Project Title:	**Sensing and predictive treatment of frailty and associated co-morbidities using advanced personalized models and advanced interventions**
Contract No:	690140
Instrument:	Collaborative Project
Call identifier:	H2020-PHC-2014-2015
Topic:	PHC-21-2015: Advancing active and healthy ageing with ICT: Early risk detection and intervention
Start of project:	1 January 2016
Duration:	36 months

Deliverable No: 6.3

First mHealth Integrated version (vers a)

Due date of deliverable:	June 2017
Actual submission date:	**27-06-2017**
Version:	1.0
Date:	23-06-2017

Lead Author:	**Gruppo SIGLA**
Lead partners:	BRAINSTORM, HYPERTECH

Horizon 2020
European Union funding
for Research & Innovation

Change History

Ver.	Date	Status	Author (Beneficiary)	Description
0.1	05-06-2017	Draft	L. Bianconi	First version
0.2	12-06-2017	Draft	F. Podda	Revision
0.3	15-06-2017	Draft	M. Toma	Revision
0.4	22-06-2017	Release	L.Bianconi, F. Podda, M. Toma	Revision
1.0	23-06-2017	Release	L.Bianconi, C. Degano, F. Podda, M. Toma	Release version
1.1	26-06-2017	Draft	E. Montesa	Review
1.1	27-06-2017	Draft	Kosmas Petridis, Stefanos Makris	Review
1.2	27-06-2017	Release	L.Bianconi, C. Degano, F. Podda, M. Toma	Release version

EXECUTIVE SUMMARY

The aim of this report is to document the state of the art and the workplan of the FrailSafe mHealth Integrated system.

TABLE OF CONTENTS

List of Figures

List of Tables

1. INTRODUCTION

One of the purposes of the FrailSafe project is to implement a system, also called *platform*, able to detect and reduce the progress of the older people towards the frailty status through the help of ICT technologies such as softwares, sensors, games, wearables, devices, etc.

Such a system is developed like a whole modular ecosystem infrastructure, composed by several different and separate parts that need to work together, harmoniously, in order to reach the FrailSafe target functionalities, described within the *Description of ActionD* as *"Technological Objectives"*. To achieve this fundamental goal, a relevant integration activity must be performed.

As described in detail in deliverable D1.3, where the architecture is examined in details, the whole system can be conceived as a set of blocks. Each one of these blocks implements a set of features, grouped for scope. Every single module exposes its data and functionalities by means of a set of web services, that can be consumed by the other ones. The described architectural approach means the implemented system is a service-oriented one. Having designed the system in accordance to this methodology suggests that every single process or use case is represented by a chain of services, therefore the objective of the integration task is to allow an effective and secure interconnection between the different parts of the system, with the aim of providing the right services to the end-users.

For this reason, the clear definition of the system architecture represents a fundamental specification of the whole project and the integration of each module and the communications between them is crucial for the good results of the system (see deliverable D1.3 and its forthcoming revision, deliverable D1.4).

The aim of this document is to describe the evolution of the integrated platform, from one step to the following one, explaining the adopted methodology for the integration task and the details of the work done – i.e. the cloud infrastructure setup, the workplan, the tests, etc.

The document is organized as follows:

- *Chapter 2* describes the methodology adopted for the integration, with details about the Service Oriented Architecture (SOA), the expected plan for the releases of the integrated platform and an overall overview of the integration approach;

- *Chapter 3* illustrates the cloud configuration with the description of the networks organization, how they interact and communicate, the allocation of infrastructural resources;

- *Chapter 4* shows the state of the integration and the plans for the forthcoming releases;

- *Chapter 5* summarizes the test methodology and reports which tests have been executed to check the features of each module.

1.1 Reference documents

The following documents are used as reference for the writing of D6.3:

- D1.2 User requirements, use cases, UCD methodology and final protocols of evaluation studies;
- D1.3 FrailSafe technical specifications and end-to-end architecture (vers.a).

1.2 Acronyms and definitions

Acronyms	Definitions
API	Application Programming Interface
AWS	Amazon Web Services
eCRF	electronic Case Report Form
NAT	Network Address Translation
OOD	Object Oriented development
SIT	System Integration Testing
SOA	Service -Oriented Architecture
UCD	User Centred Design
VPC	Virtual Private Cloud
VPN	Virtual Private Network
WWBS	Wearable WBAN System

2. PLATFORM INTEGRATION METHODOLOGY

As said before, the integration is a crucial task for the success of the whole project and this is the consequence of choosing the right methodology in relation to the system architecture. The next sections explain the type of the system architecture and the integration methodology adopted.

2.1 Service Oriented Architecture

As stated in Chapter 4 of deliverable D1.3, titled *System Deployment and Integration*, the FrailSafe system is designed according to a so-called *Service Oriented* architectural pattern.

The architectural concepts referred by a Service Oriented Architecture – also known as SOA - aims at enabling agile business processes via open, standard-based interoperability. The architectural approach aims at creating systems implemented based on autonomous services.

SOA makes easier to start with early integrations of systems, basically avoiding “Big Bang” integration practices, with the positive result of having – and making possible to have - an end solution likely to be composed of services developed in different programming languages, hosted on disparate platforms with a variety of security models and business processes that are however interoperable and integrated.

Even if such practice sounds complex, it shares many service design principles with Object Oriented Development (OOD) techniques, in any case familiar to the world of software engineering (i.e. encapsulation, abstraction and clearly defined interfaces).

SOA is essentially a collection of atomic building blocks, that are the *services*. Each service – or group of services providing the same and specific set of functionally coherent features – is a part of the system, or a sub-system. These services are then built to perform granular operations with limited knowledge of the whole system in which they are inserted (i.e. they interact with it through the interfaces they expose receiving inputs and providing outputs).

These services are expected to communicate with each other and can be interacted through well-defined message exchanges. The communication can involve simple data passing or can involve two or more services coordinating some activity. Such a communication among services rely upon standards-based interfaces and messages (e.g. RESTful services and JSON format for messages).

These services are designed and implemented according to the principles of encapsulation, abstraction and defined interfaces. As a consequence, they are robust, their aggregation is easy to change and their internal logics can be modified seamlessy. All that means that SOA is agile, flexible and resilient. In general, this means that the services could be fixed, improved or even replaced by more updated ones - time by time - without any harm for the correct functioning of the whole platform and without changing entire workflows of a system.

The concept behind SOA is to make use of a service only by its published interfaces. The benefit is that any change in the code of the service has not consequence to the consumer of it. If a consistent interface is maintained, the module, which consumes the resource, could address the request to an alternative instance of the same service type without modifying the application or to an alternative service providing the same interfaces. Since different services, the consumers and the providers use the same communication protocol, it is not necessary that they have the same technologies for their implementation and interfaces.

Consequently, this architectural pattern is in close relationship with the integration approach chosen for the project and it affects deeply the design of the system and its integration process.

2.2 Integration methodology

As briefly sketched in deliverable D1.3, chapter 4 *"System Deployment and Integration"*, an integration approach suitable for the characteristics of modularity and interoperability of the designed system, also efficient enough for performing the tasks implied in by the "System Integration", must be adopted. For this reason, it has been chosen to adopt a "Continuous Integration", like approach instead of the old-fashioned "Big Bang" methodology.

Before getting into much details about the adopted methodology, it is worth defining with a few lines what is the role and the work of the "System Integration".

It can be summarized by identifying the two sub-tasks in which the work is mainly divided into:

- *Integration:* that is assembling the parts of a system in a logical, efficient and cost-effective way;
- *Test*: that is comprehensively checking system execution and testing its adherence to the functional requirements under the full-system point of view.

definition, even tough incomplete and not scientific, should be satisfingly schematic and clear to define the responsabilities and the boundaries among whom the "System Integration" task operates.

Since the original writing of the project's proposal, it has been decided to adopt an up-to-date methodological approach for what concerns the collection of the system requirements and use-cases (see usage of User Centered Design in deliverable D1.2). Nevertheless, the decision of keeping the clinical experts of the projects constantly in the processes of design the system architecture and of the single functionalities has been taken. The choice of taking care of the end-users' needs and making them participate also to the design of the sub-systems - e.g. implementation of e-CRF, of games and of Wearable WBAN System (WWBS) - through all their progressive evolutions was taken according to the principles of agility and participatory design. As already said here above, also for the *"System Integration"* was chosen to adopt an iterative - not to use the nowadays abused term "agile" - approach to it, in the form of the so called "Continuous Integration".

As a quick reference, it is useful to give an informal definition of it: *Continuous System Integration* could be defined as an integration practice that consists of starting from an early integrated and simple version of a system that is able to evolve with flexibility, adding time by time newer features, till reaching a final integrated system through frequent and constant progressive integration iterations.

Towards this direction a proper "Integration Rhythm" has been declared already within the Project Proposal: three formal integration iterations have been scheduled officially at M18, M24 and M32 within the workplan of the project. Each official integration step is preceded by a number of individual implementations, deployments and integrations for each module of the system, according to their internal implementations plans, defining a much more frequent number of independent iterations.

For organizing and rationalizing the evolution of the system, from the first version to the final one, different steps have been, or will be, executed:

1. The overall architecture of the system has been reviewed and finalized;

2. based on the modules specifications, the processes covering the main use cases defined for the system (see deliverable D1.2) have been designed and described;

3. a *Functional Integration Matrix* (see Figure 8) has been written to have objective metrics to evaluate the progresses of the system and its submodules, to schedule and track

the progresses of each module integration status and to organize the test tasks and implementations.

Several tests are forseen to be performed automatically on a periodic basis, when possible, in order to identify eventual misfunctioning of the system and, in particular, to have a regression testing infrastructure in order to check whether the newly added features are not having bad impacts on what was already positively working.

As explained, the first step needed for the correct performance of the Continuous Integration is to have an early integrated kernel of the system, or a skeleton, around whom adding all the other modules time by time and for each of them adding features time by time.

Such a skeleton has been implemented in two consecutive iterarions during the months of March, April and May 2017. The first iteration saw, in a progressive consecutive order, the preparation of a first initial cloud resources infrastructure and, subsequently, the deployment and test of the basic security and authentication modules of the system. The second iteration consisted in the extension of the cloud resources infrastructure in order to be ready to host all the parts of the system implemented by the different WPs and in the deployment of the first release of the core data analysis modules.

When such a skeleton has been finally ready, it came the time to start deploying and integrating the first versions of the first available single system's parts.

During a third iteration, along the months of May and June 2017, a refined version of the data analysis modules, the lingtester and the ecrf have been deployed and integrated, at first, within each other. Several tests have been performed, also through the usage and the implementation of automatic test procedures.

After that, through consecutive iterations, all the other modules of the system, expected to be in an alpha version, will be deployed one by one onto the cloud infrastructure.

At each iteration, new tests will be implemented and added.

The details of each of the highlighted iterations are described below in this document, within the following chapters.

Concluding, we can summarize our *"Continuous Integration"*-like approach with a process that tries to perform as many integration iterations as possible. The main steps are:

- Establishing an integration agenda based on three official phases (M18, M24, M32), that is the main "Integration Rhythm";

- Finalization of architecture and system specifications in order to build a "functional integration matrix" in order to define, track and test the progresses for each sub-module of the system;

- Preparing an early integrated infrastructural "skeleton" of the system
    - Cloud infrastructure
    - Basic security software modules;

- Preparing a first integrated "skeleton" of the system
    - Cloud infrastructure to host other sub-modules
    - Deployment and integration of the core data analysis modules;

- Performing frequent iterations to add new modules and features to modules already deployed
    - Deploy and integrate
    - Basic Tests
    - Automated testing.

2.3 Integration Plan

The work of the integration is carefully planned since the very beginning of the project with a specific methodology and a specific timeline, as explained within the previous chapter *Integration methodology*.

The integration is organized in three consecutive evolutive iterations. They are:

1. FrailSafe mHealth Integrated version (vers a)
2. FrailSafe mHealth Integrated version (vers b)
3. FrailSafe mHealth Integrated version (vers c)

followed by a final period of debugging and bugfixing.

Each of the iterations described here above corresponds to a different stage of progressive completeness of the whole platform.

The phase 1, as described in more details here after, will see the release of an integrated basic skeleton of the system able to host and support the deployment of all the other sub-systems. This basic structure is composed by the correct preparation of the infrastructural backbone of the system, as well as the deployment and integration of different basic modules, responsible for the base features of the system – i.e. security, data, communication, etc..

According to the timeline of the deliverables – see Figure 1 -, it means that the percentage of completeness of the system's functionalities is about 30%.

This integration stage has been concluded within the end of Month 18, *June 2017*.

#	Description	Resp	due for	Alpha M18	Beta M25	RC M32
WP3						
D3.2	Preliminary WWBS Prototype	SMARTEX	M15	x		
D3.3	Final WWBS Prototype	SMARTEX	M24		x	
WP4						
D4.1	Offline Analysis of data (vers a)	UoP	M18	x		
D4.2	Offline Analysis of data (vers b)	UoP	M24		x	
D4.3	Online Analysis of data (vers a)	UoP	M18	x		
D4.4	Online Analysis of data (vers a)	UoP	M24		x	
D4.12	LingTester Test Results - Passive offline mode (vers a)	UoP	M18	x		
D4.13	LingTester Test Results - Passive offline mode (vers b)	UoP	M24		x	
D4.14	Signal processing algorithms for extraction frailty related indicators (vers a)	UoP	M12	x		
D4.15	Signal processing algorithms for extraction frailty related indicators (vers b)	UoP	M24		x	
D4.16	FrailSafe Decision Support System (vers a)	UoP	M24		x	
D4.17	FrailSafe Decision Support System (vers b)	UoP	M28			x
WP5						
D5.2	Beta Version of Synthesized AR Game system	BRAINSTORM	M16	x		
D5.3	Final Synthesized AR Game system	BRAINSTORM	M24		x	
D5.4	Personalized Context-aware, Information Visualization (ver a)	CERTH	M24		x	
D5.5	Personalized Context-aware, Information Visualization (ver b)	CERTH	M28			x
WP6						
D6.1	FrailSafe Virtual Community Platform (vers a)	UoP	M28			x
D6.2	FrailSafe Virtual Community Platform (vers b)	UoP	M32			x
D6.3	FrailSafe mHealth Integrated version (ver a)	SIGLA	M18	x		
D6.4	FrailSafe mHealth Integrated version (ver b)	SIGLA	M25		x	
D6.5	FrailSafe mHealth Integrated version (ver c)	SIGLA	M32			x

Figure 1 Deliverables and integration phases

The phase 2 will see the release and integration of the most of the sub-systems – some of them in a final version and some with reduced functionalities. According to the timeline of the deliverables (see Figure 1), it means that the percentage of completeness of the system's functionalities is about 70%.

This integration step will be completed within the end of Month 24, *December 2017*.

The phase 3 coincides with the release and integration of all the sub-systems and all of them in their final and full version. According to the timeline of the deliverables (see Figure 1 it means that the percentage of completeness of the system's functionalities is about 100%.

The deadline of this last phase is the end of Month 32, *August 2018*.

Even if an important number of tests are performed before, it is reasonable to expect that some bugs or minor issues will be discovered and need to be fixed. A period of futher debugging of the final version of the system, as published in Month 32, is expected for the last three months of the Project.

To organize the work of integration in respect of partners' internal development workplans and to have a more precise idea of the timeline of implementation of each module a comprehensive release roadmap has been collected, as depicted in Figure 2 - Sub-systems' release roadmap.

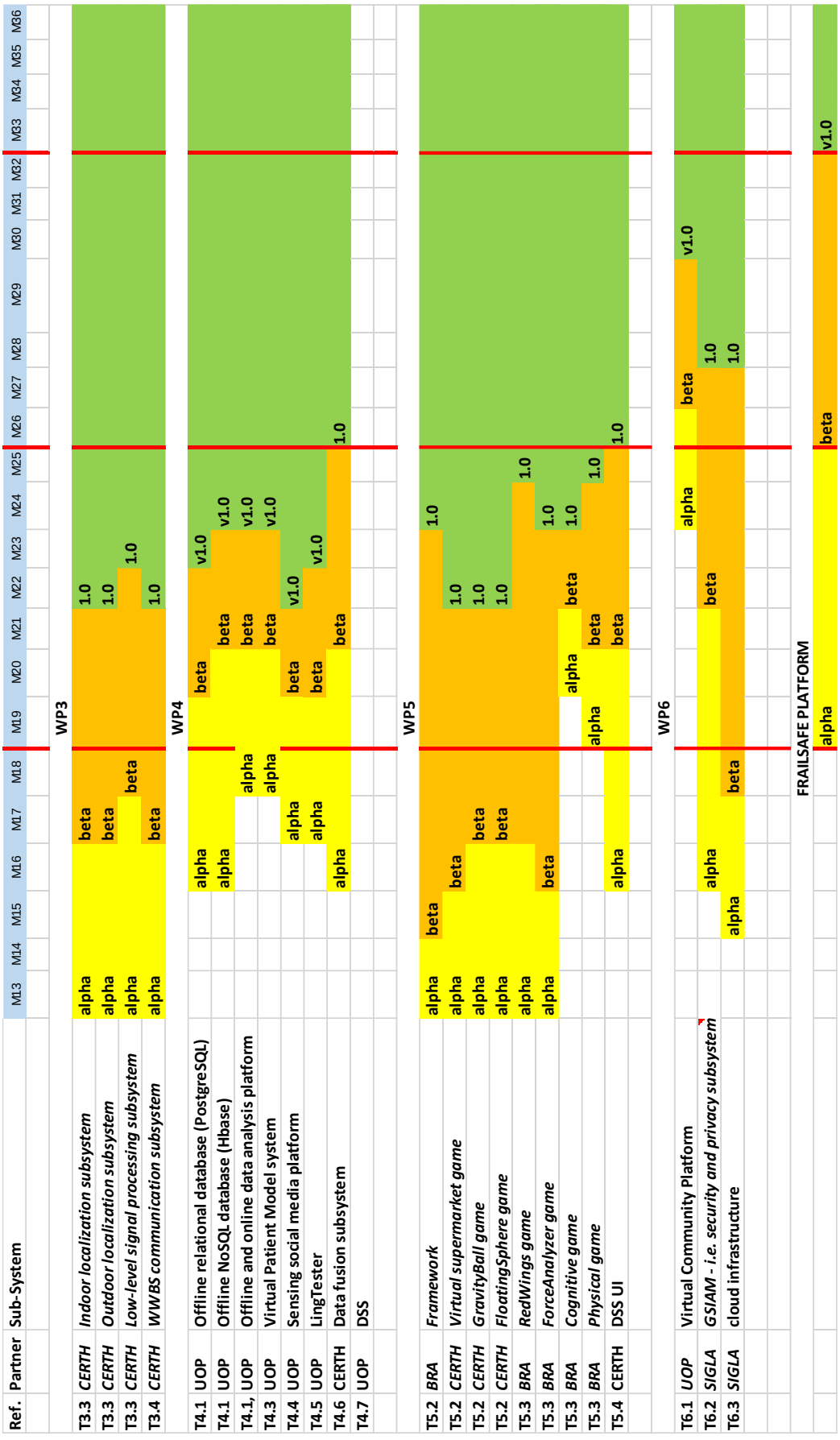

Figure 2 - Sub-systems' release roadmap

3. PLATFORM SETUP

As described in deliverable D1.3, for the implementation of the FrailSafe cloud, AWS has been chosen as service provider. To deploy all the modules defined in the FrailSafe architecture, an initial configuration activity has been carried out at the very beginning of the integration activity to create the infrastructure needed to host the different pieces of the environment. After that, the allocation of the resources has been performed and Project's partners have been allowed to deploy their respective modules. Finally, integration tests have been completed to verify the correct integration of modules inside the FrailSafe cloud. This chapter will describe, in detail, all the afore-mentioned steps.

3.1 Cloud infrastructure setup

The initial infrastructure setup represents an essential stage in order to effectively deploy almost any ICT application. In a project like FrailSafe, these assumptions come to be even more true, since the complexity of the system and the connections among different modules put on the cloud infrastructure a central importance to have an efficient, secure and maintainable ecosystem.

As said before, the configuration of the cloud was taken as the very first action in the integration chain, therefore some macro-issues have been defined in order to start:

- How do we organize different resources inside the cloud?
- How do we guarantee the security inside the cloud?
- How do we grant access to cloud resources?

To answer all these questions, the approach described in deliverable D1.3 was carried out. In detail, we firstly allocated an isolated slice of the AWS cloud, the so-called Virtual Private Cloud[1] (VPC), where then we defined the network infrastructure. The networking is an essential part of a cloud infrastructure, since it defines how resources communicate with each other and if they are reachable from where.

Considering the FrailSafe system architecture, the VPC has been divided in three different sub-networks, namely:

- Public subnet;
- Private subnet;
- Database subnet.

The main goal of this division is to make the system more secure providing a resource segregation that helps to maintain the overall infrastructure consistent. Each subnet has its own properties that define its role inside the FrailSafe cloud:

- The Public subnet has a direct connection to the internet, so that resources that are allocated inside it are directly reachable from the outside, e.g. from end-users;

- The Private subnet has no connection with the internet, hence resources in this subnet are completely isolated from the outside;

[1] https://aws.amazon.com/vpc/

- The database subnet has the same properties of the Private one, and it is dedicated to host all the database instances that are used in the cloud.

Among the single property of each subnet, we have also defined the following rule: each subnet can be reachable from the others, so that resources can talk to each other regardless of the subnet they are deployed.

After this first configuration, we can summarize with the picture below the infrastructure created so far.

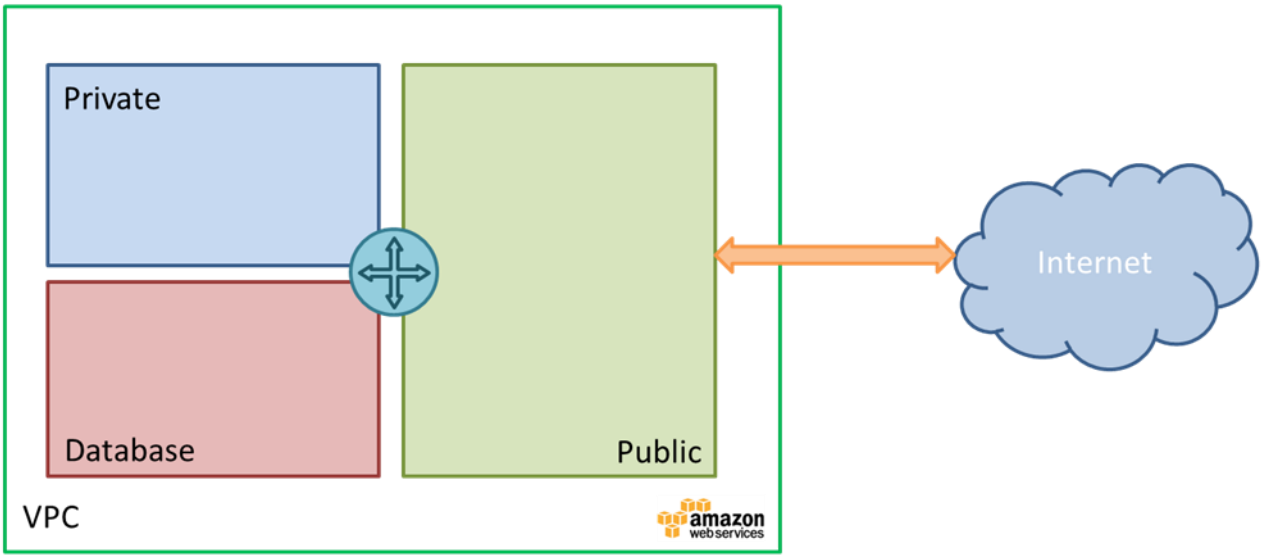

Figure 3 - Cloud infrastructure (1)

With this subnet division, we have created an infrastructure where the only entry-point of the FrailSafe cloud is the Public subnet and the resources that are allocated there, while the other services are isolated. We could say that in this way we ensure the security of the whole infrastructure, exposing only a small part of the cloud and then reduce the points of attack, but this is not sufficient. The main reason is that, even if a module could stay into an isolated part of the cloud, it probably needs to reach the internet for interacting with other services (e.g. a Social Network or a search engine). Furthermore, even if a module exposes services that could be consumed within the cloud, it probably needs to expose only a little part of them to the end-users. In these mentioned cases, the only possible solution is to deploy the modules inside the Public subnet, invalidating most of the security assumptions made before.

To solve the above-mentioned issues, we basically must introduce two components in the cloud infrastructure:

- A system that allows resources in the Private subnet to access the internet;

- A system that allows resources in the Private subnet to expose only some services without the need to expose the entire module.

For the first point, a NAT Gateway[2] has been configured in order to allow resources to initiate connections to the internet. It is worth noticing that, with a NAT Gateway, modules continue to be unreachable from the outside.

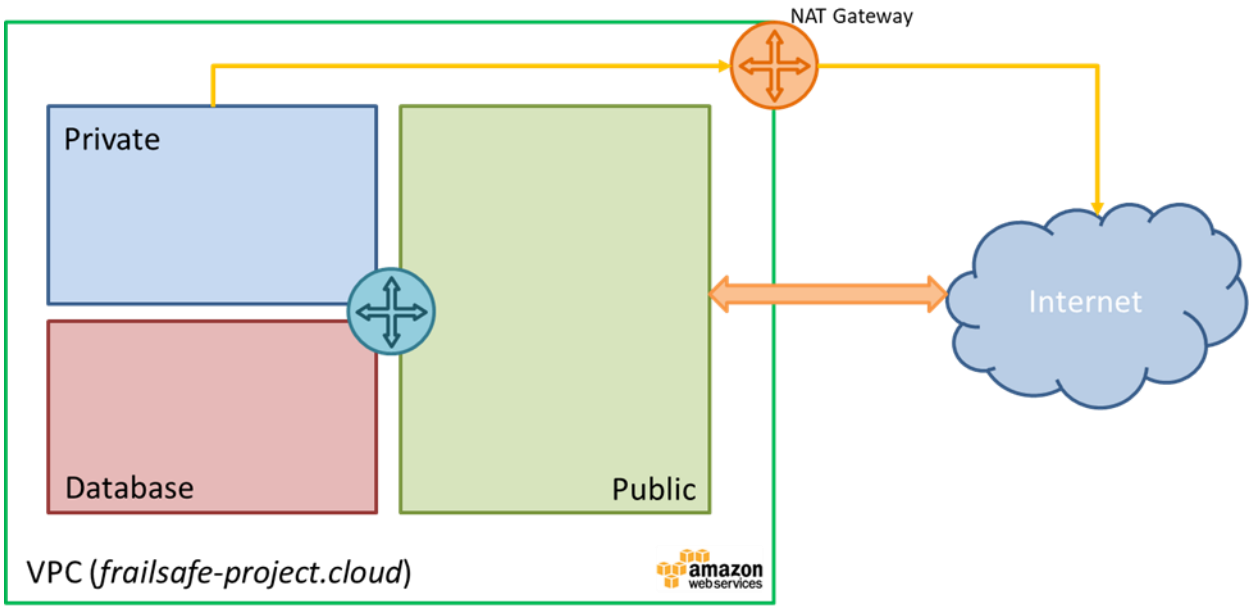

Figure 4 - Cloud Infrastructure with NAT Gateway

For the second point, we have introduced an API Gateway[3] deployed in the Public subnet. API Gateway is a software that expose APIs that are proxied to the appropriate services. API Gateway allows us to expose only the services we map into it, giving us the possibility to deploy them into the private part of the VPC, ensuring that only the necessary services are exposed to end-users. Since API Gateway is deployed in the Public subnet, it also handles the security of the connection between end-users and the target service. The picture below summarizes how the API gateway works inside the FrailSafe VPC.

[2] http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-nat-gateway.html

[3] http://microservices.io/patterns/apigateway.html

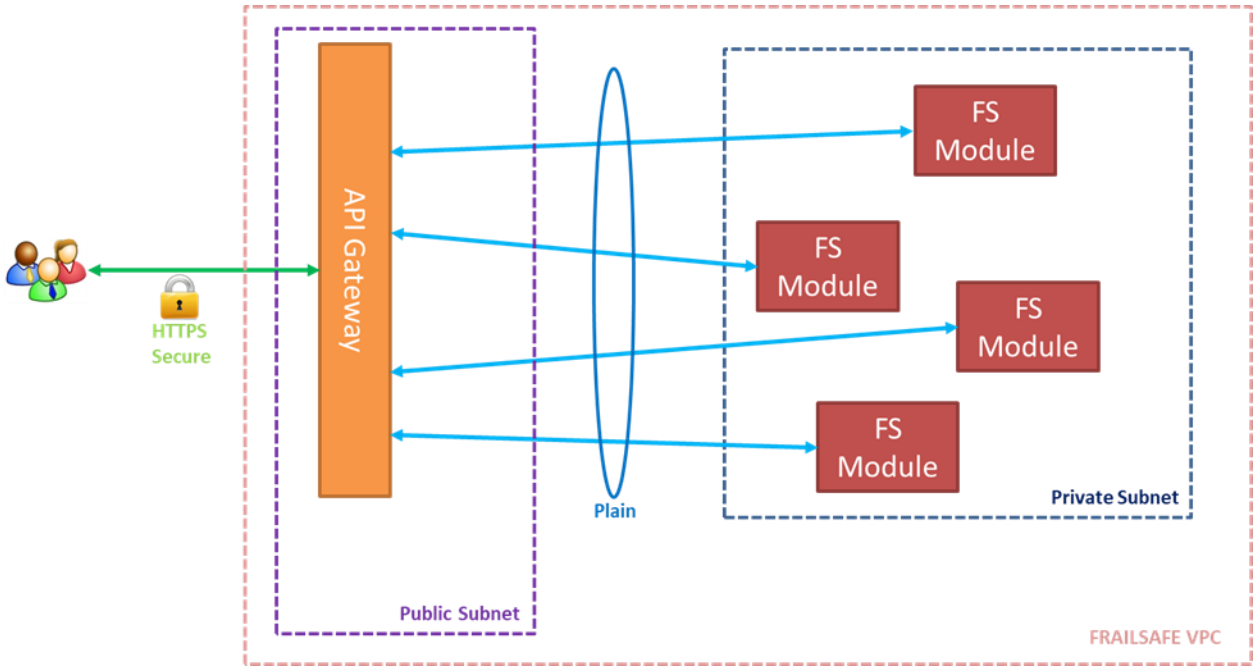

Figure 5 - API Gateway in the FrailSafe cloud infrastructure

Another important module that must be added to the infrastructure is a VPN server. This is mostly because resources deployed in the Private subnet and in the Database subnet could not be accessed in any way apart from a resource inside the VPC. Deploying a VPN server in the Public subnet gives us the possibility to establish a secure connection with the VPC and hence access to any deployed resources.

3.2 Resource deployment

After the initial configuration step described above, deployment of the different modules developed in FrailSafe was carried out. In this stage, each partner has been involved in order to clearly define technical requirement of each module to allocate resources inside the FrailSafe cloud. **Table 1** shows for each module its technical requirements and the correspondent resources that has been allocated to it.

Module	Sub-module	Resource[4]	Subnet
Data Analysis	Data Analysis cluster	4 t2.medium EC2 instances, 100 GB of EBS storage each	Private
	Data Analysis services	t2.medium EC2 instance with 50 GB of EBS storage	Private
	Data Fusion	t2.micro EC2 instance with 8 GB of EBS storage	Private

[4] The naming of quoted resources is coherent with Amazon AWS services specifications.

Social media Analysys	LingTester Backend	t2.small EC2 instance with 8 GB of EBS storage	Private
	LingTester Frontend	t2.micro EC2 instance with 8 GB of EBS storage	Public
Clinical Web Portal	eCRF Backend	t2.micro EC2 instance with 8 GB of EBS storage	Private
	eCRF Frontend	t2.micro EC2 instance with 8 GB of EBS storage	Public
	Data Visualization	t2.micro EC2 instance with 8 GB of EBS storage	Public
Sensors	Sensor data backend	t2.micro EC2 instance with 8 GB of EBS storage	Private
Games	Game backend	t2.micro EC2 instance with 8 GB of EBS storage	Private
Integration	Authentication	t2.micro EC2 instance with 8 GB of EBS storage	Private
	Integration services	t2.micro EC2 instance with 8 GB of EBS storage	Private
	API Gateway	t2.micro EC2 instance with 8 GB of EBS storage	Public
	VPN server	t2.micro EC2 instance with 8 GB of EBS storage	Public

Table 1 Technical requirements and the correspondent resources allocated to each module

To complete this section, summarizing the work done so far, Figure 6 shows the current status of the FrailSafe platform's cloud resources configured.

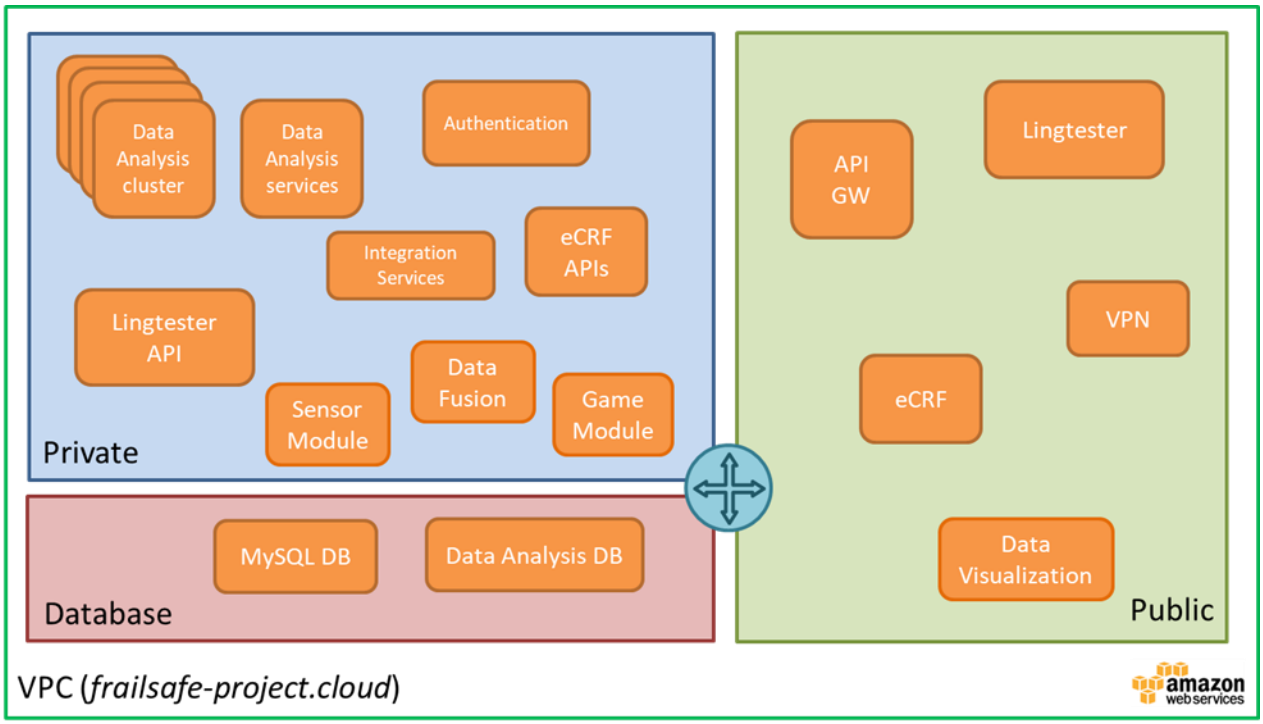

Figure 6 - Current FrailSafe cloud resources configured

4. STATE OF THE INTEGRATION

This chapter shows the state of the integration as at M18. As said before, we are now at the first integration stage and hence the major effort has been spent in configuring and deploying a robust and effective infrastructure that will host the integrated FrailSafe platform. Within this chapter, a complete table of modules and functionalities that are available will be provided.

Regarding the FrailSafe cloud, the picture below (Figure 7) shows a schema where each module is coloured with respect to its status, that can be:

- *Not present*, if the module has not been already deployed in the cloud because it was not foreseen to be ready at M18;
- *Deployed in the cloud, but in an early stage*. This happens for modules that are currently on development and their cloud deployment has been started. This also means that the module is not already integrated with the entire platform;
- *Deployed in the cloud and working*. This means that the module by itself is already working, but is not already fully integrated with the FrailSafe platform;
- *Deployed and integrated*. This means that the module, even if it is on a first release, is already fully integrated with the platform.

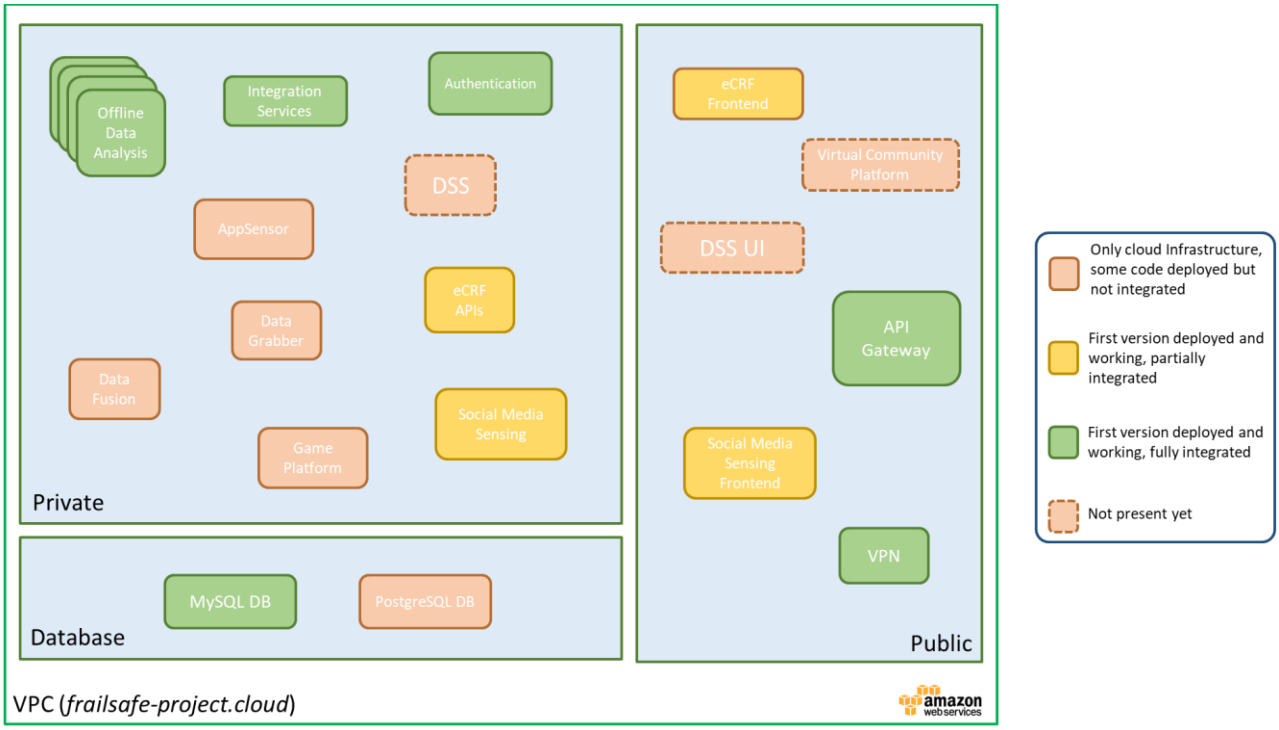

Figure 7 - Current FrailSafe cloud deployment state

As said before, the workplan of the project specifies three integration iterations that have been scheduled officially at M18, M24 and M32. Figure 8 (in paragraph 4.1) shows which features of each module will be present in the different steps of integration plan.

Aims of the integration task are to maintain in each iteration the features that was present and working of the previous one and to have all the features at least in the interation at M32.

4.1 Functional Integration Matrix

INDOOR LOCALIZATION	M18	M24	M32
Cloud Infrastructure	X		
Authentication (login/logout)			
Indoor localization preparation - monitoring enviroment setup	X		
Start/Stop the monitoring	X		
Being monitored (after start)	X		
Automatic collected data sent to server			

OUTDOOR LOCALIZATION	M18	M24	M32
Cloud Infrastructure	X		
Authentication (login/logout)			
Outdoor localization settings (GPS - autostart session - notification buttons - log file type - local folder)	X		
Start/Stop the monitoring	X		
Being monitored (after start)	X		
Automatic collected data sent to server			

WWBS + IMU	M18	M24	M32
Manual data transfer (via ECRF UI)	X		

SOCIAL MEDIA SENSING	M18	M24	M32
Cloud Infrastructure	X		
Authentication (login/logout)			
Analyse text message from social media	X		
Send data to DSS			
Send alert to DSS			
Send recommendation to patient			
Send mail to mailbox			

Send mail to patient			
Register social network personal account	X		

DATA GRABBER	M18	M24	M32
Cloud infrastructure	X		
Import data from ECRF			
Import data from Indoor & outdoor monitoring			
Import data from Auxiliary Devices			
Import data from Social Media Sensing			
Import data from Games			

OFFLINE DATA ANALYSIS	M18	M24	M32
Cloud Infrastructure	X		
Aggregate data			
Find patterns and associations			
Detect frailty level changing			
Identification of frailty signs			

DSS	M18	M24	M32
Cloud Infrastructure			
Support clinical staff in decisions			
Personalized feedback and intervention strategies to the patient			
Facilitate analysis of the collected data			
Provide personalized guidance, interventions, alerts			

ONLINE DATA ANALYSIS	M18	M24	M32
System login/logout through app			
Monitored for adverse events detection	X		
Send alert to DSS			
Receive alerts identified locally by the online processes			

GAMES	M18	M24	M32
Cloud Infrastructure	X		
Authentication (login/logout)			
Game selection			
Play selected game	X		
Automatic collected data sent to server			

AR GLASSES GAME	M18	M24	M32
Cloud Infrastructure	X		
Authentication (login/logout)			
Play selected game			
Automatic collected data sent to server			

DSS UI	M18	M24	M32
Cloud Infrastructure			
Authentication (login/logout)			
Patient selection			
View patient information			
Get/Set/Manage recommendations/suggestions/interventions			
Provide “researchers” features for testing hypothesis and querying data			
Visualize alerts generated by the DSS and by the Online Data Analysis App			

VIRTUAL COMMUNITY PLATFORM	M18	M24	M32
Cloud Infrastructure			
Authentication (login/logout)			
Communicate with other platform users			
Get recommendations			
Give recommendations			

IAM	M18	M24	M32
Cloud Infrastructure	X		

Authentication (login/logout)	**X**		
Validation request	**X**		
Patients list	**X**		

AUTHENTICATION UI	M18	M24	M32
Cloud Infrastructure	**X**		
Authentication (login/logout)	**X**		
Manage users			

ECRF	M18	M24	M32
Cloud Infrastructure	**X**		
Authentication (login/logout)	**X**		
User Management	**X**		
Patient Management	**X**		
Trials Management	**X**		
Device Management	**X**		
Collect data	**X**		
Export data	**X**		

CLINICAL WEB PLATFORM	M18	M24	M32
Cloud Infrastructure	**X**		
Authentication (login/logout)	**X**		
Wrap AUTH UI	**X**		
Wrap ECRF	**X**		
WRAP DSS UI			

Figure 8 Functional Integration Matrix

5. TESTS

In system integration, the testing phase represents an important process that it should be done periodically to verify if the whole system meets its requirements and if its performances are in accordance with the end-user expectations. This task is called *"System IntegrationTesting",* sometimes also known as SIT, and it involves generally the testing of a complete system with all the subsystem components or modules.

Usually, the beginning of the testing process of an integrated system starts when the whole system has been built and there are some features that can be tested.

However, this implies also that a previous testing step has been passed: before being released each module and subsystem should pass through an internal phase of tests where its requirements are verified and where it is internally validated by the partner individually responsible for its implementations.

After that, the subsystems are deployed but, for considering the integration procedure completed, they must pass two layers of tests: basic tests and automatic ones.

The first phase is performed manually by a human tester to double check that, at a higher-level, the expected features of the systems are working as expected.

The second phase is, when possible, performed automatically through the usage of unit tests. Specifically, the interfaces (i.e. the APIs) exposed by the FrailSafe system via their mapping into the API Gateway can be automatically tested periodically.

In the unlikely eventuality these APIs are not working properly and with the expected results, that would be made evident after being checked by the unit tests, that will give precious insights on where and how the system is bugged or failing. This testing step consists in a set of tests, written by developers, to verify whether each exposed interface, and the access they provide to the system's feautres and resources, works properly. In practice, this helps in testing the consistency of the deployed module double-checking that it provides the expected inputs to the other modules depending on its APIs.

As explained, the usage of automatic test procedures has the main objective of having a quicker and more efficient identification of bugs. This is possible because through them the bugs' origin is circumscribed to the single module and the specific function.

The testing phases should be as automated as possible; the manual check to detect bugs is often tedious, wastes a lot of time and could be ineffective. Automated integration testing can allow development teams to continue working forward and it increases the chances that small defects will be identified immediately, giving developers time to make appropriate bug-fixes.

The integration testing is finalised in a last test phase that is devoted to check the correct performance of the processes described as FrailSafe usecases in deliverable D1.2.

It's fundamental to keep in mind the differences among the different testing steps.

The "Testing" methodology of passes through four different steps:

1. *Module specific testing:* tests internal to an individual sub-system, cheking the correct working of the business logics internal to the module;

2. *Integration testing A*: after the sub-system has been deployed, its features are tested manually at a higher-level;

3. *Integration testing B*: the APIs exposed publicly by a sub-system through the API Gateway are automatically tested using unit tests implemented for the purpose;

4. *Integration Testing C*: the Use Cases of the system, as defined within the deliverable D1.2, are tested one by one using demo users, one or more per available role.

According to the "*Continuous Integration*"-like approach we adopted for the intergration of the FrailSafe system, it is worth underlining that the "Module specific testing" tests are performed for every build before being released and deployed, while the integration testing described in the last two steps are prediodically performed on the whole platform when one or more modules are updated or released.

Some differences between the step 1 – i.e. Module spcific - and the steps 2 and 3 – i.e. the integration testing - are:

- *Encapsulation*: tests in step 1 should not use external resources, instead integration tests use additional modules, the cloud infrastructures and the system's available resources;

- *Complexity*: module specific tests concern to small and distinct parts of code (since they are related to a specific sub-system). Integration tests are more complex since it tests a functionality of the system usually more than one sub-system to be implemented;

- *Test failure:* a failure in the step 1 tests is clearly a bug in the business logic of the code of the sub-system. On the contrary when an integration test fails, it's not necessary to look at the code that implements business logic but it's more likely that something has changed in the environment and needs to be addressed.

6. CONCLUSIONS

In this document, we have seen how the consortium has approached and organized the integration work, the methodology as well as design principles of the system – relevant to the integration tasks – has been presented and whose benefits analysed, the integration plan and the state of the art of the platform have been defined in details.

At month 24, December 2017, at the second integration iteration a new version of this document will be published. It is expected to have a progressive and sensible evolution of the integrated features and system for the forthcoming period in accordance with the timeline of the project and of the individual sub-systems.