Manuscript Details

Manuscript number	COMPAG_2017_433
Title	CLUeFARM: Integrated Web-Service Platform for Smart Farms
Article type	Research Paper

Abstract

Smart farming is a relatively new domain which has popularity nowadays. It emerged from the need to produce more with less effort and it consists of integrating modern technologies in conventional agriculture to raise the quality and the quantity of agricultural products. This paper describes an integrated web-service platform which aims to increase the quality of products grown in farms and to support business development in agriculture related fields. The platform enjoys the benefits of Cloud computing like flexibility, availability or security and it can be accessed at any time, in any place by using just an Internet connection. It allows users to locate and efficiently manage their farms providing access to different types of statistics and predictions. It also acts as a social network allowing users to interact with each other by sending private messages or posting on the forum or on the blog. In addition to describing the services offered by the platform and the interaction between them, this paper also presents the architecture of the system and the performance test results which prove the efficiency of the platform.

Keywords	Smart farming; Cloud computing; Web Services; Workflow; Data management
Corresponding Author	Florin Pop
Corresponding Author's Institution	University Politehnica of Bucharest
Order of Authors	Madalin Colezea, George Musat, Florin Pop, Catalin Negru, Alexandru Dumitrascu, Mariana Mocanu
Suggested reviewers	Lucia Vacariu, Radu Drobot, Stelios Sotiriadis, Aniello Castiglione

Submission Files Included in this PDF

File Name [File Type]

cover_letter.docx [Cover Letter]

ClueFarm.docx [Manuscript File]

highlights.docx [Highlights]

To view all the submission files, including those not included in the PDF, click on the manuscript title on your EVISE Homepage, then click 'Download zip file'.

Cover Letter

CLUeFARM: Integrated Web-Service Platform for Smart Farms

Madalin Colezea, George Musat, Florin Pop, Catalin Negru, Alexandru Dumitrascu, Mariana Mocanu

Smart farming is a relatively new domain which has popularity nowadays. It emerged from the need to produce more with less effort and it consists of integrating modern technologies in conventional agriculture to raise the quality and the quantity of agricultural products. This paper describes an integrated web-service platform which aims to increase the quality of products grown in farms and to support business development in agriculture related fields. The platform enjoys the benefits of Cloud computing like flexibility, availability or security and it can be accessed at any time, in any place by using just an Internet connection. It allows users to locate and efficiently manage their farms providing access to different types of statistics and predictions. It also acts as a social network allowing users to interact with each other by sending private messages or posting on the forum or on the blog. In addition to describing the services offered by the platform and the interaction between them, this paper also presents the architecture of the system and the performance test results which prove the efficiency of the platform.

Our paper describes a web platform that comes in handy to fulfill the needs of the farmers, giving them the possibility to manage monitor and control their farms from distance through any device that has an internet connection (a phone, a table or a personal computer). Before describing the architecture of the platform, we presented the architecture of the system which contains the platform. The platform is constituted of two main applications, decoupled from one another. We propose an architecture along with the technologies used and a motivation for choosing them, for each of these two applications and we also discuss the communication between them. We analyzed the performance of the platform through several tests which reveal the load times of the platform and the response times of the services. We also explained the results. These are the main contribution of our paper.

Corresponding Author,

Prof. Florin Pop

CLUeFARM: Integrated Web-Service Platform for Smart Farms

Madalin Colezea[1], George Musat[1], Florin Pop[1,2,*], Catalin Negru[1], Alexandru Dumitrascu[3], Mariana Mocanu[1]

[1]Department of Computer Science, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania

Emails: madalin.colezea@stud.cs.pub.ro, george.musat@stud.cs.pub.ro, florin.pop@cs.pub.ro, catalin.negru@cs.pub.ro, mariana.mocanu@cs.pub.ro

National Institute for Research and Development in Informatics (ICI), Bucharest, Romania

Email: florin.pop@ici.ro

[3]Department of Automatic Control and Systems Engineering, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania

Email: alexandru.dumitrascu@acse.pub.ro

Abstract

Smart farming is a relatively new domain which has popularity nowadays. It emerged from the need to produce more with less effort and it consists of integrating modern technologies in conventional agriculture to raise the quality and the quantity of agricultural products. This paper describes an integrated web-service platform which aims to increase the quality of products grown in farms and to support business development in agriculture related fields. The platform enjoys the benefits of Cloud computing like flexibility, availability or security and it can be accessed at any time, in any place by using just an Internet connection. It allows users to locate and efficiently manage their farms providing access to different types of statistics and predictions. It also acts as a social network allowing users to interact with each other by sending private messages or posting on the forum or on the blog. In addition to describing the services offered by the platform and the interaction between them, this paper also presents the architecture of the system and the performance test results which prove the efficiency of the platform.

Keywords: Smart farming, Cloud computing, Web Services, Workflow, Data management

1. INTRODUCTION

Nowadays, the Internet has become an indispensable need for most people. It has started as a small network that connects a few stations and it has evolved over time becoming the only global network that interconnects all the people around the world.

Lately, a new step has been made with the advent of the concept of IoT (Internet of Things). IoT is a network of "things" such as vehicles or buildings that have several sensors or actuators which allows the network to collect data or to control them from distance. The reunion of Internet and the IoT in a single network it is what is happening nowadays. This is called IoE (Internet of Everything), a network consisting of billions of devices which produces a huge amount of data and people that uses those data for different tasks and control the devices from distance. For instance, (Liao et. All, 2016) propose a system that monitors both environmental factors and growth traits of Phalaenopsis providing quantitative information with high spatiotemporal resolution.

The quantity of data produced and the need to process it to get useful information raises several problems: a lot of processing power and storage is needed to be able to analyze the data in real time. A viable solution has been shown to be the Cloud computing. It is a modern concept in computer science and it consists of a distributed system with enough processing and storage power to do the tasks described above. The user does not need to be aware of the physical location of the equipment he uses. It offers some advantages, including flexibility, availability or security.

A high number of classical fields starts to integrate new technologies in their daily activity to enhance productivity and the quality of their products. Such an area is agriculture, one of the most important sectors of activity worldwide. In classical agriculture, the farmer is the only one responsible for the management of all activities and he is the one who take all the decisions. A good decision can be taken only after an analysis, depending on many factors (temperature, humidity). This requires a lot of time and forces the farmer to be physically present at the farm location.

In a report publish by the FAO (Food and Agricultural Organization of the UN) the world population will reach 9.6 billion people by 2050 thus the food production must increase by 70 percent to feed all those people. Some of the barriers to fulfilling this request are the climate that keeps changing and the high consumption of energy and fresh water (the agriculture consuming 70 percent of the available fresh water resource) (Alexandratos and Bruinsma, 2012). FAO presents as a solution to these threats the use of "Smart Farming". The industrialization of greenhouses (named "farms" further on this paper) has led to the concept of "Smart Farm". A "Smart Farm" is a farm capable to automatically perform several actions for auto management and enables monitoring of several parameters.

This paper describes a web platform that comes in handy to fulfill the needs of the farmers, giving them the possibility to manage monitor and control their farms from distance through any device that has an internet connection (a phone, a table or a personal computer). Through the services integrated in this platform, a new approach for farming is encouraged, an approach that combines the classical agriculture with technology. Thus, a farm become a controlled environment and the farmer is helped to take the best decisions since he can supervise the activity in the farm or due to the information collected from the online community that is created.

In a context in which the agriculture already uses the advantages offered by new technologies, with existing complete hardware and software solutions consisting of a network of sensors and an application that displays to the user data collected by the sensors, the next real challenge is to find a working solution to centralize all the farms regardless of the hardware or the physical location of the farm. Mobility is another important aspect as the present solutions do not allow the farmer to monitor his farm from distance (Bojan et. all, 2015, So-In et. All, 2014).

The project[1] described in this paper aims to fulfill these shortcomings, proposing a solution with which the only requirement for monitoring one's farm is to have a device connected to the internet, thus the mobility is no longer a problem. Moreover, the data presented to the user are processed in real time and displayed in a graphical form offering statistics and correlations based on them.

In addition, other useful services are integrated making the platform a place where the farmers can share their thoughts or get information that helps them to take the best decisions which will increase productivity, this being the primary goal of the agriculture for the years to come.

This paper describes an intelligent platform designed exclusively for farmers and it is meant to respond to their needs. It helps the users, the farmers, to manage and supervise their farms from any device with an intuitive, user friendly and appealing interface.

To be able to meet the other goals we must first implement a web platform. It should be scalable and performant so it can be deployed in a Cloud environment. Its main purpose is to serve as a point of access for all the services offered to the farmers. Another goal of the web platform is to provide an easy way to integrate other services in the future. Security is also a crucial aspect because the platform stores personal information about the user and his farms, information that must be kept private.

Regarding the design of the platform, this should be displayed as a dashboard with a left side menu where the user can access any of the services with just one click. The interface must be responsive, intuitive and easy to use from any device (phone, tablet or laptop). The platform can be regarded both as a farm management system and as a social network.

The farm consists of several services that fulfill the farmer's needs regarding the management and supervisions of their farms. The farm management service must allow the user to register a farm into the system by providing a couple of information about it. For a user to be able to localize a farm he must have highly detailed geographical

[1] ClueFarm: http://cluefarm.hpc.pub.ro/

information so, for simplifying this process the service must allow the specification of the location of the farm using an interactive map. The user must have the possibility to remove the farm from the platform anytime he wants.

Many of the significant decisions made by farmers are based on weather predictions because the temperature or meteorological phenomena have a bearing on the agricultural activity. Therefore, the weather service is an important one. Moreover, it must provide precise weather forecast for the following seven days, giving the users the opportunity to choose between a variety of weather providers.

The data taken from the sensors are difficult to interpret in the form in which they are generate (time series) therefore, the statistics service oversees correlating those data and offers the user the possibility to view the results in a more intuitive graphical manner for each parameter. When an event occurs, the user is informed both by a notification in the platform and by email.

The other aspect of the platform is the social network one. The main goal of this aspect is to create a strong and active online community of farmers and specialist in agriculture. The platform provides several services in order to achieve this goal. The users can create or join already existent groups, they can comment on the forum or post complex articles on the blog on different subjects related to agriculture. A messaging service is also available.

2. Related Work

Nowadays, more traditional areas began to use new technologies to increase their productivity and product quality, providing intelligence to different environments. Among them there is also agriculture, this causing the appearance of the "Smart farming" concept. Another important aspect is finding a reliable source of information that helps the user to make the best decisions [3].

The appearance of the **Internet of Things** (IoT) parading was a big step further, thus allowing connecting to the Internet of many of the objects in our surrounding. This concept was created by Kevin Ashton in 1999 (Ashton, 2009) but, its meaning has changed a lot over the years with the evolution of technology. According to (Kruize et. all, 2016) IoT has three main components:

- **Hardware** – consisting of sensors, being the main source of data and actuators that allows the remote control of different things;
- **Middleware** – responsible for storage and data analysis;
- **Presentation** – widely accessible tools for visualization and interpretation of data;

Regarding IoT one of the biggest challenges is storing and interpreting in real-time the enormous quantity of data generated by the sensors. This can be done with the use of **Cloud Computing** technology. Cloud infrastructure can provide the necessary storage and processing power for doing such real-time analysis, the only limit for applications development being the human imagination.

"**Smart farming"**, also known as "precision agriculture" can be defined as a science that combines the advantages offered by new technologies with the mature agriculture industry. Smart farming is not a new concept, as it was defined a few years ago, but the research in this area has gained momentum lately with the advent of new technologies that makes the objectives of Smart farming realizable (Davis et. all, 2016). The main purpose of smart farming is to create a new crop management style to increase productivity, optimize planting and harvesting and reduce pollution. Such a crop management system can be a wireless sensor network (WSN) which generates data, collected and stored in a database. The analyses of these data help the farmer to make the right decisions. This system is also constituted of an actuator system responsible for task automatization.

Over the years, a lot of farm management systems have been developed, some of them offering complex solutions while the others are trying to solve specific problems. Most systems come as both hardware and software solutions, the great actual challenge being the development of a general solution loosely coupled with the hardware. Farm Management Information Systems have evolved from simple farm recordkeeping into sophisticated and complex systems to support production management. The aim of these systems is to "satisfy demands, to reduce production costs, comply with agricultural standards, and maintain high product quality and safety" (Fountas et. All, 2015). As mentioned above one important aspect is the existence of a reliable source of information, which can be in form of an online community. Some of these solutions are described below along with the features they offer.

OnFarm [http://www.onfarm.com] is a web platform that offers a lot of services. All farming information is disposed on a fully configurable dashboard, the user being able to add his useful widgets on it. Some of the farm

services are: farm visualization on the farm (provided by ESRI), automatic data gathering, weather widget that combines data collected from the farm with information from weather provides, a scheduler, analyzing data and their correlation to provide trends, averages and forecasts and a service for private communication. The alerting system offered by the platform can warn the user on his phone via SMS and by sending emails to any account configured by the user. The platform also offers a custom chart builder.

AgFuse [https://agfuse.com/home/about] is a social networking platform launched at the end of 2015. The platform exclusively targets the farmers. Its main purpose is to create an online community where the farmers and agriculture professionals can interact with each other by direct communication or by sharing information. Once a user created an account he must complete his profile by specifying his domains of interests, the platform giving him suggestions based on it. After that he can connect with other users, join or create groups where he can post or read useful information.

3. System Architecture and General Implementation Details

In this section of the paper is described the architecture of a scalable and performant integrated web-service platform that aims to facilitate the management of smart farms and creating an online community of farmers among with some general implementation details. This platform is part of a more complex system described in Section 3.1. The rest of the section focuses on describing the architecture of the platform, the integration with external services and technologies used. In Section 3.5 is presented the database schema of the platform.

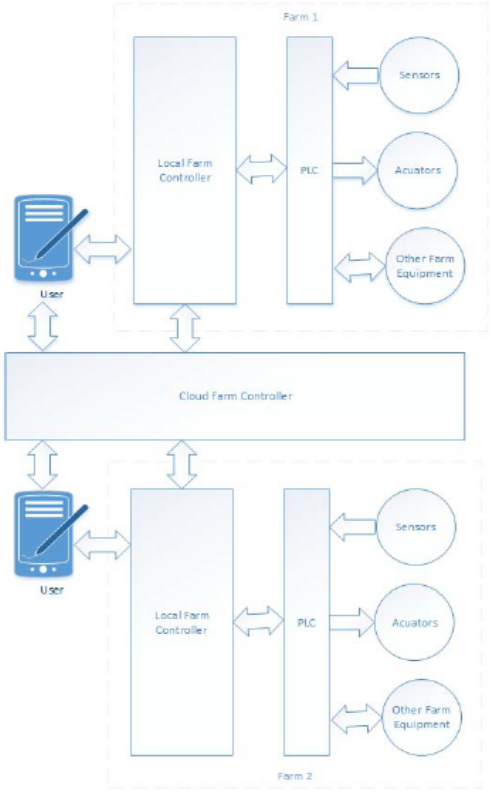

Figure 3.1 General Cloud-Based Architecture of the System (Mocanu et. all, 2015).

3.1 Cloud-Based Architecture for Smart Farms Management

In Figure 3.1 is presented a general cloud-based architecture of a system for farm management proposed in (Mocanu et. all, 2015). It consists of two main components: Local Farm Controller(LFC) and Cloud Farm Controller(CFC). LFC is located at the farm site and offers to the farmers the possibility to control the farm. Its main purpose is to collect and store the data generated by the sensors. These data are stored in a relational database having a fixed structure. To permit to the CFC to gather the stored data the LFC must be connected to the internet and must allow the access to the database using a username and a password. LFC also works as a standalone component. Cloud Farm Controller is in fact the component that makes the system a cloud based one. This must be synchronized with Local Farm Controller, aggregates the data collected from all the farms and allows the user to

monitor his farm through the internet without any location restriction. In addition, it offers to the famers access to many useful services, also integrating some external services like maps and weather. The platform which is the subject of this paper and whose architecture is further described plays the role of a Cloud Farm Controller.

3.2 Local Farm Controller

A greenhouse monitoring system is very important for plants' climate, providing all the necessary environment conditions for the harmonious growth of the crop.

The monitoring system presented in this section is represented by a sensors' network with wireless transmission, through which measures air and soil parameter values in the greenhouse (Serrouch et. all, 2015). These values are transmitted to a base radio connected to a server. The server has the option to acquire measurement data and to process the data for graphical display various parameters over time.

The sensors' type that makes up the system consists of air and soil sensors. The air sensors are represented by the air humidity sensors, the air temperature sensors and dew point calculation, the solar radiation sensors, and the leaf wetness sensors. The soil sensors are represented by the soil moisture sensors, the soil temperature sensors, and the soil water content sensors. All these sensors are shown in Figure 3.2.

Figure. 3.2 The type of sensors.

All the sensors are connected to wireless nodes for remote transmission of measured values. One wireless node has 4 ports to connect the sensors. Also, the node has a dual power source: rechargeable batteries and solar cell. The nodes have XMesh low-power networking protocol to provide plug-and-play network scalability and to extend the range of coverage.

All the wireless nodes transmit data to a base-radio, which collects all the parameters' values measured by the sensors. The base-radio are connected using a USB cable to the server. The server runs a Linux-Debian distribution and it has a sensor network management and data visualization software packages, named *XServe* and *eKoView*, respectively. Figure 3.3 shows the type of wireless node and the base radio.

Figure. 3.3. The wireless node and the base-radio.

The complete architecture of the wireless sensors network is presented in the Figure 3.4. Each node transmits data from sensors every 15 minutes. The software installed in the server computes values for every hour, day or month for long time statistics and store the parameters' values into a *SQLite* database. From this database, the users can export data in *.csv* file format for processing by various applications or simply for back-up.

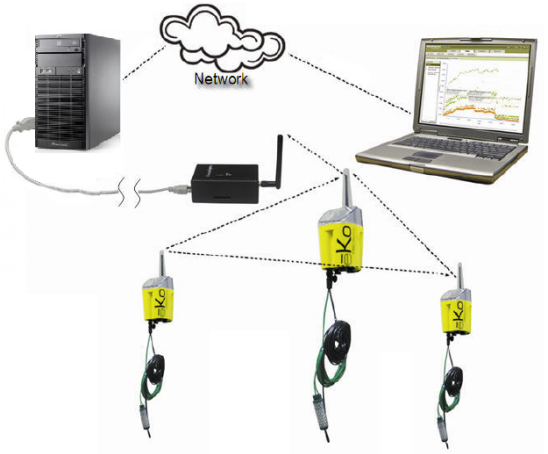

Figure. 3.4 The complete architecture of the wireless sensors network.

Figure 3.5 shows the *eKoView* web interface implemented in server, which allows users to make various settings:

- setup and configure the wireless network;
- create user-defined map to view the wireless nodes across overall network;
- manage configurations for user-defined chart;
- create trend charts of all parameters' values across customized time spans;
- real-time visualization data, which gives users the control needed to manage crop health;
- view details of measured parameters for individual wireless node;
- monitor network performance and health of each node;
- set alert levels, run report and get notifications via SMS or email.

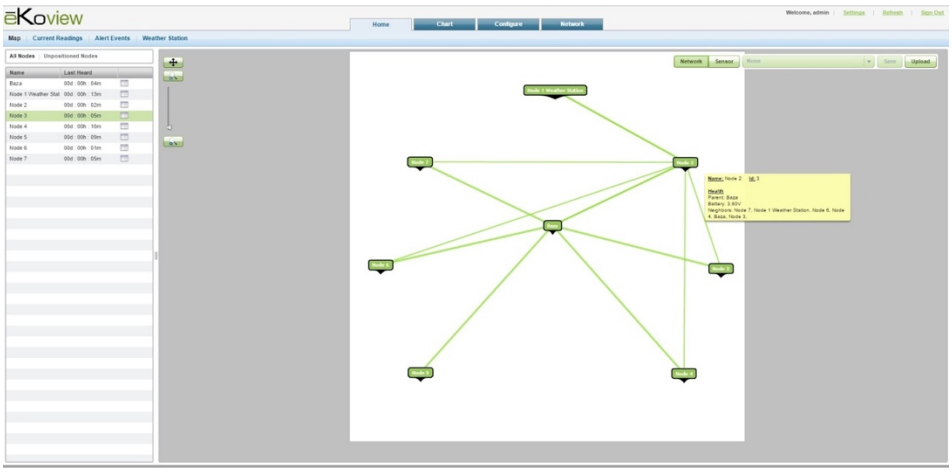

Fig. 3.5. The *eKoView* web interface.

It is observed that the wireless sensors network is composed by 7 nodes: the first nod (top of the map) transmits data from a micro-weather station and other 6 nodes (marked from 2 to 7) transmit data from greenhouse climate. All the nodes communicate their values to the base-radio (in center of the map), and then data are sent to the server.

The Figure 3.6 show the evolution graphs of two relevant parameters of the greenhouse climate - the ambient temperature and the ambient humidity. The user can choose between viewing the data, for example in Figure 3.6 data is displayed over the last 11 days, between March 20th and March 30th 2017. Also, if the user places the mouse

on the chart, one can see the information that provide the number of node which transmits data, the port number which connects the sensor, the measured values at the time, and the date of measurement.

Fig. 3.6. The ambient temperature and humidity parameter's values.

3.3 Cloud Farm Controller - General Architecture

The general architecture of the platform consists of two main applications: the front-end application, described in Section 3.3 and the back-end application described in Section 3.4. In the Figure 3.7, it is represented the two applications, along with the external components with which they interact. Both applications are hosted in the Cloud and they can be considered independent of each other. For now, on, for simplicity, we will call the front-end application "client" and the back-end application "server". The communication with most external services and with the databases (farm local databases and cloud database) is achieved through the server while using services as Google Maps and Google Places is achieved directly by the client. For sending emails the communication with the email server is made using SMTP (Simple Mail Transfer Protocol) protocol.

For most services, the communication between the two applications it is made through the REST services (over HTTP protocol) exposed by the server, the messages being formatted as JSON. For real-time notification service to obtain the smallest possible delays we use STOMP messages over a web socket.

JSON stands for JavaScript Object Notation. It is a text format used to transfer data over the network. The advantages of using JSON over XML is that it is more compact, more readable and the amount of data transferred is less because of it is format: list of values (key/value pair) [http://www.json.org/].

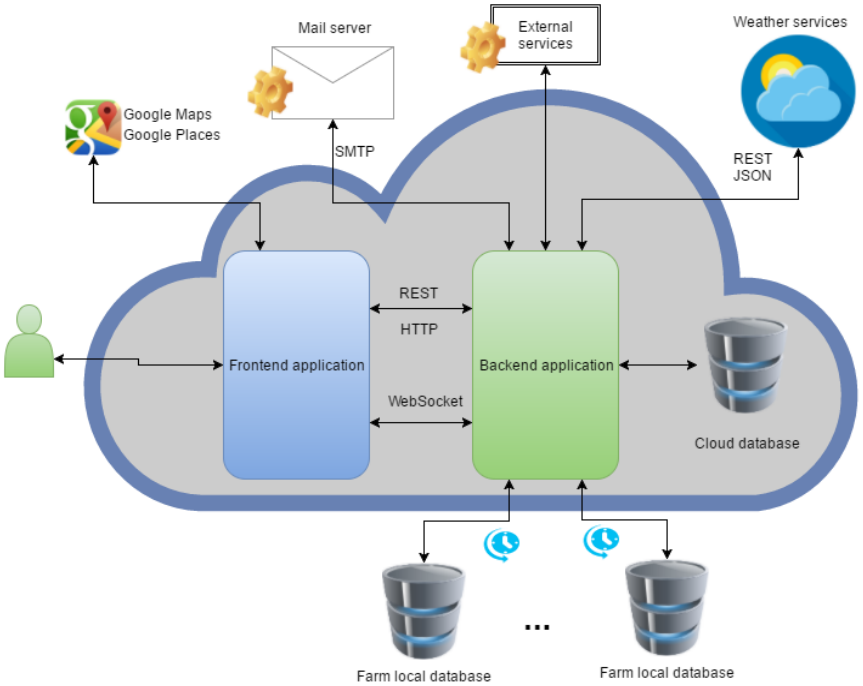

Figure 3.7 General Architecture of the Platform.

REST stands for Representational State Transfer and is a modern architectural style for building scalable and maintainable API's for web applications. A common approach, preferred in this platform too is to use HTTP protocol for communication and build an interface with CRUD (Create, Read, Update, Delete) methods for manipulating domain objects based on the standard methods of the protocol (POST, GET, PUT, DELETE) [http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html].

STOMP [https://stomp.github.io/] is a simple text-oriented messaging protocol. To configure a STOMP channel between a client and a message broker (configured on the server) it is necessary to prior establish a web socket connection. WebSocket (Fette, 2011) is a protocol that enables two-way communications between a client (a web browser in our case) and a server. At transport layer, it user TCP, the initial handshake being made with HTTP messages. After the web socket connection and the STOMP channel are established, messages from both client and server can be easily sent with few bandwidth overhead and few processing power from the client.

3.4 Front-end Application Architecture

The front-end application exposes to the user a pleasant, performant and intuitive interface through which he can access all the services exposed by the back-end application. This interface is responsive, so the user experience on the platform is of high-quality regardless of the device used (laptop, PC, table or phone). Functional details of the application are presented in Section 4, so in this section we present the structure of the application and architectural decisions related to the selection of the technology stack. The choice of technologies is a very important step in the development of any application because this influence the performance and the entire development stage. Thus, the technologies must be carefully chosen and compatible one with each other.

The functionality of the front-end application is implemented in AngularJs framework (Green, 2012). We choose this framework because is a MVC (Model View Controller) framework, compatible with all the browsers and it provides features that help in fast development of scalable and modular applications. These features include the two-way data binding mechanism (automatic synchronization between the model and the view, having an important influence on increasing user experience, the user receiving instant feedback from the application) and dependency injection mechanism (it helps on modularization and on component reuse). Due to the modularity provided by this framework, the integration of other components is simplified. For markup and style, we use HTML5 and CSS3.

To easily implement the functionalities of the platform and to create a pleasant design we use some libraries including:

- **Bootstrap** – it is the most popular HTML, CSS and JavaScript library for building responsive web pages. We use this library to make the application responsive with little effort from the developer. It also offers several reusable components like modal, dropdown, tab or carousel;
- **Font Awesome** – it is a CSS library that offers easy customizable (regarding the size or the color) vector icons. We use this to make the interface more enjoyable and intuitive;
- **Angular Chart** – it is an AngularJs module which contains a set of directives for building different types of reactive charts like line chart, pie chart or bar chart;
- **UI Bootstrap** – it is one of the most popular AngularJs library. It contains a large set of directives built on top of Bootstrap's markup and CSS. Thanks to the functionalities offered by this library the developing effort is considerably reduced;
- **SweetAlert** – it is a JavaScript library used to replace standard JavaScript alerts with ones which are easy configurable and looks better. We use it to confirm some actions made in platform;
- **AngularJS** ui-select – it is a configurable AngularJs directive which replaces standard selectors offering a lot of functionalities including multiple selection. We use this for all selectors of the platform.
- **Angular UI Grid** – it is an AngularJs directive which offers a grid with sorting, filtering, editing or grouping functions.

As shown in Figure 3.8 the front-end application is a SPA (Single Page Application) thus, when the application is accessed just one HTML page is loaded ("index.html").

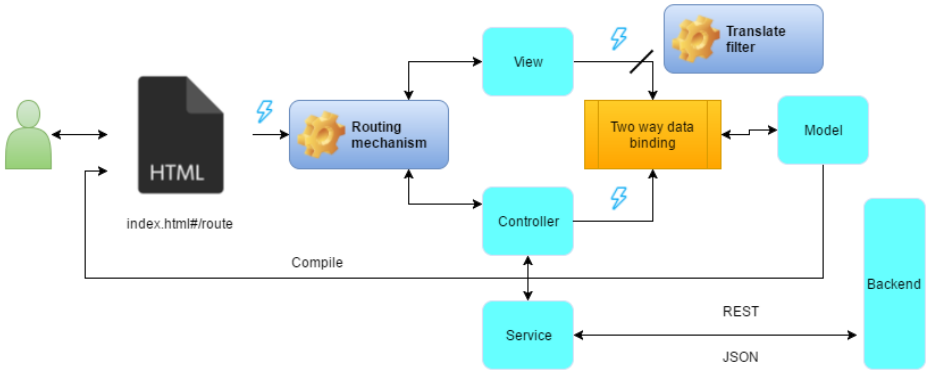

Figure 3.8 Front-end Application Functioning.

To navigate between different pages of the application is used a routing mechanism (implemented with an AngularJs module called AngularUI Router). This mechanism makes possible the association between a virtual URL a view (HTML template) and a controller (JavaScript function). When accessing a new URL, the properly view is loaded into the page, in a place specified by a special HTML directive. All the functionality corresponding to that view is implemented in the associated controller. The application also has configured an internationalization mechanism that uses angular-translate module. It interposes in loading a new view and replaces all the labels of the page with a translated version of it, taken from a JSON file. There is a JSON file with labels for every view and language in part. In our case, there are two JSON files for every view (one for English, respectively one for Romanian language). Being used an MVC framework, the application has a strict structure, each view interacting with the model only through the controller. The controller uses many services (which are singleton, reusable components) to achieve different functionalities or to communicate with the server. All the requests to the server are resolved asynchronously.

3.5 Back-end Application Architecture

The back-end application is the place where the platform services are implemented. It must expose several REST endpoints through which the communication with the front-end application is done and the services are accessed. Another important role of this application is the gather and analysis of the data produced by the sensors located at the farm. In a system with real time functionalities the performance is the most important aspect thus, prior to implementation the developers must choose the technology stack that best fits their requests and offers capabilities that facilitate a fast and efficient implementation of the services.

The technologies that we choose for implementing the back-end application are:

- **Java** – it is an object-oriented programming language. It offers portability, excellent performance, multithreading support, memory management, exception handling mechanism and security at the same time. We choose to implement the back-end application using this programming language because of the characteristics listed above and because it is well documented, there are numerous frameworks and libraries developed over Java language that allowing easy integration of other technologies (JDBC – Java Database Connectivity for connecting to a database, JavaMail for sending emails);
- **Spring Framework** – it is a Java platform that provides support for building Java applications (Johnson, 2004). It is based on POJO (Plain Old Java Object) and dependency injection, these two contributing to the development of loosely coupled components. Starting from the core modules of the framework a lot of other projects have been developed, some of them being used by the platform including Spring MVC (it imposes MVC architecture and offers several already implemented components), Spring Data JPA (it extends Hibernate framework and offers more functionality with less lines of codes), Spring Security (handles authentication and security aspects like securing endpoints), Spring Social (used for social login). We choose to use this framework because it imposes a comprehensive structure and comes with a lot of functionalities already implemented, allowing the developer to focus on application functionality.
- **Hibernate Framework** – this framework offers ORM (Object/Relational Mapping) functionality and in addition it implements the JPA (Java Persistence API) specification. We use this framework to work with

database tables in an object-oriented way and to perform database operations in a simplified manner, using methods already implemented.

- **MySQL** – is a RDBMS (relational database management system) owned by Oracle. We use this RDBMS over others because is open-source, stable, performant and offers a lot of features.

The application is designed according to the MVC pattern with an n-layer approach (Figure 3.9). The view is represented by the front-end application, while the controller is represented by the REST Controllers (Spring components). The role of this layer is to process all the requests from the client, sends them to the model and after that return the processed data to the view. The model is divided into two layers: service layer and persistence layer. The service layer is responsible for handling all the requests from the controller and is the place where all the business logic is implemented. All the services that resides in this layer are implemented using the singleton pattern. The persistence layer oversees database operations and mapping tables into domain objects. Domain objects are POJOs used by the services and are returned to the controller as DTOs (Data Transfer Objects) which are also POJOs but with less fields (only those important). This has the effect of reducing network traffic. All the REST endpoints are secured, the client must by authenticated and to have the necessary role for each endpoint individually. For now, the platform has two roles defined: ROLE_USER and ROLE_ADMIN. To access the clear majority of endpoints a logged user must have at least ROLE_USER role.

3.6 Database schema

The database schema is presented in Figure 3.10, the tables user by the services presented in this paper being highlighted with a red polygon. It is observed that the entire database is built around two main tables: user and farm. These two also represents the central entities of the platform. The user table stores personal information about the users of the platform like username, first_name, last_name, password, email and some additional fields user by the application (activated, lang_key). This table has a many-to-many relationship with authority table (this table stores all application roles). A user has at least one role. The farm table stores formation about farm (name, location, cui and so on). The list with available soil types and weather providers are stored in tables soil and weather service, the user table having a many to one relation with the first one and a relation of type many to many with the second one. The relation between user and farm tables is one to many (a user can have multiple farms).

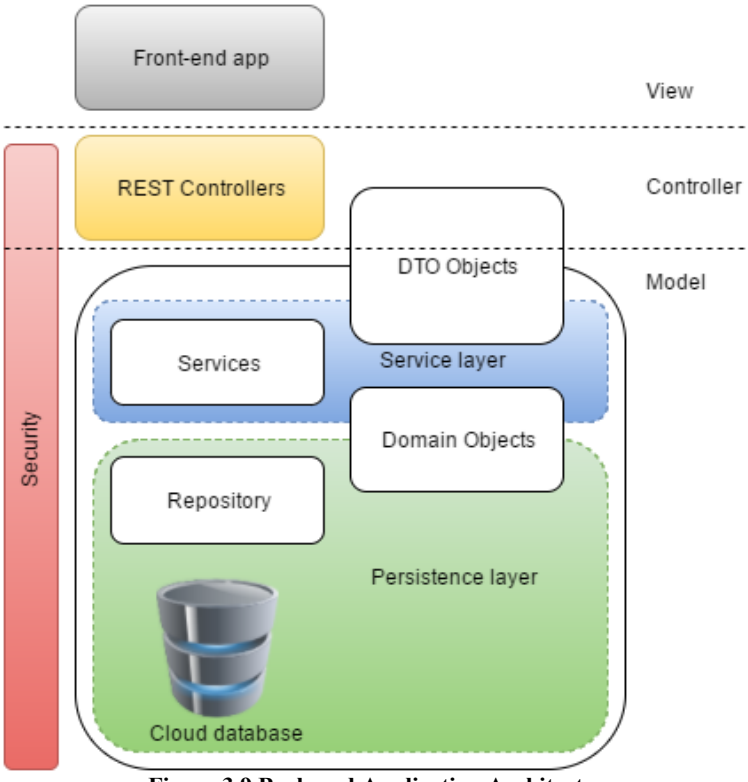

Figure 3.9 Back-end Application Architecture.

The blog service uses the blog and blog_comment tables for storing blog articles respectively article comments. Both tables have a foreign key to the user table, representing the author. Notifications history is kept in the notification table. The method used in implementing the statistics service involve the use of multiple tables (remote_data, remote_sensor, remote_data_value, hour_statistics, month_statistics, day_statistics.

4. Platform Description and Service Specification

In this section, it is presented an overview of the platform together with the specification of the implemented services. For every service, it is specified a functional description, an input, output description, the integration with external services, the moment of occurrence and how critical cases are handled, and the key aspects of its implementation.

4.1 Platform Overview

The platform has two perspectives: one for logged users and one for visitors. When a user access the platform and he is not logged he has access only to a presentation page. It is a page with a modern design where the user can view information about the platform and services offer by this (about section), a contact section and a map with which they can get an idea about the localization of the registered farm (the farms are represented on the map with an image no other information, even the name is not displayed). From this page the user can navigate only to login (where he can log in into the platform using an existing account or using social login) and register page (where he can create a new account). The only service that can be accessed when a user is not logged is the registration service.

When the user is logged, the platform looks like a dashboard. The main page contains a map which is resembling with the one described earlier, with the difference that the farm names are displayed and the farms owned by the current user are represented with a green picture. By clicking on a farm the user is redirected to the farm visualization page of the clicked farm. In the left side of the dashboard the user can see its profile picture (by clicking on it he can access his profile page) along with a menu where he can access all the services offered by the platform or can go to the view page of any of his registered farms. On its profile page the user can review his activity on the platform (view registered farms, forum interventions, blog posts). When accessing other user's profile there is a widget that allows sending a personal message to that user. When watching others farms, the number of details displayed is in accordance to the privacy options selected while adding that farm

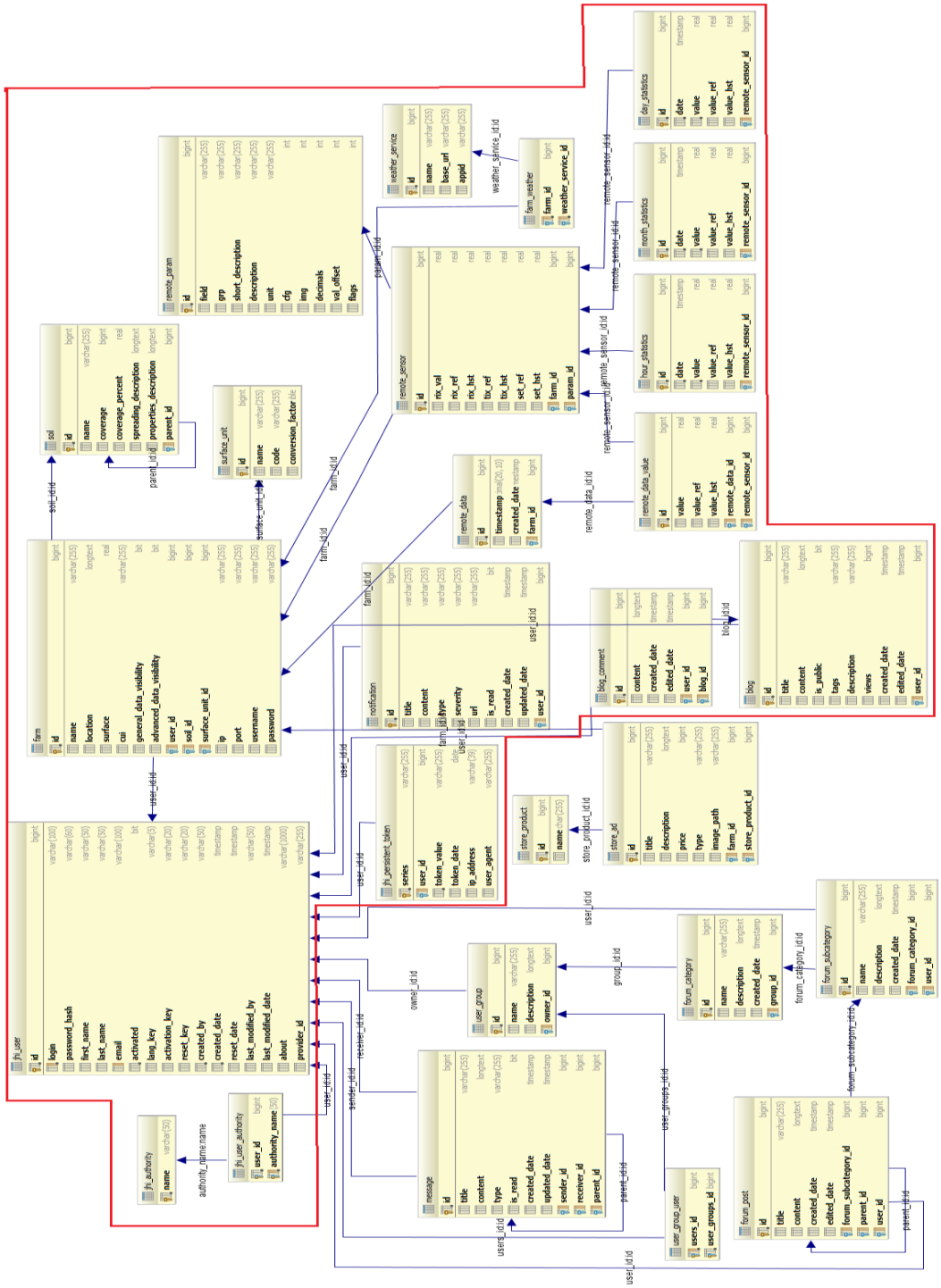

Figure 3.10 Database Structure.

In the top navigation bar (**Figure 4.1**), there is a search input where the user can search for farms, users and groups. There can be also found a new message indicator and a notification indicator. Clicking on the notifications indicator opens a dropdown menu with all the new notifications, while clicking on the new message indicator opens the messaging inbox. Also from the top bar the platform language can be changed. After authentication, the default language is the one chosen by the user while creating the account. For the moment, the available options are: Română and English.

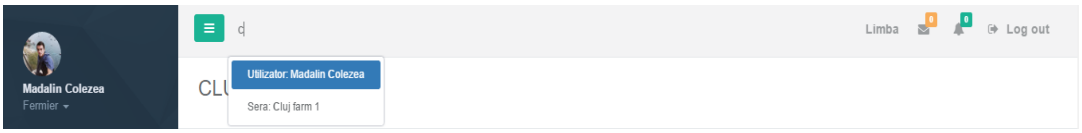

Figure 4.1 Top bar.

4.2 Identity Management Service

To gain access to other services offered by the platform, the user must have an active account. From the home page of the platform the user can choose to log in into the application, if he already has an account, or he can navigate to the registration page if he does not have one. From the log in page the users can choose to reset their password by filling a valid email and following the instructions. Further on we are going on to detail the registration process.

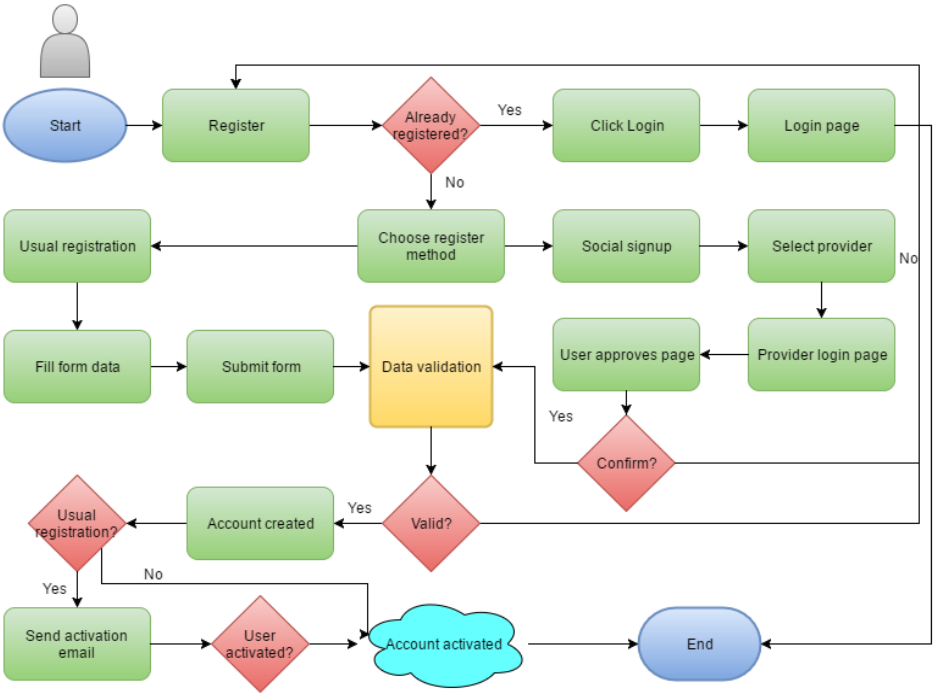

Figure 4.2 User Registration Workflow.

User registration service is a public service that allows users to register into the platform. The registration flow is represented in Figure 4.2 above. To register into the platform, the users must fill in and submit a form from the registration page. They also have the possibility to use a social sign-up, choosing an identity provider from the available ones (Google, Facebook). An identity provider is an entity able to provider user authentication and to offer to the requesting service the necessary data about the authenticated user. For the first option if the data filled in is valid, an activation email is sent. The account is activated only after the user follows the link sent in the email.

For the second choice, the process follows the provider's protocol of authentication and the user must log in to the provider's application and allow the platform to use his information. In this case an account is automatically created using the data retrieved from the identity provider. The account is also active.

For usual registration, the information that must be filled in is:

- Personal data: first name, last name, email address (used for sending activation email and other platform specific emails);
- Information used by the platform: username, password (both used for authentication) and preferred language (the default language used after the authentication);

The data is stored as JSON and is sent to the server when the form is submitted. After validation, the server responds with a success or a failure message which is also JSON formatted. In the first case the data is stored in a database and an activation email is sent.

For social sign-up, the user interacts only with the identity provider where he must authenticate with username/email and password. After the authorization process the platform communicates directly with the server and creates an account based on the information received from the server. Next, a response is sent to inform the user about the account registration and a confirmation email.

This service integrates with two external services, one for each identity provider. Both are restful services used for verifying the user's identity and obtain basic profile information. Each is a proprietary implementation of OpenID 2.0 that in the case of Google is called Google-OpenID, and in the case of Facebook is called Facebook connect.

For an account to be created the data filled in the form must pass field validations. These are:

- Username – required, unique field and must have at most 50 characters;
- First name, last name – required and must have at least 2 characters and at most 50 characters;
- Email Address – must comply email pattern and have at most 50 characters;
- Password – required and must have between 5 and 50 characters. For safety reasons the user must confirm the password and in case they do not match validation fails.
- Preferred language – a string in a predefined list (Română, English);

Validations are done both on the client and on the server-side. If client-side validation does not pass, the submit button of the form is disabled and the wrong field is indicated. If server-side validation fails, the server replies with a message indicating where the problem is to be found.

Regarding social sign-up, the creation of two different accounts with the same username/email is not permitted. Also, the refusal of the user to allow the platform to use their profile information during authorization process prevents the creation of the account.

Concerning forgot password functionality, the email address filled in by the user must belong to an active account, otherwise the server replies with an error message.

The user registration service is a REST service. The new account details are sent in the HTTP POST method's body and after validation the server responds with 201(CREATED) or 400(BAD_REQUEST) code along with a representative message. For security reasons the password is stored as a hash. An activation key (a random numeric string of 20 characters) is generated and emailed afterward to the user as part of the activation URL. When the URL is clicked, the account is activated and the activation key is removed from the database. A similar process occurs also in the case of forgot password functionality. The authority associated to a newly created account is ROLE_USER, as explained previously.

4.3 Farm Management Service

Farm registration service allows farmers to register their personal farms on the platform and it can be accessed by anyone who has an active account. The farm is the main entity of the platform, therefore the number of information that must be filled in is high.

For simplicity reasons, the form for adding a farm into the platform is divided into a series of steps (more specifically, 3) each having its role. A workflow of this service is presented in Figure 4.3.

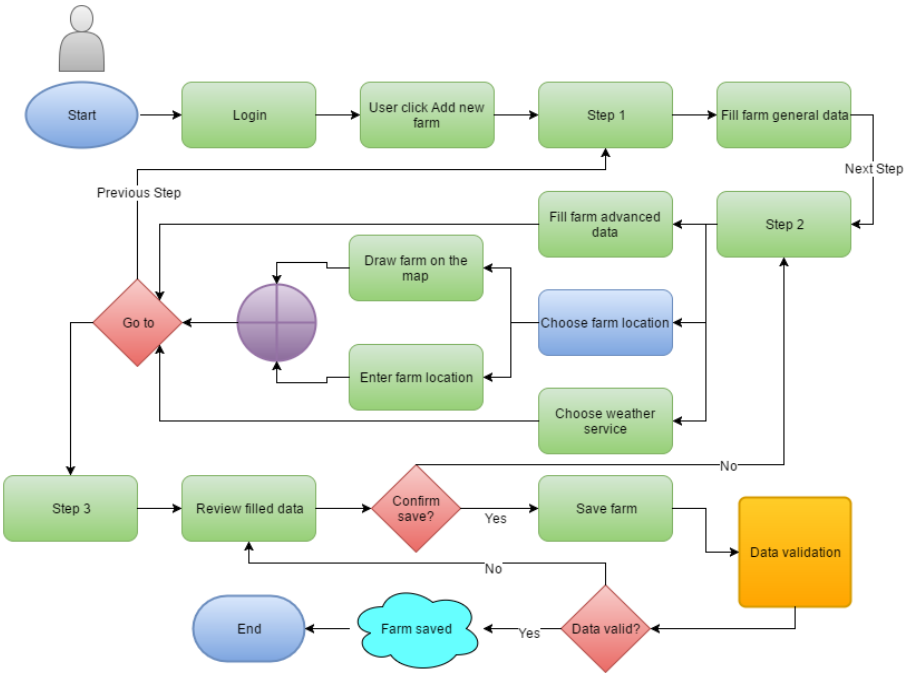

Figure 4.3 Farm Registration Workflow.

The first step (Figure 4.4) is made up of a series of general fields that the user should fill in. After this step, has been followed, the user can go further on to the second step. The fields are:

- Farm name;
- Soil class and soil type. The user has the possibility to choose from lists containing all the soil classes and types found in Romania. Selecting a soil class reduces the number of available soil types only to those which belong to the selected class. Also, selecting directly a soil type automatically fills the soil class. When hovering over a selected soil type a description about the spread of that soil type is displayed. This function helps the farmer to choose the soil correctly even if he does not know exactly the type or if he knows only the location of the farm;
- Farm surface, which can be inserted in four units: square meter, acre, hectare or are;

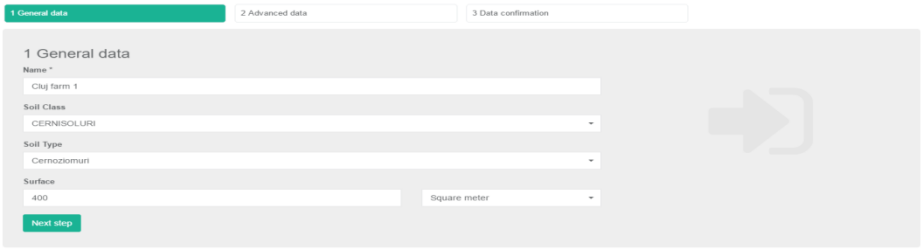

Figure 4.4 Register farm - first step.

In the second step (Figure 4.5), the user must provide advanced data about the farm. He must specify farm localization information. To do that, the farmer should draw the farm shape on the map by clicking on different locations. The selected points are automatically grouped in a polygon which can be adjusted by the user. He can also use the undo function that removes last added point or the clear function that removes all the points. For easy navigation on the map the farmers can type the name of a place on a text input with type ahead capabilities and the map focuses on it. Within this step the farmers must also choose at least one external weather provider that is used by the weather service to provide weather forecast. There are two providers available (OpenWeatherMap and Forecast.io). This step presupposes also the completion of required data for connecting to the farm local database. These are: IP, port, username and password and they are used by the statistics service. If they are not filled in the user cannot have access to any statistics. From this second step, it is possible to go back to the first step or to go further on to the third step.

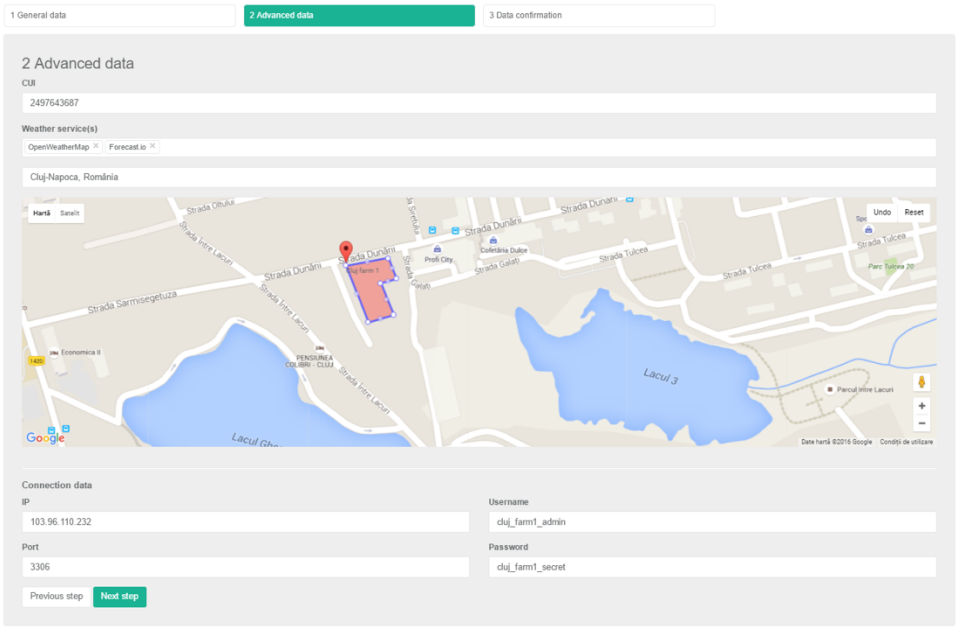

Figure 4.5 Register farm – second step.

In the third step (Figure 4.6), the user can review the data completed before and choose data visibility (public or private) for each of the previous steps. According to his choice the data is visible or not for other users. He can go back to modify the input or he can proceed to save farm instance. After successfully adding a farm, the user can view it whenever he wishes to, to edit any of the parameters and even to remove it from the system.

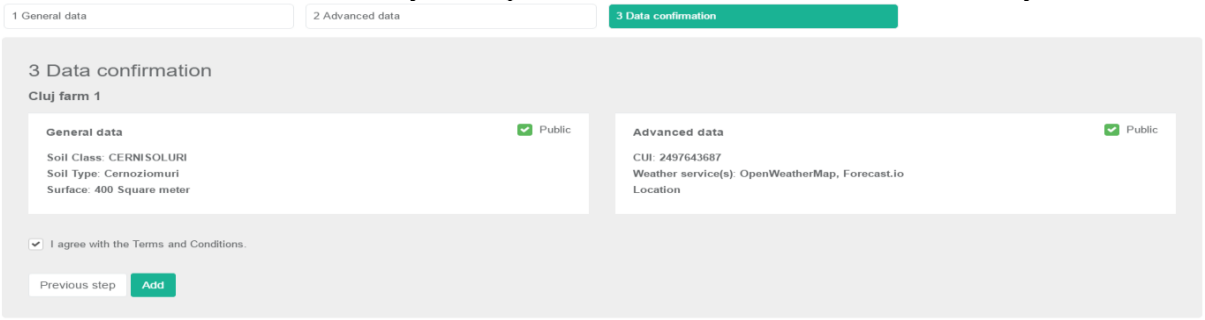

Figure 4.6 Register farm - third step.

The input consists of the data filled in the form by the user:

- Step 1: name, soil type, surface;
- Step 2: cui, farm location, weather services, IP, port, username, password;
- Step 3: data visibility;

The location is stored as a series of geospatial coordinates (pairs of latitude and longitude). Input data is formatted as JSON and is sent to the server which validates it. If the input is valid it is saved on a database. Server response is also JSON formatted and contains a success or a failure message. In second case the user must review and modify the wrong input.

This service uses two external services from Google: Google Maps and Google Places. Google Maps is a web-based service that provides detailed information about regions and sites around the world. It offers the possibility to specify farm location dynamically in a user-friendly interface. The user can draw the farm shape on the map. Google Places is a web-based service able to return place predictions based on the user's partial input. It is used to search for a place and focus the map on it.

Critical cases can occur when data does not meet field validations. These are the input validations:

- name - required and unique field;

- soil type - soil type defined in database;
- surface - numeric field;
- Farm location – list of geospatial points, required field;
- Weather service - a string in a predefined list (OpenWeatherMap, Forecast.io);
- Cui – numeric string within a specific format; required field;
- IP – must comply with the IP format;
- Port – numeric value;

Validations are done both on client and on server-side. If client-side validation fails, the save button is disabled and a message is displayed under invalid input fields. Each critical case is treated by the server which replies with messages that help the user to correct the mistakes.

The farm is drawn on the map using the API from Google Maps. It is based on an array of objects, every object representing a point on a map, having two properties (latitude and longitude). When the user clicks on the map the array is updated by adding a new point having the coordinates of location which has been clicked. The undo and clear functionalities are implemented by removing the last inserted respectively all the points of the array. After every update the map is redrawn for changes to be visible. Before submitting the form, the array of points is translated to a string with the following form: "latitude1, longitude1; latitude2, longitude2; …; latitudeN, longitudeN". This form helps in storing the location of the farm with ease.

4.4 Weather Service

Weather service is a service through which the user can view various information regarding weather and outside conditions about each farm individually, based on its location. A workflow diagram of this service is presented in Figure 4.7.

While adding a new farm the user must specify at least one weather service. For safety reasons, he can choose to use multiple services and compare the results. For each selected service the platform displays in a widget the basic weather information (temperature, rainfall, humidity and information about wind). For more detailed information and forecast for the following week the user must click on the "More details" button.

Regarding weather forecast the information taken from the weather provider and displayed by the platform are: minimum temperature, maximum temperature, humidity, precipitation intensity, precipitation probability, atmospheric pressure, wind speed together with a short description. It is possible that in time more weather providers will be available or if simply the user wants to change his initial choice the platform gives him the possibility to modify the selected providers.

The units used are:

- Celsius degrees (°C) for temperature;
- meter/sec (m/s) for wind speed;
- hectopascal (hPa) for atmospheric pressure;
- millimeter/hour (mm/h) for precipitation intensity;

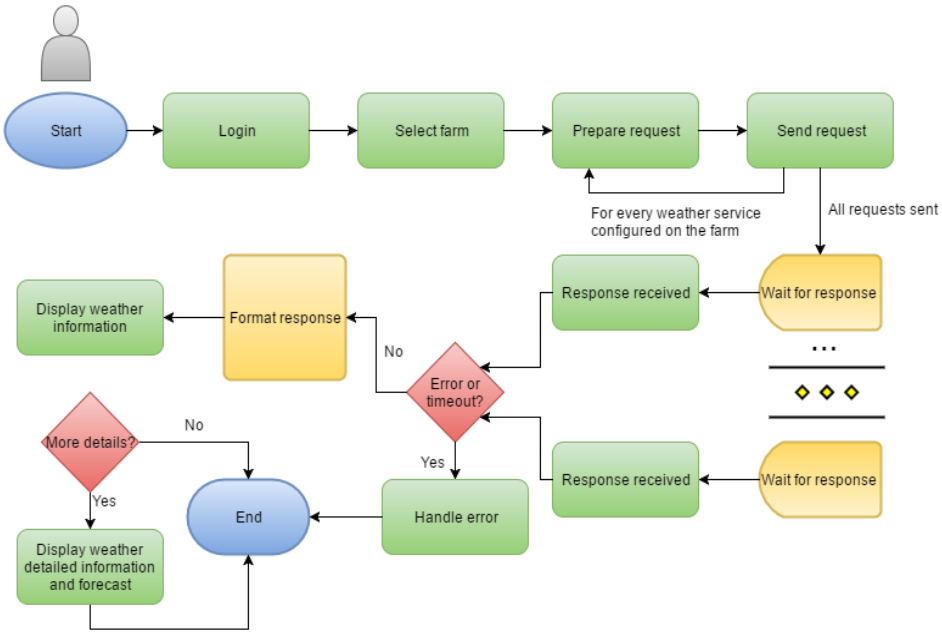

Figure 4.7 Weather Service Workflow.

The input is represented by the geospatial coordinates of a point (the middle of the polygon representing the farm). Regardless of the weather provider, longitude and latitude are sent via request parameters or are directly included in request URL. Every service requires a set of options to be specified but this is done invisibly to the user. The output is represented by a JSON which contains requested information or an error message in case of wrong formatted request. Example piece of result from OpenWeatherMap:

```
{...,"main": {
"temp": 8.46,
"pressure": 1024,
"humidity": 61,
"temp_min": 7,
"temp_max": 10
},...}
```

The service integrates with two weather services. Each is a restful service which exposes a public API.

OpenWeatherMap offers access to current weather for over 200000 places aggregating data from over 50000 weather stations. 5-day forecast includes data every 3 hours. For extended forecast, it includes data daily. It can provide hourly data about weather history for a period equal to the maximum of the previous month. Location can be specified by city name, by city ID or by geographical coordinates (used by the platform).

Forecast.io can be used to get current conditions, minute-by-minute forecasts up to 1 hour, hour-by-hour forecasts up to 48 hours and day-by-day forecasts up to 7 days. It can provide historical data up to 60 previous years or future data up to 10 years. Location can be specified by longitude and latitude.

Critical cases can occur when the external service is not available or when the request is timed out. When this happens, the weather widget contains an error message.

Another error may occur when the request is not well formatted, but this is to be avoided programmatically. For situations in which the API is changed the error is treated as described above.

Both services have limitations in terms of number of calls:

- ***OpenWeatherMap*** – no more than 60 calls per minute;
- ***Forecast***.io – no more than 1000 calls per day;

If the limits are exceeded the services are no longer available.

To reduce processing at the front-end application level the data from the weather providers are retrieved through the back-end application. Listing 4.1 presents the method which retrieve and parses the data from an external service. Knowing the weather service and the farm (needed to compute the point for which the data are requested), on line 4 a generic method that returns a well formatted URI is called. To do that the method should receive three parameters: the farm, the weather service and the type of information that should be retrieved (BASIC or ADVANCED), the last one being important for reducing the quantity of data requested from the weather server.

For example, for ***OpenWeatherMap*** service, a correct URI should look like this:

`http://api.openweathermap.org/data/2.5/find?lat=44.88&lon=25.69&units=metric&cnt=1&appid=f868f58d6d10ef036962e7aa672b946d, where "lat" and "lon" parameters are the latitude respectively the longitude of the middle of the farm and the "appid" is an application identifier previously requested from the API. It is a required parameter, always the same for each weather service individually.`

```
1       public BasicWeatherInfoDTO getBasicWeather(Long farmId, Long
weatherServiceId) {
2           WeatherService weatherService =
weatherServiceRepository.findOne(weatherServiceId);
3           Farm farm = farmRepository.findOne(farmId);
4           URI uri = WeatherUtil.buildURL(weatherService, farm,
WeatherInformationType.BASIC);
5           RestTemplate restTemplate = new RestTemplate();
6           String response = restTemplate.getForObject(uri, String.class);
7           JsonParser jsonParser = new JacksonJsonParser();
8           Map<String, Object> json = jsonParser.parseMap(response);
9           BasicWeatherInfoDTO dto = new BasicWeatherInfoDTO(json,
WeatherServiceEnum.WeatherServiceFromName(weatherService.getName()));
10          return dto;
11      }
```

Listing 4.1 Retrieve Basic Weather Information from External Weather Service.

The URI is used for the call to external ***REST*** service (using Spring's RestTemplate which is a class that helps consuming RESTful Web Service) the response of which is a string. To get useful data, the response must be parsed as ***JSON***; this is done on line 8, using Jackson JSON library.

To standardize the information presented to the user, regardless of the used service and the structure of the parsed JSON the retrieved data are processed using the algorithm presented in Appendix 1. The processing is done into the constructor of the DTO class (***BasicWeatherInfoDTO*** for basic weather info or ***AdvancedWeatherInfoDTO*** for advanced weather info).

The algorithm to compute the middle of a farm has as input the location of the farm (an array of points, each with a latitude and a longitude property). The latitude of the middle point is determined as the arithmetic mean of the lowest and the highest latitude of the input points while his longitude is the arithmetic mean of the lowest and the highest longitude of the input points.

4.5 Statistics Service

Statistics service is a service that allows the farmers to view the data generated by the sensors located at the farm site in a more user friendly manner, in the form of charts that aggregate data from a longer period.

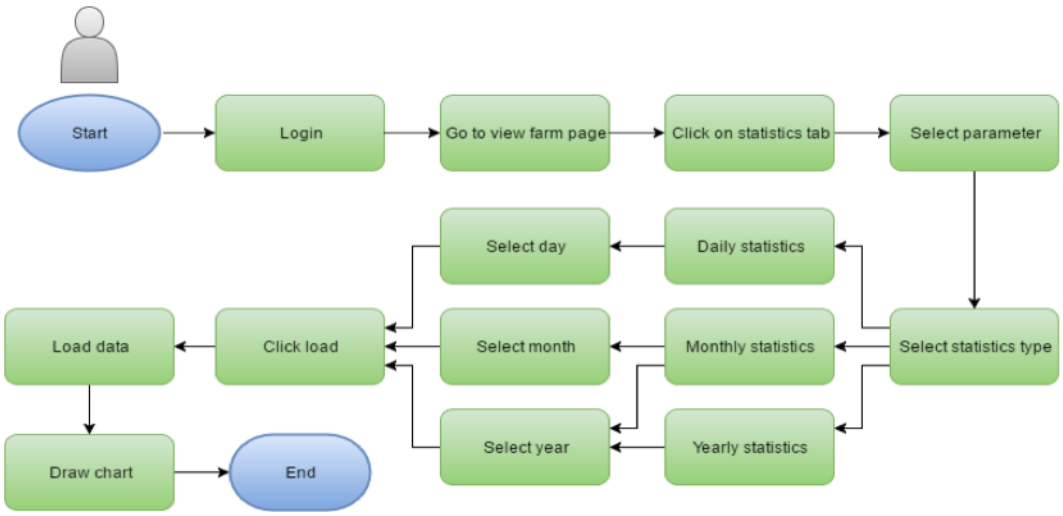

Figure 4.8 Statistics Service Workflow.

The service can be accessed from the farm view page by clicking on the statistics tab as shown in the flow diagram in Figure 4.8. The statistics are particular to each farm and for this service to be available the user must provide the data necessary to connect to the farm database while adding a farm (IP, port, username and password). Having these data, the back-end application can extract and process raw data generated by the sensors.

The list with available parameters is automatically generated based on the data found in the database of the farm. A complete list with parameters is available in Appendix 2.

The user has access to three types of statistics:

- Daily statistics – in order to view this type of statistics the user must select a parameter and a day and then click the load button. The day is selected from a calendar widget. A graph is drawn with a value for every hour of the day. The statistics are available even for the current day but only for the past hours.
- Monthly statistics – in addition to the parameter, to view this type of statistics the user must select a year and a month. The graph is drawn with a value for every day of the month. This type of statistics is also available before the end of the month.
- Yearly statistics – as with the other types of statistics the user must provide additional information, in this case the year for which he wants to see statistics. This type of statistics is available anytime, the graph being generated with the data from available months.

In Figure 4.9 it is an example of a graphical visualization of a daily statistic for temperature parameter.

The input for this service is represented by the options selected by the user on the page. They are sent to the server as parameters to the HTTP GET method. Relying on them the server-side application extracts necessary data from the database and responds with a JSON containing an array of values used for the graph.

A critical case occurs when the necessary data for the statistics are not available, for example when the user requires to view statistics for a period before the date of the registration of the farm into the platform. In such situations, a message is displayed informing the user about what was happened. While filling in the form for request statistics the user must select a value for every field. If any of the fields is not filled in the load button is disabled.

Because the response time of the platform is a critical attribute for performance measurement we propose for the statistics service an approach based on asynchronous evaluation of the data received from the farm and preparation of the statistics in advance. Therefore, the user does not have to wait for the statistics to be generated because these are already generated. This approach has a few advantages especially for user experience but also some disadvantages because this solution cannot scale with increasing number of farms and also the space for storing such data is greater.

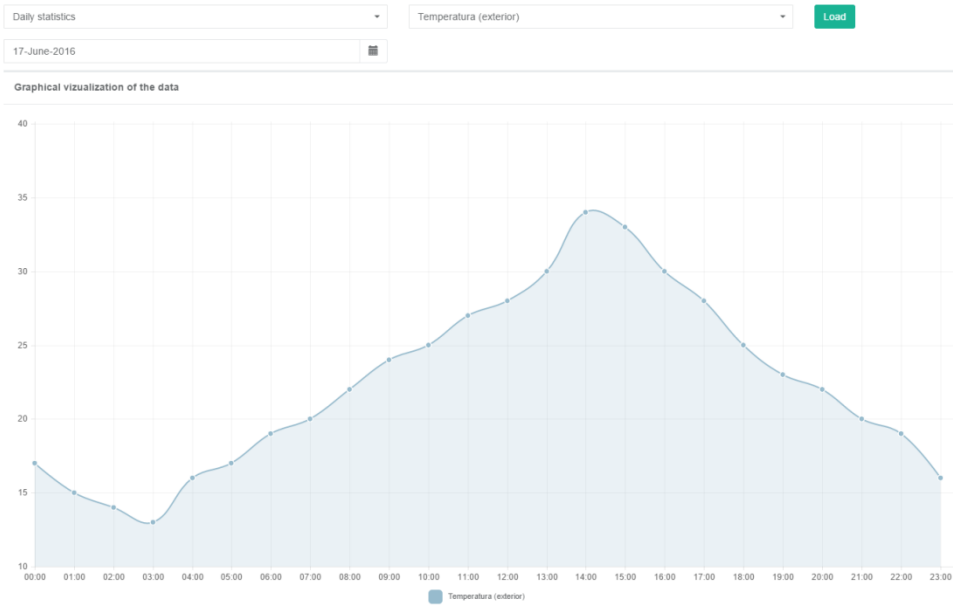

Figure 4.9 Statistics example.

This functionality is implemented using three scheduled tasks (as shown in Figure 4.10) each with a different role:

- The first one is responsible for generating useful data for daily statistics. It runs once an hour, processes raw data collected from the farms and inserts into the "**hour_statistic"** table one row for each farm representing the mean value of the data generated in the last hour. The database table with row data is populated by another job which runs once a minute, reads the data from the remote database, process it and writes the results into the database of the platform
- The second one runs once a day and has the role to generate data for monthly statistics. The input for this task is represented by the values generated by the first task. This task also produces one row for every farm every time it runs.
- The third one is like the second task except that this one runs once a month and produce data used for yearly statistics.

Next we detail the implementation of a scheduled task. It is implemented in accordance with the Replicated Workers model. This model is based on a few workers who can execute any job in work pool. A work pool is a collection of tasks waiting to be executed. Any worker executes a job and once he finishes he take another job from the work pool until the task is finished. A task is considered done when all the workers finished their job and the work pool is empty.

Listing 4.2 presents the code for a scheduled task. This is a thread that creates a work pool of jobs each time it runs (line 2). The type of job inserted into the pool as well as the running frequency is determined by the type of task. The work pool is populated with a job for every farm (as presented in line 4 to line 6). Once the work pool generation is completed it creates several NT worker threads and it starts them (line 7 to line 13). We choose NT to be 4 but if the application does not scale with increasing number of farms it can be increased easily. The workers take jobs from work pool in a **round robin** manner and execute them. When all the workers finish their jobs and the thread pool is empty, the scheduled task ends too.

```
1        public void run() {
2            WorkPool workPool = new WorkPool(NT);
3            List<Farm> farms =
farmRepository.findAll();
4            for (Farm farm: farms) {
5                workPool.putWork(new
CreateStatisticsJob(farm, statisticsType, …));
6            }
7            Worker[] workers = new Worker[NT];
8            for (int i = 0; i < NT; i++){
9               workers[i] = new Worker(workPool);
10           }
11           for (int i = 0; i < NT; i++){
12               workers[i].start();
13           }
14           for (int i = 0;i < NT; i++){
15               try {
16                   workers[i].join();
17               } catch (Exception e){
18                  e.printStackTrace();
19           }
20         }
21       }
```

Listing 4.2 Scheduled Task Implementation

4.6 Notifications service

Notifications service is a service through which the platform can notify users about the occurrence of various events generated by the other services of the platform as specified in Figure 4.11.

The input is represented by an object that contains the following fields:

- User – the user who should receive the notification;
- Title – the title of the notification or the email subject
- Message – the content of the notification or the message that is sent in the email body;
- Type – the type of notification;
- Severity – the severity of the notification. Can take one of the following values: success, info, warning, error;
- Url – the redirecting location;
- isRead – a flag to check if the notification is seen by the user;

The output is represented by the notification sent to the user if there is no error.

The service integrates with more services. They can be divided in two categories: those which generate requests for sending notifications and those which handle these requests. The first category includes: the alert service (generates critical alerts), the user management service (require sending emails when a new account is created), the store service, the forum service (sends internal notifications to users when someone posts in a discussion thread in which they subscribe) and the blog service (the user is notified when someone comments on his blog post). They send requests formatted as described above. To the second category belongs the email server, used by the service to send emails and an AngularJs service which handles the platform notifications.

Regarding the input only user and severity fields are required while the type, url and isRead fields are taking default valued if not specified. The user must be an active user on the platform. The message and title fields are strings. If these validations are not satisfied an error is thrown. Another critical situation is when the service able to send emails is down. In this situation administrator intervention is required.

The emails are sent using Spring Framework's implementation of JavaMail, the content of the email being represented by HTML templates written in Thymeleaf.

The mechanism for sending notifications is implemented as a generic service that takes as parameter an object of the type notification and according to the values of the fields it takes different actions. This service is in charge of

validation and generation of unspecified fields. If severity, which is a required field, has the value of error then in addition to a platform notification, an email is sent too.

To simplify using this service from other services, we choose to create a notification object using the builder pattern. Using this creational pattern, the need of writing more constructors is avoided and the user of the service can simply provide only those parameters he considers important, the rest of the business logic being relegated to the service.

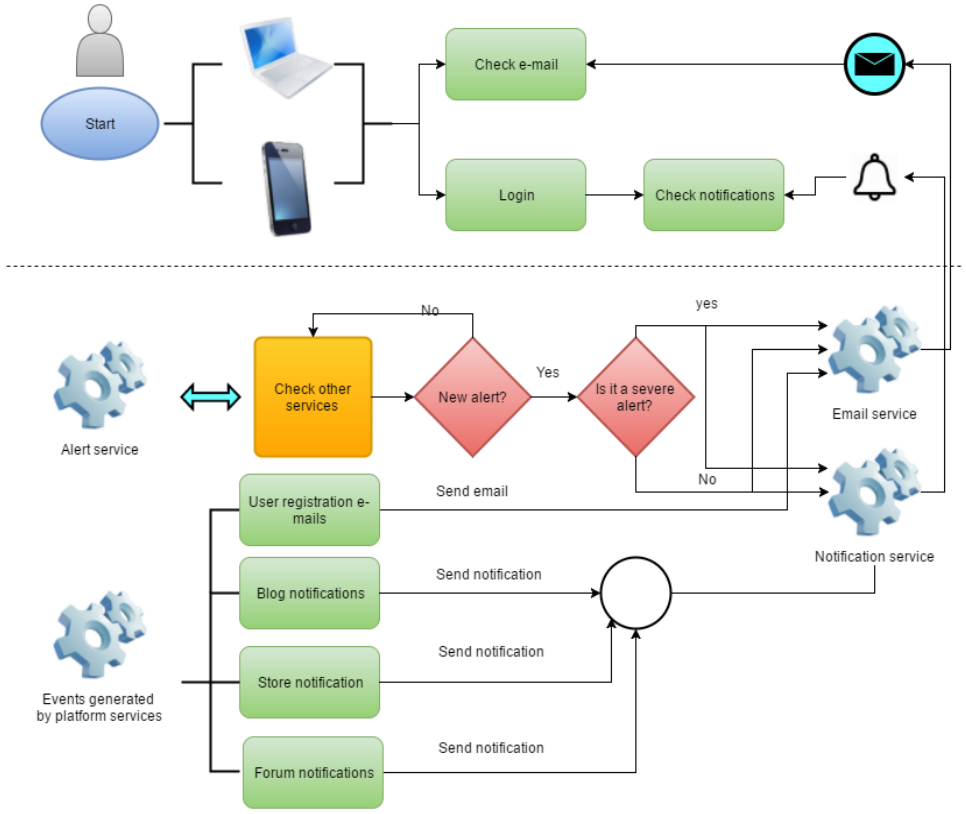

Figure 4.11 Notification Service workflow.

To notify the user through the web interface of the platform we propose a solution based on the WebSocket and STOMP protocol. When a user logs into the platform a WebSocket connection is established between the web browser of the user and the web server through the WebSocket endpoint exposed by the back-end application.

Over the established WebSocket connection, the STOMP client subscribes to a STOMP message broker through a channel. The message broker is available at an address built based on the username of the user (Figure 4.12). Thus, it is guaranteed that the channel created is unique therefore, only the right user receives the notification.

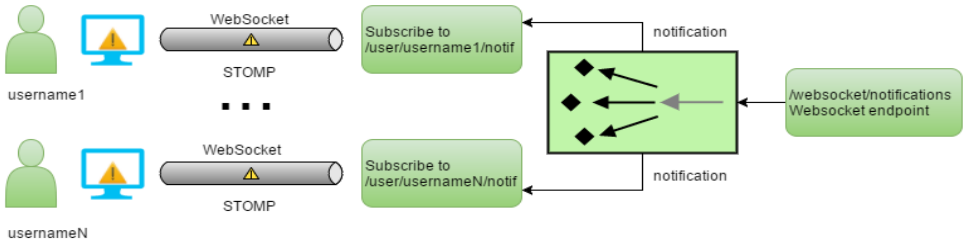

Figure 4.12 Notification Service Functioning.

Over the established WebSocket connection, the STOMP client subscribes to a STOMP message broker through a channel. The message broker is available at an address built based on the username of the user (Figure 4.12). Thus, it is guaranteed that the channel created is unique therefore, only the right user receives the notification.

Once the STOMP connection is established the user receives all unseen notification beginning with the last login and during the usage of the platform all the new notifications are sent in real time. There is a service in front-end application which receives all the notifications and generates different types of toast notifications based on severity field.

The biggest advantage of using these technologies is that it offers the possibility of implementing a mechanism of real time push notification over a bidirectional channel with a few overhead for the web browser because it does not have to request new notifications.

4.7 Blog service

Blog service is part of the social networking aspect of the platform. It allows any user to write complex articles about subjects that may be of interest to other users and to read the articles written by others. Its workflow diagram is presented in Figure 4.13.

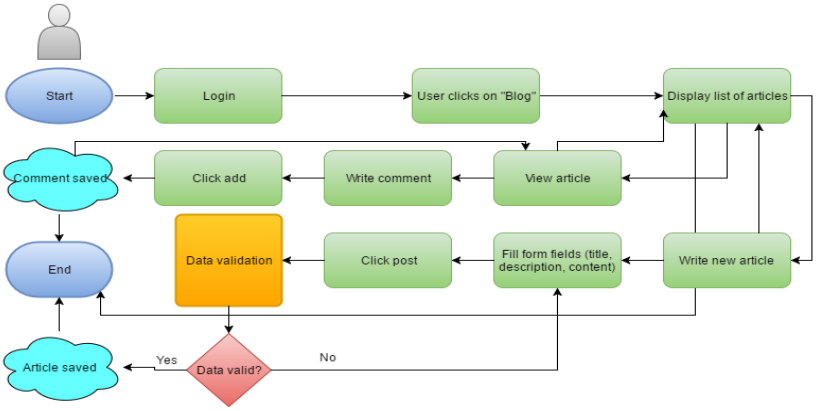

Figure 4.13 Blog Service Workflow.

The blog can be accessed from the main menu, the first page consisting of a list with all the articles (title and a short description) in descending order by the date of their creation. From this page the user can choose to read or write an article. Clicking on an article opens a new page where the user can read the content of the article or write comments. There are some statistics available including the number of comments and the number of views. Clicking on the "Write new article" button opens a page where the user must write a title, a short description and the content of the article. By submitting the form, the article is saved.

The content of the article is written with the help of a widget which allows the text to be customized. Customization options are the following:

- bold, italic, underline, superscript, subscript, strikethrough text;
- font family – more options, including "Arial", "Helvetica", "Tahoma", "Verdana", "Times New Roman";
- font size – the range of 8-36;
- line height – the range of 1.0 to 3.0;
- paragraph style;
- background and foreground color;

The elements that can be inserted are: lists (unordered, ordered), tables, links and pictures. It also offers functions such as undo and redo.

When a new article is written, the input expected from the user is represented by three fields: title, description and content. The first two are plain texts, while the last one is a string the content of which represents the HTML code generated by the widget described above. Those fields are passed to the server as a JSON where the article is preserved in the database. The output is represented by a success or failure message coming from the server also as a JSON.

Regarding the main page of the blog the output received from the server is a JSON with a list of objects having several fields: title, description, createdDate – the date on which the article was created, numberOfViews – the total

number of views, numberOfComments – the total number of comments, user – the name of the user who wrote the article. For the visualization page of the article the output is an object having the same fields as the ones named above while also incorporating the content field – a string with HTML code.

When posting an article, all fields are required. Validations are done both on the client side (the submitting button of the form is disabled if one or more fields are not completed) and on the server side (if one of the fields is missing the server responds with an error message specifying what the error is). When posting a comment, the situation is the same. The content of the comment should not be null. When there are no posts available an informative message is displayed.

Depending on the database settings, when posting an article containing uploaded images an error may occur when the total size of the article is bigger than the maximum size allowed by the database. In such situations, the user is announced through an error message.

In order to offer the text editing features for writing the content of the article we used "Summernote". This is a popular JavaScript library that offers a simple and easy configurable text editor compatible with AngularJs framework. The library formats the content as HTML code and this is stored in the database as a string in a LONGTEXT field. MySQL engine limits the size of such a field to a maximum of 5Gb but more often the maximum size for a transaction is much smaller. We set this limit to 10Mb as we considered that it is enough for any blog post. If this size is exceeded an error is thrown.

5. Performance analysis

In this section, we present a performance analysis of the platform by measuring the load time from different places around the world specifying critical points, along with the response time of the platform services.

5.1 The Performance of the Platform

Loading time is a very important aspect for measuring the performance of a web application. Together with the design, this influences a lot the user experience on the application. Several research results made by Google specialists show that a user leaves the application if the loading time is over 2 - 2.5 seconds. Besides advantages in terms of UX (User Experience), there are also advantages in terms of SEO (Search Engine Optimization), the loading time being recently added among many criteria used by Google (the most popular search engine) for computing the Page Rank.

The specifications of the server and the distance between the testing location and the server highly influences the performance of the application. For the next testing scenarios, the platform is deployed on a Tomcat 8 server located in Europe. The tests are made with the help of WebPageTest [http://www.webpagetest.org]. This is an online tool that allows complex load testing of web applications from different locations around the world providing the results in an easy to interpret manner.

Being a Single Page Application designed as presented in Chapter 2, all the static resources (HTML files, JavaScript files and CSS files) are loaded on first access of the platform, thus the worst-case scenario is the testing of the home page. Table 5.1 presents the test results from different locations using a Google Chrome browser over a cable connection (5/1 Mbps 28ms RTT).

Table 5.1. Load Time Test Results.

Location	Load Time	User Time	First Byte	Start Render
London, UK	**2.533s**	**1.878s**	0.246s	1.390s
New York, USA	**2.074s**	**1.717s**	0.338s	-
Sao Paulo, Brazil	**5.025s**	**3.728s**	0.945s	1.986s
Johannesburg, South Africa	**5.073s**	**4.800s**	1.614s	2.540s
Tokyo, Japan	**5.431s**	**3.831s**	1.046s	1.481s
Sydney, Australia	**2.812s**	**1.864s**	1.359s	1.496s

The load time column represents the total time spent by the browser to load and render the page, while the user time represents the time after which the user can use the application. From the user perspective, this time might seem shorter because the CSS files are loaded before the JavaScript files and the application seems to be ready to the user before it is. The values obtained are lower than the limit of 2 to 2.5 seconds for locations in North America or in

Europe. The load time of the other pages of the platform is considerably lower because the only static files loaded are: the HTML template, a JSON file containing the translate labels and the images used in that page. For very distant locations as South Africa or Japan the loading time for the first page are not very satisfactory, but once the application is loaded for the first time the other pages can be accessed in a time lower than 2.5 seconds given that the first Byte is received after 1 – 1.5 seconds, thus the application is still offering good performances.

In Figure 5.1 a distribution of the number of HTTP requests on file types is presented. Most files fetched from the server are JavaScript files (81.2%) followed by the number of files of CSS type (10.1%), while the HTML files represents just 5.8%.

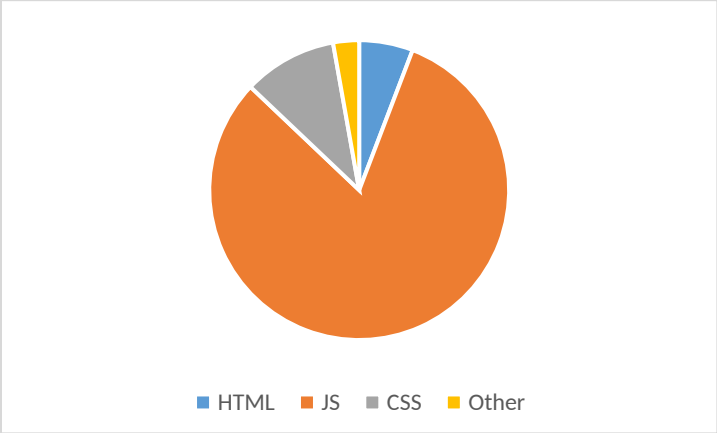

Figure 5.1 Requests Diagram.

Regarding the dimension of the fetched files (presented in Figure 5.2), the order is maintained, the JavaScript files representing 86.5% of the total size, CSS files 12.4% while HTML pages represents just 1%.

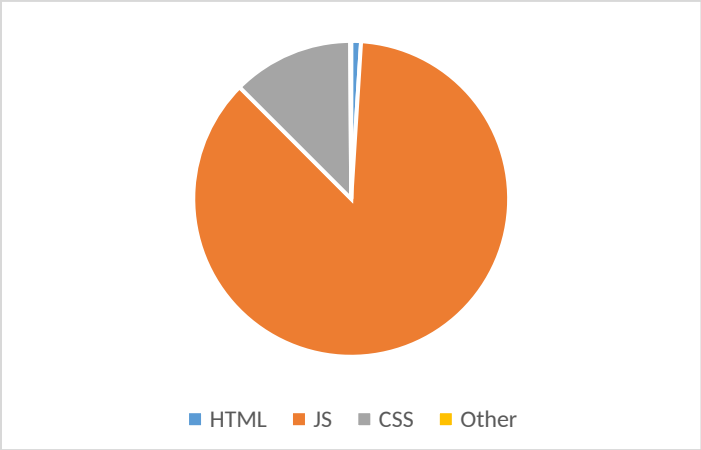

Figure 5.2 Bytes Diagram.

5.2 The Performance of the Platform Services

The response time of the services of the platform is also very important, especially for those which are dealing with critical tasks or data processing like Statistics Service or Notification Service.

For the tests described in this section we used a computer that has an Intel® CoreTM i7 – 3610QM CPU @ 2.30GHz, 8GB DDR3 RAM and a Windows 10 Pro operating system. The application is deployed on tomcat-embed-core 8.0.30. The application is accessed using Google Chrome web browser. The total time of a request is represented by the sum of the connection setup time and the Request/Response time represented by the time spent on the network plus the waiting time. In Table 5.2 only the waiting time is represented, it is the time spent by the server in processing a request. For higher accuracy, a set of 4 tests were made.

Table 5.2. Services Response Time Test Results.

Action	Test1	Test2	Test3	Test4	Average
User registration	300.85ms	264.61ms	161.47ms	265.05ms	**247.99ms**

Farm registration	185.68ms	92.64ms	67.72ms	94.56ms	**110.15ms**
OpenWeatherMap - get basic weather information	123.70ms	128.77ms	127.05ms	124.40ms	**125.98ms**
OpenWeatherMap – get advanced weather information	215.81ms	231.25ms	233.66ms	234.26ms	**228.75ms**
Forecast.io - get basic weather information	449.39ms	529.63ms	451.81ms	529.69ms	**490.13ms**
Forecast.io – get advanced weather information	453.70ms	463.31ms	480.97ms	508.86ms	**476.71ms**
Get daily statistics	13.75ms	12.94ms	12.72ms	12.92ms	**13.08ms**
Get monthly statistics	12.75ms	13.92ms	11.76ms	12.39ms	**12.71ms**
Get yearly statistics	11.17ms	9.80ms	9.75ms	9.43ms	**10.04ms**
Post blog article (1Mb content)	214.26ms	206.72ms	200.12ms	179.17ms	**200.06ms**
Receive notification	28ms	29ms	21ms	22ms	**25ms**

For user registration, the measured time includes the time spent for sending the confirmation email. For weather service the response time can vary depending on the response time of the weather provider. Considering the implementation of the statistics service presented we have succeeded in obtaining very little response times, the greatest amount of data processing being made asynchronously, the user does not have to wait for the statistics to be computed when he requires them.

For notification service the measured time represents the time interval between the occurrence of an event and the moment when the user is notified in platform. The results are obtained due to the chosen implementation (using WebSocket protocol) and they can be influenced by the physical distance between the server and the client (the time spent on the network).

6. Conclusions and future work

In this paper, we have presented a web platform the main purpose of which is to improve agriculture by providing the farmers with a way to monitor their farm from a distance with a single requirement: a connection to the internet. Another goal was to make a platform which facilitates the creation of a strong online community of farmers and agriculture experts (a social network for farmers). All the information presented in this paper shows that these objectives were accomplished.

Before describing the architecture of the platform, we presented the architecture of the system which contains the platform. The platform is constituted of two main applications, decoupled from one another. We propose an architecture along with the technologies used and a motivation for choosing them, for each of these two applications and we also discuss the communication between them.

Because the main components of a platform are the services that it offers to the users, we rendered a detailed presentation of each one of them. A functional description and some important implementation details was presented. The user management service allows a user to register in the platform. Several registration manners have been implemented. The users along with their farms are the main entities of the platform, latter being managed by other services, the farm management service which allows a user to register a farm in a platform and specify its location on a map. The weather service is another service available on the platform. It offers precise 7-days forecast for each farm. The statistics service is very important; it is the one which connects the user with his farm offering him real time statistics about monitored parameters. We implemented it in a manner that guarantees small delays. When an event that is of interest for the user occurs, he is notified through the notification service (platform notification and/or email). One of the services that categorizes the platform as a social network is the blog service. It can be used by the users to share their thoughts by posting articles on a blog.

We analyzed the performance of the platform through several tests which reveal the load times of the platform and the response times of the services. We also explained the results.

In the future, we want to extend the platform functionality by adding a couple of new functionalities or improvements including:

- adding to the notifications service the capability of sending SMSs. This is a very important aspect because important events could take place when the user does not use the platform, that is, when he is not connected to the internet. This may be the only way to send alerts to the user;
- adding a scheduler that allows the user to plan his activities better. He will be notified when a planned event is approaching;
- increasing the number of possibilities to register into the platform by adding more external identity providers like LinkedIn or Twitter. This simplifies the process of registration for users who do not have a Facebook or Google account but who still want to use the social registration feature;
- increase the number of available statistics for the statistics service;
- allowing a farmer to add contributors to a farm he owns;
- finally, improving the performance of the actual services and reducing the storage used by the platform and its loading time by partially loading JavaScript files and load libraries from CDN (Content Delivery Network) sources.

7. Acknowledgment

The research presented in this paper is supported by projects: *clue-Farm* - Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms - PN-II-PT-PCCA-2013-4-0870 and *DataWay* - Real-time Data Processing Platform for Smart Cities: Making sense of Big Data - PN-II-RU-TE-2014-4-2731.

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

8. REFERENCES

AgFuse, 2016< https://agfuse.com/home/about>.

Alexandratos, N. and Bruinsma, J., 2012. World agriculture towards 2030/2050: the 2012 revision (No. 12-03, p. 4). Rome, FAO: ESA Working paper.

Ashton, K., 2009. That 'internet of things' thing. RFiD Journal, 22(7), pp.97-114.

Bojan, V.C., Raducu, I.G., Pop, F., Mocanu, M. and Cristea, V., 2015, September. Cloud-based service for time series analysis and visualisation in Farm Management System. In Intelligent Computer Communication and Processing (ICCP), 2015 IEEE International Conference on (pp. 425-432). IEEE.

Davis, G., Casady, W.W. and Massey, R.E., 1998. Precision agriculture: An introduction. Extension publications (MU).

Fette, I., 2011. The websocket protocol.

Fountas, S., Carli, G., Sørensen, C.G., Tsiropoulos, Z., Cavalaris, C., Vatsanidou, A., Liakos, B., Canavari, M., Wiebensohn, J. and Tisserye, B., 2015. Farm management information systems: Current situation and future perspectives. Computers and Electronics in Agriculture, 115, pp.40-50.

Green, B. and Seshadri, S., 2013. AngularJS. " O'Reilly Media, Inc.".

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., Arendsen, A., Davison, D., Kopylenko, D., Pollack, M. and Templier, T., 2004. The spring framework–reference documentation. Interface, 21.

JSON, 2016< http://www.json.org/>.

Kruize, J.W., Wolfert, J., Scholten, H., Verdouw, C.N., Kassahun, A. and Beulens, A.J., 2016. A reference architecture for Farm Software Ecosystems. Computers and Electronics in Agriculture, 125, pp.12-28.

Liao, M.S., Chen, S.F., Chou, C.Y., Chen, H.Y., Yeh, S.H., Chang, Y.C. and Jiang, J.A., 2017. On precisely relating the growth of Phalaenopsis leaves to greenhouse environmental factors by using an IoT-based monitoring system. Computers and Electronics in Agriculture, 136, pp.125-139.

Mocanu, M., Cristea, V., Negru, C., Pop, F., Ciobanu, V. and Dobre, C., 2015, May. Cloud-based architecture for farm management. In Control Systems and Computer Science (CSCS), 2015 20th International Conference on (pp. 814-819). IEEE.

887 OnFarm, 2016 <http://www.onfarm.com/>.

888 RESTful Web Services, 2016< http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>

889 Serrouch, A., Mocanu, M. and Pop, F., 2015, June. Soil management services in Cluefarm. In Parallel and
890 Distributed Computing (ISPDC), 2015 14th International Symposium on (pp. 204-209). IEEE.

891 So-In, C., Poolsanguan, S. and Rujirakul, K., 2014. A hybrid mobile environmental and population density
892 management system for smart poultry farms. Computers and Electronics in Agriculture, 109, pp.287-301.

893 STOMP, 2016 <https://stomp.github.io/>

894 WEBPAGE TEST, 2016<http://www.webpagetest.org/>.

Dear Editor,

Here the highlight for the paper *CLUeFARM: Integrated Web-Service Platform for Smart Farms*:

1. This paper describes a web platform that comes in handy to fulfill the needs of the farmers, giving them the possibility to manage monitor and control their farms from distance through any device that has an internet connection (a phone, a table or a personal computer).
2. Before describing the architecture of the platform, we presented the architecture of the system which contains the platform.
3. The platform is constituted of two main applications, decoupled from one another. We propose an architecture along with the technologies used and a motivation for choosing them, for each of these two applications and we also discuss the communication between them.
4. We analyzed the performance of the platform through several tests which reveal the load times of the platform and the response times of the services. We also explained the results.

Corresponding Author,

Prof. Florin Pop