





**GRADO DE INGENIERÍA INFORMÁTICA DEL SOFTWARE**

|  |  |
| --- | --- |
|  |  |

**GestUsers: User Management System**

**Authors**: **Team I2A1 (Loader & Incident Manager):**

**Juan Aza Gutiérrez**

**Lorena Castillero Corriols**

**Marcos Chacón Cadenas**

**Team I2A2 (Agents & Incident Dashboard):**

**Alba Cotarelo Tuñón**

**Javier Díez García**

**Jesús Atorrasagasti García**

**Date**: 09/03/2018

**Version**: 2018.EN.Deliverable02

**Table of Contents**

[**Introductions and Goals**](#_2et92p0) **10**

[**Requirements**](#_tyjcwt) **11**

[Loader](#_1t3h5sf) 11

[Agents](#_4d34og8) 12

[Incident Management](#_4wpuq7tuxgxj) 12

[Incident dashboard](#_2zhfrlf3z0s) 13

[**Methodology**](#_2s8eyo1) **13**

[**Stakeholders**](#_17dp8vu) **14**

[Students that develop the assignment](#_3rdcrjn) 14

[System administrator](#_26in1rg) 14

[Users](#_lnxbz9) 14

[Developers of the Agents System](#_35nkun2) 15

[Course teachers](#_1ksv4uv) 15

[4.6 Incident Management Staff](#_949yxi4sgstc) 15

[**Quality Attributes**](#_44sinio) **15**

[List of Quality Attributes](#_2jxsxqh) 17

[Quality Attributes and stakeholders](#_z337ya) 18

[**Architecture Constraints**](#_3j2qqm3) **18**

[Technical constraints](#_1y810tw) 18

[Organizational Constraints](#_4i7ojhp) 19

[**System scope and context**](#_2xcytpi) **19**

[**Quality Scenarios**](#_3whwml4) **21**

[**Views**](#_2bn6wsx) **26**

[Context](#_qsh70q) 26

[Main overview](#_1pxezwc) 26

[Elements Catalogue](#_49x2ik5) 27

[Elements](#_2p2csry) 27

[Relationships](#_147n2zr) 27

[Interfaces /Ports](#_3o7alnk) 27

[Loader](#_23ckvvd) 27

[Agents](#_ihv636) 27

[DataBase](#_32hioqz) 27

[Behaviour](#_1hmsyys) 28

[Loader](#_41mghml) 28

[Loader](#_2grqrue) 28

[Agents](#_vx1227) 28

[Agents](#_3fwokq0) 28

[DataBase](#_1v1yuxt) 28

[Users List](#_4f1mdlm) 29

[Main overview](#_2u6wntf) 29

[Catalogue of Elements](#_19c6y18) 29

[Elements](#_3tbugp1) 29

[Relationships](#_28h4qwu) 30

[Interfaces/ Ports](#_nmf14n) 30

[Parser](#_37m2jsg) 30

[DBUpdate](#_1mrcu09) 30

[ReportWriter](#_46r0co2) 30

[Parser](#_2lwamvv) 30

[DBUpdate](#_111kx3o) 31

[ReportWriter](#_3l18frh) 31

[Context Diagram](#_206ipza) 31

[Rationale](#_4k668n3) 31

[Agents](#_2zbgiuw) 32

[Main overview](#_1egqt2p) 32

[Catalogue of elements](#_3ygebqi) 32

[Elements](#_2dlolyb) 32

[Relationships](#_sqyw64) 33

[Interfaces / Ports](#_3cqmetx) 33

[Agents](#_1rvwp1q) 33

[DBManagement](#_4bvk7pj) 33

[Behaviour](#_2r0uhxc) 33

[Agents](#_1664s55) 33

[DBManagement](#_3q5sasy) 34

[Context Diagram](#_25b2l0r) 34

[Rationale](#_kgcv8k) 34

[InciManagement](#_pulchmwszea4) 36

[9.4.1 Main Overview](#_xszg4vg6ju3r) 36

[9.4.2 Catalogue of Elements](#_2g90g0r3ses) 36

[9.4.2.1 Elements](#_rkpicse3xxe6) 36

[9.4.2.2 Relationships](#_ln3dimupz9e8) 36

[9.4.2.3 Interfaces / Ports](#_4j50sfql38xi) 37

[9.4.2.4 Behaviour](#_5d1j4ydoa84b) 37

[9.4.3 Context Diagram](#_od9t97ecfz71) 37

[9.4.4 Rationale](#_2bbvl2g1jiwe) 37

[Incidents Dashboard](#_xbjige4p07k5) 37

[Main overview](#_1jlao46) 38

[Catalogue of elements](#_43ky6rz) 38

[Elements](#_2iq8gzs) 38

[Relationships](#_xvir7l) 38

[Interfaces / Ports](#_3hv69ve) 38

[Behaviour](#_1x0gk37) 39

[Context Diagram](#_4h042r0) 39

[Rationale](#_2w5ecyt) 39

[Deployment View](#_1baon6m) 40

[Main overview](#_3vac5uf) 40

[Catalogue of elements](#_2afmg28) 40

[Elements](#_pkwqa1) 40

[Relationships](#_39kk8xu) 40

[Interfaces / Ports](#_1opuj5n) 40

[Behaviour](#_48pi1tg) 40

[Context Diagram](#_2nusc19) 40

[Rationale](#_1302m92) 40

[**References**](#_3mzq4wv) **40**

# Introductions and Goals

The goal of this document is to describe the structure of an architecture of User Management that will be reused. Although the system that we describe has its own functionality, the main goal is that it will be part of a general system of agent participation.

This document describes the first deliverable of the laboratory assignment of the course "*Software Architecture*" which is taught by the authors. The course is part of the Degree in Software Engineering, School of Computer Science Engineering, University of Oviedo.

The system is divided in two parts: Loader, to load data about users (which can be Persons, Sensors, Entities, etc) and Agents, to check if a user can participate. The students have to implement the software described in this document in two teams of 3 or 4 students during 3 weeks. One team will implement the Loader module sub-system and the other team will implement the Agents module.

In the next deliverables, the students will create the architecture and implement a prototype of the rest of the Participation System of which the current system will be part of.

# Requirements

User Management will be divided in several parts:

* Loader: loads the data from agents that can submit incidences to the system.
* Agents: checks if an agent can participate
* Incident Management(InciManagement)
* Incident Dashboard(InciDashboard)

## Loader

The loader module must be able to introduce data from the agents that can submit incidences to the system. The main difference between last year’s implementations is that there are different kinds of agents: physical people, entities, sensors, etc.

Each type of user that takes part within the system will be identified through a master.csv file as the following: “Person”, “Entity”, “Sensor” .

The introduction of data will be made from Excel files that contain a list of rows with the following information:

* Name (in the case of a person, it will contain both first and last name)
* Location (geographical coordinates). This value is optional for people and entities. If there is not location, the value will be blank.
* Email: Contact email. In the case of sensors or other kinds of automatic agents, it can contain the email of the admin person who manages it.
* Identifier: Agent's ID. In case of people or entities, it can be the National ID (Spanish NIF). This identifies will be unique in the system and will be the user name.
* Kind: Integer number that represents the agent's kind.

When importing the users data, the system will create a user (whose login name will be the identifier field) and a random password which will enable the user to enter the system to check if the data is correct as well as to later participate in the system. The system will generate personal letters that will be sent to each user by email. This task will be done by the Council and is not part of this system.

If a user appears in two different lists, this event will be recorded and informed in a log file. A user can only be created once. If the data is different from the current data available in the system, the current data will not be modified and an error will be recorded in the log.

The system now emits the letters using formats like .txt, Microsoft Word or PDF.

If the input file contains errors, the system must detect them and report the errors found.

[**Optional**] The input data parser can be configured to accept data in different formats. Although it is mandatory to import data in Excel format, the system should be ready to be extended in the future to accept other formats easily.

(**Optional**) The service can be extended to handle security aspects

## Agents

Agents should be able to login into the system to check that they can submit incidents once the notification letter has been received. In order to implement that feature, a simple web service will be created that has two parameters passed as a POST message: login name and password and returns the data available about the agent if the information is correct or reports an error if it isn't. Both the call parameters and the return information will employ JSON format.

(**Optional**) The web service can be extended to offer a simple HTML interface where a user can login and see his information in a human-friendly way.

(**Optional**) Using HTTP content negotiation, the system could handle other formats as XML.

(**Optional**) The service can be extended to enable the user to change his password.

(**Optional**) The service can be extended to handle security aspects

## Incident Management

This module will process the incidents that are delivered by the different agents. The module will request information to module 2 (Agents) to check if the agents are registered in the system and are allowed to submit incidents.

There are several kinds of agents that can submit incidents: people, entities, sensors, etc. The kind of agents is not predetermined as it will be obtained from the same master file described in the first deliverable.

Each incident can have the following fields: username and password, incident name, description, location (which will be automatically obtained if possible), tags (list of words separated by commas that will allow to categorize incidents), additional information (pictures, videos, etc). For some incidents, a list of fields can also be added with the form "property/value", where the property field refers to a property name and the value field refers to the value of that property.

If the combination "username/password" is not right, incidents will not be processed and an error will be reported. If the combination is right, incidents will be processed and the processed incident will be submitted to Apache Kafka for its further handling by module 4 (InciDashboard).

Some agents like sensors can continuously submit incidents depending on how they are configured. For example, a temperature sensor can be configured to emit incidents each hour with information about some place's temperature.

Incidents will acquire different states (open, in-process, closed, cancelled) as well as other information generated by the system like assigned operator/entity that must handle the incident, comments about the incident, etc. Incidents can also have an expiration, for example, in the case of the incidents submitted by a temperature sensor, if they are submitted each hour, the expiration time will be an hour).

The system will only store a selection of incidents (those submitted by a person or entity, or some specific values submitted by sensors, not all of them). Those agents will be able to query and visualize the incidents that they have submitted using the username and password generated in module 1. Each agent can have access to see the incidents that he or she has submitted and follow their evolution.

**[Optional] In** the case of people or entities, the incident submission system will offer a conversational user interface using a chat-bot that will be asking information to the user about the incident.

**[Optional]** An alarm system will be developed to send some specific kinds of incidents to some operators so they can assign them.

## Incident dashboard

The incidence dashboard will be used by the incident management staff and can visualize and manage the incidents that appear in the system.

The dashboard will receive incidents submitted by Incident Manager through Apache Kafka.

Some agents, like sensors, will continually submit incidents with some property values. The dashboard will be configured to know which values are allowed and how to classify the property values. In case that some property values are dangerous the system will notify the operators so they can take the corresponding actions.

The system will offer a continuous monitoring about the evolution of most representative property values, as well as the incidents that are being generated by some person or entity. It will also offer the possibility to visualize the geo-located incidents in a map, as well as the current values, their states, and the historical values of some of the incidents.

The system will offer information to the operators about the incidents they have assigned and will allow them to control those incidents, changing the state of the incident in it has been processed or not, or adding comments about the incident.

**(Optional)** The system will offer an administration module that will enable administrative people to interactively configure the behaviour of the dashboard.

**(Optional)** The system can offer some graphical visualizations or statistics about incidents.

# Methodology

This document employs the ADD (Atribute-Driven Design) methodology (Bass, Clements, & Kazman, 2003) and the SEI norm (ANSI/IEEE 1471, 2000).

The templates have also been inspired by the Arc42 templates (<http://arc42.org/>) where documentation architecture templates are defined in English, German and Spanish.

Another project that follows those templates for a biking domain is available at:

<http://biking.michael-simons.eu/docs/index.html>

# Stakeholders

The main stakeholders identified in the system are:

1. Students from team I2A1
2. Students from team I2A2
3. Course Teachers

There also are other stakeholders which belong to business environment:

1. System Administrator
2. Users
3. People responsible of the participation system
4. Incident Management Staff

|  |  |  |
| --- | --- | --- |
| Code | Stakeholder | Interests (Modules) |
| ST-01 | Students | Both |
| ST-02 | System administrator | Load files |
| ST-03 | Users | Check data |
| ST-04 | Developers of Participation System | Check data |
| ST-05 | Course Teachers | Both |
| ST-06 | Incident Management Staff | Incidence Dashboard |

**Table 1. List of stakeholders/interests**

## Students that develop the assignment

This group is formed by the team that will develop the system. Some of their goals are:

* Use of known technologies and methodologies minimizing the risks to learn new ones.
* Learn how to develop software collaboratively and in a professional way
* Use similar technologies to the group with whom they will work later to minimize incompatibilities.

## System administrator

This is the person who is in charge of loading the users list.

Some of the goals are:

* Use of simple and well-known technologies for input files
* Files that can be read by humans.
* Be able to automate the loading process.
* Be able to debug the loading process in case of failures

## Users

These are the final users of the system, the Agents. Some of their goals are:

* Get access to the system in a simple way
* Being able to query their status in the system
* Submit incidents
* Being able to update or change their information in the system, for example, their password (**Optional**)

## Developers of the Agents System

This is the team that will implement the participation system. Some of their goals are:

* Have a simple way to detect if an agent can participate in the system as soon as possible
* Use of simple technologies that can interoperate with other systems

## Course teachers

They are responsible for the results of this assignment. Some of their goals are:

* Use technologies that help students acquire skills related with Software Architecture by developing a practical assignment.
* Introduce the students in collaborative and professional software development through TDD (Test driven development) techniques.
* Show the students an example documentation of a software architecture

## 4.6 Incident Management Staff

This group of people controls the Incidence Dashboard. Some of their goals are:

* Solve emergency situations if the Dashboard notifies dangerous property values
* Manage an incident while it is been monitored: changing the state of the incident in it has been processed or not, or adding comments about the incident

# Quality Attributes

We have identified the following quality attributes:

* **Availability**
  + The system must be able to process data 24x7.
* **Modifiability**
  + Easily change some parts of the application: Change the parser of input data
  + Easily change some parts of the application: Add an error reporting feature
  + Easily modify some parts of the application: Add other output files to generate the letters
  + Easily modify some parts of the application: Enable password change by users
  + Easily modify some parts of the application: Enable different formats to be used by the web service
* **Performance**
  + The performance of the data loading system is reasonable
  + Querying information about a user through the web service should be fast
* **Security** 
  + The system should warrant the confidentiality of the users’ data
* **Testability**
  + It must be testable that the users’ data loading process is correct
  + It must be testable that the web service behaves as expected
* **Usability**
  + The data loading system must be easy to use by System administrator users which are familiar with Unix-like tools.
* **Interoperability**
  + This system will be used by the Participation System which will leverage on it for user management. The Agents web service must be used by an automated process that can query the status of a user.
* **Simplicity**
  + The two modules should be simple and easy to develop
* **Deployability**
  + The system should be easily deployable, especially in a cloud based server

## List of Quality Attributes

The list of quality attribute is the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Description** | | **Type of Attribute** | **Module** |
| **AT001** | The system must be able to process data 24x7 | Availability | Agents |
| **AT002** | Easily modify some parts of the application: Change the parser of input data | Modifiability | Loader |
| **AT003** | Easily modify some parts of the application: Add an error reporting feature | Modifiability | Loader |
| **AT004** | Easily modify some parts of the application: Add other output files to generate the letters | Modifiability | Loader |
| **AT005** | Easily modify some parts of the application: Enable password change by users | Modifiability | Agents |
| **AT006** | Easily modify some parts of the application: Enable different formats to be used by the web service | Modifiability | Agents |
| **AT007** | The performance of the data loading system is reasonable (not too slow, but not critical) | Performance | Loader |
| **AT008** | The system should warrant the confidentiality of the agents data | Security | Loader and Agents |
| **AT009** | It must be testable that the web service behaves as expected | Testability | Agents |
| **AT010** | It must be testable that the user loading process is correct | Testability | Loader |
| **AT011** | The data loading system must be easy to use by system administrator users which are familiar with Unix-like tools. | Usability | Loader |
| **AT012** | The querying web service must be used by automated processes that can query the status of the system. | Interoperability | Agents |
| **AT013** | The system must be simple and easy to develop | Simplicity | Loader and Agents  Loader and Agents |
| **AT014** | The system should be easily deployable | Deployability | Agents |
| **AT015** | The system keeps constant track of the incidents generated. | Availability | Incident dashboard |
| **AT016** | The performance of the filter by agent functionality is reasonable | Performance | Incident dashboard |
| **AT017** | Provide an interface with detailed information of an incident | Usability | Incident dashboard |
| **AT018** | Provide a way to interact with the incidents and manage them from the dashboard | Usability | Incident dashboard  Incident manager |
| **AT019** | Incidents can be submitted with personalised fields with the form “property/value” | Usability | Incident manager |
| **AT020** | Agent’s configuration such as sensors should be modifiable to emit information in the required interval | Modifiability | Incident manager |
| **AT021** | Information related to the incidents can be modified by the system administrator | Usability | Incident manager |
| **AT022** | Ability to follow the evolution of submitted incidents | Usability | Incident manager |

**Table 2. List of quality attributes and their types**

## Quality Attributes and stakeholders

The following table shows which attribute qualities are interesting for which stakeholder:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attributes**  **vs**  **Stakeholders** | **ST-01** | **ST-02** | **ST-03** | **ST-04** | **ST-05** | **ST-06** |
| **AT001** | X |  | X | X | X | X |
| **AT002** | X | X |  |  | X |  |
| **AT003** | X | X |  |  | X |  |
| **AT004** | X | X |  |  | X | X |
| **AT005** | X |  | X |  | X |  |
| **AT006** | X |  | X | X | X |  |
| **AT007** | X | X |  |  | X |  |
| **AT008** | X | X |  |  | X |  |
| **AT009** | X | X |  |  | X |  |
| **AT010** | X |  | X | X | X |  |
| **AT011** | X | X |  |  | X |  |
| **AT012** | X |  |  | X | X |  |
| **AT013** | X |  |  | X | X |  |
| **AT014** | X | X |  |  | X |  |
| **AT015** | X |  |  |  | X | X |
| **AT016** | X |  |  |  | X | X |
| **AT017** | X |  |  |  | X | X |
| **AT018** | X |  |  |  | X | X |
| **AT019** | X |  | X |  | X |  |
| **AT020** | X | X |  |  | X |  |
| **AT021** | X | X |  |  | X |  |
| **AT022** | X |  |  | X | X |  |

**Table 3. List of stakeholders: interests vs quality attributes**

# Architecture Constraints

## Technical constraints

We have detected the following set of technical constraints in the project:

|  |  |  |
| --- | --- | --- |
| **Code** | **Constraint** | **Background/Motivation** |
| **TC001** | Both systems will be implemented in Java | The developer team (ST001) has knowledge of Java |
| **TC002** | The data will be stored in a relational database. | The developer team (ST001) has knowledge of relational databases and there are a lot of libraries to work with relational databases from Java |
| **TC003** | The web service will be based on REST using JSON format | The REST style of web services using JSON is very popular and easy to implement nowadays. |
| **TC004** | The input data format to load data is Excel | Excel is a popular format for data exchange and there are several libraries to process Excel files |
| **TC005** | The output data of the loader module will be a set of text files | In order to facilitate the implementation, text files are the easier format to generate. However, the developer team can optionally implement other generators. |
| **TC007** | Automated testing | The tests should be run automatically and a continuous integration system must be used |
| **TC008** | The web service will be implemented using the Spring Boot web framework | Spring Boot web framework leverages on Spring, which is a well-known framework very popular in Industry. It contains lots of examples and help info that can help students to learn to use it. |
| **TC009** | The interaction between the Incidents Manager and Incidents Dashboard modules will be made using Apache Kafka | Apache Kafka is a distributed streaming platform that allows us to publish and subscribe to streams of messages in real-time. |

**Table 4. Technical constraints**

## Organizational Constraints

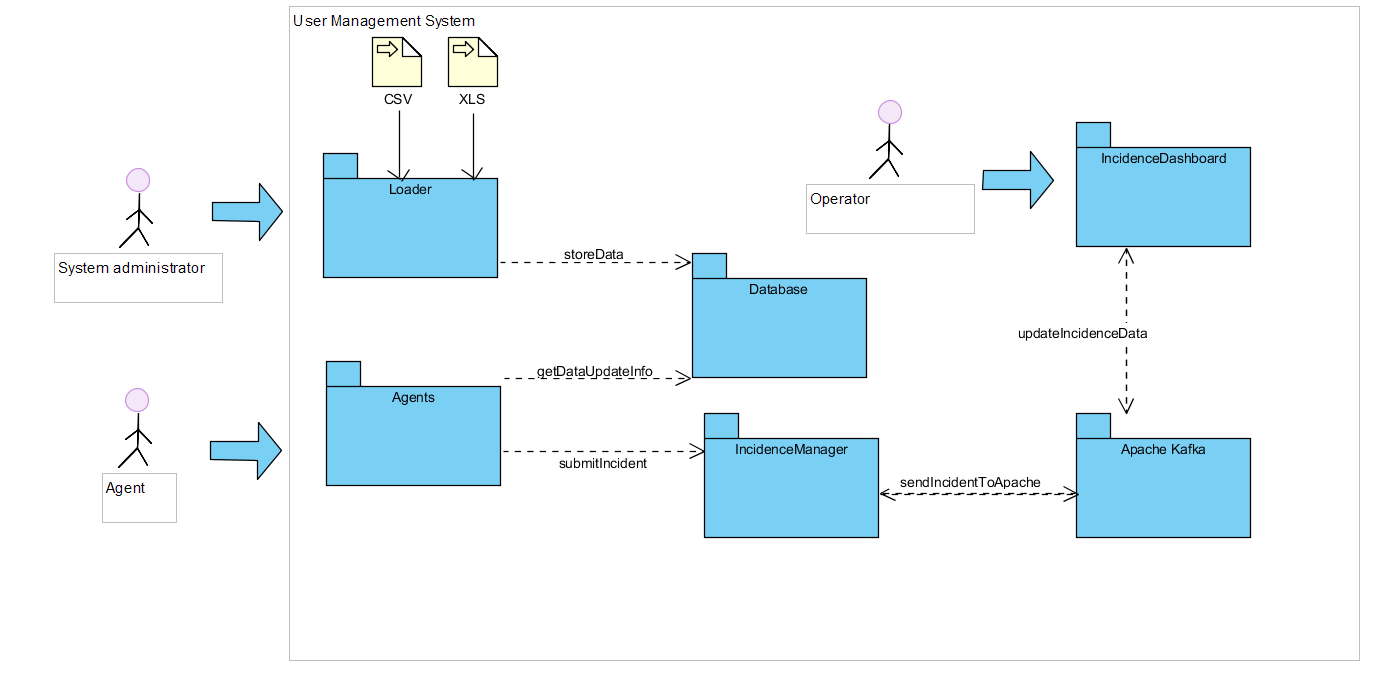
|  |  |  |
| --- | --- | --- |
| **Code** | **Constraint** | **Background/Motivation** |
| **OC001** | Each system will be implemented by a small team of student developers. | The size of the teams will be between 3 or 4 students. The goal is that students learn to work collaboratively by developing a simple project |
| **OC002** | The structure of the database will be shared by both teams. | Although the projects are designed to enable independent development by each team. The database acts as a glue between both systems so its structure must be shared by both teams |
| **OC003** | The source code will be available as a github repository | Github offers a very powerful project management tool for this kind of projects. |

**Table 5. Organizational constraints**

# System scope and context

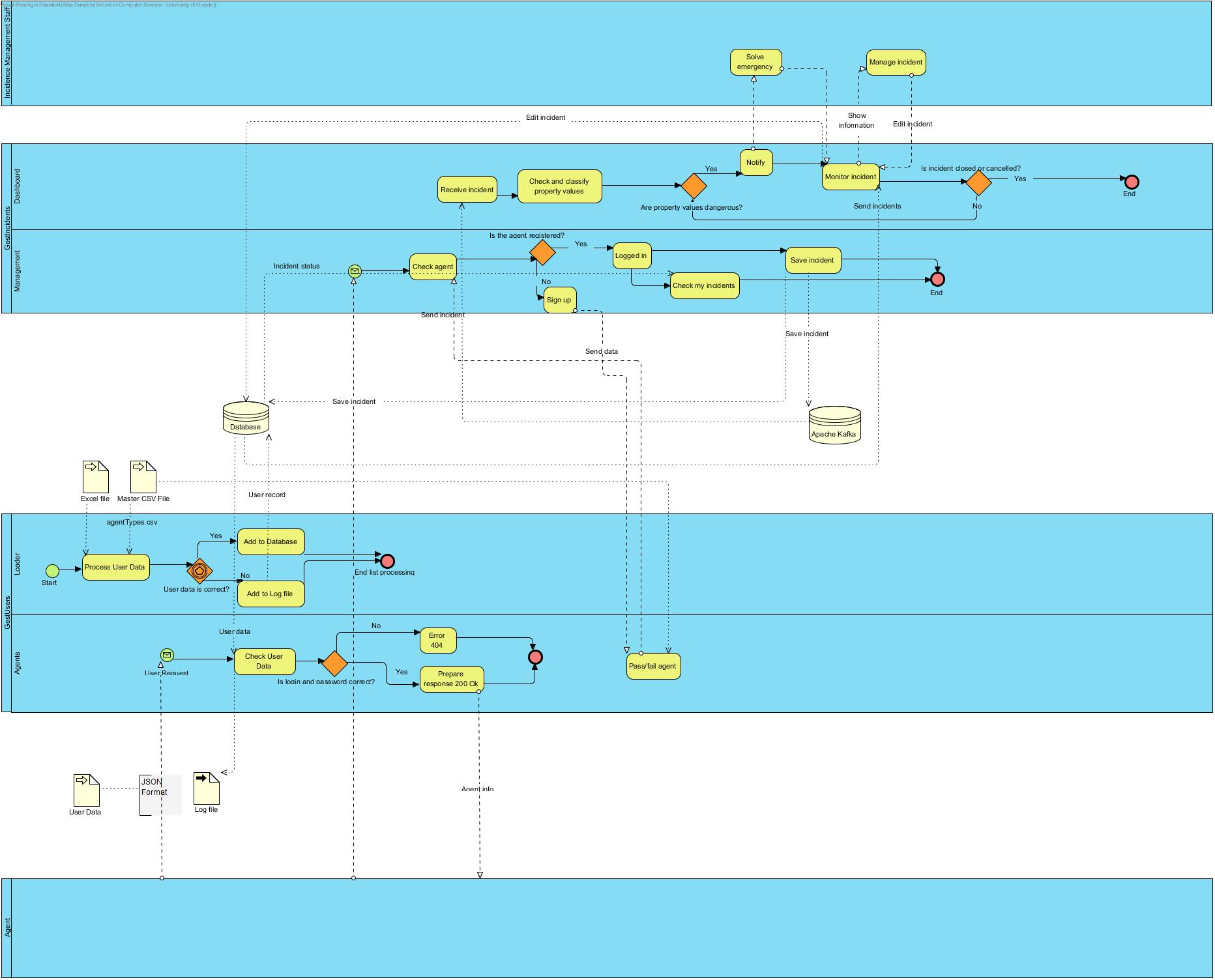
The system is decomposed in two modules:

* Loader: This module will be responsible to convert data from Excel files and load it into the database. The system will be invoked by a system administrator.
* Agents: This module will check if users can participate obtaining information from the database.



**Figure 1. Business Context**

The following figure contains a BPMN diagram showing the whole process of both sub-systems.



**Figure 2. BPMN Diagram**

* Incident Management (InciManagement): This module will process the incidents that are delivered by the different agents. These agents must be registered in the database to be allowed to submit incidents, the incident management will request information to the Agent’s module to verify the existence of the agent submitting an incident.
* Incident Dashboard (InciDashboard): The incidence dashboard will be used by the incident management staff, and can visualize and manage the incidents that appear in the system.

# Quality Scenarios

The table below contains the quality scenarios that have been identified:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Scenario | Source Stimulus | Stimulus | Environment | Artifact | Response | Measure | Affected  Attribute Quality |
| 1 | Participation System | Ask information about a user | Runtime | Agents | Participation System obtains the required information in less than 15seg at any time in the day | The required information is obtained | **AT001** |
| 2 | Student developer | A new parser is introduced | Development | Parser | Change is successfully introduced | The system is compiled and passes all the tests without errors | **AT002** |
| 3 | Student developer | A new option is implemented for the report file | Development | ReportWriter, DBUpdate and Parser | The option is implemented with minimal changes that affect only the report writer module | Less than one day of work | **AT003** |
| 4 | Student developer | A new output format is added | Development | Agents and DBManagement | The new output format is included with minimal changes to existing code. | Less than one day of work | **AT004** |
| 5 | Student developer | The option to change user's password is introduced | Development | Agents and DBManagement | The password of a user is successfully changes | Less than one day of work | **AT005** |
| 6 | Student developer | A new format is added to the web service | Development | Agents | The new format is implemented | Less than 2 days of work | **AT006** |
| 7 | System administrator | Load an Excel file into the System (DB) | Runtime | Parser, DBUpdate and ReportWriter | Loading an excel file without errors is done in a reasonable time. | < 1 second for each 10 Agent | **AT007** |
| 8 | Student developer | Load an Excel file into the system (DB) | Development/  Runtime | Parser, DBUpdate and ReportWriter (Optional) | Loading data should be done in a safe way | It is not possible to get access to the users’ personal data except by the system administrator who cannot get access to the password. | **AT008** |
| 9 | Agents | Get access to the application | Runtime | Agents | A user can get access to his data but not to other user's data | Access to data is enabled only if the pair user name/password is correct | **AT009** |
| 10 | System administrator | Loads an excel file into the DB | Runtime | Parser, DBUpdate and ReportWriter | The loading process is made in a reliable way and it is possible to check that the data has been loaded | There are no errors in the database, no repeated record, and no agent has less information than expected | **AT010** |
| 11 | System administrator | Loads an excel file into the DB | Runtime | Parser, DBUpdate and ReportWriter | The loading process behaves in a usual way and the options available to run the system are easy to understand | The system shows help options if the user asks for them. The error messages and other information can be understood by technical people | **AT011** |
| 12 | User Participation System | Access to the web service | Runtime | Agents | The participation System requests information about a user by passing a combination of user name and password | A 200 OK response is sent with the correct format if the combination is OK or a failure information is returned | **AT012** |
| 13 | Student developer | Develops the system | Development | Agents  Loader | The student developers can implement the system | The system can be implemented and testes in 2/3 weeks by third year undergraduate students. | **AT013** |
| 14 | System administrator | Deploys the system | Deployment | Loader, Agents | The system is deployed in a production environment | The system can be deployed by a system administrator in less than an hour. | **AT014** |
| 15 | Incidents Dashboard system | Shows information about the incidents | Runtime | Incidents Dashboard | The system keeps constant track of the incidents generated in the system. | The system shows any incident generated in the system in less than a minute. | **AT015** |
| 16 | Student developer | Add filters and configure the dashboard to make easier to access to important data. | Development | Incidents Dashboard | The dashboard shows us the most important properties all the time, marks the dangerous ones, let us filter by agent. | The system show us all the incidents produced by an agent when we introduce its name in less than a minute and shows in red the dangerous ones. | **AT016** |
| 17 | Student developer | Provide an interface with detailed information of an incident | Development | Incidents Dashboard | For each incident, we have the possibility of deploying an interface with more detailed information. | The system provides a map placing the incident in it, the properties that are altered by the incident and all the actions related to it when the operator clicks on it. | **AT017** |
| 18 | Student developer | Provide a way to interact with the incidents and manage them from the dashboard | Development | Incidents Dashboard, Incidents Manager | From the dashboard, the operator can access to the incidents manager. | The manager will receive the orders given from the dashboard and will follow them, and the dashboard will update that changes in less than a minute. | **AT018** |
| 19 | Student developer | Incidents can be submitted with personalised fields with the form “property/value” | Development | Incident manager | For the creation of each incident there will be the option to add personalised fields | The user submitting an incident will be able to add new fields to an incident. | **AT019** |
| 20 | Student developer | Agent’s configuration such as sensors should be modifiable to emit information in the required interval | Runtime | Incident manager | Agents such as sensors must have a modifiable state for example to change their interval period | The system administrator must be able to change the configuration of an agent for example sensor at runtime | **AT020** |
| 21 | Student developer | Information related to the incidents can be modified by the system administrator | Runtime | Incident Manager | For each incident we have the possibility to deploy an interface to modify its details | The system administrator or the user in the system in charge of updating incidents should have an interface to change all information related to a submitted incident. | **AT021** |
| 22 | Agent participation system | Ability to follow the evolution of submitted incidents | Runtime | Incident manager | A table displaying main information about the incidents submitted by each agent | < 5 seconds to deploy the information to the interface | **AT022** |

**Table 6. List of quality scenarios**

# Views

In the following paragraphs the identified the views that will be documented following the learning guide instructions.

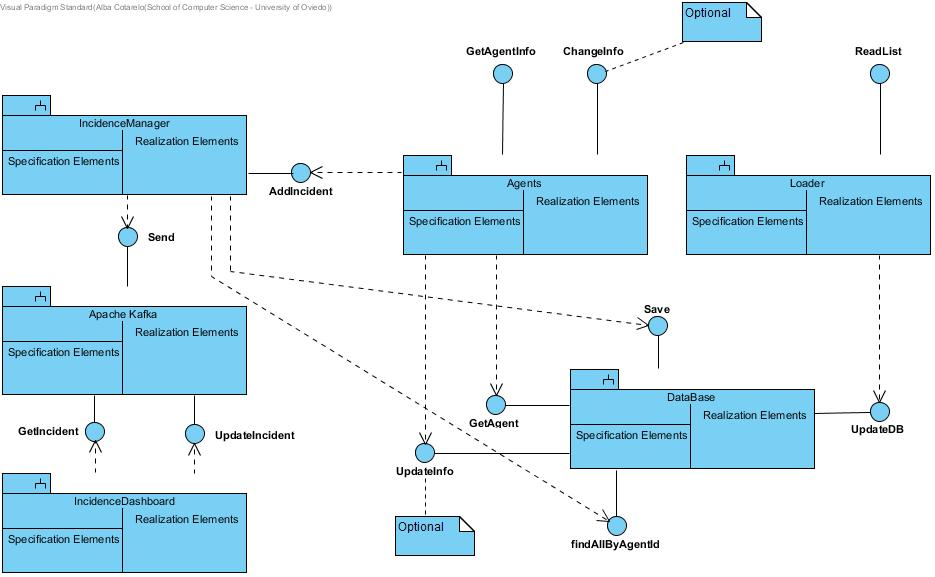
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| View | Stakeholders | Quality Attributes | Scenarios | Organizational constraints |
| Context | ST-01, ST-02, ST-03, ST-04, ST-05 | AT011, AT013, AT14 | 11, 13, 14 | OC002 |
| Loader | ST-01, ST-02, ST-04, ST-05 | AT002, AT003, AT004, AT007, AT008 y AT010, AT011, AT013, AT014 | 2, 3, 4, 7, 8, 10, 11, 13, 14 | OC001, OC003 |
| Agents | ST-01, ST-03, ST-04, ST-05 | AT001, AT005, AT006, AT008, AT009, AT012, AT013, AT014 | 1, 5, 6, 8, 9, 12, 13, 14 | OC001, OC003 |
| Incidents manager | ST-01, ST-02, ST-03, ST-04, ST-05 | AT019, AT020, AT021, AT022 | 19, 20, 21, 22 | OC001, OC003 |
| Incidents dashboard | ST-01, ST-02, ST-03, ST-04, ST-05, ST-06 | AT015, AT016, AT017, AT018, | 15,16,17,18 | OC001, OC003 |

In the catalogues and views we have described both the mandatory and some optional elements. The students can ignore those optional elements that they are not going to implement.

## Context

The System view is divided in two main sub-systems.

### Main overview



**Figura 3. Context view**

### Elements Catalogue

#### Elements

|  |  |
| --- | --- |
| Element | Properties |
| Loader | It introduces users data in the system. It reads an Excel file with data, generates passwords, personal letters and reports any errors. |
| Agents | This is the module used by users to check that their information is available in the system. They can optionally change some of their personal information and their password. |
| DataBase | This module encapsulates database access. |

#### Relationships

Agents data are introduced in the system through the interface *ReadList* from module *Loader*. For each user, a password is generated as well as a personalized letter with information about the user.

That interface sends the data to the database through the interface *UpdateDB* from the DataBase module.

The *Agents* module allows an external system to check the information about a user through the web service *GetAgentInfo*. In order to check the information, *Agents t*asks data to the *DataBase* module through the *GetAgent* interface.

Optionally, it is possible to implement the interface *ChangePassword* that will allow a user to change her password. In order to do that, the *Agents* module requests the *DataBase* to change the password through the *UpdatePasswd* interface.

#### Interfaces /Ports

##### Loader

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| ReadList | Interface | Command line invocation | This interface will be invoked from the main application as a console program |

##### Agents

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interface | Tipo | Tecnología | | Propiedades | |
| GetAgentInfo | | Interface | Web Service | | This interface will be invoked through an HTTP request |

##### DataBase

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Tipo | Tecnología | Propiedades |
| GetAgent | Interface | Method invocation | Returns data from agents |
| UpdateDB | Interface | Method invocation | Inserts into the database data about an agent included its password |
| UpdatePasswd | Interface | Method invocation | Updates the password of a user in the database |

#### Behaviour

##### Loader

##### Loader

See 9.2.2.3.4.

It can also do the following options:

* (**Optional**) the subsystem that generates the letters could implement the Adapter pattern which would enable to generate the letters in different formants in the future (Word, ODT, PDF, RTF, etc.).
* (**Optional**) If the file contains errors, those errors should be detected and a report should be generated for its later treatment
* (**Optional**) The parser of input data should be configurable using an adapter pattern to allow input data in different formats (Excel, TXT, etc.).

##### Agents

##### Agents

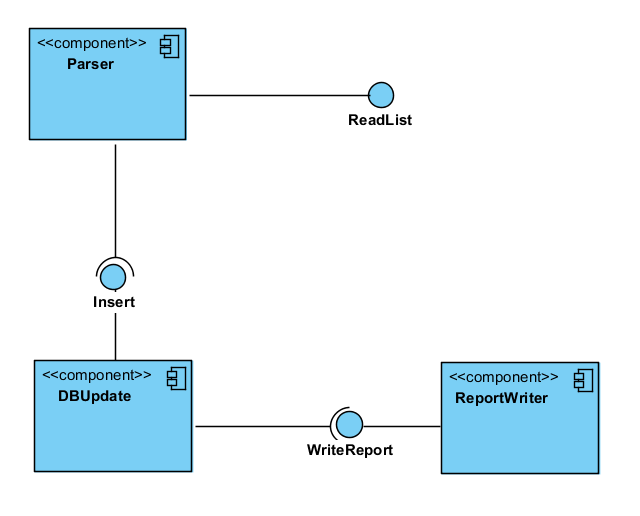
It allows users to get access into the system to check if they can participate, using the information that they received in the letter. The users may not get access directly by a web browser, but through an external participation system that invokes the Agents module as a web service.

##### DataBase

All the operations done in this module will be integrated in a *Facade pattern* which will contain the operations that offer access to the database. It encapsulates all the operations that affect the database.

## Users List

### Main overview



**Figura 4. Agents list view**

### Catalogue of Elements

#### Elements

|  |  |
| --- | --- |
| Element | Properties |
| Parser | Reads data from the Excel file and transforms them into an in-memory object container that can be later iterated to insert the data in the database.  It will also generate the *password* of the agent as well as the personal letter.  During the design and implementation this component can be divided into the sub-components needed to separate these services following the quality attributes AT002, AT003, AT004 and AT007. |
| DBUpdate | Encapsulates all the database operations using interfaces to allow the database access to be separated from some specific database implementations. |
| ReportWriter | It receives the pieces of data that were not possible to insert into the database as well as the reasons and writes a report containing all that information in a human-readable way |

#### Relationships

The *Parser* component receives the input file in Excel format and reads and converts the information about the different users. It generates a new password for each user and adds the information to the database using the *DBUpdate* component.

(**Optional**) If there are any errors during the loading phase (duplicated DNIs, empty DNI fields, etc.) or if the database component returns an error, this information will be notified to the Reportwriter component through the *WriteReport* interface.

#### Interfaces/ Ports

##### Parser

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| ReadList | Interface | Method invocation | Read the Excel file with the users data. |
| Rlist | Port |  | Creates the needed subcomponents of the parser to process the input file. |
| Insert | Interface (Required) | Method invocation | It calls a method in the *DBUpdate* component to insert the information in the database. |
| InserR | Port |  | Verifies the data and creates the object to send to the *DBUpdate* component. |

##### DBUpdate

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| Insert | Interface | Method invocation | Receives and object with the information to insert in the database. |
| InsertP | Port |  | Verifies input data and generates and error if there is a lack of some mandatory attribute. |
| WriteReport | Interface (Required) | Method invocation | Calls a method from the *ReportWriter* component to write a new item in the report file. |
| WreportR | Port |  | Verifies the data to write |

##### ReportWriter

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| WriteReport | Interface | Method Invocation | Receives the data to write in the report file. |
| WreportP | Port |  | Adds data at the corresponding date and time. |

##### Parser

Introduces the user’s data in the system obtained from Excel files that contain a row for each user. Each row (except the first one that contains the headings) contains the following columns:

* Name (string)
* Location (geographical coordinates indicating latitude and longitude)
* Email (string that follows the email format conventions)
* ID(in the case of people or entities it can be the National ID, it will be the user name)
* Kind (integer number that represents the Agent’s kind)

Invocation will be done through a batch program executed in the command line by the system administrator. During the import process a password will be generated so the combination of email/password enable a user to enter the system and participate in the system, and receive information about the polling station code where the user can participate.

This component will also generate personal emails/letters communicating the user that he has been added to the system with a username (his email) and a password.

##### DBUpdate

It updates the database. See 9.1.2.4.5.

##### ReportWriter

(**Optional**) It stores in a text file information about the errors that were produced by the conversion process. The basic information to store is:

* Date
* Time
* Original Excel file
* Error information (with all the needed information)

### Context Diagram

See 9.1.

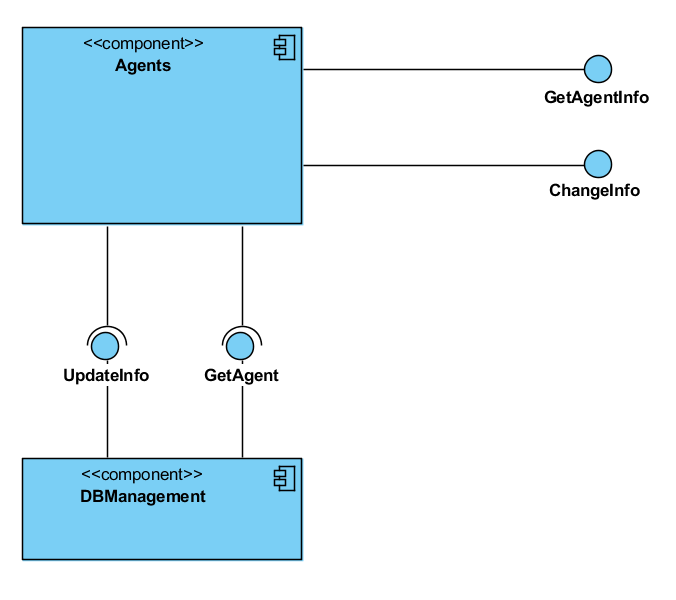
### Rationale

The main design decisions of this sub-system are:

|  |  |  |
| --- | --- | --- |
| Scenario | Quality attributes | Justification |
| 2 | AT002 | Access to the parser using an Adapter pattern facilitates to change the implementation without affecting other parts of the application. |
| 3 | AT003 | Defining an interface and an object for error reporting allows to add this functionality later. |
| 5 | AT005 | Using a relational database will improve the performance of accessing information about users. |
| 6 | AT006 | Using a relational database that offer security aspects can improve the security of the system. Sending the login name and password by regular mail avoids that the information can be accessed electronically. |
| 8 | AT008 | Using a standard database which can be queried using SQL can allow the students to verify that the data has been correctly loaded. |
| 10 | AT010 | The use of a batch application that can be executed manually or configured for its automatic execution is a common practice for system administrators. |
| 14 | AT014 | A batch application can be directly executed without any special needs for deployment |

## Agents

### Main overview



**Figure 5. Agents View**

### Catalogue of elements

#### Elements

|  |  |
| --- | --- |
| Element | Properties |
| Agents | It offers two web services: *GetAgentInfo*, which allows to obtain information about a user and (Optional) *ChangePassword* that allows to change the password of a user. |
| DBManagement | It offers two interfaces: GetAgent, that returns the data of an agent from the database and (Optional) *UndateInfo*, to update a password change in the database. |

#### Relationships

The AgentParticipation System invokes *Agents* using a web service call which is processed by *GetAgentInfo* (sending *email/password*) and it gets access to the DBManagement system using the interface *GetAgent*. If the email/password are correct the data is returned as a JSON response.

(**Optional**) The user can invoke *Agents*through a web browser to change his password invoking *ChangePassword* and sending the parameters *email/password/newPasswod*. It will invoke the interface *UndateInfo* to modify the password using the *DBManagement* component.

#### Interfaces / Ports

##### Agents

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| GetAgentInfo | Interface | Web service | Allows to get access to an agent data through the email/password combination |
| GetPIP | Port |  | Validates a user before asking the data. |
| ChangePassword | Interface | Web service | Allows to change a password using the combination: *email/password/newPasswod*. |
| ChangeInfo | Port |  | Validates a user before asking to change his password. |
| ChangeIP | Port |  | Validates a user before asking to change the password |
| UndateInfo | Interface (Required) | Method invocation | Asks a password change for a user. |
| UInfoR | Port |  |  |
| GetAgent | Interface (Requerida) | Method invocation | Asks information for the user |
| GetPR | Port |  |  |

##### DBManagement

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Tipo | Tecnología | Propiedades |
| UndateInfo | Interface | Method invocation | Handles the password change of a user. |
| UInfoP | Port |  |  |
| GetAgent | Interface | Method invocation | Handles the information request for the user. |
| GetPP | Port |  |  |

#### Behaviour

##### Agents

It implements a REST web service to handle requests of information about users. The POST HTTP request will be done to the following address:

<WebServiceURI>/user

where <WebServiceURI> represents the URI where the web service has been deployed. The POST request contains JSON data with the following structure:

{"login": user, "password": password, "kind": agent's kind}

In case that the (email, password) combination are available in the database the response will be 200 OK with the a JSON body of the form:

{ "name": Name,

"location": Coordinates (optional),

"email": Email,

"id": identifier,

"kind": agent's kind,

"kindCode": numeric code that represents the kind

}

In case that the (email, password) is incorrect, the response will be 404 Not found. The "kindCode" field is obtained from the master file in CVS format described in last section.

It is possible to implement some HTML interface so the web service can be used by humans through a web browser.

The web service can be extended to allow users to change their password.

##### DBManagement

This component encapsulates all the database access so it can be easy to change the underlying database system.

### Context Diagram

See 9.1.

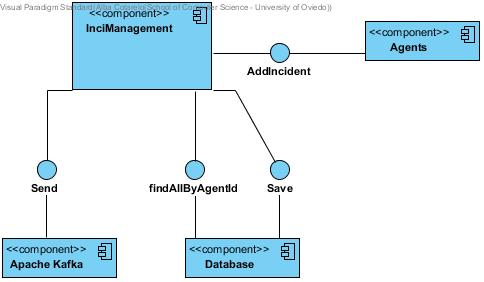
### Rationale

The main design decisions have been:

|  |  |  |
| --- | --- | --- |
| Scenario | Quality Attributes | Justification |
| 1 | AT001 | Using a REST Web Service leverages on HTTP technology and makes it easier to deploy the system in some infrastructure with high availability. |
| 4 | AT005 | The encapsulation of model features that affect the database during development and the use of a MVC framework will facilitate the addition of functionalities like password change. |
| 6 | AT006 | Using a Web framework like Spring Boot will facilitate the development of common web features like content negotiation |
| 8 | AT008 | Accessing by *email/password* is considered secure enough for this process. Passwords should be stored encrypted. |
| 9 | AT009 | The development of a REST web service based on JSON formats will facilitate the development of tests. The Spring Boot framework contains several tools for unit and integration testing of web applications that can be used. |
| 12 | AT012 | The use of a REST web service enables the automatic access to the system through a software client |
| 13 | AT013 | The web service API defined is simple and contains the minimal functionality. Leveraging on Spring Boot web framework will facilitate the development by the students given that the framework has solutions for all the required functionality |
| 14 | AT014 | The use of Spring Boot framework facilitates deployment. There are several examples that show how to deploy Spring Boot based applications to production servers |

## InciManagement

### 9.4.1 Main Overview



### 9.4.2 Catalogue of Elements

#### 9.4.2.1 Elements

|  |  |
| --- | --- |
| Element | Properties |
| InciManagement | Receives an Incident sent by an Agent(not part of this module). The Incidence Manager is responsible of checking if the incidence fields are correct. If they are not, the incidents won’t be processed and an error will be reported. |
| Apache Kafka | Encapsulates all the database operations using interfaces to allow the database access to be separated from some specific database implementations. |
| Database | Keeps a collection of the submitted incidents. It saves the same incident which are sent to Kafka, it is used as an util component to track the state of the incidents, if modified from the Dashboard. This way an Agent can keep an eye on the incident: InciManager takes that information from this Database. |

#### 

#### 9.4.2.2 Relationships

The InciManagement receives an Incident from an Agent (the Incident could be represented as an Entity). After receiving it, the incident is processed to check if it meets the desired requirements. If it is correct, it is inserted in the Apache Kafka using the Send method. The incident is also saved through Save in the MongoDB Database in order to keep an easy access and been able to edit its state from the IncidentDashboard.

#### 9.4.2.3 Interfaces / Ports

#### 

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| AddIncident | Interface | Method invocation | Allows the agent to submit an incident |
| Send | Interface | Method invocation | Allows the InciManager to send the incidents to Kafka |
| Save | Interface | Method invocation | Allows the InciManager to save an Incident to the Mongo Database. |

#### 9.4.2.4 Behaviour

It is a main component in charge of take care of the process of submitting the incidents, it connects the agents with that process. InciManager allows the agents to submit the incidents and sends them to InciDashboard. It also keeps tracking of the incidents’ state in order to keep updated the information the agents receive about the incidents they have submitted.

### 9.4.3 Context Diagram

See 9.1

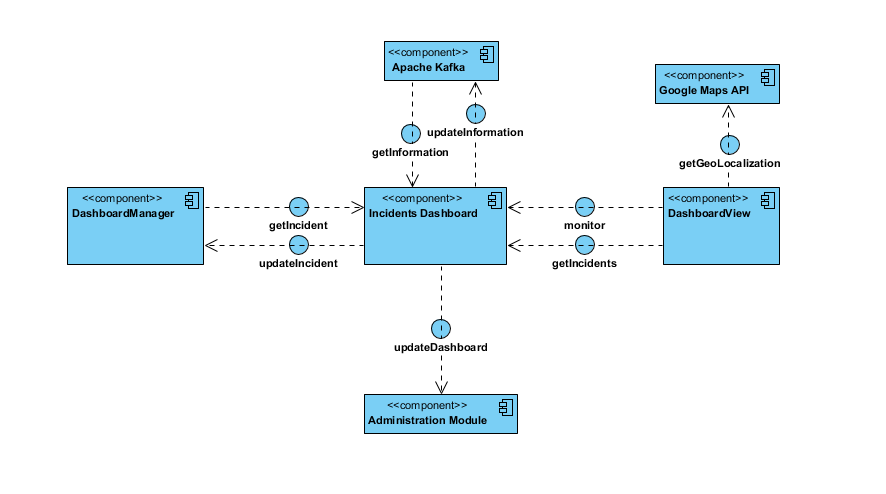
### 9.4.4 Rationale

The main design decisions have been:

|  |  |  |
| --- | --- | --- |
| Scenario | Quality Attributes | Justification |
| 18 | AT018 | InciManager is responsible for receiving orders from the dashboard and interacting with the incidents according to these orders, for the InciDashboard to manage them later. |
| 19 | AT019 | It is compulsory for the incidents’ manager to allow the users adding new fields with information to those incidents they need them. |
| 20 | AT020 | Some agents may be in the need of updating some of their fields at runtime, so the system administrator will be in charge of this operation. |
| 21 | AT021 | Users or administrators in the system must have a visual interface to see the details of each submitted incident, so that its information can be updated in case it is needed. |
| 22 | AT022 | A table interface with all the incidents submitted by one agent must be deployed in a maximum of 5 seconds. |

## Incidents Dashboard

### Main overview



### Catalogue of elements

#### Elements

|  |  |
| --- | --- |
| Element | Properties |
| Incidents Dashboard | The main component, through all the IncidentsDashboard module will pass to get the incidents, monitor them or update them. |
| DashboardManager | It offers two interfaces: *getIncident* to obtain an incident and *updateIncident* to update the information about it. |
| DashboardView | It offers two interfaces: *getIncidents* to obtain all the incidents and *monitor* to monitor some properties about them |
| AdministrationModule | **OPTIONAL:** will work in order to update all the dashboard. |
| Apache Kafka | Module that we will use in order to *GetInformation* from the Manager and *updateInformation* and send it to the manager. |
| Google Maps API | API that offers a geolocalization service to the view to locate an incident |

#### Relationships

The relationships between this components are stated with the interfaces. This module will be connected to the IncidentsManager through Apache Kafka.

#### Interfaces / Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Type | Technology | Properties |
| GetIncident | Interface | Method invocation | Allows to get access to an incident data. |
| GetIncidents | Interface | Method invocation | Allows to get access to all the incidents data. |
| Monitor | Interface | Method invocation | Allows to monitor the important data from the incidents |
| UpdateIncident | Interface | Method invocation | Allows to update the information of an incident and send it to the dashboard |
| UpdateDashboard | Interface | Method invocation | Allows to update whatever you want in the dashboard:filters, incidents to be shown… |
| UpdateInformation | Interface | Method invocation | Allows to update the information of all the incidents sending it to Apache Kafka |
| GetInformation | Interface | Method invocation | Allows to get all the information related to incidents from Apache Kafka |
| GetGeolocalization | Interface | Method invocation | Allows to get a map from the Google Maps API |

#### Behaviour

It is a main component where we will have a list of Incidents, which will be sent, obtained or altered through the different components that we have stated in our diagram (DashboardManager,DashboardView…). It would work as a coordinator, helping all the other components to work without any kind of problems.

Also, it can help by filtering the data we obtain through Apache Kafka.

### Context Diagram

See 9.1.

### Rationale

|  |  |  |
| --- | --- | --- |
| Scenario | Quality Attributes | Justification |
| 15 | **AT015** | We have to show information for all the incidents, which are managed in another module, so the best way is obtaining all of them from Apache Kafka |
| 16 | **AT016** | As we want to obtain all the information of incidents from an agent, we will use our interaction with Apache Kafka and filter the result to obtain it. |
| 17 | **AT017** | We have to allow an operator of the system to get more information about an incident, so we have an specific component that will work on that concept. |

## Deployment View

### Main overview

### Catalogue of elements

#### Elements

#### Relationships

#### Interfaces / Ports

#### Behaviour

### Context Diagram

See …

### Rationale

The main design decisions have been:

|  |  |  |
| --- | --- | --- |
| Scenario | Quality Attributes | Justification |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# References

ANSI/IEEE 1471. (2000). *Recommended Practice for Architectural Description of Software-Intensive Systems.* ANSI/IEEE.

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice, Second Edition.* Boston: Addison Wesley.