Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 01, January-2021

# Designing of A Sensor and Mobile Application for Identification of Tomato Plant Physiological State

[*] [1, 2, a] Teuma Mbezi M., [3, b] Deussom Djomadji E.M,
[4, c] Ambang Zachee, [1, d] Ekobena Fouda H.P.

[1]Laboratory of Biophysics, Department of Physics, Faculty of Sciences, University of Yaoundé I, P.O. Box 812, Yaoundé, Cameroon.
[2]National Advanced School of Posts, Telecommunications and Information and Communication Technologies.
3 College of technology, University of Bua
[4]Laboratory of Phytopathology, Department of biology and vegetal physiology Faculty of Sciences, University of Yaoundé I, P.O. Box 812, Yaoundé, Cameroon

***Abstract*:- Context/problematic: Generally, the appearance of visible symptoms such as leaf depigmentation, poor fruit quality means that the disease has already created a lot of damages in the plant. The problem is therefore, how to automatically identify a diseased plant before the visible manifestation of its first symptoms. Objective: Our goals are to design firstly a fluorescence sensor able to measure physiological state bio-indicator parameters of tomato plant. Secondly, a mobile application under android studio which will receive bio-indicator parameters emitted by the sensor, perform necessary calculations to deduce the physiological state of the plant, and to show a message concerning plant state to the user through an interface. Methods/ analysis: The sensor designing was done using Isis Proteus software, and the mobile application using four UML (Unified Modelling Language) diagrams. The code was developed under android studio. Findings: At the end of the designing work, there emerged a sensor consisting of two stages, a transmission stage and a reception stage. Each stage consists of an assembly of electronic components allowing them to perform their function properly. The mobile application, to function properly, required the use of eight UML classes. The code of each of those classes was written and the mobile application interface is presented. Application/Improvements: This work is of great importance because, its implementation will allow to act quickly in case of plant infection, which will obviously have the effect of increasing agricultural production and food quality.***

***Keywords: Sensor, fluorescence, UML diagrams, class.***

## 1. INTRODUCTION

One of the factors that inhibits tomato crop production is the presence of pathological agents that interfere with proper plant growth. Although all precautionary measures have been taken, the plantations are often damaged by the *infesting phytophthora,* the pathological agent of *tomato mildew*. Usually, the appearance of visible symptoms such as leaf depigmentation, poor fruit quality means that the disease has already created extensive damages in the plant [1]. The problem that arises is how to automatically identify a diseased tomato plant before the visible manifestation of its first symptoms?

Nowadays, there are devices in the category of connected things (Internet Of Thing) such as *Parrot Flower Power* and *Edyn* which allow their users to be informed about the physiological state of their plants during and after their growth. These devices, made up of sensors, measure the conductivity of the soil, the level of humidity and light in order to be able to deduce health status of the plant [2, 3]. Now a pathology such as *tomato mildew* does not modify any of the parameters measured by the above devices; therefore neither *Flower Power* nor *Edyn* are able to automatically identify an attack of the *infesting phytophthora* before the visible manifestation of a sick tomato plant first symptoms. The same limitation can be found when using systems such as the one name *Expert System* based on *step by step and graphical method* of plant diagnosis [4]; or method for plant disease recognition based on leaf image classification, by the use of deep convolutional neural networks [5].

The aims of this work is first of all to design a fluorescence sensor which is able to measure bio-indicator parameters of plant physiological state, and a mobile application under android which will receive bio-indicator parameters emitted by the sensor, perform calculations to deduce plant physiological state and then inform the user via an interface about the plant state.

The rest of paper is therefore presented as follows: we will present the material, explain the methods and design model used to obtain the presented, analysed and discussed results. At the end we have some concluded remarks of the paper.

## 2. MATERIALS AND METHODS

We present two categories of materials and methods. The first are materials and method used to design the sensor then the second are materials and method used to design the mobile application.

## 2.1 Materials and methods related to the designing of the sensor

Our sensor consists of two stages, a transmission stage and a reception stage. The emission stage aims to emit a supersaturated light (light with sufficient high intensity to close all the PSII centers) which will produce maximum fluorescence *Fm* (Maximum fluorescence. Fluoresce level when a hight intensity flash has been applied. All antenna sites are assumed to be close) [6] then a modulated light (analytical light with sufficiently low intensity to not induce in significantly manner the closure of PSII centers) which will cause primary fluorescence *Fo* (Minimal fluorescence. Fluorescence level when all antenna pigment complexes associated with the photosystem are assumed to be open) [6]. The reception stage aims to detect, measure those two types of fluorescence and send data through Bluetooth to the user mobile application.

**a. transmission stage**

In *Isis Proteus*, we have chosen many electronics components such has:

- a bulb which aim to produce supersaturated light which will lead to *Fm* maximum fluorescence
- a red LED (Light-Emitting Diode) which aim to produce modulated light which will lead to *Fo* primary fluorescence
- a APDS-9002 TRX photodiode, its purpose is to switch on the sensor only in night. This is to avoid interference with other light sources
- a ATEMEGA16 microcontroller. It was programed using CodeVision AVR to let lit continuously the red LED and make lit intermittently the lamp. The time space between intermittent light is one second
- a resistor of protection

The modulated and supersaturated light are both emitted towards plant leaf, which reflected them as *Fm* and *Fo* fluorescence to the reception stage.

**b. reception stage**

At the entrance of the reception stage, we have:

- a Torch-LDR (Light Dependent Resistor) which received and transformed both *Fm* and *Fo* light signal to and electric signal
- a blue LED which is a loud
- a resistor of protection
- a Bluetooth module which aim to transmit electric data correspond to *Fm* and *Fo* to a smartphone where the mobile application is installed.

The method simply consisted to connect each above electrics component whether in series or in parallel according to the research purpose. The obtained results are presented in figure 3.1 and 3.2

## 2.2 Materials and methods related to the designing of the application

We have used four UML diagrams, and android studio as development environment. For programming, we used the M.V.C (Model, View, Control) approach. In the Model package we have the '*Profil'* class, in the Control one we have "*Control*" class and "MainActivity", "Thread", "ClientClass","ServerClass" and "SendReiceive" classes are contained in View package. The application also include a Manifest file which containing all the permissions related to the use of Bluetooth and the mathematical functions used in the application.

**a. Use case diagram**

The use case diagram represents the structure of the major functionalities required by the users of the system. This is the first diagram of the UML model, the one where ensuring the relationship between the user and the objects that the system implements [7].

Our diagram consists of two actors, one *main actor* and one *secondary actor*. The main (actor) is the user who wants to assess the state of health of his plant, or send a message, the secondary (actor) is the sensor which provides necessary fluorescence information to the system (the application) for the evaluation of health status. We distinguish two use cases namely: "*Evaluate health*" and *"Send message*" because after having obtained the result concerning the state of health of his plant, the main actor can decide to send a message to request an intervention. We also distinguish seven internal use cases. Five of them namely: "*Activate Bluetooth*", "*Select address*", "*Enter duration*", "*Choose type of evaluation"*, and "*Fluorescence values*" are linked to the "Evaluate health" case by an inclusion relationship stereotyped by "include" , because when the "Evaluate health" case is requested, these five internal cases must be requested. The "Send message" case is linked to the "Enter message" case by an inclusion relation and to the "Read message" case by an extension relation stereotyped by "extend", because the execution of the "Send message" case can optionally cause the execution of the "Read message" case. The "Send message" case is also linked to the "Activate Bluetooth" and "Select address" cases by an inclusion relationship, because sending a message necessarily requires the activation of the Bluetooth and the selection of the recipient's address. The result of this use case diagram is presented in figure 3.5.

### b. Activity diagram

The activity diagrams allow to focus on the treatments. They are therefore suitable for modelling the routing of control flows and data flows. They graphically represent the behaviour of a method or the course of a use case [7].

Our activity diagram will consist of 13 action nodes, two decision nodes, one initial node and one end of activity node. It will be divided into 3 partitions namely a "user", the application named "eplanthealth" and a "*sensor*".

The "user" partition will successively contain the action nodes *"Activate Bluetooth*", "*Select address*", "*Enter duration*", "*Choose type of evaluation*", "*Evaluate health*", *"Enter message*" and *"Send message*".

The "*eplanthealth*" partition will include the action nodes "*Bluetooth search*", "*Establish connection*", *"Parameter calculation*". It will also contain one decision node (which will indicate the action to be followed depending on the state of Bluetooth connectivity) and one end activity node.

The "*sensor*" partition will have the action node "*Data transfer*" and a decision node which will decide when to transfer the fluorescence data. The result of this activity diagram is presented in figure 3.6.

### c. Sequence diagram

The sequence diagram represents the chronological succession of operations carried out by an actor. It indicates objects that actor will manipulate and the operations that move from one object to another. The main information contained in a sequence diagram are the messages exchanged between lifelines, presented in chronological order [7].

The sequence diagram will consist of three lifelines each representing one object namely, a user "*user*", the application *": Application*" and a sensor ": *sensor*". It will contain synchronous messages, response messages and two interactions fragments *"alt"*. One interaction fragment will be represented on the lifeline of the ": Application" object and another will connect the lifelines of ": *Application*" and ": *sensor*" objects. The result of this sequence diagram is presented in figure 3.7.

### d. Class diagram

The class diagram is generally considered as the most important in object-oriented development. It represents the conceptual architecture of the system: it describes the classes that the system uses, as well as their links, these represent a conceptual nesting (inheritance) or an organic relation (aggregation) [7].

- ***Profil class***

The class "*Profil*" (Profile) (Figure 2.1) will contain a constructor which takes as parameter an integer *"typeEva"*' to indicate the type of evaluation (the application will have two types of evaluations: a standard evaluation and a comparative evaluation), an integer *"dureeFluo*" to take into account the duration during which the fluorescence values *Fo* and *Fm* will be transmitted from the sensor for the application, and a table "*tabValRecu*" which will contain all the fluorescence values received. It will also contain the methods *calculEff ()* to assess the average photosynthetic efficiency *(Fm-Fo / Fm), calculMaxEff ()* to identify the maximum photosynthetic efficiency, *"message ()"* to provide the message relating to the health state of the plant. The "calculDsp ()" and "calculEcartype ()" methods to calculate respectively the power spectral density and the standard deviation of the photosynthetic efficiency.

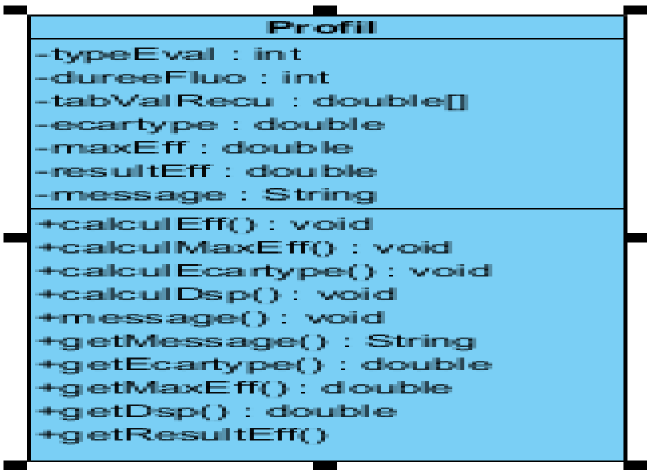


Figure 2.1: Profil class

- ***Controle class***

The *"Controle"* class (Control) (Figure 2.2) contained in the *"Controle" package* is a class that acts as an intermediary between the Model package (which contains the *Profil class*) and the "*View*" package. It creates and retrieves a working instance using the "*getInstance ()*" method and assigns it to the "instance" field. It takes the results from the "*Profile class*"that will be used in the *View* package using the "*creeProfile ()*" method and assigns it to the *"profil"* field.

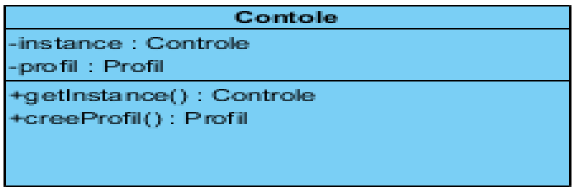


Figure 2.2: *Controle class*

- ***AppCompatActivity class***

The *"AppCompatActivity"* class (Figure 2.3) is a predefined Android studio class that is used to manage applications. The "*MainActivity*" class inherits from it.

Figure 2.3 : *AppCompatActivity class*

- ***MainActivity* class**

"*MainActivity" class* (Figure 2.4) is used to display on the screen using "*afficheResult()*" method and *"TextVie"*, information related to the health plant state coming from *Profil class* through the *"Controle" class*. It retrieves all actions coming from "*Butons*", "*RadioButons*" and "*EditText*" type objects and transfer them to the *Profil class* using *control class*. It also makes possible the establishment of Bluetooth connection using *"bluetoothAdapter"* object, and to obtain the list of identify Bluetooth devices using the "btArray" object. It contains the "*ClientClass*", "*ServerClass*" and "*SendReiceive*" classes.

- ***Thread class***

The *Thread class* (Figure 2.5) is a predefined android studio class that allows certain actions to be performed in different threads, in order to optimize the operation of the application. Thus *MainActivity class* is executed in the main thread, the *"ClientClass*", *"ServerClass*" and *"SendReiceive*" classes which inherit from *"Thread" class* will be executed in secondary threads.

- ***SendReiceive class***

The *"SendReiceive" class* (Figure 2.6) is a subclass of *"MainActivity" class*. It will create a Bluetooth access point using the "*bluetoothSocket"* object, receive the data stream from the sensor using the *"inputStream*"object, and sending a data stream from the application to another application using the *"outputStream"* object. A *"write ()"* method to write received data to a buffer. A *"calculTabBlue ()"* method to retrieve the Bluetooth data and insert it into a "*tabValBlue []"* and *"tabValRecu []"* array which are two fields of the *SendReiceive class*. *GetTabValRecu ()* and *getTbValBlueRecu ()* methods provide access to these arrays in other classes. A *"handler"* object is used to transfer messages between two classes that are executed through different threads. A *"run ()"* method to start the execution of the *SendReiceive class*.

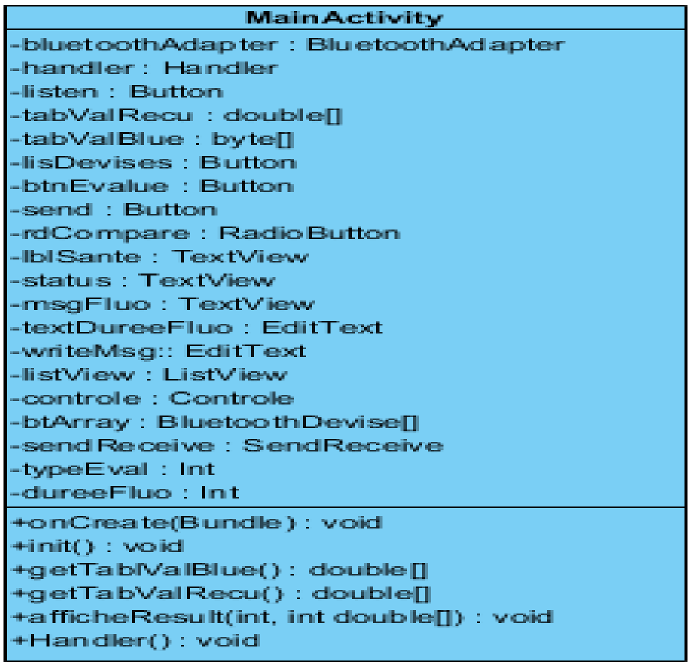


Figure 2.4: *MainActivity class*

Figure 2.5: *Thread class*

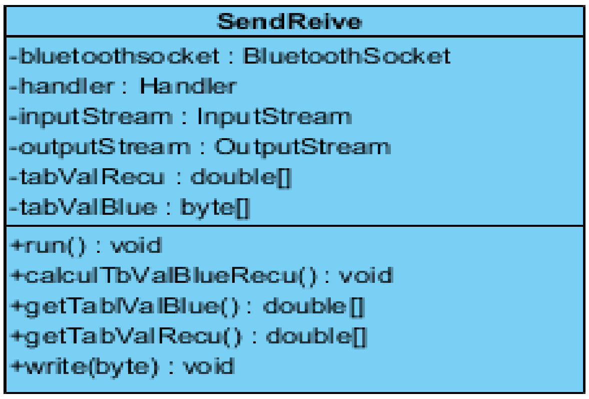

SendReive
-bluetoothsocket : BluetoothSocket
-handler : Handler
-inputStream : InputStream
-outputStream : OutputStream
-tabValRecu : double[]
-tabValBlue : byte[]
+run() : void
+calculTbValBlueRecu() : void
+getTablValBlue() : double[]
+getTabValRecu() : double[]
+write(byte) : void

Figure 2. 6: *SendReiceive class*

- ***ClientClass class***

The *"ClientClass" class* (Figure 2.7) is the one which allows the application running in client mode to receive the fluorescence values coming from a server (which is in this case a sensor) using the *"sendReiceive*"'object. It allows the creation of Bluetooth connection point using the *"socket"* object. It obtained the address of the sensor using the *"device"* object. The "*handler*" object is used to pass a message from "*ClientClass*" class to the "*MainActivity*" class for display purposes. The *"run ()"* method is used to start the execution of the class.

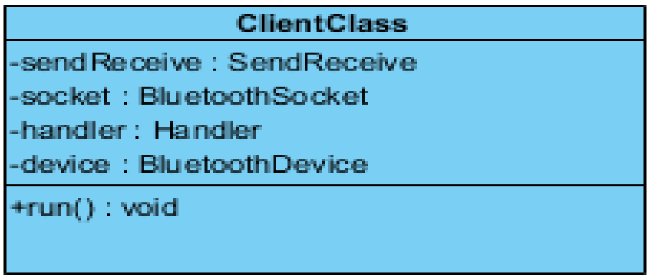

ClientClass
-sendReceive : SendReceive
-socket : BluetoothSocket
-handler : Handler
-device : BluetoothDevice
+run() : void

Figure 2.7 : *ClientClass class*

- ***ServerClass class***

The application can also run in server mode by using the "*ServerClass*" class (Figure 2.8). The use of the "*serverSocket*" object gives the application the ability to function as a server. *"sendReceive*" object allows a message to be sent to another application with which it is connected. *"Handler"* is used to pass a message from *ServerClass* class to *MainActivity* class and the *run ()* method starts the execution of the class.

The relationships between all the above classes will be presented on the class diagram in figure 3.8.

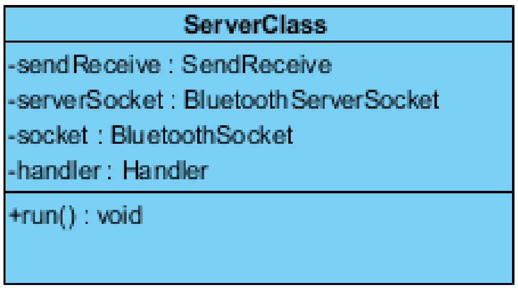

ServerClass
-sendReceive : SendReceive
-serverSocket : BluetoothServerSocket
-socket : BluetoothSocket
-handler : Handler
+run() : void

Figure 2.8: *ServerClass class*

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 01, January-2021

All materials, methods and techniques used to achieve our goals were presented. We designed our sensor and our application. In what follows, it will be presented and discuss different results obtained from the various manipulations.

## 3. RESULTS AND DISCUSSION

In this section, we present and simulate under *Isis Proteus* the two stages of our sensor. We also present the four UML diagrams, the application's interface and we make a unitary test under android studio of the "*profil class*".

### 3.1 Sensor

The stages of our sensor are presented and simulated under light and darkness.

**a. Transmission stage**

The emission stage (Figure 3.1) operates only in the darkness. It emits two types of lights. A modulated light generated by the red LED which will have the effect of producing the primary fluorescence *Fo* and a supersaturating light produced by the bulb which will have the effect of producing the maximum fluorescence *Fm.*

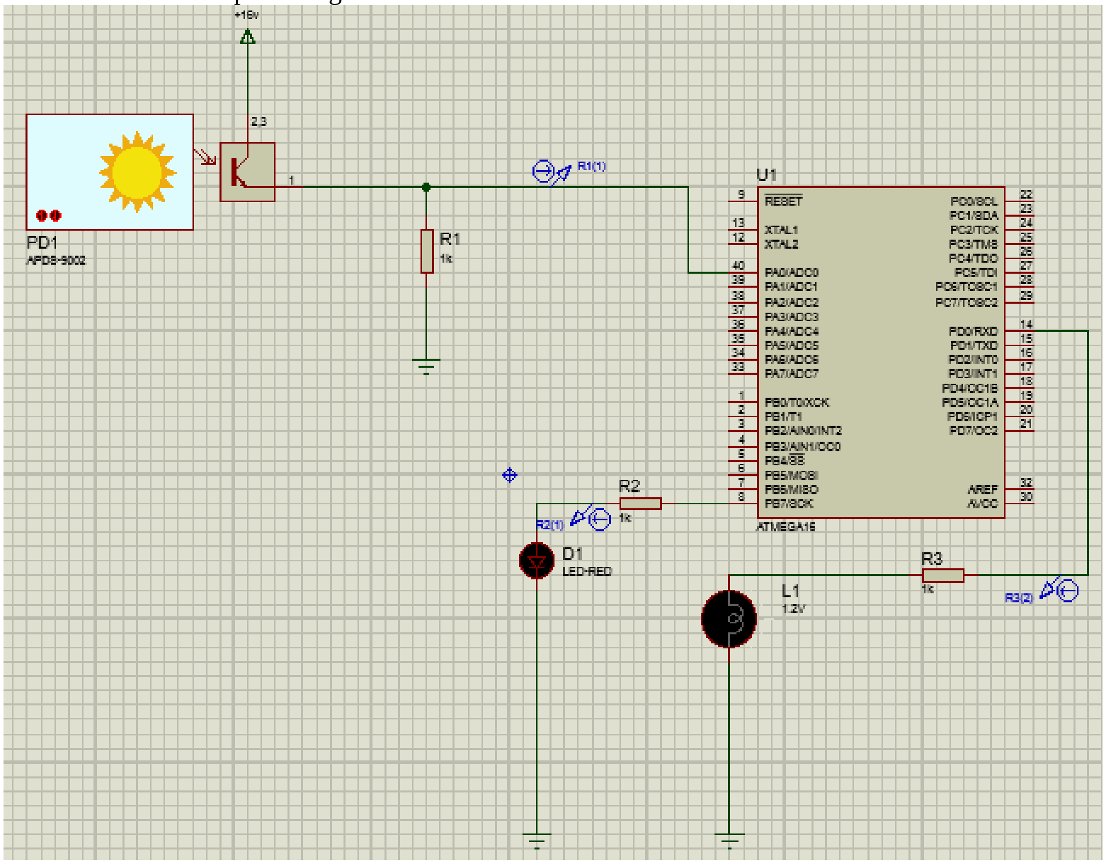


Figure 3. 1: Transmission stage

In the presence of light, the probe indicates main current of 29.1µA and that no current passes through either the LED or the bulb (Figure 3.2).

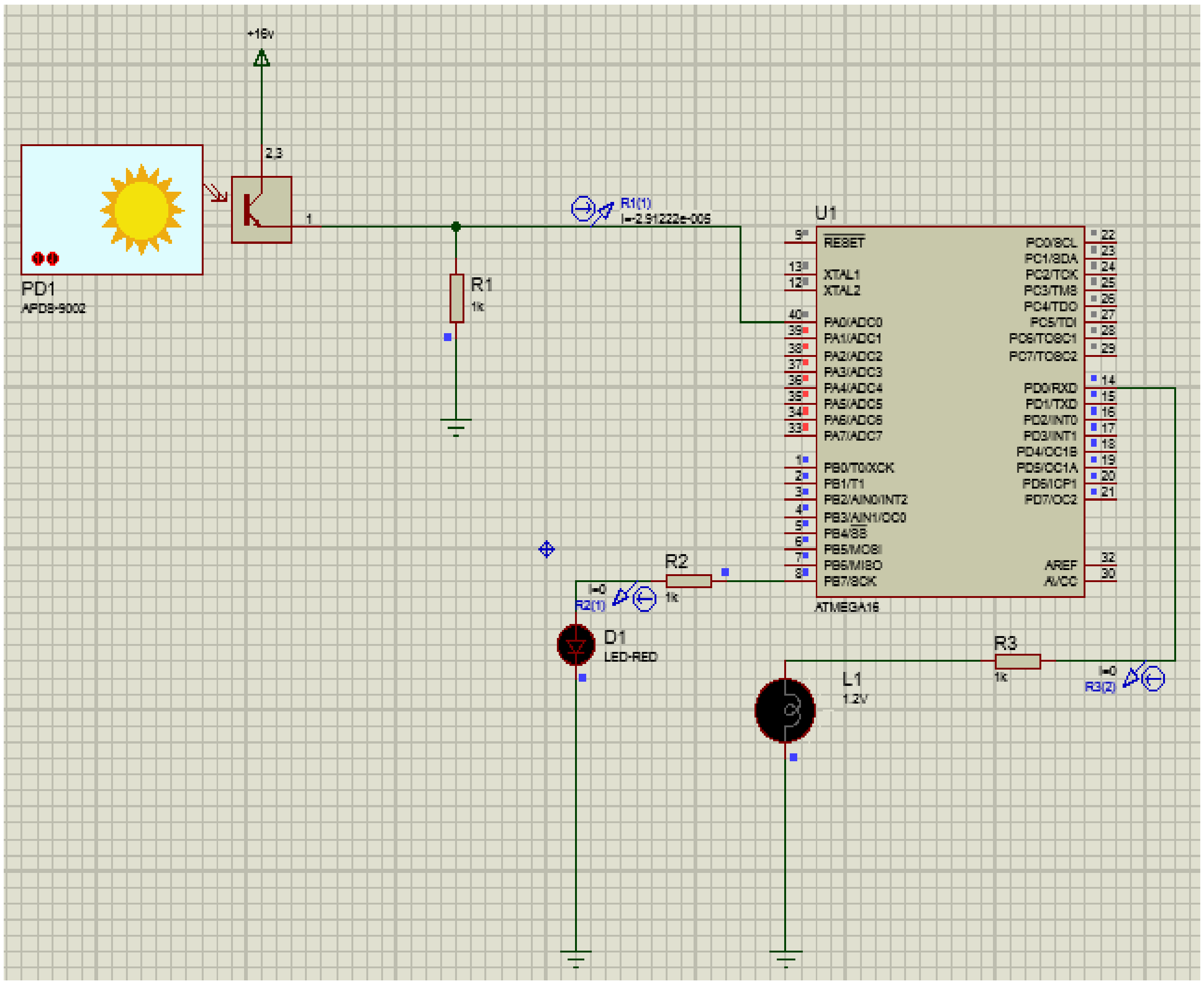


Figure 3. 2: Illuminated emission stage. In the presence of light, the probe indicates the main current of 29.1µA and that no current passes through either the LED or the bulb.

On the other hand, in the presence of darkness (identifiable by the presence of the moon and the stars at the level of the photodiode), radiation from the LED and the bulb is observed, producing modulated and supersaturing light respectively (Figure3.3).

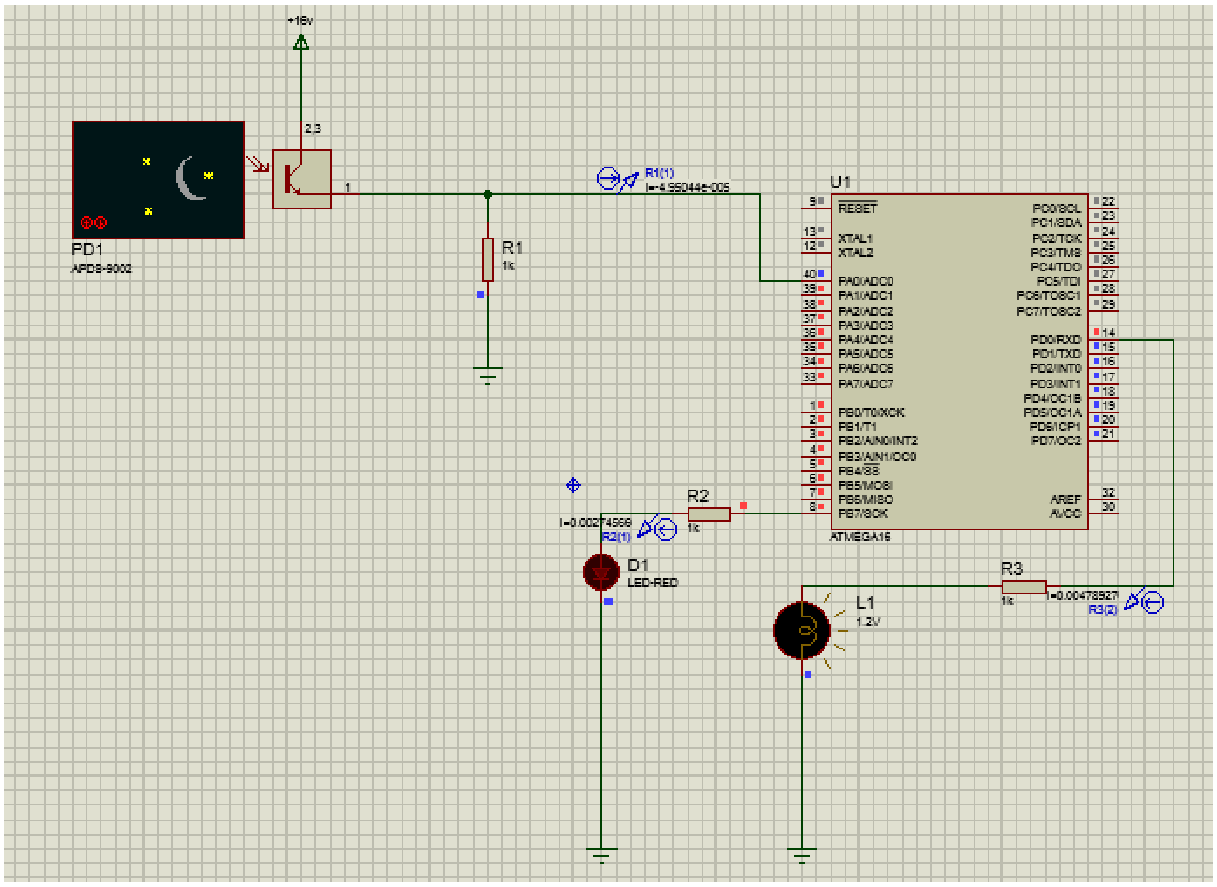


Figure 3. 3 : Emission stage in the dark. In the presence of darkness, radiation from LED and bulb is observed, producing modulated light and supersaturating light, respectively.

**b. Reception stage**
**a. Reception stage**

The role of the reception stage is to detect fluorescent radiation coming from a tomato leaf, convert it into an electrical signal and then into a Bluetooth format to be finally transmitted towards a smartphone (Figure 3.4).

Due to the fact that *Proteus* does not have a fluorescence module, we used a TORCH-LDR to emit and receive two light rays of different intensities. One of high intensity represents *Fm* fluorescence and the other of low intensity represents *Fo*.

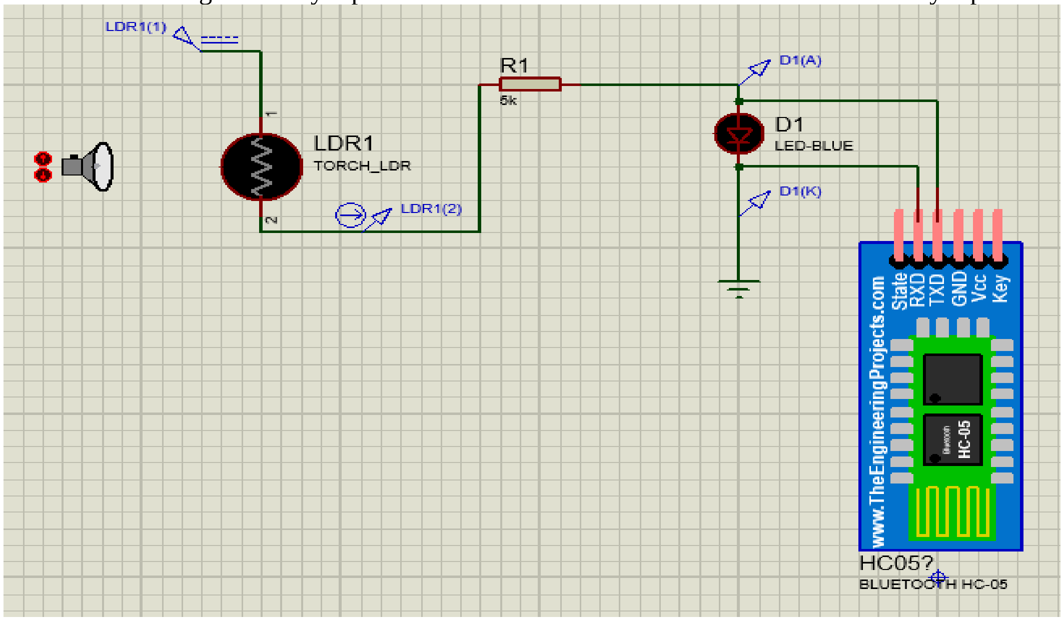


Figure 3. 4 : Reception stage

In addition, due to the fact that Proteus does not have according to our knowledge a module for converting electrical signals into Bluetooth data, we are contented to point out the difference in electrical state generated by the *Fm* and *Fo* fluorescences during the simulation of our sensor reception stage. In fact, for *Fo* fluorescence, the electrical potential identified by the probe is 1.72V and the blue LED (the indicator light) remains off (Figure 3.5).

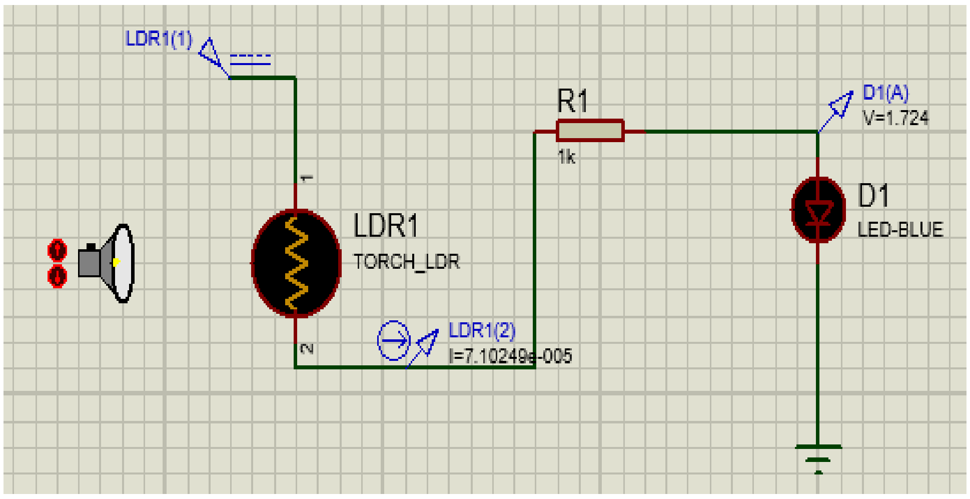


Figure 3. 5: Reception stage. Indeed for *Fo* fluorescence, the electrical potential identified by the probe is 1.72V and the blue LED (the indicator light) remains off.

On the other hand, for *Fm* fluorescence, the electrical potential identified by the probe is 2.02V and the blue LED lights up (Figure 3.6). The signals of 1.72V and 2.02V will be converted and then transmitted as text messages via Bluetooth (Figure 3.4).

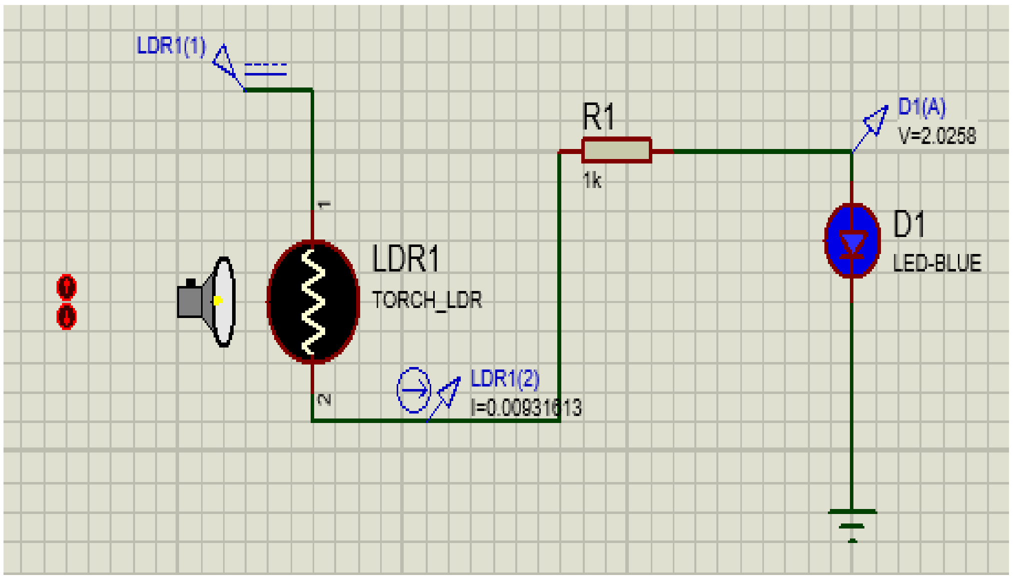


Figure 3. 6: Reception stage. For *Fm* fluorescence, the electrical potential identified by the probe is 2.02V and the indicator LED lights up.

Since most of the results concerning the designing of the sensor have been presented, we are going to do the same about the designing of the mobile application.

## 3.2 Android Application

The results concerning the application interface, UML diagrams and unitary test of the *Profil class* through the *ProfilTestjava class* are presented below.

### 3.2.1 Interface

The interface consists of a "BLUETOOTH" button used to initiate the Bluetooth search for discoverable devices including the sensor. The *"LIST"* button which displays the list of identified devices. The *"DURATION"* text box inside which we enter the length of time we would like the evaluation to face. A *"CONNECTION STATE"* display area whose role is to inform the user about the Bluetooth connection status. We also have two radio buttons such as "*Standard Assessment*" and "*Comparative Assessment*" required to specify the type of assessment requested by the user. To activate the process of transferring *Fo* and *Fm* signals from the sensor to the application (the successive values of *Fo* and *Fm* will be displayed in the *"Fluorescence"* display area), and to calculate bio-indicator parameters of plant health state (power spectral density, fluctuation, average value, and maximum value of photosynthetic efficiency), the user should click on the button "*EVALUATE THE HEALTH STATE OF YOUR PLANT"*

Once the calculations are completed, a message revealing the health state will be displayed in the *"Physiological state*" area. The health state of the plant being known, the user can write a message and send it using the text box *"Message*" and the button "*Send*" to a third person whose phone address is on the list of identified devices (Figure 3.7).

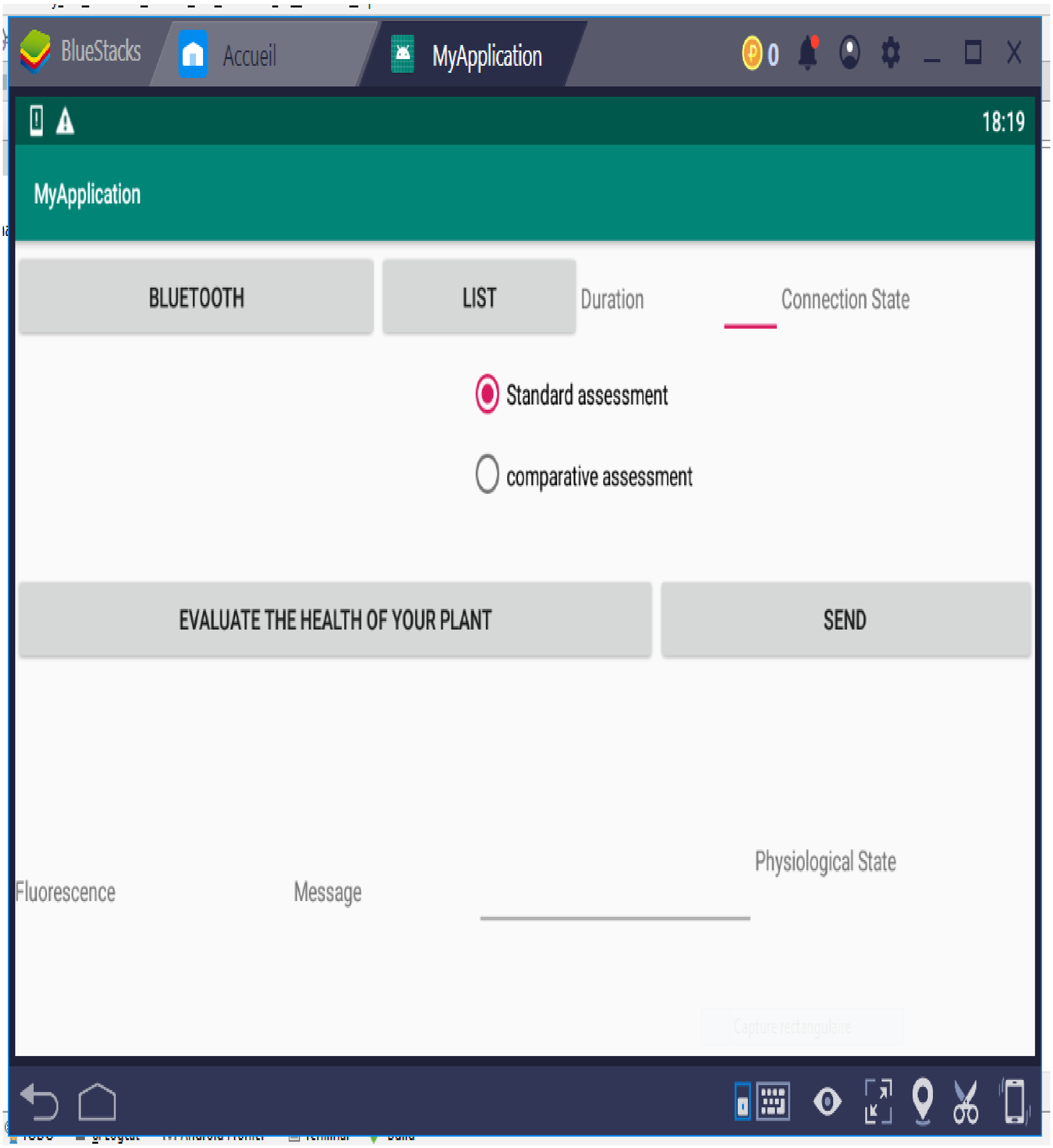


Figure 3. 7: Application's interface

### 3.2.2 UML Diagrams

#### a. Use case diagram

The diagram shows two use cases, seven internal use cases, a primary actor and a secondary actor. Two associations, eight inclusions and one extension relationship (Figure 3.8). When the case "*Evaluate health*" is requested to obtain the health state of plant, the cases *"*Activate Bluetooth", "*Select address*", *"Enter duration*", *"Choose type of assessment*" and *"Fluorescence values*", (provided by the secondary actor, which is a sensor) are also requested. Likewise, if the "Send message" case is requested, the *"Activate Bluetooth*", and "*Select address*", cases are also requested, while the *"Read message*" case is optional.

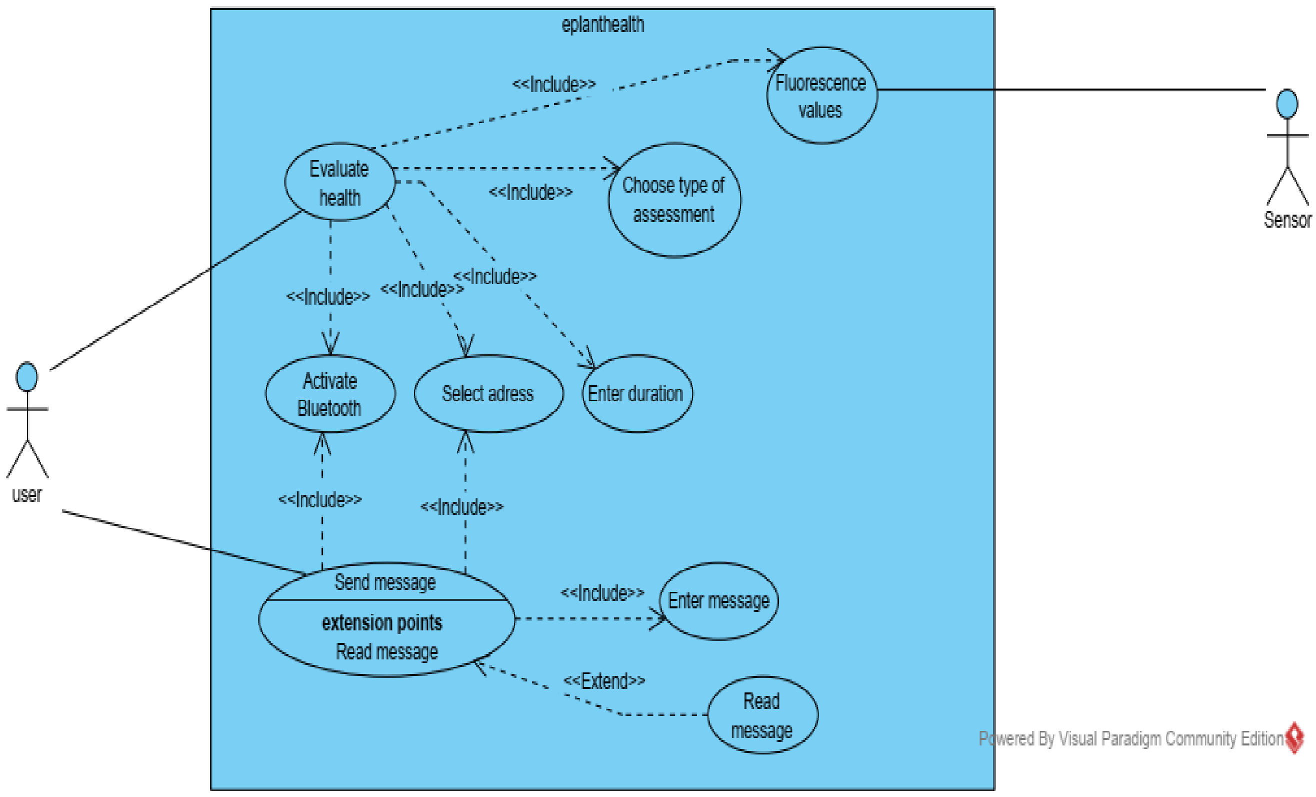


Figure 3. 8: Use case diagram

#### b. Activity diagram

Our activity diagram is made up of three partitions representing respectively the user *(user)* the application *(eplanthealth)* and the sensor (*sensor)* (Figure 3.9). It also includes thirteen actions, two decisions, one initial and one end activity nodes.

The three actions performed by the application are: searched for Bluetooth devices *"Bluetooth search*", establish connection with the sensor *"Establish connection*", receive data and calculate parameters "*Parameter calculation*".

The action of the sensor is to transmit data to the application only in the absence of any light. The nine user actions are represented in the "user" part.

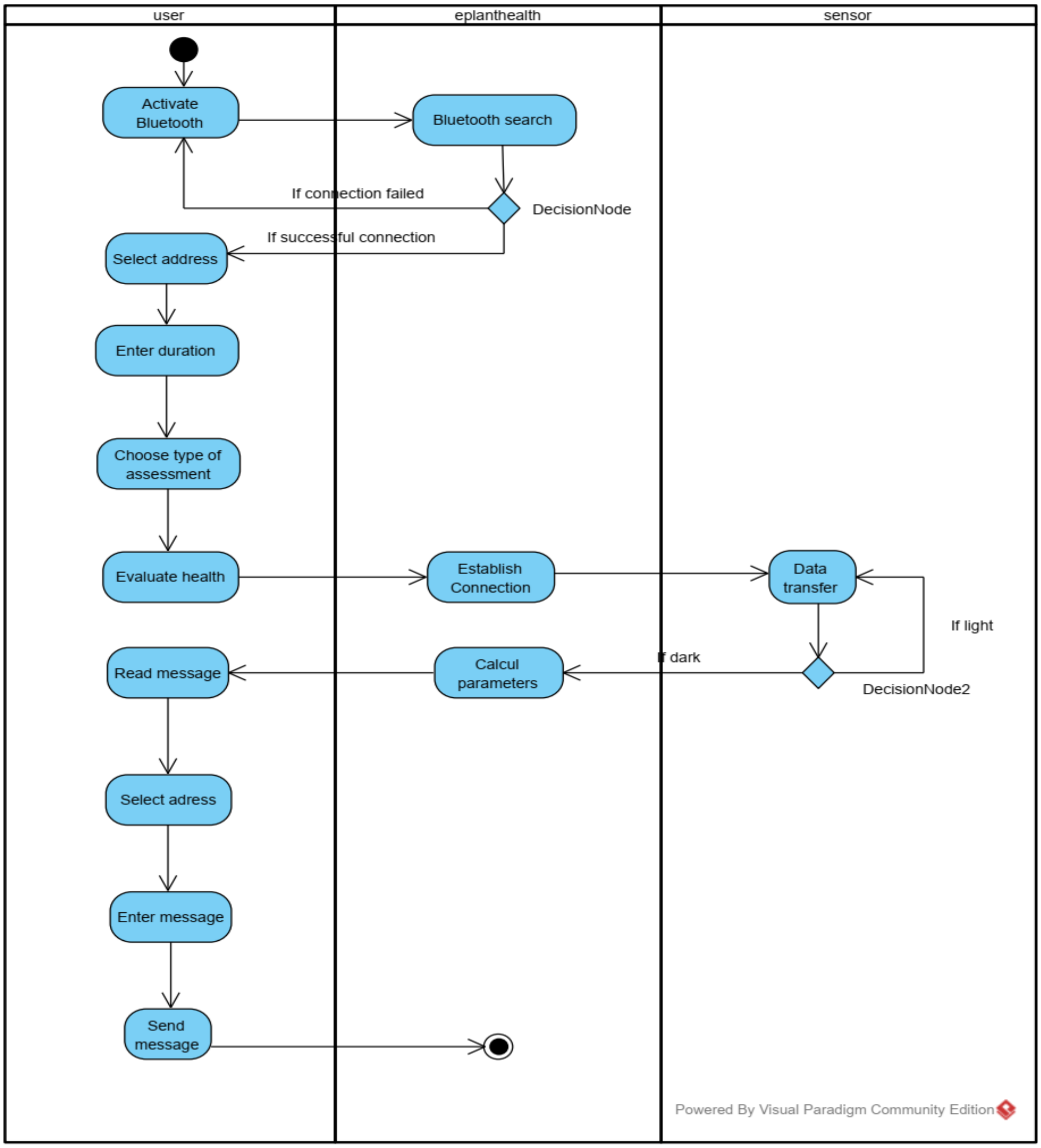


Figure 3.9: Activity diagram

### b. Sequence diagram

Our sequence diagram is made up of three lifelines, each representing the *"user", ": application"* and *"sensor"* objects (Figure 3.10). Ten synchronous messages, two response messages and two "alternative" interaction fragments are identified.

First the user activates Bluetooth, a message providing information about the connection status "*connection in progress, connected or connection failed"* is displayed by the application. If the connection is successful, the user selects the address of the sensor, enters the evaluation time, chooses the type of evaluation and clicks on "*evaluate*". A connection is established between the application and the sensor. If the initiation of connection to the sensor took place during the day, the sensor will wait until nightfall before taking the fluorescence measurements and transferring them via Bluetooth to the application. Once the data transmission time (duration of evaluations) is reached, the application starts to calculate of the bio-indicator parameters and displays a health message "*your plant is in good health*" or "*your plant would undergo biotic or abiotic stress*". The user can then

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 01, January-2021

select a new address, enter a message and send it (to ask for help or to inform a third party about the physiological state of his plant).

**a. Class diagram**

Our class diagram has eight classes, four inheritances and three aggregation relations (Figure 3.11).

The *"Profil"* class is the class where all the parameters necessary to deduce the physiological state of the plant are calculated. Its constructor takes as a parameter the type of evaluation, the duration of the evaluation and a table containing the successive values of the fluorescence's *Fm* and *Fo*. The "Control" class allows the management of data coming from the *"Profil"* class to the "*MainActivity*" class and vice versa.

The *MainActivity* class allows to retrieve all informations entered at the interface level and the triggered activities when any button is clicked. It allows the retrieval of this information by the *"Profil"* class via the *"Control"* class. The *"MainActivity"* class inherits from the "*AppCompatActivity"* class and has as subclass *"ServerClass"*, *"ClientClass"*, and "*SendReiceive*" classes. These three subclasses each inherits from the *"Thread" class* which gives them the ability to be executed in different threads from where *"MainActivity"* class is executed.

The "*ServerClass*" class allows the application to function as a server when sending messages and "ClientClass" class allows the application to function as a client when receiving fluorescence data from the sensor. The *"SendReiceive"* class allows receiving data streams from the sensor and sending messages from the application to a specific recipient.

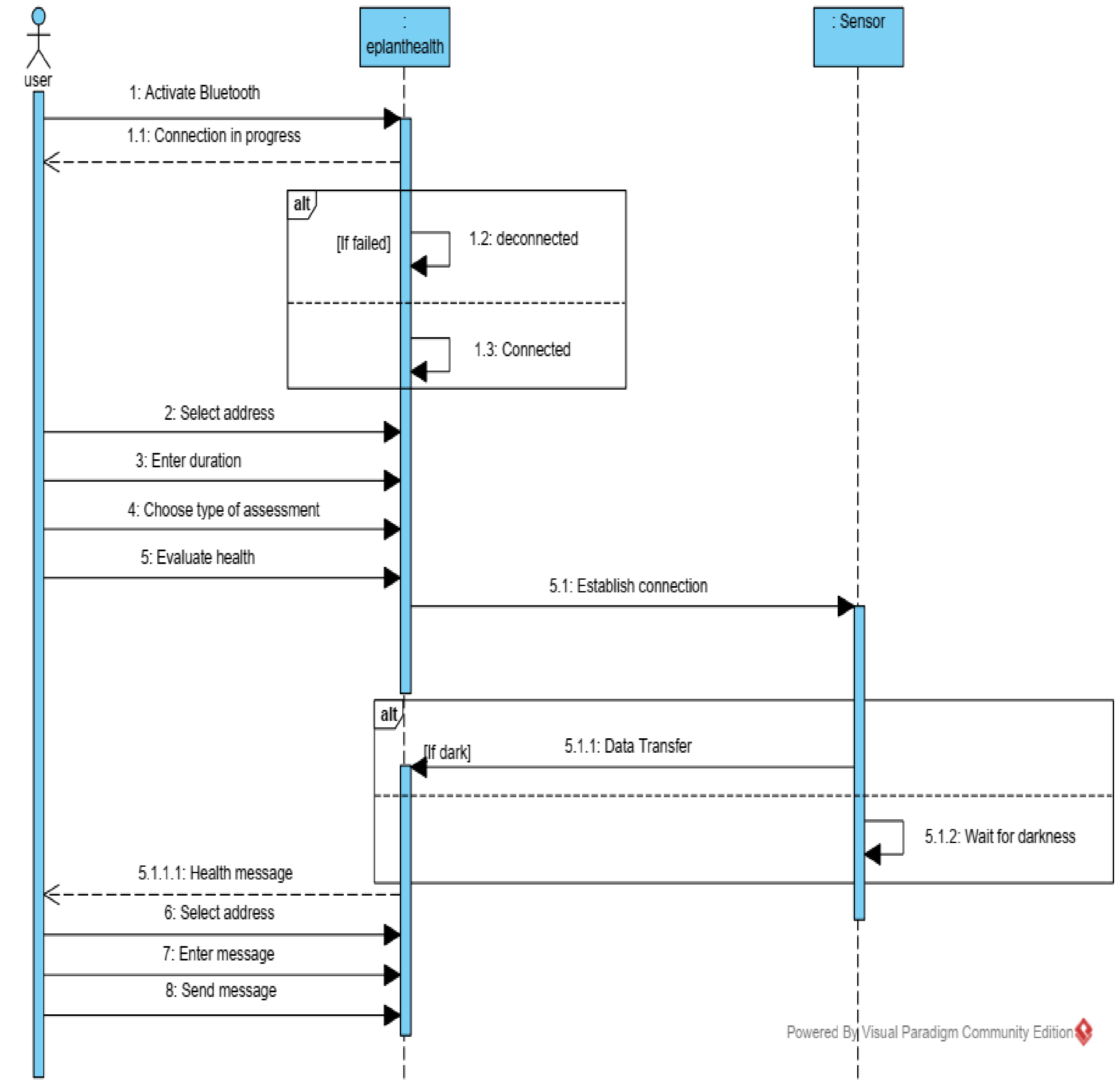


Figure 3. 10: Sequence diagram

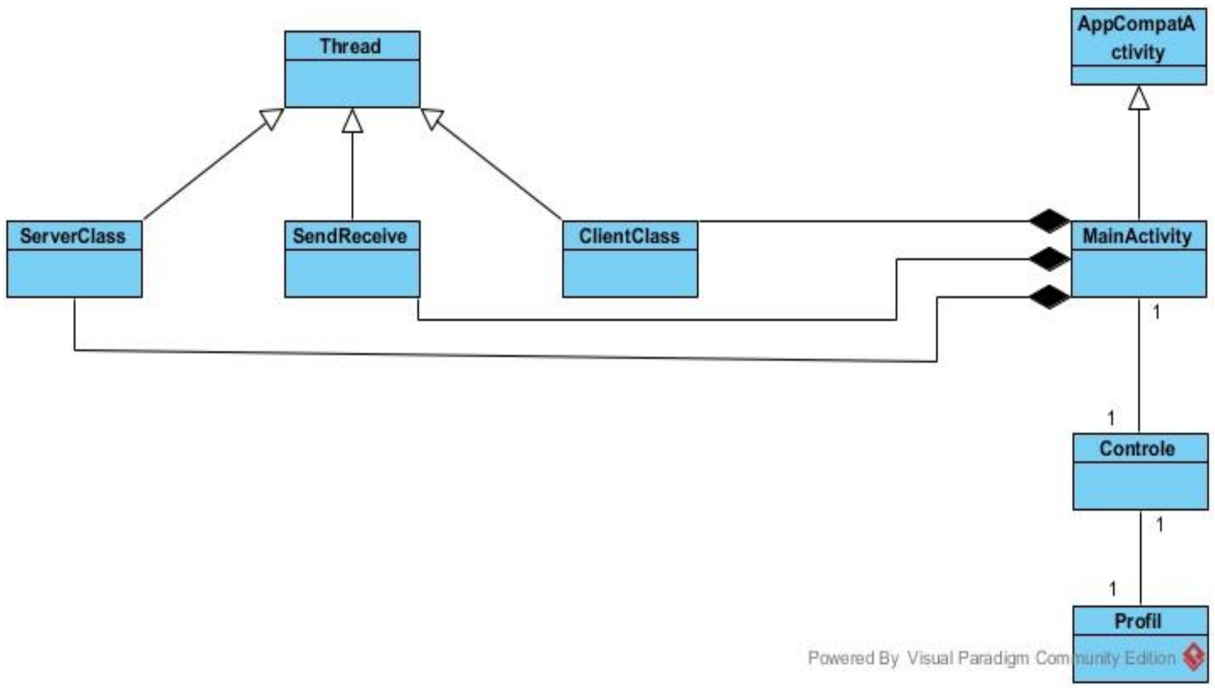


Figure 3.11: Class diagram

### 3.2.3 Unit test of the *"Profil"* class

#### a. Unit test for a successful standard assessment

The unit test is carried out in the case of a user chooses to enter an evaluation period equal to "six" and ticks on “*Standard assessment”* button (Figure 3.12). Considering that the table of fluorescence values received by the application is

*tabValRecu [] = {13, 10, 6, 3, 12, 9};*

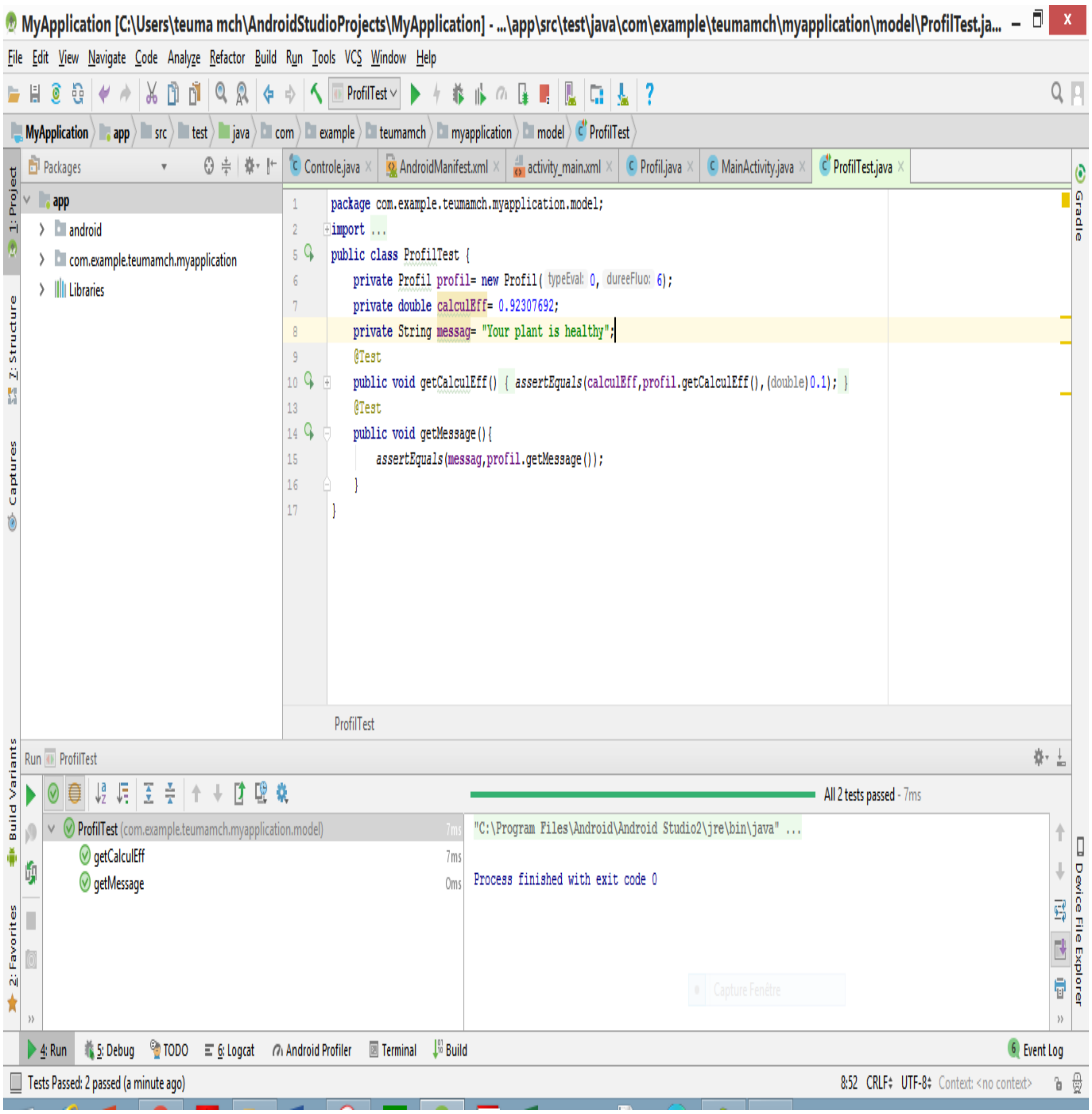
The expected and obtained calculation of the photosynthetic efficiency is

*Private double calculationEff = 0.32;*

And the expected and obtained message is

*Private String Massage= “Your plant is healthy*”

The console tells us that two tests have been carried out and these two tests have passed (All 2 tests passed). This success is also seen by green writing color, of “*getCalculEff ()”* and “*getMessage*” methods.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication app src test java com example teumamch myapplication model ProfilTest
Packages
app
android
com.example.teumamch.myapplication
Libraries
Controle.java AndroidManifest.xml activity_main.xml Profil.java MainActivity.java ProfilTest.java
4: Run 5: Debug TODO 6: Logcat Android Profiler Terminal Build Event Log
8:52 CRLF UTF-8 Context: <no context>

MyApplication [C:\Users\teuma mch\AndroidStudioProjects\MyApplication] - ...\app\src\test\java\com\example\teumamch\myapplication\model\ProfilTest.ja...

```java
package com.example.teumamch.myapplication.model;
import ...
public class ProfilTest {
    private Profil profil= new Profil( typeEval: 0, dureeFluo: 6);
    private double calculEff= 0.92307692;
    private String messag= "Your plant is healthy";
    @Test
    public void getCalculEff() { assertEquals(calculEff,profil.getCalculEff(),(double)0.1); }
    @Test
    public void getMessage(){
        assertEquals(messag,profil.getMessage());
    }
}
```

```
Run  ProfilTest
All 2 tests passed - 7ms
ProfilTest (com.example.teumamch.myapplication.model)   7ms
    getCalculEff   7ms
    getMessage     0ms

"C:\Program Files\Android\Android Studio2\jre\bin\java" ...

Process finished with exit code 0

Tests Passed: 2 passed (a minute ago)
```

Figure 3.12: Unit test for a successful standard assessment

**b. Unit test for a failed standard assessment.**

When we change the expected message to,

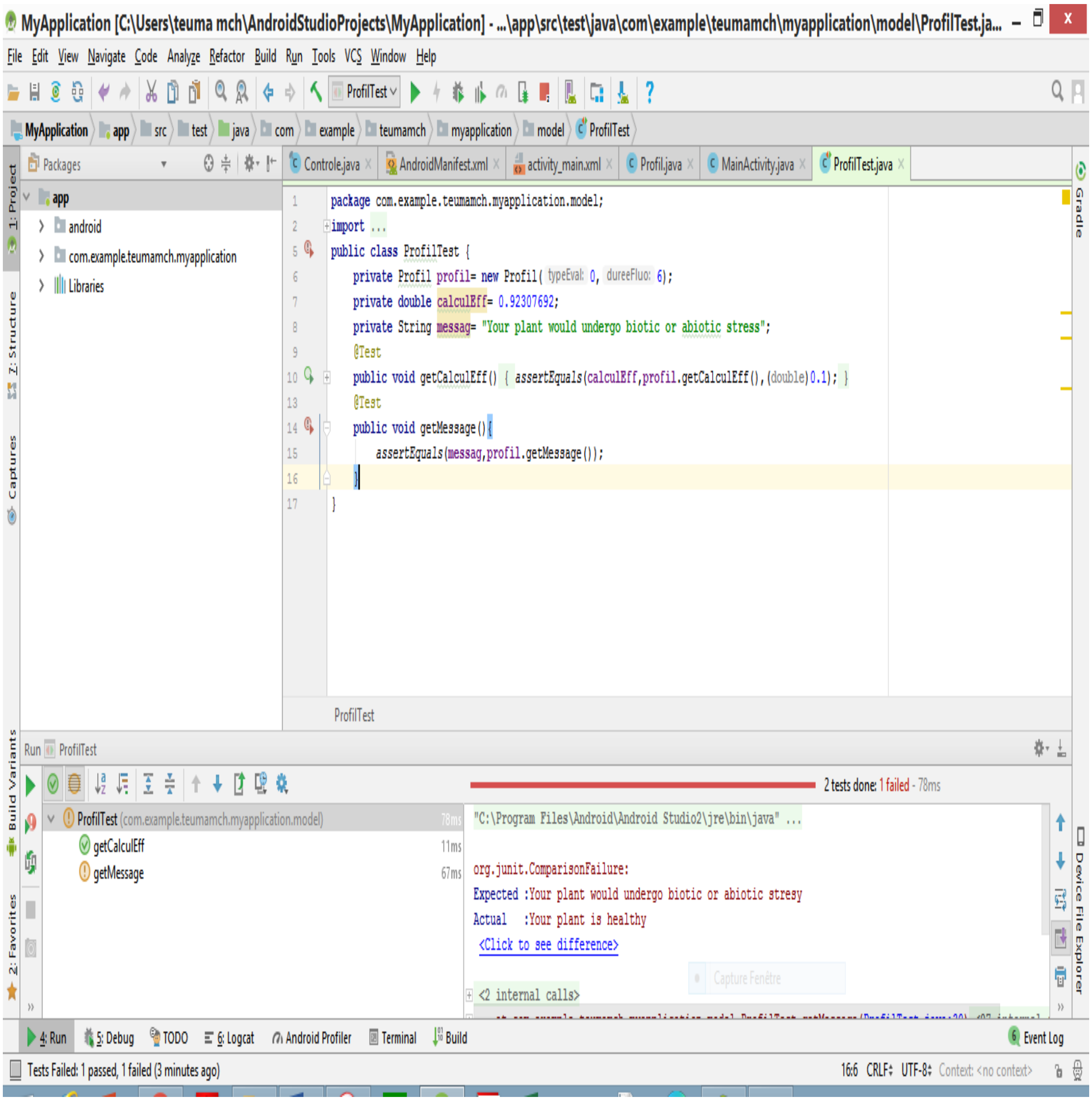
*Private String Massage= "Your plant would undergo biotic or abiotic stress";*

The console tells us that two tests have been carried out and one test has failed (2 tests done 1 failed). These results can also be seen by writing the *"getCalculEff ()"* method in green colour and the *"getMessage*" method in yellow (Figure 3.13). In the center of the console this failure is reflected by the messages,

*Expected = "Your plant would undergo biotic or abiotic stress"*
And

*Actual = "Your plant is healthy"*

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 01, January-2021



```
package com.example.teumamch.myapplication.model;
import ...
public class ProfilTest {
    private Profil profil= new Profil( typeEval: 0, dureeFluo: 6);
    private double calculEff= 0.92307692;
    private String messag= "Your plant would undergo biotic or abiotic stress";
    @Test
    public void getCalculEff() { assertEquals(calculEff,profil.getCalculEff(),(double)0.1); }
    @Test
    public void getMessage(){
        assertEquals(messag,profil.getMessage());
    }
}
```

```
2 tests done: 1 failed - 78ms

ProfilTest (com.example.teumamch.myapplication.model)   78ms
    getCalculEff   11ms
    getMessage     67ms

"C:\Program Files\Android\Android Studio2\jre\bin\java" ...

org.junit.ComparisonFailure:
Expected :Your plant would undergo biotic or abiotic stresy
Actual   :Your plant is healthy
 <Click to see difference>

<2 internal calls>
```

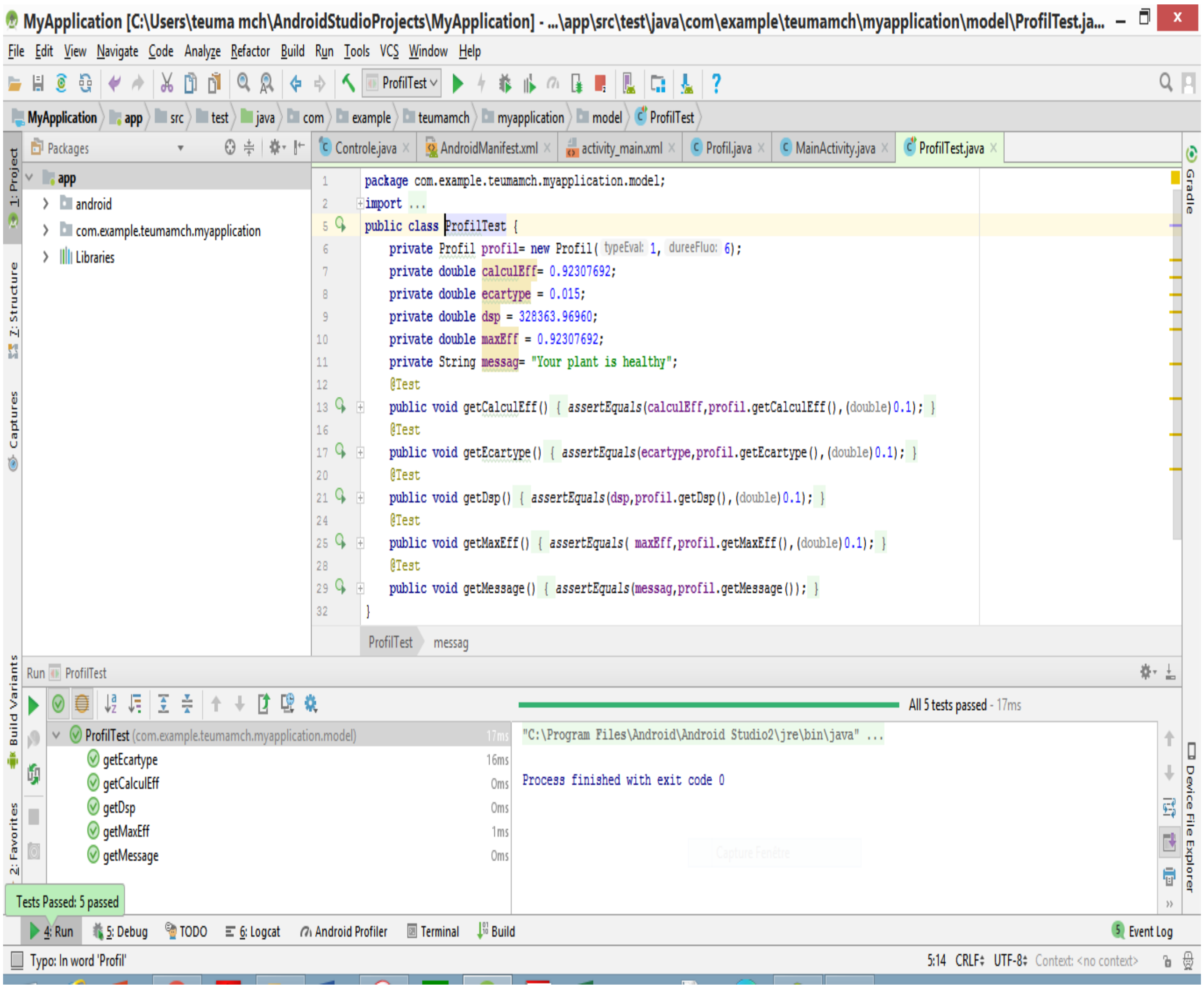
```
Tests Failed: 1 passed, 1 failed (3 minutes ago)
```

Figure 3.13: Unit test for a failed standard assessment

### c. Unit tests for successful and failed comparative evaluation

Unit tests performed when user ticks the "*Comparative assessment"* button are present on Figures 3.14 and 3.15. For a "*comparative*" evaluation type, four methods are used namely: "*getCalculEff ()*" to retrieve calculation result of the average photochemical efficiency (which is equal to: 0.32), *"getEcartype ()"* to retrieve calculation result of the standard deviation (which worth: 0.015), *"getDsp ()"* to recover calculation result of the DSP (which has the value: 34699.032), "*getMaxEff ()"* to recover the maximum photochemical efficiency (equal to: 0.5) and *"getMessage ()"* to retrieve the message corresponding to the physiological state of the plant (which corresponds to: Your plant would undergo biotic or abiotic stress). The console shows us (Figure 3.14) that five tests have been performed and all have passed. On the other hand, by changing the expected message (Expected) the console shows an error (Figure 3.15).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 01, January-2021

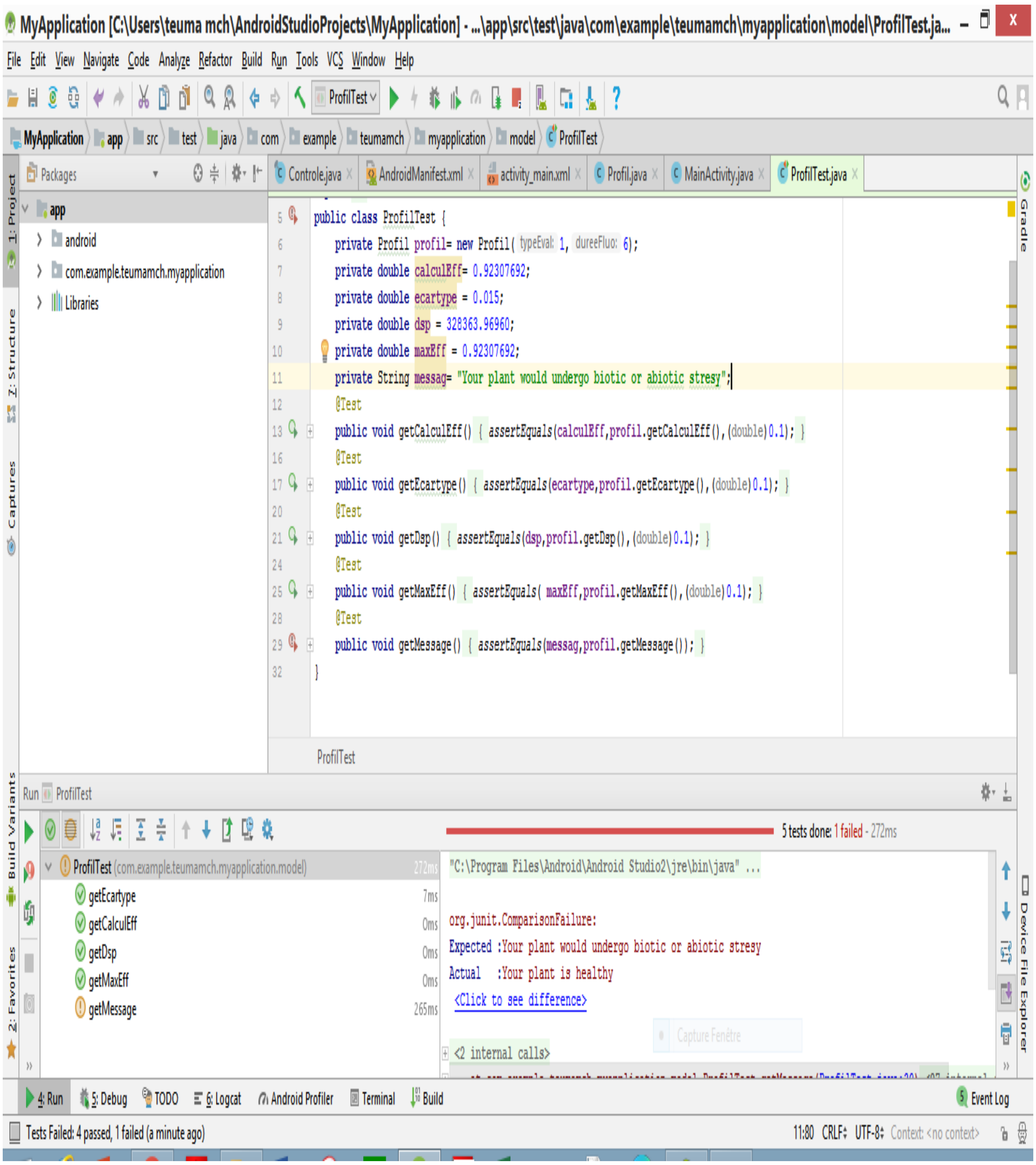


MyApplication [C:\Users\teuma mch\AndroidStudioProjects\MyApplication] - ...\app\src\test\java\com\example\teumamch\myapplication\model\ProfilTest.ja...

```
package com.example.teumamch.myapplication.model;
import ...
public class ProfilTest {
    private Profil profil= new Profil( typeEval: 1, dureeFluo: 6);
    private double calculEff= 0.92307692;
    private double ecartype = 0.015;
    private double dsp = 328363.96960;
    private double maxEff = 0.92307692;
    private String messag= "Your plant is healthy";
    @Test
    public void getCalculEff() { assertEquals(calculEff,profil.getCalculEff(),(double)0.1); }
    @Test
    public void getEcartype() { assertEquals(ecartype,profil.getEcartype(),(double)0.1); }
    @Test
    public void getDsp() { assertEquals(dsp,profil.getDsp(),(double)0.1); }
    @Test
    public void getMaxEff() { assertEquals( maxEff,profil.getMaxEff(),(double)0.1); }
    @Test
    public void getMessage() { assertEquals(messag,profil.getMessage()); }
}
```

```
All 5 tests passed - 17ms

ProfilTest (com.example.teumamch.myapplication.model)  17ms
    getEcartype   16ms
    getCalculEff   0ms
    getDsp         0ms
    getMaxEff      1ms
    getMessage     0ms

"C:\Program Files\Android\Android Studio2\jre\bin\java" ...

Process finished with exit code 0

Tests Passed: 5 passed
```

Figure 3.14: Unit test for a successful comparative assessment



```
public class ProfilTest {
    private Profil profil= new Profil( typeEval: 1, dureeFluo: 6);
    private double calculEff= 0.92307692;
    private double ecartype = 0.015;
    private double dsp = 328363.96960;
    private double maxEff = 0.92307692;
    private String messag= "Your plant would undergo biotic or abiotic stresy";
    @Test
    public void getCalculEff() { assertEquals(calculEff,profil.getCalculEff(),(double)0.1); }
    @Test
    public void getEcartype() { assertEquals(ecartype,profil.getEcartype(),(double)0.1); }
    @Test
    public void getDsp() { assertEquals(dsp,profil.getDsp(),(double)0.1); }
    @Test
    public void getMaxEff() { assertEquals( maxEff,profil.getMaxEff(),(double)0.1); }
    @Test
    public void getMessage() { assertEquals(messag,profil.getMessage()); }
}
```

```
5 tests done: 1 failed - 272ms

ProfilTest (com.example.teumamch.myapplication.model)   272ms
    getEcartype     7ms
    getCalculEff    0ms
    getDsp          0ms
    getMaxEff       0ms
    getMessage      265ms

"C:\Program Files\Android\Android Studio2\jre\bin\java" ...

org.junit.ComparisonFailure:
Expected :Your plant would undergo biotic or abiotic stresy
Actual   :Your plant is healthy
 <Click to see difference>

<2 internal calls>

Tests Failed: 4 passed, 1 failed (a minute ago)
```

MyApplication [C:\Users\teuma mch\AndroidStudioProjects\MyApplication] - ...\app\src\test\java\com\example\teumamch\myapplication\model\ProfilTest.ja...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication app src test java com example teumamch myapplication model ProfilTest
Controle.java AndroidManifest.xml activity_main.xml Profil.java MainActivity.java ProfilTest.java
Packages
app
android
com.example.teumamch.myapplication
Libraries
Run ProfilTest
4: Run 5: Debug TODO 6: Logcat Android Profiler Terminal Build Event Log
11:80 CRLF UTF-8 Context: <no context>

Figure 3.15: Unit test for a failed comparative assessment

Our sensor is very flexible due to the fact that, it is not made of any pre-fabricated elements as it is the case of many fluorescence sensors [8]. In addition to this flexibility, its realization does not require a huge financial investment because, it does not use a narrow band filter [8, 9]. This filter is typically used to reduce the effect of light interference and to allow only light of a specific wavelength to pass through. To solve this interference problem, our sensor is triggered automatically in the absence of any environmental light.

"IOT" sensors such as *Flower Power* and *Edyn* [10, 11], found in the market are limited only to the detection of abiotic stresses, so there are not able to identify malignancies due to the presence of the "*infesting phytophthora"* on a tomato plant. The designers of these devices could well add to their multifunctional sensors a fluorescence module consisting of two stages like our sensor. However systems such as *Expert System* and *Deep convolutional neural networks* systems can detect the effect of "*infesting phytophthora"* on a tomato plant. But the weakness of these systems is that they are based on image detections; this means that, plants leafs already present visible symptoms such depigmentation.

In addition providing plant physiological state information such as *Flower Power* and *Edyn*, our application can use its *"ServerClass"* class send a specific message (to request help for example). In addition, it is possible to follow the health state evolution of several plants one by one by simply selecting the address of the corresponding sensors.

In the *"standard assessment*" unit test, after receiving the fluorescence values in the table *"tabValRecu"*, the evaluation result is *"your plant would undergo biotic or abiotic stress*" is justified by the fact that, average photochemical efficiency is lower than the standard average photochemical efficiency of a healthy plant [11,12] (Figure 3.12). So it is obvious that, if the expected message is "*your plant is healthy*" then the compiler displays a "failed test" in the console (Figure 3.13).

The results of Figures 3.14 and 3.15 are explained by the fact that, concerning the "*comparative assessment*", a plant is declared to be in good health if in addition to the fact that its average photochemical efficiency is greater or equal than the healthy plant one, it is also necessary that its fluctuation (standard deviation), DSP, and maximum photochemical efficiency, are respectively greater or equal than average photochemical efficiency, fluctuation, DSP and maximum photochemical efficiency of healthy plants; otherwise we may deal with a plant which would undergo biotic or abiotic stress [13,14].

The simulation of the integral sensor-application system require the realization and the characterization of the sensor, because for the moment there is no simulation software integrating both Isis Proteus and android studio functionalities to simulate simultaneously our whole system. However, concerning sensor, its manufacture should not cause any problem since the circuit is available (Figure 2.1 and 2.4). All we have to do is to replace the Torch-LDR with a real plant sheet and insert a translator between the LED (blue LED) and the Bluetooth module.

## 4. CONCLUSION

Our aims was to design a system able of identifying a diseased tomato plant before the first symptoms appear. To do this, we designed a system consisting of a sensor and an android mobile application. The sensor is made of two stages, a transmission stage and a reception stage. The mobile application, to function properly, required the use of eight UML classes. The code of each of those classes was written, the mobile application interface was presented and unitary test of our model (*Profil Class*) were perform. It comes out that our device will provide information on the plant physiological state like *Flower Power*, *Edyn*, *Expert System* and *Deep convolutional neural networks* systems. However our system will be more flexible, less expensive and will able to identify all types of stresses, whether they are biotic or abiotic origin before the first appearance of symptoms. This is due to the fact that, contrary to other systems our plant disease detection is based on the measure of photochemical efficiency, not on leaf image. During the second phase of our project, we will optimize and manufacture our system using transimpedance amplifier and raspberry pi 3 microcomputer. This optimization will aim to reduce energy consumption, to use WIFI and the 4G network of a mobile phone operator.

## ACKNOWLEDGEMENTS

We would like to express a special thanks Nkolo Laure-Martine for her assistance

## REFERENCES

[1] Philippe Lepoivre. Phytopathologie ; Bases moléculaires et biologiques des pathosystèmes et fondements des stratégies de lutte. 2003; Editions De Boeck Université.
[2] Francesco Paolo Camarda. Parrot flower power review 24 February, 2017
[3] Edyn.Welcometotheconnectedgarden. https :// www.kickstarter.com/projet. November 2018.
[4] Abu-Naser S.S, Kashkash K.A., Fayyad M. Developing an Expert System for Plant Disease Diagnosis. Journal of Artificial Intelligence. 1(2):78-85, 2008.
[5] Srdjan S., Marko A., Andras A., Dubravko C., Darko S. Deep Neural Networks Based Recognition of plant Diseases by Leaf Image Classification. Computational Intelligence and Neuroscience. http://dx.doi.org/10.1155/2016/3289801.
[6] Stress Testing. Optisci.com.Retrieved 2011-03-28
[7] Laurent Audibert. UML 2.0. IUT de Villetaneuse. Novembre 2018.
[8] Christopher G., José A.M.N., Eduardo G. P. A Low-Cost Chlorophyll Fluorescence Sensor System. Conference Paper. DOI:10.1109/SBESC.2016.036. November 2016. Novembre 2018.
[9] Zachary S. Low-cost fluorescence sensor for accurate detection of Ph value. Master of Science in Electrical Engineering. November 2018.
[10] Parrot Flower Power review. Can technology give you a green thumb? November 2018.
[11] Bourrié B. La fluorescence chlorophyllienne comme outil de diagnostic. 8$^{eme}$ journées de fertilization raisonnée et de l'analyse de terre. GEMAS-COMIFER Fertilisation raisonnée et de l'analyse de terre: quoi de neuf en 2007.
[12] Hitmi A., Moussard C., Austruy A., Verney P., Fonctionnement des systèmes photosynthétiques. ADEME. www.ademe.fr
[13] Teuma M.M., Ekobena F.H.P., Ambang Z., Tabi C.B., Kofane T.C. Study of the passive electrical properties of tomato tissues after infection and treatment by fungicide. Indian Journal of Science and technology 2017 10(26).
[14] Teuma M.M., Ambang Z., Ekobena F.H.P., Kofane T.C. Electrical conductance analysis of Solanum lycopersicum under biotic stress Transaction on Machine learning and Artificial Intelligence Volume 8 No 2. 2020. DOI: 10.14738/tmlai.82.6799