An iOS Framework for the Indivo X Personally Controlled Health Record

An iOS Framework for the Indivo X Personally Controlled Health Record

Pascal B Pfiffner, MD, PhD[1,2], Kenneth D Mandl, MD, MPH[1,2]

[1] Children's Hospital Informatics Program at Harvard-MIT Health Sciences and Technology, Boston Children's Hospital, Boston, MA

[2] Harvard Medical School, Boston, MA

Abstract

The Indivo X personally controlled health record creates a channel between researchers and the patient/subject in several large scale projects. Indivo enables patients to access their health data through a web interface and, as an "apps platform", can be extended in functionality. Patient-facing apps, such as a medication list, may improve the data flow between researcher and patient, in both directions, and as such provide better data for the researcher and immediate benefit for the patient. However, research projects in general do not allocate large funds to patient facing apps, let alone a mobile interface. Thus we have created a framework that greatly simplifies connecting an iOS app to an Indivo X server. Our open-source framework enables novel as well as experienced iOS developers to build mobile interfaces for their research subjects, taking advantage of Indivo X.

Introduction & Background

The personally controlled health record (PCHR) platform Indivo X[1-3] is an "apps platform" that supports substitutability[4] – the capability of swapping out core components for new ones. Substitutability is an important concept because it allows the platform to adapt to the needs of the diverse patient population. The value of such substitutability is accentuated by Indivo's role as personal health information (PHI) store for large scale research projects such as The Gene Partnership [5] or CARRAnet [6]. However profound the level of substitutability, most value for patients is still achieved by offering a wide range of apps interacting with personal health data[2]. Giving patients access to their health data, especially in research environments, can improve data flow from patient to researcher, not only from researcher to the patient. Well-built patient-facing apps, for example a medication list manager, may immediately benefit the patient, in contrast to the usual patient-reported outcome collection via surveys. Often however, the resources to build patient facing interfaces in such studies are limited, and building a separate mobile interface is out of the question.

In order to facilitate building patient facing apps that interface with an Indivo X backend we have released an iOS framework. Our framework seeks to fulful the following desiderata:

1. **Authentication**. The framework handles authentication, including the presentation of a login- and record selection screen, automatically upon request.
2. **Access to Indivo**. All relevant API calls are abstracted into object logic.
3. **Data handling**. The framework provides classes for all relevant Indivo documents, automating XML serialization and deserialization.

Indivo's source code is available under GPL and L-GPL licenses; packaged versions as well as documentation are available at www.indivohealth.org. The iOS framework is available under the L-GPL license from https://github.com/chb/IndivoFramework-ios.

Methods

Indivo recognizes three categories of apps with a different set of privileges:

- **User App**. Runs in context of a record. This is the type of apps targeted at end-users – patients and health professionals alike.

- **Admin App**. Has privileges in the context of an Indivo account. It can create records, receive messages and handle account-level tasks such as resetting a password.
- **Chrome App**. Has access to Indivo's core features. Administrative interfaces use this type, as do UI servers such as our reference UI server.

The framework we created supports "User App" API calls.

Authentication

Authorization and authentication in the Indivo environment is handled through OAuth [7]. Figure 1 shows the authentication workflow between the user, the app developer, the framework and the Indivo Server. Since user apps can only authorize request tokens in scope of a record, the app first has to get a record id, ideally selected by the app user. Because user level apps however are not authorized to handle login- and record-selection, this functionality is provided by the UI server. For authentication purposes, the framework talks to a UI server, which delivers a login page that the framework displays in an embedded web view. After login, still in the web view, the UI server displays a page listing all records available to the user. Upon record selection, the app receives a callback with a record-id which is subsequently used to request a token and perform the remaining steps of the OAuth dance. Once the framework obtains valid access tokens, all calls scoped to the selected record are directly made to the server. This makes the framework dependent on two servers, one supplying login functionality (the UI server) and the other handling data exchange (the Indivo X backend server).

XML Data Model

Indivo is a document-centric container sending and accepting XML documents. Cocoa Touch, the application framework for building iOS applications, does not offer lightweight XML facilities, handling XML however is a fundamental requirement in Indivo. We therefore built a simple DOM implementation which represents XML nodes as INXMLNode objects that can be navigated using jQuery[8] inspired methods.

XML Serialization and Deserialization

To achieve the goal of automatic XML serialization and deserialization, we built a hierarchy of classes that can instantiate themselves from an INXMLNode and write their own XML representation. From the INObject base class, which merely knows its node name, we created subclasses to represent the base data types such as xs:date and xs:string. These classes can all represent one XML node that has no child nodes and we have created classes for most standard XML simple types.

Another subclass of INObject, INServerObject, adds "server awareness" to the object. Instances of this class hold a reference to their Indivo server object and can perform basic Hypertext Transfer Protocol (HTTP) GET, PUT, POST and DELETE actions. This class is the abstract superclass for all objects that live on the server, specifically the record object and all the documents belonging to a record.

The latter, representing the type of objects that will most often be interacted with, has an abstract superclass called IndivoAbstractDocument. Instances of this class can instantiate themselves from an XML tree and also serialize themselves into an XML tree representation. To achieve this, the class, when being told to instantiate itself from a given XML node, walks through all instance variables and looks for variable names matching a child node name and inheriting from INObject. If both conditions are met, the instance variable is told to instantiate itself from the matching child node. Array instance variables are handled in a similar way, except that to know which class to use for objects, the class holds a static dictionary, mapping instance variable names to INObject subclasses.

The reverse process, XML serialization, is achieved by the same principle. When an object is asked for its XML representation, it collects all instance variables that are a subclass of INObject or respond to the "xml" method. All these instances are then asked to return their XML representation from which the object itself can construct a complete XML tree.

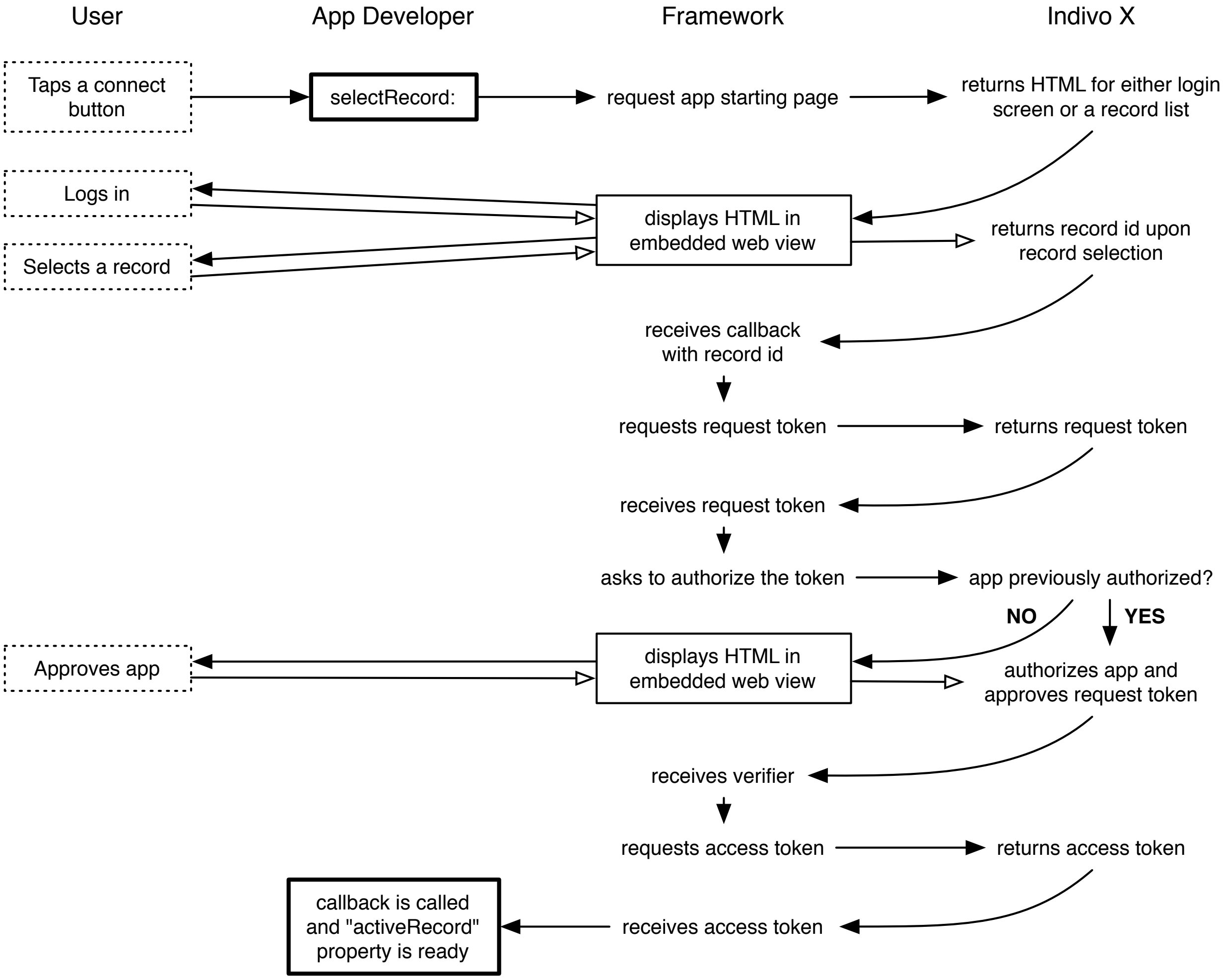
User
App Developer
Framework
Indivo X
Taps a connect button
selectRecord:
request app starting page
returns HTML for either login screen or a record list
Logs in
Selects a record
displays HTML in embedded web view
returns record id upon record selection
receives callback with record id
requests request token
returns request token
receives request token
asks to authorize the token
app previously authorized?
NO
YES
Approves app
displays HTML in embedded web view
authorizes app and approves request token
receives verifier
requests access token
returns access token
callback is called and "activeRecord" property is ready
receives access token

Figure 1: **Authentication Workflow. Most interaction of the app with the server during authentication is handled by the framework, initiated by the developer implementing the “selectRecord:” method on the Indivo server object.**

Class Generation

Indivo 2.0 introduced “pluggable schemas”, the ability to add data models to an Indivo instance by dropping a JSON-like data description by simple file drop. The core data models use the same format. To avoid building Cocoa classes around the schemas by hand and allow developers to easily support custom data models, we have built a companion application that can generate Cocoa classes from Indivo’s SDML files.as part of the framework bundle.

API Calls

The Indivo X API has iteratively been fine-tuned to the needs of different stakeholders such as academic computer scientists, practicing physicians or hospital IT staff[3]. Of the resulting definite REST [9] call set of 155 API calls, 41 pertain to record specific actions. We have abstracted most of these record-specific calls into object logic to provide a pure object-oriented interface to Indivo. The record object for example provides a “fetch reports of class” method, which directly maps to Indivo’s “GET /records/<record-id>/reports/<report-type>/” API call.

Resulting Framework

Our iOS framework interfaces with Indivo X and abstracts most "User-App" level API calls relevant to working with a record into object logic. The framework greatly facilitates building native iOS apps with Indivo as data storage backend. With this we hope to lower the barrier for creating health-related apps for clinical and translational research that can take advantage of PCHR data, patient as well as caregiver-focused.

Implementation

The framework can be used as a static library, a standard format for using foreign C code. This makes it a drop-in component to any Objective-C app and allows the developer to keep his code separate from the Indivo code, with a minimal amount of glue-code. Our use of blocks[10], a C-language extension to create closures, throughout the framework requires apps using the framework to target iOS version 4.0 and higher.

The properties of the Indivo server that the app wants to connect to, namely the URL to both the backend server and the UI server, the app id as well as the consumer key and consumer secret needed to authorize the app, can be changed at runtime. This enables an app to be run against different Indivo instances, in line with Indivo's substitutability paradigm. Still, all servers first have to be set up to allow the app to run against its data.

Framework Use

Authentication logic is hidden to the framework user, and, true to the idea of OAuth, at no point do users enter their credentials into the app itself. The framework loads Indivo's login screen, provided by an instance of an Indivo UI server, in an embedded web view. As a cautionary point, the presentation of an embedded web view technically enables the app to extract user credentials from the web view. Thus to make it impossible for the app to grab credentials, one would have to load the login and record selection screen in the OS's native browser. The app could then receive the callback through a URL-handler registered with the OS. This is possible without much changes to the framework, however since Indivo apps have to be approved by an admin before they can access an Indivo instance, we assumed a certain level of trust and we chose to use an embedded web-view by default, for better user experience.

To alleviate the hurdles posed by API calls and data schemas, we chose to abstract all of Indivo's functionality into Cocoa objects. This means developers need not know the specifics of Indivo's API or data structures, they only work with Indivo objects, instances of Cocoa classes, and interact with Indivo by calling a small set of methods. For example, to upload a new medication to Indivo, the developer instantiates a new medication object, sets its properties and then calls the "push:" method on the object.

SMART-Indivo App Challenge

An iOS app utilizing our framework won second place at the SMART-Indivo App Challenge, a developer challenge sponsored by the Office of the National Coordinator for Health Information Technology. The challenge entries can be viewed at http://www.health2con.com/devchallenge/smart-indivo-app-challenge/.

Security Issues

Mobile applications utilizing OAuth for authentication purposes inherently need to store their consumer secret and consumer key in the application bundle. For this reason, OAuth defines apps living on users' devices as "public" clients because they are incapable of maintaining credential confidentiality [11]. There are techniques to hamper attempts to extract app credentials from an app binary, such as storing the consumer secret in scrambled order and putting it back together at runtime, but this attempt at security through obscurity offers little extra protection, especially in an open source framework. There currently exists no feasible solution to this problem, not only for Indivo but for mobile apps in general.

Conclusion

The Indivo X PCHR provides a basis for storing personal health data, and the framework we developed provides iOS developers with the functionality they need to develop mobile PCHR apps. This greatly reduces the effort required to integrate Indivo with native iOS apps and thus enables researchers to build mobile interfaces for their patients, improving the data flow between patient and researcher. The iOS framework has application in the multiple Indivo testbeds, including the Gene Partnership at Boston Children's Hospital [5], the TuAnalyze application [12], and the CARRANet 60 site pediatric rheumatology registry [6]. Additionally, several components of the framework are being reused in the development of an iOS framework for the SMART platform [13].

Acknowledgements

We thank Daniel Haas for his invaluable input during the development of this framework, as well as Joshua Mandel and Ben Adida for their input on security issues. This work was supported by a scholarship from the Novartis Foundation, formerly Ciba-Geigy Jubilee Foundation, and a Swiss National Science Foundation postdoctoral fellowship (PP). This work was funded by the Strategic Health IT Advanced Research Projects Award 90TR000101 from the Office of the National Coordinator of Health Information Technology, by PO1HK000088-01 from the Centers for Disease Control and Prevention (CDC) and by The Manton Center for Orphan Disease Research at Children's Hospital Boston.

References

1 Riva A, Mandl KD, Oh DH, Nigrin DJ, Butte A, Szolovits P, et al. The personal internetworked notary and guardian. International Journal of Medical Informatics 2001;62:27–40.

2 Mandl KD, Simons WW, Crawford WCR, Abbett JM. Indivo: a personally controlled health record for health information exchange and communication. BMC Med Inform Decis Mak 2007;7:25.

3 Adida B, Sanyal A, Zabak S, Kohane IS, Mandl KD. Indivo X: Developing a Fully Substitutable Personally Controlled Health Record Platform. AMIA Annu Symp Proc 2010;2010:6.

4 Mandl KD, Kohane IS. No small change for the health information economy. N Engl J Med 2009;360:1278–81.

5 Kohane IS, Mandl KD, Taylor PL, Holm IA, Nigrin DJ, Kunkel LM. Medicine. Reestablishing the researcher-patient compact. Science 2007;316:836–7.

6 Natter MD, Quan J, Ortiz DM, Bousvaros A, Ilowite NT, Inman CJ, et al. An i2b2-based, generalizable, open source, self-scaling chronic disease registry. JAMIA 2013;20:172–9.

7 Hammer-Lahav E. The OAuth 1.0 protocol. http://tools.ietf.org/html/rfc5849

8 Resig J. jQuery. http://www.jquery.com

9 Fielding R. Architectural Styles and the Design of Network-based Software Architectures. 2000.

10 Blocks. Apple Inc. http://clang.llvm.org/docs/LanguageExtensions.html#blocks

11 Hammer-Lahav E, Recordon D, Hardt D. The OAuth 2.0 protocol. http://tools.ietf.org/html/draft-ietf-oauth-v2-23

12 Weitzman ER, Adida B, Kelemen S, Mandl KD. Sharing data for public health research by members of an international online diabetes social network. PLoS ONE 2011;6:e19256.

13 Mandl KD, Mandel JC, Murphy SN, Bernstam EV, Ramoni RL, Kreda DA, et al. The SMART Platform: early experience enabling substitutable applications for electronic health records. JAMIA 2012;19:597–603.