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](#_Toc120803975)

[1. Introduction 6](#_Toc120803976)

[1.1 Background 6](#_Toc120803977)

[1.2 Different Possible Approaches 6](#_Toc120803978)

[1.3 Pros and cons of vertical farming 7](#_Toc120803979)

[2. Requirements 7](#_Toc120803980)

[2.1 Non-Functional Requirements 7](#_Toc120803981)

[2.2 Functional Requirements 8](#_Toc120803982)

[3. Design Diagrams 11](#_Toc120803983)

[3.1 Use-Case Diagram 11](#_Toc120803984)

[3.2 Control System Diagram 14](#_Toc120803985)

[3.3 Finite State Machine 15](#_Toc120803986)

[3.4 Data flow 17](#_Toc120803987)

[4. UML Diagrams 19](#_Toc120803988)

[4.1 UML Package Diagram 19](#_Toc120803989)

[4.2 UML Class Diagrams 19](#_Toc120803990)

[5. Final Version of the Application 24](#_Toc120803991)

[5.1 Setup 24](#_Toc120803992)

[5.2 Description of Application 24](#_Toc120803993)

[5.3 Description of Packages 27](#_Toc120803994)

[5.4 Description of Classes 27](#_Toc120803995)

[5. Testing 29](#_Toc120803996)

[5.1 White-box Testing 29](#_Toc120803997)

[5.1.1 EnvironmentControllers Package Tests 29](#_Toc120803998)

[5.1.2 InputParameters Package Tests 29](#_Toc120803999)

[5.2. Black-box testing 30](#_Toc120804000)

[6. Conclusions 31](#_Toc120804001)

[6. References 32](#_Toc120804002)

List of Figures and Tables

[Figure 1 Hydroponics in vertical farming systems 5](#_Toc120010017)

[Figure 2 Use-case diagram of the Vertical Farm Control System 10](#_Toc120010018)

[Figure 3 Sequence diagram for use case 1 11](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010019)

[Figure 4 Sequence diagram for use case 2 12](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010020)

[Figure 5 Sequence diagram for use case 3 12](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010021)

[Figure 6 Sequence diagram for use case 4 13](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010022)

[Figure 7 Sequence diagram for use case 5 13](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010023)

[Figure 8 Feedback Control System Diagram 14](#_Toc120010024)

[Figure 9 Finite state machine diagram 15](#_Toc120010025)

[Figure 10 Level 0 (Context) data flow diagram 16](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010026)

[Figure 11 Level 1 data flow diagram 16](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010027)

[Figure 12 Level 2 data flow diagram 17](file:///C:\Users\andre\Dropbox\My%20PC%20(LAPTOP-AUV3VKT8)\Documents\Uni\V.Sem\Software%20engineering\Software-Engineering-Project\Team10_D3.docx#_Toc120010028)

[Figure 13 Administrator control panel 19](#_Toc120010029)

[Figure 14 Monitor System Panel 20](#_Toc120010030)

[Figure 15 Control Simulation Window 21](#_Toc120010031)

[Figure 16 Monitor System during Simulation 21](#_Toc120010032)

|  |  |  |
| --- | --- | --- |
| ***Tasks of individual team members for D3*** | | |
| 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 disadvantage 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 are concerning 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 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **When to do** |
| Air temperature | **<** *[minimum\_air\_temperature]* – [*temperature\_treshold]* | R1. Start heating the air | **>=** *[minimum\_air\_temperature]* |
| **>** *[maximim\_air\_temperature*] – [*temperature\_treshold]* | R2. Start cooling the air | **<=** *[maximum\_air\_temperature]* |

1. The system will check the air temperature every *[air\_temperature\_check\_interval]*.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **When to do** |
| Water temperature | **<** *[minimum\_water\_temperature]* **–** *[temperature\_treshold]* | R3. Start heating the water | **>=** *[minimum\_air\_tempreature]* |
|  | **<** *[maximum\_water\_temperature]* **–** *[temperature\_treshold]* | R4. Start cooling the water | **<=***[maxium\_air\_tempreature]* |

1. The system will check the water temperature every *[water\_temperature\_check\_interval]*.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **When to do** |
| Humidity | **<** *[minimum\_humidity] – [humidity\_treshold]* | R5. Start humidifier | **>=** *[minimum\_humidity]* |
|  | ***>*** *[maximum\_humidity] – [humidity\_treshold]* | R6. Start dehumidifier | **<=** *[maximum\_humidity]* |

1. The system will check the humidity level in the room every *[humidity\_check\_interval].*

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **When to do** |
| pH level | ***<*** *[minimum\_pH] – [pH\_treshold]* | R7. Start alkaline dispenser | **>=** *[minimum\_pH]* |
|  | ***>*** *[maximum\_pH] – [pH\_treshold]* | R8. Start acid dispenser | **<=** *[maximum\_pH]* |

1. The system will check the pH level in the water every *[pH\_check\_interval]*.

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment property** | **When to do** | **What to do** | **When to do** |
| Electrical conductivity | ***<*** *[minimum\_EC] – [EC\_treshold]* | R9. Increase EC | **>=** *[minimum\_EC]* |
|  | ***>*** *[maximum\_EC] – [EC\_treshold]* | R10. Decrease EC | **<=** *[maximum\_EC]* |

1. The system will check the EC level in the water every *[EC\_check\_interval]*.

The *[parameter\_name]* is representing 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 is continuing heating until the temperature gets above the minimum temperature. The system has other requirements such as:

1. The system will check the nitrogen(N) level in the water every *[nitrogen\_check\_interval]*.
2. The system will check the phosphorus(P) level in the water every *[phosphorus\_check\_interval]*.
3. The system will check the potassium(K) level in the water every *[potassium\_check\_interval]*.
4. It will ensure the levels are kept to an NPK ratio of *[NPK\_ratio]* by adding nutrients as needed.
5. The system will turn the lights on for *[light\_ON\_time]*.
6. The system will turn the lights off for *[light\_OFF\_time]*.
7. If the system cannot solve an issue it will alert the administrator.
8. The system will check the air temperature every *[air\_temperature\_check\_interval]*.
9. The system will check the water temperature every *[water\_temperature\_check\_interval]*.
10. The system will check the nitrogen(N) level in the water every *[nitrogen\_check\_interval]*.
11. The system will check the phosphorus(P) level in the water every *[phosphorus\_check\_interval]*.
12. The system will check the potassium(K) level in the water every *[potassium\_check\_interval]*.
13. It will ensure the levels are kept to an NPK ratio of *[NPK\_ratio]* by adding nutrients as needed.
14. The system will turn the lights on for *[light\_ON\_time]*.
15. The system will turn the lights off for *[light\_OFF\_time]*.
16. The system will check the humidity level in the room every *[humidity\_check\_interval].*
17. The system will check the pH level in the water every *[pH\_check\_interval]*.
18. The system will check the EC level in the water every *[EC\_check\_interval]*.
19. If the system cannot solve an issue it will alert 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

Graphical user interface, application

Description automatically generatedThe **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

Diagram

Description automatically generated with medium confidenceThe **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 are 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.

Diagram

Description automatically generated

Figure Package Diagram

4.2 UML Class Diagrams

Now the class diagrams will follow.

Graphical user interface, text, application

Description automatically generated

Figure Classes of the JDBC package

Diagram

Description automatically generated

Figure Classes of the EnvironmentControllers Package

Graphical user interface, diagram, application, Teams

Description automatically generated

Figure Classes of the InputParameters Package

Graphical user interface

Description automatically generated

Figure Classes of the EnvironmentSimulator Package

Text

Description automatically generatedGraphical user interface, application, Teams

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Figure Classes for the Utils Package

Figure Classes for the GUI Package

Figure Classes for the SystemConfiguration Package

5. Final Version of the Application

With this first version of our application, we aim to show how the temperature would behave in our vertical farm control system. A growing salad must also have other parameters adjusted, like lights to be turned on or off, the humidity, the pH level and so on. However, for simplicity, the interface shows only the temperature changes and when the heater or cooler is tuned on or off.

This version will have extensions for different parameters and for live reports and alerts. So far, the initialization and the monitoring of the system are the panels that show accurate results.

5.1 Setup

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. 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 temperature 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 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. 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.

To indicate the sensors measured values in this simulation, the sensor tendency indicates if the simulated measurements should keep increasing or decreasing with every second. The temperature is 20 degrees Celsius to start with.

The actuator tendency indicates the actions that the actuator (heater or cooler in this case) takes: if the slider is in the positive range, then the heater is turned on; if the slider is in the negative range, then the cooler is turned on.

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

Table

Description automatically generated

Figure 23 Control Simulation Window

To follow the simulation, the user should look at the monitor system window. There, the salad time is showing, the current temperature, and the actuator being on or off. After the predefined simulation is over, the time stops, and another simulation can be started.

Graphical user interface

Description automatically generated with medium confidence

Figure 24 Setting the parameters

Graphical user interface

Description automatically generated with medium confidence

Figure 25 Generate Report Window

Graphical user interface

Description automatically generated

Figure 26 Invalid Input Alert

5.3 Description of Packages

The GUI package is the graphical user interface. This allows the user to interact with the system.

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.

The ControlSystem package connects everything else with the database. It also updates the parameters in the database.

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 class 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 of the parameters into the database.

The EnvironmentControllers class contains a class that controls the EnvironmentController. 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. It also converts the time from seconds to a legible format. The EnvironmentControllers class has a class for the data collection state. 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. This balances the parameters that are given by turning on and off the actuator. It looks at the values at 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 class also has a singleton class corresponding to the balanced state. 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.

**SystemConfiguration package**

This class ensures that the class is singleton and reads the value from the configuration file corresponding to the given SystemConfigParameter. It also resolves the value of an environment variable corresponding whose name is given in the configuration file.

**EnvironmentSimulator package**

The Actuator class is an empty class.

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 static class that starts the simulation by saving the values entered into 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 measurement 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 in 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 the parameter 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. Then it ensures this class is singleton. 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.

5. Testing

5.1 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.

5.1.1 EnvironmentControllers Package Tests

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.

5.1.2 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.

updateCheckTimeTest class: the updating of the time interval of checks is updated in balanced and in balancing states. By calling the corresponding update function from the InputParameterProcessor, updateMeasurementIntervalParameter(), the balanced checking time is set to 30 minutes and the balancing check time is set to 5 minutes. 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 updating of the growth time is 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.

5.2. Black-box testing

6. Conclusions

6. 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/>