Web Technologies

Web services (I)

from SOA to SOAP, WSDL, and UDDI
“We treasure what we understand.”

Kevin Budelmann
What are the Web purposes?
Creating and interacting with an inter-human communication space

- knowledge sharing
Creating and interacting with an inter-human communication space

- knowledge sharing

social Web ("Web 2.0"), Web of data (semantic Web),...
Making use of the computational power

the access to the Web can be achieved by using devices with limited resources
Making use of the computational power

the access to the Web can be achieved by using devices with limited resources

ubiquitous Web: mobile Web, 3D Web,...

performance ▶ assuring scalability
The interaction between humans and Web is made by using Web forms and exploring links via Web addresses – URIs
How can be accessed and processed the resources – data, information, knowledge – available on the Web?
Web developers’ needs

Multi-platform, loose-connected solutions

Internet/Web-based integration (in real-time) of applications, services, and systems
Web developers’ needs

Multi-platform, loose-connected solutions

Internet/Web-based integration (in real-time) of applications, services, and systems

example: finding the service offerings, based on user’s geographic location, in the context of mobile devices
Web developers’ needs

Multi-platform, loose-connected solutions

data should be described in order to be “understood” by computers and to be easily interconnected
Web developers’ needs

Multi-platform, loose-connected solutions

data should be described in order to be “understood” by computers and to be easily interconnected

Web “puzzles”

- interconnected multiple services providing data of interest, according to the user preferences
Web developers’ needs

Pluggable & versatile services

Software as a Service – SaaS

Application Service Provider – ASP
Dividing applications into – independent – services that can be composed, meant to connect and orchestrate in a spontaneous way in the business/technical processes
solution

“The Web is the computer”
solution

“The Web is the computer”

using the architecture(s) that...

provide support for communication paradigms – based on current Web technologies – between heterogeneous applications
solution

“The Web is the computer”

using the architecture(s) that...

allow a transparent manner to locate services
solution

“The Web is the computer”

using the architecture(s) that...

facilitate adding, replacing, removing services in a dynamic way
solution

“The Web is the computer”

using the architecture(s) that...

hide the system details
solution

“The Web is the computer”

using the architecture(s) that...

assure the quality of distributed and/or parallel application development & deployment: standardization, security, availability, reusability, maintainability, etc.
Web as a middleware technology

**client proxy**
- data (de)serialization
- data receiving
- service invocation

**Web broker**
- service registration
- server localization
- establishing communication

**server proxy**
- data (de)serialization
- service invocation
- result retrieval

**client**
- calls service (at proxy level)

**service**
- waits for requests
- executes operations
How about Web services?
web service

“A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities.”

www.w3.org/TR/ws-gloss/
“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.”

www.w3.org/TR/ws-gloss/
Software providing a specific functionality

resource access – Instagram, Pinterest, Slideshare, Vimeo
news aggregation – Feedly, Reddit
mapping – Google Maps, Nokia HERE, OpenStreetMap
instant messaging – Snapchat, Telegram, Twilio
processing – IBM Watson, TensorFlow, Skyttle,...
Web statistics – e.g., Google Analytics
social networks – Facebook Open Graph, Twitter
spelling checking – Bing Spell Check, xSpell,...
data storage – Amazon S3, Dropbox, OneDrive, etc.
...
used – remotely – by other applications/services
web service

Accessed – in a standardized way – via Web
addressing resources with URI
data transfer via HTTP
messages having various data formats: CSV, JSON, XML,...
web service: example

A service provided by a travel agency

offering – and, possibly, selling –
various holiday formulas
web service: example

A service provided by a travel agency uses other (software) services available on the Web:

- mapping + weather services
- accommodation services
- financial transactions—*e.g.*, e-banking
- transport services
- social recommendation services
How can we implement a service?
web service

Standard implementation

using Web application servers + frameworks

ASP.NET, Django, Node.js,
PHP (CodeIgniter, Laravel,...),
Play!, Ruby on Rails,...
Traditionally, the application provides a user-interface available on the Web markup language – *e.g.*, HTML styles for content presentation – CSS interactivity via JavaScript (+libraries/frameworks)
web service

Traditionally, the application provides a user-interface available on the Web

requests are captured via forms + hypermedia links
web service

Traditionally, the application provides a user-interface available on the Web

human users should interpret the labels and dialog fields

33
Euro

145.71
Romanian Leu
web service

Traditionally, the application provides a user-interface available on the Web implemented service provides a response (a Web resource representation)

usually, an HTML document whose content is transferred to the client according to a protocol: HTTP(S)
How can we obtain the response in order to be (re)used in our programs?

processing the data from HTML code ▶ Web scraping
web service

How can we obtain the response in order to be (re)used in our programs?

processing the data from HTML code ➤ Web scraping

any markup modification ➤ rewriting the program that process the data from HTML document
web service: characteristics

Web services made explicit the implicit indications

the input data and result can be (rigorously) specified by using various validation techniques
web service: characteristics

Used in the interaction between applications

dynamic

lack of *a-priori* knowledge about the interaction with other Web applications/services
web service: characteristics

Endpoints used for public data processing – possibly, via open APIs
web service: characteristics

Developed on the basis of current platforms, architectures, technologies, and languages
There is an architectural model for developing Web services?
soa

Service Oriented Architecture
An architectural style for designing and developing applications considered as services that can be invoked by other applications
A software development paradigm adopting the use of services, that provides functionalities required by users.
A software development paradigm adopting the use of services, that provides functionalities required by users resources are available through a suite of independent services those implementations do not need to be known (black box)
Components of the system as a whole have a high degree of independence (de-coupling)
Components of the system as a whole have a high degree of independence (de-coupling)

services must be designed to interact without dependences between them
Services share a formal contract

necessity for a formal description of a service:
provided operations (service interface)
data interchange manner (request + response)
service discoverability
service quality (SLA – service-level agreement)
Services can be re-composed/re-orchestrated according to the requirements or deployment context

composable services
reusable services
Services will not depend on the state of communication (statelessness)
Services will not depend on the state of communication (statelessness)

in order to perform an operation, the amount of information that can be kept should be minimal
from traditionally developed Web applications to service-based architectures

Jay O’Connor (2014)
http://tinyurl.com/l62tjab
ubiGuide project
(Ionuț Dănilă & Mihaela Ghimiciu, 2013—2014)
public services: ABBYY Cloud OCR, AlchemyAPI,
Google Places API, YouTube API,...
libraries: Flickr.NET, GART (Geo Augmented Reality Toolkit),
Hammock, TweetSharp, etc.
https://www.youtube.com/watch?v=wygXE6hQ07c
http://www.slideshare.net/ionutdanila/ubi-guide
Can we specify in a standardized format the input data and the response provided by the Web service?
The need for a data communication (transport) protocol between heterogeneous platforms/applications
invocation

The protocol should offer a mechanism of **invoking** and **transmitting** data in a structured way.
invocation

The protocol should offer a mechanism of **invoking** and **transmitting** data in a structured way, facilitating complex interactions between applications and assuring extensibility + security, reliability, caching.
invocation: solution

**XML-RPC (1999)**

easy to be used, but modest

based on RPC (Remote Procedure Call)

messages are modeled in XML

http://xmlrpc.scripting.com/spec
invocation: solution

SOAP

sophisticated, more flexible

W3C standards (2007)

commonly used in the enterprise applications

http://www.w3.org/TR/soap12/
invocation: solution

Adopting other representations

CSV (Comma Separated Values)
POX (Plain Old XML)
JSON (JavaScript Object Notation)

the developer creates its own serialization methods
invocation: soap

Goal:
a protocol of communication between two machines (client and server) for XML data interchange

platform/language-independent
invocation: soap

A W3C standard (2003, 2007)

www.w3.org/TR/soap12-part0/
invocation: soap

The “dialog” between applications takes place by using **SOAP messages** ≡ XML documents

- envelope
  - header + body

XML messaging
invocation: soap

SOAP Envelope

SOAP Header
- header block
- header block

SOAP Body
- body block
- body block
- body block
- body block
invocation: soap

Can describe a model for content processing

SOAP encoding rules
invocation: soap

Can specify a route from sender to receiver, via an optional intermediary (proxy)

SOAP routing
invocation: soap

The headers could be processed by different intermediaries
invocation: soap

XML data stored by a body block can be transported regardless the protocol HTTP, usually other protocols can be adopted – e.g., SMTP, XMPP
invocation: soap

By using SOAP, it can be specified a call of a method available on other computer

SOAP RPC representation
SOAP message exchange

sender

SOAP system

XML encoding

data packing

any data type

SOAP message exchange

receiver

SOAP system

XML decoding

data unpacking

binding

transport protocol

Internet (intermediaries)

transport protocol
(instead of) break
invocation: soap – example

Scenario:
a client sends to a service a name of orange 
(as input argument) 
in order to obtain the available quantity – i.e. the response
invocation: soap – example

TCP/IP classical approach 😞

sockets ‣ a user-defined port (e.g., 7777) + a convention of data exchange is specified
invocation: **soap** – example

**RPC approach 😞**

a remote procedure call, executed by the server (using a port of RPC dispatcher), adopting XDR – External Data Representation – as a binary protocol of data (de)serialization

typical implementations in C, C++ or Java (with RMI)
invocation: soap – example

SOAP approach – XML messaging

exchanged data is wrapped into SOAP “envelopes” (messages), transported via HTTP

platform- and language-independent
Invoking a Web service – RPC style

Client invoking a Web service

calls method 2 of Web service

proxy class

sync. transfer implem.

async. transfer implem.

Implemented Web service

Web service class

method 1 implem.

method 2 implem.

method $N$ implem.

SOAP messages ✉️
invocation: a SOAP message – HTTP request

POST http://web.info/porto/
Accept: text/xml
Content-Type: text/xml

```xml
  <S-ENV:Body>
    <!-- calling the method to get the available quantity -->
    <v:getQuantity xmlns:v="http://web.info/porto/">
      <!-- input data: the orange name -->
      <v:arg0>blue</v:arg0>
    </v:getQuantity>
  </S-ENV:Body>
</S-ENV:Envelope>
```
invocation: a SOAP message – HTTP response

HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
><soap:Body>
  <tns:getQuantityResponse
    xmlns:tns="http://web.info/porto/"
  >
    <!-- the transmitted result -->
    <return>74</return>
  </tns:getQuantityResponse>
</soap:Body>
</soap:Envelope>

XML namespace defined by our service
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Body>
    <s:Fault>
      <faultcode>flickr.error.0</faultcode>
      <faultstring>Invalid SOAP envelope.</faultstring>
      <faultactor>
        http://www.flickr.com/services/soap/
      </faultactor>
      <details>
        Please see http://www.flickr.com/services/api/ for more details
      </details>
    </s:Fault>
  </s:Body>
</s:Envelope>
invocation: soap

SOAP \equiv \text{RPC for the Web}

request + response including I/O parameters (+their data types) specified in XML
invocation: soap

SOAP \equiv \text{messaging (serialization) protocol}

request contains a serialized request-object

response includes a serialized response-object
A language able to describe Web services is necessary
A language able to describe Web services is necessary

How a Web service could be found?

What is the syntax of transported messages?

How data transfer is performed?
description: wsdl

Web Service Description Language

a W3C recommendation (2007)

www.w3.org/TR/wsdl20/
A Web service is described in an XML format by a .wsdl document. Data types (input arguments + received response) are defined by using XML Schema. Specifies the syntax, not the semantics of a service.
**description: wsdl**

<table>
<thead>
<tr>
<th>Service interface (abstract definition)</th>
<th>Service implementation (actual specification)</th>
</tr>
</thead>
<tbody>
<tr>
<td>messages</td>
<td>binding</td>
</tr>
<tr>
<td>operation</td>
<td>service</td>
</tr>
<tr>
<td>interface</td>
<td>endpoint</td>
</tr>
</tbody>
</table>
Conceptually, it uses a data model represented by a set of components having various properties.
How the existing Web services could be (re)found?
discovery: uddi

Universal Description, Discovery, and Integration

a common distributed catalogue
of available (registered) Web services


www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
discovery: uddi

Universal Description, Discovery, and Integration

UDDI is in fact a Web service – invoked via SOAP
records are replicated
in the present, privately available (enterprise level)
discovery: uddi

UDDI registry

providers & services listing

service publish

search a provider

service invocation

obtained result

client application

service requester

service provider

service description

advanced
1. searches for a service
2. obtains service URL
3. gets service description
4. sends SOAP request
5. invokes service
6. returns result
7. gets data via SOAP

UDDI registry
query API

client

Web server

Web service

WSDL
discovery: uddi – implementation

Software tools – examples:

jUDDI (Apache): http://juddi.apache.org/
able to manage a private UDDI catalogue

SOA Client – a Firefox extension
performs queries against a UDDI catalogue
https://addons.mozilla.org/firefox/addon/soa-client/
inquiries on a jUDDI registry via the SOA Client extension
other specifications & initiatives (WS-*)

Addressing: WS-Addressing
Discovery: WS-Inspection, WS-Discovery
Messaging: Reliable HTTP (HTTPR), WS Attachments, WS-Routing,...
Transaction support: WS-Coordination, WS-Transaction
Interaction between Web services and users:

**WS for Remote Portlets (WSRP),**

**WS for Interactive Applications (WSIA)**

**Workflows:** Business Process Execution Language (BPEL),

WS-Choreography, WS Flow Language (WSFL),...

**Interoperability – WS-I initiative:** [www.oasis-ws-i.org](http://www.oasis-ws-i.org)

...and many others
Web services’ presence is enough?
Data and services should be available on each device and from anywhere

consider the Web ubiquity
development

The need for a service oriented infrastructure

a communication “bus” for services/components
New services could be composed by already existing Web services and transparently accessed
development

New services could be composed by already existing Web services and transparently accessed the need for a device-independent platform able to execute services middleware providing functionalities + inter-operability
Supporting the Web content in “old” way (e.g., CGI, Web application servers) + Web services

Web servers = “gateways” to Web pages and/or services
Dr. Sabin Buraga

conceptual layered structure of a Web service-based platform

Web service provider/user

API

backend server(s)

legacy systems

core services (calendar, transactions, ...)

service context (who, why, ...)

service description: WSDL

discovery services: UDDI

protocols (HTTP, SMTP, ...)

advanced

micro-/macro-serv.

virtual machine

workflow engine

virtual machine
Development

Technologies, products, applications – examples:

Apache Axis2 (C and Java), Synapse, Tuscany (C++, Java)
EasyWSDL – Java, Objective-C, Swift class generator
Eclipse SOA Tools – www.eclipse.org/soa/
goat (a SOAP client written in Go language)
gSOAP (C and C++)
JAX-WS – Java Architecture for XML: Web Services
native support provided by .NET (C# et al.)
osa, Zeep (Python implementations)
features provided by Play! framework (Java, Scala)
development

Technologies, products, applications – examples:

- built-in support in PHP5+
- Red Hat JBoss Enterprise SOA Platform (Java)
- Savon (Ruby)
- soap, soap-server (Node.js) – nodejsmodules.org/tags/soap
- SOAP::Lite (a Perl module)
- SOAPEngine (a SOAP client for iOS apps)
- WSDL2ObjC (Objective-C)
- Boomerang, Wizdler (Chrome extensions)
- ...and others
development: soap public services

Various examples:

Continental Clothing, Culture24, FedEx, FitnessWizard, Who’s Hurt, InsideSales, LyricWiki, Magento, OnTime, PDFReactor, Salesforce, ScrumWorks Pro, SugarCRM, TrackRoad Routing, ViaMichelin, Wikispaces, XML Soccer
development: case study

Invoking a public Web service by using its WSDL description
development: case study

Invoking a public Web service by using its WSDL description

we visit Programmable Web – http://www.programmableweb.com/ – in order to obtain the list of SOAP invocable services
Aonaware Dictionary API offers the description of functionalities via WSDL access without authentication no developer key is required

<table>
<thead>
<tr>
<th>API Provider</th>
<th><a href="http://aonaware.com">http://aonaware.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>API Homepage</td>
<td><a href="http://services.aonaware.com/DictService/">http://services.aonaware.com/DictService/</a></td>
</tr>
<tr>
<td>Primary Category</td>
<td>Reference</td>
</tr>
<tr>
<td>Secondary Category</td>
<td>Dictionary</td>
</tr>
<tr>
<td>Protocol / Formats</td>
<td>XML, SOAP</td>
</tr>
<tr>
<td>APIhub URL</td>
<td></td>
</tr>
<tr>
<td>API Kits</td>
<td>C#</td>
</tr>
<tr>
<td>Contact Email</td>
<td><a href="mailto:website@aonaware.com">website@aonaware.com</a></td>
</tr>
</tbody>
</table>

**Molu - The Search dog**
Molu is a super fast vertical meta search engine searching across various categories in a single go. What more, it does not store any of the user history unlike other search engines which track the...

**Scrabble Helper and Cheat**
A Scrabble helper, Scrabble cheat, and Scrabble word finder. When you find a word, you can look it up in the dictionary.

**Molu - Dictionary Search Bot**
This GTalk/Jabber bot searches the dictionary meaning of any given word and gives users the power of information at their fingertips.
we run **Boomerang** – a Chrome extension

by using the URL of WSDL Web service description, a list of implemented operations is provided: 

**Define** **DictionaryInfo** **DictionaryList** **Match** etc.
Aonaware Dictionary API

using the WSDL specification, we can learn the structure of input parameters for Define operation

```xml
<s:element name="Define">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="word" type="s:string"/>
    </s:sequence>
  </s:complexType>
</s:element>
```

Define operation has only one optional input argument of string type
a SOAP request performed with Boomerang extension
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:s="http://services.aonaware.com/webservices/">

<SOAP-ENV:Body>

<!-- specifies the operation (method) to be invoked -->

<s:Define>

<!-- input parameter; here, a term (a character string) -->

<s:word>

programming

</s:word>

</s:Define>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Dictionary API – invoking the operation (SOAP request)
Aonaware Dictionary API

using WSDL, we can know the data type of the response provided by the **Define** operation (remotely executed)

```xml
<s:element name="DefineResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
                  name="DefineResult"
                  type="WordDefinition"/>
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="WordDefinition">
  <s:sequence>
    ...
  </s:sequence>
</s:complexType>
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <DefineResponse xmlns="http://services.aonaware.com/webservices/">
      <DefineResult>
        <Word>programming</Word>
        <Definitions>
          <Definition>
            <Word>programming</Word>
            <Dictionary><Id>wn</Id><Name>WordNet</Name></Dictionary>
            <WordDefinition>program: a system…</WordDefinition>
          </Definition>
          <Definition> <!-- other definitions --> </Definition>
        </Definitions>
      </DefineResult>
    </DefineResponse>
  </soap:Body>
</soap:Envelope>

Dictionary API – obtained data (SOAP response)
a SOAP response visualized with Boomerang extension
development: php

**SoapServer** class serves SOAP requests

useful methods:

- **addFunction ()** – adds a function processing a request
- **setClass ()** – specifies a class implementing a service
- **handle ()** – "listens to" SOAP request from clients
- **fault ()** – emits an error message (SOAP fault)
- **setPersistance ()** – sets the connection persistence

try {
$server = new SoapServer (null, // no WSDL description is provided
    array ('uri' => 'http://web.info/porto')); // service namespace

    // adding implemented methods (i.e. functionalities, operations)
$server->addFunction ('getQuantity');
$server->handle (); // waiting for SOAP requests
}

} catch (SOAPFault $exception) {
    // an exception occurred :
    die ('Oh, a problem… ' . $exception);

} // function providing the quantity of an orange sort
function getQuantity ($product) {
    // usually, we’ll performed a SQL query, a (CSV, JSON, XML,...)
    // data processing, a call of other Web service, etc.
    switch ($product) {
        case 'gray'  : return 33;
        case 'blue'  : return 74;
        default      : return 'n/a';
    }
}
SoapClient class performs SOAP requests to a service

methods of interest:

__soapCall () – sends a SOAP request (an operation call)
__getLastRequest () – offers info about the last request
__getLastResponse () – gets info about the response
__setSoapHeaders () – sets various headers (see SoapHeader)
__setCookie () – sets a cookie

```php
$client = new SoapClient(null,
    array ('location' => 'http://web.info/porto.php', // Web service address
        'uri'    => 'http://web.info/porto')); // namespace

// performing several calls of the desired method (operation)
foreach (array ('blue', 'gray', 'celestial') as $product) {
    $res = $client->__soapCall('getQuantity', array ($product));
    echo "<p>The quantity of $product oranges is <strong>$res</strong>.</p>";
    ...
}
```
The quantity of gray oranges is 33.

The quantity of celestial oranges is n/a.
Using annotations to specify the service

```java
package ro.infoiasi.oranges;

import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService
public class Oranges {
    // a class implementing the Web service
    @WebMethod // a public method providing the quantity
    public Integer getQuantity (String product) {
        ... }
    @WebMethod // another public method offering the price
    public Double getPrice () {
        ... }
}
```
development: java

Publishing the Web service

package ro.infoiasi.oranges.service;

import javax.xml.ws.Endpoint;
import ro.infoiasi.oranges.*;

public class ExposedService {
    public static void main (String[] args) {
        try {
            // publishing the Web service to a specified URL
            Endpoint.publish("http://localhost:8888/porto", new Oranges());
        } catch (Exception e) {
            /* an exception occurred... */
        }
    }
}
development: java

Accessing (consuming) the Web service by a client

```java
public class ClientCravingForOranges {
public static void main (String[] args) {
    try {
        // instantiating the service by using
        // the proxy class already generated with 'wsimport' utility
        OrangesService serv = new OrangesService ();
        Oranges oranges = serv.getOrangesPort ();
        // calling the methods exposed by service
        System.out.println("The stock of blue oranges has the value " +
                           oranges.getQuantity ("blue") * oranges.getPrice ());
    } catch (Exception e) {
        /* an exception occurred... */
    }
}
```
Web services based on Java according to the ESB (Enterprise Service Bus) model experiment GlassFish project – glassfish.java.net

Web services based on .NET WCF (Windows Communication Foundation)

also, study L. Alboaie, S. Buraga, Web Services, Polirom, 2006 [RO]
www.slideshare.net/busaco/l-alboaie-s-buraga-servicii-web-concepte-de-baz-i-implementri-2006
“conclusion”

Web services

“definitions”, characteristics, architecture, SOAP, technologies and applications
next episode: **Web services with REST**

<table>
<thead>
<tr>
<th>Users</th>
<th>Show/Hide</th>
<th>List Operations</th>
<th>Expand Operations</th>
<th>Raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/Users/login</td>
<td>Login a user with username/email and password</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/Users/logout</td>
<td>Logout a user with access token</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users/confirm</td>
<td>Confirm a user registration with email verification token</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/Users/reset</td>
<td>Reset password for a user with email</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users/{id}/accessTokens/{fk}</td>
<td>Find a related item by id for accessTokens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>/Users/{id}/accessTokens/{fk}</td>
<td>Delete a related item by id for accessTokens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>/Users/{id}/accessTokens/{fk}</td>
<td>Update a related item by id for accessTokens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users/{id}/accessTokens</td>
<td>Queries accessTokens of User.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/Users/{id}/accessTokens</td>
<td>Creates a new instance in accessTokens of this model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>/Users/{id}/accessTokens</td>
<td>Deletes all accessTokens of this model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users/{id}/accessTokens/count</td>
<td>Counts accessTokens of User.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/Users</td>
<td>Create a new instance of the model and persist it into the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>/Users</td>
<td>Update an existing model instance or insert a new one into the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users</td>
<td>Find all instances of the model matched by filter from the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users/{id}/exists</td>
<td>Check whether a model instance exists in the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEAD</td>
<td>/Users/{id}</td>
<td>Check whether a model instance exists in the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Users/{id}</td>
<td>Find a model instance by id from the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>/Users/{id}</td>
<td>Delete a model instance by id from the data source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>/Users/{id}</td>
<td>Update attributes for a model instance and persist it into the data source</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>