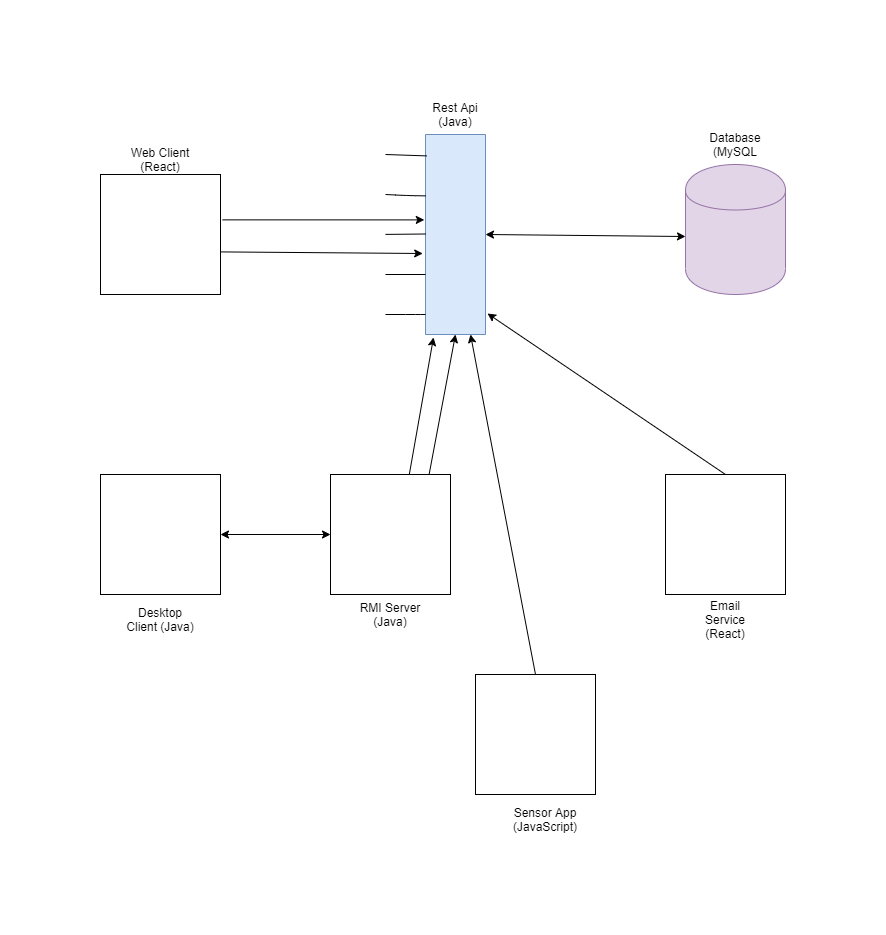
**Architectural Diagram**



**Service Interfaces Of Each Service**

REST API

There are two web services. One is to handle database operations regarding sensor details. This has three service interfaces.

1. getSensors ( )

This returns the id, location, CO2 level, smoke level and the status of each sensor object as JSON objects. This is used in the RMI server and the web client to get and display sensor details. This can be accessed by calling

[**http://localhost:8080/rest/webapi/sensors**](http://localhost:8080/rest/webapi/sensors).



1. addSensor (Sensor sensor)

This accepts a sensor object sent in JSON format as the input parameter and returns it after adding it to the database. This is used in the RMI server to add new sensors to the database. This can be accessed by calling [**http://localhost:8080/rest/webapi/sensors/add**](http://localhost:8080/rest/webapi/sensors/add).



1. updateSensor (Sensor sensor)

This accepts a sensor object sent in JSON format as the input parameter and returns it after updating its details in the database. This is used in the sensor applications to simulate the behavior of a fire alarm sensor. This can be accessed by calling [**http://localhost:8080/rest/webapi/sensors/update**](http://localhost:8080/rest/webapi/sensors/update).



The other service is used for the RMI server to notify the REST API whenever the CO2 level or smoke level is high and for the dummy email service to send emails.

1. getStatus ( )

This returns an email object as a JSON object with the value of sendEmail. This is used in the email service to send an email after checking whether the CO2 level or smoke level has gone up in any sensor. This can be accessed by calling [**http://localhost:8080/rest/webapi/emails**](http://localhost:8080/rest/webapi/emails).



1. updateStatus(Email email)

This accepts an email object sent in JSON format as the input parameter and returns it after updating its details in the database. This is used in the RMI server to notify the REST API whenever the CO2 level or the smoke level is high in any sensor. This can be accessed by calling [**http://localhost:8080/rest/webapi/emails/update**](http://localhost:8080/rest/webapi/emails/update).



**Sequence Diagram**

A screenshot of a cell phone

Description automatically generated

**Appendix**

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** DS\_Project;
8. **import** java.rmi.AccessException;
9. **import** java.rmi.NotBoundException;
10. **import** java.rmi.RemoteException;
11. **import** java.rmi.registry.LocateRegistry;
12. **import** java.rmi.registry.Registry;
13. **import** java.util.logging.Level;
14. **import** java.util.logging.Logger;
15. **import** DS\_Project.add\_or\_update;
17. /\*\*
18. \*
19. \* @author Kesara
20. \*/
21. **public** **class** Add **extends** javax.swing.JFrame {
23. /\*\*
24. \* Creates new form Add
25. \*/
26. **public** Add() {
27. initComponents();
28. }
30. /\*\*
31. \* This method is called from within the constructor to initialize the form.
32. \* WARNING: Do NOT modify this code. The content of this method is always
33. \* regenerated by the Form Editor.
34. \*/
35. @SuppressWarnings("unchecked")
36. // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
37. **private** **void** initComponents() {
39. addBtn = **new** javax.swing.JButton();
40. jTextFieldId = **new** javax.swing.JTextField();
41. jTextFieldLocation = **new** javax.swing.JTextField();
42. jButton1 = **new** javax.swing.JButton();
43. jLabel1 = **new** javax.swing.JLabel();
44. jLabel2 = **new** javax.swing.JLabel();
45. jLabel3 = **new** javax.swing.JLabel();
47. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
49. addBtn.setText("Add");
50. addBtn.addActionListener(**new** java.awt.event.ActionListener() {
51. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
52. addBtnActionPerformed(evt);
53. }
54. });
56. jTextFieldId.addActionListener(**new** java.awt.event.ActionListener() {
57. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
58. jTextFieldIdActionPerformed(evt);
59. }
60. });
62. jButton1.setText("Go Back");
63. jButton1.addActionListener(**new** java.awt.event.ActionListener() {
64. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
65. jButton1ActionPerformed(evt);
66. }
67. });
69. jLabel1.setText("Enter Sensor ID:");
71. jLabel2.setText("Sensor Location:");
73. jLabel3.setFont(**new** java.awt.Font("Tahoma", 1, 24)); // NOI18N
74. jLabel3.setText("Add New Sensors Here");
76. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
77. getContentPane().setLayout(layout);
78. layout.setHorizontalGroup(
79. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
80. .addGroup(layout.createSequentialGroup()
81. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
82. .addGroup(layout.createSequentialGroup()
83. .addGap(175, 175, 175)
84. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
85. .addComponent(jLabel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 112, javax.swing.GroupLayout.PREFERRED\_SIZE)
86. .addComponent(jLabel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 112, javax.swing.GroupLayout.PREFERRED\_SIZE))
87. .addGap(86, 86, 86)
88. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
89. .addComponent(jTextFieldLocation, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 269, javax.swing.GroupLayout.PREFERRED\_SIZE)
90. .addComponent(jTextFieldId, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 269, javax.swing.GroupLayout.PREFERRED\_SIZE)))
91. .addGroup(layout.createSequentialGroup()
92. .addGap(279, 279, 279)
93. .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 283, javax.swing.GroupLayout.PREFERRED\_SIZE))
94. .addGroup(layout.createSequentialGroup()
95. .addGap(251, 251, 251)
96. .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 121, javax.swing.GroupLayout.PREFERRED\_SIZE)
97. .addGap(69, 69, 69)
98. .addComponent(addBtn, javax.swing.GroupLayout.PREFERRED\_SIZE, 127, javax.swing.GroupLayout.PREFERRED\_SIZE)))
99. .addContainerGap(208, Short.MAX\_VALUE))
100. );
101. layout.setVerticalGroup(
102. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
103. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
104. .addGap(45, 45, 45)
105. .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 42, javax.swing.GroupLayout.PREFERRED\_SIZE)
106. .addGap(65, 65, 65)
107. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
108. .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 42, javax.swing.GroupLayout.PREFERRED\_SIZE)
109. .addComponent(jTextFieldId, javax.swing.GroupLayout.PREFERRED\_SIZE, 42, javax.swing.GroupLayout.PREFERRED\_SIZE))
110. .addGap(32, 32, 32)
111. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
112. .addComponent(jTextFieldLocation, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE)
113. .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE))
114. .addGap(45, 45, 45)
115. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
116. .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE)
117. .addComponent(addBtn, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE))
118. .addContainerGap(89, Short.MAX\_VALUE))
119. );
121. pack();
122. }// </editor-fold>//GEN-END:initComponents
124. //passing newly added sensor details to the server
125. **private** **void** addBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_addBtnActionPerformed
127. **try** {
128. Registry reg = LocateRegistry.getRegistry("localhost",1099);
129. RMI\_Interface server = (RMI\_Interface) reg.lookup("rmi://localhost/server");
131. **int** id = Integer.valueOf(jTextFieldId.getText());
132. **boolean** check = **false**;
133. String location = jTextFieldLocation.getText();
134. **int** smokeLvl = 0;
135. **int** co2Lvl = 0;

138. server.addSensors(id, check, location, smokeLvl, co2Lvl);

141. } **catch** (RemoteException ex) {
142. Logger.getLogger(Add.**class**.getName()).log(Level.SEVERE, **null**, ex);
143. } **catch** (NotBoundException ex) {
144. Logger.getLogger(Add.**class**.getName()).log(Level.SEVERE, **null**, ex);
145. }

148. }//GEN-LAST:event\_addBtnActionPerformed
150. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed
151. add\_or\_update back = **new** add\_or\_update();
152. back.setVisible(**true**);
153. }//GEN-LAST:event\_jButton1ActionPerformed
155. **private** **void** jTextFieldIdActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jTextFieldIdActionPerformed
156. // TODO add your handling code here:
157. }//GEN-LAST:event\_jTextFieldIdActionPerformed
159. /\*\*
160. \* @param args the command line arguments
161. \*/
162. **public** **static** **void** main(String args[]) {
163. /\* Set the Nimbus look and feel \*/
164. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
165. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
166. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
167. \*/
168. **try** {
169. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
170. **if** ("Nimbus".equals(info.getName())) {
171. javax.swing.UIManager.setLookAndFeel(info.getClassName());
172. **break**;
173. }
174. }
175. } **catch** (ClassNotFoundException ex) {
176. java.util.logging.Logger.getLogger(Add.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
177. } **catch** (InstantiationException ex) {
178. java.util.logging.Logger.getLogger(Add.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
179. } **catch** (IllegalAccessException ex) {
180. java.util.logging.Logger.getLogger(Add.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
181. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
182. java.util.logging.Logger.getLogger(Add.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
183. }
184. //</editor-fold>
186. /\* Create and display the form \*/
187. java.awt.EventQueue.invokeLater(**new** Runnable() {
188. **public** **void** run() {
189. **new** Add().setVisible(**true**);
190. }
191. });
192. }
194. // Variables declaration - do not modify//GEN-BEGIN:variables
195. **private** javax.swing.JButton addBtn;
196. **private** javax.swing.JButton jButton1;
197. **private** javax.swing.JLabel jLabel1;
198. **private** javax.swing.JLabel jLabel2;
199. **private** javax.swing.JLabel jLabel3;
200. **private** javax.swing.JTextField jTextFieldId;
201. **private** javax.swing.JTextField jTextFieldLocation;
202. // End of variables declaration//GEN-END:variables
203. }
204. /\*
205. \* To change this license header, choose License Headers in Project Properties.
206. \* To change this template file, choose Tools | Templates
207. \* and open the template in the editor.
208. \*/
209. **package** DS\_Project;
210. **import** DS\_Project.Update;
212. /\*\*
213. \*
214. \* @author Janith Perera
215. \*/
216. **public** **class** add\_or\_update **extends** javax.swing.JFrame {
218. /\*\*
219. \* Creates new form add\_or\_update
220. \*/
221. **public** add\_or\_update() {
222. initComponents();
223. }
225. /\*\*
226. \* This method is called from within the constructor to initialize the form.
227. \* WARNING: Do NOT modify this code. The content of this method is always
228. \* regenerated by the Form Editor.
229. \*/
230. @SuppressWarnings("unchecked")
231. // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
232. **private** **void** initComponents() {
234. jPanel1 = **new** javax.swing.JPanel();
235. jButton1 = **new** javax.swing.JButton();
236. jButton2 = **new** javax.swing.JButton();
237. jLabel1 = **new** javax.swing.JLabel();
239. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
240. setPreferredSize(**new** java.awt.Dimension(850, 450));
242. jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(**new** java.awt.Color(0, 0, 0), 5));
244. jButton1.setText("Add");
245. jButton1.addActionListener(**new** java.awt.event.ActionListener() {
246. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
247. jButton1ActionPerformed(evt);
248. }
249. });
251. jButton2.setText("Update");
252. jButton2.addActionListener(**new** java.awt.event.ActionListener() {
253. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
254. jButton2ActionPerformed(evt);
255. }
256. });
258. javax.swing.GroupLayout jPanel1Layout = **new** javax.swing.GroupLayout(jPanel1);
259. jPanel1.setLayout(jPanel1Layout);
260. jPanel1Layout.setHorizontalGroup(
261. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
262. .addGroup(jPanel1Layout.createSequentialGroup()
263. .addGap(296, 296, 296)
264. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
265. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
266. .addComponent(jButton1)
267. .addGap(10, 10, 10))
268. .addComponent(jButton2, javax.swing.GroupLayout.Alignment.TRAILING))
269. .addContainerGap(316, Short.MAX\_VALUE))
270. );
271. jPanel1Layout.setVerticalGroup(
272. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
273. .addGroup(jPanel1Layout.createSequentialGroup()
274. .addGap(78, 78, 78)
275. .addComponent(jButton1)
276. .addGap(54, 54, 54)
277. .addComponent(jButton2)
278. .addContainerGap(82, Short.MAX\_VALUE))
279. );
281. jLabel1.setFont(**new** java.awt.Font("Tahoma", 1, 36)); // NOI18N
282. jLabel1.setText("Admin Page");
284. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
285. getContentPane().setLayout(layout);
286. layout.setHorizontalGroup(
287. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
288. .addGroup(layout.createSequentialGroup()
289. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
290. .addGroup(layout.createSequentialGroup()
291. .addGap(62, 62, 62)
292. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
293. .addGroup(layout.createSequentialGroup()
294. .addGap(295, 295, 295)
295. .addComponent(jLabel1)))
296. .addContainerGap(93, Short.MAX\_VALUE))
297. );
298. layout.setVerticalGroup(
299. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
300. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
301. .addContainerGap(36, Short.MAX\_VALUE)
302. .addComponent(jLabel1)
303. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
304. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
305. .addGap(83, 83, 83))
306. );
308. pack();
309. }// </editor-fold>//GEN-END:initComponents
311. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed
312. Add add = **new** Add();
313. add.setVisible(**true**);
314. }//GEN-LAST:event\_jButton1ActionPerformed
316. **private** **void** jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton2ActionPerformed
317. Update update = **new** Update();
318. update.setVisible(**true**);
319. }//GEN-LAST:event\_jButton2ActionPerformed
321. /\*\*
322. \* @param args the command line arguments
323. \*/
324. **public** **static** **void** main(String args[]) {
325. /\* Set the Nimbus look and feel \*/
326. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
327. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
328. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
329. \*/
330. **try** {
331. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
332. **if** ("Nimbus".equals(info.getName())) {
333. javax.swing.UIManager.setLookAndFeel(info.getClassName());
334. **break**;
335. }
336. }
337. } **catch** (ClassNotFoundException ex) {
338. java.util.logging.Logger.getLogger(add\_or\_update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
339. } **catch** (InstantiationException ex) {
340. java.util.logging.Logger.getLogger(add\_or\_update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
341. } **catch** (IllegalAccessException ex) {
342. java.util.logging.Logger.getLogger(add\_or\_update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
343. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
344. java.util.logging.Logger.getLogger(add\_or\_update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
345. }
346. //</editor-fold>
348. /\* Create and display the form \*/
349. java.awt.EventQueue.invokeLater(**new** Runnable() {
350. **public** **void** run() {
351. **new** add\_or\_update().setVisible(**true**);
352. }
353. });
354. }
356. // Variables declaration - do not modify//GEN-BEGIN:variables
357. **private** javax.swing.JButton jButton1;
358. **private** javax.swing.JButton jButton2;
359. **private** javax.swing.JLabel jLabel1;
360. **private** javax.swing.JPanel jPanel1;
361. // End of variables declaration//GEN-END:variables
362. }
363. /\*
364. \* To change this license header, choose License Headers in Project Properties.
365. \* To change this template file, choose Tools | Templates
366. \* and open the template in the editor.
367. \*/
368. **package** DS\_Project;
370. **import** DS\_Project.Add;
371. **import** DS\_Project.HomePage;
372. **import** javax.swing.JOptionPane;
374. /\*\*
375. \*
376. \* @author Geetha
377. \*/
378. **public** **class** AdminLogin **extends** javax.swing.JFrame {
380. /\*\*
381. \* Creates new form AdminLogin
382. \*/
383. **public** AdminLogin() {
384. initComponents();
385. }
387. /\*\*
388. \* This method is called from within the constructor to initialize the form.
389. \* WARNING: Do NOT modify this code. The content of this method is always
390. \* regenerated by the Form Editor.
391. \*/
392. @SuppressWarnings("unchecked")
393. // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
394. **private** **void** initComponents() {
396. jPanel1 = **new** javax.swing.JPanel();
397. jLabel2 = **new** javax.swing.JLabel();
398. jLabel3 = **new** javax.swing.JLabel();
399. jPasswordField1 = **new** javax.swing.JPasswordField();
400. jTextField1 = **new** javax.swing.JTextField();
401. jButton1 = **new** javax.swing.JButton();
402. jButton2 = **new** javax.swing.JButton();
403. jButton3 = **new** javax.swing.JButton();
404. jLabel1 = **new** javax.swing.JLabel();
406. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
408. jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(**new** java.awt.Color(0, 0, 0), 5));
409. jPanel1.setPreferredSize(**new** java.awt.Dimension(800, 300));
411. jLabel2.setText("User Name");
413. jLabel3.setText("Password");
415. jPasswordField1.addActionListener(**new** java.awt.event.ActionListener() {
416. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
417. jPasswordField1ActionPerformed(evt);
418. }
419. });
421. jTextField1.addActionListener(**new** java.awt.event.ActionListener() {
422. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
423. jTextField1ActionPerformed(evt);
424. }
425. });
427. jButton1.setText("Login");
428. jButton1.addActionListener(**new** java.awt.event.ActionListener() {
429. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
430. jButton1ActionPerformed(evt);
431. }
432. });
434. jButton2.setText("Reset");
435. jButton2.addActionListener(**new** java.awt.event.ActionListener() {
436. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
437. jButton2ActionPerformed(evt);
438. }
439. });
441. jButton3.setText("Go Back");
442. jButton3.addActionListener(**new** java.awt.event.ActionListener() {
443. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
444. jButton3ActionPerformed(evt);
445. }
446. });
448. javax.swing.GroupLayout jPanel1Layout = **new** javax.swing.GroupLayout(jPanel1);
449. jPanel1.setLayout(jPanel1Layout);
450. jPanel1Layout.setHorizontalGroup(
451. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
452. .addGroup(jPanel1Layout.createSequentialGroup()
453. .addGap(145, 145, 145)
454. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
455. .addGroup(jPanel1Layout.createSequentialGroup()
456. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
457. .addComponent(jLabel3)
458. .addComponent(jLabel2))
459. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
460. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
461. .addComponent(jPasswordField1)
462. .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, 138, javax.swing.GroupLayout.PREFERRED\_SIZE)))
463. .addGroup(jPanel1Layout.createSequentialGroup()
464. .addComponent(jButton1)
465. .addGap(25, 25, 25)
466. .addComponent(jButton2)
467. .addGap(18, 18, 18)
468. .addComponent(jButton3)))
469. .addContainerGap(342, Short.MAX\_VALUE))
470. );
471. jPanel1Layout.setVerticalGroup(
472. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
473. .addGroup(jPanel1Layout.createSequentialGroup()
474. .addGap(54, 54, 54)
475. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
476. .addComponent(jLabel2)
477. .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
478. .addGap(18, 18, 18)
479. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
480. .addComponent(jLabel3)
481. .addComponent(jPasswordField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
482. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 73, Short.MAX\_VALUE)
483. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
484. .addComponent(jButton1)
485. .addComponent(jButton2)
486. .addComponent(jButton3))
487. .addGap(60, 60, 60))
488. );
490. jLabel1.setFont(**new** java.awt.Font("Tahoma", 0, 36)); // NOI18N
491. jLabel1.setText("Welcome to Admin Login");
493. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
494. getContentPane().setLayout(layout);
495. layout.setHorizontalGroup(
496. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
497. .addGroup(layout.createSequentialGroup()
498. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
499. .addGroup(layout.createSequentialGroup()
500. .addGap(45, 45, 45)
501. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 745, javax.swing.GroupLayout.PREFERRED\_SIZE))
502. .addGroup(layout.createSequentialGroup()
503. .addGap(204, 204, 204)
504. .addComponent(jLabel1)))
505. .addContainerGap(60, Short.MAX\_VALUE))
506. );
507. layout.setVerticalGroup(
508. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
509. .addGroup(layout.createSequentialGroup()
510. .addGap(11, 11, 11)
511. .addComponent(jLabel1)
512. .addGap(18, 18, 18)
513. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 284, javax.swing.GroupLayout.PREFERRED\_SIZE)
514. .addContainerGap(93, Short.MAX\_VALUE))
515. );
517. pack();
518. }// </editor-fold>//GEN-END:initComponents
520. **private** **void** jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jTextField1ActionPerformed
522. }//GEN-LAST:event\_jTextField1ActionPerformed
524. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed
525. String username = jTextField1.getText();
526. String password = jPasswordField1.getText();
528. **if**(username.equals("admin") && password.equals("admin")){
529. JOptionPane.showMessageDialog(**this**, "Login Successfull! redirecting to admin page, press 'OK' to continue");
530. add\_or\_update addupdate = **new** add\_or\_update();
531. addupdate.setVisible(**true**);
532. }
533. **else** **if**(username.equals("") || password.equals("")){
534. JOptionPane.showMessageDialog(**this**, "Please fill up both Username & Password");
535. }
536. **else**{
537. JOptionPane.showMessageDialog(**this**, "invalid login please try again!");
538. }
539. }//GEN-LAST:event\_jButton1ActionPerformed
541. **private** **void** jPasswordField1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jPasswordField1ActionPerformed
543. }//GEN-LAST:event\_jPasswordField1ActionPerformed
545. **private** **void** jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton2ActionPerformed
546. jTextField1.setText(**null**);
547. jPasswordField1.setText(**null**);
548. }//GEN-LAST:event\_jButton2ActionPerformed
550. **private** **void** jButton3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton3ActionPerformed
551. HomePage home = **new** HomePage();
552. home.setVisible(**true**);
553. }//GEN-LAST:event\_jButton3ActionPerformed
555. /\*\*
556. \* @param args the command line arguments
557. \*/
558. **public** **static** **void** main(String args[]) {
559. /\* Set the Nimbus look and feel \*/
560. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
561. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
562. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
563. \*/
564. **try** {
565. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
566. **if** ("Nimbus".equals(info.getName())) {
567. javax.swing.UIManager.setLookAndFeel(info.getClassName());
568. **break**;
569. }
570. }
571. } **catch** (ClassNotFoundException ex) {
572. java.util.logging.Logger.getLogger(AdminLogin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
573. } **catch** (InstantiationException ex) {
574. java.util.logging.Logger.getLogger(AdminLogin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
575. } **catch** (IllegalAccessException ex) {
576. java.util.logging.Logger.getLogger(AdminLogin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
577. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
578. java.util.logging.Logger.getLogger(AdminLogin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
579. }
580. //</editor-fold>
582. /\* Create and display the form \*/
583. java.awt.EventQueue.invokeLater(**new** Runnable() {
584. **public** **void** run() {
585. **new** AdminLogin().setVisible(**true**);
586. }
587. });
588. }
590. // Variables declaration - do not modify//GEN-BEGIN:variables
591. **private** javax.swing.JButton jButton1;
592. **private** javax.swing.JButton jButton2;
593. **private** javax.swing.JButton jButton3;
594. **private** javax.swing.JLabel jLabel1;
595. **private** javax.swing.JLabel jLabel2;
596. **private** javax.swing.JLabel jLabel3;
597. **private** javax.swing.JPanel jPanel1;
598. **private** javax.swing.JPasswordField jPasswordField1;
599. **private** javax.swing.JTextField jTextField1;
600. // End of variables declaration//GEN-END:variables
601. }
602. /\*
603. \* To change this license header, choose License Headers in Project Properties.
604. \* To change this template file, choose Tools | Templates
605. \* and open the template in the editor.
606. \*/
607. **package** DS\_Project;
609. **import** java.awt.Dimension;
610. **import** java.awt.Insets;
611. **import** java.awt.Toolkit;
612. **import** java.rmi.NotBoundException;
613. **import** java.rmi.RemoteException;
614. **import** java.rmi.registry.LocateRegistry;
615. **import** java.rmi.registry.Registry;
616. **import** java.util.List;
617. **import** java.util.Timer;
618. **import** java.util.TimerTask;
619. **import** java.util.logging.Level;
620. **import** java.util.logging.Logger;
621. **import** javax.swing.table.DefaultTableModel;
623. /\*\*
624. \*
625. \* @author ASUS
626. \*/
627. **public** **class** Alert **extends** javax.swing.JFrame {
629. /\*\*
630. \* Creates new form Alert
631. \*/
632. **public** Alert() {
633. initComponents();
635. start();
637. setAlwaysOnTop(**true**);
638. Insets toolHeight = Toolkit.getDefaultToolkit().getScreenInsets(getGraphicsConfiguration());
639. Dimension scrSize = Toolkit.getDefaultToolkit().getScreenSize();
641. **new** Thread(**new** Runnable() {
642. @Override
643. **public** **void** run() {
645. **for**(**int** i=0; i < getHeight(); i++){
646. setLocation(scrSize.width - getWidth(), scrSize.height - toolHeight.bottom - i);
647. **try** {
648. Thread.sleep(2);
649. } **catch** (InterruptedException ex) {
650. Logger.getLogger(Alert.**class**.getName()).log(Level.SEVERE, **null**, ex);
651. }
652. }
653. }
654. }).start();
656. **try**{
657. getAlert();
659. }
660. **catch**(RemoteException e){
661. System.out.println(e.getMessage());
662. }
663. }
665. //RMI client get the sensor status every 15 second
666. Timer timer = **new** Timer();
668. TimerTask task = **new** TimerTask() {
669. @Override
670. **public** **void** run() {
671. **try** {
672. jTableAlert.setModel(**new** DefaultTableModel(**null**, **new** Object[]{"ID", "Active/Not Active", "Location", "Smoke Level", "CO2 Level"}));
673. getAlert();
675. } **catch** (Exception e) {
676. System.out.println(e.getMessage());
677. }
679. }
680. };

683. **public** **void** start() {
684. timer.scheduleAtFixedRate(task, 20000, 20000);
685. }
687. /\*\*
688. \* This method is called from within the constructor to initialize the form.
689. \* WARNING: Do NOT modify this code. The content of this method is always
690. \* regenerated by the Form Editor.
691. \*/
692. @SuppressWarnings("unchecked")
693. // <editor-fold defaultstate="collapsed" desc="Generated Code">
694. **private** **void** initComponents() {
696. jLabel1 = **new** javax.swing.JLabel();
697. jButton1 = **new** javax.swing.JButton();
698. jScrollPane1 = **new** javax.swing.JScrollPane();
699. jTableAlert = **new** javax.swing.JTable();
701. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
703. jLabel1.setFont(**new** java.awt.Font("Tahoma", 1, 18)); // NOI18N
704. jLabel1.setText("ALERT");
706. jButton1.setText("OK");
707. jButton1.addActionListener(**new** java.awt.event.ActionListener() {
708. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
709. jButton1ActionPerformed(evt);
710. }
711. });
713. jTableAlert.setModel(**new** javax.swing.table.DefaultTableModel(
714. **new** Object [][] {
716. },
717. **new** String [] {
718. "ID", "Active/Not Active", "Location", "Smoke Level", "CO2 Level"
719. }
720. ));
721. jScrollPane1.setViewportView(jTableAlert);
723. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
724. getContentPane().setLayout(layout);
725. layout.setHorizontalGroup(
726. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
727. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
728. .addGap(0, 32, Short.MAX\_VALUE)
729. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 603, javax.swing.GroupLayout.PREFERRED\_SIZE)
730. .addGap(24, 24, 24))
731. .addGroup(layout.createSequentialGroup()
732. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
733. .addGroup(layout.createSequentialGroup()
734. .addGap(258, 258, 258)
735. .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 119, javax.swing.GroupLayout.PREFERRED\_SIZE))
736. .addGroup(layout.createSequentialGroup()
737. .addGap(291, 291, 291)
738. .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 66, javax.swing.GroupLayout.PREFERRED\_SIZE)))
739. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
740. );
741. layout.setVerticalGroup(
742. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
743. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
744. .addContainerGap()
745. .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 51, javax.swing.GroupLayout.PREFERRED\_SIZE)
746. .addGap(18, 18, 18)
747. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 282, javax.swing.GroupLayout.PREFERRED\_SIZE)
748. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 36, Short.MAX\_VALUE)
749. .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 35, javax.swing.GroupLayout.PREFERRED\_SIZE)
750. .addGap(22, 22, 22))
751. );
753. pack();
754. }// </editor-fold>
756. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
758. **new** Thread(**new** Runnable() {
759. @Override
760. **public** **void** run() {
762. dispose();
763. **try** {
764. Thread.sleep(10);
765. } **catch** (InterruptedException ex) {
766. Logger.getLogger(Alert.**class**.getName()).log(Level.SEVERE, **null**, ex);
767. }
769. }
770. }).start();
772. }
774. /\*\*
775. \* @param args the command line arguments
776. \*/
778. **public** **void** getAlert() **throws** RemoteException{
779. **try** {
780. Registry reg = LocateRegistry.getRegistry("localhost",1099);
781. RMI\_Interface server = (RMI\_Interface) reg.lookup("rmi://localhost/server");
783. List<Sensors> list = server.displayAlert();
785. DefaultTableModel tableModel = (DefaultTableModel) jTableAlert.getModel();
787. Object[] row = **new** Object[5];
789. **for**(Sensors s : list){
790. row[0] = s.getId();
791. row[1] = s.isActive();
792. row[2] = s.getLocation();
793. row[3] = s.getSmokeLvl();
794. row[4] = s.getCO2Lvl();
796. tableModel.addRow(row);
797. }
799. }  **catch**(RemoteException | NotBoundException e) {
800. System.out.println(e.getMessage());
801. }
802. }
804. **public** **static** **void** main(String args[]) {
805. /\* Set the Nimbus look and feel \*/
806. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
807. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
808. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
809. \*/
810. **try** {
811. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
812. **if** ("Nimbus".equals(info.getName())) {
813. javax.swing.UIManager.setLookAndFeel(info.getClassName());
814. **break**;
815. }
816. }
817. } **catch** (ClassNotFoundException ex) {
818. java.util.logging.Logger.getLogger(Alert.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
819. } **catch** (InstantiationException ex) {
820. java.util.logging.Logger.getLogger(Alert.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
821. } **catch** (IllegalAccessException ex) {
822. java.util.logging.Logger.getLogger(Alert.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
823. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
824. java.util.logging.Logger.getLogger(Alert.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
825. }
826. //</editor-fold>
828. /\* Create and display the form \*/
829. java.awt.EventQueue.invokeLater(**new** Runnable() {
830. **public** **void** run() {
831. **new** Alert().setVisible(**true**);
832. }
833. });

836. }
838. // Variables declaration - do not modify
839. **private** javax.swing.JButton jButton1;
840. **private** javax.swing.JLabel jLabel1;
841. **private** javax.swing.JScrollPane jScrollPane1;
842. **private** javax.swing.JTable jTableAlert;
843. // End of variables declaration
844. }
845. /\*
846. \* To change this license header, choose License Headers in Project Properties.
847. \* To change this template file, choose Tools | Templates
848. \* and open the template in the editor.
849. \*/
850. **package** DS\_Project;
852. /\*\*
853. \*
854. \* @author ACER
855. \*/
856. **public** **class** Emails {
857. **private** **boolean** sendEmail;
859. **public** Emails(**boolean** sendEmail){
860. **this**.sendEmail = sendEmail;
861. }
863. **public** **boolean** isSendEmail() {
864. **return** sendEmail;
865. }
867. **public** **void** setSEndEmail(**boolean** sendEmail) {
868. **this**.sendEmail = sendEmail;
869. }
870. }
871. /\*
872. \* To change this license header, choose License Headers in Project Properties.
873. \* To change this template file, choose Tools | Templates
874. \* and open the template in the editor.
875. \*/
876. **package** DS\_Project;
878. **import** DS\_Project.AdminLogin;
879. **import** DS\_Project.viewSensors;
880. **import** java.rmi.NotBoundException;
881. **import** java.rmi.RemoteException;
882. **import** java.rmi.registry.LocateRegistry;
883. **import** java.rmi.registry.Registry;
884. **import** java.util.List;
885. **import** java.util.Timer;
886. **import** java.util.TimerTask;
887. **import** javax.swing.table.DefaultTableModel;
889. /\*\*
890. \*
891. \* @author Geetha
892. \*/
893. **public** **class** HomePage **extends** javax.swing.JFrame {
895. /\*\*
896. \* Creates new form HomePage
897. \*/
898. **public** HomePage() {
899. initComponents();
901. display();
903. start();

906. }
908. Timer timer = **new** Timer();
909. TimerTask task = **new** TimerTask() {
910. @Override
911. **public** **void** run() {
912. **try** {
914. display();


918. } **catch** (Exception e) {
919. System.out.println(e.getMessage());
920. }


924. }
925. };

928. **public** **void** start() {
929. timer.scheduleAtFixedRate(task, 15000, 15000);
930. }
932. **public** **void** display(){
933. **try** {
934. Registry reg = LocateRegistry.getRegistry("localhost",1099);
935. RMI\_Interface server = (RMI\_Interface) reg.lookup("rmi://localhost/server");
937. List<Sensors> list = server.displayAlert();
938. **int** lenght = list.size();
939. **if**(lenght > 0){
940. Alert alert = **new** Alert();
941. alert.setVisible(**true**);
942. }
944. }  **catch**(RemoteException | NotBoundException e) {
945. System.out.println(e.getMessage());
946. }
947. }
949. /\*\*
950. \* This method is called from within the constructor to initialize the form.
951. \* WARNING: Do NOT modify this code. The content of this method is always
952. \* regenerated by the Form Editor.
953. \*/
954. @SuppressWarnings("unchecked")
955. // <editor-fold defaultstate="collapsed" desc="Generated Code">
956. **private** **void** initComponents() {
958. jPanel1 = **new** javax.swing.JPanel();
959. jButton1 = **new** javax.swing.JButton();
960. jButton2 = **new** javax.swing.JButton();
961. jLabel1 = **new** javax.swing.JLabel();
963. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
965. jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(**new** java.awt.Color(0, 0, 0), 5));
967. jButton1.setText("Admin Login");
968. jButton1.addActionListener(**new** java.awt.event.ActionListener() {
969. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
970. jButton1ActionPerformed(evt);
971. }
972. });
974. jButton2.setText("View Sensor Details");
975. jButton2.addActionListener(**new** java.awt.event.ActionListener() {
976. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
977. jButton2ActionPerformed(evt);
978. }
979. });
981. javax.swing.GroupLayout jPanel1Layout = **new** javax.swing.GroupLayout(jPanel1);
982. jPanel1.setLayout(jPanel1Layout);
983. jPanel1Layout.setHorizontalGroup(
984. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
985. .addGroup(jPanel1Layout.createSequentialGroup()
986. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
987. .addGroup(jPanel1Layout.createSequentialGroup()
988. .addGap(195, 195, 195)
989. .addComponent(jButton2))
990. .addGroup(jPanel1Layout.createSequentialGroup()
991. .addGap(216, 216, 216)
992. .addComponent(jButton1)))
993. .addContainerGap(195, Short.MAX\_VALUE))
994. );
995. jPanel1Layout.setVerticalGroup(
996. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
997. .addGroup(jPanel1Layout.createSequentialGroup()
998. .addGap(69, 69, 69)
999. .addComponent(jButton1)
1000. .addGap(45, 45, 45)
1001. .addComponent(jButton2)
1002. .addContainerGap(94, Short.MAX\_VALUE))
1003. );
1005. jLabel1.setFont(**new** java.awt.Font("Tahoma", 0, 36)); // NOI18N
1006. jLabel1.setText("Home");
1008. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
1009. getContentPane().setLayout(layout);
1010. layout.setHorizontalGroup(
1011. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1012. .addGroup(layout.createSequentialGroup()
1013. .addGap(137, 137, 137)
1014. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1015. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
1016. .addGroup(layout.createSequentialGroup()
1017. .addGap(224, 224, 224)
1018. .addComponent(jLabel1)))
1019. .addContainerGap(166, Short.MAX\_VALUE))
1020. );
1021. layout.setVerticalGroup(
1022. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1023. .addGroup(layout.createSequentialGroup()
1024. .addGap(33, 33, 33)
1025. .addComponent(jLabel1)
1026. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
1027. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
1028. .addContainerGap(92, Short.MAX\_VALUE))
1029. );
1031. pack();
1032. }// </editor-fold>
1034. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
1035. AdminLogin adminLogin = **new** AdminLogin();
1036. adminLogin.setVisible(**true**);
1037. }
1039. **private** **void** jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
1040. viewSensors sensorDetails = **new** viewSensors();
1041. sensorDetails.setVisible(**true**);
1042. }
1044. /\*\*
1045. \* @param args the command line arguments
1046. \*/
1047. **public** **static** **void** main(String args[]) {
1048. /\* Set the Nimbus look and feel \*/
1049. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
1050. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
1051. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
1052. \*/
1053. **try** {
1054. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
1055. **if** ("Nimbus".equals(info.getName())) {
1056. javax.swing.UIManager.setLookAndFeel(info.getClassName());
1057. **break**;
1058. }
1059. }
1060. } **catch** (ClassNotFoundException ex) {
1061. java.util.logging.Logger.getLogger(HomePage.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1062. } **catch** (InstantiationException ex) {
1063. java.util.logging.Logger.getLogger(HomePage.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1064. } **catch** (IllegalAccessException ex) {
1065. java.util.logging.Logger.getLogger(HomePage.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1066. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
1067. java.util.logging.Logger.getLogger(HomePage.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1068. }
1069. //</editor-fold>
1071. /\* Create and display the form \*/
1072. java.awt.EventQueue.invokeLater(**new** Runnable() {
1073. **public** **void** run() {
1074. **new** HomePage().setVisible(**true**);
1075. }
1076. });
1077. }
1079. // Variables declaration - do not modify
1080. **private** javax.swing.JButton jButton1;
1081. **private** javax.swing.JButton jButton2;
1082. **private** javax.swing.JLabel jLabel1;
1083. **private** javax.swing.JPanel jPanel1;
1084. // End of variables declaration
1085. }
1086. /\*
1087. \* To change this license header, choose License Headers in Project Properties.
1088. \* To change this template file, choose Tools | Templates
1089. \* and open the template in the editor.
1090. \*/
1091. **package** DS\_Project;
1093. **import** java.rmi.Remote;
1094. **import** java.rmi.RemoteException;
1095. **import** java.util.List;
1097. /\*\*
1098. \*
1099. \* @author Kesara
1100. \*/
1101. **public** **interface** RMI\_Interface **extends** Remote{
1103. //declare a method to get all the sensor details at 15 seconds intervals
1105. /\*\*
1106. \*
1107. \* @return
1108. \* @throws RemoteException
1109. \*/
1110. **public** List<Sensors> getAllSensors() **throws** RemoteException;
1112. //declare a method to add sensors
1113. **public** **void** addSensors(**int** id, **boolean** active, String location, **int** smokeLevel, **int** co2Level) **throws** RemoteException;
1115. //decalre a method to update sensors
1116. **public** **void** updateSensors(**int** id, **boolean** active, String location, **int** smokeLevel, **int** co2Level) **throws** RemoteException;
1118. //declare a method to display an alert
1119. **public** List<Sensors> displayAlert() **throws** RemoteException;
1121. //declare a method to send status of sensors regarding CO2 and smoke levels
1122. **public** **void** sendStatus() **throws** RemoteException;
1123. }
1124. **package** DS\_Project;
1126. **import** java.io.Serializable;
1128. //create sensor attributes and setters and getters
1130. **public** **class** Sensors **implements** Serializable{
1132. **private** **boolean** active;
1133. **private** String location;
1134. **private** **int** smokeLvl;
1135. **private** **int** CO2Lvl;
1136. **private** **int** id;
1138. **public** Sensors(**int** id,**boolean** active, String location, **int** smokeLvl, **int** CO2Lvl){
1139. **this**.id = id;
1140. **this**.active = active;
1141. **this**.location = location;
1142. **this**.smokeLvl = smokeLvl;
1143. **this**.CO2Lvl = CO2Lvl;

1146. }
1148. **public** Sensors(**int** id,String location){
1149. **this**.id = id;
1150. **this**.location = location;
1151. }

1154. **public** **int** getId() {
1155. **return** id;
1156. }
1158. **public** **void** setId(**int** id) {
1159. **this**.id = id;
1160. }
1162. **public** **boolean** isActive() {
1163. **return** active;
1164. }
1166. **public** **void** setActive(**boolean** active) {
1167. **this**.active = active;
1168. }
1170. **public** String getLocation() {
1171. **return** location;
1172. }
1174. **public** **void** setLocation(String location) {
1175. **this**.location = location;
1176. }
1178. **public** **int** getSmokeLvl() {
1179. **return** smokeLvl;
1180. }
1182. **public** **void** setSmokeLvl(**int** smokeLvl) {
1183. **this**.smokeLvl = smokeLvl;
1184. }
1186. **public** **int** getCO2Lvl() {
1187. **return** CO2Lvl;
1188. }
1190. **public** **void** setCO2Lvl(**int** CO2Lvl) {
1191. **this**.CO2Lvl = CO2Lvl;
1192. }
1194. **public** String toString() {
1195. **return** "> "+**this**.id+**this**.active+**this**.location+**this**.smokeLvl+**this**.CO2Lvl;
1196. }
1198. }
1199. /\*
1200. \* To change this license header, choose License Headers in Project Properties.
1201. \* To change this template file, choose Tools | Templates
1202. \* and open the template in the editor.
1203. \*/
1204. **package** DS\_Project;
1206. **import** com.google.gson.Gson;
1207. **import** java.io.BufferedReader;
1208. **import** java.io.IOException;
1209. **import** java.io.InputStreamReader;
1210. **import** java.io.Reader;
1211. **import** java.io.UnsupportedEncodingException;
1212. **import** java.net.HttpURLConnection;
1213. **import** java.net.MalformedURLException;
1214. **import** java.net.URL;
1215. **import** java.rmi.RemoteException;
1216. **import** java.rmi.registry.LocateRegistry;
1217. **import** java.rmi.registry.Registry;
1218. **import** java.rmi.server.UnicastRemoteObject;
1219. **import** java.util.ArrayList;
1220. **import** java.util.LinkedHashMap;
1221. **import** java.util.List;
1222. **import** java.util.Map;
1223. **import** java.util.Timer;
1224. **import** java.util.TimerTask;
1225. **import** java.util.logging.Level;
1226. **import** java.util.logging.Logger;
1227. **import** org.json.JSONArray;
1228. **import** org.json.JSONObject;
1230. /\*\*
1231. \*
1232. \* @author Kesara
1233. \*/
1234. **public** **class** Server **extends** UnicastRemoteObject **implements** RMI\_Interface {
1236. **private** **static** HttpURLConnection connection;
1238. Sensors s1 = **new** Sensors(0,**false**,"\*",0,0);
1240. **public** Server() **throws** RemoteException {
1241. **super**();
1243. }
1245. //implementation of the getAllSensors method in RMI\_Interface
1246. @Override
1247. **public** List getAllSensors() **throws** RemoteException {
1249. BufferedReader reader;
1250. String line;
1251. StringBuffer response = **new** StringBuffer();
1253. **try** {
1254. URL url = **new** URL("http://localhost:8080/rest/webapi/sensors");
1255. connection = (HttpURLConnection) url.openConnection();
1257. connection.setRequestMethod("GET");
1258. connection.setConnectTimeout(5000);
1259. connection.setReadTimeout(5000);
1261. **int** status = connection.getResponseCode();
1263. **if** (status > 299) {
1264. reader = **new** BufferedReader(**new** InputStreamReader(connection.getErrorStream()));
1265. **while** ((line = reader.readLine()) != **null**) {
1266. response.append(line);
1267. }
1269. reader.close();
1270. } **else** {
1271. reader = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));
1272. **while** ((line = reader.readLine()) != **null**) {
1273. response.append(line);
1274. }
1276. reader.close();
1277. }
1279. System.out.println(status);
1281. List<Sensors> list = **new** ArrayList<>();
1283. JSONArray sensors = **new** JSONArray(response.toString());
1284. **for** (**int** i = 0; i < sensors.length(); i++) {
1285. JSONObject sensor = sensors.getJSONObject(i);
1286. **int** id = sensor.getInt("id");
1287. **boolean** isActive = sensor.getBoolean("active");
1288. String location = sensor.getString("location");
1289. **int** smokeLvl = sensor.getInt("smokeLvl");
1290. **int** co2Lvl = sensor.getInt("CO2Lvl");
1292. list.add(**new** Sensors(id, isActive, location, smokeLvl, co2Lvl));
1294. }
1296. **return** list;
1298. } **catch** (MalformedURLException ex) {
1299. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1300. } **catch** (IOException ex) {
1301. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1302. } **finally** {
1303. connection.disconnect();
1304. }
1306. **return** **null**;


1310. }

1313. //implementation of the addSensors method in RMI\_Interface
1314. @Override
1315. **public** **void** addSensors(**int** id, **boolean** active, String location, **int** smokeLevel, **int** co2Level) **throws** RemoteException {
1317. s1 = **new** Sensors(id,active,location,smokeLevel,co2Level);
1318. System.out.println(s1);
1320. **if**(s1.getId() == 0){
1321. System.out.println("No sensor added");
1322. }
1323. **else**{
1324. Gson json = **new** Gson();
1325. String res = json.toJson(s1);
1327. System.out.println(res);

1330. **try** {
1331. URL url = **new** URL("http://localhost:8080/rest/webapi/sensors/add");
1332. Map<String, String> params = **new** LinkedHashMap<>();
1333. params.put("Sensors", res);
1335. **byte**[] postDataBytes = res.getBytes("UTF-8");
1337. System.out.println(postDataBytes);
1338. HttpURLConnection conn = (HttpURLConnection) url.openConnection();
1339. conn.setRequestMethod("POST");
1340. conn.setRequestProperty("Content-Type", "application/json");
1341. conn.setRequestProperty("Content-Length", String.valueOf(postDataBytes.length));
1342. conn.setDoOutput(**true**);
1343. conn.getOutputStream().write(postDataBytes);
1345. System.out.println(postDataBytes);
1347. Reader in = **new** BufferedReader(**new** InputStreamReader(conn.getInputStream(), "UTF-8"));
1348. StringBuilder sb = **new** StringBuilder();
1349. **for** (**int** c; (c = in.read()) >= 0;) {
1350. sb.append((**char**) c);
1351. }
1353. String response = sb.toString();
1355. System.out.println(response);
1357. JSONObject myResponse = **new** JSONObject(response.toString());
1358. System.out.println("result after Reading JSON Response");
1359. System.out.println("origin- " + myResponse.getString("origin"));
1360. System.out.println("url- " + myResponse.getString("url"));
1362. JSONObject form\_data = myResponse.getJSONObject("form");
1363. System.out.println("id= " + form\_data.getString("id"));
1364. System.out.println("active= " + form\_data.getString("isActive"));
1365. System.out.println("location= " + form\_data.getString("location"));
1366. System.out.println("smoke= " + form\_data.getString("smokeLvl"));
1367. System.out.println("co2= " + form\_data.getString("CO2Lvl"));
1369. } **catch** (MalformedURLException e) {
1370. System.out.println(e.getMessage());
1371. } **catch** (UnsupportedEncodingException e) {
1372. System.out.println(e.getMessage());
1373. } **catch** (Exception e) {
1374. System.out.println(e.getMessage());
1375. }
1377. }

1380. }


1384. @Override
1385. **public** **void** updateSensors(**int** id, **boolean** active, String location, **int** smokelevel, **int** co2level) **throws** RemoteException {
1387. s1 = **new** Sensors(id, active, location, smokelevel, co2level);
1388. System.out.println(s1);
1390. **if** (s1.getId() == 0) {
1391. System.out.println("No sensor added.");
1392. } **else** {
1393. Gson json = **new** Gson();
1394. String res = json.toJson(s1);
1396. System.out.println(res);
1398. **try** {
1399. URL url = **new** URL("http://localhost:8080/rest/webapi/sensors/update");
1400. Map<String, String> params = **new** LinkedHashMap<>();
1401. params.put("Sensors", res);
1403. **byte**[] postDataBytes = res.getBytes("UTF-8");
1405. System.out.println(postDataBytes);
1407. HttpURLConnection conn = (HttpURLConnection) url.openConnection();
1408. conn.setRequestMethod("PUT");
1409. conn.setRequestProperty("Content-Type", "application/json");
1410. conn.setRequestProperty("Content-Length", String.valueOf(postDataBytes.length));
1411. conn.setDoOutput(**true**);
1412. conn.getOutputStream().write(postDataBytes);
1414. System.out.println(postDataBytes);
1416. Reader in = **new** BufferedReader(**new** InputStreamReader(conn.getInputStream(), "UTF-8"));
1417. StringBuilder sb = **new** StringBuilder();
1418. **for** (**int** c; (c = in.read()) >= 0;) {
1419. sb.append((**char**) c);
1420. }
1422. String response = sb.toString();
1424. System.out.println(response);
1426. JSONObject myResponse = **new** JSONObject(response.toString());
1427. System.out.println("result after Reading JSON Response");
1428. System.out.println("origin- " + myResponse.getString("origin"));
1429. System.out.println("url- " + myResponse.getString("url"));
1431. JSONObject form\_data = myResponse.getJSONObject("form");
1432. System.out.println("id= " + form\_data.getString("id"));
1433. System.out.println("active= " + form\_data.getString("isActive"));
1434. System.out.println("location= " + form\_data.getString("location"));
1435. System.out.println("smoke= " + form\_data.getString("smokeLvl"));
1436. System.out.println("co2= " + form\_data.getString("CO2Lvl"));
1438. } **catch** (MalformedURLException e) {
1439. System.out.println(e.getMessage());
1440. } **catch** (UnsupportedEncodingException e) {
1441. System.out.println(e.getMessage());
1442. } **catch** (Exception e) {
1443. System.out.println(e.getMessage());
1444. }
1445. }
1447. }
1449. @Override
1450. **public** List<Sensors> displayAlert() **throws** RemoteException {
1452. BufferedReader reader;
1453. String line;
1454. StringBuffer response = **new** StringBuffer();
1456. **try** {
1457. URL url = **new** URL("http://localhost:8080/rest/webapi/sensors");
1458. connection = (HttpURLConnection) url.openConnection();
1460. connection.setRequestMethod("GET");
1461. connection.setConnectTimeout(5000);
1462. connection.setReadTimeout(5000);
1464. **int** status = connection.getResponseCode();
1466. **if** (status > 299) {
1467. reader = **new** BufferedReader(**new** InputStreamReader(connection.getErrorStream()));
1468. **while** ((line = reader.readLine()) != **null**) {
1469. response.append(line);
1470. }
1472. reader.close();
1473. } **else** {
1474. reader = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));
1475. **while** ((line = reader.readLine()) != **null**) {
1476. response.append(line);
1477. }
1479. reader.close();
1480. }
1482. System.out.println(status);

1485. List<Sensors> list = **new** ArrayList<>();
1487. JSONArray sensors = **new** JSONArray(response.toString());
1489. **for** (**int** i = 0; i < sensors.length(); i++) {
1490. JSONObject sensor = sensors.getJSONObject(i);
1491. **int** id = sensor.getInt("id");
1492. **boolean** isActive = sensor.getBoolean("active");
1493. String location = sensor.getString("location");
1494. **int** smokeLvl = sensor.getInt("smokeLvl");
1495. **int** co2Lvl = sensor.getInt("CO2Lvl");
1497. **if**(smokeLvl > 5 || co2Lvl > 5){
1498. list.add(**new** Sensors(id, isActive, location, smokeLvl, co2Lvl));
1499. }
1501. }
1503. **return** list;
1505. } **catch** (MalformedURLException ex) {
1506. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1507. } **catch** (IOException ex) {
1508. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1509. } **finally** {
1510. connection.disconnect();
1511. }
1513. **return** **null**;
1515. }
1517. @Override
1518. **public** **void** sendStatus() **throws** RemoteException {
1520. BufferedReader reader;
1521. String line;
1522. StringBuffer response = **new** StringBuffer();
1523. List<Sensors> list = **new** ArrayList<>();
1524. **try** {
1525. URL url = **new** URL("http://localhost:8080/rest/webapi/sensors");
1526. connection = (HttpURLConnection) url.openConnection();
1528. connection.setRequestMethod("GET");
1529. connection.setConnectTimeout(5000);
1530. connection.setReadTimeout(5000);
1532. **int** status = connection.getResponseCode();
1534. **if** (status > 299) {
1535. reader = **new** BufferedReader(**new** InputStreamReader(connection.getErrorStream()));
1536. **while** ((line = reader.readLine()) != **null**) {
1537. response.append(line);
1538. }
1540. reader.close();
1541. } **else** {
1542. reader = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));
1543. **while** ((line = reader.readLine()) != **null**) {
1544. response.append(line);
1545. }
1547. reader.close();
1548. }
1550. System.out.println(status);
1552. JSONArray sensors = **new** JSONArray(response.toString());
1553. **for** (**int** i = 0; i < sensors.length(); i++) {
1554. JSONObject sensor = sensors.getJSONObject(i);
1555. **int** id = sensor.getInt("id");
1556. **boolean** isActive = sensor.getBoolean("active");
1557. String location = sensor.getString("location");
1558. **int** smokeLvl = sensor.getInt("smokeLvl");
1559. **int** co2Lvl = sensor.getInt("CO2Lvl");
1561. list.add(**new** Sensors(id, isActive, location, smokeLvl, co2Lvl));
1563. }
1564. } **catch** (MalformedURLException ex) {
1565. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1566. } **catch** (IOException ex) {
1567. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1568. } **finally** {
1569. connection.disconnect();
1570. }
1572. **int** count = 0;
1573. **for**(Sensors s : list) {
1574. **if**(s.getCO2Lvl()> 5 || s.getSmokeLvl() > 5) {
1575. count++;
1576. }
1577. }
1579. **boolean** sendEmail;
1580. **if**(count > 0){
1581. sendEmail = **true**;
1582. } **else** {
1583. sendEmail = **false**;
1584. }
1586. Emails e1 = **new** Emails(sendEmail);
1587. System.out.println(e1);
1589. Gson json = **new** Gson();
1590. String res = json.toJson(e1);
1592. System.out.println(res);
1594. **try** {
1595. URL url = **new** URL("http://localhost:8080/rest/webapi/emails/update");
1596. Map<String, String> params = **new** LinkedHashMap<>();
1597. params.put("Emails", res);
1599. **byte**[] postDataBytes = res.getBytes("UTF-8");
1601. System.out.println(postDataBytes);
1603. HttpURLConnection conn = (HttpURLConnection) url.openConnection();
1604. conn.setRequestMethod("PUT");
1605. conn.setRequestProperty("Content-Type", "application/json");
1606. conn.setRequestProperty("Content-Length", String.valueOf(postDataBytes.length));
1607. conn.setDoOutput(**true**);
1608. conn.getOutputStream().write(postDataBytes);
1610. System.out.println(postDataBytes);
1612. Reader in = **new** BufferedReader(**new** InputStreamReader(conn.getInputStream(), "UTF-8"));
1613. StringBuilder sb = **new** StringBuilder();
1614. **for** (**int** c; (c = in.read()) >= 0;) {
1615. sb.append((**char**) c);
1616. }
1618. String r = sb.toString();
1620. System.out.println(r);
1622. JSONObject myResponse = **new** JSONObject(r.toString());
1623. System.out.println("result after Reading JSON Response");
1624. System.out.println("origin- " + myResponse.getString("origin"));
1625. System.out.println("url- " + myResponse.getString("url"));
1627. } **catch** (MalformedURLException e) {
1628. System.out.println(e.getMessage());
1629. } **catch** (UnsupportedEncodingException e) {
1630. System.out.println(e.getMessage());
1631. } **catch** (Exception e) {
1632. System.out.println(e.getMessage());
1633. }


1637. }
1639. //RMI server check the sensor status every 15 seconds to get uptodate readings
1640. Timer timer = **new** Timer();
1642. TimerTask task = **new** TimerTask() {
1643. @Override
1644. **public** **void** run() {
1645. **try** {
1646. getAllSensors();
1647. sendStatus();
1648. displayAlert();
1650. } **catch** (Exception e) {
1651. System.out.println(e.getMessage());
1652. }
1653. }
1654. };
1656. **public** **void** start() {
1657. timer.scheduleAtFixedRate(task, 15000, 15000);
1658. }

1661. **public** **static** **void** main(String[] args) {
1662. **try** {
1663. Registry reg = LocateRegistry.createRegistry(Registry.REGISTRY\_PORT);
1665. Server server = **new** Server();
1667. reg.rebind("rmi://localhost/server", server);
1668. System.out.println("Server is running...");
1670. server.start();


1674. } **catch** (RemoteException ex) {
1675. Logger.getLogger(Server.**class**.getName()).log(Level.SEVERE, **null**, ex);
1676. }
1677. }
1679. }
1680. /\*
1681. \* To change this license header, choose License Headers in Project Properties.
1682. \* To change this template file, choose Tools | Templates
1683. \* and open the template in the editor.
1684. \*/
1685. **package** DS\_Project;
1687. **import** java.awt.Color;
1688. **import** java.awt.Component;
1689. **import** javax.swing.JLabel;
1690. **import** javax.swing.JTable;
1691. **import** javax.swing.table.DefaultTableCellRenderer;
1693. /\*\*
1694. \*
1695. \* @author Kesara
1696. \*/
1697. **public** **class** Table\_Colour **extends** DefaultTableCellRenderer {
1699. //If the smoke level or CO2 level is above 5 they will be marked in red
1701. **public** Table\_Colour() {
1702. }
1704. **public** Component getTableCellRendererComponent(JTable jtable, Object value, **boolean** isSelected, **boolean** hasFocus, **int** row, **int** column) {
1705. JLabel label = (JLabel) **super**.getTableCellRendererComponent(jtable, value, isSelected, hasFocus, row, column);
1707. Color back = Color.WHITE;
1709. Object obj = jtable.getValueAt(row, 3);
1710. Object ob = jtable.getValueAt(row, 4);
1712. **try** {
1713. **int** co2 = Integer.parseInt(obj.toString());
1714. **int** smoke = Integer.parseInt(ob.toString());
1716. **if** (co2 > 5 || smoke > 5) {
1717. back = Color.RED;
1718. }
1720. } **catch** (Exception e) {
1722. }
1724. label.setBackground(back);
1725. **return** label;
1726. }
1728. }
1729. /\*
1730. \* To change this license header, choose License Headers in Project Properties.
1731. \* To change this template file, choose Tools | Templates
1732. \* and open the template in the editor.
1733. \*/
1734. **package** DS\_Project;
1736. **import** java.rmi.NotBoundException;
1737. **import** java.rmi.RemoteException;
1738. **import** java.rmi.registry.LocateRegistry;
1739. **import** java.rmi.registry.Registry;
1740. **import** java.util.logging.Level;
1741. **import** java.util.logging.Logger;
1743. /\*\*
1744. \*
1745. \* @author Janith Perera
1746. \*/
1747. **public** **class** Update **extends** javax.swing.JFrame {
1749. /\*\*
1750. \* Creates new form Update
1751. \*/
1752. **public** Update() {
1753. initComponents();
1754. }
1756. /\*\*
1757. \* This method is called from within the constructor to initialize the form.
1758. \* WARNING: Do NOT modify this code. The content of this method is always
1759. \* regenerated by the Form Editor.
1760. \*/
1761. @SuppressWarnings("unchecked")
1762. // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
1763. **private** **void** initComponents() {
1765. jPanel1 = **new** javax.swing.JPanel();
1766. jLabel2 = **new** javax.swing.JLabel();
1767. jLabel3 = **new** javax.swing.JLabel();
1768. jTextField1 = **new** javax.swing.JTextField();
1769. jTextField2 = **new** javax.swing.JTextField();
1770. jButton1 = **new** javax.swing.JButton();
1771. jButton2 = **new** javax.swing.JButton();
1772. jLabel1 = **new** javax.swing.JLabel();
1774. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
1776. jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(**new** java.awt.Color(0, 0, 0), 5));
1778. jLabel2.setText("Enter Sensor ID :");
1780. jLabel3.setText("Sensor Location : ");
1782. jTextField1.addActionListener(**new** java.awt.event.ActionListener() {
1783. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
1784. jTextField1ActionPerformed(evt);
1785. }
1786. });
1788. jButton1.setText("Go Back");
1789. jButton1.addActionListener(**new** java.awt.event.ActionListener() {
1790. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
1791. jButton1ActionPerformed(evt);
1792. }
1793. });
1795. jButton2.setText("Update");
1796. jButton2.addActionListener(**new** java.awt.event.ActionListener() {
1797. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
1798. jButton2ActionPerformed(evt);
1799. }
1800. });
1802. javax.swing.GroupLayout jPanel1Layout = **new** javax.swing.GroupLayout(jPanel1);
1803. jPanel1.setLayout(jPanel1Layout);
1804. jPanel1Layout.setHorizontalGroup(
1805. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1806. .addGroup(jPanel1Layout.createSequentialGroup()
1807. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1808. .addGroup(jPanel1Layout.createSequentialGroup()
1809. .addGap(146, 146, 146)
1810. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
1811. .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
1812. .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
1813. .addGap(122, 122, 122)
1814. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
1815. .addComponent(jTextField1, javax.swing.GroupLayout.DEFAULT\_SIZE, 157, Short.MAX\_VALUE)
1816. .addComponent(jTextField2)))
1817. .addGroup(jPanel1Layout.createSequentialGroup()
1818. .addGap(189, 189, 189)
1819. .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 94, javax.swing.GroupLayout.PREFERRED\_SIZE)
1820. .addGap(95, 95, 95)
1821. .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED\_SIZE, 102, javax.swing.GroupLayout.PREFERRED\_SIZE)))
1822. .addContainerGap(231, Short.MAX\_VALUE))
1823. );
1824. jPanel1Layout.setVerticalGroup(
1825. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1826. .addGroup(jPanel1Layout.createSequentialGroup()
1827. .addGap(77, 77, 77)
1828. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
1829. .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE)
1830. .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, 46, javax.swing.GroupLayout.PREFERRED\_SIZE))
1831. .addGap(52, 52, 52)
1832. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
1833. .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE)
1834. .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE))
1835. .addGap(45, 45, 45)
1836. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
1837. .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE)
1838. .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE))
1839. .addContainerGap(59, Short.MAX\_VALUE))
1840. );
1842. jLabel1.setFont(**new** java.awt.Font("Tahoma", 1, 36)); // NOI18N
1843. jLabel1.setText("Update Sensor Details");
1845. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
1846. getContentPane().setLayout(layout);
1847. layout.setHorizontalGroup(
1848. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1849. .addGroup(layout.createSequentialGroup()
1850. .addGap(48, 48, 48)
1851. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
1852. .addContainerGap(32, Short.MAX\_VALUE))
1853. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
1854. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
1855. .addComponent(jLabel1)
1856. .addGap(218, 218, 218))
1857. );
1858. layout.setVerticalGroup(
1859. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
1860. .addGroup(layout.createSequentialGroup()
1861. .addGap(30, 30, 30)
1862. .addComponent(jLabel1)
1863. .addGap(18, 18, 18)
1864. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
1865. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
1866. );
1868. pack();
1869. }// </editor-fold>//GEN-END:initComponents
1871. **private** **void** jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jTextField1ActionPerformed
1872. // TODO add your handling code here:
1873. }//GEN-LAST:event\_jTextField1ActionPerformed
1875. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed
1876. add\_or\_update back = **new** add\_or\_update();
1877. back.setVisible(**true**);
1878. }//GEN-LAST:event\_jButton1ActionPerformed
1880. **private** **void** jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton2ActionPerformed
1881. **try** {
1882. Registry reg = LocateRegistry.getRegistry("localhost",1099);
1883. RMI\_Interface server = (RMI\_Interface) reg.lookup("rmi://localhost/server");
1885. **int** id = Integer.valueOf(jTextField1.getText());
1886. **boolean** check = **false**;
1887. String location = jTextField2.getText();
1888. **int** smokeLvl = 0;
1889. **int** co2Lvl = 0;

1892. server.updateSensors(id, check, location, smokeLvl, co2Lvl);

1895. } **catch** (RemoteException ex) {
1896. Logger.getLogger(Add.**class**.getName()).log(Level.SEVERE, **null**, ex);
1897. } **catch** (NotBoundException ex) {
1898. Logger.getLogger(Add.**class**.getName()).log(Level.SEVERE, **null**, ex);
1899. }
1900. }//GEN-LAST:event\_jButton2ActionPerformed
1902. /\*\*
1903. \* @param args the command line arguments
1904. \*/
1905. **public** **static** **void** main(String args[]) {
1906. /\* Set the Nimbus look and feel \*/
1907. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
1908. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
1909. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
1910. \*/
1911. **try** {
1912. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
1913. **if** ("Nimbus".equals(info.getName())) {
1914. javax.swing.UIManager.setLookAndFeel(info.getClassName());
1915. **break**;
1916. }
1917. }
1918. } **catch** (ClassNotFoundException ex) {
1919. java.util.logging.Logger.getLogger(Update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1920. } **catch** (InstantiationException ex) {
1921. java.util.logging.Logger.getLogger(Update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1922. } **catch** (IllegalAccessException ex) {
1923. java.util.logging.Logger.getLogger(Update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1924. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
1925. java.util.logging.Logger.getLogger(Update.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
1926. }
1927. //</editor-fold>
1929. /\* Create and display the form \*/
1930. java.awt.EventQueue.invokeLater(**new** Runnable() {
1931. **public** **void** run() {
1932. **new** Update().setVisible(**true**);
1933. }
1934. });
1935. }
1937. // Variables declaration - do not modify//GEN-BEGIN:variables
1938. **private** javax.swing.JButton jButton1;
1939. **private** javax.swing.JButton jButton2;
1940. **private** javax.swing.JLabel jLabel1;
1941. **private** javax.swing.JLabel jLabel2;
1942. **private** javax.swing.JLabel jLabel3;
1943. **private** javax.swing.JPanel jPanel1;
1944. **private** javax.swing.JTextField jTextField1;
1945. **private** javax.swing.JTextField jTextField2;
1946. // End of variables declaration//GEN-END:variables
1947. }
1948. /\*
1949. \* To change this license header, choose License Headers in Project Properties.
1950. \* To change this template file, choose Tools | Templates
1951. \* and open the template in the editor.
1952. \*/
1953. **package** DS\_Project;
1955. **import** DS\_Project.RMI\_Interface;
1956. //import DS\_Project.Sensors;
1957. **import** java.rmi.NotBoundException;
1958. **import** java.rmi.RemoteException;
1959. **import** java.rmi.registry.LocateRegistry;
1960. **import** java.rmi.registry.Registry;
1961. **import** java.util.ArrayList;
1962. **import** java.util.List;
1963. **import** java.util.Timer;
1964. **import** java.util.TimerTask;
1965. **import** javax.swing.JOptionPane;
1966. **import** javax.swing.table.DefaultTableModel;
1968. /\*\*
1969. \*
1970. \* @author Kesara
1971. \*/
1972. **public** **class** viewSensors **extends** javax.swing.JFrame {

1975. /\*\*
1976. \* Creates new form viewSensors
1977. \*/
1978. **public** viewSensors() {
1979. initComponents();
1981. **try**{
1982. getSensorsView();
1983. }
1984. **catch**(RemoteException e){
1985. System.out.println(e.getMessage());
1986. }
1988. start();
1990. jTableViewSensors.setDefaultRenderer(Object.**class**, **new** Table\_Colour());
1992. }
1994. //adding sensor details to the table
1995. **public** **void** getSensorsView() **throws** RemoteException{
1997. **try** {
1998. Registry reg = LocateRegistry.getRegistry("localhost",1099);
1999. RMI\_Interface server = (RMI\_Interface) reg.lookup("rmi://localhost/server");
2001. List<Sensors> list = server.getAllSensors();
2003. DefaultTableModel tableModel = (DefaultTableModel) jTableViewSensors.getModel();
2005. Object[] col = **new** Object[5];

2008. **for**(Sensors s : list){
2009. col[0] = s.getId();
2010. col[1] = s.isActive();
2011. col[2] = s.getLocation();
2012. col[3] = s.getSmokeLvl();
2013. col[4] = s.getCO2Lvl();
2015. tableModel.addRow(col);
2016. }


2020. }  **catch**(RemoteException | NotBoundException e) {
2021. System.out.println(e.getMessage());
2022. }
2024. }
2026. //RMI client get the sensor status every 15 second
2027. Timer timer = **new** Timer();
2029. TimerTask task = **new** TimerTask() {
2030. @Override
2031. **public** **void** run() {
2032. **try** {
2033. jTableViewSensors.setModel(**new** DefaultTableModel(**null**, **new** Object[]{"ID", "Active/Not Active", "Location", "Smoke Level", "CO2 Level"}));
2034. getSensorsView();


2038. } **catch** (Exception e) {
2039. System.out.println(e.getMessage());
2040. }


2044. }
2045. };

2048. **public** **void** start() {
2049. timer.scheduleAtFixedRate(task, 30000, 30000);
2050. }


2054. /\*\*
2055. \* This method is called from within the constructor to initialize the form.
2056. \* WARNING: Do NOT modify this code. The content of this method is always
2057. \* regenerated by the Form Editor.
2058. \*/
2059. @SuppressWarnings("unchecked")
2060. // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
2061. **private** **void** initComponents() {
2063. jPanel1 = **new** javax.swing.JPanel();
2064. jScrollPane1 = **new** javax.swing.JScrollPane();
2065. jTableViewSensors = **new** javax.swing.JTable();
2067. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
2069. jTableViewSensors.setModel(**new** javax.swing.table.DefaultTableModel(
2070. **new** Object [][] {
2072. },
2073. **new** String [] {
2074. "ID", "Active/Not Active", "Location", "Smoke Level", "CO2 Level"
2075. }
2076. ));
2077. jScrollPane1.setViewportView(jTableViewSensors);
2079. javax.swing.GroupLayout jPanel1Layout = **new** javax.swing.GroupLayout(jPanel1);
2080. jPanel1.setLayout(jPanel1Layout);
2081. jPanel1Layout.setHorizontalGroup(
2082. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
2083. .addGroup(jPanel1Layout.createSequentialGroup()
2084. .addContainerGap()
2085. .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 1074, Short.MAX\_VALUE)
2086. .addContainerGap())
2087. );
2088. jPanel1Layout.setVerticalGroup(
2089. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
2090. .addGroup(jPanel1Layout.createSequentialGroup()
2091. .addContainerGap()
2092. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 744, javax.swing.GroupLayout.PREFERRED\_SIZE)
2093. .addContainerGap(27, Short.MAX\_VALUE))
2094. );
2096. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
2097. getContentPane().setLayout(layout);
2098. layout.setHorizontalGroup(
2099. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
2100. .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
2101. );
2102. layout.setVerticalGroup(
2103. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
2104. .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
2105. );
2107. pack();
2108. }// </editor-fold>//GEN-END:initComponents
2110. /\*\*
2111. \* @param args the command line arguments
2112. \*/
2113. **public** **static** **void** main(String args[]) {
2114. /\* Set the Nimbus look and feel \*/
2115. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
2116. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
2117. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
2118. \*/
2119. **try** {
2120. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
2121. **if** ("Nimbus".equals(info.getName())) {
2122. javax.swing.UIManager.setLookAndFeel(info.getClassName());
2123. **break**;
2124. }
2125. }
2126. } **catch** (ClassNotFoundException ex) {
2127. java.util.logging.Logger.getLogger(viewSensors.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
2128. } **catch** (InstantiationException ex) {
2129. java.util.logging.Logger.getLogger(viewSensors.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
2130. } **catch** (IllegalAccessException ex) {
2131. java.util.logging.Logger.getLogger(viewSensors.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
2132. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
2133. java.util.logging.Logger.getLogger(viewSensors.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
2134. }
2135. //</editor-fold>
2137. /\* Create and display the form \*/
2138. java.awt.EventQueue.invokeLater(**new** Runnable() {
2139. **public** **void** run() {
2140. **new** viewSensors().setVisible(**true**);
2141. }
2142. });






2150. }
2152. // Variables declaration - do not modify//GEN-BEGIN:variables
2153. **private** javax.swing.JPanel jPanel1;
2154. **private** javax.swing.JScrollPane jScrollPane1;
2155. **private** javax.swing.JTable jTableViewSensors;
2156. // End of variables declaration//GEN-END:variables
2157. }
2158. // email model class
2159. **package** com.ds.rest;
2161. **public** **class** Email {
2162. **private** **boolean** sendEmail;
2164. // getters and setters
2165. **public** **boolean** isSendEmail() {
2166. **return** sendEmail;
2167. }
2169. **public** **void** setSendEmail(**boolean** sendEmail) {
2170. **this**.sendEmail = sendEmail;
2171. }
2172. }
2173. // email repository class
2174. // talks to the DB
2175. **package** com.ds.rest;
2177. **import** java.sql.\*;
2178. **import** java.util.ArrayList;
2179. **import** java.util.List;
2181. **public** **class** EmailRepository {
2183. Connection connection = **null**;
2185. **public** EmailRepository() {
2187. String url = "jdbc:mysql://localhost:3306/restdb?useSSL=false";
2188. String username = "root";
2189. String password = "root";
2191. **try** {
2192. Class.forName("com.mysql.jdbc.Driver"); // load the driver
2193. connection = DriverManager.getConnection(url, username, password); // get connection object
2194. } **catch** (SQLException e) {
2195. e.printStackTrace();
2196. } **catch** (ClassNotFoundException e) {
2197. e.printStackTrace();
2198. }
2200. }
2202. // get boolean value from the DB
2203. **public** Email getStatus() {
2204. String sql = "select \* from email";
2205. Email email = **new** Email();
2207. **try** {
2208. Statement st = connection.createStatement();
2209. ResultSet rs = st.executeQuery(sql);
2211. // assign returned values to the created email object
2212. **if**(rs.next()) {
2213. email.setSendEmail(rs.getBoolean(1));
2214. }
2216. } **catch** (Exception e) {
2217. e.printStackTrace();
2218. }
2220. **return** email;
2221. }


2225. // update value
2226. **public** Email updateEmail(Email email) {
2228. String sql = "update email set sendEmail=?";
2229. **try** {
2230. PreparedStatement st = connection.prepareStatement(sql);
2231. // set values from the passed email object to update
2232. st.setBoolean(1, email.isSendEmail());
2234. st.executeUpdate();
2236. } **catch** (SQLException e) {
2237. e.printStackTrace();
2238. }
2240. **return** email;
2241. }
2242. }
2243. // email resource class
2244. **package** com.ds.rest;
2246. **import** javax.ws.rs.Consumes;
2247. **import** javax.ws.rs.GET;
2248. **import** javax.ws.rs.PUT;
2249. **import** javax.ws.rs.Path;
2250. **import** javax.ws.rs.Produces;
2251. **import** javax.ws.rs.core.MediaType;
2253. @Path("emails") // this resource will be called when a request is made with "emais" in it
2254. **public** **class** EmailResource {
2256. EmailRepository repo = **new** EmailRepository();
2258. // this method will be called when a GET request is made with /emails at the end
2259. @GET
2260. @Produces(MediaType.APPLICATION\_JSON)
2261. **public** Email getStatus() {
2262. System.out.println("called");
2263. **return** repo.getStatus();
2264. }

2267. // this method will be called when a PUT request is made with /emails/update at the end
2268. @PUT
2269. @Path("update")
2270. @Consumes(MediaType.APPLICATION\_JSON)
2271. **public** Email updateStatus(Email email) {
2272. System.out.println("update called");
2274. **return** repo.updateEmail(email);
2275. }
2277. }
2278. // sensor model class
2279. **package** com.ds.rest;
2281. **public** **class** Sensor {
2282. **private** **int** id;
2283. **private** **boolean** isActive;
2284. **private** String location;
2285. **private** **int** smokeLvl;
2286. **private** **int** CO2Lvl;
2288. // getters and setters
2289. **public** **boolean** isActive() {
2290. **return** isActive;
2291. }
2292. **public** **void** setActive(**boolean** isActive) {
2293. **this**.isActive = isActive;
2294. }
2295. **public** String getLocation() {
2296. **return** location;
2297. }
2298. **public** **void** setLocation(String location) {
2299. **this**.location = location;
2300. }
2301. **public** **int** getSmokeLvl() {
2302. **return** smokeLvl;
2303. }
2304. **public** **void** setSmokeLvl(**int** smokeLvl) {
2305. **this**.smokeLvl = smokeLvl;
2306. }
2307. **public** **int** getCO2Lvl() {
2308. **return** CO2Lvl;
2309. }
2310. **public** **void** setCO2Lvl(**int** cO2Lvl) {
2311. CO2Lvl = cO2Lvl;
2312. }
2313. **public** **int** getId() {
2314. **return** id;
2315. }
2316. **public** **void** setId(**int** id) {
2317. **this**.id = id;
2318. }
2319. }
2320. // sensor repository class
2321. // talks to the DB
2322. **package** com.ds.rest;
2324. **import** java.sql.\*;
2325. **import** java.util.ArrayList;
2326. **import** java.util.List;
2328. **public** **class** SensorRepository {
2330. Connection connection = **null**;
2332. **public** SensorRepository() {
2334. String url = "jdbc:mysql://localhost:3306/restdb?useSSL=false";
2335. String username = "root";
2336. String password = "root";
2338. **try** {
2339. Class.forName("com.mysql.jdbc.Driver"); // load the driver
2340. connection = DriverManager.getConnection(url, username, password); // get connection object
2341. } **catch** (SQLException e) {
2342. e.printStackTrace();
2343. } **catch** (ClassNotFoundException e) {
2344. e.printStackTrace();
2345. }
2347. }
2349. // get details of all the sensors from the DB
2350. **public** List<Sensor> getSensors() {
2351. List<Sensor> sensors = **new** ArrayList<>();
2352. String sql = "select \* from sensor";
2354. **try** {
2355. Statement st = connection.createStatement();
2356. ResultSet rs = st.executeQuery(sql);
2357. // for each result, create a sensor object and assign the returned values and add the object to sensors
2358. **while**(rs.next()) {
2359. Sensor sensor = **new** Sensor();
2360. sensor.setId(rs.getInt(1));
2361. sensor.setActive(rs.getBoolean(2));
2362. sensor.setLocation(rs.getString(3));
2363. sensor.setSmokeLvl(rs.getInt(4));
2364. sensor.setCO2Lvl(rs.getInt(5));
2366. sensors.add(sensor);
2367. }
2369. } **catch** (Exception e) {
2370. e.printStackTrace();
2371. }
2373. **return** sensors;
2374. }
2376. // get details of a particular sensor from the DB
2377. **public** Sensor getSensor(**int** id) {
2378. String sql = "select \* from sensor where id="+id;
2379. Sensor sensor = **new** Sensor();;
2381. **try** {
2382. Statement st = connection.createStatement();
2383. ResultSet rs = st.executeQuery(sql);
2385. // assign returned values to the created sensor object
2386. **if**(rs.next()) {
2387. sensor.setId(rs.getInt(1));
2388. sensor.setActive(rs.getBoolean(2));
2389. sensor.setLocation(rs.getString(3));
2390. sensor.setSmokeLvl(rs.getInt(4));
2391. sensor.setCO2Lvl(rs.getInt(5));
2392. }
2394. } **catch** (Exception e) {
2395. e.printStackTrace();
2396. }
2398. **return** sensor;
2399. }
2401. // add a sensor to the DB
2402. **public** Sensor addSensor(Sensor sensor) {
2403. String sql = "insert into sensor values (?, ?, ?, ?, ?)";
2404. **try** {
2405. PreparedStatement st = connection.prepareStatement(sql);
2406. // set values from the passed sensor object to insert it to the database
2407. st.setInt(1, sensor.getId());
2408. st.setBoolean(2, sensor.isActive());
2409. st.setString(3, sensor.getLocation());
2410. st.setInt(4, sensor.getSmokeLvl());
2411. st.setInt(5, sensor.getCO2Lvl());
2413. st.executeUpdate();
2415. } **catch** (SQLException e) {
2416. e.printStackTrace();
2417. }
2419. **return** sensor;
2420. }
2422. // update details of a sensor
2423. **public** Sensor updateSensor(Sensor sensor) {
2424. String sql;
2426. **if** (sensor.getCO2Lvl() == 0) {
2427. // this will be executed when the admin tries to update the location of a sensor
2428. sql = "update sensor set location=? where id=?";
2429. **try** {
2430. PreparedStatement st = connection.prepareStatement(sql);
2431. // set id and location from the passed sensor object to update its location
2432. st.setString(1, sensor.getLocation());
2433. st.setInt(2, sensor.getId());
2435. st.executeUpdate();
2437. } **catch** (SQLException e) {
2438. e.printStackTrace();
2439. }
2440. } **else** {
2441. // this will be executed when the sensor app tries to update sensor details
2442. sql = "update sensor set isActive=?, location=?, smokeLvl=?, CO2Lvl=? where id=?";
2443. **try** {
2444. PreparedStatement st = connection.prepareStatement(sql);
2445. // set values from the passed sensor object to update its details
2446. st.setBoolean(1, sensor.isActive());
2447. st.setString(2, sensor.getLocation());
2448. st.setInt(3, sensor.getSmokeLvl());
2449. st.setInt(4, sensor.getCO2Lvl());
2450. st.setInt(5, sensor.getId());
2452. st.executeUpdate();
2454. } **catch** (SQLException e) {
2455. e.printStackTrace();
2456. }
2457. }
2459. **return** sensor;
2460. }
2461. }
2462. // sensor resource class
2463. **package** com.ds.rest;
2465. **import** java.util.List;
2467. **import** javax.ws.rs.Consumes;
2468. **import** javax.ws.rs.GET;
2469. **import** javax.ws.rs.POST;
2470. **import** javax.ws.rs.PUT;
2471. **import** javax.ws.rs.Path;
2472. **import** javax.ws.rs.PathParam;
2473. **import** javax.ws.rs.Produces;
2474. **import** javax.ws.rs.core.MediaType;
2476. @Path("sensors") // this resource will be called when a request is made with "sensors" in it
2477. **public** **class** SensorResource {
2479. SensorRepository repo = **new** SensorRepository();
2481. // this method will be called when a GET request is made with /sensors at the end
2482. @GET
2483. @Produces(MediaType.APPLICATION\_JSON)
2484. **public** List<Sensor> getSensors() {
2486. **return** repo.getSensors();
2487. }
2489. // this method will be called when a POST request is made with /sensors/add at the end
2490. @POST
2491. @Path("add")
2492. @Consumes(MediaType.APPLICATION\_JSON)
2493. **public** Sensor addSensor(Sensor sensor) {
2495. **return** repo.addSensor(sensor);
2496. }
2498. // this method will be called when a PUT request is made with sensors/update at the end
2499. @PUT
2500. @Path("update")
2501. @Consumes(MediaType.APPLICATION\_JSON)
2502. **public** Sensor updateSensor(Sensor sensor) {
2504. **return** repo.updateSensor(sensor);
2505. }
2506. }
2507. // provider class
2508. **package** com.ds.rest.filters;
2510. **import** java.io.IOException;
2512. **import** javax.ws.rs.container.ContainerRequestContext;
2513. **import** javax.ws.rs.container.ContainerRequestFilter;
2514. **import** javax.ws.rs.container.ContainerResponseContext;
2515. **import** javax.ws.rs.container.ContainerResponseFilter;
2516. **import** javax.ws.rs.ext.Provider;
2518. @Provider
2519. **public** **class** AccessControlResponseFilter **implements** ContainerRequestFilter, ContainerResponseFilter {
2521. @Override
2522. **public** **void** filter(ContainerRequestContext requestContext, ContainerResponseContext responseContext)
2523. **throws** IOException {
2524. responseContext.getHeaders().add("Access-Control-Allow-Origin", "\*"); // defines who is allowed to make requests to the server
2525. responseContext.getHeaders().add("Access-Control-Allow-Headers","origin, content-type, accept, authorization"); // defines which headers are allowed to be passed up to the server
2526. responseContext.getHeaders().add("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS, HEAD"); // defines which HTTP methods that the server will response to
2527. }
2529. @Override
2530. **public** **void** filter(ContainerRequestContext requestContext) **throws** IOException {
2531. // TODO Auto-generated method stub
2533. }
2535. }
2536. sendDetails(); // initially call sendDetails()
2537. setInterval(sendDetails, 10000); // call sendDetails() after every 10 seconds
2539. **function** sendDetails(){
2541. **const** no = Math.random(); // generates a random number in the range of 0 to less than 1
2542. **const** active = no < 0.8;
2543. let CO2Lvl;
2544. let smokeLvl;
2545. **if** (active) { // if the sensor is active, assign a random number between 1 to 10 for CO2 level and smoke level
2546. **const** no = Math.random(); // generates a random number in the range of 0 to less than 1
2547. **if** (no > 0.9) {
2548. CO2Lvl = randomInt(6, 10); // generates a random number between 6 & 10
2549. smokeLvl = randomInt(6, 10); // generates a random number between 6 & 10
2550. } **else** {
2551. CO2Lvl = randomInt(1, 5); // generates a random number between 1 & 5
2552. smokeLvl = randomInt(1, 5); // generates a random number between 1 & 5
2553. }
2555. } **else** { // if the sensor is not active, consider CO2 level and smoke level as -1
2556. CO2Lvl = -1;
2557. smokeLvl = -1;
2558. }
2560. **const** sensor = { // create a sensor object with updated sensor details
2561. CO2Lvl: CO2Lvl,
2562. active: active,
2563. id: 1,
2564. location: "F1-R2",
2565. smokeLvl: smokeLvl
2566. };
2568. // ajax request to update sensor details
2569. jQuery.ajax({
2570. url: "http://localhost:8080/rest/webapi/sensors/update",
2571. type: "PUT",
2572. contentType: "application/json",
2573. dataType:'json',
2574. data: JSON.stringify(sensor),
2575. success: **function**(data, textStatus, errorThrown) {
2577. $("#location").text(data.location);
2579. **if** (data.CO2Lvl === -1)
2580. $("#CO2Lvl").text("No signal");
2581. **else**
2582. $("#CO2Lvl").text(data.CO2Lvl);
2584. **if** (data.smokeLvl === -1)
2585. $("#smokeLvl").text("No signal");
2586. **else**
2587. $("#smokeLvl").text(data.smokeLvl);
2589. $("#active").text(data.active);
2591. },
2592. error : **function**(jqXHR, textStatus, errorThrown) {
2593. },
2594. timeout: 120000,
2595. });
2596. }
2598. // function to generate a random number between two values
2599. **function** randomInt(min, max) {
2600. **return** Math.floor(Math.random() \* (max - min + 1) + min);
2601. }
2602. <!DOCTYPE html**>**
2603. **<html** lang="en"**>**
2604. **<head>**
2605. **<meta** charset="UTF-8"**>**
2606. **<title>**Sensor 1**</title>**
2607. **<script** src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"**></script>**
2608. **<script** src="send-sensor-details.js"**></script>**
2609. **<link** rel="stylesheet" href="sensor.css"**>**
2610. **</head>**
2611. **<body>**
2612. **<h1>**Sensor 1**</h1>**
2613. **<hr>**
2614. **<h2>**Location: **<span** id="location"**></span></h2>**
2615. **<h2>**CO2 Level: **<span** id="CO2Lvl"**></span></h2>**
2616. **<h2>**Smoke Level: **<span** id="smokeLvl"**></span></h2>**
2617. **<h2>**Active: **<span** id="active"**></span></h2>**
2618. **</body>**
2619. **</html>**
2620. import React, {Component} from 'react';
2621. import axios from 'axios';
2623. class App extends Component{
2625. constructor() {
2626. super();
2627. this.state = {
2628. sendEmail: false
2629. };
2630. }
2632. componentDidMount() {
2633. this.getStatus(); // initially call loadSensorDetails()
2634. setInterval(this.getStatus, 5000); // call loadSensorDetails() after every 5 seconds
2635. setInterval(this.sendEmailFunction, 15000); // call loadSensorDetails() after every 15 seconds
2636. }
2638. // method to get sensor details of all the sensors
2639. getStatus = async () =**>** {
2640. try {
2641. const res = await axios.get('http://localhost:8080/rest/webapi/emails'); // make a GET request to get status
2642. const sendEmail = res.data; // assign res.data to sensorEmail
2643. this.setState({
2644. sendEmail: sendEmail.sendEmail // set state
2645. });
2646. console.log(this.state.sendEmail);
2647. } catch (err) {
2648. console.log(err) // log errors if there are any
2649. }
2650. };
2652. sendEmailFunction = () =**>** {
2653. if (this.state.sendEmail) // display alert message if sendEmail is true
2654. alert("Email sent")
2655. };
2657. render() {
2658. return (
2659. **<div>**
2660. **<h1>**Email service**</h1>**
2661. **</div>**
2662. );
2663. }
2664. }
2666. export default App;
2667. **import** React, {Component} from 'react';
2668. **import** axios from 'axios';
2670. **class** App **extends** Component{
2672. constructor() {
2673. **super**();
2674. **this**.state = {
2675. sendEmail: **false**
2676. };
2677. }
2679. componentDidMount() {
2680. **this**.getStatus(); // initially call loadSensorDetails()
2681. setInterval(**this**.getStatus, 5000); // call loadSensorDetails() after every 5 seconds
2682. setInterval(**this**.sendEmailFunction, 15000); // call loadSensorDetails() after every 15 seconds
2683. }
2685. // method to get sensor details of all the sensors
2686. getStatus = async () => {
2687. **try** {
2688. **const** res = await axios.get('http://localhost:8080/rest/webapi/emails'); // make a GET request to get status
2689. **const** sendEmail = res.data; // assign res.data to sensorEmail
2690. **this**.setState({
2691. sendEmail: sendEmail.sendEmail // set state
2692. });
2693. console.log(**this**.state.sendEmail);
2694. } **catch** (err) {
2695. console.log(err) // log errors if there are any
2696. }
2697. };
2699. sendEmailFunction = () => {
2700. **if** (**this**.state.sendEmail) // display alert message if sendEmail is true
2701. alert("Email sent")
2702. };
2704. render() {
2705. **return** (
2706. <div>
2707. <h1>Email service</h1>
2708. </div>
2709. );
2710. }
2711. }
2713. **export** **default** App;