Raindrop

Paulo Monteiro Ferreira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. João Manuel Brisson Lopes

Examination Committee

Chairperson: Prof. José Carlos Alves Pereira Monteiro

Supervisor: Prof. João Manuel Brisson Lopes

Members of the Committee: Prof. Duarte Nuno Jardim Nunes

November 2018
Acknowledgements

After finishing this long journey, I would like to thank my mentor, Professor João Brisson Lopes, for giving me the opportunity to work on this project. It was due to him that I achieved success, through his relentless help and vast knowledge.

To Dr João Pedro Sanches, owner of Quinta das Chantas, who kindly received me at his premises and provided documents and knowledge that revealed to be extremely important for the realisation of this work.

To Eng. Daniel Ribeiro, partner at HIDROSOPH, who helped expand my knowledge in agriculture by sharing key information on the operation of HIDROSOPH’s products.

To all Maretec team and associates, which offered me the possibility to develop this system using their weather services.

To Imprensa Nacional-Casa da Moeda (INCM), my employer, which values the education of their employees and who has given me time to carry out this project.

A special thanks to my family, girlfriend and friends. For their great support, advice and suggestions.

Thank you.
Abstract

Predicting crop water requirements is a hard task given the number of variables involved. Smart irrigation systems help farmers calculate such requirements using weather forecasts and/or soil moisture values and irrigate accordingly. After analyzing the existing solutions, we concluded that one major irrigation problem is the lack of interoperability, this is, when two or more controllers are required to acquire the needs of a field, they do not share information with each other, thus preventing any intelligence capabilities.

This document describes all the processes behind the conception and implementation of a solution, *Raindrop*, designed to tackle the problem above. Raindrop is a smart irrigation system that can be remotely accessed through a smart device. It combines meteorological forecasts with soil moisture values to generate irrigation schedules to provide the required amounts of water at the right time, achieving water and energy savings and healthy crops.

Our solution was tested in two ways: data analysis and usability. Through data analysis, we were able to compare the results obtained by algorithms with the data observed in a farm, and we concluded that for better system operation both meteorological predictions and readings of the humidity probes must be used. The usability of our application was tested with a group of real users, who performed tasks inherent to the critical functionalities, together with a SUS test that showed a high final score.

**Keywords:** Smart Irrigation, Evapotranspiration, Soil Moisture Sensors, Weather Forecast.
Resumo

Calcular as necessidades de água de uma cultura é uma tarefa difícil, dado o número de variáveis envolvidas para o efeito. Os sistemas de rega inteligentes ajudam os agricultores a calcular tais necessidades, utilizando previsões meteorológicas e/ou monitorizando os valores de humidade do solo, ativando o sistema de rega quando necessário. Após a análise das soluções existentes, concluímos que existe um problema fundamental, a falta de interoperabilidade quando dois ou mais controladores são necessários, perdendo, assim, capacidades de inteligência.

Este documento descreve todos os processos que estão por detrás da conceção e implementação de uma solução, Raindrop, um sistema de rega que resolve o problema acima mencionado e que pode ser acedido remotamente através de um dispositivo inteligente. Este sistema combina as previsões meteorológicas com os valores de humidade do solo para gerar calendarizações de rega de modo a fornecer as quantidades de água necessárias em tempos oportunos, de forma a economizar água e energia e obter culturas saudáveis.

A nossa solução foi testada de duas formas: análise de algoritmos e usabilidade. Através da análise de algoritmos pudemos comparar os resultados obtidos pelos nossos algoritmos com os dados observados numa quinta de cultivo. Concluímos que para um melhor funcionamento do sistema devem ser usadas tanto as previsões meteorológicas como as leituras das sondas de humidade. A usabilidade da nossa aplicação foi testada com um grupo de participantes representativos de usuários reais que realizaram tarefas inerentes às funcionalidades críticas, tendo o teste SUS obtido uma boa pontuação.

Palavras-chave: Rega Inteligente, Evapotranspiração, Sensores de Humidade do Solo, Previsões Meteorológicas.
# Table of Contents

Acknowledgements .................................................................................................................. iii

Abstract ..................................................................................................................................... v

Resumo ....................................................................................................................................... vii

Table of Contents ......................................................................................................................... ix

List of Figures ............................................................................................................................... xiii

List of Tables ................................................................................................................................. xv

List of Acronyms ............................................................................................................................ xviii

1. Introduction ............................................................................................................................... 20
   1.1. Context ................................................................................................................................. 20
   1.2. The problem ......................................................................................................................... 20
   1.3. Project goals ......................................................................................................................... 21
   1.4. Ground research .................................................................................................................... 21
   1.5. Document structure ............................................................................................................. 24

2. Theoretical framework ............................................................................................................. 25
   2.1. Crop water requirements .................................................................................................... 25
   2.2. Soil texture ......................................................................................................................... 25
   2.3. Soil moisture ....................................................................................................................... 26
   2.4. Water balance equation ...................................................................................................... 27
   2.5. Crop evapotranspiration calculation .................................................................................. 27
   2.6. Forecasting irrigation ......................................................................................................... 30

3. Related work ............................................................................................................................ 32
   3.1. Smart irrigation controllers ............................................................................................... 32
       3.1.1. Weather-based irrigation controllers ........................................................................... 32
       3.1.2. Soil moisture-based irrigation controllers ................................................................. 33
   3.2. Irrigation systems products ............................................................................................... 33
       3.2.1. Blossom ....................................................................................................................... 34
       3.2.2. Netro Smart ................................................................................................................. 34
       3.2.3. Rachio Smart Sprinkler ............................................................................................ 34
       3.2.4. RainMachine .............................................................................................................. 35
List of Figures

Figure 1. Soil moisture sensor placed at olive grove landscape (Quinta das Chantas, 30 November 2017). ......................................................................................................................................................... 22
Figure 2. Olive grove landscape (Quinta das Chantas, 30 November 2017). .................................................. 23
Figure 3. Landscape preparation for walnut plantation (Quinta das Chantas, 30 November 2017). .... 23
Figure 4. Graph extracted from a report of Quinta das Chantas, showing irrigation on a rainy day. .... 24
Figure 5. Soil texture triangle, showing soil classes according to the percentage of sand, silt, and clay [8]. ......................................................................................................................................................... 25
Figure 6. Soil water balance inputs and outputs [6]. ......................................................................................... 27
Figure 7. Water stress coefficient, Ks [6, p. 167]. ............................................................................................ 29
Figure 8. Total available soil water to the crop depends on the root’s depth [10]. ........................................ 30
Figure 9. Native development model, adapted from [22]. ............................................................................ 38
Figure 10. Cross-platform development model, adapted from [22]............................................................... 39
Figure 11. Application test flow. .................................................................................................................. 41
Figure 12. Layered view of the proposed solution. ....................................................................................... 49
Figure 13. Relation client-server between components. ............................................................................... 51
Figure 14. User model schema, containing a username, a name, an e-mail, a hashed password, the registry date, the list of controllers associated with such user and the user areas. ........................................... 53
Figure 15. User model validations: check if the username is alphanumeric, verify that username meets minimum and maximum length and confirm that there are no other users with the same username. 54
Figure 16. Water balance example, without irrigation and with irrigation, on the left and right respectively. ........................................................................................................................................ 55
Figure 17. List of items with two dimensions on the left, and the same items packed in a bin, on the right, using the Sleator algorithm [29]. ..................................................................................................... 57
Figure 18. Software test, white-box technique, create a user model in the database. .............................. 59
Figure 19. Software test, black-box technique, register a user account using the REST API. ............... 59
Figure 20. Example of server logs in the development environment. ............................................................ 60
Figure 21. Raindrop mobile app view navigation. ......................................................................................... 62
Figure 22. Logged out state, from the left to the right: login view and register view. ............................ 63
Figure 23. Raindrop mobile app, areas view. ................................................................................................ 64
Figure 24. Area view: list of zones, weekly predictions and location. ........................................ 64
Figure 25. Zone details......................................................................................................................... 65
Figure 26. Zones watering schedule example...................................................................................... 65
Figure 27. Zone predicted water content in the soil. .............................................................................. 66
Figure 28. Zone predicted water balance. ............................................................................................. 66
Figure 29. Devices view.......................................................................................................................... 67
Figure 30. Profile View........................................................................................................................... 67
Figure 31. Flexible layout is responding to the device orientation. ......................................................... 68
Figure 32. Area creation view, from left to right: name, location and weather provider. .................... 69
Figure 33. Location - text search............................................................................................................ 70
Figure 34. Location - map marker........................................................................................................... 70
Figure 35. Comparison between observed water content collected from each zone. ......................... 72
Figure 36. Predicted water content using weather forecasts versus average observed zone water content. ................................................................................................................................. 73
Figure 37. Predicted water content using weather forecasts and soil moisture sensors versus average observed zone water content. ................................................................................................................. 74
Figure 38. Development cycle............................................................................................................... 75
Figure 39. User demographics: age and gender.................................................................................... 76
Figure 40. User previous experience ................................................................................................... 76
Figure 41. The time required to complete each task............................................................................... 77
Figure 42. On the left, location search by street, zip or city; on the right, mark the location. ............... 78
Figure 43. Change weather provider..................................................................................................... 79
List of Tables

Table 1. Typical soil water content levels, expressed in m$^3$ m$^{-3}$, for different soil types [6, p. 144]................. 26
Table 2. Smart irrigation products comparison. .................................................. 36
Table 3. Comparison between native, web, hybrid, interpreted and cross-compiled approaches [20, 23]. .................................................. 40
Table 4. Smart devices used for testing. .......................................................... 43
Table 5. App loading time comparison (lower is better) ........................................ 43
Table 6. Packages used to develop the server among NodeJS. .................................. 52
Table 7. Irrigation amount algorithm .................................................................. 56
Table 8. Sleator algorithm [29] ........................................................................ 58
Table 9. Type of logs for each environment.......................................................... 60
Table 10. Packages used with react native to develop the mobile application. .......... 61
Table 11. Colour palette used in the mobile app. .................................................. 62
Table 12. Weather icons decision table, using cloud percentages and rain amount. .... 65
Table 13. Observed water content versus predicted water content ....................... 73
Table 14. Task 1 - Registration. ........................................................................ 77
Table 15. Task 2 - Login.................................................................................. 77
Table 16. Task 3 - Add a device......................................................................... 78
Table 17. Task 4 - Create an area. ..................................................................... 78
Table 18. Task 5 - Update area......................................................................... 79
Table 19. Task 6 - Create a zone. ..................................................................... 80
Table 20. Task 7 - Rename zone. ..................................................................... 80
Table 21. Task 8 - Check irrigation schedule ..................................................... 80
Table 22. Task 9 - Check the weather forecast. .................................................. 81
Table 23. Task 10 - Change units. ................................................................... 81
Table 24. Task 11 - Log out.............................................................................. 81
Table 25. System Usability Scale spreadsheet. ..................................................... 82
List of Acronyms

$ET_c$  
Crop evapotranspiration

$TAW$  
Total available water

$RAW$  
Readily available soil water

$MAD$  
Management allowed depletion

$K_c$  
Crop coefficient

$ET_o$  
Reference crop evapotranspiration

$K_s$  
Water stress coefficient

$ET_{c\,adj}$  
Crop evapotranspiration under non-standard conditions

$SWAT$  
Smart Water Application Technologies

$API$  
Application Programming Interface

$OS$  
Operating System

$SDK$  
Software Development Kits

$HTTP$  
Hypertext Transfer Protocol

$GPS$  
Global Positioning System

$IP$  
Internet Protocol

$JSON$  
JavaScript Object Notation

$UI$  
User Interface

$XML$  
Extensible Markup Language

$FAO$  
Food and Agriculture Organization

$REST$  
Representational State Transfer

$SUS$  
System Usability Scale
1. Introduction

This chapter introduces the project scope, exposes the problems in current solutions, lists our goals, and illustrates ground research.

1.1. Context

Irrigation is a technique used in farming that allows the cultivation of crops in areas where rainfall is not enough to meet crop development requirements.

According to R.N. Reddy [1], farming uses close to 66% of the world’s water usage and most irrigation systems supply at least from 50 to 60% more water than necessary. Water usage from limited supplies is not only a high-cost exercise but also raises environmental concerns [2].

Water is a life requirement. We must ensure that when we exploit this resource, we are not compromising the lives of the others now or in the future [3].

1.2. The problem

Throughout the year, the water requirements of crops change due to different weather conditions and crop development. These requirements are complicated to track, especially in what concerns water usage. Smart irrigation controllers were created to solve this problem.

As we will discuss in Chapter 3, there are two main types of smart irrigation controllers: weather-based and soil moisture-based.

Weather-based controllers focus mainly on weather conditions to estimate crop irrigation needs. This approach may fail due to the use of a non-representative weather data source. Furthermore, it does not track the actual amount of water that infiltrates the soil.

Soil moisture-based controllers collect data from sensors placed in representative areas and trigger irrigation when moisture levels drop down a certain threshold. This approach may fail, since water needs can be different within a zone. Furthermore, these systems may trigger irrigation even when rainfall is highly probable.

From our research, we can conclude that smart controllers, independently of the approach, have a limited number of zones and that that limit might be surpassed by adding an extension (if available) or adding a new controller (keeping the previous).

On the one hand, adding an extension might mean that cables connecting the irrigation controller to the sensors and actuators, will have significant lengths, increasing cost and the probability of errors (e.g., cable break).
On the other hand, adding a new controller might require extra attention from the user. None of the products reviewed enable communication across controllers in the same location to share data (interoperability). Consequently, there may be overlapping irrigation schedules that lower water pressure and irrigate less than planned.

1.3. Project goals

Our solution to solve the problem comprehends:

- A centralised server that remotely manages the irrigation controllers;
- An irrigation algorithm based on weather conditions and soil humidity sensor data;
- An abstraction to handle multiple controllers as one controller;
- A mobile application for irrigation controller configurations and commands, display statistics and transmit decision making that is transparent to the user.

1.4. Ground research

For a better understanding of how a real-world farm environment works, we contacted Mr João Pedro Sanches, owner of Quinta das Chantas, a farm located in Santarém, Portugal. We were able to visit it, on 30 November 2017, with the owner presence and his guiding.

Two separate areas constitute Quinta das Chantas: one for olive grove cultivation, with 53.57 ha; and another for walnut cultivation, with 22 ha. Both are located on top of a water table that, according to the owner, might be one of the biggest in the country.

Magos\(^1\), a company expert in irrigation systems, created a planning proposal for this farm, in late 2016, that included both an olive area and a walnut area, and was split as follows:

- Project details. Numerical information for each area, i.e., crop plantation area and respective spacing, amount of water sprinklers and respective spacing, maximum flow for each sprinkler, total daily crop water needs, and the time necessary to fulfil water needs for each day.
- Technical data. Description of motors, water filters, hydraulic valves and flow regulators accordingly to their maximum water flow/pressure, power consumption, and efficiency.
- Pricings. List of products to purchase and their assembly labour, e.g., water pumping, water filtration, valve manifolds, primary and secondary conducts, irrigation controller, soil moisture sensors.
- Commercial conditions. Report's time-frame validity, payment instructions, warranty and excluded properties.
- Appendix. Two topographic views of the area for each crop: crop lines view, and watering zones. Crop lines were designed to be perpendicular to level contours, so lateral slopes could be

\(^1\) http://magos.pt/
avoided, and oriented as much north-south as possible to be able to get most of the daylight. Zones were created to group crop lines with similar water needs.

**Olive grove landscape**

In early 2017, tractors ploughed the olive grove area, specialists installed the required instruments, and the farmers cultivated the crop.

The irrigation controller that was installed works as a clock system (this can be upgraded in the future). Therefore, the farmer needs to change the watering schedule manually.

Four moisture sensors like the one in Figure 1 were installed. Sensors were installed one per zone, at locations previously selected from topographic maps. Each sensor can collect data from multiple depths, up to a maximum of 80 cm. This range is particularly important as with the crop growth, the root zone will increase, and plants will be able to collect water from deeper zones.

![Figure 1. Soil moisture sensor placed at olive grove landscape (Quinta das Chantas, 30 November 2017).](image)

The irrigation controller does not connect directly to the sensors. Instead, those sensors send the information to a web server that generates weekly reports, providing decision support for water management. With those reports, farmers adjust the controller’s watering cycles.

The owner, as of to date, is thrilled with the current results and says that the crop is growing at an exceptional rate, as seen in Figure 2.
Walnut landscape

As of the visit day, the farmers were preparing the walnut landscape for plantation (see Figure 3): tractors have ploughed the field; primary and secondary water conducts have been installed. For each line, there are four conducts installed. However, only two of them will be watering in the beginning. With crop development and consequent water requirement increase, the two remaining conducts will be activated in the future.

Visit conclusions

Farmers need to consider many factors even before the plantation phase. For a profitable business every detail count: water, for irrigation; electricity, how to power the system (e.g., pumps to extract the water from groundwater table); fuel, used for tractors; and fertiliser.

The water provided to this farm comes from the groundwater table. The extractions costs are related to the licensing and the machinery and power required to extract it. However, this resource needs to be carefully explored, given the limited quantity, especially during dry seasons. Also, the Portuguese government supports the efficient use of water, providing monetary support to farmers that meet specific criteria [4]. In this farm, all watering operations to be placed during off-peak hours, reducing electricity cost. Furthermore, tractors are equipped with GPS devices to compute the exact amount of fuel required and apply fertiliser uniformly given speed.

This farm’s controller lacks the ability to automatically adjust watering schedules based on the sensor’s readings and the weather forecast. Moreover, the reports from the sensors are only generated once a
week. Therefore, the farmer does not have immediate feedback of moisture values and, possibly, over-irrigate, or worst, under-irrigate crops.

An example of how an automated system provided irrigation inadequately can be seen in Figure 4. As the report shows, irrigation was provided on the 14th of October, two days before a predicted rainfall. Furthermore, after a rainy day, on the 18th of October, irrigation was provided, even though it rained on that day too.

![Figure 4. Graph extracted from a report of Quinta das Chantas, showing irrigation on a rainy day.](image)

In sum, we can suppose that the farmer may not have read the reports and act immediately accordingly; also, he might have forgotten to modify the programmed schedule, or only change it a few days later. In addition, weather forecasts can change when considering extended periods.

### 1.5. Document structure

This document’s first chapter introduces the project scope, exposes the problems of current solutions, lists our goals, and illustrates its ground research. The second chapter aims to establish a basic understanding of the theoretical framework involved in this project. The third chapter goes through the different types of smart irrigation controllers and evaluates current solutions, presents the data offered by meteorological services, and covers several mobile development approaches. The fourth chapter lists the requirements that must be met by our proposed solution. The fifth chapter provides a broad overview of the solution’s domain model, components and communications. The sixth chapter details the implementation of both the server and the mobile app components of this solution. The seventh chapter aims to test the implemented solution via data analysis and user testing. Finally, the eighth chapter concludes this work with a brief discussion of what is done so far and its limitations and what future work can be built upon ours.
2. Theoretical framework

This chapter presents the basic theoretical concepts underlying our project, which are relevant for a better understanding of the related work and our solution. It explains how soil type influences the amount of water that can be stored, how much water is required to meet the needs of a crop, and how those needs can be predicted and watering schedules generated.

2.1. Crop water requirements

Crop water requirements, also named crop evapotranspiration, are usually represented as $ET_c$ and must be fulfilled to attain potential crop yields [5]. Evapotranspiration is the combination of two separate processes: evaporation of liquid water from the soil surface; and transpiration of water from the plants themselves [6].

Stored soil water, precipitation, and irrigation supply water to the crop. Therefore, irrigation is required when the crop water requirements exceed the supply of water from the soil and precipitation.

2.2. Soil texture

The soil is composed of all kinds of particles of different sizes [7]. Most of them originate from the degradation of rocks, the mineral particles. Some originate from residues of plants or animals, the organic particles. Water and air are stored in soil particles spaces, called pores.

The proportion of sand, silt, and clay present in the soil composition determine its texture (see Figure 5). The amount of water and air that type of soil can store is directly related to the soil texture since texture affects the pore size.

Figure 5. Soil texture triangle, showing soil classes according to the percentage of sand, silt, and clay [8].
2.3. Soil moisture

The soil moisture content or soil water content, denoted as $\theta$, indicates the amount of water present in the soil and can be calculated from the following equation [7]:

$$\theta = \frac{V_w}{V_t},$$

where $\theta$ is the soil moisture content [m$^3$ m$^{-3}$], $V_w$ the total volume of water [m$^3$] and $V_t$ the total volume of air, water, and soil [m$^3$].

There are three essential water contents in irrigation planning [7]:

- Saturation ($\theta_s$), where water fills all the soil pores, and no air is left in the soil. Many crops cannot withstand saturated soil conditions for longer than 2-5 days;
- Field capacity ($\theta_{fc}$), where the large soil pores are filled with both air and water while the smaller pores are full of water. This is ideal for crop growth;
- Permanent wilting point ($\theta_{wp}$), where the soil becomes mostly dry, and it gets hard for the plant roots to extract water. At this point the plant wilts.

The relation between those states can be expressed as follows:

$$\theta_{wp} < \theta_{fc} < \theta_s$$

Common values for $\theta_{fc}$ and $\theta_{wp}$, for different soil types, are presented in Table 1.

<table>
<thead>
<tr>
<th>Soil type</th>
<th>$\theta_{fc}$</th>
<th>$\theta_{wp}$</th>
<th>$\theta_{fc} - \theta_{wp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sand</td>
<td>0.07 - 0.17</td>
<td>0.02 - 0.07</td>
<td>0.05 - 0.11</td>
</tr>
<tr>
<td>Loamy sand</td>
<td>0.11 - 0.19</td>
<td>0.03 - 0.10</td>
<td>0.06 - 0.12</td>
</tr>
<tr>
<td>Sandy loam</td>
<td>0.18 - 0.28</td>
<td>0.06 - 0.16</td>
<td>0.11 - 0.15</td>
</tr>
<tr>
<td>Loam</td>
<td>0.20 - 0.30</td>
<td>0.07 - 0.17</td>
<td>0.13 - 0.18</td>
</tr>
<tr>
<td>Silt loam</td>
<td>0.22 - 0.36</td>
<td>0.09 - 0.21</td>
<td>0.13 - 0.19</td>
</tr>
<tr>
<td>Silt</td>
<td>0.28 - 0.36</td>
<td>0.12 - 0.22</td>
<td>0.16 - 0.20</td>
</tr>
<tr>
<td>Silt clay loam</td>
<td>0.30 - 0.37</td>
<td>0.17 - 0.24</td>
<td>0.13 - 0.18</td>
</tr>
<tr>
<td>Silty clay</td>
<td>0.30 - 0.42</td>
<td>0.17 - 0.29</td>
<td>0.13 - 0.19</td>
</tr>
<tr>
<td>Clay</td>
<td>0.32 - 0.40</td>
<td>0.20 - 0.24</td>
<td>0.12 - 0.20</td>
</tr>
</tbody>
</table>

*Table 1. Typical soil water content levels, expressed in m$^3$ m$^{-3}$, for different soil types [6, p. 144].
2.4. Water balance equation

The water balance equation expresses the equality between the input and output of water from a farming system, as depicted in Figure 6, and can be used to estimate the required amount and timing of irrigation for crops.

![Figure 6. Soil water balance inputs and outputs [6].](image)

The amount of water required to re-establish up to the field capacity level is the difference between field capacity and current soil water level and corresponds to the soil water depletion \( (D_r) \) [5]. The daily water balance equation, expressed in terms of depletion at the end of the day is [6]:

\[
D_{r,i} = D_{r,i-1} - (P - RO)_i - I_i - CR_i + ET_{c,i} + DP_i,
\]

(3)

where \( D_{r,i} \) is the soil water depletion on day \( i \) [mm], \( D_{r,i-1} \) the soil water depletion on the previous day \( (i-1) \) [mm], \( P_i \) the precipitation on day \( i \) [mm], \( RO_i \) the runoff on day \( i \) [mm], \( I_i \) the net irrigation depth on day \( i \) that infiltrates the soil [mm], \( CR_i \) the capillary rise from day \( i \) [mm], \( ET_{c,i} \) the crop evapotranspiration on day \( i \) [mm], and \( DP_i \) the deep percolation on day \( i \) [mm].

2.5. Crop evapotranspiration calculation

According to Allen et al. [6], there are two distinct methods to calculate crop evapotranspiration:

- single crop coefficient: for daily, 10-day or monthly calculations, where a single coefficient \( (K_c) \) integrates both evaporation and transpiration. Indicated for non-frequent water applications (surface and sprinkler irrigation);

- dual crop coefficient: for daily calculations, where both transpiration and evaporation are evaluated separately, resulting in two coefficients: basal crop \( (K_{cb}) \) and soil evaporation coefficient \( (K_e) \). Indicated for irrigation with high-frequency water application.

For this project, we will use the single crop coefficient \( (K_c) \), as it should be enough to provide adequate results for daily computations.
Crop coefficient \((K_c)\)

The crop coefficient, \(K_c\), is the ratio of the crop \(ET_c\) to the reference \(ET_o\), and it represents an integration of the effects of four primary characteristics that distinguish the crop from reference grass: crop height, the albedo of the crop-soil surface, canopy resistance and evaporation from soil [6].

As the crop develops, the ground cover, crop height, and leaf area change. Due to differences in evapotranspiration during the various growth stages, the \(K_c\) for a given crop will vary over the growing period. The growing period can be divided into four distinct growth stages:

- Initial stage \((K_{c\,\text{ini}})\),
- Crop development stage \((K_{c\,\text{dev}})\),
- Mid-season stage \((K_{c\,\text{mid}})\),
- Late season stage \((K_{c\,\text{end}})\).

Common crop coefficient values, \(K_c\), for different plants and development stages are presented in Appendix A.

Reference evapotranspiration \((ET_o)\)

Reference evapotranspiration, \(ET_o\), is “the evapotranspiration of a hypothetical grass reference crop with an assumed crop height of 0.12 m, a fixed surface resistance of 70 s m\(^{-1}\) and an albedo of 0.23” [6].

The FAO Penman-Monteith method is recommended to calculate \(ET_o\), as it provides consistent values for all regions and climates. For a daily calculation, FAO Penman-Monteith equation is [6]:

\[
ET_o = \frac{0.408 \Delta (R_n - G) + \frac{900}{T + 273} u_2 (e_s - e_a)}{\Delta + \gamma (1 + 0.34 u_2)},
\]

where \(ET_o\) is the reference evapotranspiration [mm day\(^{-1}\)], \(R_n\) the net radiation at the crop surface [MJ m\(^{-2}\) day\(^{-1}\)], \(G\) the soil heat flux density [MJ m\(^{-2}\) day\(^{-1}\)], \(T\) the mean daily air temperature at 2 m height [\(^\circ\)C], \(u_2\) the wind speed at 2 m height [m s\(^{-1}\)], \(e_s\) the saturation vapour pressure [kPa], \(e_a\) the actual vapour pressure [kPa], \(\Delta\) the slope vapour pressure curve [kPa \(^\circ\)C\(^{-1}\)], and \(\gamma\) the psychrometric constant [kPa \(^\circ\)C\(^{-1}\)].

A fully detailed explanation of how to calculate \(ET_o\) with the FAO Penman-Monteith method is available in [6, pp. 15-86] and, an abridged one, in [9].

The meteorological data required from weather data services for the calculation with the FAO Penman-Monteith method is [6]:

- Location: altitude above sea level [m], latitude (degrees north or south), and height of measurement above ground surface \((h)\) [m];
- Air temperature: daily maximum \((T_{\text{max}})\) and minimum temperature \((T_{\text{min}})\) [\(^\circ\)C];
- Air humidity: daily maximum \((RH_{\text{max}})\) and minimum relative humidity \((RH_{\text{min}})\) [%];
- Radiation: average daily net radiation \((R_n)\) [MJ m\(^{-2}\) day\(^{-1}\)];
- Wind speed: average daily wind speed \((u_2)\) [m s\(^{-1}\)].

**Crop evapotranspiration under standard conditions**

The crop evapotranspiration under standard conditions \((ET_c)\) is “the evapotranspiration from disease-free, well-fertilized crops, grown in large fields, under optimum soil water conditions, and achieving full production under the given climatic conditions” [6]. \(ET_c\), normally expressed in millimeters (mm) per unit time (1 day, usually), can be estimated as:

\[
ET_c = K_c ET_o,
\]

where \(ET_c\) is the crop evapotranspiration [mm d\(^{-1}\)], \(K_c\) the crop coefficient and \(ET_o\) the reference crop evapotranspiration [mm d\(^{-1}\)].

**Water stress coefficient \((K_s)\)**

The water stress coefficient \((K_s)\) describes the effect of water stress on crop transpiration [6]. When the root zone depletion \((D_r)\) exceeds the \(RAW\) threshold \(\theta_r\), the root zone depletion is high enough to limit evapotranspiration to less than potential values and the crop evapotranspiration begins to decrease in proportion to the amount of water remaining in the root zone (see Figure 7). In this state \((D_r > RAW)\), \(K_s\) is [6]:

\[
K_s = \frac{TAW - D_r}{TAW - RAW},
\]

where \(K_s\) is a dimensionless transpiration reduction factor dependent on available soil water [0 - 1], \(TAW\) the total available soil water in the root zone [mm], \(D_r\) the root zone depletion [mm], and \(RAW\) the readily available soil water in the root zone [mm].
Crop evapotranspiration under non-standard conditions

The crop evapotranspiration under non-standard conditions ($ET_{c\,adj}$) is “the evapotranspiration from crops grown under management and environmental conditions that differ from the standard conditions” [6]. It is calculated by using a water stress coefficient ($K_s$), and/or by adjusting $K_c$ for all kinds of other stresses and environmental constraints on crop evapotranspiration:

$$ET_{c\,adj} = K_s K_c ET_o,$$

where $ET_{c\,adj}$ is the crop evapotranspiration adjusted [mm d$^{-1}$], $K_s$ the water stress coefficient, $K_c$ the crop coefficient and $ET_o$ the reference crop evapotranspiration [mm d$^{-1}$].

The different formulas should be used according to the depletion level ($D_{r,i}$):

- $D_{r,i} \leq TAW$, depletion has not reached stress levels, therefore $ET_c$ should be used;
- $D_{r,i} > TAW$, depletion has reached stress levels, therefore $ET_{c\,adj}$ should be used.

2.6. Forecasting irrigation

Irrigation is required when rainfall is insufficient to compensate for the water lost by evapotranspiration. Since different crops may have different root depths, the total available water for a crop, $TAW$, may vary (see Figure 8). The capacity of a soil to retain water that is available to plants can be computed as [6]:

$$TAW = 1000(\theta_{fc} - \theta_{wp})Z_r,$$

where $TAW$ is the total available soil water in the root zone [mm], $\theta_{fc}$ the water content at field capacity [m$^3$ m$^{-3}$], $\theta_{wp}$ the water content at wilting point [m$^3$ m$^{-3}$], and $Z_r$ the rooting depth [m].

![Figure 8. Total available soil water to the crop depends on the root's depth [10].](image)
However, crops can only extract a fraction of $TAW$ without suffering water stress. This fraction named readily available water ($RAW$), can be computed as follows [6]:

$$RAW = p TAW,$$

where $RAW$ is the readily available soil water in the root zone [mm] and $p$ the fraction of $TAW$ that can be depleted before stress occurs [0 - 1]. Common values of $p$, for different crops, can be found in Appendix A.

Management allowed depletion ($MAD$) is the maximum allowed depletion before irrigation refill occurs. It is influenced by management and economic factors in addition to the physical factors. This value can be:

- $MAD \leq RAW$, where the crop does not suffer water stress;
- $MAD > RAW$, where water stress is an intentional part of soil water management.

Therefore, when depletion ($D_{r,i}$) is getting close to management allowed depletion ($MAD$), irrigation should be provided.

One should be aware that an irrigation system has high inertia: from the time when the depletion value is computed (via predictions or soil moisture sensors) to the time when the plant absorbs water by infiltration in the soil, there could be a moment where depletion is higher than management allowed depletion ($D_{r,i} > MAD$), reaching non-intended values of depletion. For this reason, a margin should be considered before reaching $MAD$.

If no precipitation ($P_i$) is forecasted, the net irrigation depth ($I_i$) can be the same as root zone depletion ($D_{r,i}$), to bring the water soil deficit to zero. To avoid deep percolation losses that may leach relevant nutrients, $I_i$ should be smaller than or equal to $D_{r,i}$. 
3. Related work

This chapter introduces the most relevant state of the art. It goes through the different types of smart irrigation controllers and evaluates current solutions, presents the data offered by meteorological services, and covers the various mobile development approaches.

3.1. Smart irrigation controllers

Smart irrigation controllers are an evolution of clock-based controllers, as they only provide water as crops needs, instead of operating on a predefined constant schedule, watering every day for a fixed period and a fixed water quantity.

With the evolution of technology, smart irrigation controllers have improved substantially. Materials for the creation of the products and their manufacture process have become cheaper, internet access has grown around the world, and watering calculation formulas have been improved.

The two main types of smart irrigation controllers are weather-based and soil moisture-based [11]. Some controllers have the option to operate with both options, combining the best of two worlds. Regardless of the type, they both share the same target: calculate the irrigation quantity to be applied at the right time.

3.1.1. Weather-based irrigation controllers

Weather-based controllers use weather conditions to plan irrigation schedules. Weather data can be collected either in real-time or via historical records and is used to calculate evapotranspiration, as described in Chapter 2. With the calculation results from the Soil Water Balance Equation, the system keeps track of depletion and eventually compensates water loss with irrigation.

The quality of the collected data represents a significant factor in how good the controller will operate. There are two ways to obtain real-time weather data: on-site measurements; or via a remote weather station site. Both those methods have their trade-offs.

On the one hand, a limited data set will be generated with on-site sensors, and therefore the evapotranspiration calculation might be weaker than using a full data set from a weather station. On the other hand, a remote weather station data may not be representative of the irrigation site, and therefore the results might not be accurate.

The weather station from where the data is collected can either be public or private. A centralised computer server usually processes this data that is then transmitted to the irrigation controllers. This operation has costs that can either be paid by the controller provider or by the controller users.

In this approach, irrigation controllers have no feedback on the amount of water that infiltrates the soil. Therefore, crops might not receive the necessary amount of water.
3.1.2. Soil moisture-based irrigation controllers

Soil moisture-based controllers schedule irrigation according to soil moisture conditions measured on-site with one or more soil moisture sensors. Those controllers usually initiate watering when soil moisture level drops to a threshold point set by the user – management allowed depletion (MAD) - and stop when soil moisture level is near field capacity.

Some controllers uniformly irrigate all zones using the data acquired from a single sensor. Others use multiple sensors, placed in representative zones, and apply different amounts of water to each zone.

With the evolution of technology, moisture sensors have improved and became cheaper. Many different soil moisture sensor types exist with various methods to calculate moisture [11]. Since sensors may work differently for different soil types, temperatures and salinity, a calibration is usually required. For irrigation, consistency of a sensor is as important a factor, or more so than, accuracy. If the values read by a sensor, for a specific soil moisture content, stay similar through time, the controller will be able to interpret data correctly.

However, the sensors used to trigger irrigation might not represent the entire zone correctly, and furthermore can even be misplaced with time due to soil movements, providing incorrect data to the controller.

Moreover, those controllers follow a reactive approach that does not take into consideration the weather forecast, e.g., such system may trigger irrigation till soil moisture achieves field capacity even though there might be high probability of rainfall.

3.2. Irrigation systems products

Many smart irrigation systems are currently available in the market. Most of them usually take advantage of old controller’s wires installation, promoting easy installation. Those controllers are either sold as stand-alone controllers or as an add-ons to an existing one [11].

Comparing the efficiency of the devices is a hard task, as they must be tested in similar conditions (e.g., weather, soil and crop type) and time frames. Most of them claim a percentage range of possible water savings. However, the reference to what those values are being compared to is unknown to the user. To verify if the products results meet specific standards, associations such as SWAT² and WaterSense³ oversee the development of irrigation tools.

In the following, we will present the products Blossom, Netro Smart, Rachio Smart Sprinkler, RainMachine, and Skydrop. Note that this list is not extensive but shows the different qualities and issues.

---

² [https://www.irrigation.org/SWAT](https://www.irrigation.org/SWAT)
³ [https://www.epa.gov/watersense](https://www.epa.gov/watersense)
3.2.1. Blossom

Blossom⁴ is a weather-based irrigation controller available in the United States. It does not contain any screen, buttons or controls [12], so configuration must be made via one of the available mobile applications. It has a status light indicator that changes colours (green, blue-purple, yellow and red) and modes (solid and pulsating) for the different system states [13].

For installations where Wi-Fi is not the best option, Blossom provides an alternative to connect via the powerline, taking advantage of the electric system to connect to the router.

It lacks statistics data, like water usage, meteorological predictions, and notifications for completed or skipped water cycles.

3.2.2. Netro Smart

Netro Smart⁵ is a weather-based irrigation controller that can optionally combine both weather and soil moisture characteristics. This product's company has a proprietary wireless sensor that can be installed on the slope and will connect to the cloud via Wi-Fi, relaying values of solar exposure, soil humidity, and temperature.

The watering schedule is decided automatically according to the configuration for each zone, adjusting the timing and quantity required.

It cannot water multiple zones at the same time. Moreover, it does not allow manual cycles for more than 60 minutes [14].

3.2.3. Rachio Smart Sprinkler

Rachio Smart Sprinkler⁶ is a weather-based irrigation controller that can optionally combine both weather and soil moisture characteristics.

This product is one of the most complete. It allows many configurations for each zone, can connect to four sensors (flow sensor, rain sensor or soil moisture sensor), is remotely available via an iOS app, an Android app or even a Web browser, and finally allows integration with many different services, such as Amazon Alexa⁷ and Google Assistant⁸.

By default, the controller receives weather data from a service named Aeris Weather Service⁹. However, the user can select another weather service.

---

⁴ https://www.myblossom.com/
⁵ http://netrohome.com/
⁶ http://rachio.com/
⁷ https://developer.amazon.com/alexa
⁸ https://assistant.google.com/
⁹ https://www.aerisweather.com/
If the internet connection fails, this device has a limited prediction saved in memory and irrigates as scheduled. The device also allows manual irrigation, using physical buttons, making the system usable even after the prediction schedule ends. When the device reconnects, the prediction is updated as usual.

### 3.2.4. RainMachine

RainMachine\(^{10}\) is a weather-based watering controller available worldwide. This product’s manufacturer freely provides the device’s source code, so that users can verify, modify and trust the device behaviour. A public API is also available, enabling users to create their applications and communicate with the controller.

This device does all calculations required itself, without having to rely on RainMachine servers. Therefore, internet connection is only necessary to connect to meteorological service providers. Remote user interfaces can connect to the devices directly, instead of connecting through RainMachine servers.

The controller collects weather data from one or more meteorological services providers, and the user can choose which ones to use from a list. If the user selects more than one service, a combined average is used [15].

### 3.2.5. Skydrop

Skydrop\(^{11}\) is a smart weather-based watering controller, officially supported in the United States, Canada, Australia and New Zealand. However, the company informs that the device will work anywhere with a Wi-Fi Internet connection and a weather station within a reasonable distance (using Weather Underground service) [16].

This device comes in two pieces: a back-plate, where all the wiring is done; and a front-piece, that has the controller screen and buttons.

During the initial configuration, the user is required to input a zip-code so that the system can select the best weather station. However, the system does not report the station it has chosen [17].

This device receives the weather conditions every hour through a connection to Skydrop’s Cloud via Wi-Fi [18]. If the internet connection fails, the system uses a watering calendar backup based on history.

The user can configure push notifications, e.g., receive notifications when a watering cycle starts or ends.

This controller cannot pause watering. So, if a user needs to stop a watering cycle, instead of recommencing where it was previously stopped and only water for the remaining amount that was required, it will start from the beginning. It also does not allow the user to manually initiate a watering for a specific zone and time frame.

---

\(^{10}\) [http://www.rainmachine.com/](http://www.rainmachine.com/)

\(^{11}\) [https://www.skydrop.com/](https://www.skydrop.com/)
### 3.2.6. Comparison

A comparison of the different devices previously presented can be seen in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>Blossom</th>
<th>Netro Smart</th>
<th>Rachio Smart Sprinkler</th>
<th>RainMachine HD-12, HD-16</th>
<th>Skydrop</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC WaterSense Certified</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Screen Interface</td>
<td></td>
<td></td>
<td>✓ (touch screen)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Outside protection</td>
<td>✓ (optional)</td>
<td>✓ (optional)</td>
<td>✓ (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zones</td>
<td>6, 12, 32</td>
<td>12</td>
<td>8, 16</td>
<td>12, 16</td>
<td>8, 16</td>
</tr>
<tr>
<td>Zone variables</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crop type</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Soil type</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Soil slope</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Solar exposure</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Nozzle type</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sensors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rain</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Soil moisture</td>
<td>✓ (proprietary)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Internet connection</td>
<td>Wi-Fi, Powerline</td>
<td>Wi-Fi, Ethernet, mobile modem</td>
<td>Wi-Fi</td>
<td>Wi-Fi</td>
<td>Wi-Fi</td>
</tr>
<tr>
<td>Remote access availability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>iOS</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Web</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Integrations</td>
<td>Amazon Alexa, Google Assistant</td>
<td>Nest, Amazon Alexa, Google Assistant, IFTTT, SmartThings</td>
<td>Amazon Alexa, IFTTT</td>
<td>Nest</td>
<td></td>
</tr>
<tr>
<td>Public APIs</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓ (opensource)</td>
<td></td>
</tr>
</tbody>
</table>

*Table 2. Smart irrigation products comparison.*
3.3. Weather forecast providers

Weather forecasting is a process that involves observing the current state of the atmosphere and analysing the collected data.

According to Vasquez [19], observation data is usually encoded using one of the following formats: METAR observations, Synoptic observations (SYNOP) or Surface aviation observations (SAO). Its elements are temperature, dew point, wind, pressure, visibility, weather and obstructions to vision, cloud height, cloud cover, cloud type, and flying conditions. This data is then processed using forecast models to generate predictions.

Currently, many weather providers offer weather predictions through websites, mobile applications, and radio and TV channels. Some allow users to search for forecasts by entering a region name or location coordinates, like Meteo|Técnico12, where there are hourly predictions for up to seven days. Other providers like The Weather Channel13 and AccuWeather14 can produce predictions up to a month. Services like Weather Underground15 even allow users to add their stations and share their data with other users.

Some weather providers supply APIs so that developers can embed weather forecasts into their apps. This resource is usually free up to a few requests per day. If that number of requests is not enough, a paid subscription is usually available.

3.4. Mobile application development

Choosing a development platform for a mobile application can be a hard task given the number of tools available and factors to be considered, e.g., budget, development skill and device functionality [20]. One can choose between developing a native application for each operative system (OS) that wants to support, or use a cross-platform framework, creating a single application which different OS can use.

In the following, we will address the differences and pros and cons of the various mobile development approaches.

3.4.1. Native development

Native apps have binary executable files that users usually download from an app store. Those apps are stored and installed on the device [21], and interface directly with the OS, without any intermediary or container and can access all application programming interfaces (API) available by the OS vendor.

12 http://meteo.tecnico.ulisboa.pt/
13 https://weather.com
14 https://www.accuweather.com
15 https://www.wunderground.com
To create a native app, the developer writes the source code and uses tools provided by OS vendors, to compile the code and pack it with any additional resources, e.g., images, audio files.

OS vendors provide their developing tools, usually as software development kits (SDK). For example, for iOS\textsuperscript{16}, XCode\textsuperscript{17} is used with the programming language Swift\textsuperscript{18} or Objective-C\textsuperscript{19}; for Android\textsuperscript{20}, Android Studio\textsuperscript{21} is used with the programming language Java\textsuperscript{22}.

These differences result in the most prominent disadvantage of the native development approach – code written for one mobile platform cannot be used on another, forcing developers to maintain separate development cycles for each platform (see Figure 9). However, creating and maintaining multiple applications is an expensive process.

![Diagram](image)

*Figure 9. Native development model, adapted from [22].*

However, from the user point-of-view, native applications provide the best experience, as they exhibit fast performance, good look and feel, and have access to the entire device hardware and data [23].

### 3.4.2. Cross-platform development

The cross-platform development aims to help developers create applications for different platforms. According to Raj and Tolety [23], the four primary cross development approaches are the web approach, the hybrid approach, the interpreted approach and the cross-compiled approach. Independently of the approach, the point is to improve the development cycle process, by sharing a part-of or totality-of code for the different platforms (Figure 10).

---

\textsuperscript{16} https://developer.apple.com/ios/
\textsuperscript{17} https://developer.apple.com/xcode/
\textsuperscript{18} https://developer.apple.com/swift/
\textsuperscript{19} https://developer.apple.com/documentation/objectivec
\textsuperscript{20} https://developer.android.com/
\textsuperscript{21} https://developer.android.com/studio/index.html
\textsuperscript{22} https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html
Web approach.

Web apps use standard web technologies like HTML5, CSS, and JavaScript. Users can access these apps by navigating to a website through the smart device's browser, without having to install them. Therefore they are cross-platform by default.

This approach uses languages that are easy to learn and develop. Moreover, developers can easily maintain and update the app since the server hosts all application files.

However, this approach comes with some issues: web view rendering is slow compared to native rendering; internet connection is required to access and use the app; many native APIs cannot be accessed.

Hybrid apps.

The hybrid approach combines both native and web development. On the one hand, using web technology allows developers to write significant amounts of code that will work cross-platform, on the other hand, native development allows direct access to native APIs.

Like native apps, hybrid applications require an installation. Those applications run inside a native container that, using the operative system native APIs, creates an embedded HTML rendering engine (WebView) to provide the user interface through a browser [21].

This approach can provide full access to the device’s APIs through an abstraction layer [23], execute offline and provides a user interface that developers can reuse across platforms. However, hybrid apps are slower than native, since they execute inside a browser. They will also lack the look and feel of native apps.

Interpreted approach.

The interpreted approach allows developers to program with a non-native, interpreted language such as JavaScript, and run it on the different mobile platforms.

Like hybrid apps, there is a container that runs the interpreted language and provides bindings to call native APIs. For example, if the interpreted language is JavaScript, then JavaScriptCore Virtual
Machine\textsuperscript{23} can be used as a container [24]. However, instead of running the interface through a WebView, interpreted apps use platform-specific native user interface elements to generate the user interface.

By creating the user interface with native elements, interpreted languages will look and feel like native apps and will run faster than hybrid applications.

**Cross compiled approach.**

In this approach, developers can write the program in a common language that is converted at build-time to native binary with a cross-compiler, unlike interpreted applications that use an interpreter at run-time.

This approach allows the application to access all device features and maintain performance comparable to native. However, developers cannot reuse the user interface, nor reuse access to specific platform features, as those are different for each platform.

### 3.4.3. Comparison

Table 3 presents a comparison of mobile development approaches. Each has its benefits and trade-offs that the software architect must consider before choosing. Furthermore, if low battery consumption is a must, native development should be considered, as the adoption of cross-platform solutions implies an increase in energy consumption [25].

<table>
<thead>
<tr>
<th>Approach</th>
<th>Native</th>
<th>Web</th>
<th>Hybrid</th>
<th>Interpreted</th>
<th>Cross compiled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development language</td>
<td>Swift, Objective-C, Java</td>
<td>HTML5, CSS, JavaScript</td>
<td>HTML5, CSS, JavaScript</td>
<td>JavaScript</td>
<td>C#, C++, JavaScript</td>
</tr>
<tr>
<td>Code portability</td>
<td>None</td>
<td>High</td>
<td>High</td>
<td>Medium-High</td>
<td>Medium</td>
</tr>
<tr>
<td>Access device features</td>
<td>High</td>
<td>Low</td>
<td>Medium-High</td>
<td>Medium-High</td>
<td>High</td>
</tr>
<tr>
<td>Upgrade flexibility</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>Low-Medium</td>
</tr>
<tr>
<td>Look and feel</td>
<td>Good</td>
<td>Poor</td>
<td>Medium</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

Table 3. Comparison between native, web, hybrid, interpreted and cross-compiled approaches [20, 23].

\textsuperscript{23} https://developer.apple.com/documentation/javascriptcore
3.4.4. Mobile development tools

As we will discuss in Chapter 4, we are planning to develop an application for Android and iOS, as those systems predominantly occupy the mobile’s operating systems market, with a combined share of 94% of the market, as of December 2017 [26].

Therefore the cross-platform approach best suits our needs, as source code can be reused across platforms, facilitating development cycle.

Web apps have insufficient access to the device native APIs, and, furthermore, cannot be accessed offline. Cross compiled apps require specific user interfaces and API accesses to be created for each platform, slowing down the development process. That leaves us with hybrid apps and interpreted apps. There are many development tools available for those approaches; therefore, before proposing a technology, we decided to test one hybrid framework and one interpreted framework.

For this test, we created two similar applications, one for each framework. Those applications represent the basis required for our solution, and we ran it on different devices. Both applications consist of three views (Figure 11):

- The first view displays a “Hello World” message and a button to switch to the next view;
- The second view obtains the GPS location, by accessing the native mobile API, and displays latitude and longitude, and a button to switch to the next view, providing the coordinates;
- The third view creates an HTTP request to a remote weather data provider (with the given coordinates), parses the JSON response and displays the current weather data.
Hybrid app approach

We created the first application with Ionic\textsuperscript{24}, an open source framework for hybrid mobile development. This framework is built on top of Apache Cordova\textsuperscript{25}, a hybrid mobile development framework, and AngularJS\textsuperscript{26}, a front-end web application framework. It took us a day-work to understand the basic platform operation, develop the test app, configure the devices, run the app and test it.

Interpreted approach

The second application was developed with React Native\textsuperscript{27}, an open source framework to build interpreted mobile apps with JavaScript. This platform uses specific native APIs to construct the user interface. Therefore the developer must choose the appropriate interface blocks for each operating system. However, React Native provides an option to render common user interface controls, where those controls behave alike on both platforms, to facilitate the development cycle. Taking advantage of those features, we created an almost non-platform specific code that ran on both OSs. The only specific code that we had to make was a run-time Location Permission Request\textsuperscript{28} for Android. This approach, likewise to the previous one, also took us a day-work to understand the basics, develop, run and test the app.

Tests and conclusions

Nearly all the code created was run independently of the operating system. We tested the applications on two Android devices and one iOS emulator (Table 4), without any performance tweaks, other than building the applications in release mode, i.e., deactivating development tools.

The apps created with Ionic and React Native, look similar across devices (Appendix B), with a few exceptions:

- React Native uses native UI elements, and, for this reason, we can see a difference in the buttons\textsuperscript{29}: on the one hand, Android fills the button’s background with a given colour; on the other hand, iOS sets the button’s text colour with the given colour, maintaining a white background;
- Ionic uses web elements to generate the interface, therefore is dependent on the device’s browser rendering engine. We can see that the buttons have different sizes across devices.

\textsuperscript{24}https://ionicframework.com/
\textsuperscript{25}https://cordova.apache.org/
\textsuperscript{26}https://angularjs.org/
\textsuperscript{27}https://facebook.github.io/react-native/
\textsuperscript{28}https://developer.android.com/training/permissions/requesting.html
\textsuperscript{29}https://facebook.github.io/react-native/docs/button.html
We could have removed those differences, to make them look similar. However, it was out of the test purpose.

We noticed that one of the applications had a loading time lighter than the other, so we decided to test loading times. Both apps were launched three times on each physical device (LG Nexus 5x and Asus Nexus 7), and we recorded the process. Table 5 presents the average loading time values from those tests.

<table>
<thead>
<tr>
<th>Load time (seconds)</th>
<th>Ionic</th>
<th>React Native</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LG Nexus 5x</td>
<td>Asus Nexus 7</td>
</tr>
<tr>
<td>Test 1</td>
<td>3.025</td>
<td>5.018</td>
</tr>
<tr>
<td>Test 2</td>
<td>2.899</td>
<td>4.797</td>
</tr>
<tr>
<td>Test 3</td>
<td>3.121</td>
<td>4.899</td>
</tr>
<tr>
<td>Average</td>
<td>3.015</td>
<td>4.905</td>
</tr>
</tbody>
</table>

Table 5. App loading time comparison (lower is better).

The app created with React Native loads faster than the one created with Ionic, on all devices. More interestingly, the React Native app load times are similar on both devices, contrasting with the Ionic app that increases load time significantly when using the older device (Asus Nexus 7).

Even though both approaches took us nearly the same time to build and look similar across devices, we believe that with app growth, developing with React Native will require more time to design the user interface than with Ionic, as, most-likely, OS-specific UI components will be necessary. However, the look and feel provided by React Native will be superior due to the use of those native elements compared to Ionic’s web rendering.

30 http://www.lg.com/uk/mobile-phones/lg-H791
31 https://www.asus.com/Tablets/Nexus_7/specifications/
32 https://support.apple.com/kb/sp727
Furthermore, both approaches have their way to provide a consistent layout for different screen sizes: Ionic’s Responsive Grid and React Native’s Flexbox. Therefore, designing for various targets is not an issue for either.

Taking into consideration all the above information, we decided to use React Native. Not only it looks and feels like a native application but also runs faster. This is why well-known companies prefer React Native, e.g., Facebook, Airbnb, Skype, and Tesla. Also, there is active community support.
4. Project requirements

In this chapter we present the intended requirements for our product, stating what our product should do or how it should perform. Jenny et al. [27] present a narrow list of different requirements, representative of what needs to be captured, that we will follow. In Appendix C, a more detailed list is present.

Track watering needs

Irrigation clock systems work without any knowledge of the real-world environment variables, generating dummy schedules such as \( x \) amount of water for every \( y \) time. An essential feature of any smart irrigation is guessing when irrigation is needed to avoid water loss and achieve a healthy crop development. Therefore, our system must be able to predict crop watering needs.

Automated irrigation schedules

The tracking of watering needs is exceptionally complicated due to the number of variables required to do so and how quickly those may change with crop development and weather factors, e.g. maximum daily temperature or soil moisture content.

Irrigation schedules are directly related to the watering needs – after the calculation of the amount of water required for a day, it is necessary to study the best timing to provide water fitting the user irrigation system. This system must generate or update a watering schedule automatically when a noticeable variable change occurs.

Multiple weather data providers

Any system that is dependent on another system is vulnerable if the last one misbehaves, e.g. data errors or downtimes, either scheduled or not. To lower the odds of unintended results, one can avoid or, at least, mitigate by supporting other systems that can work as replacements.

This system is highly dependent on weather forecasts to predict and track watering needs and, therefore, irrigation schedules. Misbehaviours from outside systems could cause over-irrigation, or worst, under-irrigation. Therefore, the system must be able to collect data from different weather providers and offer the option to choose which one to use.

Moreover, each weather provider has its interface to supply data, e.g., different file formats and metrics. For that reason, this system must implement a driver for each weather provider to collect data and, afterwards, normalise this data to a project common.

Save weather predictions and watering schedules

Analysing previously recorded data and decision making based on such data can help improve systems. For instance, algorithms can be adjusted or improved, or even artificial intelligence can be introduced to understand how variables change and how to react to changes appropriately. Furthermore,
it allows the recreation of scenarios to help developers discover and fix software errors. Even if the data is not useful in the short term, it can become highly relevant in the long term.

In irrigation systems, recorded data can be used to show statistics to users such as crop progress, water used and saved, and use it as a motivation for users to feel that the system is useful and helpful.

Therefore, the system must save collected weather forecasts and generated watering schedules for future use.

**Remote control**

With the advancement of technology, many tasks became remotely achievable, that is, users are no longer required to move to a specific location to perform them, resulting in time and economic savings. This system should also follow such path; that is, users must be able to remotely control irrigation, providing users with the ability to simultaneously control multiple fields, access while on vacations, and so on.

For this functionality, the system must provide a public application programming interface (API), to which software applications will connect and interact. Moreover, smartphones invaded people’s lives and opened the opportunity to do hard tasks with a sturdy pocket device. This project must provide an application that covers most of the mobile market share: Android and iOS.

**Multiple controllers**

Smart irrigation solutions usually have a controller, a device that connects to pumping stations, valves and sensors, and saves irrigation schedules in non-volatile memory.

Those controllers have physical limitations as their hardware can only support a limited number of valves. If that limit is not enough, a new controller is then required, to be installed together with the previous one.

However, current solutions do not have scalability adequately thought out, as shown in Chapter 3, as those controllers will not know the existence of each other. Our product will do away with this restriction, by creating a way for users to have multiple smart controllers that collaborate, sharing vital information for better collective work.

**Language and units**

When designing a product, software engineers must consider many factors for its success, such as scalability, maintainability and security. Moreover, some factors, like usability, require the knowledge of the targeted consumer group.

This system’s interface must be written in English, one of the most widely spoken languages, and must support different unit metrics, that can be chosen by the user, to achieve a significant group of users.
5. Solution overview

In this chapter, we will present our software architecture - “a set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both” [28].

5.1. Domain model

To fulfil the requirements presented in Chapter 4, we created a domain model - a conceptual model that aims to describe and model real-world concepts and their behaviour, representing the problem domain space. This model was designed carefully so that this software solution might be easily maintainable, testable and upgradable in the future (see Appendix E).

Our project relies on this model, to persist data that can be inserted, updated or deleted in runtime. In the following, we will describe each entity created.

User

This entity is a representation for each user of our system, storing user personal information. Since users will interact with our platform, they are required to authenticate, so our system can identify them and prove their identity. This authentication process requires a username and a password, that, for security reasons, is one-way hashed.

Besides the username, users are also required to have an e-mail address. On the one hand, the e-mail address is required to recover an account, send notifications and important messages. On the other hand, the system requires a username to share content with other users without exposing e-mail address.

This entity is associated directly or indirectly with every other entity, as everything else is associated with a user and depends on user preferences and actions.

Weather Provider

Weather providers are a crucial part of our system, as those are used to obtain weather predictions and, consequently, generate irrigation plans. Users are allowed to choose their preferred weather provider source. Therefore this entity is required and contains its name, a description and its current availability.

Area

An area is an entity that represents a real-world terrain – a stretch of land with different slopes, soil textures and crop types, where irrigation systems are installed.

Some parts of an area share similar conditions and watering needs. Therefore, users can group those into zones that will then be treated alike when planning irrigation. Moreover, zones share a close-by location. Therefore, the weather forecast predictions are the same.

Consequently, a name, a location and a list of zones define this entity.
**Soil Texture**

The different minerals present in the soil define its texture. As discussed in Chapter 2, space in between those particles can be filled with water, defining the amount of water that soil can hold. This entity aims to represent the soil texture, containing the theoretical maximum and minimum values of available water (field capacity and wilting point).

**Crop**

Each crop has its watering requirements, as discussed in Chapter 2. Therefore the user must define what kind of crop it is, and when was it cultivated. This entity specifies crop details such as name, maximum root depth and height, and evapotranspiration coefficient.

**Controller**

This entity represents a hardware module device with a non-volatile memory to which the pumping station, valves and sensors are connected, that can be acquired and used by consumers. Each controller has its serial number - a random string, non-sequential - the users use such serial numbers during the association process.

**Controller Part**

As previously stated, a controller connects to pumping stations, valves and sensors. A controller part represents each of those connections in this entity. By doing so, a user can combine parts from different controllers to create and setup zones, allowing intercommunication between controllers by default.

**Zone**

Each zone is a piece of an area, where the crops share the same watering needs and conditions, and consequentially, the irrigation plan. Therefore, this entity stores plantation date, soil slope and texture, crop type, and the controller parts associated to it: a pumping station and a valve, and a soil moisture sensor.

**Schedule**

A schedule is an entity directly related to a zone that represents its watering program for the ensuing days. This entity contains the weather data collected, the result of the calculation using previous data evapotranspiration, the daily water balance, and irrigation start and end times.
5.2. Components

As depicted in Figure 12, the system architecture is split into three components: a server, an irrigation controller, and a mobile application.

Server

The server is a cloud-based application that manages irrigation controllers, generates watering schedules, receives weather forecasts and provides a remote interface for mobile applications.

This server is responsible for receiving data from Irrigation Controllers sensors, parse that data and send updated watering schedules to the Irrigation Controller and its owner. Actuators can also be invoked by the server if so allowed by the user (e.g., the user wants to water a zone for five minutes, with immediate effect).

As discussed in Chapter 2, one can predict the water usage of a crop using a combination of the weather forecast, crop type and development, and soil characteristics. Moreover, as examined in Chapter 3, one can collect data from moisture sensors to verify if a crop needs irrigation, based on a threshold. The server combines the previous two methods, predicting when irrigation is required and confirming the predictions by measuring soil moisture. Therefore, to generate watering schedules, the server needs to borrow user zone configurations, weather data, and soil moisture levels.

Weather forecasts are requested by the server, for each required location, from different sources. These sources usually provide different APIs; therefore, the Weather Data Normalization component will adjust the information so that Schedule Manager can use it.
The Cloud Server do also provide an API for remote access from mobile applications so that users can configure irrigation controllers, receive watering schedules information and view statistics.

**Irrigation Controller**

The Irrigation Controller is emulated in this project, as well all sensors and actuators connected to it. This controller is connected to the Cloud Server and provides the following functionalities: read sensor values and return data values; invoke actuators; update the watering schedule; and, water or fertigate accordingly to schedule.

The sensors for this solution are: flow sensors, to measure the amount of water that is used during a watering cycle; rain sensors, to immediately stop irrigation in case of rainfall; and, soil moisture sensors, to measure the amount of water present in the soil. The actuators for this solution are valve actuators: for both primary and secondary conducts so that the system can water a specific zone; and, for fertiliser injection, so that the right fertiliser is added to the water supply.

This device saves watering schedules in non-volatile memory so that schedules may resume after a power outage. Furthermore, watering schedules will be for a week, at least, so that watering cycles continue even with network failures.

**Mobile App**

The mobile application allows users to add new irrigation controllers and configure them, view watering and fertiliser schedules, manually start or pause irrigation and receive notifications of events.

### 5.3. Communication protocols

For this proposal, the Cloud Server is both a client and a server, as depicted in Figure 13. On the one hand, Cloud Server behave as a server for irrigation controllers and mobile applications, as it is the component providing services and does not know about the client’s identity’s. On the other hand, the cloud server behaves as a client of weather forecast providers, since it knows about the selected weather forecast server identity. Although Irrigation Controllers and Mobile Applications may be clients of Cloud Server, those will follow different protocols.

Irrigation Controllers must identify themselves to the server when they start, as their internet protocol (IP) address can change or be unknown (e.g., new controllers), and need a communication protocol that allows the server to send messages without a client request (e.g., start irrigation now). Furthermore, Irrigation Controllers can be behind firewalls, excluding the possibility to work as a remote server out-of-the-box. For this reason, a WebSocket communication protocol is used, allowing bidirectional conversation.
Mobile Applications only connect to the server as needed. Therefore a client-pull approach can be used. For this case, we will use representational state transfer (REST).

Weather Providers have their APIs and protocols. Usually, the client creates an HTTP request with the data (e.g., API key, location) and receives a JSON or XML file, using a client-pull strategy.
6. Solution implementation

In this chapter we describe how the system was implemented, pointing out the challenges encountered during the development phase and the adopted solutions to overcome.

6.1. Server

The server application is the main component of this system, as everything else depends on its correct operation. As mentioned in the previous chapter, this component exposes services to which other devices connect to and interact with. Those services are available through a RESTful API – an application programming interface that uses hypertext transfer protocol (HTTP) requests GET, to view; PUT, to update; POST; to add; and DELETE, to delete data (see Appendix D).

6.1.1. Development tools

To develop the server component, we used Node.js\(^{33}\), a JavaScript\(^{34}\) framework designed to build scalable network applications, with a large community-maintained open source plugin library that we used, as listed in Table 6.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bcrypt</td>
<td>Hash and compare passwords</td>
</tr>
<tr>
<td>Lodash.clonedeeep</td>
<td>Recursively clone array values</td>
</tr>
<tr>
<td>Moment-timezone</td>
<td>Parse and display dates in any time zone</td>
</tr>
<tr>
<td>Mongoose</td>
<td>MongoDB object modelling tool</td>
</tr>
<tr>
<td>Node-schedule</td>
<td>Schedule jobs for execution at specific dates</td>
</tr>
<tr>
<td>Restify</td>
<td>REST framework</td>
</tr>
<tr>
<td>Restify-errors</td>
<td>Collection of errors shared across restify components</td>
</tr>
<tr>
<td>Passport-restify</td>
<td>Authentication middleware compatible with restify</td>
</tr>
<tr>
<td>Passport-local</td>
<td>Local username and password authentication strategy for passport</td>
</tr>
<tr>
<td>Eslint</td>
<td>JavaScript pattern checker</td>
</tr>
<tr>
<td>Eslint-config-airbnb</td>
<td>Airbnb’s Eslint config</td>
</tr>
<tr>
<td>Mocha</td>
<td>Test framework for Node.JS</td>
</tr>
</tbody>
</table>

Table 6. Packages used to develop the server among NodeJS.

\(^{33}\) https://nodejs.org/en/
\(^{34}\) https://developer.mozilla.org/en-US/docs/Web/JavaScript
6.1.2. Database

The persistent data system used for the server component is MongoDB\(^{35}\), an open-source, cross-platform database, that uses JSON documents to store and retrieve data records. The server communicates with the database through mongoose\(^{36}\), an object document mapper (ODM) that allows the creation of schema models, documents, and validations and performs queries and triggers events.

**Schema**

A schema defines the shape of a collection of documents within a MongoDB collection. Each time an entity creates a document, it must fulfil the schema requirements, e.g. as seen in Figure 14, *username*, *name*, *email*, *hashedPassword* and *registerDate* are required, and, therefore, will be needed to create a user document.

```javascript
const UserSchema = new Schema({
  username: {
    type: String, unique: true, trim: true, required: true,
  },
  name: {
    type: String, required: true
  },
  email: {
    type: String, unique: true, lowercase: true, trim: true, required: true,
  },
  hashedPassword: { type: String, select: false, required: true },
  registerDate: { type: Date, required: true, default: Date.now },
  controllers: [[ type: Schema.Types.ObjectId, ref: 'Controller' ]],
  areas: [[ type: Schema.Types.ObjectId, ref: 'Area' ]],
});
```

*Figure 14. User model schema, containing a username, a name, an e-mail, a hashed password, the registry date, the list of controllers associated with such user and the user areas.*

**Validations**

An essential step in database development is the creation of validations that a system must perform when an entity creates or modifies a document. This validation confirms that the data that is being added or changed complies with specified constraints. For instance, a User’s username must be alphanumeric, its length must be within a given range, and it must be unique, see Figure 15.

---

\(^{35}\) [https://www.mongodb.com/](https://www.mongodb.com/)

\(^{36}\) [https://mongoosejs.com/](https://mongoosejs.com/)
Figure 15. User model validations: check if the username is alphanumeric, verify that username meets minimum and maximum length and confirm that there are no other users with the same username.

6.1.3. Weather prediction

This system allows its users to choose a weather provider for each area from a list. Each available weather provider has its application programming interfaces. Therefore, custom drivers were designed to communicate and request data (see Appendix F).

Meteo|Técnico

Meteo|Técnico is a weather provider available for the Portuguese territory, and its clients can request predictions for specific location coordinates. The system connects to it through a private application programming interface (API) with previously acquired access credentials. This provider’s API returns a JSON document with a forecast for the next seven days.

WunderGround

WunderGround services are available worldwide, and its users can use it through a free API for a limited number of requests. The server can request predictions by providing its user key and a location - either by coordinates or name. The response is a JSON document that includes a 10-day forecast, however, to normalise data, the system only uses seven days, discarding the remaining.

Test

We developed a weather provider simulator, intended to be used during the development and testing of this system. This provider generates random weather values with some cohesion, e.g. daily minimum temperature is lower than the daily maximum temperature. However, this provider generates days without taking into consideration the previous day, i.e., abnormal temperature changes may occur.

With this provider generating random weekly predictions, we were able to achieve odd results that were used to test and bullet-proof our irrigation algorithm.
6.1.4. Algorithms

The server component automatically generates watering schedules, by using a combination of soil and crop properties and weather data predictions.

As mentioned in Chapter 2, the daily reference evapotranspiration can be calculated using the weather data as an input to the FAO Penman-Monteith correlation. Then, using crop properties, crop evapotranspiration can be predicted.

With the daily crop evapotranspiration calculated for a given timeframe, the water balance equation is then used to figure if irrigation is required and, if so, how much to irrigate.

Finally, with the weekly irrigation requirements, a schedule can be generated. This schedule takes into account user preferred irrigation times, and maximum irrigation flow available (pumping station and valves).

**Irrigation amount**

Irrigation amount is an algorithm that we developed to compute the amount of watering required for a timeframe, see Table 7. Given the input list of one or more days, this algorithm generates an output list with the amount of irrigation to provide for each day.

With the calculation of daily water balance, the irrigation needs can be estimated, by predicting the soil water depletion at the end of each day and verifying if its value reaches the management allowed depletion (MAD), as seen in Figure 16.

![Water depletion graph](image)

*Figure 16. Water balance example, without irrigation and with irrigation, on the left and right respectively.*

A list is created to add every consecutive day with a rising depletion above management allowed depletion (MAD). When depletion stops increasing or the day's list is over, the last day in the list (the day with the maximum depletion) is then selected to calculate how much irrigation is required in provide to the first day in the list (the day when depletion surpassed MAD). The algorithm repeats this process until the list of days is exhausted.
**Input:** A list $I$ of items and it’s dimensions $(D_{\text{ini}}(I_i), D_{\text{end}}(I_i), \text{MAD}(I_i), \text{ETc}(I_i), Pr(I_i), Ir(I_i)))$

**Output:** Items $I$ irrigation amount.

| 1. If $D_{\text{ini}}(I_i) > \text{MAD}(I_i)$ then |
| 2. $\text{Ir}(I_i) ← D_{\text{ini}}(I_i) + \text{ETc}(I_i) - \text{MAD}(I_i)$ |
| 3. $D_{\text{end}}(I_i) ← D_{\text{ini}}(I_i) + \text{ETc}(I_i) - \text{Ir}(I_i)$ |
| 4. end if |
| 5. If $D_{\text{end}}(I_i) > \text{MAD}(I_i)$ then |
| 6. $\text{Ir}(I_i) ← D_{\text{ini}}(I_i) + \text{ETc}(I_i) - Pr(I_i) - \text{MAD}(I_i)$ |
| 7. $D_{\text{end}}(I_i) ← D_{\text{ini}}(I_i) + \text{ETc}(I_i) - \text{Ir}(I_i) - Pr(I_i)$ |
| 8. end if |
| 9. $i ← 1$ |
| 10. let ConsecutiveDays be a list of consecutive days below MAD |
| 11. while $i < \text{length}(I)$ do |
| 12. $D_{\text{ini}}(I_i) ← D_{\text{end}}(I_{i-1})$ |
| 13. $D_{\text{end}}(I_i) ← D_{\text{ini}}(I_i) + \text{ETc}(I_i) - \text{Ir}(I_i) - Pr(I_i)$ |
| 14. if $D_{\text{end}}(I_i) < 0$ then |
| 15. $D_{\text{end}}(I_i) ← 0$ |
| 16. end if |
| 17. added $←$ false |
| 18. if $D_{\text{ini}}(I_i) < D_{\text{end}}(I_i)$ and $D_{\text{end}}(I_i) > \text{MAD}$ then |
| 19. ConsecutiveDays $←$ ConsecutiveDays + $I_i$ |
| 20. added $←$ true |
| 21. end if |
| 22. if added $=$ true or $I_i$ is the last item then |
| 23. if $D_{\text{ini}}(I_i) < D_{\text{end}}(I_i)$ then |
| 24. irrigationValue $←$ $D_{\text{end}}(I_i) - \text{MAD}(I_i)$ |
| 25. else |
| 26. irrigationValue $←$ $D_{\text{ini}}(I_i) - \text{MAD}(I_i)$ |
| 27. end if |
| 28. $\text{Ir}(I_{\text{length}(\text{ConsecutiveDays})}) ← \text{irrigationValue}$ |
| 29. $j ← i - \text{length}(\text{ConsecutiveDays})$ |
| 30. while $j < \text{length}(\text{ConsecutiveDays})$ do |
| 31. $D_{\text{ini}}(I_j) ← D_{\text{end}}(I_{j-1})$ |
| 32. $D_{\text{end}}(I_j) ← D_{\text{ini}}(I_j) + \text{ETc}(I_j) - \text{Ir}(I_j) - Pr(I_j)$ |
| 33. $j ← j + 1$ |
| 34. end while |
| 35. end if |
| 36. $i ← i + 1$ |
| 37. end while |

**Table 7. Irrigation amount algorithm.**

This algorithm has some limitations which we propose as future work improvements in Chapter 8. On the one hand, weather predictions are not entirely accurate, and, furthermore, correlations used to calculate evapotranspiration are no more than approximations of the real values. On the other hand, the irrigation system calibration might not be correct, and, for instance, the soil may only absorb a portion of the irrigated water.
Irrigation sequence

The irrigation sequence algorithm aims to order and introduce concurrent irrigation for different zones. To figure which zones the system can group in a specific time frame, we used an algorithm that solves the bin packing problem, to pack different items in a bin so that the wasted space in the bins is minimum.

Figure 17. List of items with two dimensions on the left, and the same items packed in a bin, on the right, using the Sleator algorithm [29].

Ortmann [29, p. 119] lists different algorithms and their complexity and compares them. Our server was developed to run within NodeJS, a run-time environment that by nature is single threaded. Therefore, any CPU intensive tasks will be blocking the execution of others. With that in mind, we choose Sleator [30], an algorithm created in 1980, which had the lowest time complexity, $O(n \log n)$. 
**Input:** A list $I$ of items, the dimensions $(w(I_i), h(I_i))$ of those items and the strip width $W$.

**Output:** Items $I$ packed into a strip of width $W$.

| 1. sort items according to decreasing height |
| 2. stack all items of width $> 0.5 \times W$ |
| 3. let $i$ be the index of the first unpacked item, $\text{RemainingW} \leftarrow W$ |
| 4. while $w(I_i) \leq \text{RemainingW}$ do |
| 5. pack $I_i$ at a height of $h_0$ |
| 6. $\text{RemainingW} \leftarrow \text{RemainingW} - w(I_i)$ |
| 7. let $i$ be the index of the next unpacked item in $I$ |
| 8. end while |
| 9. determine the height of the left and right columns |
| 10. while there are unpacked items do |
| 11. $\text{RemainingW} \leftarrow 0.5 \times W$ |
| 12. if height of the left column $\leq$ height of the right column then |
| 13. while $w(I_i) \leq \text{RemainingW}$ do |
| 14. pack $I_i$ into the left-hand column |
| 15. $\text{RemainingW} \leftarrow \text{RemainingW} - w(I_i)$ |
| 16. let $i$ be the index of the next unpacked item |
| 17. end while |
| 18. else |
| 19. while $w(I_i) \leq \text{RemainingW}$ do |
| 20. pack $I_i$ into the right-hand column |
| 21. $\text{RemainingW} \leftarrow \text{RemainingW} - w(I_i)$ |
| 22. let $i$ be the index of the next unpacked item |
| 23. end while |
| 24. end if |
| 25. end while |

*Table 8. Sleator algorithm [29].*

The same way as for the previous algorithm, this one also has its limitations. In optimal conditions, the system should provide irrigation to use pumping station optimal flow by introducing concurrent irrigation of different zones, to achieve optimal energy efficiency. However, for this project, we introduced an algorithm that only arranges irrigation to accomplish irrigation in the shortest time. Moreover, it does not have a time limit, that is, the system could potentially schedule irrigation outside the users preferred time frame.

### 6.1.5. Software testing

Software testing is an essential step of any software development process, as it allows bug discovery and prevention, and brings reliability and quality, among others [31]. Several tests were created to verify the behaviour of the different components of this project, aiming to determine if they are fit to use, applying the dynamic testing black-box and white-box techniques.
Tests using a black-box technique or functional testing, only consider functional requirements, i.e. structure or logic of software is not considered. On the other hand, tests using a white-box technique or glass-box testing, are meant to expose everything that the developers are required to implement, i.e. the entire design, structure, and code of software have to be studied [31].

Figure 18 depicts an example of a white-box test, where a user model method is invoked to create a new document with a non-alphanumeric username, expecting an exception to be thrown. Since the database has a username validator, the system will not create the document, and an error is thrown, passing the test.

In Figure 19, an example of a black-box test is shown, where an hypertext transfer protocol (HTTP) request is made to the server representational state transfer (REST) application programming interface (API) to register a new user account. However, similarly to the previous test, the username provided in the request body is not alphanumeric, therefore the HTTP response status is 409 (Conflict - used in situations where the user might be able to resolve the conflict and resubmit the request), and the HTTP response body contains a message referring to the alphanumeric error, passing the test.

6.1.6. Deployment environments

During the creation and delivery of software, there are many development phases, in which each requires different environment configurations. For instance, the domain name, the server certificate, the project folder path or even log files detail level. For this server, we used the following environments:
• **Development.** Where speed is not a concern and the information is presented to help the developer while programming. In this environment, the database gets cleaned up and populated with default data each time the server starts.

• **Test.** As previously mentioned, this project contains unitary tests, and, therefore, requires a test environment, where the database is cleaned up before executing any test.

• **Production.** When the application is ready to be used by the clients, the system is set to the production environment. Performance and security are some of the must qualities for the final product. Detailed logs shall not exist, only basic messages or errors that are essential - information must be carefully printed or saved. Furthermore, the database shall be untouched when starting or restarting the app.

6.1.7. Logs

Logging is a crucial part of any server. Not only can it help solve possible bugs but also track unintended accesses, e.g. a user edits something that he/she should not be able to, see Figure 20. A library named `log4js`\(^{37}\) was used to generate logs, which provides a list of different log levels: trace, debug, info, warn, error and fatal.

```
at Area.findOne.then.then.area (c:\Projects\raindrop\server\app\controllers\zones.js:2:1359)
at <anonymous>
jse_shortmsg: 'Area not found.',
jse_info: {},
message: 'Area not found.',
body: { code: 'ResourceNotFound', message: 'Area not found.' },
context: null }
[2018-06-20T11:23:42.022] [INFO] development - [pmf] Zone created.
```

*Figure 20. Example of server logs in the development environment.*

For each previously described environment, different logs levels were defined, as Table 9 shows.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Log types</th>
<th>Console Print</th>
<th>File Print</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>trace, debug, info, warn, error and fatal</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Test</td>
<td>trace, debug, info, warn, error and fatal</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Production</td>
<td>info, warn, error and fatal</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

\(^{37}\) https://github.com/log4js-node/log4js-node
6.2. Mobile app

6.2.1. Development tools

The mobile application was developed with React Native, a cross-platform mobile development framework. Identically to the server component’s development tool, React Native also has a great community-maintained open source plugin library that we used, as listed in Table 10.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>React-native-camera</td>
<td>Camera module</td>
</tr>
<tr>
<td>React-native-datepicker</td>
<td>Date picker component</td>
</tr>
<tr>
<td>React-native-elements</td>
<td>Set of components for React Native</td>
</tr>
<tr>
<td>React-native-geocoder</td>
<td>Geocoding services</td>
</tr>
<tr>
<td>React-native-keychain</td>
<td>KeyStore access</td>
</tr>
<tr>
<td>React-native-maps</td>
<td>Map component</td>
</tr>
<tr>
<td>React-native-popup-menu</td>
<td>Extensible popup menu component</td>
</tr>
<tr>
<td>React-native-qrcode-scanner</td>
<td>QR Code scanner using react-native-camera</td>
</tr>
<tr>
<td>React-native-step-indicator</td>
<td>Step indicator widget</td>
</tr>
<tr>
<td>React-native-svg</td>
<td>Drawing SVG interface</td>
</tr>
<tr>
<td>React-native-vector-icons</td>
<td>Bundle of scalable icons</td>
</tr>
<tr>
<td>React-navigation</td>
<td>App navigation system</td>
</tr>
<tr>
<td>Moment</td>
<td>Date parse, validate, manipulate and format</td>
</tr>
<tr>
<td>Convert-units</td>
<td>Convert quantities in different units</td>
</tr>
<tr>
<td>Node-fetch</td>
<td>Interface for fetching resources</td>
</tr>
</tbody>
</table>

Table 10. Packages used with react native to develop the mobile application.
6.2.2. App layout

**Colours**

To choose the applications’ colour palette we used the split complementary harmony, a combination of a key colour or primary colour with two others directly on either side of the complementary colour, introducing more visual interest and variety [32].

The primary colour that we picked is blue, representing the most significant challenge – water. After that, using the previously mentioned harmony, the secondary and tertiary colours, brown and red respectively, were generated, as seen in Table 11. Furthermore, we created different shades for each of the previous colours.

<table>
<thead>
<tr>
<th></th>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lighter</td>
<td>#345B84</td>
<td>#846734</td>
<td>#843E34</td>
</tr>
<tr>
<td>Normal</td>
<td>#003366</td>
<td>#664100</td>
<td>#660E00</td>
</tr>
<tr>
<td>Darker</td>
<td>#002952</td>
<td>#523300</td>
<td>#520C00</td>
</tr>
</tbody>
</table>

*Table 11. Colour palette used in the mobile app.*

**Navigation**

The mobile application has two primary states: logged-in and logged-out that represent if the user is authenticated or not, as seen on Figure 21.

*Figure 21. Raindrop mobile app view navigation.*

When a user first opens the app in a device, and it is not yet logged in, it is in the logged-out state, where the user can interact with two views: login and register, as depicted in Figure 22. The register view allows users to create new accounts given a username, a password, an e-mail address and a full name. On
the other hand, the login view allows users to authenticate themselves by providing their username and password.

![Login view and register view](image)

**Figure 22.** Logged out state, from the left to the right: login view and register view.

After the login process, the user enters the logged-in state, a restricted space containing areas - a view with the list of areas, their zones and weather and irrigation predictions; devices - a list of associated controllers; and, profile – account customisation and app settings.

### 6.2.3. App views

This section presents the most relevant views created for the mobile app when a user is logged in.

**Areas**

After a successful login, the user enters the logged-in state. The default navigation tab is Areas, a view containing a list of areas, with their names, images and watering needs, and a button to create a new area, as seen in Figure 23.

Each area shown has a list of associated zones and their weekly schedules. For each scheduled weekday, a watering can is displayed if irrigation is required, or an x mark otherwise. The ratio of water required for the maximum water that the plant can absorb defines the amount of water present in each watering can.
Area

When a user chooses an area from the previous list, a new screen is presented, showing the selected area, with a list of zones, weather predictions and area location, as depicted in Figure 24.

Each zone listed in the area view has a cup with water. The amount of water present in the cup is the ratio of the sum of the required water for the week to the maximum water that the plant can absorb in a week.
Weather predictions contain information, for each weekday, of maximum and minimum temperatures, relative humidity, precipitation, reference evapotranspiration and average wind speed. Furthermore, an icon is drawn to represent the weather condition, given the percentage of clouds and the predicted rain, see Table 12.

<table>
<thead>
<tr>
<th>Description</th>
<th>Sunny</th>
<th>Partly Cloudy</th>
<th>Cloudy</th>
<th>Rainy</th>
<th>Pouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clouds (%)</td>
<td>&lt;= 10</td>
<td>&gt; 10 and &lt; 60</td>
<td>&gt; 60</td>
<td>&gt; 10</td>
<td>&gt; 10</td>
</tr>
<tr>
<td>Rain (mm)</td>
<td>&lt; 2</td>
<td>&lt; 2</td>
<td>&lt; 2</td>
<td>&gt; 2 and &lt; 7</td>
<td>&gt;= 7</td>
</tr>
</tbody>
</table>

**Table 12. Weather icons decision table, using cloud percentages and rain amount.**

**Zone**

A zone view contains a list of days where irrigation is required, a graph representing the water content present in the soil, a graph of daily water balances and the details of such zone - the crop type and age, the soil texture and area, and the maximum flow for irrigation, see Figure 25.

![Zone details](image)

**Figure 25. Zone details.**

The first card shown to the user is the irrigation plan: when to start, for how long and how much. Likewise, as previously described in the areas listing, the watering can also is used here to represent the required irrigation, as depicted in Figure 26.

![Zones watering schedule example](image)

**Figure 26. Zones watering schedule example.**

A significant challenge during graphs design is transforming and transmitting considerable amounts of data in such a way that allows a more straightforward interpretation by the user. That is the case of graphs for irrigation that, as previously mentioned, take into account many variables and constraints.
We designed two graphs – one representing the predicted amount of water present in the soil and the other providing the water balance given irrigation, precipitation and evapotranspiration.

Water content gives the user an idea of how much water that soil will have for each day of the week, and how close it gets to the stress zone. Therefore, this graph uses the maximum absorbable water; the management allowed depletion and soil moisture. Moreover, as seen in Figure 27, the stress level is different for each day due to the growth of the plant.

![Figure 27. Zone predicted water content in the soil.](image)

Water balance takes into account the gains and losses of water at the end of each day. If the amount of water reaches the management allowed depletion, irrigation is then required. With this graph, the user can easily understand how much water the evapotranspiration is consuming from the soil and how much it is gaining from rain and irrigation, see Figure 28.

![Figure 28. Zone predicted water balance.](image)

**Devices**

A device is a piece of hardware that a user associates to an account which saves and executes the watering schedules as well as reading sensor data. This view aims to list the controllers associated with a user account.

Each controller has a unique serial number that is used to associate the device with a user account. A QR Code, a machine-readable optical label that a smart device camera can scan, can represent this serial number, for a more comfortable setup, see Figure 29.

When a device is added, either by entering a device’s serial number or scanning its QR Code, the device’s parts get available for the user to create new zones, i.e. master pump, valves and sensors.
Profile

The third navigation tab is Profile, a view designed to customise the user account properties and the application behaviour, as depicted in Figure 30.

A user can change the username, full name, e-mail and password, if those meet the required criteria, e.g. maximum length, e-mail not in use. Some changes such as password change force the session to restart. Moreover, the user can change the units that are displayed by the app, e.g. volume as $m^3$, area to $km^2$ and speed to $km/h$. 
6.2.4. Other features

Flexibility

There is a vast diversity of mobile devices, with many different screen sizes and resolutions. A significant challenge for any developer targeting those platforms is handling as many devices as possible with the lowest changes in code.

React-native has a layout algorithm, flexbox, which aims to provide a consistent layout on different screens. However, some parts of the design must be taken care of manually, e.g. SVG graphs, see Figure 31. For those, react-native provides an interface that gives the screen dimensions and allows the developer to add an event listener that informing the application in case of changes of such dimensions.

We developed this app with a combination of both previously mentioned techniques. Therefore it is ready for many smart devices with different screen sizes.

![Figure 31. Flexible layout is responding to the device orientation.](image)

Loading icon

Jakob Nielsen gives basic advice regarding response times, where 0.1 second is the limit for the user to feel that the system is reacting instantaneous, 1.0 second is the limit for the user’s flow of thought to stay uninterrupted, and 10 seconds is about the limit for keeping the user’s attention [33].

The mobile application creates multiple internet requests to the server during its normal usage, that can take from some milliseconds up to a few seconds, depending on the connection between the two and the load of the server.

During software development, the notion of response time can be quickly forgotten given that, most likely, it uses a local connection and an idle server. However, in a production environment latency cannot
be despised for various reasons and the user needs to be aware that the system is processing after a click or input.

A loading icon is displayed each time the application is expecting a delay, to keep the user's attention. For example, when navigating between views, where the data needs to be fetched from the server and be analysed and prepared to show the user.

**Breadcrumb**

Breadcrumb is a graphical element used to aid navigation in user interfaces, providing users with a way to keep track and awareness of their location in the dialogue structure. This element is present in many applications that allow exploring files in a computer, where a considerably sized hierarchy exists, and there is the need to aid the user during the navigation.

![Breadcrumb](image)

*Figure 32. Area creation view, from left to right: name, location and weather provider.*

This system uses this method during the creation of areas and zones, drawing a path at the bottom of the view containing the steps required to accomplish the task. Each step relates to a group type of information, and the user can easily navigate back by clicking on it.

**Location**

Irrigation planning using weather predictions require the knowledge of the farm’s location, as weather can and will vary accordingly.

As previously mentioned, each area has a list of zones that share the same location. A user defines this location during the creation of an area, where the user is given the option to set the location from one of the following: collect GPS location; search by street, zip code or city; or, mark the location on a map.

GPS location can be retrieved using the smart device GPS native API, which acquires location coordinates with a set precision. One should be aware that weather predictions most likely will not change with a small difference between spaces, and, therefore, the smart device will probably return
good results even with low precision. However, if the user is configuring at a location different from the one he wants to, this method cannot be used.

Figure 33. Location - text search.

Another option is to input a street, zip code or city and search using Google’s Geocoding API[38], as depicted in Figure 33. The app detects when the user stops typing and creates an HTTP request to Google’s geocoding services by sending the text input and receiving a list of possibilities that are presented to the user to select one.

Figure 34. Location - map marker.

A combination of the previous two approaches was used to create the map marker, see Figure 34. By the one hand, using the GPS location the map is positioned at the user's location; on the other hand, providing a text box for searching for a street, zip code or city, the user can quickly jump to a new location without scrolling too much the map. Afterwards, the user can mark a position on the map and confirm its selection.

---

[38] https://developers.google.com/maps/documentation/geocoding/start
7. Solution evaluation

Our solution aims to save water and energy and contribute to proper crop development. Such goals are difficult to test, due to the time required for a good sample, space and location for crop plantation and development, and other economic factors.

This project’s evaluation was split into two parts: data analysis, where irrigation algorithms were used to simulate a scenario that was compared to real-world data; and user testing, where a group of participants tested the user interface.

7.1. Data analysis

For this evaluation, we used data reports that were generated on a weekly basis for an olive crop, located at Quinta das Chantas, from the 24th of February to the 10th of April, 2018, containing information such as date, moisture sensor values, weather conditions and water applied to each crop zone, and then we compared it to the data obtained from the algorithms of this solution. This crop is in a developed state; its coefficient is one point one ($K_c = 1.1$), and its minimum water content before stress is fifty millimetres.

7.1.1. Zone soil moisture sensor

As previously stated, farms need to be split into different zones that share similar crop and soil properties and weather conditions, resulting in same watering needs. A different soil moisture sensor should be placed at each of those zones.

In Quinta das Chantas, there are four zones, each with a soil moisture sensor. However, during the report’s timeframe one sensor was malfunctioning and, therefore, was not considered for our analysis.

Figure 35 shows a graph with the observed water content in the considered timeframe. As one can see, values collected for each day vary for each different zone, demonstrating the importance of having sensors placed in relevant locations.
Figure 35. Comparison between observed water content collected from each zone.

A peak can be seen on the 9th of March; however, on the following day, on the 10th of March, the observed water content drops down to a value close to the one before the peak. This phenomenon happened due to the significant quantity of water and how fast it was provided (heavy rainfall), at a rate much faster than what the soil could absorb, with most water ending up in the drainage canal.

Weather-based irrigation planning usually considers the total amount of precipitation forecasted for a day and relies on an estimate of the amount partially absorbed by the soil. However, it is unknown how much infiltrates the soil, possibly leading to irrigation miscalculation, as we will see in the following.

7.1.2. Irrigation planning

To test our irrigation algorithm, we ran it using weather forecasts for the previously selected time frame, resulting in predictions of water loss by evapotranspiration and gains from precipitation. Table 13 presents the values of water content observed in the starting day and the ones predicted and observed on the last day.

The generated predictions have an average error of 47.51% that can be lowered by:

- Changing the crop coefficient, resulting in a different crop evapotranspiration;
- Choosing an alternative weather provider, for more representative weather forecasts;
- Adding soil moisture sensors, to observe how much water infiltrates the soil.
<table>
<thead>
<tr>
<th></th>
<th>Zone 1</th>
<th>Zone 2</th>
<th>Zone 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed water content on first day</td>
<td>72.35 mm</td>
<td>77.69 mm</td>
<td>80.73 mm</td>
<td>76.92 mm</td>
</tr>
<tr>
<td>Predicted overall precipitation</td>
<td></td>
<td></td>
<td></td>
<td>194.5 mm</td>
</tr>
<tr>
<td>Predicted overall irrigation</td>
<td></td>
<td></td>
<td></td>
<td>0 mm</td>
</tr>
<tr>
<td>Predicted overall crop evapotranspiration</td>
<td></td>
<td></td>
<td></td>
<td>103 mm</td>
</tr>
<tr>
<td>Predicted water content on the last day</td>
<td>163.85 mm</td>
<td>169.19 mm</td>
<td>172.23 mm</td>
<td>168.42 mm</td>
</tr>
<tr>
<td>Observed water content on the last day</td>
<td>105.46 mm</td>
<td>110.13 mm</td>
<td>128.99 mm</td>
<td>114.86 mm</td>
</tr>
<tr>
<td>Prediction error</td>
<td>55.37%</td>
<td>53.63%</td>
<td>33.52%</td>
<td>47.51%</td>
</tr>
</tbody>
</table>

Table 13. Observed water content versus predicted water content.

A daily comparison of the average zone water content with predictions can be seen in Figure 36. Furthermore, neither the predictions nor observed values ever reach management allowed depletion. Therefore irrigation was not provided or predicted.

![Predicted water content using weather forecasts (mm) - 2018](image)

Figure 36. Predicted water content using weather forecasts versus average observed zone water content.

In Figure 37 we can see how soil moisture sensors can help irrigation. Using the previous daily forecasts and the daily observed values of soil water content, we predicted the water content for each day and adjusted the predictions thus reducing the error margin.
7.1.3. Conclusions

The water levels throughout a terrain can be entirely different, even if only a single type of plant is grown there. The observed values depend on the exact location, so the more probes there are, the better the acquired values of the amount of water present in a soil. However, probes are expensive, so these must be purchased for strategic locations that best represent a particular area.

Although apparently all soil receives the same amount of water, we perceive that the observed values vary according to zone. This makes models based only on weather forecasting fail, especially for long periods of time, as it happened in the test described above.

We concluded that our weather-based algorithm combined with its subsequent confirmation and adjustment, by reading the soil moisture values, obtains the best results and that these are accurate.

7.2. User testing

User testing or usability testing aims to test whether the product developed is usable by the intended user population to achieve their tasks [27], “breaking down the wall between those who create the product and those who use it” [34].

Our Mobile Application was tested using this methodology. We created tasks with specific goals, solved by a group of participants, representing real users. During the execution of such tasks, we recorded efficiency, effectiveness and satisfaction. Finally, after analysing the recorded data, the problems found were diagnosed and fixed. This process was repeated till users could operate the product as intended, see Figure 38.
7.2.1. Protocol

The usability testing carried out by following a specific protocol to ensure that all participants had similar environmental conditions, used the same mobile device, and began at the same starting state.

We prepared a room with the lowest possible disturbance, to avoid interrupting the participant’s line of thinking, and provided an Android smartphone, capable of connecting to our servers, with our application installed and opened in the logged-out state (the starting state).

We gave the participant a document, which was available for consultation during the extent of the test (see Appendix G) that contained the following:

- A brief introduction to Raindrop and essential information required for the execution of the test. The user was also given five minutes to explore the application and sort any possible doubts;
- An initial questionnaire, to obtain demographic data and previous experience of the user;
- A list of tasks, representative of crucial functionalities of the application, where we recorded the time on task, the number of clicks and the errors at each task;
- A satisfaction questionnaire, using the System Usability Scale (SUS) methodology.

7.2.2. Results

Demographics and previous experience

With the results from the initial questionnaire, we can observe that the participants of this testing were mainly ranging from eighteen to fifty-four years old, and the male-female ratio was sixty to forty, as depicted in Figure 39.
The vast majority consider themselves in an advanced level of smartphone experience usage. The primary operating system used on a daily basis is Android, followed by iOS, see Figure 40.

Tasks

Each user was given eleven tasks to perform. During the tasks’ execution, we collected, for each task, the elapsed time, the number of clicks and the number of errors. Those results were then compared to a reference test – a test that was performed by ourselves aiming to figure the minimum number of clicks to solve a task and the amount of time that an experienced user would need without any mistakes.

Each collected time was considered as acceptable if its value was under three times the reference value. As for the number of clicks, the acceptable range was up to one point five times the reference value.

Figure 41 shows an overview of the time spent by each user at each task. Most of the results are within the defined acceptable range. However, there are some outliers that we show next.
Figure 41. The time required to complete each task.

**Task 1. Registration - create a new account.**

Each user performed this task within the acceptable time range and number of clicks.

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>30</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>55.866</td>
<td>8.3</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>14</td>
<td>0.699</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 14. Task 1 - Registration.

**Task 2. Login – authenticate using the previously created account.**

Most of our users had no issues logging in, doing it within the reference time and number of clicks. One user, however, wrote the username, during the registration process, with a capital letter (“John”) and was trying to log in, without success, with the same username but in lowercase ("john"). This proved to be a system error, as usernames should not be case-sensitive in authentication. This issue was marked as a bug, and was fixed later.

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>8</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>9.7</td>
<td>3</td>
<td>0.07</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>5.8</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>Number of issues</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 15. Task 2 - Login.

**Task 3. Add device - associate a device, either by using a serial number or a QR-Code.**

The participants were able to perform this task without any issues. We found out that QR-Code was extremely well accepted and used, proving its efficiency during the association process.
Task 3 - Add a device.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td>18.9</td>
<td>4</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>7.6</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 16. Task 3 - Add a device.

Task 4. Create an area, insert a new area, given the required data.

The time reference for the task was 38 seconds, which multiplied by 3 is 114 seconds – the maximum acceptable time. Four users had times well above this value. Moreover, the reference value for clicks was 9, which multiplied by 1.5 equals 13.5 – the maximum acceptable number of clicks. Two users exceeded this number.

From the previous data, we can see that this task raised some issues. In this task the user had to create an area, by providing a name, selecting a location and picking a weather provider. We observed that the users that took more time and made more clicks had issues in the selection of the location. To select a location, a user could either collect the GPS coordinates; search by street, zip code or city; or, mark the location on a map.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>38</td>
<td>9</td>
</tr>
<tr>
<td>Average</td>
<td>84.53</td>
<td>10.53</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>35.13</td>
<td>2.7</td>
</tr>
<tr>
<td>Number of issues</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 17. Task 4 - Create an area.

Most of the users preferred to search by street, zip code or city. However, the task provided the location “Instituto Superior Técnico, Taguspark”, which, when searched by the user, returned a street name (“Av. Prof. Dr. Cavaco Silva 13, 2740 Porto Salvo, Portugal”) that, even though it was indeed correct, the users could not identify as the correct one, see Figure 42.

![Figure 42. On the left, location search by street, zip code or city; on the right, mark the location.](image)

On the one hand, the task could have been simplified by providing the street address, thus avoiding confusion. On the other hand, we intended to observe if the users, when in doubt, would try to use other options, such as “mark the location”.
But even though task time and the number of clicks were higher than the reference test, all users were able to complete the task correctly.

Task 5. Update area - change a previously created area’s location and weather provider.

This task consisted of picking another weather provider and changing the location from “Instituto Superior Técnico, Taguspark” to “Instituto Superior Técnico, Alameda”, which, unlike the previous task, did not cause any issues. It was performed within the expected values except for one participant that could not figure how to change the weather provider.

The application view for picking another weather provider has a button labelled “Change”, which the user tried to click, expecting a new window to pop up with the available choices, see Figure 43.

![Figure 43. Change weather provider.](image)

We considered this issue as a bug since the button should have been greyed out, not to induce the user in error. The button looked like it was clickable, however, since the dropdown value did not change, the button did not act when pressed.

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>34</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>67.67</td>
<td>12.87</td>
<td>0.07</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>37.24</td>
<td>1.14</td>
<td>0.25</td>
</tr>
</tbody>
</table>

| Number of issues | 1 | 0 | 1 |

Table 18. Task 5 - Update area.

Task 6. Create zone – insert a new zone inside the previously created area.

In this task, the participant was requested to create a zone, with a name, a dimension, a crop type, management allowed depletion, a soil texture, a soil slope, a pumping station and a valve and their respective flows, and a moisture sensor. Those properties had to be inserted by the user in a form that used the breadcrumb method to aid navigation, by splitting the form into different steps that the user could go back to if required.

This task proved to be the most laborious task in the tests, and many users had trouble completing it. Common issues were the incorrect usage of units; forgetting to change crop type and soil texture from the drop-down list; and, pressing the back button, which restarted the test instead of going back to the previous step.
As listed in Table 19, six users exceeded the maximum acceptable time for the task; seven users made more clicks than acceptable; and, five users ended up with errors. Therefore, such results cannot be treated as outliers, and the zone creation process needs to be reviewed in future work.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>53</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>134.33</td>
<td>17.07</td>
<td>0.33</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>51.12</td>
<td>4.49</td>
<td>0.47</td>
</tr>
<tr>
<td>Number of issues</td>
<td>6</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 19. Task 6 - Create a zone.

Task 7. Rename zone – change the name of the previously created zone.

In this task, the user had to rename the previously created zone, by following steps similar to the ones required for the fifth task. The results were outstanding, leading us to believe that the user easily learned how to edit a zone based on previous experience.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>24</td>
<td>6.33</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>14.25</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 20. Task 7 - Rename zone.

Task 8. Check irrigation schedule - view the predicted watering schedules for the created zone.

Assuming that the user had just completed the previous task, a press of the back button was all that was required to view the predicted watering schedule. However, many users immediately tried to scroll down, after pressing the back button, searching for more information; others even navigated to other views, ending up taking much longer than expected, as listed in Table 21.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>23.1</td>
<td>2.1</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>10.15</td>
<td>1.86</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>13</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 21. Task 8 - Check irrigation schedule.

As in the sixth task, the values obtained cannot be considered outliers, and a future redesign should be considered to make the watering schedule resemble more of a “schedule”.

Task 9. Check weather forecast - view the weather forecast for the following three days.

Just like in the previous task, a back button was just enough to enter the Area view that contains weather forecasts. Afterwards, the user had to do three clicks to view the three following days’ predictions. However, in contrast to the previous task, the users were able to recognise much faster the intended information.
### Task 9 - Check the weather forecast.

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>13</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>24.5</td>
<td>4.9</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>11.4</td>
<td>1.24</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 22. Task 9 - Check the weather forecast.

### Task 10. Change units - update the application’s temperature unit to Fahrenheit.

To accomplish this task, the user had to change the application’s temperature units to Fahrenheit. In this task, there was a significant outlier, who performed the task in 132 seconds, almost 90 seconds more than average.

The user first pressed the temperature unit displayed in the weather predictions; afterwards the user entered the area settings and even the zone settings; ending up returning to the starting state and finding the option under the profile tab. Even though we did not consider it as an error, we firmly believe that the first instinct of trying to switch a unit while viewing the data would add significant value for a better experience and should be implemented in future work.

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>18</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>40.47</td>
<td>9.1</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>28.8</td>
<td>4.01</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 23. Task 10 - Change units.

### Task 11. Logout – exit the user account.

The final task consisted of exiting from the user account, which the users had no issues with, as seen in Table 24.

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Clicks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>4.3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>1.14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Number of issues</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 24. Task 11 - Log out.

### Satisfaction questionnaire

In the satisfaction questionnaire the method System Usability Scale (SUS) was used, a “quick and dirty usability scale” [35], where the participants were asked to score the following ten questions from one (strongly disagree) to five (strongly agree):

1) I think that I would like to use this system frequently.
2) I found the system unnecessarily complex.
3) I thought the system was easy to use.
4) I think that I would need the support of a technical person to be able to use this system.
5) I found the various functions in this system were well integrated.
6) I thought there was too much inconsistency in this system.
7) I would imagine that most people would learn to use this system very quickly.
8) I found the system very cumbersome to use.
9) I felt very confident using the system.
10) I needed to learn a lot of things before I could get going with this system.

The answers were transformed into a SUS score (see Table 25) by executing the following steps:

1. Swap the scale from one to five to zero to four, by:
   o subtracting one from the answer, if the question number was odd;
   o subtracting answer from five, if the question number was even;
2. Sum the scaled answers, resulting in a score from zero to forty;
3. Multiply the sum by two point five, resulting in a scale from zero to one hundred;
4. Average of all tests, resulting in a system usability scale score in a range from zero to one hundred.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 4</td>
<td>77.5</td>
</tr>
<tr>
<td>2</td>
<td>2 5</td>
<td>85</td>
</tr>
<tr>
<td>3</td>
<td>2 5</td>
<td>87.5</td>
</tr>
<tr>
<td>4</td>
<td>4 5</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>2 5</td>
<td>67.5</td>
</tr>
<tr>
<td>6</td>
<td>3 4</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>3 4</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>5 3</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>4 5</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>4 3</td>
<td>62.5</td>
</tr>
<tr>
<td>11</td>
<td>4 3</td>
<td>70</td>
</tr>
<tr>
<td>12</td>
<td>5 2</td>
<td>67.5</td>
</tr>
<tr>
<td>13</td>
<td>5 4</td>
<td>60</td>
</tr>
<tr>
<td>14</td>
<td>5 2</td>
<td>70</td>
</tr>
<tr>
<td>15</td>
<td>5 2</td>
<td>82.5</td>
</tr>
<tr>
<td>Average</td>
<td>4.53</td>
<td>72.33</td>
</tr>
</tbody>
</table>

Table 25. System Usability Scale spreadsheet.

The obtained system usability scale score was 72.33, a value above the average (68) and graded as ‘C+’ according to Jeff Sauro [36]. Therefore, even though the Raindrop system usability is ‘good’, there is still room for future improvement. Mainly, tasks six and eight, which scored lower than average should be addressed in future work.
8. Conclusions

8.1. Approach overview

Crop irrigation requirements not only are hard to calculate but also change in time and with plant growth stage. Different methods and products are available to help farmers track those needs.

Some products are weather based, using weather forecasts to predict the amount of water in the soil over a certain period, failing to know the actual amount of water that infiltrates the soil. Others are based on soil water content, with the aid of soil moisture sensors that help determine the amount of water in the soil and irrigation is applied when it falls below a certain limit, but fail when applying irrigation in additional quantities in rainy days due to the absence of rain forecast.

The most advanced products combine the previous methods, taking advantage of the best of the two worlds. However, none of the solutions has an ideal plan for when more than one irrigation controller is needed since there is the lack of interoperability when there is more than one controller for an area. Our solution was devised with the purpose of adding value to existing solutions, by introducing interoperability between controllers.

The solution we have created allows for the convenience of our users that operations can be controlled remotely through a smart device running Android or iOS. Once we have studied the best way to develop an application for a smart device, we concluded that a cross-platform approach would be the most appropriate. We tested a technology based on the hybrid methodology, Ionic, and another based on the interpreted method, React Native. From these, we choose React Native for being faster and providing a look and feel closer to native applications.

So that irrigation schedules are automatically provided and do not force the user to run our application on a constant basis, we created a server whose responsibility is to collect all the necessary data and generate the entire schedule and provide both the irrigation controllers and users’ devices with irrigation schedules. In order to achieve a solution with good results, the irrigation schedules generated by our system take into account weather forecasts and readings of soil moisture content.

Once we got a usable prototype, we ran two tests: one to test the irrigation algorithms, and the other to test the usability of the system with users.

From the tests developed for the irrigation algorithms, and only taking into account the forecasts of meteorology, we concluded that at the end of forty-five days the error between the predicted amount of water in the soil in comparison to what was observed was 47.51%. However, this error is practically null when the observed values of the humidity sensors are taken into account since forecasts were updated daily.

Testing with users proved to be very important as it allowed us to discover and fix various errors and features that were corrected. In the end, a satisfaction questionnaire was applied using the System
Usability Scale method, and our application achieved a score of 72.33, thus signalling that the application is "good" but can still be improved.

8.2. Limitations and weaknesses

Our system has some limitations and weaknesses. For example, the Internet connection is essential; without it, the central server cannot connect to the weather providers nor transmit the irrigation information to the users.

This project’s system requires minimal knowledge of agriculture since various data are needed for the creation of zones, such as the type of plant and its coefficient, the type of soil and so forth.

One of the problems of agriculture is floods, which not only make the terrain lose essential minerals but can also damage crops. Weather conditions are not controllable, and therefore our application cannot handle everything automatically, thus requiring human intervention.

A significant difficulty in developing applications for agriculture is to perform tests in a real environment with fields and plants. To do so requires several features and significant time and availability since such tests cannot be done overnight, given the natural temporal need for plant growth. Also, the tests cannot be replicated accurately, since weather conditions are ever changing.

8.3. Future work

This project initially contained extremely ambitious requirements, due to the high complexity inherent in the agriculture world. There are many things we would like to have done that stand as proposals for future work.

In this document it was proposed the realisation of an irrigation controller, that was wholly simulated. Its actual implementation would be a great asset in future work, since it allows for automation in irrigation, creating a bridge between software and hardware. The development of this component also allows tests to be carried out on the field.

There is an extensive list with the different types of plants that can hardly be maintained by the application development team. Also, the same type of plant may have different needs in different climates. For better convenience, users should be able to add their crops and evapotranspiration coefficients and be able to share them with other users.

Moisture sensors are subject to the most diverse problems that may jeopardise their viability. The system could collect data and decide if the acquired values are within the normal parameters and whether they are correct or not. In case of failure, affected users would be informed of the occurrence.

A good way for the system to interact with users could be to add notifications to the mobile application. These would inform the user of suggested changes to irrigation plans, system maintenance needs or even tips.
This project uses algorithms to decide the amount and times of irrigation. Since the flow needed to provide such quantities for periods may be insufficient, it may be necessary to define which areas require water more urgently, thus setting priorities.

In order to facilitate the recognition of zones in an area, we suggest the possibility of drawing them on a map. This map could then operate as a way of zone selection, or, even better, as a heat map, where different levels of water content would be visible, in order to realise which areas need watering.

The mobile application interface takes advantage of the wide use of the English language throughout the world, allowing us to broaden our user base. Nonetheless, additional languages should further increase this number. This setting should be customizable by users according to their preferences or regions.

Finally, it is important to mention the need for an administration panel. This panel would be the back-office of the application, where only users with a certain level or role would be able to access to view and modify users, their areas and zones. It would also give the possibility to add and modify types of plants and weather providers.
References


“Native, web or hybrid mobile-app development,” IBM, 2012.


# Appendix

## Appendix A

Single crop coefficients ($K_c$), adapted from [6, pp. 110-114].

<table>
<thead>
<tr>
<th>Crop</th>
<th>$K_{c\text{ini}}$</th>
<th>$K_{c\text{mid}}$</th>
<th>$K_{c\text{end}}$</th>
<th>Maximum crop height (m)</th>
<th>Maximum Root Depth (m)</th>
<th>Depletion Fraction ($p$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomato</td>
<td>0.6</td>
<td>1.15</td>
<td>0.70 - 0.90</td>
<td>0.6</td>
<td>0.7-1.5</td>
<td>0.40</td>
</tr>
<tr>
<td>Sunflower</td>
<td>0.35</td>
<td>1.0 - 1.15</td>
<td>0.35</td>
<td>2</td>
<td>0.8-1.5</td>
<td>0.45</td>
</tr>
<tr>
<td>Maize, Field (grain) (field corn)</td>
<td>0.3</td>
<td>1.20</td>
<td>0.35 - 0.60</td>
<td>2</td>
<td>1.0-1.7</td>
<td>0.55</td>
</tr>
<tr>
<td>Maize, Sweet (sweet corn)</td>
<td>0.3</td>
<td>1.15</td>
<td>1.05</td>
<td>1.5</td>
<td>0.8-1.2</td>
<td>0.50</td>
</tr>
<tr>
<td>Berries (bushes)</td>
<td>0.30</td>
<td>1.05</td>
<td>0.50</td>
<td>1.5</td>
<td>0.6-1.2</td>
<td>0.50</td>
</tr>
<tr>
<td>Grapes, Table or Raisin</td>
<td>0.30</td>
<td>0.85</td>
<td>0.45</td>
<td>2</td>
<td>1.0-2.0</td>
<td>0.35</td>
</tr>
<tr>
<td>Grapes, Wine</td>
<td>0.30</td>
<td>0.70</td>
<td>0.45</td>
<td>1.5-2</td>
<td>1.0-2.0</td>
<td>0.45</td>
</tr>
<tr>
<td>Walnut Orchard</td>
<td>0.50</td>
<td>1.10</td>
<td>0.65</td>
<td>4-5</td>
<td>1.7-2.4</td>
<td>0.50</td>
</tr>
<tr>
<td>Olives</td>
<td>0.65</td>
<td>0.70</td>
<td>0.70</td>
<td>3-5</td>
<td>1.2-1.7</td>
<td>0.65</td>
</tr>
</tbody>
</table>
Appendix B

Test application screenshots scaled according to the devices screen size (Android, LG Nexus 5x; iOS, iPhone 6s PLUS).

First view: “Hello World.”

Second view (A): request location permissions.
Second view (B): display current location (latitude and longitude).

Third view: create an HTTP request, containing the previously generated position, to a remote weather data provider; receive and parse the response JSON file; display weather.
Appendix C

Requirements

Functional requirements.

1. **Generate a watering schedule automatically.** The system must generate a watering schedule for each zone, with a minimum period of one week. This schedule must be based on forecasted weather, soil moisture values, and zone configurations.

2. **Restrict the watering schedule.** The system should allow the user to restrict the watering schedule to daily schedules (e.g., limiting watering schedule to Mondays from 22:00h-23:00h and Fridays from 02:00h-04:00h).

3. **Update watering schedules.** The system must update the watering schedule if a change of weather predictions, soil moisture values or zone configuration occurs.

4. **Multiple weather data providers.** The system should receive weather data from multiple sources, to avoid interruptions if a weather provider is unavailable, and cross-data to discard possible data errors.

5. **Support multiple controllers.** The system must support multiple irrigation controllers, each with a finite number of zones.

6. **Support multiple controllers per user.** The system must allow one user to have more than one irrigation controller. The system must provide an abstraction layer so that the user can combine multiple controllers as one (e.g., watering multiple zones at the same time can cause issues, as water pressure might not be enough; power consumption may be higher due to the need of a higher circuit breaker panel capacity; furthermore, some zones might require water first).

7. **Share sensors across controllers.** The system should be able to use sensor data from different controllers to generate and update watering schedules (e.g., rain sensor, a user with two controllers in the same area probably does not need two different rain sensors).

8. **Detect sensor failures.** The system should detect sensor failures, e.g., no data received or received incorrect data.

9. **Prevent sensor failures.** The system should prevent sensor failures by notifying the user to perform maintenance tasks, such as check the correct placement of moisture sensors.

10. **Manage actuators.** The system must control actuators, activating or deactivating them (e.g., activate the pumping station and a valve, resulting in watering a zone).

11. **Fertigation.** The system could be able to add different fertilisers to the water supply during irrigation cycles.

12. **Manual control.** The system should allow the user to water a zone manually at any given time.

13. **Notifications.** The system should send notifications to users, e.g., water cycle start/end information, maintenance warnings and schedule updates.

14. **Offline.** The system could save watering schedules and statistics in the user’s devices so that users can access those offline.
15. **Rainfall.** If rainfall is detected, the system *should* suspend any current irrigation cycles and recompute irrigation schedules.

**Data requirements.**

16. **Receive weather data.** The system *must* receive forecast from a remote weather data provider, given a region name or location coordinates, for a period of at least 24h. The information obtained *should* contain the following: elevation above sea level, latitude and height of measurement above the ground surface, maximum and minimum air temperature, average daily net radiation, average daily wind speed and maximum and minimum relative humidity.

17. **Read sensor data.** The system *should* read data from the following sensors: air temperature, precipitation, and soil moisture.

18. **Store zone information.** The system *must* store the following information for each zone: crop type, crop development state, soil type, soil slope, amount of daily solar exposure, nozzle type and required fertiliser type(s) and quantity.

19. **Save watering schedules.** The system *must* save the watering schedule in a non-volatile memory, continuing normal operations in case of network failures or electric power.

20. **Allow different units of measurement.** The system *should* work with different measurement unit systems.

21. **Record predictions, measurements and schedules.** The system *should* record the data received from weather servers, the data collected from sensors and the watering schedules (e.g., statistics and future tests and analysis).

**Environmental requirements.**

22. **Temperature.** If the temperature is less than or equal to zero, the system *should* suspend irrigation to prevent freezing.

23. **Internet connection.** The system *must* have internet access to communicate with users and weather data providers.

24. **Remote access.** The system *must* be remotely accessible and controllable for the users.

25. **Location.** The system *should* access user GPS positioning during the controller’s setup.

26. **Smart device.** The user *must* have access to a tablet or a smartphone running Android or iOS.

27. **Availability.** The system *must* be available for the operating systems Android and iOS.

**Usability goals.**

28. **Effectiveness.** Can the system help the user save water and energy?

29. **Utility.** Does the system provide all the required features for proper irrigation?

30. **Learnability.** How much time will take to the user to configure controller zones?
## Appendix D

Server application programming interface (API) paths

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Body</th>
<th>Login</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Users</strong></td>
<td><strong>GET</strong> /users</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>POST</strong> /users</td>
<td>Username, name, email, password</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td><strong>PUT</strong> /users</td>
<td>Username, name, email, password</td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>POST</strong> /users/login</td>
<td>Username, password</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td><strong>GET</strong> /users/logout</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td><strong>Areas</strong></td>
<td><strong>GET</strong> /areas</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>POST</strong> /areas</td>
<td>Name, latitude, longitude, weatherProvider</td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>DEL</strong> /areas/{Areaid}</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>GET</strong> /areas/{Areaid}</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>PUT</strong> /areas/{Areaid}</td>
<td>Name, latitude, longitude, weatherProvider</td>
<td>⬤</td>
</tr>
<tr>
<td><strong>Zones</strong></td>
<td><strong>POST</strong> /areas/{Areaid}/zones</td>
<td>Name, areaSize, crop, soilTexture, pumpingStation, valve, moistureSensor</td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>DEL</strong> /areas/{Areaid}/zones/{ZoneId}</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>GET</strong> /areas/{Areaid}/zones/{ZoneId}</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td><strong>PUT</strong> /areas/{Areaid}/zones/{ZoneId}</td>
<td>Name, areaSize, crop, soilTexture, pumpingStation, valve, moistureSensor</td>
<td>⬤</td>
</tr>
<tr>
<td><strong>Crops</strong></td>
<td><strong>GET</strong> /crops</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td><strong>Soil Textures</strong></td>
<td><strong>GET</strong> /soiltextures</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td><strong>Weather Providers</strong></td>
<td><strong>GET</strong> /weatherproviders</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td><strong>Controllers</strong></td>
<td><strong>GET</strong> /controllers</td>
<td></td>
<td>⬤</td>
</tr>
<tr>
<td>Method</td>
<td>Endpoint</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>POST</td>
<td>/controllers</td>
<td></td>
<td>serialNumber ●</td>
</tr>
<tr>
<td>GET</td>
<td>/controllers/{ControllerId}</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>GET</td>
<td>/controllerparts/</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>GET</td>
<td>/controllerparts/listPumpingStations</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>GET</td>
<td>/controllerparts/listValves</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>GET</td>
<td>/controllerparts/listMoistureSensors</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>GET</td>
<td>/controllerparts/listAvailablePumps</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>PUT</td>
<td>/controllerparts/{Partid}</td>
<td>flow</td>
<td>●</td>
</tr>
</tbody>
</table>
## Appendix F

### Weather provider request and response

#### i. Meteo|Tecnico

<table>
<thead>
<tr>
<th>Request</th>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Host</td>
<td><a href="http://meteo.tecnico.ulisboa.pt">http://meteo.tecnico.ulisboa.pt</a></td>
</tr>
<tr>
<td></td>
<td>Port</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Path</td>
<td>/service/raindrop?lat=38.736946&amp;lon=-9.142685&amp;date=20180811&amp;days=7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{ &quot;response&quot;: [ { &quot;temp&quot;: { &quot;max&quot;: 28.31, &quot;min&quot;: 18.37, &quot;average&quot;: 23.34 }, &quot;hum&quot;: { &quot;average&quot;: 65.75 }, &quot;rain&quot;: { &quot;daily&quot;: 0 }, &quot;cloud&quot;: { &quot;average&quot;: 0 }, &quot;wind_vel&quot;: { &quot;average&quot;: 5.1 }, &quot;wind_dir&quot;: { &quot;average&quot;: 354.55 }, &quot;atm_press&quot;: { &quot;average&quot;: 1019.6 }, &quot;radiation&quot;: { &quot;daily&quot;: 28.09 } }, { ... } ]</td>
</tr>
</tbody>
</table>

#### Underground

<table>
<thead>
<tr>
<th>Request</th>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Host</td>
<td><a href="http://api.wunderground.com">http://api.wunderground.com</a></td>
</tr>
<tr>
<td></td>
<td>Port</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Path</td>
<td>/api/{key}/forecast10day/conditions/q/38.736946,-9.142685.json</td>
</tr>
</tbody>
</table>
{ "forecast": { "simpleforecast": { "forecastday": [ { "date": { "epoch": "1534010400", "pretty": "7:00 PM WEST on August 11, 2018", "day": 11, "month": 8, "year": 2018, "yday": 222, "hour": 19, "min": "00", "sec": 0, "isdst": "1", "monthname": "August", "monthname_short": "Aug", "weekday_short": "Sat", "weekday": "Saturday", "ampm": "PM", "tz_short": "WEST", "tz_long": "Europe/Lisbon" }, "period": 1, "high": { "fahrenheit": "89", "celsius": "32" }, "low": { "fahrenheit": "64", "celsius": "18" }, "conditions": "Clear", "icon": "clear", "icon_url": "http://icons.wxug.com/i/c/k/clear.gif", "skyicon": "", "pop": 10, "qpf_allday": { "in": 0, "mm": 0 }, "qpf_day": { "in": null, "mm": null }, "qpf_night": { "in": 0, "mm": 0 }, "snow_allday": { "in": 0, "cm": 0 }, "snow_day": { "in": null, "cm": null }, "snow_night": { "in": 0, "cm": 0 }, "maxwind": { "mph": 15, "kph": 24, "dir": "", "degrees": 0, }, "avewind": { "mph": 4, "kph": 6, "dir": "NW", "degrees": 322, }, "avehumidity": 62, "maxhumidity": 0, "minhumidity": 0 }, ... }, ... }, ... }
Appendix G

User testing – mobile application

i. Environment and equipment preparation

We have prepared a room with close to no distractions to avoid interfering with the user’s line of thinking, granting the same conditions for all users, and facilitated an Android smartphone with the Raindrop mobile application installed and opened, in the *logged-out state*, to begin the testing. This device is connected to the internet and can connect to Raindrop servers. Furthermore, a paper with a QR-Code and a Serial Number is provided to simulate a controller.

Moreover, a stopwatch will be used to measure the time required to accomplish each task.

ii. Introduction

First and foremost, we would like to thank you for your time. Your help is highly appreciated and is a fundamental step to the success of this project.

In the following, we will provide tasks that are meant to evaluate if the developed application can achieve the intended results. Any questions should be made before commencing the tasks or afterwards, as each task must be completed without any help. You can take as much time as required and if you are unable to achieve the requested task you should then ask to interrupt the task, and that test will be discarded. Our intent is not to test you, but the system.

In the following five minutes you will be able to test the application and ask any questions.

All information will be gathered anonymously and will not identify the participant. Furthermore, the collected data will only be used in the context of this project.

iii. Initial questionnaire

1) What is your age?

- [ ] ≤17
- [ ] 18-34
- [ ] 35-54
- [ ] 55-74
- [ ] ≥75
2) **What is your gender?**

- [ ] Female
- [ ] Male

3) **Do you have or take care of any plants?**

- [ ] Yes
- [ ] No

4) **If your previous answer was “no”, please skip the following questions.**

   a) **How often do you water them?**

      - [ ] Every day
      - [ ] Once a week
      - [ ] Once a month
      - [ ] Whenever I believe it is needed

   b) **How do you choose the amount of watering?**

      - [ ] Always the same
      - [ ] Randomly
      - [ ] Guessing the requirements
      - [ ] What a book or article recommended

   c) **Do you check weather forecasts to figure if watering is necessary?**

      - [ ] Yes
      - [ ] No

   d) **Would you like to use an application that calculates the watering needs of a plant for you? (When and how much should you water)**

      - [ ] Yes
      - [ ] No
5) How often do you use devices with a touchscreen? Please choose what best matches your usage.

☐ Every day
☐ Two or three days a week
☐ At least a few days a month
☐ I do not use

6) Which operating system do you usually use?

☐ iOS
☐ Android
☐ Other

iv. Tasks

1. Registration

Register a new account with the following:

Username: john
Name: John Doe
E-mail: jdoe@mail.com
Password: doe

2. Login

Login with the previously created account.

3. Add device

Associate a device to the account using the following:

SN: ABC-123
4. Create area

Create a new area with the following:

Name: Técnicas Alameda
Location: Instituto Superior Técnico, Taguspark
Weather provider: meteo|Técnico

5. Update area

Change the previously created area’s location and weather provider to “Instituto Superior Técnico, Alameda” and “Test”, respectively.

6. Create zone

Create a new zone, within the previously created area, with the following:

Name: Test
Dimensions: 1000 m²
Crop type: Olives
Management allowed depletion 0.8
Soil texture: Silty clay
Soil slope: 0°
Pumping Station: Cntr, pumpingStation1, flow 10 m³
Valve: Cntr, valve2, flow 1 m³
Moisture Sensor: Cntr, moistureSensor1

7. Rename zone

Rename the previously created zone to “Olival”.

8. Check zone irrigation schedule

View the days that will require irrigation for the previously created zone.

9. Check weather forecast

View the weather forecast for the following three days.

10. Change units

Change the application’s temperature unit to Fahrenheit [F].

11. Logout

Log out from the application.
v. **Satisfaction questionnaire**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly agree</th>
<th>Neither agree nor disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I think that I would like to use this system frequently.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the system unnecessarily complex.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I thought the system was easy to use.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I think that I would need the support of a technical person to be able to use this system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the various functions in this system were well integrated.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I thought there was too much inconsistency in this system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I would imagine that most people would learn to use this system very quickly.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the system very cumbersome to use.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt very confident using the system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I needed to learn a lot of things before I could get going with this system.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>