

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science



Monitoring simulation and automated regulation of parameters of hydroponic system solution

Bachelor thesis

Nikita Bondarev

Faculty: Faculty of Electrical Engineering
Study programme: Software engineering
Supervisor: Ing. David Kadleček, Ph.D.

Prague, 2021

Thesis Supervisor:

Ing. David Kadleček, Ph.D.
Center for Knowledge Management
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
160 00 Prague 6
Czech Republic



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bondarev** Jméno: **Nikita** Osobní číslo: **469810**
 Fakulta/ústav: **Fakulta elektrotechnická**
 Zadávací katedra/ústav: **Katedra počítačů**
 Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Simulace monitoringu a automatizovaná regulace parametrů roztoku hydroponického systému

Název bakalářské práce anglicky:

Monitoring simulation and automated regulation of parameters of hydroponic system

Pokyny pro vypracování:

Vytvořte simulaci monitoringu a automatizované regulace parametrů roztoku hydroponického systému.

Monitorované parametry:

- Úroveň PH (potenciál vodíku)
- Elektrická vodivost
- Teplota

- Objem (výška hladiny)

Automatizace regulace:

- Peristaltická pumpa s nádržemi na PH +- a hnojiva
- Topení
- Chlazení

- Přívodní ventil na vodu a odpust'

Výstup:

- Navrh komponent a jejich integrace
- Prototyp formou počítačové simulace nebo fyzického nasazení

Seznam doporučené literatury:

[1] William Texier – Hydroponie pro každého. ISBN: 978-2-84594-161-8

[2] Stuart Russell / Peter Norvig – Artificial Intelligence: A Modern Approach. ISBN: 9781292153964

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Kadleček, Ph.D., Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **21.02.2021**

Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. David Kadleček, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Declaration

I hereby declare I have written this bachelor thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, 2021

.....
Nikita Bondarev

Abstract

This bachelor thesis is aimed at automatization of work with a hydroponic system's nutrient solution, i.e. monitoring and regulating its parameters depending on a plants' growth phase or a gardener's will. Nowadays the topic is very important since the hydroponics market grows from year to year and sown areas do not. The work contains the study about hydroponics, full hardware, software and technologies analysis, implementation and evaluation.

Keywords: Hydroponics, Automated Hydroponic System, Microservices, Regulation, Sensors.

Tato bakalářská práce je zaměřena na automatizaci práce s živným roztokem hydroponického systému, tj. monitorování a regulaci jeho parametrů v závislosti na fázi růstu rostlin nebo vůli zahradníka. V dnešní době je toto téma velmi důležité, protože trh hydroponie roste, ale oseté plochy se zmenšují. Práce obsahuje teorii o hydroponii, úplnou hardwarovou a softwarovou analýzu, analýzu použitých technologií, implementaci a zhodnocení.

Keywords: Hydroponie, Automatizovaný hydroponický systém, Mikroslužby, Regulace, Senzory.

Acknowledgements

I would like to thank Ing. David Kadleček, Ph.D. for being the project's supervisor. Also would like to thank Pavel Wimmer, Jakub Szasz, Adam Kučera and Jiří Šebek for the support during the project.

List of Tables

3.1	Summarizing computer devices	16
3.2	Required sensors	16
3.3	Required actuators	18
4.1	Comparison of SQL and No-SQL databases	21
4.2	Summarizing programming languages	24
4.3	Comparison of Microservice and Monolith architectures	26
5.1	Functional requirements	32

List of Figures

2.1	DWS system scheme	4
2.2	NFT system scheme	5
2.3	Dutch bucket system scheme	6
2.4	Pebble substrate	7
2.5	Ceramsite substrate	8
2.6	Perlite substrate	8
2.7	Rockwool substrate	9
3.1	Entire system's general scratch scheme	13
3.2	Arduino Uno microcontroller	14
3.3	RaspberryPi microcomputer	15
3.4	System communication scratch scheme	19
4.1	Microservice architecture scratch scheme	25
4.2	Monolith architecture scratch scheme	25
5.1	Communication using channels	33
5.2	Arduino application sequential diagram	34
5.3	Local persistence service sequential diagram	35
5.4	AWS persistence service sequential diagram	36
5.5	Regulation service sequential diagram	37
5.6	System communication scratch scheme	38
6.1	Device configuration format	39
6.2	Sensor data message format	41
6.3	Command message format	41
6.4	Regulation coefficients	46
6.5	Regulation rules	46
7.1	Local persistence service log	49
7.2	AWS persistence service log	49
7.3	AWS DynamoDB console	50
7.4	Low temperature log	50
7.5	Temperature is back to normal log	50
7.6	Low water level log	51
7.7	Water level is back to normal log	51
7.8	Wrong pH log	51
7.9	Wrong pH log	51

Contents

Abstract	vii
Acknowledgements	ix
List of Tables	xi
List of Figures	xiii
1 Introduction	1
2 Introduction to hydroponics	3
2.1 Back to history	3
2.2 Types of hydroponic systems	3
2.2.1 Deep Water Culture (DWS)	4
2.2.2 Nutrient film technique (NFT)	4
2.2.3 Dutch Bucket	5
2.3 Substrates	6
2.3.1 Stones and pebbles	6
2.3.2 Expanded clay (ceramsite)	7
2.3.3 Perlite and Rockwool	8
2.4 Required parameters	9
2.4.1 pH Level	9
2.4.2 Electrical conductivity (EC)	9
2.4.3 Water temperature	10
2.4.4 Air temperature	10
2.4.5 Air humidity	10
2.4.6 Lighting	10
2.4.7 CO2 level	11
3 Hardware analysis	13
3.1 General scheme	13
3.2 Required components	14
3.2.1 Computing power	14
3.2.1.1 Arduino Uno	14
3.2.1.2 Raspberry Pi	15
3.2.1.3 Summarizing	15
3.2.2 Sensors	16
3.2.3 Actuators	18
3.3 Communication	19

4	Used technologies	21
4.1	Data storage	21
4.1.1	SQL vs NoSQL	21
4.1.2	Dynamo Database	22
4.1.3	Redis Database	22
4.2	Programming languages	23
4.2.1	Arduino	23
4.2.2	Python	23
4.2.3	Summarize	24
4.3	Microservice Architecture	24
4.4	Communication protocols	26
4.4.1	I2C	26
4.4.2	UART Protocol	27
5	Software analysis	29
5.1	AS-IS state	29
5.2	Strategic intent	29
5.3	Business intent	30
5.4	5F analysis	30
5.5	PEST analysis	31
5.6	Functional requirements	32
5.7	Non-functional requirements	32
5.8	Communication between Services	32
5.9	Sequential diagrams	33
5.9.1	Arduino application	33
5.9.2	Raspberry Pi Local Persistence Service	34
5.9.3	Raspberry Pi AWS Persistence Service	36
5.9.4	Raspberry Pi Regulation Application	37
5.10	Deployment diagram	38
6	Implementation	39
6.1	Arduino Uno Application	39
6.2	Local persistence service	42
6.3	AWS Persistence Service	44
6.4	Regulation Service	45
6.4.1	Parameters regulation	46
7	Testing and Evaluation	49
7.1	Storing the data	49
7.2	Parameters regulation	50
8	Conclusion	53
A	List of Abbreviations	55
B	Source code	57
	Bibliography	59

1. Introduction

Hydroponics is a long time known method of cultivating plants without any kind of soil by using a solution which contains dissolved nutrients.

Even when hydroponics is one of the most effective methods, which allows gaining both quality and quantity, nowadays it is still less preferred than traditional cultivation methods. There are many reasons for this: starting with still great possibilities to grow plants the old way, finishing with a fear of something new.

According to the UN's message about population increasing, the number of people on the planet will reach up to 10 billion in the next 30 years, which will affect the increase in food demand and reduction of acreage of sown areas.[1]

I believe that this is our common problem now and in the future, and that's why I have chosen it as a theme for my bachelor thesis.

Hydroponics is a complex method. It requires monitoring and regulation of many parameters, which can be automated.

The goal of this project is to create a system that allows automation of the work with a hydroponic system solution. The thesis will cover all the project sides: from hydroponics study up to hardware analysis and software implementation.

The project is a part of a larger one. Another part was managed by my colleague Akhmadov Ali and covers the environment of hydroponic systems. Both parts are integrated into one automated system.[2]

2. Introduction to hydroponics

2.1 Back to history

As outlined in the introduction, hydroponics has been known for a long time. It is not known who started cultivating plants without using any soil, but first written confirmations of people growing them using something like the hydroponic method are found in medieval Southern America. Aztecs and others used to create “artificial islands” with a frame made of a tied reed, filled with soil and placed them on a surface of lakes rich in dissolved salts. That allowed the roots to get nutrients from the water. Such “islands” were found in China and in other parts of the world.

In a year of 1699 English naturalist and historian, John Woodward conducted the first experiment proving that plants get nutrients from water and soil. The experiment also proved that plants can be cultivated using only water. The problem was that people didn’t know what exactly plants needed to grow.

New discoveries and experiments followed in the next 200 years. Were determined parameters and nutrient formulas for plant growth.

William Gericke was another important person, who introduced the term “Hydroponics” in the early 1920s. He was the first who began cultivating plants without using soil but with special nutrient solutions industrially. [3]

To this day, hydroponics constantly changes, adapting to new realities, and rapidly grows.

2.2 Types of hydroponic systems

When it comes to choosing a particular system, there are many different types and options. Some of them work better and more effectively than others depending on the plant we want to grow and the space we have. In this section I will go through the most common of them.

2.2.1 Deep Water Culture (DWS)

The most common and cheapest type of hydroponic system, which can be made even from a usual plastic bucket in every home.

In such hydroponic systems, the container is almost completely filled with a nutrient solution, and the plants' roots are completely immersed in it. Oxygen is provided to them with an air stone connected to an air pump.

The system is suitable for almost all plants but is especially effective for large ones with large root systems.

The main disadvantage of DWS is that the system is passive, therefore, the solution doesn't flow and requires renewals, which is a complex task. [3]

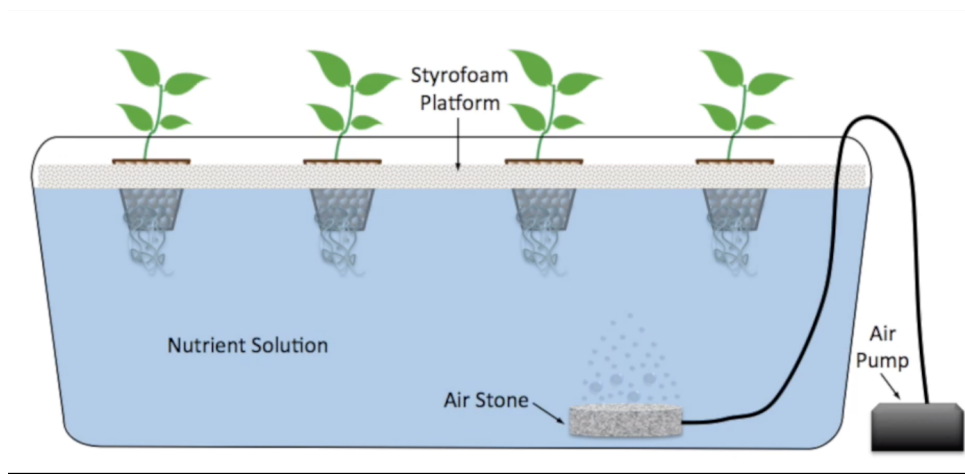


Figure 2.1: DWS system scheme

2.2.2 Nutrient film technique (NFT)

One of the most famous types of systems. This is because it's very productive, since it is suitable for fast-growing plants and is easy to set up to grow more of them at once.

The nutrient solution in such systems is being pumped to channels that hold plants. It flows through them over the plant's roots.

The system is easy to automate.

Disadvantages are unsuitability for large root plants and risk of water pump breakdown, which can cause the death of all plants within a couple of hours. [4]

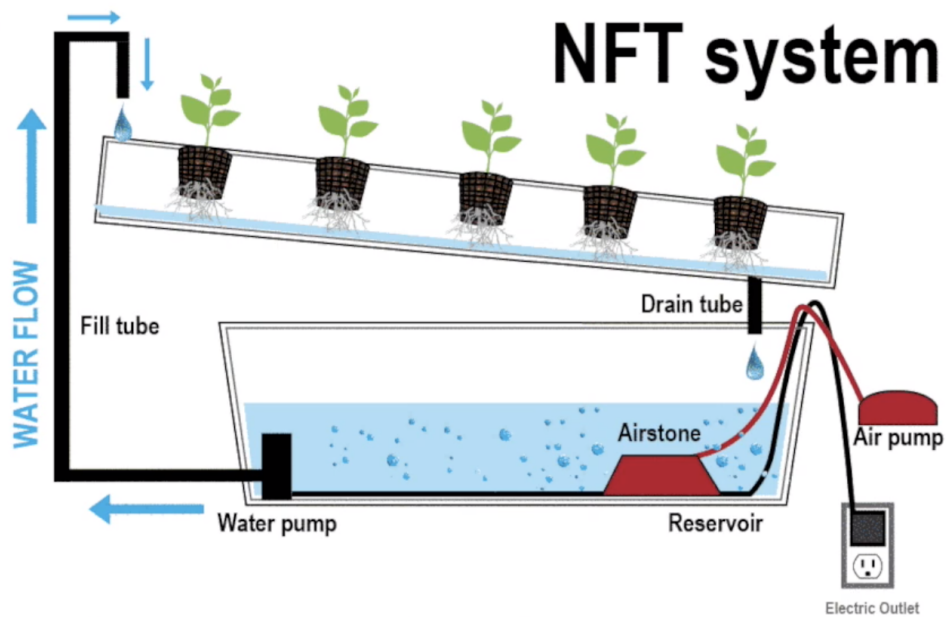


Figure 2.2: NFT system scheme

2.2.3 Dutch Bucket

The system is similar to NFT but cancels its disadvantage of unsuitability for large plants.

The plants are being located in separate pots. There is a water line running from the container with a nutrient solution, which has drip hoses at each plant. The solution flows over their roots and exits a drain, which leads back to the solution container.

The system still has the disadvantage of risk of water pump breakdown and also has a new risk of clogging of an increased amount of pipes and hoses. [5]

Chilli pepper was chosen as a plant, which will be grown during this project. Since its root system is large and we would like to grow as much as possible, the Dutch bucket hydroponic system is the most suitable one for the purpose of the project.

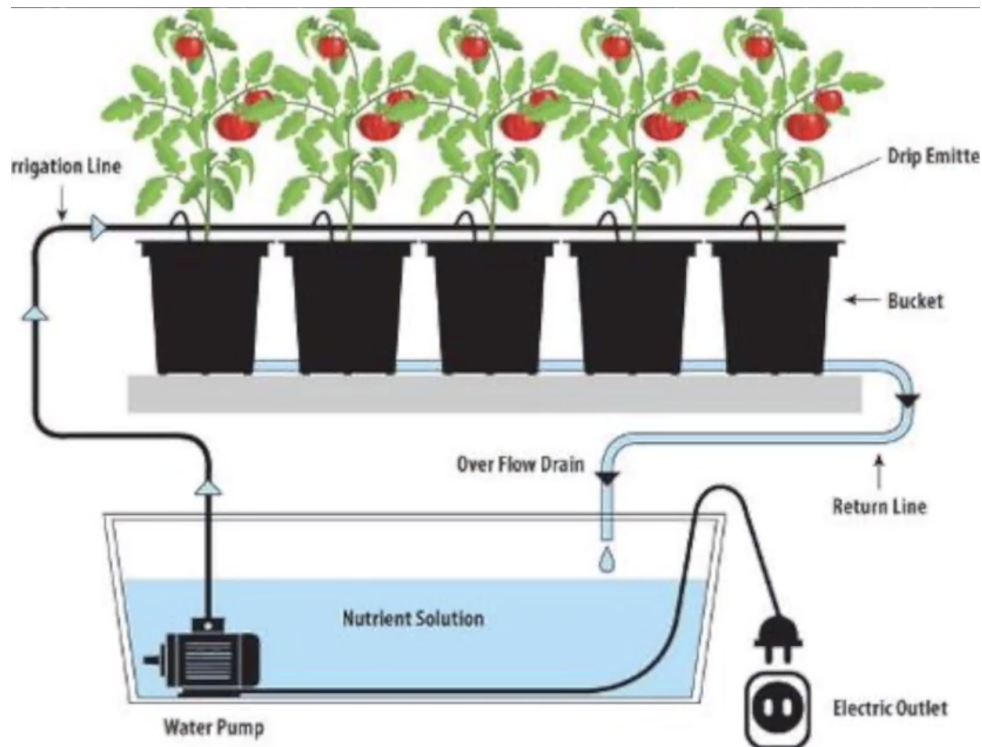


Figure 2.3: Dutch bucket system scheme

2.3 Substrates

Although there is no soil in hydroponic systems, we still need something to keep and hold the plant tight in the pot or any other container depending on the chosen system. This role is played by substrates, which will be covered in this section.

2.3.1 Stones and pebbles

Starting with a substrate which can be found literally on the ground.

Obviously, this is not the best solution since the hygroscopicity of stones is very low. So the nutrient solution must be fed from above.

It will not be enough to just wash the stones or pebbles found on the ground. [6] The substrate needs to be specially prepared, which I will talk about later.



Figure 2.4: Pebble substrate

2.3.2 Expanded clay (ceramsite)

Material that is almost completely free of dust and dirt. There are types that are specifically developed for hydroponics. Despite all of this, it still needs to be prepared.

Expanded clay is not neutral, i.e., it can affect pH level in the nutrient solution. [6] The preparation (neutralization) process is:

- Wash and boil expanded clay
- Soak in distilled water for 2 days
- Measure pH level
- Repeat, if $\text{pH} > 6$



Figure 2.5: Ceramsite substrate

2.3.3 Perlite and Rockwool

Perlite and rockwool are initially neutral materials that don't affect the pH level of nutrient solution. Both have high hygroscopicity. Due to their qualities, they are the most common substrate types. [6]

It is worth noticing that perlite should be carefully cleaned from dust before use.



Figure 2.6: Perlite substrate



Figure 2.7: Rockwool substrate

2.4 Required parameters

Cultivating plants using a hydroponic method is a very complex process. It requires measuring and adjusting many parameters for plants to grow actively and healthy. Starting with parameters of the nutrient solutions (pH level, EC, solution temperature), which I will cover in detail in this section, and finishing with the whole environment (air temperature, humidity, lighting and CO₂ level), which will be mentioned too.

2.4.1 pH Level

pH itself represents an acid-base balance of water. pH level always decreases and increases as a result of the life activity of bacteria. The more H⁺ particles, the more the acid. Otherwise, the more OH⁻, the more alkaline concentration in water. Neutral pH level is 7.0.

pH is required for nutrient assimilation. Incorrect Ph level will reduce or completely lock out the amount of nutrients the plants can absorb.

A suitable level of pH: 5.5 - 6.2. It can be adjusted by adding “pH-Down” (phosphoric or nitric acid) in case of higher level or by adding “pH-Up” (any base) in case of a lower one. [3]

2.4.2 Electrical conductivity (EC)

Electrical conductivity (EC) is the most common measure of salinity, which shows the total amount of dissolved salts.

Pure water without salt impurities has almost infinite electrical resistance. The more salt dissolved in water, the less is its resistance; therefore, by measuring EC, we can have an idea of the amount of salts.

EC affects plants' morphology and the result quality and quantity. The more EC, the more nutrients will plants absorb. Measured in millisiemens [mS].

EC should be around 0.5 - 1.0 mS for sprouts, 0.8 - 1.2 for plants in the active vegetation phase and 1.2 - 1.8 for plants in the flowering phase.

EC can be adjusted by adding pure water in case of higher EC values or by adding fertilizers in case of lower values. [7]

2.4.3 Water temperature

The temperature at the root zone plays a crucial role. It impacts plants' growing speed by affecting the amount of oxygen in the solution. The higher the temperature, the more oxygen there will be.

The problem is, that the higher the temperature is, the more oxygen plants will need; therefore, it is necessary to determine the optimal temperature, which is 18 - 22°C.

A little lower or higher temperature doesn't mean that plants will die, but their growth will slow down massively. [3]

2.4.4 Air temperature

Here I start talking about other parameters associated with the whole environment and won't go into details.

Air temperature adjusts the rate of photosynthesis, breathing and other physiological and biochemical processes.

The optimal value is 22-27°C for a day and 18-22°C for a night. [2]

2.4.5 Air humidity

Air humidity affects plants' final size. The optimal value is around 65 - 75%, depending on the growth phase. [2]

2.4.6 Lighting

Plants are the only organisms that can feed by sunlight, so-called photosynthesis. Therefore common lamps won't be suitable, but lamps, which produce sufficient PAR(Photosynthetically Active Radiation). [2]

2.4.7 CO₂ level

CO₂ is also an element that is needed for plant feeding, since it is a part of photosynthesis. [2]

3. Hardware analysis

This chapter is devoted to analyzing and describing all the devices and hardware used in the project. Communication between them will also be mentioned.

3.1 General scheme

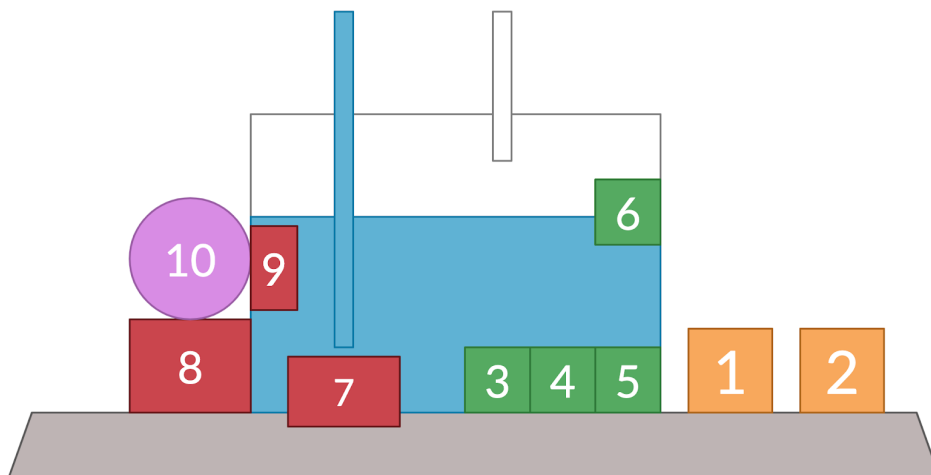


Figure 3.1: Entire system's general scratch scheme

1. **Arduino Uno** - microcontroller responsible for communication with RaspberryPi, gathering sensors data and sending commands to actuators.
2. **RaspberryPi** - microcomputer responsible for data processing and storing them to the databases.
3. **Water temperature sensor** - measuring temperature in °C.
4. **EC sensor** - measures electrical conductivity in milisiemens.
5. **pH sensor** - measures pH level.
6. **Water level sensor** - floating sensors that indicate if they are touched by water.

7. **Feeding pump** - pump circulating the solution around the system.
8. **Dosing pumps** - doses fertilizers,ph-Ups/ph-Downs and pure water into the solution while turned on.
9. **Water cooler** - reduces water temperature.
10. **Containers** - contain fertilizer solutions, ph-Ups/ph-Downs and pure water.

3.2 Required components

In this section I will describe all components mentioned at Figure (3.1).

3.2.1 Computing power

3.2.1.1 Arduino Uno

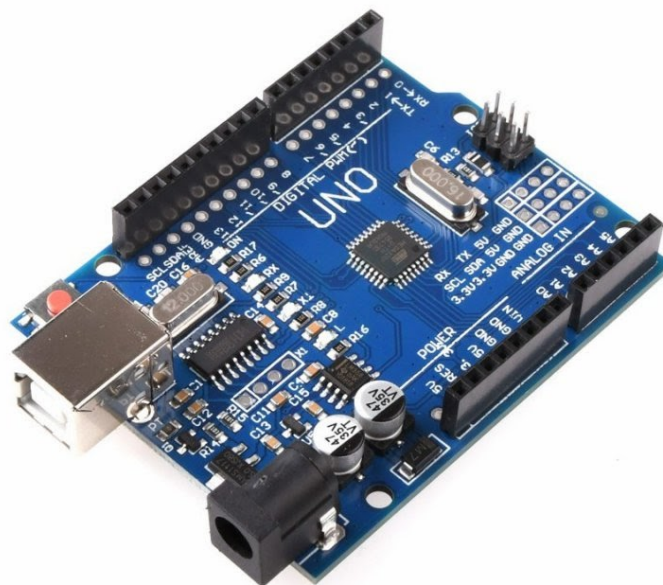


Figure 3.2: Arduino Uno microcontroller

The most famous microcontroller. Due to low CPU resources and small memory (about 2 KB) is not really suitable for big programs and hard computations. [8] Lack of the ability to connect to the internet without additional shields is its next disadvantage.

But on the other hand due to lower power consumption and a lack of operating system (therefore, low maintenance) Arduino Uno is really good at real-time processing. [9] In combination with the fact that it has good support for analog signals [10], Arduino Uno is a good choice for measuring sensors' data, sending it for computation via serial link, receiving responses and sending commands to actuators (relays).

3.2.1.2 Raspberry Pi

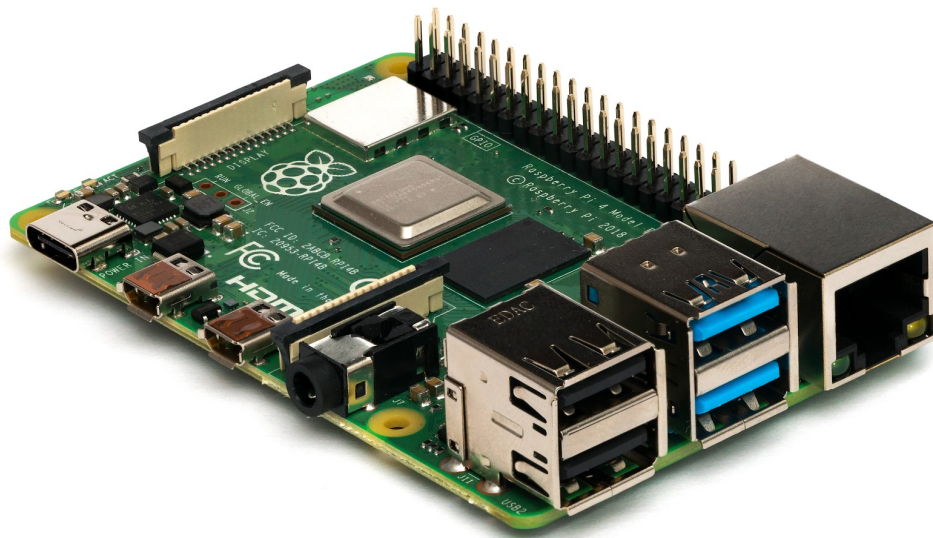


Figure 3.3: RaspberryPi microcomputer

RaspberryPi is the most famous microcomputer with high CPU resources, memory, internet connection and other advantages. [11]

Due to its operating system (which does not grant us control over resources) and a bad support for analog signals (reading analog sensors requires extra hardware assistance) RaspberryPi is not really suitable for work with sensors, which the project requires. [12]

Otherwise, it may serve as a normal computer running Raspbian (Linux-like) OS. Raspberry's OS allows to run programs as services, i.e. to work with them semi-automatically by setting boot and other options. Therefore, it perfectly suits for processing of data received from Arduino.

3.2.1.3 Summarizing


In the following table I will summarize each computer's pros and cons. The information in the table repeats previous sections and serves for better lucidity.





Table 3.1: Summarizing computer devices

	Arduino Uno	Raspberry Pi
System	Microcontroller	Single-Board computer
Operating System	Has no operating system	Runs an operating system
Dynamic memory	2 kb	Up to 8 gb
Resources control	Full resources control because of lack of the OS	OS doesn't grant full resources control
Internet connection	Requires external HW and additional code	Can be easily connected via Wi-Fi or Ethernet
Measuring analog data	Native support	Requires external HW and additional libraries
Program execution	Can run only one program loop	Can run multiple programs in parallel
Computing power	Rather low	High
Data-Transfer protocols	I2C, UART	UART, Ethernet/Wi-Fi
Real-Time processing	Suitable due to lack of the OS	Not really suitable because of a lower resources control

3.2.2 Sensors

Table 3.2: Required sensors

Name	Photo	Description
ISE Probe interface		Reading data from pH probe

Lab pH probe		Measuring pH level
EC Probe interface		Reading data from EC probe
Lab EC probe		Measuring EC
DS18B20		Measuring water temperature

<p>HC-SR04</p>		<p>Measuring water level</p>
----------------	--	------------------------------

3.2.3 Actuators

Table 3.3: Required actuators

Name	Photo	Description
<p>Dosing pump</p>		<p>Dosing of fertilizers, ph-Ups/ph-Downs and water</p>
<p>Relay</p>		<p>Opening and closing circuits for turning the actuators on and off</p>

<p>Water cooler</p>		<p>Reduces water temperature</p>
----------------------------	---	----------------------------------

3.3 Communication

Most of the sensors are connected to Arduino Uno via I2C interface (will be described in more details later), the remaining ones and actuators are connected directly to its analog or digital pins.

Therefore, Arduino does all the work with them: it collects data from the sensors and constantly sends it to RaspberryPi via serial link. Arduino also checks its serial port for messages from RaspberryPi, i.e. commands for actuators, which are represented by relays. Then sends these commands directly to them.

RaspberryPi is responsible for processing data gathered from Arduino, deciding if parameters' values are beyond required and sending commands back to it via serial link. RaspberryPi is also responsible for storing data in local and cloud databases for further use.

Data flow is described at Figure (3.4).

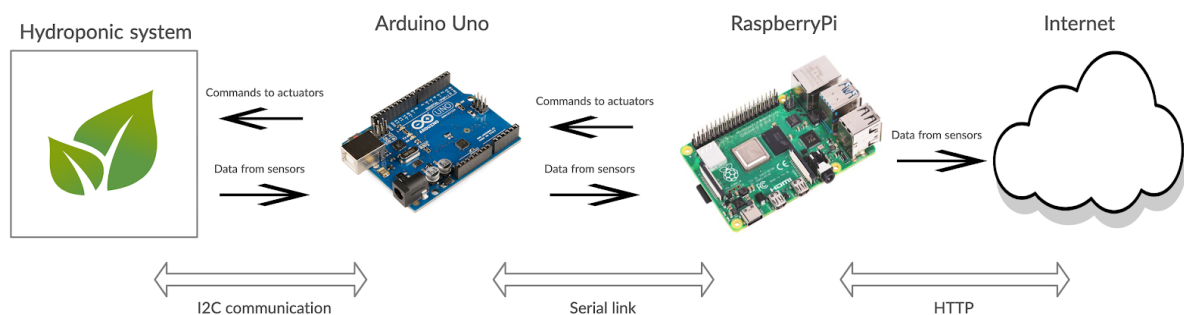


Figure 3.4: System communication scratch scheme

4. Used technologies

4.1 Data storage

Each system requires reliable data storage. When it comes to the system which is responsible for life support, the reliability of its data must be even greater. It also must be flexible and scalable. The project requires cloud data storage data to be accessible from the internet. But since we can not rely on the internet connection, the project needs to have another one local storage.

Nowadays when it comes to deciding, there is a question: relative (SQL) or non-relative (No-SQL) database. This section goes through it.

4.1.1 SQL vs NoSQL

Relative database is an old classic. Static tables with predefined columns and powerful SQL language for querying data. Is really suitable for storing highly structured data, but is not when it comes to flexible systems such as hydroponics. Variable data and data structures generated by the hydroponic system lead us to No-SQL databases.

Table 4.1: Comparison of SQL and No-SQL databases

	SQL	No-SQL
Type	Relational	Non-relational
Data	Structured. Stored in tables	Unstructured. Stored in JSON
Schema	Static	Dynamic
Scalability	Vertical	Horizontal
Flexibility	Rigit schema	Non-rigit flexible schema
Software	Oracle, PostgreSQL, MySQL	DynamoDB, MongoDB, RedisDB
Elasticity	Mostly requires downtime	Automatic. No downtime required

4.1.2 Dynamo Database

DynamoDB is a part of Amazon Web Services. It is a No-SQL database for applications that require strong consistency and low latency (under 10ms) at any scale. It is also fully managed and supports both document and key-value store models.

DynamoDB gives the ability to auto-scale by tracking how close the usage is to the upper bounds (auto-sharding) and the ability to work with really large amounts of data (thousands of reads or writes per second).

The database is multi-regional, which enables the maintenance of identical copies as replicas of a DynamoDB master table in one or more regions. [13]

Since DynamoDb supports a document-oriented data model, we will store data as “items”, which correspond to “rows” in SQL. Each item will have the following attributes: ”system_id”, ”sensor_type”, ”sensor_name”, ”value” and ”timestamp”. Since attributes are dynamic, there may be new ones in the future.

The project’s database will contain 1 table with sensors’ data with a composite primary key defined as ”system_id” + ”timestamp”.

Alternatives:

- **Any SQL database.** Is not as flexible as DynamoDb, because they work only with structured data and predefined tables. Also, they are not as performant as No-SQL when it comes to processing large amounts of data. [14]
- **MongoDB.** Good candidate with similar benefits as DynamoDB. But it is less scalable (since it doesn’t support auto-sharding and multi-shard transactions) and is not very low latency. [15]

4.1.3 Redis Database

Provides almost equal functionality as Dynamo Database, but has advantages, which made it the best decision to be the local data storage:

- Free to use offline.
- Provides publish/subscribe channels which allow simulating subscription to the table, which we need for communication between applications.
- Not only stores data in heap but in some other data structures, such as list and sorted set.

Alternatives:

- **Same as DynamoDB's.** Plus disadvantage of not having publish/subscribe or easy-to-implement table subscription.

4.2 Programming languages

4.2.1 Arduino

Arduino itself has native support for a language that is called the same (“Arduino programming language”). It is basically a framework built on top of C/C++, therefore follows its logic and syntax. [16]

The language also provides prepared libraries with functions suitable for the project. Most modern sensors provide libraries to work with Arduino. So mainly will be used these libraries and also one called “Wire” serving for I2C communication[17]. Also often used are functions as `analogRead()` for reading analog data from sensors that don't provide libraries and `digitalWrite()` for sending electric signals to actuators.

Arduino programs (due to their simplicity also called scratches) are saved with the `.ino` extension and their main difference from programs written in C/C++ is that the whole code is wrapped into 2 main functions: `setUp()`, where there is a common initialization, and `loop()`, which is, obviously, a loop containing entire program's logic.

Alternatives:

- **None.** Since Arduino Programming Language is built on top of C/C++, we could use pure versions of these languages, which is unnecessary because Arduino provides built-in explicit extensions to control its hardware.

4.2.2 Python

Python nowadays is a very popular and widespread high-level programming language. According to the TIOBE programming community index is ranked second. [18]

The language kernel's syntax is minimalistic, but in the meantime, its Standart library provides many useful functions. one of the main reasons why Python is so popular and why it was chosen for the project is its variety of external libraries, some of which will be used in the project: “serial” for communication with Arduino Uno via UART protocol [19], “boto3” for accessing Amazon's DynamoDB [20] and ”redis” for accessing RedisDb.

Python is an interpreted language, i.e. it doesn't compile, but interprets line by line. Which makes it slower than some other languages (Java, C/C++). But despite this, it is used in such Real-Time applications as Instagram, DropBox, Reddit and others. [21]

Python is also great for working with AI, which is its next benefit for the project, as the project has the potential to use AI logic to work with actuators. [22]

Alternatives:

- **Java.** Also a popular and widely used language. Since it is a compiled language, Java is fast and performant. Also provides many useful libraries. But on the other hand, libraries for communication via Serial link and working with DynamoDB are subjectively not as convenient to work with as Python's.

The second disadvantage is not good support for AI. [23]

4.2.3 Summarize

The following table doesn't compare the languages but summarizes their advantages and reasons why they were chosen for the project.

Table 4.2: Summarizing programming languages

Arduino Language	Python
Built on top of C/C++	One of the most popular languages
The only opportunity to write a program on Arduino	The standard library provides many useful functions
Provides in-built functionality to read analog signals and send electrical	Provides many useful libraries
Provides in-built functionality to read and write to/from a serial link	Supports DynamoDB and RedisDB well
Provides a library for working with an I2C bus	Supports serial communication well
	Great for working with AI

4.3 Microservice Architecture

Microservices is a modern architecture that gradually takes on trends and replaces the previous most popular and widespread architecture, i.e. monolith.

Microservice architecture represents independent components, each doing its job (Separation Of Concerns) and communicating with each other. A system built on microservices loses a lot of software's critical point: Single Point Of Failure. Therefore it is a flexible

architecture that perfectly suits the project's purposes.

Communication between services will be set up via publish/subscribe messaging channels.

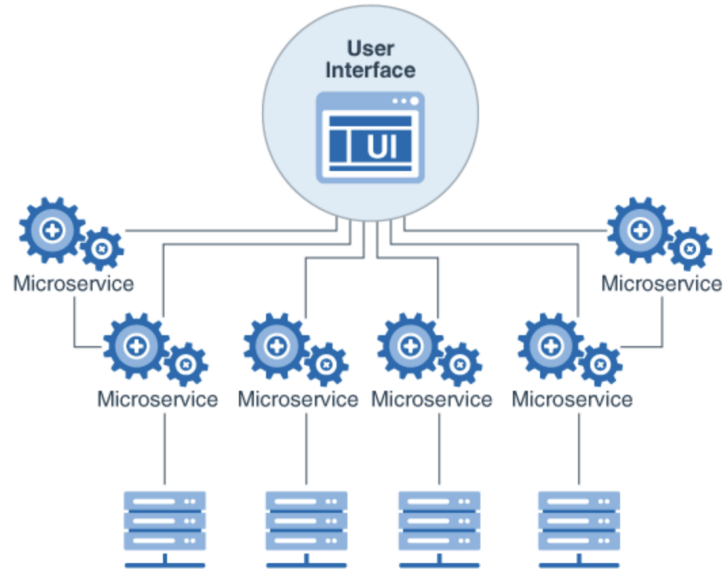


Figure 4.1: Microservice architecture scratch scheme

Alternatives:

- **Monolith.** Old fashioned style of development. All the software is managed in one place: databases, client- and server applications. That leads to many bottlenecks and points of failure, reduces system flexibility. Which is unacceptable for the project.

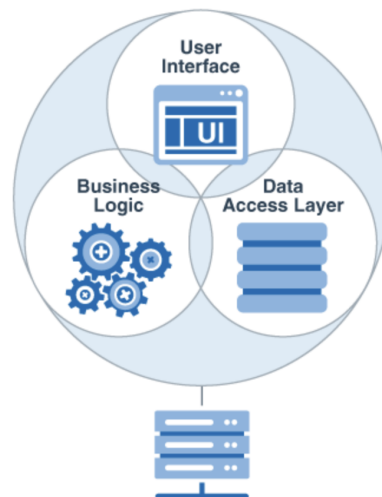


Figure 4.2: Monolith architecture scratch scheme

The following table describes the main differences between the 2 architectures mentioned above.

Table 4.3: Comparison of Microservice and Monolith architectures

	Monolith	Microservices
Deployment	Large code base mostly deployed on one device	Multiple independent applications. It May be deployed on different devices
Scalability	Bad scalability, since the project is deployed on one device which has hardware limitations	Easy to scale
Flexibility	Bad flexibility. Since one small change may cause an avalanche effect of changes	Good flexibility because of a Separation Of Concerns
Data Storage	Shared storage	Each service mostly has own storage
SPOF	A big problem	Almost eliminates the SPOF
Reusage	Mostly is very difficult	Services can be easily reused

4.4 Communication protocols

Since there are 2 communication pairs in the project (Sensors/Actuator <> Arduino Uno and Arduino Uno <> Raspberry Pi), it is necessary to define the protocols that will be used for their communication.

The only requirement for the protocols is for them to be wired to keep the system immune to such problems as the internet- or wireless signal loss.

4.4.1 I2C

Inter-Integrated Circuit is a synchronous serial protocol for data transfer designed for microcontrollers. The absolute majority of modern sensors do communicate over this protocol. It requires only 2 wires to work: first for sending and receiving data and the second that carries the clock signal for synchronizing.

Protocol's Master-Slave communication allows us to connect up to 128 devices (sensors in our case), which makes it perfect for the project. [24]

Alternatives:

- **Serial Peripheral Interface (SPI).** Another communication protocol designed for microcontrollers. Mostly used in special sensors or other devices when low response time is required. Unlike I2C doesn't allow to create buses of up to 128 devices, i.e. supports only 1 master device and 4 slaves.
- **Direct connection.** Will certainly be used in the project to connect less complex sensors and relays directly to the Arduino Uno.

4.4.2 UART Protocol

Universal asynchronous receiver-transmitter is one of the oldest and commonly used physical data transfer protocols, which uses asynchronous serial communication.

Asynchronous stands for no clock signal to synchronize the output bits from the transmitting device going to the receiving end.

Arduino Uno and Raspberry Pi will be connected via USB, since Arduino has an in-built converter from USB to Serial which RaspberryPi can handle well. [25]

Alternatives:

- **Binary protocol.** Unnecessarily difficult to implement since Arduino Uno and RaspberryPi may be easily connected via USB using UART protocol to communicate. Therefore, is not taken into account.

5. Software analysis

This chapter is devoted to complete project software analysis. It includes analysis of state around hydroponics from a business point of view, diagrams describing projects from a technical point of view and analysis of technologies used in the project and their alternatives.

5.1 AS-IS state

Nowadays hydroponics is still not a widespread technology; therefore, many people treat and use it the old way.

They have to keep in mind a value for a special parameter for each growth phase of every single plant. During the plant's growth gardeners have to constantly measure all these parameters with common instruments, such as thermometer, EC - meter and so on. Manipulations with the solution in order to change the parameter's values are also made manually. Which is long, ineffective and leads to mistakes, many of which can completely ruin the solution.

Industrial hydroponics is mostly widespread in the USA and UK, where its market sizes are around 2.000.0000.000\$, production sizes are around 22.500.000 kg a year and will grow further in the future. [26]

5.2 Strategic intent

The intent of the project is to create a system, which provides a partial automatization of the plants' cultivating process using hydroponic systems.

The user will be able to efficiently and reliably keep track of information about the state of the hydroponic system's solution and to set up its parameters, depending on the plants' growth phase or a gardener's will.

The system, in turn, is responsible for keeping the parameters at given values and storing gathered data to the databases.

5.3 Business intent

The system will make gardeners sure that their systems work just as required and at the same time will relieve them of much routine work allowing them to concentrate less on maintaining the farms and giving them an opportunity to further expand them.

Also making it easier to work with hydroponics systems will bring new people, which will have a good effect on gardening progress itself and the whole world's food availability.

5.4 5F analysis

- **Rivalry**

- Rather small amount of offers with similar functionalities.
- In majority automated hydroponic systems are made for industrial plant cultivation and are very expensive. [21]
- Systems for common use are rare on the market. At most are handmade and low-quality.

- **Potential entrants**

- Entry threshold to IT business is low.
- Required knowledge of plant cultivation and hydroponics systems, which is not hard to gain.
- Cheap hardware and not really complex software may bring many entrants.
- Ready for the appearance of new entrants.

- **Suppliers**

- Hardware suppliers.
- Fertilizers suppliers.
- Plant seeds suppliers.
- Many offers in different price categories. Substitute inputs are present.

- **Buyers**

- The practice of hydroponics is expanding.
- Information about hydroponics is becoming easier to get.
- Every social category is in the target group.

- **Substitutes**

- Substitutes' appearance possibility is real.

5.5 PEST analysis

- **Political**

- Situation is stable.

- **Economical**

- Situation is unstable due to coronavirus.
- Hardware prices grow due to a deficit of components and production facilities.

- **Social**

- Situation is unstable due to coronavirus.
- The situation is hard to predict.

- **Technological**

- Situation is stable. We are using the technologies, which are considered to grow in terms of usage in the future:
 - * **Arduino** is an in-built programming language. Widely used and supported, since it is the only way to work with Arduino.
 - * **Python** programming language. One of the most popular nowadays. Well supported and provides a large number of libraries, which are suitable for the project.
 - * **DynamoDB** is a No-SQL database powered by Amazon, one of the world's richest and most stable companies. It is fully managed, multi-region and widely used, which suits the project's purposes well.
 - * **RedisDB** is a No-SQL database powered by RedisLabs. Is also widely used, and well supported.

5.6 Functional requirements

Table 5.1: Functional requirements

ID	Description	Priority
1	System allows to gather data from sensors	High
2	System allows to send data for computation	High
3	System allows to evaluate the data	High
4	System allows to store data to local and cloud databases	High
5	System allows to send commands to actuators	High
6	System allows users to set up parameters	High
7	System allows users to change parameters while system is up	Medium
8	System allows to retrieve data about solution's state	Low

5.7 Non-functional requirements

- The system must be independent of internet connection.
- The system must be available 24/7/365.
- The system must be highly performant, e.g. the time between sending a command to an actuator and actual turning on must be within a second.
- The system must be scalable. This applies both for HW (adding new sensors and actuators) and SW (simple implementation of new functionalities)

5.8 Communication between Services

Since Microservice Architecture implies separate and independent components, it is necessary to find a way of communication between them. Publish/Subscribe message

channels provided by Redis Database are a good choice for it.

Publish/Subscribe pattern allows publishers to send their messages to an arbitrary number of receivers. They, in turn, may subscribe to an arbitrary number of channels and receive messages from publishers.

For the project we need 2 channels: for notifying about new data and for sending commands to actuators.

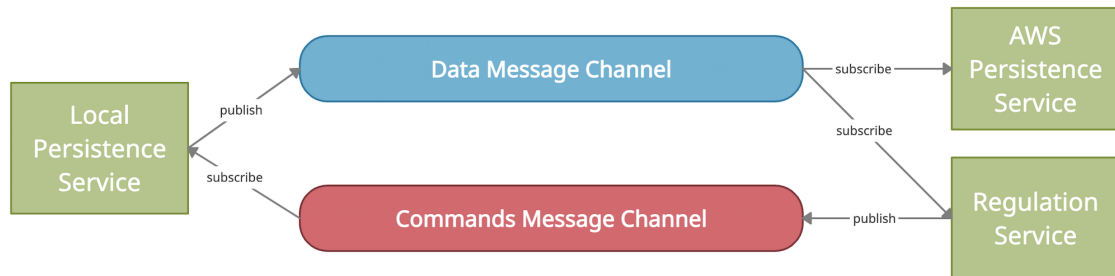


Figure 5.1: Communication using channels

5.9 Sequential diagrams

The following diagrams describe in a more formal (UML) way the communication process mentioned in chapter 3.3 and described informally at Figure (3.4).

5.9.1 Arduino application

This diagram describes the app running on Arduino Uno which is connected to sensors, actuators and RaspberryPi. At startup the app sends a request for device data stored on Raspberry Pi. After receiving a response and setting devices up, the main program loop starts.

There is another loop inside the main one. It iterates through sensors and, if required time passed, reads data from them and sends it to RaspberryPi via serial link.

Every iteration app checks its serial port for commands which continuously come from RaspberryPi.

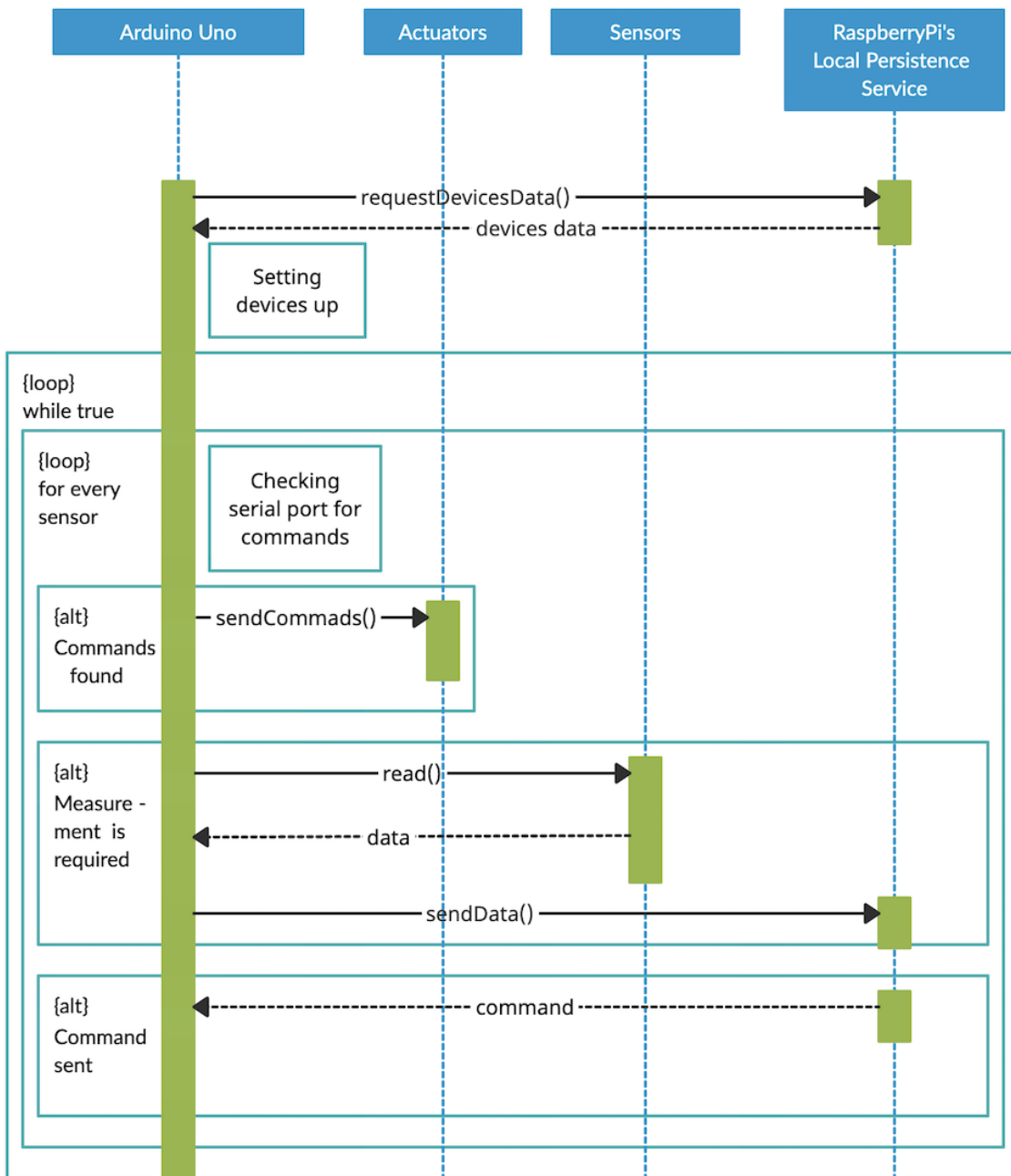


Figure 5.2: Arduino application sequential diagram

5.9.2 Raspberry Pi Local Persistence Service

This diagram describes the first of 3 apps running on Raspberry Pi. It is responsible for gathering data from Arduino Uno, storing it to the local Redis Database and sending commands back to Arduino.

At startup the app sets the connection to the database up. Since Raspberry ap-

plications do communicate using message channels provided by Redis, so the app also subscribes to the commands message channel .

There are two phases in the main loop: the first checks the command channel for new commands and sends them to the Arduino Uno in case of presence; the second checks the serial port where messages from Arduino constantly arrive.

In case of a data request message, the app will send back configuration device data which is stored internally, in case of a sensors data message, the app will store it to the local Redis database and send a message to the data message channel, notifying all the channel subscribers about the newly stored data.

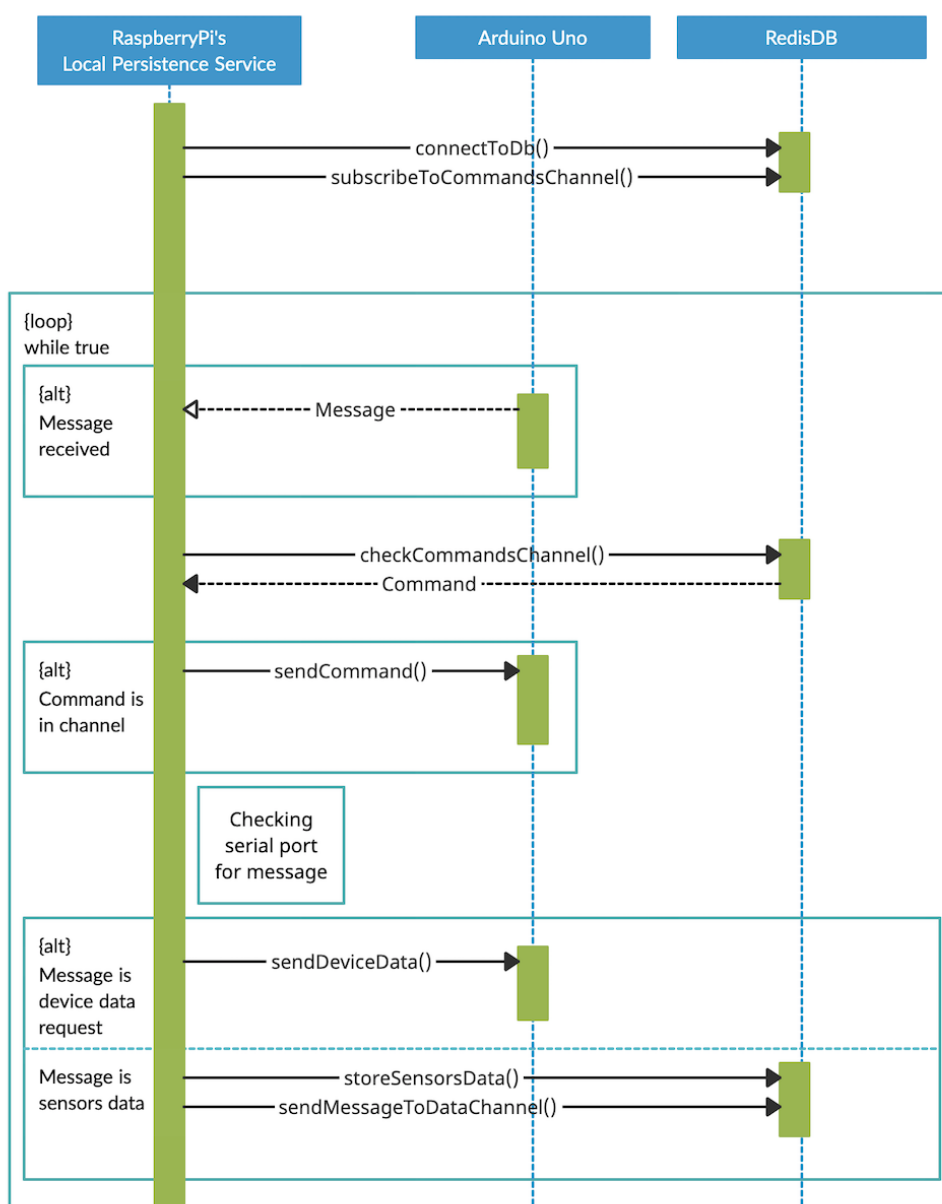


Figure 5.3: Local persistence service sequential diagram

5.9.3 Raspberry Pi AWS Persistence Service

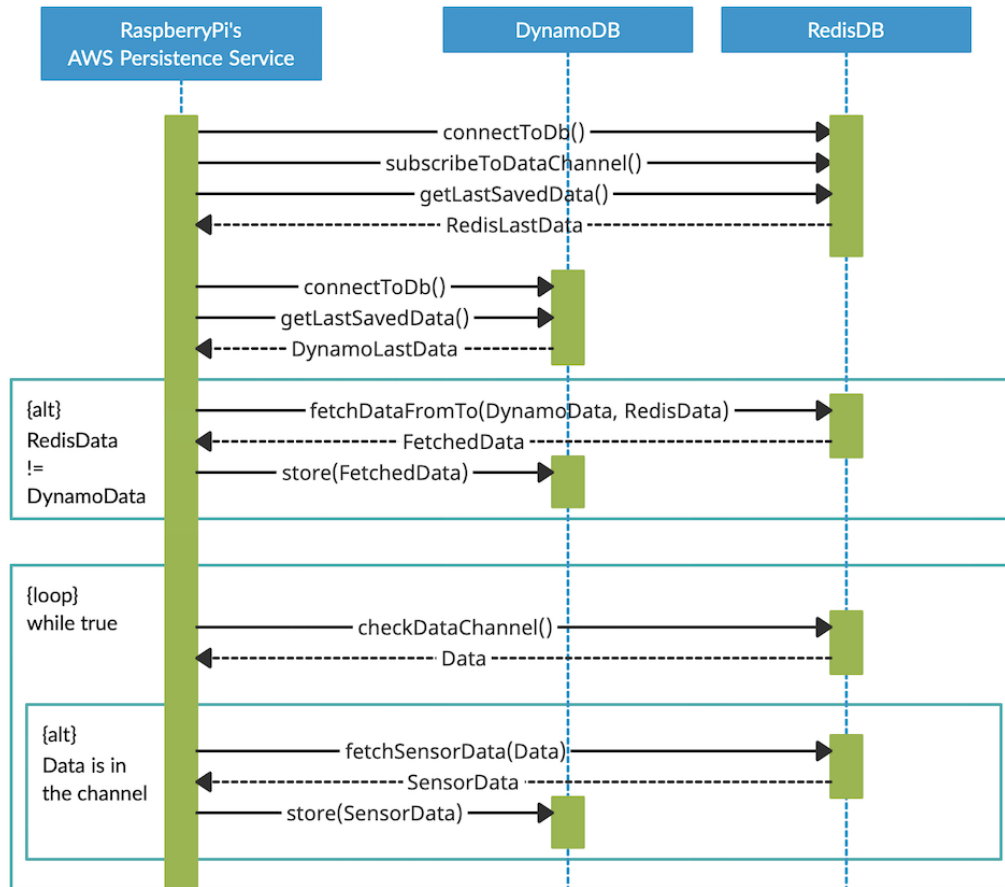


Figure 5.4: AWS persistence service sequential diagram

The following diagram describes the second of 3 apps running on Raspberry Pi. This one is responsible for storing gathered data to the cloud Dynamo Database.

The main goal of the application is to keep data consistent. To that end, except for connecting to the databases and subscribing to the data message channel, the app will also fetch the last data saved locally (from the RedisDb) and in the cloud (DynamoDb). In case of data inequality (meaning the data, which were saved locally were not saved in the cloud for some reason), missing data will be fetched from RedisDb and stored to the DynamoDb. There is also a failed connection handling mechanism which will be described in Chapter 6.

In the main loop app checks the data message channel, which notifies about new locally stored data. It will fetch new data and store it in the cloud.

5.9.4 Raspberry Pi Regulation Application

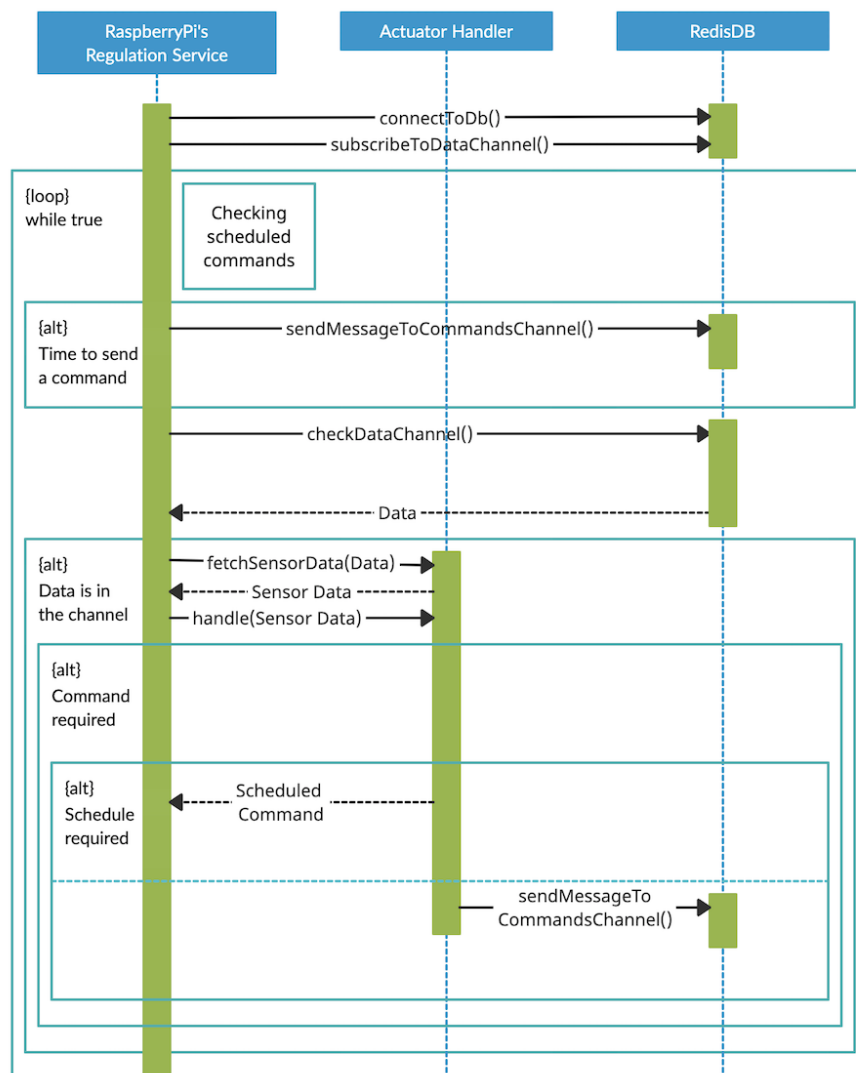


Figure 5.5: Regulation service sequential diagram

The last of 3 apps running on Raspberry Pi is responsible for regulating hydroponic solution parameters based on required values by sending commands for actuators to turn up or down. This is an app where all the interesting logic happens which will be described in more details in Chapter 4.

Like the other 2 apps, this one connects to the RedisDb and subscribes to the data message channel at startup.

The main loop consists of 2 phases: during the first one the app checks its list of scheduled commands and, if it is the time, sends a command to the command message channel; during the second one the app checks the data message channel and in case of new data presented, it fetches it and sends to the associated handler (depending on

parameter type: temperature, pH-level, etc.). Handler decides, if the value is beyond required and, if necessary, send a message to the commands message channel.

Some actuators require to be turned off after the required amount of time (so-called scheduled commands). In that case this information will be also saved to the local list of scheduled commands.

5.10 Deployment diagram

The following diagram shows the physical deployment of the system.

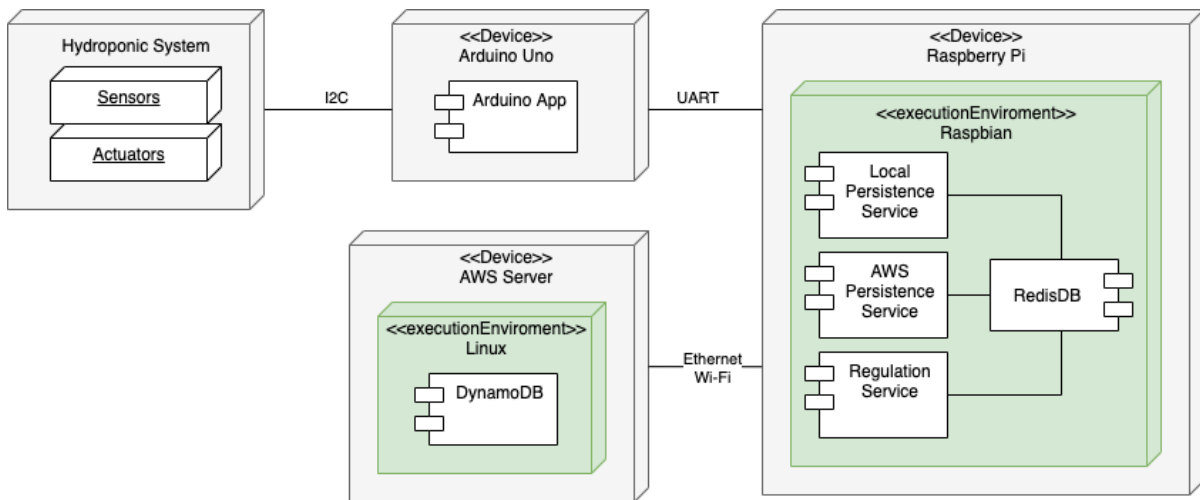


Figure 5.6: System communication scratch scheme

- **Hydroponics System**

All the sensors and actuators are physically located inside of a hydroponic solution tank or near it and connected to the Arduino Uno via I2C.

- **Arduino Uno**

The microcontroller that runs the script gathering data from the sensors, sending it to the Raspberry Pi and sending commands to the actuators. Is connected to the Raspberry Pi via UART.

- **Raspberry Pi**

Main system computer. Runs 3 applications described above and a local Redis Database.

- **AWS Server**

An external server that runs Dynamo Database. Raspberry Pi is connected to it via Ethernet or Wi-Fi.

6. Implementation

In this chapter I move on to the practical part of the project. The chapter describes each of the 4 project's applications and their programmatic solutions.

6.1 Arduino Uno Application

As mentioned in previous chapters, arduino application consists of 2 parts:

- `setup()`

Listing 6.1: Setup Function

```
char* device_data[POSSIBLE_NUMBER_OF_DEVICES][CONFIG_ENTRIES];

void setup() {
    Serial.begin(9600);
    Wire.begin();
    fetch_config_data();
    setup_devices();
}
```

The function that is called at the application startup. It starts serial communication and sets the Wire library up for work with I2C devices.

Then the "HELLO" message is sent and device data are fetched from Raspberry Pi with a response. It is stored in the `device_data` variable which is a double-dimensional list.

The response represents lines of config entries for every device. The format is the following:

<code>super_sensor</code>	<code>temperature_sensor</code>	<code>0x63</code>	<code>120</code>
Sensor name	Sensor type	Address	Tempo time

Figure 6.1: Device configuration format

Tempo time stands for the amount of time in seconds that must pass after the last reading before sensors can send new data. Tempo time is 0 for actuators.

- **loop()**

Listing 6.2: Setup Function

```

void loop() {
  for (int i = 0; i < number_of_sensors; i++){
    check_for_actuator_commands();

    const char* sensor_name = device_data[i][0];
    const char* sensor_type = device_data[i][1];
    int device_address = int(atoi(device_data[i][2]));
    unsigned long sensor_tempo = int(atoi(device_data[i][3]));
    float sensor_value = 0;
    bool measuring_required = time_passed(last_sensor_timestamps[i], sensor_tempo);

    if (!measuring_required) continue;
    if (!strcmp(sensor_type, PH_S)) {
      sensor_value = pH_lib.measurepH(mv_lib.measureTemp());
    }
    else if (!strcmp(sensor_type, EC_S)) {
      sensor_value = ec_lib.measureEC(mv_lib.measureTemp());
    }
    else if (!strcmp(sensor_type, TEMP_S)) {
      sensor_value = mv_lib.measureTemp();
    }
    else if (!strcmp(sensor_type, MID_WATERLEVEL_S) || !strcmp(sensor_type,
      LOW_WATERLEVEL_S) || !strcmp(sensor_type, UP_WATERLEVEL_S)) {
      sensor_value = (digitalRead(device_address) == HIGH) ? 1 : 0;
    }
    else if (!strcmp(sensor_type, FLOW_S)) {
      sensor_value = getFlow(device_address);
    }
    last_sensor_timestamps[i] = millis();
    send_data(sensor_name, sensor_type, sensor_value);
  }
}

```

The function where application constantly iterates through the sensors and, if

required time passed, read its data and sends this information to the RaspberryPi. Since Arduino Uno has a really small amount of memory, the message format is the following:

```
"sensors_data super_sensor temperature_sensor 23.75"
           Sensor name      Sensor type      Value
```

Figure 6.2: Sensor data message format

- Another function which is worth to mention is `check_for_actuator_commands()` which is called every iteration.

Listing 6.3: Setup Function

```
void check_for_actuator_commands() {
  while (Serial.available()) {
    String act_type = Serial.readStringUntil('\n');
    String act_command_value = Serial.readStringUntil('\n');

    for (int i = 0; i < POSSIBLE_NUMBER_OF_DEVICES; i++){
      if (!strcmp(act_type.c_str(), device_data[i][1])) {
        if (act_command_value.equals("0")) {
          digitalWrite(int(atoi(device_data[i][2])), HIGH);
        } else if (act_command_value.equals("1")) {
          digitalWrite(int(atoi(device_data[i][2])), LOW);
        }
      }
    }
  }
}
```

The function checks Arduino's serial port and reads all the messages (commands) of the following format:

```
"temperature_actuator 1"
           Actuator type      Command
                               value
```

Figure 6.3: Command message format

Command value equal to 1 stands for turning the actuator up, 0 - for turning it down.

6.2 Local persistence service

As mentioned in previous chapters, this is the program responsible for the communication with Arduino Uno, storing its data to the local Redis Database and notifying others about the new data by sending messages to the Redis' Data Message Channel.

- 1st phase

Listing 6.4: Serial Link Handling

```
def check_act_command():
    sub_message = redis_sub.get_message() #Checking Command
    Message Channel
    if sub_message:
        command = sub_message['data']
        [act_type, act_command_value] = command.split()
        ser.write((act_type + "\n" + act_command_value +
                  "\n").encode("ascii"))
        ser.flush()
        logger.info('Command sent to %s with value %s',
                    act_type, act_command_value)
```

At the beginning of every iteration application checks Redis' Command Message Channel and if there is one, the app sends it to the Arduino Uno.

- 2nd phase

Listing 6.5: Serial Link Handling

```
if ser.in_waiting > 0:
    line = ser.readline().decode('utf-8')
    logger.info('received: ' + line)

    if line == "HELLO":
        logger.info('Hello message received')
        file = open(constants.CONFIG_FILE_NAME)
        config_data = file.read().replace("\n", " ")
        file.close()

        ser.write(config_data.encode("ascii"))
        ser.flush()

    elif line.startswith("sensor_data"):
        measured_data = line.split()
```

```

sensor_data = {
    "system_id": constants.SYSTEM_ID,
    "sensor_name": measured_data[1],
    "sensor_type": measured_data[2],
    "value": measured_data[3],
    "timestamp": datetime.now().strftime("%Y/%m/%d
        %H:%M:%S")
}

save_data(sensor_data)

```

The application checks its serial port and if there is a "HELLO" message (device data request), it will send the data from the associated config file. Otherwise, if the message represents sensors data, it will be saved to the Redis Database.

- `save_data()`

Listing 6.6: Serial Link Handling

```

def save_data(sensor_data):
    timestamp = sensor_data["timestamp"]
    redis_client.hset(timestamp, mapping=sensor_data)
    redis_client.rpush(sensor_data["sensor_name"] + "_timestamps",
        timestamp)
    redis_client.rpush(constants.TIMESTAMPS_LIST_NAME, timestamp)
    new_timestamp_id =
        redis_client.zadd(constants.TIMESTAMPS_FOR_IDS_SORTED_SET_NAME,
            timestamp) # Sorted set which increases IDs of inserted items
    redis_client.publish(constants.PUB_SUB_SENSORS_CHANNEL_NAME,
        timestamp + " " + str(new_timestamp_id))

```

- Since sensors data are to be read sequentially, each timestamp will be unique, which makes it a perfect key.
- Redis provides 3 data structures: the heap where all objects are stored, list and sorted set. We use every one of them to be able to work with data in a No-SQL database.
- After saving the data app sends the timestamp and its id to the Data Message Channel so other services can query the full object from the heap.

6.3 AWS Persistence Service

This is a simple service responsible for storing gathered data in the cloud. Its goal is to keep data consistent. There are two mechanisms helping to achieve this goal: the first one runs on the very startup and was well described on the sequential diagram and won't be described in this chapter. The second one is the Connection Loss Handling Mechanism.

Listing 6.7: Serial Link Handling

```

sub_message = redis_handler.redis_sub.get_message()
if sub_message:
    timestamp, current_id = utils.split_timestamp_and_id(sub_message)
    sensor_data = redis_handler.redis_client.hgetall(timestamp)

    is_saved = save_item(sensor_data)

    if not is_saved and not disconnected:
        logger.warning('Internet connection is lost'.)
        disconnected = True
        first_unsuccessful_id = current_id
    elif is_saved and disconnected:
        logger.info('Internet connection has been reestablished.')
```

```

        disconnected = False
        last_unsuccessful_id = current_id - 1
        unsent_timestamps =
            redis_handler.redis_client.zrangebyscore(min=first_unsuccessful_id,
            max=last_unsuccessful_id)
        save_all(unsent_timestamps)
```

Every iteration application checks Data Message Channels and if there is a new message (new data's timestamp and id) it will query the whole object from Redis and store it to the Dynamo Database. In case of internet loss `save_item()` method returns False and the algorithm begins:

- If the connection had just been lost, i.e. it was the first data that failed to save, app will set its id as first unsuccessful.
- If the connection had been lost a while ago, i.e. first unsuccessful id is already set, nothing happens.

- When an item is saved and service was disconnected, i.e. the connection reestablished, the app will fetch unsaved data (from first unsuccessful id up to current) from Redis Database and save them all together.
- Mechanism is also present in the `save_all()` method.

Listing 6.8: Serial Link Handling

```
def save_all(timestamps_for_ids):
    for unsaved_redis_ts in timestamps_for_ids:
        unsent_sensor_data =
            redis_handler.redis_client.hgetall(unsaved_redis_ts)
        is_saved = save_item(unsent_sensor_data)
        if not is_saved:
            disconnected = True
            first_unsuccessful_id = unsaved_redis_ts
            logger.warning('Internet connection is lost')
            break
```

6.4 Regulation Service

Like other applications, this one establishes the connection to the Redis Database and subscribes to the Data Message Channel at startup.

At the beginning of every iteration application checks scheduled commands (will be described later) and modifications of associated configuration files. In case of modification new data are saved. Configurations files are:

- **Coefs.json**

These are the coefficients that will be used in formulas for regulations. `phUp`-, `phDown`- and `ecUp`- coefficients stand for the amount of a solution in milliliters required to change the required parameter by 1 in 1 liter of hydroponic solution.

```

"tankVolume": 60, [L]
"phUpCoef": 5, [ml/L]
"phDownCoef": 0.125, [ml/L]
"ecUpCoef": 5, [ml/L]
"ecToPpm": 700, [Parts Per Mln]
"ecDownCoef": 4, [ml/L]
"dosPumpCoef": 60 [ml/s]

```

Figure 6.4: Regulation coefficients

- Rules.json

```

"temp": {
  "required": 22,
  "tolerance": 2,
  "sensor_type": "temperature_sensor",
  "act_type": "temperature_actuator"
},
"ec": {
  "required": 1.4,
  "tolerance": 0.1,
  "stabilizing_tolerance": 0.15,
  "sensor_type": "ec_sensor",
  "act_up_type": "ec_up_actuator",
  "act_down_type": "ec_down_actuator"
},
"pH": {
  "required": 6.0,
  "tolerance": 0.1,
  "stabilizing_tolerance": 0.15,
  "sensor_type": "ph_sensor",
  "act_type": "temperature_actuator",
  "act_up_type": "ph_up_actuator",
  "act_down_type": "ph_down_actuator"
}

```

Figure 6.5: Regulation rules

These are the variable values defining required parameters, tolerances and types of sensors and actuators associated with a given parameter. The screenshot is an example where not all parameters are provided.

6.4.1 Parameters regulation

The application provides BaseHandler class which is to be extended for every actuator type. The class has handle() method where regulation logic is. At the moment there are

4 of them:

- **Water Level Handler**

The first parameter to be regulated. Until the water level is at the required mark, none of the other parameters can be regulated.

Logic is the following:

- There are 2 level sensors: the lower and the upper ones.
- When Water Level Handler gets information that water is below the lower sensor, i.e. it sent "0" in its data, command for water level actuator to on is sent.
- When Water Level Handler gets information that water is above the upper sensor, i.e. it sent "1" in its data, command for water level actuator to off is sent.
- When the water level has been regulated the next parameter regulation is possible.

- **EC Handler**

The next parameter for regulation is EC. The handler determines if a gathered value is higher or lower than the required one and based on this it turns on ecUp- or ecDown actuator.

- **ecUp**

ecUp actuator is a dosing pump which pumps fertilizers. We need to determine the amount of time for actuator to be turned on to reach required value. We need to use the following formula for it:

$$act_time = \frac{Coefficient_{ecUp} \cdot V_{water}}{Coefficient_{dosPump}} \quad (6.1)$$

Where $Coefficient_{ecUp}$ is a coefficient defining the amount of fertilizers in milliliters required to raise EC by 1 in 1 liter of hydroponic solution, V_{water} is a volume of water and $Coefficient_{dosPump}$ is the amount of solution which dosing pumps pump in 1 second.

- **ecDown**

ecDown actuator is a dosing pump that pumps pure water. We also need to determine the amount of time for an actuator to be turned on.

$$water_volume_to_add = \frac{V_{water} \cdot Coefficient_{ecToPpm} \cdot current_EC}{required_EC \cdot Coefficient_{ecToPpm}} - V_{water} \quad (6.2)$$

This formula determines the amount of water that is necessary to add to change EC to the required. It comes out of the fact, that EC may be converted to PartsPerMillion (concentration), which may be reduced by raising the water volume. Where $Coefficient_{ecUp}$ is a coefficient defining the amount

- Application stores information about the time, when each actuator must be turned off. Then checks it at the beginning of every iteration and sends the commands to turn off.
- To continue regulation, the EC level must stabilize, i.e. new obtained value must not differ from last stored by more than the value specified in the corresponding rule called "stabilizing tolerance".

• pH Handler

pH Handler works almost the same way as the previous one. The only difference is the formula used to calculate the amount of time for an actuator to stay turned on.

$$regulator_volume_to_add = \frac{\frac{V_{water}}{required_PH} - \frac{V_{water}}{current_PH}}{\frac{1}{phLevel_{regulator}} - \frac{1}{required_PH}} \quad (6.3)$$

Here the regulator is a phUp or a phDown solutions which depends whether we need to raise or lower hydroponic solution's pH level.

• Temperature Handler

The most common and simple handler. Its algorithm:

- If temperature is higher than required, turn on the cooler
- If temperature is lower or equal to the required and the cooler is ON, then turn it off.

7. Testing and Evaluation

Since the project is a system that works with real-time sensors data, automated testing is difficult to implement, therefore in this chapter I will evaluate the results of the work, describe manual testing use-cases and illustrate them using screenshots of logs and AWS console.

7.1 Storing the data

As mentioned in previous chapters, there are 4 parameters which we follow:

- Temperature
- EC level
- pH level
- Water level

To test that all the parameters are successfully measured, gathered and stored in the database we need to take a look at the Local Persistence Service's logs, the Aws Persistence Service's logs and the AWS console.

```
May 15 13:12:11 pi local_persistence_service: [INFO] received: sensors_data temp_sens_1 temperature_sensor 23.07
May 15 13:12:12 pi local_persistence_service: [INFO] received: sensors_data low_lev_sens low_water_level_sensor 0
May 15 13:12:13 pi local_persistence_service: [INFO] received: sensors_data up_lev_sens up_water_level_sensor 1
May 15 13:12:15 pi local_persistence_service: [INFO] received: sensors_data ec_sens_1 ec_sensor 1.08
May 15 13:12:17 pi local_persistence_service: [INFO] received: sensors_data ph_sens_1 ph_sensor 6.25
May 15 13:14:12 pi local_persistence_service: [INFO] received: sensors_data temp_sens_1 temperature_sensor 23.09
May 15 13:14:14 pi local_persistence_service: [INFO] received: sensors_data low_lev_sens low_water_level_sensor 0
May 15 13:14:15 pi local_persistence_service: [INFO] received: sensors_data up_lev_sens up_water_level_sensor 11
May 15 13:14:17 pi local_persistence_service: [INFO] received: sensors_data ec_sens_1 ec_sensor 1.06
May 15 13:14:18 pi local_persistence_service: [INFO] received: sensors_data ph_sens_1 ph_sensor 6.25
```

Figure 7.1: Local persistence service log

Local persistence successfully obtains the sensors data from the Arduino Uno

```
May 15 13:12:11 pi aws_persistence_service: [INFO] Saving item with sensor name temp_sens_1 and value 23.07
May 15 13:12:12 pi aws_persistence_service: [INFO] Saving item with sensor name low_lev_sens and value 0
May 15 13:12:13 pi aws_persistence_service: [INFO] Saving item with sensor name up_lev_sens and value 1
May 15 13:12:15 pi aws_persistence_service: [INFO] Saving item with sensor name ec_sens_1 and value 1.08
May 15 13:12:17 pi aws_persistence_service: [INFO] Saving item with sensor name ph_sens_1 and value 6.25
```

Figure 7.2: AWS persistence service log

Data are successfully being stored to the Dynamo Database.

system_id	timestamp	sensor_name	sensor_type	value
1	2021/05/15 13:14:18	ph_sens_1	ph_sensor	6.25
1	2021/05/15 13:14:17	ec_sens_1	ec_sensor	1.06
1	2021/05/15 13:14:15	up_lev_sens	up_water_level_sensor	1
1	2021/05/15 13:14:14	low_lev_sens	low_water_level_sensor	0
1	2021/05/15 13:14:12	temp_sens_1	temperature_sensor	23.09
1	2021/05/15 13:12:17	ph_sens_1	ph_sensor	6.25
1	2021/05/15 13:12:15	ec_sens_1	ec_sensor	1.08
1	2021/05/15 13:12:13	up_lev_sens	up_water_level_sensor	1
1	2021/05/15 13:12:12	low_lev_sens	low_water_level_sensor	0
1	2021/05/15 13:12:11	temp_sens_1	temperature_sensor	23.07

Figure 7.3: AWS DynamoDB console

As it can be seen from the above, the data are not only successfully stored but stored regularly based on values in the configuration file (2 minutes). New data does not rewrite the old which is suitable for the possible future implementation of Artificial Intelligence algorithms. Data is easily queryable for the purposes of the future front-end application.

7.2 Parameters regulation

In this section I will go through use-cases that will take place during the system work.

- Low temperature

```
May 17 12:07:56 pi temperature_handler: [INFO] Received from temperature sensor value '26' is beyond required
May 17 12:07:56 pi regulation_service: [INFO] Sending command to temperature_actuator with value 1
```

Figure 7.4: Low temperature log

When temperature value is higher than required, the signal is sent to the temperature actuator which starts cooling the water. Nothing happens until temperature lowers. Then the signal to turn off is sent.

```
May 17 12:45:26 pi temperature_handler: [INFO] Temperature is back to normal: 22.5
May 17 12:45:26 pi regulation_service: [INFO] Sending command to temperature_actuator with value 0
```

Figure 7.5: Temperature is back to normal log

- Water level is beyond required

```
May 17 14:01:34 pi temperature_handler: [INFO] Received from low_water_level sensor value '1' is beyond required
May 17 14:01:34 pi regulation_service: [INFO] Sending command to water_level_actuator with value 1
```

Figure 7.6: Low water level log

When water level goes below the lower level sensors, it sends a signal to the water level actuator to turn off, i.e. to the feeding pump. Nothing happens until water touches the upper-level sensor. Then the signal to turn off is sent.

```
May 17 14:04:21 pi temperature_handler: [INFO] Water tank has been filled up
May 17 14:04:21 pi regulation_service: [INFO] Sending command to water_level_actuator with value 0
```

Figure 7.7: Water level is back to normal log

- Low/high pH level

```
May 17 14:25:05 pi ph_nadler: [INFO] Received from ph_sensor sensor value '5.0' is beyond required
May 17 14:25:05 pi ph_nadler: [INFO] Volume of phUp to add: 5 ml
May 17 14:25:05 pi ph_nadler: [INFO] phUp actuator will be up for 5 seconds
May 17 14:25:05 pi regulation_service: [INFO] Sending command to water_level_actuator with value 1
May 17 14:25:10 pi regulation_service: [INFO] Sending command to water_level_actuator with value 0
```

Figure 7.8: Wrong pH log

When pH level is lower or higher than required, the volume of a regulator to add is calculated using the formula (5.3). Then the time for an actuator, i.e. dosing pump, to run is calculated and stored and the signal to turn on is sent. After the calculated time passes, the signal to turn off is being automatically sent.

The process repeats until pH level is stable.

- Low/high EC level

```
May 17 14:41:17 pi ph_nadler: [INFO] Received from ec_sensor sensor value '0.3' is beyond required
May 17 14:41:17 pi ph_nadler: [INFO] Volume of phUp to add: 20 ml
May 17 14:41:17 pi ph_nadler: [INFO] phUp actuator will be up for 20 seconds
May 17 14:41:17 pi regulation_service: [INFO] Sending command to water_level_actuator with value 1
May 17 14:41:37 pi regulation_service: [INFO] Sending command to water_level_actuator with value 0
```

Figure 7.9: Wrong pH log

The algorithm is the same as for regulation of the pH level. The only difference is the formula to calculate the volume of a regulator: (5.1) for ecUp and (5.2) for ecDown.

8. Conclusion

The project was a part of a larger one aimed at almost complete automation of work with hydroponic systems by automated monitoring and regulation of its parameters.

During the project, a list of required HW was compiled, assembled the module containing the HW, proposed and implemented the software system for parameters monitoring and regulation, carried out manual testing. Project goals are considered as accomplished.

This thesis covers the study of hydroponics, hardware and software analysis, analysis of used technologies and their alternatives and a practical part covering implementation and testing.

There are many opportunities for further development:

- Data visualization using **front-end application**.
- Using **Artificial Intelligence** to make parameters regulation even more effective.
- Connection of separate hydroponic systems into one **network**.
- Improved **analysis of the data** stored in the Dynamo Database.

A. List of Abbreviations

- **HW** - Hardware
- **SW** - Software
- **DWC** - Deep Water Culture
- **NFT** - Nutrient film technique
- **EC** - Electrical conductivity
- **RPi** - RaspberryPi
- **SQL** - Structured Query Language
- **DB** - Database
- **AI** - Artificial Intelligence
- **I2C** - Inter-Integrated Circuit
- **SPI** - Serial Peripheral Interface
- **UART** - Universal Asynchronous Receiver-Transmitter
- **SPOF** - Single Point Of Failure
- **AWS** - Amazon Web Services

B. Source code

Project's source code on GitLab:



Bibliography

- [1] *Growing at a slower pace, world population is expected to reach 9.7 billion in 2050 and could peak at nearly 11 billion around 2100: UN Report*, [Online] <https://www.un.org/sustainabledevelopment/blog/2019/06/growing-at-a-slower-pace-world-population-is-expected-to-reach-9-7-billion-in-2050-and-could-peak-at-nearly-11-billion-around-2100-un-report>
- [2] Ali Akhmadov. *Monitoring and automated regulation of parameters of hydroponic system environment*. Bachelor Thesis, Czech Technical University in Prague, Faculty of Electrical Engineering and Technology, 2021.
- [3] William Texier. *Hydroponics for Everybody: All About Home Horticulture*. 2015, ISBN 978-2845941205.
- [4] Pak. J. Agri., Agril. Engg., Vet. Sc. *HYDROPONICS: KEY TO SUSTAIN AGRICULTURE IN WATER STRESSED AND URBAN ENVIRONMENT*, [Online] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.514.4323&rep=rep1&type=pdf>
- [5] Keith Roberto. *How-To Hydroponics, Fourth Edition*. 2003. ISBN 978-0967202617.
- [6] *Hydroponics For Beginners – The Definitive Guide*, [Online] <https://www.trees.com/gardening-and-landscaping/hydroponic-gardening>
- [7] *Why EC Is Important In Hydroponics*, [Online] <https://www.hydroponics.co.uk/news/why-ec-is-important-in-hydroponics/>
- [8] *Arduino Uno technical specifications*, [Online] <https://store.arduino.cc/arduino-uno-rev3>
- [9] *Arduino Uno Board with Real-Time Application Projects*, [Online] <https://www.wa-telectronics.com/arduino-uno-board-tutorial-and-its-applications>
- [10] *Analog Input Pins*, [Online] <https://www.arduino.cc/en/Tutorial/Foundations/AnalogInputPins>
- [11] Simon Monik. *Raspberry Pi Cookbook*, 2013. ISBN 978-1449365226.
- [12] *Capturing Analogue Signals with a Raspberry Pi*, [Online] <https://www.rs-online.com/designspark/capturing-analogue-signals-with-a-raspberry-pi>
- [13] *Amazon DynamoDb documentation*, [Online] <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide>

- [14] *Sql and NoSq: An overview with advantages and disadvantages*, [Online] <https://acodez.in/sql-and-nosql-an-overview/>
- [15] *MongoDb documentation. Sharding*, [Online] <https://docs.mongodb.com/manual/sharding/>
- [16] Brian Evans. *Beginning Arduino Programming*, 2011. ISBN 978-1430237778.
- [17] *Wire library*, [Online] <https://www.arduino.cc/en/reference/wire>
- [18] *TIOBE Index for December 2020*, [Online] <https://www.tiobe.com/tiobe-index/>
- [19] *PySerial documentation*, [Online] <https://pythonhosted.org/pyserial/>
- [20] *Getting Started Developing with Python and DynamoDB*, [Online] <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.html>
- [21] *7 Popular Software Programs Written in Python*, [Online] <https://codeinstitute.net/blog/7-popular-software-programs-written-in-python/>
- [22] Prateek Joshi. *Artificial Intelligence with Python: A Comprehensive Guide to Building Intelligent Apps for Python Beginners and Developers*, 2017. ISBN 978-1786464392.
- [23] *The best AI Programming Languages – Java vs Python*, [Online] <https://huddle.eurostarsoftwaretesting.com/the-best-ai-programming-languages-java-vs-python/>
- [24] *I2C-bus specification and user manual*, [Online] <http://www.nxp.com/docs/en/user-guide/UM10204.pdf>
- [25] *Raspberry Pi Arduino Serial Communication*, [Online] <https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/>
- [26] *Hydroponics Market Size, Share Trends Analysis Report By Type*, [Online] <https://www.grandviewresearch.com/industry-analysis/hydroponics-market>