Technical University of Crete

School of Electrical and Computer Engineering

Diploma Thesis

Management of Farms and Crops Data in the Cloud

Apostolos Rousalis

Thesis Committee

Euripides Petrakis, Professor (Supervisor)

Vasilis Samoladas, Assistant Professor

Stelios Sotiriadis, The Edward Rogers Sr. Department of Electrical and Computer Engineering University of Toronto, Canada, Research Associate

Chania 2017

This page intentionally left blank

Acknowledgement

I would like first to express my special appreciation and thanks to my supervisor professor Euripides Petrakis for his guidance and his commitment until the time completion of this thesis. I would also like to thank the members of the lab Spyros and Kostas, and all the postgraduate and undergraduate students, for helping me on this project. Finally, I must express my very profound gratitude to my family and to my best friends for providing me with unfailing support throughout my studies and through the process of writing this thesis. Thank you.

Abstract

Recent technological advances in cloud computing paved the way for developing and offering advanced services for remote monitoring in the agricultural sector. In this thesis, we developed a cloud-based Farm Management Information System, referred as CloudFarm. CloudFarm offers a variety of services that helps farmers to manage and keep control of their farms and crops, by supporting all the basic agricultural activities according to set of Good Agricultural Practices. CloudFarm is accessible to farmers, as a mobile application running on a smart-phone or tablet that connects with the cloud, and as a Web application running in the cloud. The “on the go” system means that farmers can manage and capture scheduled or performed activities directly from fields, while using a mobile application. Mobile application is synchronized with information stored permanently in the cloud so that farmers can have always access to up-date information. Furthermore, CloudFarm offers a Web application where farmers can monitor their data and certified advisors can monitor the farmers’ agricultural activities and advise them about their farm status and best planning of farm and crops works.

Contents

Chapter 1 - Introduction 1

1.1 Motivation 1

1.2 Proposed Solution 2

1.3 Thesis Structure 3

Chapter 2 – Background & Existing Technologies 4

2.1 Cloud Computing 4

2.1.1 Cloud Computing Service Models 5

2.1.2 Cloud Computing Deployment Models 7

2.1.3Cloud Computing Providers 7

2.1.4 Cloud Computing Virtualization of Resources 9

2.2 Internet of Things (IoT) 9

2.2.1 Cloud Computing and the Internet of Things 10

2.3 Service-Oriented Architecture (SOA) 11

2.4 JSON (JavaScript Object Notation) 12

2.5 Web Services 12

2.5.1 HTTP (HyperText Transfer Protocol) 12

2.5.2 REST (Representational state transfer) 14

2.5.3 API (Application Programming Interface) 16

2.5.4 Jersey – RESTful Web Services in Java 16

2.6 Mobile Application Development 16

2.6.1 Native 17

2.6.2 Web 17

2.6.3 Hybrid 17

2.7 Web Application Development 18

2.7.1 Single Page Architecture (SPA) 18

2.7.2 AngularJS Framework 19

2.8 Apache CouchDB 20

Chapter 3 – CloudFarm Design & Implementation 22

3.1 General CloudFarm System Design 22

3.2 User Roles 23

3.3 Functional Requirements 23
3.4 Use Case Diagrams 25
3.5 Activity Diagrams 27
3.6 Class Diagram 33
3.7 Non-functional Requirements 37
3.8 CloudFarm System Implementation 37
3.8.1 CloudFarm Architecture 37
3.8.1 CloudFarm Back-End 38
3.8.1.1 Application Logic 38
3.8.1.2 Storage Service 38
3.8.1.3 Authentication & Authorization Service 39
3.8.1.4 Plant Protection Service 40
3.8.1.5 Users Service 40
3.8.1.6 Fields Service 41
3.8.1.7 Crops Service 42
3.8.1.8 Harvests Service 43
3.8.1.9 Storehouses Service 44
3.8.1.10 Sprays Service 45
3.8.1.11 Fertilizations Service 46
3.8.1.12 Advisors Service 47
3.8.1.13 API Call Request Headers 49
3.8.2 CloudFarm Front-End 49
3.8.2.1 Mobile Application 49
3.8.2.2 Web Application 51
3.9 Launching the instance in FIWARE lab 52
3.10 CloudFarm System Deployment 52
3.11 CloudFarm Presentation 53
Chapter 4 – Performance Evaluation 65
4.1 API Performance 65
Chapter 5 – Conclusion and Future Work 68
5.1 Conclusion 68
5.2 Future Work 69
References 70

Chapter 1 - Introduction

Nowadays, there is a constant challenge for food producers to grow safe, healthy products in a responsible way. There is a huge need for farmers to conform with Good Agricultural Practices[1] (GAP) codes, standards and regulations in order to guarantee food safety and product quality. Farmers must be well organized and able to manage all the details of the farm and be certified from certified organizations by providing information that covers a period from the seeding process to the multiple farming activities until post-harvest management.

By adopting the use of modern information technology along with approved management techniques and engineering technology, the development of farm management information systems is the new way of optimizing farm production in terms of product quality and operation efficiency [23,24]. There is an information flow that provides the farmer with external knowledge and decision support in order to perform efficient field operations, and it serves as a means of transmitting data about farm and field operations. A large amount of data from fields activities are collected by electronic devices and transmitted to remote storage infrastructures. As an additional benefit, various stakeholders such, as agronomists, government and legislation bodies, industries, tap into this information system of data flow to collect and analyze information.

1.1 Motivation

The increasing pressure on farmers to follow the Good Agricultural Practices enforced them to provisions and restrictions in the use of input products such as fertilizers and agrochemicals. Farmers deal with huge managerial load and need to keep track of huge information loads in order to keep up to date records of farm operations (e.g., crop operations). This process is time-consuming and suspicious to human errors, due to the fact that all information must be checked and recorded by the farmer, that can easily lead to false results and decisions. Most farmers are using spreadsheets to keep their data organized and they have to insert all information manually and cross-check the performed agricultural activities themselves, such as remembering the exact days of the performed activities, the chemical amount consumed in a spraying activity, updating the stock amount after a spraying or fertilization activity, keep track of the field and crop history and more. All these types of cross-checking are time consuming for farmers and there is also a huge risk of forgetting details about performed activities such as dates, used pesticides, used quantities and false or incomplete information in spreadsheets, might contribute to losing a certification, due to the fact that their data is not according to good agriculture principles.

The increasing use of computers, automate the way of handling and processing farm data but still many farm management (FMS) software systems are made to be used by farmers as well as advisors who are working in front of a desktop computer rather than on the field and they are not able to leave the computer desk where the systems are developed. Besides the risk of incomplete or false information (as data input is done off-line), there is also the risk of data loss, as farm data stored locally in a computer, data can be damaged because of a computer virus or mechanical or software damage. So, there is a need of a farm management information system that runs from

[1] http://extension.psu.edu/food/safety/farm/gaps

a safe remote infrastructure that can be accessed from a Web browser over the internet, minimizing this way the farmers' hardware maintain costs.

1.2 Proposed Solution

This dissertation focusses on the development of a farm management system that is not running locally on the farmer's personal computer but is an "on the go" system meaning that farmers can capture performed activities directly from fields, using a mobile application. Both farmers and their advisors can monitor the farm data through a web application accessed from a web browser over the internet. Our proposed solution exploits state-of-the technology solutions in computer and internet technology using the internet such as remote activity management and monitoring via mobile devices (smartphones, tables). The concept of cloud computing gets significant attention during the latest years and many companies recognized the advantages and the impact can have in people's life. Today, it is evolved to an important technology for application developers and end users by allowing on demand and remote access of resources. Cloud allows real time data collection and analysis in an efficient way by offering an infinitive view of resources, remote data management, easy access and economic benefits. Taking cloud computing benefits into consideration, we develop an application that is easy to use and is made accessible to farmers, as a mobile application running on a smart-phone or tablet and connects with the cloud, and as a web application running in the cloud. Our ambition is to increase the acceptance of farmers with the minimum exposure to technology, as all communication is through user-friendly interfaces and other means (e.g. instructions by narration and iconic interfaces).

The goal of the thesis is the development of a farm management information system, called CloudFarm, running on the cloud with main purpose to increase farmers' farms productivity. It will support the management (e.g. monitoring, recording) of all the basic agricultural activities such as spraying, harvest, fertilization according to a collection of agricultural rules (G.A.P). As referred to above, CloudFarm is running on the cloud, which means it can be accessed anywhere and anytime by a mobile application or a web application with an internet connection.

The following summarizes the advantages and main characteristics of CloudFarm application:

- It runs on the cloud and assists farmers in keeping track of their crop and harvesting activities, anytime, anywhere, through friendly interfaces running on a smartphone, tablet or desktop.
- It reduces the paperwork and increases safer data collection by minimizing human errors and information loss.
- It can be used off-line in cases of loss of internet connection and allows data collection off-line.
- It syncs local data with data stored on the cloud when the internet connection is back.
- It helps farmers to get a certification for their products and processes, which increases the direct revenues for their products and the loss of income due to products non-sold.

- It helps farmers to save money in production by maximizing the economic benefit for the farmer. The benefit comes from reduction of inputs (agrochemicals) and / or achieved better performance.

Also, the fact that the system is on the cloud also provides the following advantages:

- Users can access the services from any place at any time.
- If necessary, a user can acquire more storage resources.
- Users are charged per use of the resources. This way they know how much they are paying and can manage it according to their needs.
- Minimizes hardware and software procurement and possession (e.g., maintenance) costs. This responsibility is left to the cloud provider.
- CloudFarm application resorts entirely to the cloud provider to guarantee good services provision based on good practices and Service Level Agreements (SLA) agreed with the end-users (the farmers).

1.3 Thesis Structure

The rest of this text is structured as follows:
Chapter 2, presents the background technologies that are used in this thesis. Chapter 3, describes the design and implementation of the system. It begins with the analysis of functional and non-functional requirements following by the mapping of requirement to a set of UML diagrams and ends by describing the implementation of the system. Chapter 4, presents the evaluation of the system performance. Chapter 5, concludes the thesis and discusses issues for future work.

Chapter 2 – Background & Existing Technologies

This chapter presents background of Web technologies such as Cloud Computing, Internet of Thing, REST Web Services, APIs, Web and Mobile application development technologies.

2.1 Cloud Computing

Cloud Computing enables convenient, on demand internet-based access to a shared pool of configurable computing resources (networks, servers, storage, applications, and services), rapidly available to the user with the minimum management effort or interaction with the service provider. Cloud Computing offers three service models, **Software as a Service (SaaS)**, **Platform as a Service (PaaS**) and **Infrastructure as a Service (IaaS).** Moreover, there are four common deployment models, each of them represent a specific category of cloud environment, the **Public Cloud**, the **Private Cloud**, the **Hybrid Cloud** and the **Community Cloud** [1].

Current research appears to validate the view that there is a tremendous increment of organizations and users who adopting cloud solutions. The reason why, is because cloud computing offers numerous **advantages** such as [1][2]:

- **Reduced Cost**: Is the most significant benefit of cloud computing, businesses no longer need to worry about infrastructure and maintenance costs. In order to host their applications in the cloud, they just use services provided and maintained by cloud providers. Thus, businesses need for specialized staff is minimized or eliminated. Furthermore, users' costs are also minimized because they can easily access an application through non-expensive devices over the internet.

- **Flexibility**: Users can immediately and easily access the cloud anytime, from anywhere they want, using web-enabled devices, connected to the internet.

- **Elasticity:** Users can scale-up or down their demands for service resources according to their needs. For instance, they can scale up specific computing resources (storage, memory, network etc.) to improve application's performance.

- **"Pay-as-you-go":** It's a payment model according to which users are charged based on their usage of computing resources. Users can scale up/down, customize computing resources and only be charged by what services or resources they used, rather than paying a fixed price for resources that most likely they will not be using at all times. So, users and mainly businesses no longer need to pay for excessive resources in order to handle occasionally extreme workloads.

Although cloud computing advantages benefit many organizations by reducing costs and allowing them to manage and customize their computing resources, cloud computing is based on internet connection. Thus, despite its beneficial advantages there are some concerns

about cloud technology, below will be presented some of **disadvantages of the Cloud Computing** such as:

- **Downtime:** This is the worst nightmare of the users, businesses and cloud providers. Cloud computing systems are internet based which means without an internet connection a cloud based system cannot be accessed by the users or businesses. Moreover, cloud platforms are systems with hardware too, so service outages are possible due to hardware failures so it is possible someday a cloud provider to claim immunity to service outages due to a hardware failure [3].

- **Security and Privacy:** Are data stored in cloud infrastructures safe? Many users are very skeptical about saving their data in the cloud because they don't know exactly where their data are stored. To put it another way, cloud providers are hiding technical details about their infrastructure facilities. Recently Apple's iCloud was hacked and a collection of 500 private pictures of celebrities, were allegedly obtained from iCloud storage. Of course, cloud providers are always trying to apply the best security standards and certifications for their hardware infrastructure, but when users exchanging data by using a remote internet access, nothing is perfectly secure [3].

- **Legal Issues:** Although there is always a service level agreement (SLA) in the contract between the consumer and the cloud provider, there are some scenarios that can raise some serious legal questions. For example, consumers store their sensitive data to a trustful cloud provider but how the consumers are sure that the cloud administrator or an unauthorized user won't have access to their sensitive data? Does the cloud infrastructure notify the user when a violation of the agreement occurs?These are few of the legal issues that the cloud computing technology is dealing with.

- **Vendor-Lockin:** Adopting a cloud solution means binding to a specific platform and vendor, using specific protocols, standards and tools of the cloud and finally, running into a vendor lock-in situation. The end-user may opt to migrate its application to another cloud provider if the current provider decides to raise prices or fails to provide or conform with the agreed SLA. Migrating the application from one provider to another may not be easy or can be costly.

2.1.1 Cloud Computing Service Models

Cloud computing services can be categorized by three service models, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [1]

- **Software as a Service (SaaS):** Delivers a variety of cloud based software applications, running on a cloud infrastructure, accessible to the users via the internet, usually from a web browser. All the underlying cloud infrastructure and cloud services functionalities are controlled and managed by the cloud providers, users can only use the applications through provided interfaces and APIs (Application Programming Interface). SaaS is the most famous model among those three because it offers many advantages. First, SaaS is

reducing the consumers' costs from installing and running applications in their own computers, as also the costs of hardware upgrade or maintenance. Second, SaaS model provides a "pay-as-you-go" or monthly subscriptions, as payment methods. Finally, a SaaS cloud provider automatically perform software updates. Few SaaS applications are Dropbox, Gmail, Google Drive [1].

- **Platform as a Service (PaaS):** Delivers a platform interface with the capability to allow consumers to develop, debug and test applications and services into a cloud infrastructure, using tools supported through the platform's interface. PaaS platforms can ease software development allowing developers huge flexibility on system customization. In addition, consumers are not required to maintain any of the provided software or hardware of the platform because that is a responsibility of the cloud provider [1]. With PaaS anyone can easily create instances of operating system, programming language execution environment, databases, web servers and get these instances working together. Some examples of PaaS platforms are Windows Azure by Microsoft, App Engine by Google.

- **Infrastructure as a Service (IaaS**): Provides to the consumers virtualized computing resources through virtual machines over the internet. Consumers can rent virtual machines with specific hardware such as storage, network or computing power and they're responsible for installing, and updating the software of the virtual machines. The maintenance, update and upgrade of the hardware is the cloud provider responsibility [1]. IaaS examples are the Amazon Web Services (AWS), Google Compute Engine, Windows Azure.

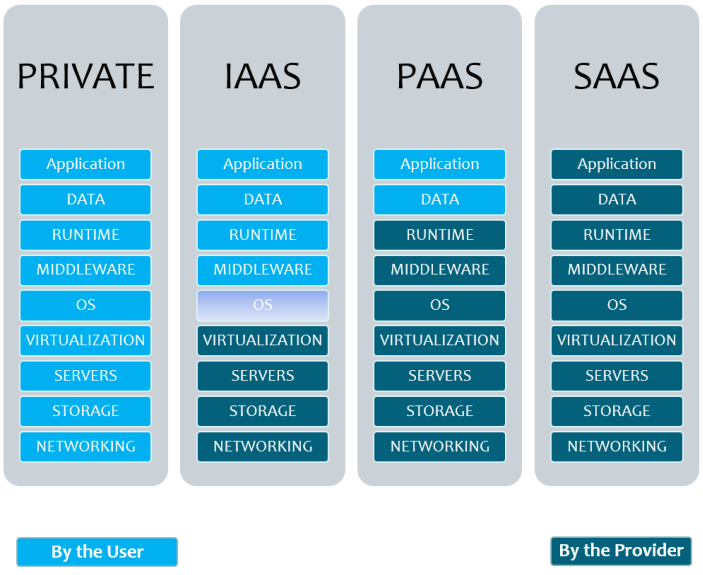


Figure 1: Cloud computing service models
(source: NIST[2])

[2] https://www.nist.gov/

Figure 1 illustrates the differences between the cloud computing service models. In the first column to the left, all computing resources are provided, managed and maintained by the enterprise itself. In IaaS model, the second column, the hardware is provided to the enterprise as a service such as networking, storage, servers. The third column PaaS model is more wider and provides the entire platform such as infrastructure services, operating systems, databases and the enterprises only required to manage the applications and their data. In the last column to the right, (SaaS model) all the components are served by the service provider.

2.1.2 Cloud Computing Deployment Models

Cloud computing infrastructure can be categorized by four deployment models the Public Cloud, Private Cloud, Community Cloud and Hybrid Cloud [1].

- **Public Cloud:** As the name suggests**,** is an open use infrastructure consisted of multiple computing resources which can be accessed by the general public over the internet. Most of the services are made available either for free or according to a "pay-as-you-go" payment model. Public cloud is flexible due to direct provision of services, the end-user can scale-up or down their utilization of computing resources on demand.

- **Private Cloud:** As the name implies**,** is a private infrastructure with computing resources owned and operated by a company or organization. Private cloud is the security and privacy solution that organizations always want in order to have the overall control of their sensitive data and services. However, an organization must build and maintain a datacenter infrastructure.

- **Community Cloud:** That type of cloud infrastructure is shared by multiple organizations who are members of a specific community, with shared concerns (e.g., policies, security, requirements, goals) and needs. Community cloud can be owned, controlled and operated by the organizations or a third-party or some combination of them.

- **Hybrid Cloud:** Is the mixture of all the above cloud computing development models well combined in order to work seamlessly together. Hybrid cloud exploits multiple benefits from the different development models. For instance, a hybrid cloud can combine public cloud's flexibility and scalability and private cloud's data privacy. All these models combinations, in certain cases, can obtain greater flexibility to businesses by providing multiple compute resources and by charging according to utilization of resources.

2.1.3 Cloud Computing Providers

There are several key players in cloud market, each offering different services and infrastructure with different advantages. Some of the most famous tech companies providing big cloud infrastructure are mentioned below.

Salesforce[3]

Salesforce is mainly a PaaS (Platform as a Service) provider for enterprises and business customers. Salesforce is a public cloud infrastructure provider, providing a platform that allows consumers to deploy customized cloud based business applications with minimum programming knowledge using a collection of tools and services through drag and drop web browser environment.

Amazon[4]

Amazon Web Services (AWS), a subsidiary of Amazon.com, is a IaaS (Infrastructure as a service) provider, offering a broad set of infrastructure services such as compute power, networking, database, storage options, available on-demand, with "pay-as-you-go" pricing model. Among services are offered by AWS are Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (S3) which belongs to IaaS service model.

Microsoft Azure[5]

Microsoft Azure is a PaaS (Platform as a Service) cloud computing platform created by Microsoft for building, deploying, debugging, testing and managing applications and services. Microsoft Azure provides a range variety of services including those for compute, data storage, networking, analytics and identity and access management.In addition, Microsoft Azure platform supports many different development tools, programming languages and frameworks, including Microsoft-specific and third-party software and systems.

FIWARE[6]

FIWARE is an open cloud-based infrastructure for cost-effective creation and delivery of Future Internet (FI) applications and services. It has been funded mainly the EU's Future Internet Public-Private Partnership programme (FI-PPP) for Internet-enabled innovation. FIWARE introduced an innovative infrastructure for cost effective creation and delivery of services on the Web. It offers an open architecture and a reference implementation of a novel service infrastructure, building upon generic and reusable building blocks referred to as Generic Enablers (GEs) which can be accessed through a public catalogue. GEs enable the rapid development of FI applications following the successful paradigm of Service Oriented Architecture (SOA) and relies REST-based communication of services. In particular, developers build applications by utilizing already deployed services offered as Cloud enablers encapsulating common functionalities (e.g. user authentication, data storage, context data management etc.), instead of re-engineering and implementing services from scratch. One of the FIWARE Lab Generic enablers is the Keyrock Identity Manager. One of the main uses of this GE is to allow developers to add identity management (authentication and authorization) to their applications based on FIWARE identity and provide access information. To achieve that Keyrock follows the OAuth[27] protocol.

[3] https://www.salesforce.com
[4] https://aws.amazon.com
[5] https://azure.microsoft.com
[6] https://www.fiware.org
[7] https://oauth.net/2/

2.1.4 Cloud Computing Virtualization of Resources

Cloud computing virtualization enables provision of computing services to a wide range of users and applications through Virtualization. It is a very important procedure because the basic idea of cloud is to provide computing resources to consumers. So how cloud providers manage the physicals resources in order to be shared among consumers? The solution to that problem is the virtualization of resources. A virtualized cloud infrastructure combines the multiple physical resources into shared pools from which consumers can receive virtual computing resources, dynamically allocated. Virtualization maximizes utilization of physical resources, separates physical infrastructure from applications, running multiple operating systems, or versions of an operating system on the same server at the same time. The cloud determines how these virtualized resources are allocated, delivered, presented to the end-users. The software that is responsible for the distribution of the cloud computing resources is referred to as Hypervisor.

The additional software layer shown in figure 2, between the virtual machines and the physical infrastructure it called Virtual Machine Monitor (Hypervisor). The Hypervisor is responsible for handling the requests made by virtual machines and for accessing resources on the physical hardware. The Hypervisor guarantees resources isolation, meaning that a virtual machine of a consumer cannot affect the operation of another virtual machine even if it crashes.

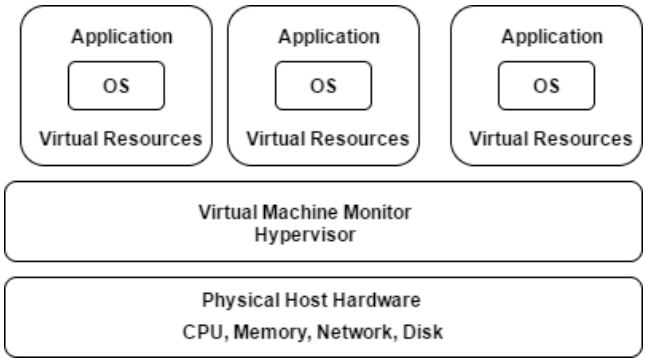


Figure 2: Physical Infrastructure – Hypervisor – Virtual Machine Relation

2.2 Internet of Things (IoT)

The Internet of Things is a network of physical objects, or “things”, embedded with electronics, software and sensors. Internet of Things generally refers to scenarios where network connectivity and computing capability extends to objects, sensors and everyday items not normally considered computers, allowing these devices to generate, exchange and consume data with minimal human intervention [4]. Nowadays, people’s everyday lives are surrounded by multiple kinds of electronic devices and machines which are connected to a network and gathering information from the environment through sensors. Companies around the globe are developing innovative devices and applications in order to ease people’s daily life. Monitoring the environment, transportation systems, health care systems and home appliances are few of the numerous aspects in which companies are focus on developing that kind of applications. For instance, health and fitness monitoring devices, watches, and even human implanted devices are connected with smartphones, offering users multiple health and wellness services with personalized data, such as the everyday calories they burned and their heart rate.

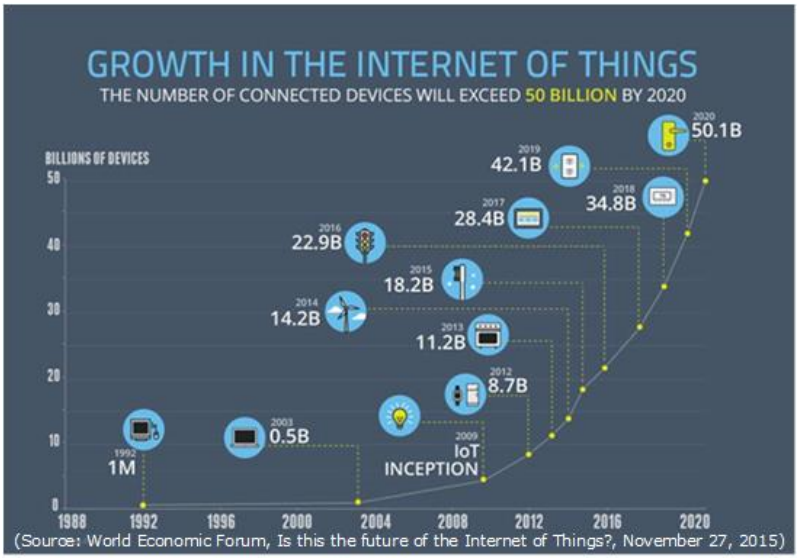


Figure 3: Number of connected devices over the years

(Source: World Economic Forum,Is this the future of the Internet of Things?[8], November 27,2015)

As shown in Figure 3, there are currently 22.9 billion connected devices to the internet and it is projected that by 2020 the number of connected devices will be 50 billion devices. It is said that a third of them will be computers, smartphones, tablets, and TVs.

Modern smartphones are equipped with numerous sensors, up to 10, able to collect information such as location, device orientation, light conditions ,temperature, environment pressure and can also enable connection and exchange information with sensors by using a wireless protocol (e.g. Bluetooth Low Energy). Collectively, these sensors produce a huge amount of data, both in unstructured form as well as structured, such as GPS or acceleration data. With the rise of the wearables, such as Android Wear or the Apple Watch smartphones play an additional role. Smart phones not only work as devices collecting data from embedded sensors but also acts as a middle-man, gateways that transmit sensor information to the internet and from there to the cloud for further processing, analysis and persistent storage.

2.2.1 Cloud Computing and the Internet of Things

There is no doubt that cloud computing is the most suitable technological environment for the development of IoT applications. The development of applications based on IoT's and cloud computing, allows these devices to continuously transmit, process and store huge amounts of the data, by exploiting the scalability of the cloud computing resources. Moreover, cloud offers a variety of big data and analytics services which simultaneous recording and processing data and generating in real time statistical results for improving applications business logic (e.g. improved decision making and optimized monitoring).

Many companies have already started to adopt cloud computing services for storing, processing and analyzing data from IoT devices, because the cloud data services helps companies to gather data across from all their domains, internal or third-party sources, in order to form a real-time view of the business picture. All the data is remotely centralized stored instead of being stored

[8] https://www.weforum.org/agenda/2015/11/is-this-future-of-the-internet-of-things/

locally on companies' machines so it reduces drastically the time was spending on gathering data from all domains in order to generate the reports. Cloud also reduces the infrastructure costs by automatically store all data in cloud. Overall, cloud computing makes businesses run more efficiently on managing their IoT data.

2.3 Service-Oriented Architecture (SOA)

The Service-oriented architecture is based on the logic that any large complex problem can be divided into separate smaller problems, simple enough to be solved. The solved sub-problems are effectively combined in order to solve the original problem. SOA is an approach for creating a software architecture or system as a collection of loosely coupled communicating services (typically, Web services that can connect and exchange XML information over Web using an HTTP protocol). This has the advantages of software re-usability making it possible to use, replace or upgrade individual services in a plug-and-play way and without the need to start application development from scratch. This is offers a competitive benefit to businesses by saving time and money for application development. One requirement of the SOA style is to provide meaningful descriptions for web services so that software applications can understand their features and learn how to interact with them. WSDL[9] is the first W3C[10] standard widely used for publishing service descriptions [5][25].

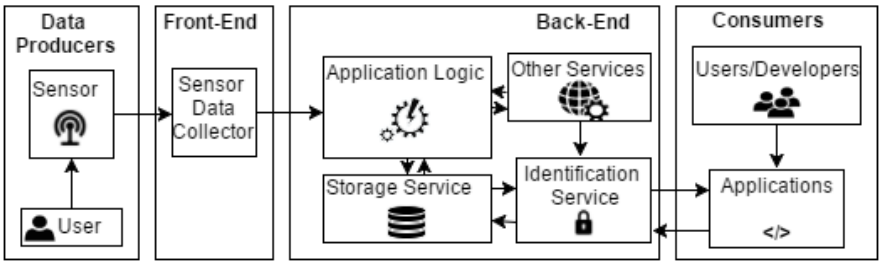


Figure 4: IoT General Service-Oriented Architecture

A general example, of an IoT system following SOA is given in figure 4. The system is divided in four major parts. The data Producers are the producers which acquire data using sensors and interact with users and the cloud. The Front-End is the service endpoint and plays the role of a gateway by receiving data from the sensors and transmitting data over the internet to application's Back-End. The Back-End runs on the cloud and combines all system entities, data process, storage and authentication services which work as a whole by providing services to the Front-End. All these services communicate with the Application Logic which is the crucial part of the system in which all the necessary logic rules and conditions on how data will be stored, changed and accessed are included. In addition, Application Logic service orchestrates the transferring of the information to the appropriate individual services (storage, identification and information manager) and implements application intelligence for handling context events (e.g., a rule based system) or decides whether the consumers must handle these events. The consumers-end refers to application and end users that communicate with the cloud application programming interfaces (API's).

[9] https://www.w3.org/TR/wsdl
[10] https://www.w3.org/Consortium

2.4 JSON (JavaScript Object Notation)

JSON[11] is a lightweight data format which is human and machine readable text in a way that humans can read and write it and machines parse and generate it. JSON is a text object derived from the object literals of JavaScript and consisted of key/value pairs. The key is always a string, the value can be a string, number, boolean, array, object or null. JSON also supports nested key-value pair inside a key-value pair. It's the most common data format for asynchronous server and web application communication. JSON is also the most common data format that modern web services use for exchanging information [6]. Some advantages on using JSON to transfer data over the internet are listed below:

- Loaded asynchronously much more easily than XML[12].
- Light-weight format that is easier to load and parse than XML.
- Human and Machine readable with more clear structure compared to XML.

2.5 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It is the basic part of the Service Oriented Architecture (SOA) because multiple web services can communicate with each other to work as a whole system. A Web service has an interface described in a machine-processable format (specifically WSDL[13]) and with this description other systems can interact with the Web service and exchange messages typically using HTTP with a data format such as JSON. REST (Representation State Transfer) is the most popular architecture style of web services [7][8].

2.5.1 HTTP (HyperText Transfer Protocol)

HTTP is a communication protocol. It is the most common underlying communication protocol used by World Wide Web and defines how web browsers exchange data between a web server and a client. When a user access any web page by entering a URL, this actually sends an HTTP command to the server to fetch and transmit a web page. Nowadays several new technologies are using HTTP such as Java[14] and JavaScript[15]. Basically, HTTP is a TCP/IP[16] based communication protocol, that is used to deliver data (HTML[17] files, image files, query results, etc.) on the World Wide Web [9]. HTTP is so powerful protocol for three main reasons:

- **HTTP is connectionless:** After a HTTP request from a client to server is made, the client disconnects from the server and waits for server's response. The server processes the request and re-establishes a connection with the client, in order to send the response back to the client.

[11] http://www.json.org/
[12] https://en.wikipedia.org/wiki/XML
[13] https://en.wikipedia.org/wiki/Web_Services_Description_Language
[14] https://java.com
[15] https://en.wikipedia.org/wiki/JavaScript
[16] https://en.wikipedia.org/wiki/Internet_protocol_suite
[17] https://en.wikipedia.org/wiki/HTML

- **HTTP is media independent:** Any data format can be sent over HTTP protocol if the client and the server know how to handle the data format. It is required for client requests and server responses to indicate with a header the data content type. Most common types are JSON , XML[18] and text.
- **HTTP is stateless:** each command is executed separately, without any information about previous commands. The server and client are aware of each other only during a current request.

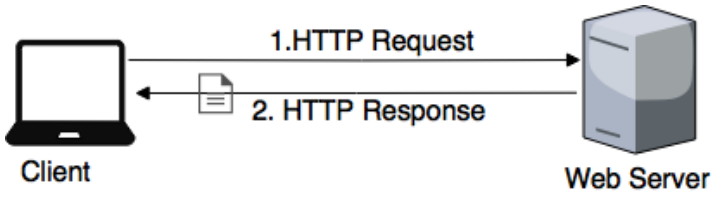


Figure 5: Client-Server HTTP Communication

As shown in figure 5, the client sends a request using a request method to server, that request contains URI, protocol version, client information and content. The server responds with a response message which contains a status code, server information and possibly some content. The request method shows how the resource is going to be accessed and processed. The URI is a Uniform Resource Identifier and identifies the resource in which the request is performed.

The four main methods defined by HTTP protocol are listed below:

- **GET**, when the client requests data from a specified resource
- **POST**, when the client sends data to the server to be processed and inserted to a specified resource.
- **PUT**, when client sends data to a server into an existing resource, replaces the resource currently exists on the given URL with a new resource. Also with this method, if a resource is not existing on a given URL can create a new resource like POST.
- **DELETE**, when the client wants to delete a particular resource from the server.

HTTP protocol uses the internet as everyone knows it and sometimes errors might occur. The HTTP protocol provides the server with numerous status codes and error messages for every asked HTTP client request. For instance, when a user tries to access a webpage that either not exists or has been deleted an error message occurs in browser saying “404 File Not Found”. These errors called HTTP status codes and they are response codes given by servers to help users to identify the cause of the problem [9]. These code numbers are divided into five broad categories:

- Informational - 1XX
- Successful - 2XX
- Redirection - 3XX
- Client Error - 4XX
- Server Error - 5X

[18] https://en.wikipedia.org/wiki/XML

2.5.2 REST (Representational state transfer)

Representational state transfer (REST) is an architectural style for distributed hypermedia systems such as the World Wide Web. To implement this RESTful architecture there is a set of guidelines called Resource-Oriented Architecture which are the rules to be followed for designing RESTful web services [10]. The REST architectural style is based on the following basis: Client-Server Communication, Resources, Addressability, Statelessness, Connectedness and Uniform Interface [10] [11].

Client-Server Communication: All applications built in the RESTful style must be follow the client-server principle shown in figure 5. HTTP is the most common client-server protocol used to implement RESTful architectures.

Resources: A Restful web service provides a set of resources which identify the targets of the interaction with its clients. Resources can be physical or abstract objects (cars, objects, customers, pictures, etc.). Each resource is identified by URIs, which provides a global address space for resource and service discovery. If an object is not exposed with a URI, it is not a resource and it is not accessible through the Web. Resources can be represented with many format types, such as JSON.

Addressability: Every resource should be addressable with URI, so there are at least as many URIs as the resources. Multiple URI's may refer to the same resource. The idea is that every web service resource is reachable through a unique identifier. A clear and reasonable addressability for resources keeps a web service simple and understandable. Each HTTP request contains the URI of the resource object and the format of a URI is standardized as follows:

http://host:port/path

The host is the IP address followed by the port which represents the location of the resources on the network and at the end is the path consisted of segments of "/" character and represents the directory path where the resource can be accessed. Furthermore, a URI links to a specific resource and the web service publish the data of the resource

Statelessness: All HTTP requests should happen in a complete isolation from each other. The RESTful web service should never rely on information from previous requests, so the client must present all the necessary information required to make a request. In case that some data from a previous request is needed, the client must send it again to the server because the server which hosting the web service does not keep any information about client's previous requests. This makes the system more reliable, simpler and scalable.

Connectedness: A Restful Web service provides to client links from one state to another inside each resource representation. Representations are hypermedia that contain links (URI's) to other resources, that offers a well-connected web environment. Clients can discover the web service interface by visiting the embedded links in each of its resource.

Uniform Interface: The client interacts with the server through HTTP methods such as GET, POST, PUT, DELETE. These methods define how a resource can be accessed and processed.

- **GET:** the client sends a request using this method to retriever the information identified by the given Request-URI which is the representation of a resource. GET method is called a safe method, because it cannot change the state of the resource, it can only retriever a resource from the server. So, multiple requests can be performed having the same response as a single request. A successful request will return a resource as a response consisted of a message body and a response code status 200 (OK). A non-successful request will send as response an error status code such as 404 (Not Found).

- **POST:** the client sends a request using this method to create a new resource to the given URI with the data enclosed in the request. When the Request-URI in not linked to an existing resource, the new resource is created with response status code 201 (Created) which indicates that the request has been fulfilled and the new resource has been created.

- **PUT:** the client sends a request using this method to modify and update an existing resource on the given Request-URI, with the new data enclosed in the request. If the URI refers to an already existing resource, then the resource is updated with the new data with a response status code 200 (OK) indicating the successful completion of the request. However, if the URI refers to a no existing resource then the PUT method acts as a POST and creates the new resource on the requested URI, with the given data in the request with a response code status 201 (Created). Both POST and PUT can create a new resource but the main difference is that with the PUT method the client indicates the id of the resource but with POST method the id of the resource is automatically generated. Furthermore, multiple same PUT requests will have the same result as a single request, as well as the GET method, but with POST each request will create new resource.

- **DELETE**: the client sends a request using this method to delete an existing resource on the given Request-URI. The server deletes the resource identified by the request. As GET and PUT, the DELETE method is also an idempotent method so executing a multiple same request will have the same result as execute a single request. A successful response has a status code 200 (OK) meaning that the action has been performed and the resource has been removed.

Below is a table summarizing the primary HTTP methods in combination with the CRUD operations (Create, Read, Update, Delete) on a resource at a given URI.

HTTP Method	CRUD	Description
POST	CREATE	Create a resource
READ	GET	Retrieve the current state of resource
UPDATE	PUT	Initialize or Update a resource.
DELETE	DELETE	Delete a resource. The resource at the given URI is no longer exist.

2.5.3 API (Application Programming Interface)

An API (Application Programming Interface) is a software application consisted from multiple software components such as web services which provides a solution to a problem and can be accessed through the internet. Providers of services offer APIs interfaces so the public can access the services and developers can easily design applications using those API's. In addition, APIs allow a program to interact with another without involving the user. For instance, when a web application offers the user the ability to login via his Twitter account, the web application uses the API provided by Twitter for user identification. In other words, we assign the identification of the users of our application to an external service to provide additional security, reliability and convenience to the user.

Modern software applications are typically built on several APIs. An API could be seen as a problem abstraction which specifies how software components should interact with other software components that provide a solution to that problem. The main purpose of an API is to provide a clear and simple logical interface to a component and its functionality, while hiding unnecessary implementation details. An API that is difficult to be understood and used by developers leads to decreased programming productivity which contradicts the purpose of APIs to increase efficiency.

HTTP is the most commonly used application protocol in conjunction with REST architecture so modern public API's are REST API's and they can be used by practically any programming language and are easy to test.

2.5.4 Jersey – RESTful Web Services in Java

Jersey is a REST open source framework for developing RESTful Web services and is the reference implementation of Java[19] JAX-RS, a Java API for RESTful Web Services, which uses annotations such as @GET, @Produces @Consumes to define the REST relevance of Java classes. Jersey provides its own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful services and client development. Jersey implement Restful Web services in a Java servlet container. This servlet handles the HTTP requests and selects the correct class and method to respond to this request. A REST web developed with Jersey is consisted of data classes for the resources and services classes [12].

2.6 Mobile Application Development

There are three distinct mobile application development approaches the native, the web and the hybrid. Each one of them has numerous advantages but also some disadvantages. Thus, a developer must choose wisely among these approaches in order to develop a mobile application because different types of applications suits best with different development approaches.

[19] https://java.com

2.6.1 Native

Native mobile applications are written in the native language of the platform (Java for Android, Objective-C for iOS). Native applications have direct access to native functionality and services such as camera, geolocation, address book through API's provided by the platform's SDK. The main advantage of native mobile applications is the fast performance and the high reliability. However, this kind of applications are expensive to be developed because are close source and proprietary, and thus do not work on any other operating systems. So, companies must develop different versions of the same application for each mobile platform. In general, developers choose this approach when they need to build a mobile application with high performance, 3D graphics and complicated UIs [13] [14].

2.6.2 Web

Web mobile applications are accessed from the device's web browser and usually are developed with web technologies such as HTML, CSS and JavaScript. Basically, they are not real mobile applications, they are websites that are adjusted to function on mobile devices. The main advantage of web mobile applications is that only one version of the application is developed and can be accessible from all the mobile platforms through a web browser. However, this type of mobile applications can only be accessed over the internet because they are not installed locally on the device so in places with a poor or without Internet connection the application cannot be used. Another drawback of the web development approach is the lack of integration with the native functionality and services of the device. In general, developers choose this approach when they need to build a mobile application compatible with multiple platforms, with no need to access native device services and it is a non-intensive computing application [13] [14].

2.6.3 Hybrid

The hybrid mobile applications combine the advantages of the native and web mobile applications. Hybrid applications are installed on the device as the native applications and are developed using a mixture of web technologies such as HTML[20], CSS[21] and JavaScript[22] hosted inside the application's WebView[23] (an embedded browser window that runs within the application). So, Hybrid applications are comprised of two parts, the web application and a wrapper that opens the WebView and provides JavaScript APIs for mobile platform's specific services such as an accelerometer or camera. Some of the top companies are having hybrid mobile applications instead of native, like Apple's AppStore, Twitter and Gmail [13] [14].

The reason why hybrid applications are popular is the portability to different platforms, meaning that the same HTML, CSS and JavaScript code components can be run on different mobile platforms without modifications for each platform. This, reduces the development costs. In addition, web developers can easily develop mobile applications without being familiar with the peculiarities of each vendor or mobile platform or its programming language and environment.

[20] https://www.w3schools.com/html/
[21] https://en.wikipedia.org/wiki/Cascading_Style_Sheets
[22] https://en.wikipedia.org/wiki/JavaScript
[23] https://developer.android.com/reference/android/webkit/WebView.html

Moreover, companies build hybrid apps as wrappers of their existing web app because only few code components changes are needed.

The most famous hybrid mobile application development framework is the Apache Cordova which let the developers to build an application using existing web technologies that can run to more than one platform just by changing a line or two of code. Apache Cordova can be seen as an abstraction layer between the business application code and the operating system. It provides numerous of API's through JavaScript to do basic device manipulations on the operating system level such as accessing the camera and GPS. Apache Cordova runs the CSS3[24], HTML5[25] and JavaScript code inside a single WebView[26], which renders the web content inside the native application. HTML5 can provide access to device's native hardware such as the accelerometer, camera, and GPS. However, HTML5 is a newer technology and a lot of mobile browsers, especially older versions do not support HTML5 [18]. To bypass this problem, Apache Cordova embeds the HTML5 code inside the native WebView on the mobile and provides a range variety of native plug-ins through JavaScript API's such as camera, file system, microphone which communicate with HTML pages, allowing developers to add more functionalities.

The major problem with hybrid applications is their performance. If a hybrid application is not designed properly using best practices, the powerful and compute demanded JavaScript engines might slow down the application response time and the overall performance.

2.7 Web Application Development

This section will describe web technologies used in modern browser-based applications that take benefit from the powerful features of modern web browsers.

2.7.1 Single Page Architecture (SPA)

When a web application follows Single Page Architecture, it means that the application fits on a single web page, providing a smoother and more fluid user experience. Users don't have to reload the Web page in order to get the latest updates, because the user interface is composed of individual components which can be updated/replaced independently. At the beginning the Web browser sends a request to server and receives the Web page (HTML documents). Afterwards, on every single user interaction, the user interface will stay responsive, while new data transferred from the server and displayed to the user without re-rendering the whole Web page. All the calls to the server are performed asynchronously to the application. JavaScript frameworks, such as AngularJS have adopted SPA principles. A web application developed according to SPA architecture is referred as Single Page Application [15]. An example of a Single Page Applications is Gmail.

[24] https://www.w3schools.com/css/css3_intro.asp
[25] https://www.w3.org/TR/html5/
[26] https://developer.android.com/reference/android/webkit/WebView.html

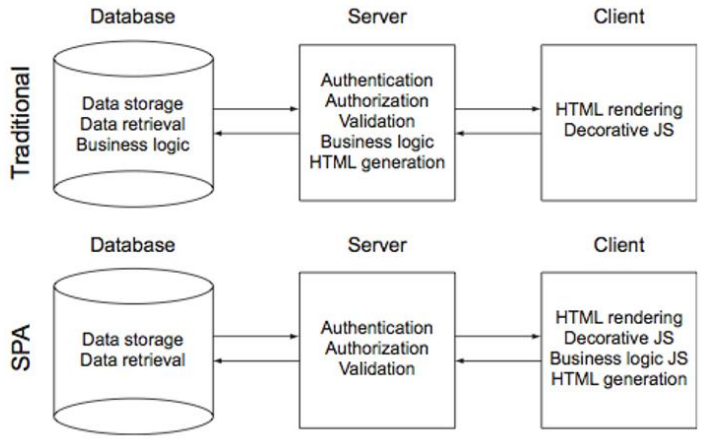


Figure 7: Traditional Web Application Architecture and Single Page Web Application Architecture

The figure 7, shows a traditional web application architecture (server-side application) and a single page application architecture (client-side application). In the first one, JavaScript is mainly used for decorative functions and basic data validation. The web server is responsible for providing the application logic and the user interface (HTML). In the second one, after the SPA application is started a huge part of application logic and HTML documents are stored to the client side. The Client can keep track of the application state (HTML template) without asking the web server on every user interaction and only asks the server when data are needed to be transferred. The independent nature of SPA application can support offline functionality, because relevant data stored in the browser's local storage and then synchronized with the server when connection is again available. However, there are still areas where data processing is performed on the server side for security reasons such as user authentication and authorization.

2.7.2 AngularJS Framework

AngularJS is an open-source, client-side JavaScript MVC framework provided and maintained by Google, for building dynamic single-page web applications that take advantage of the features of modern browsers and devices [16]. MVC (Model-View-Controller) is a design architecture pattern which separates the different components used in application development to individual modules. These modules are developed separately and the interaction between the modules is limited. The MVC separates the business data (Models) from the user interface (Views), with a third component (Controllers) which manages application logic. Separating the application logic from the user interface ease of code reuse and maintenance. MVC design pattern improves the overall application's organization and structure. The three basic components of MVC are described below [16] [17]:

- **Model**: Represents the data model of an application. Model objects are very simple data objects.
- **View**: Provides the user interface of the application. All user interactions such as pressing buttons, are triggered on the View. The View represents the visual current state of Model objects. When a Model changes, the View receives a change event in order to update the View.

- **Controller**: Contains all the application logic and manages the user interface. All the user interactions from the View are forwarded to the Controller. The controller takes decisions on what is shown in the user interface (View). The Controller acts as an intermediate between a model and a view of the application. It receives events for the user's actions (inputs such as mouse and keyboard) and retriever's data from the underlying data storage system (a database).

AngularJS extends the capabilities of HTML by custom elements, attributes and classes called directives in order to deliver rich dynamic webpages. A directive is something that introduces a new HTML syntax. Directives are markers on a DOM (Document Object Model) element which gives a special behavior to it or even transform the DOM element. Another great feature of AngularJS is the scope object refers to application model and is the way to pass data from a controller to a view. This allows the directives to be notified of property value changes, which allows the directive to render the updated value to the DOM. Moreover, Angular's JavaScript components complement Apache Cordova, the framework used for developing hybrid mobile applications.
The main features of AngularJS are presented below:

- Two-way data binding between models and views. It binds a background data object with a user interface object. Any change to the background data object is automatically copied to the user interface object and vice-versa. This eliminates DOM manipulation.
- MVC patter support that helps to separate the application into three different areas, Model (the data), Controller (the logic rules that operates on data) and the user interface (View).
- Built-in services for RESTful server communication.

Mainly AngularJS is a framework which is used for the development of complex and high performance client-side applications with a clean MVC architecture. This kind of web applications are easily maintainable, testable and extendable.

2.8 Apache CouchDB

CouchDB is an open source, schema less document database management system which is completely suitable for the Web. CouchDB server hosts multiple databases which store JSON documents and each document has a unique Id in each database. It provides a well-structured REST HTTP/JSON API for accessing and processing the database documents, according to CouchDB API documentation [18].

To retrieve document, a user can send a simple get request with curl[27] to the CouchDB database:

```
curl -X GET http://host:5984/database_name/document_id
```

A user can simply create a database by sending a PUT request with curl to the CouchDB server:

```
curl -X PUT http://host:5984/database_name
```

[27] https://curl.haxx.se/

Moreover, CouchDB provides Map/Reduce functionality and is used to combine and sort out the result from multiple documents by a given key or expression.

CouchDB security imposes powerful security mechanisms on data and operations. Access to a database is granted only to authorized users [19]. There are three types of users:

- Server Admins, can do anything across the entire CouchDB server. This includes creating/deleting databases, managing users, and full admin access to all databases.
- Database Admins, have full read and write access only on a specific database and can also modify security settings on the specific database.
- Database Readers, can only read documents and views on a specific database, and have no other permissions.

Each database has its own security document where it stores the users who have access to the database. Thus, a database in a CouchDB server can be completely secured and isolated from other databases and users, improving this way data confidentiality [20].

Chapter 3 – CloudFarm Design & Implementation

This chapter focus on system design and implementation. The discussion starts with the analysis of functional and non-functional requirements of farm and crops procedures followed by the mapping of these requirements to a series of UML diagrams including activity (workflow), deployment and architecture diagrams capturing all aspects of system design. At the beginning of the thesis several farmers and agronomists were interviewed in order to capture their needs according to their intents on system usage. Differences in their viewpoints are mapped to different UML diagrams per user type (i.e. farmers, agronomists and system administrators).

3.1 General CloudFarm System Design

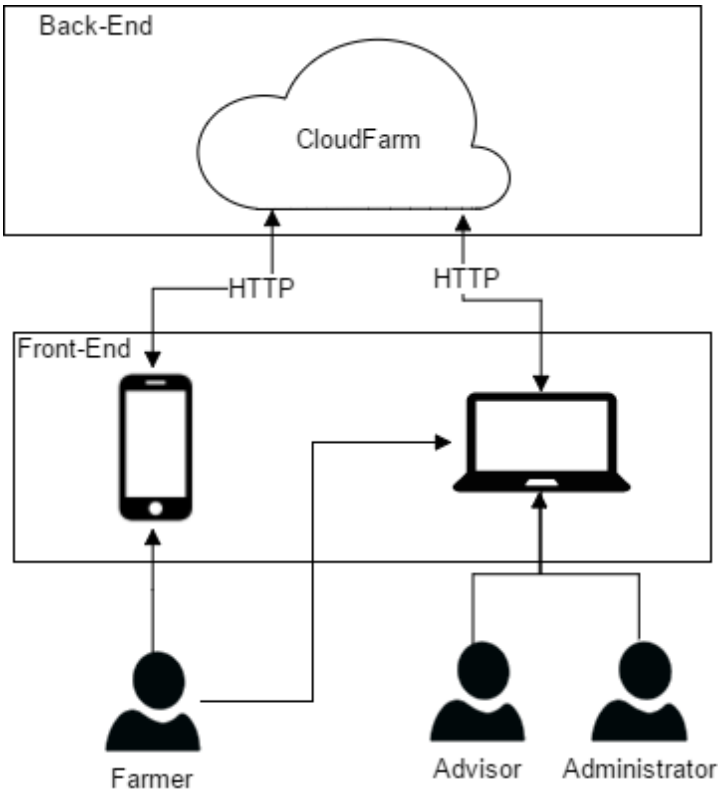


Figure 10: General CloudFarm design idea

Users can interact with the CloudFarm system running in the cloud using a mobile application that runs on a smart-phone or tablet and a web application that runs on the cloud, as figure 10 illustrates. The web application provides a digital dashboard allowing users to monitor input data. On the cloud, farmers can view reports from agriculture activities, advisors can supervise multiple farmers by monitoring their farm data, and administrators can manage users and keep the database updated. The mobile application is only used by farmers when a farm activity (e.g. spray, harvest, fertilization) takes place.

3.2 User Roles

- **Farmers**: can control the overall farm condition through the Web application and also input information about farm activities (e.g. spraying, fertilizations, cropping) performed at a specific point in time using the mobile application.
- **Advisors**: monitoring the data of a farmer through the web application.
- **Administrators:** they are responsible for the stability, maintenance and the management of users of the CloudFarm.

3.3 Functional Requirements

In general, CloudFarm offers a variety of features such as Crop Management, Field Management, Storehouse Management and a Plant Protection.

The main features of CloudFarm are:

- Capture information directly from the filed during an agriculture activity using a mobile device. A GPS confirms the location of the performed activity.
- It is fully functional in agriculture areas without a network connectivity.
- Complies to Crop, Field and Storehouse Management procedures according to Good Agriculture Practices directives.

System functionality is exposed to farmers in two ways. Either via a Mobile Application running on a mobile device (smartphone or tablet) or via a Web interface running on the cloud. The former is intended to be used by farmers while on the field while the latter is intended to be used for accessing data online in the cloud. The main difference between the two applications is that the Mobile application can work in two modes i.e., either online as the Web application does and also offline in case of loss of internet connection in which case all data are cached locally and synchronize with data stored persistently in the cloud when internet is back.

Functionality for farmers on Mobile Application:

- **Create crop** (sowing): Registers the process where farmers plant seeds into an empty field. The plants category and species are selected from a catalogue provided from a Plant Protection Service of the Greek Ministry of Agriculture.

- **Harvest crop**: Registration of the crop work (e.g., harvesting vegetables from a field) and registering weights or units. Harvest is allowed only if the minimum waiting date after the last spraying is over. Otherwise if there isn't any spraying activity recently performed the crop can be harvested.

- **Spray crop**: Registration of spraying work (for protection from harmful insects, or diseases). The allowed chemicals are selected from the catalogue of the Greek ministry of Agriculture.

- **Fertilize crop:** Registration of the process of using natural general type fertilizer or synthetic new-type fertilizer essential to the growth of plats.

- **Remove crop:** Removes the crop and the field turns inactive which signals the end of the crop season.
- **Insert supply to Storehouse:** Registration of purchased supplies such as fertilizers or chemicals from the list of the Ministry of Agriculture.
- **Stock supervision**: View available quantities of supplies in the storehouse.
- **Login into CloudFarm**: A user can log in to CloudFarm with a FIWARE account.
- **Registration:** Upon first successful login into CloudFarm using a FIWARE account, the user is prompted to fill up a registration form in order to become known as a CloudFarm user.

Functionality for farmers in the cloud (exposed via Web Application):

- **Crop Management**:
  - Easy access to Crops Details using a Map with fields.
  - View Crops History.
  - View detailed reports about crop's overall harvests, sprayings and fertilizations.
- **Storehouse Management**:
  - View supplies availability.
  - View supply consumption.
  - View detailed supply consumption reports on where and when fertilizers or sprays have been used.
- **Fields Management**:
  - View fields on the Map.
  - Insert or delete a field.
- **Advisors Management**:
  - View their advisors.
  - Accept a pending supervision advisor request.
  - Remove an advisor from advisors list.

Functionality for Advisors in the cloud (exposed via Web Application):

- View overall farm condition (Crops, Harvests, Sprayings and Fertilizations) per farm and per field.
- View Storehouses with details and consumption of supplies per farmer and per field.
- Start advising a farmer by sending an advising request to farmer.
- Stop advising a farmer.

Functionality for system administrators in the cloud (exposed via Web Application):

- Manage Pending Users: Decline or approve pending request.
- Manage Users: Add User, Delete Users, View Users.
- Add, Update Plant Protection Products Details such as chemical dosage per crop and chemical packages.
- Manage Fields: Add or Remove a farmer's Field.

3.4 Use Case Diagrams

UML[28] Use case diagrams provide a good way of getting an overall picture of what is happening in the system. The use case diagram is an effective means of communicating with users and other stackholders about the system and what is intended to do [22]. Below there are presented four use case diagrams which represent sequences of actions carried out by users of the system that interact with the system.

Figure 11, summarizes farmer's functionality accessible in mobile application.
Figure 12, summarizes functionality for farmers accessible via a Web interface running in the cloud.
Figure 13, summarizes functionality for advisors accessible on web application running on cloud.
Figure 14, summarizes functionality for systems administrators accessible via Web Interface. running in the cloud.

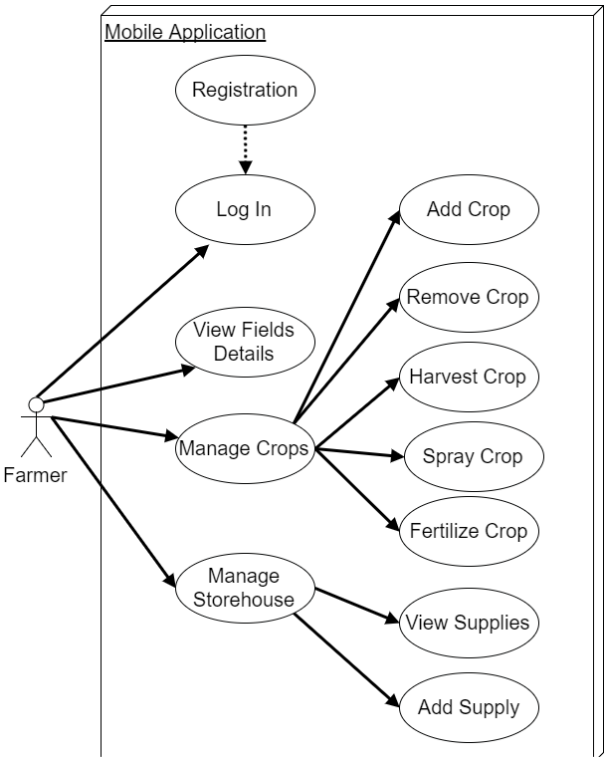


Figure 11: Farmer's Mobile Application Use Case Diagram

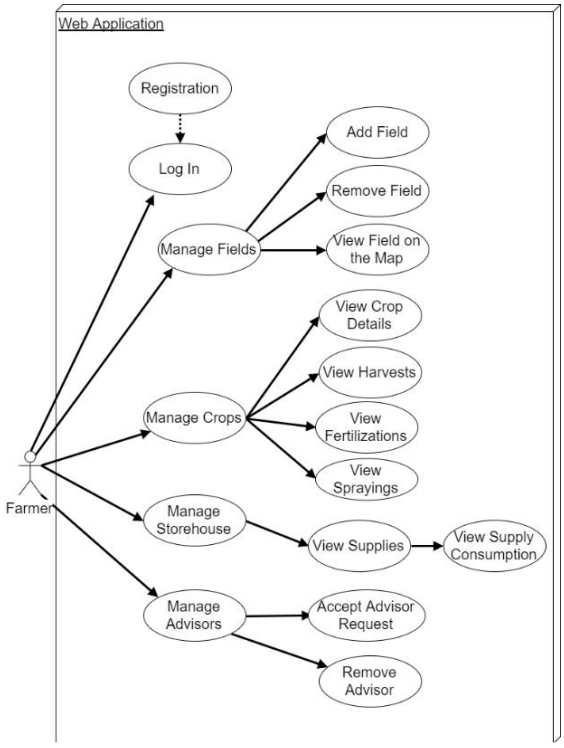


Figure 12: Farmer's Web Application Use Case Diagram

[28] https://en.wikipedia.org/wiki/Unified_Modeling_Language

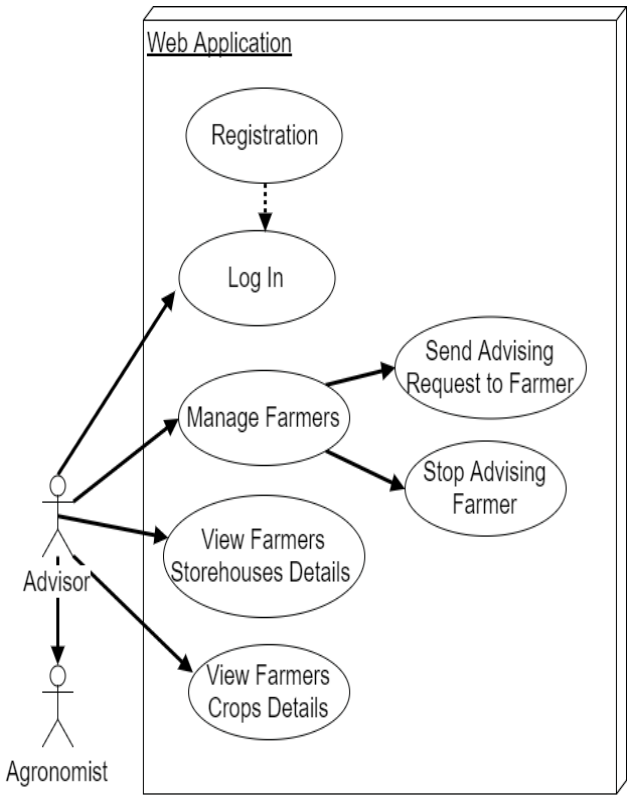


Figure 13: Advisor's Web Application Use Case Diagram

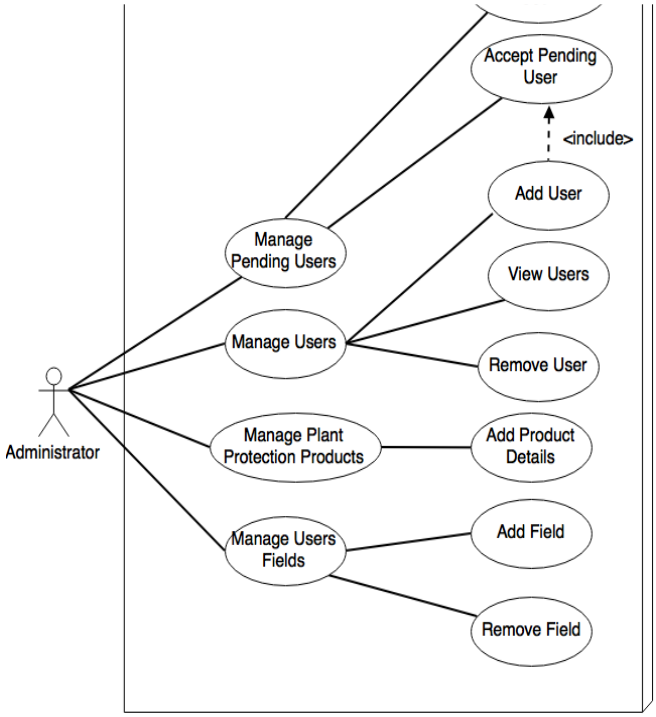


Figure 14: Admin's Web Application Use Case Diagram

3.5 Activity Diagrams

UML Activity diagrams are the means of describing graphical an executed set of procedural system activities. Activity diagrams can be used to describe the complexity of use cases at a detailed level. At the design level, we used activity diagrams to describe in detail the details of the activities presented in the use case diagrams [22].

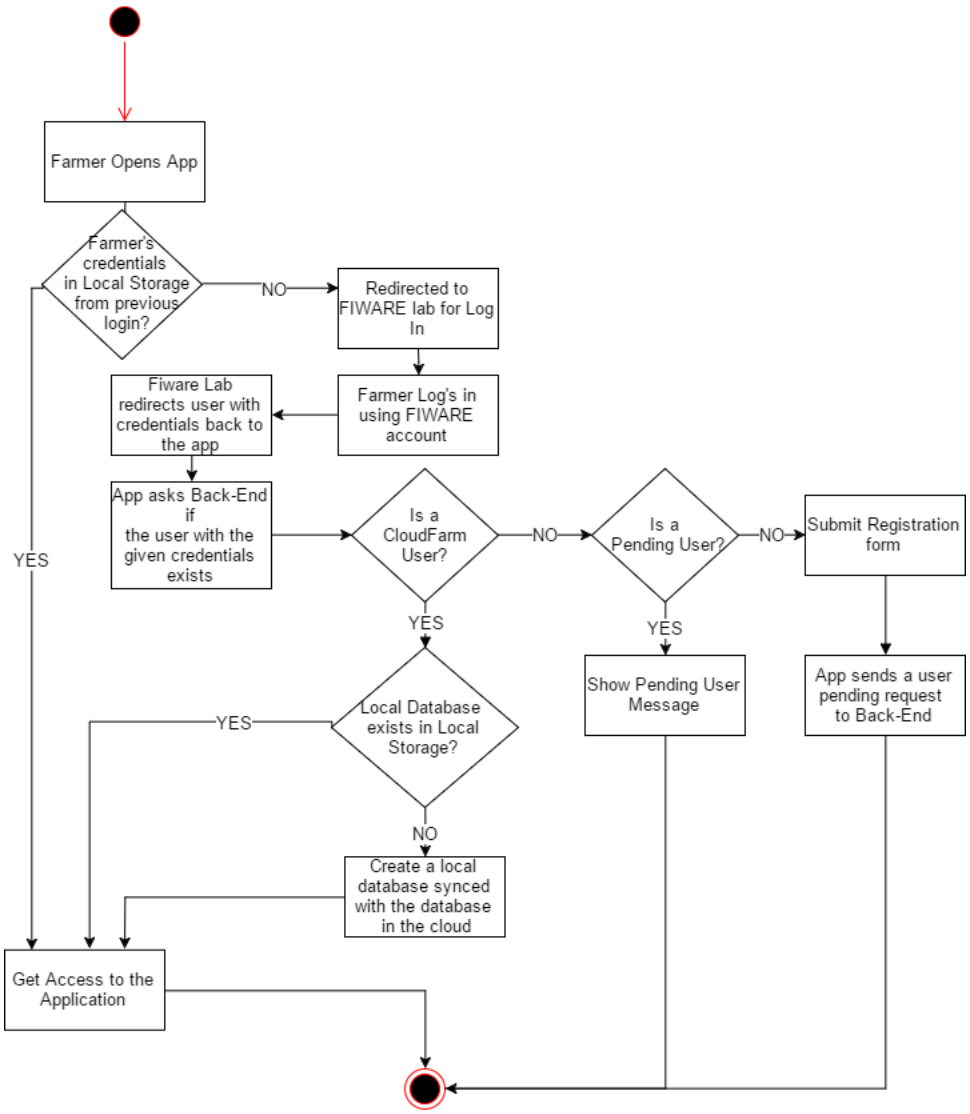


Figure 15: Activity Diagram – Login into CloudFarm via Mobile Application.

Figure 15 illustrates, the log in process of a farmer in the CloudFarm mobile application. First the farmer opens the mobile application and if he has logged in before then all the needed user credentials to access the CloudFarm system exist in the mobile application and the farmer can access the application. If the farmer has not logged in before, the application is redirected to Keyrock Identity Manager (FIWARE Lab) in order to login, and then Keyrock sends back to the application the user credentials. The application sends a request to CloudFarm among with the given user credentials to authenticate, if the user that logged in with FIWARE account, is a

CloudFarm user or not. If the user is a CloudFarm user, then the farmer is ready to use the application. Else, if the user in not a CloudFarm user, must submit a registration form and the application will send a request in the cloud to add user as a pending user. Else, if the user is a pending user waiting for approval of CloudFarm Administrators, a message will inform the farmer that his registration is not yet approved.

In figure 16, shows a business logic workflow, which describes how crop activities are performed using mobile application. First the farmer must log into the CloudFarm system using FIWARE account, as described in figure 15. Then the farmer selects the crop management in order to perform a crop activity (e.g. harvest, spraying, fertilization) and the mobile application uses a geolocation service which returns the location of the device (farmer's location). The application process farmer's location and detects if the farmer is located inside a field (if the geolocation service returns wrong location the farmer can choose the field manually). If the farmer is located inside field, the application checks if there is an active crop, in which case the field has crop then all crop activities are displayed as options, otherwise the farmer is prompt to create a new crop activity. In the Add Crop activity the farmer can only select plant species according to a list from the Greek ministry of Agriculture, and them fulfils the required steps as shown in the figure 16, and the crop is submitted and stored in to the system. In a Crop Spraying activity only permitted chemicals for this specific crop according to the Greek ministry of Agriculture can be selected from a list. The chemical dosage calculated automatically by multiplying filed acres with the chemical dosage per acre. The minimum harvest date after a spaying is considered the current spraying date plus some waiting days without harvest due to spraying (the waiting days are provided along with the chemical details at the selection step). The number of waiting days depends on crop type. In a Crop Fertilization activity, the farmer can use two types of fertilizers. For the new type fertilizers, the farmer is prompted to select the fertilizer from a list of new type fertilizers according to the Greek Ministry of Agriculture. Because the Greek Ministry of Agriculture is not providing a list for the general type fertilizers, the farmer fulfils manually the fertilizer's name.

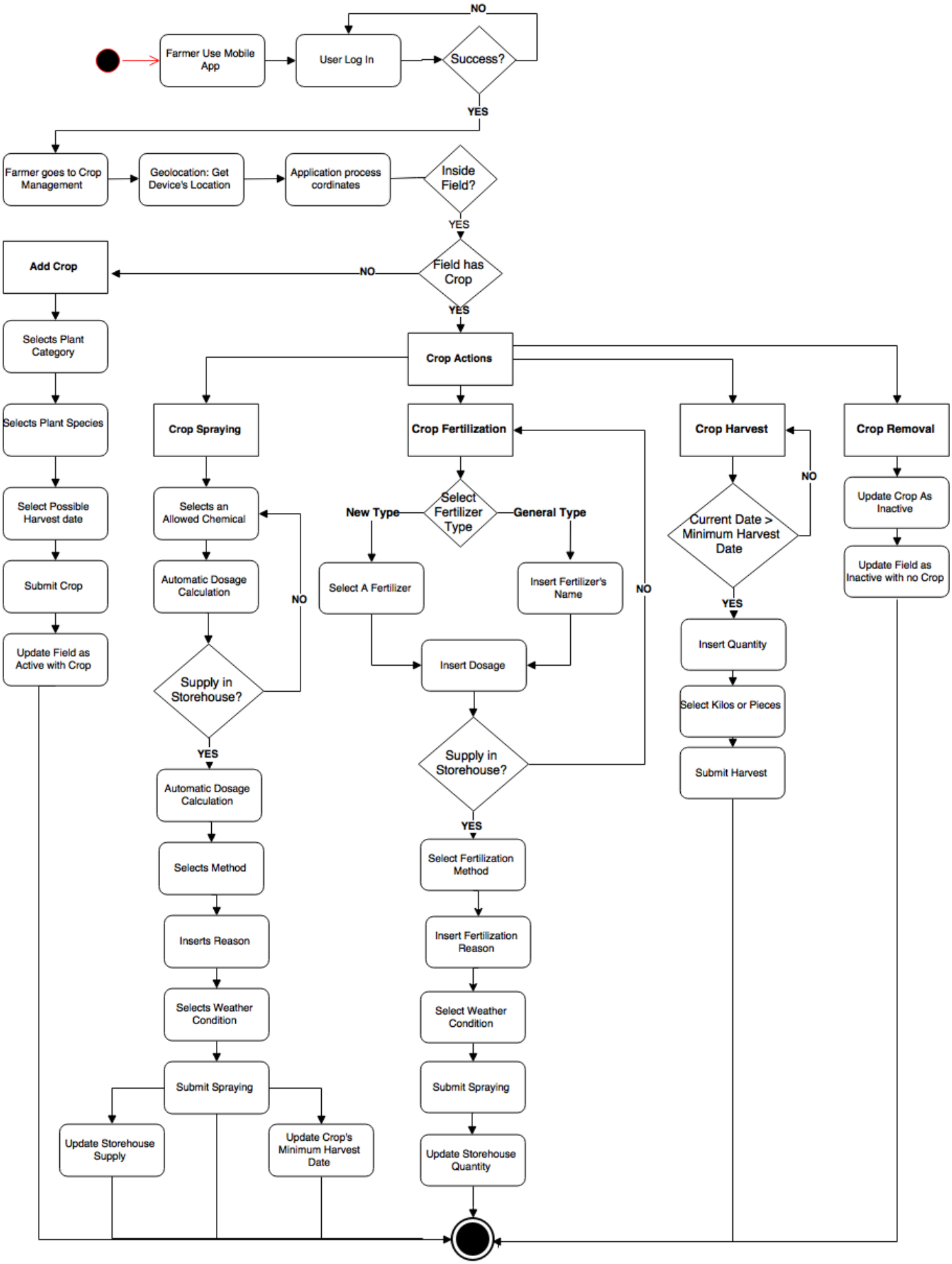


Figure 16: The farmer Mobile Application Activity Diagram

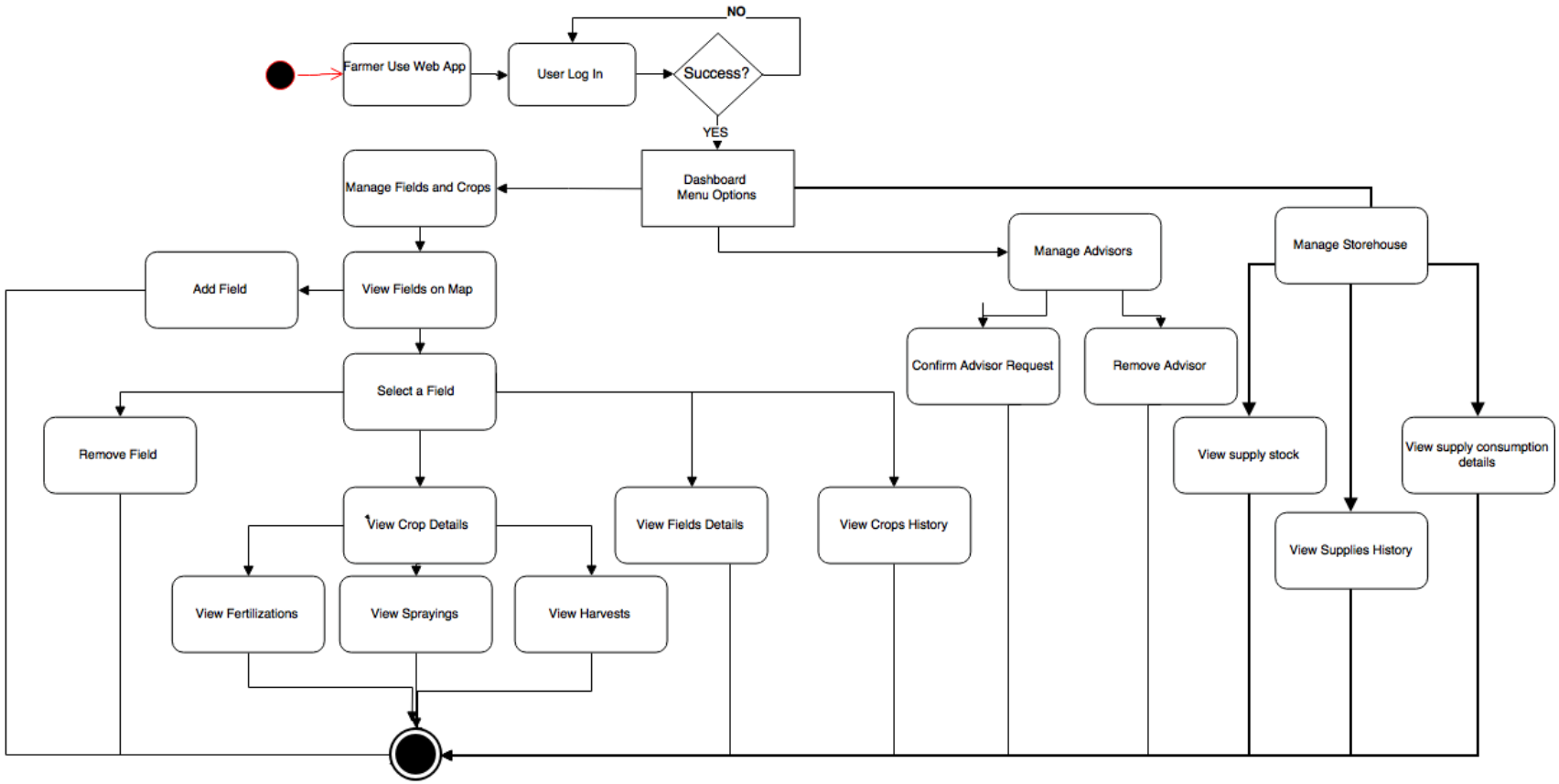


Figure 17: Activity Diagram – Farmer's functionalities via Web Application

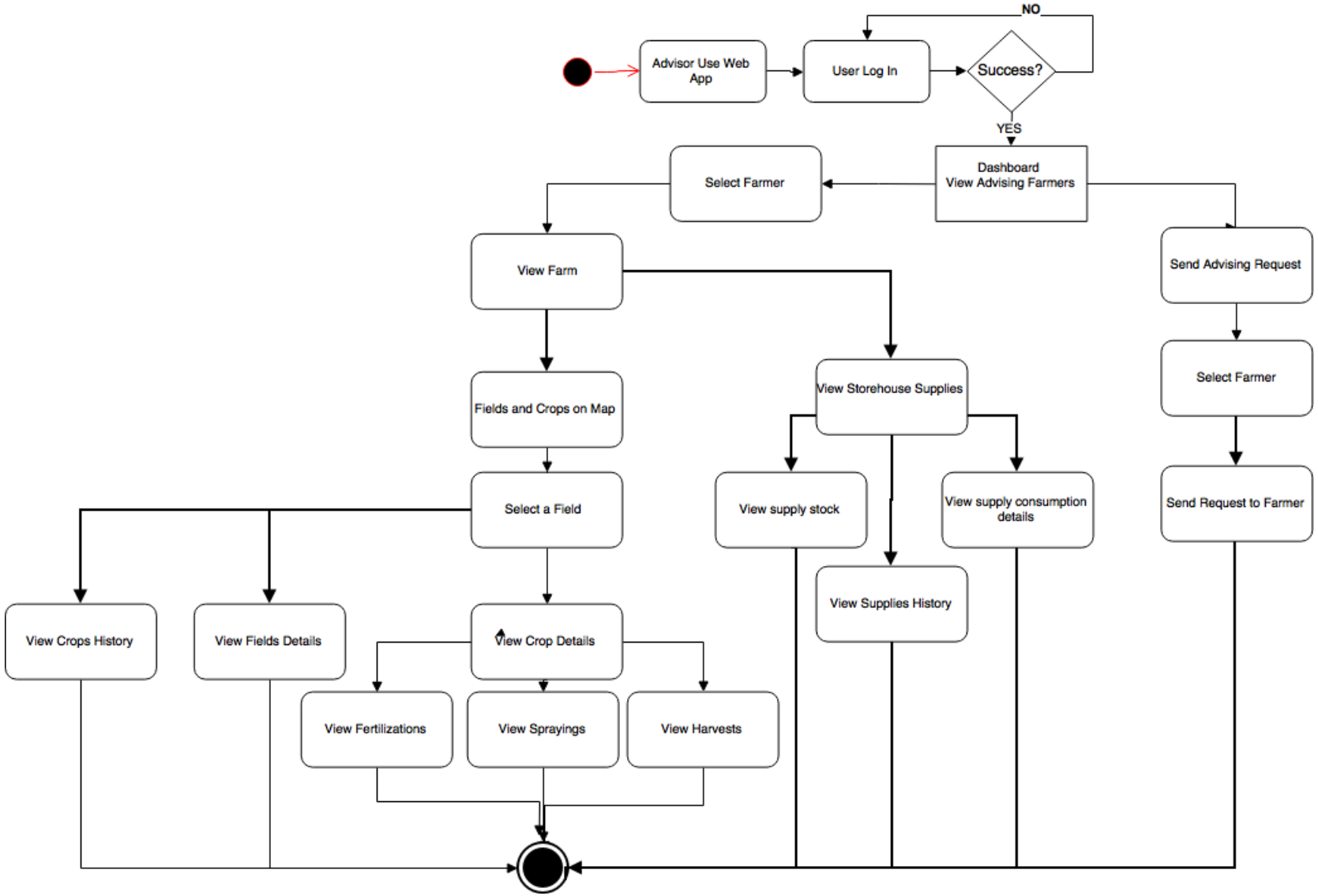


Figure 18: Activity Diagram – Advisor's functionalities via Web Application

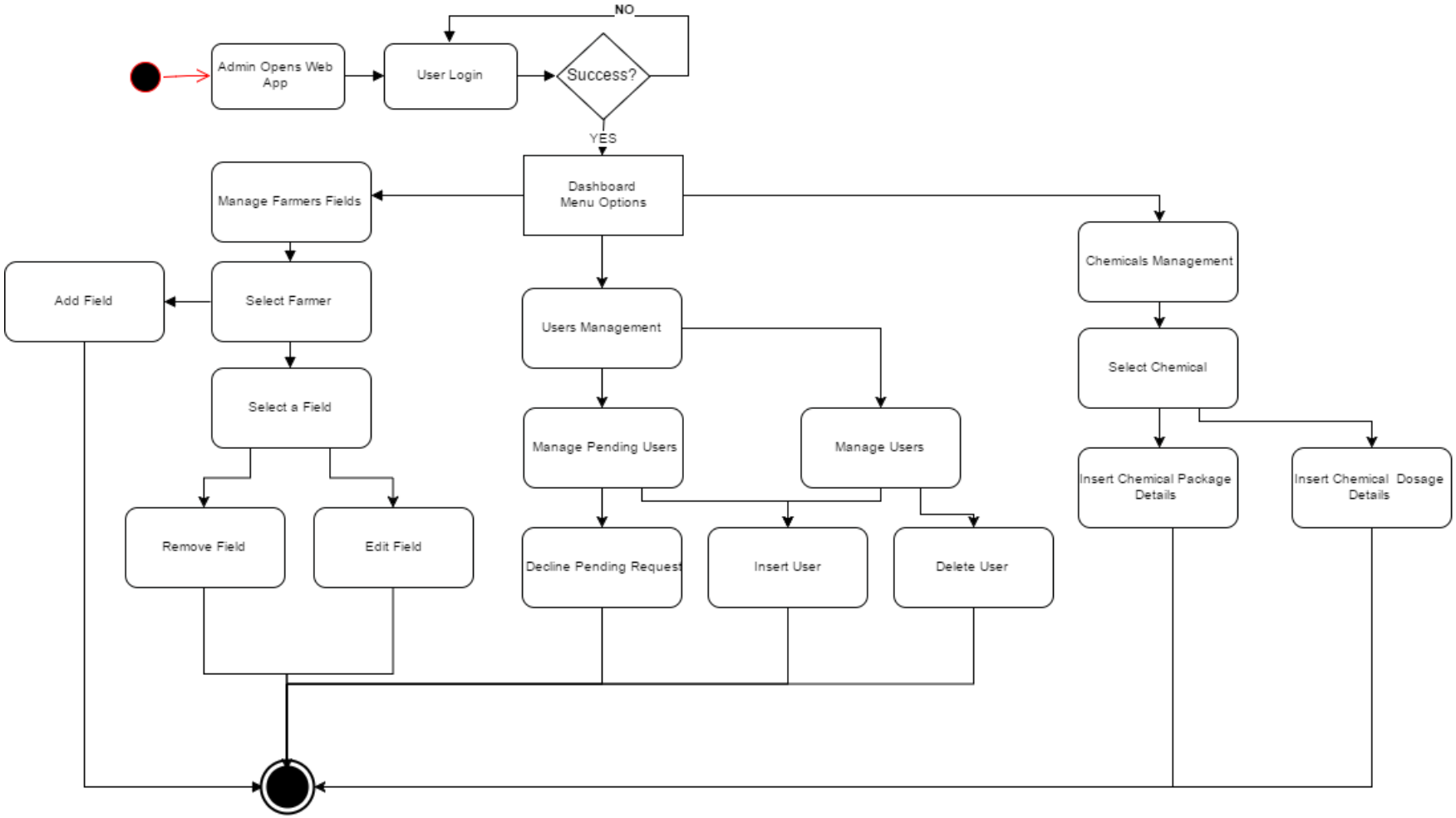


Figure 19: Administrator's Web Application Activity Diagram

Figure 17, shows in detail the steps that farmer follows in order to access information in the cloud exposed via the Web Application. The figure covers the work flow for farmers' functionalities described in the use case diagram in figure 12. The steps are analyzed below:

- The farmer is accessing the Web application from a Web browser and log's in using the FIWARE account. The web application sends a request to CloudFarm system in the cloud to check if the farmer is a valid user. If he is, then he can use the Farmer Dashboard to manage fields, crops, advisors and the storehouse, else he must fulfill a registration form.
- With Manage Fields option, all the fields are displayed on a map. By selecting a field on the map, the farmer can view field's and crops details (i.e., performed fertilizations, sprayings and harvests) and can also remove the field from the system.
- With Manage Advisors option, all the farmer's advisors, as well as the pending advisors' requests are displayed. Pending advisors' requests are the advisors who have sent a request to the farmer in order to gain access to farmer's data and start monitoring the farm's data. Farmer can confirm or decline a pending request from an advisor and can also remove an already accepted advisor.
- With Manage Storehouse option, all the farmer's purchased supplies with stock details are displayed. The farmer can choose among supplies to view detailed supply consumption.

Figure 18, shows all the advisors functionalities via the Web application running in the cloud. First the advisor must login into the CloudFarm system in the same way as described in figure 17. Then after a successful login the advisor is ready to use all the functionalities presented in the use case diagram (figure 13) and explained in the functionality for advisors in the cloud (chapter 3.3).

Figure 19, shows all the administrator functionalities via a Web application running in the cloud. First the advisor must login into the CloudFarm system as described in figure 17. Then after a successful login the advisor is ready to use the functionalities presented in the use case diagram (figure 14).

3.6 Class Diagram

UML Class diagrams are the means of describing graphical, the building blocks of the system such as the classes that composes our system and the collaboration among those classes. It shows the behavioral and data management responsibilities of each class and thus how the responsibility is delegated across the class model [22]. Figure 20 shows, the class diagram of our system.

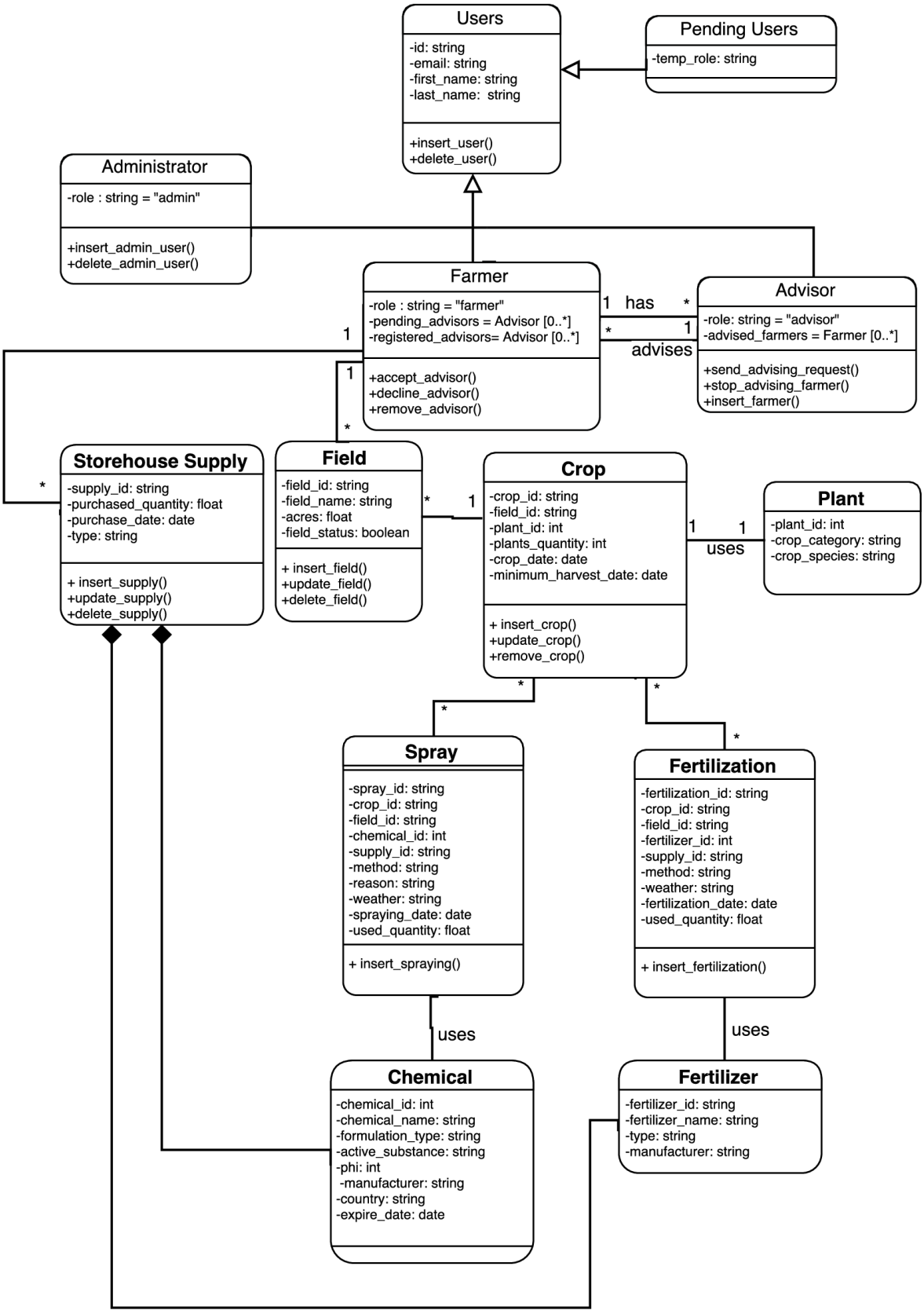


Figure 20: Class Diagram (Information Viewpoint)

Users Class: It's a super class and provides the abstract operation of all the system users. It has three child classes of Farmer, Advisor and Administrator each implement their own version of the abstract operation of insert and delete user. Child classes contains all the necessary information for the permanent CloudFarm users, meaning users that have been approved by system Administrators. This class stores the id, the email, the first name and the last name of a user which are common to all child classes.

Permanent Users Class: It contains the information for the permanent CloudFarm users, meaning users that have been approved by system Administrators. This class stores the first name, the last name and the role type of a CloudFarm user (farmer, advisor, administrator). It inherits all the other user details and functionality from the super class, combining all the necessary user information.

Pending Users Class: This class describes the details of a pending user who has completed a user registration into the system. This type of user is waiting the approval of the registration. It inherits from the user class the id, email, first name and last name but it also stores the temporary role of a pending user (the temporary role is the selected during user registration).

Administrator Class: This class contains the information of system administrator. It inherits all the other user details and functionality from the super class. It also stores and initialize the role (user role) information as "admin" It adds extra functionalities by overriding the super class functionalities such as the insertion and deletion of an administrator user.

Farmer Class: This class contains the information of a farmer. It stores a list of the pending advisors who are waiting the approval of farmer and a list of farmer's advisors (registered advisors). It also inherits all the other user details and functionality from the super class and initializes the user role information as "farmer". It adds extra functionalities such as accept an advisor which adds an advisor in the farmer's registered advisors, decline an advisor which removes advisor's pending request and remove a registered advisor.

Advisor Class: This class contains the information of an advisor. It stores a list of farmers who the advisor advises. It also inherits all the other user details and functionality from the super class and initializes the user role information as "advisor". It adds functionalities such as, send advising request to farmer (adds an advisor into farmer's pending advisors list), stop advising a farmer (removes a farmer from the advised farmers list) and insert farmer in the advised farmer list.

Field Class: This class contains the details of farmer's field. It stores information such as field id, field name, acres and field status (active with a crop or inactive without crop). A farmer class is associated with fields class. This means that a farmer can have multiple fields. A field class is also associated with a specific crop. It has functionalities such as insert, update and delete a field object.

Crop Class: This class contains the details of a crop. It stores information such as crop id, plant id, field id, quantity of plants, the crop date (creation date) and the minimum harvest date (the date where a crop can be harvested). Crop class is associated with Field class, and knows about this association by storing the field id. A crop class is also associated with a plant class and only the

first one knows that the relationship exists. It has functionalities such as insert, update and delete a crop object.

Plant Class: This class contains the details of a plant. It stores information such as plant id, crop category and crop species. Plant Class structure is approaching the structure of a plant object of the Plant Protection catalogue, of the Greek Ministry of Agriculture.

Chemical Class: This class contains the details of a chemical. It stores information such as chemical id, chemical name, formulation type, active substance, phi (is the number of waiting days after spraying), manufacturer, country of origin and expiration date. Chemical class structure is approaching the structure of a chemical object of the Plant Protection catalogue, from the Greek Ministry of Agriculture. There is an association with a composition aggregation relationship which indicates that a Chemical class instance lifecycle is depended on Storehouse Supply class's instance lifecycle. This means that a chemical object can only outlive through a Storehouse Supply object.

Fertilizer Class: This class contains the details of a fertilizer. It stores information such fertilizer id, fertilizer name, fertilizer type and the manufacturer. Fertilizer class structure is approaching the structure of a fertilizer object of the Plant Protection catalogue, of the Greek Ministry of Agriculture. There is an association with a composition aggregation relationship which indicates that a Fertilizer class instance lifecycle is depended on Storehouse Supply class's instance lifecycle. This means that a fertilizer object can only outlive through a Storehouse Supply object.

Spray Class: This class contains the details of a crop's spraying. It stores information such as spray id, crop id, field id, chemical id, supply id (purchased supply in storehouse), the spraying method, the spraying reason, the weather conditions, the spraying date and the supply used quantity. A spray class is associated with crop class. This means that a crop can have multiple sprayings. A spray class is associated with a storehouse supply because it uses a chemical object which only exists through a Storehouse Supply object. Spray class has the insert spraying functionality.

Fertilization Class: This class contains the details of a crop's fertilization. It stores information such as fertilization id, crop id, field id, fertilizer id, supply id (purchased supply in storehouse), the fertilization method, the weather conditions, the fertilization date and the supply used quantity. A fertilization class is associated with crop class. This means that a crop can have multiple fertilizations. A fertilization class is associated with a storehouse supply because it uses a fertilizer object which only exists through a Storehouse Supply object. Fertilization class has the insert fertilization functionality.

Storehouse Item Class This class contains the details of purchased supply. It stores information such as supply id, purchased quantity, purchase date and the type of supply (fertilizer or chemical). A farmer class is associated with storehouse supply class. This means that a farmer can have multiple storehouse supplies. It has functionalities such as insert, update and delete a storehouse supply object.

3.7 Non-functional Requirements

- The application is available on multiple mobile platforms (iOS, Android).
- Farmers must download to their mobiles or tablets the application.
- Minimum android version supported, 4.
- CloudFarm system should provide reliability and stability. The system needs to ensure its reliability, stability and data security in the case of high concurrency.
- System scalability: the system should be highly cohesive so that it can be easily extended to incorporate future changes and needs of the cloud system.

3.8 CloudFarm System Implementation

In this section, first we present the CloudFarm system architecture diagram as shows figure 21. The system comprises of three main architectural blocks referred to as, a) Front-End comprising of the Web application and the mobile platform application, b) The Back-End comprising of services for the management of information stored permanently in the cloud. Building upon principles of Service Oriented Architectures (SOA) CloudFarm is realized by means of well-defined components (services) communicating with each other using REST protocol. Next, we present those services, and then we separately analyze the functionality and the implementation of each.

3.8.1 CloudFarm Architecture

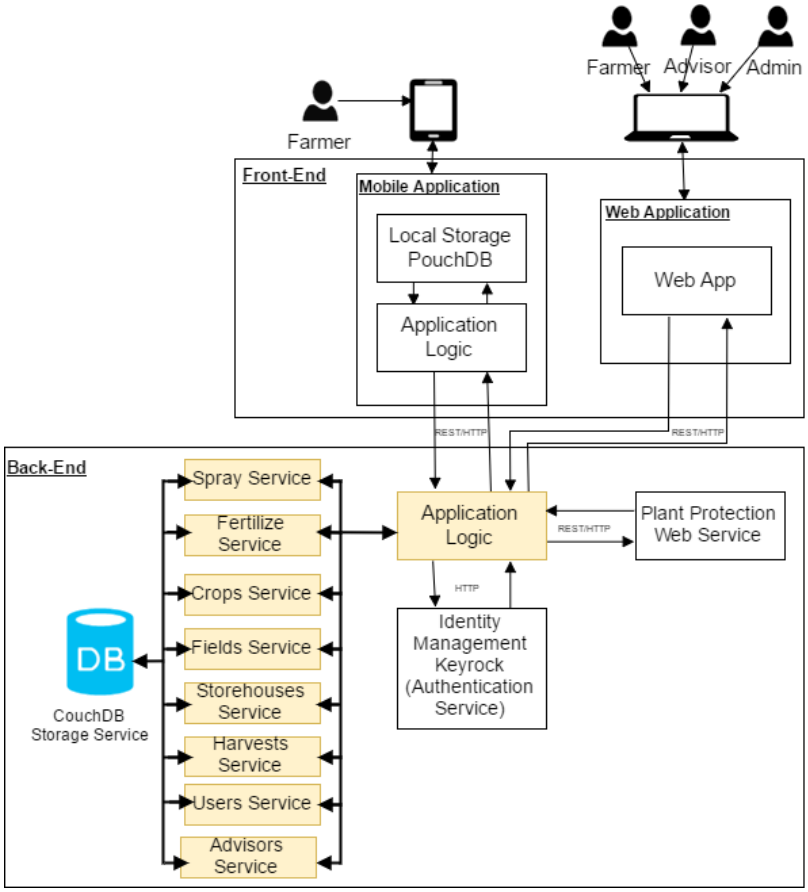


Figure 21: CloudFarm Architecture Diagram

Figure 21, illustrates CloudFarm architecture. The building blocks of the architecture are the Front-End the part of the CloudFarm system which could be visible to the users through a mobile application running on a smartphone or tablet and a Web application running on the Back-End. The Back-end comprised of services running in the cloud, and the Web Application (a Web interface to allow system users interact with the cloud using a Web browser).

3.8.2 CloudFarm Back-End

The Back-End as shown in figure 21, is the main part of the CloudFarm system hosted entirely on the cloud and it combines all system services such as storage, authentication, plant protection Web service and Application logic that fulfil the system requirements. The application logic is the "brain" of the CloudFarm system and provides a REST API that supports multiple endpoints and methods to system services and it can be used by clients in order to access the system's resources. Every single request to the API is processed first by the application logic that communicates with the necessary services in order to return the required information. The Back-End application logic which provides the REST API was implemented in Java with the Jersey framework, supporting a complete Create, Read, Update and Delete (CRUD) functionality. During the implementation phase, we tested API functionality using Postman[29] tool. In order to submit a request with Postman, we entered the API's endpoint service URL and then we chose the appropriate HTTP method (GET,POST,PUT,DELETE) and the request headers.

3.8.2.1 Application Logic

This module implements the business logic of the CloudFarm application, as it encompasses intelligence for handling, processing and exchanging data stored in storage service between other services. It provides the REST API which exposes the CloudFarm data and can be consumed of the Web application running in the cloud or the mobile application. Each API service fulfils the CloudFarm functionalities (e.g. Users Service, Sprays Service, Fields Service). Each request to REST API is processed first by the Application Logic to validate the data, as also to authenticate the user who submits the request via the Authentication Service.

3.8.2.2 Storage Service

This service is responsible for storing and retrieving data from a database. Its main functionalities are offered as a REST API making data storing and retrieving easy for developers and users. It is implementing using a CouchDB server which provides a well-structured REST HTTP API[30] for accessing or processing the data. It can host multiple databases which store JSON documents. In our system, each farmer has his own database in CouchDB server, comprised of multiple JSON documents (crops, harvests etc.). Farmers can read, edit, and insert database documents in their databases by sending requests to the API, along with credentials obtained in the form of access tokens by the user identification and authorization service. Each database defines its own security policy document by storing the users who have access to the database. Server admins have full access and can read, edit, and delete documents in every database in the CouchDB server. Advisors can only read documents from farmers who advise.

[29] https://www.getpostman.com/
[30] http://docs.couchdb.org/en/2.0.0/api .

3.8.2.3 Authentication & Authorization Service

The authentication service is provided by Keyrock Identity Management Generic Enabler (IdM) of FIWARE. The FIWARE IdM complies with the OAuth2[31] and supports token-based authentication. The reason why we choose a token-based authentication, is because it supports the stateless principle of REST API, so there is no need to keep a session stored in the server, the token is a self-contained entity that conveys all user information. The token lives in local storage on the Front-End. Figure 22 illustrates the authentication process.

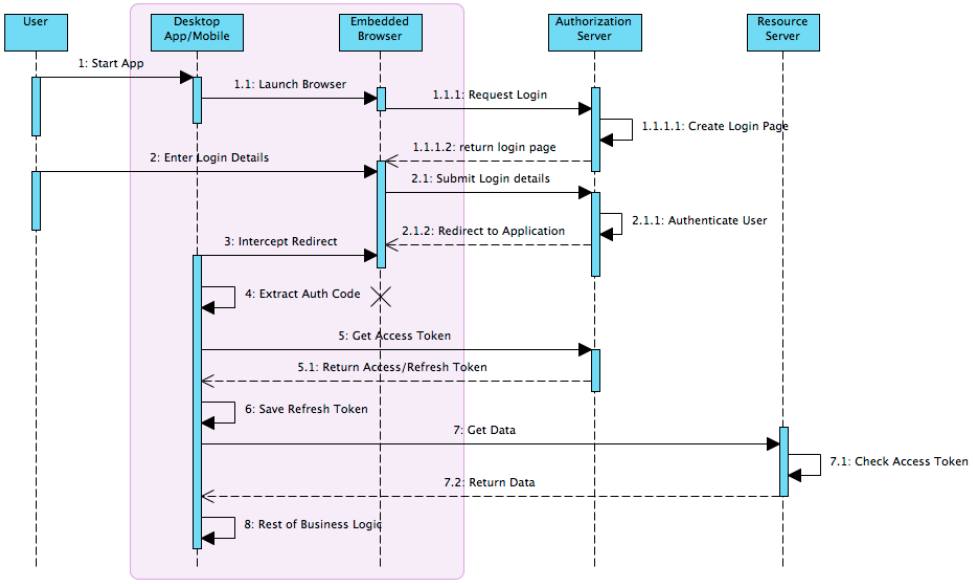


Figure 22: CloudFarm Authorization using Keyrock Identity Manager, Sequence Diagram.
(source: https://fhirblog.com/2014/06/18/fhir-oauth2-and-the-mobile-client)

- An end-user visits CloudFarm web Application or opens the mobile app.
- A user entering Cloud Farm application is redirected to Keyrock Identity Management GE of FIWARE in order to login into the CloudFarm.
- The use clicks a "Log In" button on CloudFarm site or app and are redirected to FIWARE IdM (Identity Manager) website and is prompted to log in with a FIWARE account and accept certain permissions.
- On successful login, the authorization server will redirect the user back to application along with an access token.
- On every single request to CloudFarm API (Resource server) must provide the access token, so the Back-End can verify the user's identity by asking Keyrock to identify the user with the given access token.

REST API implementation and methods of Keyrock Identity Manager can be found at FIWARE Lab Instance documentation[32].

[31] https://oauth.net/2/
[32] http://fiware-idm.readthedocs.io/en/v4.4.1/oauth2.html .

3.8.2.4 Plant Protection Service

This service was developed in order to provide all the information on plant protection products and fertilizers according to the catalogues of the Greek Ministry of Agriculture. Constitutes an effort of a REST Web Service for the implementation of the Plant Protection Catalogues provided by the Greek Ministry of Agriculture. It's a public RESTful Web service which fully provides all the details for plants, chemicals and fertilizers used in crop activities. Currently the Greek Ministry of Agriculture does not offer any web service where developers or third-party applications can consume agricultural data. For the implementation of this service we download a public instance of the database of Greek Ministry of Agriculture and replicate it into a database to our CouchDB server. It uses as format type JSON documents and it's a public web service for anyone interested in the development of any related agricultural application which needs that kind of agricultural information. Clients can send requests to the web service in order to fetch data in JSON format. This Web service was developed using Java Jersey framework and supports methods as shown in Table 1.

Resource	URI	HTTP Method	Method Description
Plants	/plant_protection_service/plants	GET	Get list of all plants
Plant	/plant_protection_service/plants/{plant_id}	GET	Get plant details
Chemicals	/plant_protection_service/chemicals	GET	Get List of all Chemicals
Chemical	/plant_protection_service/chemicals/{chemical_id}	GET	Get chemical details
Search	/plant_protection_service/chemicals/search?plant_id=	GET	Get list of permitted chemicals for a specific plant

Table 1: The REST implementation of Plant Protection Web Service.

The JSON body of this service is presented below:


```
{
  "crop_category": "Vegetable",
  "crop_species": "Tomato",
  "plant_id": 1062000
 }
```


3.8.2.5 Users Service

This services manages users by creating or removing one. There are two kind of users the CloudFarm users and the Pending users. When a user is registered in in the CloudFarm by

submitting a form, the user is considered as a pending user and must wait to be approved by the administrators. Administrators can accept or decline pending requests. If an administrator decides to accept a pending request, then the user is being removed from the pending users and inserted to CloudFarm users with a role given from the administrator. Table 2, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Users	/users	POST	JSON-encoded	Add a User
		GET		Get a list with all users
User	/users/{user_id}	GET		Get a User
		DELETE		Delete a User
		PUT	JSON-encoded	Update a User
Pending Users	/pending_users	GET		Get Pending Users
		POST	JSON-encoded	Add Pending User
Pending User	/pending_users/{user_id}	GET		Get Pending User
		DELETE		Delete Pending User

Table 2: The REST API implementation of Users Service.

An example of JSON-encoded request body of this service is presented below:

```
{
    "_id": "1767639282",
    "role": "advisor",
    “first_name”: “Apostolos”,
   “last_name” :”Rousalis”,
   "email": apostolosr@hotmail.com
 }
```

The user id for each user is the email encrypted using Adler-32[33] encryption hash function. When the user is registered into the system, the user’s email that obtained from Keyrock Identity Management GE is hashed in order the system to create the user id. So, the user id and the user email are related.

3.8.2.6 Fields Service

The main purpose of this service is to create, read, update and delete a farmer’s field. Any request to this service must be authenticated first from the authentication service to identify the user who wants to access field resource. A farmer can create or delete a field resource stored in the farmer’s

[33] https://en.wikipedia.org/wiki/Adler-32

database. An advisor can read the fields of the advised farmers. Administrators have full access to farmer's fields and use all CRUD operations on a field resource. Table 3, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Field	/users/{user_id}/farmer/fields	POST	JSON-encoded	Add Farmer's Field
		GET		Get Farmer Field
Fields	/users/{user_id}/farmer/fields/ {field_id}	PUT	JSON-encoded	Update Farmer's Field
		DELETE		Delete Farmer's Field

Table 3: The REST API implementation of Fields Service.

An example of JSON-encoded request body of this service is presented below:

```
 {
   "_id": "9beae871beae0a942849680609b17dfb",
  "fieldname": "B",
  "acres": "30",
  "active": true,
  "type": "field",
  "crop_id": "c7cbbe3f-ebae-4ef2-217d-5159953de21f",
  "points": [{  "point": "35.7108378,23.7524414"
              },{ "point": "34.9940038,23.7744141"
             },{  "point": "35.2994355,24.9609375"
             },{  "point": "35.960223,25.0048828"
           } ]
}
```

If the field has no crop, then the crop_id and the active field must be set as false. Points field is a JSON array which holds all the polygon points (latitude, longitude) of a field.

3.8.2.7 Crops Service

The main purpose of this service is to create, read, update and delete (remove) a farmer's crop. In crop creation, the field where the crop is going to be created must be checked if already has an active crop. After a successful crop creation, the field resource is updated as an active field with the basic information about the crop. So, crop service uses the fields service to update the field in which a farmer wants to create a crop. In addition, a crop resource can be updated with given data. Moreover, a crop can be removed, means that the end of crop period is over and the crop must also be removed from the field. This method is the delete function which does not basically remove the crop resource from the database just updates the crop status to inactive. After a

successful removal of crop, the Crops Service calls the Fields Service in order to update field's status from active to inactive. Any request to this service must be authenticated first from the authentication service to identify the user who wants to access the crop resource. A farmer can create, read, update or delete(remove) a crop. An advisor can read crops data of farmers who advises. Administrators have full access to farmers' crops data and can create, read, update or delete any crop resource. Table 4, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Crops	/users/{user_id}/farmer/ crops	GET		Get Farmer's Crops
		POST	JSON-encoded	Add Farmer's Crop
Crop	/users/{user_id}/farmer/ crops/{crop_id}	GET		Get Farmer's Crop
		PUT	JSON-encoded	Update Farmer's Crop
		DELETE		Delete Farmer's Crop

Table:4 The REST API implementation of Crops Service.

An example of JSON-encoded document for the request body of this service is presented below:


```
{
 "_id": "c7cbbe3f-ebae-4ef2-217d-5159953de21f",
 "type": "crop",
 "field_id": "9beae871beae0a942849680609b17dfb",
 "plant_id": 1055100,
 "crop_species": "Zucchini ",
 "crop_date": "2017-01-30T21:56:50.163Z",
 "min_harvest_date": "2017-02-02T21:58:16.869Z",
 "active": true,
 "quantity": "200"
}
```


3.8.2.8 Harvests Service

The main purpose of this service is to create and read a harvest performed on farmer's crop and to protect the consumer's health from the chemicals side-effects. A Harvest is only performed if the minimum required harvest date of crop after a spraying action has passed. In order to get the minimum harvest date of crop, the service uses the Crops Service to obtain crop's information. The quantity harvested is measured in kilos or pieces. Any request to this service must be authenticated first from the authentication service to identify the user who wants to perform harvest and access crop's information. Only a farmer can perform a harvest (create harvest). An advisor can read harvests data of farmers who advises. Administrators have full access to farmers'

harvests data and can read, update or delete any harvest resource. Table 5, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Harvests	/users/{user_id}/farmer/crops/{crop_id}/harvests	GET		Get crop's harvests
		POST	JSON-encoded	Add a Harvest
Harvest	/users/{user_id}/farmer/crops/{crop_id}/harvests/{harvest_id}	GET		Get a harvest
		DELETE		Delete a harvest

Table 5: The REST API implementation of Harvests Service.

An example of JSON-encoded document for the request body of this service is presented below:


```
 {
  "_id": "A97BE67F-A102-3637-B827-A5EC744165D4",
  "crop_id": "c7cbbe3f-ebae-4ef2-217d-5159953de21f",
  "field_id": "9beae871beae0a942849680609b17dfb",
  "harvest_date": "2017-02-16T18:25:01.115Z",
  "quantity": 500,
  "measurement": "pieces",
  "type": "harvest"
}
```


3.8.2.9 Storehouses Service

The main purpose of this service is to manage farmer's storehouse supplies (items). A farmer can register (create) a purchased supply and read supplies details. A supply in storehouse is automatically updated after a spraying or fertilization action. A storehouse item can be removed (deleted) only when the supply was inserted by mistake and it hasn't consumed anywhere. Any request to this service must be authenticated first from the authentication service to identify the user who wants to access the supplies. Only a farmer can insert, update and delete a supply. An advisor can only read supplies of farmers' who advises. Administrators have full access to farmers' storehouses data and can read, update or delete any storehouse item resource. Table h, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Storehouse Supplies	/users/{user_id}/farmer/storehouse/	GET		Get Storehouse Supply

		POST	JSON-encoded	Add Storehouse Supply
Storehouse Supply	/users/{userid}/farmer/storehouse/{supply_id}	GET		Get A Storehouse Supply
		PUT	JSON-encoded	Update a Storehouse Supply
		DELETE		Delete a Storehouse Supply

Table 6: The REST API implementation of Storehouses Service.

Examples of JSON-encoded documents for the request body of this service are presented below:

For a Chemical:


```
{
   "_id": "4f90e6b2-c455-e992-c8cc",
   "type": "storehouse-chemical",
   "date": 1485727200000,
   "chemical_id": 1592,
   "chemical_name": "PYRINEX 5 GR",
   "kind": "box",
   "quantity": " 2000",
   "current_quantity": " 2000",
   "measurement": "gr"
 }
```


For a New Type of Fertilizer:


```
{
   "_id": "4f90e6b2-c455-e992-c8cc",
   "type": "storehouse-fertilizer-new",
  "fertilizer_name": "BORLASSA 308",
   "quantity": 50,
   "current_quantity": 50
}
```


For a General Type of Fertilizer:


```
{
   "_id": "17681e31-4527-a161-9af1",
   "type": "storehouse-fertilizer-general",
   "date": 1485295200000,
   "fertilizer_name": " Ammonia",
   "quantity": 20,
   "current_quantity": 15
    }
```


3.8.2.10 Sprays Service

The main purpose of this service, is to manage sprayings performed on farmer's crops according to agricultural rules and principles in order to protect the consumer's health from the chemicals side-effects. A spraying is only performed with a permitted chemical and only if the minimum required harvest date of crop after a spraying action has passed. In order to get the minimum harvest date of crop, the service uses the Crops Service to obtain crop's information. Moreover, this service communicates with the Plant Protection Service to get the chemical details and calculates the crop's new possible harvest date using those details. The new possible harvest date calculated as the number of days that the crop cannot be harvested due to chemical side-effects from the spraying date. Before spraying the availability of the chemical in storehouse should be checked. Thus, the storehouse service is called to provide the amount of the chemical in the

storehouse. The needed proportion of the chemical is automatic calculated (in cube centimeters) according to the chemical details obtained from the Plant Protection Service and the field's acres. After a successful spraying the Sprays Service calls the Crops Service to update the crop's harvest date and also calls the Storehouse Service to update the new stock amount. This service uses an efficient algorithm on how the quantity will be consumed on a spraying when more than one packages of the supply are existing in storehouse. Moreover, this services is responsible to restore stock to previous amounts after a removal of wrong spraying insertion. Any request to this service must be authenticated first from the authentication service to identify the user who wants to do a spraying. Only a farmer can perform a spraying activity. An advisor can read spraying data from farmers who advises. Administrators have full access to farmers' spraying data and can read, update or delete any spraying resource. Table 7, shows the REST API implementation of this system service.

Resource	URI	HTTP Method	Request Body	Method Description
Sprays	/users/{user_id}/farmer/crops/{crop_id}/sprays	GET		Get all Sprays
		POST	JSON-encoded	Add a Spray
Spray	/users/{user_id}/farmer/crops/{crop_id}/sprays/{spray_id}	GET		Get a Spray
		DELETE		Delete a Spray

Table 7: The REST API implementation of Sprays Service

An example of JSON-encoded document for the request body of this service is presented below:


```
{
  "_id": "c474d8c3-0e70-a708-77c6-9b85c157dbc1",
  "type": "spray",
  "field_id": "9beae871beae0a942849680609b17dfb",
  "crop_id": "c7cbbe3f-ebae-4ef2-217d-5159953de21f",
  "chemical_id": 2188,
  "measurement": "cc",
  "spraying_date": "2017-01-30T21:58:16.869Z",
  "quantity": 900,
  "weather": "Sunny Day",
  "method": "Turbine Sprayers",
  "reason": "pesticide"
}
```


3.8.2.11 Fertilizations Service

The main purpose of this service, is to manage fertilizations performed on farmer's crops. A fertilization can use two types of fertilizers, the general type fertilizer and the new type fertilizers.

The user must insert in kilograms the dosage of the fertilizer. In order to perform crop's fertilization, the service calls the Crops Service to obtain crop's information, and the Storehouse Service to check the fertilizer's availability in the storehouse. This service uses an efficient algorithm on how the quantity will be consumed on a fertilization when more than one packages of the supply are existing in storehouse. After a successful fertilization, the Fertilization Service calls the Storehouse Service to update the new stock amount. Any request to this service must be authenticated first from the authentication service to identify the user who wants to do a fertilization. Only a farmer can perform a fertilization. An advisor can read fertilizations data from farmers who advises. Administrators have full access to farmers' fertilizations and can read, update or delete any fertilization resource. Table 8, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Fertilizations	/users/{user_id}/farmer/crops /{crop_id}/fertilizations	GET		Get all crop's fertilizations
		POST	JSON-encoded	Add a fertilization
Fertilization	/users/{user_id}/farmer/crops /{crop_id}/fertilizations/ {fert_id}	GET		Get a fertilization
		DELETE		Delete a fertilization

Table 8: The REST API implementation of Fertilizations Service

An example of JSON-encoded document for the request body of this service is presented below:


```
{
  "_id": "fff87241-c479-f9e3-34e2-802b6c0ee1f5",
  "type": "fertilization",
  "field_id": "9beae871beae0a942849680609b17dfb",
  "crop_id": "c7cbbe3f-ebae-4ef2-217d-5159953de21f",
  "fertilizer_name": "Ammonia",
  "measurement": "kg",
  "date": "2017-01-30T21:59:41.395Z",
  "quantity": 5,
  "weather": "Sunny Day",
  "method": "Hand Linear"
}
```


3.8.2.12 Advisors Service

The main purpose of this service, is to manage farmers' advisors and the pending advising requests which are sending from the advisors to the farmers. In the one hand, an advisor can read all the farmers details who advises and send (create) an advising request to a farmer. An advisor

can also stop advising a farmer by deleting him from his farmers advising list. In the other hand, a farmer can, read all the advisors who advise him, accept or decline an advising request and delete an advisor from his advisors list. When the farmer accepts advisor's pending request, then the advisors is added in the farmer's advisors list. Any request to this service must be authenticated first from the authentication service to identify the farmer or advisor who wants to access the service. Administrators have full access to this service and can read, update or delete an advisor resource. Table 9, shows the REST API implementation of this service methods.

Resource	URI	HTTP Method	Request Body	Method Description
Farmer's Pending Advisors	/users/{user_id}/farmer/advisors/pending_advisors	GET		Get Pending Advisors
		POST	JSON-encoded	Add Pending Advisor
Farmer's Pending Advisor	/users/{user_id}/farmer/advisors/pending_advisors/{pending_id}	GET		Get Pending Advisor
		DELETE		Delete Pending Advisor
Farmer's Advisors	/users/{user_id}/farmer/advisors	GET		Get Farmer's Advisors
		POST	JSON-encoded	Add Farmer Advisor
Farmer's Advisor	/users/{user_id}/farmer/advisors/{advisor_id}	GET		Get Farmer Advisor
		DELETE		Delete Farmer Advisor
Advised Farmers	/users/{user_id}/advisor/advising_farmers	GET		Get Farmers who are Advised by Advisor
Advised Farmer	/users/{user_id}/advisor/advising_farmers/{farmer_id}	GET		Get Farmer Advised by Advisor
		DELETE		Stop from Advising Farmer

Table 9: The REST API implementation of Advisors Service.

An example of JSON-encoded document for the request body of this service is presented below:

Farmer's Pending Advisor JSON Document:

```
{
    "advisor_id": "fcf05cb1f4537d37f57e0cbc6400401d",
   "type": "pending_advisor"
 }
```

Farmer's Advisors JSON Document:

```
{
    "advisor_id": "fcf05cb1f4537d37f57e0cbc6400401d",
   "type": "farmer_advisor"
 }
```

3.8.2.13 REST API Request Headers

All the requests to REST API, must have the following request headers as shown in table 10, including the access token obtained from Keyrock Identity Manager as described in Authentication Service.

Header type	Value
Content-Type	application/json
Authorization	*Bearer Access-Token*

Table 10: REST API Request Headers

3.8.2 CloudFarm Front-End

3.8.2.1 Mobile Application

The mobile application developed as a hybrid application using the web technologies such as HTML[34], CSS[35] and JavaScript[36] hosted in application's WebView[37]. The reason why the hybrid approach was chosen is:

1. The need of a mobile application to work on multiple platforms such as Android and iOS without developing versions for every platform. We have one codebase for mobile platforms (Android, iOS). So, we will be definitely saving on development time.
2. We can easily develop the mobile application using Web technologies without being familiar with the programming language and environment of each mobile platform.
3. The ability of application to be distributed through the app stores.

The CloudFarm mobile application is developed using Ionic[38] framework based on Apache Cordova, AngularJS, HTML and CSS. The plugins were used are the geolocation to locate farmer's position, the network information which quickly checks the network status and the InAppBrowser[39] which launch a URL in another in-app browser (redirects from services).

One of the main goals of the mobile application, was to support offline functionality so it can be used by farmers, even if there is no internet connection, because some agricultural lands might not be covered with a 3G or 4G network. The restriction of the internet connection led to implement services that can run offline scenarios, locally inside the mobile application logic. Those services exchange data with a local storage without the need to interact with the cloud when the application is offline. With the offline syncing end-users, can perform agricultural activities without internet connection. After the device is back online, the changes are synced with the Back-End using the API. Another main goal of the mobile application was the ability to locate a farmer inside a field.

[34] https://www.w3schools.com/html/
[35] https://www.w3.org/Style/CSS/
[36] https://www.w3schools.com/js/
[37] https://cordova.apache.org/docs/en/latest/guide/hybrid/webviews
[38] https://ionicframework.com
[39] https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-inappbrowser/

Application Logic

Is the most crucial service of the mobile application. It has the logic intelligence which is used for the communication between the CloudFarm system (Back-End) and the services of the application. It is also supports the ability to perform agricultural activities when the mobile application is offline and when is back online it begins the syncing process between the local data stored in local storage service and the Back-End. It also manages the User interface of the mobile application where farmers can interact with the system. It follows the Good Agriculture Practices principles and standards for every agricultural activity performed by a farmer, through user interface, such as harvest, spraying, fertilization. Of course, the log in function cannot be provided offline and it is only available when the device is connected to the internet because it must interact with the Authorization Service and Users Service in the Back-End in order to verify the user. So, the application can work offline only when the farmer has previously logged in.

As described earlier, farmers should be able to capture farm activities performed on fields instantly by using the mobile application. Thus, the application should detect the field in which the farmer is performing those activities. For that reason, the application logic uses, the Geolocation Service (an Apache Cordova plugin) which returns the geographic location of the farmer and the radius based on information about cell towers, WiFi[40] nodes, GPS[41]. The application logic in order to identify in which field the farmer is located it uses an algorithm which uses the farmer's location gained from Geolocation Service. The algorithm uses a point (latitude, longitude) which indicates the current location of the smartphone, gained from geolocation service (GPS). Next, it should check if the point is really within a field. A field is represented as a polygon, consisted of multiple points. If the farmer's point is inside a field is calculated by iterating all over the polygon's points (every point being x,y) and finds the min ,max values of x and y. A point is inside a polygon if it has not have x value smaller than xmin and greater than xmax or y value smaller than ymin and greater than ymax. It is a very simple algorithm and runs very fast. This algorithm quickly excludes a point if it is not within a polygon. The application logic runs this algorithm for each field in order to identify if the user is locating inside a field or not.

```
if (farmer.x < Xmin || farmer.x > Xmax || farmer.y < Ymin || farmer.y > Ymax) {
  // Definitely not within the field (polygon)  }
```

Local Storage

This service is responsible to store application's data locally in the device. The mobile application can be used offline, if there is no internet connection, so the application should be able to store data locally using PouchDB[42] database which provides datastore for offline applications [21]. PouchDB is a lightweight open-source database written in JavaScript inspired by Apache CouchDB that stores JSON documents and runs well within the hybrid application's WebView[43]. When the

[40] http://www.wi-fi.org
[41] http://www.gps.gov/systems/gps
[42] https://pouchdb.com/
[43] https://developer.android.com/reference/android/webkit/WebView.html

application is connected to the internet, in order to support offline functionality, fetches farmers needed data from the Back-End such as active crops, crops sprayings, crops harvests, crops fertilizations, fields, storehouse items, and stores the data in the local database. In that way, if the device goes offline it has all the required data. In the offline mode, during an agricultural activity the application stores the data in the local database as JSON objects. When the internet connection is back, the local database syncs data with the remote database on the cloud using the API. For example, if a farmer has performed a spraying activity offline, a spraying JSON document is stored in the local database and when the application is online sends the spraying data to Sprays Service in the Back-End using asynchronous call (AJAX[44]) to Sprays Service endpoint. While the application, is online all the performed activities are sent directly to the Back-End and synced with the local database by fetching from the cloud only the data that changed by the performed activity. So, if the application suddenly loses the internet connection it can be fully functional by having all the required data.

Moreover, the mobile application must support the Plant Protection Service which provides all the necessary information about plants and chemicals according to the Greek ministry of Agriculture. Thus, if there's is no internet connection the mobile application cannot get the data from the Plant Protection Service. To solve this problem, when the application is online it gets the plants and chemicals data from the Plant Protection Service running in the cloud (Back-End) and stores them in the local storage.

3.8.2.2 Web Application

As we discus in system architecture (chapter 3.8.1) the second Front-End is a Web interface that allows system users to interact with the Back-End in the cloud using a Web browser. The development process of the web application is based on the functional requirements, use case and activity diagrams (chapter 3.3 – 3.5). The web application is as a single page application (SPA) developed with AngularJS framework, HTML, CSS, PHP[45]. The AngularJS $http[46] service is a core AngularJS service that facilitates communication with the REST API via AJAX[47] (asynchronous HTTP) requests, for transferring data between client and server. By using AJAX asynchronous HTTP requests to communicate with the Back-End, the user can still use a responsive Web page, while the request is waiting for a response from the Back-End. In that way, the web application does not need to reload the whole page to get new information. The Front-End works with the following logic. A user opens a Web browser and enters the URL of CloudFarm Web application. Then the Web applications is stored and runs inside the Web browser. If the user requests to view data, then an asynchronous http request is sent from the browser to the REST API, by using the AngularJS http service, in order the browser to get the required data. The AngularJS displays dynamically this data in a visual representation in the HTML document without refreshing and re-rendering the whole web page.

[44] https://www.w3schools.com/xml/ajax_intro.asp
[45] https://en.wikipedia.org/wiki/PHP
[46] https://docs.angularjs.org/api/ng/service/$http
[47] https://en.wikipedia.org/wiki/Ajax_(programming)

3.9 Launching the instance in FIWARE lab

The whole Back-End CloudFarm system, virtual infrastructure was based on FIWARE Lab Cloud. In order to get access to FIWARE Lab Cloud we asked the cloud's administrators. After we got the access to the cloud we clicked on the Instance tab and then clicked on Launch New Instance and we chose the image type (Ubuntu 14.04). We chose a medium flavor instance meaning that it has 4GB RAM, 2 VCPUs and 40GB Disk as shown in figure 25. We accessed our instance over a given IP via any SSH client such as PuTTY[48] using a security keypair.

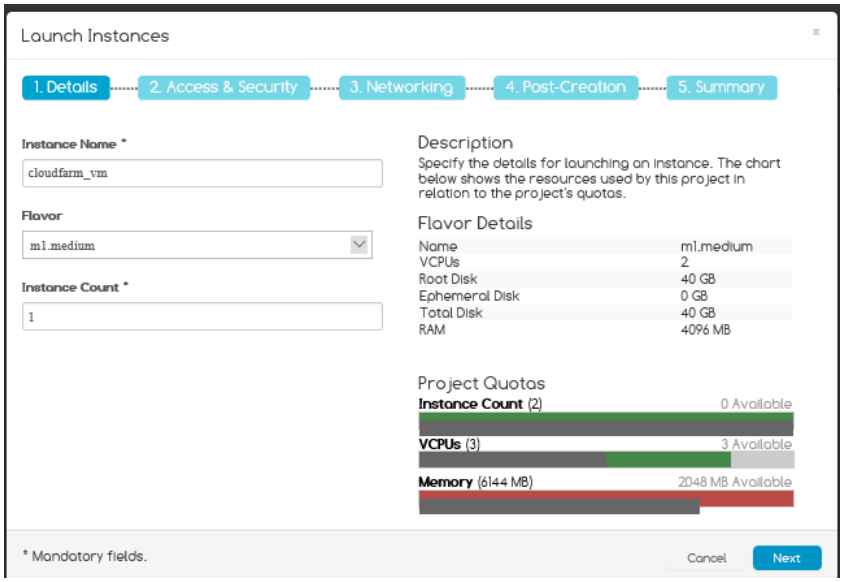


Figure 25: FIWARE Lab Cloud Platform : Launch the Instance as a VM.

3.10 CloudFarm System Deployment

Our virtual cloud infrastructure is consisted of one Virtual Machine (VM).It is our instance running in FIWARE lab as presented in previous section and it has the following specifications:

- 2 Virtual CPUs (VCPUs).
- CPU Model: Intel Xeon E3-1242.
- CPU Clock Rate: 2800 Mhz.
- CPU Cache size 4096 KB.
- CPU architecture x86_64
- 4GB RAM.
- 40 GB Hard Disk.
- Operating System: Ubuntu 14.04

[48] http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

In our Virtual Machine runs:

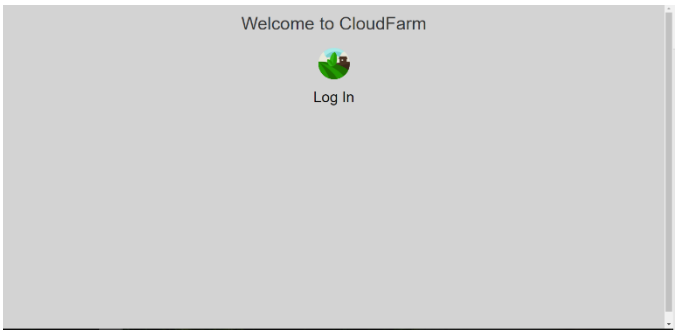
- An Apache Tomcat server (port 8080) which serves the REST API that can be accessed from http://147.27.60.206:8080/api/webapi and the Plant Protection Web Service that can be accessed from http://147.27.60.206:8080/plant_protection_service .
- An Apache server which provides a Web Application to end-users through a Web browser at http://147.27.60.206/cloudfarm.
- A CouchDB server.

As described in system architecture, CloudFarm Back-End uses Keyrock Identity Manager GE as authentication service which runs on a public instance in the FIWARE Lab[49].

3.11 CloudFarm Presentation

CloudFarm User Interface is in Greek as it is intended to be used by Greek farmers and uses agricultural data provided by the Greek Ministry of Agriculture. All screenshots appearing in this thesis are translated in English by Google Translate. Any language mistakes are due to automatic translation.

Use Case 1 – A Registered CloudFarm Farmer uses the Web Application



As shown in figure 26, the user must click on the log in button in order to redirected to Keyrock IdM for Logining into the CloudFarm system.

Figure 26: CloudFarm Login Page.

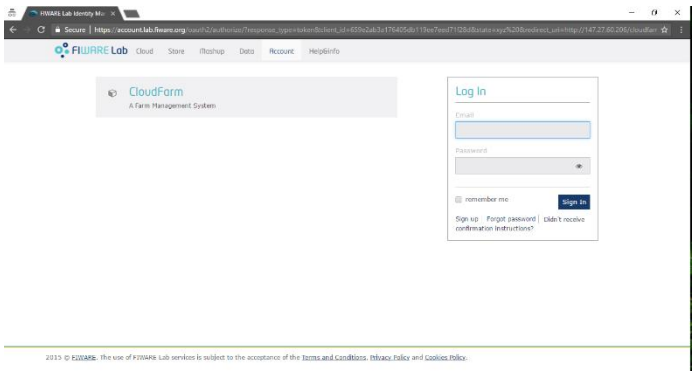


Figure 27 shows the FIWARE Lab login page. After a successful login, the user will be authenticated and redirected to the application's dashboard. In this use case scenario, the user is already a registered farmer, so the farmer's dashboard will appear after a successful log in.

Figure 27: FIWARE Lab, Login page.

[49] https://account.lab.fiware.org

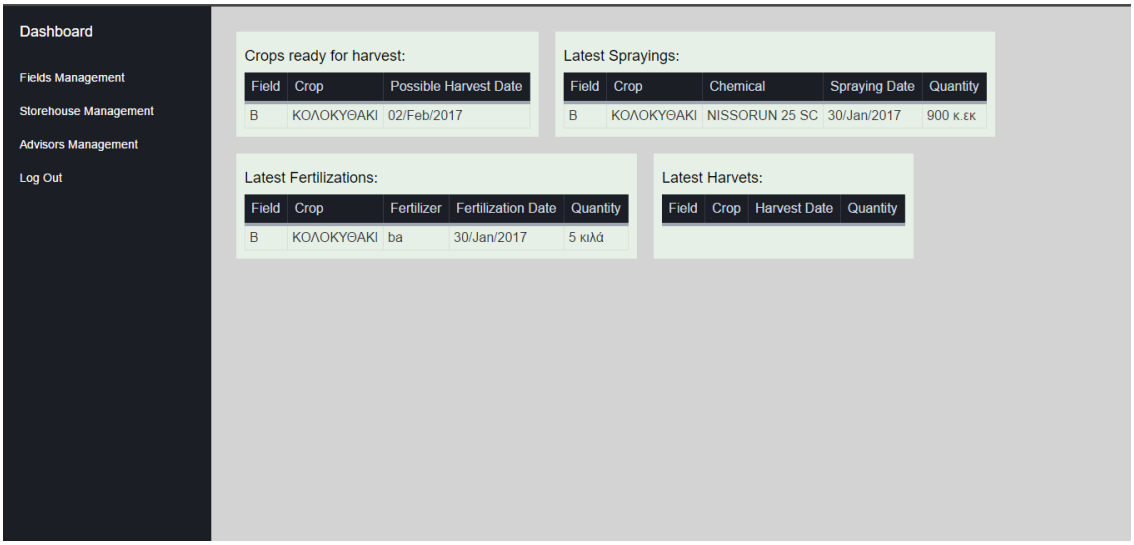


Figure 28: Farmer's Dashboard page.

Figure 28, shows that through dashboard a farmer can view an overall image of crops like the latest harvests, spraying, fertilizations performed in crops and the crops which are ready to be harvested. From options menu, the farmer can visit the Fields Management, the Storehouse Management and the Advisors Management pages. The Log Out option sing out the farmer from the system.

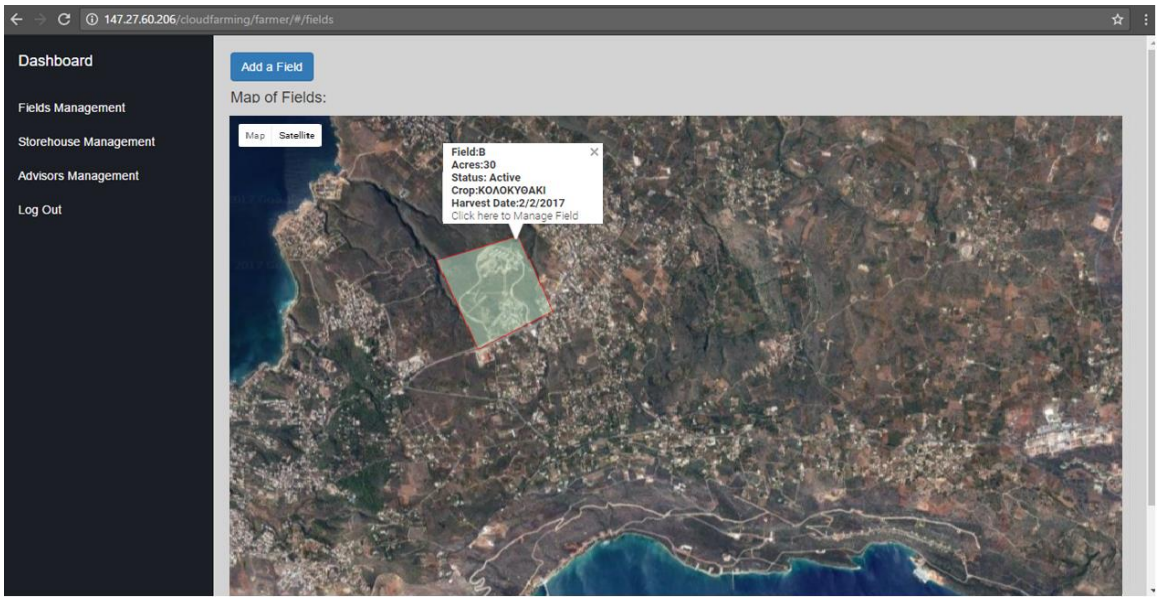


Figure 29: Fields Management page.

Figure 29 shows the Fields Management page. Farmer can view all the fields on a Map. This page uses Google Maps which converts the points of a field to polygon. When the farmer mouse over a field he can view a summary of the field status such as field's name, acres, status (active with

crop) and the crop category (if it has a crop). By clicking a field the farmer can view the Field & Crop details.

Figure 30: Field & Crop Details page.

In figure 30, the farmer can use the Field Actions to overview field details such as the field's total sprayings and fertilizations which shows a history of all fertilizations and sprayings performed in that field. Moreover, the farmer can view the current (active) crop and the crops history from all the previous crops. The Crop Details page as shown in figure 31, presents all the detailed information about the crop such as sprayings, fertilizations and harvests.

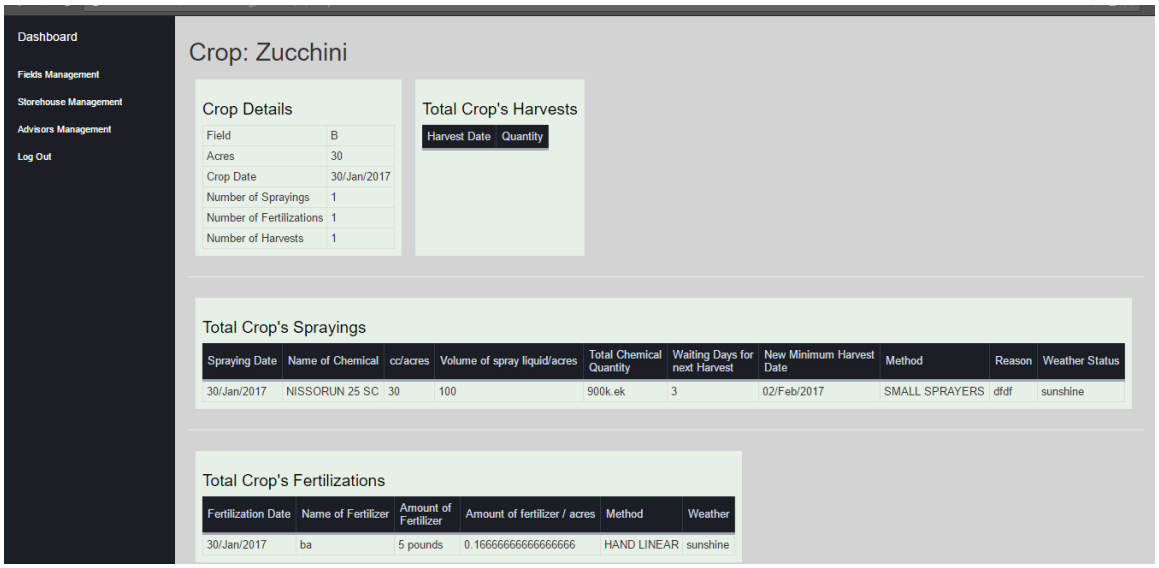


Figure 31: Crop Details page.

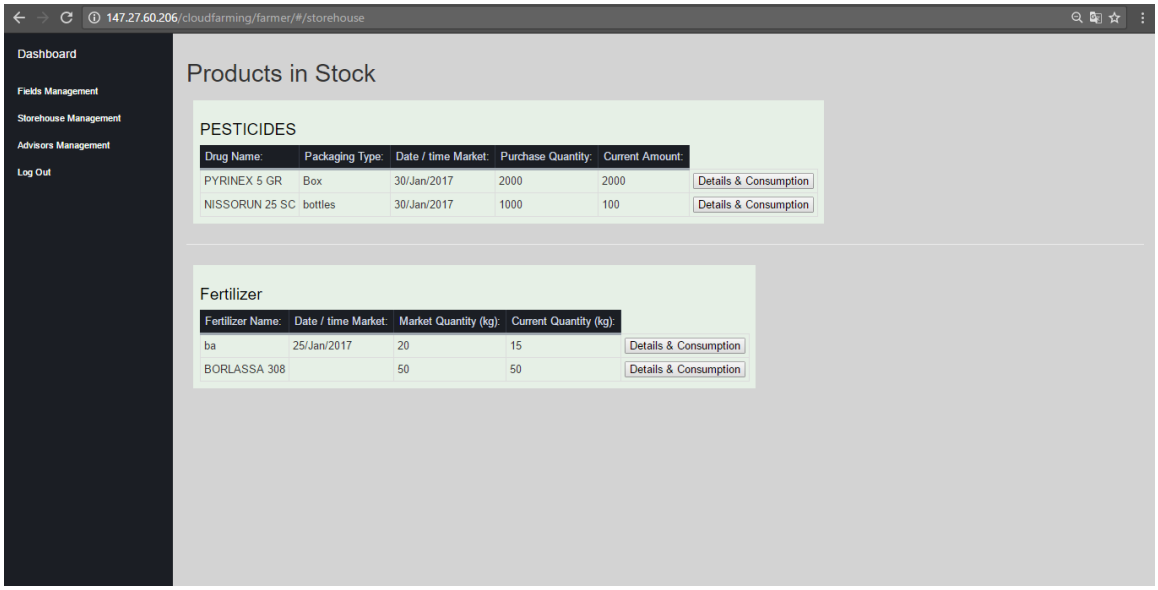


Figure 32: Storehouse Management page.

In the Storehouse Management page, shown in figure 32, the farmer can view all the purchased storehouse items (supplies) and details such as the current amount of the supply, where the supply was used and how much quantity were consumed (Details & Consumption page shown in Figure 33).

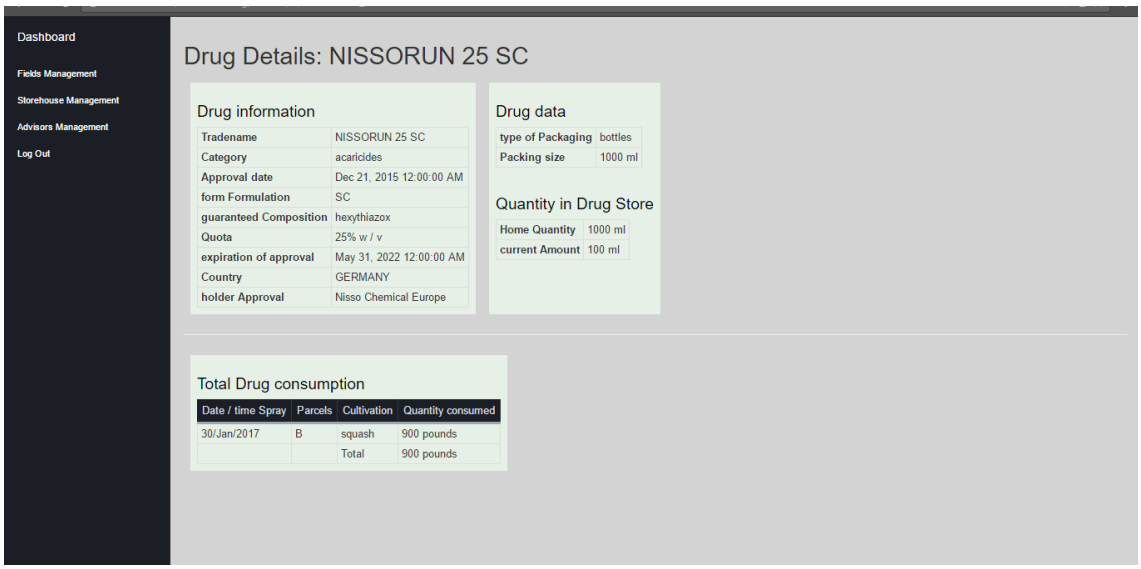


Figure 33: Details & Consumption page.

Use Case 2 – An advisor uses the CloudFarm Web application

The user log's in the CloudFarm as shown in figures 26, 27 and is redirected back to the advisor's dashboard.

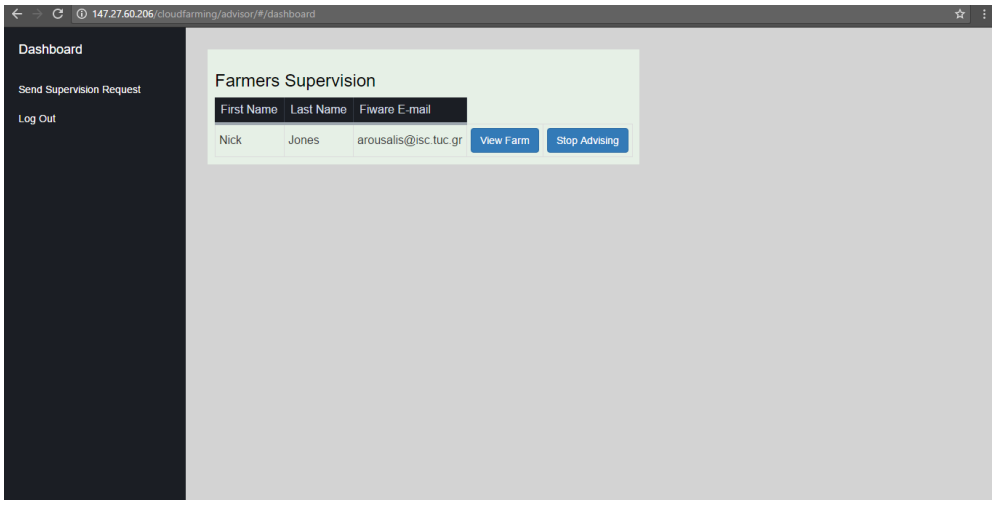


Figure 34: Advisor's Dashboard page.

In figure 34, the advisor can view all the advised farmers and by clicking on the View Farm button can view a farmer's farm details.

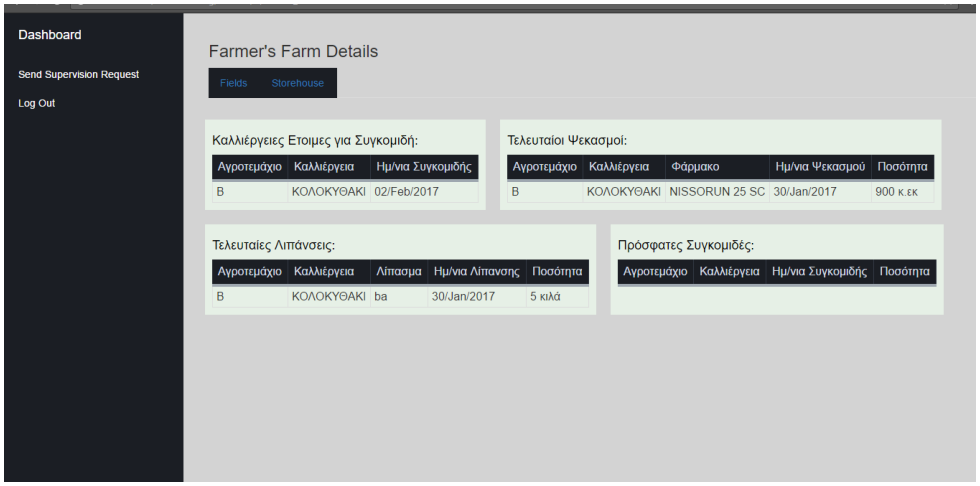


Figure 35: Advisor monitoring farmer's farm.

In figure 35, the advisor can view all the agricultural activities performed by the farmer and all fields and storehouse details. The page is familiar with the farmer's page shown in figure 28. By clicking on the Fields button the advisor can visit the fields page as shown in figure 29 and by clicking on the storehouse button he can visit the storehouse page familiar with the farmer's page as shown in figure 32.

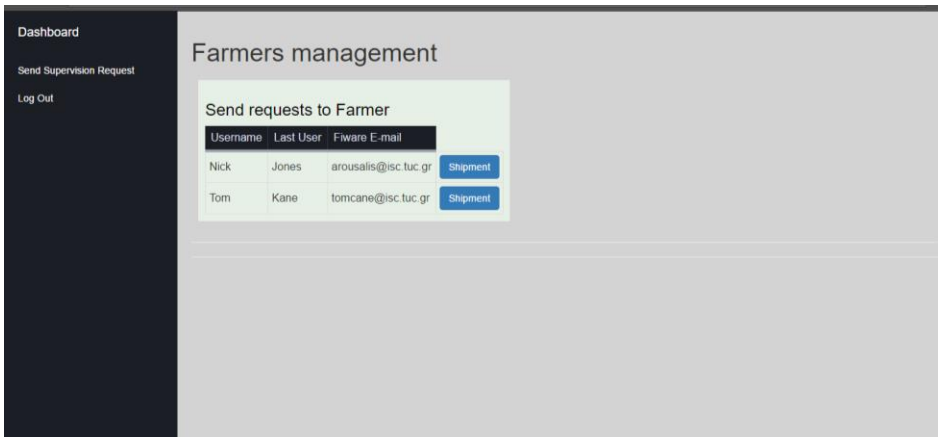


Figure 36: Advisor's Send Supervision Request page.

In figure 36, an advisor through this page can send supervision requests to farmers. A table with all the farmers is shown. The advisor in order to send the advising request should press the send button. The request is sent to a farmer as a pending advising request and the farmer should accept the request to give access to the advisor.

Use Case 3 – A new user is using the CloudFarm system (Web or Mobile application)

In figure 37, a user after a successful login through Keyrock Identity Manager, if it is not a CloudFarm user is redirected back to the registration page. The user fulfils the first name, the last name and the user role (can register as farmer or as an advisor). A successful registration means that a request to the Users Service has sent and the user is added to the system as a pending user.

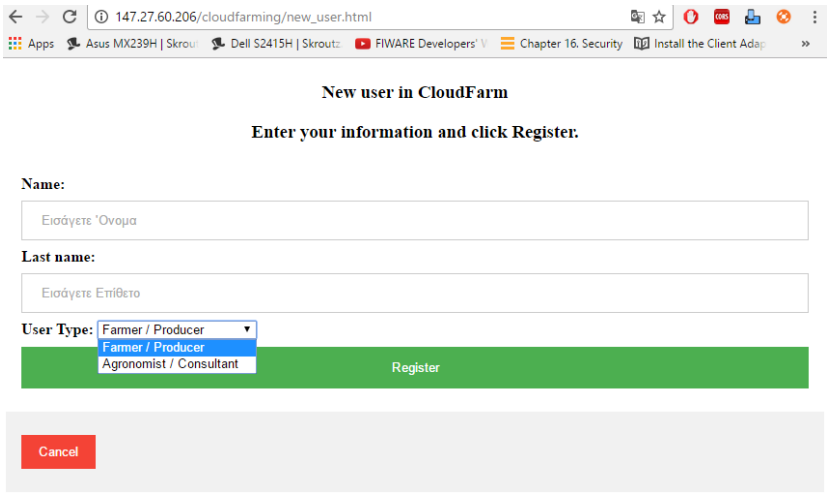


Figure 37: User Registration Web application page.

Use Case 4 – An administrator uses CloudFarm Web application

Administrator can access the Web application by logging into the CloudFarm with FIWARE account as shown in figure 27. After a successful login, the administrator will be authenticated and redirected to the administrators' dashboard with the available menu options.

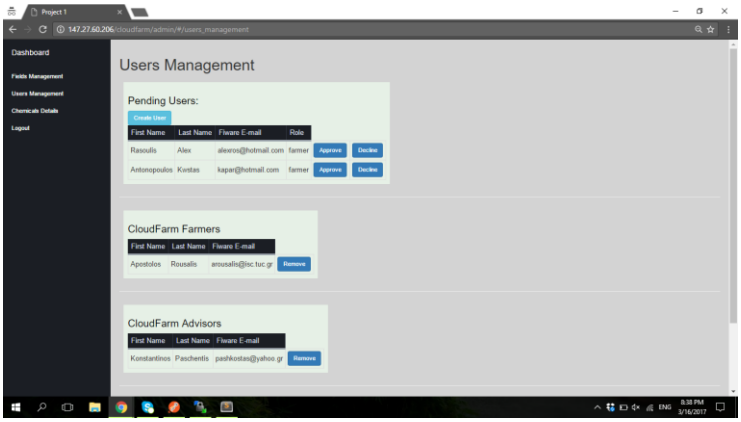


Figure 38: Administrators' Users Management page

Figure 38, shows the Administrators Users Management page. Administrators can accept or decline pending users' requests, that awaiting for approval, and can also view or remove permanent CloudFarm users.

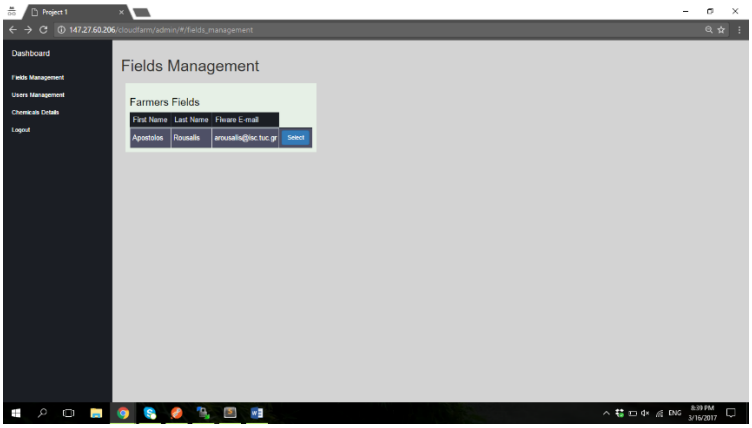


Figure 39: Administrators' Fields Management page.

Figure 39, shows the Administrators' Fields Management page where administrators can manage farmers' fields. The page displays all CloudFarm farmers and an administrator can select a farmer in order to manage farmer's fields. Figure 40, shows the fields management page, of a selected farmer where the administrator can view or insert fields. This page is familiar with the farmers' field management page presented in figure 29.

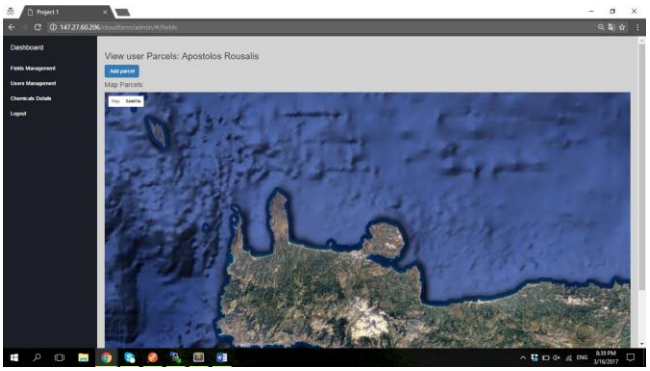


Figure 40: Administrator uses farmer's fields management page.

Presentation of Mobile Application

Figure 41, shows the main page of the mobile application. This page provides a summary of the last crops, harvests and fertilizations. The navigation bar has four options, Home, Fields, Crop and Storehouse.

Figure 42, shows the Fields page, the farmer can view his fields status.

Figure 43, shows the Crops page, the application first detects the field in which the farmer is located using GPS and asks the farmer to verify that he is currently located inside that field and after the verification the crop page provides all the crop options. If the field has not a crop, the only available option is the crop creation shown in figure 48. If the farmer is not located in the suggested field because of wrong GPS coordinates, there is a cancel option where the farmer can choose manually the field he is currently located.

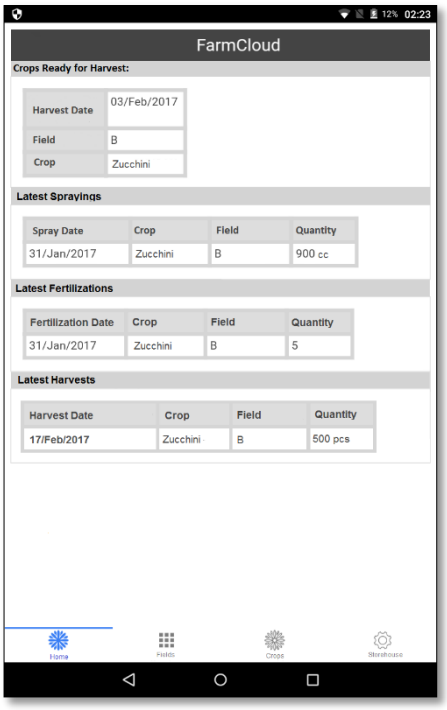


Figure 41: The main page.

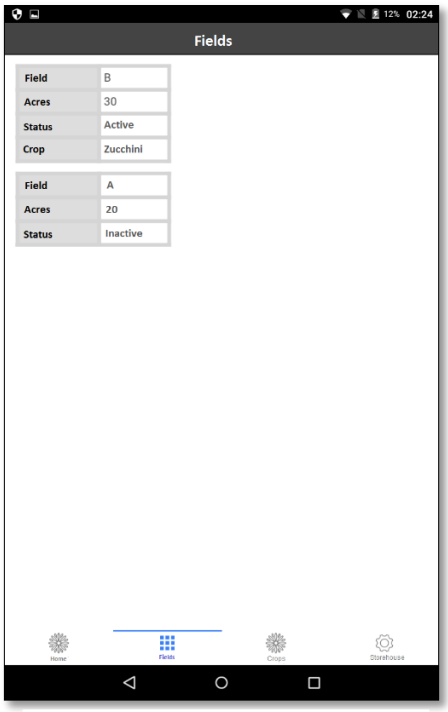


Figure 42: The Fields page.

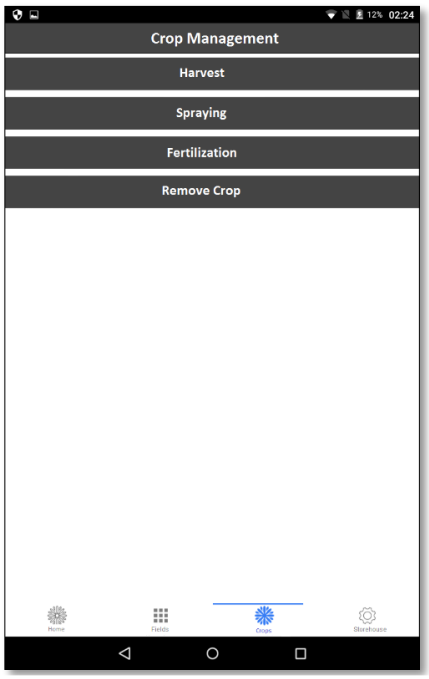


Figure 43: The Crops page when first locates the farmer.

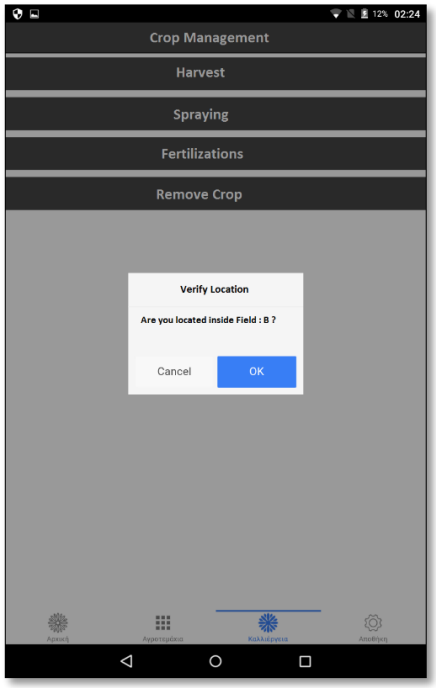


Figure 44: The Crops Page when provides the crop menu of agricultural activities.

Figure 44, shows the available crop's option after the farmer verifies that he is located inside a field. If the field has an active crop the page will show the four available agricultural activities, harvest, spraying, fertilization and crop removal.

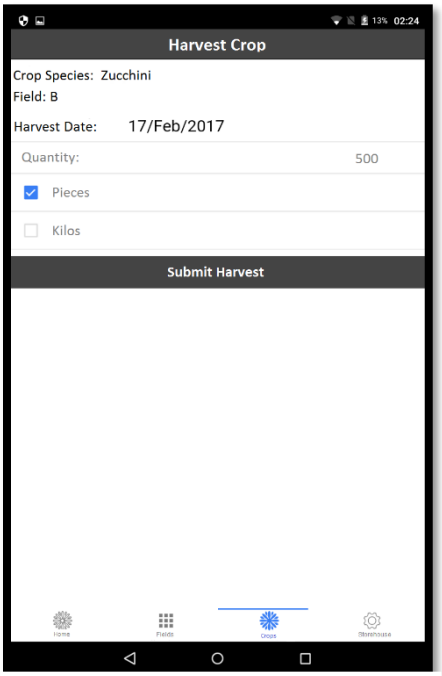


Figure 45: Crop's harvest page.

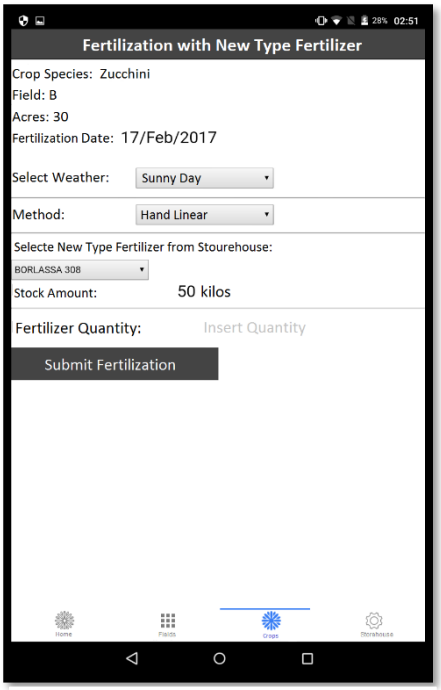


Figure 46: Crop's Fertilization page.

Figure 45, shows the crop's harvest page. The farmer inserts the harvested quantity in kilograms or pieces. The harvest date is automatic inserted and it's the current date of the farmer capturing the harvest activity through the mobile application. Before the harvest submission, the application checks all the selected parameters and asks the farmer to confirm the harvest.

Figure 46, shows the crop's fertilization page. First the farmer selects the weather condition, the fertilization method and the fertilizer from the storehouse fertilizers list. Then the application displays the stock amount of the fertilizer in the storehouse and the farmer inserts the used quantity (kilograms) of fertilizer. Before the fertilization submission, the application checks all the selected parameters and asks the farmer to confirm the fertilization.

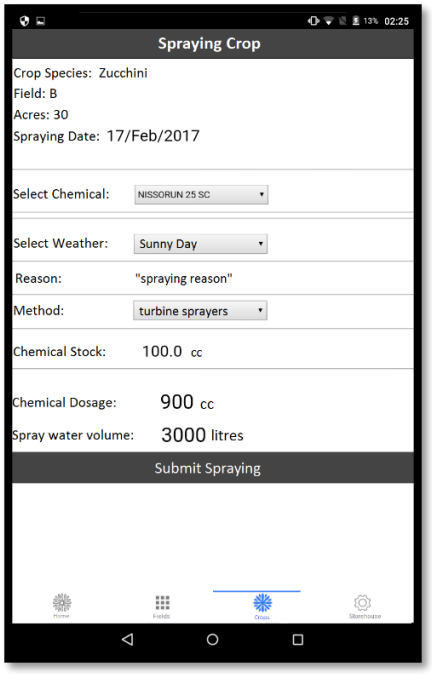


Figure 47: Crop's Spraying page.

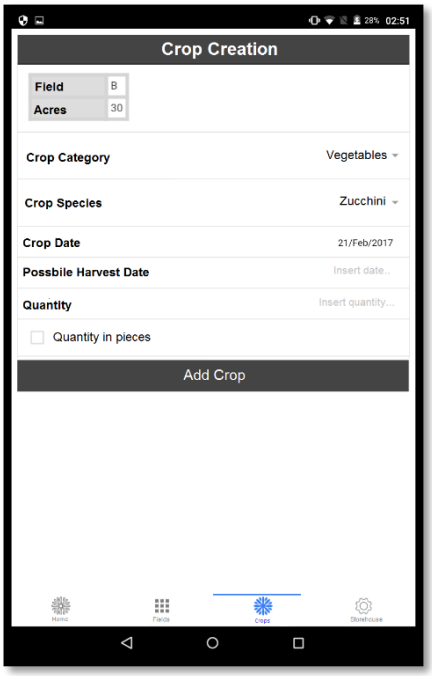


Figure 48: Crop creation page.

Figure 47, shows the crop's spraying page. First the farmer selects a chemical through a list of permitted chemicals, suitable for the specific crop species. Then the farmer selects the weather condition, the spraying method and inserts the spraying reason. At the end, the applications display the stock quantity of the selected chemical in the storehouse and the automatic calculated dosage which the farmer must use for this spraying. Before the spraying submission, the application checks all the selected parameters and asks the farmer to confirm the spraying. After a successful spraying the crop's harvest date is automatically updated to a new one, depending on chemical's waiting days due to chemical's side-effects.

Figure 48, shows the crop's creation page. A farmer selects crop's category and species through a list, and inserts the possible harvest date by using a calendar like in figure 52 and then inserts the quantity of plant pieces. The crop date is automatic inserted and it's the current date.

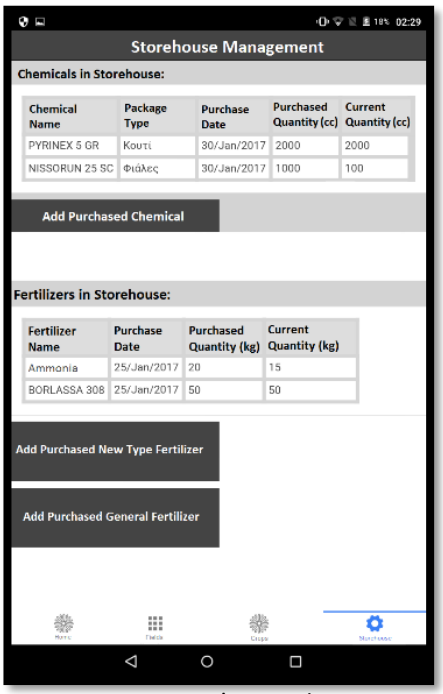


Figure 49: Farmer's Storehouse page with the options to add a purchased supply into the storehouse

Figure 50: First step of the option, Add a purchased chemical, into the storehouse. All the chemicals are listed in a searchable table.

Figure 49, shows the storehouse page. The farmer can view all the purchased supplies details such as purchase date, current quantity and can also add a supply in the storehouse.

Figure 50 and 51, show how a farmer can add a chemical supply in the storehouse. First the farmer must search and select a chemical and then the chemical details will be displayed in order to inform the farmer that he is trying to insert the right chemical. Then the farmer must select the supply's package type and package size according to the registered chemical details and chemical's packages. The package size is measured in cube centimeters. Finally, the farmer must insert the purchase date of the package by clicking in the purchase date field, where a calendar pops-up, as shown in figure 52, in order to select the date. Before the item insertion, the application checks all the selected parameters and asks the farmer to confirm the item insertion. After a successful insertion, the inserted item can be found in the storehouse main page as a purchased item. The insertion of a purchased fertilizer in the storehouse, is the same procedure as the chemical insertion with the only difference that the farmer must select if the fertilizer is a new type or a general type of fertilizer.

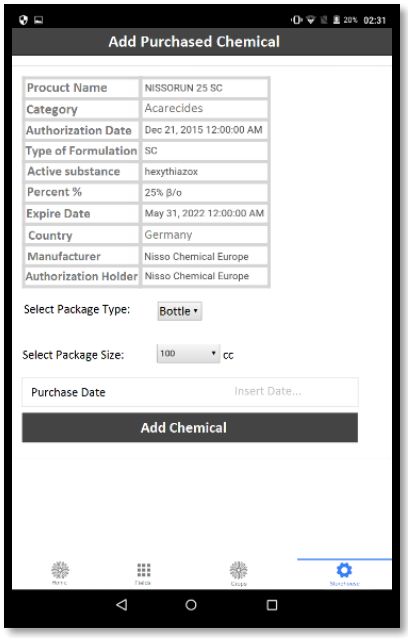


Figure 51: Second step of the option, Add a purchased chemical, into the storehouse. The selected chemicals details are displayed as the supply's package types and sizes.

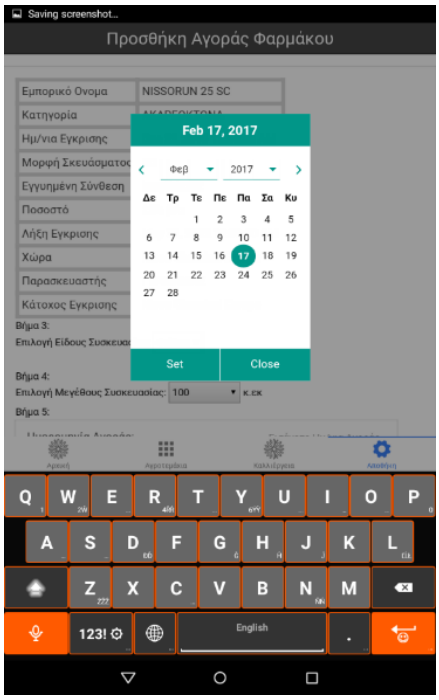


Figure 52: Shows the calendar pop-up page. The user can select the month, the year and the date.

Chapter 4 – Performance Evaluation

In order to evaluate the system performance, we chose a demanding use case scenario in terms of computing resources. This scenario is the execution of a spraying activity. Inserting a spraying into the system communicates numerous system services such as Keyrock Identity Manager (Authorization Service), Sprays Service, Crops Service, Storehouse Service and Storage Service. This scenario is a POST request (/users/{user_id}/farmer/crops) to the REST API.

All the measurements are taken from our virtual machine presented in chapter 3.7.14 which hosts all the CloudFarm services.

4.1 API Performance

In this section, we evaluate the performance of the REST API which runs in a Tomcat Server, with apache benchmarking tool called ApacheBench (ab)[50]. It shows how many requests per second the server is capable of serving.

An example on how to run a benchmark using ab tool through a Linux machine is shown below:

ab -n 1000 -c 5 http://host:port/path

- -n, is the number of requests submitted to server.
- -c, is the **concurrency** number and is the number of simultaneously requests to perform at a time to the server.

The CPU and RAM usage measurements are taken using the Linux command top[51] which provides a dynamic real-time view of the resources of a running system.

Test 1: Multiple requests submitted one at a time (concurrency = 1)

In this experiment, we set concurrency equal to 1 in order to run 2000 requests which were submitted one at a time to the server (sequential requests). Concurrency equal to 1 means that all the requests are submitted sequential to the API. The test completed 2000 sequential requests in 164.2 seconds with total data transferred 1556 kb. Before we execute the benchmark the virtual machine CPU usage was at 1%. While benchmark was running the highest virtual machine CPU usage was 24% which occurred after 1800 completed requests. More test results are shown in table 11 and table 12. As expected, while the test was running we noticed that there is a low CPU usage, due to the fact that only one request was executed per time.

Number of sequential requests	Fastest response time (ms)	Mean response time (ms)	Slowest response time (ms)	Tranfer Rate (kb/sec)	Virtual machine max CPU usage (%)	Virtual machine max RAM usage (%)
2000	124	165	320	9.22	24 %	6 %

Table 11: shows the test results of 2000 requests performed one at a time.

[50] https://httpd.apache.org/docs/2.4/programs/ab.html
[51] http://www.computerhope.com/unix/top.htm

Percentage of the requests served within a certain time	Time (ms)
50 %	160
66 %	164
75 %	168
80 %	168
90 %	172
95 %	176
98 %	180
99 %	185
100 %	320 (longest request)

Table 12: shows the time distribution of 2000 requests performed one at a time.

Test 2 - API Performance: Multiple requests performed simultaneously (concurrency = 50)

In this experiment, we set the concurrency number to 50 in order to run 2000 requests where simultaneously 50 different users, are doing 40 sequential requests. The test completed in 8.9 seconds with total data transferred 1556 kb. Before we execute the test the virtual machine CPU usage was at 1%. While the test was running the highest virtual machine CPU (peak) usage was 60 %, which occurred after 1000 completed requests. The test results are shown in table 13 and table 14.

Number of Requests	Fastest response time (ms)	Mean response time (ms)	Slowest response time (ms)	Tranfer Rate (kb/sec)	Virtual machine max CPU usage (%)	Virtual machine max RAM usage (%)
2000	132	221	665	170	60 %	9 %

Table 13: shows the test results of 2000 requests performed with concurrency 50.

Percentage of the requests served within a certain time	Time (ms)
50 %	208
66 %	221
75 %	231
80 %	241
90 %	271
95 %	313
98 %	398
99 %	466
100 %	665 (longest request)

Table 14: shows the time distribution of 2000 requests performed with concurrency 50.

The above results show that the API performance is quite good, beside the incensement of CPU and RAM usage. This occurs because parallel requests bind the system resources until they get the response from the system. Each request to API can generate multiple sub-requests to CloudFarm services in order to return the required information. The demand of processing all those requests at the same time, increases the CPU and RAM usage of our virtual machine.

Test 3 - API Performance: Multiple requests performed simultaneously (concurrency = 200)

In this experiment, we set concurrency equal to 200 in order to run 2000 requests where simultaneously 200 different users, are doing 10 sequential requests. The test completed in 10.2 seconds with total data transferred 1556 kb. Before we execute the test the virtual machine CPU usage was at 1%. While the test was running the highest virtual machine CPU (peak) usage was 99 %, which occurred after 1200 completed requests. The test results are shown in table 15 and table 16.

Number of Requests	Fastest response time (ms)	Mean response time (ms)	Slowest response time (ms)	Transfer Rate (kb/sec)	Virtual machine max CPU usage (%)	Virtual machine max RAM usage (%)
2000	260	860	3868	238.9	98 %	12 %

Table 15: shows the test results of 2000 requests performed with concurrency 200.

Percentage of the requests served within a certain time	Time (ms)
50 %	792
66 %	855
75 %	906
80 %	948
90 %	1142
95 %	1708
98 %	1846
99 %	1933
100 %	3868 (longest request)

Table 16: shows the time distribution of 2000 requests performed with concurrency 200.

The above results show that the API struggles to handle all requests because the CPU is reaching the highest limits. The above results show, that after increasing more the number of parallel requests to the API, the system performance drops dramatically. This happens because more parallel requests bind more system resources, until the time they will get served. Each request to API can generate multiple sub-requests to CloudFarm services in order to return the required information. This demand of processing all those sub-requests at the same time, increases the CPU and RAM usage of our virtual machine. When the CPU approaches its limits, all the requests at that time are delayed because they CPU cannot handle more requests at that time.

Chapter 5 – Conclusion and Future Work

5.1 Conclusion

The goal of this thesis was to create a system for managing and monitoring farmers' farms and their crops data using services which take advantage of Cloud Computing capabilities, combined with the idea of the Internet of Things. To achieve that, we implemented a mobile application for smart-phones or tables which is used by farmers to perform agricultural activities inside the field and a web application which can be used by farmers and advisors in order to monitor the farm data. As a result of this thesis we developed the Front-End and the Back-End of CloudFarm system. The mobile application and the web application consist the Front-End, and the Back-End is all the implemented services that runs in our cloud infrastructure. The developed system is further evaluated by load testing to ensure the applicability of CloudFarm under severe load conditions.

CloudFarm will be an essential part for farmers, helping them to manage their farms organization, supervision and certification. It will help farmers to smartly monitor their farm and to have access to useful information that will help them make the best decisions. They can also be able to view in every moment the efficiency of every crop and they can act objectively.

Summarizing the most important CloudFarm benefits for farmers:

- CloudFarm can be accessed on mobile devices or tablets or through a Web application via a Web browser, anytime, from anywhere.
- Captures the performed agricultural activities (sprayings, fertilizations, harvests) directly from field using the mobile application, even without an internet connection.
- Knowledge, in real-time, the situation of the farm.
- Knowledge of the crop history, activities and the used supplies.
- Adopts Good Agricultural Practices (G.A.P) in order to grow harmless products of higher quality, because food safety is a major concern for all food producers and handlers.
- Reduces the paperwork by keeping organized all agricultural data according to Good Agricultural Practices that helps farmers to be certified for their products quality.
- Elimination of human error, because the system is validating the input data on each performed agricultural activity.
- Cost reduction related to farm management. Farmers do not need to purchase expensive infrastructure (software, hardware) and aren't responsible for infrastructure upgrades and maintenance, which reduces cost greatly.

Our conclusions about the used technologies during CloudFarm system development are listed below:

- Cloud computing offers a variety of services ready to be used such as Keyrock Identity Manager GE of FIWARE, that ease the development of our system.
- We easily deployed our cloud-based system using a virtual infrastructure (Virtual Machine) with preconfigured computing resources, provided and maintained by FIWARE Cloud Lab.

- Service Oriented Architecture (SOA) help us to divide our system into a loosely coupled components of functionality (services) that communicate with each other to provide required information and can be easily tested separately. Our SOA-based system is scalable because services can run on different servers at the same time.
- Using REST (Representational State Transfer) style architecture, we manage to provide a well-defined REST API that support the CRUD (create, read, update, delete) methods which map directly to HTTP methods (GET, POST, PUT, DELETE) and it can be easily accessed by anyone.
- Apache Cordova is a very useful tool for the development of mobile applications because we used Web technologies such AngularJS, HTML, CSS, and JavaScript, instead of learning the programming languages and frameworks of each mobile platform.

5.2 Future Work

Some improvements and additions for CloudFarm system are presented below:

- The Plant Protection Catalogue database that we obtained from the Greek Ministry of Agriculture in order to create the Plant Protection Service, does not provide all the nessecarry information about the chemicals such as the dosage per acre at specific plant species and the types of registered chemical packages. These details are provided only on PDF documents, that's why we implement the ability of administrator to add some extra chemical details, because we couldn't find the given details in any digital form. It will be a huge improvement if the Greek Ministry of Agriculture provide all chemical details through a Web Service, helping in the developers work.
- At this point the CloudFarm system supports only the management of vegetables but it can be expanded with new types of plant categories such as trees.
- Implementation of a Financial Management Service which will provide data analytics by using the collected data and offer smart financial analysis such as profitability analysis per field, crop and farm, cost analysis per field and more.
- Implementation of a Task Scheduler Service where farmers can schedule agriculture activities and can be notified when a scheduled activity is delayed.

References

[1] The NIST Definition of Cloud, by Peter Mell, Timothy Grance http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf ,retrieved on February 2017.

[2] A view of Cloud Computing by Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia : http://cacm.acm.org/magazines/2010/4/81493-a-view-of-cloud-computing/fulltext , retrieved on February 2017.

[3] Disadvantages of Cloud Computing, March 17, 2015 by Sudhi Seshachala. http://cloudacademy.com/blog/disadvantages-of-cloud-computing , retrieved on February 2017.

[4] The Internet of Things (IoT): An Overview, Karen Rose, et.al. The Internet Society, October 2015. p. 5.

[5] Perrey R., Lycett M. Service-oriented architecture.

[6] Introducing JSON. http://www.json.org/ , retrieved on February 2017.

[7] Definition of the W3C for Web Services. http://www.w3.org/TR/ws-arch , retrieved on February 2017.

[8] L. Richardson, S. Ruby, et al. Restful Web Services. O'Reilly, 1st edition, May 2007.

[9] Fielding, R. 1993. Hypertext Transfer Protocol – HTTP/ 1.1. [Document] Internet Engineering Task Force. http://tools.ietf.org/html/rfc2616#section-1 , retrieved on February 2017.

[10] L. Richardson, S. Ruby, et al. Restful Web Services. O'Reilly, 1st edition, May 2007.

[11] C. Pautasso, O. Zimmermann, and F. Leymann. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. IW3C2, April 2008.

[12] Jersey, RESTful Web Services in Java. https://jersey.java.net/ , retrieved on February 2017.

[13] Mobile: Native Apps, Web Apps, and Hybrid Apps by Raluca Budiu on September 14, 2013: https://www.nngroup.com/articles/mobile-native-apps , retrieved on February 2017.

[14] Native vs. Hybrid vs. Web Apps. Posted by Mobomo on June 30, 2016. https://www.mobomo.com/2016/06/native-vs-hybrid-apps , retrieved on February 2017.

[15] Mesbah, A., and van Deursen, A. Migrating Multi-page Web Applications to Single-page AJAX Interfaces. In 11th European Conference on Software Maintenance and Reengineering, 2007. CSMR '07 (Amsterdam, March 2007), IEEE, pp. 181–190.

[16] A.Freeman. Pro AngularJS (Expert's Voice in Web Development) 1st ed. ISBN: 1430264489

[17] Krasner & Pope 1988. A Cookbook for Using View-Controller User Interface Paradigm in Smalltalk-80. https://www.lri.fr/~mbl/ENS/FONDIHM/2013/papers/Krasner-JOOP88.pdf , retrieved on February 2017.

[18] Apache Software Foundation. Apache couchdb. http://couchdb. apache.org , retrieved on February 2017.

[19] Wikipedia CouchDB Security features overview. http://wiki.apache.org/couchdb/Security_Features_ Overview, retrieved on February 2017.

[20] Apache Software Foundation. 10.3.9. /db/ security - apache couchdb 2.0.0 documentation. http://docs.couchdb.org/en/latest/ api/database/security.html, retrieved on February 2017.

[21] PouchDB is a pocket-sized database. https://www.github.com/pouchdb/pouchdb , retrieved on February 2017.

[22] Simon Bennett, John Skelton & Ken Lunn. Schaum's Outline's UML Second Edition. Dec 2004.

[23] LR Malcolm. “Farm Management analysis: a core discipline, simple sums, sophisticated thinking”. http://ageconsearch.umn.edu/bitstream/120918/2/Malcolm06.pdf , retrieved on February 2017.

[24] Nebojsa Novkovic , Christoph Huseman , Tihomir Zoranovic , Beba Mutavdzic. “Farm Management Information Systems”. http://ceur-ws.org/Vol-1498/HAICTA_2015_paper80.pdf , retrieved on February 2017.

[25] T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.