D3.1.5

Proof of concept for home healthcare

Project number:	257243
Project acronym:	TClouds
Project title:	Trustworthy Clouds - Privacy and Resilience for Internet-scale Critical Infrastructure
Start date of the project:	1st October, 2010
Duration:	36 months
Program:	FP7 IP

Deliverable type:	Prototype
Deliverable reference number:	ICT-257243 / D3.1.3 PU / 1.0
Activity and Work package contributing to the deliverable:	Activity 3 / WP 3.1
Due date:	September 2013 – M36
Actual submission date:	8th October 2013

Responsible organization:	PHI
Editor:	Mina Deng
Dissemination level:	Public
Revision:	1.0

Abstract:	This document describes what has been accomplished during the development of the Home Healthcare Proof of Concept as a Trusted Platform as a Service, on TClouds trustworthy OpenStack.
Keywords:	Healthcare Trusted PaaS, cloud based healthcare platform, proof of concept

Editor

Mina Deng (PHI)

Contributors

Marco Abitabile (FCSR)

Kees Wouters, Andres Monge Moreno (PHI)

Disclaimer

This work was partially supported by the European Commission through the FP7-ICT program under project TClouds, number 257243.

Executive Summary

This document describes the final proof concept of the Healthcare Trustworthy Platform as a Service, or TPaaS, for TClouds home healthcare. It describes the requirements, the architecture, the underlying implementation modules, and the user manual of the proof of concept, which is built on top of TClouds as a benchmark application and user-centric evaluation.

The implementation of TPaaS involves a service-oriented architecture, including services that guarantee security, manage databases, and provide a user interface. The Healthcare TPaaS offers a set of on-board software security modules and trust protocols.

In TClouds, Activity 3 specifies cloud applications, functionalities and requirements, defines application prototypes to be implemented on the cloud platform, and validates the application prototypes and the TClouds platform. Two applications have been defined. In this document we describe the proof of concept for home Healthcare Trusted Platform as a Service (TPaaS), which is the use case discussed in work package 3.1.

The Healthcare TPaaS aims to be a platform that runs in the cloud in a secure and trustworthy fashion, allowing third party healthcare applications to use it and benefit from the service it provides. The Healthcare TPaaS is deployed on TClouds Trustworthy OpenStack, integrated with effort from Activity 2. To manage the (third party) healthcare applications, trust protocols and a user interface for access control has been provided, so users can be in full control of their data and who has access to the data.

The Healthcare TPaaS is a multilevel platform that aims to provide novel services such as:

- Store trustworthily health-related data (relying on a trusted IaaS);
- Provide API for 3rd party apps to access to users' health data, in a privacy-preserving manner (PaaS);
- Provide API for 3rd party apps to use identity/role management services (PaaS);
- Provide an interface to allow 3rd party app developers to register their application(s) in order to access the users' data (SaaS);
- Allow End Users to manage their data and specify privacy policy about which data a particular application/ user can access (SaaS).

Healthcare TPaaS follows the best practices in terms of software development. Modularity has been taken into account in order to allow different development teams to interact among them and build the platform in parallel.

TPaaS is built in layer abstractions, namely the JPA layer, as the lowest layer abstraction, the Domain layer that implements all the entity and main security features, the Application layer that acts as a use case container, the Log Service, the User interface that allows users to access the platform and user data, Rest Services that allow the application layer to communicate to the outside world, and trust protocol Oauth2Share that enforces authorization and access control between application and end user.

The Healthcare Platform consists in 4 Virtual Machines to be deployed on the TClouds trustworthy Openstack, namely Appliance as the main VM, the PHR VM: it is a virtual machine that stores all the PHR data, the EHR_IT VM that stores the EHR data of legal entities that resides in Italy, and the EHR_DE VM that stores all the EHR data of legal entities that resides in Germany.

Please note that the list of verified security components provided by the TClouds Trustworthy OpenStack that the healthcare TPaaS is built on is provided in D3.3.4.

The generic trust protocols describe the trust protocols used at the application interface of the trustworthy healthcare platform. The goal of the trust protocols is to enforce the authorization rights given by a user to another user or an app with respect to accessing his or hers PHR/EHR data, and to do this in a REST full manner.

In order to define the trust protocols, four use cases are discussed. In the normal use case the data owner wants access to his or her data. In this case no delegation of rights is needed; the owner can authorize the access. An independent device is a device (including an app) that can access data without direct owner control. In the authenticated user use case the data owner gives consent to an app to access his or her data for any user subscribed to the app. In the authorized user use case, the data owner explicitly gives another user the right to access his or her data.

Based on the use cases, the relations between the various components are defined. Privacy rights as delegated to the apps or other users in the “delegates to” relation are stored in the privacy policies database. When an app wants to retrieve information from the PHR/EHR database, it first has to check the privacy policies on whether or not this is allowed. Creating the “delegates to” relations and putting them into the privacy policies database is not part of the trust protocols, this is done by the UI part of the trustworthy healthcare platform. The trust protocols ensure that this policy is enforced.

Trust protocols is designed and developed based on and extends the general idea behind OAuth2.0. The normal and independent device use cases can be handled by the protocolsof standard OAuth2.0. To support the authenticated user and authorized user use cases, the OAuth2.0 protocol is extended.

The document also provides a user manual on how the proof of concept of the Healthcare TPaaS may be operated. The distribution of the proof of concept of the Healthcare TPaaS is not provided in the form of binary files due to the complex installation. Instead, this platform is accessible online (i.e. on the TClouds Trustworthy Openstack cloud). With a graphical user interface, it enables functionalities such as login/sign-in, adding new friends, privacy policy specification, and auditing. The Platform is accessible from the web at https://tpaas.eservices4life.org

Contents

List of tables

List of figures

Chapter 1

Introduction

Chapter Author:

Mina Deng (PHI)

1.1 TClouds — Trustworthy Clouds

TClouds aims to develop trustworthy Internet-scale cloud services, providing computing, network, and storage resources over the Internet. Existing cloud computing services today are generally not trusted for running critical infrastructure, which may range from business-critical tasks of large companies to mission-critical tasks for the society as a whole. The latter includes water, electricity, fuel, and food supply chains. TClouds focuses on power grids, electricity management and patient-centric health-care systems as main applications.

The TClouds project identifies and addresses legal implications and business opportunities of using infrastructure clouds, assesses security, privacy, and resilience aspects of cloud computing and contributes to building a regulatory framework enabling resilient and privacy-enhanced cloud infrastructure.

The main body of work in TClouds defines an architecture and prototype systems for securing infrastructure clouds, by providing security enhancements that can be deployed on top of commodity infrastructure clouds (as a cloud-of-clouds) and by assessing the resilience, privacy, and security extensions of existing clouds.

Furthermore, TClouds provides resilient middleware for adaptive security using a cloud-of-clouds, which is not dependent on any single cloud provider. This feature of the TClouds platform will provide tolerance and adaptability to mitigate security incidents and unstable operating conditions for a range of applications running on a cloud-of-clouds.

1.2 Activity 3 — Benchmark Applications & User-centric Evaluation

Activity 3 focuses on the applications and the evaluation of the TClouds platform. The activity's objective is to define and validate cloud applications' architecture and specifications (the medical and the smart grid use case). For this purpose, Activity 3 has specified cloud and application functionalities and requirements, has defined application prototypes to be implemented on the cloud platform, and has validated the application prototypes and the TClouds platform. For this purpose, the requirements defined in Activity 1 have served as a generic guideline, which has been refined and consolidated in Activity 3. Finally, there was a continuous and close interaction between Activity 3 and Activity 2 in order to make sure that the platform and applications match the specifications and that the TClouds project achieves its overall objectives.

1.3 Work package 3.1 — Cloud Applications Data Structures for Home Healthcare Benchmark Scenario

WP3.1's main objective was to define the cloud architecture and specification for running TClouds selected application, i.e., to provide technical requirements for cloud computing for the medical sector, especially in the area of home healthcare devices and then turn the analysis into an architecture, API, and protocols for client side. This architecture is closely linked and integrated with WP2.1, WP2.2, and WP2.3. The protocols will be validated in collaboration with the demonstrator implementation.

1.4 Deliverable 3.1.5 — Proof of concept for home healthcare

1.4.1 Overview

This document provides explanatory information of the proof of concept for home healthcare trustworthy platform. The document details the implementation, design, architecture and embedded trust protocols, which extend the draft proof of concept for home healthcare in D3.1.3. It also considers security and resilience requirements for the underlying cloud environments. The document provides some instructions on how the prototype can be used.

1.4.2 Structure

Chapter 2 introduces design and implementation decisions of the proof of concept for home healthcare, which is the Healthcare TPaaS (Trusted Platform as a Service). Discussions on design and implementation decisions will be provided, with an overall architecture and underlying implementation modules.

Chapter 3 provides the trust protocols used at the application interface of the trustworthy healthcare platform. The goal of the trust protocols is to enforce the authorization rights given by a user to another user or an app with respect to accessing the user's PHR/EHR data.

Chapter 4 provides a user manual on how the proof of concept of home healthcare may be operated.

1.4.3 Deviation from Work plan

The implementation of this prototype was fully compliant with the defined work plan.

1.4.4 Target Audience

Target audience of this deliverable includes all TClouds partners and parties that wish to have a deeper view of the development result of the trustworthy healthcare platform. The deliverable, to be fully understood, requires some background in software engineering and healthcare IT systems.

1.4.5 Relation to Other Deliverables

D3.1.5 covers Task 3.1.3 trust protocols and Task 3.1.4 Proof of concept prototypes. Trust protocols are related to WP1.2 legal implications of cross-border cloud implementations and WP2.3 cross-layer security and privacy management. Moreover, D3.1.5 is based on D3.1.3 the draft proof of concept, and provides a list of security and privacy requirements, which can be covered by TClouds security components as presented in D2.4.2.

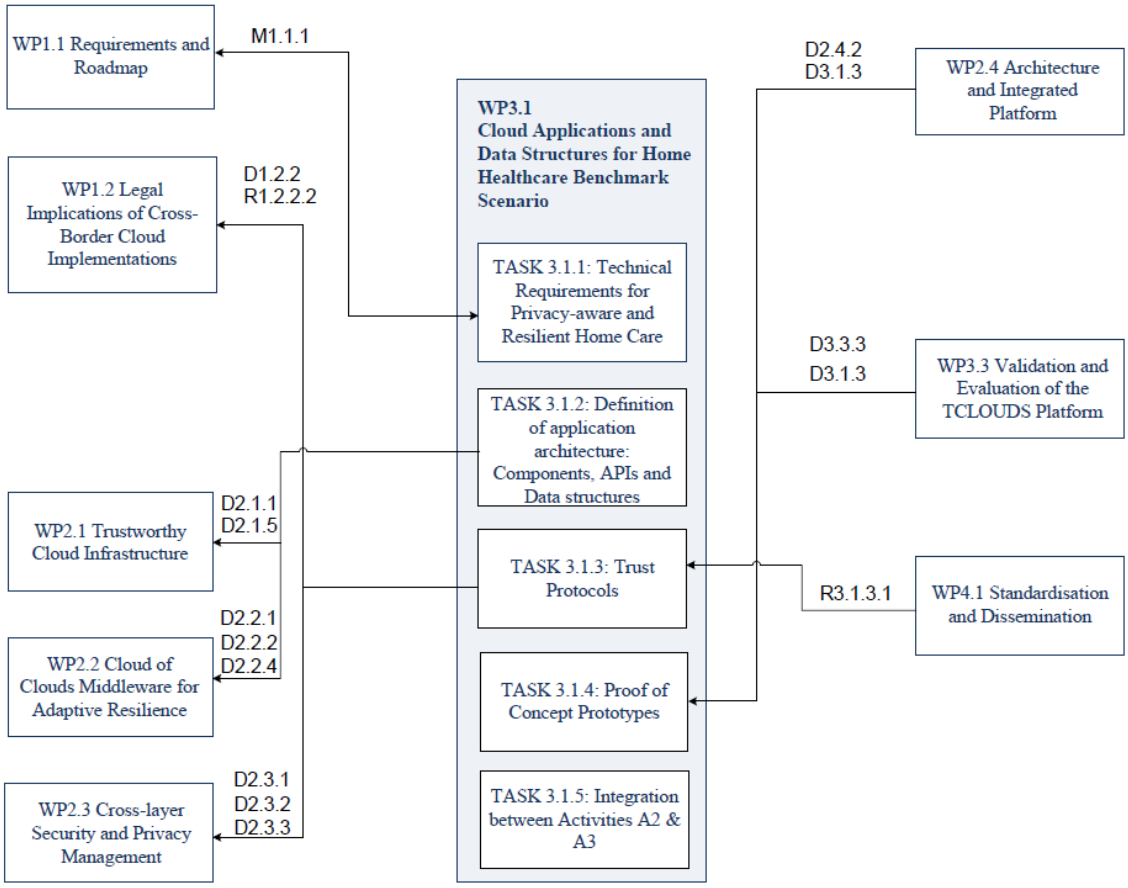

Figure 1: Interdependency chart for WP3.1

Chapter 2

Home Healthcare proof of concept implementation

Chapter Author:

Marco Abitabile (FCSR)

In this chapter we provide some insights on the implemented Home Healthcare Platform (namely the Healthcare TPaaS).

2.1 Introduction

The Healthcare TPaaS aims to be a platform that runs in the cloud in a secure and trustworthy fashion, allowing third party healthcare applications to use it and benefit from the service it provides. In particular, the TPaaS platform is deployed on top if the trustworthy OpenStack provided by TClouds.

This section describes in detail the functionalities and services offered at the Platform as a Service (PaaS) layer. In this we try to outline the meaning of “Healthcare Trusted PaaS”, or simply “Healthcare TPaaS”. Indeed, some of the services actually have an interface for End Users, and we consider them to be Software as a Service (SaaS). The Healthcare TPaaS is a multilevel platform that aims to provide novel services such as:

- Store trustworthily health-related data (relying on a trusted IaaS);
- Provide API for 3rd party apps to access to users’ health data, in a privacy-preserving manner (PaaS);
- Provide API for 3rd party apps to use identity/role management services (PaaS);
- Provide an interface to allow 3rd party app developers to register their application(s) in order to access the users’ data (SaaS);
- Allow End Users to manage their data and specify privacy policy about which data a particular application/ user can access (SaaS).

2.2 Architecture

It is well known that the more lines of code are written, the more the software is prone to have errors. The main discriminators of this rule are good programmers and good software architecture.

TPaaS aims to be a Trustworthy platform. Its “trustworthy” concept leverages most of TClouds infrastructure features; nonetheless if the Healthcare platform itself is not well designed, an attacker may pass directly through the VM instead of trying to pass through an extremely hard-to-brake and complex cloud system.

We evolved these ideas by enhancing the layering concept and introducing stronger design theories and a precise layering of the platform, supporting ourselves with tools that allow TPaaS to be oriented to large concurrent systems.

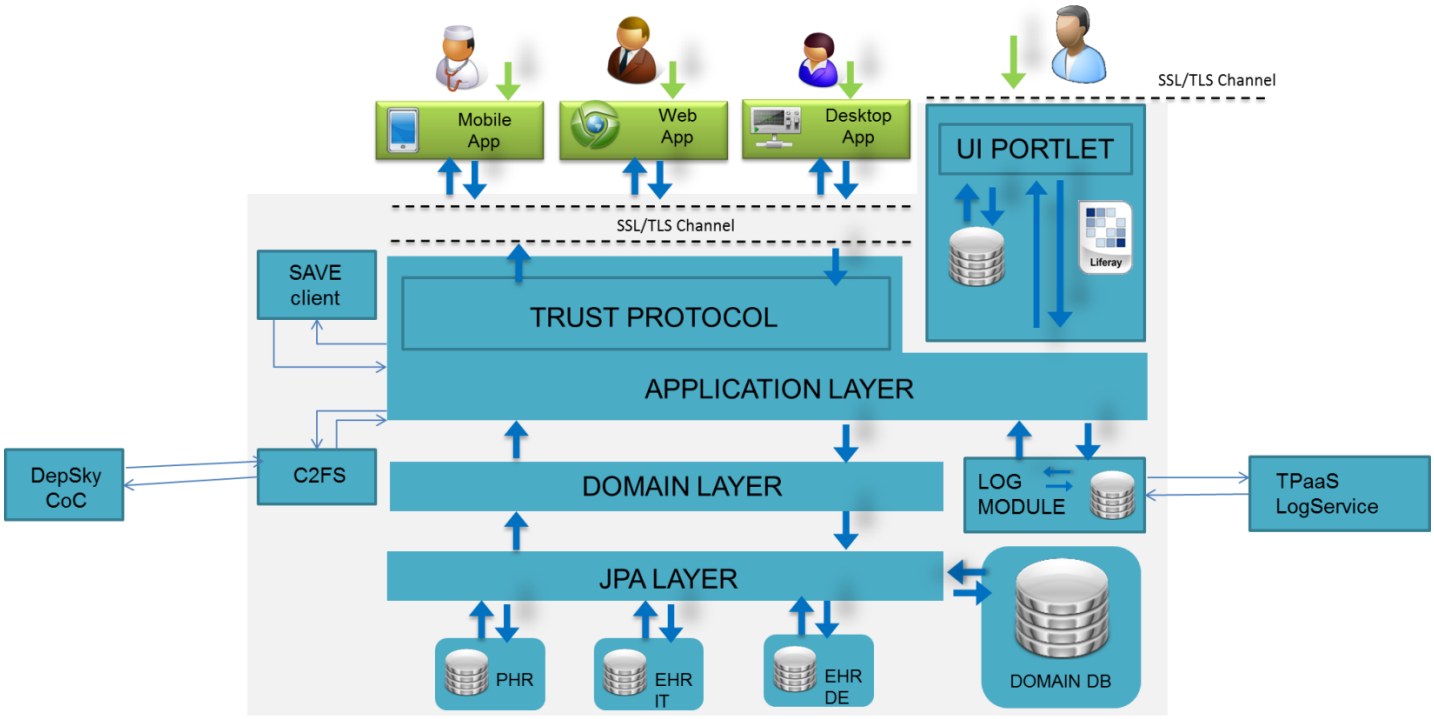

Figure 2: Healthcare TPaaS architecture

Health TPaaS follows the best practices in terms of software development. In addition, modularity has been taken into account in order to allow different development teams to interact among them and build the platform in parallel.

TPaaS – Trustworthy Healthcare Platform's architecture takes inspiration of Domain Driven Design concepts and theory. The whole system has been developed maintaining the Domain as the core of the problem. This allows building safe, strong and reliable code.

In fact, building application and complex platform systems requires programming techniques that allow expanding the system capabilities without damaging other part or without creating new bugs of pitfalls.

TPaaS layering is done taking into account some concepts considering (Evans, 2003)'s best practices in terms of application's layers.

Layer	Description
SaaS/PaaS Interface	Responsible for showing information to the user and interpreting the user's commands. The external actor might sometimes be another computer system rather than a human user.
Application Layer	Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems. The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems. This layer is kept thin. It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down. It does not have state reflecting the business situation, but it can have state that reflects the progress of a task for the user or the program.
Domain Layer (or Model Layer)	Responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure. This layer is the heart of business software.
Infrastructure Layer	Provides generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, and so on. The infrastructure layer may also support the pattern of interactions between the four layers through an architectural framework.

Table 1: Healthcare TPaaS layer description. As written in (Evans, 2003)

Moreover, the layering strategy adopted has to respect some straightforward principles:

- Given a layer abstraction, it is able to use lower levels (It has references to it) and knows how lower levels exchange data, meanwhile it does not know upper levels functionalities nor have their references.
- The most of the calls should be singletons, giving “session” concept to higher levels (e.g.: UI)

2.3 Software abstractions and Modules

As shown in Figure 2 and described in the previous chapter, TPaaS is built in layer abstractions. In this chapter we will describe each of them.

- JPA layer: this is the lowest layer abstraction. It manages all the persistency with the databases. It is able to store health data in a specific database depending on the user policies and geo-location; moreover it manages all the transactions with the persistence unit and ensures that each task submitted to it has an atomic nature, allowing highly parallelism and integrity of data.
- TPaaS domain implements all the entities involved into the system, from users, to applications to relationship among them (including privacy policies). The domain layer “knows” all the actors involved and offers a set of interfaces that allows upper layers to use the domain properly. The domain implements all the main security features, from policy change to relationship management. This

means that any other layer needs to “ask” the domain layer in order to persist or modify a policy.

- Application layer, acts as a use case container, in which every single use case in the platform is implemented and satisfied.
- Log Service is an independent module used to store log information in the remote Log Service and used to record successful and failed access data attempts, as well as privacy policy updates
- User interface (UI) consists in specific view to allow users to access the platform and their user area
- Oauth2Share: is the trust protocol that enforces authorization and access control between application and end user.

2.4 Deployment scenario

Below it is described the deployment scenario that will be used into the TClouds Trustworthy Openstack Infrastructure.

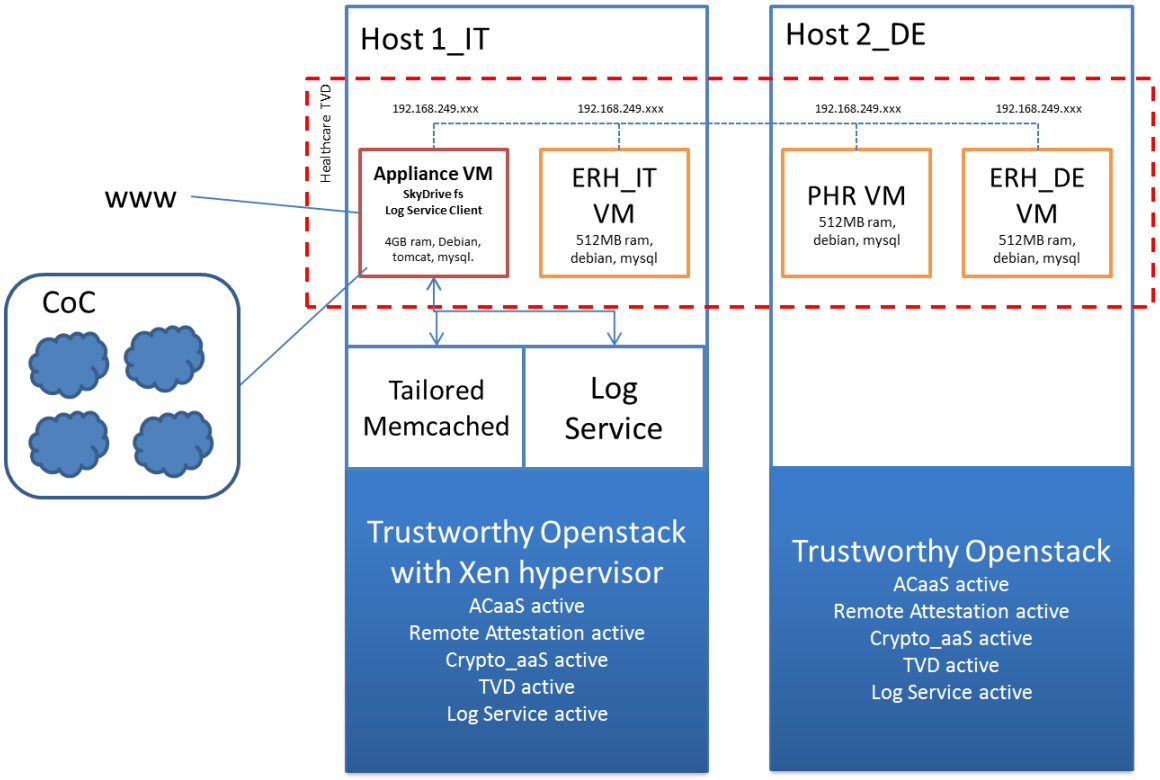

Figure 3: Home healthcare platform deployment block diagram

The Healthcare Platform consists in 4 Virtual Machines:

- Appliance: it is the main VM. In here resides all the logic of the platform. It manages platform users, their relations, UI and all the business logic to work with DepSky, LogService and Tailored Memecached sub-component.

It is able to receive data from external requests and dispatch it in the correct Database VM.

Due to its nature to have sensitive data (mainly username and password plus all the user information) we decided that it needs to have an encrypted VM in order to increase the security.

- PHR VM: it is a virtual machine that stores all the PHR data.
- EHR_IT VM: it stores all the EHR data of legal entities that resides in Italy (e.g. an Italian hospital).
- EHR_DE VM: it stores all the EHR data of legal entities that resides in Germany (e.g. a German hospital).

Chapter 3

Trust protocols

Chapter Author:

Kees Wouters, Andres Monge Moreno (PHI)

3.1 Introduction

This chapter describes the trust protocols used at the application interface of the trustworthy healthcare platform (see Figure 4). The goal of the trust protocols is to enforce the authorization rights given by a user to another user or an app with respect to accessing his or hers health data, providing easy and straightforward REST interfaces.

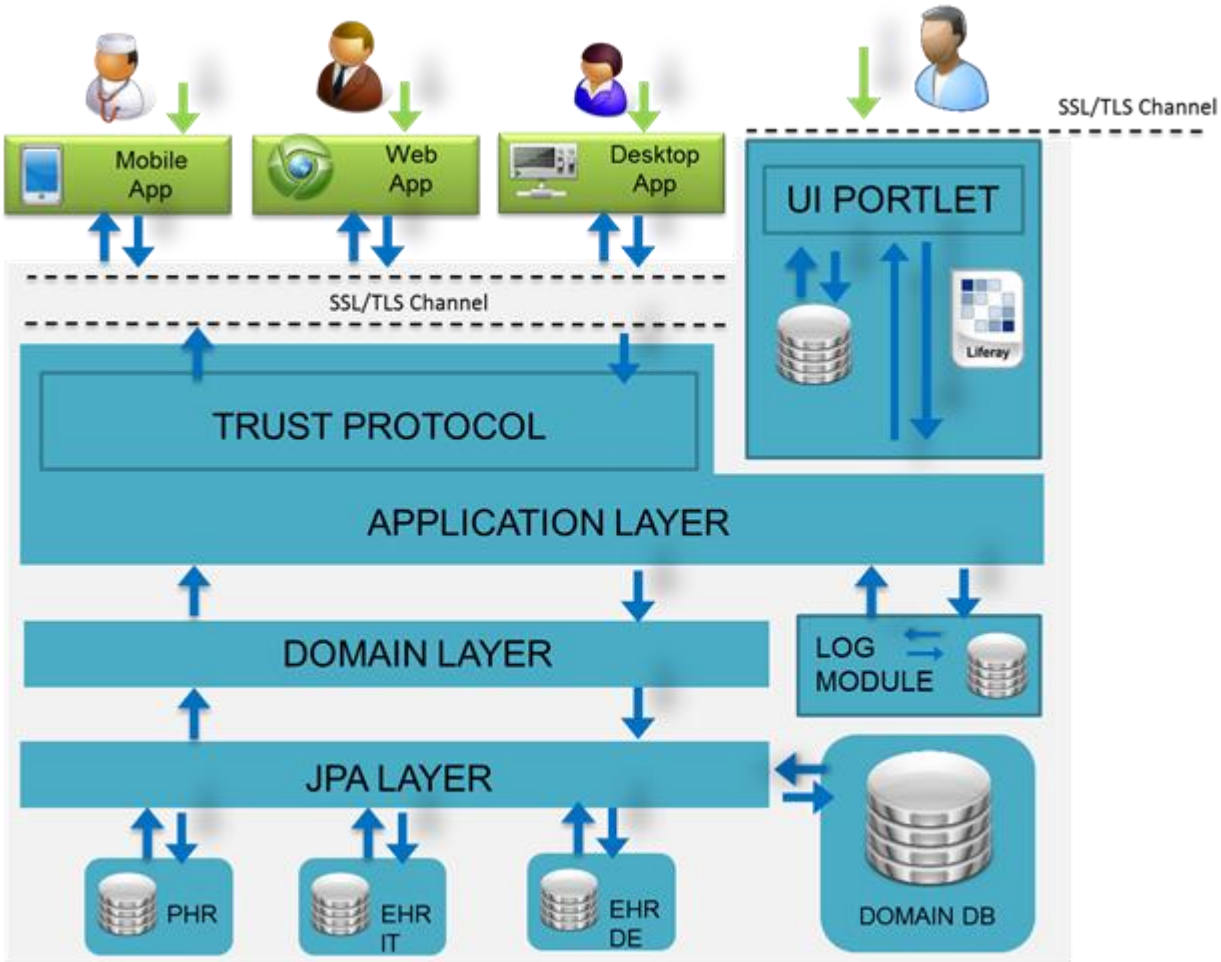

Figure 4: Trustworthy Healthcare Architecture

3.2 Requirements

In order to define the trust protocols, first the use cases will be discussed. Based on the use cases the relations between the various components will be defined. Both use cases and relations will be used to define the trust protocols, a boundary condition is to define RESTful protocols.

3.2.1 Use cases

Four use cases have been identified. But before going into the specifics of each use case first the context will be sketched. The trustworthy healthcare platform allows for the creation of applications (apps in Figure 4). The trust protocols provide the means to give an app access to PHR/EHR data in a secure and RESTful way (authenticated and authorized) when the PHR/EHR data owner or his delegate has given consent. The four use cases, explained in the following chapters, focus on the different ways consent can be given.

3.2.1.1 Normal

In the normal use case the data owner wants access to his or her data (see Figure 5). In this case no delegation of rights is needed; the owner can authorize the access. The owner delegates the rights needed for the app to access the requested data. Furthermore the owner subscribes to the app to make him known to the app. The app takes care of the data access to the trustworthy healthcare platform.

Figure 5: Normal use case

3.2.1.2 Independent device

An independent device is a device (including an app) that can access data without direct owner control. An example of such a device is the ActiWatch (see Figure 6). This is a wrist wearable device that measures the activity of the wearer, and the light conditions under which the activity is taking place.

Figure 6: ActiWatch

In case of an independent device, the data owner has given consent to the app controlling the device to access the data (see Figure 7). The consent is given as part of the “delegates to” relation between the data owner and the app. As the device will run without direct owner control, one should be careful with respect to the rights given to the device. There are several ways to restrict the rights, e.g. by limiting the data fields which can be accessed, by limiting the time that the device can work before renewal of the consent is needed etc..

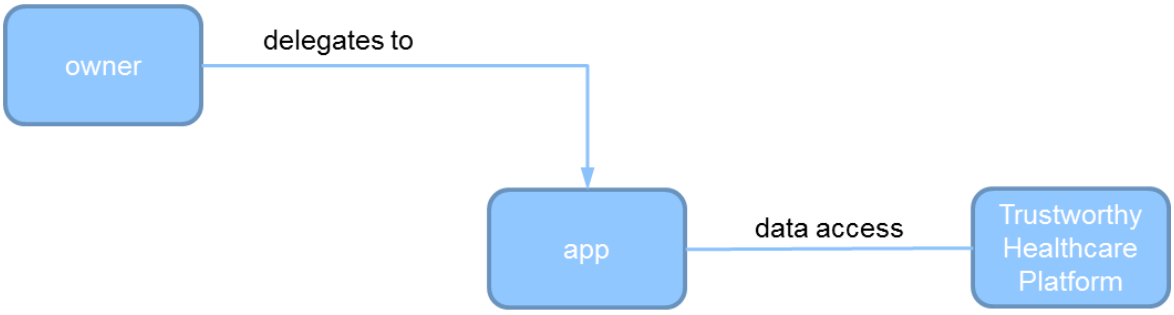

Figure 7: Independent device use case

3.2.1.3 Authenticated user

In the authenticated user use case the data owner gives consent to an app to access his or her data (see Figure 8, “delegates to”) for any user subscribed to the app (see Figure 8, “subscribes to”). An example of such an app is a phone-book; any user of the phone-book has access to the phone numbers published. Although in this case the user has to identify him before he is allowed to look into the phone-book. Like in the independent device use case, the data owner should be careful with respect to the rights given to the app, as the data owner has no prior knowledge on who will access the data.

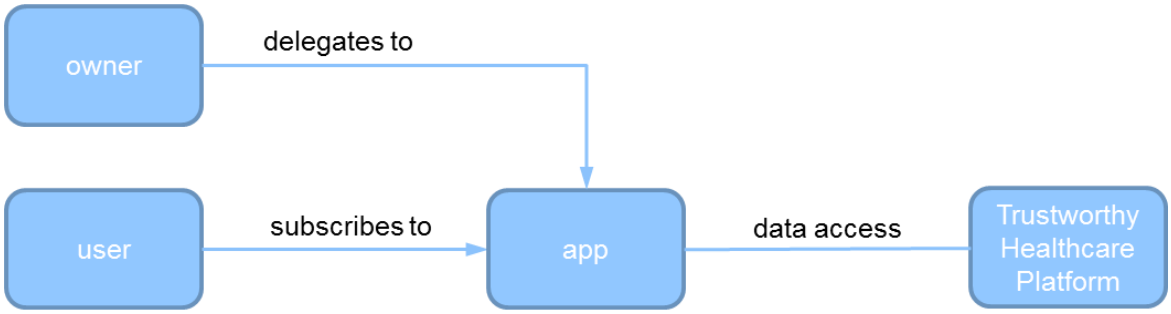

Figure 8: authenticated user use case

3.2.1.4 Authorized user

In the authorized user use case, the data owner explicitly gives another user the right to access his or her data (see Figure 9, “delegates to” between owner and user). From the app point of view, the user is considered to be a data owner. Hence the “delegates to” relation is between the user and the app. To use the app, the user “subscribes to” the app. The access rights of the user can delegates are determined by the rights the owner granted him or her in the “delegates to” relation.

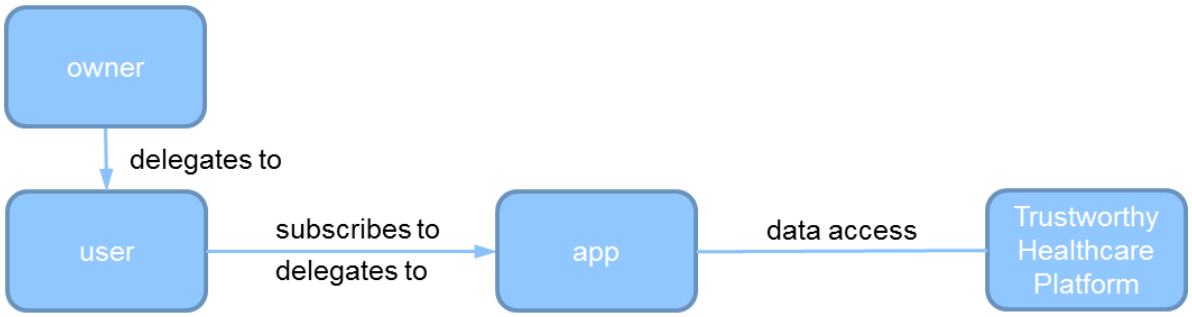

Figure 9: authorized user use case

3.2.2 Relationships

The use cases base their effectiveness on the rights that have been delegated to the apps or to other users in the “delegates to” relation. Properties of the “delegates to” relation are stored in the privacy policies database. When an app wants to retrieve information from the PHR/EHR database, it first has to check the privacy policies database, whether or not this is allowed (see Figure 10). Creating the “delegates to” relations and putting them into the privacy policies database is not part of the trust protocols, this is done by the UI part of the trustworthy healthcare platform (see Figure 4). The trust protocols ensure that this policy is enforced.

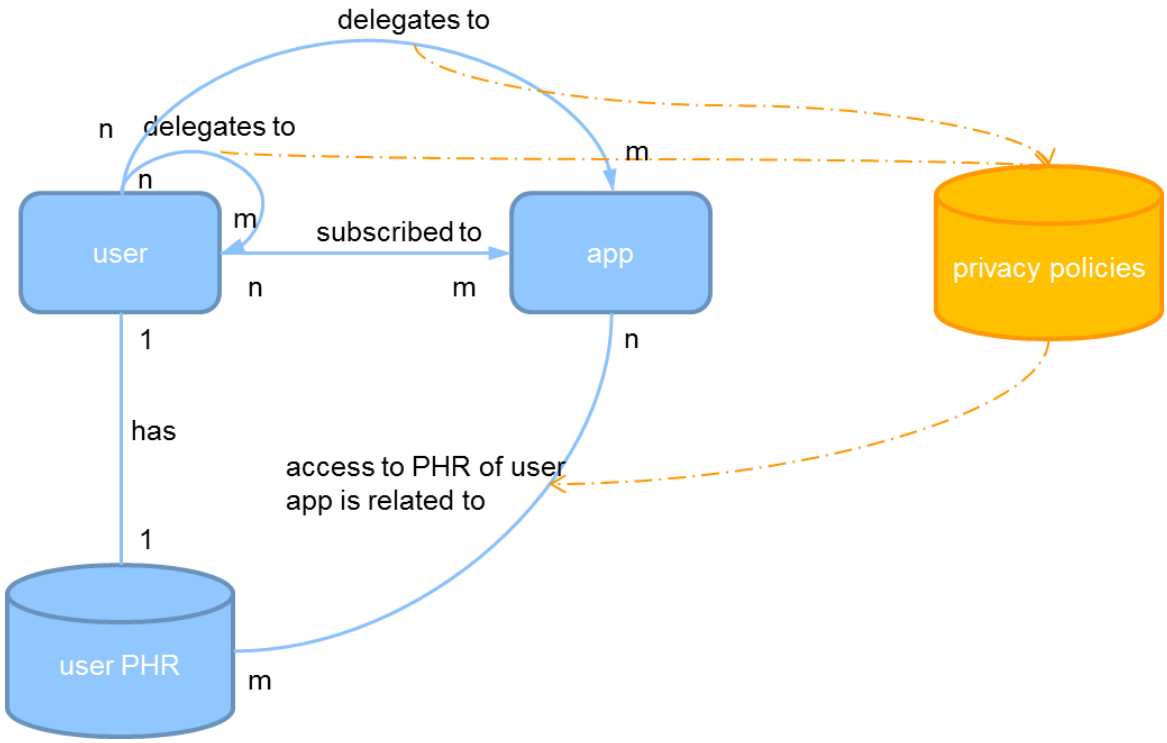

Figure 10: Relationship diagram

3.3 Selected protocol

For implementing the requirements, the OAuth2.0 protocol has been chosen. The choice has been based on three properties:

1. OAuth2.0 complies with the REST requirement
2. OAuth2.0 already supports the normal and the independent device use case
3. OAuth2.0 allows for extension of the protocol

In this chapter the general idea behind OAuth2.0 will be explained. In the section 3.4 the normal and independent device use cases will be handled, these are standard OAuth2.0 and section 3.5 the extensions needed for the authenticated user and authorized user use cases will be introduced.

3.3.1 OAuth2.0

According to abstract in RFC6749 “The OAuth2.0 Authorization Framework”:

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

Thus OAuth2.0, out of the box, supports the normal use case (on behalf of the resource owner) and the independent device use case (by allowing the third-party application to obtain access on its own behalf).

To support this OAuth2.0 has defined four different roles and a protocol flow between these roles. In this document only the basic flows will be described. For an in-depth view, please refer to the official Oauth2.0 RFC[1]

The roles are:

Resource owner: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

Resource server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Client: An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

Authorization server: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The term access token used in the roles definition will be explained after the protocol flow description. The protocol flow (see Figure 11) shows three exchanges of information. Starting point is when the client wants to access to a protected resource. In order to achieve this, the client first performs a request to the resource owner for an authorization grant. The authorization grant is a credential representing the resource owner's authorization. Please keep in mind that the grant can be limited in scope and in time. In the next step, the client by presenting the authorization grant, requests the authorization server for an access token, while in the third step, the client can get the protected resource from the resource server by presenting the access token.

The lifetime of the authorization grant and the access token are different, typically the grant has a very short lifetime whereas the access token has a lifetime in line with the usage of the protected resource.

[1] http://tools.ietf.org/html/rfc6749

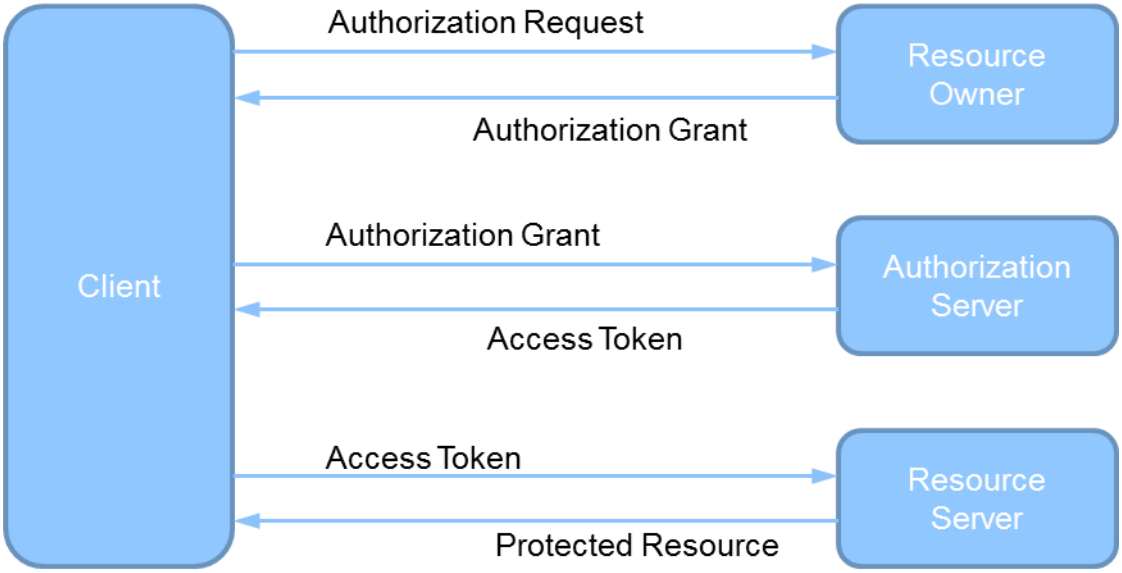

Figure 11: OAuth2.0 protocol flow

The access token provides an abstraction layer, replacing different authorization constructs (e.g., username and password) with a single token recognized by the resource server. This abstraction enables issuing more restrictive access tokens than the authorization grant used to obtain them, as well as removing the resource server's need to recognize a wide range of authentication methods.

The format of the access token is up to the implementation, and is no part of OAuth2.0. This leaves room for adapting the token's format to the security requirements of the resource server.

Some applications need very powerful access tokens (e.g. long lifetime), which are undesirable from a security point of view. To handle this, the client should be able to request access tokens on its own account. To accommodate this, OAuth2.0 has defined a refresh token. The refresh token allows the client to request additional access tokens, thus reducing the need for very powerful access tokens. Normally this construction is used when a longer lifetime is needed than normally issued to an access token. As described in the protocol flow in Figure 11, when requesting a resource with an invalid access token an error will occur. The only option for the client is to ask a new grant from the resource owner. In case of a refresh token, the client can directly ask the authorization server for a new access token based on the evidence given by presenting the refresh token.

Security

As the reader may notice, security has been barely mentioned beside that the format of the tokens is not part of OAUth2.0 and can be defined according to the security requirements of the resource server. OAuth2.0 itself relies on the HTTPS protocol to secure the communication channel.

Remark: In the Trustworthy Healthcare Platform the "protected resource" (see Figure 11) normally is data, so in the remainder the protected resource of mostly referred to as data.

3.4 Basic protocol

The basic protocol supports the regular use cases i.e. normal and independent device are standard OAuth2.0 use cases. Logging on onto the system is kept implicit in these cases.

3.4.1 Normal use case

Figure 12 depicts the communication flow of the normal use case. The main difference with respect to the flow in Figure 11 is that a refresh token is used.

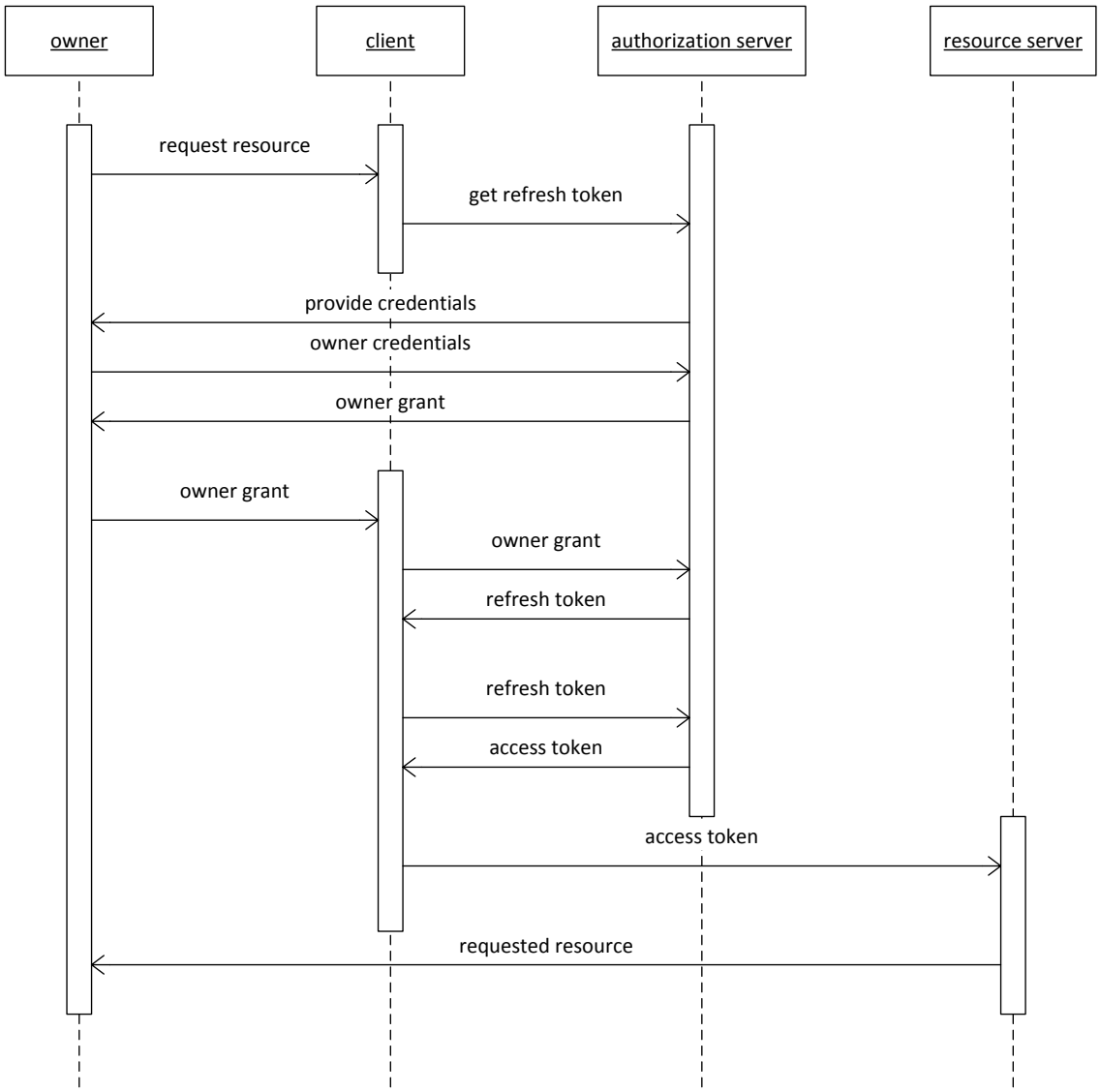

Figure 12: normal use case protocol flow

An example of the normal communication flow is a patient requesting to read his blood pressure measurements of the last week.

3.4.2 Independent device use case

The communication flow of the independent device use case looks a bit different since the owner is not going to be the initiator of the communication, instead the client itself will be. To show this more prominent, the activity bars have been split, so the upper part of Figure 13 is about setting up the client (device) and the lower part is about the client (device) retrieving or writing resources.

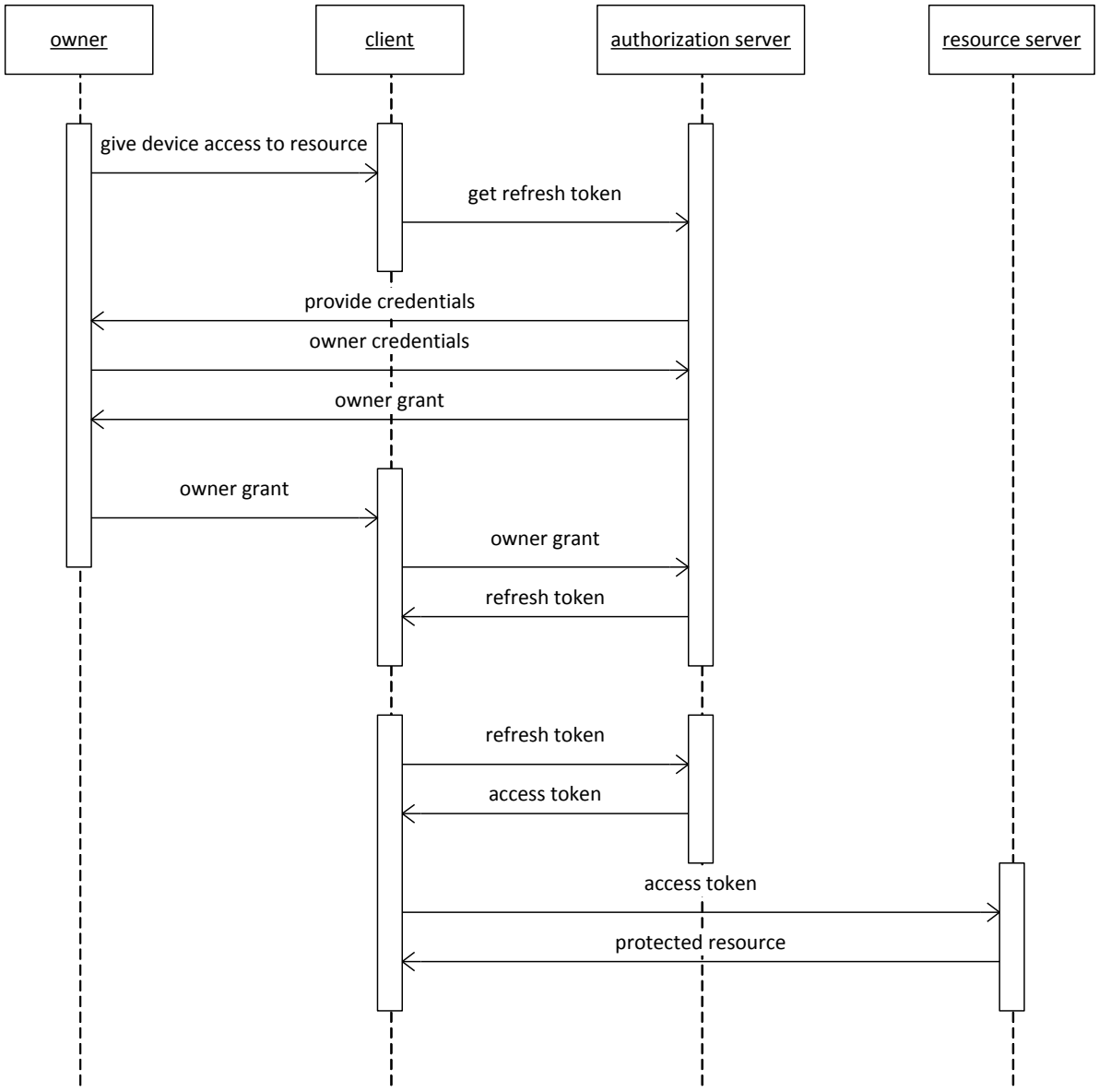

Figure 13: Independent device protocol flow

An example of the independent device communication flow is a blood pressure measurement device which automatically measures the blood pressure and writes the data to the trustworthy healthcare platform.

3.5 Protocol extensions

The Oauth2.0 protocol has been extended for the authenticated user and authorized user use cases. Part of this will be solved in OAuth2.0 and part via the privacy policies of the trustworthy healthcare platform.

It is important to be aware that the refresh and access token do not need to refer to the identity of the owner or user. In other words anybody providing the correct refresh and access tokens can have access to the protected resource. One of the main issues for the trustworthy healthcare platform is traceability, so the extensions are made in such a way that they provide the required traceability.

3.5.1 Authenticated user extension

The communication flow of the authenticated user use case is similar to the independent device flow; rights are delegated to the app, but there is another user (which is not the owner) controlling the app (see Figure 14). In order to access the data, the user has to authenticate himself (logon onto the system.

To support this, the independent device flow is extended to handle the identity information. This is done in several places.

- The owner identification is explicitly made part of the refresh token (see first part of protocol flow in Figure 14).
- The user who wants access to the data logs on, she or he will get a refresh token including her or his identification (second part, Figure 14).
- To access the data both refresh tokens need to request access tokens and both access tokens are needed to get the actual access (last part, Figure 14).

Part of the process on the server side (not shown in Figure 14) is logging of the identity of the user accessing the data and logging of the owner identity.

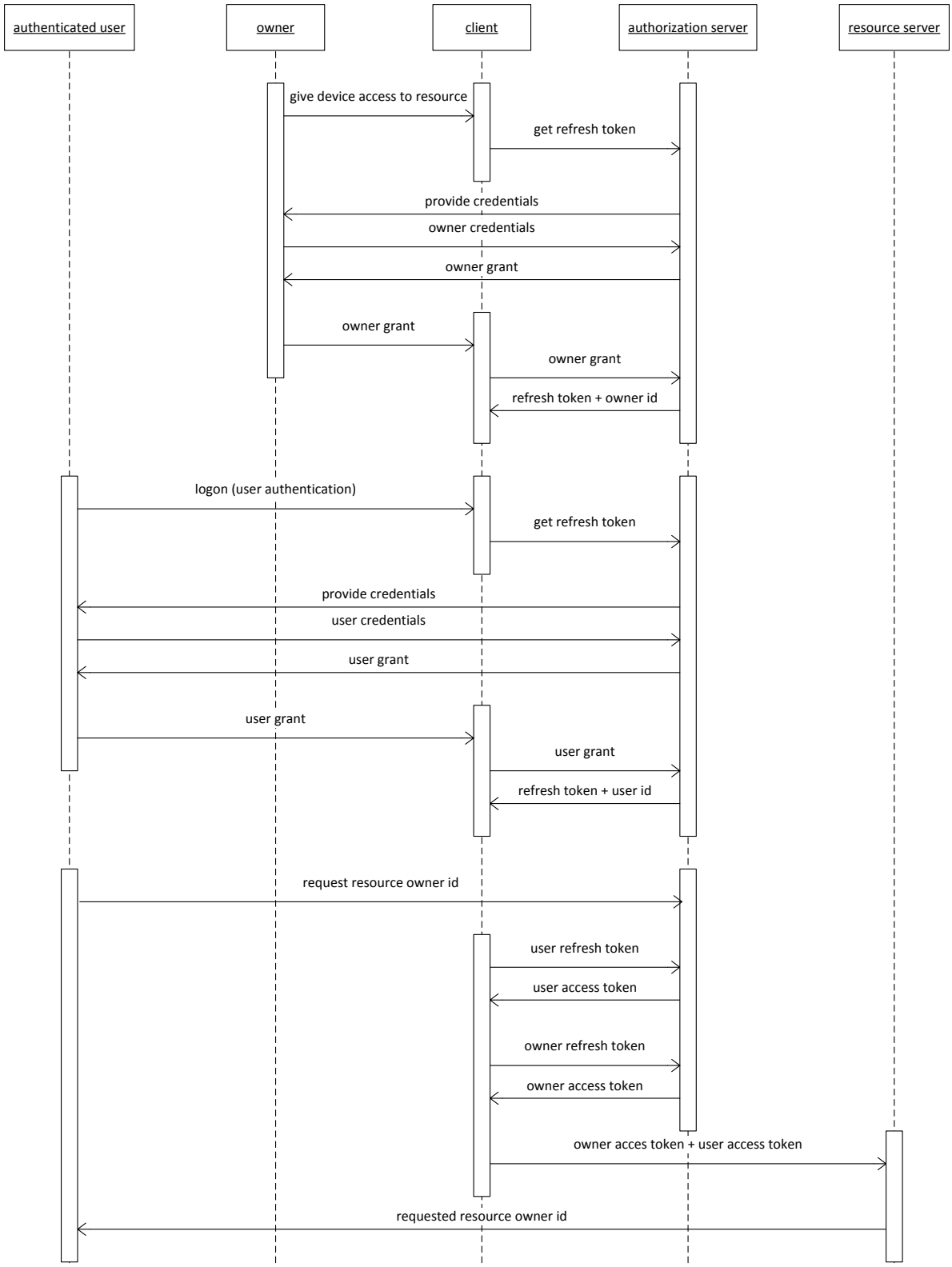

Figure 14: Authenticated user protocol flow

3.5.2 Authorized user extension

The authorized user extension is based on the normal communication flow. The only difference being that the data owner has delegated the right to grant access to another user. However to be able to verify who accessed the data, the authorized user has to logon. This is described in the first part of Figure 15, the second part shows the access itself. The actual delegation is not part of the trust protocol, this is functionality of the privacy policies layer of the trustworthy healthcare platform, and the extension uses this information via the authorization server.

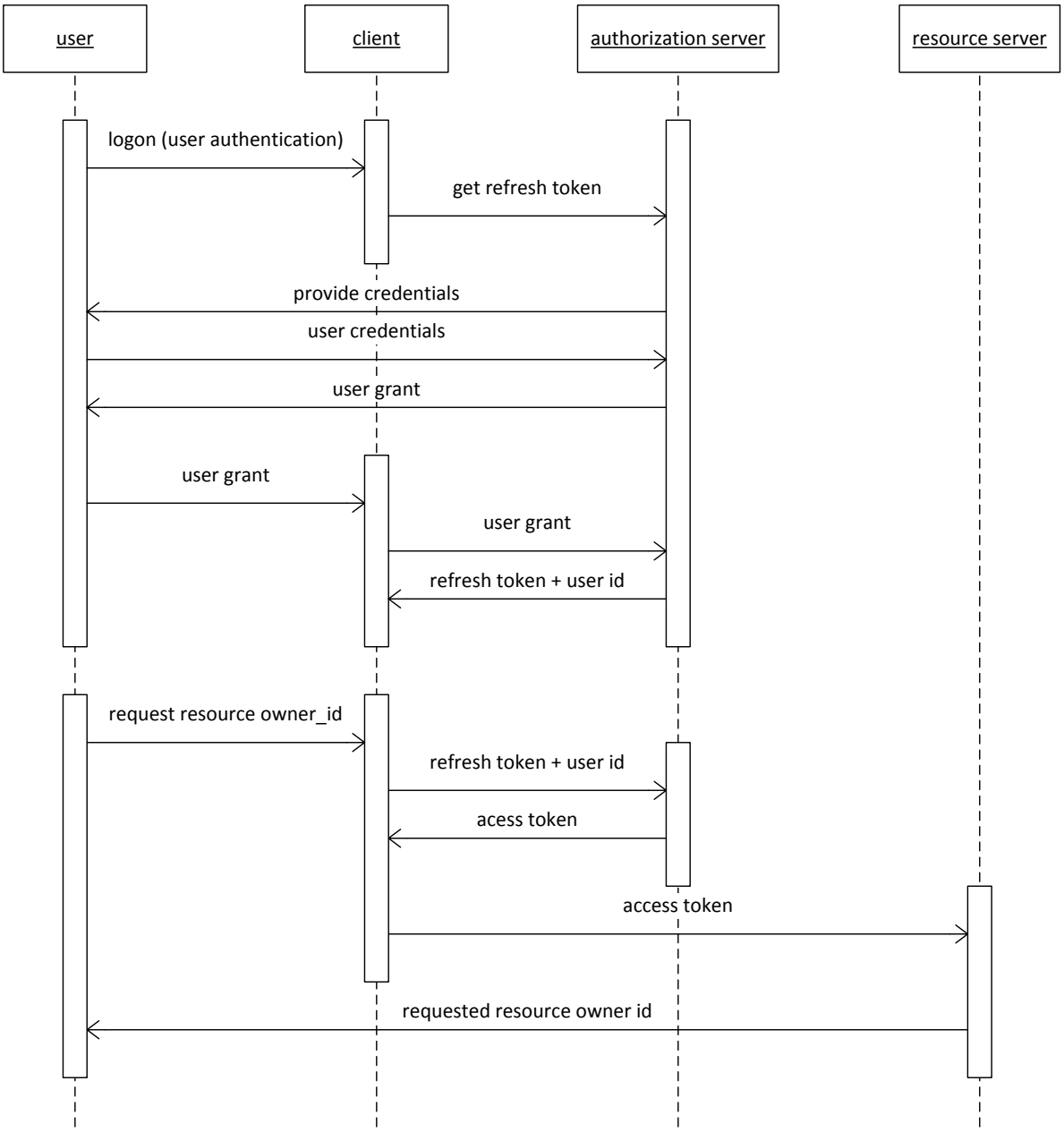

Figure 15: Authorized user protocol flow

3.6 Implementation

The extensions described in the previous chapters are implemented by extending the OAuth2.0 implementation of the Spring Security package (http://projects.spring.io/spring-security-oauth/). The actual implementation can be found in the archive of the trustworthy healthcare platform.

Chapter 4

User manual

Chapter Author:

Marco Abitabile (FCSR)

In this chapter we describe the main functionalities of the home healthcare platform and application from a user perspective.

4.1 General Description

4.1.1 Installation

4.1.1.1 The easy way

Just download TPaaS Virtual Machines and deploy them into a cloud environment

Appliance VM requirements:

- Min 4GB Ram
- Min 4CPUs
- Min 20GB HDD

PHR/EHRs VM requirements:

- Min 512MB Ram
- Min 1CPU
- Min 20GB HDD

4.1.1.2 The Hard way

Installing the healthcare platform requires many subcomponents and software to be installed. We will provide a brief description of the steps needed to make the system work.

Preparation of PHR/EHRs Virtual Machines:

First of all, it is necessary to setup the VMs that acts as remote databases for PHR and HER data.

Requirements are:

- Min 512MB Ram
- Min 1CPU

- Min 20GB HDD
- Linux Debian Wheezy
- Mysql

Once installed, provide the VM with a specific IP address and write it down for later use.

Preparation of the Appliance Virtual Machine:

Appliance VM is the one that have all the platform logic. In order to make it run it is necessary to install the following software:

- Linux Debian Wheezy
- Java 7
- Tomcat-Liferay Bundle
- Mysql
- TClouds DepSky subcomponent client
- TPaaS Healthcare Java code to be deployed into the Tomcat-Liferay bundle
- Libvirt package installed
- TClouds SAVE client

DepSky code, SAVE code and Healthcare platform code can be retrieved by the respective TClouds partner and/or downloaded from http://svn.tclouds-project.eu

Once everything is in place it is necessary to change the IP addresses written into the following config files:

Starting from the Tomvat-Liferay bundle: $TOMCAT_BUNDLE/webapps/TCloud-portlet/WEB-INF/classes/

/application/properties/application.properties

/log/properties/auditmanager.properties

/log/properties/remoteseclog.properties

/log/properties/dbsessionmanager.properties

/log/properties/seclog.properties

/log/properties/plainlog.properties

/log/properties/seclogconstants.properties

/log/properties/plainlogVM.properties

/audit/auditmanager.properties

/audit/dbsessionmanager.properties

/audit/plainlog.properties

/com/tcloud/portlet/configuration/properties/configuration.properties

Configuration files has to be changed in order to connect to the remote databases, and to the remote TClouds LogService

4.1.2 Login / Sig-in

TPaaS Healthcare platform is available online at https://tpaas.eservices4life.org. DNS points to TClouds Infrastructure, as we can see in the images below:

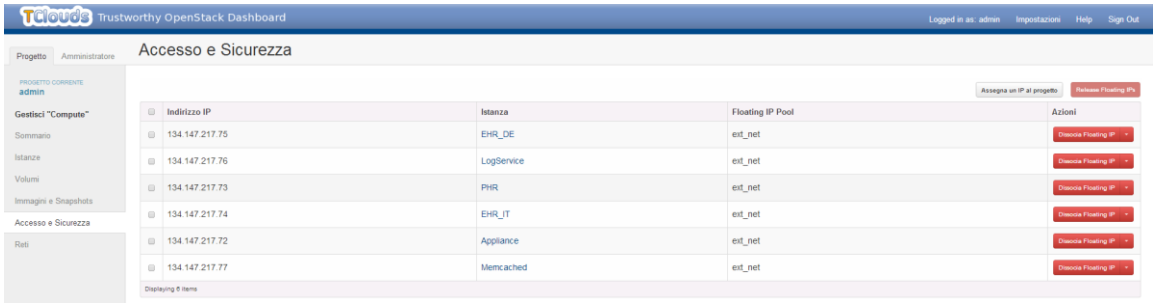

Figure 16: IP bounding with Healthcare appliance on TClouds TrustworthyOpestack dashboard

```
root@core ~# ping tpaas.eservices4life.org
PING tpaas.eservices4life.org (134.147.217.72) 56(84) bytes of data.
64 bytes from comp72.rubtrust.ruhr-uni-bochum.de (134.147.217.72): icmp_seq=1 ttl=63 time=2.51 ms
64 bytes from comp72.rubtrust.ruhr-uni-bochum.de (134.147.217.72): icmp_seq=2 ttl=63 time=2.10 ms
64 bytes from comp72.rubtrust.ruhr-uni-bochum.de (134.147.217.72): icmp_seq=3 ttl=63 time=2.17 ms
64 bytes from comp72.rubtrust.ruhr-uni-bochum.de (134.147.217.72): icmp_seq=4 ttl=63 time=1.88 ms
^C
--- tpaas.eservices4life.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.882/2.169/2.517/0.230 ms
root@core ~#
```

Figure 17: Ping of tpaas.eservices4life.org

eservices4life.org domain is a domain owned by San Raffaele and refers to the IRIS research unit that whose researcher cooperates within TClouds consortium.

In order to login into the platform is necessary to be registered. You can freely chose to register yourself or use one of the pre-registered demo users:

NOTE: password is “12345” for all users

username	patient	developer	administrator	Doctor
tpaas_admin@eservices4life.org	X	X	X	
john.smith@eservices4life.org	X			X
katherine.nolan@eservices4life.org	X			
patrick.burrows@eservices4life.org	X			

Table 2: Demo users saved into the platform

Following is depicted the relationship among users:

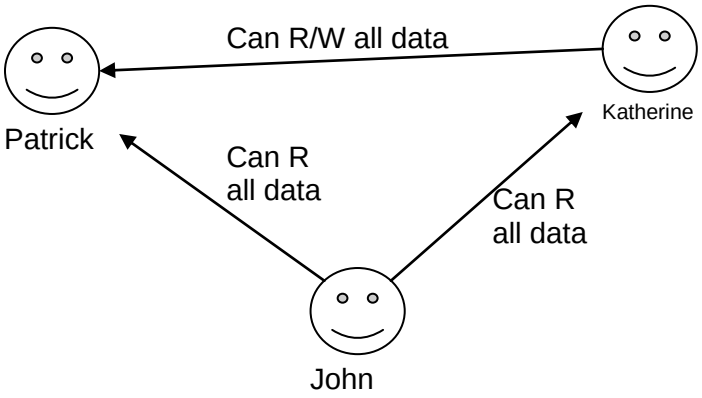

Figure 18: Relationship among demo users into TPaaS

The idea is to reproduce a very simple relationship between Katherine and his son, Patrick (for this reason Katherine has all privileges on Patrick's data, being his guardian). John, instead, is a Doctor and has Read abilities on Patrick and Katherine's data.

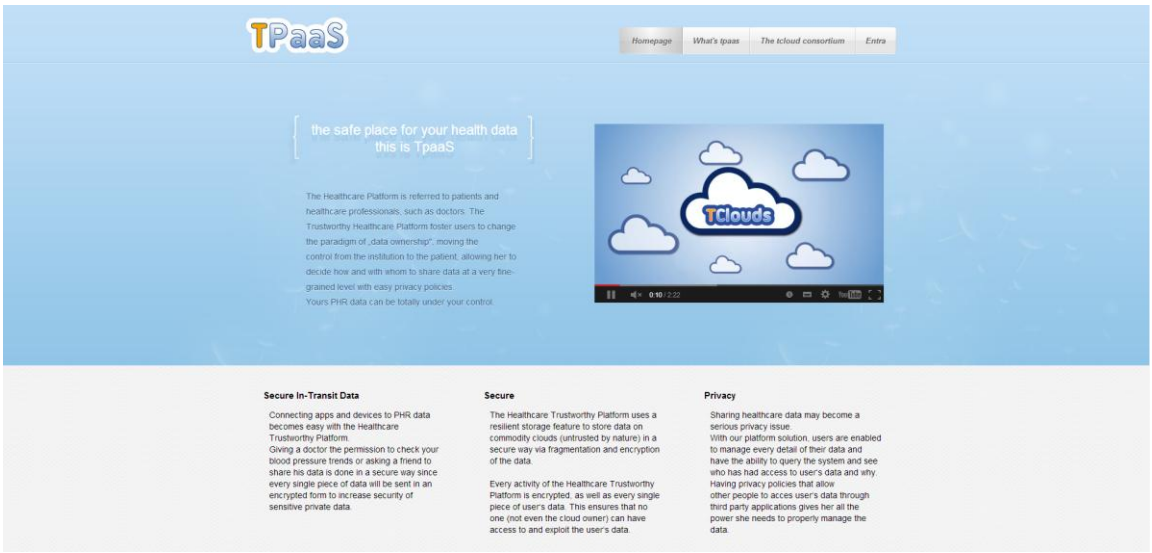

Figure 19: TPaaS homepage

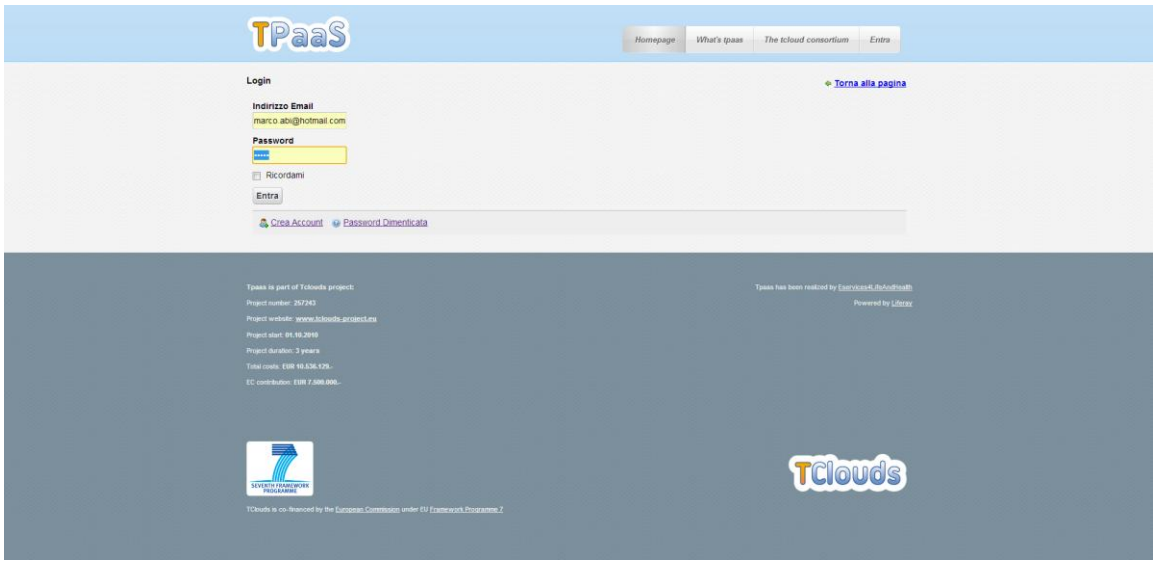

Figure 20: Login

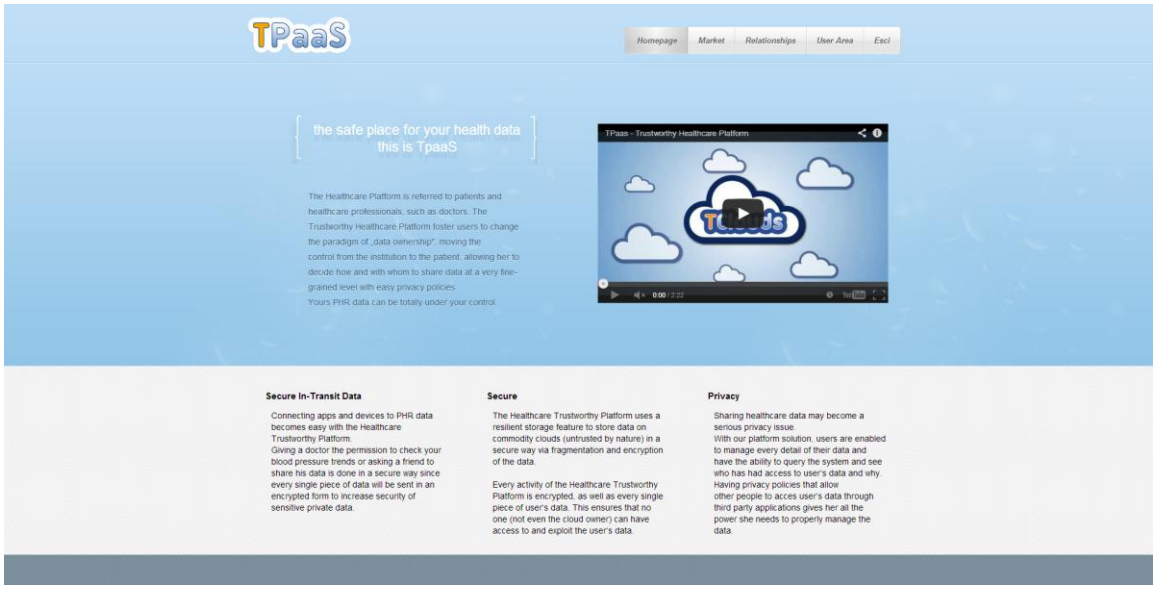

Figure 21: Logged in

4.1.3 App Market

Every user can access to the app market. Here he/she can see:

- Whether an application still owns an Oauth2Share access token, providing the ability to remove it
- Add/remove grants to applications

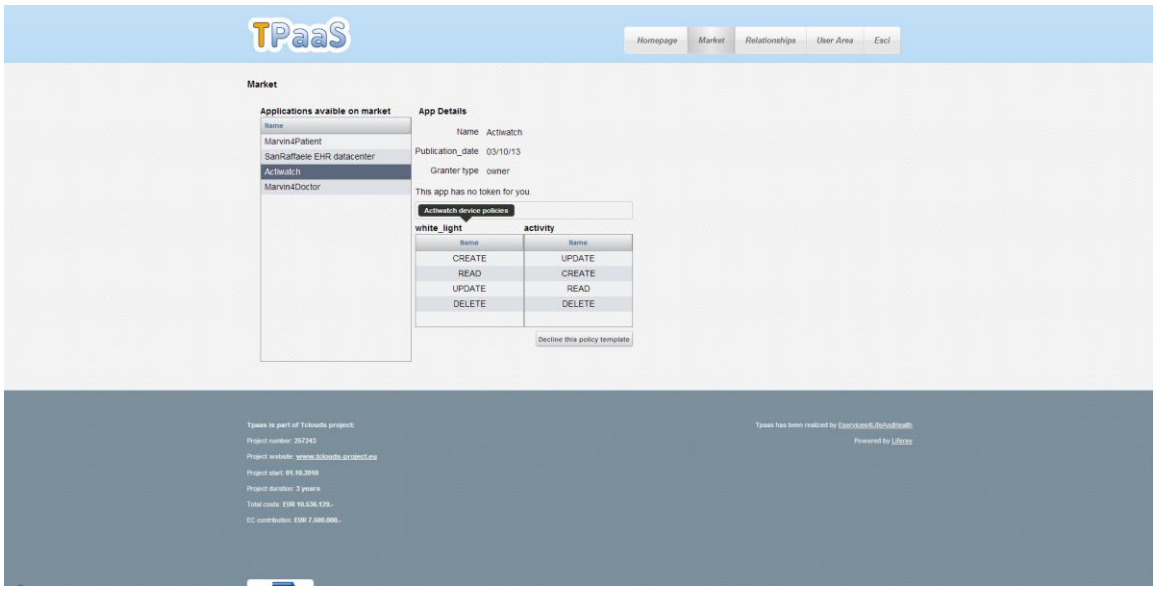

Figure 22: App market

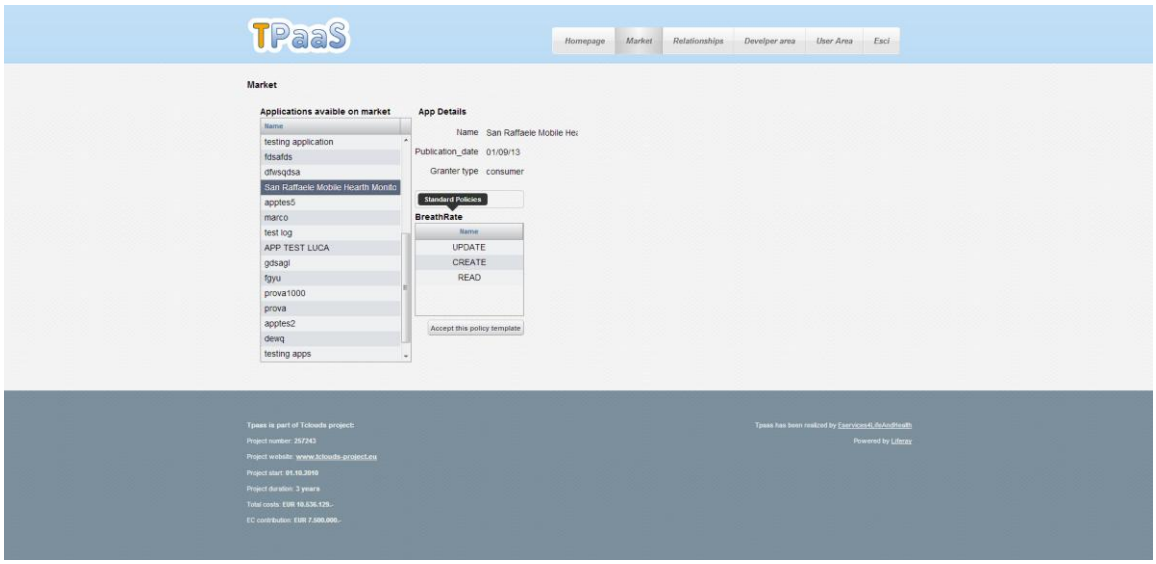

Figure 23: Add a new app and accept policies

4.1.4 Adding new friends

From the “Relationship” tab is possible to see all the users that are registered into the platform and create grants to them. You can also see who is giving you grants on their data.

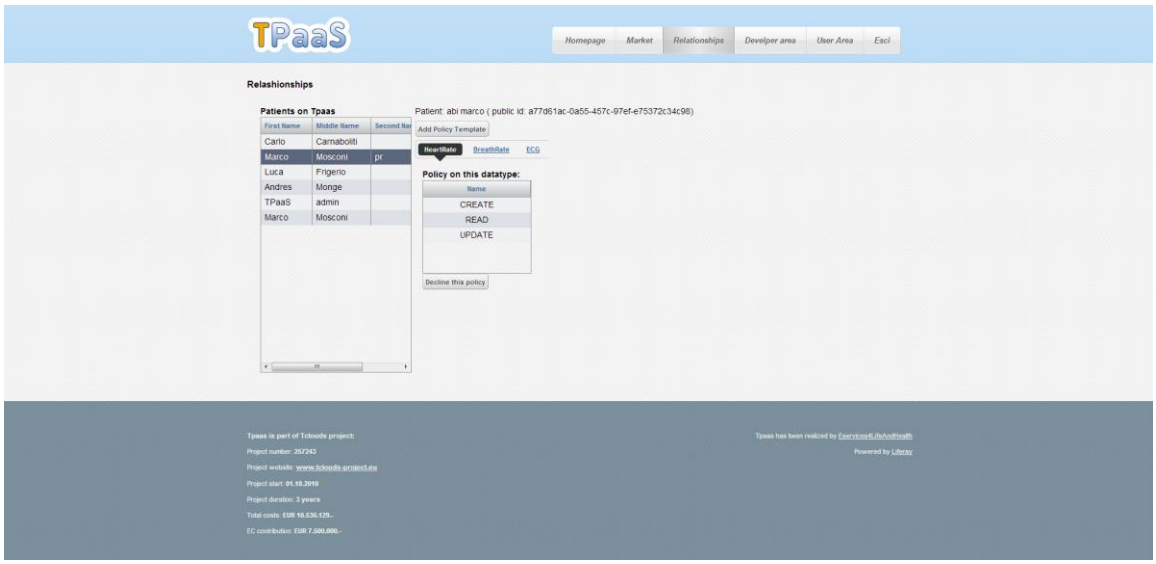

Figure 24: Friends list and details

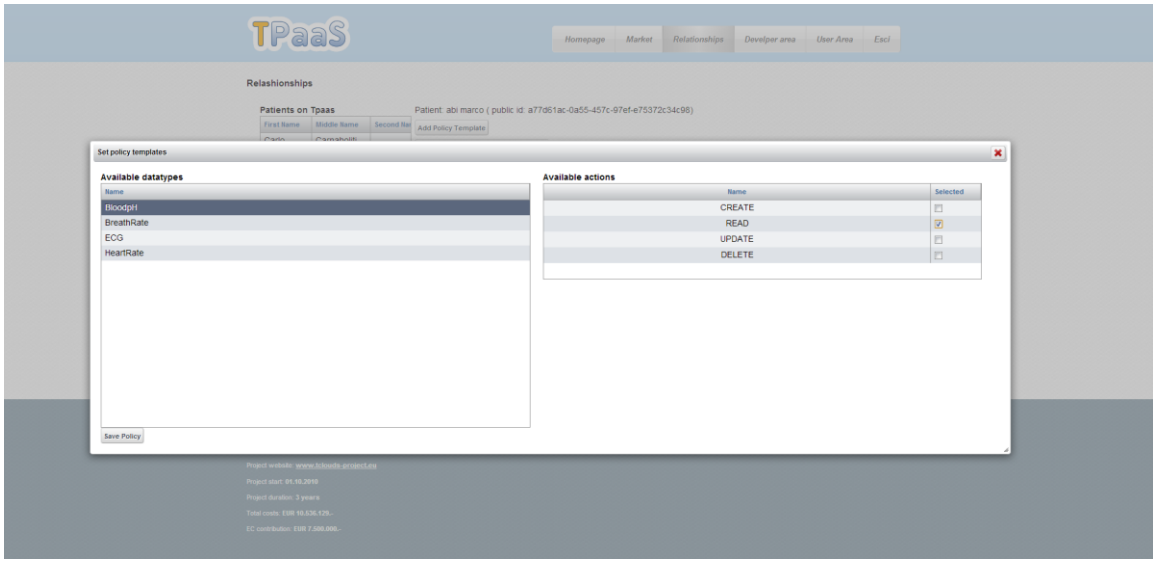

Figure 25: Add new policy to an user

4.1.5 Developer Area

Users that ask to become developers can access al to the “developer” area. In this area you can register a new app (that will appear into the app market) and create policy templates that users have to accept in order to allow the app to access data.

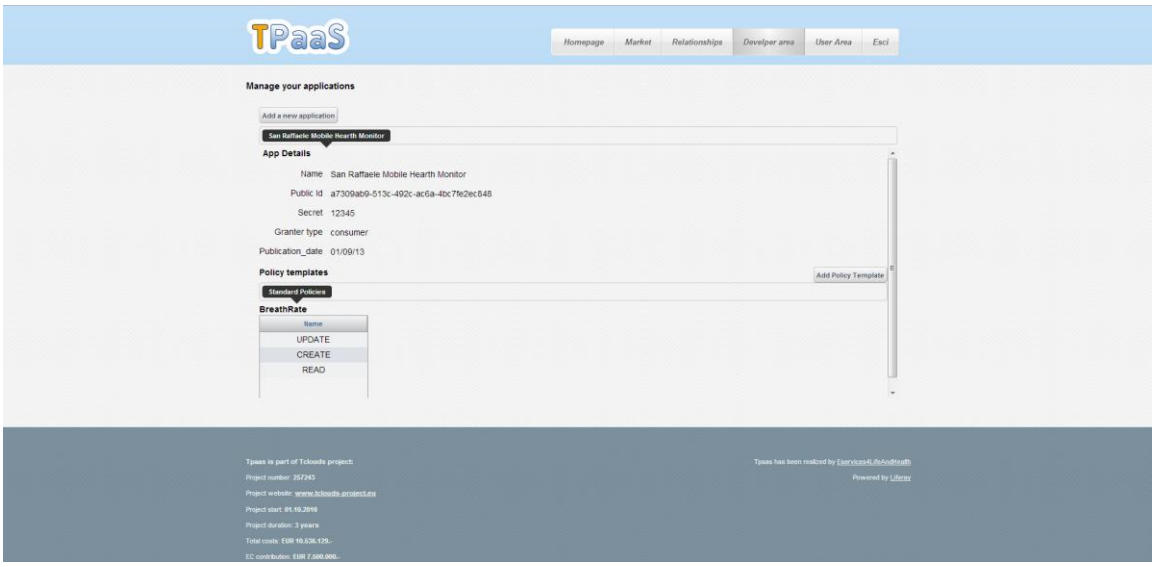

Figure 26: Developer area

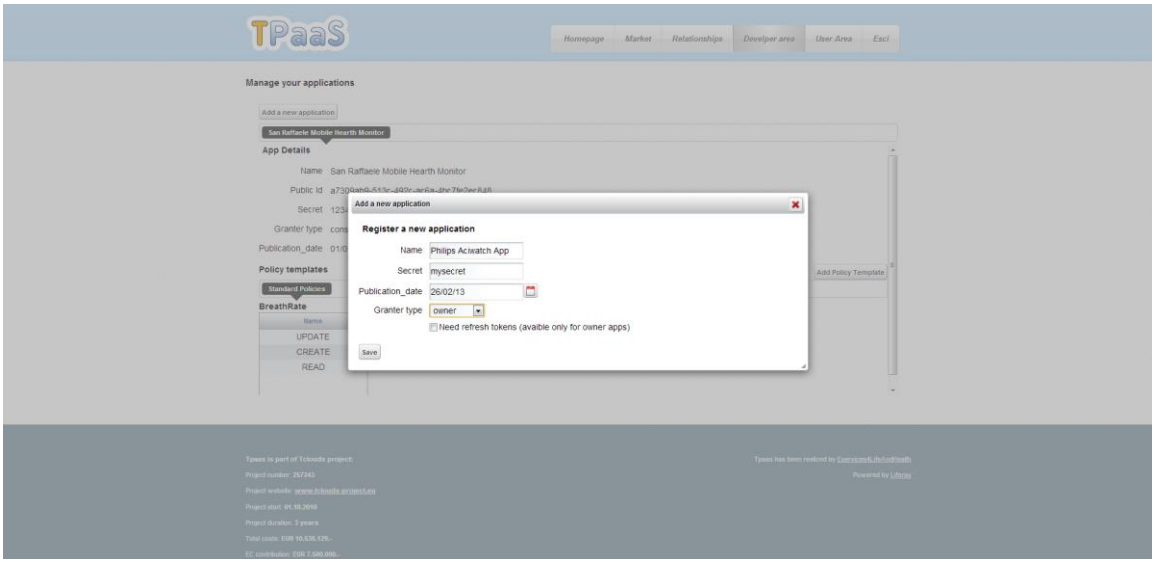

Figure 27: Create a new App

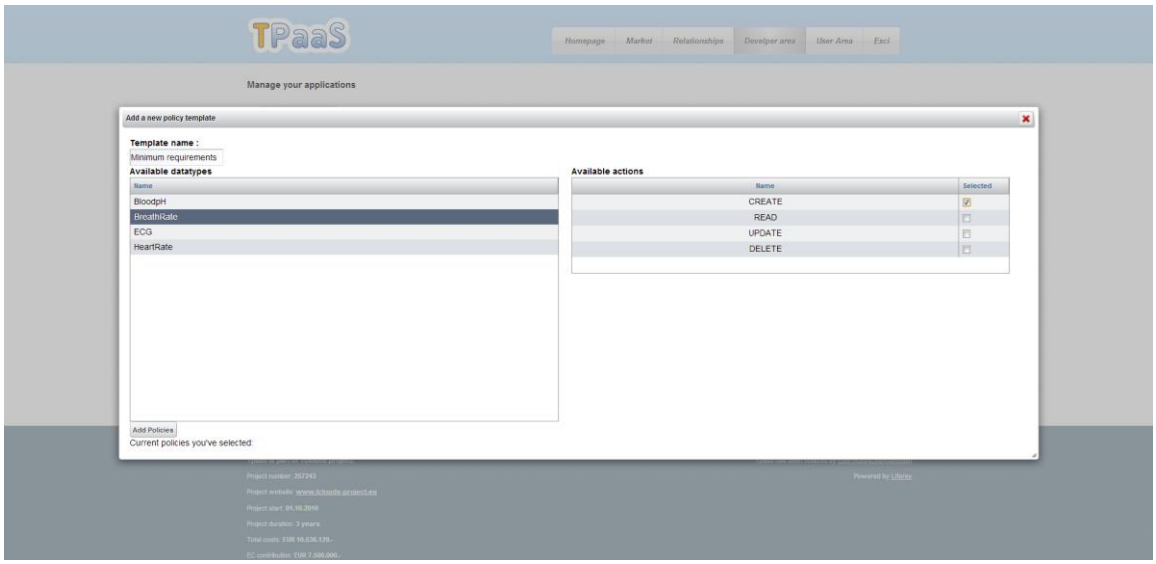

Figure 28: Add minimum policy requirements to the new app

4.1.6 User Area

The user area provides users with the ability to see some information about their developer activity (whether the user is a developer), and provides audit features, to know who is using his data and how. This feature access and uses TClouds LogService subcomponent, in fact, is possible to perform validation checks on the logged data.

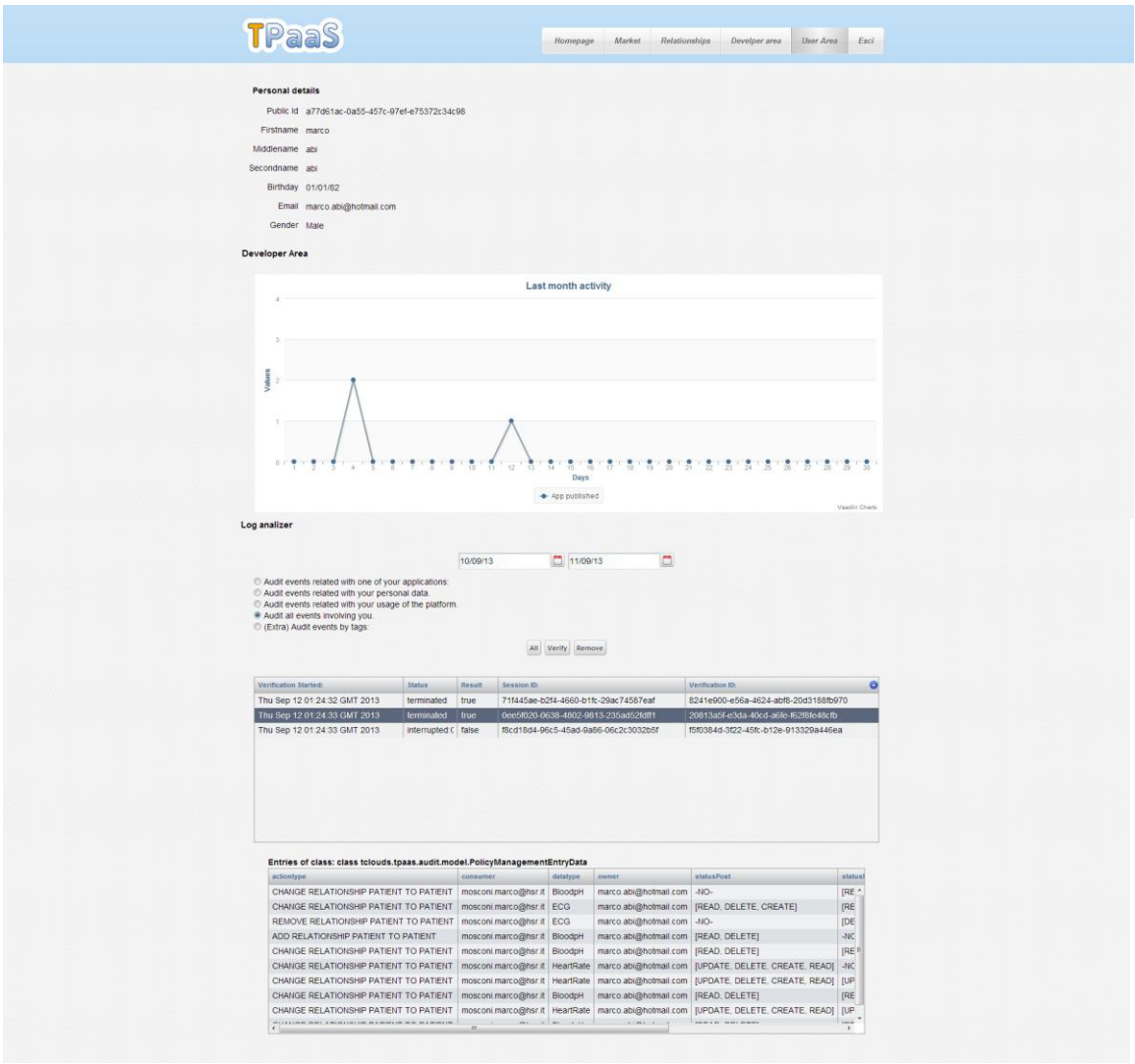

Figure 29: User area and audit

4.1.7 Admin area

The platform administrator has some extra features not available to other user type:

- DB backup by using DepSky TClouds subcomponents
- Extended log audit features over all the activities performed by all the users into the platform by using TClouds LogService subcomponent
- TClouds SAVE deployment check

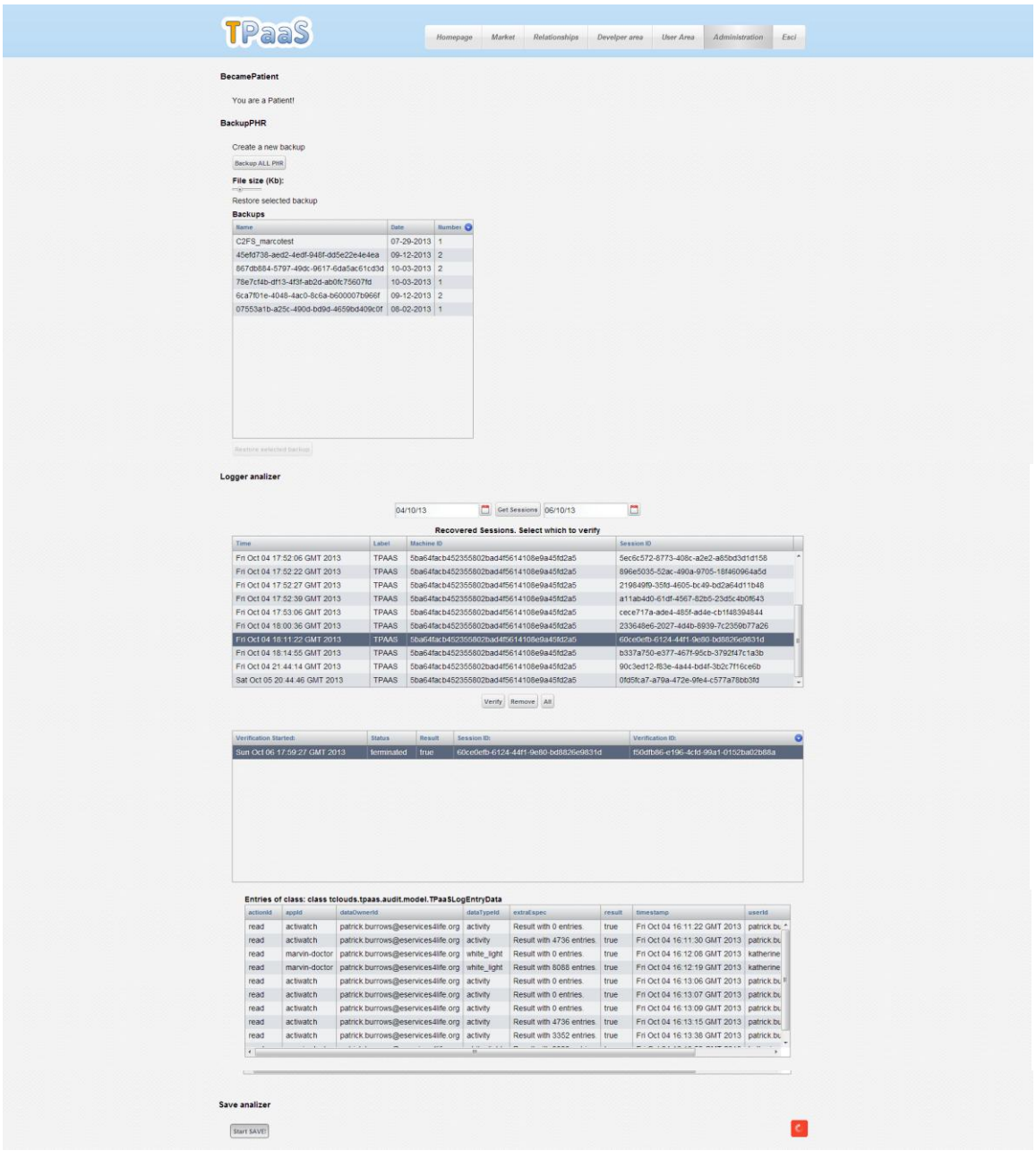

Figure 30: Admin area, auditing on user activities

4.1.8 Third Party applications

In order to show potentialities of the Healthcare Platform we developed three third party applications. They are “owned” by tpaas_admin@eservices4life.org user (thus you can discover more information on these app my accessing the developer area of this user).

The three applications are:

- Actigraphs: allows users to upload Philips Actiwatch data and show that data into graphs. Actigraphs application can be used directly by accessing to: http://actiwatch.eservices4life.org and by using the related Android device.

- Marvin 2: it consists of porting of Y1 Marvin application. Marvin has been developed with the idea to develop a portal for patients and doctors to monitor patient-generated data related with depression (among this data there is Actiwatch light data). Marvin 2 is the porting of the doctor features and allows doctors to access patients' data collected by different devices into the Healthcare platform. Marvin2 web app can be used directly by accessing to: http://marvin2.eservices4life.org:8080
- DocMob android App: it represents a demo application that is used by doctors to store and retrieve users' data.

Chapter 5

Conclusion

Chapter Author:

Mina Deng (PHI)

In TClouds, Activity 3 has specified cloud applications, functionalities and requirements, has implemented application prototypes on the cloud platform, and has validated the application prototypes and the TClouds platform. In this document we describe the proof of concept for home Healthcare Trusted Platform as a Service (TPaaS), which is the use case discussed in work package 3.1.

The Healthcare TPaaS is deployed on TClouds Trustworthy OpenStack, integrated with effort from A2. The logging/auditing service has been deployed on TClouds as well. Furthermore, the user interface allows a user to manage the third party applications. Finally, several third party applications of medical use cases have been developed, which can access users data under their privacy control.

The Healthcare TPaaS aims to be a platform that runs in the cloud in a secure and trustworthy fashion, allowing third party healthcare applications to use it and benefit from the service it provides.

The idea is to build a trusted platform to manage health records on top of TClouds security components. In particular, the TPaaS platform is deployed on top if the trustworthy OpenStack provided by TClouds A2.

The Healthcare TPaaS is a multilevel platform that aims to provide novel services such as:

- Store trustworthy health-related data (relying on a trusted IaaS);
- Provide API for 3rd party apps to access to users' health data, in a privacy-preserving manner (PaaS);
- Provide API for 3rd party apps to use identity/role management services (PaaS);
- Provide an interface to allow 3rd party app developers to register their application(s) in order to access the users' data (SaaS);
- Allow End Users to manage their data and specify privacy policy about which data a particular application/ user can access (SaaS).

TPaaS is built in layer abstractions. In this chapter we will describe each of them.

- JPA layer: this is the lowest layer abstraction. It manages all the persistency with the databases. It is able to store health data in a specific database depending on the user policies and geo-location; moreover it manages all the transactions with the persistence unit and ensures that each task submitted to it has an atomic nature, allowing highly parallelism and integrity of data.
- TPaaS domain implements all the entities involved into the system, from users, to applications to relationships among them (including privacy policies). The domain layer "knows" all the actors involved and offers a set of interfaces that allows upper layers to use the domain properly. The domain implements all the main security features, from policy change to relationship management. This

means that any other layer needs to “ask” the domain layer in order to persist or modify a policy.

- Application layer, instead, acts as a use case container, in which every single use case in the platform is implemented and satisfied.
- Log Service is an independent module used to store log information in the remote Log Service
- User interface consists in specific view to allow user to access the platform and their user area
- Oauth2Share: is the trust protocol that enforces authorization and access control between application and end user.
- The Healthcare Platform consists in 4 Virtual Machines to be deployed on the TClouds trustworthy Openstack:
- Appliance: it is the main VM. In here resides all the logic of the platform. It manages platform users, their relations, UI and all the business logic to work with DepSky, LogService and Tailored Memecached sub-component.
- PHR VM: it is a virtual machine that stores all the PHR data.
- EHR_IT VM: it stores all the EHR data of legal entities that resides in Italy (e.g. an Italian hospital).
- EHR_DE VM: it stores all the EHR data of legal entities that resides in Germany (e.g. a German hospital).

The generic trust protocols describe the trust protocols used at the application interface of the trustworthy healthcare platform. The goal of the trust protocols is to enforce the authorization rights given by a user to another user or an app with respect to accessing his or hers PHR/EHR data, and to do this in a REST full manner.

In order to define the trust protocols, four use cases are discussed. In the normal use case the data owner wants access to his or her data. In this case no delegation of rights is needed; the owner can authorize the access. An independent device is a device (including an app) that can access data without direct owner control. In the authenticated user use case the data owner gives consent to an app to access his or her data for any user subscribed to the app. In the authorized user use case, the data owner explicitly gives another user the right to access his or her data.

Based on the use cases, the relations between the various components are defined. Privacy rights as delegated to the apps or other users in the “delegates to” relation are stored in the privacy policies database. When an app wants to retrieve information from the PHR/EHR database, it first has to check the privacy policies on whether or not this is allowed. Creating the “delegates to” relations and putting them into the privacy policies database is not part of the trust protocols, this is done by the UI part of the trustworthy healthcare platform. The trust protocols ensure that this policy is enforced.

Trust protocols is designed and developed based on and extends the general idea behind OAuth2.0. the normal and independent device use cases can be handled by the protocols of standard OAuth2.0. To support the authenticated user and authorized user use cases, the OAuth2.0 protocol is extended.

The distribution of the proof of concept of the Healthcare TPaaS is not provided in the form of binary files due to the complex installation. Instead, this platform is accessible online. With the graphical user interface of this platform, a user can create his profile, log in this platform, add his friends and audit the behaviors of third party applications. The Platform is accessible from the web at https://tpaas.eservices4life.org.

Bibliography

Evans, E. (2003). *Domain Driven Design.* Addison Wesley.