Advanced Services for Efficient Management of Smart Farms

George-Alexandru Musat[a], Mădălin Colezea[a], Florin Pop[a,b], Catalin Negru*[a], Mariana Mocanu[a], Christian Esposito[c], Aniello Castiglione[c]

[a] *Computer Science Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania*
[b] *National Institute for Research and Development in Informatics (ICI), Bucharest, Romania*
[c] *Computer Science Department, University of Salerno, Italy*

Abstract

Lately, the technology has been developed quickly causing the appearance of smart software products in many domains such as agriculture which are designed to ease people's work. Due to bad weather phenomena and growing demand for agricultural products, have started to appear a lot of greenhouses, where there is a strict control around each parameter, by sensors, contributing to the increase of the production. This paper presents a smart platform which helps farmers to efficient manage their greenhouses and to interact with other farmers. We present the software architecture and each implemented service. We describe the general implementation details and then we present the data gathering algorithm. This algorithm is one of the most important features of the platform because it deals with sending notifications to users in case of a problem in greenhouses. It also offers data for processing to other services like the statistics service. Performance tests show the necessary time for gathering a large amount of data and for their processing. In addition, we will analyze the time needed to send notifications to users.

Keywords: Distributed Systems, Internet of Things, Internet of Data, Smart Farming.

1. Introduction

Nowadays, technology has started to make its presence felt in more and more domains, helping them through smart software solutions [1] that are designed to ease the people's work, to increase production in some cases, even to decrease the consumption of resources [2], [3], [4]. A concrete case is represented by the intelligent houses which have systems that automatically manage the resource

*Corresponding author, Tel.: +40-720 383-408; Fax: +40-318-145-309; *Email address*: catalin.negru@cs.pub.ro.

consumption like the light, the heat, the air conditioning, the purpose being to decrease the resource consumption and the environmental impact [5].

In addition, the concept of Internet of Things (IoT) picks up more sense being influenced by the multitude of intelligent devices that communicate with each other through the Internet [6], [7], [8], [9]. It is also used in the cultural heritage protection oriented networks for the environment monitoring and security enhancement issues [10], [11]. Agriculture appeared until beginning of the world, the main purpose of this domain being the feeding of the population [12]. The intensifying of weather phenomena, and the growing demand for agricultural products has led to broadening the idea of a greenhouse. The greenhouse is a complex system where the plants are being grown into a monitored medium by moisture sensors, temperature sensors and light intensity sensors which can determine the automatic realization of certain actions to increase production [13], [14].

All these sensors and smart appliances can provide a large amount of data and the big challenge is to analyze this data to provide useful information about the current state of the farm [15]. Moreover, this data can be used to create statistics that may be useful to increase production in future [16]. So, this led us to the concept of Internet of Data which is a network composed by data entities coming from the IoT. It may also refers to exploiting location-based services and technologies to provide smart data services and applications [17], [18]. The focus is to gain insight [19] about the efficiency of management operations [20], about the supply chain and the interactions with other players in the field [21].

In this paper we propose an integrated platform for efficient management of smart farms which aims to gather this large amount of data to notify farmers if a problem occurs and to provide useful information like statistics, tendencies and correlations. In addition, the platform has a social networking area where farmers can interact.

The main contributions of this paper are the following: smart platform which helps farmers to efficient manage their greenhouses and to interact with other farmers; a software architecture and description of each implemented service and the general implementation; a data gathering algorithm and performance test of the proposed platform. The proposed platform is based on data analytics, Cloud computing, management of data models and services.

The integrated platform should offer a wide range of services that help farmers to improve production in greenhouses and also to share knowledge with other farmers. To fulfill these two conditions in a single platform is really challenging because farmers must not be confused about how the application works. Besides complex functions that must provide for processing data received from sensors, the application should be intuitive and easy to use by any person. The challenging features of this platform are:

- Farm registration is an intuitive and simple task.It is the main functionality to help farmer to efficient manage their greenhouses. Afterwards, gathering data from sensors must be done in a transparent manner to the user, the important thing for him being the result of the data processing;

- Implementation of the notification system is very challenging because users must be warned in no time on the occurrence of a problem through suggestive messages or by email;
- Statistics, tendencies and correlations have to offer consistent data with witch farmers can take decisions to increase production in future;
- Social media services should be easy to use and provide high flexibility in communication and efficient exchange of impressions and knowledge.

The objective of this paper is to develop a smart platform, which can be able to help all the farmers by giving them the possibility of managing their farms in an efficient way. In the same time, this platform has a social networking area, where all the users can communicate and can change opinions about the harvests. This part is as important as the first one, because we know that the experienced advices of these farmers can help the other ones and can grow up their production. The objectives can be divided like this:

- **Internal notification service** - proposes to help all the users to send messages in a faster and easier way through the platform. When a user receives a new message, he must be informed no matter where he is placed in the application, because in this way he is able to see and to answer as fast as he can. In the same time, the internal notification service proposes to inform the user if there is a problem within the farms so he can be able to resolve it immediately. In addition, this service sends informative alerts, which are generated by other platform's services;
- **Groups service** - create groups which are formed by users with mutual interests. A group can contain, for example, all the farmers from a certain area, or the farmers which are interested in a certain type of a harvest like tomatoes, or cucumbers. In order to add value to platform, this service must offer some facilities within the other services, like the forum, or the internal notification service;
- **Store service** - all the users are able to post rent, sale or purchase advertisements within all types of useful tools, or products which are being used in their greenhouses. All these must be seen by all the users and if someone is interested, he will be able to contact the seller by using the internal messages, or the internet;
- **Forum service** - provide a section where users can discuss on different topics of mutual interest. Users will be able to add new posts or subcategories and to comment on existing posts, the main purpose being to help people to interact on topics related to production, how to improve the management of their farms, best practices for increasing production and others;
- **Tendencies and correlations service** - users will be able to access important data, which will be easy to follow and easy to understand.

This service exposes the current situation of the greenhouses and it will be made a parallel between the previous and the current situation. So, the user must see the growth and the decrease factor of all the parameters within the last hours, days and months. The correlations must offer a clear and a new perspective within the dependency between two parameters.

This paper is organized as follows. In the Section 2 we present some solutions which can offer us support for the management of the farms or which facilitates the way of communication between all the farmers. Section 3 presents the software architecture of the intelligent platform and the used technologies. In Section 4 we describe the platform in a general way and then we describe each implemented service, too. In addition, we present the structure of the database and we detail the tables that we used for the services. Section 5 presents some general implemented details, which are being used within all the services and in some particularities cases too. In addition, we describe the gathering mechanism of the local farms data and the way in which is processed and saved in the application database. In the Section 6 we analyze the performances of some services. Firstly, we notice the length of the time which is necessary for the processing of data received and we analyze how this length of time varies depending on the number of measurements and sensors which are associated to the farm. We also analyze the time lengths which are necessary for the calculation of tendencies and we notice how this can affect the loading time of the farm's page. Finally, in Section 7, we present the conclusions and future work.

2. Background and related work

2.1. Related work

FarmLogs [22] is a platform which offers support for managing multiple farms through many services. Firstly, the application allows users to add fields associated with their farms by identifying them on the map. Automatically is being calculated the size of the field and are taken the geographical coordinates. These are used to get weather information from external weather services. In addition, the application automatically identifies the soil type of the field. After the field was added, the application offers to user the possibility to add many activities related to its fields like irrigating, tilling, planting, spraying, harvesting and fertilizing. The platform offers detailed info forecast for the next days. FarmLogs offers a support for administrating the budget of the farm and offers approximate parameters about the profit which can be realized by the harvest. In addition, users can add notes for each field, can log storage centers with their capacity and equipment used in farms. Paid services give to users details about the level of nitrogen, crop health monitoring, automatic activity record and guidance on increasing the efficiency of the field.

British farming forum [23] is a specialized forum platform which offers to users the possibility to discuss around themes from the domain of agriculture. The forum structure consists in a main category, which contains subcategories as: Agricultural matters, cropping, machinery and others. It also has an extra category which contains subcategories as: Buildings and Infrastructure, Computer Issues. At the same time, the application offers some external links to another forums which are being specialized in agriculture.

Farm Time [24] is an American platform which offer social networking services in the agricultural context. This application has a basic forum service which offers to users the possibility to discuss around agricultural themes. Another function is represented by a section of blog where are being published interesting articles for farmers. Farm Time offers a groups section, in which users can join depending on the area of interest. Farm Events gives to users the possibility to find out what events related to agriculture will occur in the future. Finally, in Farm Media zone, users can add photos or videos about their farms.

The authors of [25] present a farm connected through IoT systems with the goal to provide a farming system for end users. The paper presents the design and implementation for connected farms and the advantages of using such systems. Furthermore, service scenarios that compare smart interconnected farms to previous smart farms are presented.

In [26] the authors present a novel approach to solve the problem of pest infestation in crops through monitoring and video processing, using technologies such as cloud computing and robotics. Are presented methodologies to detect pests in the tomato and how Internet of Things paradigm can be conceptualized in solving the problem of cultures infestation.

2.2. System architecture

The platform for efficient management of smart farms is made of two completely separated applications: the frontend and the backend one. The connection is made through the Representational State Transfer (REST) services, which are exposed by the backend application. There are used web sockets for processing the data in real time. The web socket represent a protocol, which offers us a full-duplex communication over a TCP connection. If we use it in the frontend application, it won't be necessary to poll the server to see if there are new data.

The backend application communicates with external weather services to gather weather forecast. The connection is realized through REST services and all the data are received in JSON format. This application also communicates with an email server for sending emails to users. The connection is made up using the Simple Mail Transfer Protocol (SMTP). Beside MySQL local database, the backend application can access the farms' databases. This is happening periodically through jobs.

2.2.1. Backend architecture

The backend application is implemented in Java language by using the Spring framework. Spring framework represents a large platform. It is intended to simplify the writing of Java applications. It is mainly used for the Java Enterprise Edition (Java EE) platform, but it can be used in any other application based on Java language. The Spring MVC framework is based on the Model-View-Controller (MVC) model and it has started from Spring framework. The Model-View-Controller is an architectural model, which is highly used in the software engineering. Its purpose is to isolate the functionality components (controllers) from the model and the view ones.

For keeping the security we used Spring Security framework, which offers a special support for session-based authentication and authorization. We used Spring Social module for signing in using external authentication services. This is an extension which makes the connection with external services like Facebook or Google through the exposed APIs. For mapping the database's tables, we used Hibernate, which is a JPA implementation. We used Spring Data JPA for having a view within the database. This framework is bringing built-in methods for writing, reading, updating and deleting. It allows to create custom queries by using a resembling SQL language, named Java Persistence Query Language.

We used the Spring framework because it can offer a various range of modules, which can make easier the programmer's job (Spring Security, Spring Social and others). Another reason is that it offers a lot of reusable implemented components and modularity, too. We used MySQL for the database because is one of the most popular databases and it is very easy to use it. It can also offer a various documentation if a problem occurs. We chose Liquibase for adding default data and for changing the database. Its purpose is to manage and update the changing databases scripts. After a database script runs, an informational entry is saved into a Liquibase table, which contains a MD5 codification of the script. If the script is adjusted, it will have a different MD5 and it will generate an error. Gradle is a system which is used to build the application. It helps us to open the application, it can pack the application as jar or war and it can generate the application's documentation. For establishing the order in which every task run, Gradle uses directed acyclic graphs (DAG).

2.2.2. Frontend architecture

The frontend application is implemented with AngularJS framework. This is a JavaScript framework, which can offer us a lot of facilities for developing dynamic web sites. For the markup and style part we used the newest Hyper Text Markup Language (HTML5) and Cascading Style Sheets (CSS3) versions. In addition, we used the Bootstrap 3, which is an HTML and CSS framework. It can offer us support for creating responsive applications, with a good look on all devices.

As we can notice in Figure 1, the frontend application has a mechanism for routing which offers to users the possibility to access all sections. The "Two-way data binding" mechanism is used to synchronize data between the model and the view. That's why is not necessary to reload the page for data's updating. This approach is called "Single-page application". This model provides a fluid user experience and all necessary files like HTML, JS, CSS are loaded on the first access. We used the angular translate module for the internationalization. The module deals with labels replacing from the HTML view with a translated version from specific JSON files. To facilitate the application development we used the following tools:

- Bower - it facilitates the installation of JavaScript libraries and AngularJs modules;
- JsHint - it can find the JavaScript errors;
- LiveReload - it can allow the reloading of the page in real time, when a HTML, JS or CSS file have been modified;
- Grunt - it includes all of the above.

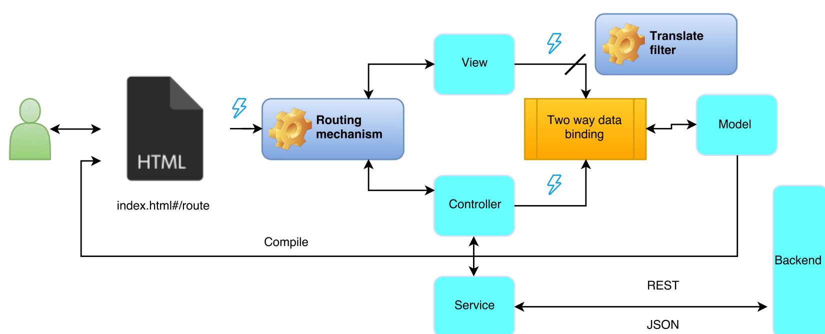

Figure 1: Front End Architecture.

3. Platform description and services specifications

The management platform of the smart farms gives us a large number of useful services which can help the farmers to notice how the parameters evolves in their farms. They can also receive notifications if there are some problems, they can communicate with other farmers within changing some useful information and they can sell or rent some agricultural products.

The guests of this platform must create an account or they can authenticate through an external service like Facebook or Google. The main page of the platform contains only items of information about contacts, features offered by the platform and a map with all the farms registered in the platform. The map can be annotated with informations based on satellite image processing [27, 28, 29]. After login, we have access to each service from the menu which can be found in the left part of the page. In this section, we are going to describe every implemented service from the platform and the structure of the database, too.

3.1. Internal notifications service

Internal notifications represent the service that allows direct interaction between users within the platform. The first feature provides a mailbox system through which users can send and receive messages inside the platform. It can be compared with an email client offering basic functionalities.

In addition, another feature of this service is to notify the user in case of an internal event. In the context of the platform this feature is very important because it provides useful alerts when problems are identified in the data collected from farms. Furthermore, this feature is used by other services of the application to send informative notifications to users. An important goal for the service is to provide an interface through which any external service can easily send notifications without requiring additional implementations.

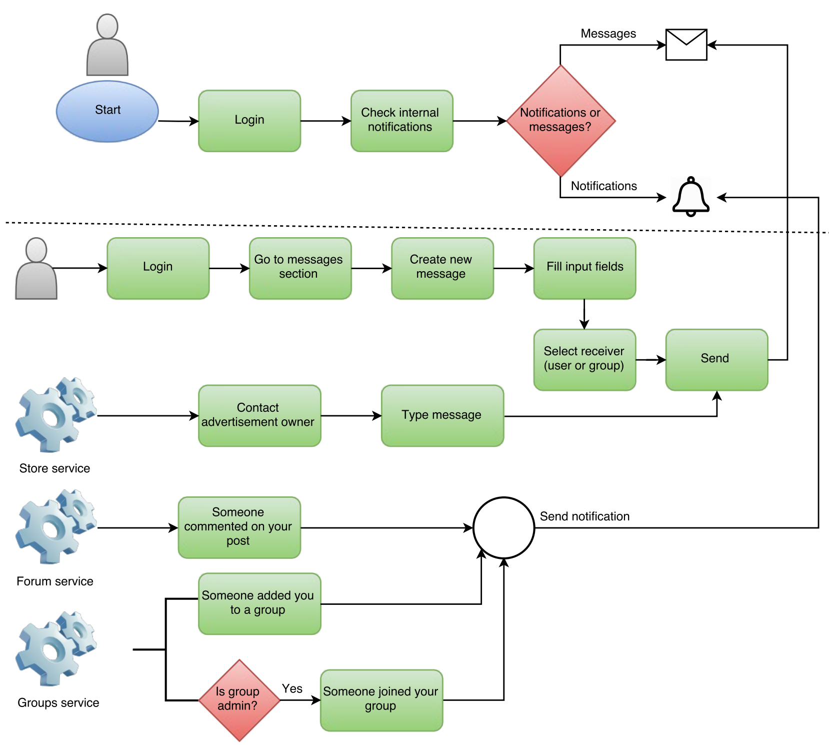

Figure 2: Internal Notification Workflow.

As we said, Internal Notification Service (Figure 2) is a public service which consists of two different features, internal message system and internal notification system. The main difference between the two features is that messages are sent by users while notifications are generated by events from other services.

Messages section consists of an inbox, outbox and a message create page. If a user wants to send a new message he should fill in a form. After pressing the submit button a new message will appear in receiver's inbox. Messages have a status attribute with two possible values, read and unread. Unread messages count will be displayed on each page, in the header section so users can easily access inbox when a new message arrives.

Internal notifications appear only in the page header as a list containing all unread notifications. The user can click on notifications and he will be redirected to a suggestive page in the application.

In message system, input is represented by the following fields:

- Receiver: username or group name;
- Title: message title;
- Message: message text.

The data is sent to the server in JSON format. Output is represented by server response and it is also in JSON format. In notification system, other services should provide the same data structure consisting of user id, notification message and a URL where the user will be redirected after pressing the notification. This structure helps service to create and send notifications to the user. Otherwise, an exception will be thrown and the programmer must identify and fix the problem.

There are three external services that integrate with notification system. Forum Service uses this system to notify a user when he receives a comment on a post. For doing this, every time a user comments on a post, notification system will receive a request and will send a notification to the post author. Groups Service uses this system to notify a group administrator when someone joins the group. Also, a user will be notified when he is invited to join a group.

Store Service uses this service to put user in touch with an advertisement author. On view advertisement page, there is a button that redirects the user to the create message page with the receiver field already filled. In addition, the algorithm that deals with gathering data from farms will use the notification system to send an alert to the user if a problem occurs.

In message system, critical cases can occur when data validations fail. User input should pass the following validations:

- Receiver: required field and valid username or group;
- Title: required field;
- Message: required field and should have more than 5 characters.

Validations are done both on client and server-side. If they are not met, users can not send the message and an error will be displayed. In notification system, critical cases can occur if other services do not respect data structure. In this case, notification system will log the received data to be verified by an administrator. Another problem can occur if user id from data structure does not exist in database. In this case, the notification cannot be added.

3.2. Forum service

Forum is a public service where users can interact on various topics, the main purpose being to help each other. This service is not as complex as dedicated applications because it is designed to provide minimum required support to users (Figure 3).

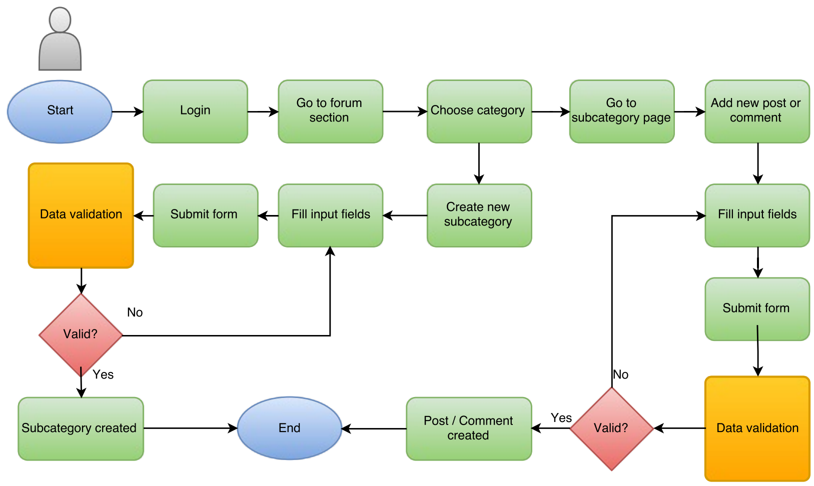

Figure 3: Forum Workflow.

Forum contains categories, subcategories, posts and comments. Categories are read-only, only administrators can add or edit them. Subcategories, posts and comments can be added by any user. Only the owner can edit his post or comment. If a user wants to add or edit a subcategory, a post or a comment, he must fill and submit a form with specific fields. Subcategories cannot be edited, only administrators can do this. Each subcategory has its own page. This page contains all posts in subcategory ordered by creation date. Comments are displayed below each post ordered by creation date, too. When a user comments on a post, the post owner will be notified through another service.

When a user wants to add, or edit a subcategory, a post or a comment he must fill in:

- Subcategory: title
- Post or comment: title, description Data is sent to the server in JSON format. The server processes the data and then sends a success or a failure message to the user. In case of success, the data is stored in the database.

Output is represented by categories and subcategories on the main page and by posts and comments on subcategory page. Output data is also in JSON format.

As we have mentioned earlier, an external service will notify the post owner when another user comments on it. This service is another public service of this platform called Notification Service. In our case, it will be triggered by comment submit button and it sends an email or an internal notification to the post owner. Users can choose between email and internal notification in account settings. Groups service also integrates with the forum service. It automatically creates a forum category when a new group is added. Only group members can add subcategories, posts and comments in this category.

When adding or editing subcategories, posts and comments, critical cases can occur if the following validations fail:

- subcategory title - required field;
- post and comment title - required field;
- post and comment description - required field, more than 5 characters.

Validations are done both on client and on server-side. If client-side validation fails, the user can not submit the form and a message will be displayed. In addition, a critical case will occur if users are trying to create a subcategory with a name existing in the current category. This case is treated to avoid adding redundant subcategories Another critical case can occur when a user is trying to use insults in posts or comments. In this case, after submitting a post or a comment, an algorithm will parse the text and will determine if it contains insults or not. If the algorithm detects something, the post or the comment will not be saved and a message will be displayed. The algorithm will detect only obvious cases. For all others, there will be forum administrators.

3.3. Groups service

Groups service is a public service that allows users to create groups to communicate more easily and to encourage mutual help on different areas of interest. Groups are entities that are designed to bring users with common interests in one place. The functionality of this service is not so useful but other services of the application are responsible to provide some features, the main purpose being to facilitate the communication between group members (Figure 4).

If a user wants to add a new group, he must fill in a form and after validations the group will be saved. After it has been successfully added, the user can access its page and can manually invite other users to join the group. The user who added the group is the only one who can edit it. He can also delete other users from group. A user can join any group without requiring administrator approval and any member of a group can invite other users to join. The user can leave the group at any time and from that moment he will not be included in any event of an external service that involves the group. Each group has its own page where all users can see its members and description. Furthermore, users can send internal messages to all group members. The service will automatically append the group name to the title of the message to inform users that the message was sent to all group members.

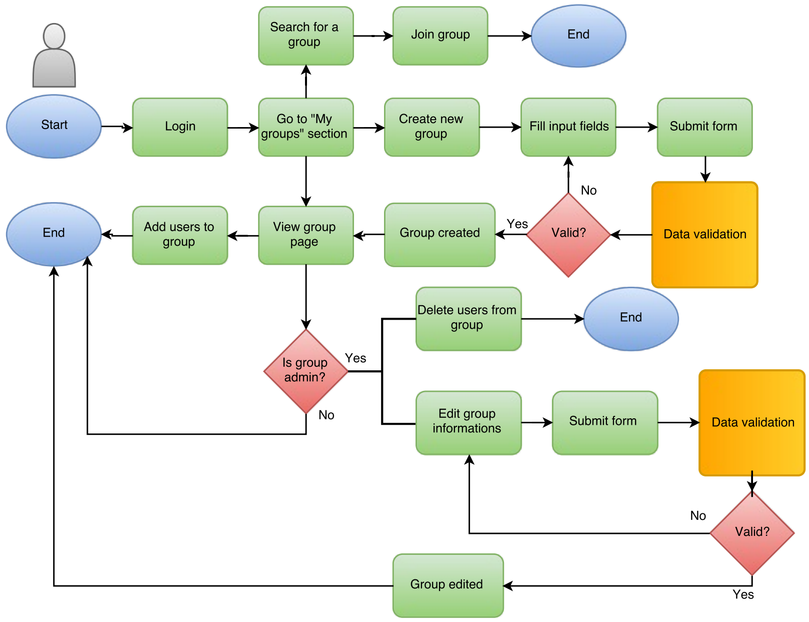

Figure 4: Groups Workflow.

When adding, or editing a group, input is represented by the following attributes:

- Name: a short text that describes the group;
- Description: group's description that will appear on its page.

When adding a user to a group, the input is represented by user's id. The data is sent to the server in JSON format. Server response is also in JSON format and represents a success or a failure message. Description's input field is a "What You See Is What You Get" (WYSIWYG) editor which allows users to format the input text and to insert images, videos and links.

Groups Service is integrated with Forum Service and Internal Notification Service. Forum service uses this service to dynamically create new forum category when a group is added. This category can only be seen by users enrolled in group. Basically, this category represents a private forum section for users in group. Internal Notification Service uses this service to send notifications to a group, not individually, if necessary. It also allows users to send internal messages to all members of a group fast and easy.

There are three critical cases that can occur when working with groups. The first one represents data validation when adding or editing a group. All fields are required and each one has some validations:

- Name: should be unique and should have more than 5 characters;

- Description: should have more than 20 characters to be a relevant description.

For this scenario, validations are done both on client and server-side. If client-side validation fails, the user can not submit the form and a message will be displayed under invalid fields. Another critical case can occur when a user is trying to invite other users to join a group. For any reason, the user that he is trying to invite may not exist. The third critical case can occur when a user is trying to join a group that does not exists. In these two cases, validations are done on the server-side. Even if the search field have an auto-complete function, the user can submit the form at any time and the server should verify if user or group exists in database. If it does not exist an error message will be displayed.

3.4. Store Service

Store service is a public service that allows users to create advertisements. This service comes to help users to rent, sell or buy products needed in farms. It provides a basic implementation for an online store, without dealing with products' stock for example, the main reason being to encourage users to discuss details in private, by email or through internal messaging system (Figure 5).

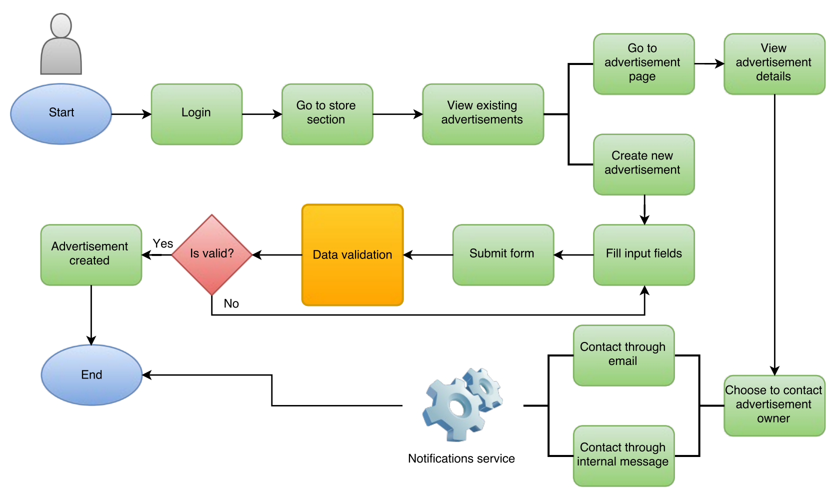

Figure 5: Store Workflow.

In store service, there are three types of advertisements, sale, rental and buy advertisements. Store section consists of a page for each category of advertisement and a page with all advertisements added by current user. Each page contains advertisements published by users sorted by creation date. If a user wants to add a new advertisement he should fill in a form. Store Service does not manage the communication between users that are interested to trade. It simply passes responsibility to Notifications Service. There is a button on advertisement page where users can contact the owner through email or internal message.

Input is represented by following fields when users are trying to add a new advertisement:

- Ad type: advertisement type (rental, sale or buy);
- Title: advertisement title;
- Product name: the product to be rented or sold;
- Price: rental or sale price;
- Description: product description and other details.

Output data is represented by a success or a failure message depending on the validity of the input data. Output is also in JSON format.

As we have mentioned earlier, Store Service integrates with Notifications Service which is responsible for communication between users interested to trade. Given that Notification Service is divided into internal and external notifications, users can choose how they want to communicate. There is also the possibility to prevent the publication of personal email in own advertisements. For doing this, the user should choose this in account settings and other users can contact him only through internal messages.

Critical cases can occur when input data does not meet the following requirements:

- Ad type: should be rental or sale;
- Title: required field, at least 3 characters;
- Product name: required field, at least 3 characters;
- Price: optional field, decimal value;
- Description: required field, at least 5 characters.

Validations are done both on client and server-side. If one of these validations fails, the user can not create new advertisement and error messages will be displayed under invalid fields.

3.5. Tendencies and correlations service

Tendencies and correlations service is a private service that provides useful information obtained through additional processing of data calculated by statistics service (Figure 6).

Tendencies and correlations are displayed in the view farm page on two separate tabs. Data provided by the two features are calculated when accessing the page because the values can change very quickly depending on data received from sensors. Tendencies tab contains a table that displays the increase or

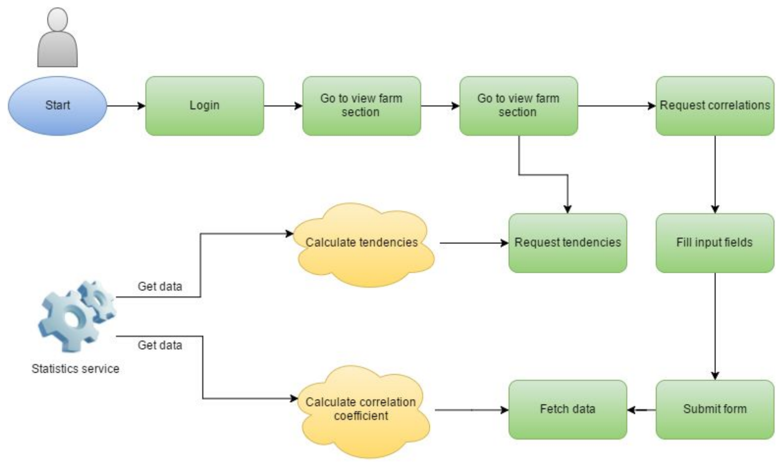

Figure 6: Tendencies and correnlations workflow.

decrease percent of recently received data from sensors compared to the average of the last three values provided by the statistics service. The data obtained gives the user an overview of farm current state compared to the data obtained in the past.

Correlations tab contains a form where the user must choose two parameters and time frame for who wants to calculate the correlation factor. After submitting, the Pearson's correlation coefficient is calculated according to the formula given below and displayed a graph with data from selected parameters. Pearson's correlation coefficient is a number between -1 and 1 and represent linear dependence of the values of the two parameters. In 1 we will consider first parameter values as X and second parameter values as Y.

$$r = \frac{\sum (X - \bar{X}) \times (Y - \bar{Y})}{\sqrt{(X - \bar{X})^2 \times (Y - \bar{Y})^2}} \tag{1}$$

For tendencies feature input is represented by recent data collected from sensors and calculated data in the statistics service. The output is represented by tendencies calculated for each sensor. Correlations also use data provided by the service statistics and, in addition, the user must fill in the following fields:

- The two parameters;
- Time frame: current day/month/year.

Output is represented by correlation coefficient calculated for desired parameters. Furthermore, a chart representing the linear dependency between the two parameters will be displayed.

Both features are integrated with statistics service. They use the data obtained by statistics service to calculate the required parameters. In both features, a critical case may occur if there is no data received from selected sensors. In this situation, a suggestive message will be displayed.

3.6. Database structure

The database of the platform is very extended and the main tables, which have the most connections with the other ones, are represented by the user table and by the farm table. This highlights that the users and the farms are the main entities of the platform. Around them revolve all the implemented services. Further we will describe the tables which are used by every service.

The internal notification service is using two tables: the message and the notification one. These two tables have the same structure, with some small exceptions. Both contain fields which describe the title, the content, the type of the notification or the message. In the same time, both have an attribute which can specify if the notification or the message has been read by the user to whom it is addressed. The date type fields show us the certain time of their adding. The user id area represents a foreign key for knowing its user. It must be known the sender and the receiver of these messages. In addition, there is a reference to the same table which allow the sending of the messages in the same thread of discussion. All the sent messages have a connection with the initial one.

The forum service is using three tables which represent the main entities of the service. These are the categories, the subcategories and the posts. Categories and subcategories contain fields which signify the name, description and the creation time. In addition, subcategory table contain a foreign key that represents the person who added them. It also contains a reference to the forum category table, which shows us from where it belongs. The forum post table has fields which describe the title, the content and the date type fields which represent the creation and edit time. To know the subcategory which includes posts, we added a foreign key to the subcategory table and for retaining the post's author it has been added a reference to the users table. In addition, there is a foreign key referencing the same table which allows adding comments to posts. These comments are posts, too, with a set parent id.

The groups service is using the "user_group" table, where the name and the description can be found. This contains a reference to the user table, which represents its author. In addition, we created another table named "user_group_user" to realize the many-to-many connection between the group and user tables.

The store service is using two tables for saving the added advertisements and for showing the sold, rent, or the bought products. All these advertisements contain the title, the description, the price and the type of the product. We can also find its picture, which is added by the user in a possible future implementation. Every advertisement is related to a farm by using a foreign key to the farm table and to a product from "store_product" table.

For gathering the farms data are being used the following tables:

- remote_param: it describes all the parameters which can be measured in a farm. This contains fields where we can find the name, the description, the unit of measure and other specific attributes of the parameters;
- remote_sensor: it contains fields where we should find the limits of the measured values. In the same time, this table achieves a mapping between the farms and the parameters. It is made by two foreign keys referencing the farm table and the remote_param table;
- remote_data: this table contains the received timestamps from the farms databases and a reference to the main farm, too;
- remote_data_value: this table contains a field that represents the value which is measured by the sensors. It is also necessary a reference to the correspondent line from the remote data table and another to the sensor which has measured the value, from the remote sensor table.

4. Implementation details

For the implementation of the services presented in the previous section, several features are being developed on server-side and client-side to achieve the objectives. In this section, we present details of how services have been implemented in general and we describe some particular elements identified in implementation.

4.1. Backend implementation details

The necessary data for each service are being stored in the database in accordance with the structure presented in the fourth section. Data from each table are mapped into classes called entities. These classes contain all fields presented in tables and objects from relationships with other tables. To avoid a big load, additional collections are taken only in the moment when they are used, by the "lazy-initialization" mechanism. For retrieving data from the database, we used repository interfaces which, by default, provides methods of writing, reading, updating and deleting. For retrieving data with certain clauses, we have built custom queries using Java Persistence Query Language (JPQL) which is as like as the SQL.

The logic of the application is implemented in services. These are singleton reusable components. Every entity has a service with implemented methods for collecting data from database, processing and sending them to the user. For being used, the services are introduced in the web controllers using the dependency injection mechanism, which is offered by the Spring framework. The web controllers receive requests from the frontend application. They must also respond to them with the requested data.

The data are sent to the client through DTO objects, not through entity objects. The DTO objects have only the necessary attributes. It is very important

Table 1: Tendencies table.

Sensor	Hourly tendency (Relative to last 3 hours)	Daily tendency (Relative to last 3 days)	Monthly tendency (Relative to last 3 months)
Temperature (indoor)	21,3 ↑	25 ↓	24.5 ↑
Temperature (outdoor)	32,3 ↓	33,3 ↓	31,3 ↑
Humidity (indoor)	66,3 −	64,3 −	62,3 −
Humidity (outdoor)	68 ↓	65 ↓	64 ↓
Wind speed	25 ↑	26 ↑	28 ↑

to send only the useful data to the frontend application, because their dimension can influence the loading time in a negative way. The mapping between entities and DTOs was realized using MapStruct and constructors. MapStruct is an external library which generates code. It simplifies the mapping between the entities and the DTO by creating an interface with two methods of mapping: entity to DTO and DTO to entity.

As we mentioned in the previous section, the tendencies are calculated when the view farm page is accessed. Firstly, the last values received for all sensors from remote database are taken. For each identified sensor, we use only the last three calculated hourly, daily and monthly statistics. If there are calculated statistics for the respectively sensor, the arithmetic mean will be made in a separated hourly, daily and monthly way. The increasing or decreasing percentage is calculated by using the average value and the last value received from the sensor as in 2, where *PR* represents the Percent Rate, *Vpresent* represents the last value received from the sensor and *Vpast* represents the average value calculated before.

$$PR = \frac{(V_{present} - V_{past})}{V_{past}} \times 100 \qquad (2)$$

If the statistics are not calculated for a certain sensor, then the algorithm moves on. In this case, the customer receives a suggestive message. The data are displayed in a table as in the Table 1 for being followed easier by the user. The table contains the farm's sensors on the lines and on the columns, we can find the time's reference.

Correlations are calculated after the form consisting of the two parameters and the time period is submitted. The algorithm fetch the calculated statistics for the two parameters from the database. As we know, there are three tables where statistics are stored according to the period for which they were calculated. If the algorithm finds data, correlation factor is calculated using the formula (1) defined in the previous section. Considering that the statistics are created for each sensor for the same period, there will be no problems of inconsistency between the two datasets.

4.2. Frontend implementation details

For the frontend part we used the AngularJs framework and Hyper Text Markup Language (HTML) plus Cascading Style Sheets (CSS). For each visible page, three important files are created in frontend. For functionality, we implemented an angular controller where the data received from backend are being processed and then are being set on the $scope variable. That's how we can use them in the template's file. We have also implemented an angular configuration file, where are being defined the URL of the page's accessing, the title, the used template and the permissions of the access. In this file are being overtaken the initial data from the server and is configured the translation module. In template files are being implemented the visual functionalities, by using the AngularJs directives. For receiving data from the server, we used Angular services, which have the role to send requests to the backend application. All requests are sent in JSON format. The server must process the request and provide a response in JSON format, too.

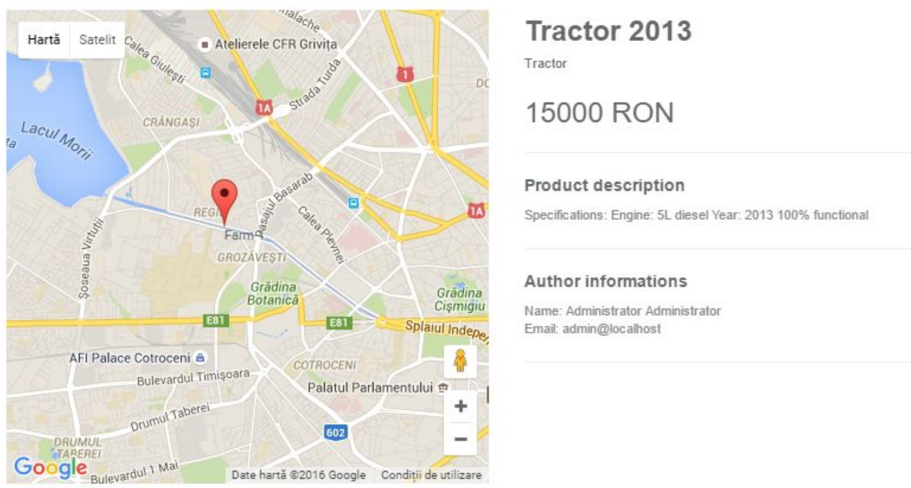

Figure 7: View advertisement page.

As we have discussed in previous sections, in store service each advertisement have reference to a farm. This link is useful because it helps users to find the location where they can buy or rent the product. In order to avoid showing some geographical coordinates to the user we used Google Maps to display a map centered on the farm as in Figure 7 We implemented this functionality by using the AngularJs module called Angular Google Maps. This module provides a set of angular directives that automatically connect with the Google Maps API to simplify the programmer's work. We used "ui-gmap-google-map" directive that renders the map with specific attributes like zoom value and center location. Inside this, we used another angular directive called "ui-gmap-maker" that deals with marking a point on the map according to given coordinates.

4.3. Gathering data from farms

As part of the platform for efficient management of smart farms, an important share is represented by the gathering data from farms' databases. This data will be later processed and other services will offer to user relevant information as regard to their farms. At the same time, the user will be notified by the Notification Service in case that will appear an unexpected event.

We suppose that all farms have databases with identical structure to process and save data on the platform database. It is very important that data are saved in application's database because we need them to make statistics and another request to take the same data from remote databases would be inefficient.

The remote database must contain a table with data received from sensors which has the following structure: IDS (farm id), timestamp and a column for each parameter. At the same time, it is necessary a table of parameters where are specified all parameters measured by sensors from farm containing the following fields: name, unit of measure and other attributes. So, each farm will have sensors which will measure some parameters. In this table are also specified the normal limits between must be the measures.

For gathering data, a job runs periodically. This job creates a *workpool* with four threads, and then, for each farm it is added a task in a queue. The threads process tasks in parallel until all new information of all farms are gathered. The threads and tasks' creation are presented in the following code sequence. First, we have defined the number of threads. We chose to use four threads because it represents the number of cores that we have on the system. If the application will run on a better system, this number can be easily changed.

```
public static final Integer NT = 4; //num threads

public void run() {
        //create workpool
        WorkPool workPool = new WorkPool(NT);

        //get all farms from database
        List<Farm> farms = farmRepository.findAll
            ();

        //create jobs
        for (Farm farm: farms) {
                workPool.putWork(new
                    GatherDataJob(farm,
                    remoteDataRepository,
                    remoteDataValueRepository,
                    remoteSensorRepository));
        }

        //create workers
        Worker[] workers = new Worker[NT];
        for (int i = 0; i < NT; i++){
                workers[i] = new Worker(workPool)
                    ;
```

```
            }

            // start workers
            for (int i = 0; i < NT; i++){
                workers[i].start();
            }
            // wait workers to finish
            for (int i = 0;i < NT; i++){
                try {
                    workers[i].join();
                } catch (Exception e){
                    e.printStackTrace();
                }
            }
        }
```

Task processing is divided on three different steps. The first step is gathering data from the remote database. To gather new added data, queries to the remote database contains the following clause: the timestamp must be bigger than last timestamp saved in database for the current farm. The request to the remote database will return an object which contains all measures made during the last examination. It is important that the timestamp taken from remote database to indicate the moment when the measures have been made. It helps for a good accuracy in the calculation of statistics and in sending of alerts to the user. In the second step, each data received is processed and if the sensor value is not between defined limits, an alert is sent to the user informing him about the measured value. In the third step, processed data are saved in the database. To provide flexibility, the local database has a different structure compared to the remote database. There is no limits related to the number of parameters. In the main table will be kept only the timestamp value and, in another table, we will be keep the values for each parameter in the main table. If there will be a lot of parameters, we will avoid the adding of the columns in the main table. This technique is inspired by the entity-attribute-value (EAV) model.

Entity-attribute-value is a model of database construction known as "vertical database model". It proposes a disposition on the vertical way of the tables with many columns and few values. So, each column, which represents the parameter, is being eliminated from the main table. All these parameters are being added, line by line in a table of parameters. The values for each parameter are then saved in a table of values which have reference to the main table and the table of parameters. If the parameters can have different type of value, for each type of data should be made a new table of values. In the parameters' table, will be specified the type of the data to know the location of the values. That's how the database will be designed but the processing is hardly because are necessary some extra joins to fetch all the values for the parameter.

For storing the data taken from remote databases, we need only a value table because all sensors record float type data. The data which we add here are processed only by the jobs that deals with statistics calculation and run on a different thread.

5. Performance analysis

The performance analysis is extremely useful in any application for appreciating the quality offered by its services. The loading time is a representative factor for the web applications. It is not recommended to make intensive operations on large sets of data after receiving a request because the response time can be influenced and the customer can lose his patience.

In this section we are going to present the time needed for gathering data from farms and saving them in the application and we are going to analyze its evolution depending on various factors. In addition, we analyze the time needed to process the data depending on the number of notifications sent to the user. We are also going to analyze the required time for calculating the tendencies and we'll notice if the user is going to be directly affected within this time.

5.1. Gathering data algorithm

The algorithm of gathering the data from the farms represents a critical point in the application because it announces the user if there is a problem while the data are processed. The time which is between the measurement of a value and when the user receives the notification depends by a lot of factors.

Firstly, there is a time until the gathering data's job starts to run. This temporal value cannot be estimated because it is not known the time when a problematic measurement can appear. However, we can tell that the value is between 0 and the frequency at which the job is set to run. The time spent by the farm in the tasks' queue is the next factor.

Because we don't know the number of farms which will be processed before the respectively farm, we can't calculate or estimate this factor. The connection to the remote database and its response time represent another factor. Finally, the processing of each line and the sending time of the notification represent the last factor.

We are going to simplify the equation supposing that the factors that cannot be measured or estimated have the 0 value. In measurements, we consider that the farm's database is on the same system with the platform. In this way, we eliminate the possible latency which can appear because of the network.

In Table 2 we can see the execution time of the gathering data algorithm. We chose to measure the time during step two and step three together because we assumed that in step two will not be sent any notification to the user. Thus, the data processing time is negligible and we decided to measure it together with step three.

As we can notice, in measurements, we used to vary two parameters. The first parameter is represented by the number of new measurements identified in greenhouse's database from the last execution of the algorithm. By varying this parameter, we can analyze how the algorithm can manage large amount of

Table 2: Gathering data algorithm - time performance.

Number of measurements received per sensor and the number of sensors	Step1 duration (ms)	Step2 + Step3 duration (ms)	Total time (ms)
100 with 1 sensor	100	5300	5400
100 with 3 sensor	115	6100	6215
100 with 7 sensor	98	7300	7398
300 with 1 sensor	115	16500	16615
300 with 3 sensors	102	18400	18502
300 with 7 sensors	120	21300	21420
500 with 1 sensors	118	28600	28718
500 with 3 sensors	112	29700	29812
500 with 7 sensors	131	32400	32531
1000 with 1 sensor	122	45200	45322
1000 with 3 sensors	121	48300	48421
1000 with 7 sensors	110	50400	50510

data. We did these measurements only to test the performance of the algorithm because, in production, it will run quite often so the amount of new data will be quite small.

The second parameter is represented by the number of sensors related to current farm. Varying this parameter helps us to understand how the "entity-attribute-value" model used to store the data received from sensors can influence the execution time of the algorithm.

First, we analyze the results by varying the second parameter. As we can see, the step one duration is not influenced by the variation of the number of sensors. This is normal because the table from the remote database contains one column for each sensor and thus no matter how many sensors the farm has, a measurement will mean a single line in this table. So, the number of lines received from the database does not depend on the number of sensors but the number of new measurements. As we can see, there is a huge difference between the duration of the first step and the duration of the second and third step. This can be explained by the fact that in the first step we read data from a remote database and in the third step we must save this data in the local database which consume more time. The time differences in step two and step three observed by varying the number of sensors are caused by local database structure. As we know from the previous section, "entity-attribute-model" implies additional inserts to save the value of each sensor. So, if we have 100 new measurements with 7 sensors, we should insert 100 lines in the main table ("remote_data") and 100 * 7 lines in the value table ("remote_data_value"). Comparing with the 100 new measurements for one sensor case, we have to insert 600 more values in the value table. As we can see, the time difference is not as big as we would expect from theoretical analysis.

Secondly, we will analyze the result by varying the second parameter. As we can see, the necessary time to execute the first step is almost constant which shows that the remote database selects data from the table very quickly. If we will try to read a larger volume of data, the execution time will certainly increase. As we can see, in the second and third step, the time required to process and save the data increases linearly. We can estimate that the saving of 100 measurements regardless of the number of sensors lasts about six seconds.

In the next time, we will analyze the speed of processing data received from the remote databases. Just before, we analyzed the execution time of steps one and three and now, we will analyze only the second step. An important parameter that directly influences the duration of the data processing is the number of notifications needed to be sent to the user. In order to observe how this parameter influences the processing time, we will consider a fixed number of measurements in tests.

As we can see in Figure 8, we measured the execution time of the second step by varying two parameters: the number of new measurements received from the remote database and the percentage of measurements for which notification was sent. First of all, we decided to vary percentage of measurements for which notification was sent because this can show us how the sending of notifications influence the data processing. The dependence between the execution time and the percentage of notifications sent is not linear (while the execution time grows by 4 times, the amount of notifications sent grows by 5 times). Secondly, we decided to vary the number of measurements received from the remote database because this can show us how execution time evolves depending on the number of measurements. The execution time increases linearly, depending on the number of measurements to be processed.

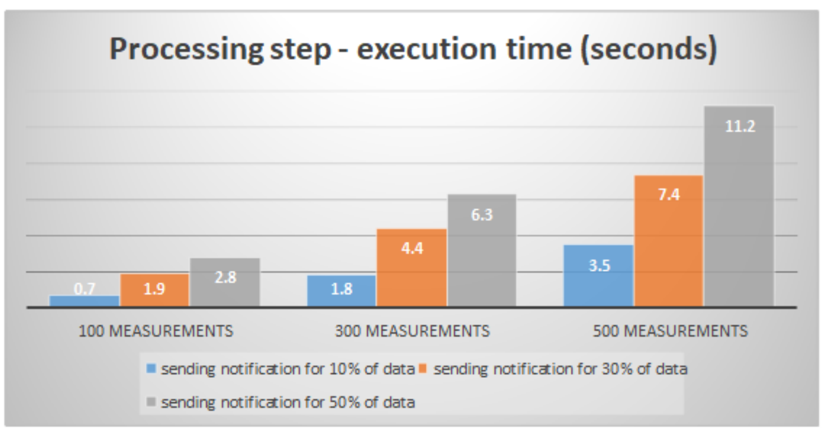

Figure 8: Processing step – performance analysis.

In conclusion, the longest period is used to save data in the local database, data processing time depends linearly on their amount and the fastest task is represented by the retrieving of data from the remote database.

In order to improve the performance of the services offered by the platform we could move it into the Cloud. Cloud providers offer a wide range of options to transfer data into (or out of) cloud platform through simple and reliable APIs. For instance, AWS Direct Connect provides consistently high bandwidth and low latencies for transferring large amounts of data to AWS using a dedicated network connection. Also, the cloud storage service has a CDN (content distribution network) close to the network edge that improve the latency.

The platform is scalable and can support unlimited number of farms if we move it into the Cloud. Regarding data distribution and collection over multiple farms, the system is designed to collect data from a large geographical area (e.g. surface of Romania).

In the case when connectivity with the monitoring service is lost the user is informed about the current state of connection with platform services. The monitored data is stored locally until the network connection is restored.

5.2. Calculation of tendencies

As we mentioned in previous sections, tendencies calculation represents a critical area within the application because the loading time of the view farm page depends on the rapidity with which these tendencies are calculated.

As we can see in Figure 9, the time necessary to calculate tendencies for 1 to 7 sensors is between 100 and 500 milliseconds. This time is decent and cannot influence the user experience in a visible way. We can say that user will be affected only if he has more than 30 sensors on the farm. We can notice that the dependence between the time and the number of sensors is linear.

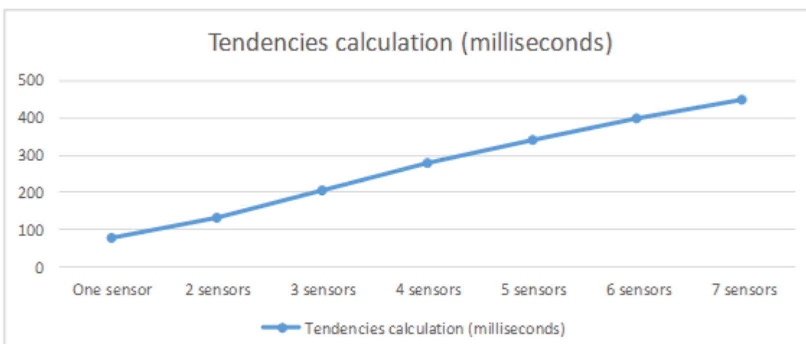

Figure 9: Tendencies calculation – performance analysis.

6. Conclusion and future work

In this paper we have presented a platform which will offer to the farmers a place where they will can manage their smart farms in an efficient manner, but a place where they can interact with another farmers. In the first time, we have proposed a software architecture based on two independent applications, frontend application and backend application, this two communicating by REST services or by web sockets. This architecture, used technologies as well as the implementation offers to users a pleased experience on any device with very low response time. So, we achieved an important objective of the project.

Forwards, we have been described a part of platform's services: internal notification service, forum service, groups service, store service and tendencies and correlations service. We have been highlighted each operating mode, input and output, integration with external services and critical cases that can occur. In addition, we have been described the database and the tables used in services' implementation. Implementation details have been brought a clear vision around how services were created. In the same time, we have been described the data gathering algorithm from farm, realized in 3 steps, gather, process and save in the local database. Finally, we have presented some performance criteria about the data gathering from farms. We have been analyzed the time of each step and we have been measured the time of data processing according to the number of notifications which must be sent to users in case of a problem. In the same time, we have been analyzed the time of tendencies calculation, this time directly influencing the loading time of view farm page.

In the future implementations, we intend to integrate the platform with an external service which sends SMS. This feature would be very useful for sending critical notifications by SMS, too. In this way, the user will be informed by the occurrence of an error, directly on the phone, not being necessary to access the platform or the email account. Inside the project, we have intended to integrate such a service, but we have found a few solutions and these offered only paid services.

Another feature which we intend to implement is represented by creating an Application Programming Interface (API) through which users that have not the correct structure of database will can send data directly in the platform. These data must be sent into a standard structure and in the application, these will be processed in the same manner of actual implementation. This feature is important because it comes to help users which cannot afford to modify the database structure of the farm as well as the functionality. A script which regularly sends data to platform is easier to implement than to change the database structure.

In the future, we intend to develop a mobile application for the platform. Even if the platform, in the actual state, is responsive and it look fine on all mobile devices, a native application would help the farmers by sending notifications directly to mobile, not inside of the platform. Finally, we want to analyze each implemented service and we want to identify where we can bring new features for a better experience for the users.

By this platform, we have tried to create an environment where farmers can efficiently manage their farms and, simultaneously, help farmers to interact, the main purpose being to help the production to increase. As use case for deploying the platform in real-world environment, it can be offered as a package for a farm together with the monitoring system situated at farm site.

Future research also include integrating forensic principles in the design of the platform, an approach coined as forensic-by-design by Ab Rahman et al. [30], [31]. Grispos, Glisson and Choo also emphasize the importance of having a forensic-drive approach when ensuring the security and resiliency of cyber physical systems [32].

Acknowledgment

The research presented in this paper is supported by projects: *DataWay*: Real-time Data Processing Platform for Smart Cities: Making sense of Big Data - PN-II-RU-TE-2014-4-2731; *clueFarm*: 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.

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

References

[1] I. You, J. Choi, C. Choi, P. Kim, Intelligent healthcare service based on context inference using smart device, Soft Computing 18 (12) (2014) 2577–2586.

[2] K.-K. R. Choo, The cyber threat landscape: Challenges and future research directions, Computers & Security 30 (8) (2011) 719–731.

[3] K.-K. R. Choo, A conceptual interdisciplinary plug-and-play cyber security framework, in: ICTs and the Millennium Development Goals, Springer, 2014, pp. 81–99.

[4] J. Liu, X. Yu, Z. Xu, K.-K. R. Choo, L. Hong, X. Cui, A cloud-based taxi trace mining framework for smart city, Software: Practice and Experience 47 (8) (2017) 1081–1094.

[5] M. Tao, J. Zuo, Z. Liu, A. Castiglione, F. Palmieri, Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes, Future Generation Computer Systems.

[6] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future generation computer systems 29 (7) (2013) 1645–1660.

[7] D. Merezeanu, G. Vasilescu, R. Dobrescu, Context-aware control platform for sensor network integration in iot and cloud, Studies in Informatics and Control 25 (4) (2016) 489–498.

[8] M. S. Shin, B. C. Kim, S. M. Hwang, M. C. Ko, Design and implementation of iot-based intelligent surveillance robot, Studies in Informatics and Control 25 (4) (2016) 421–432.

[9] A. Chianese, F. Piccialli, G. Riccio, Designing a smart multisensor framework based on beaglebone black board, in: Computer Science and its Applications, Springer, 2015, pp. 391–397.

[10] Y. Wang, X. Dai, J. J. Jung, C. Choi, Performance analysis of smart cultural heritage protection oriented wireless networks, Future Generation Computer Systems.

[11] Y. Wang, X. Su, D. Choi, C. Choi, Coordinated scheduling algorithm for system utility maximization with heterogeneous qos requirements in wireless relay networks, IEEE Access 4 (2016) 8351–8361.

[12] M. Roopaei, P. Rad, K.-K. R. Choo, Cloud of things in smart agriculture: Intelligent irrigation monitoring by thermal imaging, IEEE Cloud Computing 4 (1) (2017) 10–15.

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

[14] A. Serrouch, M. Mocanu, F. Pop, Soil management services in cluefarm, in: Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on, IEEE, 2015, pp. 204–209.

[15] A. Kaloxylos, R. Eigenmann, F. Teye, Z. Politopoulou, S. Wolfert, C. Shrank, M. Dillinger, I. Lampropoulou, E. Antoniou, L. Pesonen, et al., Farm management systems and the future internet era, Computers and electronics in agriculture 89 (2012) 130–144.

[16] V.-C. Bojan, I.-G. Raducu, F. Pop, M. Mocanu, V. Cristea, 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, IEEE, 2015, pp. 425–432.

[17] A. Chianese, F. Marulli, V. Moscato, F. Piccialli, A" smart" multimedia guide for indoor contextual navigation in cultural heritage applications, in: Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on, IEEE, 2013, pp. 1–6.

[18] S. Cuomo, P. De Michele, A. Galletti, F. Piccialli, A cultural heritage case study of visitor experiences shared on a social network, in: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015 10th International Conference on, IEEE, 2015, pp. 539–544.

[19] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, R. Buyya, Big data computing and clouds: Trends and future directions, Journal of Parallel and Distributed Computing 79 (2015) 3–15.

[20] M. S. Hossain, M. A. Rahman, G. Muhammad, Cyber–physical cloud-oriented multi-sensory smart home framework for elderly people: An energy efficiency perspective, Journal of Parallel and Distributed Computing 103 (2017) 11–21.

[21] Y. Sun, H. Song, A. J. Jara, R. Bie, Internet of things and big data analytics for smart and connected communities, IEEE Access 4 (2016) 766–773.

[22] J. Shriver, J. Soloff, N. Molen, Data driven farming: Delivering the benefits of remotely sensed data and decision support tools to farmers, in: AGU Fall Meeting Abstracts, Vol. 1, 2014, p. 0225.

[23] L. M. Jespersen, J. P. Hansen, G. Brunori, A. L. Jensen, K. Holst, C. Mathiesen, I. Rasmussen, Ict and social media as drivers of multi-actor innovation in agriculture–barriers, recommendations and potentials, Aarhus, Denmark: International Centre for Research in Organic Food Systems. Retrieved from http://icrof. eu/pdf/2013% 20SCAR_CWG_AKIS_ SocialmediaICTandinnovationprocesses% 20report 2031 (2005) (2013) 202013.

[24] D. Nainggolan, M. Termansen, M. S. Reed, E. D. Cebollero, K. Hubacek, Farmer typology, future scenarios and the implications for ecosystem service provision: a case study from south-eastern spain, Regional Environmental Change 13 (3) (2013) 601–614.

[25] M. Ryu, J. Yun, T. Miao, I.-Y. Ahn, S.-C. Choi, J. Kim, Design and implementation of a connected farm for smart farming system, in: SENSORS, 2015 IEEE, IEEE, 2015, pp. 1–4.

[26] S. R. Rupanagudi, B. Ranjani, P. Nagaraj, V. G. Bhat, G. Thippeswamy, A novel cloud computing based smart farming system for early detection of borer insects in tomatoes, in: Communication, Information & Computing Technology (ICCICT), 2015 International Conference on, IEEE, 2015, pp. 1–6.

[27] D. Gorgan, V. Bacu, D. Rodila, F. Pop, D. Petcu, Experiments on esip—environment oriented satellite data processing platform, Earth Science Informatics 3 (4) (2010) 297–308.

[28] C. Gruia, F. Pop, V. Cristea, Distributed algorithm for change detection in satellite images in grid environments, in: Parallel and Distributed Computing, 2007. ISPDC'07. Sixth International Symposium on, IEEE, 2007, pp. 41–41.

[29] D. Petcu, D. Zaharie, D. Gorgan, F. Pop, D. Tudor, Mediogrid: a grid-based platform for satellite image processing, in: Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007. 4th IEEE Workshop on, IEEE, 2007, pp. 137–142.

[30] N. H. Ab Rahman, N. D. W. Cahyani, K.-K. R. Choo, Cloud incident handling and forensic-by-design: cloud storage as a case study, Concurrency and Computation: Practice and Experience 29 (14).

[31] N. H. Ab Rahman, W. B. Glisson, Y. Yang, K.-K. R. Choo, Forensic-by-design framework for cyber-physical cloud systems, IEEE Cloud Computing 3 (1) (2016) 50–59.

[32] G. Grispos, W. B. Glisson, K.-K. R. Choo, Medical cyber-physical systems development: A forensics-driven approach, in: Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2017 IEEE/ACM International Conference on, IEEE, 2017, pp. 108–113.

*Author Biography & Photograph

Biographies

George-Alexandru Musat is a Computer Science diplomat engineer graduated University Politehnica of Bucharest, Faculty of Automatic Control and Computers in 2016. He is currently studying Advanced Software Systems within the same faculty and he is working as a software engineer in a Romanian company. His research interests include distributed systems, scalable Cloud architectures and Internet of things.

Mădălin Colezea is a Computer Science diplomat engineer graduated University Politehnica of Bucharest, Faculty of Automatic Control and Computers in 2016. He is currently studying Advanced Software Systems within the same faculty and he is working as a software engineer on a multinational company. His current research interests include areas such as high performance Cloud systems, Internet of Things and big data analysis.

Florin POP received his PhD in Computer Science at the University POLITEHNICA of Bucharest in 2008. He received his MSc in Computer Science in 2004 and the Engineering degree in Computer Science in 2003, at the same University. He is Professor within the Computer Science Department and an active member of Distributed System Laboratory. His research interests are in scheduling and resource management, multi-criteria optimization methods, Grid middleware tools and applications development, prediction methods, self-organizing systems, contextualized services in distributed systems. He is the author or co-author of more than 150 publications (books, chapters, papers in internationals journals and well-established and ranked conferences). He served as guest-editor for *International Journal of Web and Grid Services* and he is Managing Editor for *International Journal of Grid and Utility Computing*. He was awarded with "Magna cum laude" distinction for his results during his PhD, one IBM Faculty Award in 2012 or the project "*CloudWay – Improving resource utilization for a smart Cloud infrastructure*", two Prizes for Excellence from IBM and Oracle (2008 and 2009), *Best young researcher in software services Award*, FP7 SPRERS Project, Strengthening the Participation of Romania at European R&D in Software Services in 2011 and two Best Paper Awards. He worked in several international (EGEE III, SEE-GRID-SCI, ERRIC) and national research projects in the distributed systems field as coordinator and member as well. He is scientific researcher within National Institute for Research and Development in Informatics (ICI), Bucharest. He is a senior member of the IEEE, ACM and euroCRIS.

Catalin Negru is a system engineer and researcher at the Computer Science Department of the Faculty of Automatic Control and Computers and an active member of Distributed System Laboratory at University Politehnica of Bucharest. He obtained his PhD from the same faculty in 2016. His research interests include distributed systems, energy efficiency, cloud storage, cyber-physical systems, GIS. His research has led to the publishing of numerous papers and articles at important scientific journals (such as Future Generation Computer Systems, Soft Computing, International Journal of Applied Mathematics and Computer Science and conferences (IEEE, PODC, ICCP, AQTR, HPCC, CISIS etc.). He is involved in several national and international research projects.

Mariana Mocanu is a professor of the Computer Science Department, University Politehnica of Bucharest, and has a long experience in developing information systems for industrial and economic processes, and in project management. She performs teaching for both undergraduate and master's degree in software engineering, systems integration, software services and logic design. At the University of Regensburg, as visiting professor, she thought Process Computers. She worked for ten years in a multidisciplinary research team for vehicles, being co-author of two patents. She participated in numerous research projects, implementing information systems for control / optimization of processes in various areas (transport, environment, medicine, natural resources management). Her results are reflected in articles published in journals, in papers presented at national and international conferences, and books.

Christian Esposito received the Ph.D. degree in computer engineering and automation from the University of Napoli ``Federico II'', Italy. Actually, he is adjunct professor at the University of Naples "Federico II", Italy, and at the University of Salerno, Italy, where he is also a research fellow. He regularly serves as a reviewer and guest editor for several international journals, and conferences (with about 200 reviews being done). He has been a PC member or involved in the organization of about 40 international conferences/workshops. He also serves as guest editor for several journals, and member of two editorial boards. His research interests include reliable and secure communications, middleware, distributed systems, positioning systems, multi-objective optimization, and game theory.

Aniello Castiglione received the Ph.D. degree in computer science from the University of Salerno, Italy. Actually, he is an Adjunct Professor with the University of Salerno, and University of Naples '``Federico II'', Italy. He received the Italian national habilitation as an Associate Professor of Computer Science. He serves as a Reviewer for around 50 international journals, He acted as a Guest Editor for several journals. He also serves as a Managing Editor of two international journals, an Editor of several editorial boards. He served as a Program Chair and TPC Member of around 90 international conferences. He has authored more than 140 papers in international journals and conferences. He has been involved in forensic investigations, collaborating with several Law Enforcement Agencies as a Consultant. His current research interests include information forensics, digital forensics, security and privacy on Cloud, communication networks, and applied cryptography.