

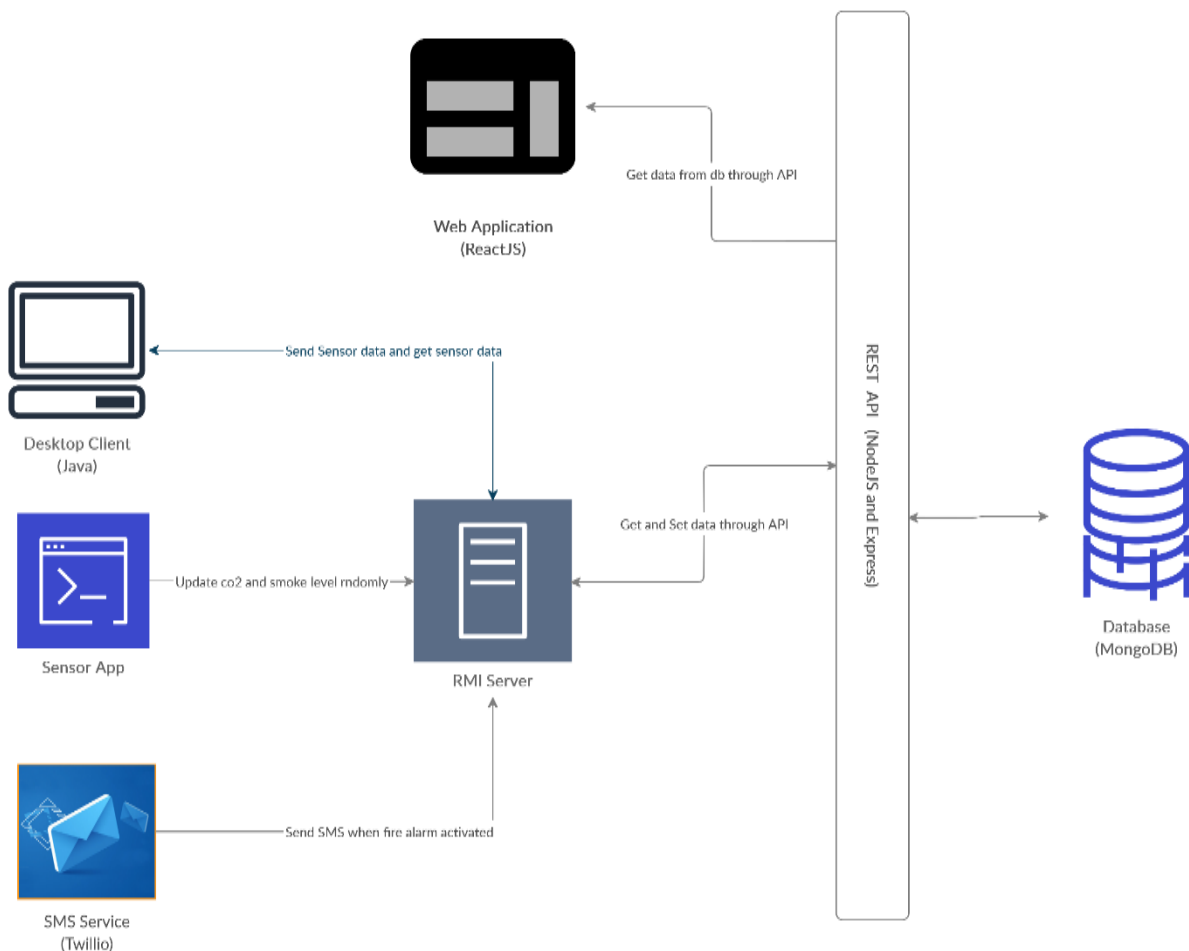
## **Fire Alarm Monitoring System**

### **Assignment 2** **Distributed System**

<b>Registration Number</b>	<b>Name</b>
IT18001808	S.W.P.N.M.Weerasinghe (Leader)
IT18012866	L.G.I.Sathsara
IT17084796	H.S.D.N.Gunasekara
IT18066944	R.M.D.D.Rajapaksha


## 1) High Level Architectural Diagram


Fire Alarm Monitoring System contains REST API, RMI server and client application, Web client and a database. REST API has implemented using Node Js and Express Js and it handles the http requests which came from the web client and also it handles http requests which came from the RMI server. Web client has implemented using React Js and it displays the details of fire alarms. Web client send http requests to REST API and REST API response with the details which obtained from the database. RMI server and client is a desktop application which use to monitor the fire alarm sensors. Desktop client can send http requests through RMI server and those requests are handled by REST API. The RMI sensor checks API from 15 seconds and update the desktop client. If gas levels have increased, the RMI sensor sends a SMS alert. Desktop client can add or edit sensor details. A dummy app called Sensor App has used to simulate the behaviors of the sensors.



## 2) System Interfaces

### I. Web Application


**Fire Alarm System**




## Fire Alarm System

Use Technology for Safety | Get Know Before Burn

Current Status : Danger : Alarm Activated

### Fire Alarms Details

Sensor Id	Floor	Room	CO <sub>2</sub> Level	Smoke Level	Alarm Status
101	1 Floor	1 Room	7.5	5.5	On
102	2 Floor	1 Room	5.5	6.5	On
120	1 Floor	2 Room	2.5	2.5	Off
121	2 Floor	4 Room	2.5	2.5	Off
122	2 Floor	3 Room	1.5	1.5	Off


 Fire Alarm System | © 2020 Copyright Reserved

### II. Desktop Application

### Fire Alarm Sensor Details

Sensor ID	Floor Number	Room Number	Smoke Level	Carbon Dioxide Level
101	1	1	5.5	7.5
102	2	1	6.5	5.5
120	1	2	2.5	2.5
121	2	4	2.5	2.5
122	2	3	1.5	1.5

Add/Edit Sensor

Message

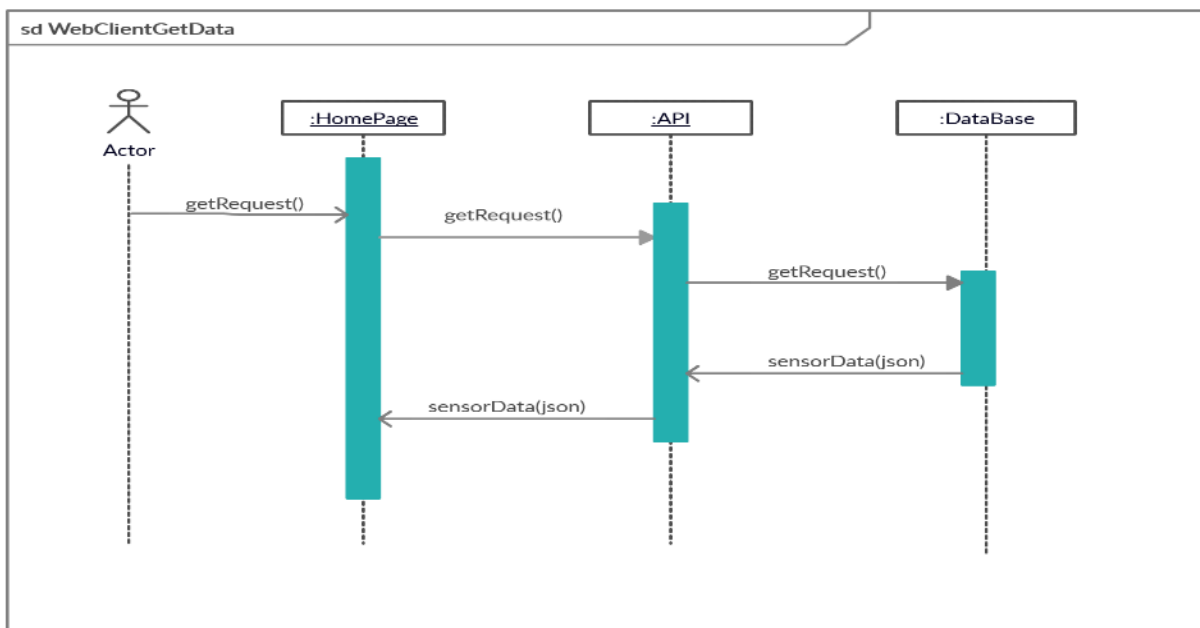
**Warning!!**  
Smoke Level is increasing

OK

## 3) System Workflow

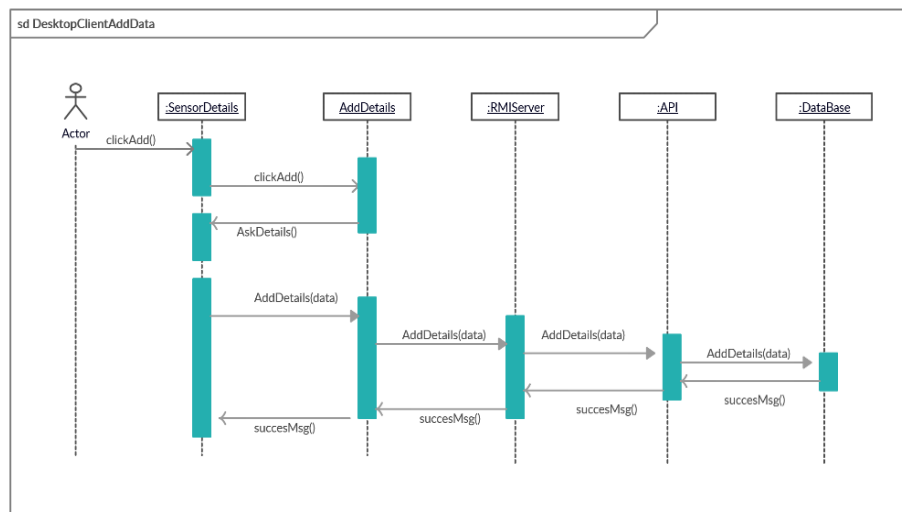
### I. Web Application

Web client sends http get() requests to the API and API handles that request by retrieving sensor details from the database and send http response to the client in json format.



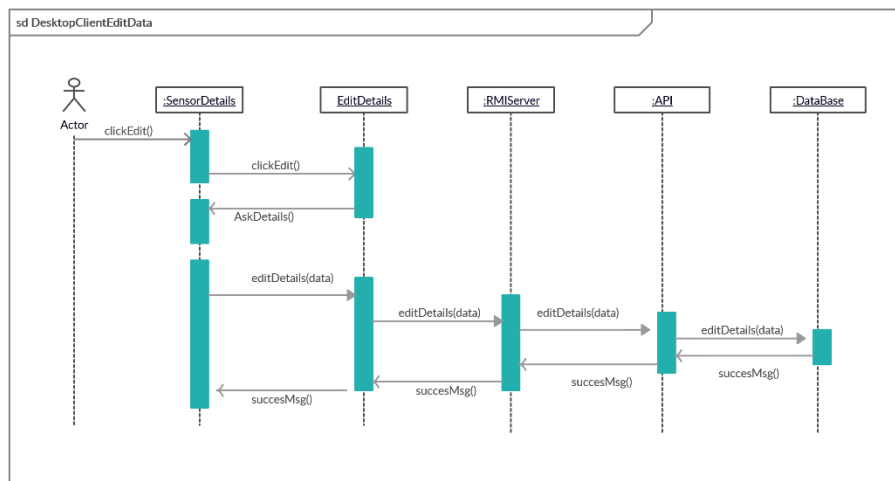
### II. Desktop Application – Add Sensor

New sensors can register to the system through desktop client. RMI server send http post() request to the API and it stores the new sensor in the database. After that API sends an success msg to the client through RMI server.



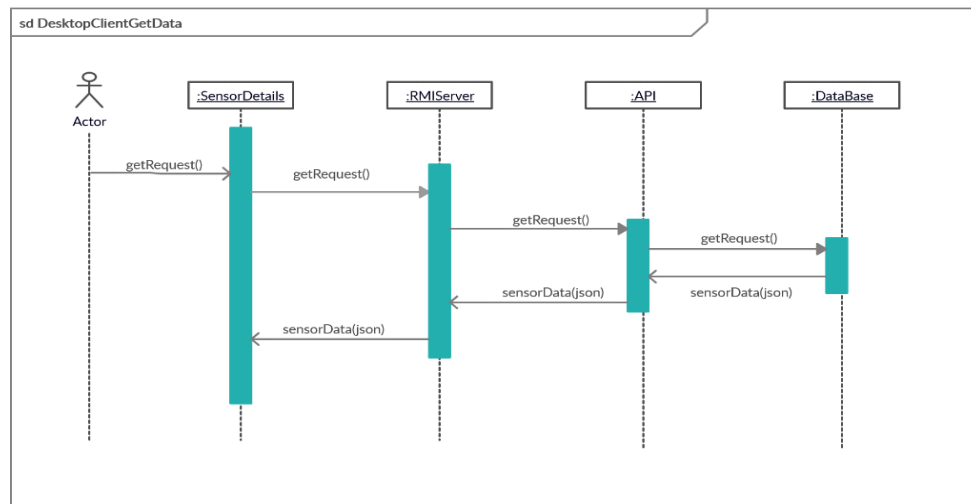
### III. Desktop Application – Edit Sensor

Sensor details can edit through RMI server. RMI server send http request to the API and it stores the new details of edited sensor in the database. After that API sends an success msg to the client through RMI server.



### IV. Desktop Application – Get Data

RMI server send http get() request to the API and API retrieve the sensor details from the database and sends them to the RMI server and desktop client displays the details in a table which retrieve from the server.



## 4) Appendix

### I. Fetch API data to web application

```
const getSensors = async () => {
  const response = await fetch('http://localhost:4000/sensors');
  const data = await response.json();
  setSensors(data);
}
```

### II. Assign Fetch data to the web application table

```
const SensorDetails = ({ sensorDetail }) => {
  return (
    <div className="container">
      <center><h2>Fire Alarms Details</h2></center><br />
      <table className="table table-hover border-rounded mb-3" style={{ textAlign: 'center' }}>
        <thead className="thead-dark">
          <tr>
            <th scope="col">Sensor Id</th>
            <th scope="col">Floor</th>
            <th scope="col">Room</th>
            <th scope="col">CO<sub> 2</sub> Level</th>
            <th scope="col">Smoke Level</th>
            <th scope="col">Alarm Status</th>
          </tr>
        </thead>
        <tbody>
          {sensorDetail.map(sensor => (
            <tr>
              <td>{sensor.sensorid}</td>
              <td>{sensor.floor} Floor</td>
              <td>{sensor.room} Room</td>
              <td>{sensor.colevel}</td>
              <td>{sensor.smokelevel}</td>
              <td><button className={ (sensor.colevel>5 || sensor.smokelevel>5) ? "btn btn-danger" : "btn btn-
success" }>
                {(sensor.colevel>5 || sensor.smokelevel>5) ? "On" : "Off" }
              </button></td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  )
}
```

**III. Establish Database connection in API**

```
//inside .env file
DB_CONNECTION=mongodb+srv://fireadmin:Fire1234@fas-tidp1.mongodb.net/Alarms?retryWrites=true&w=majority
//inside app.js
mongoose.connect(
  process.env.DB_CONNECTION,
  {useNewUrlParser:true,useUnifiedTopology: true},
  ()=>console.log('Database connected!')
)
```

**IV. Create SensorDetails & User Model to store sensor details**

```
const mongoose = require('mongoose');
const PostSchema = mongoose.Schema({
  sensorid: {type:Number,required:true},
  floor: {type:Number,required:true},
  room: {type:Number,required:true},
  colevel: {type:Number,default:0},
  smokelevel: {type:Number,default:0}
});
module.exports=mongoose.model('SensorDetails',PostSchema);

const UserSchema = mongoose.Schema({
  userid: {type:Number,required:true},
  username: {type:String,required:true},
  password: {type:String,required:true}
});
```

**V. Post method for send Sensor & User to database in API**

```
router.post('/',(req,res)=>{
  const sensor = new SensorDetails({
    sensorid:req.body.sensorid,
    floor:req.body.floor,
    room:req.body.room,
    colevel:req.body.colevel,
    smokelevel:req.body.smokelevel
  });
  sensor.save()
  .then(data=>{
    res.json(data);
  })
  .catch(err=>{
    res.json({message:err})
  })
});

router.post('/',(req,res)=>{
  const user = new UserDetails({
    userid:req.body.userid,
    username:req.body.username,
    password:req.body.password
  });
  user.save()
  .then(data=>{
    res.json(data);
  })
  .catch(err=>{
    res.json({message:err})
  })
});
```

**VI. Get Method for get Sensor & User details in API**

```
router.get('/',async (req,res)=>{
  try{
    const sensors = await SensorDetails.find();
    res.json(sensors);
  }catch(err){
    res.json({message:err})
  }
});

router.get('/',async (req,res)=>{
  try{
    const users = await UserDetails.find();
    res.json(users);
  }catch(err){
    res.json({message:err})
  }
});
```

**VII. Update sensor details method in API**

```
router.post('/:sensorid', async (req, res) => {
  try {
    const updateSensor = await SensorDetails.updateOne({_id: req.params.sensorid},
      { $set : { sensorid: req.body.sensorid,
        floor: req.body.floor,
        room: req.body.room,
        colevel: req.body.colevel,
        smokelevel: req.body.smokelevel } });
    res.json(updateSensor);
  } catch (err) {
    res.json({ message: err });
  }
})
```

**VIII. sensorApp.java**

```
//generate random number
Random r = new Random();
for(;;) {
  try {
    Thread.sleep(1000);
  }
  catch (InterruptedException ie) {
  }
  this.updateArrayList();
  num = r.nextInt();
  for (Sensor s: sensorList) {
    double smoke = s.getSmokeLevel();
    double cd = s.getCdLevel();
    if (num > 0) {
      smoke = smoke - 0.5;
      cd = cd - 0.5;
      if (smoke > 0 && cd > 0) {
        s.setCdLevel(cd);
        s.setSmokeLevel(smoke);
      }
    }
    else if (num < 0) {
      smoke = smoke + 0.5;
      cd = cd + 0.5;
      s.setCdLevel(cd);
      s.setSmokeLevel(smoke);
    }
  }
}
```



```
//send change value to db
```

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://localhost:4000/sensors");
JSONParser parser = new JSONParser();
Object obj;
try {
    obj = parser.parse(target.request(MediaType.TEXT_XML).get(String.class));
    JSONArray array = (JSONArray)obj;
    for(int i = 0 ;i<array.size();++i) {
        JSONObject obj2 = (JSONObject)array.get(i);
        Sensor sensor = new Sensor();
        sensor.set_id(obj2.get("_id").toString());
        sensor.setId(obj2.get("sensorid").toString());
        sensor.setFloorNo(new Integer(obj2.get("floor").toString()));
        sensor.setRoomNo(obj2.get("room").toString());
        sensor.setSmokeLevel(new Double( obj2.get("smokelevel").toString()));
        sensor.setCdLevel(new Double( obj2.get("colevel").toString()));
        sensorList.add(sensor);
    }
}
```

#### **IX. fireAlarmSensorServer.java**

```
public void saveSensorsToList() {
    Client client = ClientBuilder.newClient();
    WebTarget target = client.target("http://localhost:4000/sensors");
    JSONParser parser = new JSONParser();
    Object obj;
    try {
        obj = parser.parse(target.request(MediaType.TEXT_XML).get(String.class));
        JSONArray array = (JSONArray)obj;
        List<Sensor> sList = new CopyOnWriteArrayList<Sensor>();
        for(int i = 0 ;i<array.size();++i) {
            JSONObject obj2 = (JSONObject)array.get(i);
            Sensor sensor = new Sensor();
            sensor.set_id(obj2.get("_id").toString());
            sensor.setId(obj2.get("sensorid").toString());
            sensor.setFloorNo(new Integer(obj2.get("floor").toString()));
            sensor.setRoomNo(obj2.get("room").toString());
            sensor.setSmokeLevel(new Double( obj2.get("smokelevel").toString()));
            sensor.setCdLevel(new Double( obj2.get("colevel").toString()));
            sList.add(sensor);
        }
        sensorList = sList;
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

**X. Get admin details for authentication**

```
public void getAdminsFromApi() {
    client = ClientBuilder.newClient();
    WebTarget target = client.target("http://localhost:4000/users");
    JSONParser parser = new JSONParser();
    Object obj;
    try {
        obj = parser.parse(target.request(MediaType.TEXT_XML).get(String.class));
        JSONArray array = (JSONArray)obj;
        for(int i = 0 ;i<array.size();++i) {
            JSONObject obj2 = (JSONObject)array.get(i);
            Admin admin = new Admin();

            admin.setUserName(obj2.get("username").toString());
            admin.setPassword(obj2.get("password").toString());
            adminList.add(admin);
        }
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

**Xi ) Send SMS notification when smoke and co2 increase**

```
public void sendSMS(int floor, String room, double smokeLevel, double cdLevel) throws RemoteException {
    Twilio.init(ACCOUNT_SID, AUTH_TOKEN);
    Message.creator(
        new com.twilio.type.PhoneNumber("+94715443619"),
        new com.twilio.type.PhoneNumber("+13343731533"),
        "Warning!!\n"+"Smoke Level and carbon Dioxide level is increasing\n"+"Floor Number:"+floor+"\n"+"Room
Number:"+room)
        .create();
}
```

**END.**