props\*/*}        </div>      );    }  }  *export* *default* App; |

**Web Client Back end (REST API)**

1. **rest api APP.js**

|  |
| --- |
| const express = require("express");  const bodyParser = require("body-parser");  const mongoose = require("mongoose");  const alarmRoutes=require("./routes/fire-alarm-routes");  const adminRoutes=require("./routes/admin-routes");  const app = express();  app.use(bodyParser.json());*//extract the entire body portion of an incoming request.*  */\**  *\*this custom middleware use to solve the error when connecting the react.*  *\*without these settings browser will throw bunch of CORS Errors errors.*  *\*but Postman can still work without this middleware.*  *\*/*  app.use((req, res, next) => {    res.setHeader('Access-Control-Allow-Origin', '\*');    res.setHeader(      'Access-Control-Allow-Headers',      'Origin, X-Requested-With, Content-Type, Accept, Authorization'    );    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PATCH, DELETE');    next();  });  app.use('/api/sensors',alarmRoutes);  app.use('/api/admin',adminRoutes);  */\**  *\* This middle ware will handle all othe url requests excepts the ones which are mentioned above.*  *\* Those requset are invalid so we throw page not found with error code - 404*  *\*/*  app.use((req,res,next)=>{      const error=new Error('page not found! - Fire Sensor System');      error.code=404;      next(error);  });  */\* This middleware will handle the all errors which are throwen by other middlewares \*/*  app.use((error,req,res,next)=>{  *if* (res.headerSent) {  *return* next(error);      }      res.status(error.code||500).json({message:error.message || 'Unknown -- Error !!'})  });  */\**  *\* If Connection to the MongoDB success,*  *\* then it will start the node server on port 9000*  *\*/*  mongoose    .connect('mongodb+srv://crhunter:Pass4mongodb@cluster0-hqncx.mongodb.net/fire-alarm?retryWrites=true&w=majority')    .then(()=>{      app.listen(9000);      console.log('server & db are up and running!!!');    })    .catch((err) => {      console.log(err);    }); |

1. **admin routes**

|  |
| --- |
| const express = require("express");  const { check } = require("express-validator");*//to validate the requsts*  const router = express.Router();  const adminController = require("../controllers/admin-controller");  */\* signup route - 'POST' method \*/*  */\* validating requst info using chck() method of the express-validator package  \*/*  router.post(    "/signup",    check("name").not().isEmpty(),    check("email").normalizeEmail().isEmail(),    check("password").isLength({ min: 3 }),    adminController.signUp  );  router.post("/login", adminController.login);*//login route- 'POST'*  module.exports = router; |

1. **fire-alarm-routes**

|  |
| --- |
| const express = require("express");  const { check } = require("express-validator");  const router = express.Router();  const fireAlarmController = require("../controllers/fire-alarm-controller");    */\* create sensor route - 'POST' Method \*/*  */\* validating requst info using chck() method of the express-validator package \*/*  router.post(    "/",    check("status").not().isEmpty().isBoolean(),    check("floorNo").not().isEmpty().isInt(),    check("roomNo").not().isEmpty().isInt(),    check("co2").isInt({ min: 0, max: 10 }),    check("smoke").isInt({ min: 0, max: 10 }),    fireAlarmController.createSensor  );  router.get("/", fireAlarmController.getAllSensors);*//get all sensor route - 'GET' Method*  router.patch("/:sid", fireAlarmController.updateSensor);*//update sensor route - 'PATCH' Method*  module.exports = router; |

1. **Sensor.js**

|  |
| --- |
| const mongoose=require('mongoose');  const Schema=mongoose.Schema;  const sensorSchema=new Schema({      status:{type:Boolean,required:true},      floorNo:{type:Number,required:true},      roomNo:{type:Number,required:true},      co2:{type:Number,required:true,min:0,max:10},      smoke:{type:Number,required:true,min:0,max:10}  });  */\**  *\*This is where we converted the above Schema into a mongoose model*  *\*here,SensorSchema will be stored as a collection in the mongoDB.*  *\*Sensor->('sensors' will be the name of the collection.)*  *\*/*  module.exports=mongoose.model('Sensor',sensorSchema); |

1. **Admin.js**

|  |
| --- |
| const mongoose=require('mongoose');  const uniqueValidator=require('mongoose-unique-validator');*//to validate the emails*  const Schema=mongoose.Schema;  const adminSchema=new Schema({      name:{type:String,required:true},      email:{type:String,required:true,unique:true},*//this will create an seperate id for email. but to uniqe this email from the database we have to import the mongoose-unique-validator package.*      password:{type:String,required:true,minlength:3},  });  adminSchema.plugin(uniqueValidator);*// to add the unique behaviour to the email, have used mongoose-unique-validator package as mentioned above.*  */\**  *\*This is where we converted the above Schema into a mongoose model*  *\*here,AdminSchema will be stored as a collection in the mongoDB.*  *\*Admin->('admins' will be the name of the collection.)*  *\*/*  module.exports=mongoose.model('Admin',adminSchema); |

1. **Fire alarm controller**

|  |
| --- |
| const Sensor = require("../models/sensor");  const { validationResult } = require("express-validator"); *//to validate the request informations*  const createSensor = async (req, res, next) => {  */\* validating request using express validator package \*/*    const validationErrors = validationResult(req);  *if* (!validationErrors.isEmpty()) {      const error = new Error("Double check your inputs!!");      error.code = 422;  *return* next(error);    }  */\**  *\* Creating a newSensor using Sensor schema model.*  *\* Then saving it in mongoDB using save() query*  *\* if the query operation successfull, gives newly created sensor object as a response with code 201*  *\* otherwise it will throw a error.*  *\*/*    const { status, floorNo, roomNo, co2, smoke } = req.body;    const newSensor = new Sensor({      status,      floorNo,      roomNo,      co2,      smoke,    });  *try* {  *await* newSensor.save();    } *catch* (err) {  *return* next(err);    }    res.status(201).json({ sensor: newSensor.toObject({ getters: true }) });  };  */\**  *\* Retreving all the sensors using mongoose find() query.*  *\* if the query operation successfull, outputs the retrived sensor array using a map() as a response with code 200*  *\* otherwise it will throw a error with code 500.*  *\*/*  const getAllSensors = async (req, res, next) => {    let sensors;  *try* {      sensors = *await* Sensor.find();    } *catch* (err) {      const error = new Error(        "Could not retrive the sensors form the DB - database-error"      );      error.code = 500;  *return* next(error);    }    res      .status(200)      .json({ sensors: sensors.map((s) => s.toObject({ getters: true })) });  };  const updateSensor = async (req, res, next) => {  *//checks for any validations error.*    const validationErrors = validationResult(req);  *if* (!validationErrors.isEmpty()) {      const error = new Error("Please double check your inputs");      error.code = 422;  *return* next(error);    }    const { status, floorNo, roomNo, co2, smoke } = req.body; *//retrieves all the request data using object destructuring.*    */\**  *\* sId = retrives the passed user id from the url using params*  *\* Finds the given user id in the db to select the realted sensor using findById() method*  *\* then the user will store in selectedSensor variable.*  *\*/*    const sId = req.params.sid;    let selectedSensor;  *try* {      selectedSensor = *await* Sensor.findById(sId);    } *catch* (err) {      const error = new Error(        "Could not retrive the exact sensor details from the DB"      );      error.code = 500;  *return* next(error);    }  */\* filter the null inputs and only the valid ones will updated\*/*  *if* (status != null) {      selectedSensor.status = status;    }  *if* (co2 != null) {      selectedSensor.co2 = co2;    }  *if* (smoke != null) {      selectedSensor.smoke = smoke;    }  *if* (roomNo != null) {      selectedSensor.roomNo = roomNo;    }  *if* (floorNo != null) {      selectedSensor.floorNo = floorNo;    }  */\**  *\*After updating the selectedSensor object,*  *\*save the updated object in the db using save()*  *\*/*  *try* {  *await* selectedSensor.save();    } *catch* (err) {      const error = new Error("Could not save the updated sensor in db");      error.code = 500;  *return* next(error);    }    res.status(200).json({ place: selectedSensor.toObject({ getters: true }) });  };  exports.createSensor = createSensor;  exports.getAllSensors = getAllSensors;  exports.updateSensor = updateSensor; |

1. **admin controller**

|  |
| --- |
| const Admin=require("../models/admin");  const { validationResult } = require("express-validator");*//to validate the request informations*  const signUp = async (req, res, next) => {  */\* validating request using express validator package \*/*      const validationErrors = validationResult(req);  *if* (!validationErrors.isEmpty()) {        const error = new Error("Double check your inputs!!");        error.code = 422;  *return* next(error);      }  *//To find out if the email already exists in the db*      const { name,email,password } = req.body;      let hasRegistered;  *try*{          hasRegistered=*await* Admin.findOne({email:email});      }*catch*(err){        const error = new Error("Something went wrong on DB!");        error.code = 422;  *return* next(error);      }  *if*(hasRegistered){        const error = new Error("Email already in use, Try Login instead!");        error.code = 422;  *return* next(error);      }  */\**  *\* Creating a newAdmin using Admin schema model.*  *\* Then saving it in mongoDB using save() query*  *\* if the query operation successfull, gives newly created admin object as a response with code 201*  *\* otherwise it will throw a error.*  *\*/*      const newAdmin = new Admin({        name,        email,        password      });    *try* {  *await* newAdmin.save();      } *catch* (err) {  *return* next(err);      }      res.status(201).json({ admin: newAdmin.toObject({ getters: true }) });    };  const login=async(req,res,next)=>{      const {email,password}=req.body; *//retrieves request body data using object destructuring method*    */\**  *\* finds the given email is exists in the db using findOne() query*  *\* if the email exists it will match given password with the real password which stored in the DB*  *\* when operation success, gives outputs admins details as a response.*  *\*/*      let identifiedAdmin;  *try*{            identifiedAdmin= *await* Admin.findOne({email:email})      }*catch*(err){          const error = new Error("Something went wrong, when finding the admin's details from db");          error.code = 500;  *return* next(error);      }    *if*(!identifiedAdmin || identifiedAdmin.password !==password){          const error = new Error("Invalid Email or Password ");          error.code = 422;  *return* next(error);      }      res.json({message:"Admin logged in!!",user:identifiedAdmin.toObject({getters:true})});  }      exports.signUp=signUp;    exports.login=login; |