VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

CAPSTONE PROJECT

STUDYING AND DEVELOPING A SOLUTION TO MANAGE AND UPDATE IRRIGATION SCENARIOS TO MANY END-DEVICES

Major: Computer Science

THESIS COMMITTEE: CLC Khoa học Máy tính 6
SUPERVISOR(s): Assoc. Prof. Thoại Nam
Dr. Nguyễn Quang Hùng
REVIEWER: Assoc. Prof. Lê Trọng Nhân
—o0o—

STUDENT 1: Tiêu Viết Trọng Nghĩa (1852611)
STUDENT 2: Phạm Văn Minh Toàn (1953028)
STUDENT 3: Nguyễn Hoàng (1952255)

HO CHI MINH CITY, MAY 2023

Declaration

We guarantee that this research is our own, conducted under the supervision and guidance of Assoc. Prof. Thoại Nam And Dr. Nguyễn Quang Hùng. The result of our research is legitimate and has not been published in any forms prior to this. All materials used within this researched are collected by ourselves, by various sources and are appropriately listed in the references section. In addition, within this research, we also used the results of several other authors and organizations. They have all been aptly referenced. In any case of plagiarism, we stand by our actions and are to be responsible for it. Ho Chi Minh City University of Technology therefore is not responsible for any copyright infringements conducted within our research.

Acknowledgements

First and foremost, we would like to express our heartfelt appreciation to our beloved research supervisors, Assoc. Prof. Thoại Nam and Dr. Nguyễn Quang Hùng, for providing us the opportunity to conduct research throughout the semester and make contribution to the Smart Village System. During the course of this semester, all of their valuable orientations, evaluations, and instructions help us to gradually improve our work. We would like to show our gratitude to the previous researchers in charge of this study last year — Mr. La Hoàng Lộc and Mr. Đình Phúc Hưng — for their active support and suggestions. Their knowledge on the research topic, as well as their motivations and visions, have greatly influenced us. Last but not least, we would like to express our gratefulness to our beloved university — Ho Chi Minh University of Technology — for allowing us to conduct this research and for their continuous encouragement throughout the semester. It is inevitable that there are some mistakes in our presentations and demonstrations. We are all ears to any suggestions, and improvements are welcome and encouraged.

Abstract

In the recent years, the Vietnamese agriculture sector has made great progress in terms of securing its food and nutrition, solving the occupation problem in rural regions, contributing to GDP, etc., but it has also faced many challenges, most of which due to climate change and natural disasters. A proposed solution is smart agriculture, and it has been observed to be heading in the right direction. Small farmers in Vietnam has been utilizing smart agriculture to solve the aforementioned challenges, which gives farmers access to complete and timely information, skills, and strategies that will allow them to make better decisions regarding their production and trading. As a result, the farmers' productivity, outputs, income, and profits are expected to rise. However, the usage of smart agriculture in Vietnam is still constrained by some fundamental issues such as high technology investment costs, inefficiency at small scale, limited access to capital, land, suitable technologies, etc. This project aims to create a solution to lessen the burden put on a farmer when migrating to the smart agriculture model. With a view to creating a complete solution to the smart agriculture demand, we will discuss and extend a related project, how a previous project and ours can work side-by-side, and how they will be united into one package.

Contents

List of Figures

List of Figures

List of Tables

List of Tables

Chapter 1

Introduction

1.1 Overview

Smart agriculture describes the practice of using technologies such as Internet of Things, Artificial Intelligence, sensors, robots, etc. in agricultural activities. The high-tech machineries and equipment utilized can provide an extensive range of abilities such as climate monitoring, irrigation management, automation and more, helping to make better decisions and improving the efficiency of the farming process. In the Asia-Pacific region alone, smart agriculture market share is expected to grow with Compound Annual Growth Rate (CAGR) of 10.10% and generate $14396.90 in revenue by 2028, mainly due to increasing awareness of consumers in the benefits of smart agriculture[1].

Accounting for nearly 40% of total employment, the agriculture sector of Vietnam currently creates about 20 million jobs[6]. Although the figures show a declining trends in the recent years, Vietnam being an agriculture-centered country bears a lot of potential to lift off in this sector, thus has been widely supported and invested by domestic as well as foreign entities.

The demand for a smart farming solution reveals a lot of promises along with various challenges. The agriculture process concerns multiple factors like air temperature, soil properties, irrigation, etc., and with the support of IoT devices, most of these properties can be measured, transferred and saved at a date warehouse. With the data readily available, study and experience can be applied to create a decision model such that a computer can use this model to create the most optimal environment for a crop, improving the efficiency of the farming process, eventually helping the farmers cut down on costs and create more competitive selling price.

The Southern regions of Vietnam, specifically at the Mekong Delta, sees a lot of support not only from the businesses but also from the local government when it comes to smart agriculture. At Ho Chi Minh University of Technology (HCMUT), the High Performance Computing (HPC) Lab has created and been maintaining a smart agriculture solution. They have done a great job at

taking the climate data, processing these data to generate prediction models for some dedicated users. However, there are still unsolved problems like distributing the models, versioning these models and more, and our project aims to address as many problems as possible. We hope that our work will provide a better view on what we are trying to achieve so that later work can further improve what we have done.

1.2 Scope

With permissions from Assoc. Prof. Thoại Nam, our project plans to implement a system supporting an irrigation ecosystem. For this project, we define an irrigation scenario, which is a plan or instruction to irrigate a specie of plant at a designated location. Our system will allow the uploading of irrigation scenarios at the central server, then propagate them to the mobile application of the users, then they can configure the scenario used at the gateway devices at their designated location. We will also discuss the use of fog servers and how we can guarantee transparency of trust when the users want to control a gateway device.

The irrigation scenarios are created from the gathered data about a specific specie of plant at a specific location, also called a sample. The management and deployment of sensors, engineering of the data gathered, and generating scenarios are the works of another project, and we have been given permissions from Assoc. Prof. Thoại Nam to use the project and cooperate with the related group, hence the creation or generation of the irrigation scenarios is not within the scope of our project.

Moreover, we have been given access to the existing irrigation device solutions. They consist of irrigation pumps of various output, various irrigation nozzles, and a gateway device to receive irrigation scenarios or commands and control previously mentioned devices. These devices are also the work of another project, and we have been given permissions to work with this project and provide suggestions if necessary.

Our stakeholders are the end users at the mobile applications and the administrators at the servers.

1.3 Outcome

We define an irrigation scenario as a plan or guide on how to irrigate a type of plant at a specific location at a future time, and our project aims to implement a system to deliver these irrigation scenarios from a central server to the end users. This requires that we get to know how our stakeholders expect and use our system, specifically the end users on the mobile applications and at the gateway devices. One of our challenges is that there are multiple possibilities for the

mobile application to communicate with the gateway devices, and we will need to cover as many cases as possible while discussing how we can reduce the cost to maintain such system.

As such, our project aims to serve a web interface for each of the server application, and a mobile application for the end user. The project aims to cover as much technical aspects as possible, thus requiring less technical knowledge from the end users, allowing for easier transition to our system. Moreover, we will provide statistics dashboard for the administrators at the server so they can manage their system and make better predictions and choices.

This project has been a reflection of what our group has learnt during the time at the university: algorithms, software development process, database designs, etc. and has allowed us to practice on industry-standard tools and technologies: source code management, web frameworks, packaging, etc.

1.4 Structure

This project report is divided into multiple chapters. Each chapter will focus on an idea as follows:

Chapter 2: Study on factors and challenges that the agriculture sector concerns and the procedures and techniques applied in their current solutions.

Chapter 3: Deep dive into discovering what problems needed to solve and encountered problems during the implementation.

Chapter 4: Determining stakeholders and functionalities of the system.

Chapter 5: Brief summary about the main components of the system.

Chapter 6: Implementation of our system.

Chapter 7: Evaluation of the results.

Chapter 8: Retrospective of what have be done and can be done for the project.

Chapter 2

Survey on The Application of Internet Of Things in Agriculture

2.1 Automation in Irrigation

2.1.1 Overview of Vietnamese Agriculture

Vietnam is a tropical country in South East Asia where agriculture is very important. More than half of the people work in agriculture and grow crops in the lowlands and some parts of the highlands. The most important areas for farming are the Red River delta, the Mekong River delta, and the southern terrace.

Recently, Vietnam's agriculture has faced challenges because of the changing climate. Climate change has hurt rice production in the Mekong Delta, which is the most important place for growing rice in Vietnam. Drought and saltwater have made it hard for farmers to grow crops and caused a lot of damage. In 2020, the situation is worse than it was in 2016. Many people don't have enough water for daily use. Five provinces in the Mekong Delta have declared an emergency.[15].

The government is trying to use technology to help farmers and combat climate change. They want to use new technology to help farmers use less water and grow crops more efficiently. They are also working with other countries like Japan, Ireland, the Netherlands, and Australia to learn from their experience.[22].

Foreign and domestic companies are interested in investing in technology like the Internet of Things (IoT) and blockchain. This can help farmers improve their production without spending a lot of money. Most Vietnamese farmers are small and need affordable ways to improve their crops.

2.1.2 Current Solutions for Irrigation

Smart irrigation is a system built from multiple electronic devices including pumps, vents, solenoid valves, etc. integrated with a control center which helps to operate everything smoothly and consistently. Such systems are remotely controlled through smart devices, saving a considerable amount of time and labor for the users.

Nowadays, smart irrigation has been a common concept around agriculture areas in Vietnam. Most of the modern farmers are utilizing it in various ways, the most basic of which is scheduled irrigation.

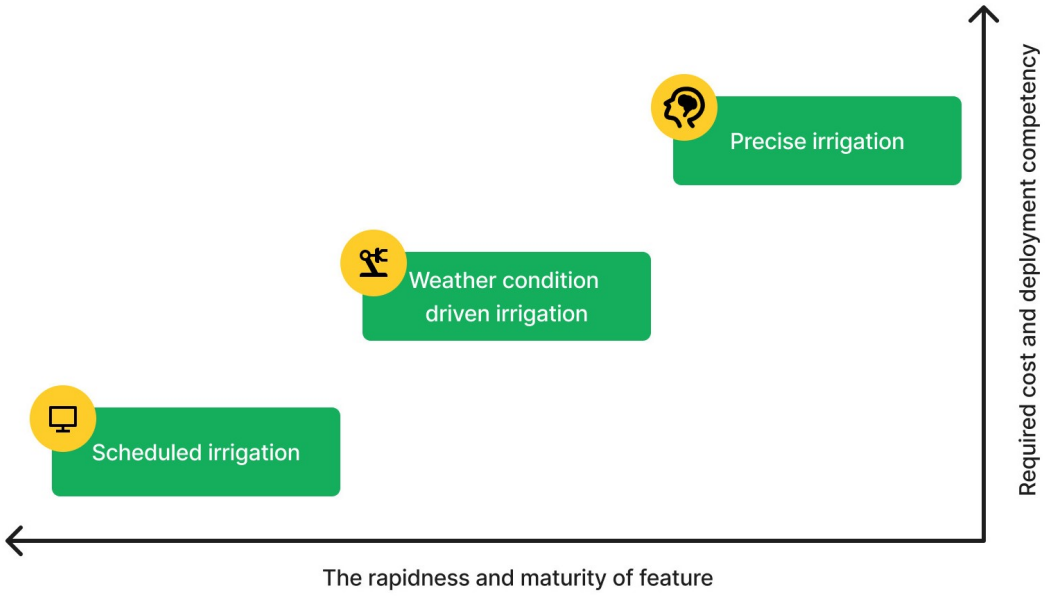


Figure 2.1.1: Main features of automated irrigation system

To give a brief overview of irrigation system, these are features of automatic irrigation system ordered by their competency:

1. Scheduled irrigation: Scheduled irrigation systems will automatically irrigate following a predefined schedule. In order to make this happen, an irrigation system needs an automatic timer or an automatic controller integrated with the irrigating devices. The controller will take charge of operating the system.

2. Weather condition driven irrigation: This system is a higher-level irrigation system. It uses sensors to collect data about weather conditions. This system is more precise than scheduled irrigation. It operates on different weather conditions such as rain, sun, etc. The

data collected by the sensors is transmitted in real-time. This system helps farmers to save water and grow better crops in a more efficient manner.

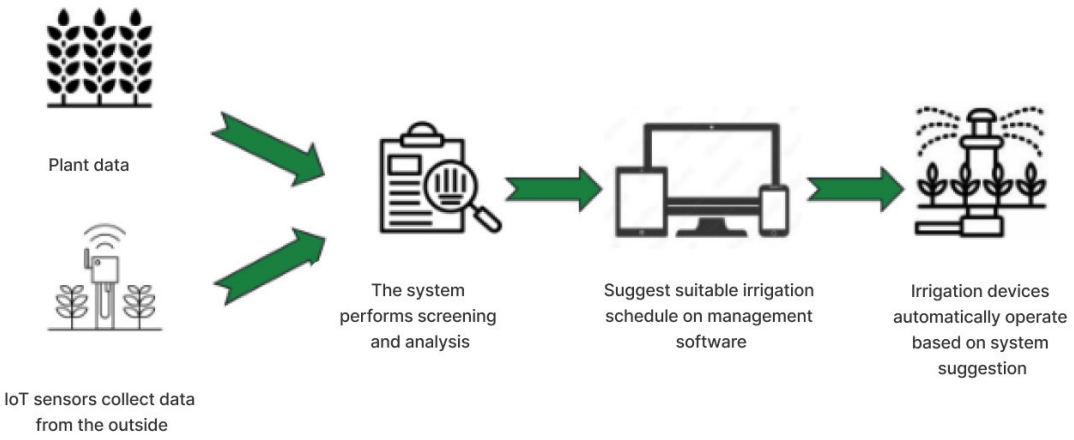


Figure 2.1.2: Irrigation control flow in general

3. Precision water control: This is the most advanced irrigation system. It waters crops at the right time, place, and amount. It uses irrigation sensors to monitor the growth and development of crops. The system collects data from environmental sensors, including water, soil, and air conditions. It also takes into account the average growth history of each plant type. This system is very efficient and economical. It helps farmers to increase yields, increase fertility, protect the soil, and preserve precious water resources.

   In the context of crops, the most significant benefit of irrigation systems is ensuring high yields and product quality, regardless of the natural climate conditions in a year. Not all farmers have the luxury to be located in humid climates where rains are frequent and the land is moist enough for the crops to thrive. With irrigation systems, farmers can grow crops in different weather conditions, including dry ones.

   In terms of usability, engineers and farmers who are in charge of the irrigation process can monitor and control the system remotely through applications on both web and mobile platforms. This feature saves time and labor for farmers.

2.1.3 Advantages of Smart Irrigation

Currently, most traditional irrigation systems are manually configured through a timer to turn on or off. The disadvantage of this approach is that watering will occur unnecessarily

throughout the day and night or in sudden changes in the weather like showers. This is inefficient in saving irrigation costs when the weather constantly changes. Studies have shown that worldwide, 70% of water usage is allocated to irrigation, and half of that is wasted due to inefficient traditional approaches[25]. Therefore, a new approach to managing irrigation and landscaping more efficiently is to know the amount of water needed for crops on any given day or to determine if crops are receiving too much water. Smart irrigation systems are becoming increasingly popular in countries around the world, even in the most remote areas of the planet. Smart irrigation systems can control plant watering and monitor daily activities of the garden by continuously collecting and inputting real-time weather data at a specific location.

2.2 Smart Village System

2.2.1 The Emergence of Smart Village Around the World

Smart Villages in EU Rural Areas

Rural regions in the European Union (EU) are diverse in terms of their nature, geographical patterns, development levels and socio-economic and demographic trends. Covering 44.6% of the EU and home to 93.1 million people (20.8% of the total EU population), the EU's rural regions are multifunctional spaces facing a range of challenges[9]. These include but not limited to: demographic ageing due to rising life expectancy and lower fertility rates, lack of infrastructure and service provision, digital gap and divide (i.e. lack of reliable internet connections limiting both individuals and businesses), and people moving to urban areas in search of better job prospects and provision of public services.

2019 data (GDP: 2016 data)	Territory (Km2)	%	Population (1 000 h.)	%	Density (inhabitant/Km2)	GDP (million EUR)	%	EUR/ inhabitant	Index
Rural regions	1 882 884	44.6%	93 160	20.8%	50	1 812 156	14.6%	19 302	68
Intermediate regions	1 930 008	45.7%	173 999	38.9%	90	4 269 836	34.3%	24 551	87
Urban regions	412 235	9.8%	179 662	40.2%	436	6 366 275	51.1%	35 786	127
Total EU - 27	4 225 127	100%	445 168	100%	105	12 448 267	100%	28 200	100

Figure 2.2.1: Rural regions vs intermediate and urban region: EU-27 structural data

Despite these challenges, rural areas offer many opportunities. Their diversity is one of the EU's richest resources. They provide food and environmental resources and can contribute to the fight against climate change, providing alternatives to fossil fuels and developing circular economy.

In 2020, a survey[8] carried out by the European Leader Association for Rural Development (ELARD) showed that rural citizens feel that the vital importance of rural territories for society should be recognized and public investment in the development of rural areas should be equal to that devoted to urban areas.

Moreover, the Commission communication *The future of rural society* acknowledged the specific problems of rural society and underlined the need to try out new development approaches, involving rural communities in seeking appropriate solutions. The communication concluded that the management approach to policies relevant for the development of rural areas needed to be coordinated, integrated and multi-sectoral in its implementation.

2.2. Smart Village System

While a range of responses exist to address these challenges, the smart village concept is seen as a way to future-proof rural communities and ensure their survival. The 2016 *Cork 2.0 Declaration for a Better Life in Rural Areas* set out a 10-point manifesto to improve quality of life in EU rural areas. It highlighted the need to overcome the digital divide between rural and urban areas and to develop the potential offered by connectivity and digitalization in rural areas.

Figure 2.2.2: A school, a kindergarten, and a medical center in Aghali smart village

In 2019, following the second *Consultation on the working definition of "Smart Villages"* and extended discussions in the *smart eco-social villages* pilot project steering group, a working definition was established. Involvement of the local community and the use of digital tools were seen as core elements, focusing on a number of key features:

- A smart village strategy identifies challenges, needs, assets and opportunities.
- Cooperation involving partnerships and support of the local authorities is key. This includes cooperation and partnership between villages, and between villages and nearby urban areas.
- Smart villages seek solutions rooted in the local territory that can generate value and benefits for the community
- Social and digital innovation are characteristic of smart villages

The EU Parliament allocated €3.3 million to support the development of 10 smart villages. The concept was further reinforced by the work of the European Network for Rural Development (ENRD). The ENRD website's smart villages portal provides access to case studies and publications on issues such as: digital and social innovation in rural services, the European Green

Deal, and rural digitalization transformation. The ability of the smart village concept to infuse many aspects of living and working in the EU's remote and sparsely populated rural areas and be illustrated by the following themes:

- Rural digitalization: In Lormes, Burgundy, France, the 'Digital Mission' association was established in 203 to offer digital inclusion and education support to the community. A local digital hub opened in 2008 offering training and educational facilities.
- Renewable energy: In Oberrosphe, Germany, the local community invested €700,000 in a project to connect 120 houses to a wood chip-fired heating plant managed collectively by a cooperative, resulting in CO_2 reductions.
- Mobility: In Yllas, Finland, a cooperation project formed by the municipality of Kolari of Yllas Travel Association launched a mobility project to enable tourists to buy tickets from a range of public and priate transport services
- Social innovation: The Iberian Ecovillage Network involves 13 eco-villages in Spain and Portugal.
- Health and social care: In Castellon Province, Valencia, Spain, in response to rural de-population, the provincial government launched a rural taxi service for medical purposes initiative.
- Culture: In Piscu Village, Ilfov County, Romania, a project is raising awareness of cultural and local heritage among younger generations, operating from a pottery centre.

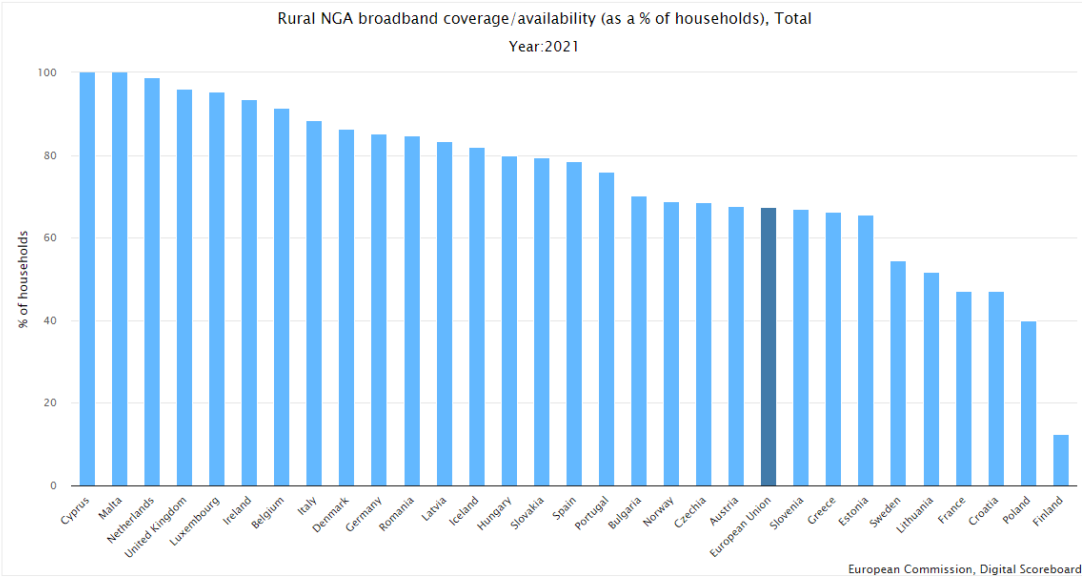


Figure 2.2.3: Rural next generation access (NGA) broadband coverage (2021)

In conclusion, the smart village concept is gaining traction on the rural development agenda, coinciding with the ongoing reform of the common agricultural policy (CAP). The European Parliament has made a significant contribution to the smart village concept, taking part in a pilot project on smart eco-villages and supporting the European Commission's 2017 action plan for smarter villages.

How Smart Villages Are Changing Lives in Niger

Niger is a landlocked country in West Africa that is named after the Niger River. With two-thirds of the country lying within the Sahara Desert, it is one of the hottest countries in the world. Niger has a developing economy based largely on agriculture and mining. The country has a poorly diversified economy with agriculture accounting for 40% of its GDP, with more than 10 million people (41.8% of the population) living in extreme poverty in 2021.

To better serve rural populations, the government of Niger together with various international organizations are collaborating on an initiative to bring smart technologies to underserved village communities.

Launched in 2019 to improve rural livelihoods through technology, the Smart Villages initiative uses a cross-sectoral, whole-of-government approach to design, deploy and scale digital solutions that contribute to the United Nations Sustainable Development Goals[20]. The Smart Villages approach now serves as a blueprint for the inclusive digital transformation of rural villages across the world.

Since 2019, two new digital healthcare services were launched in Sadore and Borgo Darey, focuses on supporting healthcare workers with dermatology and early childhood illness diagnostics. In one case, a young patient arrived at the Borgo Darey clinic with a suspected case of leprosy. His parents, seeing his rash, thought amputating his hands would save his life. Fortunately, the village nurse, unconvinced, snapped a photo using her digital tablet and sent it to a dermatologist based 3 hours away in Niamey for verification. After two weeks of taking locally available prescription medication, the patient was cured.

Malaysia's Smart Village Project

Since 2019, Malaysia's Ministry of Rural Development has been working with partners to provide rural areas with smart solutions, close the digital divide between urban and rural areas and build digital skills among village communities.

The pilot smart village programme — which will initially start in 191 villages but is intended to expand to 15,000 villages across Malaysia — plans to roll-out high-speed internet that will power smart classrooms and digital libraries, allow villagers to sell their wares via dig-

ital marketplaces and farmers to use drones, and enable hospitals to monitor the health of their patients from city areas and deliver care via telemedicine[17]. A Smart Map is also being built to share data in real-time and increase flooding awareness.

Figure 2.2.4: Rimbunan Kaseh, a model community to the north-east of Kuala Lumpur

On a broader scheme, the Smart Village initiative forms part of KPLB's Rural Development Policy 2030. This is a plan to create prosperous rural community with a focus on 10 areas spanning social, economic and environmental progress. These include the areas of entrepreneurship, human capital, sustainable biodiversity, housing and regional development.

2.2.2 Smart Village in Vietnam

Director, Chief of Office of Central New-style Rural Coordination Office Nguyễn Minh Tiến emphasized:

> Bringing digital transformation into agriculture in general and into new-style rural construction in particular is the main policy of the Government and the Ministry of Agriculture.[16]

Acknowledging this issue, the Central New-style Rural Coordination Office has promoted the implementation of the *Digital transformation in building new-style countryside towards a smart new-style countryside, period 2021 – 2025* project in order to improve the efficiency in the administration and management of the National Target Program on New-style Rural areas, improving the methods, efficiency in production and business of farmers, cooperatives and

enterprises. The scheme sketches 4 specific goals by 2025, including: Building a digital government, Digital Economy, Digital Society, and a set of National Criteria for smart new-style rural areas period 2026 – 2030.

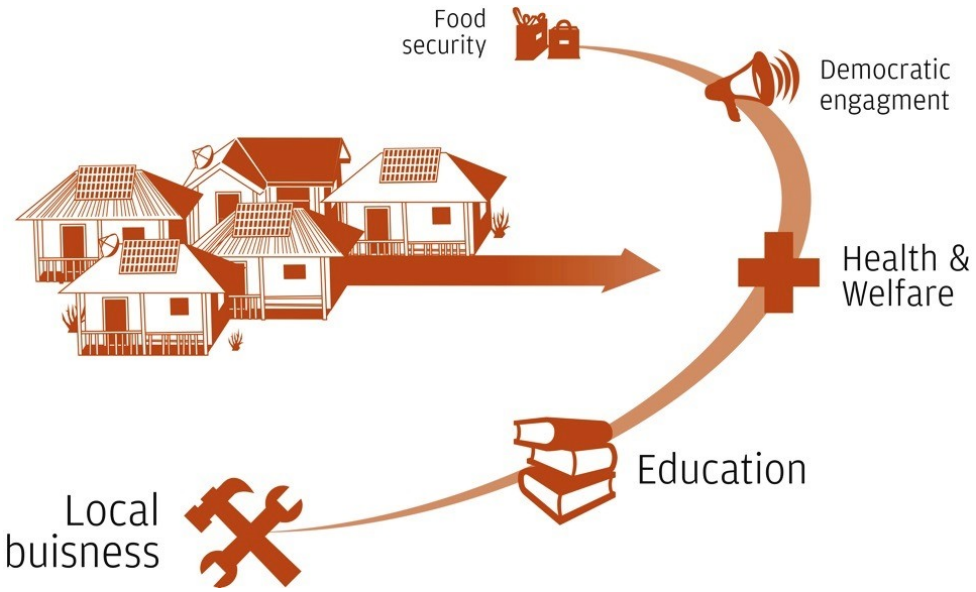


Figure 2.2.5: Smart village model in Vietnam

According to the Central New-style Rural Coordination Office, the *smart village/commune* is the optimal development model and target of the rural areas in the future. However, to develop such comprehensive model requires human, financial and time resources to convert, apply and adapt appropriately. In the short term, it is very challenging to develop an overall smart village model that can pilot and deploy smart villages in each specific field. Localities, depending on local conditions, resources and potential, can choose to deploy one or more components.

According to Mr. Nguyễn Minh Tiến, a smart village is a community of hamlets, villages and communes in rural areas that uses digital technology-based solutions to take advantage of local strengths and opportunities for sustainable development. Thereby, gradually narrow the rural – urban gap, improving the economic, social and environmental conditions of rural areas.

Thừa Thiên Huế is the leading locality in piloting the construction of the "smart commune" model in two communes: Vĩnh Hưng (Phú Lộc) and Quảng Thọ (Quảng Điền) in the 2016 – 2020 period. Director of the Department of Agriculture and Rural Development of Thừa Thiên Huế Province Nguyễn Minh Đức expressed that the pilot model is in its first steps but has shown multiple benefits: commune-level governing, smart commune management, applying science and technology to agricultural production, weather forecasting, sharing and monitoring data for local farming and production, etc.

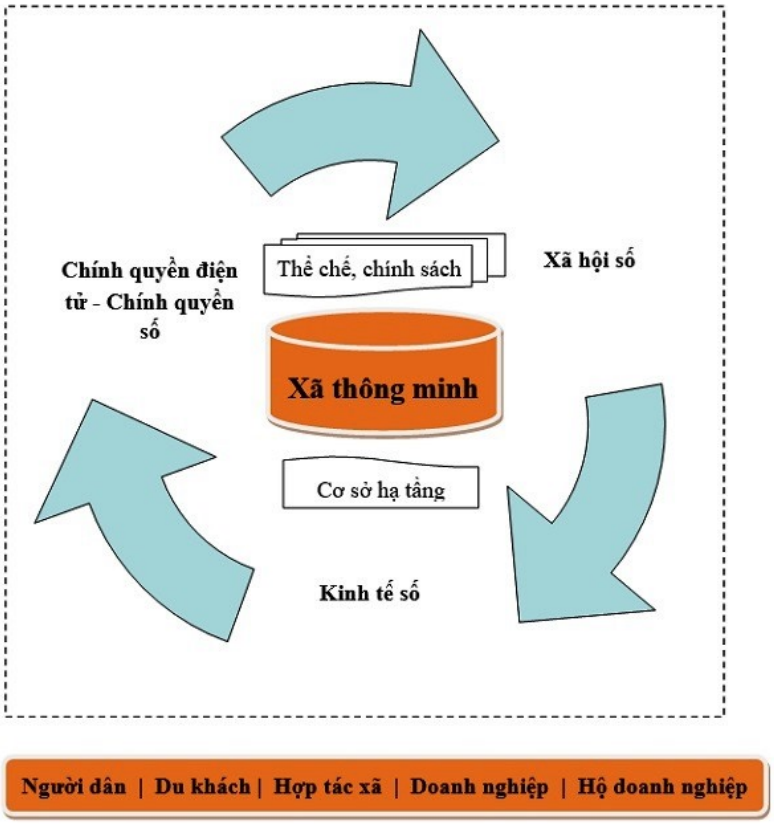


Figure 2.2.6: "Smart village – New-style rural village" model at Huế Province

To the south of Vietnam, Bình Dương has been selected to become the next region to pilot the smart village program. Bạch Đằng commune covers an area of just over 1090 hectares and is surrounded by the Đồng Nai river. The entire commune has nearly 500 households growing pomelo with an area of 400 hectares. Two local traditional specialty pomelos that are preserved and developed by the locals are orange-leaved sweet pomelo and guava pomelo.

According to the Provincial Department of Rural Development Bình Dương, the Bạch Đằng pomelo can greatly benefit from the smart village plan and can become a characteristic subject of study. The People's Committee of Bạch Đằng Commune has planned to organize the implementation of the *Building a Model New-style Rural Commune* project in the 2021 – 2025 period, in which the pilot contents of Smart Villages are integrated, with the focus on infrastructure investment in the 2021 – 2023 period and digital transformation in the 2023 – 2025 period.

2.2. Smart Village System

Figure 2.2.7: Bạch Đằng specialty pomelo

2.3 The Smart Village System in Đồng Tháp

Following the adaptation of the smart village model around the world, Assoc. Prof. Thoại Nam at the Ho Chi Minh University of Technology aims to raise awareness of the technology. Having observed the transition of the smart village all over the world, it is concluded that there is no one-size-fits-all solution[18] and each locality requires a specific solution. Acknowledging the call for a solution, in 2020, a proposal for a Smart Village System (SVS) at Đồng Tháp province was raised, built on the basis of the Provincial Farmers' Assembly model at Đồng Tháp province.

Figure 2.3.1: Setting up an environment monitoring station at a smart village

The proposal strives to create a smart village model to stimulate rural development, improving living and production environments, livelihood opportunities and services as well as governance for sustainable development and recovery based on local strengths and resources. The SVS will solve local challenges in a modern way by applying digital technology in information and communication, as well as other advanced techniques in combination with indigenous values to preserve and develop the values. With the utility of the Internet of Things (IoT), artificial intelligence (AI), blockchain and renewable energy like solar power, the SVS has started permeating multiple aspects of life: security closed-circuit television (CCTV), irrigation combined with monitoring, digital diary, public illumination, etc.

2.3. The Smart Village System in Đồng Tháp

Figure 2.3.2: Smart village platform at Đồng Tháp

At the heart of the SVS is a central big data system that can be connected and shared to operate smart services and applications. Electronic portals and mobile applications help connect residents, authorities and businesses to publish and share legal information, techniques, experience, market information and environmental data. By providing a channel to share information, the users are at an active position to make flexible decisions in production and business, allowing the users to actively select a beneficial strategy over another, thus helping to cut costs and become more competitive, which in turns thrives the economy.

A key difference in the SVS deployment at Đồng Tháp compared to other places is the orientation of developing a base software to become a guideline to replicate and disseminate the system to many different areas, such that it can promote the specificity of a locality but can stay connected and share common information throughout the system. With this model, the implementation of the smart village in Đồng Tháp has many advantages as it has a good starting point.

2.4 Irrigation in the Smart Village System at Đồng Tháp

The Mekong Delta region of Vietnam is at the downstream part of the Mekong River, spreading over 13 cities and provinces. With fertile fields blessed by the nature, the area holds the highest figures in rice production as well as aquatic products, seafood and fruits for both domestic consumption and export.

However, in the dry season, when the water from upstream is low, seawater tends to seep into the fields, causing difficulties for daily live and production. Almost every year, the Mekong Delta provinces experience drought and saltwater intrusions. However, in recent years, this situation became more serious due to negative effect of climate change.

Figure 2.4.1: Damaged crops due to drought

In the dry seasons of 2015 – 2016, 10 of the 13 provinces and cities in the Mekong Delta declared natural disasters caused by drought and saltwater intrusion, and got worse in the dry season of 2019 – 2020. However, due to early forecasting, localities have implemented many solutions to proactively prevent and reduce the damage significantly.

2.4. Irrigation in the Smart Village System at Đồng Tháp

Figure 2.4.2: Locals at saltwater intruded areas receiving freshwater

According to a report from the Ministry of Agriculture and Rural Development, the total area of rice damaged by drought and saltwater is 58,400 hectares, or 14% of that of the previous period; total damaged fruit tree area is 25,120 hectares, or 88% of that of the previous period; about 96,000 households faced difficulty, or 54% of that of the previous period[7].

After this event, it can be concluded that mitigating the damage is possible if more proactive methods are used. These methods include but not limited to: monitoring the source of water, forecasting and promptly provide information about drought and saltwater intrusions, monitoring the use of freshwater and increasing the efficiency of irrigation.

Lead by Assoc. Prof. Thoại Nam and his High Performance Computing (HPC) lab, the Smart Village System (SVS) aims to provide a solution at Đồng Tháp province. With the utility of Internet of Things and High Performance Computing, SVS helps farmers with monitoring their resource usage and adjust behaviours via many channels (i.e.: web, mobile application). By having data collected from sensors that monitor environmental conditions (e.g: lighting, moisture, temperature, etc.) and automatically control devices, the SVS can predict and provide an optimized irrigation management solution for farmers, hence, increase productivity and diminish manual labor.

Their implementation consists of 3 main components: data core, fog node, and edge devices.

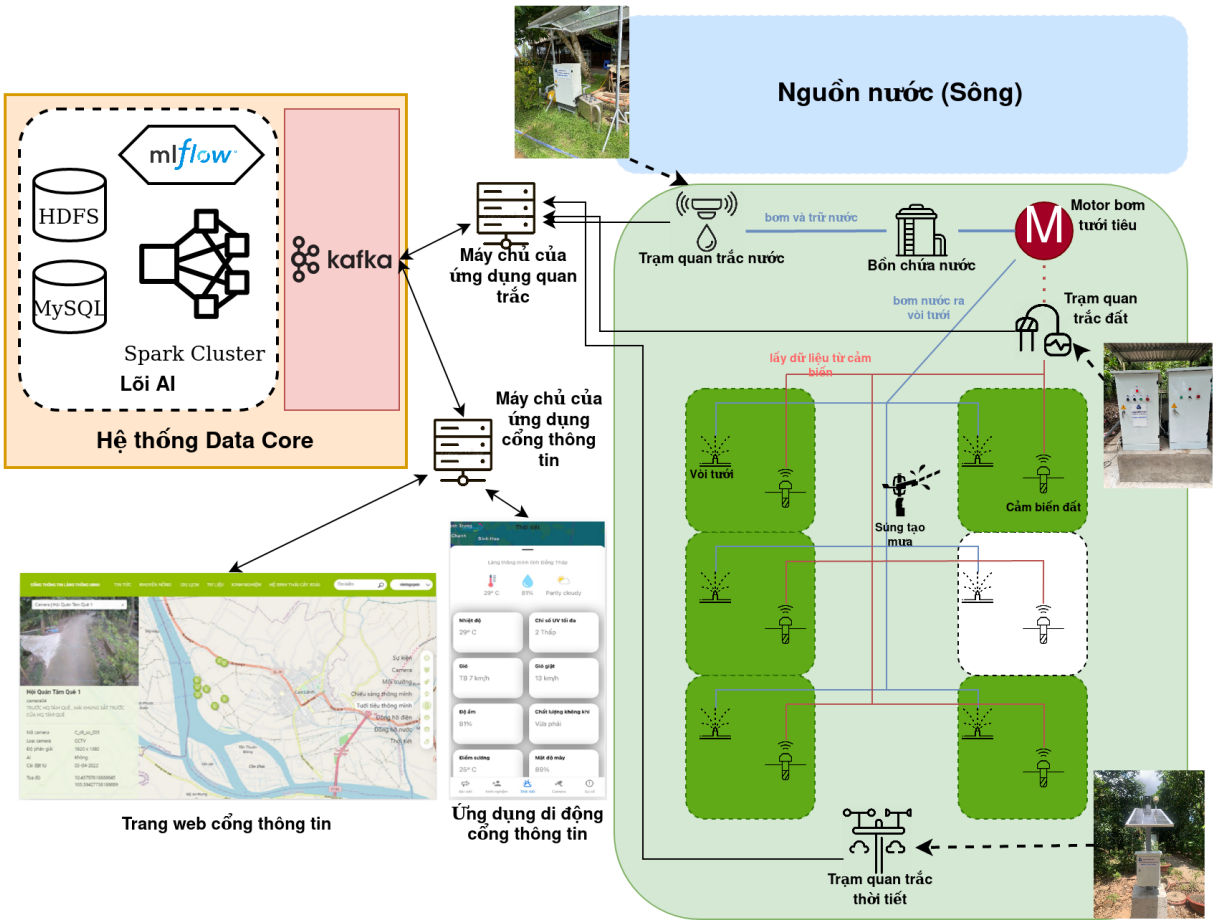


Figure 2.4.3: Architecture of Smart System Village

2.4.1 Data Core System

Data Core System is a physical cluster of high performance computers located on Ho Chi Minh University of Technology at Dĩ An City, Bình Dương Province. It is the multi-purpose core processor for applications such as environmental monitoring, healthcare, education, smart lighting, irrigation, etc. In the scope of Smart Village System, the Data Core System performs calculations, predicts and provides real-time irrigation instructions for users.

At the data core, Spark instances pulling data from the Kafka gateway when the Kafka receives new aggregated data from the fog nodes, then they reformat and store them into a Hadoop Distributed File System (HDFS). As in Figure 2.1.2, the irrigation application at the Data Core has three major working flows:

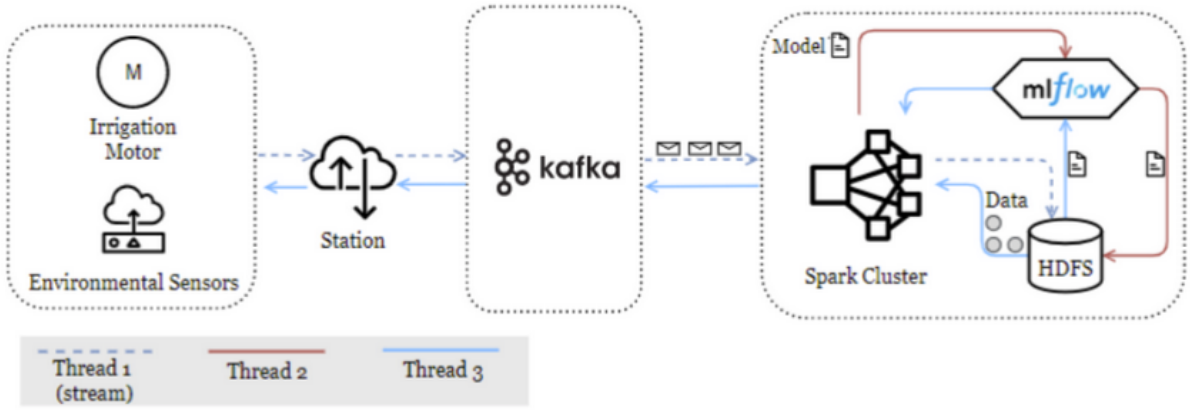


Figure 2.4.4: An overview of the workflows at data core for the irrigation application

- Thread 1: Periodically, environmental sensor data will be gathered, aggregated at the fog server, and sent to the data core cluster. In addition to serving irrigation applications, this cluster also supports various other applications, including environmental monitoring dashboards, crop management apps, etc. Therefore, the data will be delivered to a Kafka module's topic in order to serve several applications. Regarding our irrigation application, we use a Spark cluster to listen to the relevant topic, get data from the Kafka module, and preprocess them prior to putting them in the HDFS store.
- Thread 2: In this flow, our application will get historical data from the HDFS, train a new model, and submit it to an MLflow tracking server. In addition, MLflow is a machine learning monitoring application that seeks to manage, monitor, and codify the machine learning model development process. Therefore, we use this tool for experiment management.
- Thread 3: The application will recommend an appropriate watering strategy based on environmental data. This decision will be sent to the irrigation motor for implementation.

2.4.2 Fog Server

Each application will have a fog server of its own. The fog server will gather data from many gateways for irrigation applications.

Fog Node is essentially a combination of two application types. The first application is responsible for gathering monitoring data from local gateways, as well as receiving instructions and running the irrigation system. Other application is a portal for customers to monitor and manually operate their gardens.

2.4.3 Edge Devices

Edge devices in the system including two main component: gateway and sensor.

There are air gateway, water gateway, soil gateway. Different kinds of gateway are located at different location in the local area to collect, aggregate, and process data from the network of sensors every 5 minutes before sending to the irrigation fog node. The Figure 2.4.5, 2.4.6, and 2.4.7 respectively, shows the gateways collecting data from air, water and soil sensors.

Device (Sensor) Name	**Quantity**
Total Suspended Solid Controller	1
Water pH sensor (water proton H+)	1
Wastewater temperature sensor	1
Water dissolved oxygen sensor	1
Water salinity sensor	1
Soil pH sensor	2
Soil moisture sensor	8
Soil electrical conductivity sensor kit	2
Particulate matter sensor kit (PM2.5, PM10)	1
Weather sensor kit (air temperature, humidity, light intensity and rain flow)	2

Table 2.4.1: Hardware kits and Sensors

Figure 2.4.5: Gateway to collect data from air sensor

2.4. Irrigation in the Smart Village System at Đồng Tháp

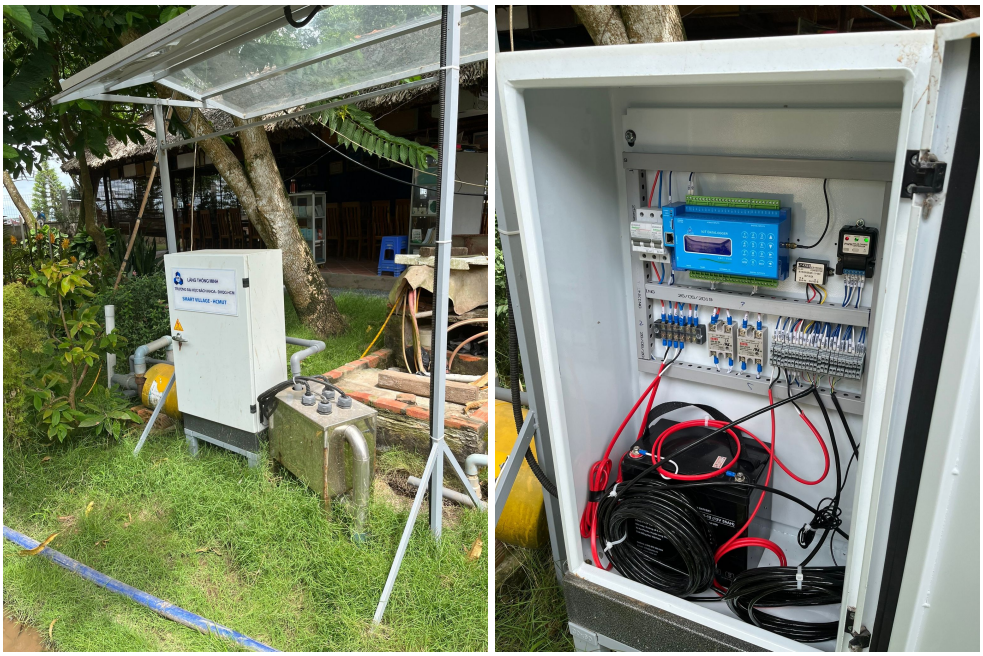


Figure 2.4.6: Gateway to collect data from water sensor

Figure 2.4.7: Gateway to collect data from soil sensor

2.5 Conclusion

Climate is hitting new extremes in the recent years, affecting crops and biodiversity and causing heavy negative effect in many rural areas. Such impact can be mitigated as seen in Section 2.3, but we lack a support and guidance system, nor do we have professional counselling. As we seek for a solution, the smart rural environment appears to solve our problems and handles the lack of infrastructure and services at these areas.

The idea behind smart villages revolves around a shift the paradigm, so that instead of considering small rural populations as recipients of aid, they are equipped with the necessary means to proactively make decisions and productive centers[11]. Some new technologies coming to help the development of rural areas are:

- IOT: e.g. sensors, allowing precise monitoring of crops and taking efficient actions, making the crops more competitive.
- Artificial Intelligence: the adoption of artificial intelligence systems allows for making localized predictions, improving crops and environmental management, as well as in optimizing healthcare and infrastructure use.
- Big Data: The system will generate a huge amount of data, allowing changes in natural conditions to be analyzed more accurately.
- Drones: Hectares of land can be monitored automatically, enabling the monitoring of temperature, crop status, animal and pest movement, thus making the use of pesticide more appropriate.
- Blockchain: Transparency and traceability in agricultural supply chains can be maintained, increasing competitiveness.
- Robots: From autonomous tractors to drones equipped with fertilizer sprayers, the introduction of robots in rural areas helps to cut down on manual labor.

The main obstacle is the digital divide, the difficulty in internet coverage outside large cities, the digital illiteracy, and the lack of training to equip the population with the necessary skills. By applying the internet and associated technologies, the quality of life of the people at rural areas can be significantly improved one step at a time.

Chapter 3

Problem Statement

3.1 Initial Problem

As can be seen in Section 2.1, integrating digital technology into agricultural processes can provide a lot of benefits. In order to have automation in a system, it needs to know a desired output and compares it with the input from the environment, then the control output can be adjusted accordingly to achieve a desired state. By using accurate feedback from monitoring an aspect of the environment, the system can eliminate disturbances, achieve better consistency and stability, all while eliminating the need for human interaction.

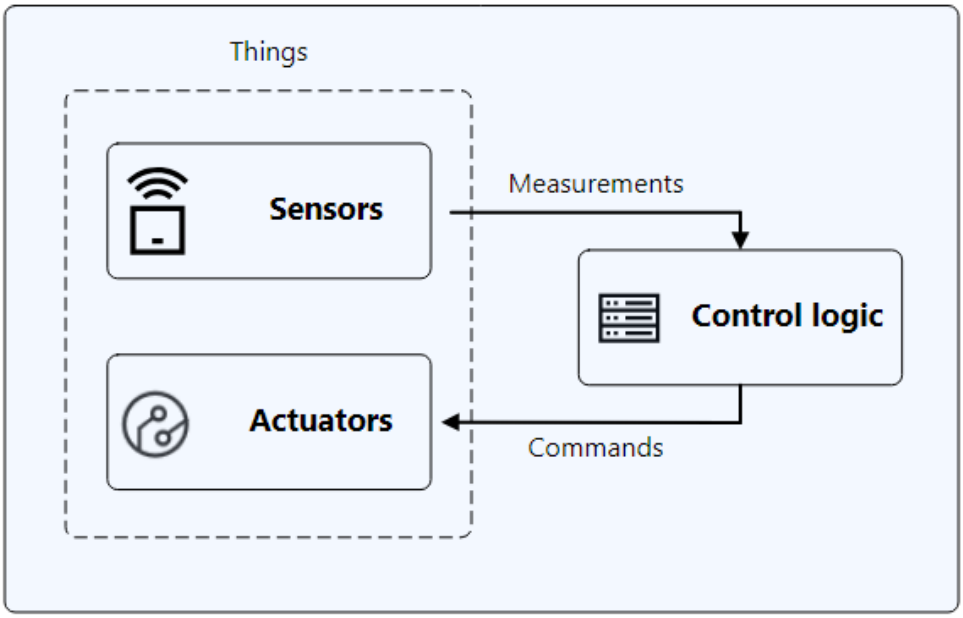


Figure 3.1.1: Basic measure and control loop

A closed measure and control loop is composed of sensors, actuators and a controller. All

three components work in concert to make sure that the device is operating in its environment as expected[12].

- Sensors continually measure current environment conditions and report them to the controller.
- Actuators are the physical components that affect device state. Examples include water pumps, door locks or power switches.
- Control logic uses data collected from sensors and make corrective actions by sending commands to the actuators.

Companies are willing to invest in this technology but the rate of adoption by independent farmers are not particularly high. This is mainly due to the inability to afford the upfront cost for such technology, and their lack of knowledge in this field.

Under the supervision of Assoc. Prof. Thoại Nam, our group have been given permissions to use and extend a solution of the High Performance Computing (HPC) Lab. Instead of having a standalone feedback loop (Figure 3.1.1) at each farm land, only a few sample farms will have the sensors deployed. Data from these sensors will then be processed and used to generate a so-called *scenario*, which is a model containing rules for the controllers. These scenarios are then distributed via the Internet to the controllers at the end user/farmer location. The project has been deployed at Đồng Tháp province (Section 2.2) and has surpassed a lot of expectations.

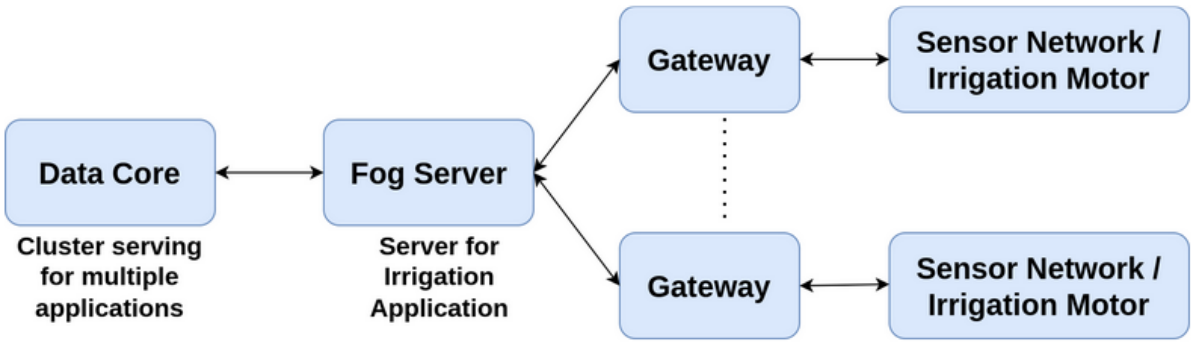


Figure 3.1.2: An overview of the deployment of Irrigation Application

According to a provided reference[14], the current system can take all the aggregated data to create an irrigation scenario, then the gateway devices will fetch these scenarios and use. This is great in terms that the pool of data is big, which helps in creating a more accurate corrective plan. However, this design requires all 3 components of a measure control loop (Figure 3.1.1) to be present at a location of concern, and there is no method to programmatically deliver these scenarios to the gateway devices. Additionally, this design is not very efficient when multiple

kinds of plants or crops are introduced into the equation due to the lack of a schema to manage the irrigation scenarios.

As we worked on the project, we set goals to work and improve based on the current design. We found various solutions and various new challenges, but we are aware that there are no solutions, only trade offs. In the following sections, we have selected some challenges we think is important and will discuss how and why we decided to pick such resolution.

3.2 Challenges and Solutions

3.2.1 Ease the Transition to Smart Irrigation

One of our goals for this project is to lower the investment cost to migrate to the smart village model, we found that the subscription model is a good choice, as compared to the traditional one-off purchase model. With the subscription model, the cost to use and maintain the SVS can be spread out over a longer time span, the customers can always have access to the latest and greatest piece of software, and we can achieve guaranteed income.

With that said, we want to introduce the subscription model into the SVS. Every once in a while, users will be asked to pay a subscription fee to register or renew their license in order get the latest irrigation schedules for their farm based on their location and plant types. Users can subscribe to whatever types of plant type their want or customize the schedule on their needs, we will make sure to provide the flexibility for end users as much as possible.

For free users, they have already installed our equipment and devices, so they have all the access to their devices, including manual device control and manually scheduling irrigation for their land.

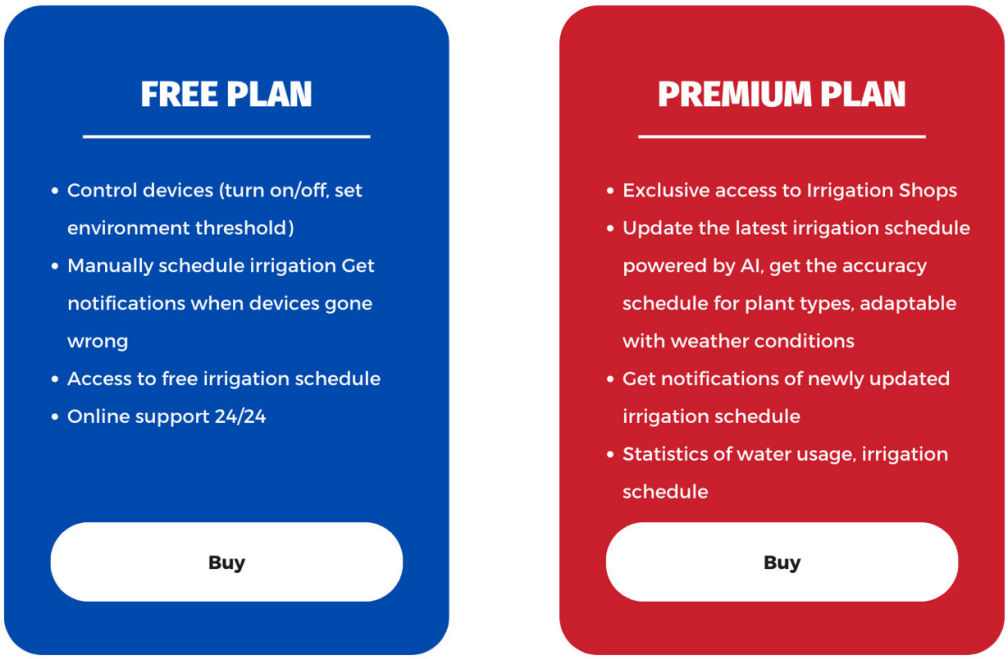


Figure 3.2.1: Example of plans in a subscription

For premium users, they have access to all features a free user have. Plus, they gain exclusive access to the Irrigation Shop, which will display all of our current seasonal irrigation

schedule for subscription. Premium user can subscribe to what schedule fit their plants and their devices. They will also get notifications of new version of irrigation schedule arrives and view the statistics of water usage, irrigation schedule from all of their devices.

Ideally, the subscription fee we charge users comes with the responsibility to provide them an end to end service, that is when the local service providers take their role. Service provider is a third party company located in the same area with users, acts as the intermediary between the central node and end users. They will be in charge of taking care of users when they need support, providing installation and maintaining the equipment. They will have their own Irrigation Shop, which received all the latest schedule from the Central Node, manage and adding advertise or description before distributing them to subscribed users.

In that way, we would still maintain the technical aspects of Central Node and still doing our best to enhance the experience and the accuracy of product without having to worried much about the business side. Meanwhile, the local service provider will take care and strive the business and receive a partial of subscription charge from them.

3.2.2 Scenario versioning

Based on plant type, season and geographical location, the *DataCore* system synthesize an irrigation scenario that eventually used by end-users. The current scenario is now lacking a versioning mechanism that separate one scenario from one another. As we work on the project, we will answer the following questions to versioning sequences of scenarios that users can decide to update new one or reverse to old version:

- Which criteria to distinguish a unique scenario?
- Different version labels for long term vs. short term plants

Attempt 1

In this iteration, we use an incremental mechanism to label the version for new arrival irrigation scenarios. We came up with 2 solutions:

- Integer labelling: from 1 to infinity. New arrival scenario will increase the label by 1.
- Date labelling: new arrival scenario is labelled with the current arrival date. Sequentially, the new scenario has the higher version than the old one. E.g: 20220103, 20220104

However, as we worked with the group in charge of irrigation, we found that the nature of a scenario can be separated into 3 category:

- Short term scenario: act as immediate action, to combat with sudden events like a shower.
- Long term scenario: schedule irrigation, irrigate instructions for several months.
- Conditional scenario: irrigate based on environmental conditions.

As such, we need to provide a better labelling mechanism with more semantics. Sometimes, there will be an updated scenario for a specific types of plant to combat with a harsh weather condition. How should we know that information if the label is incremental?

Attempt 2

Semantic Versioning (SemVer) is a versioning scheme for using meaningful version numbers. SemVer works by structuring each version identifier into three parts: **MAJOR**, **MINOR**, and **PATCH**. Together, they can notate a version of a piece of data as seen in Figure 3.2.2.

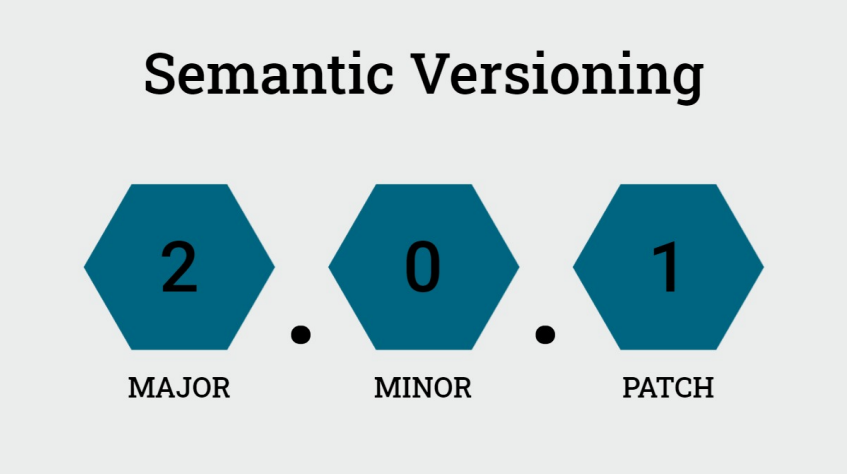


Figure 3.2.2: Structure of Semantic Versioning

Each of these parts is managed as a number and incremented according to the following rules:

- **PATCH** is incremented for bug fixes, or other changes that do not change the behavior of the API.
- **MINOR** is incremented for backward-compatible changes of the API, meaning that existing consumers can safely ignore such a version change.
- **MAJOR** is incremented for breaking changes, i.e. for changes that are not within the backwards compatibility scope. Existing consumers have to adapt to the new API, very likely by adapting their code.

The semantic versioning release structure is best modelled as a tree. At the top, you have your public interface changes, each of which result in a new major number. Every major series has its own set of minor releases, where new functionality is added in a backwards compatible manner. Finally, minor releases may receive bug-fixing patches from time-to-time

Adapting the definition with the nature of scenario , we have a semantic versioning of our own, which can be defined as:

- **PATCH** is incremented for immediate actions scenario (turn on/off sprinkler immediately), which required to done as soon as possible.
- **MINOR** is incremented for each fixed time period, when scenario is updated to adapt with the growth stage of plants.
- **MAJOR** is incremented for breaking changes, i.e.: new crop, increment once a year

Further discussion on the fixed period for new update for short-term and long-term plant is discussed in Section 3.2.4

Conclusion

So far, **Semantic Versioning** has proven to be a better choice to provide more meaningful labels for the irrigation scenario. But for the sake of simplicity, we will not implement the whole features. Specifically, modelling as tree structure, allowing the deployment of multiple branches at the same time will not be presented in our thesis. Our thesis will use *Semantic Versioning* in the manner of *incremental labelling* only.

3.2.3 Connectivity Between the Mobile Application and Gateway Devices

The mobile applications and gateways devices will need to communicate with each other. There are multiple methods to achieve such results, however they all come with their drawbacks. We will discuss how our findings work and what trade off we are willing to take.

Attempt 1

As we go through the use cases of our system, we found that both the mobile applications and gateway devices are likely behind a feature called Network Address Translation (NAT), which is used when multiple devices want to share a network. On the gateway device side, this is understandable because you wouldn't want to run a new internet connection for every new device. The same case applies for mobile devices, either via WiFi connection or cellular

network. People usually fall into a misconception that only one connection is allowed by the carrier for each SIM card, and each SIM card can only be used by one mobile device, thus there is no connection being shared. This is untrue because the SIM card is only used to identify and authenticate a user with a carrier provider, upon successful authentication, a mobile device will connect to a network access point over wireless cellular network, which is the same as connecting to a router over WiFi.

In short, Network Address Translation (NAT) is used to share a connection to multiple devices. The devices live in their own local area network and are assigned private IP addresses, while the shared connection is assigned a public IP address. NAT works by mapping private addresses to the public address and sometimes public address to private addresses. However, mapping public addresses to private addresses is troublesome as most of the configuration is manual. Additionally, there may be cases where two devices requests for different information from a remote address, but NAT is not aware of this content and forwards all information to all configured devices regardless, causing undefined behavior. Stateful firewalls are introduced to work with NAT to solve such cases. With this dual setup, the firewall can remember which private address *recently* requested some data from what remote address and tell NAT to forward the response accordingly, while blocking all connections initiated from the public address.

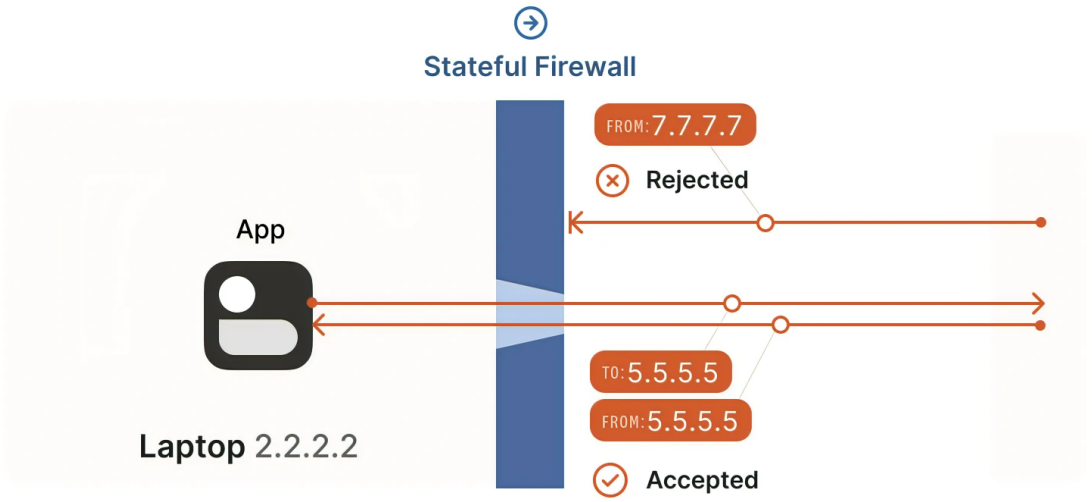


Figure 3.2.3: Stateful firewall

Manually forwarding a port over NAT requires that the ports of the public and private addresses be known and be set on the devices respectively. This method is highly technical and tightly coupled for each case, and might create further risks if the configurations are unattended.

Fortunately, the original solution from the HPC Lab brings a solution. A server always reachable via its public IP address will host a Kafka topic, acting as a common channel for

communication. This means that when a device wants to send some data to one another, a server will store this signal, then the other device can read this message and act accordingly.

However, with this design, we have just successfully created a single point of failure. The topic on the server needn't have a problem, but any degradation between one device to another is enough to cause disruption in the service, e.g. breakage in the internet infrastructure or lack of internet coverage in a certain area.

Attempt 2

As we focus on breaking free from the server, we found that peer-to-peer networking is a viable option, but then we are brought back to the NAT and stateful firewall problem.

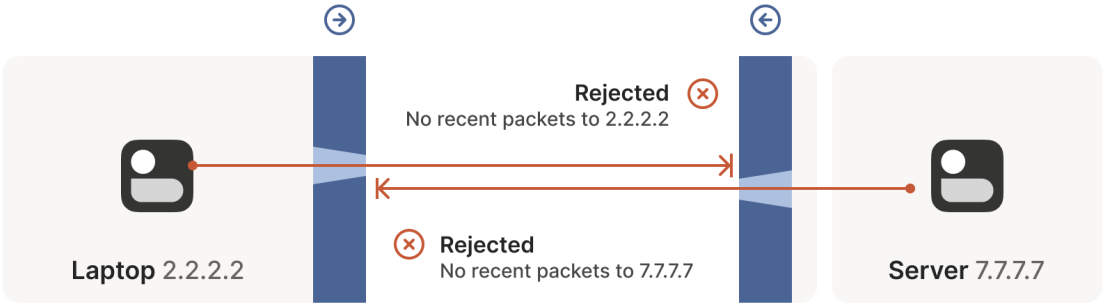


Figure 3.2.4: Firewalls facing each other

Many attempts have been made, to name a few: Session Traversal Utilities for NAT (STUN) protocol, Traversal Using Relays around NAT (TURN) protocol, and Interactive Connectivity Establishment (ICE) protocol, and since all of these are based on TCP, the general term for these protocols is TCP hole punching. Similarly, there is UDP hole punching, which is arguably more simple.

TCP hole punching adds a layer complexity over UDP hole punching due to how TCP is defined. TCP is a connection-oriented protocol that requires a three-way handshake to establish a connection between two endpoints, meaning the NAT device needs to be aware of the connection and allow it to pass through.

In contrast, UDP is connectionless and requires no handshakes. The rule for UDP is very simple: the firewall allows an inbound UDP packet if it previously saw a matching outbound packet. In other words, packets must flow out before packets can flow back in[2].

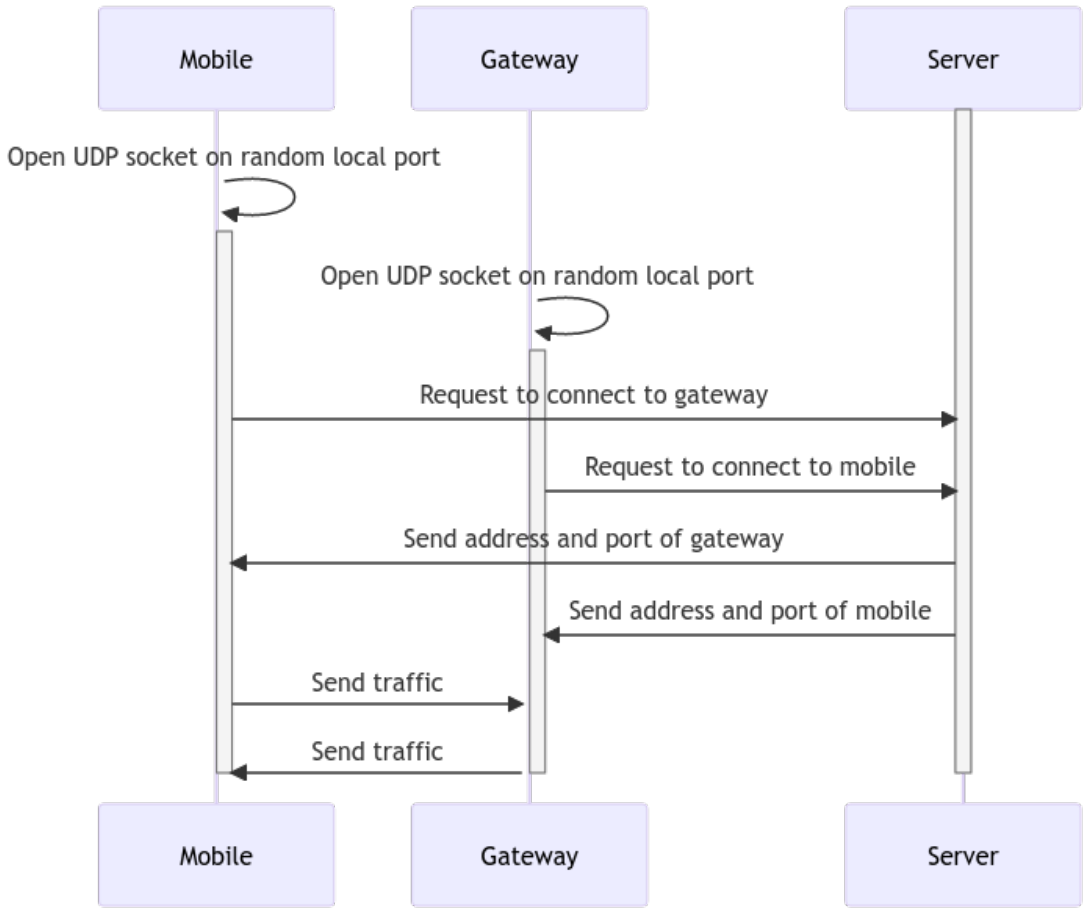


Figure 3.2.5: UDP hole punching

To keep things simple, UDP hole punching goes something like:

1. Two peers both open a UDP socket bound to a local random port.
2. Both peers contact a server on the internet.
3. The server reads the public address of both peers and exchange them.
4. Both peers start trying to send traffic to each other.
5. NATs will see that there have been recent outbound connections to an external address, and said external address is now send some data. Hopefully, NATs will happily forward this data to the local devices.

This is not fully peer-to-peer due to a server being needed to initiate the connection. However, by having a mechanism for the devices to connect directly to each other, we have paved paths for some interesting ideas.

Consider the case where internet coverage is unavailable, we can still have a router set up to manage a local network of gateway devices. Because we no longer need a server for our main operations, we can have the mobile application connect directly with the gateways within a local network, either via static IP addresses or via mDNS. This is great in terms that the server cannot control the operation of the gateways anymore, meaning we have just achieved a great boost in transparency and trust in the service. Additionally, in the case of a nested-router setup, a light-weight local discovery service may be provided.

Nonetheless, implementing such feature is not an easy task:

1. Direct control over the network socket is required. As a rule, we cannot take an existing network library and make it traverse NATs, because we are sending traffic that are not part of the *main* protocol. For example, the gateways managed by HPC lab accepts MQTT packets, which is based on TCP. As a result, a complete rewrite of the MQTT library is required, and an implementation has do be done for each type of device used in our system.

2. Considering the case of lack of internet connectivity, we need to find a method to verify the device ownership offline. Security has to be the top priority as the users have all the time in this world to mess with our devices, like packet spoofing.

Conclusion

With the current goals of the project, we find that UDP hole punching is the most suitable in the long term. Regardless, implementing such feature for the system requires the collaboration of all parties involved. As such, we believe it is better to wait for more opinions from more related group before we move on with this connectivity challenge.

3.2.4 Scenario Update Period

A scenario is generated based on the recent data, and once it is pushed into gateways, it will not change. This is troublesome as the climate is not the same every year, and each growth state of a crop might vary in different kinds of environment. Furthermore, long-term plants tend to have less strictly controlled irrigation than short-term plants. The irrigation scenarios for short-term plants need to be updated more frequently due to the rapid change of their growth state. This requires restructuring of the scenarios and distribution mechanism to adapt the frequency of new update.

3.2.5 Scenario Distribution

As we have one common data processor serving a group of gateways, we need to control what scenario to deliver to which gateway, for example you would not want to apply a watermelon irrigation scenario for pepper.

Resolving to a low level solution like distributing scenarios via different internet address is too complicated and impractical, meaning we need to implement a high-level solution such that the user can configure what scenario goes to which controller. By providing an application, we can track which gateway uses what kind of scenario, and we can allow the users to change this option if they are registered with the system.

3.2.6 Event Notification

During the growth of a plant, there will be some special environmental events (e.g: sudden increase in northeast wind, heavy rain, drought, pests, etc.), subscription or scenario-related notifications that needs specific action from users. We need a general method to listen all of these events on mobiles and act with defined actions.

With the introduced services from Section 6.1.7, we use Firebase Cloud Messaging (FCM) as a carrier to distribute our messages to all available devices. Clients once registered to the Firebase will receive a messaging token to submit to our server. When our server have notifications that need distributing, it can decide which users to send to, equivalently which messaging tokens to send notifications to.

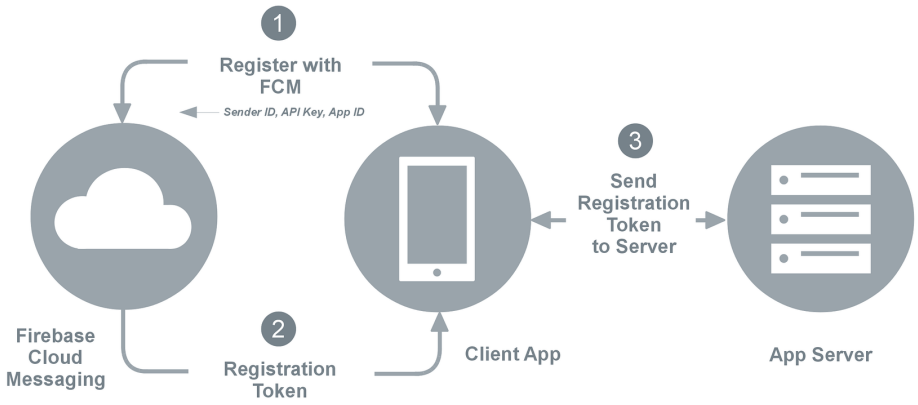


Figure 3.2.6: Firebase Cloud Messaging working principles

However, when the token expires or user has been inactive for a while, the notification won't be sent to user's device via FCM. That's when we need a mailbox storage to store all the previous notifications, so we can still see all notifications from the past.

3.2.7 Enhance User Experience and Retention

It will be ineffective to only use scenarios from the sample garden if we do not consider how users use our scenarios. Sometimes, the sample gardens have different soil moisture or temperature than a farmer's garden, thus they would want to turn off the pump based on their knowledge of having taking care the garden for years. In that case, it is more beneficial to collect their action than to utilise a pre-made template. Hence, using that collected data, our data center can enhance the our products (irrigation scenario) to better solutions for users.

Furthermore, users after applying irrigation scenarios to their garden, without anything else to do, they tend to abandon the application for a long time. We want to collect the agricultural activities on the user's garden to support the Digital Diary mentioned in Section 2.2. More information about the user's input may mean more details we have about the garden. Hence, the products harvested from the garden can be easily valuated and certified with standards (e.g.: *VietGAP*, *GlobalGAP*, *USDA*, ..), increasing the internal values of the products. Only then, users are more willing to use the application for their benefits.

Solution

In this matter, we're collecting irrigation scenarios that customized by users as well as their irrigation activities, so we can have a better grasp on how users actually use the application. It can came from:

- When user manually turn on/off devices
- How often user used our pre-made scenarios, which plants are most used
- Water amount used for each irrigation
- Related activities: fertilizing, harvesting, pesticides,...

Figure 3.2.7: Allowed activities

Chapter 4

Requirements Analysis

4.1 Functional Requirements

After the valuable time discussing with Assoc. Prof. Thoại Nam, we could draw out the user requirements, including functional and non-functional ones. We first investigate all parties engaged in the creation and operation of the system. They are the system's stakeholders, as shown in the table below.

Stakeholder	**Description**
End User	Farmer - subscribe, receive irrigation schedule from service providers and control the automated irrigation system via mobile.
Service Provider	Local company - accountable for monitoring, providing irrigation schedule for end-user, managing end users and irrigation devices.

Table 4.1.1: System stakeholders

4.1.1 End User

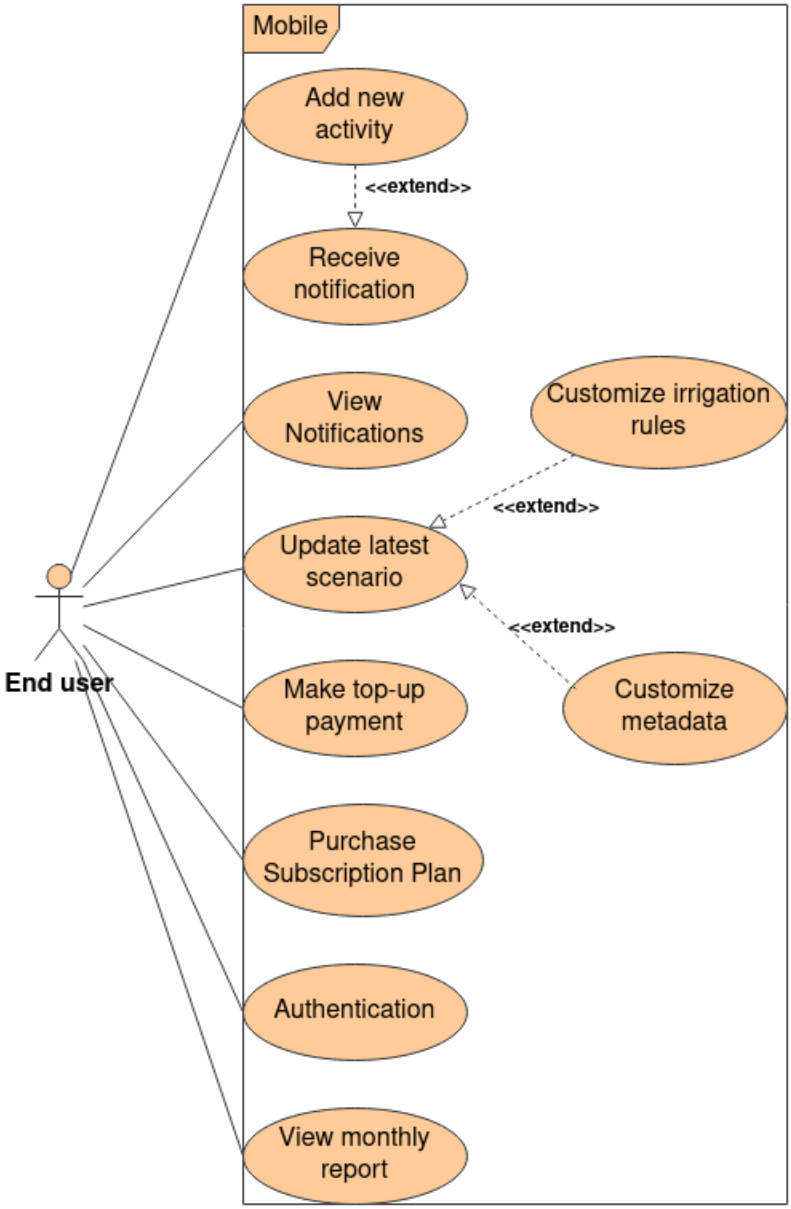


Figure 4.1.1: End user's use case diagram

- Manage personal account:
  - Login
  - Logout
  - Password recovery
- Manage subscription:
  - View available subscription plans
  - Register new subscription plan for a plant type
- Manage irrigation scenarios:
  - Create new user irrigation scenarios based on registered template scenarios with the service provider
    - Irrigate instantly with a certain amount of water
    - Schedule automatic irrigation with irrigation amount and rainwater threshold
    - Irrigate based on different weather conditions (rain threshold, temperature threshold, soil threshold)
  - Execute irrigation scenarios to pump devices through gateway
  - Update user's irrigation scenario information
  - View detailed information of user's irrigation scenarios
  - Delete user's irrigation scenarios
  - Receive the latest template irrigation scenarios from the service provider
- Manage notification:
  - Receive new notifications from service providers (New scenarios, Scenario expiration, Weather alerts, ...)
  - Execute or update scenarios from new notifications
- Manage farming activities:
  - Manage farm:
    - Create new activity based on plant type, action type and labor cost
    - Update information on farm management activity
    - View detailed information on farm management

- Manage fertilization for plants:
  - Create new activity based on plant type, fertilization method, fertilizer brand, quantity, and labor cost
  - Update information on fertilization activity for plants
  - View detailed information on fertilization activity for plants
- Manage pesticide spraying:
  - Create new activity based on plant type, pest type, pesticide type, brand, quantity, and labor cost
  - Update information on pesticide spraying activity
  - View detailed information on pesticide spraying activity
- Manage harvesting:
  - Create new activity based on plant type, quantity of plants harvested, and labor cost
  - Update information on harvesting activity
  - View detailed information on harvesting activity
- Manage product sales:
  - Create new activity based on plant type, quantity of plants sold, and revenue
  - Update information on product sales activity
  - View detailed information on product sales activity
- Manage natural disaster or pest alerts:
  - Create new activity based on plant type, image of diseased plants or natural disaster
  - Update information on natural disaster or pest alert
  - View detailed information on natural disaster or pest alert activity

4.1.2 Service Provider

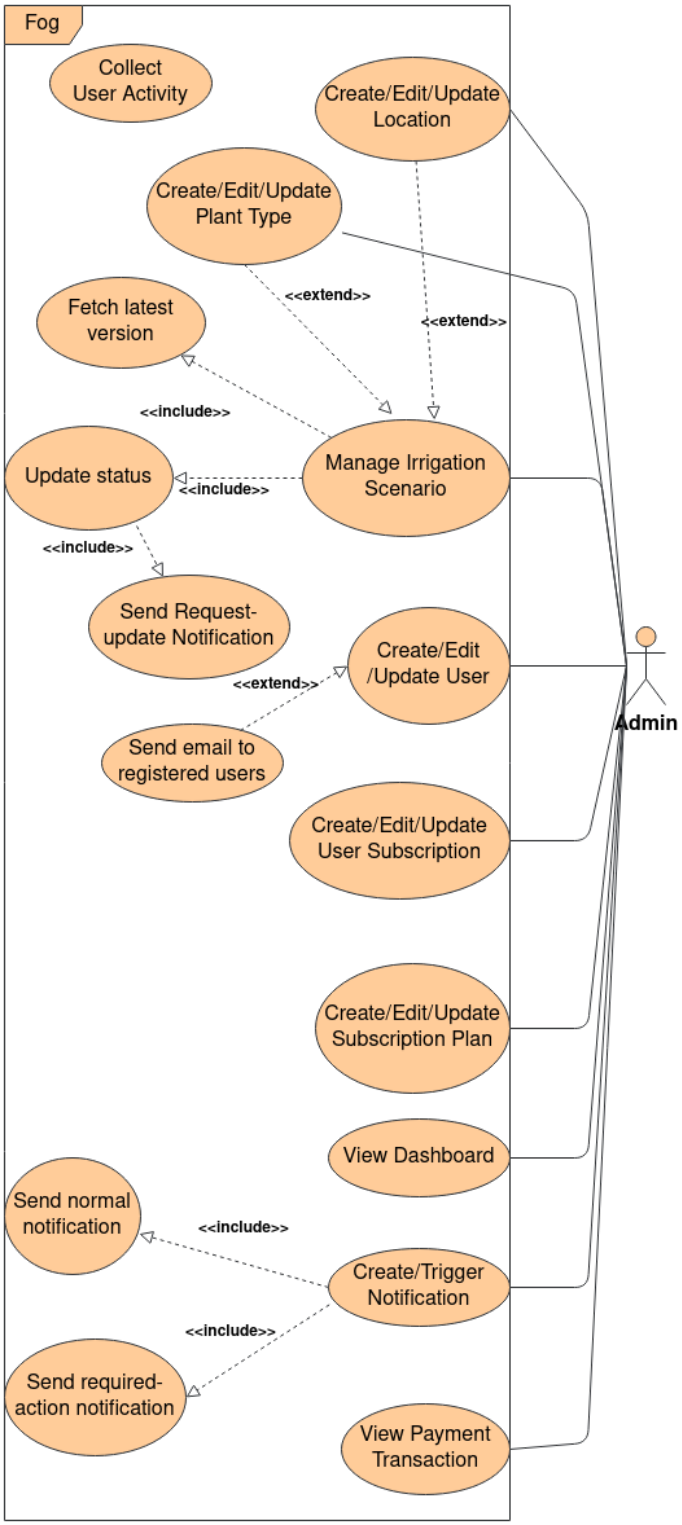


Figure 4.1.2: Service provider's use case diagram

4.1. Functional Requirements

- Manage end-users (customer)
  - View list of users
  - Create new user with defined password
  - Remove user
- Manage plant type
  - View current defined plant types
  - Create new plant type with defined attributes (name, description, unique code,..)
  - Edit/remove plant type
- Manage location
  - View current defined locations
  - Create new location with defined attributes (name, description, unique code,..)
  - Edit/remove location
- Manage subscription plan
  - Create new subscription plan
  - View current subscription plan
  - Edit subscription plan
  - Remove subscription plan
- Manage subscription of irrigation scenarios for end-users:
  - Create new user subscription
  - View users subscription
  - Edit user subscription
  - Remove user subscription
- Manage irrigation scenarios:
  - Fetch new scenarios from *Data Core system*
  - Edit metadata from fetched irrigation scenarios (plant type, season, description, and versioning)
  - Remove scenarios

  - Update irrigation scenario status. **RUNNING** status will make the scenario published and displayed to Irrigation Shop. **STOP** status make the scenario removed from Irrigation Shop.

- Manage notifications:
  - Push notifications to end-users based on their subscription plan, geographic location, and plant type
  - Send event notifications, with assigned action
  - Send notifications to all users

- Statistics of end-user activities:
  - Includes charts and tables of recent end-user activities. Based on this information, the service provider has an overview and can see the effectiveness of the user's activities and current scenarios.

4.2 Non-functional Requirements

Next, we will review all of the system's non-functional requirements, which include limitations and anticipated system behavior from various perspectives. The following table categorizes all non-functional system requirements into product requirements and organizational requirements.

4.2.1 Product requirements

- Under optimal network circumstances, the application may instantaneously update the irrigation scenario to devices with user's consent.
- Ensure that users may get notifications of upcoming irrigation scenarios under less-than-ideal network circumstances.
- Irrigation mobile application must work reliably under any condition, not having internet.
- All user's irrigation scenario still work regardless the decommission of that scenario in Irrigation Shop.
- All scenarios appeared in Irrigation Shop is the most updated scenarios for a specific plant type and location.

Compatible requirements

- Application must compatible with multiple devices, web browser.

4.2.2 Security requirements

- Use the SHA-256 cryptographic hash algorithm to store passwords.
- Authorize user with owned resources, but not with others resources.
- Tolerant with SQL injections, especially with login process.
- Authenticate users using Json Web Token (JWT)

4.2.3 UX/UI requirements

- The interface should be easy to view, easy to use, and use colors that are relevant to the agriculture industry.
- Good interactivity capabilities.

- Workflow for each feature is straight forward, user-friendly
- The application needs to be easy to install

4.2.4 Availability requirements

- Operating time: always available.
- Maintenance time: holidays, system upgrades,...

4.3 Use-case Tables

4.3.1 End users (e.g. farmers)

Login

User story: As an end-user, I want to be able to log in to use the application. Login information includes email and password. If I haven't logged in, the system will redirect to the login page.

Name	Login
ID	IR-01
Actor	Farmer
Description	The user must enter an account, password to use the application
Pre-condition	The user has been provided with an account by the service provider and is on the login screen
Basic flow	1. Enter the email the user has received 2. Enter the password 3. Click the login button
Alternative Flow	N/A
Exception Flow	• 3a. The user enters an incorrect email. Request login again. • 3b. The user enters an incorrect password. Request login again.
Post-condition	The user has logged in and can use the application

Table 4.3.1: Use case specification: Login

Create user irrigation scenario

User story: As a user of the system, I want to be able to create a new user scenario based on a template scenario attached to the plant type registered with the premium plan from the service provider. If I have not registered that plant type with the service provider, I will not be able to create a new user scenario.

Name	Create user irrigation scenario
ID	IR-02
Actor	Farmer
Description	The user creates a new irrigation scenario based on a subscription plan.
Pre-condition	The user has registered the plant type with the service provider along with the devices they have. The user has successfully logged into the application and is on the Scenario screen.
Basic flow	1. Click the "+" button on the right side of the header bar 2. Choose one of the available template scenarios from the displayed list of template scenarios 3. Click the "Add" button in the New Rules section to add a new rule for that scenario 4. Enter detailed information based on the type of template scenario the user has selected 5. Click the "Next" button on the right side of the header bar to go back to the previous screen 6. Click the "Next" button on the right side of the header bar to proceed 7. Enter the name of the user scenario the user wants to create 8. Click the "Save" button to create the new scenario
Alternative Flow	• 2a. The user clicks the "Back" button to cancel creating a new scenario • 6a. If the user does not click the "Next" button but instead clicks the "Add" button, they will go back to step 3 to add a new rule.
Exception Flow	• 5a. If the user does not fill in all the required fields, the "Next" button will be disabled • 8a. If the user does not enter a scenario name, the "Save" button will be disabled
Post-condition	The user has successfully created a new scenario.

Table 4.3.2: Use case specification: Create user irrigation scenario

Execute User Scenario

User story: As an end-user, I want to be able to execute the irrigation scenario based on the scenarios that I have created before.

Name	Execute User Irrigation Scenario
ID	IR-03
Actor	Farmer
Description	The user executes the irrigation scenario from the scenarios they have created.
Pre-condition	The user has logged into the application and is on the Scenario screen. They have also created various types of scenarios.
Basic flow	1. Select a scenario that the user has created. 2. Press the switch button to change the scenario status.
Alternative Flow	• 2a. If the status is off. Press the switch button to change the status to on. • 2b. If the status is on. Press the switch button to change the status to off.
Exception Flow	N/A
Post-condition	The user successfully executes the scenario.

Table 4.3.3: Use case specification: Execute User Scenario

View User Scenario

User story: As a user of the system, I want to be able to view the irrigation scenario based on the scenarios I have created before.

Name	View User Irrigation Scenario
ID	IR-04
Actor	Farmer
Description	The user views the irrigation scenario from the scenarios they have created.
Pre-condition	The user has logged into the application and is on the Scenario screen. They have also created various types of scenarios.
Basic flow	1. Select a scenario that the user has created. 2. The screen displays detailed information about the user's created scenario.
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	The user can view the details of the scenario they have created.

Table 4.3.4: Use case specification: View User Scenario

Delete User Scenario

User story: As a user of the system, I want to be able to delete the irrigation scenario based on the scenarios I have created before.

Name	Delete User Irrigation Scenario
ID	IR-05
Actor	Farmer
Description	The user deletes the irrigation scenario from the scenarios they have created.
Pre-condition	The user has logged into the application and is on the Scenario screen. They have also created various types of scenarios.
Basic flow	1. Select a scenario that the user has created 2. Swipe left to select the delete button 3. Click the "Trash" button to delete the scenario
Alternative Flow	2a. If the user swipes right, the scenario will not be deleted
Exception Flow	N/A
Post-condition	The user can view the details of the script they have created.

Table 4.3.5: Use case specification: Delete User Scenario

Creating Farm Management Activity

User story: As a system user, I want to be able to manually create a farm management activity to record farming activities at a specific time.

Name	Creating Farm Management Activity
ID	IR-06
Actor	Farmer
Description	The user creates a farm management activity with information such as plant type, action type, labor cost, and notes.
Pre-condition	TThe user has logged into the application and is on the Activity screen.
Basic flow	1. Select the "+" button to display a list of activity types. 2. Select "Farm Management." 3. Enter information such as plant type, action type, labor cost, and notes. 4. Select the "Save" button to create a new activity.
Alternative Flow	N/A
Exception Flow	3a. If the user leaves a required field blank, the "Save" button will be disabled.
Post-condition	The user successfully creates a farm management activity.

Table 4.3.6: Use case specification: Creating Farm Management Activity

Create Fertilizer Activity

User story: As a system user, I want to be able to create a manual fertilizer activity to record farming operations at a specific time.

Name	Create fertilizer activity
ID	IR-07
Actor	Farmer
Description	The user creates a farm management activity with information such as plant type, fertilization method, fertilizer brand, unit, quantity, labor cost, and note.
Pre-condition	TThe user has logged into the application and is on the Activity screen.
Basic flow	1. Select the "+" button to display a list of activity types. 2. Select "Fertilizer" 3. Enter information such as plant type, fertilization method, fertilizer brand, unit, quantity, labor cost, and note 4. Select the "Save" button to create a new activity.
Alternative Flow	N/A
Exception Flow	3a. If the user leaves a required field blank, the "Save" button will be disabled.
Post-condition	The user has successfully created a fertilizer activity.

Table 4.3.7: Use case specification: Create Fertilizer Activity

Creating a Pesticide Spraying Activity

User story: As a system user, I want to be able to manually create a pesticide spraying activity to record farming operations at a specific time.

Name	Creating a Pesticide Spraying Activity
ID	IR-08
Actor	Farmer
Description	The user creates a farm management activity with information such as plant type, pest type, pesticide type, pesticide brand, unit, quantity, labor cost, and notes.
Pre-condition	TThe user has logged into the application and is on the Activity screen.
Basic flow	1. Select the "+" button to display a list of activity types. 2. Select "Pesticides" 3. Enter information such as plant type, pest type, pesticide type, pesticide brand, unit, quantity, labor cost, and notes 4. Select the "Save" button to create a new activity.
Alternative Flow	N/A
Exception Flow	3a. If the user leaves a required field blank, the "Save" button will be disabled.
Post-condition	The user has successfully created a pesticide spraying activity.

Table 4.3.8: Use case specification: Creating a Pesticide Spraying Activity

Create Harvest Activity

User story: As a system user, I want to be able to create a manual harvest activity to record farming activities at a specific time.

Name	Create Harvest Activity
ID	IR-09
Actor	Farmer
Description	The user creates a farm management activity with information such as plant type, quantity of plants harvested, units, labor cost, and notes.
Pre-condition	The user has logged into the application and is on the Activity screen.
Basic flow	1. Select the "+" button to display a list of activity types. 2. Select "Harvest" 3. Enter information such as plant type, quantity of plants harvested, units, labor cost, and notes 4. Select the "Save" button to create a new activity.
Alternative Flow	N/A
Exception Flow	3a. If the user leaves a required field blank, the "Save" button will be disabled.
Post-condition	The user has successfully created a harvest activity.

Table 4.3.9: Use case specification: Create Harvest Activity

Creating pest and disaster alert activities

User story: As a system user, I want to be able to manually create pest and disaster alert activities to record farming activities at a specific time.

Name	Create pest and disaster alert activities
ID	IR-10
Actor	Farmer
Description	The user creates a farm management activity with information such as plant type, captured image, and description.
Pre-condition	The user has logged into the application and is on the Activity screen.
Basic flow	1. Select the "+" button to display a list of activity types. 2. Select "Disaster/Pest Warning" 3. Enter the plant type and note. 4. Select the "Take photo" button to capture evidence or "Select photo" button to get photo from albums. 5. Select the "Save" button to create a new activity.
Alternative Flow	N/A
Exception Flow	3a. If the user leaves a required field blank, the "Save" button will be disabled.
Post-condition	The user successfully creates a pest and disaster alert activity.

Table 4.3.10: Use case specification: Creating pest and disaster alert activities

Notification Management

User story: As a system user, I want to be able to configure whether or not I receive notifications from the service provider. If I decline to receive notifications, I will not be bothered by new notifications but can still view the details of that notification in the Notification section. If I allow notifications, the notification will take me directly to the notification details screen.

Name	Notification Management
ID	IR-11
Actor	Farmer
Description	The user can configure to turn on/off notifications from the service provider.
Pre-condition	The user has logged in to the application and is on the Account screen.
Basic flow	1. Select "Notifications" 2. Select "Change" to navigate to the phone settings to configure notification on/off
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	The user can change the notification status of the application.

Table 4.3.11: Use case specification: Notification Management

View list of notifications

User story: As a user of the system, I want to be able to view a list of notifications from the application based on various events. Notifications could be updates to the latest version of the irrigation script, weather change notifications,...

Name	View list of notifications
ID	IR-12
Actor	Farmer
Description	The user can view detailed notifications sent from the service provider
Pre-condition	The user has logged into the application and is on the Notification screen.
Basic flow	1. Select any notification displayed on the screen 2. Display notification details and some actions the user can interact with the scenario
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	The user can view detailed incoming notifications

Table 4.3.12: Use case specification: View list of notifications

4.3.2 Local service providers

Login

User story: As a staff of service provider, you want to able to login into the system using provided email and password.

Name	Login
ID	AD-01
Actor	Admin
Description	The user must enter an account, password to use the application
Pre-condition	The user has been provided with an assigned account and have clicked on the login screen
Basic flow	1. Enter the email the user has received 2. Enter the password 3. Click the login button
Alternative Flow	N/A
Exception Flow	• 3a. The user enters an incorrect email. Request login again. • 3b. The user enters an incorrect password. Request login again.
Post-condition	The user has logged in and can use the application

Table 4.3.13: Use case specification: Login

Logout

User story: As a staff of service provider, you want to able to logout of the application.

Name	Login
ID	AD-01
Actor	Admin
Description	The user want to logout
Pre-condition	The user has logged in
Basic flow	1. Click on the top-right profile button 2. Click "Logout" button
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	The staff has logged in and can use the dashboard

Table 4.3.14: Use case specification: Logout

View available scenarios

User story: As a staff of service provider, you want to see all available scenarios on the system.

Name	View irrigation scenarios
ID	AD-02
Actor	Admin
Description	User want to see all irrigation scenarios
Pre-condition	The user has already logged in
Basic flow	1. Click on "Scenario" tab from side panel 2. Click on "Reload" button to fetch the latest scenarios from Data Core
Alternative Flow	• Iterate through pages if there many pages
Exception Flow	N/A
Post-condition	Staff can view all available scenarios on the system

Table 4.3.15: Use case specification: View available scenarios

Edit a chosen scenarios

User story: As a staff of service provider, you want to modify a scenario.

Name	Edit a irrigation scenario
ID	AD-03
Actor	Admin
Description	User want to modify a irrigation scenario
Pre-condition	The user has already logged in
Basic flow	1. Click on "Scenario" tab from side panel 2. Choose a desired scenario to modify by clicking on "Edit" button 3. Choose field to modify (name, description, plant type category, location,..) 4. Click Save button
Alternative Flow	• 4a. If a specified plant type or location is not only the list, proceed to Location or Plant Type tab to register it first.
Exception Flow	N/A
Post-condition	Staff edit successfully a scenario

Table 4.3.16: Use case specification: Edit a scenario

Set status for a chosen scenarios

User story: As a staff of service provider, you want to set status for a scenario.

Name	Edit a irrigation scenario
ID	AD-04
Actor	Admin
Description	User want to modify a irrigation scenario
Pre-condition	The user has already logged in
Basic flow	1. Click on "Scenario" tab from side panel 2. Choose a desired scenario to modify by clicking on "Edit" button 3. Choose a desired status (NEW, PRODUCTION, STOP) 4. Click Save button
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	Mobile user will able to see new irrigation scenarios if the chosen scenario is on PRODUCTION or STOP, receive notifications about it

Table 4.3.17: Use case specification: Set status for a scenario

Set status for a chosen scenarios

User story: As a staff of service provider, you want to set status for a scenario.

Name	Edit a irrigation scenario
ID	AD-05
Actor	Admin
Description	User want to modify a irrigation scenario
Pre-condition	The user has already logged in
Basic flow	1. Click on "Scenario" tab from side panel 2. Choose a desired scenario to modify by clicking on "Edit" button 3. Choose a desired status (NEW, PRODUCTION, STOP) 4. Click Save button
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	Mobile user will able to see new irrigation scenarios if the chosen scenario is on PRODUCTION or STOP, receive notifications about it

Table 4.3.18: Use case specification: Set status for a scenario

Delete a chosen scenarios

User story: As a staff of service provider, you want to delete a scenario.

Name	Delete a irrigation scenario
ID	AD-06
Actor	Admin
Description	User want to delete a irrigation scenario
Pre-condition	The user has already logged in
Basic flow	1. Click on "Scenario" tab from side panel 2. Choose a desired scenario to delete by clicking on "Delete" button 3. Click "Delete" again to confirm
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	Staff delete successfully a scenario

Table 4.3.19: Use case specification: Delete a scenario

View available plant types

User story: As a staff of service provider, you want to see all available plant types.

Name	View plant types
ID	AD-07
Actor	Admin
Description	User want to see all plant types
Pre-condition	The user has already logged in
Basic flow	1. Click on "Plant Type" tab from side panel
Alternative Flow	Navigate through pages if there are many pages
Exception Flow	N/A
Post-condition	Display all plant types

Table 4.3.20: Use case specification: View all plant types

Create new plant types

User story: As a staff of service provider, you want to create a new plant type.

Name	Create plant types
ID	AD-08
Actor	Admin
Description	User want to create a plant type
Pre-condition	The user has already logged in
Basic flow	1. Click on "Plant Type" tab from side panel 2. Click on "New" button 3. Input name, description and unique code for new plant type 4. Click on "Save" button
Alternative Flow	N/A
Exception Flow	• 3a. If unique code is duplicated with previous plants. Try input a different unique code
Post-condition	Staff create successfully a plant type

Table 4.3.21: Use case specification: Create new plant type

Edit a plant types

User story: As a staff of service provider, you want to edit a new plant type.

Name	Edit a plant types
ID	AD-09
Actor	Admin
Description	User want to edit a plant type
Pre-condition	The user has already logged in
Basic flow	1. Click on "Plant Type" tab from side panel 2. Click on "Edit" button 3. Modify name, description and unique code as desired 4. Click on "Save" button
Alternative Flow	N/A
Exception Flow	• 3a. If unique code is duplicated with previous plants. Try input a different unique code
Post-condition	Staff edit successfully a plant type

Table 4.3.22: Use case specification: Edit a plant type

Delete a chosen plant type

User story: As a staff of service provider, you want to delete a plant type.

Name	Delete a plant type
ID	AD-10
Actor	Admin
Description	User want to delete a plant type
Pre-condition	The user has already logged in
Basic flow	1. Click on "Plant Type" tab from side panel 2. Choose a desired plant type to delete by clicking on "Delete" button 3. Click "Delete" again to confirm
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	Staff delete successfully a plant type

Table 4.3.23: Use case specification: Delete a plant type

View available loccation

User story: As a staff of service provider, you want to see all available locations.

Name	View locations
ID	AD-11
Actor	Admin
Description	User want to see all available locations
Pre-condition	The user has already logged in
Basic flow	1. Click on "Location" tab from side panel
Alternative Flow	Navigate through pages if there are many pages
Exception Flow	N/A
Post-condition	Display list all locations

Table 4.3.24: Use case specification: View all locations

Create a new location

User story: As a staff of service provider, you want to create a new location for your irrigation scenario.

Name	Create a new location
ID	AD-12
Actor	Admin
Description	User want to create a location
Pre-condition	The user has already logged in
Basic flow	1. Click on "Location" tab from side panel 2. Click on "New" button 3. Input name, description and unique code for new location, optional for latitude and longitude 4. Click on "Save" button
Alternative Flow	N/A
Exception Flow	• 3a. If unique code is duplicated with previous locations. Try input a different unique code
Post-condition	Staff create successfully location

Table 4.3.25: Use case specification: Create new location

Edit a location

User story: As a staff of service provider, you want to edit a location.

Name	Edit a location
ID	AD-13
Actor	Admin
Description	User want to edit a location
Pre-condition	The user has already logged in
Basic flow	1. Click on "Location" tab from side panel 2. Click on "Edit" button 3. Modify name, description and unique code, longitude, latitude as desired 4. Click on "Save" button
Alternative Flow	N/A
Exception Flow	• 3a. If unique code is duplicated with previous locations. Try input a different unique code
Post-condition	Staff edit successfully location

Table 4.3.26: Use case specification: Edit a location

Delete a chosen location

User story: As a staff of service provider, you want to delete a location.

Name	Delete a location
ID	AD-14
Actor	Admin
Description	User want to delete a location
Pre-condition	The user has already logged in
Basic flow	1. Click on "Location" tab from side panel 2. Choose a desired location to delete by clicking on "Delete" button 3. Click "Delete" again to confirm
Alternative Flow	N/A
Exception Flow	N/A
Post-condition	Staff delete successfully location

Table 4.3.27: Use case specification: Delete a location

Chapter 5

System Design

5.1 System Architecture

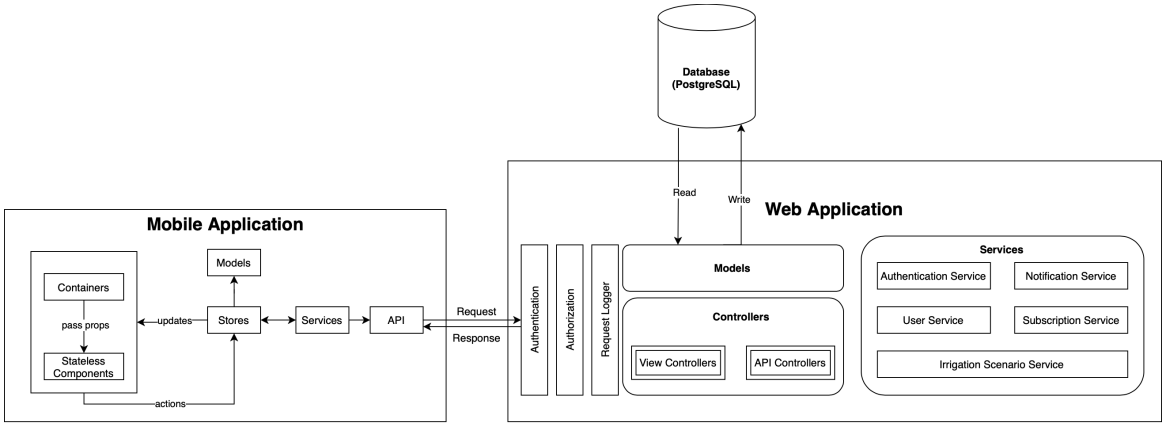


Figure 5.1.1: Final system architecture

System architecture is designed as Figure 5.1.1 including the following components:

5.1.1 Mobile Application

In this section, we will describe how the mobile application manages states using stores based on the following criteria:

- Easy communication between stores
- Root store includes children stores
- Independent communication between stores.

To implement this, we used the Model-View-Controller (MVC) design pattern and layered architecture approach, which breaks the code into different parts. Here's how it works:

- Service layer: communicates with the backend through APIs.
- Stores layer: contains all application states. All service functions are called only in the store. Components execute actions in the store when the state changes.
- Presentational Component: use the store directly by injecting the store or Props from Container Component can be passed in it.
- Container or Presentational Component: invoke store actions and automatic rendering of components will be done by Mobx.

We use services to communicate between the application and backend services. This makes our code more organized and flexible. If we were to call services inside the store, the code would become more complicated as the app scales. Inside a store, we only call the service method and update the store. Service methods are only for communication and do not modify stores. We can only modify observable variables in stores.

5.1.2 Web Application + API Server

Controllers: contains 2 kinds of controllers: API controller and View Controller. Each controller in charge of receiving and performing a specific action based on predefined method and URL path. API controllers serve data in the form of JSON format. Meanwhile, View Controllers render a interface for users to interact with.

Additionally, before entering controller, the request have go through 3 (or more) layers:

- Authentication: authenticate user, decide if given user details is exists in database, whether the declared user is valid.
- Authorization: decide if the authenticated user has permission to conduct action on specified resources.
- Middle-ware: perform an action for each received request from user (logging,..)

Services: perform a high-level abstract tasks that requested from the controller. A service can call to other services as well.

- Authentication service: handle user authentication with given email and password. Serve user 2 kinds of authentication: session-based and token-based. Authorize user with roles.

- User Service: handle create new user, managing users.
- Notification Service: handle cloud messaging and sending notifications to users.
- Irrigation Scenario Service: handle fetching and managing irrigation scenarios.
- Subscription Service: handle managing users subscriptions, extending or renew subscription plan.

Models: a model accurately maps a relationship from a database and its associated constraints, with corresponding names. The models are interconnected through the relationship of objects.

5.2 Database Design

Database is where all the necessary data of the application is stored, ensuring it can be accessed anytime and anywhere. The controllers in the database are mapped and work directly with the Models layer. The database will execute all SQL commands to query, update, create new data, etc..

5.2.1 Entity Relational Diagram

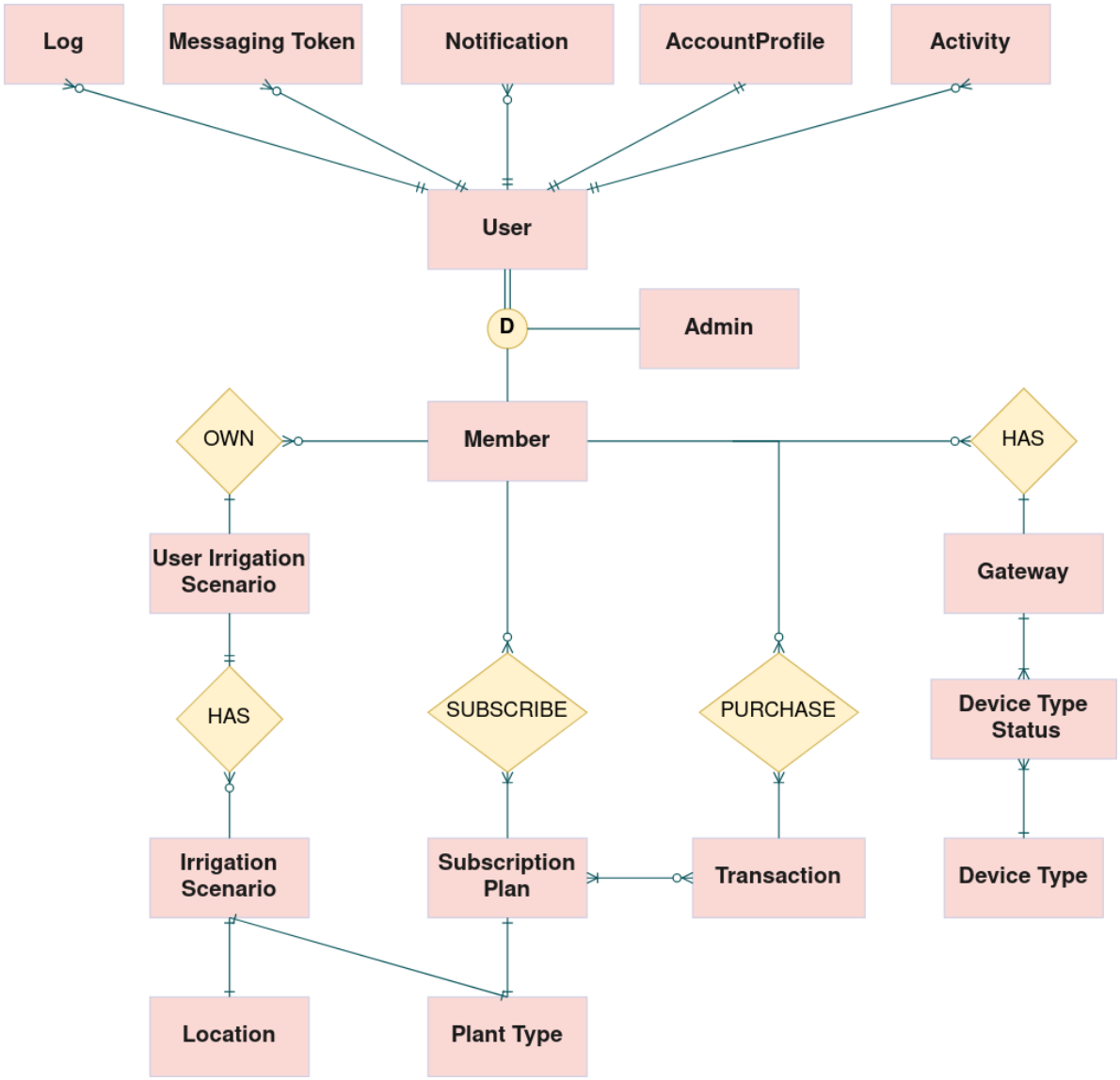


Figure 5.2.1: System ERD design

5.2.2 Relation Database Schema

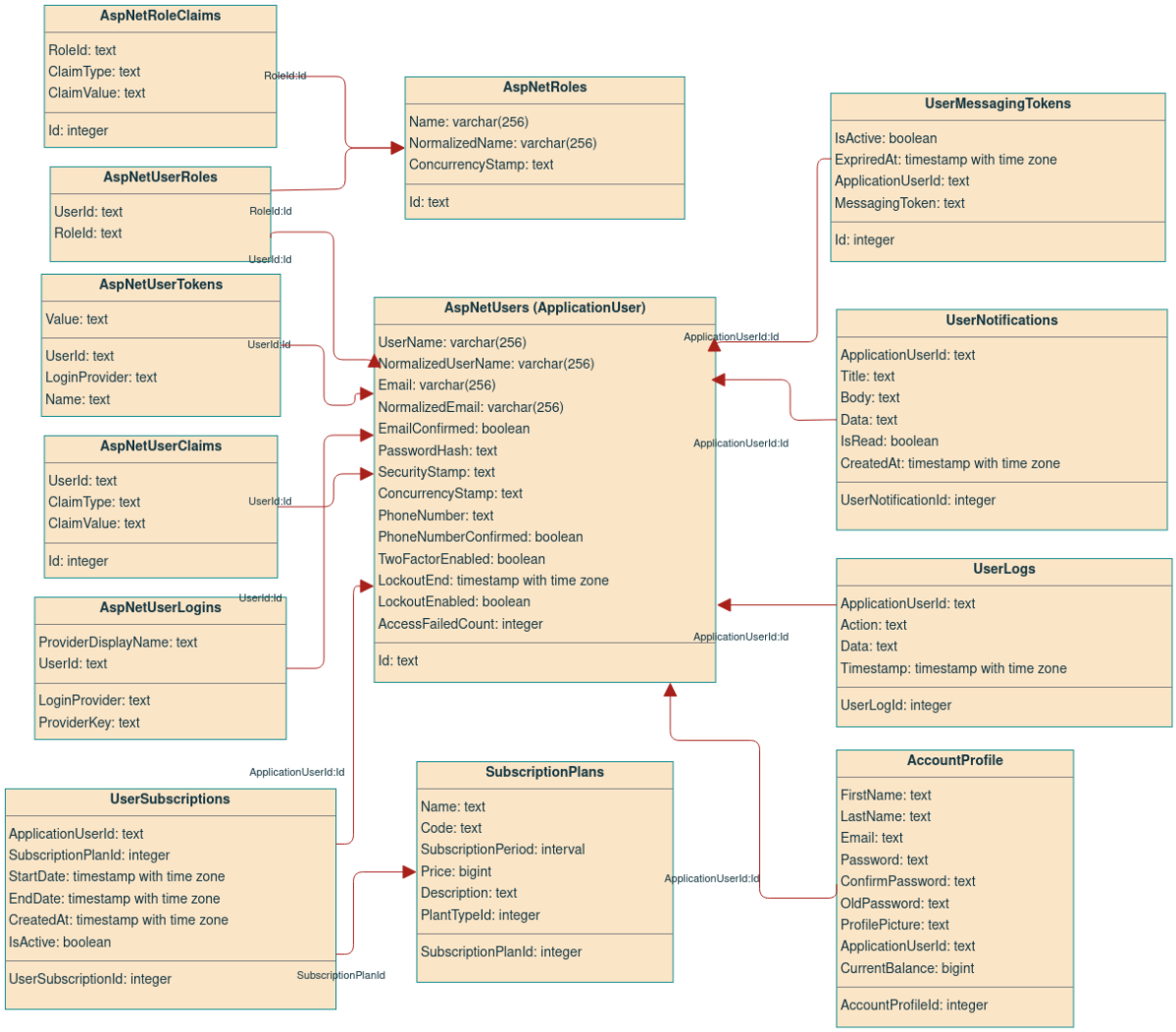


Figure 5.2.2: User-related Features

5.2. Database Design

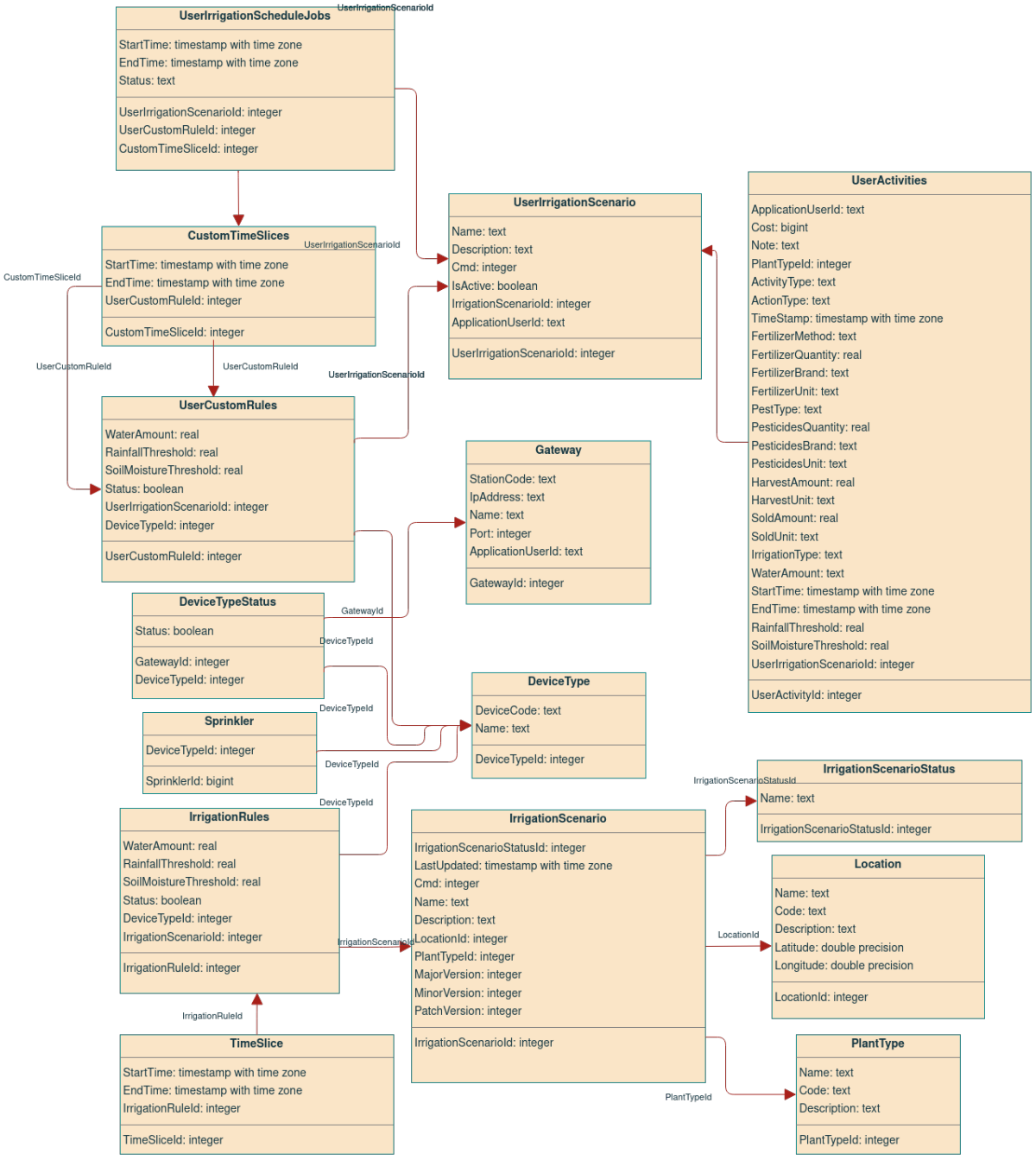


Figure 5.2.3: Irrigation-related Features

5.2.3 Table Descriptions

UserLog

Attribute Name	Variable Type	Description
UserLogId	int	Unique identifier for the user log entry
ApplicationUserId	string?	The ID of the user who performed the action (can be null)
Action	string	The type of action performed by the user
Data	string	The data associated with the user action
Timestamp	DateTime	The date and time when the user action was performed (in UTC)

Table 5.2.1: Attributes of the UserLog class.

UserIrrigationScenario

Attribute Name	Variable Type	Description
UserIrrigationScenarioId	int	Unique identifier for the user irrigation scenario
Name	string	The name of the user irrigation scenario
Description	string	The description of the user irrigation scenario
Cmd	int	The command code associated with the user irrigation scenario
IsActive	bool	A flag indicating whether the user irrigation scenario is active or not
IrrigationScenarioId	int	The ID of the irrigation scenario associated with the user irrigation scenario
ApplicationUserId	string	The ID of the user who owns the user irrigation scenario
Rules	ICollection <User-Custom-Rule>	A collection of user-defined rules

Table 5.2.2: Attributes of the UserIrrigationScenario class

UserNotification

Attribute Name	Variable Type	Description
UserNotificationId	int	Unique identifier for the user notification
ApplicationUserId	string	The ID of the user who the notification is for
Title	string?	The title of the notification (can be null)
Body	string?	The body of the notification (can be null)
Data	string?	The data associated with the notification (can be null)
IsRead	bool	A flag indicating whether the notification has been read or not
CreatedAt	DateTime	The date and time when the notification was created (in UTC)

Table 5.2.3: Attributes of the UserNotification class.

UserSubscription

Attribute Name	Variable Type	Description
UserSubscriptionId	int	Unique identifier for the user subscription
ApplicationUserId	string	The ID of the user who is subscribed
SubscriptionPlanId	int	The ID of the subscription plan
StartDate	DateTime	The date and time when the subscription starts (in UTC)
EndDate	DateTime	The date and time when the subscription ends (in UTC)
CreatedAt	DateTime	The date and time when the subscription was created (in UTC)
IsActive	bool	A flag indicating whether the subscription is active or not

Table 5.2.4: Attributes of the UserSubscription class

UserIrrigationScheduleJob

Attribute Name	Variable Type	Description
UserIrrigationScenarioId	int	The ID of the user's irrigation scenario
UserCustomRuleId	int	The ID of the user's custom rule
CustomTimeSliceId	int	The ID of the custom time slice
StartTime	DateTime	The start time of the irrigation schedule job
EndTime	DateTime	The end time of the irrigation schedule job
Status	string	The status of the irrigation schedule job

Table 5.2.5: Attributes of the UserIrrigationScheduleJob class

UserActivity

Attribute Name	Variable Type	Description
UserActivityId	int	Unique identifier for the user activity entry
ApplicationUserId	string?	The ID of the user who performed the activity (can be null)
Cost	long	The cost associated with the activity
Note	string?	The note associated with the activity
PlantTypeId	int	The ID of the plant type associated with the activity
ActivityType	string	The type of activity performed
ActionType	string?	The type of action performed (if applicable)
TimeStamp	DateTime	The date and time when the activity was performed (in UTC)
FertilizerMethod	string?	The method used for fertilization (if applicable)
FertilizerQuantity	float?	The quantity of fertilizer used (if applicable)
FertilizerBrand	string?	The brand of fertilizer used (if applicable)
FertilizerUnit	string?	The unit of fertilizer used (if applicable)
PestType	string?	The type of pest (if applicable)
PesticidesQuantity	float?	The quantity of pesticide used (if applicable)
PesticidesBrand	string?	The brand of pesticide used (if applicable)
PesticidesUnit	string?	The unit of pesticide used (if applicable)
HarvestAmount	float?	The amount of harvested crop (if applicable)
HarvestUnit	string?	The unit of harvested crop (if applicable)
SoldAmount	float?	The amount of crop sold (if applicable)
SoldUnit	string?	The unit of sold crop (if applicable)
IrrigationType	string?	The type of irrigation used (if applicable)
WaterAmount	string?	The amount of water used (if applicable)
StartTime	DateTime?	The start time of the irrigation activity (if applicable)
EndTime	DateTime?	The end time of the irrigation activity (if applicable)
RainfallThreshold	float?	The rainfall threshold for the irrigation activity (if applicable)
SoilMoistureThreshold	float?	The soil moisture threshold for the irrigation activity (if applicable)
UserIrrigationScenarioId	int?	The ID of the user irrigation scenario associated with the activity (if applicable)

Table 5.2.6: Attributes of the UserActivity class

UserCustomRule

Attribute Name	Variable Type	Description
UserCustomRuleId	int	The ID of the user's custom rule
WaterAmount	float?	The amount of water to be used for irrigation
RainfallThreshold	float?	The threshold of rainfall for irrigation
SoilMoistureThreshold	float?	The threshold of soil moisture for irrigation
Status	bool	The status of the custom rule
TimeSlices	ICollection <CustomTimeSlice>	A collection of custom time slices
UserIrrigationScenarioId	int	The ID of the user's irrigation scenario
DeviceTypeId	int	ID of device type used for irrigation

Table 5.2.7: Attributes of the UserCustomRule class.

CustomTimeSlice

Attribute Name	Variable Type	Description
CustomTimeSliceId	int	The ID of the custom time slice
StartTime	DateTime	The start time of the time slice
EndTime	DateTime	The end time of the time slice
UserCustomRuleId	int	The ID of the user's custom rule that the time slice belongs to

Table 5.2.8: Attributes of the CustomTimeSlice class.

TransactionSubscription

Attribute Name	Variable Type	Description
SubscriptionTransactionId	long	The ID of the subscription transaction
UserSubscriptionId	int	The ID of the user subscription related to the transaction
Amount	long	The amount of the transaction
CreatedAt	DateTime	The date and time the transaction was created

Table 5.2.9: Attributes of the SubscriptionTransaction class.

TimeSlice

Attribute Name	Variable Type	Description
TimeSliceId	int	The ID of the time slice
StartTime	DateTime	The start time of the time slice
EndTime	DateTime	The end time of the time slice

Table 5.2.10: Attributes of the TimeSlice class.

SubscriptionPlan

Attribute Name	Variable Type	Description
SubscriptionPlanId	int	The ID of the subscription plan.
Name	string	The name of the subscription plan.
Code	string	The unique code for each subscription plan.
SubscriptionPeriod	TimeSpan	The duration of the subscription plan.
Price	long	The price of the subscription plan.
Description	string	The description of the subscription plan.
PlantTypeId	int?	The ID of the plant type associated with the subscription plan.

Table 5.2.11: Attributes of the SubscriptionPlan class.

Plant Type

Attribute Name	Variable Type	Description
PlantTypeId	int	The unique ID of the plant type
Name	string	The name of the plant type
Code	string	The unique code of the plant type, allows upper-cases and digits
Description	string	A description of the plant type (optional)

Table 5.2.12: Attributes of the PlantType class.

Location

Attribute Name	Variable Type	Description
LocationId	int	The ID of the location
Name	string	The name of the location
Code	string	The unique code of the location (allow upper-cases and digits)
Description	string	The description of the location
Latitude	double?	The latitude of the location
Longitude	double?	The longitude of the location

Table 5.2.13: Attributes of the Location class

IrrigationScenario

Property	Type	Description
IrrigationScenarioId	int?	ID for the irrigation scenario
IrrigationScenarioStatusId	int?	Status for the irrigation scenario
LastUpdated	DateTime	Last updated of the irrigation scenario
Rules	ICollection <Irriga-tion-Rule>	Collection of irrigation rules
Cmd	int	Command for the irrigation scenario
Name	string	Name of the irrigation scenario
Description	string	Description of the irrigation scenario
LocationId	int	ID for the location
PlantTypeId	int	ID for the plant type
MajorVersion	int	Major version number
MinorVersion	int	Minor version number
PatchVersion	int	Patch version number
Version	string	Version number

Table 5.2.14: Attributes of the IrrigationScenario class

IrrigationScenarioStatus

Attribute Name	Variable Type	Description
IrrigationScenarioStatusId	int	The ID of the irrigation scenario status
Name	string	The name of the irrigation scenario status

Table 5.2.15: Attributes of the IrrigationScenarioStatus class.

IrrigationRule

Attribute Name	Variable Type	Description
IrrigationRuleId	int	The ID of the irrigation rule
WaterAmount	float?	The amount of water to be applied
RainfallThreshold	float?	The rainfall threshold for the rule to be active
SoilMoistureThreshold	float?	The soil moisture threshold for the rule to be active
Status	bool	The status of the irrigation rule
TimeSlices	ICollection <TimeSlice>?	The time slices when the rule is active
DeviceTypeId	int	The ID of device type that associated with the rule

Table 5.2.16: Attributes of the IrrigationRule class.

DeviceType

Attribute Name	Variable Type	Description
DeviceTypeId	int	The ID of the device type
DeviceCode	string	The code of the device type
Name	string	The name of the device type

Table 5.2.17: Attributes of the DeviceType class.

DeviceTypeStatus

Attribute Name	Variable Type	Description
Status	bool	The status of the device type
GatewayId	int	The ID of the gateway associated with the device type status
DeviceTypeId	int	The ID of the device type associated with the device type status

Table 5.2.18: Attributes of the DeviceTypeStatus class.

Gateway

Attribute Name	Variable Type	Description
GatewayId	int	The ID of the gateway
StationCode	string	The station code of the gateway
IpAddress	string	The IP address of the gateway
Name	string	The name of the gateway
Port	int	The port of the gateway
DeviceTypeStatusCollection	ICollection <DeviceTypeStatus>	A collection of device type statuses associated with the gateway
ApplicationUserId	string	The ID of the application user

Table 5.2.19: Attributes of the Gateway class.

AccountProfile

Attribute Name	Variable Type	Description
AccountProfileId	int	The ID of the account profile
FirstName	string	The first name of the account owner
LastName	string	The last name of the account owner
Email	string	The email address of the account owner
Password	string	The password of the account
ConfirmPassword	string	The confirmation password of the account
OldPassword	string	The old password of the account (optional)
ProfilePicture	string	The URL of the profile picture of the account owner
ApplicationUserId	string	The ID of the application user associated with the account (optional)
PhoneNumber	string	The phone number of the account owner (optional)
Address	string	The address of the account owner (optional)
LocationId	int	The ID of the location associated with the account (optional)
CurrentBalance	long	The current balance of the account
UserSubscriptions	ICollection <UserSubscription>	A collection of user subscriptions associated with the account

Table 5.2.20: Attributes of the AccountProfile class.

AspNetRoleClaims

Attribute Name	Variable Type	Description
RoleId	text	The ID of the role associated with the claim
ClaimType	text	The type of the claim
ClaimValue	text	The value of the claim
Id	integer	The ID of the role claim

Table 5.2.21: Attributes of the AspNetRoleClaims class.

AspNetRoles

Attribute Name	Variable Type	Description
Name	varchar(256)	The name of the role
NormalizedName	varchar(256)	The normalized name of the role
ConcurrencyStamp	text	The concurrency stamp of the role
Id	text	The ID of the role

Table 5.2.22: Attributes of the AspNetRoles class.

AspNetUserClaims

Attribute Name	Variable Type	Description
UserId	text	The ID of the user associated with the claim
ClaimType	text	The type of the claim
ClaimValue	text	The value of the claim
Id	integer	The ID of the user claim

Table 5.2.23: Attributes of the AspNetUserClaims class.

AspNetUserLogins

Attribute Name	Variable Type	Description
ProviderDisplayName	text	The display name of the provider
UserId	text	The ID of the user associated with the login
LoginProvider	text	The login provider name
ProviderKey	text	The provider key

Table 5.2.24: Attributes of the AspNetUserLogins class.

ApplicationUser

Attribute Name	Variable Type	Description
UserName	varchar(256)	The user name
NormalizedUserName	varchar(256)	The normalized user name
Email	varchar(256)	The user email
NormalizedEmail	varchar(256)	The normalized user email
EmailConfirmed	boolean	Indicates whether the email is confirmed or not
PasswordHash	text	The hashed password
SecurityStamp	text	The security stamp
ConcurrencyStamp	text	The concurrency stamp
PhoneNumber	text	The user phone number
PhoneNumberConfirmed	boolean	Indicates whether the phone number is confirmed or not
TwoFactorEnabled	boolean	Indicates whether two-factor authentication is enabled or not
LockoutEnd	timestamp with time zone	The lockout end date
LockoutEnabled	boolean	Indicates whether lockout is enabled or not
AccessFailedCount	integer	The number of access failed attempts
Id	text	The ID of the user

Table 5.2.25: Attributes of the ApplicationUser class

AspNetUserRoles

Attribute Name	Variable Type	Description
UserId	text	The ID of the user
RoleId	text	The ID of the role

Table 5.2.26: Attributes of the AspNetUserRoles class.

AspNetUserTokens

Attribute Name	Variable Type	Description
Value	text	The token value
UserId	text	The ID of the user
LoginProvider	text	The login provider
Name	text	The token name

Table 5.2.27: Attributes of the AspNetUserTokens class.

Chapter 6

System Implementation

6.1 Technology Stack

6.1.1 Source Code Management

Since we are working on a software and work in a group, we need a tool that can help us in tracking the history of changes and enable collaborative work. Any of us should be able to make changes to the project, and any earlier version of the project can be recovered at any time.

Version Control Systems (VCS) bring a unified and consistent view of a project, surfacing work that is already in progress. With distributed VCS, developers don't need a constant connection to a central repository, and still has a full copy of the project and project history. Git is the most popular tool in this field, with significant benefits for individuals, teams and businesses:

- Git lets developers see the entire timeline of changes, decisions and progression of any project in one place.
- Collaboration can happen at any time while maintaining source code integrity. Using branches, developers can safely propose changes to production code.
- Communication barriers between individuals and teams can be broken down, and the developers can focus more on doing their work.

For our common Git server, we have decided to use GitHub. GitHub hosts Git repositories and provides tools to ship better code.

(a) Docker

(b) Compose

Figure 6.1.1: Git and GitHub

6.1.2 Mobile Application

React Native Framework

React Native is a framework developed by the famous technology company Facebook to solve the performance problem of Hybrid and the cost problem of having to write multiple native languages for each mobile platform.

We can build a Native application, and we can also build the application in a multi-platform way, not a "mobile web app", not an "HTML5 app", and not a "hybrid app" or just build on iOS or Android but we can build and run both ecosystems.

Another great point is reducing the recompile cost of Native by using Hot-Loading, which means that we don't need to build the application from scratch, so editing happens quickly. This helps programmers see their edits quickly and visually, without spending too much time building and running the application.

And the next amazing feature of React Native is that we only need to use JS to develop a complete mobile application, while solving the problems that Native App encounters as we mentioned above. And then there is also a combination with native code like Swift, Java, etc...

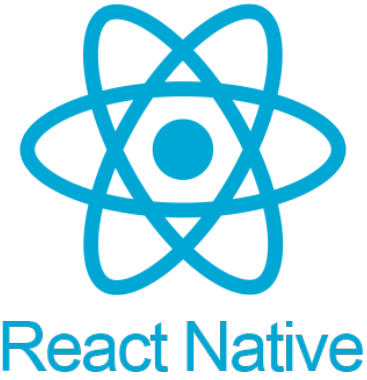


Figure 6.1.2: React Native for mobile development

Advantages of React Native:

- Easy to use: React is an open source JavaScript GUI library that focuses on a specific task, efficiently completing the UI task.
- Supports Reusable Component: React allows you to reuse components that have been developed into other applications with the same function. The ability to reuse components is a distinct advantage for programmers.
- Writing components is easy because it uses JSX
- Relatively stable performance.
- Strong development community
- Save money
- Build for multiple different operating systems with the least native code.

Disadvantages of React Native:

- Does not have features like calling APIs, managing routes, managing state, ... need to use external libraries - developed by Facebook or the community like Redux, React Router, Redux Saga ...
- Still requires native code.
- Performance will be lower than purely native code apps.
- Low security due to relying on JS.
- Memory management
- Customizability is not really good for some modules.

MobX

MobX-State-Tree (MST) is a functional reactive state library that provides structure and tools for state management. MST is useful in both large and small applications and offers better performance and less boilerplate code than Redux. It is one of the most popular Redux alternatives and is used by companies worldwide, including Netflix, Grow, IBM, DAZN, Baidu, and more. MST supports a modern state management system and has zero dependencies other than MobX itself.

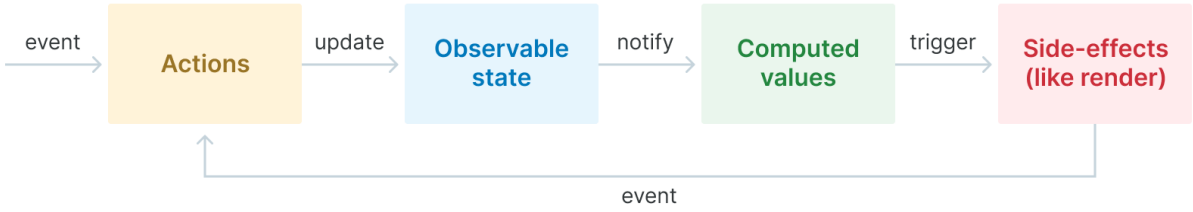


Figure 6.1.3: MobX Flow

MST offers centralized stores for data, allowing for easy and protected mutation of mutable data within "actions." With runtime type checking, MST prevents accidental assignment of the wrong data type to a property. TypeScript can infer static types from runtime types automatically. Every data update is traced, and snapshots of the state can be generated at any time. MST has built-in support for references, allowing for normalization of data across an entire application. Side effects can be managed via async flows, which are long-running actions. Time-travel debugging and logging of state changes over time are possible using snapshots. MST has a robust community and a large, active core team.

In summary, MST is a state container system built on MobX and offers centralized stores, runtime type checking, and support for references. It is a popular alternative to Redux, used worldwide by companies in various industries. MST supports a modern state management system and offers time-travel debugging and logging of state changes over time.

6.1.3 Web Application

.NET Core

The center and fog nodes will spend most of its life listening for requests, doing most of the data processing and storage, and giving responses, in other words, it requires a web application. To help with the workload on this project, we want to implement a web application using a well-known web framework. After some research, we have our sight on the Microsoft .NET ecosystem.

Figure 6.1.4: Microsoft .NET

.NET Framework was originally released as a closed-source Windows-only solution in 2016, but later changed its name to .NET together with a lot of changes to its policy: it is now open-source, cross-platform and has been receiving a lot of support and contributions from the community. The recent .NET 7 saw a lot of performance upgrades, as well as supports to create a lightweight executable to be used in the microservice architecture.

As previously mentioned, .NET itself is an ecosystem, consisting of multiple component that will benefit our project. At first glance, the .NET ecosystem comes with a subset of tools that will benefit our project:

- ASP.NET Core: Acronym for Active Server Pages, this is the web framework that we will work with.
- Entity Framework (EF) Core: Object-Relational Mapping (ORM) framework to handle interaction with the database.
- Dapper: A lightweight alternative to EF Core.

Empowering the .NET ecosystem is the Common Intermediate Language (CIL), which is an intermediate language binary instruction set, meaning higher-level languages can translates into CIL instead of platform- or processor-specific binary code and .NET will do the heavy lifting. The CIL supports both Object Oriented Programming (OOP) and Functional Programming (FP), which translates to the F# programming language and the LINQ library in the C# programming language, giving us a lot of versatility where needed.

AdminLTE Template

AdminLTE is an open-source web-based dashboard built on top of Bootstrap 3, which is designed to be a responsive and user-friendly interface for managing and monitoring web applications. It provides a clean and intuitive interface for managing various aspects of an application

such as user management, form controls, data tables, and charts. AdminLTE has a modular architecture, which means that different modules can be added or removed depending on the needs of the application. With its extensive collection of UI components, AdminLTE is an excellent choice for building professional-looking admin panels or back-office dashboards for web applications. It is also fully customizable, allowing developers to tailor the look and feel of the interface to match their specific needs.

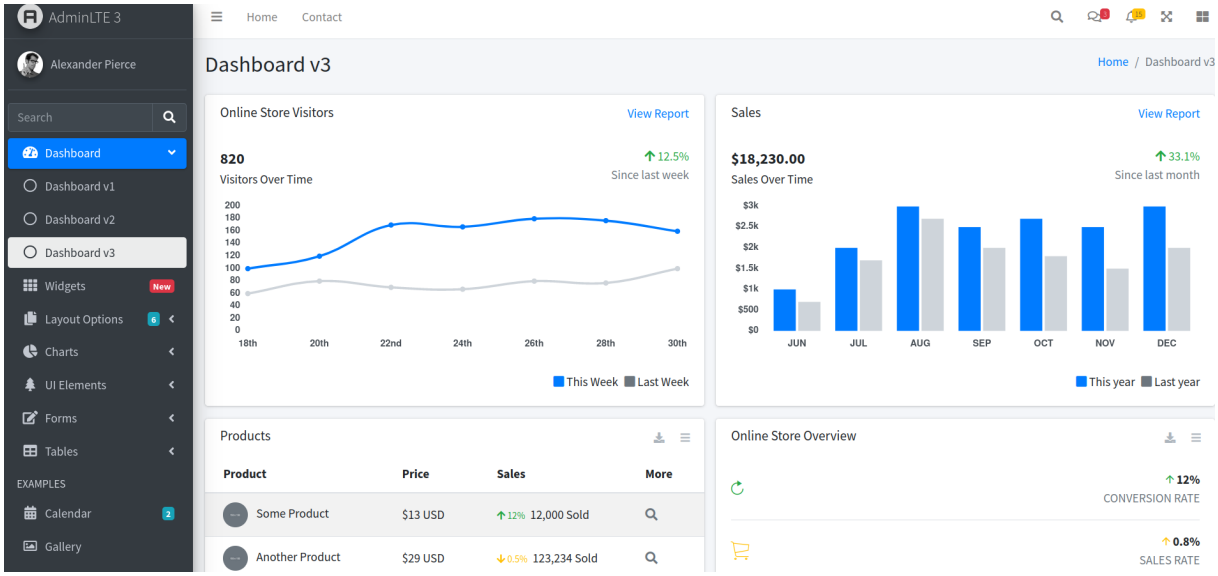


Figure 6.1.5: Dashboard template from AdminLTE

6.1.4 Database

PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance. It has been established as a major player in the Open Source database world and is challenging big players such as Oracle, Sybase, and IBM. PostgreSQL is professionally maintained and developed software, capable of running complex, data-driven applications. As such, the advantages of PostgreSQL are huge.

- Data types: Postgres offers a wider variety of data types than MySQL. If your application deals with any of the unique data types it has available, or unstructured data, PostgreSQL may be a better pick. If you're using only basic character and numeric data types, both databases will suit you.

- Indexes: Databases use indexes to speed up queries. With multiple indexing options to pick from, you can fine-tune your database performance as your data grows to get faster

query responses from your database and an improved user experience for your application users.

- Security: Both databases support user and group management and granting SQL privileges to roles. PostgreSQL supports IP-based client filtering and authentication using PAM and Kerberos, while MySQL supports PAM, native windows services, and LDAP for user authentication. In terms of security, the two databases have comparable options.

6.1.5 Infrastructure

We are aware that managing the runtime environment and packaging software is a pain, hence we want to package our software in such way that it requires as little setup as possible. Additionally, its runtime environment should be well-defined so that the behavior of our software is more predictable and easier to reproduce if needed. We took some time looking for a solution that suits our needs, and the Linux container hits the sweet spot.

A lot of people think that containers are virtual machines or some sort of magic, but they really are just processes running on the Linux kernel.

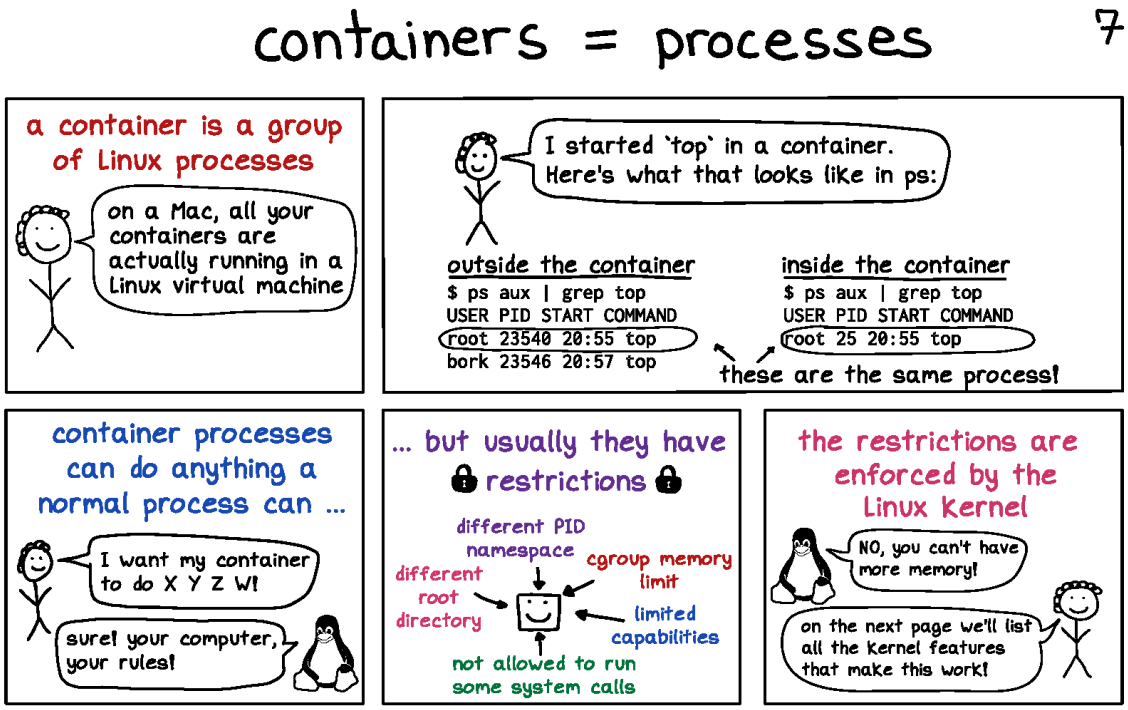
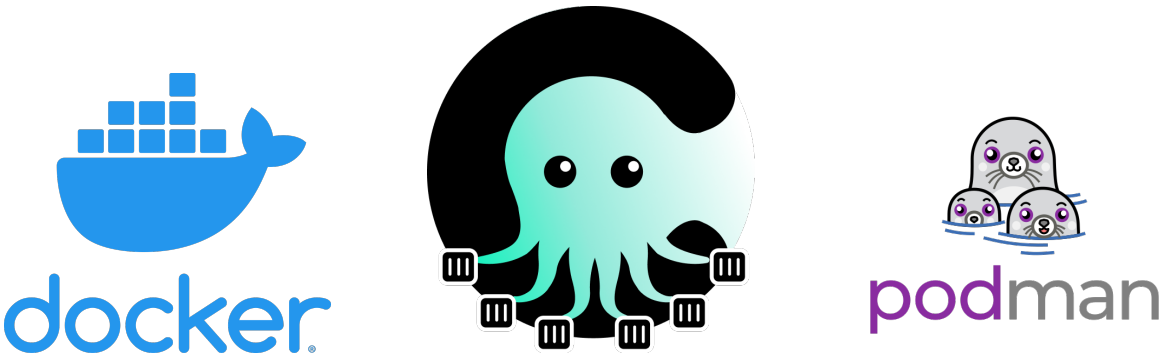


Figure 6.1.6: How containers work (Wizard Zines)

Debuted in 2013, Docker was the first open-sourced container engine and runtime, and later became the standard, just like how we say Google to mention a search engine. Similarly, in 2014, Google released Kubernetes (K8s), an open-source container orchestration system to support microservice systems, and it also quickly became a standard. As there were no other standards, K8s and Docker quickly embraced each other and became cohesive. However, people quickly saw that Docker has a lot of limits and constraints, such as the Docker engine runs in the server-client model, unnecessarily giving the daemon root access. A lot of other container implementations have been released since, and they all want to be compatible with K8s. The engineers at Google saw that there was a need for a standard for K8s API, thus they dropped[1] support for Dockershim[2] and created their own standards.

To ensure that our software is backed with modern standards, we will select a container runtime compatible with the Kubernetes's Container Runtime Interface (CRI).



(a) Docker (b) Compose (c) Podman

Figure 6.1.7: Tools for the infrastructure

There are multiple implementations, and we have decided to use the popular ones:

- Docker: A containerization platform that uses a daemon to manage containers. It is designed to be used with a single daemon running on a host machine. Docker is widely used and has a large community of users and contributors.
- Compose: A tool helping to define and run multi-container applications with a single command. We will mainly use Compose during development.
- Podman: A daemonless, open source, Linux native tool designed to make it easy to find, run, build, share and deploy applications. Podman is similar to Docker, except that the images it creates follow the Open Containers Initiative (OCI) standards, providing better

[1] Kubernetes dropped support for Dockershim in v1.24
[2] A Docker-specific container runtime interface to allow interaction between K8s and a container

integrations if a container orchestration tool like Kubernetes is used. Podman also provides a feature named Pod, which can be used to run multiple related containers like but provides more controls.

6.1.6 Continuous Integration and Delivery

Our project implements a lot of new ideas, and we break things all the time. Additionally, our project does need a deployment for other systems — e.g. mobile application — to interact with. This means we need to embrace a solution where our software development cycle can be automated, especially and the building and deployment stages.

Continuous integration and continuous delivery (CI/CD) is methodology about frequently delivering apps to customers by introducing automation into the stages of app development. CI/CD introduces ongoing automation and continuous monitoring throughout the lifecyle of software, from integration and testing phases to delivery and deployment. Together, these connected practices are often referred to as *CI/CD pipeline* and are supported by development and operations teams working together.

There are multiple tools and platforms for CI/CD, and our tool of choice is Jenkins. Jenkins is an open-source CI/CD server that helps automate the parts of software development related to building, testing, and deploying. It is a server-based system that runs in servlet containers and can be self-hosted. The base Jenkins platform only contains the bare minimal to run the software, further features can be enabled through the use of plugins.

Figure 6.1.8: Jenkins

6.1.7 Cloud Messaging

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably send messages at no cost. Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4000 bytes to a client app.

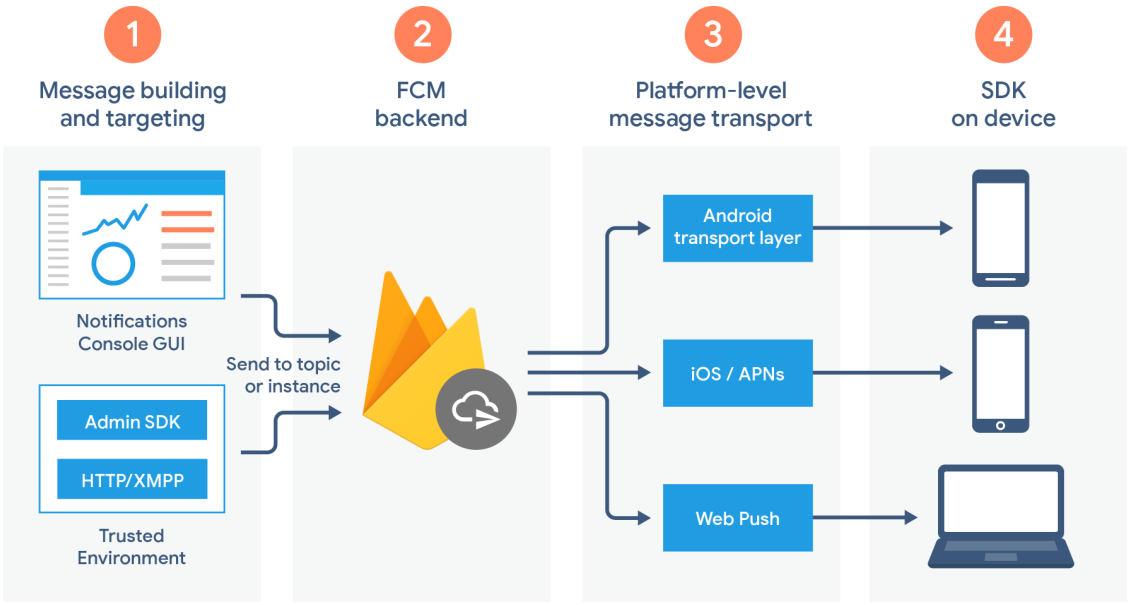


Figure 6.1.9: Firebase Cloud Messaging diagram

Key features:

- Send notification messages or data messages: Send notification messages that are displayed to your user. Or send data messages and determine completely what happens in your application code.
- Versatile message targeting: Distribute messages to your client app in any of 3 ways—to single devices, to groups of devices, or to devices subscribed to topics.
- Send messages from client apps: Send acknowledgments, chats, and other messages from devices back to your server over FCM's reliable and battery-efficient connection channel.

An FCM implementation includes two main components for sending and receiving:

- A trusted environment such as Cloud Functions for Firebase or an app server on which to build, target, and send messages.

- An Apple, Android, or web (JavaScript) client app that receives messages via the corresponding platform-specific transport service.

Usage

In this project, we will use this application to send notifications to users, including new update irrigation scenario notification, seasonal event, weather warning (heavy rain, thunder, northest wind, etc).

6.1.8 Google Analytics

Google Analytics is an app measurement solution, available at no charge, that provides insight on app usage and user engagement. Analytics reports help us to understand clearly how your users behave, which enables us to make informed decisions regarding app marketing and performance optimizations. The SDK automatically captures a number of events and user properties and also allows us to define our own custom events to measure the things that uniquely matter to the product. Once the data is captured, it's available in a dashboard through the Firebase console. This dashboard provides detailed insights about your data — from summary data such as active users and demographics, to more detailed data such as identifying your most purchased items.

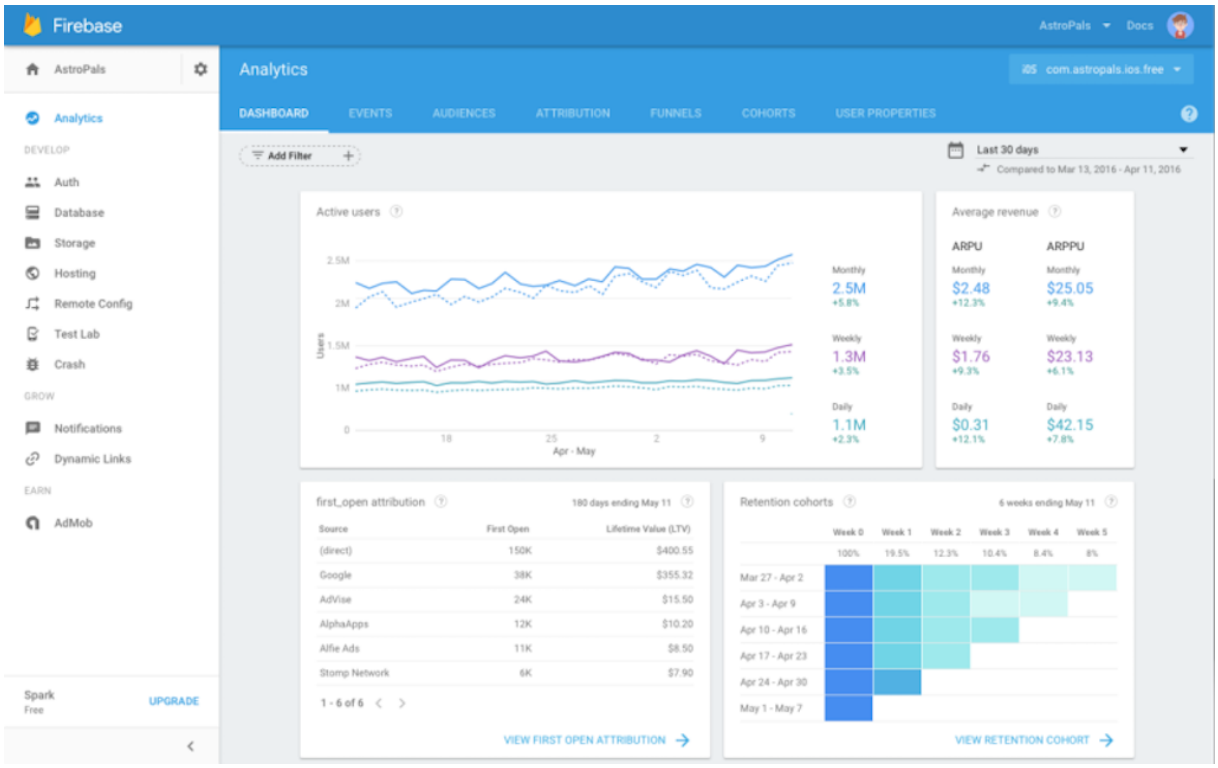


Figure 6.1.10: Google Analytics dashboard enables us to gain insight from users

Analytics also integrates with a number of other Firebase features. For example, it automatically logs events that correspond to notification messages sent via the Notifications composer and provides reporting on the impact of each campaign.

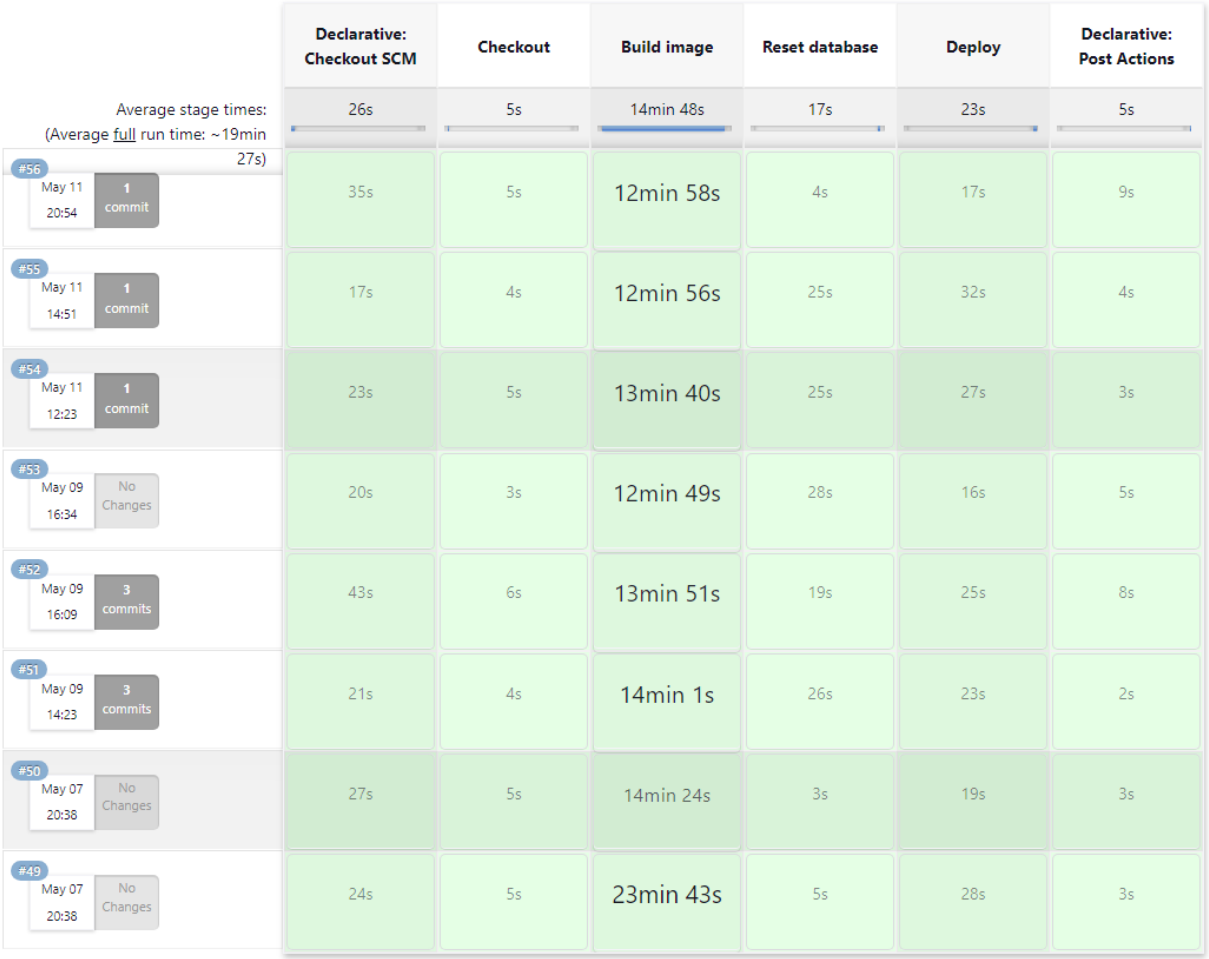
Usage

In this project, we will use this application to log specific user behaviors. Including:

- User's active time
- Manual control devices action
- Subscribe irrigation scenario action
- Update new irrigation scenario action
- Modify irrigation scenario action
- Failed error codes

6.2 Deployment

We provide the ability to run our software using containers, meaning that our server application can be run easily on a lot of platforms, from on premise, with infrastructure-as-a-service (IaaS), to platform-as-a-service (PaaS). This provides a lot of possibilities for mix and match, and due to the encapsulated runtime environment of containers, we can rest assure that their behavior will be consistent on all kinds of deployment.



	Declarative: Checkout SCM	Checkout	Build image	Reset database	Deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~19min 27s)	26s	5s	14min 48s	17s	23s	5s
#56 May 11 20:54 — 1 commit	35s	5s	12min 58s	4s	17s	9s
#55 May 11 14:51 — 1 commit	17s	4s	12min 56s	25s	32s	4s
#54 May 11 12:23 — 1 commit	23s	5s	13min 40s	25s	27s	3s
#53 May 09 16:34 — No Changes	20s	3s	12min 49s	28s	16s	5s
#52 May 09 16:09 — 3 commits	43s	6s	13min 51s	19s	25s	8s
#51 May 09 14:23 — 3 commits	21s	4s	14min 1s	26s	23s	2s
#50 May 07 20:38 — No Changes	27s	5s	14min 24s	3s	19s	3s
#49 May 07 20:38 — No Changes	24s	5s	23min 43s	5s	28s	3s

Figure 6.2.1: Some of our latest builds

Currently, we build our software and run locally using Podman. We choose Podman because it can be invoked without superuser permissions by default. We believe limiting the permissions of a user is important to the security of a system, and we would love to embrace this idea early on.

6.2. Deployment

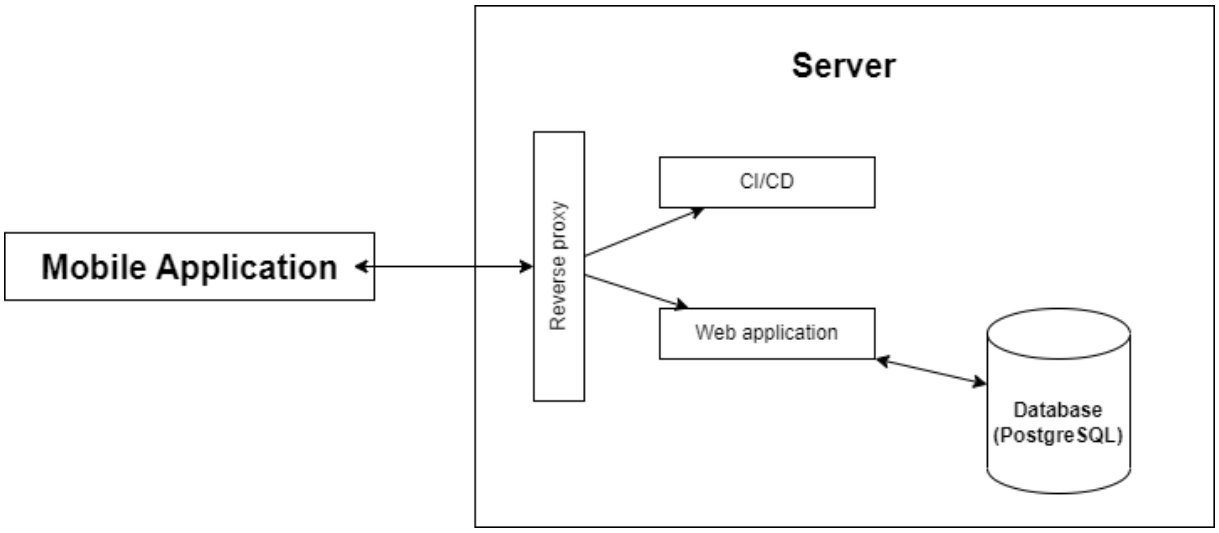


Figure 6.2.2: Deployment of the system

Pod is another feature that not a lot of people have heard of, but can allow for great degree of control over our deployment as compared to Compose. The term is borrowed from Pods of Kubernetes, where Pods are the smallest deployable units. A pod is a group of one or more containers with shared storage and network resources, and its content are always co-located and co-scheduled and run in a shared context. The shared context is a set of Linux namespaces, cgroups and potentially other facets of isolation (Figure 6.1.6). In addition, we provide the option to run our software via Compose for quick and easy inspections.

```
● ❯ podman container ls --filter pod=capstone
CONTAINER ID  IMAGE                                COMMAND               CREATED         STATUS         PORTS                                                 NAMES
c666207b8273  localhost/podman-pause:4.5.0-1681486872                    6 weeks ago     Up 6 weeks     0.0.0.0:443->443/tcp, 0.0.0.0:11232->11232/tcp  6c2f117eaf70-infra
722ff0b707cf  docker.io/library/haproxy:latest     haproxy -f /usr/l...  6 weeks ago     Up 6 weeks     0.0.0.0:443->443/tcp, 0.0.0.0:11232->11232/tcp  haproxy
4133fedfb3b7  docker.io/library/postgres:15        postgres              14 hours ago    Up 14 hours    0.0.0.0:443->443/tcp, 0.0.0.0:11232->11232/tcp  fog-db
21bd9c842a98  localhost/fog-mgmt:latest                                  14 hours ago    Up 14 hours    0.0.0.0:443->443/tcp, 0.0.0.0:11232->11232/tcp  fog-mgmt
289412275b1a  localhost/cicd-service:lts                                 13 minutes ago  Up 13 minutes  0.0.0.0:443->443/tcp, 0.0.0.0:11232->11232/tcp  jenkins
```

Figure 6.2.3: Containers running on our server

The server we use to run our deployment is completely owned and managed by us, but our software will just run anywhere thanks to the power of containers. Different variations of our deployment have been made, like running our database on https://railway.app and running the web application with Azure Container Instances, and all we needed to change is an environment variable. To demonstrate this, we have extracted our deployment stage from the CI/CD pipeline. The highlighted lines show how we can pass environment variables into the container to change its behavior.

```
55          stage('Deploy') {
56              steps {
```

```
57                  sh 'systemctl --user stop
                    capstone-${GIT_REPO}.service'
58                  sh 'podman run -d --replace --pod capstone --name
                    ${GIT_REPO} \
59                          -e
CUSTOMCONNSTR_DefaultConnection="Username=postgres; Password=postgres;
Server=localhost; Port=11232; Database=${GIT_REPO}" \
60                          -e ASPNETCORE_URLS="http://localhost:22232" \
61                          localhost/${GIT_REPO}:latest'
62              }
63              post {
```

There are several takeaways that we want to take note here:

1. We use reverse proxy because our server hosts multiple web applications. Our reverse proxy of choice is HAProxy.

2. We have a *Reset database* stage due to us constantly changing and improving our database schema. This also means we lack a backup and recovery plan.

3. Our integration pipeline implements continuous integration and continuous deployment without continuous delivery. This is due to our support being available only for the latest version. However, we do properly tag each iteration of our software, we just don't push the images out of our server.

```
❯ podman image ls
REPOSITORY                  TAG                                         IMAGE ID      CREATED       SIZE
localhost/fog-mgmt          latest                                      0a8cbf6b8dd9  20 hours ago  464 MB
localhost/fog-mgmt          e077d79df027d7b597675447b98b229309510970    0a8cbf6b8dd9  20 hours ago  464 MB
localhost/fog-mgmt          ea4c13ba345927711122b7b43aaea15fd666bed3    0a8cbf6b8dd9  20 hours ago  464 MB
localhost/fog-mgmt          e6c9aa9cf14697634346f6a631c0764177e06e8a    884c6a6cc633  23 hours ago  496 MB
localhost/fog-mgmt          584af82a857eaa4c5486c2dd91357f0de8dfc07c    d54833cd23a0  2 days ago    496 MB
localhost/fog-mgmt          b420ab7f39ec9dbdc1a9e85d796200f290729f2e    41115cd8ccfa  2 days ago    529 MB
localhost/fog-mgmt          46721f8352892608f40a9d3eb7b144c509e77bfa    359fb0d2f99f  4 days ago    529 MB
localhost/fog-mgmt          bbb4bd4c3c1b714cc2b91015844797f55fd16046    40fbe397bc3c  6 days ago    497 MB
localhost/fog-mgmt          d8f46af9f294b3748047da764d1fa4bb56950271    cea6b8c982ba  6 days ago    497 MB
localhost/fog-mgmt          0387d7745b9b9edf6b6346bb27c8630608394119    03367505b03a  7 days ago    497 MB
localhost/fog-mgmt          4257e73ab09942e2a5aa11a39e1bff6f2bb4c6ea    8b68df96e2f0  7 days ago    495 MB
```

Figure 6.2.4: Some of our created images

Chapter 7

Testing and Evaluation

7.1 Implementation Result

After completing the graduation project using the technologies and knowledge presented in previous sections, the group has completed a relatively complete **The solution for managing irrigation scenarios** in the form of a mobile application which is named **GardenCare**, and a website application is also available.

In the following section, our group will present the interface of the implemented application, as well as solutions to problems encountered during implementation.

7.1.1 Mobile Application

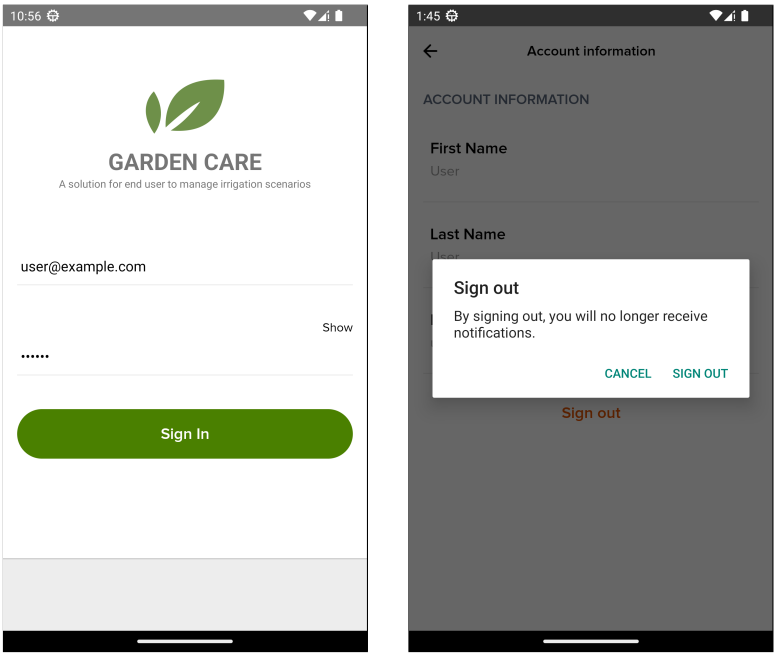
Authentication

After being provided with an account by the service provider, the farmer can log in to the application on the login page. Upon successful login, the server will provide this account with a token to perform API requests throughout the application usage process.

The token is stored in Local Storage, with the purpose of retaining the information when the user reloads the app and all the application's state is deleted. At this point, the saved token will be used to retrieve information without requiring the user to log in again.

After logging out of the application, this token will also be removed from Local Storage as well as the notification token to prevent the case where notifications are sent to this device but using a different account.

7.1. Implementation Result

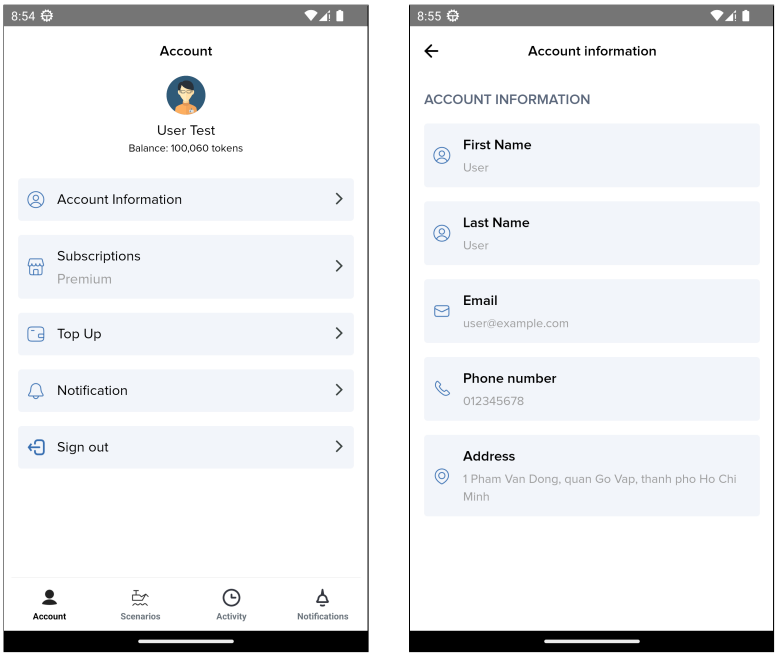


(a) Sign In (b) Sign Out

Figure 7.1.1: Authentication

Account Management

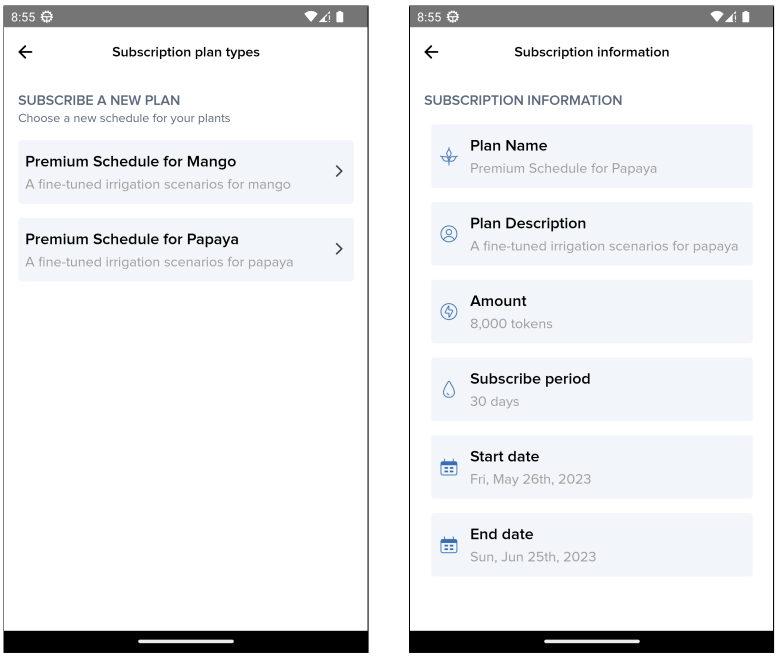
The following screens display some basic information about the user including their first name, last name, email address, and the subscription plan that the user has registered for.



(a) Account screen (b) Account information

Figure 7.1.2: Account Management

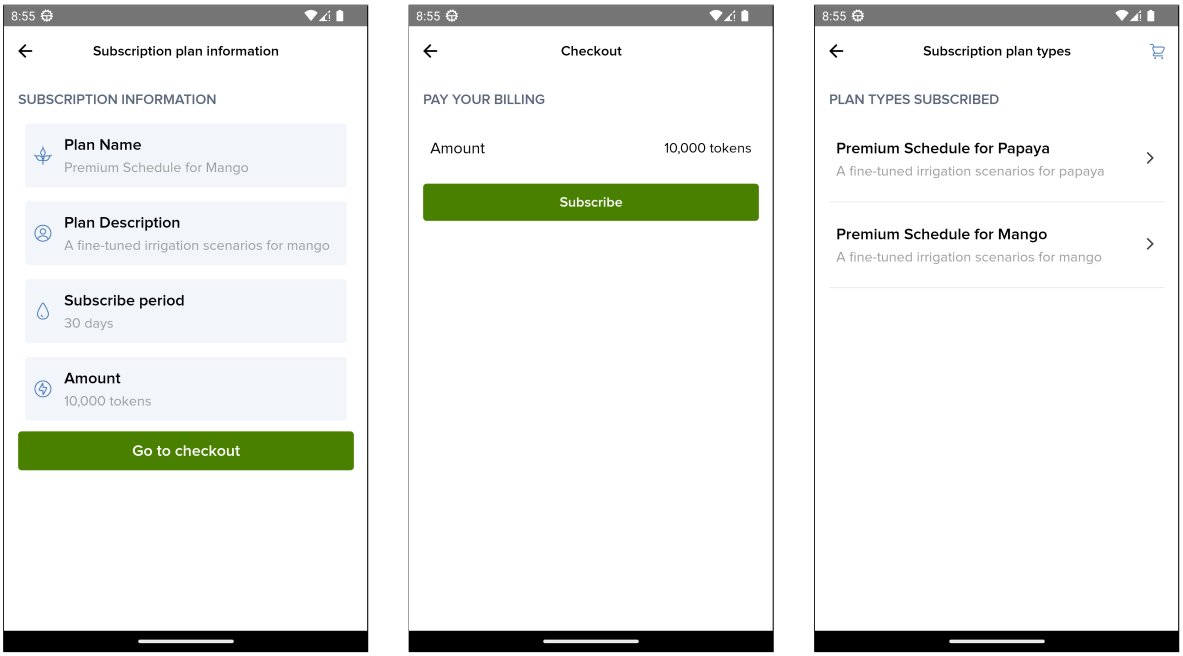
Subscription Management



(a) List subscription for plant types (b) View detail of a subscription for a plant type

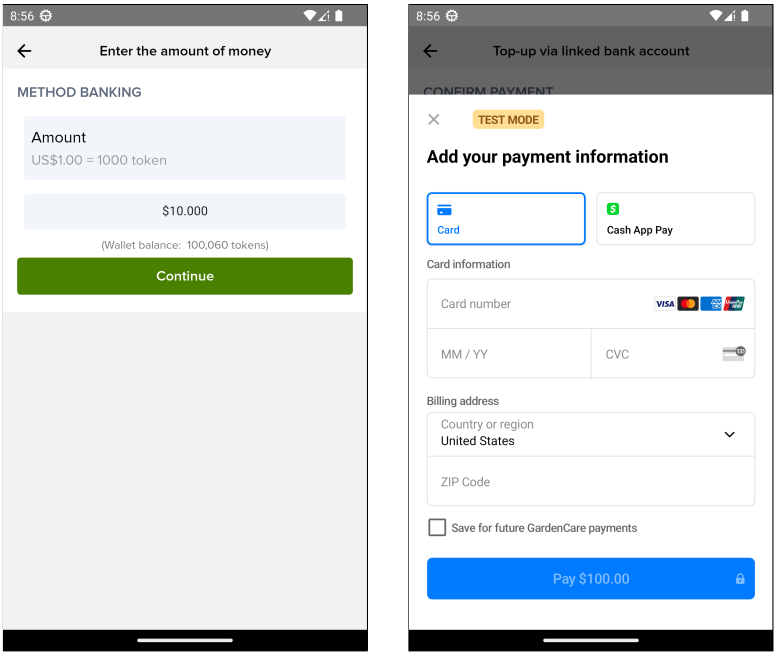
Figure 7.1.3: Subscription Management

7.1. Implementation Result



(a) Scenario template details (b) Checkout scenario template (c) List plant types subscribed

Figure 7.1.4: Subscription Management



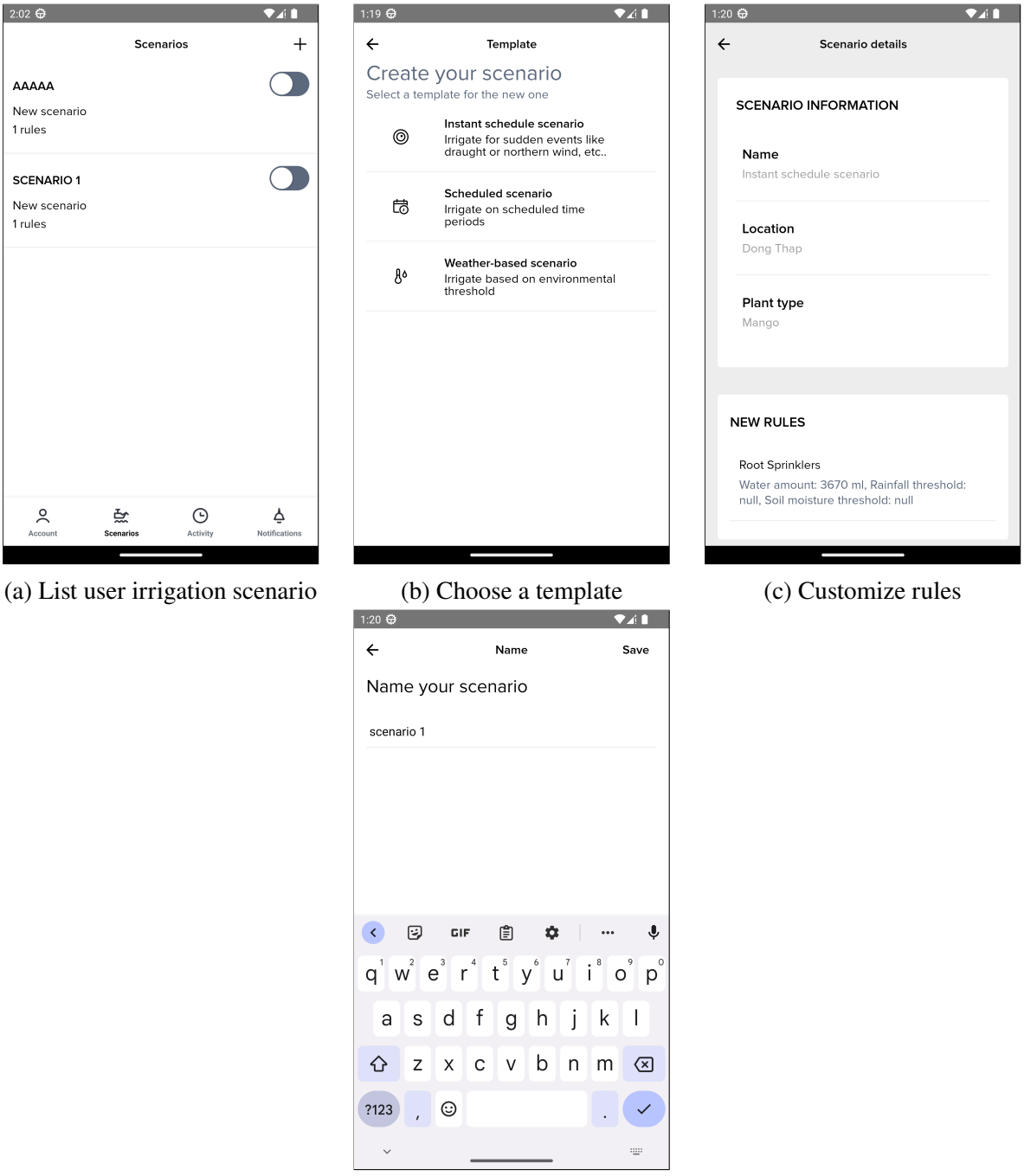
(a) Input money for top up (b) Payment with Stripe

Figure 7.1.5: Subscription Management

Irrigation Scenario Management

Below are the feature group related to creating an irrigation scenario based on a template scenario from the service provider, including screens for displaying the list of user-created scenarios, a screen for selecting a template scenario, a screen for customizing scenario details by adding rules, and a screen for naming the scenario.

7.1. Implementation Result



(a) List user irrigation scenario

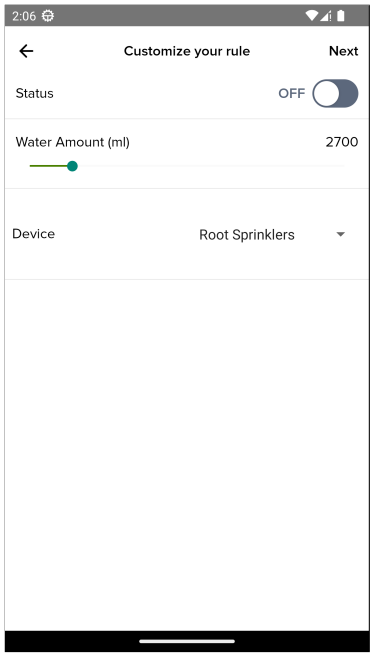
(b) Choose a template

(c) Customize rules

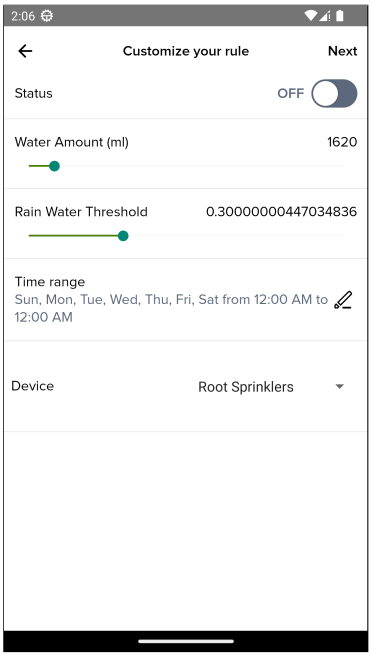
(d) Set name for user irrigation scenario

Figure 7.1.6: Irrigation Scenario Management

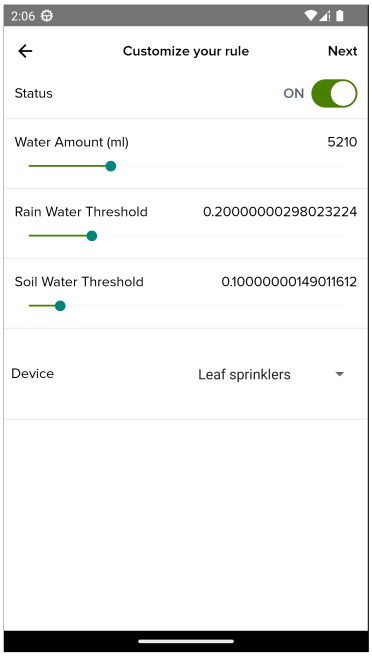
Below are the groups of template scenarios, corresponding to each type of template scenario will have different adjustments about water amount, irrigation time, irrigation status, rainwater threshold, soil threshold and irrigation device type.



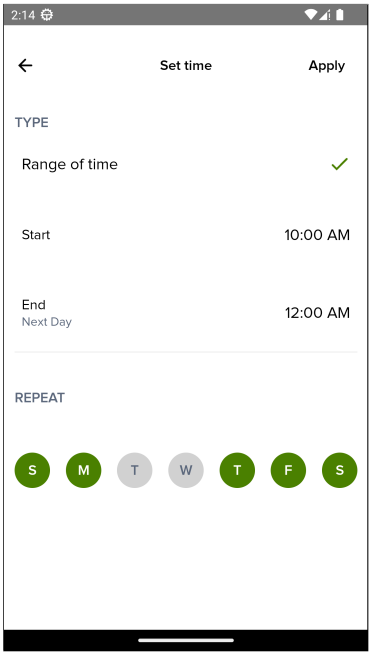
(a) Irrigation scenario template for constant irrigation



(b) Irrigation scenario template for scheduled irrigation



(c) Irrigation scenario template for weather-based irrigation

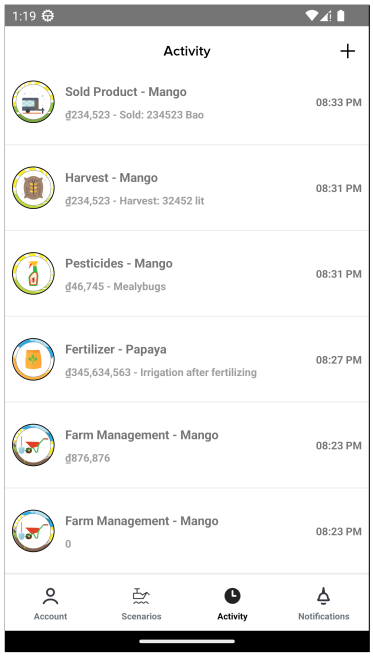


(d) Customize time range for scheduled irrigation scenario

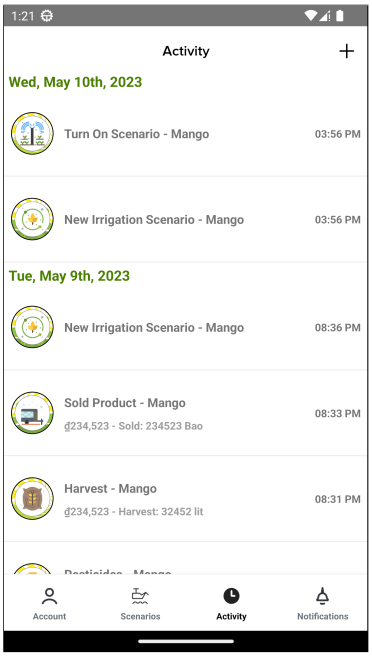
Figure 7.1.7: Configurations for irrigation scenario

Activity Management

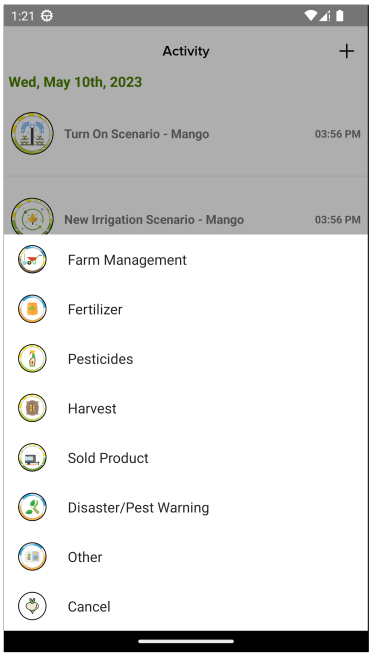
Below are the user-related feature group, displaying a list of activities that the user creates as well as activities that are self-generated from creating scenarios. After being created, the following activities will push data back to the Data Core for training user habits, thereby proposing more advanced irrigation scenarios.



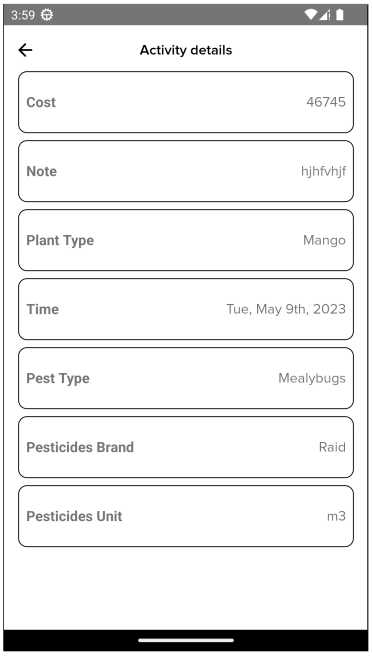
(a) List activity generated manually



(b) List activity generated from user irrigation scenario



(c) List options for creating activity manually

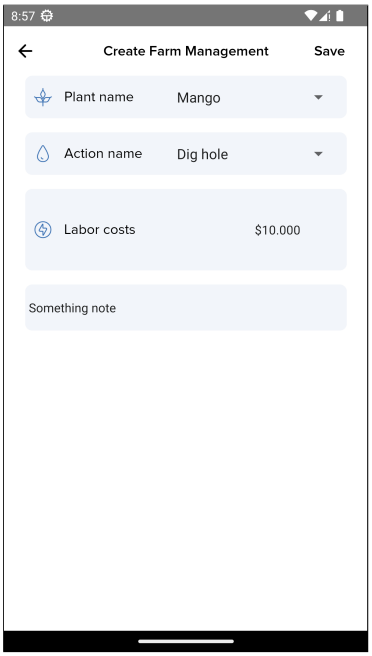


(d) View activity details

Figure 7.1.8: Activity Management

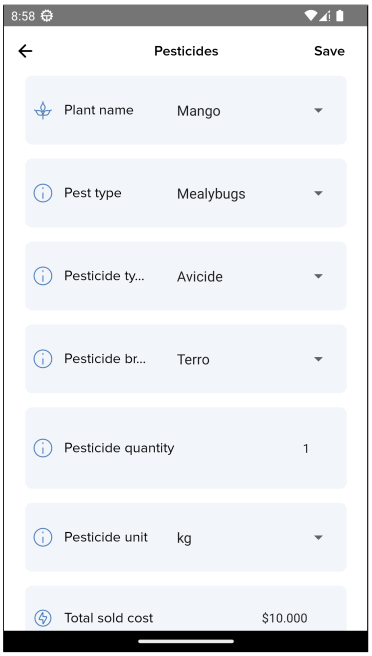
Here are the options for creating manual activities for users. Each type of activity will have different information fields for that activity.

7.1. Implementation Result

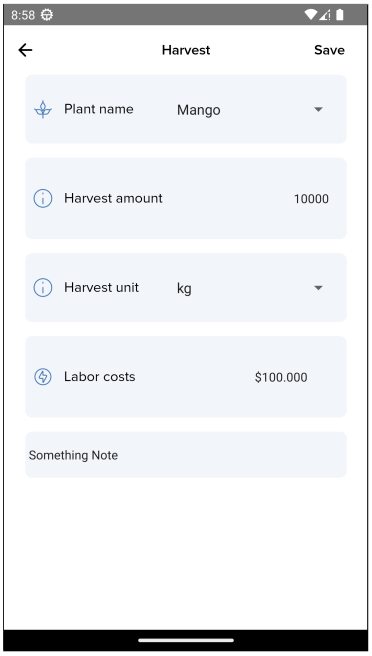


(a) Creating activities for farm management

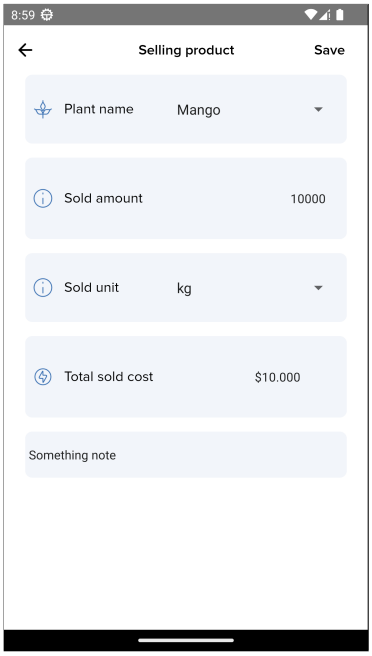
(b) Creating activities for crop fertilization



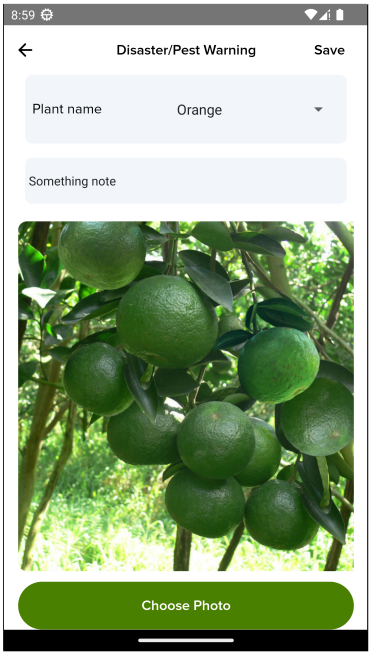
(c) Creating activities for pesticide spraying



(d) Creating activities for crop harvesting



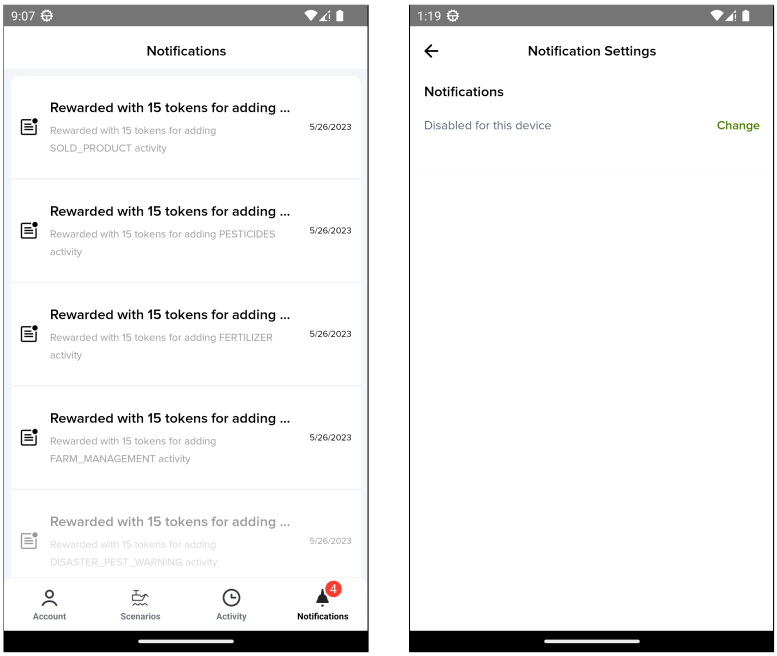
(e) Creating activities for selling crop products



(f) Creating activities for warning disasters and pest

Figure 7.1.9: Activity Types

Notification Management



(a) List notifications (b) Configure notification

Figure 7.1.10: Notification Management

7.1.2 Web Application

Authentication

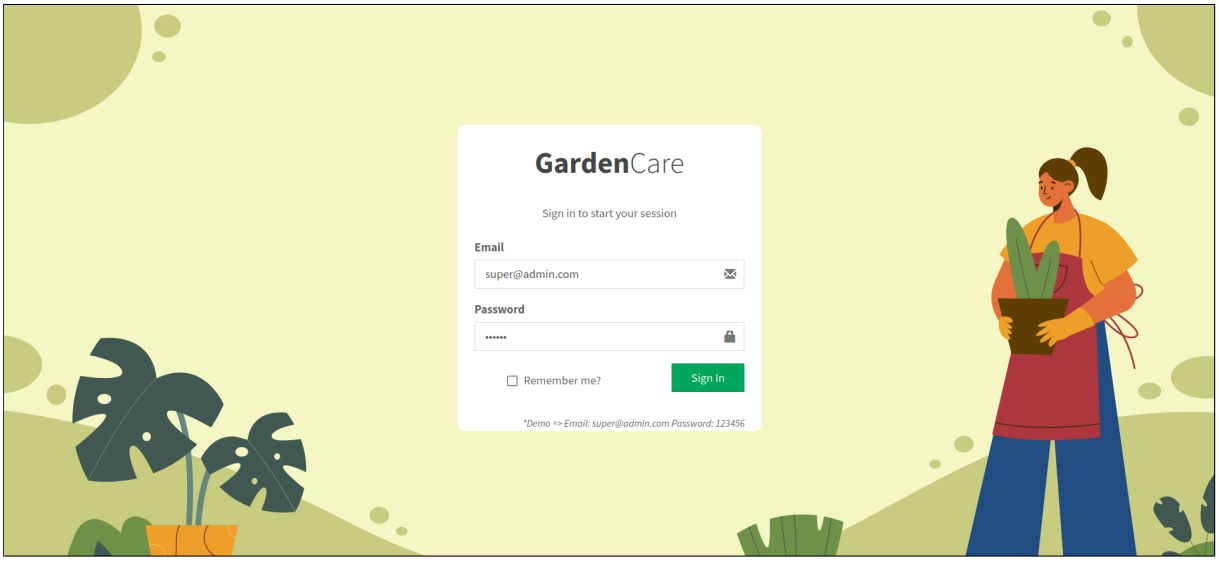


Figure 7.1.11: Login Page

Customer Management

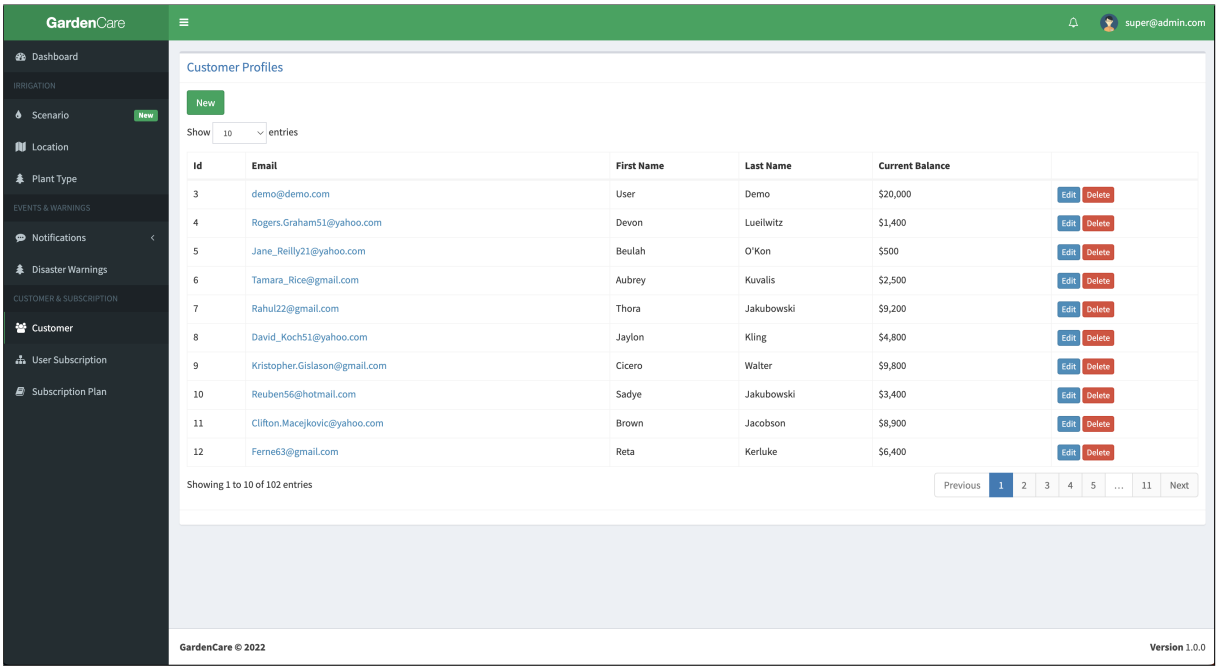


Figure 7.1.12: Index Page

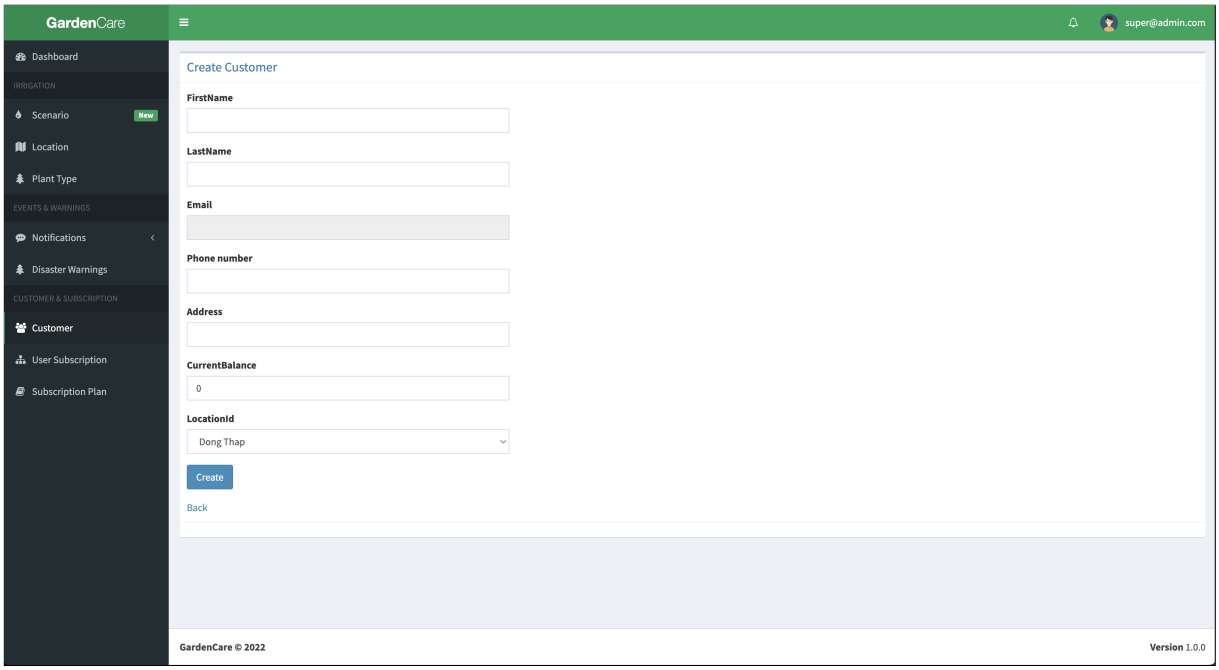


Figure 7.1.13: Create new customer

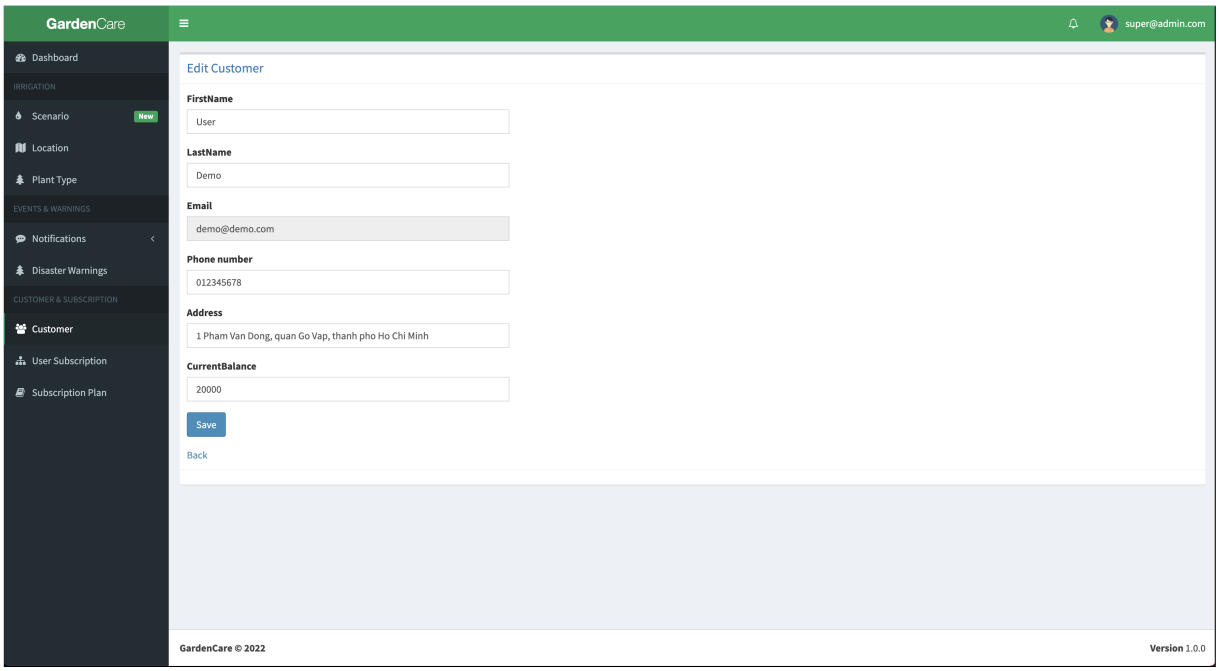


Figure 7.1.14: Edit customer information

7.1. Implementation Result

Scenario Management

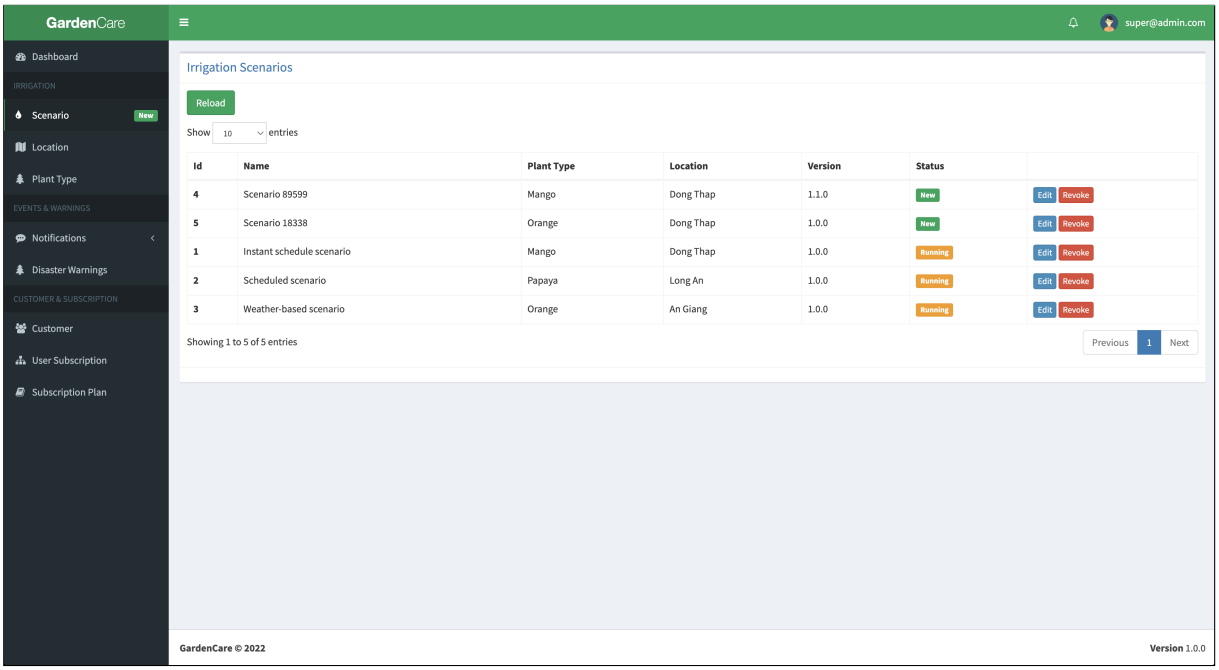


Figure 7.1.15: List of available scenarios and its status

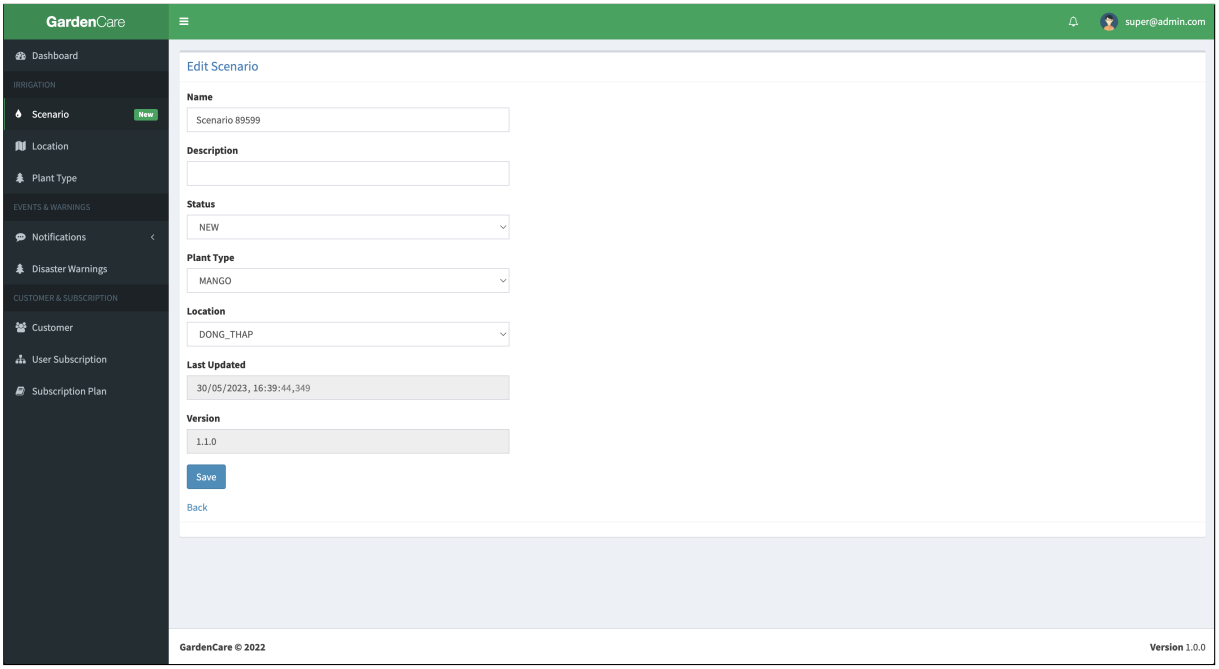


Figure 7.1.16: Edit a irrigation scenario

Plant Type Management

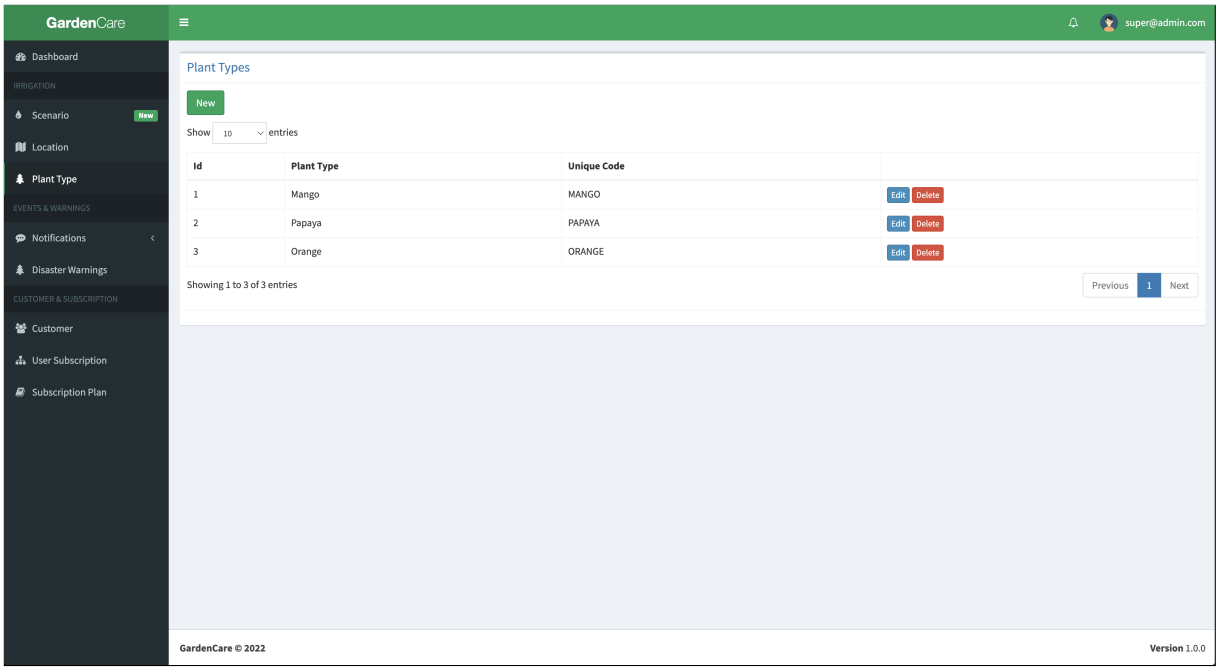


Figure 7.1.17: List of plant types available

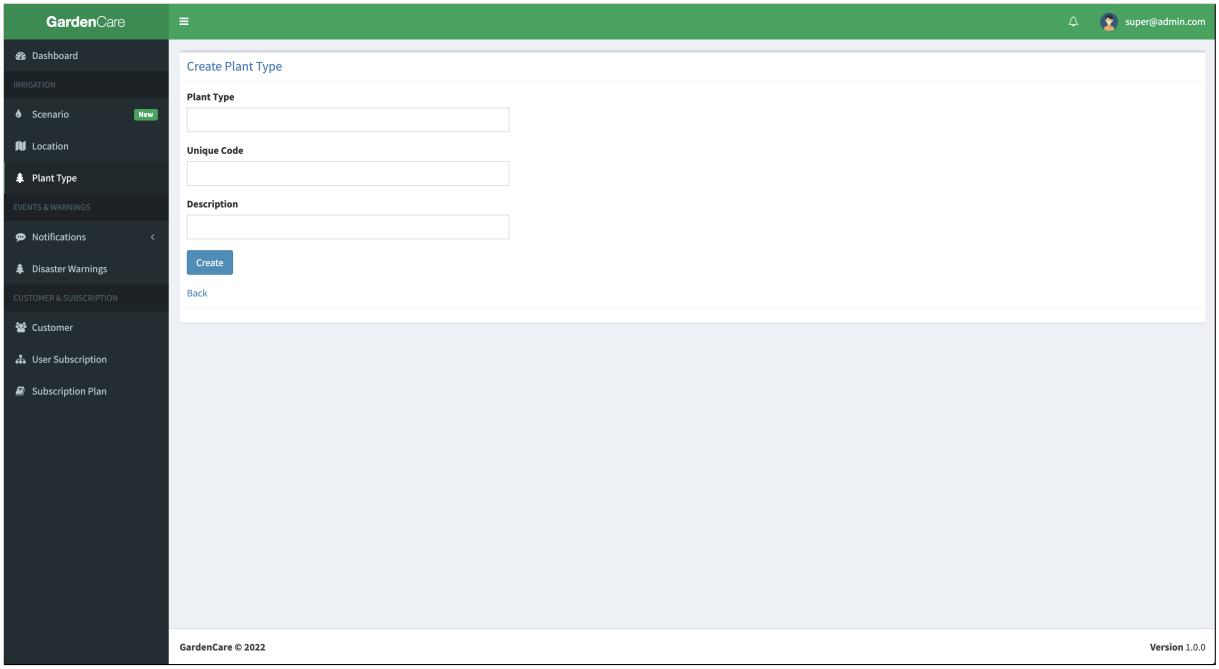


Figure 7.1.18: Create new plant type

7.1. Implementation Result

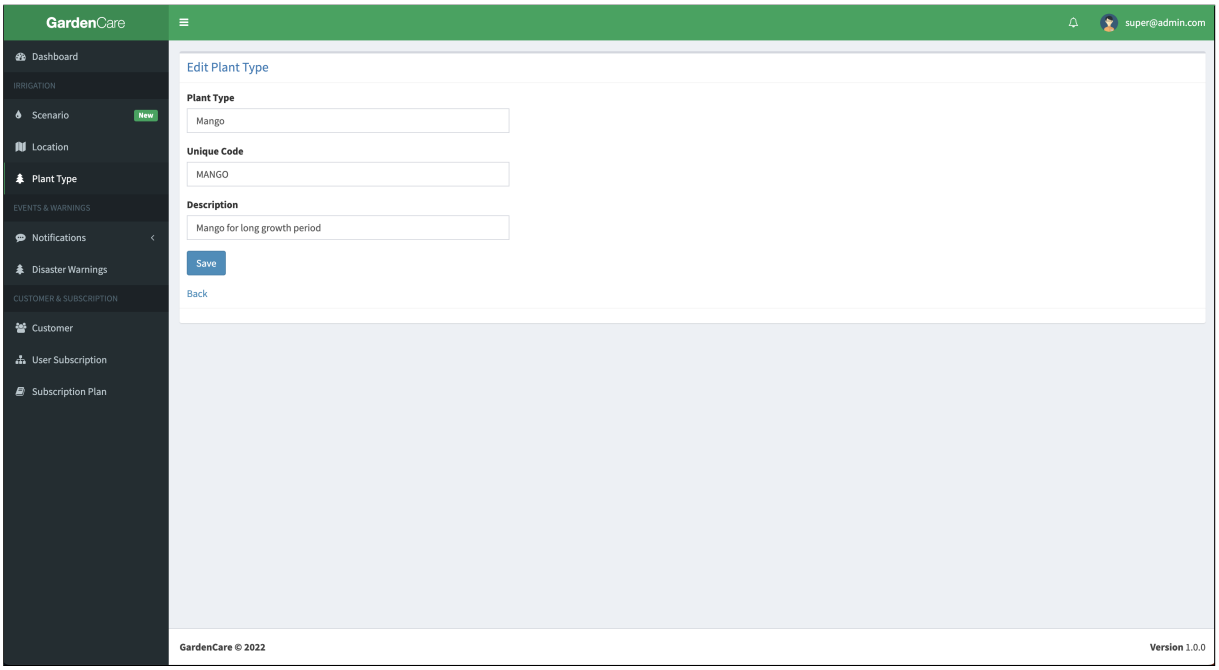


Figure 7.1.19: Edit information for a plant type

Location Management

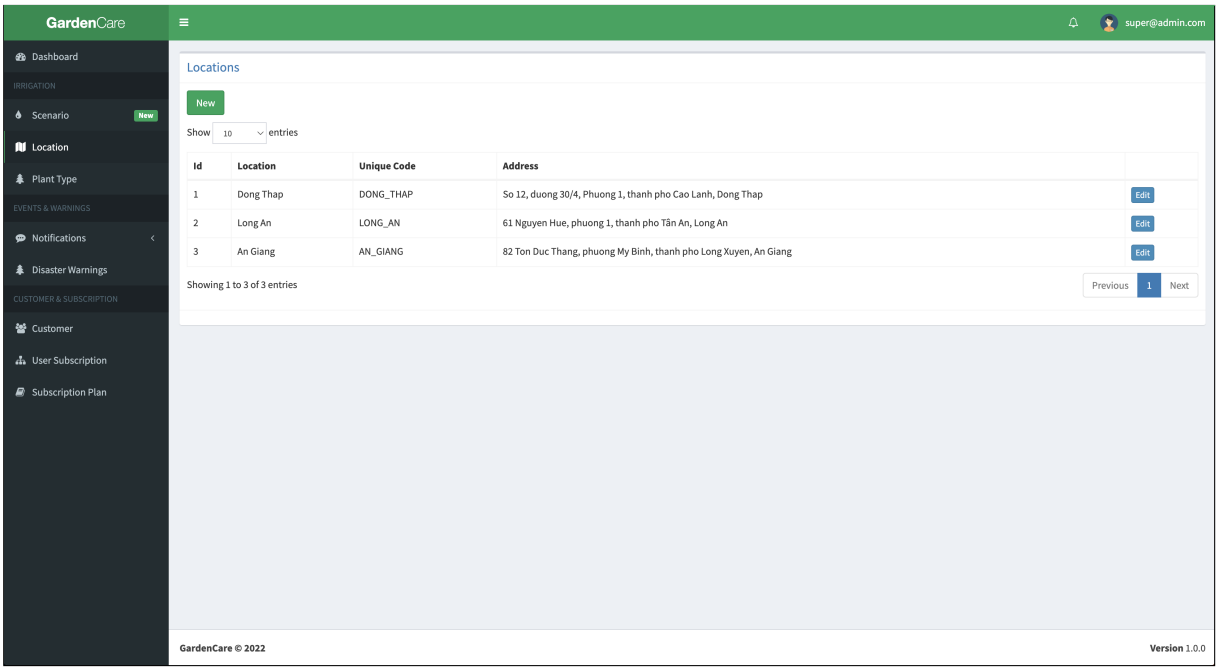


Figure 7.1.20: List of locations available

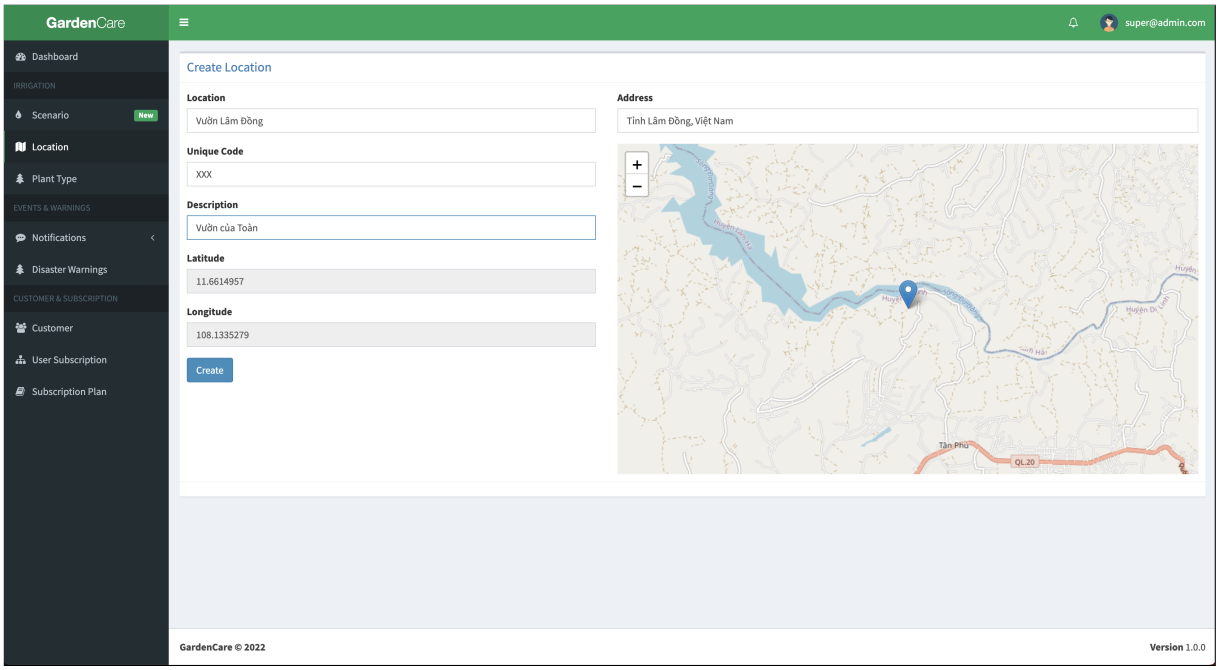


Figure 7.1.21: Create a location

Notification Management

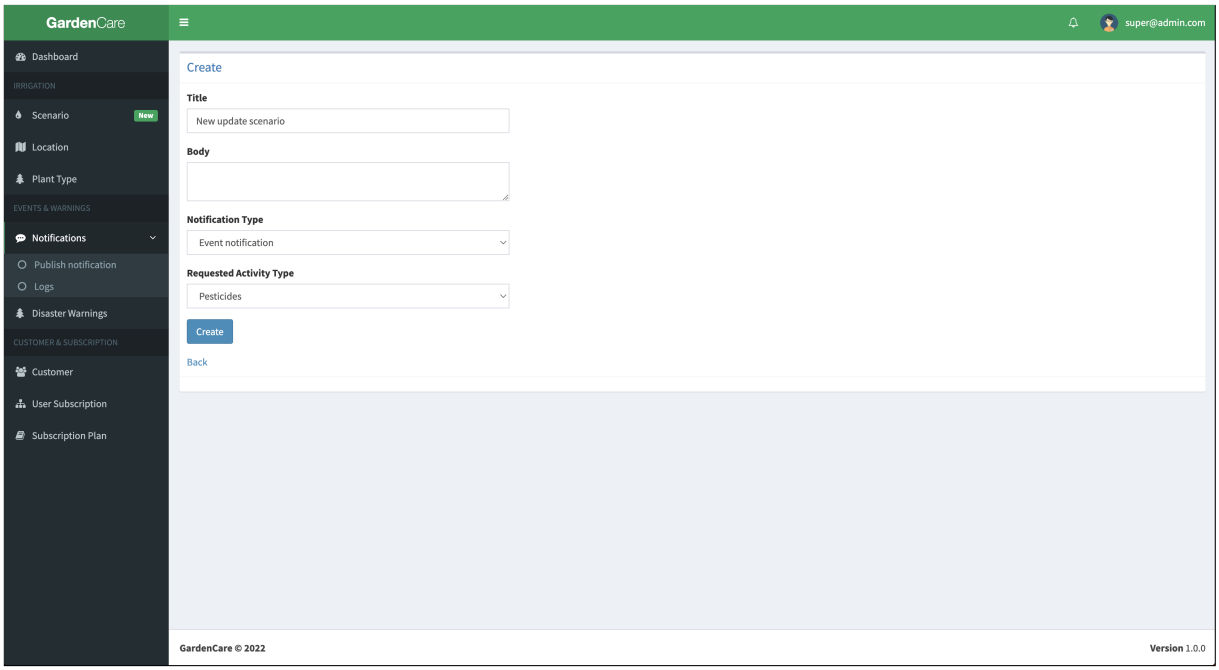


Figure 7.1.22: Create new notification

7.1. Implementation Result

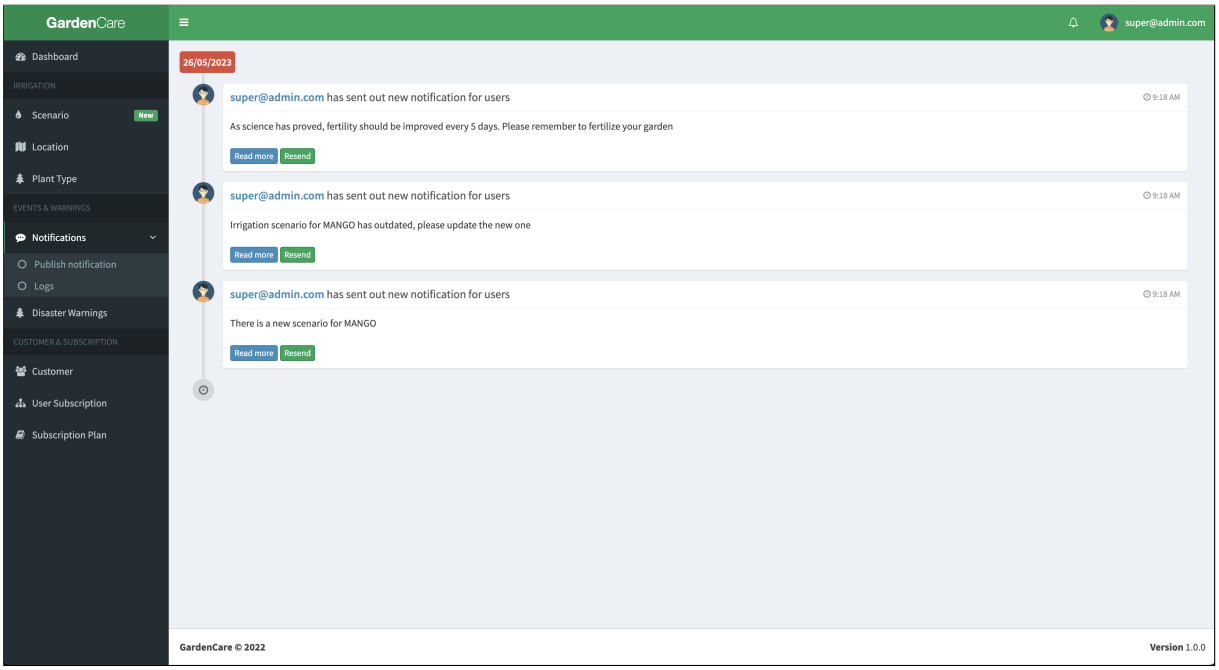


Figure 7.1.23: Tracking notification

7.2 Testing Plan

Testing software is a crucial step in the software development process. In order for a system to operate stably, developers need to test all features and components of the system, ensuring that everything runs smoothly before it can be delivered to users. Any minor error that is not carefully checked can have consequences later on, making the system unreliable and receiving little response from users. The testing process needs to go through many different types: Unit test, End to end test, Integration test... Nowadays, there are many efficient and powerful testing libraries available...

Goal

Software testing will help ensure the system's functionality is fully implemented, including functional and non-functional requirements. Testing software also ensures that the system is ready for use.

In addition, software testing verifies that the system meets various requirements including performance, reliability, safety, applicability, etc. This verification is done to ensure that the team is building a suitable system.

Scope

Test the system's functions to ensure that it operates correctly according to the requirements outlined in the description.

- For end-user (farmer)
  - Manage personal account :
    - Login
    - Logout
  - Manage irrigation scenarios:
    - Create/delete irrigation scenarios
    - View detailed information list of irrigation scenarios
    - Receive the latest irrigation scenario template
  - Manage notification:
    - Receive new notifications from service providers
    - Execute or update scenarios from new notifications
    - Configure notifications

  - Manage farming activities:
    * Create activities based on different types
    * View a list/details of activities based on different types of activities
  - Manage subscriptions:
    * Subscribe to new template scenarios with the provider
    * Implement the mechanism of depositing money into the system

- For fog node (service provider)
  - Manage end-users (customer)
    * View list of users
    * Create/Remove new user
  - Manage plant type
    * View current defined plant types
    * Create/edit/remove plant type
  - Manage location
    * View current defined locations
    * Create/edit/remove location
  - Manage subscription plan
    * View current subscription plan
    * Create/edit/remove subscription plan
  - Manage subscription of irrigation scenarios for end-users:
    * View users subscription
    * Create/edit/remove user subscription
  - Manage irrigation scenarios:
    * Fetch new scenarios from Data Core system
    * Edit metadata from fetched irrigation scenarios
    * Remove scenarios
    * Update irrigation scenario status.
  - Manage notifications:
    * Push notifications to end-users

Testing environment

Tested on Ubuntu, Window 11, MacOS Ventura, and Android Galaxy A03 operating systems.

7.3 Functional Testing

Due to the nature of the topic focusing on business operations, the functions of the system will be tested by the testing team after each function is implemented. For mobile, the team selected the function of creating an irrigation scenario based on the template irrigation scenario and the function of creating a farm management activity for testing. For the dashboard, the team selected the function of creating a Customer for testing.

Authentication

STT	Test case description	Expected result	Status
1	Enter email: **user@example.com** Enter password: **string** Click "Login" button	Successfully logged in, go to Account screen	PASS
2	Login with incorrect email or password	Display error message for incorrect email or password	PASS
3	Login with empty email/password	Display error message to require filling in the required fields	PASS

Table 7.3.1: Test Case Results for Authentication

Irrigation scenarios

STT	Test case description	Expected result	Status
1	Create a new irrigation scenario but not register a subscription plan for each type of tree	Can't proceed to the Irrigation scenario configuration screen, do not show a list of sample irrigation scenarios	PASS
2	Create a new irrigation scenario with a subscription plan for each corresponding tree type but not fill in enough fields when adding a new rule	Unable to create a new rule for that scenario, unable to create a new scenario in case there is no rule	PASS
3	Create a new irrigation scenario with a subscription plan for each corresponding tree type and fill in all necessary fields for each rule	Successfully create a new irrigation scenario	PASS
4	Delete an irrigation scenario	The scenario is removed from the displayed list	PASS

Table 7.3.2: Test Case Results for Irrigation Scenario

Activity

STT	Test case description	Expected result	Status
1	Select an activity type to create a farming activity but not fill in enough information	Cannot create an activity, display error message to require filling in enough information	PASS
2	Select an activity type to create a farming activity and fill in all necessary information	Successfully create a new activity, display corresponding icon and activity type	PASS
3	Creating a new irrigation scenario will create a new irrigation activity on the Activity screen	Successfully create a new irrigation activity, display corresponding icon and activity type	PASS
4	Changing the irrigation scenario status will create a new irrigation activity on the Activity screen	Successfully create a new irrigation activity, display corresponding on/off icon for each scenario status	PASS

Table 7.3.3: Test Case Results for Activity

Notification

STT	Test case description	Expected result	Status
1	Change notification permission, when the notification arrives, it will not be displayed when rejecting the notification permission, and it will be displayed when allowing the notification permission	Successfully display corresponding notification when having permission and hide notification when rejecting permission	PASS
2	When the user clicks on the notification, take the user to the notification detail screen	Successfully navigate the user to the notification detail screen	PASS

Table 7.3.4: Test Case Results for Notification

7.4 Non-functional Requirements Testing

Compatible Testing

As we all know, the same website will be displayed differently on different browsers. We need to check if the application is displayed correctly with without losing any critical components on different browsers.

It's important to test the web application on as many browsers as possible (IE, Firefox, Chrome, Safari, Opera, etc.) to test for compatibility. Test on different versions of each browser.

#	Description	Result
1	Accessible from Chrome	PASS
2	Accessible from Firefox	PASS
3	Accessible from Opera	PASS

Table 7.4.1: Compatibility Testing

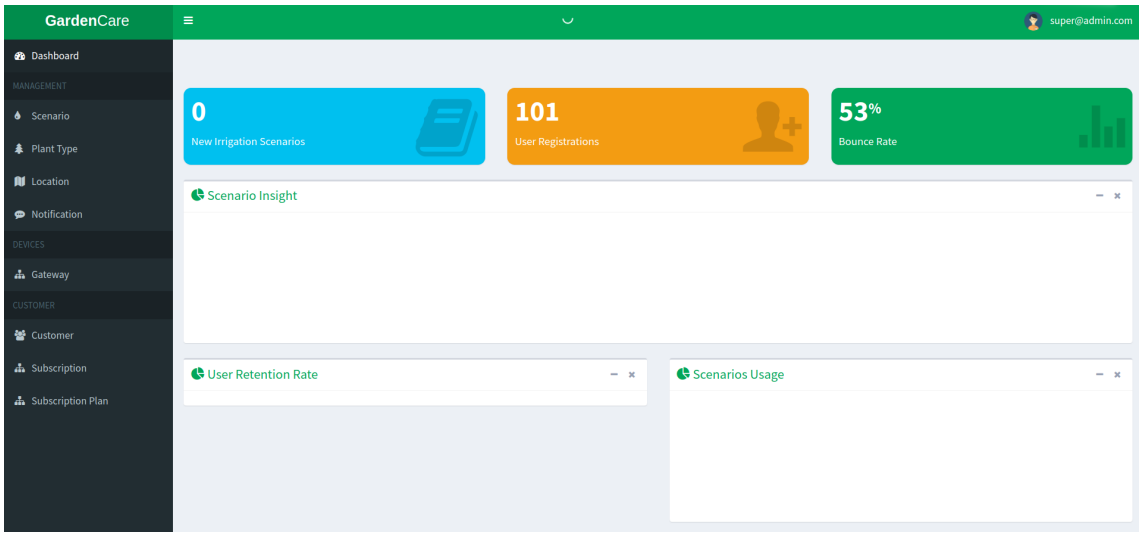


Figure 7.4.1: Compatibility test with Chrome

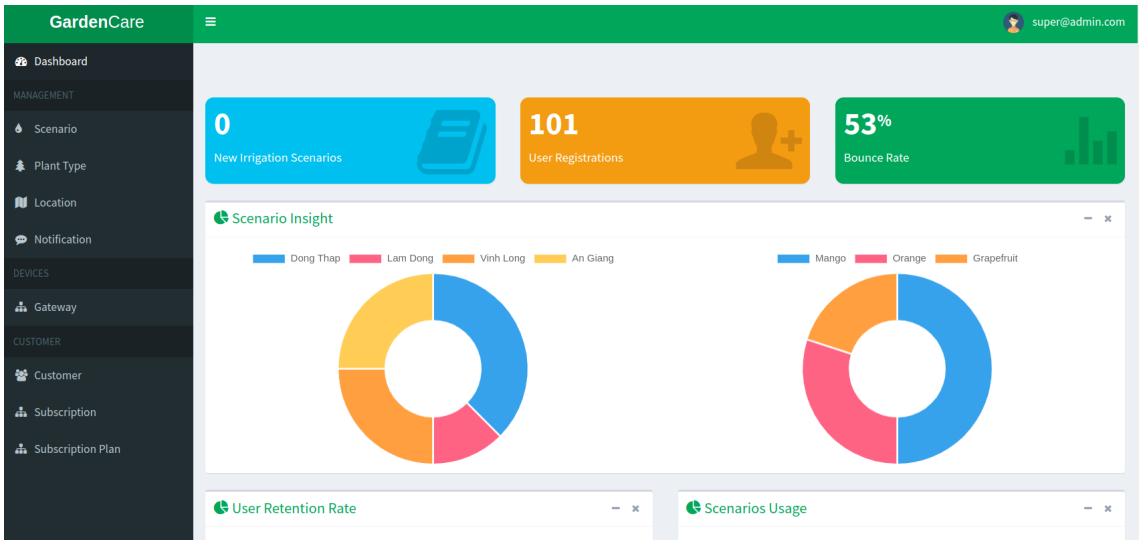


Figure 7.4.2: Compatibility test with Opera

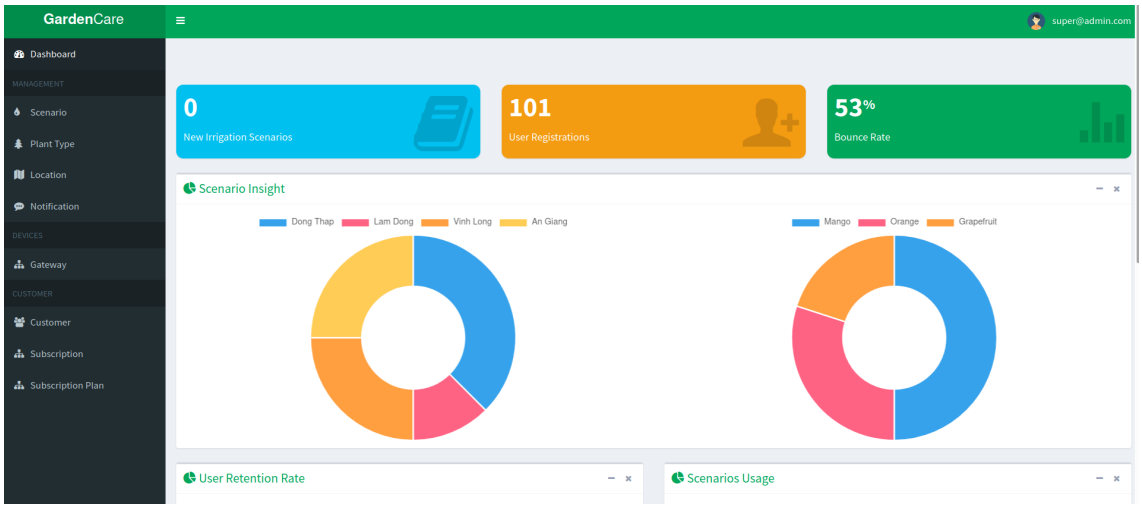


Figure 7.4.3: Compatibility test with Firefox

Security Testing

Security Testing is vital important for all applications that contain personal information and purchase details of users. Hence, we have some tests to ensure that basis of security measurement has been done, including:

1. Directly input URL will not direct user to unauthorized pages without granting permission beforehand.

2. User can't access to unauthorized resources from other users.

3. Authentication token (JWT token) and cookies will be expired over the time, needs to refresh in order to access the application.

4. Invalid username, password with unexpected characters will cause validation errors, required user to re-input.

5. All transactions, unusual activities, error logs are collected and backup-ed for further monitoring and tracking for any malicious activities.

7.5 Acceptance Testing

Acceptance testing is a vital process of testing conducted by the customer in order to confirm that the application operates as expected according to the customer's requirements. The acceptance test is the final phase of testing before the system is put into official operation.

In the scope of the capstone project, we have publicize the application on *Google Play*, available to to all interested users. We will collect feedback from users to complete the acceptance testing as soon.

7.6 Conclusion

Software testing and evaluation is a vital part of software development. Each process plays an important role in ensuring the application is working as expected. Through the testing section, we conclude that application (both mobile and web app) has passed all non-functional and functional requirements that explicitly listed from Section 4.

Chapter 8

Conclusion

8.1 Achievements

Through the process of conducting our capstone project, we have concluded some achievements as below:

- Conduct research and have meetings to gather business ideas and requirements that made available by High Performance Computing Lab to finalize the idea, then actualize it into real usable products. It is an arduous yet necessary process for a well-rounded future engineer.
- Our team found trivial solutions for numbers of problems in order to serve users with better experience.
- More hand-on experience with application framework for React Native, .NET.

Mobile Application

- Intuitive and easy-to-use application, runs smoothly and compatible with most mobile devices.
- Supports users in creating and executing irrigation scenarios according to templates and can be self-adjusted.
- Provides good user support in creating farm management activities as desired.
- Provides good support for notifying users of upcoming events.

Web Application

- Intuitive website, easy to use, compatible with most available web browser.

- Provide a mechanism to fetching and distributing scenarios from *Data Core System* to each users with highly customized configurations that suited users' need.
- Collect user data in many kind of terms: API log, cultivation activities on garden, customized scenarios by users,.. to provide the center with a large data source to do further investigations and academic researches.
- Capable of providing stats like monthly revenue.
- Capable of providing total user, active users by month, user retention rate.
- Capable of providing usage of irrigation scenarios by criteria.

8.2 Deficiencies

8.2.1 Mobile Application

- The application lacks many notification processing threads that are convenient for users to interact with.
- It is not diverse in irrigation scenarios.
- It does not support users to view statistics on revenue as well as costs for activities created.
- It does not entirely ensure the reliability of users.

8.2.2 Web Application

- User Interface is unattractive with users.
- Lot of statistics that may necessary are not been introduced.
- Not introduced any payment method to purchase subscription.
- Haven't introduced a full solution to connect and handle control with gateways. Still working with mock gateways instead.

8.3 Future Plan

- Add payment method to help users purchase subscriptions (Paypal, Momo, Banking,..)
- Introduce more stats into dashboard, provide admin with more useful information about users and the products.

- Enhance UI of admin dashboard.
- Make mobile application more user-friendly with better details graphics.
- Develop a solution, protocol or SDK (software development kit) - a set of software tools and programs provided by hardware and software vendors that developers can use to build applications for specific platforms. Hence, multiple kinds of gateways work with together regardless of their manufacturer and locality.

References

[1] Research Dive Analysis. Smart agriculture market by agriculture (precision farming, livestock, aquaculture, and greenhouse), component (solution, service, and connectivity technology), and regional analysis (north america, europe, asia-pacific, and lamea): Global opportunity analysis and industry forecast, 2021–2028, Aug 2021.

[2] David Anderson. How NAT traversal works, Aug 2020.

[3] Ahmad Muzaffar bin Baharudin, Mika Saari, Pekka Sillberg, Petri Rantanen, Jari Soini, Hannu Jaakkola, and Wanglin Yan. Portable fog gateways for resilient sensors data aggregation in internet-less environment. *Engineering Journal*, 22:221–232, 06 2018.

[4] Xianghui Cao, Jiming Chen, Yan Zhang, and Youxian Sun. Development of an integrated wireless sensor network micro-environmental monitoring system. *ISA Transactions*, 47(3):247–255, 2008.

[5] Roberto Casado-Vara, Francisco Prieto-Castrillo, and Juan M. Corchado. A game theory approach for cooperative control to improve data quality and false data detection in wsn. *International Journal of Robust and Nonlinear Control*, 28(16):5087–5102, 2018.

[6] Wendy Cunningham and Obert Pimhidzai. Vietnam's future jobs. *Word Bank*, 2018.

[7] Minh Duyên and Hoàng Linh. Sống chung với hạn hán, xâm nhập mặn, September 2020.

[8] ELARD/REDR. Which future do you want in rural areas?, 2020.

[9] EPRS. *Smart villages: Concept, issues and prospects for EU rural areas*. European Parliament Research Service, 2021.

[10] Sheikh Ferdoush and Xinrong Li. Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications. *Procedia Computer Science*, 34:103–110, 2014. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops.

[11] Iberdrola. Smart villages, when technology arrives at a village to stay, July 2021.

[12] Hanu Kommalapati. Iot measure and control loops, November 2022.

[13] Michael Lesk and Brian Kernighan. Computer typesetting of technical journals on UNIX. In *Proceedings of American Federation of Information Processing Societies: 1977 National Computer Conference*, pages 879–888, Dallas, Texas, 1977.

[14] La Hoang Loc. Introduction to the irrigation system, 2022.

[15] Vũ Long. Thiếu nước, xâm nhập mặn nghiêm trọng tại đồng bằng sông cửu long. *Báo Lao động*, 2021.

[16] Vũ Long. Thí điểm mô hình "làng thông minh" trong xây dựng nông thôn mới thông minh. *Báo Lao động*, 2021.

[17] Jo Morrison. Smart cities need smart villages, 2021.

[18] Thoại Nam. Làng thông minh từ mô hình hội quán nông dân tại Đồng tháp, 2020.

[19] Thoại Nam. Xây dựng working flow và mô hình điều khiển tưới tiêu tự động, 2021.

[20] ITU News. Digital necessity: How smart villages are changing lives in niger, 2022.

[21] Linh Van Hoai Nguyen. Nghiên cứu phát triển công cụ phân tích dữ liệu nông nghiệp dùng cho làng thông minh. Bachelor's thesis, Ho Chi Minh University of Technology, 2021.

[22] Dezan Shira and Associates. Why the agtech industry will aid vietnam's hi-tech growth, June 2021.

[23] Manuel Suárez-Albela, Tiago M. Fernández-Caramés, Paula Fraga-Lamas, and Luis Castedo. A practical evaluation of a high-security energy-efficient gateway for iot fog computing applications. *Sensors*, 17(9), 2017.

[24] Athanasios Tsipis, Asterios Papamichail, Ioannis Angelis, George Koufoudakis, Georgios Tsoumanis, and Konstantinos Oikonomou. An alertness-adjustable cloud/fog iot solution for timely environmental monitoring based on wildfire risk forecasting. *Energies*, 13, 07 2020.

[25] Worldometers. Water use statistics, Apr 2023.

[26] Tiffany Yeung. What's the difference between edge computing and cloud computing?, Jan 2022.

Appendix A

Getting Started

Our software makes use of the .NET Core framework, and can be built and run using their dotnet toolchain. Alternatively, we provide the ability to run our software in containers.

A.1 Containers

A.1.1 System Requirements

The system requirements depends on the container runtime that you use. For example, this is the system requirements for Podman:

- 64-bit processor with Second Level Address Translation
- BIOS-level hardware virtualization support must be enabled in the BIOS settings
- 2 CPU cores
- 4GB of RAM
- 5GB of storage

A.1.2 Dependencies

- Any container runtime that is compliant with the Docker format or OCI format will work, e.g.: Docker, Podman, or kaniko.
- PostgreSQL v15 running locally or remotely

A.1.3 Bootstrapping

We are building and deploying our software using Podman. Other container runtimes will work too, but some command and parameter changes may be needed.

```
podman build -f Containerfile -t fog-mgmt:latest
podman run -d --replace -p 80:80 --name fog-mgmt \
  -e CUSTOMCONNSTR_DefaultConnection="Username=postgres;
  Password=postgres; Server=localhost; Port=5432; Database=fog-mgmt" \
  -e ASPNETCORE_URLS="http://0.0.0.0:80" \
  fog-mgmt:latest
```

where `CUSTOMCONNSTR_DefaultConnection` is a string containing the information to connect to the database.

A.2 Direct Run

A.2.1 System Requirements

Including the .NET SDK, our software requires at least:

- .NET version 6 or 7
- 2 CPU cores
- 4GB of RAM
- 5GB of storage

A.2.2 Dependencies

- .NET Core SDK to build and run, or .NET Core runtime to run.
- PostgreSQL v15 running locally or remotely.

A.2.3 Bootstrapping

With the introduction of .NET Core, software built on .NET has the ability to run on multiple platforms. Here we provide a generic instruction to build and run our software in Bash shell on Linux.

```
dotnet restore
dotnet build -c Release -o app/build
dotnet publish -c Release -o app/publish /p:UseAppHost=false

export CUSTOMCONNSTR_DefaultConnection="Username=postgres;
Password=postgres; Server=localhost; Port=5432; Database=fog-mgmt"
dotnet run app/publish/fog-mgmt.dll
```

where the `CUSTOMCONNSTR_DefaultConnection` variable must be set in the environment of the system.

On Windows, you can add it to the system via the `Environment Variables` dialog, or you can do this the PowerShell way:

```
$env:CUSTOMCONNSTR_DefaultConnection="Username=postgres;
Password=postgres; Server=localhost; Port=5432; Database=fog-mgmt"
dotnet run app/publish/fog-mgmt.dll
```