A Deliverable 4 Report on

Vertical Farm Control System

**CS3500**

**Software Engineering**

Computer Science

2022-2023

Submitted by:

**Group 10**

Jakab-Gyik Sarolta - 122118473

Smith Deirbhle - 120338191

Varga Zoltán - 122118466

Veres Noémi - 122118476

Text

Description automatically generated

**University College Cork**

Ireland

Table of contents

[Abstract 5](#_Toc121156746)

[1. Introduction 6](#_Toc121156747)

[1.1 Background 6](#_Toc121156748)

[1.2 Different Possible Approaches 6](#_Toc121156749)

[1.3 Pros and cons of vertical farming 7](#_Toc121156750)

[2. Requirements 7](#_Toc121156751)

[2.1 Non-Functional Requirements 7](#_Toc121156752)

[2.2 Functional Requirements 8](#_Toc121156753)

[3. Design Diagrams 11](#_Toc121156754)

[3.1 Use-Case Diagram 11](#_Toc121156755)

[3.2 Control System Diagram 14](#_Toc121156756)

[3.3 Finite State Machine 15](#_Toc121156757)

[3.4 Data flow 16](#_Toc121156758)

[4. UML Diagrams 19](#_Toc121156759)

[4.1 UML Package Diagram 19](#_Toc121156760)

[4.2 UML Class Diagrams 20](#_Toc121156761)

[5. Final Version of the Application 25](#_Toc121156762)

[5.1 Setup 25](#_Toc121156763)

[5.2 Description of Application 25](#_Toc121156764)

[5.3 Description of Packages 29](#_Toc121156765)

[5.4 Description of Classes 30](#_Toc121156766)

[6. Testing 32](#_Toc121156767)

[6.1 White-box Testing 32](#_Toc121156768)

[6.1.1 TemperatureControl Package Tests 32](#_Toc121156769)

[6.1.2 InputParameters Package Tests 32](#_Toc121156770)

[6.1.3 EnvironmentSimulator Package Tests 34](#_Toc121156771)

[6.2 Black-Box Testing 34](#_Toc121156772)

[6.2.1 BlackBox Package Tests 34](#_Toc121156773)

[7. Conclusions 35](#_Toc121156774)

[8. References 37](#_Toc121156775)

List of Figures and Tables

[Figure 1 Hydroponics in vertical farming systems 6](#_Toc121156776)

[Figure 2 Use-case diagram of the Vertical Farm Control System 11](#_Toc121156777)

[Figure 3 Sequence diagram for use case 1 12](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156778)

[Figure 4 Sequence diagram for use case 2 13](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156779)

[Figure 5 Sequence diagram for use case 3 13](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156780)

[Figure 6 Sequence diagram for use case 4 14](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156781)

[Figure 7 Sequence diagram for use case 5 14](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156782)

[Figure 8 Feedback Control System Diagram 15](#_Toc121156783)

[Figure 9 Finite state machine diagram 16](#_Toc121156784)

[Figure 10 Level 0 (Context) data flow diagram 16](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156785)

[Figure 11 Level 1 data flow diagram 17](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156786)

[Figure 12 Level 2 data flow diagram 18](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156787)

[Figure 13 Package Diagram 19](#_Toc121156788)

[Figure 14 Classes of the JDBC package 20](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156789)

[Figure 15 Classes of the EnvironmentControllers Package 20](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156790)

[Figure 16 Classes of the InputParameters Package 21](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156791)

[Figure 17 Classes of the EnvironmentSimulator Package 22](#_Toc121156792)

[Figure 18 Classes for the Utils Package 23](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156793)

[Figure 19 Classes for the SystemConfiguration Package 23](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156794)

[Figure 20 Classes for the Reporting Package 24](file:///D:\UTCN\AN3\SEM1%20-%20Erasmus\Software%20Engineering%20CS3500\Lab\Software-Engineering-Project\Software-Engineering-Project\Team10_D4.docx#_Toc121156795)

[Figure 21 Administrator control panel 26](#_Toc121156796)

[Figure 22 Monitor System Panel 26](#_Toc121156797)

[Figure 23 Control Simulation Window 27](#_Toc121156798)

[Figure 24 Setting the Parameters 27](#_Toc121156799)

[Figure 25 Generate Report Window 28](#_Toc121156800)

[Figure 26 Invalid Input Alert 28](#_Toc121156801)

[Figure 27 Example of a report 29](#_Toc121156802)

[Figure 28 Flowchart for a test case 33](#_Toc121156803)

[Figure 29 Control Flow Graph for setActuatorCurrentStrength() method 34](#_Toc121156804)

|  |  |  |
| --- | --- | --- |
| ***Tasks of individual team members for D4*** | | |
| Jakab-Gyik Sarolta | GUI + documentation + test cases | Add new parameters to GUI + report GUI + description of application updated + test cases and description of them |
| Smith Deirbhle | Documentation + UML diagrams | UML package and class diagrams updated, description of classes and packages updated + conclusions from the project |
| Varga Zoltán | Backend + documentation | Adding new logic for the new parameters + debugging + documentation for setting up the application on any PC + Screen recording of the final application |
| Veres Noémi | GUI + test cases + documentation | GUI environment simulation updated with different windows + black-box and white-box test cases and description of them + Screen recording of the final application |

Abstract

Vertical farms are indoor farms that grow vegetables stacked on the vertical axis. In this manner, more crops can be cultivated on a smaller footprint than in traditional agriculture. The controlled environment offered by an indoor farm eliminates the risk of diseases and insects. Combined with hydroponics it increases 10 times the crop yield compared with traditional agriculture and reduces water consumption by 90% since it is recirculated in the system. Hydroponics means that the roots of the vegetables are placed in water enriched with nutrients instead of soil.

Our team has decided to create a control system for an indoor vertical farm that uses hydroponics. For simplicity, this farm grows butterhead lettuce exclusively but can be extended to manage the environment of other vegetables as well.

*Keywords*: agriculture, energy efficient, environmentally friendly, green farm

1. Introduction

1.1 Background

Some of the vital reasons why people started experimenting with vertical farming systems were connected to the exponential growth of the population and the inefficiency of traditional agricultural methods. Most places worldwide still use the same methods in growing crops as our ancestors did years ago. Which constitutes a major problem: these methods depend on various factors that the farmers cannot control. Insects, diseases, the nutrients in the soil, the general structure of the soil change from year to year, the limitation of space, and other issues.

To address them, vertical farming requires only a fraction of a farm’s space to grow just as many vegetables. The insecticides and the biochemical materials that farmers use could be avoided. Since they are also harmful to the human body, this is a considerable side effect of traditional agriculture. The consistency of the soil or water that the plants are growing in is controlled by machines, the sensors are measuring the nutrient level from time to time, ph level, air, and water temperature, and the salinity of the water which can be determined by the EC level (electrical conductivity). This way the water can provide the ideal environment for all kinds of plants, carefully determined for every species, what would be the range of the EC level that results in the fastest growth.

1.2 Different Possible Approaches

These systems, however, do not use soil to plant the seeds in. They use special growing systems, as mentioned above, which come in 6 different forms. (*seen in* Figure *1*)



Figure 1 Hydroponics in vertical farming systems

[https://www.hydroponicschina.com/types-of-hydroponic-system/]

1. ***Deep Water system:***the roots of the plants are floating in nutrient-dense water with oxygen. A container acts as a reservoir for this solution.
2. *Wick system:* does not consume electricity but it uses the wicks of the plants to provide nutrient-dense water for the plants growing in the soil. The least complex and most energy-efficient approach.
3. *Drip system:* the nutrient solution is dripped either to the roots of the plants or to the leaves, coming from above. It depends on the types of plants the farm wants to cultivate.
4. *Ebb and Flow system (also called flood and drain):* the roots of the plants are periodically flooded with nutrient-dense water.
5. *Aeroponics system:* the roots of the plants are sprinkled with the nutrient solution.
6. *Nutrient Film technique:* the system works with pipes that hold the nutrient solution for the plants, then the water is collected from them by letting it flow back to the reservoir from these pipes. The pipes are moving; the incline makes the water flow down and then a new amount is pumped into the pipe. The water is recycled this way; less water consumption.

In this control system, the first approach is taken since it does not require electricity and it is one of the simpler ways to realize the physical structure of the project.

1.3 Pros and cons of vertical farming

Among the **advantages** of vertical farming and hydroponics, we can mention the small footprint of the growing area, the reuse of water, and the elimination of pesticides and harmful mind substances. These all contribute to the growth of healthy and nutrient-rich crops. The production can be made carbon neutral by capturing rainwater and installing solar panels. Plants can be cultivated stably and predictably, isolated from the unforeseeable outside environment.

Vertical farms and hydroponic cultivation have **drawbacks** as well. Among the biggest disadvantages we can mention the limited variety of plants that can be grown in such an environment, and the natural processes, such as pollination, the system needs to replace. Artificial lighting can increase energy consumption to a great extent. Also, the initial phase of fine-tuning and high dependence on technology can carry possible vulnerabilities too.

2. Requirements

We detailed two types of requirements: non-functional requirements, and functional requirements.

2.1 Non-Functional Requirements

Non-functional requirements contain demands that concern the conceptual properties of a product. They do not say what to do, but what properties the system needs to have while doing the actions stated by functional requirements. The system should be:

1. *Intuitive and easy to use by the administrator:* The GUI of the application should not be a burden for the administrator to use. It should be clean and intuitive.
2. *Reliable results and reports:* The reports generated by the system at the request of the administrator should reflect reality. Therefore, they should be taken in real-time and displayed accurately
3. *Precise:* Sensors should be calibrated before use and maintained so. Unprecise data would have terrible effects on the environment and yield. The whole system relies on the data provided by the sensors.
4. *Deal with incorrect user inputs and display messages that can help the admin find the issue:* The application should always validate the input of the admin. The initial parameters provided are the base of the environment that will be maintained; therefore, no error should occur in this phase.
5. *Efficient*: The whole point of vertical farms and hydroponics is to be as efficient as possible with the available scarce resources. Thereby the control system should manage these resources as well as possible.
6. *Highly productive*: High efficiency is obtained not only by resource management but also maximizing the yield and the space used.
7. *Environmentally friendly*: Combined with the two requirements above, environmental friendliness is achieved. No waste and efficient use of resources contribute to this aspect.

2.2 Functional Requirements

Functional requirements say what the system should be able to do, but without mentioning how that should be achieved. They should be compliant with the SMART requirement’s properties; they should be specific, measurable, attainable, realistic, and traceable. Our functional requirements are presented below. Note that these requirements change concerning the given parameters of the system administrator.

The *[parameter\_name]* represents the corresponding system parameter. For example, the heating of air will be turned on when the temperature drops below the minimum temperature - a threshold. And it is continuing heating until the temperature gets above the minimum temperature.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **Until when to do** |
| Air temperature | **<** *[minimum\_air\_temperature]* | R1. Start heating the air | **>=** *([minimum\_air\_temperature] + [maxiumum\_air\_temperature])* / 2 |
| **>** *[maximum\_air\_temperature*] | R2. Start cooling the air | **<=** *([minimum\_air\_temperature] + [maxiumum\_air\_temperature])* / 2 |

1. The system will check the air temperature when it is in the balanced state every *[air\_temperature\_balanced\_check\_interval] minutes*.
2. The system will check the air temperature when it is in the balancing state every *[air\_temperature\_balancing\_check\_interval]* minutes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **Until when to do** |
| Water temperature | **<** *[minimum\_water\_temperature]* | R3. Start heating the water | **>=** *([minimum\_water\_temperature] + [maxiumum\_water\_temperature])* / 2 |
|  | **<** *[maximum\_water\_temperature]* | R4. Start cooling the water | **<=***([minimum\_water\_temperature] + [maxiumum\_water\_temperature])* / 2 |

1. The system will check the water temperature when it is in the balanced state every *[water\_temperature\_balanced\_check\_interval]* minutes.
2. The system will check the water temperature when it is in the balancing state every *[water\_temperature\_balancing\_check\_interval]* minutes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **Until when to do** |
| Humidity | **<** *[minimum\_humidity]* | R5. Start humidifier | **>=** *[minimum\_humidity] + [maximum\_humidity] / 2* |
|  | ***>*** *[maximum\_humidity]* | R6. Start dehumidifier | **<=** *[minimum\_humidity] + [maximum\_humidity] / 2* |

1. The system will check the humidity level when it is in the balanced state every *[humidity\_balanced\_check\_interval]* minutes*.*
2. The system will check the humidity level when it is in the balancing state every *[humidity\_balancing\_check\_interval]* minutes*.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **Until when to do** |
| pH level | ***<*** *[minimum\_pH]* | R7. Start alkaline dispenser | **>=** *([minimum\_pH] + [maximum\_pH]) / 2* |
|  | ***>*** *[maximum\_pH]* | R8. Start acid dispenser | **<=** *([minimum\_pH] + [maximum\_pH]) / 2* |

1. The system will check the pH level in the water when it is in the balanced state every *[pH\_balanced\_check\_interval]* minutes.
2. The system will check the pH level in the water when it is in the balancing state every *[pH\_balancing\_check\_interval]* minutes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **Until when to do** |
| Electrical conductivity | ***<*** *[minimum\_EC]* | R9. Increase EC | **>=** *([minimum\_EC] + [maximum\_EC]) / 2* |
|  | ***>*** *[maximum\_EC]* | R10. Decrease EC | **<=** *([minimum\_EC] + [maximum\_EC]) / 2* |

1. The system will check the EC level in the water when it is in the balanced state every *[EC\_balanced\_check\_interval]* minutes.
2. The system will check the EC level in the water when it is in the balancing state every *[EC\_balancing\_check\_interval]* minutes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **Until when to do** |
| Water level | ***<*** *[minimum\_water\_level]* | R9. Pump water in the reservoir | **>=** *([minimum\_water\_level] + [maximum\_water\_level]) / 2* |
|  | ***>*** *[maximum\_water\_level]* | R10. Open the valve of the reservoir | **<=** *([minimum\_water\_level] + [maximum\_water\_level]) / 2* |

1. The system will check the water level when it is in the balanced state every *[waterlevel\_balanced\_check\_interval]* minutes.
2. The system will check the water level when it is in the balancing state every *[waterlevel\_balancing \_check\_interval]* minutes.
3. If the system cannot solve an issue, it will alert the administrator.
4. The system generates reports based on the inputs of the administrator.

3. Design Diagrams

3.1 Use-Case Diagram

The use-case diagram (*seen in* *Figure 2*) is aimed to graphically capture the system’s actors, i.e. the administrator, and the actions which can be performed: initialize the system parameters, adjust system parameters, generate reports, receive alerts and reset the system. This diagram provides a structure for our application as it helps to identify components in the design phase. It also helps to capture the requirements which are presented in detail.

The system administrator should be a biologist/botanist who can supervise the growing phase, detect problems in time and adjust the parameters accordingly. He plants the seeds and initializes the system. Also, he harvests the plants and resets the system.

Diagram

Description automatically generated

Figure 2 Use-case diagram of the Vertical Farm Control System

Now the use cases will be presented:

**Use Case 1**: initialize system parameters *(for sequence diagram see Figure 3)*

**Primary Actor**: administrator

**Main Success Scenario**:

1. The administrator inserts the values for the: time, air temperature, water temperature, nutrient levels, light intensity, light spectrum, humidity, pH level, and conductivity level

2. The administrator clicks on the “validate input parameters” button

3. The application validates the data and displays a message informing the administrator to start the system

4. The administrator clicks on the “start” button

5. The application starts the system

**Alternative Sequence**: Invalid values for the setup parameters

The administrator inserts invalid values for the application’s setup parameters

The application displays an error message and requests the administrator to insert valid values

Diagram

Description automatically generatedThe scenario returns to step 1

Figure 3 Sequence diagram for use case 1

**Use Case 2**: adjust system parameters *(for sequence diagram see Figure 4)*

**Primary Actor**: administrator

**Main Success Scenario**:

1. The administrator selects the parameter to edit

2. The administrator clicks on the “Edit” button

3. The administrator enters the new parameter

4. The new parameter is saved

5. A confirmation message is displayed

**Alternative Sequence**: Invalid values for the parameters

The administrator inserts invalid values for the application’s parameters

The application displays an error message and requests the admin to insert valid values

The scenario returns to step 3



Figure 4 Sequence diagram for use case 2

**Use Case 3**: get a live status report *(for sequence diagram see Figure* 5*)*

**Primary Actor**: administrator

**Main Success Scenario**:

1. The administrator should select the data based on which they want to create the reports

2. The administrator presses the “Generate report” button

3. A report is generated



Figure 5 Sequence diagram for use case 3

**Use Case 4**: act on alerts *(for sequence diagram see Figure* 6*)*

**Primary Actor**: administrator

**Main Success Scenario**:

1. The application senses unusual behavior which requires further action from the administrator

2. The application displays an alert message informing the administrator about some specified faults

3. The administrator reads the alert and takes further action.

Figure 6 Sequence diagram for use case 4

**Use Case 5**: reset the system *(for sequence diagram see Figure 7)*

**Primary Actor**: administrator

**Main Success Scenario**:

1. The administrator should press the “Reset” button

2. The application returns to an initial phase

Figure 7 Sequence diagram for use case 5

3.2 Control System Diagram

The block diagram of the system (*seen in* *Figure* *8*) shows the interaction between the components of the system (sensors, actuators, the control algorithm, and the plant). Each block represents one of these components. Since the input depends also on the measured quantities, we are using a feedback loop that returns the measured values from the sensors.



Figure 8 Feedback Control System Diagram

3.3 Finite State Machine

The finite state machine diagram below (seen in *Figure 9*) captures the transitions between the states of the system. It starts with the “system initialized” state, after the admin inputs all the necessary system parameters, and then it continues until the crops are harvested. The system has 6 states in total: “system initialized”, “data collection”, “environment balancing”, “balanced environment”, “alerted” and “mature crops”.

The “alerted” state is one of the more complex states that needs two measurements and has 2 different cases in which the system needs to enter this state. Firstly, in the positive-value case, the system determines by subtraction if the current measurement is less than the target value. If it is, then it takes a second measurement value after it has taken some action to correct the problem. This is the m2 value. Then it compares the first measured value and the second one. If the target value of the measurement has not improved (in less than or equal to the first measurement), then something in the system must be broken. Thus, an alert must signal this to the admin. In the negative-value case, if we need to decrease a measured value but the second measurement seems to be greater than or equal to the first one, then the system sends an alert.

Symbols

T = predefined time interval when the sensors are off, the interval at which we make measurements in a balanced state

t = the current time since the system has entered the current state

xtarget = the target value that the system is aiming to achieve

xm1 = the value of the first measurement

xm2 = the value of the second measurement

predefined\_time = the interval at which we make measurements in the environment balancing state

Diagram

Description automatically generated

Figure 9 Finite state machine diagram

3.4 Data flow

The flow of data in our system is illustrated by the Data Flow Diagrams below. They capture the movement of data between external entities, processes, and data stores.

Graphical user interface, application

Description automatically generatedOn **Level 0** there is a **Context Diagram** *(see Figure 10)* which is a high-level data flow diagram. It shows that we have an *administrator* and the *physical system* as entities that interact with the *application* which represents the whole control system as a process. The data sent by the *administrator* to the *application* are the initial parameters. It gets back from the *application* the status report. On the other hand, we have the *physical system* as an entity as well. It gets the start command from the *application*, and it sends back the measured values

Figure 10 Level 0 (Context) data flow diagram

The **level 1** *(see Figure* 11*)* diagram puts more detail in the context diagram. It highlights the main functions of the system. In addition to the diagram, a level below it adds the *database* which is used by the *start system* process to save measurement data into it. It is also queried by the *generated live* *report* process for which it provides the measured values in the past. The *generate live report* process then sends the report to the *administrator*.

Figure 11 Level 1 data flow diagram

Graphical user interface, application

Description automatically generatedThe **Level 2** *(see Figure 12)* Data Flow Diagram adds even more detail than the previous ones. The *start system* process is divided further into *validate input* process that receives the input from the user upon initialization, sends the data to the database, and the *environmental balancing* process. This process constantly compares the values from the sensors, using a new process called *retrieve sensor values,* to the ones set by the admin. In case the expected values are different from the actual values the process sends command data to the physical system to start the required actuators. If something unusual is detected this process sends the unusual values to the *show alerts* process to generate an alert for the administrator. Another process that resulted from the initial *start system* process is the *periodically compare of ideal and actual state* process. This will get its data from the database (ideal state) and the sensors through the *retrieved sensor values* process (actual state) and if those do not match it sends the values that Diagram

Description automatically generated with medium confidenceare wrong to the *environment balancing* process. The *show parameters and report* process gets the data from the database and sends a report or just plain data to the administrator to take further action.

Figure 12 Level 2 data flow diagram

4. UML Diagrams

4.1 UML Package Diagram

The *ControlSystem* package contains the main logic of the controller. It is divided into 3 packages: *JDBC (*holds classes that communicate directly with the database*)*, *EnvironmentControllers*, and *InputParameters*. The *GUI* package contains the classes for the admin and environment simulation panels, the *EnvironmentSimulator* package contains the logic for simulating the environment. The *SystemConfiguration* package is responsible for reading system parameters from the app.conf file and from environment variables. Finally, the *Utils* package holds logic for time conversions so far.

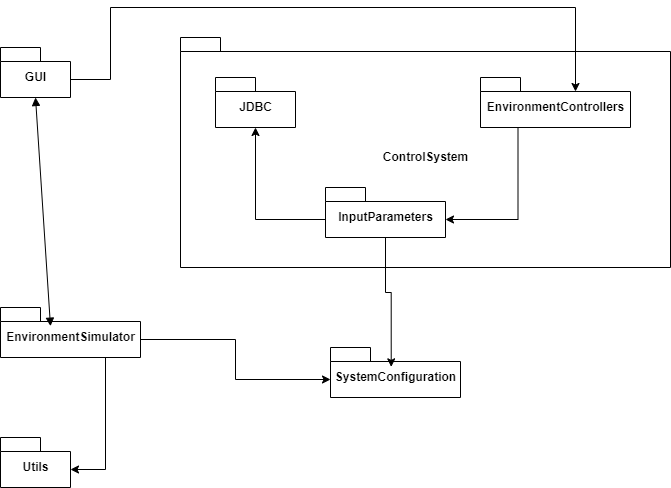


Figure 13 Package Diagram

4.2 UML Class Diagrams

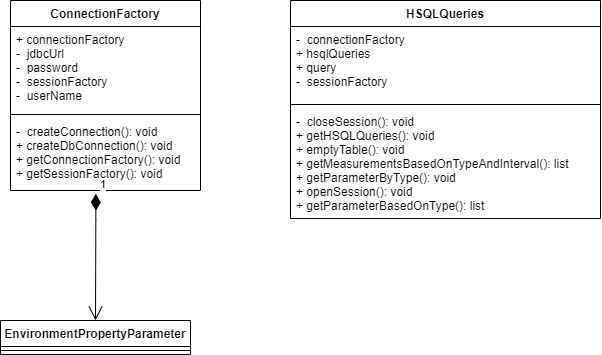
Now the class diagrams will follow.

Figure 14 Classes of the JDBC package

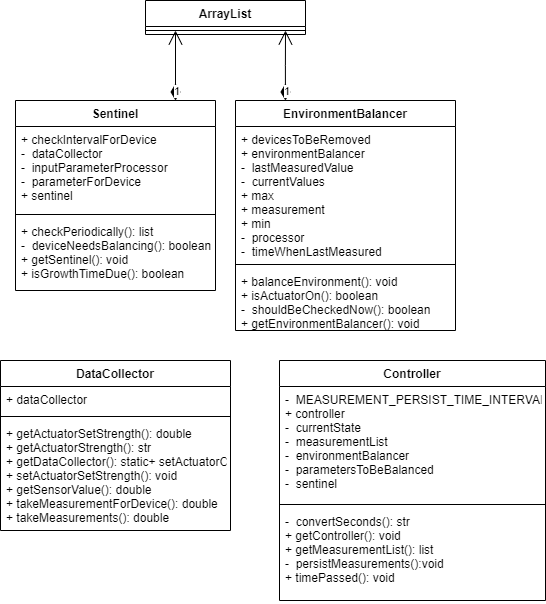


Figure 15 Classes of the EnvironmentControllers Package

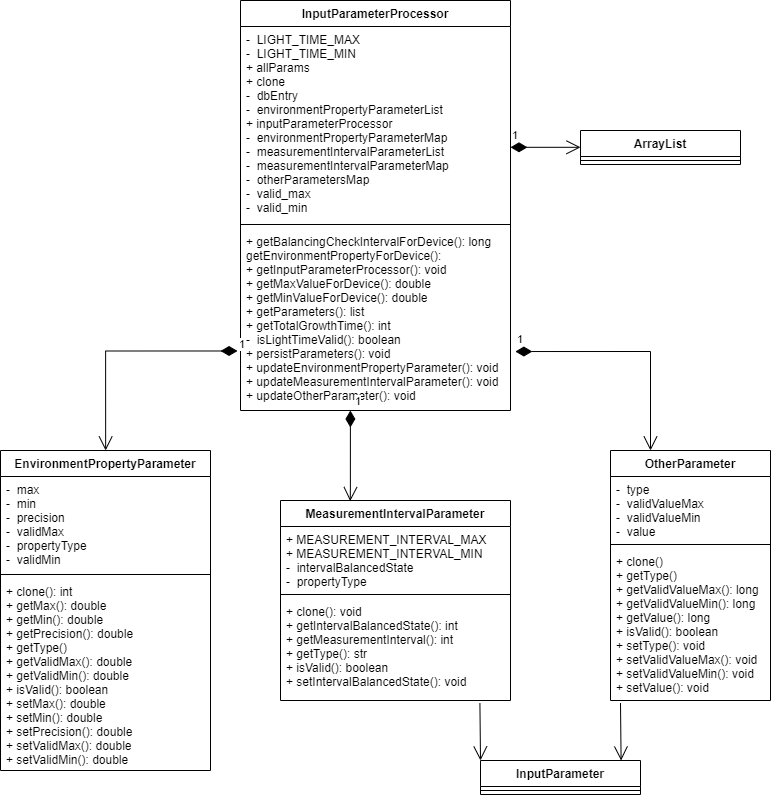


Figure 16 Classes of the InputParameters Package

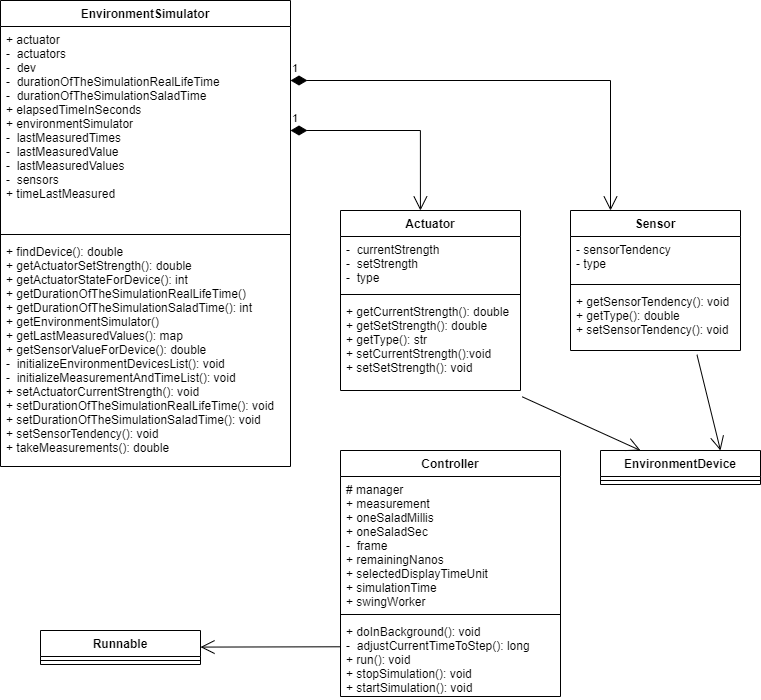
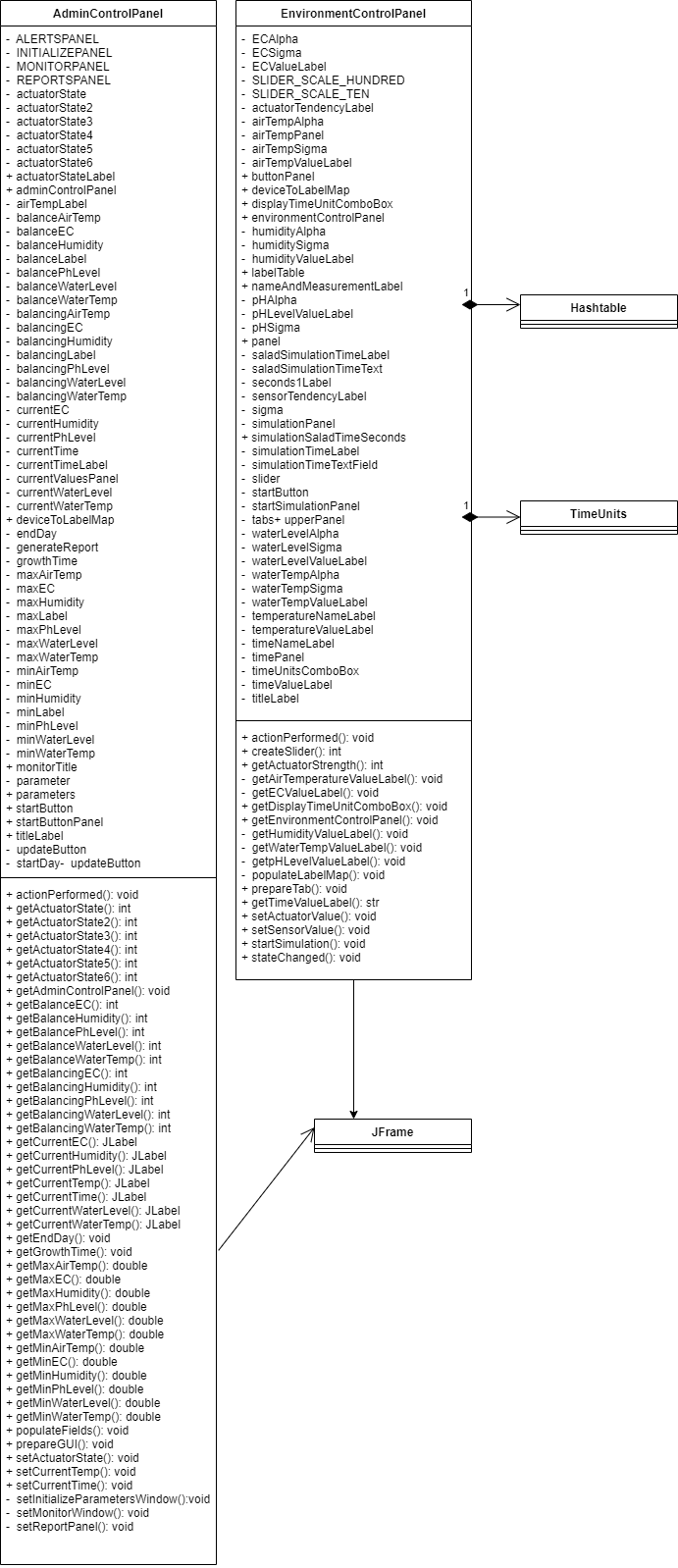


Figure 17 Classes of the EnvironmentSimulator Package

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Figure 18 Classes for the Utils Package

Figure 19 Classes for the SystemConfiguration Package

Figure 18 Classes for the GUI Package

Graphical user interface, text

Description automatically generated

Figure 20 Classes for the Reporting Package

5. Final Version of the Application

The link to the github repository where the source code is:

<https://github.com/JakabSarolta/Software-Engineering-Project/tree/code>

With this latest version of our application, we aim to show how the whole system would behave. Updates could be added to upgrade the system. However, in this phase, the control system would be able to functionally operate and produce salads.

5.1 Setup

To run the project, the host machine must have installed a MySql database server, since the input parameters and the measurement values for reports are persisted in the database. The first step to set up the project is to create a new schema. After that, 3 environment variables need to be created on the machine: the first one has to have the name *CS\_DB\_USER* and as value has to contain the url of the schema created (jdbc:mysql:*{URL}*), the second environment needs to have the name *CS\_DB\_USER* and to contain the username for the DB connection, and the third variable has to be the password of the connection within a variable named *CS\_DB\_PASS*. In case of conflict the name of the environment variables used can be modified, but then the corresponding field in the app.conf needs to be updated as well.

Also, in the setup phase the app.conf file can be modified to give the valid ranges for the input parameters.

5.2 Description of Application

This is the user manual that explains step by step how to use the application.

When you open the application, the first panel that pops up is the admin panel (Figure 21). The admin is the only user of this system. There are 3 different tabs for 3 different actions possible: Initialize/Update parameters, Monitor system, and Reports.

Table

Description automatically generated

Figure 21 Administrator control panel

To initialize the system, the user must type in the desired values: how long is the growing phase (in days), the minimum and maximum parameter thresholds (their units are specified individually), and the how often should the system perform a check on them (different for a well-balanced state and for a state in which the actuators are changing the values constantly - balancing). Then the update button saves these values in the database. Next time when the application starts, the admin will automatically see these previous values that he/she set.

The Monitor panel (Figure 22) shows the current time of the growing phase during the simulation, the parameters, and the actuator’s state (on/off). When the start button is pushed, the simulation window will pop up.

Table

Description automatically generated

Figure 22 Monitor System Panel

For the simulation to start, we need to set certain parameters, specific to the simulation environment (Figure 23). The concept of time divides into simulation time (in real life how long is the simulation going to run), and salad simulation time (how long are we growing salads in real life for – tipically a couple of days). This salad growing interval can be set to seconds, minutes, hours, or days.

After pressing the “start” button, the simulation starts running in real time.

Table

Description automatically generated

Figure 23 Control Simulation Window

To indicate the sensors’ measured values in this simulation, we use sensor tendency slider. It indicates if the simulated measurements should keep increasing or decreasing with every second. The use can change this value during simulation and the effect will be visible on the monitor panel.

The actuator tendency indicates the actions that the actuators take: if the slider is in the positive range, then a heater would turn on, in case of temperature for example (Figure 24). If the slider is in the negative range, then the cooler would turn on.

Graphical user interface

Description automatically generated with medium confidence

Figure 24 Setting the Parameters

To follow the simulation, the user should look at the monitor system window. There, the salad time is shown, the current parameter values are shown, and the actuators that are being turned on or off. After the predefined simulation is over, the time stops, and a report can be generated, or another simulation can be started.

To generate a report, the user must switch to the Reports tab (Figure 26). There, he must type in a start day and a final day for the report. Then, he must specify for which parameter he wants to generate a report for. When pressing the GENERATE button, a pdf is generated with the corresponding graph that shows the changes through this time interval.

Graphical user interface

Description automatically generated with medium confidence

Figure 25 Generate Report Window

In case the user indicates days that do not follow these conventions, no report gets generated. (Figure 27)

Graphical user interface

Description automatically generated

Figure 26 Invalid Input Alert

After the desired reports are generated, the simulation can be started again.

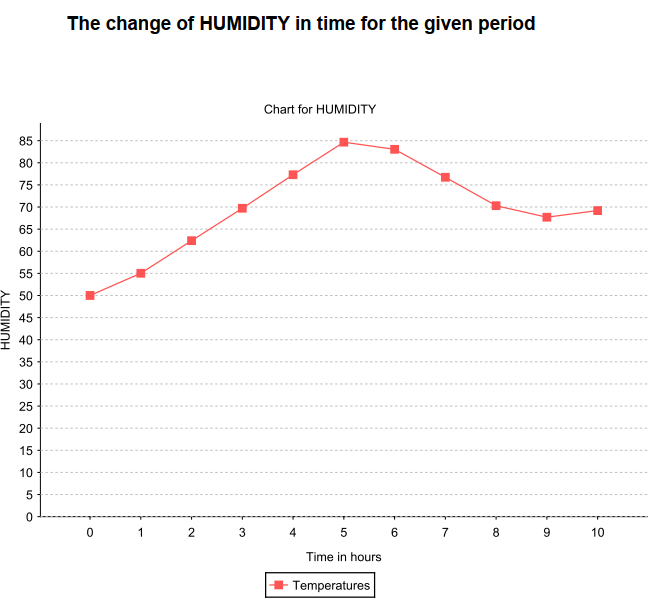


Figure 27 Example of a report

In case of abnormalities during the simulation, the admin gets notified by a window that would pop up called Alert. After the admin acknowledges the problem and fixes it, the simulation continues.

5.3 Description of Packages

The GUI package is the graphical user interface. This allows the user to interact with the system. This can be seen in Figure 18.

The EnvironmentSimulator is the package that is used to simulate a pretend environment. This would not be a package used in a real-life application of this prototype, as they would not need fake data and time changes. This can be seen in Figure 17.

The ControlSystem package connects everything else with the database. It also updates the parameters in the database. This can be seen in Figure 15.

5.4 Description of Classes

The more important classes in some detail:

**ControlSystem package**

The JDBC class is used to create the connection with the database. It also contains some HQL queries.

The InputParameters package contains three entity classes that will be stored in the database. They hold the ranges for the parameters. It also contains a singleton class for processing the input parameters. This will get all the input parameters from the database. If the new values from the database are valid it updates an entity that holds the ranges of a parameter, measurement intervals and a special parameter. This class also initializes the otherParametersMap. If the database doesn't contain values for that parameter type it sets a default value of 0. It initializes the measurementIntervalList and the environmentPropertyParameterList with values from the database. If the database does not contain values for these parameter types, it also initializes them with 0. Then it saves all the parameters into the database.

The EnvironmentControllers package contains a class named Controller. It is called when a second passed. If the system is not in a special state, and if the crops are not ready to be harvested yet, it performs check for parameters in the balanced state and for devices in the balancing state. The EnvironmentControllers package has a class for the data collection state called DataCollector. This class just calls functions from the environment simulator. Its only job is to collect data without processing it. There is also a class that corresponds to the balancing state called EnvironmentBalancer. This balances the parameters that are given by turning on and off the actuators depending on the measurements taken and the target values set by the administrator. It looks at the time intervals set in the intervalBalancingState field of the parameter. It tries to bring the values to the average of the min and max range. If a parameter is balanced it removes it from the list. The EnvironmentControllers package also has a singleton class corresponding to the balanced state, called Sentinel. It is called periodically to check whether parameters need to be balanced. It does the check at intervals set in the intervalBalanced field. Checks if the current value measured by the sensor is between the normal ranges given and if the salad should be harvested. If the values are not in the normal ranges, it adds to the list of parameters that need to be balanced by the balancer.

The Reporting package has the logic to generate reports. The ReportGenerator class processes the input sent by the GUI, retrieves the data using the HSQLQueries class, and builds a Jasper Report whose template is in the resources in the ReportTemplates folder. It saves the generated file in a pdf in a GeneratedReports file.

**SystemConfiguration package**

This package contains logic to read the system parameters from the app.conf file and from the environment variables. Beside the class it contains an enumeration too, that the matches the names of the parameters to the values in the config file

**EnvironmentSimulator package**

The Actuator class is a POJO class that helps modelling an actuator that has a strength and a type.

The Sensor class models the sensors that a hydroponic vertical farm would need to keep the salad plants healthy.

The Controller class is the class that runs the environment simulation. It is a class that starts the simulation by saving the values entered in the GUI and creates a new thread so that it can run in parallel with the GUI. The main thread simulates the change of time. It runs the while loop every second (in the life of a salad). Each time it takes a measurement, calls the control system, and displays the values to the environment simulation control panel.

The EnvironmentSimulator class contains the current state of the environment simulation. It holds data of the measurement values, the time they were measured, the list of actuators and sensors (and their state), growth time, and the simulation duration. It is also responsible for computing the value of the next measurement. This class takes measurements for all the sensors and returns a map of them and their measured values. It also takes measurements for only one sensor. It computes the current measurement value by taking into consideration the last measured value and the time it was measured, the amount the parameter changes in one second, and the strength of the actuator. The class also sets the amount with which the value of a sensor changes in one second (real life) and the amount with which the actuator tries to counter-act the growth of the parameter. It finds an entity of a device from the list of devices based on its type. The class also initializes the lists of environment devices (actuators and sensors) and the list of last measured values by the default values written in.

6. Testing

6.1 White-box Testing

For white box testing we have tested separate methods and classes on their own. Since the most important package that contains the logic behind the control system is the ControlSystem, the test cases are targeting the classes mostly from this package. We used the Junit framework for these tests.

Having a complex input domain, we partitioned the inputs into valid and invalid ones. Our program makes sure that whenever an invalid input is inserted, a pop-up window will appear, providing the user some information to help them correct the values. Valid inputs can be further decomposed into equivalence classes. Depending on time, values outside the minimum and maximum range should be balanced. These out-of-range values are in one class and in the other class should be the values where no balancing is needed. However, the out-of-range can be decomposed into below limit and over limit classes. When a value is under the limit, we should increase that value. Similarily, when a value is over the limit, it should be decreased.

6.1.1 TemperatureControl Package Tests

This package contains a few test cases for controlling the air temperature. Our program uses the same methods for controlling the other parameters, so it is enough to test it on only one parameter, i.e. air temperature in this case.

For testing the methods in the EnvironmentControllers package, we made a few test classes with several methods, each one checking a certain aspect of the controller. So, in the TemperatureControl package, we have the following test classes: coolingTest, growthTimeDueTest, heatingTest, isActuatorOnTest and temperatureStepChangeTest.

The coolingTest and heatingTest classes contain similar methods. Here we checked whether the temperature increases or decreases accordingly if the temperature value is outside the given range. We called the timePassed() function at a time in which the temperature should be checked and then called that function again 10 times. The sensor tendency being 0.1, the temperature should increase or decrease (depending on the test type) by 1 °C, at each second 0.1 °C.

The temperatureStepChangeTest contains a method which checks whether the temperature truly changes with the difference between the actuator and the sensor tendency. With the sensor tendency set to -0.1 and the actuator tendency set to 0.2, the temperature should increase by 0.1 °C in each second.

The growthTimeDueTest and the isActuatorOnTest classes both contain 2 methods. One is for checking a True and another for checking a False case. We checked whether the isGrowthTimeDue() method returns True for 30 days and False for 29 days, the actual growth time being set to 30 days. Vertical Farm Control System Group: 10 30 The actuator should only be on when the temperature is outside the limits and the system enters the balancing state.

6.1.2 InputParameters Package Tests

These tests aim to check the updating of the different types of parameters. In all cases, a test case tests for true and another for false comparisons.

Diagram

Description automatically generated

Figure 28 Flowchart for a test case

updateCheckTimeTest class: checks whether the time interval of checks is updated in balanced and in balancing states. By calling the corresponding update function from the InputParameterProcessor, updateMeasurementIntervalParameter(). Two methods check if the values after completing the update are equal to these predefined values. To realize this, assert statements are used.

updateGrowthTimeTest class: the fact that the growth time is updated will be checked. By calling the corresponding update function from the InputParameterProcessor, updateOtherParameter(), the growth time is set to 15 days. Two methods check if the value after completing the update is equal to these predefined values. To realize this, assert statements are used.

updateMinAndMaxTempTest class: the updating of the minimum and maximum temperature is checked. By calling the corresponding update function from the InputParameterProcessor, updateMeasurementIntervalParameter(), the minimum and maximum temperatures are set to the hardcoded values 14 and 19. Two methods check if the values after completing the update are equal to these predefined values. To realize this, assert statements are used. The flowchart diagram for this test-case can be seen in Figure 28.

6.1.3 EnvironmentSimulator Package Tests

This test package is aimed to test the methods of the EnvironmentSimulator. The tested methods were the following: findDevice(), setActuatorCurrentStrength(), setActuatorSetStrength(), setSensorTendency(), takeMeasurement().

The findDeviceTest() checks whether the corresponding method returns the device of the required type. We only checked it for one device type, pH level, as the method should work in the same manner for other types.

The setActuatorCurrentStrengthTest() method checks whether a value is truly set for a certain actuator. The tested method is quite straightforward, the flow of control might only take different directions at the point where the code checks whether the actuator exists (see Figure 29).

Shape

Description automatically generated with medium confidence

Figure 29 Control Flow Graph for setActuatorCurrentStrength() method

The setActuatorStrengthTest() method is similar to the one before, as the methods they are testing are both similar. They only differ in where the actuator value is taken from. In the latter one, the actuator strength is taken from the slider, and in this case, the strength will not directly influence the measurements. The other function sets the value with which our actuator actually counter acts.

setSensorTendencyTest() works in a similar manner, we only had to test it for a certain device type.

The takeMeasurementTest class tests a more complex method, which takes a measurement whenever it is invoked. It also works with different device types, so we chose one among these arbitrarily.

6.2 Black-Box Testing

6.2.1 BlackBox Package Tests

The BlackBox package contains the tests for black box testing. For this, we made first a TestDriver class, which can start the simulation for some hard-coded values, without having to insert values using the interface. Here, we considered the equivalence classes constructed at the beginning of this section. The invalid inputs were not tested here, as we made sure in the GUI that the user can start the simulation only if they inserted valid values.

Our system is complex, it changes in every second. However, when testing, it can be reduced to some most important basic cases. We used the requirements to capture them. We created three test classes, which would test different aspects of our system.

The BlackBoxTest10Sec class checks whether the parameter values change accordingly after 10 seconds have passed in the simulation. We only had to calculate how much the sensor values would change within these seconds. By the hardcoded values we assured that no balancing is done in this stage.

The BlackBoxTest32Min checks whether we entered in the balancing process for a certain parameter, i.e. the humidity in our case. 32 minutes of salad time correspond to approximately 16 seconds in our simulation. We could say that in the first 30 minutes (first 15 seconds), no balancing is needed. But after this, the humidity should enter in the balancing state. The other values will continue to change in the same manner as before, but the value of the humidity will change in the opposite direction, taking into consideration not only the sensor, but the actuator value as well.

The BlackBoxBalancedTest class checks whether a certain parameter will stop at a value inside the range after balancing was made. We only checked this value for humidity, as this was the one which needed balancing in the previous test case.

7. Conclusions

Our team has learned a lot while working on this project. The main thing that we learned was how to work more effectively in a team. To work effectively we had to make sure that each of us listened to each other without interrupting and to make sure everyone had a chance to give input on the project. We had to learn how to work with and around the level of knowledge each of us had about the topic and the system. We learned how to communicate effectively to make sure that we could meet regularly and get all of the tasks completed when we needed them to be. We also had to learn how to communicate in a respectful way when we wanted to change things that other people had done. We also had to learn how to divide the tasks that needed to be completed. We had to make sure that the tasks that we asked everyone to complete played to their strengths, and that they were given a reasonable about of time to complete the tasks. Even though we wanted to get everything done when we said we would have it done, we had to learn to be flexible and work with each other when something could not be completed in time. One way we did this was by using Jira software. Jira was a new product to us and it took a bit of trial and error to get it working properly and effectively. Once we had it working, it became a very useful tool for us to keep an eye on how the project was progressing and what parts needed more work and attention than others. Some of the members of the team also had to learn how to use Hibernate and Jasper Reports in Java. They also had to take architectural decisions to make any modifications and developments that need to be done in the future easier.

One of the things that we learned was how to use GitHub as a team. One of the members of our group had a very limited knowledge of how to use GitHub, so they had to learn how to use GitHub from the very beginning. Using GitHub as a team is different from using it as a singular person, as you must make sure you are working on the most recent update when making changes to any documents. This may lead to changes not being made when they were planned. We also had to learn how to work on different branches on GitHub which raised another challenge. Another thing we learned was how to write and format a document for a software project. This was different from previous documentation we had done which meant we had to reformat things occasionally. Another thing we learned was how to create many different types of diagrams. We first had to learn about the diagrams and then put them into practice. Some of the diagrams seemed very complicated to make at first. However, after we did research they became easier to do and we created many complete diagrams. There were many different pieces of information we needed to create these diagrams and we had to learn which pieces were more important and needed to be included as a part of the diagram. We also learned about how creating diagrams can make planning the system much easier as we can just follow them. We learned that working in a team can make many tasks easier to complete than they would be if we were doing them alone. Completing them in a team meant that we would get them done faster and that if we had any problems we could get help from one of our team mates instead of trying to figure it out on our own, which would have taken a lot longer.

8. References

1. “IoT-based Hydroponic Farms” [Google Scholar]

Retrieved from: <https://ieeexplore.ieee.org/document/8748447>

1. “Nutrient Use in Vertical Farming: Optimal Electrical Conductivity of Nutrient Solution for Growth of Lettuce and Basil in Hydroponic Cultivation” [Google Scholar]

Retrieved from: <https://www.mdpi.com/2311-7524/7/9/283/htm>

1. Data Flow Diagrams:   
   Retrieved from: <https://www.youtube.com/watch?v=euI2AFobS0w>  
   <https://www.geeksforgeeks.org/levels-in-data-flow-diagrams-dfd/>