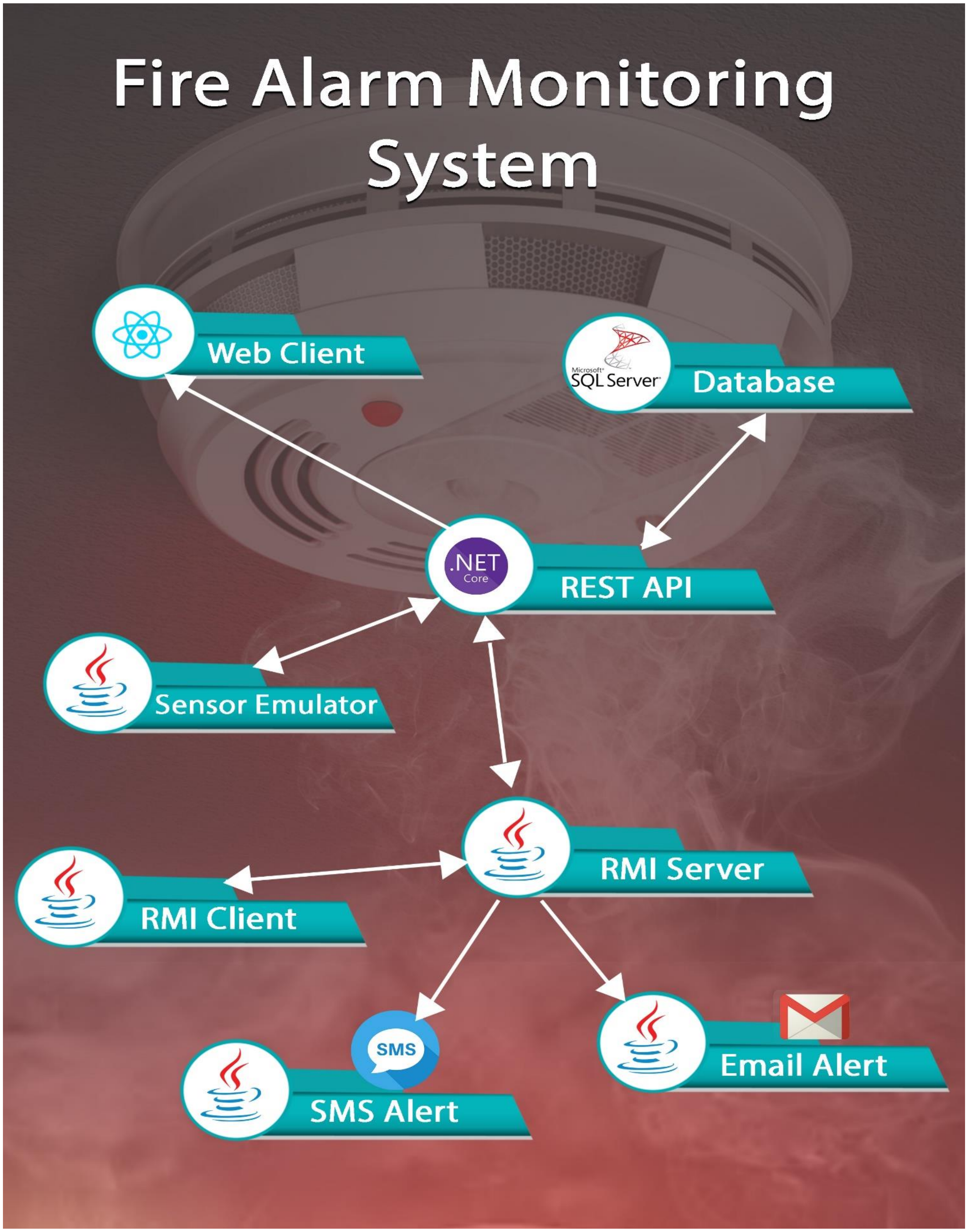
FIRE ALARM MONITORING SYSTEM

IT18178678 - SENEVIRATHNE S.M.A.S.
IT18178500 - PERERA K.K.P.
IT18180626 - WELAGEDARA H.M.A.N.
IT18084146 ABEYSINGHES.M.D.N.


Fire Alarm Monitoring System
Web Client
Database
SQL Server
.NET Core
REST API
Sensor Emulator
RMI Server
RMI Client
SMS
SMS Alert
Email Alert

Abstract

This report depicts the process of an implemented fire alarm monitoring system. This system mainly consists of a web client application and a desktop client application. The main components of this system are API, RMI server, Web Client, RMI Client and Sensor Simulator. Admin and the normal user can login to the system. After sign in admin has the access to register new fire alarm sensors, delete and edit those sensors. Web client can log in to the system and check the status of the fire alarm system, the location, smoke level or CO2 level. All these sensor details are updated in every 10 seconds. Smoke level and the CO2 level of the room is given as a number (1 - 10). If the CO2 and Smoke level of a particular room is higher than 5, an email and a message should be sent immediately.

Introduction

This fire alarm system mainly consists of an Admin, Web Client, RMI Client, Sensor Simulator and an API. The end points are defined as sign in, Get sensor details, Register sensor, Edit sensor, Delete sensor and Set sensor status.

In this system admin can have the access to the API. When the admin supplies the username and the password the access token is generated through this system. The access token is appended with the response body. Therefore, in this system all the authentications are done through an access token. This helps to improve the security of the system. Admin can access register, edit and delete all the sensors. There are two types of clients in this system. First type of client is web client. Web client post a login request and then a JSON object is passed for the client as a token. This web client can send get sensor details request for the API.

The other client is the RMI client. RMI client connects with the RMI server. Admin and a normal user who connect as a desktop client can be considered as a RMI client. RMI client can connect with the RMI server. RMI server checks the sensor status every 15 seconds to take the updated readings. Then a normal user can get sensor details. Admin can register, edit and delete sensors. This RMI server sends emails/ updates when relevant event happened. Here in this system the used SMS service offers this service only for the verified numbers. Sensor simulator take all the details of the room. A relational database is used for storing all sensor details. Then simple client applications have been implemented. These applications send fire alarm status every 10 seconds.

The frontend implementations of the system are designed in React. API is designed with .net core. Data base is designed with Microsoft sql server.

CONTENTS

Chapter 1 : High Level Architecture Diagram 4

Chapter 2 : API endpoints 5

Chapter 3 : API 6

Chapter 4 : Security of the Fire Alarm Monitoring system 9

Chapter 5 :Sensor 11

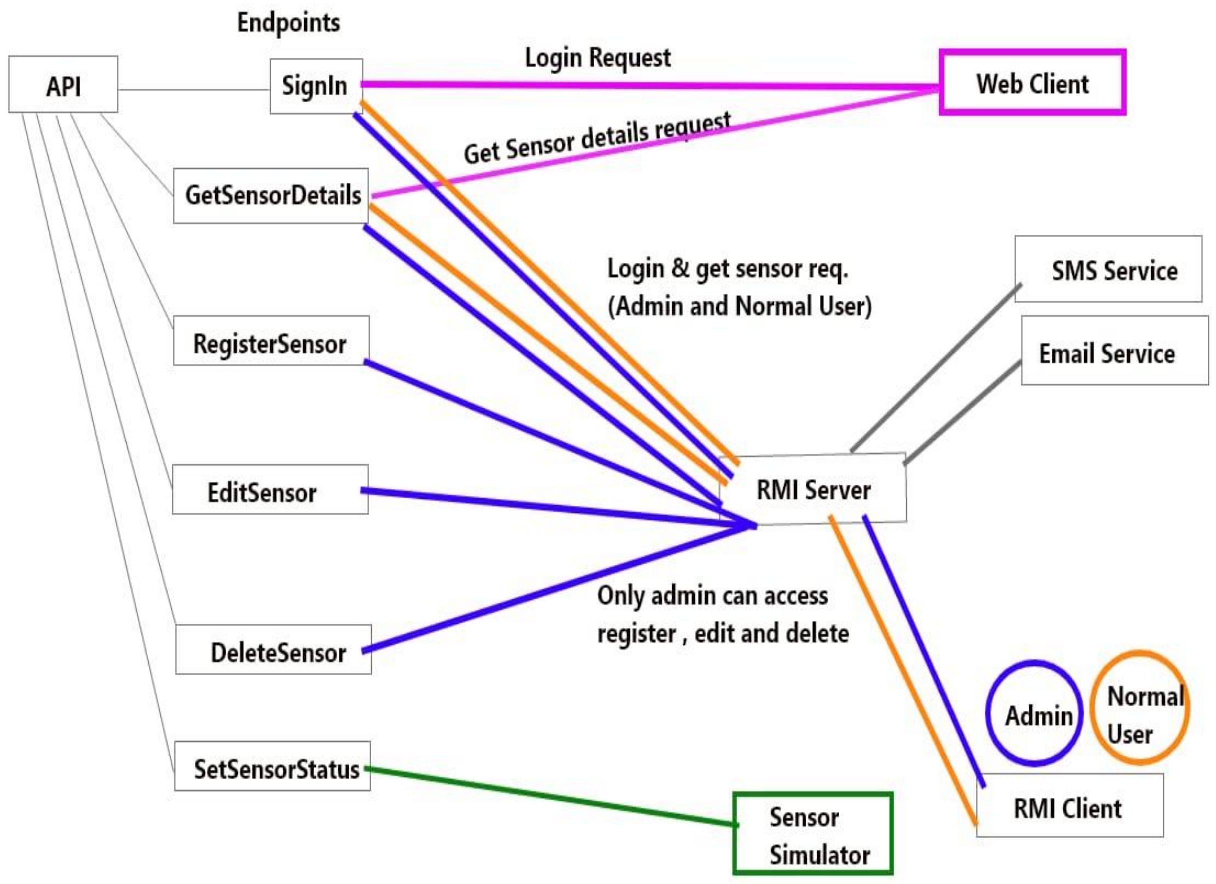
Chapter 6 : Web Client 13

Chapter 7 : RMI Client and Server 15

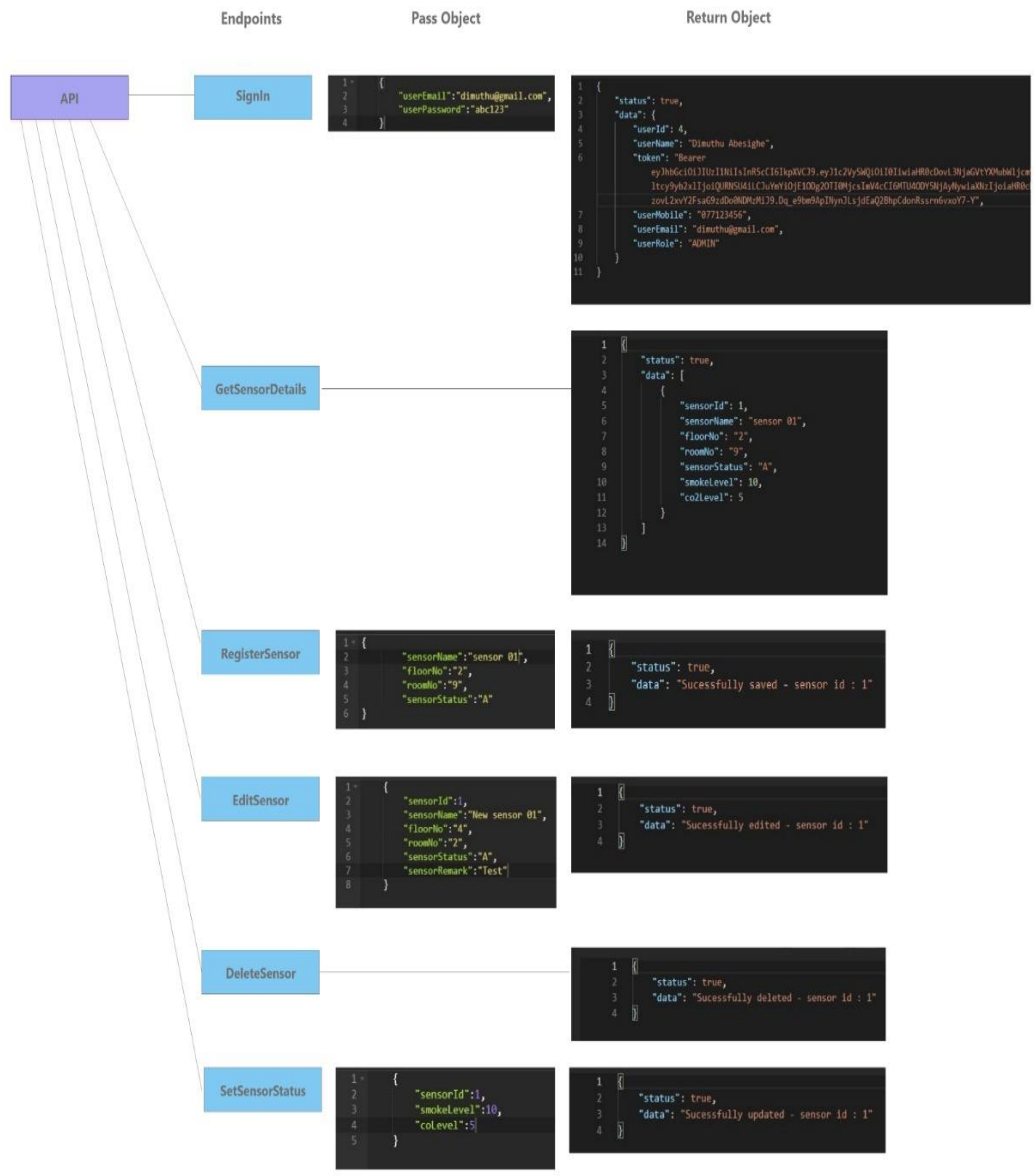
Chapter 8 :Tools and Technology 18

Chapter 9 :Appendix 19

CHAPTER 1 : HIGH LEVEL ARCHITECTURE DIAGRAM



CHAPTER 2 : API ENDPOINTS



```
    {
        "userEmail":"dimuthu@gmail.com",
        "userPassword":"abc123"
    }
```

```
{
    "status": true,
    "data": {
        "userId": 4,
        "userName": "Dimuthu Abesighe",
        "token": "Bearer
            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI0IiwiaHR0cDovL3NjaGVtYXMubWljcm
            ltcy9yb2xlIjoiQURNSU4iLCJuYmYiOjE1ODg2OTI0MjcsImV4cCI6MTU4ODY5NjAyNywiaXNzIjoiaHR0c
            zovL2xvY2FsaG9zdDo0NDMzMiJ9.Dq_e9bm9ApINynJLsjdEaQ2BhpCdonRssrn6vxoY7-Y",
        "userMobile": "077123456",
        "userEmail": "dimuthu@gmail.com",
        "userRole": "ADMIN"
    }
}
```

```
{
    "status": true,
    "data": [
        {
            "sensorId": 1,
            "sensorName": "sensor 01",
            "floorNo": "2",
            "roomNo": "9",
            "sensorStatus": "A",
            "smokeLevel": 10,
            "co2Level": 5
        }
    ]
}
```

```
{
        "sensorName":"sensor 01",
        "floorNo":"2",
        "roomNo":"9",
        "sensorStatus":"A"
}
```

```
{
    "status": true,
    "data": "Sucessfully saved - sensor id : 1"
}
```

```
    {
        "sensorId":1,
        "sensorName":"New sensor 01",
        "floorNo":"4",
        "roomNo":"2",
        "sensorStatus":"A",
        "sensorRemark":"Test"
    }
```

```
{
    "status": true,
    "data": "Sucessfully edited - sensor id : 1"
}
```

```
{
    "status": true,
    "data": "Sucessfully deleted - sensor id : 1"
}
```

```
    {
        "sensorId":1,
        "smokeLevel":10,
        "coLevel":5
    }
```

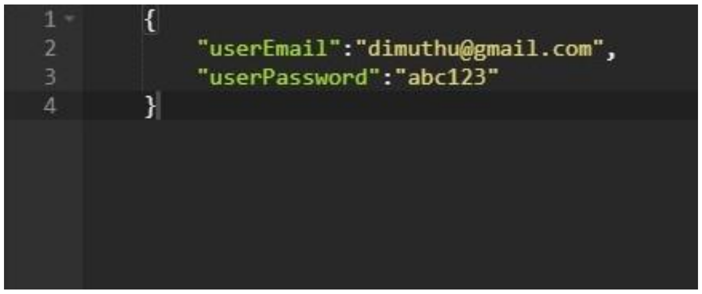
```
{
    "status": true,
    "data": "Sucessfully updated - sensor id : 1"
}
```

CHAPTER 3 : API

The API of this fire alarm monitoring system handles two main sections such as user related requests and sensor related requests. In the user related request, two main end points are being defined.

User related requests

01)SignIn - http://localhost:44332/api/user/SignIn

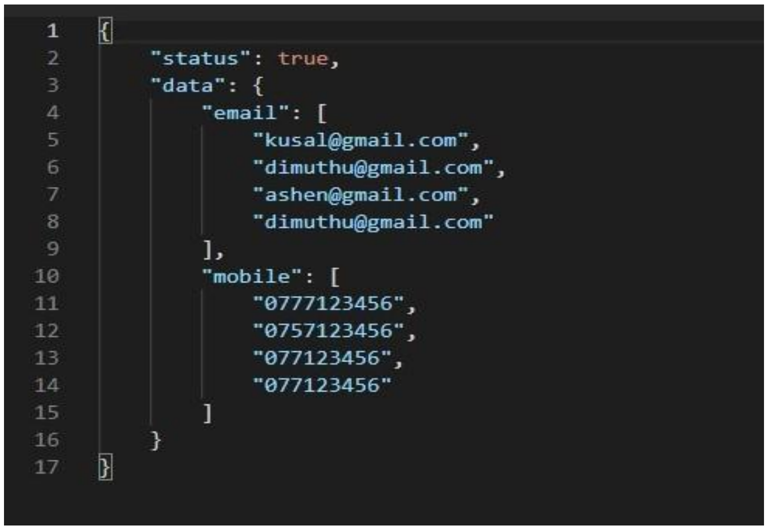


```
{
    "userEmail":"dimuthu@gmail.com",
    "userPassword":"abc123"
}
```

This endpoint is used to sign in to the API. To sign in to this API, user provides username and password. Then this username and the password are sent as a request for the API. API checks whether this username and password are user related. If username and password are user related, then API generates a token using JwtBearer token library. This token stores user information, as well as some other information like Issuer, Audience and expiry time.

02)GetUserEmailMobile - **http://localhost:44332/api/user/GetUserEmailMobile**

This endpoint is used to get the email list and the mobile number list.



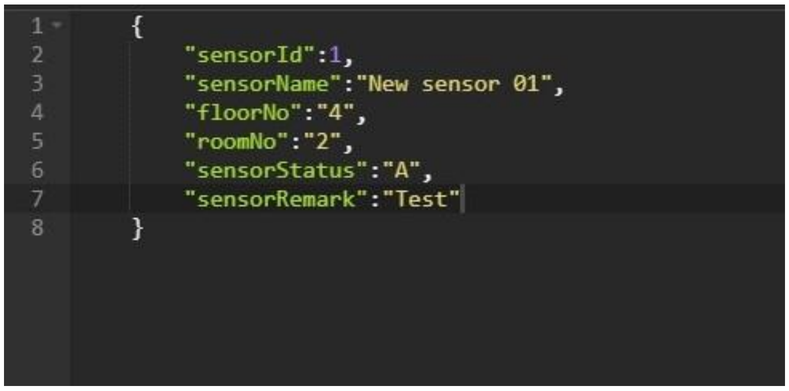
```
{
    "status": true,
    "data": {
        "email": [
            "kusal@gmail.com",
            "dimuthu@gmail.com",
            "ashen@gmail.com",
            "dimuthu@gmail.com"
        ],
        "mobile": [
            "0777123456",
            "0757123456",
            "077123456",
            "077123456"
        ]
    }
}
```

Sensor related requests

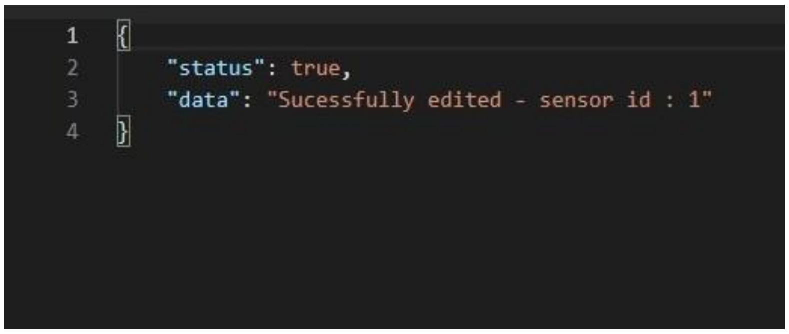
01)RegisterSensors (POST) - **http://localhost:44332/api/sensor/RegisterSensor**

This endpoint is used to register new sensor. Here the valid object should be passed. Registering sensors is a process which can be only done by admins. So access for this method is also only for admins. Therefore, token should be passed to the end point if not this will show 401 error.

02) EditSensor (PUT) - http://localhost:44332/api/sensor/EditSensor



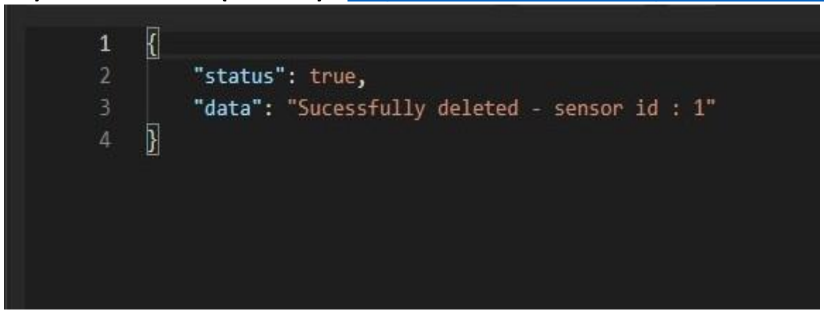
```
{
    "sensorId":1,
    "sensorName":"New sensor 01",
    "floorNo":"4",
    "roomNo":"2",
    "sensorStatus":"A",
    "sensorRemark":"Test"
}
```



```
{
    "status": true,
    "data": "Sucessfully edited - sensor id : 1"
}
```

EditSensor method is mainly used to update the sensor information. Here too a valid object should be Passed and this procedure Is also can be accessed by the admin. Token is a must in this method. A 401 error can be occurred if the token is not passed.

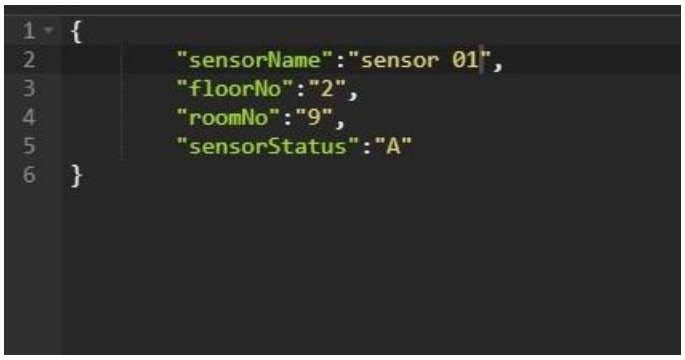
03) DeleteSensor (DELETE) - http://localhost:44332/api/sensor/DeleteSensor/5



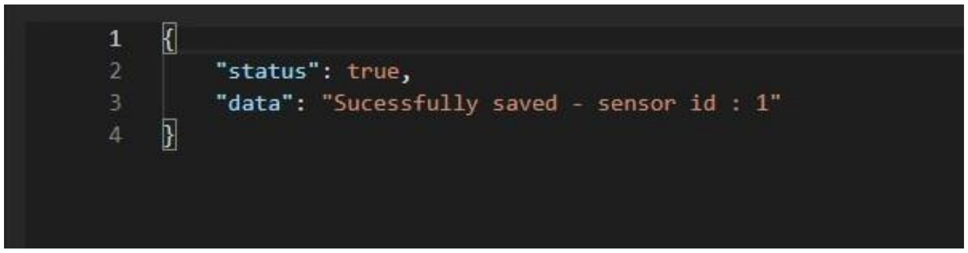
```
{
    "status": true,
    "data": "Sucessfully deleted - sensor id : 1"
}
```

This method is used to delete the registered sensor. A valid sensor is should be passed to success this process. This method is also only authorized by the admin. Token is a must in this method. A 401 error can be occurred if the token is not passed.

04) SetSensorState (POST) - http://localhost:44332/api/sensor/SetSensorState



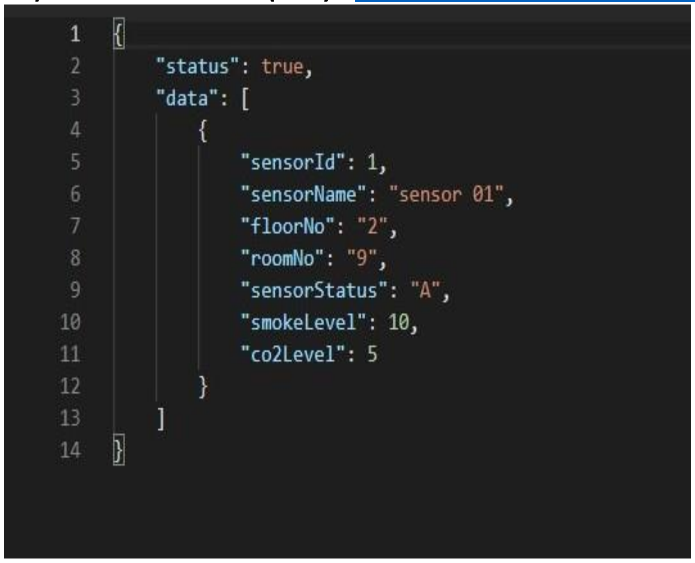
```
{
        "sensorName":"sensor 01",
        "floorNo":"2",
        "roomNo":"9",
        "sensorStatus":"A"
}
```



```
{
    "status": true,
    "data": "Sucessfully saved - sensor id : 1"
}
```

This method is used to update the sensor smoke and CO2 value. A valid sensor is should be passed to success this process. This method is also only authorized by the admin. Token is a must in this method. A 401 error can be occurred if the token is not passed.

06) GetSensorDetails (GET) - http://localhost:44332/api/sensor/GetSensorDetails



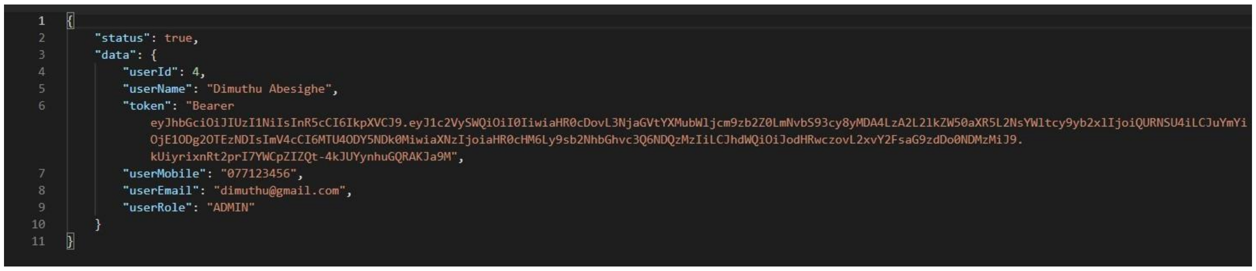
```
{
    "status": true,
    "data": [
        {
            "sensorId": 1,
            "sensorName": "sensor 01",
            "floorNo": "2",
            "roomNo": "9",
            "sensorStatus": "A",
            "smokeLevel": 10,
            "co2Level": 5
        }
    ]
}
```

Here anyone can access the end points and return all the sensor details.

CHAPTER 4 : SECURITY OF THE FIRE ALARM MONITORING SYSTEM

Usage of token

After a successful login a token can be generated based on the important details like Issuer, Audience, User claims, expiry time and signInCredintial. The token is appended with the body. Here this token is used to improve the security of the system.



```
{
    "status": true,
    "data": {
        "userId": 4,
        "userName": "Dimuthu Abesighe",
        "token": "Bearer
            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI0IiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93cy8yMDA4LzA2L21kZW50aXR5L2NsYWltcy9yb2xlIjoiQURNSU4iLCJuYmYi
            OjE1ODg2OTEzNDIsImV4cCI6MTU4ODY5NDk0MiwiaXNzIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6NDQzMzIiLCJhdWQiOiJodHRwczovL2xvY2FsaG9zdDo0NDMzMiJ9.
            kUiyrixnRt2prI7YWCpZIZQt-4kJUYynhuGQRAKJa9M",
        "userMobile": "077123456",
        "userEmail": "dimuthu@gmail.com",
        "userRole": "ADMIN"
    }
}
```

- Issuer - contain the issuer of the token
- Audiance - contain the Audience of the token
- User claims - container userid, user roles like information
- expire time - expire time of token
- signInCredintial - We define the Secret Key and encrypt the security key using "SecurityAlgorithms.HmacSha256" algorithm. Secret key is defined and encrypted.
  When user sends the request with the token, API validates the token using above mentioned factors. If token matches with these points, then this request cannot access the any authorized method from API. Also we can define Authorize method with User roles. User roles are in the User claims of the token. Invalid requests cannot be accessed by these authorized methods.

Hashing passwords

- When a new user is created, those details are stored inside the database. But if it is a String value, it is considered as the best practice. So the password can be hashed and stored inside the database. Anyone cannot read the password as it is encrypted.

Prevent SQL Injection

- Here in this system some best practices were followed to prevent sql injection.
- Using Stored Procedures instead of query for CRUD operations
- Using Object/Relation Mapping Framework (ORM)
- In this project, the entity framework is used.

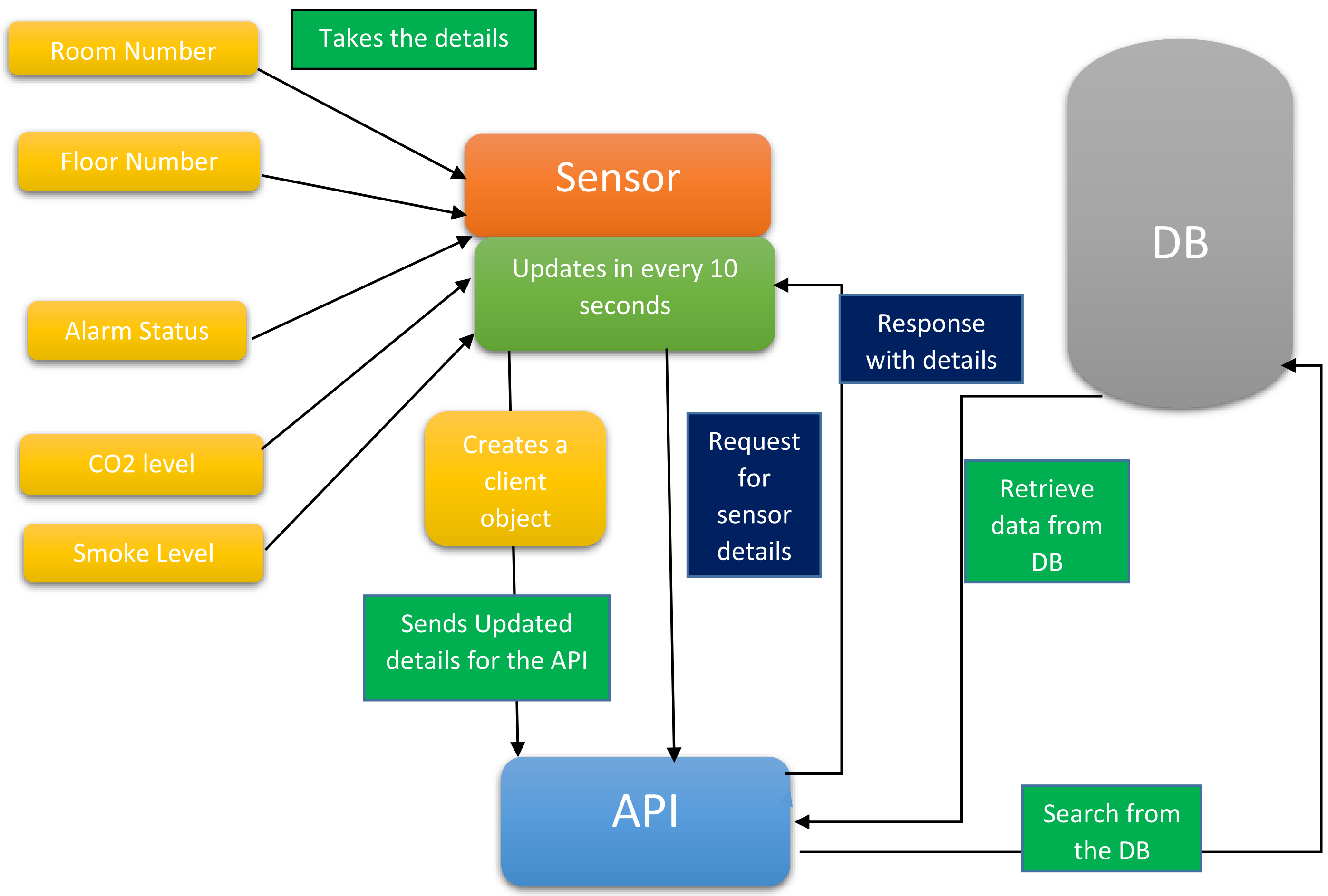
Logging

Log details can be used to ensure the request come to the API and output response for the users. structured logging tools can be used to save logging information in database. Logging details of the database is checked when an error occurred.
for example once user send request to API first of all API save the logging records with some important information like who is the user, userid , the IP of request, the endpoints of request. These informations will help when any error or security error happens.

- In the c# there are libraries like Serilogs for structured logging.

CHAPTER 5 :SENSOR

All the users of this system can be aware of latest updates which is supplied by the fire sensor, such as room number, alarm status, smoke level and CO2 level. All these sensor details should be updated with in every 10 seconds. Sensor sends updated details for the API and store them in a database. If the sensor needs to take one of the details. Then that sensor sends a request to the API. API sends a request for the database and the specific data is sending as a JSON response to the sensor.



Here, mainly a constant class has been created to store the constant values of the sensor. Then the user interface for the sensor is designed with all the relevant information in a class called SensorUI.java. This class has used java JFRAMES for the implementations. There are two model classes named Sensor.java and SensorState.java. Sensor.java is designed to contain all the information of this sensor as attributes. This class is mainly designed to represent the sensor object. Then the other class called sensorState.java is designed to represent the sensor state object.

ISensorServices.java class is created to get a list of sensor details. Sensor can take all the functional details from this class. SensorServicesImpl.java implements ISensorServices.java class. Mainly list object is created to store sensor details. Then a client object is created to send request for the API.

In this response, status should be checked. Here it checks whether the status is 200 or not. If the status is 200 then everything is considered as valid. But if the status is not equal to 200, then it is considered as an error. This response is coming as a JSON object. The status of the JSON object is checked. In the API every response consists of two attributes such as status and data. Status is the Boolean value. If it is true, that means that API responses a valid response. Data contains sensor objects. If the status is false, that means the data attribute contains an error.

SensorUI.java is created to display all the information.

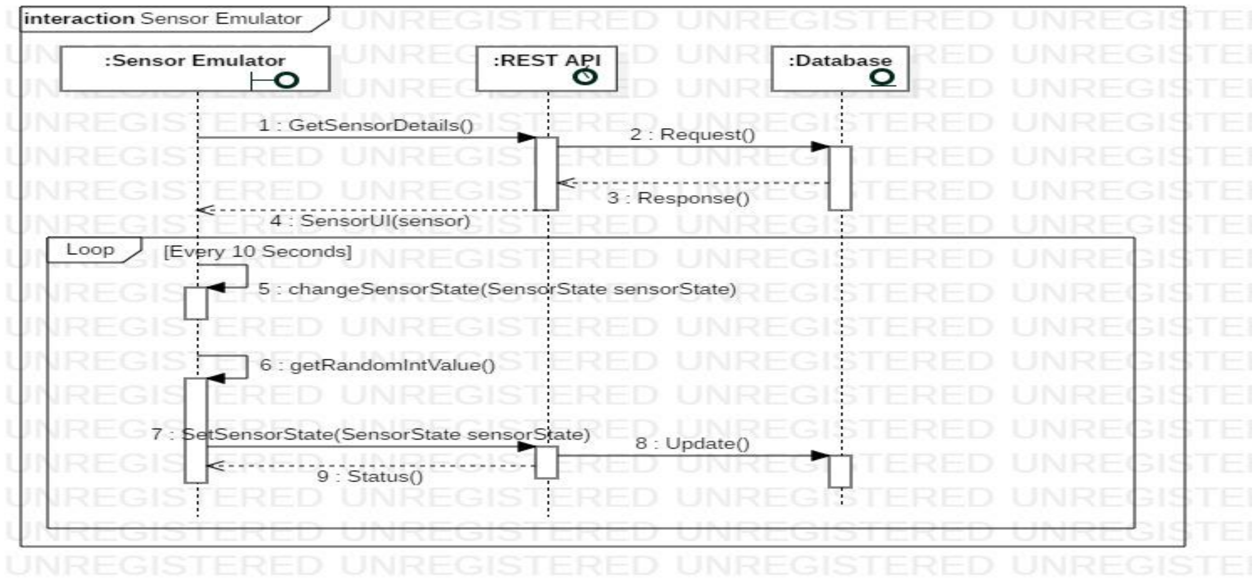
changeSensorState()

This function is implemented to handle state change of the sensor. This method is called and stimulate iSensorServices to send updated data to API.

getRandomValue()

This method is implemented to return random integer values from 0-10.

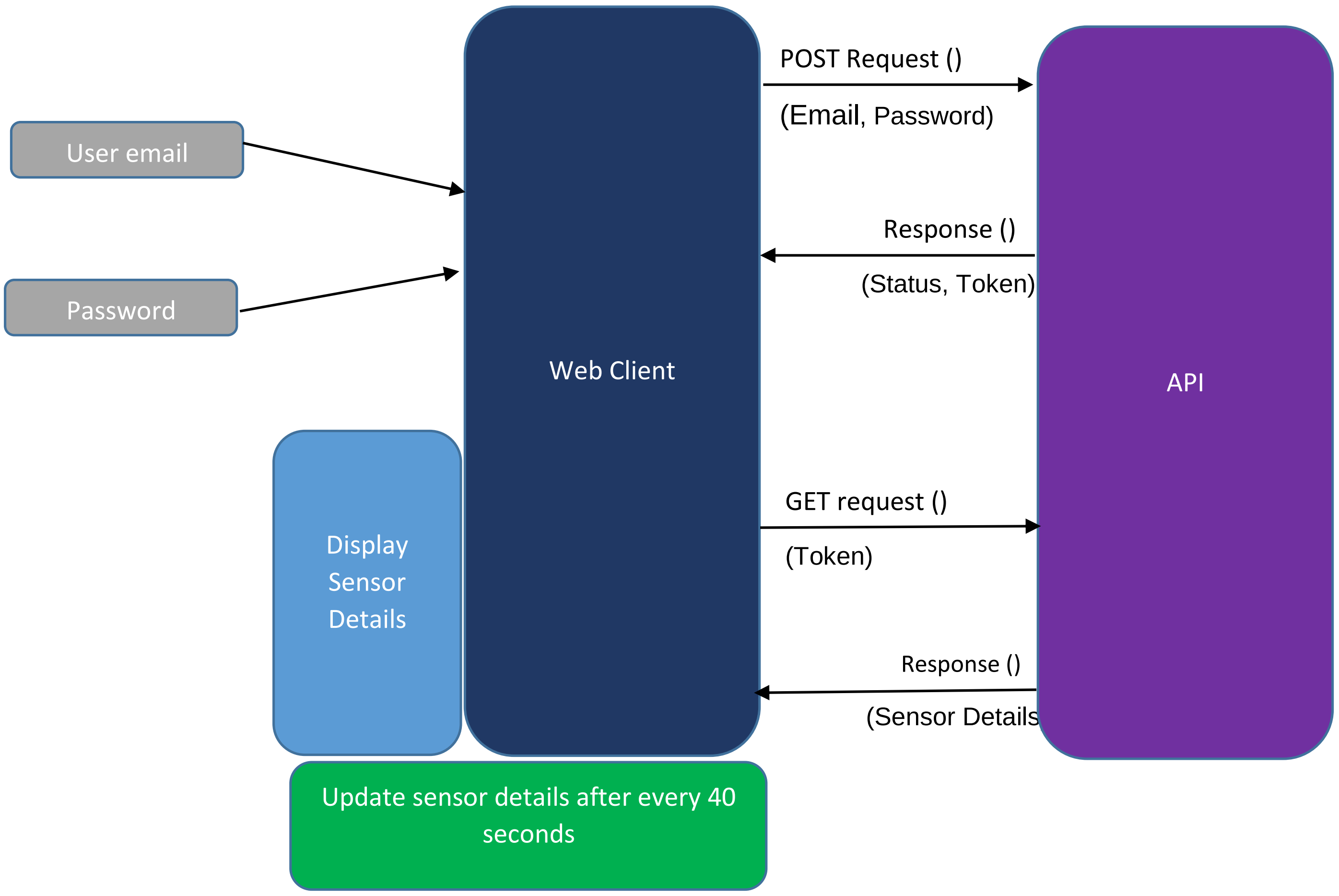
Sequence diagram for the Sensor



CHAPTER 6 : WEB CLIENT

A user can log in to the system through the log in interface. Provided email and password have been sent to the API as a POST method. Then the API response sends the status and the token. If the status is true, the client dashboard should be displayed. If it is not true, an error message is displayed.

After logged in, client app sends get request with a token (Previously got from API response) to the API. Then displays the sensor information's taken from the API. The system automatically updates the sensor information after every 40 seconds.



These two main functions handle the web client.

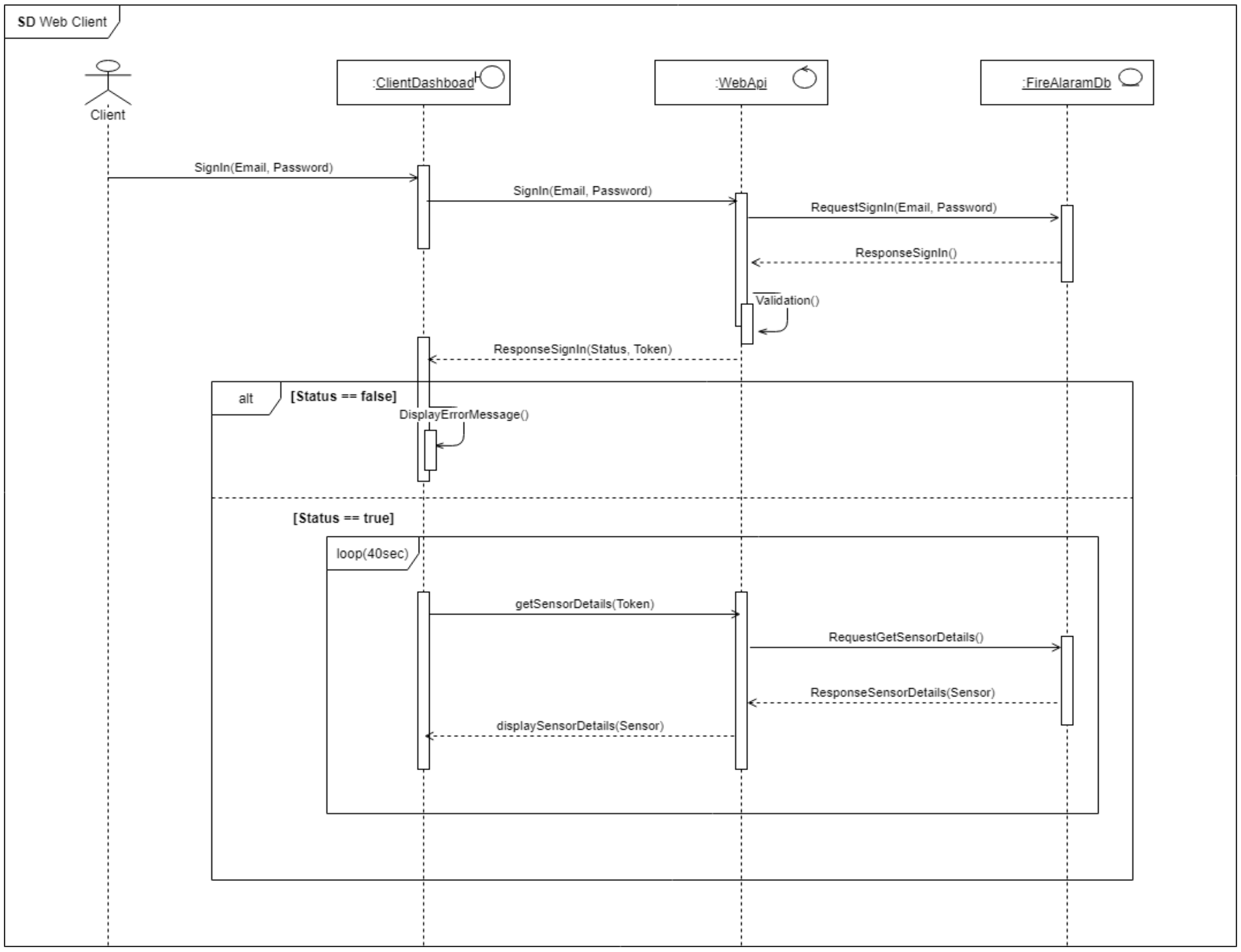
onSubmitHandler()

This method is used for sending POST request to the API with user email and password. The response gives status and token. Then check the status and when the status is true then the dashboard is displayed for the client. When this is false an error message is displayed.

fetchData()

This method is used for sending GET method to API and get sensor details. The method should be sent with previously received token. This fetchData() method is called after every 40 seconds and then the sensor details are updated automatically.

Sequence diagram for the Web Client



CHAPTER 7 : RMI CLIENT AND SERVER

First of all, RMI server creates the objects from UserServicesImpl & SensorServicesImpl classes. These classes include remote interface functions. So RMI Client can access the functions remotely. Then RMI server Binds that objects to RMI registry. RMI client get the access to that functions by using Naming. lookup("ServerURl/ObjectRegisteredName") function. Afterwards The login UI is Shown to the client. Here client can enter log in credentials.

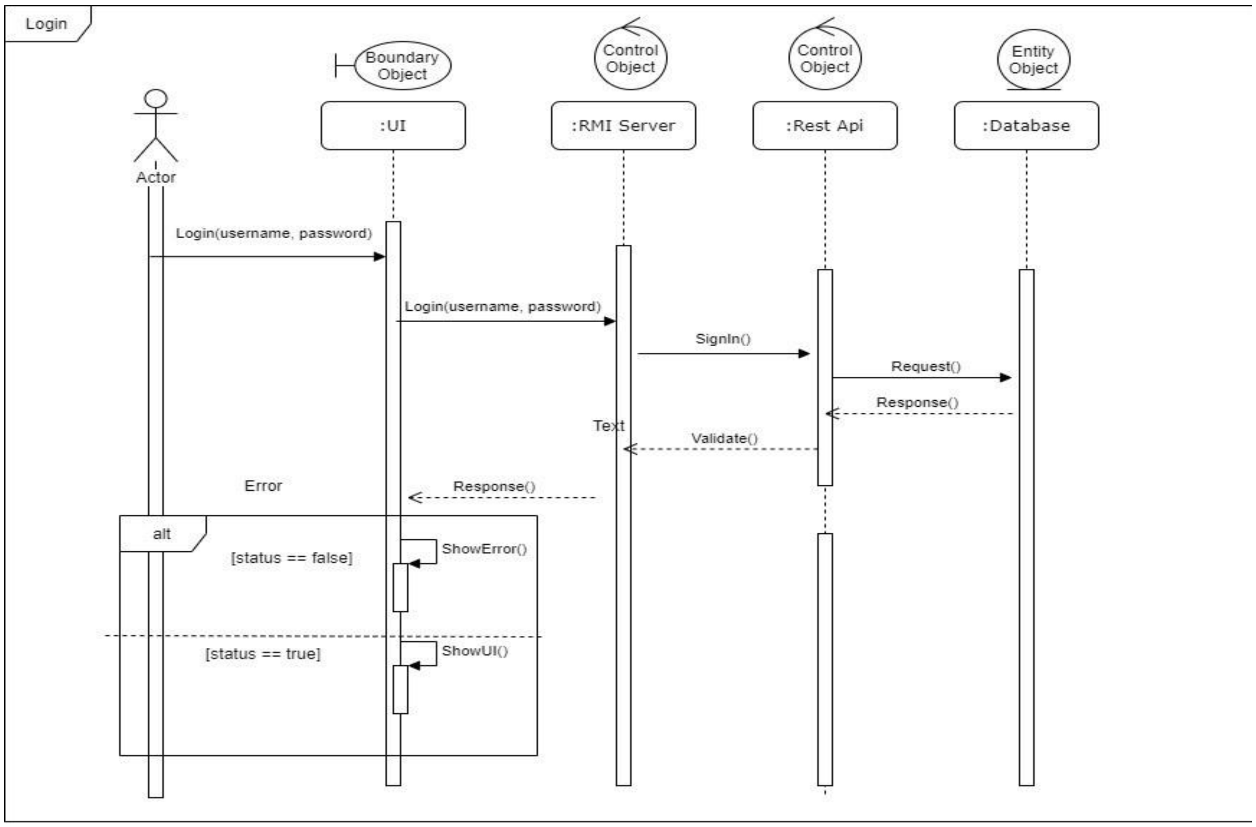
Then System sends post request to the API with log in credential data. API validates data. Then the log in settings are validated. There are two attributes in JSON response object such as "status" and "data". This token is used for the further procedures. This token is appended with the further requests. This token is used for validate these requests. If the response 200. It is considered as a success response. If it is success , this creates an object. This object is sent to the place where the function was called. If the response is an error, this object is assigned to null.

When the object is null in the called function, It displays an error message. But when it is not null it checks whether it is a normal user or the admin and displays the specific interface. After that sensor details are fetched with in a particular time interval. This is called in RMI server. Inside the RMI server it checks whether the CO2 level/smoke level is greater than 5. If the CO2 level and the smoke level are higher than 5, the function SendEmailsAndSms(JsonObject jsonResponse) is invoked.

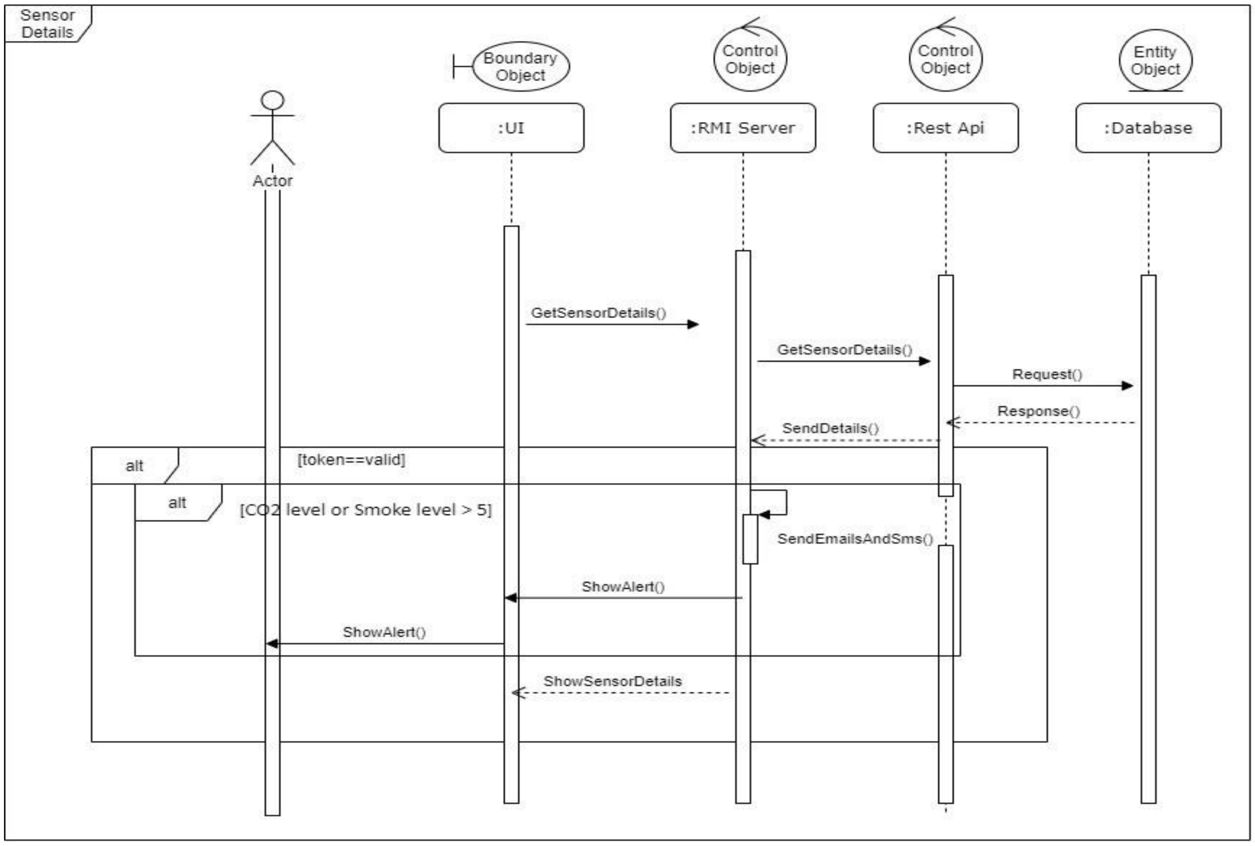
This function helps to send an email or message to the Admin. The service is allowed to send the SMS for one phone number for the testing. These sensor details are showed for the admin as well as the users. Even though, the user is an admin or a normal user the status is displayed. Admin can add, update or delete a sensor. Before requesting these the token is checked to confirm whether it is a valid or an invalid token. This courses manipulates the data from the DB if the token is valid. Then the manipulated data is displayed to the admin.

Sequence Diagram for RMI client and Server

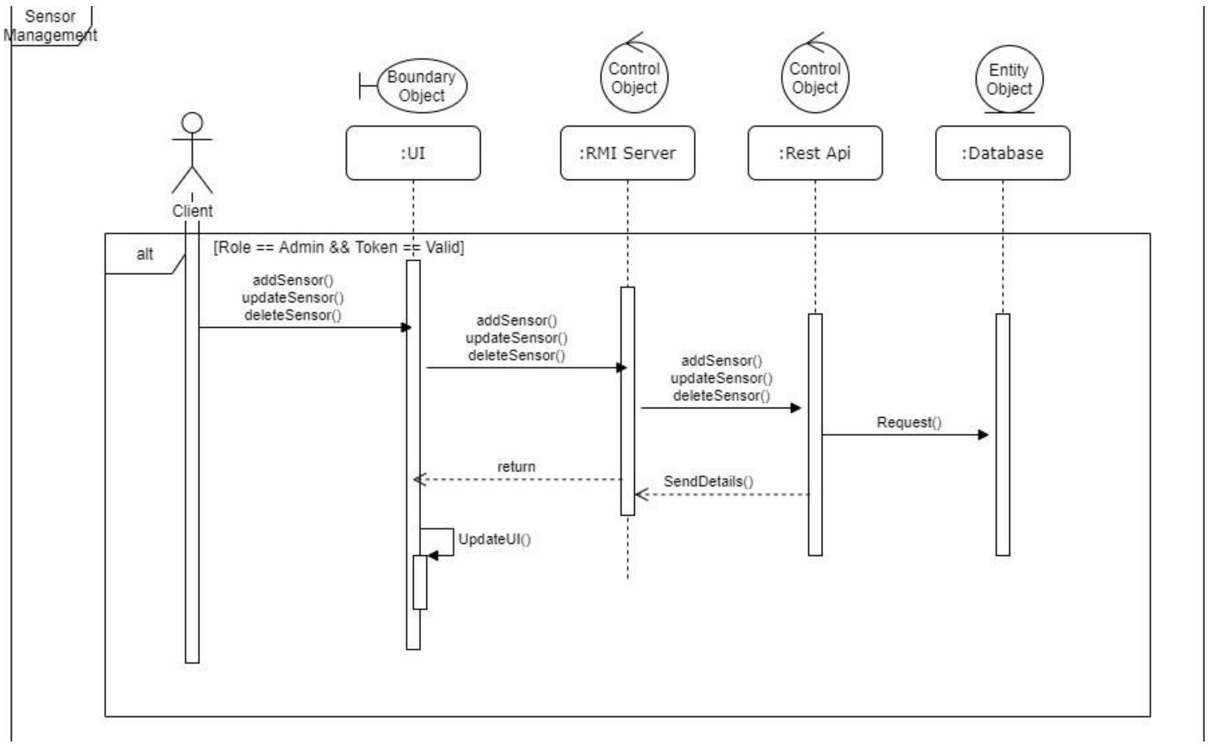
Login



Get Sensor Details



Sensor Management


Sensor Management
Boundary Object
Control Object
Control Object
Entity Object
Client
:UI
:RMI Server
:Rest Api
:Database
alt
[Role == Admin && Token == Valid]
addSensor() updateSensor() deleteSensor()
addSensor() updateSensor() deleteSensor()
addSensor() updateSensor() deleteSensor()
Request()
return
SendDetails()
UpdateUI()

CHAPTER 8 :TOOLS AND TECHNOLOGY

- Visual Studio 2019 and Sql server management studio
- Used .NET core to develop the API.
- Used entity framework to handle database related functions.
- SMS library (Twilio)
- Email Libarary (JavaMail API)
- Java and JFrame to develop RMI server,Client and Sensor emulator
- Web client React JS
- Web token library

CHAPTER 9 :APPENDIX

-----------------API--------------

File Name : SensorController

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   SensorController
 * @Description :   SensorController class inherite from the ControllerBase class. SensorController class basically handle the
 *                  all endpoints related to the Sensors.
*/

namespace FireAlarm.Web.API.Controllers
{
    // Set the API URL Path - api/SensorController
    [Route("api/[controller]")]
    [ApiController]
    public class SensorController : ControllerBase
    {
        // Create object from FireAlarmDbContext class
        private readonly FireAlarmDbContext _context;
        // Create object from ISensorService - ISensorService handle the all CRUD operations related to the sensors
        private readonly ISensorService _sensorService;

        // Constructor
        public SensorController(FireAlarmDbContext context, ISensorService sensorService)
        {
            _context = context;
            _sensorService = sensorService;
        }

        // Used to get all registered sensors in API.
        // Define API URL Path - api/SensorController/GetSensorDetails (GET)
        // Call the GetSensorDetails method inside ISensorService interface

        // Return the All registered sensor details

        [HttpGet("GetSensorDetails")]

        public async Task<ApiResult> GetSensorDetails()

        {

            return await _sensorService.GetSensorDetails();

        }


        // Used to register new sensor

        // Define API URL Path - api/SensorController/RegisterSensor (POST)

        // Call the RegisterSensor method inside ISensorService interface

        // This is a authorize method. Only "ADMIN" users can acess the this method.

        // Return sucessfully registered or not

        [Authorize(Roles = "ADMIN")]

        [HttpPost("RegisterSensor")]

        public async Task<ApiResult> RegisterSensor(SensorDetails sensorDetails)

        {

            return await _sensorService.RegisterSensor(sensorDetails);

        }


        // Used to edit the sensor

        // Define API URL Path - api/SensorController/EditSensor (PUT)

        // Call the EditSensor method inside ISensorService interface

        // This is a authorize method. Only "ADMIN" users can acess the this method.

        // Return sucessfully edited or not

        [Authorize(Roles = "ADMIN")]

        [HttpPut("EditSensor")]

        public async Task<ApiResult> EditSensor(SensorDetails sensorDetails)

        {

            return await _sensorService.EditSensor(sensorDetails);

        }


        // Used to delete the sensor

        // Define API URL Path - api/SensorController/DeleteSensor/{SensorId} (DELETE)
```

```
        // Call the DeleteSensor method inside ISensorService interface
        // This is a authorize method. Only "ADMIN" users can acess the this method.
        // Return sucessfully deleted or not
        [Authorize(Roles = "ADMIN")]
        [HttpDelete("DeleteSensor/{sensorId}")]
        public async Task<ApiResult> DeleteSensor(int sensorId)
        {
            return await _sensorService.DeleteSensor(sensorId);
        }

        // Used to update the sensor state
        // Define API URL Path - api/SensorController/SetSensorState (POST)
        // Call the SetSensorState method inside ISensorService interface
        // Return the sensor state sucessfully updated or not
        [HttpPost("SetSensorState")]
        public async Task<ApiResult> SetSensorState(SensorDetails sensorState)
        {
            return await _sensorService.SetSensorState(sensorState);
        }
    }
}
```

File Name : UserController

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   UserController
 * @Description :   UserController class inherite from the ControllerBase class. UserController class basically handle the
 *                  all endpoints related to the Users.
*/

namespace FireAlarm.Web.API.Controllers
{
```

```
// Set the API URL Path - api/UserController
[Route("api/[controller]")]
[ApiController]
public class UserController : ControllerBase
{
    // Create object from FireAlarmDbContext class
    private readonly FireAlarmDbContext _context;
    // Create object from IUserService - IUserService handle the all CRUD operations related to the users
    private readonly IUserService _userService;
    // Configuration object
    private IConfiguration _configuration;

    // Constructor
    public UserController(FireAlarmDbContext context, IUserService userService, IConfiguration configuration)
    {
        _context = context;
        _userService = userService;
        _configuration = configuration;
    }

    // Used to signin to the system
    // Define API URL Path - api/UserController/SignIn (POST)
    // Call the SignIn method inside IUserService interface
    // Return the User object if sucessfully registered
    [HttpPost("SignIn")]
    public async Task<ApiResult> SignIn(User user)
    {
        User loginUser = await _userService.SignIn(user);
        Authentication authentication = new Authentication(_configuration);
        return authentication.GetToken(loginUser);
    }

    // Used to get email list and mobile list of all users
    // Define API URL Path - api/UserController/GetUserEmailMobile (GET)
```

```
        // Call the GetUserEmailMobile method inside IUserService interface
        // Return the Users email and mobile list
        [HttpGet("GetUserEmailMobile")]
        public async Task<ApiResult> GetUserEmailMobile()
        {
            return await _userService.GetUserEmailAndMobileList();
        }
    }
}
```

File Name : ApiResult

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   ApiResult
 * @Description :   ApiResult class handle the api return object.
 *                  Every response from API send the ApiResult object.
 *                  ApiResult has two property.
 *                      * status - this is a boolean values and if it is true that mean response come without any error.
 *                               but if it is false that means error happe in the api.
 *                      * data - data property contains the details about response. if status true data property contains
 *                             the result and if status false data property contains the error message.
*/


namespace FireAlarm.Web.Data.Entities
{
    public class ApiResult
    {
        [JsonProperty("STATUS")]
        public bool STATUS { get; set; }
        [JsonProperty("DATA")]
        public object DATA { get; set; }
    }
}
```

File Name : FireAlarmDbContext

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   FireAlarmDbContext
 * @Description :   FireAlarmDbContext inherite from the DbContext in Entity framework
*/

namespace FireAlarm.Web.Data.Entities
{
    public class FireAlarmDbContext : DbContext
    {
        public FireAlarmDbContext(DbContextOptions<FireAlarmDbContext> options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // Set the relationship of User table and User role table.
            modelBuilder.Entity<User>()
                .HasOne<UserRole>(u => u.userRole)
                .WithMany(ur => ur.Users)
                .HasForeignKey(u => u.userRoleId);

            // Add dummy user roles to the db
            modelBuilder.Entity<UserRole>().HasData(
                new UserRole { roleId = 1, roleDescription = "SDUSER" },
                new UserRole { roleId = 2, roleDescription = "ADMIN" });

            // Add dummy users to the db
            modelBuilder.Entity<User>().HasData(
                new User { userId = 1, userName = "Kusal Priyanka", userPassword = "abc123", userEmail = "kusalpriyanka782@gmail.com", userMobileNo = "+940755628231", userRoleId = 2 },
                new User { userId = 2, userName = "Dimuthu Abesighe", userPassword = "abc123", userEmail = "dimuthuc2@gmail.com", userMobileNo = "+940766944088", userRoleId = 1 });


            // Add dummy sensors roles to the db
            modelBuilder.Entity<SensorDetails>().HasData(
                new SensorDetails { sensorId = 1, sensorName = "Sensor One", floorNo = "2", roomNo = "1", sensorStatus = "A"},
                new SensorDetails { sensorId = 2, sensorName = "Sensor Two", floorNo = "1", roomNo = "8", sensorStatus = "A" },
                new SensorDetails { sensorId = 3, sensorName = "Sensor Three", floorNo = "3", roomNo = "3", sensorStatus = "A" },
                new SensorDetails { sensorId = 4, sensorName = "Sensor Four", floorNo = "6", roomNo = "1", sensorStatus = "A" },
                new SensorDetails { sensorId = 5, sensorName = "Sensor Five", floorNo = "5", roomNo = "2", sensorStatus = "A" });


        }


        // Set DbSet by passing the model class
        public DbSet<User> Users { get; set; }
        public DbSet<UserRole> UserRoles { get; set; }
        public DbSet<SensorDetails> SensorDetails { get; set; }
    }
}
```

File Name : SensorDetails

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   SensorDetails
 * @Description :   SensorDetails class store the all properties of the sensor
*/

namespace FireAlarm.Web.Data.Entities
{
    public class SensorDetails
    {
        [Key]
        [JsonProperty("sensorId")]
        public int sensorId { get; set; }
        [JsonProperty("sensorName")]
        public string sensorName { get; set; }
        [JsonProperty("floorNo")]
        public string floorNo { get; set; }
        [JsonProperty("roomNo")]
        public string roomNo { get; set; }
        [JsonProperty("sensorStatus")]
        public string sensorStatus { get; set; }
        [JsonProperty("sensorRemark")]
        public string sensorRemark { get; set; }
        [JsonProperty("smokeLevel")]
        public int smokeLevel { get; set; }
        [JsonProperty("coLevel")]
        public int coLevel { get; set; }
    }
}
```

File Name : User

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   User
 * @Description :   User class store the all properties of the Users
*/

namespace FireAlarm.Web.Data.Entities
{
    public class User
    {
        [Key]
        [JsonProperty("userId")]
        public int userId { get; set; }
        [JsonProperty("userName")]
        public string userName { get; set; }
        [JsonProperty("userPassword")]
        public string userPassword { get; set; }
        [JsonProperty("userEmail")]
        public string userEmail { get; set; }
        [JsonProperty("userMobileNo")]
        public string userMobileNo { get; set; }
        public int userRoleId { get; set; }
        public UserRole userRole { get; set; }
    }
}
```

File Name : UserRole

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   UserRole
 * @Description :   UserRole class store the all properties of the User Roles
*/

namespace FireAlarm.Web.Data.Entities
{
    public class UserRole
    {
        [Key]
        [JsonProperty("roleId")]
        public int roleId { get; set; }
        [JsonProperty("roleDescription")]
        public string roleDescription { get; set; }
        public ICollection<User> Users { get; set; }
    }
}
```

File Name : ISensorService

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   ISensorService
 * @Description :   ISensorService interface contains the all methods
 *                  related to the sensors
*/



namespace FireAlarm.Web.Data.Persistence
{
    public interface ISensorService
    {
        // To Get All Sensors
        Task<ApiResult> GetSensorDetails();
        // To Register New Sensor
        Task<ApiResult> RegisterSensor(SensorDetails sensorDetails);
        // To Edit Exising Sensor
        Task<ApiResult> EditSensor(SensorDetails sensorDetails);
        // To Delete Sensor
        Task<ApiResult> DeleteSensor(int sensorId);
        // To Update The Sensor State
        Task<ApiResult> SetSensorState(SensorDetails sensorState);
    }
}
```

File Name : IUserService

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   IUserService
 * @Description :   IUserService interface contains the all methods
 *                  related to the users
*/

namespace FireAlarm.Web.Data.Persistence
{
   public interface IUserService
   {
      // To Signin To The Users
      Task<User> SignIn(User user);
      // To Get All Mobiles No And Emails List
      Task<ApiResult> GetUserEmailAndMobileList();
   }
}
```

File Name : SensorServiceImpl

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   SensorServiceImpl
 * @Description :   SensorServiceImpl class implement the all methods in
 *                  ISensorService interface
*/

namespace FireAlarm.Web.Data.Persistence
{
   public class SensorServiceImpl : ISensorService
   {
```

```
// Create FireAlarmDbContext object
private readonly FireAlarmDbContext _context;

// Constructor
public SensorServiceImpl() { }
public SensorServiceImpl(FireAlarmDbContext context)
{
   _context = context;
}

// Delete sensor
public async Task<ApiResult> DeleteSensor(int sensorId)
{
   // First get a sensor from database by passing sensor id
   SensorDetails sensorDetails = await _context.SensorDetails.FindAsync(sensorId);
   // If sensorDetails is null pass error
   if (sensorDetails == null)
      return new ApiResult { STATUS = false, DATA = "There is no sensor related to the sensor id" };

   // Otherwise delete the sensor object from database and save changes in database.
   _context.SensorDetails.Remove(sensorDetails);
   await _context.SaveChangesAsync();

   // return the result object
   return new ApiResult { STATUS = true, DATA = "Sucessfully deleted - sensor id : " + sensorId };
}

// Edit sensor
public async Task<ApiResult> EditSensor(SensorDetails sensorDetails)
{
   // First get a sensor from database by passing sensor id
   SensorDetails dbSensorObj = await _context.SensorDetails.FindAsync(sensorDetails.sensorId);
   // If sensorDetails is null pass error
   if (dbSensorObj == null || sensorDetails == null)
```

```
        return new ApiResult { STATUS = false, DATA = "There is no sensor related to the sensor id" };

    // Update the properties of sensor object
    dbSensorObj.sensorName = sensorDetails.sensorName;
    dbSensorObj.floorNo = sensorDetails.floorNo;
    dbSensorObj.roomNo = sensorDetails.roomNo;
    dbSensorObj.sensorStatus = sensorDetails.sensorStatus;
    dbSensorObj.sensorRemark = sensorDetails.sensorRemark;

    // Save the changes in database
    await _context.SaveChangesAsync();
    // return the result object
    return new ApiResult { STATUS = true, DATA = "Sucessfully edited - sensor id : " + sensorDetails.sensorId };
}

// Get sensors
public async Task<ApiResult> GetSensorDetails()
{
    // Create a result object using Sensor Details
    // Get sensor details if sensor status equals to the "A"
    var resultObj = await _context.SensorDetails.Where(sensorDetails => sensorDetails.sensorStatus.Equals("A"))
        .Select(sensorObj => new
        {
            sensorId = sensorObj.sensorId,
            sensorName = sensorObj.sensorName,
            floorNo = sensorObj.floorNo,
            roomNo = sensorObj.roomNo,
            sensorStatus = sensorObj.sensorStatus,
            smokeLevel = sensorObj.smokeLevel,
            co2Level = sensorObj.coLevel
        }
    ).ToListAsync();
    // return the result object
    return new ApiResult { STATUS = true, DATA = resultObj };
```

```
}

// Register sensor
public async Task<ApiResult> RegisterSensor(SensorDetails sensorDetails)
{
   // Check sensorDetails object is null or not
   if (sensorDetails != null)
   {
      // Save the Sensor object
      await _context.AddAsync(sensorDetails);
      try
      {
         // Save the changes in database
         await _context.SaveChangesAsync();
         // return the result object
         return new ApiResult { STATUS = true, DATA = "Sucessfully saved - sensor id : " + sensorDetails.sensorId };
      }
      catch(DbUpdateException dbUpdateEx)
      {
         return new ApiResult { STATUS = false, DATA = "Error saving data - " + dbUpdateEx };
      }
      catch(Exception ex)
      {
         return new ApiResult { STATUS = false, DATA = "Error saving data - " + ex };
      }
   }

   // return the result object
   return new ApiResult { STATUS = false, DATA = "There is no sensor related to the sensor id" };
}

// Set sensor state
public async Task<ApiResult> SetSensorState(SensorDetails sensorState)
{
```

```
            // Check sensorState object is null or not
            if (sensorState != null)
            {
                // Get a sensor from database by passing sensor id
                SensorDetails dbSensorObj = await _context.SensorDetails.FindAsync(sensorState.sensorId);
                // If sensorDetails is null pass error
                if (dbSensorObj == null || sensorState == null)
                    return new ApiResult { STATUS = false, DATA = "There is no sensor related to the sensor id" };

                // Update the sensor smoke and co2 values of sensor object
                dbSensorObj.smokeLevel = sensorState.smokeLevel;
                dbSensorObj.coLevel = sensorState.coLevel;

                // Save the changes in database
                await _context.SaveChangesAsync();
                // return the result object
                return new ApiResult { STATUS = true, DATA = "Sucessfully updated - sensor id : " + sensorState.sensorId };
            }
            return new ApiResult { STATUS = false, DATA = "Please enter sensor details to pass object" };
        }
    }
}
```

File Name : UserServiceImpl

```
/*
 * @Author      :   Kusal Priyanka
 * @Class Name  :   UserServiceImpl
 * @Description :   UserServiceImpl class implement the all methods in
 *                  IUserService interface
*/

namespace FireAlarm.Web.Data.Persistence
```

```
{
    public class UserServiceImpl : IUserService
    {
        // Create FireAlarmDbContext object
        private readonly FireAlarmDbContext _context;

        public UserServiceImpl() { }

        // Constructor
        public UserServiceImpl(FireAlarmDbContext context)
        {
            _context = context;
        }

        // Get mobile no and email list
        public async Task<ApiResult> GetUserEmailAndMobileList()
        {
            // Get email list
            var emailList = await _context.Users.Select(user => user.userEmail).ToListAsync();
            // Get mobile no list
            var mobileList = await _context.Users.Select(user => user.userMobileNo).ToListAsync();

            // Create result object
            var resultObj = new
            {
                email = emailList,
                mobile = mobileList
            };

            // return the result object
            return new ApiResult { STATUS = true, DATA = resultObj };
        }

        public async Task<User> SignIn(User user)
        {
            // Check any user exist given username and password
            User loginUser = await _context.Users.FirstOrDefaultAsync(
                dbUser => dbUser.userEmail.Equals(user.userEmail) && dbUser.userPassword.Equals(user.userPassword)
            );

            // Check user is null or not
            if (loginUser == null)
                return null;

            // Get user role
            loginUser.userRole = await _context.UserRoles.FirstOrDefaultAsync(userRole => userRole.roleId == loginUser.userRoleId);

            // return user
            return loginUser;
        }
    }
}
```

-----------------WEB CLIENT--------------

File Name : WebClient.jsx

```
export default class extends Component{

    state = {
        email : '',
        password: '',
        showHome: false,
        loading: true,
        error: "",
        data: null,
        token: null
```

```
};

//handle change event for email
onChangeHandlerEmail = (e) => {
   e.preventDefault();
   //const {name, value} = e.target;
   this.setState({
      email:e.target.value
   })
};
//handle change event for password
onChangeHandlerPwd = (e) => {
   e.preventDefault();
   //const {name, value} = e.target;
   this.setState({
      password:e.target.value
   })
};
//handle click submit button
onSubmitHandler = (e) =>{
   e.preventDefault();

   // Send a POST request to API
   axios.post("http://localhost:44332/api/user/SignIn", {
         userEmail: this.state.email.toString(),
         userPassword: this.state.password.toString()
      })
      .then(result => {
         //console.log(result);
         this.setState({
            data: result.data,
            loading: false,
            error: false,
            token: result.data.data.token
```

```
        });
        if(this.state.loading === true){
            return(
                <div>
                    Loading...
                </div>
            )
        }
        if (this.state.data.status){    //check whether response status is true, then login success
            this.setState({
                showHome:true
            })
        }
        else{
            Swal.fire({     //when email or password was incorrect
                icon: 'error',
                title: 'Oops...',
                text: 'There is no user related to the user credentials',
            })
        };
    })
    .catch(err => console.error(err));
};

render() {
    return (
        <div>
            {this.state.showHome ?
                <Home
                    token={this.state.token}
                /> :
                <Login
                    email={this.state.email}
                    password={this.state.password}
                    onChangeHandlerEmail={this.onChangeHandlerEmail}
                    onChangeHandlerPwd={this.onChangeHandlerPwd}
                    onSubmitHandler={this.onSubmitHandler}
                  />
               }
            </div>
         );
      }
   }
```

File Name : Home.jsx

```
export default class Home extends Component{
   constructor(props) {
      super(props);
      this.state = {
         content: []     //use for get data from API
      };
      this.fetchData = this.fetchData.bind(this);  //bind function
   }

    componentWillMount() {
       this.fetchData();
       setInterval(this.fetchData, 40000); //Refresh every 40 seconds.
    }

    //This function is use to get sensor details from API
    async fetchData(){
      const {token} = this.props;

      const config = {
         headers: { Authorization: `Bearer ${token}` }
      };
```

```
        if (typeof fetch == 'undefined') return
        const response = await fetch('http://localhost:44332/api/sensor/GetSensorDetails', config)  //send token in the url header
and get the response from API
        const content = await response.json()   //response convert in to json object


        this.setState({
            content: content.data
        });
        //console.log(this.state.content);
    }
    render() {
        const { content } = this.state;
        console.log(content);
        return (
            <div id="Body">
                <div className="header jumbotron">
                    <h1 >Fire Alarm Application</h1>
                    <h5>It is not death, it is dying that alarms me.</h5>
                </div>
                <div className="row m-5" id="Sensor">
                    {content.map((item, index) => {
                        return <Sensor key = {index}
                                       sensorName = {item.sensorName}
                                       co2Level = {item.co2Level}
                                       smokeLevel = {item.smokeLevel}
                                       floorNo = {item.floorNo}
                                       roomNo = {item.roomNo}
                                       sensorStatus = {item.sensorStatus}
                        />;
                    })}
                </div>
            </div>
        );
```

```
   }
}
```

File Name : Sensor.jsx

```
export default class Sensor extends Component{

   render() {
      const {sensorName, co2Level, smokeLevel, floorNo, roomNo, sensorStatus} = this.props;
      let color = '';
      ( co2Level > 5 || smokeLevel > 5) ? color = "danger": color = "success";   //Check whether co2level or smokeLevel grater 5

      return (
         <div className="col-md-4">
            <div className="m-5" >
               <div className={ `card text-white bg-${color} mb-3`}>
                  <div className={`card-header border-${color}`} id="cardHeader"><h4 className="card-title">{sensorName}</h4></div>
                  <div className="card-body " id="cardBody">
                     <p className="card-text">Floor No : {floorNo}</p>
                     <p className="card-text">Room No : {roomNo}</p>
                     <p className="card-text">Smoke Level : {smokeLevel}</p>
                     <p className="card-text">CO2 level : {co2Level}</p>
                  </div>
                  <div className={`card-footer border-${color}`} id="footer">
                     <span>{(sensorStatus === "A") ? "Status : Active":"Status : Inactive"}</span>
                  </div>
               </div>
            </div>
         </div>
      );
   }
}
```

File Name : Login.jsx

```
export default class Login extends Component{

  render() {
    const {email, password, onChangeHandlerEmail, onChangeHandlerPwd, onSubmitHandler} = this.props;

    return(
      <div className="wrapper fadeInDown" style={{marginTop: "8%"}}>
        <div id="formContent">

          <h2> Sign In </h2><br/><br/>
          <p className="login-text">
              <span className="fa-stack fa-lg">
                <i className="fa fa-circle fa-stack-2x"></i>
                <i className="fa fa-lock fa-stack-1x"></i>
              </span>
          </p><br/><br/>
          <form onSubmit={onSubmitHandler}>
            <input name="email" onChange={onChangeHandlerEmail} type="email" className="login-username"
required={true}
                placeholder="Email" value={email}/>
            <input name="password" onChange={onChangeHandlerPwd} type="password" className="login-password"
required={true}
                placeholder="Password" value={password}/>

            <input type="submit" name="Login" value="Login" className="login-submit" style={{marginTop: "10%"}}/>
          </form>

        </div>
      </div>
    );
  }
}
```

-----------------Sensor--------------

File Name : ApiConstant

```
/*
 *          @Author             :           Anjani Welagedara
 *          @Class Name                     :           ApiConstant
 *          @Description        :           This class store the constant values of the program.
 *
*/

public class ApiConstant {

        // FireAlaram API server URL
        public static String API_SERVER = "http://localhost:44332/api/sensor/";

        // API endpoint for get available sensor details
        public static String GET_SENSOR_DETAILS_PATH = "GetSensorDetails";
        // API endpoint for set sensor state
        public static String SET_SENSOR_STATE_PATH = "SetSensorState";
        // Time interval for update sensor state
        public static int TIME_PERIOD = 10000;
        // Time delay
        public static int TIME_DELAY = 0;
}
```

File Name : Sensor

```
/*
 *          @Author             :           Anjani Welagedara
 *          @Class Name                 :           Sensor
 *          @Description                :           This class represent the sensor object.
 *
*/

public class Sensor {

            // Attributes of the sensor object
    private int sensorId;
    private String sensorName;
    private String floorNo;
    private String roomNo;
    private String sensorStatus;
    private int smokeLevel;
    private int co2Level;

    // Constructors
    public Sensor() { }

            public Sensor(int sensorId, String sensorName, String floorNo, String roomNo, String sensorStatus, int smokeLevel,
                                    int co2Level) {
                        super();
                        this.sensorId = sensorId;
                        this.sensorName = sensorName;
                        this.floorNo = floorNo;
                        this.roomNo = roomNo;
                        this.sensorStatus = sensorStatus;
                        this.smokeLevel = smokeLevel;
                        this.co2Level = co2Level;
            }

            // Getters and Setters
            public int getSensorId() {
                        return sensorId;
            }

            public void setSensorId(int sensorId) {
                        this.sensorId = sensorId;
            }

            public String getSensorName() {
                        return sensorName;
            }

            public void setSensorName(String sensorName) {
                        this.sensorName = sensorName;
            }

            public String getFloorNo() {
                        return floorNo;
            }

            public void setFloorNo(String floorNo) {
                        this.floorNo = floorNo;
            }

            public String getRoomNo() {
                        return roomNo;
            }

            public void setRoomNo(String roomNo) {
                        this.roomNo = roomNo;
            }

            public String getSensorStatus() {
                        return sensorStatus;
            }
```

```
	public void setSensorStatus(String sensorStatus) {
		this.sensorStatus = sensorStatus;
	}

	public int getSmokeLevel() {
		return smokeLevel;
	}

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

	public int getCo2Level() {
		return co2Level;
	}

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

}
```

File Name : SensorState

```
/*
 *	@Author		:	Anjani Welagedara
 *	@Class Name			:	Sensor
 *	@Description	:	This class represent the sensor state object.
 *
*/

public class SensorState {

	// Attributes of the sensor object
}

	private int sensorId;
private int smokeLevel;
private int coLevel;

// Constructors
public SensorState() { }

	public SensorState(int sensorId, int smokeLevel, int coLevel) {
		super();
		this.sensorId = sensorId;
		this.smokeLevel = smokeLevel;
		this.coLevel = coLevel;
	}

	// Getters and Setters
	public int getSensorId() {
		return sensorId;
	}

	public void setSensorId(int sensorId) {
		this.sensorId = sensorId;
	}

	public int getSmokeLevel() {
		return smokeLevel;
	}

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

	public int getCoLevel() {
		return coLevel;
	}
	public void setCoLevel(int coLevel) {

		this.coLevel = coLevel;

	}


}
```

File Name : SensorServicesImpl

```
/*
 *	@Author			:		Anjani Welagedara
 *	@Interface Name		:		ISensorServices
 *	@Description		:		This interface represent the functions of sensor object.
 *
*/

public interface ISensorServices {


	// Get all sensors objects
	public List<Sensor> GetSensorDetails();


	// Set sensor state
	public boolean SetSensorState(SensorState sensorState);


}
```

File Name : ISensorServices

```
/*
 *	@Author		:		Anjani Welagedara
 *	@Class Name			:		SensorServicesImpl
 *	@Description	:		Implement the methods of ISensorServices
 *
*/
```

```
public class SensorServicesImpl implements ISensorServices{

	@Override
	public List<Sensor> GetSensorDetails() {

		// Create List object to store the sensors
		List<Sensor> sensorList = new ArrayList<Sensor>();

		// Create client object to send request to API
		Client client = ClientBuilder.newClient();

		// Call the API and get response
		Response response = client
				.target(ApiConstant.API_SERVER)			// Server URL
				.path(ApiConstant.GET_SENSOR_DETAILS_PATH) // Endpoint path
				.request(MediaType.APPLICATION_JSON)		// Request Media Type
				.get();
	// Request method type

		// Check response status equals to 200
		// If it is 200 that mean status is OK. If not that mean some error happen.
		if(response.getStatus() == 200) {
			// Get a json object from response
			JsonObject jsonResponse = response.readEntity(JsonObject.class);
			// Then check the json object status
			// In the API every response have a two attribute. One is "status" and another one is "data".
			// "status" is boolean value. If it's true that mean API return valid response and "data" contains the
Sensor objects
			// If "status" is false data attribute contains the error message.
			if(jsonResponse.getBoolean("status")) {
				// Using Gson library get sensor object array from json response variable
				Sensor list[] = new Gson().fromJson(jsonResponse.get("data").toString(), Sensor[].class);
				// Array store in the list
				sensorList = Arrays.asList(list);
```

```
                    return sensorList;

                }

                // If "status" is false return null value

                return null;

        }

        // If status not equals to 200 return null value

        return null;

    }


    @Override

    public boolean SetSensorState(SensorState sensorState) {


        // Create List object to store the sensors

        Client client = ClientBuilder.newClient();


        // Call the API and get response

        Response response = client

                    .target(ApiConstant.API_SERVER)
            // Server URL

                    .path(ApiConstant.SET_SENSOR_STATE_PATH)
    // Endpoint path

                    .request(MediaType.APPLICATION_JSON)
        // Request Media Type

                    .post(Entity.entity(sensorState, MediaType.APPLICATION_JSON));     // Request method
type and pass object




        // Check response status equals to 200

        // If it is 200 that mean status is OK. If not that mean some error happen.

        if(response.getStatus() == 200) {

            // Get a json object from response

            JsonObject jsonResponse = response.readEntity(JsonObject.class);

            // Then check the json object status

            // In the API every response have a two attribute. One is "status" and another one is "data".

            // "status" is boolean value. If it's true that mean API return valid response and "data" contains the
updated sensor Id
```

```
                        // If "status" is false data attribute contains the error message.
                        if(jsonResponse.getBoolean("status")) {
                                // If "status" true that mean sensor state updates. Then return true.
                                return true;
                        }
                        // If "status" is false return false value
                        return false;
                }
                // If status not equals to 200 return false value
                return false;
        }

}
```

File Name : Main

```
/*
 *      @Author         :       Anjani Welagedara
 *      @Class Name             :       Main
 *      @Description    :       Handle sensor object and return view for sensor object
 *
*/

public class Main {

        public static void main(String[] args) {

                // Create instance of sensor service object
                ISensorServices iSensorServices = new SensorServicesImpl();
                // Get sensor objects list from API.
                List<Sensor> sensorList = iSensorServices.GetSensorDetails();

                // Check sensorList not null
                if(sensorList != null) {
```

```
				for(Sensor sensor : sensorList) {
					// Return view for every sensor object
					new SensorUI(sensor);
				}
			}
		}

}
```

File Name : SensorUI

```
/*
 *	@Author		:	Anjani Welagedara
 *	@Class Name		:	SensorUI
 *	@Description	:	Handle user interface for sensorobject
 *
*/

public class SensorUI {

	// Attributes
	private Sensor sensor;
	private ISensorServices iSensorServices;

	// Constructor
	public SensorUI(Sensor sensor) {
		this.sensor = sensor;
		this.iSensorServices = new SensorServicesImpl();
		InitiUI();
	}

	// return UI
	private void InitiUI() {
```

```
// Initialize Frame
JFrame frame = new JFrame(sensor.getSensorName());

// Display sensor id
JLabel lblSensorId = new JLabel("Sensor Id : ");
lblSensorId.setBounds(12, 15, 90, 16);

// Display sensor name
JLabel lblSensorName = new JLabel("Sensor Name : ");
lblSensorName.setBounds(12, 45, 90, 16);

// Display floor no
JLabel lblFloorNo = new JLabel("Floor No : ");
lblFloorNo.setBounds(12, 75, 90, 16);

// Display room no
JLabel lblRoomNo = new JLabel("Room No : ");
lblRoomNo.setBounds(12, 105, 90, 16);

// Display value of sensor id
JLabel lblSensorIdVal = new JLabel(sensor.getSensorId() + "");
lblSensorIdVal.setBounds(168, 15, 56, 16);

// Display value of sensor name
JLabel lblSensorNameVal = new JLabel(sensor.getSensorName());
lblSensorNameVal.setBounds(168, 45, 90, 16);

// Display value of floor no
JLabel lblFloorNoVal = new JLabel(sensor.getFloorNo());
lblFloorNoVal.setBounds(168, 75, 56, 16);

// Display value of room no
JLabel lblRoomNoVal = new JLabel(sensor.getRoomNo());
```

```
lblRoomNoVal.setBounds(168, 105, 56, 16);

// Display smoke level
JLabel lblSmokeLevel = new JLabel("Smoke Level : ");
lblSmokeLevel.setBounds(48, 152, 90, 16);

// Display co2 level
JLabel lblCo2 = new JLabel("Co2 Level : ");
lblCo2.setBounds(200, 152, 90, 16);

// Display value of smoke level
JLabel lblSmokeLevelVal = new JLabel(sensor.getSmokeLevel() + "");
lblSmokeLevelVal.setFont(new Font("Tahoma", Font.BOLD, 20));
lblSmokeLevelVal.setBounds(74, 176, 40, 16);

// Display value of co2 level
JLabel lblCo2Val = new JLabel(sensor.getCo2Level() + "");
lblCo2Val.setFont(new Font("Tahoma", Font.BOLD, 20));
lblCo2Val.setBounds(213, 176, 40, 16);

// Add UI components to frame
frame.getContentPane().add(lblSensorId);
frame.getContentPane().add(lblSensorName);
frame.getContentPane().add(lblFloorNo);
frame.getContentPane().add(lblRoomNo);

frame.getContentPane().add(lblSensorIdVal);
frame.getContentPane().add(lblSensorNameVal);
frame.getContentPane().add(lblRoomNoVal);
frame.getContentPane().add(lblFloorNoVal);

frame.getContentPane().add(lblSmokeLevel);
frame.getContentPane().add(lblCo2);
```

```
}

			frame.getContentPane().add(lblSmokeLevelVal);
			frame.getContentPane().add(lblCo2Val);

			// Set frame size
			frame.setSize(350,260);
			// Set frame layout
			frame.getContentPane().setLayout(null);
			frame.setLocationRelativeTo(null);
			// Set frame visible
			frame.setVisible(true);

			// Initialize the timer object
			Timer timer = new Timer();
			// Define scheduled method using timer object
			timer.schedule(new TimerTask() {

				@Override
				public void run() {
					// Get random value set to the view
					lblSmokeLevelVal.setText(getRandomIntValue() + "");
					lblCo2Val.setText(getRandomIntValue() + "");

					// Call the changeSensorState
					changeSensorState(new SensorState(
							sensor.getSensorId(),
			// Pass sensor Id
							Integer.parseInt(lblSmokeLevelVal.getText()),	// Pass smoke level
							Integer.parseInt(lblCo2Val.getText())			//	Pass
co2 level
							));
				}
			}, ApiConstant.TIME_DELAY, ApiConstant.TIME_PERIOD);			//	Define	delay	and	time
interval

		}
```

```
        // Handle the sensor state change
        // Call the method of iSensorServices to send updated data to API
        private void changeSensorState(SensorState sensorState) {
                iSensorServices.SetSensorState(sensorState);
        }

        // Return random int value with in 0 - 10
        private int getRandomIntValue() {
                return ThreadLocalRandom.current().nextInt(0, 10);
        }
}
```

-----------------RMI CLIENT/SERVER--------------

------RMI Client

File Name : Main

```
/*
 *      @Author         :       Ashen Senevirathne
 *      @Class Name             :       Main
 *      @Description            :       Main class is used to get services from RMI server
 *
*/

public class Main {

        public static void main(String[] args) {

                // Set the policy file as the system security policy
                System.setProperty("java.security.policy", "file:allowall.policy");
```

```
			// Define remote interfaces
			ISensorServices sensorService = null;
			IUserServices userService = null;

			try {
				// Initialize the remote interface from RMI server
				sensorService = (ISensorServices) Naming.lookup("//localhost/SensorService");
				userService = (IUserServices) Naming.lookup("//localhost/UserService");

				// If remote interfaces not null return the login UI
				if(sensorService != null && userService != null) {
					new LoginUI(sensorService,userService);
				}

			} catch (Exception ex) {
				System.out.println("Error : " + ex.getMessage());
			}
		}

}
```

File Name : Dashboard

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name		:	Dashboard
 *	@Description	:
 *
*/

public class Dashboard {

	private ISensorServices sensorService;
	private User loginUser;
	private List<Sensor> sensorList;
	DefaultTableModel model = new DefaultTableModel();
	DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");

	public Dashboard(ISensorServices sensorService, User loginUser) {
		this.sensorService = sensorService;
		this.loginUser = loginUser;
		dashboard();
	}

	private List<Sensor> getAllSensors(){

		try {
			Sensor list[] = new Gson().fromJson(sensorService.GetAllSensors().toString(), Sensor[].class);
			showNotification(list);
			return Arrays.asList(list);
		} catch (JsonSyntaxException e) {
			e.printStackTrace();
		} catch (RemoteException e) {
			e.printStackTrace();
		}

		return null;
	}

	private void showNotification(Sensor list[]) {
		String head = "Fire Alarm alert !\nFollowing location are detected. \n";
		String fireLocation = "";
		for(Sensor sensor : list) {
			if(sensor.getCo2Level() > 5 && sensor.getSmokeLevel() > 5) {
				fireLocation +=  "Sensor " + sensor.getSensorId() + " = Room No : " + sensor.getRoomNo()
+ " Floor No : " + sensor.getFloorNo() + "\n";
			}
		}
```

```
                    if(fireLocation != "") {
                              JOptionPane.showConfirmDialog(null,
                   head + fireLocation, "Alert Warning !", JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
                    }
          }

          private Sensor getSensorById(int sensorId) {
                    for(Sensor sensor : sensorList) {
                              if(sensor.getSensorId() == sensorId)
                                        return sensor;
                    }
                    return null;
          }

          private void fillTable() {
                    sensorList = null;
                    this.sensorList = getAllSensors();
                    model.setRowCount(0);
                    for(Sensor sensor : sensorList) {
                              String sensorState = "N";
                              if(sensor.getSmokeLevel() > 5 && sensor.getCo2Level() > 5)
                                        sensorState = "Y";
                              model.addRow(new Object[] { sensor.getSensorId(), sensor.getSensorName(), sensor.getFloorNo(),
sensor.getRoomNo(),
                                                  sensor.getSmokeLevel(), sensor.getCo2Level(), sensorState});


                    }
          }

          private void dashboard() {

                    JFrame frame = new JFrame("Fire alarm Dashboard");
                    JTable table = new JTable() {
                              @Override
```

```
			public Component prepareRenderer(TableCellRenderer renderer,int row,int column) {
				Component comp = super.prepareRenderer(renderer,row, column);
				comp.setBackground(getBackground());
				int modelRow = convertRowIndexToModel(row);
				String type = (String)getModel().getValueAt(modelRow, 6);
				if ("Y".equals(type))
					comp.setBackground(Color.RED);
				return comp;
			}
		};

		frame.setBounds(430, 200, 533, 530);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);

		Object[] colomns = { "Sensor Id", "Sensor Name", "Floor No", "Room No", "Smoke Level", "Co2 Level" ,
"Status"};

		model.setColumnIdentifiers(colomns);
		table.setModel(model);

		table.getColumnModel().getColumn(6).setMinWidth(0);
		table.getColumnModel().getColumn(6).setMaxWidth(0);

		JScrollPane scrollPane = new JScrollPane(table);
		scrollPane.setBounds(12, 98, 491, 348);
		frame.getContentPane().add(scrollPane);

		JLabel caption = new JLabel("Fire Alaram Monitoring System");
		caption.setFont(new Font("Tahoma", Font.BOLD, 20));
		caption.setBounds(94, 16, 319, 25);
		frame.getContentPane().add(caption);
```

```
JLabel lblLastSync = new JLabel("Last Sync : ");
lblLastSync.setFont(new Font("Tahoma", Font.BOLD, 12));
lblLastSync.setBounds(15, 456, 215, 16);
frame.getContentPane().add(lblLastSync);

JLabel lblLoginUser = new JLabel("Login User : " + loginUser.getUserName());
lblLoginUser.setFont(new Font("Tahoma", Font.BOLD, 12));
lblLoginUser.setBounds(309, 456, 194, 16);
frame.getContentPane().add(lblLoginUser);

frame.setLocationRelativeTo(null);
frame.setVisible(true);

if(loginUser.getUserRole().equals("ADMIN")) {

        JButton btnAddNewSensor = new JButton("Add New Sensor");
        btnAddNewSensor.setBounds(94, 57, 146, 25);
        frame.getContentPane().add(btnAddNewSensor);

        JButton btnEditDelete = new JButton("Edit/Delete Sensor");
        btnEditDelete.setBounds(265, 56, 146, 25);
        frame.getContentPane().add(btnEditDelete);

        btnAddNewSensor.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                        frame.setVisible(false);
                        registerSensor(null);
                }
        });

        btnEditDelete.addActionListener(new ActionListener() {
```

```
					@Override
					public void actionPerformed(ActionEvent e) {
						frame.setVisible(false);
						editDeleteOption();
					}
				});

		}

		Timer timer = new Timer();

		timer.schedule(new TimerTask() {

				@Override
				public void run() {
					LocalDateTime now = LocalDateTime.now();
					lblLastSync.setText("Last Sync : " + dateFormat.format(now));
					fillTable();
				}
		}, ClientConstants.TIME_DELAY, ClientConstants.TIME_PERIOD);

	}

	private void editDeleteOption() {

		JFrame frame = new JFrame("Edit/Delete Sensor");

		JLabel lblSensorId = new JLabel("Sensor ID");
		lblSensorId.setBounds(12, 15, 106, 16);

		JComboBox<String> sensorLst = new JComboBox<String>();
		sensorLst.setBounds(12, 44, 358, 30);

		for(Sensor sensor : sensorList) {
				sensorLst.addItem(sensor.getSensorId() + " - " + sensor.getSensorName());
			}

			JButton edit  = new JButton("Edit");
			edit.setBounds(22, 99, 150, 30);
			JButton delete  = new JButton("Delete");
			delete.setBounds(192, 99, 150, 30);

			frame.setSize(400,187);
			frame.getContentPane().add(lblSensorId);
			frame.getContentPane().add(sensorLst);
			frame.getContentPane().add(edit);
			frame.getContentPane().add(delete);
			frame.getContentPane().setLayout(null);
			frame.setLocationRelativeTo(null);
			frame.setVisible(true);

			edit.addActionListener(new ActionListener() {
				@Override
				public void actionPerformed(ActionEvent e) {
					frame.setVisible(false);
					int  selectedItem  =  Integer.parseInt(sensorLst.getSelectedItem().toString().substring(0,
sensorLst.getSelectedItem().toString().indexOf('-')).trim());
					registerSensor(getSensorById(selectedItem));
				}
			});

			delete.addActionListener(new ActionListener() {
				@Override
				public void actionPerformed(ActionEvent e) {
					JOptionPane.showMessageDialog(frame, "Sucessfully Deleted!");

					int  selectedItem  =  Integer.parseInt(sensorLst.getSelectedItem().toString().substring(0,
sensorLst.getSelectedItem().toString().indexOf('-')).trim());
					try {
```

```
                              sensorService.DeleteSensor(selectedItem, loginUser.getToken());
                              frame.setVisible(false);
                              dashboard();
                         } catch (RemoteException e1) {
                              e1.printStackTrace();
                         }
                    }
               });
          }

          private void registerSensor(Sensor sensor) {

               String windowName = "New Sensor Register";
               String btnName = "Register Sensor";

               if(sensor != null) {
                    windowName = "Edit Sensor";
                    btnName = "Edit Sensor";
               }

               JFrame frame = new JFrame(windowName);

               JLabel lblsensorName = new JLabel("Sensor Name");
               lblsensorName.setBounds(12, 13, 106, 16);

               JTextField txtsensorName = new JTextField(40);
               txtsensorName.setBounds(12, 38, 358, 30);

               JLabel lblfloorNo = new JLabel("Floor No");
               lblfloorNo.setBounds(12, 78, 106, 16);

               JTextField txtfloorNo = new JTextField(40);
               txtfloorNo.setBounds(12, 103, 358, 30);
```

```
JLabel lblroomNo = new JLabel("Room No");

lblroomNo.setBounds(12, 141, 106, 16);


JTextField txtroomNo = new JTextField(40);

txtroomNo.setBounds(12, 165, 358, 30);


JLabel lblSensorStatus = new JLabel("Sensor Status");

lblSensorStatus.setBounds(12, 200, 106, 16);


JComboBox<String> sensorStatus = new JComboBox<String>();

sensorStatus.setBounds(12, 225, 358, 30);


sensorStatus.addItem("A");

sensorStatus.addItem("D");


JButton register  = new JButton(btnName);

register.setBounds(12, 333, 358, 30);


frame.setSize(400,430);

frame.getContentPane().add(lblsensorName);

frame.getContentPane().add(txtsensorName);

frame.getContentPane().add(lblfloorNo);

frame.getContentPane().add(txtfloorNo);

frame.getContentPane().add(lblroomNo);

frame.getContentPane().add(txtroomNo);

frame.getContentPane().add(lblSensorStatus);

frame.getContentPane().add(sensorStatus);;

frame.getContentPane().add(register);


frame.getContentPane().setLayout(null);

frame.setLocationRelativeTo(null);

frame.setVisible(true);


if(sensor != null) {
				txtsensorName.setText(sensor.getSensorName());
				txtfloorNo.setText(sensor.getFloorNo());
				txtroomNo.setText(sensor.getRoomNo());
				sensorStatus.setSelectedItem(sensor.getSensorStatus());
			}

			register.addActionListener(new ActionListener() {

				@Override
				public void actionPerformed(ActionEvent arg0) {
					if(txtsensorName.getText().trim().length() > 0 &&  txtfloorNo.getText().trim().length() > 0 
&&
							txtroomNo.getText().trim().length() > 0 ) {

						if(sensor != null) {
							try {
								sensorService.EditSensor(sensor.getSensorId(), 
txtsensorName.getText().trim(), txtfloorNo.getText().trim(), txtroomNo.getText().trim(),

	sensorStatus.getSelectedItem().toString(), loginUser.getToken());
								frame.setVisible(false);
								dashboard();
							} catch (RemoteException e) {
								e.printStackTrace();
							}
						}
						else {
							try {

	sensorService.RegisterSensor(txtsensorName.getText().trim(), txtfloorNo.getText().trim(), txtroomNo.getText().trim(),

	sensorStatus.getSelectedItem().toString(), loginUser.getToken());
								frame.setVisible(false);
								dashboard();
							} catch (RemoteException e) {
								e.printStackTrace();
```

```
                                                }
                                        }


                                }
                                else {
                                        JOptionPane.showConfirmDialog(null,
                               "Please  Fill  All  Fields  !",  "Error  Message",  JOptionPane.DEFAULT_OPTION,
JOptionPane.ERROR_MESSAGE);
                                }
                        }
                });
        }


}
```

File Name : LoginUI

```
/*
 *      @Author         :       Ashen Senevirathne
 *      @Class Name             :       LoginUI
 *      @Description    :       Handle the Login function
 *
*/

public class LoginUI {


        // Attributes of LoginUI
        private ISensorServices sensorService;
        private IUserServices userService;


        // Constructor
        public LoginUI(ISensorServices sensorService, IUserServices userService) {


                this.sensorService = sensorService;
                this.userService = userService;

                JFrame frame = new JFrame("Fire Alaram Monitoring");

                JLabel lblUserEmail = new JLabel("Enter User Email");
                lblUserEmail.setBounds(12, 15, 358, 16);

                JTextField txtUserEmail = new JTextField(40);
                txtUserEmail.setBounds(12, 44, 358, 30);

                JLabel lblUserPassword = new JLabel("Enter User Password");
                lblUserPassword.setBounds(12, 87, 358, 16);

                JPasswordField txtUserPassword = new JPasswordField(40);
                txtUserPassword.setBounds(12, 116, 358, 30);

                JButton login  = new JButton("Login");
                login.setBounds(114, 159, 150, 30);

                frame.setSize(400,250);
                frame.getContentPane().add(lblUserEmail);
                frame.getContentPane().add(txtUserEmail);
                frame.getContentPane().add(lblUserPassword);
                frame.getContentPane().add(txtUserPassword);
                frame.getContentPane().add(login);
                frame.getContentPane().setLayout(null);
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);

                login.addActionListener(new ActionListener() {

                        @SuppressWarnings("deprecation")
                        @Override
                        public void actionPerformed(ActionEvent e) {
```

```
                                        if(txtUserEmail.getText().length() == 0 || txtUserPassword.getText().length() == 0) {
                                                  JOptionPane.showConfirmDialog(null,
                                       "Please enter valid user email and password!", "Error Message",
JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
                                        }
                                        else {

                                                  try {
                                                            String apiResult = userService.SignIn(txtUserEmail.getText().trim(),
txtUserPassword.getText().trim());
                                                            User loginUser = new Gson().fromJson(apiResult, User.class);
                                                            if(loginUser != null) {
                                                                      frame.setVisible(false);
                                                                      new Dashboard(sensorService, loginUser);
                                                            }
                                                            else {
                                                                      JOptionPane.showConfirmDialog(null,
                                                           "Please enter valid user email and password!", "Error Message",
JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
                                                            }
                                                  } catch (RemoteException e1) {
                                                            e1.printStackTrace();
                                                  }

                                        }
                              }
                    });
          }

}
```

File Name : ClientConstants

```
/*
```

```
 *		@Author			:		Ashen Senevirathne
 *		@Class Name				:		ClientConstants
 *		@Description		:		Used to store the constants value of client programe
 *
*/

public class ClientConstants {

		// Time interval for update sensor state
		public static int TIME_PERIOD = 15000;
		// Time delay
		public static int TIME_DELAY = 0;

}
```

-----------------RMI Server--------------

File Name : Service

```
/*
 *		@Author			:		Ashen Senevirathne
 *		@Class Name				:		Service
 *		@Description				:		Service class used to run the RMI server and bind the remote interfaces
 *
*/

public class Service {

		public static void main(String[] args) {

				// Set the policy file as the system security policy
				System.setProperty("java.security.policy", "file:allowall.policy");
		try {
			// Define remote interfaces
			UserServicesImpl userServices = new UserServicesImpl();
			SensorServicesImpl sensorService = new SensorServicesImpl();

			// Create object of Registry
			Registry registry = LocateRegistry.getRegistry();
			// Bind the interfaces with Registry
			registry.bind("UserService", userServices);
			registry.bind("SensorService", sensorService);

			System.out.println("FireAlaram Service Started!");

		} catch (Exception ex) {
			System.out.println("Error : " + ex.getMessage());
		}

	}

}
```

File Name : AlertConstant

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name			:	AlertConstant
 *	@Description	:	This class store information used in send email and SMS
 *
*/

public class AlertConstant {

	// Email Configuration and Information
```

```
        // Define email host
        public static String EMAIL_HOST = "smtp.gmail.com";
        // Define email port
        public static String EMAIL_PORT = "465";
        // Define ssl verification
        public static String EMAIL_SSL = "true";
        // Define email authentications
        public static String EMAIL_AUTH = "true";
        // Set sender email address
        public static String EMAIL_SENDER = "hotelbluedragon@gmail.com";
        // Set sender email address password
        public static String EMAIL_SENDER_PWD = "mad@2019";

        // Set email subject
        public static String EMAIL_SUBJECT = "Fire Alert - Fire Alaram Monitoring System";
        // Set email message body
        public static String EMAIL_MESSAGE = "Fire Alarm alert !\nFollowing location are detected. \n";

        // SMS Configuration and Information
        // Set twilio account SID
        public static final String SMS_ACCOUNT_SID = "AC7a1e0939de604287ae78feb05fac1f3e";
        // Set twilio account Authentication token
        public static final String SMS_AUTH_TOKEN = "6eb39265c35acce4f189ffeabe0e28c0";
        // Set sender phone number
        public static final String SMS_MY_NUMBER = "+12057360906";
}
```

File Name : ApiConstant

```
/*
 *      @Author         :       Ashen Senevirathne
 *      @Class Name             :       ApiConstant
 *      @Description    :       This class store endpoints path of API
 *
```

```
*/

public class ApiConstant {

        // FireAlaram API server URL
        public static String API_SERVER = "http://localhost:44332/api/";

        // API endpoint for SignIn
        public static String SIGNIN_METHOD = "user/SignIn";
        // API endpoint for Get user emails and mobile numbers
        public static String GET_USER_EMAIL_MOBILE = "user/GetUserEmailMobile";

        // API endpoint for get sensor details
        public static String GET_SENSOR_DETAILS_PATH = "sensor/GetSensorDetails";
        // API endpoint for register new sesor
        public static String REGISTER_NEW_SENSOR = "sensor/RegisterSensor";
        // API endpoint for edit sensor
        public static String EDIT_SENSOR = "sensor/EditSensor";
        // API endpoint for delete sensor
        public static String DELETE_SENSOR = "sensor/DeleteSensor";

}
```

File Name : EditSensor

```
/*
 *      @Author         :       Ashen Senevirathne
 *      @Class Name             :       EditSensor
 *      @Description    :       This class store information about Edit sensor object.
 *                                              This class extends from the RegisterSensor class
 *
*/
```

```
public class EditSensor extends RegisterSensor{

	// Edit sensor object properties
	private int sensorId;

	// Constructors
	public EditSensor() { }

	public EditSensor(int sensorId,String sensorName, String floorNo, String roomNo, String sensorStatus) {
		super(sensorName,floorNo,roomNo,sensorStatus);
		this.sensorId = sensorId;
	}

	// Getters and Setters
	public int getSensorId() {
		return sensorId;
	}

	public void setSensorId(int sensorId) {
		this.sensorId = sensorId;
	}

}
```

File Name : RegisterSensor

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name			:	RegisterSensor
 *	@Description	:	This class store information about Register sensor object.
 *
*/
```

```
public class RegisterSensor {

	// Register sensor object properties
	private String sensorName;
	private String floorNo;
	private String roomNo;
	private String sensorStatus;

	// Constructors
	public RegisterSensor() { }

	public RegisterSensor(String sensorName, String floorNo, String roomNo, String sensorStatus) {
		super();
		this.sensorName = sensorName;
		this.floorNo = floorNo;
		this.roomNo = roomNo;
		this.sensorStatus = sensorStatus;
	}

	// Getters and Setters
	public String getSensorName() {
		return sensorName;
	}

	public void setSensorName(String sensorName) {
		this.sensorName = sensorName;
	}

	public String getFloorNo() {
		return floorNo;
	}

	public void setFloorNo(String floorNo) {
		this.floorNo = floorNo;
	}

	public String getRoomNo() {
		return roomNo;
	}

	public void setRoomNo(String roomNo) {
		this.roomNo = roomNo;
	}

	public String getSensorStatus() {
		return sensorStatus;
	}

	public void setSensorStatus(String sensorStatus) {
		this.sensorStatus = sensorStatus;
	}

}
```

File Name : Sensor

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name		:	Sensor
 *	@Description	:	This class represent the sensor object.
 *
*/

public class Sensor {

	// Attributes of the sensor object
   private int sensorId;
   private String sensorName;
private String floorNo;

private String roomNo;

private String sensorStatus;

private int smokeLevel;

private int co2Level;


// Constructors

public Sensor() { }


        public Sensor(int sensorId, String sensorName, String floorNo, String roomNo, String sensorStatus, int smokeLevel,

                                int co2Level) {

                super();

                this.sensorId = sensorId;

                this.sensorName = sensorName;

                this.floorNo = floorNo;

                this.roomNo = roomNo;

                this.sensorStatus = sensorStatus;

                this.smokeLevel = smokeLevel;

                this.co2Level = co2Level;

        }


        // Getters and Setters

        public int getSensorId() {

                return sensorId;

        }


        public void setSensorId(int sensorId) {

                this.sensorId = sensorId;

        }


        public String getSensorName() {

                return sensorName;

        }
```

```
public void setSensorName(String sensorName) {

        this.sensorName = sensorName;

}


public String getFloorNo() {

        return floorNo;

}


public void setFloorNo(String floorNo) {

        this.floorNo = floorNo;

}


public String getRoomNo() {

        return roomNo;

}


public void setRoomNo(String roomNo) {

        this.roomNo = roomNo;

}


public String getSensorStatus() {

        return sensorStatus;

}


public void setSensorStatus(String sensorStatus) {

        this.sensorStatus = sensorStatus;

}


public int getSmokeLevel() {

        return smokeLevel;

}


public void setSmokeLevel(int smokeLevel) {

        this.smokeLevel = smokeLevel;
```

```
	}

	public int getCo2Level() {
		return co2Level;
	}

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

}
```

File Name : LoginUser

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name		:	LoginUser
 *	@Description		:	This class represent the LoginUser object.
 *
*/

public class LoginUser {

	// Attributes of the LoginUser object
	private String userEmail;
	private String userPassword;

	 // Constructors
	public LoginUser() {}

	public LoginUser(String userEmail, String userPassword) {
		super();
		this.userEmail = userEmail;
		this.userPassword = userPassword;
	}

	// Getters and Setters
	public String getUserEmail() {
		return userEmail;
	}

	public void setUserEmail(String userEmail) {
		this.userEmail = userEmail;
	}

	public String getUserPassword() {
		return userPassword;
	}

	public void setUserPassword(String userPassword) {
		this.userPassword = userPassword;
	}

}
```

File Name : User

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name		:	User
 *	@Description	:	This class represent the User object.
 *
*/

public class User {

	// Attributes of the User object
```

```
private String userId;
private String userName;
private String token;
private String userMobile;
private String userEmail;
private String userRole;

// Constructors
public User() { }

public User(String userId, String userName, String token, String userMobile, String userEmail, String userRole) {
        super();
        this.userId = userId;
        this.userName = userName;
        this.token = token;
        this.userMobile = userMobile;
        this.userEmail = userEmail;
        this.userRole = userRole;
}

// Getters and Setters
public String getUserId() {
        return userId;
}

public void setUserId(String userId) {
        this.userId = userId;
}

public String getUserName() {
        return userName;
}

public void setUserName(String userName) {
	this.userName = userName;
}

public String getToken() {
	return token;
}

public void setToken(String token) {
	this.token = token;
}

public String getUserMobile() {
	return userMobile;
}

public void setUserMobile(String userMobile) {
	this.userMobile = userMobile;
}

public String getUserEmail() {
	return userEmail;
}

public void setUserEmail(String userEmail) {
	this.userEmail = userEmail;
}

public String getUserRole() {
	return userRole;
}

public void setUserRole(String userRole) {
	this.userRole = userRole;
}
```

```
}
```

File Name : EmailService

```
/*
 *	@Author		:	Ashen Senevirathne
 *	@Class Name		:	EmailService
 *	@Description	:	This class used to send the emails for the users
 *
*/

public class EmailService {

	public static void SendEmail(String emailList, String Emailmessage) {

		// Initialize the properties
		Properties properties = System.getProperties();

		// Set properties details
	  properties.put("mail.smtp.host", AlertConstant.EMAIL_HOST);
	  properties.put("mail.smtp.port", AlertConstant.EMAIL_PORT);
	  properties.put("mail.smtp.ssl.enable", AlertConstant.EMAIL_SSL);
	  properties.put("mail.smtp.auth", AlertConstant.EMAIL_AUTH);

	  // Start the session and pass the sender authentication attributes
	  Session session = Session.getDefaultInstance(properties, new Authenticator() {
		protected PasswordAuthentication getPasswordAuthentication() {
				return		new		PasswordAuthentication(AlertConstant.EMAIL_SENDER,
AlertConstant.EMAIL_SENDER_PWD);
		}
		});
```

```
        // Get the emails list
        String[] recipientList = emailList.split(",");
        // Initialize the InternetAddress to store the emails
        InternetAddress[] recipientAddress = new InternetAddress[recipientList.length];

        try {
                        // Add email to the InternetAddress object
                for(int i = 0; i < recipientList.length; i++) {
                        recipientAddress[i] = new InternetAddress(recipientList[i].trim());
                }

                // Initialize MimeMessage object
              MimeMessage message = new MimeMessage(session);
              // Set Sender Email address
              message.setFrom(new InternetAddress(AlertConstant.EMAIL_SENDER));
              // Set recipient list
              message.addRecipients(Message.RecipientType.TO, recipientAddress);
              // Set email subject
              message.setSubject(AlertConstant.EMAIL_SUBJECT);
              // Set email message
              message.setText(AlertConstant.EMAIL_MESSAGE + Emailmessage);
              // Send email
              Transport.send(message);
              System.out.println("Email Send!");

              } catch (Exception ex) {
                        System.out.println(ex);
              }
       }

}
```

File Name : SmsService

```
/*
 *      @Author         :       Ashen Senevirathne
 *      @Class Name             :       SmsService
 *      @Description    :       This class used to send the SMS for the users
 *
*/

public class SmsService {

        public static void SendSms(String MobList, String Smsmessage) {

                // Initialize twilio object and pass SID and Auth Token to verify user
                Twilio.init(AlertConstant.SMS_ACCOUNT_SID, AlertConstant.SMS_AUTH_TOKEN);

                // Store mobile number list in array
                String[] recipientList = MobList.split(",");

                /*
                 * For send the SMS to the user we used the Twilio service.
                 * Using Twilio we can create a free trial account and get free mobile no for the testing purposes.
                 * There is some limitation in the free trial. One of the limitation is using free trial mobile no we can only send
messages for the verified numbers.
                 * So that in this example we already registered the those users numbers in Twilio. Otherwise SMS will not
send for the unverified numbers.
                 *
                 */

                // Send sms for the Users
                for(int i = 0; i < recipientList.length; i++) {
                        Message smsMessage = Message.creator(new PhoneNumber(recipientList[i]),
                        new     PhoneNumber(AlertConstant.SMS_MY_NUMBER),      AlertConstant.EMAIL_MESSAGE     +
Smsmessage).create();
                        System.out.println(smsMessage.getSid());
                }
        }

}
```

File Name : ISensorServices

```
/*
 *      @Author             :       Ashen Senevirathne
 *      @Interface Name             :       ISensorServices
 *      @Description                :       Define the all remote methods related to the sensors
 *
*/

public interface ISensorServices extends Remote{

        // Get all sensors from API
        public String GetAllSensors() throws RemoteException;
        // Register new sensor
        public boolean RegisterSensor(String sensorName, String floorNo, String roomNo, String sensorStatus, String token)
throws RemoteException;
        // Edit Sensor
        public boolean EditSensor(int sensorId, String sensorName, String floorNo, String roomNo, String sensorStatus, String
token) throws RemoteException;
        // Delete sensor
        public boolean DeleteSensor(int sensorId, String token) throws RemoteException;
}
```

File Name : IUserServices

```
/*
 *      @Author             :       Ashen Senevirathne
 *      @Interface Name             :       IUserServices
 *      @Description                :       Define the all remote methods related to the users
 *
*/
```

```
public interface IUserServices extends Remote{

        // Sign in method
        public String SignIn(String userEmail, String userPassword) throws RemoteException;

}
```

File Name : SensorServicesImpl

```
/*
 *      @Author         :       Ashen Senevirathne
 *      @Class Name             :       SensorServicesImpl
 *      @Description    :       Implementation of all methods in ISensorServices
 *
*/

public class SensorServicesImpl extends UnicastRemoteObject implements ISensorServices{

        // Constructors
        public SensorServicesImpl() throws RemoteException {
                super();
        }

        @Override
        public String GetAllSensors() throws RemoteException {

                // Create client object to send request to API
                Client client = ClientBuilder.newClient();

                // Call the API and get response
                Response response = client
                                .target(ApiConstant.API_SERVER)                 // Server URL
                                .path(ApiConstant.GET_SENSOR_DETAILS_PATH) // Endpoint path
                                .request(MediaType.APPLICATION_JSON)                    // Request Media Type
                                .get();
        // Request method type

                // Check response status equals to 200
                // If it is 200 that mean status is OK. If not that mean some error happen.
                if(response.getStatus() == 200) {
                        // Get a json object from response
                        JsonObject jsonResponse = response.readEntity(JsonObject.class);
                        // Then check the json object status
                        // In the API every response have a two attribute. One is "status" and another one is "data".
                        // "status" is boolean value. If it's true that mean API return valid response and "data" contains the
Sensor objects
                        // If "status" is false data attribute contains the error message.
                        if(jsonResponse.getBoolean("status")) {
                                // Initialize the tread and call SendEmailsAndSms function to send emails and SMS for the
users
                                Thread newThread = new Thread(() -> {
                          for(double i=0; i<5; i++){
                                // Call SendEmailsAndSms method with passing sensor details
                                //SendEmailsAndSms(jsonResponse);
                          }
                                });
                                newThread.start();
                                // Return data
                                return jsonResponse.get("data").toString();
                        }
                }

                // return null if status code not 200
                return null;

        }

        private void SendEmailsAndSms(JsonObject jsonResponse) {
```

```
                    // Get sensor details into sensor array
                    Sensor list[] = new Gson().fromJson(jsonResponse.get("data").toString(), Sensor[].class);
                    // Create List object suing array
                    List<Sensor> sensorList = Arrays.asList(list);
                    // Firealaram message
                    String fireLocation = "";

                    // Check every sensor to identify smoke and co2 value > 5
                    for(Sensor sensor : sensorList) {
                              if(sensor.getCo2Level() > 5 && sensor.getSmokeLevel() > 5) {
                                        // Set Firelocation message
                                        // Store the sensor id and location of the sensor to send the users
                                        fireLocation +=  "Sensor " + sensor.getSensorId() + " = Room No : " + sensor.getRoomNo()
+ " Floor No : " + sensor.getFloorNo() + "\n";
                              }
                    }

                    // Check if fireLocation is not null. If it's not null means at least one of the sensor are detected the fire
                    if(fireLocation != "") {

                              // Create client object to send request to API
                              Client client = ClientBuilder.newClient();

                              // Call the API and get response
                              Response response = client
                                                  .target(ApiConstant.API_SERVER)                          // Server URL
                                                  .path(ApiConstant.GET_USER_EMAIL_MOBILE)    // Endpoint path
                                                  .request(MediaType.APPLICATION_JSON)                    //   Request   Media
Type
                                                  .get();
                    // Request method type

                              // Check response status equals to 200
                              // If it is 200 that mean status is OK. If not that mean some error happen.
```

```
if(response.getStatus() == 200) {
        // Get a json object from response
        JsonObject apiRes = response.readEntity(JsonObject.class);
        // Check json object status
        if(apiRes.getBoolean("status")) {

                // Get data body from the json object
                JsonObject data = apiRes.getJsonObject("data");

                // Get email and mobile number lists as arrays
                JsonArray jsonEmailList = data.getJsonArray("email");
                JsonArray jsonMobileList = data.getJsonArray("mobile");

                // Set Email and Mobile No list to String
                String emailList = "";
                String mobileList = "";

                for(int i = 0; i < jsonEmailList.size(); i++) {
                        emailList += jsonEmailList.getString(i) + ",";
                        mobileList += jsonMobileList.getString(i) + ",";
                }

                // Finalize the email and mobile list
                emailList = emailList.substring(0, emailList.length() - 1);
                mobileList = mobileList.substring(0, mobileList.length() - 1);

                // Call the Email service method to send email
                EmailService.SendEmail(emailList,
fireLocation.substring(0,fireLocation.length() - 1));
                // Call Sms service method to send sms
                //SmsService.SendSms(mobileList,
fireLocation.substring(0,fireLocation.length() - 1));
        }
}
```

```
		}
	}

	@Override
	public boolean RegisterSensor(String sensorName, String floorNo, String roomNo, String sensorStatus, String token)
			throws RemoteException {

		// Create client object to send request to API
		Client client = ClientBuilder.newClient();

		// Create a RegisterSensor object using pass parameters
		RegisterSensor registerSensor = new RegisterSensor(sensorName, floorNo, roomNo, sensorStatus);

		// Call the API and get response
		Response response = client
				.target(ApiConstant.API_SERVER)
				// Server URL
				.path(ApiConstant.REGISTER_NEW_SENSOR)
			// Endpoint path
				.request(MediaType.APPLICATION_JSON)
			// Request Media Type
				.header(HttpHeaders.AUTHORIZATION, token)
		// Set AUTHORIZATION header
				.post(Entity.entity(registerSensor, MediaType.APPLICATION_JSON)); // Request method
type

		// Check response status equals to 200
		// If it is 200 that mean status is OK. If not that mean some error happen.
		if(response.getStatus() == 200) {
			// Get a json object from response
			JsonObject jsonResponse = response.readEntity(JsonObject.class);
			// Then check the json object status
			if(jsonResponse.getBoolean("status")) {
				return true;
			}
		}
```

```
        // return null if status code not 200
        return false;
    }

    @Override
    public boolean EditSensor(int sensorId, String sensorName, String floorNo, String roomNo, String sensorStatus, String
token)
            throws RemoteException {

        // Create client object to send request to API
        Client client = ClientBuilder.newClient();

        // Create a EditSensor object using pass parameters
        RegisterSensor editSensor = new EditSensor(sensorId, sensorName, floorNo, roomNo, sensorStatus);

        // Call the API and get response
        Response response = client
                .target(ApiConstant.API_SERVER)
            // Server URL
                .path(ApiConstant.EDIT_SENSOR)
            // Endpoint path
                .request(MediaType.APPLICATION_JSON)
        // Request Media Type
                .header(HttpHeaders.AUTHORIZATION, token)
    // Set AUTHORIZATION header
                .put(Entity.entity(editSensor, MediaType.APPLICATION_JSON));        // Request method
type


        // Check response status equals to 200
        // If it is 200 that mean status is OK. If not that mean some error happen.
        if(response.getStatus() == 200) {
            // Get a json object from response
            JsonObject jsonResponse = response.readEntity(JsonObject.class);
            // Then check the json object status
            if(jsonResponse.getBoolean("status")) {
```

```
                        return true;
                    }
                }
                // return null if status code not 200
                return false;
            }

            @Override
            public boolean DeleteSensor(int sensorId, String token) throws RemoteException {

                Client client = ClientBuilder.newClient();

                Response response = client
                        .target(ApiConstant.API_SERVER)                         // Server URL
                        .path(ApiConstant.DELETE_SENSOR)                  // Endpoint path
                        .path(String.valueOf(sensorId))                          //   Request   Media
        Type
                        .request(MediaType.APPLICATION_JSON)             //      Set      AUTHORIZATION
        header
                        .header(HttpHeaders.AUTHORIZATION, token)     // Request method type
                        .delete();

                // Check response status equals to 200
                // If it is 200 that mean status is OK. If not that mean some error happen.
                if(response.getStatus() == 200) {
                    // Get a json object from response
                    JsonObject jsonResponse = response.readEntity(JsonObject.class);
                    // Then check the json object status
                    if(jsonResponse.getBoolean("status")) {
                        return true;
                    }
                }

                // return null if status code not 200
                return false;
        }

}
```

File Name : UserServicesImpl

```
/*
 *      @Author             :       Ashen Senevirathne
 *      @Class Name                 :       SensorSeUserServicesImplrvicesImpl
 *      @Description        :       Implementation of all methods in UserServicesImpl
 *
*/

public class UserServicesImpl extends UnicastRemoteObject implements IUserServices{

        public UserServicesImpl() throws RemoteException {
                super();
        }

        @Override
        public String SignIn(String userEmail, String userPassword) throws RemoteException {

                // Create client object to send request to API
                Client client = ClientBuilder.newClient();

                // Create object of LoginUser using passed parameters
                LoginUser loginUser = new LoginUser(userEmail, userPassword);

                // Call the API and get response
                Response response = client
                                .target(ApiConstant.API_SERVER)
                        // Server URL
                                .path(ApiConstant.SIGNIN_METHOD)
                // Endpoint path
```

```
                    .request(MediaType.APPLICATION_JSON)
        // Request Media Type

                    .post(Entity.entity(loginUser, MediaType.APPLICATION_JSON));       // Request method
type



        // Check response status equals to 200

        // If it is 200 that mean status is OK. If not that mean some error happen.

        if(response.getStatus() == 200) {

            // Get a json object from response

            JsonObject jsonResponse = response.readEntity(JsonObject.class);

            // Then check the json object status

            if(jsonResponse.getBoolean("status")) {

                // return information of login user

                return jsonResponse.get("data").toString();

            }

            return null;

        }


        return null;


    }


}
```