TICE.Healthy Framework for Developing an Ecosystem of Applications in the Domain of Informal Health

Pedro Catré, Jorge Dias
Faculty of Sciences and Technology
University of Coimbra
Coimbra, Portugal
catre@student.dei.uc.pt, jorge@deec.uc.pt

João Quintas, Alcides Marques
IPN Laboratory of Automatic and Systems and Laboratory of Informatics and Systems
Instituto Pedro Nunes
Coimbra, Portugal
jquintas@ipn.pt, alcides.marques@ipn.pt

***Abstract*— This paper presents a platform being developed in a large scale national project that aims to create an ecosystem of services and applications where patients, relatives and healthcare professionals can cooperate in day-to-day activities. It will implement functionalities for managing medical, social and context information, for integrating mobile devices, remote control and sensors, with insurance of information security.**

Keywords-eHealth; informal healthcare; service-oriented architecture; REST.

I. INTRODUCTION

TICE.Healthy – Health and Quality of Life Systems – is a research and development project that represents a collaborative effort to design and deploy innovative products and applications in the fields of eHealth and Ageing-Well. The main goal of this project is to create a web-based platform, named eVida, designed to ensure a set of services/features to support the creation of new products, processes, services and applications [1], [2].

From a global perspective, the platform being developed plays the central role within the project TICE.Healthy. Its primary objectives are data acquisition and processing from various collaborating entities (data providers – supply side) so that it can be shared and used by service providers. The eVida website can be consulted at [3], and a promotional video demonstrating some of its features is available at [4].

It is true that we are now witnessing a proliferation of eHealth platforms [5]; however, they are mainly focused on offering a single service. eVida's innovation comes precisely from breaking free of this paradigm by allowing the creation of an ecosystem of applications that can provide complementing capabilities and whose value can potentially supersede the sum of the benefits delivered by each individual application.

This paper presents the architectural approach and a brief description of the main modules comprising the eVida platform.

The paper is organized as follows. Section II presents an analysis of related work. Section III gives an overview of the platform's architecture. Section IV presents the main modules that make up the platform. Section V describes the Web Portal, more specifically: its purpose, the main features it provides for developers, the types of applications that are supported in the marketplace and a high-level view of its architecture. Section VI provides conclusions for this paper.

II. STATE OF THE ART

This section analyses current market solutions that present comparable functionalities to the ones required by the eVida platform. Although this analysis includes eHealth platforms it does not focus exclusively on that use case.

A. Commercial Web Portals

Google Health has been permanently discontinued since it did not reach the initially expected product adoption. However, it is still relevant since it was one of the primary commercial platforms available providing personal health records. This platform allowed users to persist, manage and share the user's health data with other users. It also provided an API allowing the users to share data with external applications that could provide several services like scheduling appointments, processing health data, among others [6].

Microsoft HealthVault is an XML-over-HTTP web service exposing a set of XML-based methods that developers can leverage to build applications that connect to HealthVault. Also, a number of SDKs are available that deliver platform-specific abstractions for working with HealthVault [7]. It is worth noting that this platform is only available in the United States of America and the United Kingdom. Although Microsoft Health Vault supports an ecosystem of other platforms and partner services to leverage the user's data, assuming the required permissions over the data were granted, it lacks many important features. Specifically, this platform does not integrate web applications in a single place and it does not provide the final user with a transparent and singular experience while using the platform. In other words, HealthVault does not support embedded applications, meaning each application runs independently, outside of HealthVault and no mechanisms are provided for inter-app communication. Furthermore, it does not provide Single-Sign-On [8], which is a facility through which a user logs in once and gains access to all systems without being prompted to log in again at each of them. In fact, in order to start using a new application the user is taken to the service provider's website where he

 ISBN: 978-1-61208-314-8

needs to register, sign-in and link the new account to the HealthVault account.

iGoogle is a service provided by Google that consists of a web portal that supports web feeds and web gadgets. The gadgets are defined through an open custom specification called Google Gadgets [9]. Both the development and use of the applications is open to the public [10]. It works similarly to other portals such as My Yahoo!, Netvibes and Pageflakes [11].

Facebook Apps consist of applications hosted in an external server that are accessed in Facebook in a page called Canvas. Together, they provide functionality through APIs, such as the Social Channels API that includes bookmarks, notifications, News Feed stories and search [12]. Facebook also offers an authentication system, called Facebook Connect, as an alternative mechanism to the registration of a user. In other words, Facebook Connect allows external applications to support Facebook accounts [13].

LinkedIn supports applications that follow the OpenSocial specification [14]. These applications can be added to the user's homepage and to his personal profile [15].

Twitter does not support applications in the same way as the previous examples. It offers a REST API that enables third-party applications to interact with the majority of the web site's functionalities [16]. It also provides an authorization and authentication mechanism that can be used by external applications (similarly to Facebook Connect). To that end, Twitter employs OAuth. The permissions that a user can grant are granular enough that he can limit the access to specific data and communication on his behalf.

Chrome Web Store allows the specification of applications published as a compressed archive composed of a manifest, HTML, CSS, and JavaScript files [17] or as a pointer to an external server [18]. Neither type of application is embedded in the Chrome Web Store. Also, the mechanism for executing the applications is not interoperable; it's restricted to the Chrome browser.

Podio is a web system that supports collaborative work for companies and provides extensibility through external web applications. It allows the discovery and purchase of applications similarly to the Chrome Web Store [19].

Table I presents a comparison of the functionalities of the various commercial projects that were presented.

TABLE I. COMPARISON OF THE FUNCTIONALITIES OF THE COMMERCIAL PROJECTS THAT WERE ANALYSED

	Embedded apps	**Single sign-on**	**Granular Permissions**	**Consistent UI**	**Cross-browser**	**Supports web apps**	**Supports native mobile apps**
Google Health	✗	✗	✗	✗	✓	✓	✓
Health Vault	✗	✗	✓	✗	✓	✓	✓
iGoogle	✓	✓	✗	✗	✓	✓	✗
Facebook Apps	✓	✓	✓	✓	✓	✓	✓
LinkedIn Apps	✓	✓	✓	✓	✓	✓	✗
Twitter Apps	✗	✓	✓	✗	✓	✓	✓
Chrome Web Store	✗	✗	✓	✗	✗	✓	✗
Podio App Store	✓	✓	✗	✓	✓	✓	✓
eVida	✓	✓	✓	✓	✓	✓	✓

eVida already supports the most common features provided by major web portals. However, it is an ongoing project with many required functionalities still in development, such as a payment system for the marketplace and a fully featured interoperability module.

III. PLATFORM ARCHITECTURE OVERVIEW

The platform's architecture is based on the design principles of service-oriented architectures (SOA), because of its distributed and extensible nature. This approach increases the abstraction and encapsulation within the system. The platform exposes a set of Representational State Transfer (REST) services as the communication mechanism with the data and service providers. Fig. 1 illustrates the high-level architecture of the platform. Three main components can be easily identified, which implement the information and interaction channels for selling products and health services: the database stores all the administrative data, and configuration data of the platform; the backend services provide security, interoperability and information services; and the frontend is comprised by the platform's web portal, which exposes the tools to submit and publish applications within the platform's ecosystem.

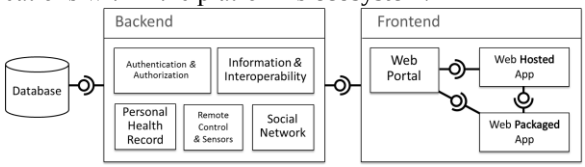

Figure 1. The platform's high-level architecture.

This platform implements functionalities to manage medical, social and context information, for integrating mobile devices, remote control and sensors, with insurance of information security.

IV. PLATFORM MODULES

This section will briefly present the modules that make up the platform.

 ISBN: 978-1-61208-314-8

A. Mobile Devices, Remote Control and Sensors

The purpose of this module is to simplify the development and integration of new and innovative services for the provision of health care and communication. This kind of services is often referred to as home care in assisted environments and domotics (e.g., measuring biometric signals and leveraging environmental sensors). This module is also responsible for accessing services deployed in eVida with mobile devices, following the technological trends of accessing Internet services through smartphones and tablets.

The platform includes mechanisms for collecting and transmitting data in real-time. For this purpose it provides a Jabber/Extensible Messaging and Presence Protocol (XMPP) [20] interface for gathering and sharing data on a continuous basis. An example of usage of this infrastructure is the real-time acquisition of sensor data.

B. Security

The security module aims to ensure that applications and services that work on the platform are reliable in terms of confidentiality, integrity and availability of information. These three aspects are essential to ensure an acceptable level of security and privacy of the user's personal information and data stored in the platform. This module's components will cover all logical layers of the platform, from the access control of users and systems, to the use of encryption in network protocols. Specifically, this module will be responsible for the *authentication* – by verifying the identity of the applications that interact with the platform using the OAuth 1.0 or OAuth 2.0 protocol (eVida provides both) – and *authorization* – by verifying data access permissions.

The *permissions* module that was developed is flexible due to the dynamic nature of our health data repository. In the current integration it allows platform administrators to:

- Create roles with (Create, Read, Update and Delete) permissions over archetypes, records and the personal health record viewer application's menus.
- Define permissions directly associated with users and associate them with roles.

In the personal health data viewer application the user can:

- Create and manage groups to whom he associates permissions and users.
- Access other users' personal health data with proper authorization, limited to the permissions provided by his roles and by the groups to which the data owner has associated him with.

We chose to distinguish between role and group (this concepts are not consistent across the literature [21]) and implement both. In our system groups are a convenient method for users to associate a name to a set of subjects and permissions and use this name for access control. On the other hand, roles are defined at the system level, by the platform managers, as a collection of privileges required to perform specific actions in the system.

The logical diagram of the permissions module is presented in Fig. 2.

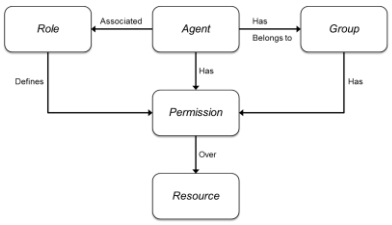

Figure 2. Logical diagram of the permissions module.

An agent can have several permissions over the same resource defined in different ways. For example, he can be associated to a profile and be part of a group that both define permissions over the resource.

The permissions are defined as capabilities [22]. A capability consists of an object the user must have in order to execute a specific action over a resource. Each capability has a resource id and operations a user can execute over a resource. Fig. 3 illustrates this concept.

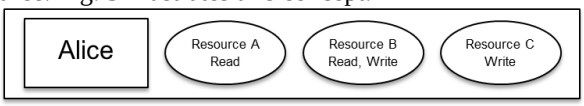

Figure 3. Conceptual representation of the use of capabilities in the permissions module.

Alice can execute a Read operation over resource A, but she cannot execute a write operation over the same resource. To perform this validation the module uses binary masks, the same mechanism used by the Unix file system [23]. This technique consists of assigning a sequential number that is a power of 2 to each type of action (permission). This way, the set of permissions a user has over a resource is given by the sum of all individual permissions. For example, if we assign the following numbers to the actions:

- Read: 1 (2^0)
- Write: 2 (2^1)
- Edit: 4 (2^2)

If an agent has Read and Write permissions over a resource, then the set of permissions is given by the disjunction of all the individual permissions (1):

$$\text{set} = \text{value(Read)} \text{ OR } \text{value(Write)} = 3 \quad (1)$$

This technique enables efficient validations, since they become basic logic operations like the one presented above. It is also memory efficient, given that it only requires a 64 bits Long to store all the permissions a user has over a resource.

C. Information and Interoperability

The information and interoperability module facilitates the presentation of information to a large number of services that wish to receive it in a particular format. Specifically, there will be a medical information component, designed to work with the Health Level 7 (HL7) versions 2.x and 3.0 and Digital Imaging and Communications in Medicine (DICOM) standards to provide a comprehensive solution to

the needs of professional caregivers and a connection to other legacy information systems. The components comprising this module allow the development of new services and products that can communicate using the same syntax and semantics.

D. Personal Health Record

A Personal Health Record is a repository of clinical information of an individual whose maintenance and updating can be performed by himself or by his caregivers [5]. This module is responsible for storing the platform's clinical data, which is made available through a REST application programming interface (API). It also presents a user interface where personal health data can be consulted in a secure and private manner. The users will be able to share parts of this data with family, friends, caregivers and service providers.

The PHR is created over an application builder [24] that was designed under the TICE.Healthy initiative. Using this component it is possible to create related business/clinical entities, design forms and views and deploy it as a full application that can be dynamically extended and changed without the need for a redeploy.

Fig. 4 presents the application builder and the PHR.

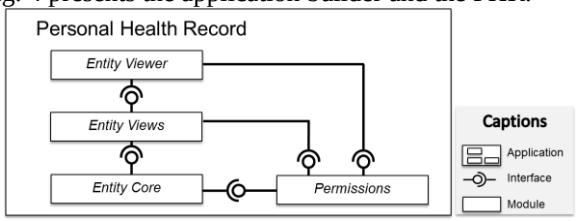

Figure 4. Static diagram of the Personal Health Record's Modules.

Fig. 4 presents four modules that serve as a foundation for the PHR application:

- The permissions module, that was previously presented, handles the system's authorization.
- The Entity Core is the layer that handles the configurable clinical data repository. It provides an internal API with business logic that enables abstraction from the dynamic nature of the data structure. It also provides a REST API for data manipulation.
- Entity Views is responsible for interpreting the configuration of templates and entities/archetypes in order to generate interface elements. It uses Entity Core to manipulate the data and the Permissions Module to check authorizations for data manipulation.
- Entity Viewer presents a dynamic application (the PHR viewer) that can be configured through an administration panel. This high-level module uses the previously described modules, allowing platform administrators to create and manage entities and templates, create application menus, define their content and manage permissions.

V. WEB PORTAL

The platform's frontend includes a marketplace for applications developed by third parties (including applications developed under the scope of TICE.Healthy).

This Web Portal aggregates and integrates applications and offers the final user a unified interface. Currently, the platform provides mechanisms so:

- Developers can add and manage applications in the platform.
- End users can use these applications in the platform without the need to register and sign-in to each of the applications, through a Single-Sign-On mechanism.
- Applications can make use of the platform's REST API and user data, given the necessary permissions.
- Applications can make use of the JavaScript APIs for interacting with the portal, other applications and users.

Furthermore, the platform allows the publication of web applications that can take one of two formats: packaged apps and hosted apps. In addition to the web applications, the marketplace also offers support for mobile applications.

A. Packaged Apps

These applications execute entirely in the browser and their business logic is programmed in JavaScript. They can also make use of the new capabilities of browsers related to HyperText Markup Language 5 (HTML5), such as working in offline mode. These packaged applications consist of an archive that follows the World Wide Web Consortium (W3C) widgets specification. With this specification W3C intends to standardize the way client-side web applications are written, digitally signed, protected, compacted and deployed independently of the platform [25].

Both the portal's API and the web interface communicate with a widgets server. Because the definition of the packaged applications follows an open standard it was possible to choose an open-source implementation to manage them. Apache Wookie [26] was selected for that purpose.

The execution process of packaged apps is described in Fig. 5.

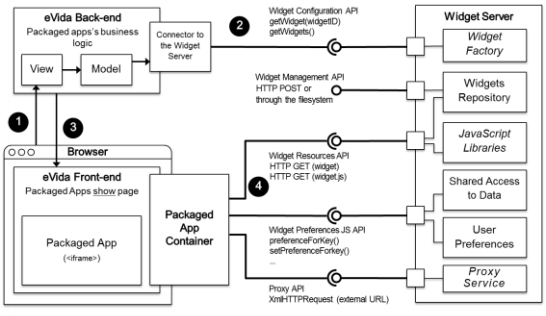

Figure 5. The execution process of packaged apps.

When accessing a packaged app (1) the corresponding view is executed which, in turn, will communicate through the model with the widgets server. The widgets server instantiates the widget and returns the application's URL. At that moment it is possible to present it to the user (3). While

it is being loaded, the resources required for the use of the JavaScript APIs are fetched (4).

Although most packaged applications are highly responsive, fast and interactive, they force the use of a particular programming model which might not always be adequate or desirable. In fact, programming the application entirely in JavaScript can be restrictive in certain contexts. To overcome this constraints the platform also supports hosted applications.

B. Hosted Apps

This type of application is remotely accessible and housed outside of the platform, being supported by its own servers.

Among the choices available for including hosted applications (such as inline JavaScript, content obtained through asynchronous JavaScript and XML – AJAX calls, iframes and script tags), we adopted the use of iframes with additional capabilities, which allow the URL of the portal to be updated as the user navigates in the application and make the application sensitive to events, such as resizing the screen and personalized messages. Also, the applications' content is downloaded asynchronously, thus not affecting the rendering of the portal page.

C. Comparison Between Hosted and Packaged Apps

Table II presents a comparison of the main characteristics of the formats that have been described.

TABLE II. COMPARISON BEWTEEN HOSTED AND PACKAGED APPLICATIONS

Criteria	Hosted App	Packaged App
Hosting	Responsibility of the developer	In the platform
Technological paradigm	Any language and technology	JavaScript executed in the browser
Server	Mandatory	Optional
Communication with the portal and other applications	Supported with limitations[a]	Supported
Data persistence	Responsibility of the developer	Supported

The communication is processed in a secure manner in recent browsers, however, the fallback that takes place in older browsers (Internet Explorer $\leqslant$ 7, Mozilla Firefox $\leqslant$ 2.0, Safari $\leqslant$ 3.2, Opera $\leqslant$ 9) is unsafe, so it cannot be used to transmit sensitive data.

It is also worth mentioning that, from the platform's perspective, it is better to have a packaged application since it provides a more seamless integration and, typically, a more responsive feel to its users. In terms of managing the applications it is also easier for administrators to control packaged applications because whenever a developer makes a change he needs to resubmit the application for approval. On the other hand, hosted applications can be changed without going through this process, thus requiring periodical verifications to make sure they still abide by the platform's terms and policies.

D. Isolated Environments

The portal's domain is different from the domains that serve both the hosted and packaged applications. This adds security since the browsers restrict the interaction between frames in which the domain, port or protocol differ. However, what is gained in security is lost in interaction capabilities. In the case of packaged apps that can execute JavaScript code using shared objects a different solution is required. In those cases, the Google Caja [27] web service is applied, which is a compiler for making third-party HTML, CSS and JavaScript safe for embedding, that follows the Object Capabilities [28] security model. This allows the isolation of the execution of the code so that the application integrated in the portal can only manipulate certain objects.

E. General Objectives of the JavaScript APIs

Each type of application (hosted and packaged) has its own technical specificities and associated scripts to guarantee the correct access to the APIs. In the case of hosted applications, the developer needs to import the script manually.

Presently, the developed APIs are:

- Inter-App Communication – offers mechanisms for communicating through a Publish-Subscribe model. A producer (application) can share data publicly (all users) or privately (across browser sessions of one user). Only private channels require user login. In both the public and private channels if the producer decides to share data he can specify which are the authorized receivers (based on their application IDs) or he can share the data with everyone (any application can become a subscriber). In the public channels the user (who may or may not be logged in) is prompted to authorize or reject the channel when the application tries to perform its registration. In the private channels the user can have his sessions initiated in multiple browsers and devices and he is prompted to authorize or reject the channel the application tries to subscribe.
- Remote Communication & Debugging – set of functions that facilitate the debugging process allowing asynchronous communication with external resources.
- Widget Properties – allows access to the properties that define the application, such as metadata included in the applications configuration file and properties related to the execution of the application in the portal (i.e., the language setting for the user in the portal). This API follows the Widget Interface specification defined by the W3C and is partially implemented by Apache Wookie.
- Widget Extensions – offers methods that extend the abilities of the applications. One of the most important methods allows the application to know if the user is logged in.

 ISBN: 978-1-61208-314-8

- Widget Preferences – supports the manipulation of a data persistence area unique to the application instance. The application can use this to store customizable configurations that are specific to a user. This API follows the Widget Interface specification defined by W3C and is implemented by Apache Wookie.
- Wookie Utilities – Helper functions that enable the dynamic update of the web page with content from external sources like servers or the user's input. This API is implemented by the Direct Web Remoting library [29].

You can learn more about these APIs at [30].

F. Inter-App Communication API

In order to support communication through private channels, a flow was defined that would guarantee the authenticity of the user. Inspired by the way the web Pusher service provides a similar mechanism [31], the following process was defined (Fig. 6):

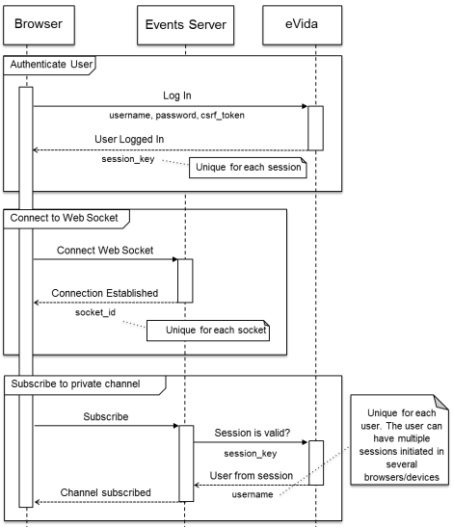

Figure 6. Protection mechanism for private communication channels.

Firstly, the user needs to be authenticated in the portal in order to use the private channels. The user's session key is generated during the login process. As soon as the user executes the first web app, a connection to the events server is established. That connection is uniquely identified by its session id. Afterwards, when a private channel is subscribed, the user's session key allows the identification of the user (provided that the session key is valid).

G. Remote Storage

eVida's users and developers are encouraged to store their data in the platform's repository but are also free to choose other options. To illustrate one alternative to eVida's developers an example packaged app [32], that enables the user to store their data at a place of their choice, was created. This example uses the remoteStorage.js library [33], a client side implementation of the remoteStorage specification [34]. This approach can have several advantages from both the users' and developers' perspectives. From a user's perspective he can effectively own is data and have everything in one place. He can setup a storage account with a provider he trusts or, ultimately, setup his own storage server and the data will always be with him regardless of his location or the status of the applications he uses (i.e., sometimes companies shutdown their services and users may lose their data). From the developers point of view they can develop their web app without worrying about hosting or even developing a backend since the users will connect their own backend at runtime.

H. High-Level View of the Portal and its Connections With Other Entities

Fig. 7 presents the high-level view of the static perspective of the web portal with the representation of the connections to external entities.

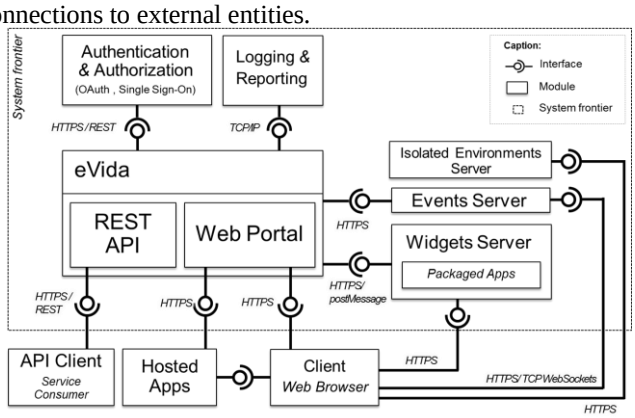

Figure 7. High-level view of the portal's architecure and connections with other modules and external entities.

Note that although the final user only accesses the link to the web interface, in practice, his browser communicates internally in a direct manner with the events, widgets and isolated environment servers, as well as with external applications. The connection established with the events server uses web sockets, or AJAX requests in case web sockets are not supported by the user's browser. The applications communicate with the portal using the postMessage method, which allows the communication between frames of an HTML page through JavaScript (thus not involving network requests).

VI. CONCLUSION

In summary, TICE.Healthy provides the infrastructure and support for an ecosystem of smart and innovative Information and Communications Technology (ICT) services, applications and products for the eHealth and Ageing-Well market. It provisions the developers and the solution providers with basic platform services, such as: authentication and authorization mechanisms, a repository for personal health data, flexible sensor integration and many more. In this manner, TICE.Healthy helps to create an ecosystem of interoperable hardware and software products with greater joint benefit for the end user.

This paper also presented a web portal that operates as information and interaction channel for selling products and

 ISBN: 978-1-61208-314-8

health services. This channel is used to process the exploring of applications and integrate them by allowing them to work around the same context and use common mechanisms. Each user is able to associate his profile with applications provided by the platform, which is responsible for sharing his context and assures a transparent, uniform and consistent user experience.

ACKNOWLEDGMENT

The TICE.Healthy project is co-financed by the European Community Fund through COMPETE - Programa Operacional Factores de Competitividade.

Jorge Manuel Miranda Dias is on sabbatical leave at Khalifa University of Science, Technology and Research (KUSTAR), Abu Dhabi, UAE.

REFERENCES

[1] "TICE.Healthy Candidatura - SISTEMA DE INCENTIVOS À INVESTIGAÇÃO E DESENVOLVIMENTO TECNOLÓGICO - PROJECTOS DE I&DT EMPRESAS MOBILIZADORES," unpublished

[2] "TICE Healthy - We develop products and services for the health market, catalyzing the consortium companies and other partners to create web-based services." [Online]. Available: https://www.evida.pt/. [Accessed: 20-Sept-2013].

[3] "eVida." [Online]. Available: https://www.evida.pt/. [Accessed: 13-Sept-2013].

[4] Instituto Pedro Nunes, "DEMO TICE.Healthy," 2013. [Online]. Available: http://www.youtube.com/watch?v=fwHnHbcpVHY&feature=youtu.be. [Accessed: 20-Sept-2013].

[5] C. Ogbuji, K. Gomadam and C. Petrie, "Web Technology and Architecture for Personal Health Records," IEEE Internet Computing, Jul. 2011, vol. 15, pp. 10-13, doi:10.1109/MIC.2011.99

[6] A. Sunyaev, D. Chornyi, C. Mauro and H. Krcmar, "Evaluation Framework for Personal Health Records: Microsoft HealthVault Vs. Google Health," 43rd Hawaii International Conference on System Sciences, IEEE Press, Jan. 2010, pp. 1-10, doi:10.1109/HICSS.2010.192.

[7] Microsoft, "HealthVault Developer Center," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/healthvault/jj127014. [Accessed: 20-Sept-2013].

[8] Microsoft, "Developer Network – Authentication, Authorization, and Single Sign-In," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff803610.aspx. [Accessed: 20-Sept-2013].

[9] E. Mills, "Welcome to iGoogle," 2007. [Online]. Available: http://www.google.com/ig/adde?moduleurl=www.google.com/ig/modules/education.xml&source=thed. [Accessed: 20-Sept-2013].

[10] A. Chitu, "The New iGoogle, Publicly Launched," 2008. [Online]. Available: http://googlesystem.blogspot.pt/2008/10/new-igoogle-publicly-launched.html. [Accessed: 20-Sept-2013].

[11] J. Price, "The Battle For Your Browser's Homepage: iGoogle vs. Netvibes vs. Pageflakes," 2010. [Online]. Available: http://www.maketecheasier.com/igoogle-vs-netvibes-vs-pageflakes/2010/07/20. [Accessed: 20-Sept-2013].

[12] Facebook, "Apps on Facebook.com — Facebook Developers," [Online]. Available: https://developers.facebook.com/docs/guides/canvas/. [Accessed: 20-Sept-2013].

[13] Facebook, "Apps on Facebook.com — Facebook Developers," [Online]. Available: https://developers.facebook.com/docs/guides/web/. [Accessed: 20-Sept-2013].

[14] OpenSocial, "OpenSocial," 2011. [Online]. Available: http://opensocial.org/. [Accessed: 20-Sept-2013].

[15] LinkedIn, "LinkedIn Apps - LinkedIn Learning Center," [Online]. Available: http://learn.linkedin.com/apps/. [Accessed: 20-Sept-2013].

[16] Twitter, "REST API Resources | Twitter Developers," [Online]. Available: https://dev.twitter.com/docs/api. [Accessed: 20-Sept-2013].

[17] Google, "Packaged Apps - Google Chrome Extensions - Google Code," [Online]. Available: http://developer.chrome.com/extensions/apps.html. [Accessed: 20-Sept-2013].

[18] Google, "Hosted Apps - Installable Web Apps - Google Code," [Online]. Available: https://developers.google.com/chrome/apps/docs/developers_guide. [Accessed: 20-Sept-2013].

[19] Podio, "Podio App Store | Podio," [Online]. Available: https://podio.com/store. [Accessed: 20-Sept-2013].

[20] P. Saint-Andre, "Jabber.org." [Online]. Available: http://www.jabber.org/. [Accessed: 20-Sept-2012].

[21] T. Ryutov, and C. Neuman, "Representation and Evaluation of Security Policies for Distributed System Services, in Proceedings of the DARPA Information Survivability Conference and Exposition," Hilton Head, SC., Jan. 2000.

[22] P. Laskov, "Introduction to Computer Security: Access Control and Authorization," 2005. [Online]. Available: http://www.ra.cs.uni-tuebingen.de/lehre/ss11/introsec/08-unix.pdf. [Accessed: 20-Sept-2013].

[23] K. Oldfield, "Introduction to Unix file permissions," 2003. [Online]. Available: http://oldfield.wattle.id.au/luv/permissions.html. [Accessed: 20-Sept-2013].

[24] P. Catré, A. Marques, J. Quintas, and D. Jorge, "TICE-Healthy: A Dynamic Extensible Personal Health Record," in HEALTHINF2013, Feb. 2013.

[25] L. Haan, A. Vagner, and Y. Naudet, "Palette web portal specification," unpublished.

[26] The Apache Software Foundation, "Apache Wookie (Incubating)." [Online]. Available: http://incubator.apache.org/wookie/. [Accessed: 20-Sept-2013].

[27] "Google Caja." [Online]. Available: https://developers.google.com/caja/. [Accessed: 20-Sept-2013].

[28] "Object-capability model." [Online]. Available: http://en.wikipedia.org/wiki/Object-capability_model. [Accessed: 20-Sept-2013].

[29] "Direct Web Remoting." [Online]. Available: http://directwebremoting.org/dwr/documentation/browser/util/. [Accessed: 20-Sept-2013].

[30] "EVIDA JavaScript APIs." [Online]. Available: https://developer.evida.pt/js-api/template.html#comm-api/en. [Accessed: 20-Sept-2013].

[31] Pusher, "Authenticating users — Pusher," 2012. [Online]. Available: http://pusher.com/docs/authenticating_users. [Accessed: 20-Sept-2013].

[32] "nunoar/remoteStorage.js." [Online]. Available: https://github.com/nunoar/remotestorage.js. [Accessed: 20-Sept-2013].

[33] "remotestorage.js JavaScript client library to connect to a remoteStorage server." [Online]. Available: https://github.com/remotestorage/remotestorage.js. [Accessed: 20-Sept-2013].

[34] "remoteStorage An open protocol for per-user storage." [Online]. Available: http://remotestorage.io/. [Accessed: 20-Sept-2013].

 ISBN: 978-1-61208-314-8