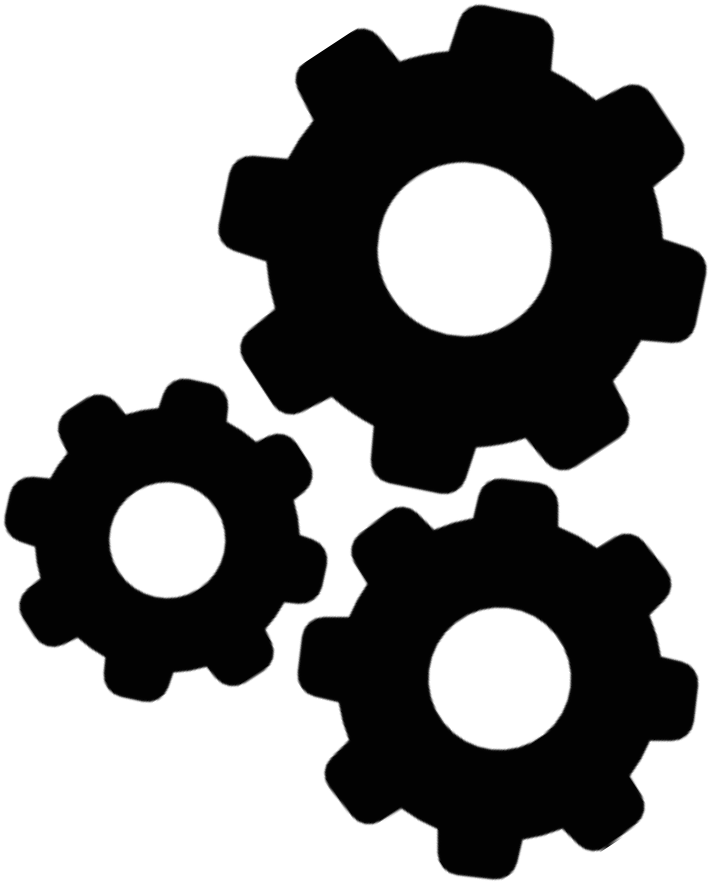
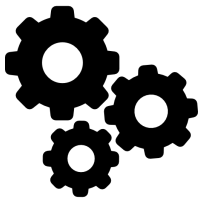
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Software Engineering & UML : Design

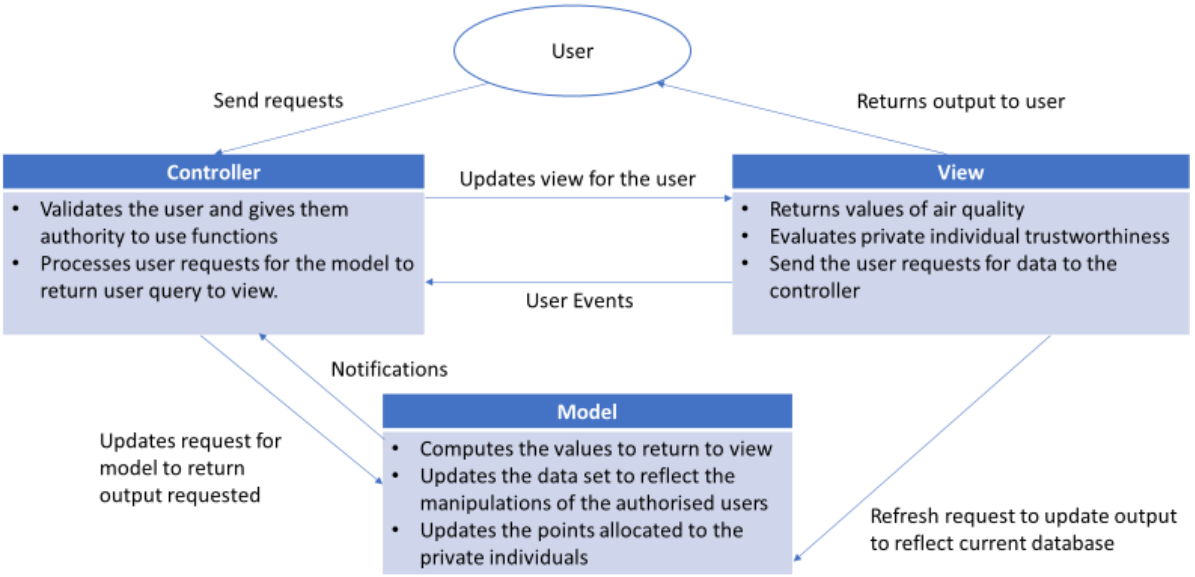


Group : 3IF-3

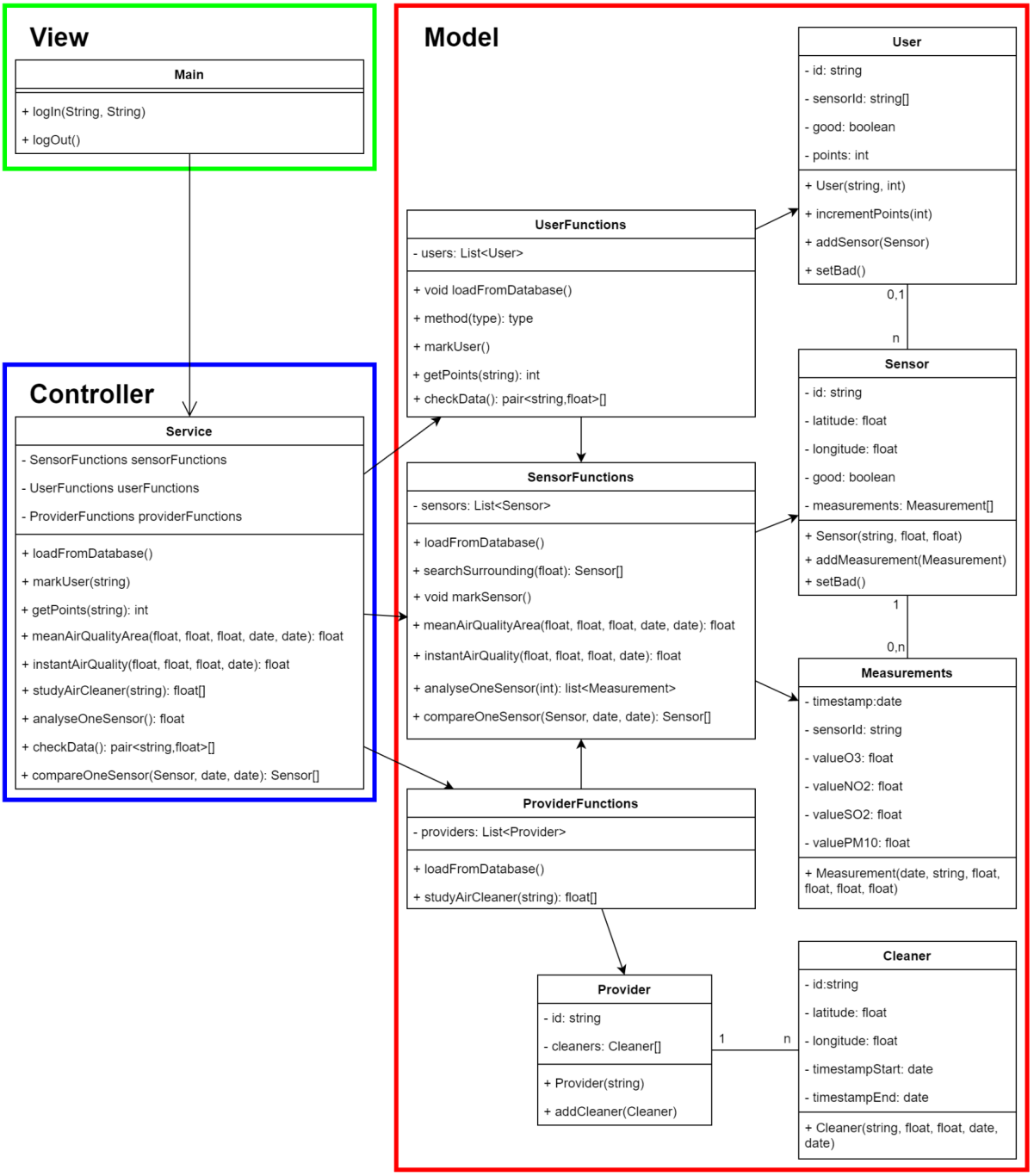
Pairs : B3320 - B3331



Architecture, modular decomposition

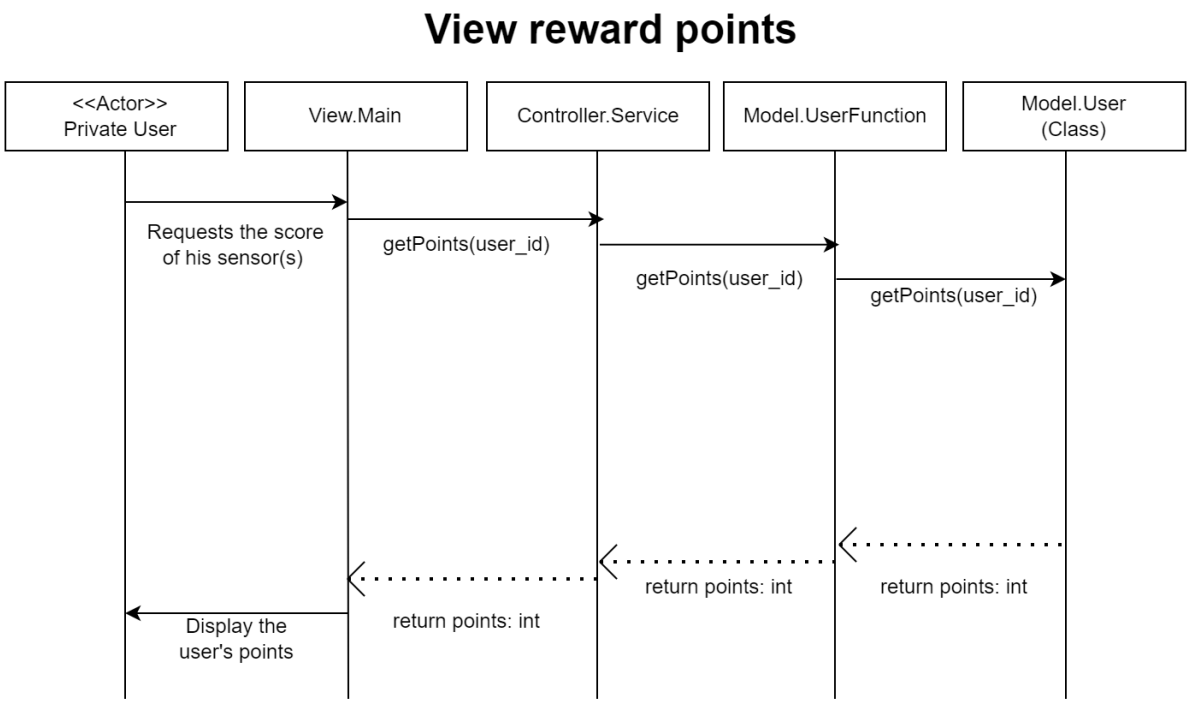


Class diagram

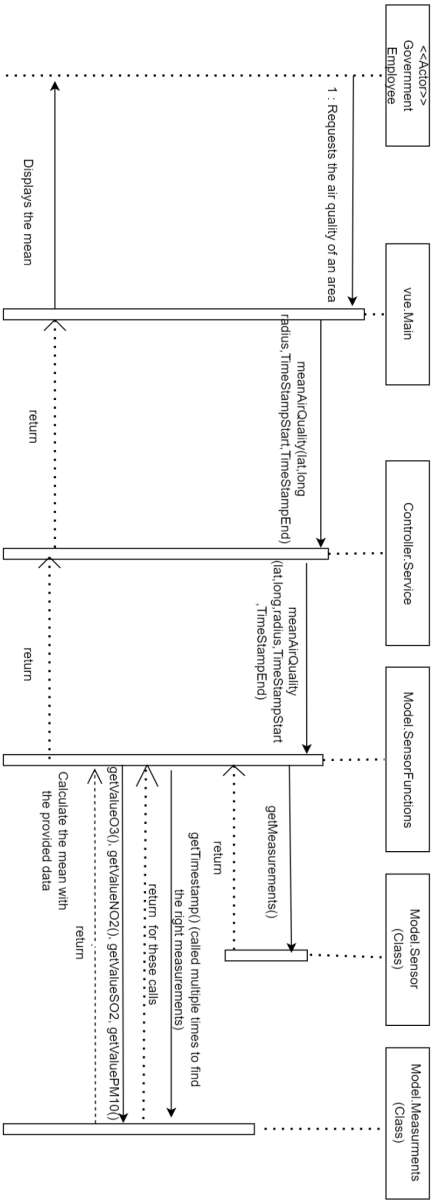


Getters are always set, but were omitted to simplify the class diagram.
The controller is mainly in charge of passing the user's inputs to the other classes to carry out the functions so as to ensure security of the data values as the user does not interact directly with the data.
When a private individual is marked as untrustworthy, the data associated with their sensors are deleted but the instance of the individual remains.
The parameters to assess the effectiveness of the air cleaners are intrinsically programmed into the classes and hence the program will analyze the effectiveness without need of further input from the user.

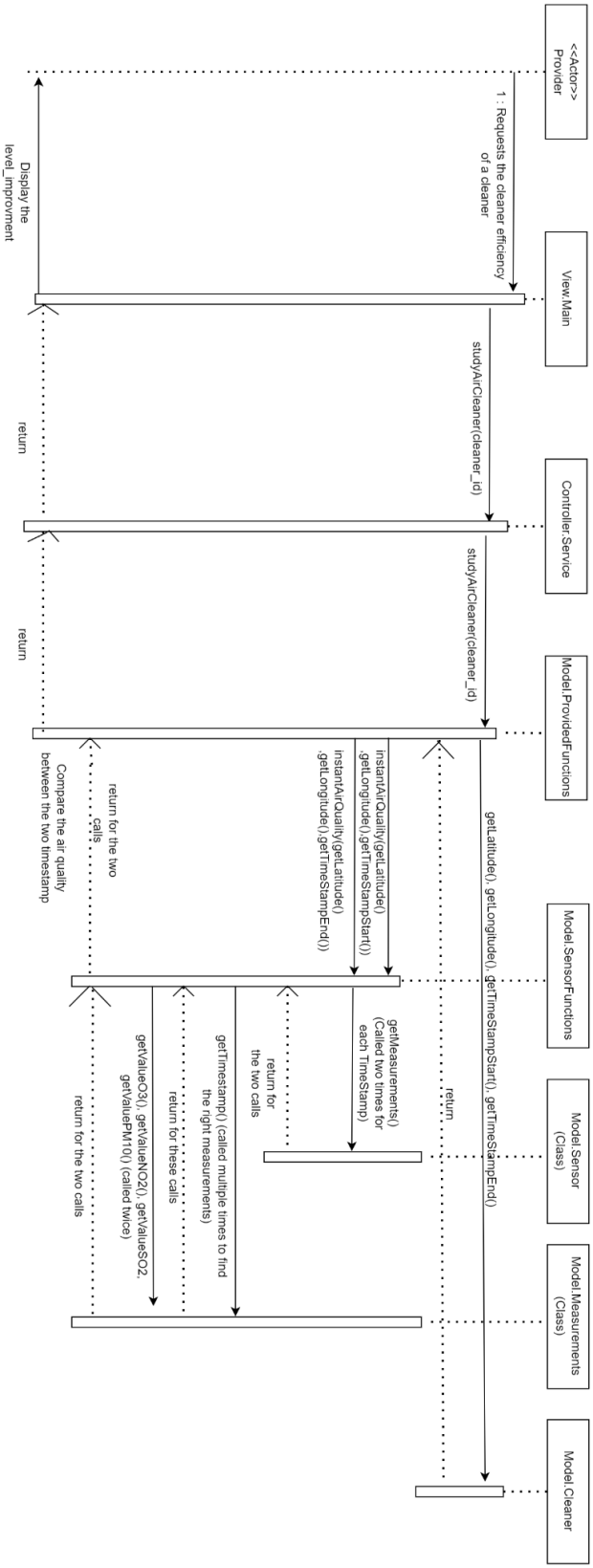
Sequence diagrams of three major scenarios



Mean Air Quality within an area



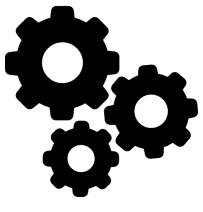
Analysis of cleaner efficiency



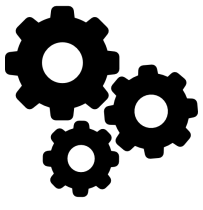
Description and pseudo-code of three major algorithms

```
Function      analyzeOneSensor
Description   analyze a sensor, if the score is too high the sensor is not trustworthy
Input         Sensor sensor
Output        float score
Precondition  sensor exists
Declaration   float average
              float AQI
              float sum
Begin
    for each date
        average = average AQI of surrounding sensors
        AQI = getAQI(sensor, date)
        sum += abs(average - AQI) / average
    score = sum / numDate
End
```

```
Function      meanAirQualityArea
Description   calculate the mean of AQI of a circular area in a given timeframe
Input         float latitude, longitude, radius
              Sensor[] sensorList
              date timeframe
Output        float mean
Precondition  valid latitude and longitude in degrees, reasonable radius in meter
Declaration   float sum = 0
              int num = 0
Begin
    for each date in timeframe
        for each sensor in sensorList
            if sensor is in the circle
                num++
                sum += getAQI(sensor, date)
    mean = sum / num
End
```

```
Function      compareOneSensor
Description   find areas with similar air quality of a certain sensor in a given timeframe
Input         sensor targetSensor
              Sensor[] sensorList
              date timeframe
Output        Sensor[] sorted
Precondition  valid targetSensor
Declaration   float difference[] = {0}
              float AQI
              int i
Begin
        sorted = full copy of sensorList

        for each date in timeframe
                AQI = getAQI(targetSensor, date)
                for (i = 0; i < sensorList.size; i++)
                        difference[i] += abs(getAQI(sensorList[i], date) - AQI)
        sort difference and sorted the same time by ascending difference
End
```

Unit Tests

Measurement

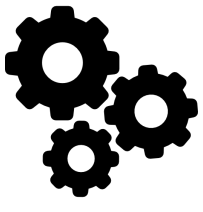
Test creation

```
public void testMeasurementCreation(){
    time_t date = "2022-05-04 12:00:00";
    string sensorId = 1;
    float valueO3 = 0.05;
    float valueNO2 = 0.05;
    float valueSO2 = 0.05;
    float valuePM10 = 0.05;
    Measurement measurement = new Measurement(date, sensorId, valueO3,
valueNO2, valueSO2, valuePM10);
    assert(measurement != NULL);
    assert(measurements.getDate() == date);
    assert(measurement.getSensorId() == sensorId);
    assert(measurement.getValueO3() == valueO3);
    assert(measurement.getValueNO2() == valueNO2);
    assert(measurement.getValueSO2() == valueSO2);
    assert(measurement.getValuePM10() == valuePM10);
}
```

Sensor

Test creation

```
public void testSensorCreation(){
    string id = 1;
    float lat = 49.05;
    float lon = -85.23;
    Sensor sensor = new Sensor(id, lat, lon);
    assert(sensor != NULL);
    assert(sensor.getId() == id);
    assert(sensor.getLatitude() == lat);
    assert(sensor.getLongitude() == lon);
    assert(sensor.getGood() == true);
    assert(sensor.getMeasurements() != NULL);
    assert(sensor.getMeasurements().size() == 0);
}
```

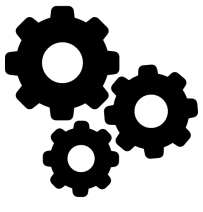



Test flagging

```
public void testSensorFlag(){
    string id = 1;
    float lat = 49.05;
    float lon = -85.23;
    Sensor sensor = new Sensor(id, lat, lon);

    sensor.setBad();
    assert(sensor.getGood() == false);
}
```

Test adding Measurements

```
public void testAddMeasurement(){
    String id = 1;
    float lat = 49.05;
    float lon = -85.23;
    Sensor sensor = new Sensor(id, lat, lon);

    time_t date = "2022-05-04 12:00:00";
    string sensorId = 1;
    float valueO3 = 0.05;
    float valueNO2 = 0.05;
    float valueSO2 = 0.05;
    float valuePM10 = 0.05;
    Measurement measurement = new Measurement(date, sensorId, valueO3,
valueNO2, valueSO2, valuePM10);

    sensor.addMeasurement(measurement);
    assert(sensor.getMeasurements().size() == 1);
    /* Needs a redefinition of the == operator for Measurement */
    assert(sensor.getMeasurements().front() == measurement);
}
```

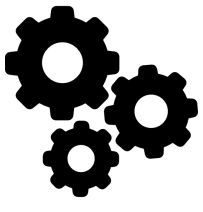
User

Test creation

```
public void testUserCreation(){
    string id = 1;
    int initialPoints = 0;
    User user = new User(id, initialPoints);
    assert(user != NULL);
    assert(user.getId() == id);
    assert(user.getPoints() == initialPoints);
    assert(user.getGood() == true);
    assert(user.getSensors() != NULL);
    assert(user.getSensors().size() == 0);
}
```

Test flagging

```
public void testSensorFlag(){
    string id = 1;
    int initialPoints = 0;
    User user = new User(id, initialPoints);

    user.setBad();
    assert(user.getGood() == false);
}
```

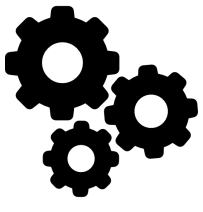
Test adding Sensors

```
public void testAddSensor(){
    string id = 1;
    int initialPoints = 0;
    User user = new User(id, initialPoints);

    string sensorId = 1;
    float lat = 49.05;
    float lon = -85.23;
    Sensor sensor = new Sensor(sensorId, lat, lon);

    user.addSensor(sensor);
    assert(user.getSensors().size() == 1);
    /* Needs a redefinition of the == operator for Sensor */
    assert(user.getSensors().front() == sensor);
}
```

Cleaner

Test creation

```
public void testCleanerCreation(){
    string id = 1;
    float lat = 49.05;
    float lon = -85.23;
    time_t timestampStart = "2022-05-04 12:00:00";
    time_t timestampEnd = "2022-05-04 13:00:00";
    Cleaner cleaner = new Cleaner(id, lat, lon, timestampStart,
timestampEnd);
    assert(cleaner != NULL);
    assert(cleaner.getId() == id);
    assert(cleaner.getLatitude() == lat);
    assert(cleaner.getLongitude() == lon);
    assert(cleaner.getTimestampStart() == timestampStart);
    assert(cleaner.getTimestampEnd() == timestampEnd);
}
```

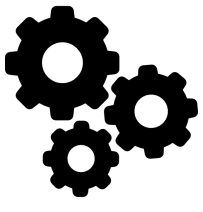
Provider

Test creation

```
public void testProviderCreation(){
    string id = 1;
    Provider provider = new Provider(id);
    assert(provider != NULL);
    assert(provider.getId() == id);
    assert(provider.getCleaners() != NULL);
    assert(provider.getCleaners().size() == 0);
}
```

Test adding Cleaners

```
public void testAddCleaner(){
    string id = 1;
    Provider provider = new Provider(id);

    String cleanerId = 1;
    float lat = 49.05;
    float lon = -85.23;
    time_t timestampStart = "2022-05-04 12:00:00";
    time_t timestampEnd = "2022-05-04 13:00:00";
    Cleaner cleaner = new Cleaner(cleanerId, lat, lon, timestampStart,
timestampEnd);

    provider.addCleaner(cleaner);
    assert(provider.getCleaners().size() == 1);
    /* Needs a redefinition of the == operator for Cleaner */
    assert(provider.getCleaners().front() == cleaner);
}
```

Service

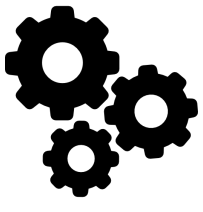
Most of the tests assume that there is a relationship of friendship (in the C++ sense) between the test class and the other classes.

Test creation

```
public void TestServiceCreation(){
    Service service = new Service();
    assert(service != NULL);
    assert(service.sensorFunction != NULL);
    assert(service.userFunctions != NULL);
    assert(service.providerFunctions != NULL);
}
```

Test loading from database (csv)

```
public void testLoadFromDatabse(){
    Service service = new Service();
    service.loadFromDatabse();
    assert(service.sensorFunctions.sensors.size() > 0);
    assert(service.userFunctions.users.size() > 0);
    assert(service.providerFunctions.providers.size() > 0);
}
```

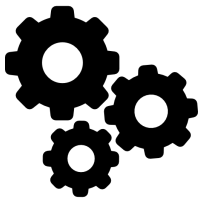



Test flagging User

```
public void testMarkUser(){
    /* Initialize Service */
    Service service = new Service();
    service.loadFromDatabase();
    /* Get list of score per user */
    Map<string,float> scores = NULL;
    scores = service.checkData();
    assert(scores != NULL);
    assert(scores.size() > 0);
    /* Marking the first User */
    service.flag(scores.begin()->first);
    /* Testing that there is a user that is marked as not good. */
    bool isBad = false;
    for (User user : service.userFunctions.users){
        if (user.getGood() == false){
            /* Asserting the right user is marked */
            assert(user.getId() == scores.begin()->first);
            isBad = true;
            break;
        }
    }
    assert(isBad == true);
}
```

Test getting points of an User

```
public void testGetPoints(){
    /* Initialize the service */
    Service service = new Service();
    service.loadDatabase();
    int points =
service.getPoints(service.userFunctions.users.get(0).getId());
    assert(points == service.userFunctions.users.get(0).getPoints());
}
```

Test calculation of the mean air quality in an area

```
public void testMeanAirQuality(){
    /* Initialize the serice */
    Service service = new Service();
    service.loadDatabase();
    /* the loaded database should be a mock database where all values
are known */
    float lat = 49.05;
    float lon = -85.23;
    float radius = 85;
    time_t date = '2021-05-03';
    float quality = service.meanAirQuality(lat, lon, radius, date);
    /* assert the quality has been properly calculated */
    float expectedQuality = /* Should be known in the mock database */
    assert(quality == expectedQuality);
}
```

Test calculation of air quality in a specific location

```
public void testInstantAirQuality(){
    * Initialize the serice */
    Service service = new Service();
    service.loadDatabase();
    /* the loaded database should be a mock database where all values
are known */
    float lat = 49.05;
    float lon = -85.23;
    time_t date = '2021-05-03 13:00:00';
    float quality = service.instantAirQuality(lat, lon, date);
    /* assert the quality has been properly calculated */
    float expectedQuality = /* Should be known from the data in the
mock database */;
    assert(quality == expectedQuality);
}
```