USOS API — how to open universities to Web 2.0 community by data sharing

Janina Mincer-Daszkiewicz, Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Banacha 2, 02-097 Warszawa, jmd@mimuw.edu.pl

Keywords
API, information sharing, interoperability, user-centered approach, OAuth, request token, JSON, mobile app, social network, University Study-Oriented System

1. ABSTRACT

In recent years more and more higher education institutions in Poland have deployed computer systems supporting various aspects of their activities. *University Study-Oriented System* (in short: USOS [USOS]) is an integrated suite of applications, running on top of a central database, installed in over 35 higher education institutions (HEIs) in Poland, altogether offering educational services to almost ¼ of the population of students from public sector HEIs. Some of these HEIs, after almost 10-years of using USOS, gathered in their databases a plethora of data, vital for their didactic duties, but also of use in many other aspects of virtual lives of involved students, staff members, university administration and authorities, who want/need access to gathered data not only by standard interface of USOS applications but also in other context. For example students might want to get information of obtained grades straight to mobile phones, staff members might want to share courseware among a group of professionals gathered on Facebook, administrators of departmental web portals might want to present statistical data on offered courses using drawing tools developed by a third party software company. The problem is how to get access to the data which is owned by university, stored in its databases, and only available through the interface provided by USOS developers, who have to prioritize requests concerning software development coming from all universities from MUCI [MUCI] consortium.

Companies like Google, Yahoo, Facebook, or Amazon, which gather data and deliver it to many people all over the world, have already recognized the need of their users for sharing the data across the systems. They offer **API** — an interface used by software components to communicate with each other and exchange data [API]. University is also part of a virtual world, not an island. University systems could integrate with the outside world and support flow of data. USOS developers decided to give university authorities the possibility to open university repositories for public access while still holding them under control, to become more user-oriented, like the aforementioned companies. This possibility comes as **USOS API** — a standard interface to data gathered in USOS databases, publicly available, well documented, with guaranteed backward compatibility.

In this paper we present USOS API, goals we are going to achieve, architecture of the solution, security and privacy issues, design of API methods, sample applications. USOS API is available at http://apps.usos.edu.pl/developers/api/.

2. GOALS

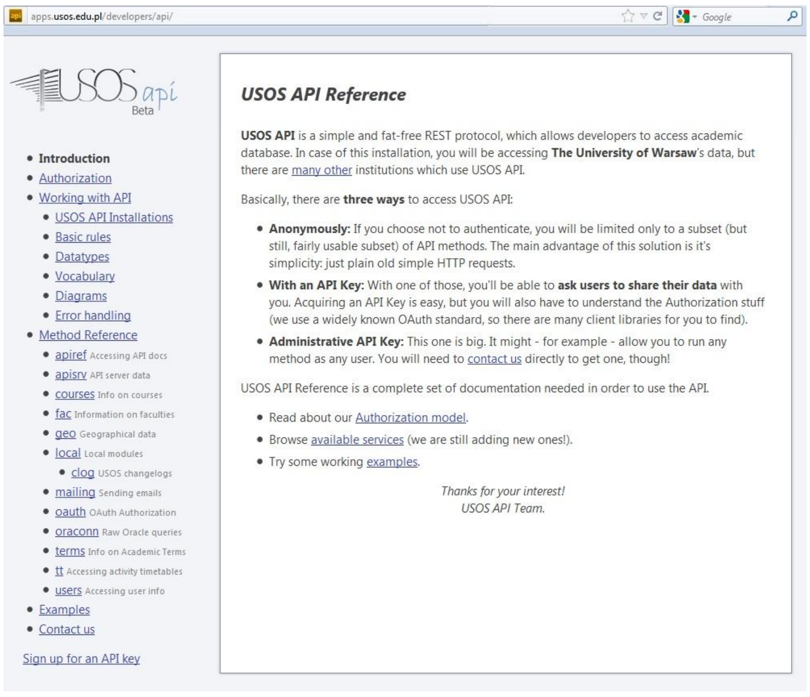
Why should USOS developers be interested in developing API to their systems? Why should university authorities be willing to open access to university data? How to control privacy and security? The last question is particularly important, we will address it in the next chapter. Questions concerning reasonability of the idea are crucial for the project and will be dealt with first.

USOS is a suite of software applications built in a distributed architecture. There is a central Oracle database with many packages, functions, triggers, jobs, etc. There is an interface for the university administration built in Oracle Forms and Reports. Recently Oracle Reports are being replaced by reports developed using *BIRT* — Java-based *Business Intelligence Reporting Tools*. There is a couple of web-based applications, developed in PHP, Smarty, and Javascript, with local MySQL database, which is automatically synchronized with the central Oracle database. There is also a couple of web-based applications developed in Django (framework based on Python), some of them as diploma projects of students of computer science. Business logic which is not gathered in the database, is duplicated and distributed among Oracle Forms, PHP and Django code. This software construction is historically justified, however with the growing number of new modules, becomes more and more difficult to maintain. We decided to identify parts of business logic, which might be gathered in one place and delivered through USOS API. USOS API thus plays a role of an **application server** which implements business logic and makes it available to other modules of the software system. Examples will be given in the following chapters.

USOS API also helps to **integrate** applications from the USOS suite with commodity software already available in university environment, like library system, e-learning, HR and financial systems, which are delivered by independent suppliers. Some data is needed in all and must somehow be shared and exchanged.

University is part of a virtual world, cannot neglect Web 2.0. Students and staff members want to smoothly travel between their private and professional virtual environments, share opinions on Facebook with participants of the same course, browse course catalogues by mobile phones, obtain grades there or register to courses. There are many possible uses of USOS API methods by Facebook plugins, mobile applications, or other tools supporting social networking.

Last but not least, USOS is owned by MUCI consortium, which is a non-profit organization. USOS is a home-made product, developed by the team financed from fees paid by HEIs gathered in MUCI. Such model of financing is very cost-effective, the estimated cost of development of a new software module when calculated per university is very low. It would be difficult for a third party software supplier to deliver a similar product of a comparable cost. However a particular university may be interested in getting a product better suited to its needs, or just different, or with a specific functionality. By publishing standard API giving access to the university data, on the one hand we make it possible for other software companies to work on competitive products, and on the other make the enterprise more economically reasonable, since the same products may also be offered to other HEIs running USOS. **Competitiveness** is profitable for MUCI participants, as is usually the case when there is a higher supply of good quality products on the market. In particular, students' Master and course projects are also easier to carry on and may lead to good quality products, when some basic functionality around USOS data is already available through API.



- **Introduction**
- Authorization
- Working with API
  - USOS API Installations
  - Basic rules
  - Datatypes
  - Vocabulary
  - Diagrams
  - Error handling
- Method Reference
  - apiref Accessing API docs
  - apisrv API server data
  - courses Info on courses
  - fac Information on faculties
  - geo Geographical data
  - local Local modules
    - clog USOS changelogs
  - mailing Sending emails
  - oauth OAuth Authorization
  - oraconn Raw Oracle queries
  - terms Info on Academic Terms
  - tt Accessing activity timetables
  - users Accessing user info
- Examples
- Contact us

Sign up for an API key

USOS API Reference

USOS API is a simple and fat-free REST protocol, which allows developers to access academic database. In case of this installation, you will be accessing **The University of Warsaw**'s data, but there are many other institutions which use USOS API.

Basically, there are **three ways** to access USOS API:

- **Anonymously:** If you choose not to authenticate, you will be limited only to a subset (but still, fairly usable subset) of API methods. The main advantage of this solution is it's simplicity: just plain old simple HTTP requests.
- **With an API Key:** With one of those, you'll be able to **ask users to share their data** with you. Acquiring an API Key is easy, but you will also have to understand the Authorization stuff (we use a widely known OAuth standard, so there are many client libraries for you to find).
- **Administrative API Key:** This one is big. It might - for example - allow you to run any method as any user. You will need to contact us directly to get one, though!

USOS API Reference is a complete set of documentation needed in order to use the API.

- Read about our Authorization model.
- Browse available services (we are still adding new ones!).
- Try some working examples.

Thanks for your interest!
USOS API Team.

Figure 1 USOS API Home Page

USOS API is publicly available in internet, at http://apps.usos.edu.pl/developers/api/ (see **Figure 1**). It is well documented, written in English, what makes it available also for companies from outside Poland. Backward compatibility is guaranteed. Project is in progress, new methods will be published over time.

3. AUTHORIZATION

University databases store private data (like citizen identity numbers, names, birth dates etc.), confidential data (like information about handicapped students or social aid), data of business value (like mailing lists of students and staff members), data which is vulnerable for theft or destruction (like grades or thesis reviews). USOS API should give university authorities the possibility to open the data repositories for public access while still holding them under control, protecting against misuse, violation of privacy, or destruction.

Basically, there are three ways to access USOS API:

- **Anonymously**: if an application chooses not to authenticate, it will be limited only to a subset (but still fairly usable subset) of API methods. The main advantage of this solution is its simplicity: just plain HTTP requests are passed.
- **With an API Key**: application will be able to ask users to share their data with it. Acquiring an API Key is easy but good understanding of the Authorization procedure is necessary.
- **With an Administrative API Key**: it might — for example — allow an application to run any method as any user. Such key is only granted to trusted applications.

Model of authorization followed in USOS API is based on **OAuth protocol**, version 1.0a [OAuth] (used also by Facebook, Twitter, and Google applications). It is a standard method of secure API authorization for web, desktop, and mobile applications.

There are three players taking part in the authorization procedure of the OAuth protocol:

- **Application** (in short **App**) — also called a **Consumer**,
- **USOS API** — also called a **Provider**,
- **User** — this is always one of USOS users who interacts with the application.

Here's what happens during the authorization procedure (see **Figure 2**):

1. App developer generates an API key for the application. This consists of two strings of random characters called a **Consumer Key** and a **Consumer Secret**. Having the keys App can (using OAuth client libraries) make a signed 2-legged API call. (This is called 2-legged, because no User is yet involved.)
2. App can make 2-legged USOS API calls having the Consumer Key only, but usually it needs more complex API methods, which require Users to share their data with it. Here's where the 3-legged authentication begins.
3. User connects to App's site to share his data.
4. App makes a 2-legged services/oauth/request_token API call to acquire a **Request Token** and **Request Token Secret**. App MUST provide an oauth_callback and MAY provide **scopes argument** at this stage. These will affect the authorization flow below.
5. App generates USOSapps authorization URL and asks User to visit it.
6. User visits the authorization page. (If he's not logged in, USOS API will first ask him to log in.) User will be asked to share his data.
7. User grants access to his private data. Now the Request Token becomes an **Authorized Request Token**, which is bound to User who was logged in and authorized it.

a. If App has supplied a callback URL while generating authorization URL above, User's browser is redirected to this URL. USOS API will append a **PIN code** (OAuth Verifier) to the URL GET parameters, where the Consumer can read it.

b. If not, USOS API will display a PIN code to User. App will have to ask User to enter it manually.

8. In either case, App has an Authorized Request Token (bound to a specific User) and a PIN code. They will be used to acquire an **Access Token**.
9. App makes a 3-legged services/oauth/access_token API call signing it with the Consumer Secret and Request Token Secret. In exchange App receives an **Access Token** and **Access Token Secret**.
10. Now App uses **Access Token** (along with the Consumer Key) to call User-related API methods.

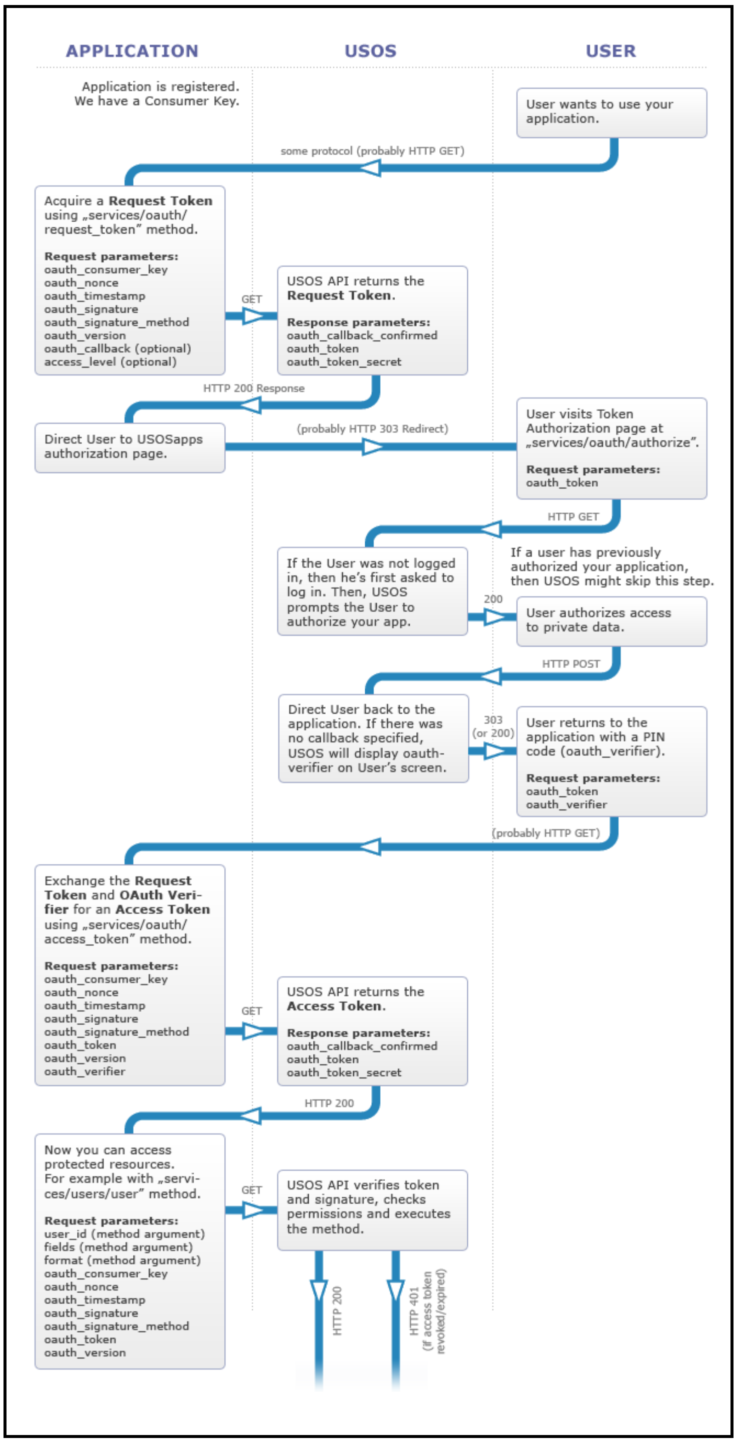
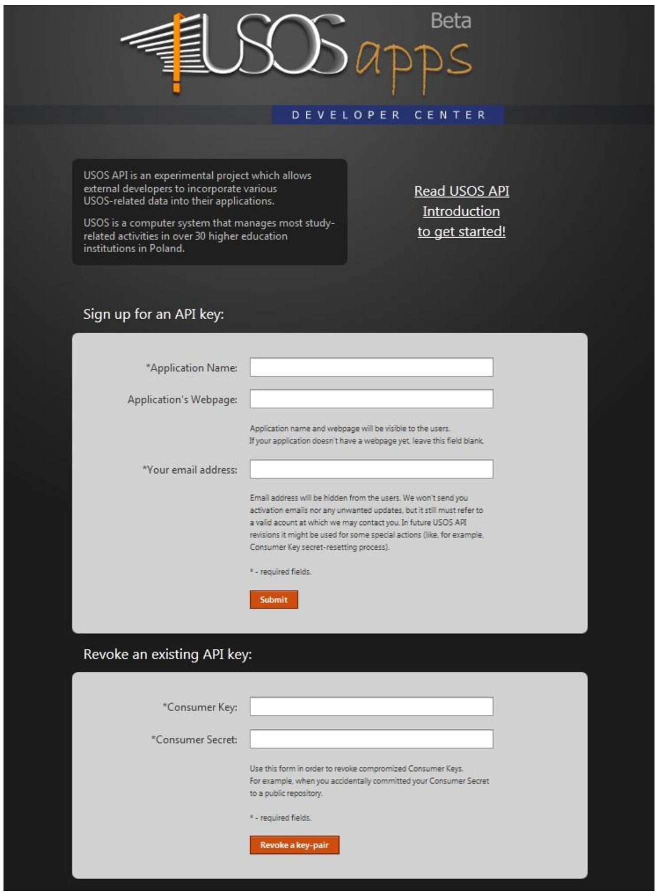

APPLICATION
USOS
USER
Application is registered. We have a Consumer Key.
User wants to use your application.
some protocol (probably HTTP GET)
Acquire a Request Token using „services/oauth/request_token" method.
Request parameters: oauth_consumer_key oauth_nonce oauth_timestamp oauth_signature oauth_signature_method oauth_version oauth_callback (optional) access_level (optional)
GET
USOS API returns the Request Token.
Response parameters: oauth_callback_confirmed oauth_token oauth_token_secret
HTTP 200 Response
Direct User to USOSapps authorization page.
(probably HTTP 303 Redirect)
User visits Token Authorization page at „services/oauth/authorize".
Request parameters: oauth_token
HTTP GET
If the User was not logged in, then he's first asked to log in. Then, USOS prompts the User to authorize your app.
If a user has previously authorized your application, then USOS might skip this step.
200
User authorizes access to private data.
HTTP POST
Direct User back to the application. If there was no callback specified, USOS will display oauth-verifier on User's screen.
303 (or 200)
User returns to the application with a PIN code (oauth_verifier).
Request parameters: oauth_token oauth_verifier
(probably HTTP GET)
Exchange the Request Token and OAuth Verifier for an Access Token using „services/oauth/access_token" method.
Request parameters: oauth_consumer_key oauth_nonce oauth_timestamp oauth_signature oauth_signature_method oauth_token oauth_version oauth_verifier
GET
USOS API returns the Access Token.
Response parameters: oauth_callback_confirmed oauth_token oauth_token_secret
HTTP 200
Now you can access protected resources. For example with „services/users/user" method.
Request parameters: user_id (method argument) fields (method argument) format (method argument) oauth_consumer_key oauth_nonce oauth_timestamp oauth_signature oauth_signature_method oauth_token oauth_version
GET
USOS API verifies token and signature, checks permissions and executes the method.
HTTP 200
HTTP 401 (if access token revoked/expired)

Figure 2 OAuth protocol

API keys are obtained/revoked from the USOSapps Administration Panel (see **Figure 3**).



USOS API is an experimental project which allows external developers to incorporate various USOS-related data into their applications.

USOS is a computer system that manages most study-related activities in over 30 higher education institutions in Poland.

Read USOS API Introduction to get started!

Sign up for an API key:

*Application Name:	
Application's Webpage:	
	Application name and webpage will be visible to the users. If your application doesn't have a webpage yet, leave this field blank.
*Your email address:	
	Email address will be hidden from the users. We won't send you activation emails nor any unwanted updates, but it still must refer to a valid acount at which we may contact you. In future USOS API revisions it might be used for some special actions (like, for example, Consumer Key secret-resetting process).
	* - required fields.

Revoke an existing API key:

*Consumer Key:	
*Consumer Secret:	
	Use this form in order to revoke compromized Consumer Keys. For example, when you accidentally committed your Consumer Secret to a public repository.
	* - required fields.

Figure 3 USOSapps Administration Panel

OAuth tokens may **expire**. Request Tokens expire quickly, they are intended to be immediately exchanged for Access Tokens. Access Tokens live longer, they expire after app. two hours OR after user logs out. All Access Tokens become invalidated when User **revokes access** to his USOS account from the application (using USOSapps Administration Panel).

When App requests a Request Token, it declares a **scopes argument**, which describes permissions required for it to run. Each API method may require different scopes – they are described in the specification.

When App asks User to authorize its Request Token, USOS API will notify User which scopes it requires. It should be chosen wisely – users may discard App request if it wants too much!

These are examples of available scope keys:

- *What can be obtained by default*: permission to read basic user information (such as user's name and id). App need not request this permission explicitly – it receives it by default with each Access Token.
- email: provides access to user's email address.
- offline_access: enables App to perform authorized requests on behalf of User at any time. By default, Access Tokens expire after a short time period to ensure that applications only make requests on behalf of users when they are actively using the applications. This scope makes Access Tokens long-lived.
- personal: provides access to user's personal data, such as citizen identity number, date of birth, etc.

- photo: provides access to user's photo.
- studies: provides access to lists of study programs, courses, classes and groups which the user attends (as a student).

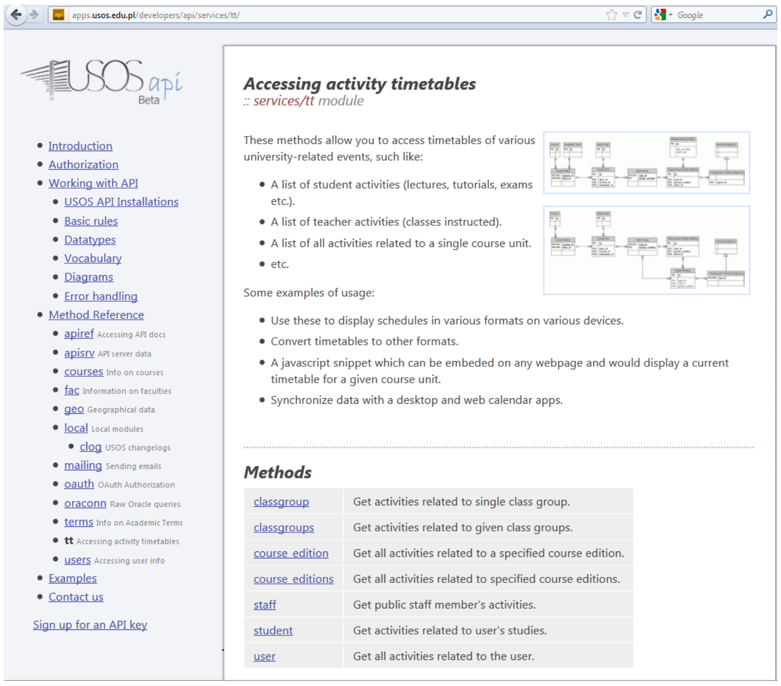
There are some API services that allow access to data that is not available to ordinary users. Therefore, there is no user that could authorize access to such data – the access scopes policy described earlier does not apply.

In order for App to access such sensitive data, App developer needs to contact USOS API administrators directly to acquire an **Administrative Consumer Key** (and **Secret**).

Administrative Consumer Key can be used like an ordinary one, but it grants an administrative access to some of the methods. This means that App may execute standard user methods with a as_user_id argument (instead of a valid Access Token), plus it might get access to methods that are not usually available to ordinary developers.

4. API DESIGN

USOS API Reference is a complete set of documentation needed in order to use the API. Each API method is designed according to some general rules and well documented on the web.



apps.usos.edu.pl/developers/api/services/tt/

USOS api Beta

- Introduction
- Authorization
- Working with API
  - USOS API Installations
  - Basic rules
  - Datatypes
  - Vocabulary
  - Diagrams
  - Error handling
- Method Reference
  - apiref Accessing API docs
  - apisrv API server data
  - courses Info on courses
  - fac Information on faculties
  - geo Geographical data
  - local Local modules
    - clog USOS changelogs
  - mailing Sending emails
  - oauth OAuth Authorization
  - oraconn Raw Oracle queries
  - terms Info on Academic Terms
  - tt Accessing activity timetables
  - users Accessing user info
- Examples
- Contact us

Sign up for an API key

Accessing activity timetables
:: *services/tt module*

These methods allow you to access timetables of various university-related events, such like:

- A list of student activities (lectures, tutorials, exams etc.).
- A list of teacher activities (classes instructed).
- A list of all activities related to a single course unit.
- etc.

Some examples of usage:

- Use these to display schedules in various formats on various devices.
- Convert timetables to other formats.
- A javascript snippet which can be embeded on any webpage and would display a current timetable for a given course unit.
- Synchronize data with a desktop and web calendar apps.

Methods

classgroup	Get activities related to single class group.
classgroups	Get activities related to given class groups.
course_edition	Get all activities related to a specified course edition.
course_editions	Get all activities related to specified course editions.
staff	Get public staff member's activities.
student	Get activities related to user's studies.
user	Get all activities related to the user.

Figure 4 Timetables module with methods and ERD diagrams

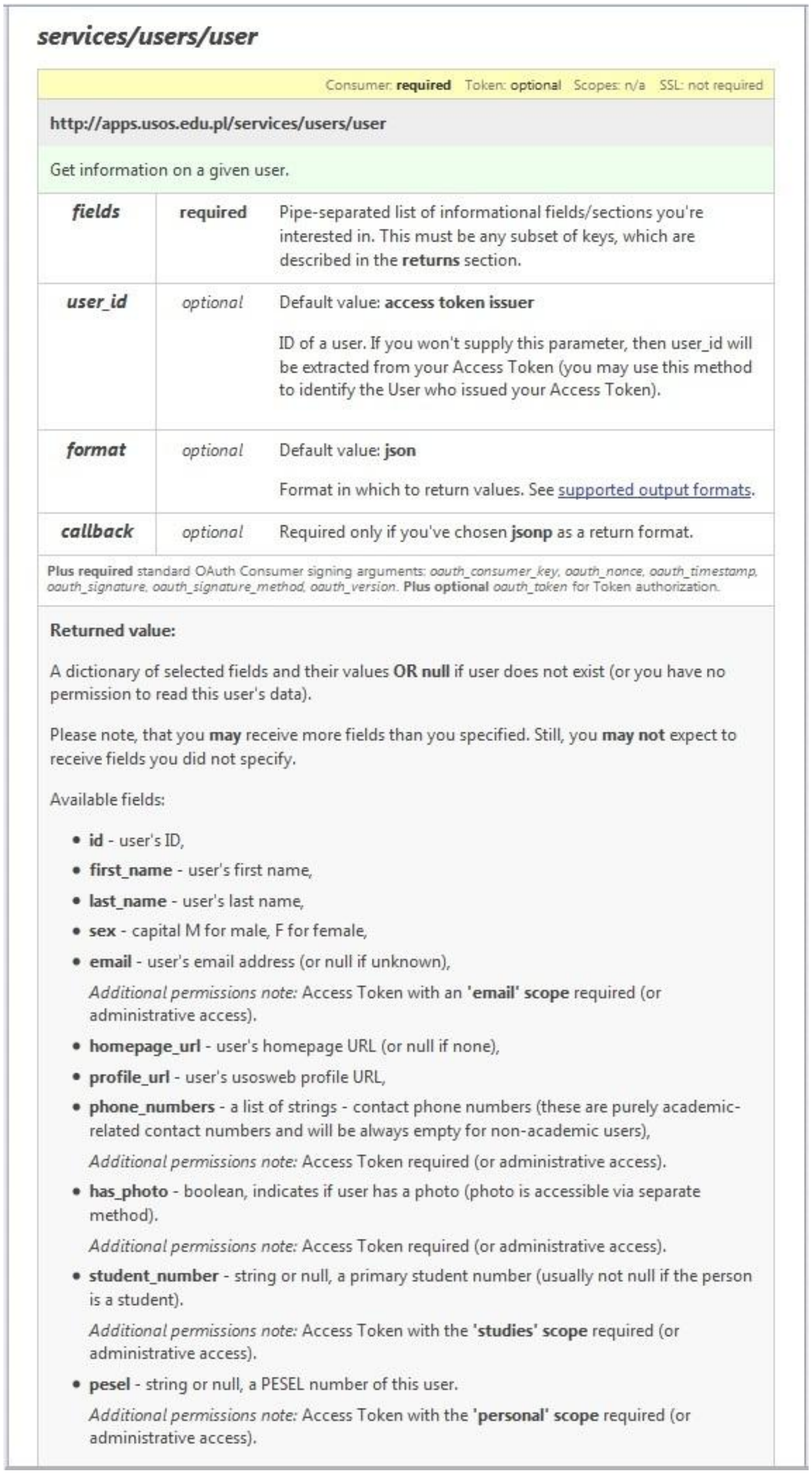
Entities stored in USOS database and handled by API methods are defined, and their relations are illustrated on **Entity-Relationship Diagrams,** using **Crow's Foot Notation**.

All methods are divided into modules, like users, course, terms, geo, mailing, etc. Each method is thoroughly documented, its arguments listed and commented.

Figure 4 illustrates a sample module (**Timetables**). Entities are explained, ERD diagrams shown, methods listed. A detailed specification of each method follows down the page (a sample method **user** from module **users** is shown in **Figure 5**).

Most of the methods come with an optional format argument. It defines in which format the results should be passed. The recommended format is ***JSON*** (*JavaScript Object Notation*), but others are also available (e.g. ***XML map***). Most of the data is available in two languages – Polish and English. All methods use **LangDict** objects to express data that comes in many languages. LangDict object is very simple – it is a dictionary of two keys (**pl** and **en**) and their values.

Many methods return multiple objects, referenced (explicitly) by multiple keys. For example, when App asks for course descriptions of 30 different courses and one of these courses does not exist, it will get the HTTP 400 error. It is possible to ask for **partial response**, which will allow to retrieve 29 of 30 referenced courses.



services/users/user

Consumer: **required** Token: **optional** Scopes: n/a SSL: not required

http://apps.usos.edu.pl/services/users/user

Get information on a given user.

fields	**required**	Pipe-separated list of informational fields/sections you're interested in. This must be any subset of keys, which are described in the **returns** section.
user_id	*optional*	Default value: **access token issuer** ID of a user. If you won't supply this parameter, then user_id will be extracted from your Access Token (you may use this method to identify the User who issued your Access Token).
format	*optional*	Default value: **json** Format in which to return values. See supported output formats.
callback	*optional*	Required only if you've chosen **jsonp** as a return format.

Plus required standard OAuth Consumer signing arguments: *oauth_consumer_key, oauth_nonce, oauth_timestamp, oauth_signature, oauth_signature_method, oauth_version*. **Plus optional** *oauth_token* for Token authorization.

Returned value:

A dictionary of selected fields and their values **OR null** if user does not exist (or you have no permission to read this user's data).

Please note, that you **may** receive more fields than you specified. Still, you **may not** expect to receive fields you did not specify.

Available fields:

- **id** - user's ID,
- **first_name** - user's first name,
- **last_name** - user's last name,
- **sex** - capital M for male, F for female,
- **email** - user's email address (or null if unknown),

  Additional permissions note: Access Token with an **'email' scope** required (or administrative access).
- **homepage_url** - user's homepage URL (or null if none),
- **profile_url** - user's usosweb profile URL,
- **phone_numbers** - a list of strings - contact phone numbers (these are purely academic-related contact numbers and will be always empty for non-academic users),

  Additional permissions note: Access Token required (or administrative access).
- **has_photo** - boolean, indicates if user has a photo (photo is accessible via separate method).

  Additional permissions note: Access Token required (or administrative access).
- **student_number** - string or null, a primary student number (usually not null if the person is a student).

  Additional permissions note: Access Token with the **'studies' scope** required (or administrative access).
- **pesel** - string or null, a PESEL number of this user.

  Additional permissions note: Access Token with the **'personal' scope** required (or administrative access).

Figure 5 Method specification

Figure 5 specifies a simple method **user** from module **users** which returns personal data of a given user (access token issuer, by default, or the one pointed to by the argument user_id). The returned value is a dictionary of (key, value) pairs. Values returned depend on the scope and the passed key.

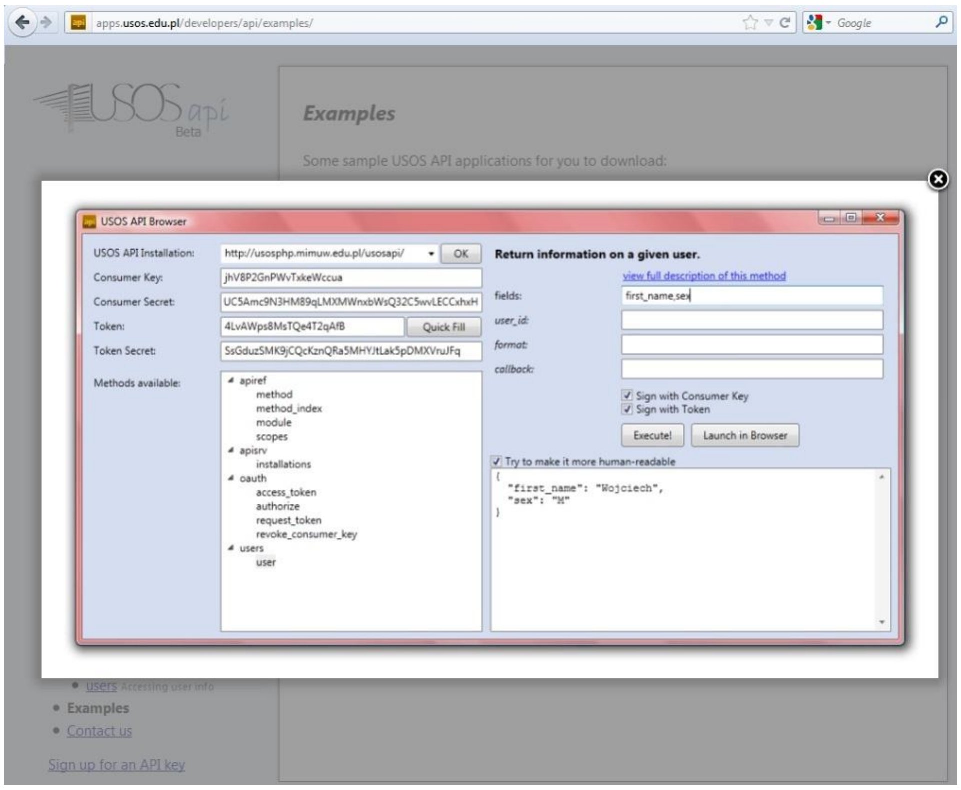
API methods follow **REST** (*Representational State Transfer*) protocol.

API documentation is automatically generated based on the output of **apiref** module, which is part USOS API itself.

5. EXAMPLES OF USING USOS API

Some simple applications which use USOS API have been prepared and posted on the project site to demonstrate to prospective developers how to incorporate authorization and methods invocations into the code.

There is ***Hello World*** written in PHP, ***Simple Proxy*** in PHP, ***Today's schedule*** in Python, and the most useful, ***USOS API Browser*** written in C#. The last one (see **Figure 6**) is Windows desktop application which allows to browse USOS API methods (thanks to the **apiref** module) and execute them with the supplied arguments. This is a very convenient tool for getting acquainted with USOS API itself.



Some sample USOS API applications for you to download:

USOS API Installation: http://usosphp.mimuw.edu.pl/usosapi/
Consumer Key: jhV8P2GnPWvTxkeWccua
Consumer Secret: UC5Amc9N3HM89qLMXMWnxbWsQ32C5wvLECCxhxH
Token: 4LvAWps8MsTQe4T2qAfB
Token Secret: SsGduzSMK9jCQcKznQRa5MHYJtLak5pDMXVruJFq
Methods available:
- apiref
  - method
  - method_index
  - module
  - scopes
- apisrv
  - installations
- oauth
  - access_token
  - authorize
  - request_token
  - revoke_consumer_key
- users
  - user

Return information on a given user.

fields: first_name,sex
user_id:
format:
callback:

[✓] Sign with Consumer Key
[✓] Sign with Token

[✓] Try to make it more human-readable

```
{
  "first_name": "Wojciech",
  "sex": "M"
}
```

Figure 6 USOS API Browser

Some of USOS API methods are already used in the production software. USOSweb which is the main web application of the USOS suite, used daily by students and academic teachers, invokes methods from the Timetable module to draw all schedules, for students, teachers, courses, groups of courses, didactic rooms, etc. Information about public holidays, when there are no classes, is also available from the same source. Information about ECTS points accumulated by a student is obtained by a method which calls a procedure from an Oracle package. Some web applications (USOSweb, Electronic Archive of Diplomas, Survey Tool, USOS-Mailer) use methods from the mailing module to send emails to individual users and groups of users (recognized by USOS). Mails may be sent with attachments, which are files stored in a dropbox, which is also handled by USOS API. Simple search engines of students, teachers, courses etc. are also used. There are some projects under development, like mobile applications (one for iPhone and one for Android) and SocialUSOS (Facebook plugin) which require information about groups of students attending the same courses, grades obtained by students, notifications sent by USOS. Some applications use photo module to obtained photos stored in USOS database.

Some software companies already enquired about the possibility of offering services which would need access to data stored in USOS database.

6. CONCLUSIONS

USOS API is a very young project. We are still experimenting with various solutions. For example there is an issue of granularity of methods and expectations concerning data freshness. Compare a method which delivers a single photo, with another which is responsible for registering a student to a course, according to some predefined policy. The first one is very simple, implemented as a single SQL statement — data may be sent straight from the central Oracle database, because we should not expect many invocations of the method (photos do not change so often and will most probably be cached on the application side). The other is much more complex, may require a sophisticated algorithm run in a transactional mode, since results are needed in real time and data consistency has to be maintained (e.g. number of students who have successfully registered should not exceed the predefined limits). If registration is conducted on a first come first served basis, there is also a problem of scalability on the one hand and system responsiveness on the other. Registering students should not disturb daily work of staff from students offices, so it might be reasonable to run such OLTP-like requests, needing access to the latest versions of the data, on local MySQL database of USOS API, but then keeping data consistency between Oracle and MySQL becomes a challenge.

Users are very much interested in being notified by USOS about various events (new grades, changes in schedules etc.), however such notifications should be kept under control — too many might degrade system operation.

We have not yet run stress tests of the most demanding methods, this is one of the tasks for the near future.

The new methods will be added to USOS API as needs arise. New modules and subsystems of USOS will be based on functionality of USOS API, old modules may be rewritten in new architecture in the future. One of the projects planned for the near future is to rewrite the interface between USOS and Moodle, which is an open source Course Management System, very popular in Poland, used to support e-learning [Moodle]. Many data have to be exchange between student management system and e-learning platform, for example lists of registered students are sent from USOS (which is responsible for registrations) to Moodle, whereas grades are sent in the opposite direction. Now data are synchronized on the database level, which means that both systems are much more vulnerable to changes made in the other.

It should be stressed again that the privacy and security of data is crucial for the project, since data is made available to untrusted and partially trusted third parties. It is ensured by OAuth protocol and application keys of various levels. Unlimited access to data is only granted to applications with the Administrative API Key, most probably developed by the USOS team. Other applications need permission of the user, who gives access to his data in USOS to another application (like mobile app or Facebook plugin). Before user grants access he is first authorized with his USOS account.

There are two options for releasing API: protecting it from the general public or making it freely available. We decided to open USOS API to the public. Such API allows web communities to create an open architecture for sharing content between communities and other applications. Content that is created in one place (e.g. student management information system like USOS) can be dynamically posted and processed in multiple locations on the web by many different service providers. User is in the center of such design. This model also encourages contributions of code from students, freelance programmers, as well as professional software companies.

For USOS developers API is mainly a tool for improving software engineering aspects of the project, like *Rapid Application Development*, easier code maintenance, code standardization and unification of the offered functionality. For universities it may become part of PR strategy, building an image of the educational institution, which is user-centered and open to new trends in social networking and Web 2.0. Both motivations are profitable.

Acknowledgements

The idea of USOS API came from Wojtek Rygielski who also designed and implemented the first version. Many thanks.

7. REFERENCES

[API] Application Programming Interface in Wikipedia, with many links to available APIs. Retrieved in January 2012 from: http://en.wikipedia.org/wiki/API

[Moodle] Home page of the Moodle community, http://moodle.org/

[MUCI] Home page of the MUCI consortium, http://muci.edu.pl

[OAuth] OAuth Community Site, http://oauth.net/

[USOS] Home page of the University Study-Oriented System, http://usos.edu.pl

[USOSAPI] USOS API, http://apps.usos.edu.pl/developers/api/