# Application Programming Interface (API)

**Ivan Pepelnjak (@ioshints, ip@ioshints.info)**
**NIL Data Communications**

# High-Level Overview

- External access to structured data (XML or JSON)
- Read-only or read-write (transactions)
- Public or authenticated
- Server-to-server or client-to-server

Typical use cases

- Mashups (Web Application Hybrids)
- Application Integration
- Multiple UIs from same data set
- Third-party platform support

# What Is JSON?

```
{
 "firstName": "John",
 "lastName" : "Smith",
 "age" : 25,
 "address" : {
      "streetAddress": "21 2nd Street",
      "city" : "New York", "state" : "NY",
      "postalCode" : "10021" },
 "phoneNumber": [
      { "type" : "home", "number": "212 555-1234" },
      { "type" : "fax", "number": "646 555-4567" } ]
}
```

Source: en.wikipedia.org/wiki/JSON

# Keep It Short & Simple (KISS)

REST (Representational State Transfer)

- Client-server
- Stateless
- Cacheable
- Resources identified in requests (URIs)
- Self-Descriptive Messages
- Hypermedia (links) describe application state

**Don’t overcomplicate matters with SOAP or RPC/CORBA**

# RESTful Services Guidelines

Use HTTP(S)

- GET for read-only requests
- POST to create new objects
- POST or PUT (preferably PUT) to modify objects
- DELETE to delete objects
- Use redirect-after-modify status codes

# Sample ATOM Feed

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns='http://www.w3.org/2005/Atom'
    xmlns:gd='http://schemas.google.com/g/2005'>
  <id>tag:blogger.com,1999:blog-23021255</id>
  <updated>2012-04-21T22:55:07.918+02:00</updated>
  <title type='text'>ipSpace.net</title>
  <subtitle type='html'>...</subtitle>
  <link rel='self' type='application/atom+xml' href='...'/>
  <link rel='alternate' type='text/html' href='...'/>
  <link rel='next' type='application/atom+xml' href='...'/>
  <entry>...</entry>
  ...
</feed>
```

Extensible

Unique ID

Timestamps

Typing

Links

Application state

# Sample ENTRY In ATOM Feed

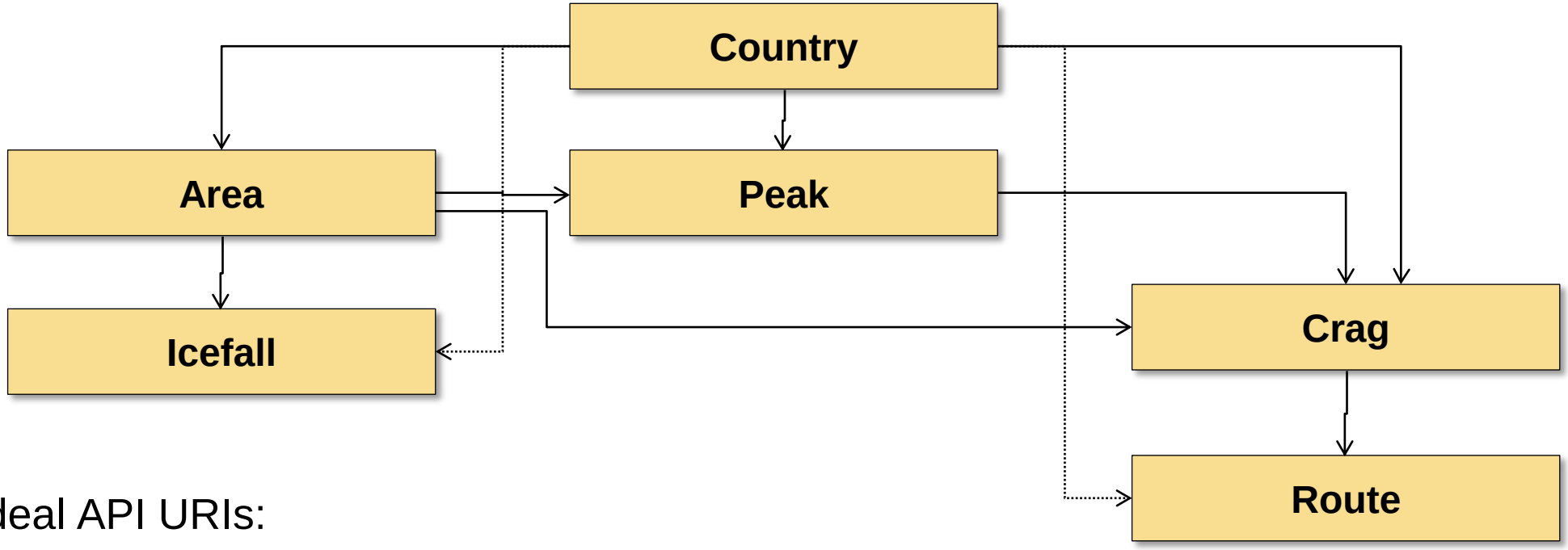
```
<?xml version='1.0' encoding='UTF-8'?>
<entry>
  <id>tag:blogger.com,1999:blog-23021255.post-8084141914020006669</id>
  <published>2012-04-21T07:51:00.000+02:00</published>
  <updated>2012-04-21T07:51:00.767+02:00</updated>
  <title type='text'>Interesting links</title>
  <content type='html'>...</content>
  <link rel='replies' type='application/atom+xml' href='...'
        title='Post Comments'/>
  <link rel='replies' type='text/html' href='...'/>
  <link rel='edit' type='application/atom+xml' href='...'/>
  <link rel='self' type='application/atom+xml' href='...'/>
  <link rel='alternate' type='text/html' href='...' title='...'/>
</entry>
```

Child references

Alternate format

Edit URL

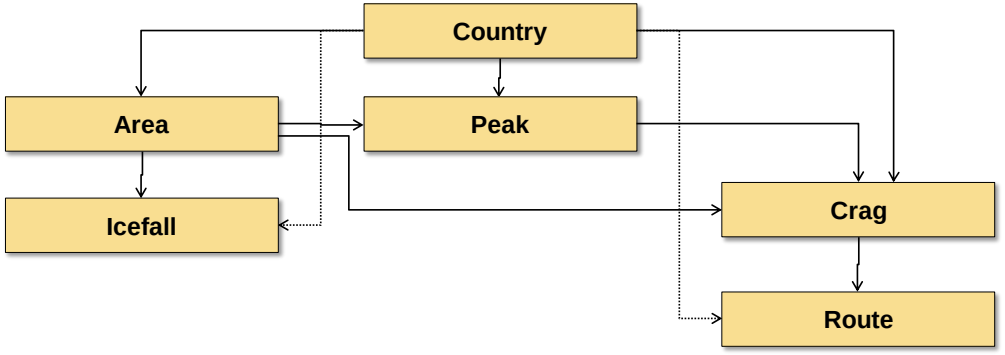
# Sample Database Schema




Ideal API URIs:

- /api/1.0/countries — list all countries
- /api/1.0/country/SI — details about Slovenia
- /api/1.0/country/SI/crags — crags in Slovenia
- /api/1.0/crag/<id>/routes — routes in crag *id*
- /api/1.0/route/<id> — details about route *id*

# Real-Life API URIs




/path/xml_countries

xml_areas?country=*ISO*

xml_showArea?area=*id*

xml_crags?country=*ISO*&select=*type*

xml_showCrag?crag=*id*

Challenges: nice-looking URIs

- Hard to implement in most scripting languages
- Require tight integration (mod_perl) or configuration rewrites

# Example: List of Countries

```
▼<CountryList>
    <country code="AL" name="Albanija" edit="A"/>
    <country code="AT" name="Avstrija" edit="A"/>
    <country code="BG" name="Bolgarija" edit="A"/>
    <country code="BO" name="Bolivija" edit="A"/>
    <country code="CZ" name="Češka" edit="A"/>
    <country code="ME" name="Črna gora" edit="A"/>
    <country code="FR" name="Francija" edit="A"/>
    <country code="GR" name="Grčija" edit="A"/>
    <country code="HR" name="Hrvaška" edit=""/>
    <country code="IN" name="Indija" edit="A"/>
    <country code="IT" name="Italija" edit="A"/>
    <country code="JO" name="Jordanija" edit="A"/>
    <country code="CA" name="Kanada" edit="A"/>
    <country code="CN" name="Kitajska" edit="A"/>
    <country code="LA" name="Laos" edit="A"/>
    <country code="MK" name="Makedonija" edit="A"/>
    <country code="MA" name="Maroko" edit="A"/>
    <country code="MX" name="Mehika" edit="A"/>
    <country code="DE" name="Nemčija" edit="A"/>
    <country code="NP" name="Nepal" edit="A"/>
```

# Example: List of Crags

```
<CragList country="SI" order="area">
  <country code="SI" name="Slovenija"/>
  <area id="721" name="Dolenjska" type="G" usage="*">
    <area id="722" name="Kolpa" type="V" usage="*"/>
    <crag id="824" extid="" name="Kuželjska stena" peak="53"/>
    <crag id="913" extid="" name="Loška stena, jugovzhodna stena" peak="79"/>
    <crag id="896" extid="" name="Loška stena, jugozahodna stena" peak="79"/>
    <crag id="912" extid="" name="Loška stena, južna stena" peak="79"/>
    <crag id="600" extid="" name="Luknja" peak=""/>
  </area>
  <area id="666" name="Gorenjska" type="G" usage="*">
    <crag id="573" extid="" name="Bodešče" peak=""/>
    <crag id="592" extid="" name="Kamnitnik" peak=""/>
    <crag id="595" extid="" name="Kovačevec" peak=""/>
    <crag id="604" extid="" name="Matjaževe kamre" peak=""/>
    <crag id="615" extid="" name="Pod Sušo" peak=""/>
    <crag id="1214" extid="" name="Podljubelj" peak=""/>
    <crag id="638" extid="" name="Turnc" peak=""/>
    <crag id="648" extid="" name="Zminec" peak=""/>
  </area>
```

# Example: Crag Data

```
▼<crag>
   <CragID>824</CragID>
   <CragCountry>SI</CragCountry>
   <CragName>Kuželjska stena</CragName>
   <ParentID>721</ParentID>
   <orient>S</orient>
   <accessTime>1</accessTime>
  ▼<access rich="yes">
     Izhodišče za steno je zaselek Ograja, ki se nahaja tik pred vasjo Kuže
     Gospodar je lovec in z veseljem nam bo pokazal svoje trofeje, med kate
     desno (gledano navzgor) ob JZ razu .
    ▼<html>
       Izhodišče za steno je zaselek Ograja, ki se nahaja tik pred vasjo Ku:
       Gospodar je lovec in z veseljem nam bo pokazal svoje trofeje, med ka
       <br/>
       <br/>
       Od tam poševno levo na zaobljen hrbet in po njem do stene.
       <br/>
       <br/>
```

**XML attributes or child tags? It depends ...**

# Using APIs

# Client-Side Example (HTML)

```
function showAscents() {
  $('#insertClimbData')
        .html($('#insertClimbDataWait').html())
        .load('script?format=HTML&routeID='+routeID);
}
```

Use jQuery.load method to load server-side HTML into DOM object

# Client-Side Example (XML)

```
function loadClimbsTable() {

  function loadClimbsData(data) {
    $.each(data.documentElement.getElementsByTagName("climb"),
      function(key,climb) {
        $("#r_"+climb.getAttribute("routeID“))
          .html("(<a href=‘script?id="+climb.getAttribute("id")+"'>"+
                climb.getAttribute("date")+"</a>)");
      });
  }

  $.ajax({
    url: “script?cragID="+cragID,
    dataType: "xml",
    timeout: 2000,
    success: loadClimbsData});
}
```

**Option: use $(data).find(“climb”).each**

**Option: use $(this).attr(*value*)**

# Going on a Tangent: Loading Indicator

```
function loadAPIData {


  var loading = $("#loadWait");
  loading.show();
  $.ajax({
    url: “script", dataType: "xml", timeout: 2000,
    error: function(data) {
          loading.hide(); ... Error processing ... }
    success: function(data) {
          loading.hide(); ... Process data ... }
  });
}
```

**Exercise: dim the screen and show centered *Loading* indicator**

# Cross-Domain API Calls Used to Work

- GET never worked, POST did

```
<form id="register"
        action=”http://cms.ipspace.net/bin/wg/register_agent"
        onsubmit="return doRegister()”>

<script>
function doRegister() {
  dataString = $("#register").serialize();
  $.post(frm.attr("action"),dataString,function(data) { …});
  return false;
}
</script>
```

# Workaround: JSONP

- Create a dynamic SCRIPT element instead of using XMLHttpRequest
- GET only, no POST
- Requires a wrapper function (specified in URL)

Example: www.example.com/getUser?callback=foo returns

```
foo({
  "firstName": "John",
  "lastName" : "Smith",
  "age" : 25,
});
```

# Solution: Cross-Origin Resource Sharing (CORS)

- Simple GET and POST requests are sent with **Origin:** header
  **Warning:** check the Origin header if you're worried about CSRF
- Share the results only if the response includes **Access-Control-Allow-Origin** header

More complex requests are *pre-flighted*

- Send OPTIONS request with **Access-Control-Request-*** headers and form data
- Check response status (must be 200)
- Check **Access-Control-Allow-*** headers
- Do the actual request

**Warning: jQuery generates requests that require pre-flighting**

# Server-Side API Call (PERL)

```
sub getRequest() {
  my ($url,$params) = @_ ;
  if ($params) { $url .= “?".$params; }
  my $response = $lwp->get($url);
  if ($response->is_success) {
    return $response->decoded_content;
  } else {
    die $response->status_line;
  }
}
```

# Server-Side API Call (ASP, XML)

```
Function GetExternalXML(URL)
  Dim HttpReq,xml
  On Error Resume Next

  Set HttpReq = Server.CreateObject("MSXML2.ServerXMLHTTP")
  HttpReq.open "GET", URL, False
  HttpReq.send
  If Err.Number <> 0 Then GetExternalXML = "Error: " & Err.Description, False : Exit Function
  On Error Goto 0

  If HttpReq.status <> 200 Then
    GetExternalXML = "HTTP request failed " & CStr(HttpReq.Status) : Exit Function
  Else
    Set xml = HttpReq.responseXML
    If xml.parseError.errorCode <> 0 Then
      GetExternalXML = "XML parsing failed: " & CragDoc.parseError.reason
    Else
      Set GetExternalXML = HttpReq.responseXML
    End If
  End If
End Function
```

# API Authentication

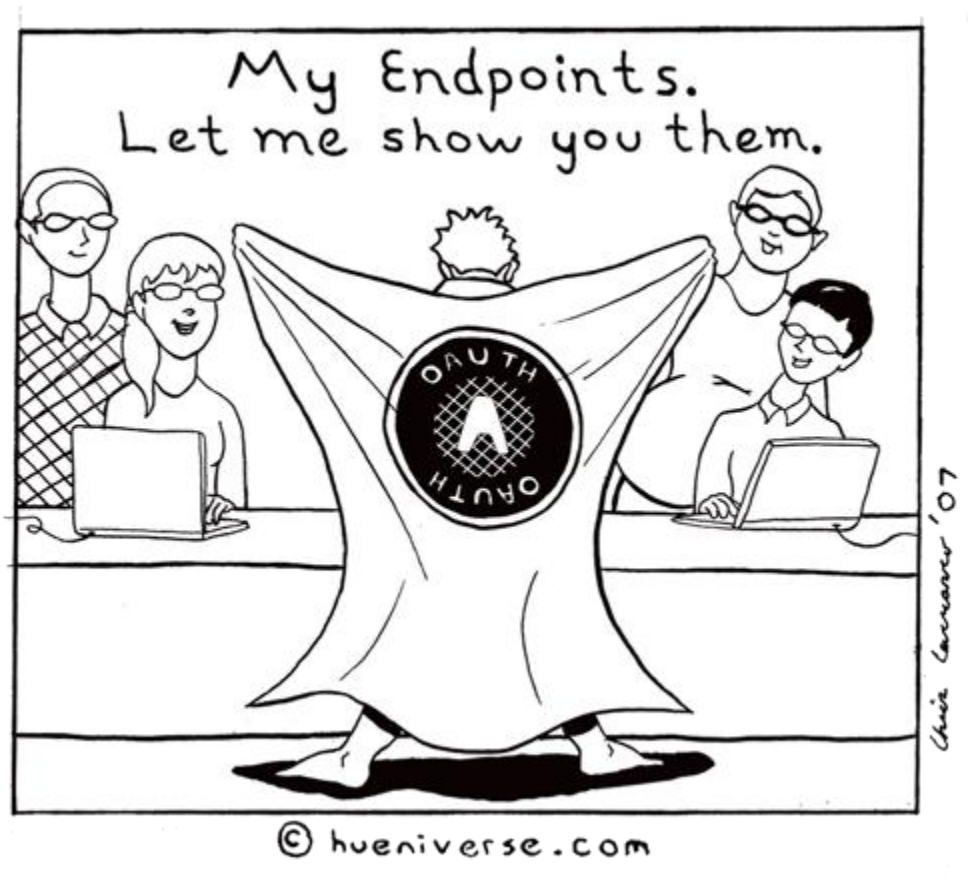
# Authentication Basics

Client-Server API

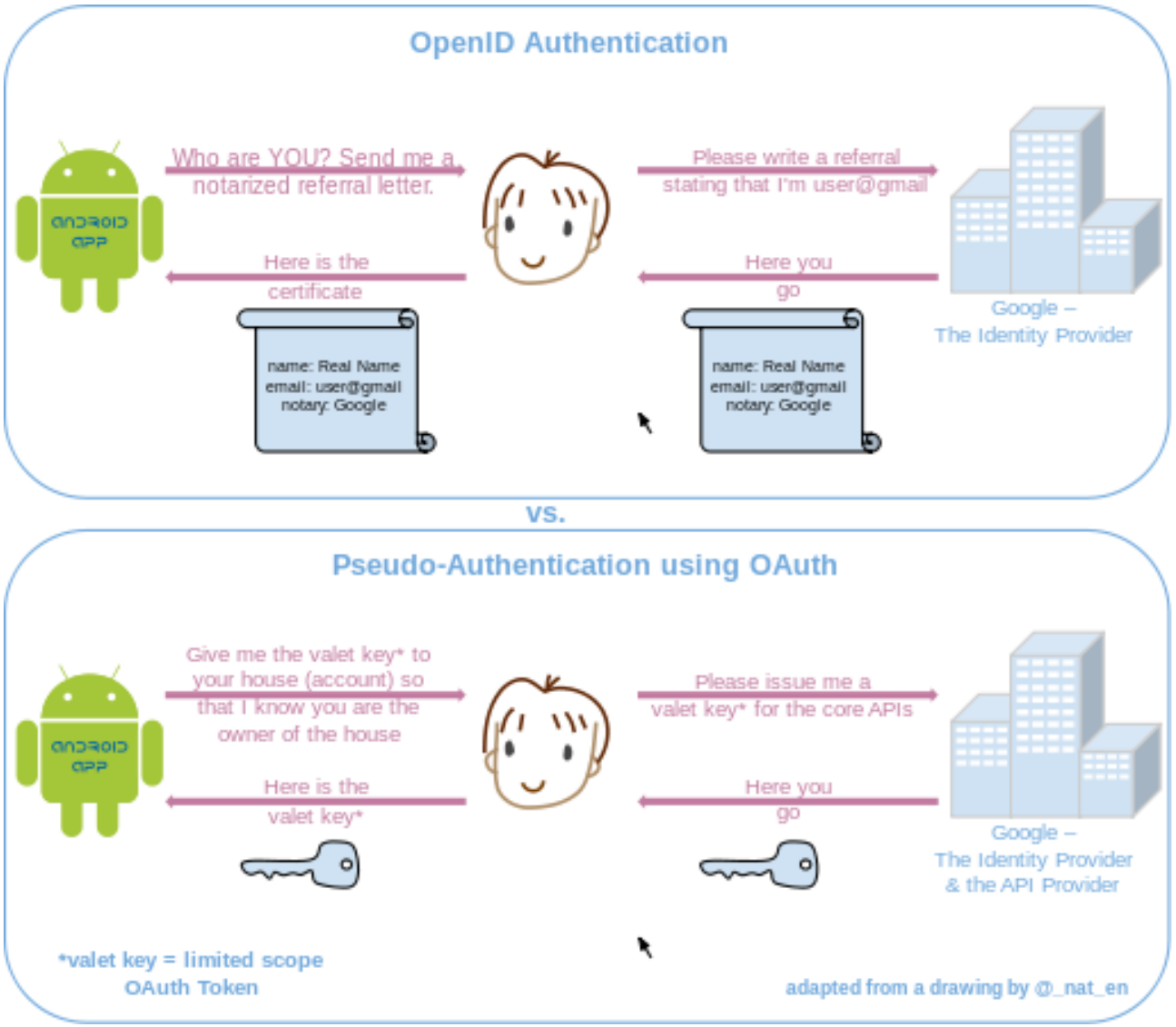
- Use regular authentication mechanisms (ex: session cookie)

Server-to-Server API

- Application key
- Login credentials (session cookie)
- Third-party authentication (OAuth 2.0)

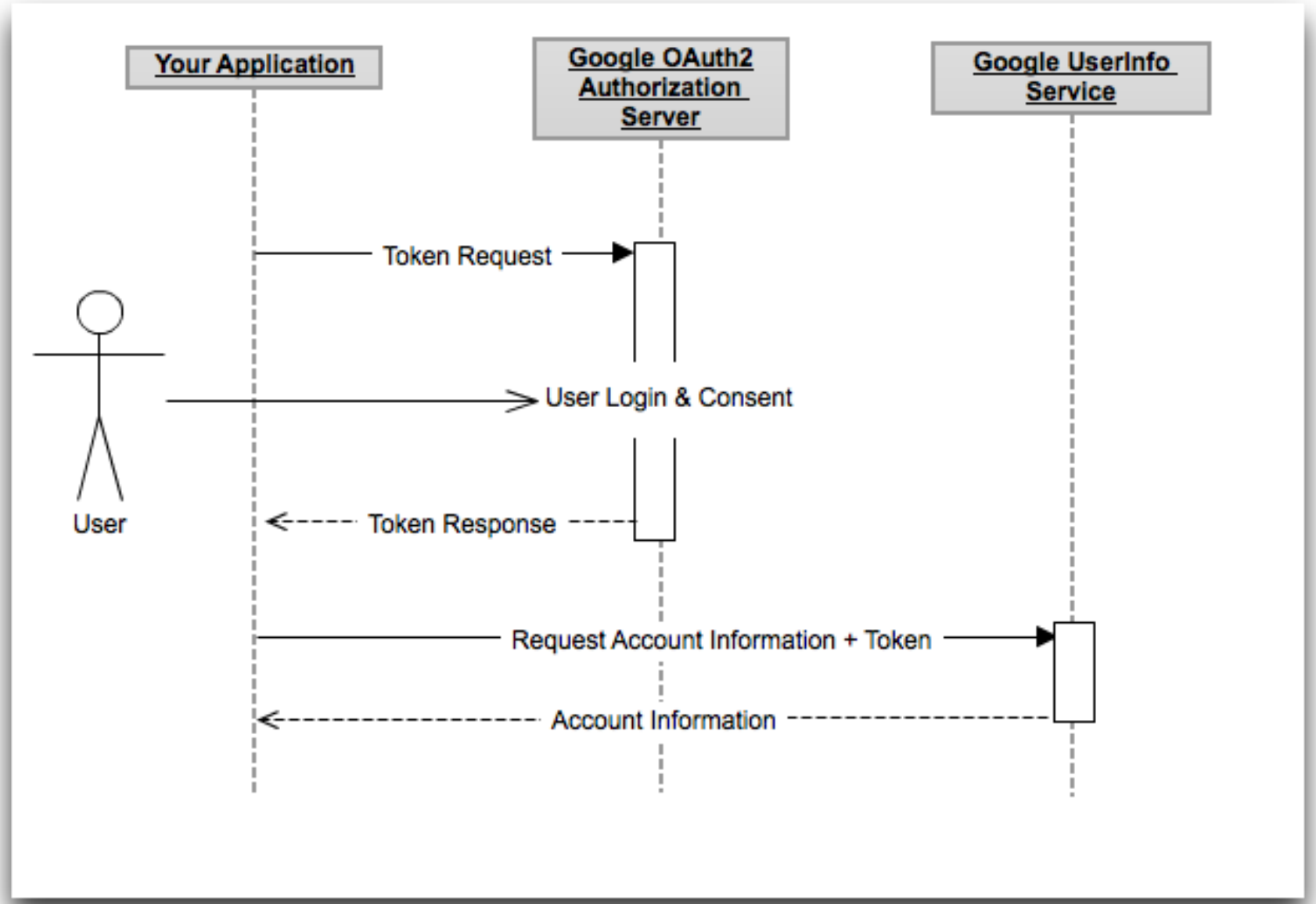


# OAuth versus OpenID



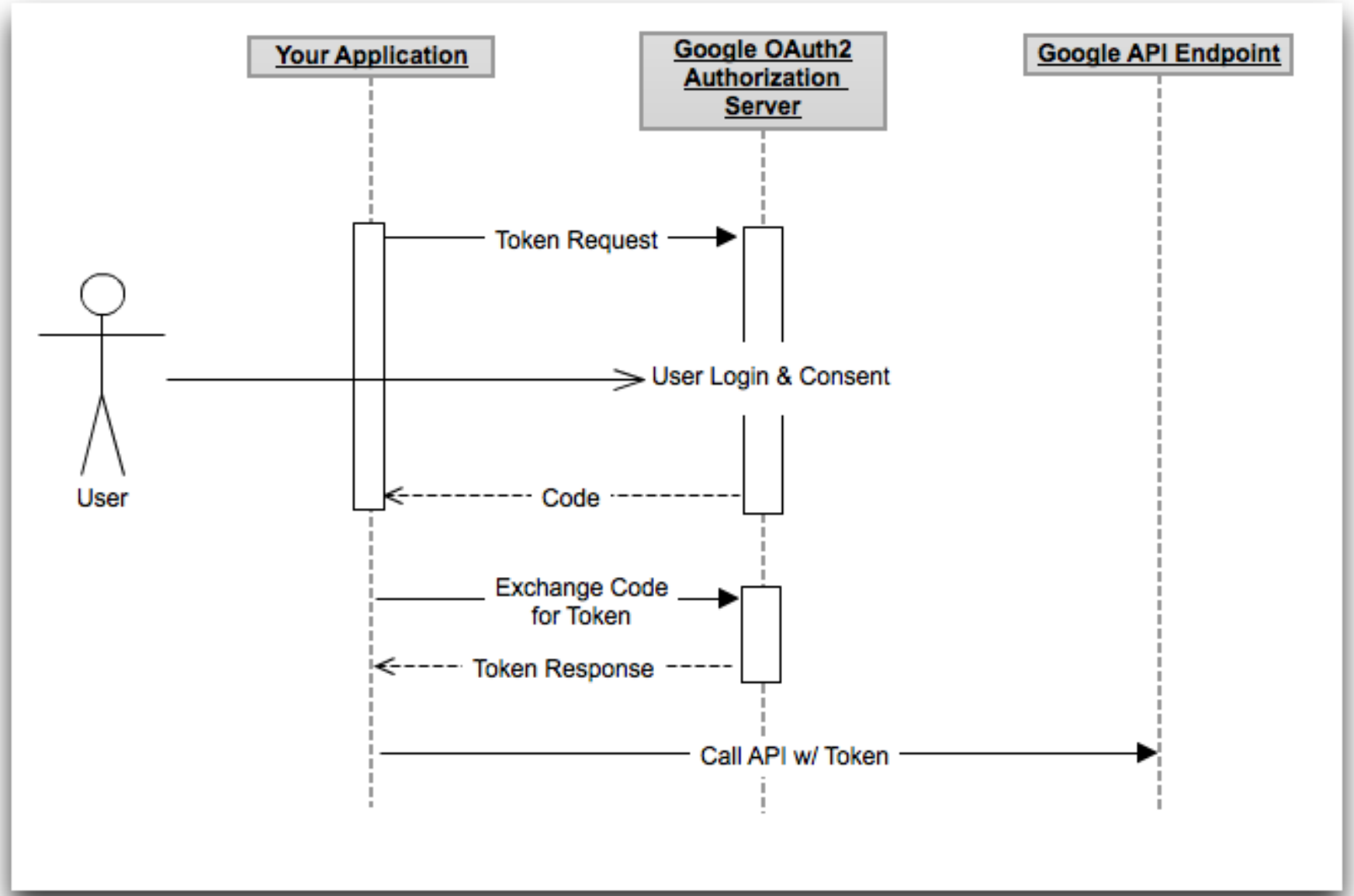

**Source: en.wikipedia.org/wiki/OAuth**

# OAuth Authentication



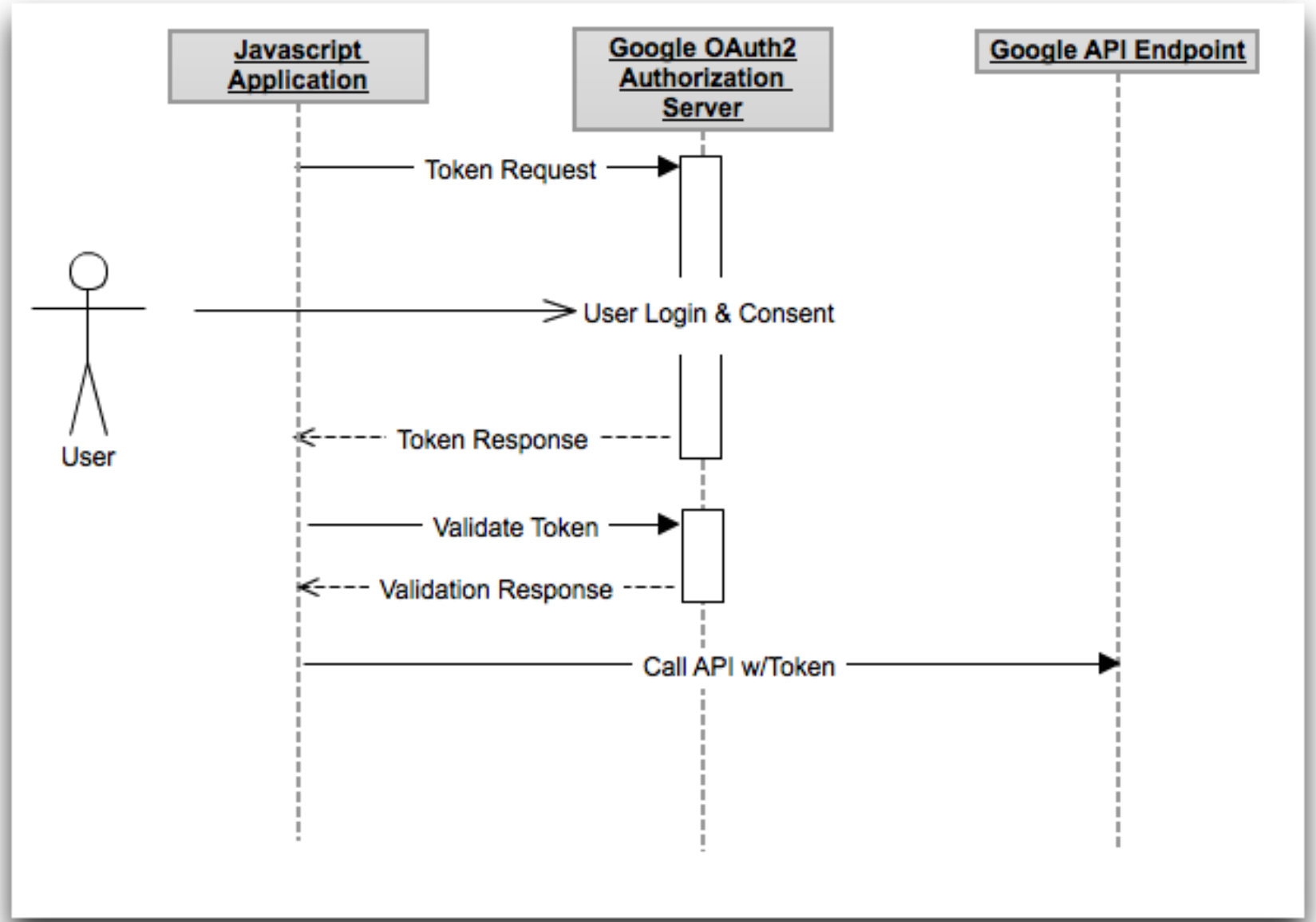

Source: developers.google.com/accounts/docs/OAuth2

# OAuth Web Server and Device Application




Source: developers.google.com/accounts/docs/OAuth2

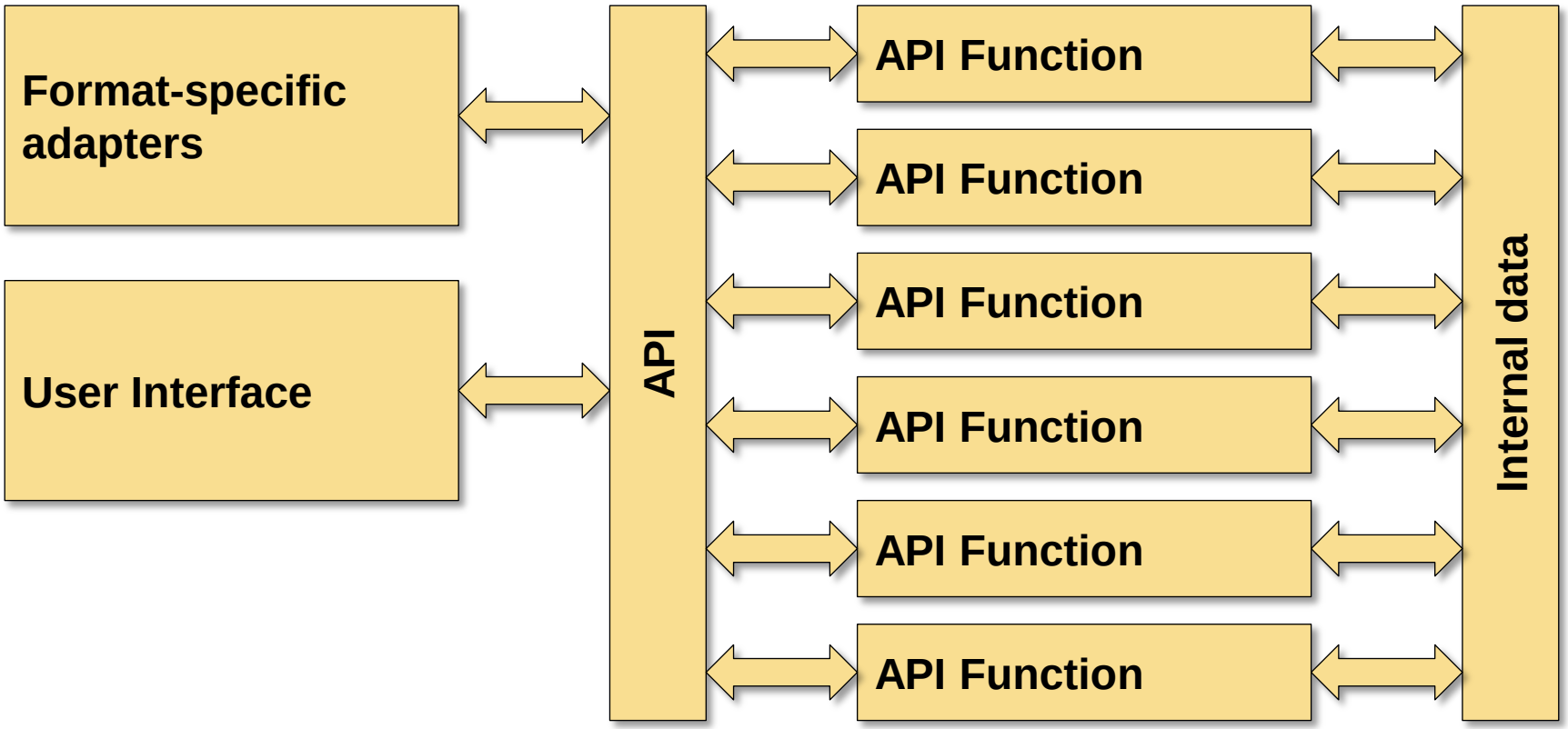
# OAuth Web Client Application



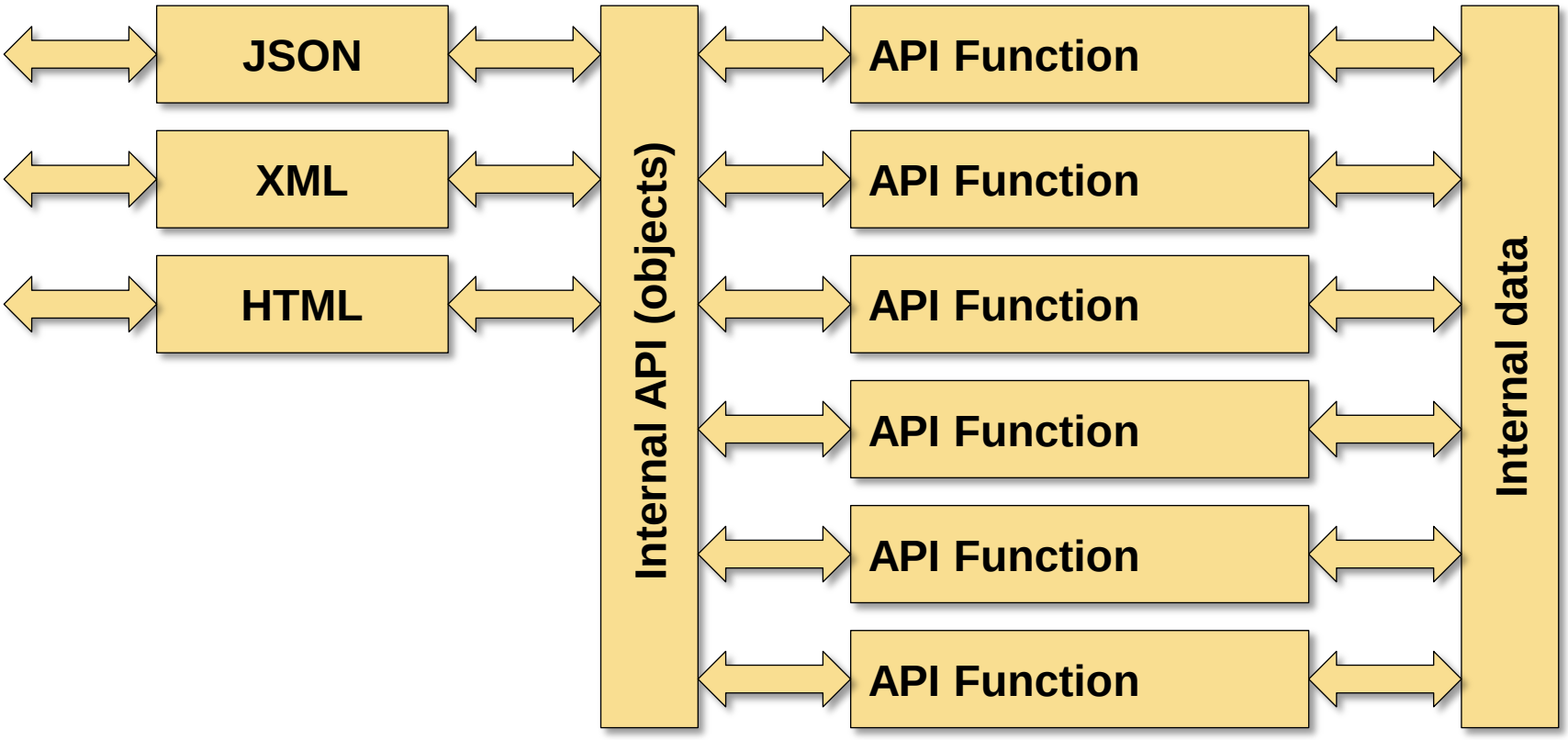

Source: developers.google.com/accounts/docs/OAuth2

# Application Design with APIs

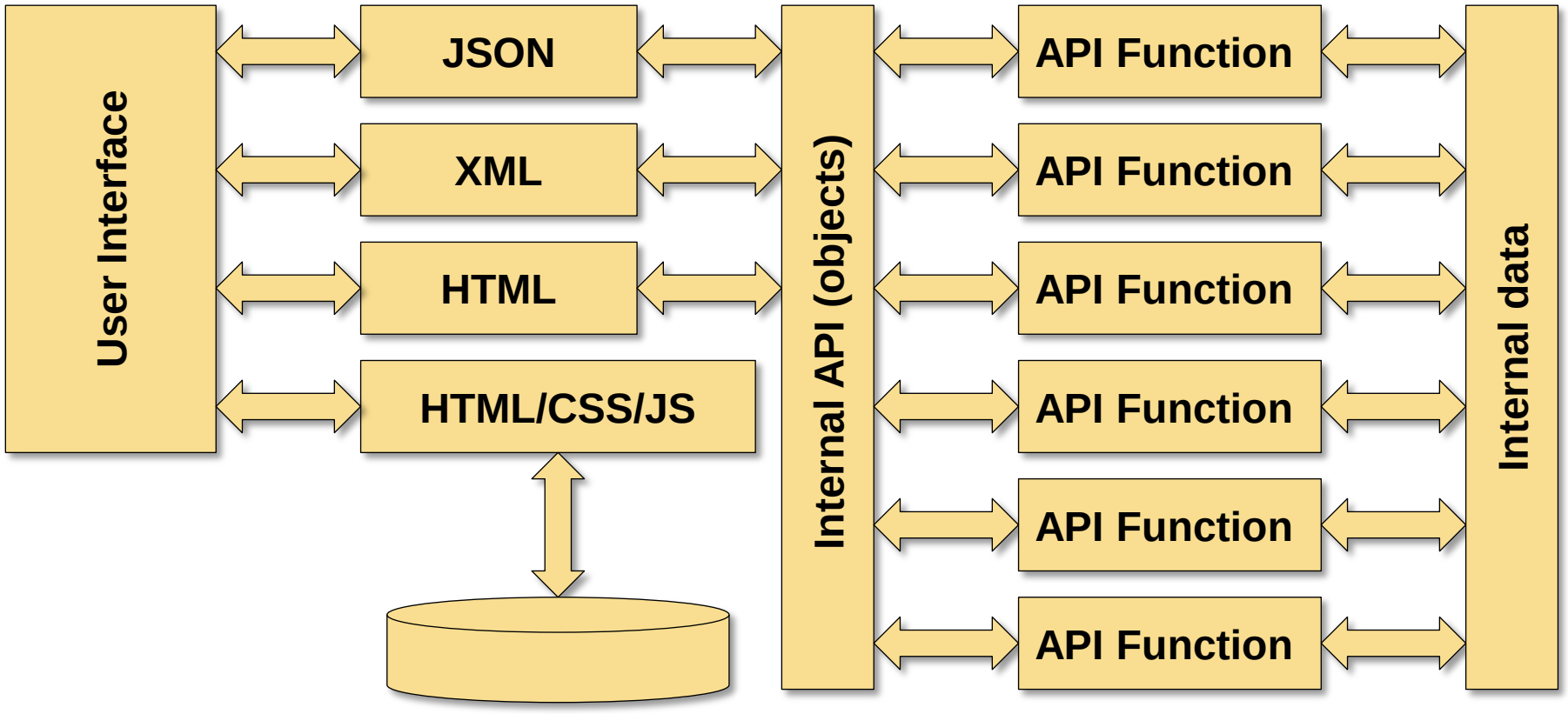
# Overall Application Structure



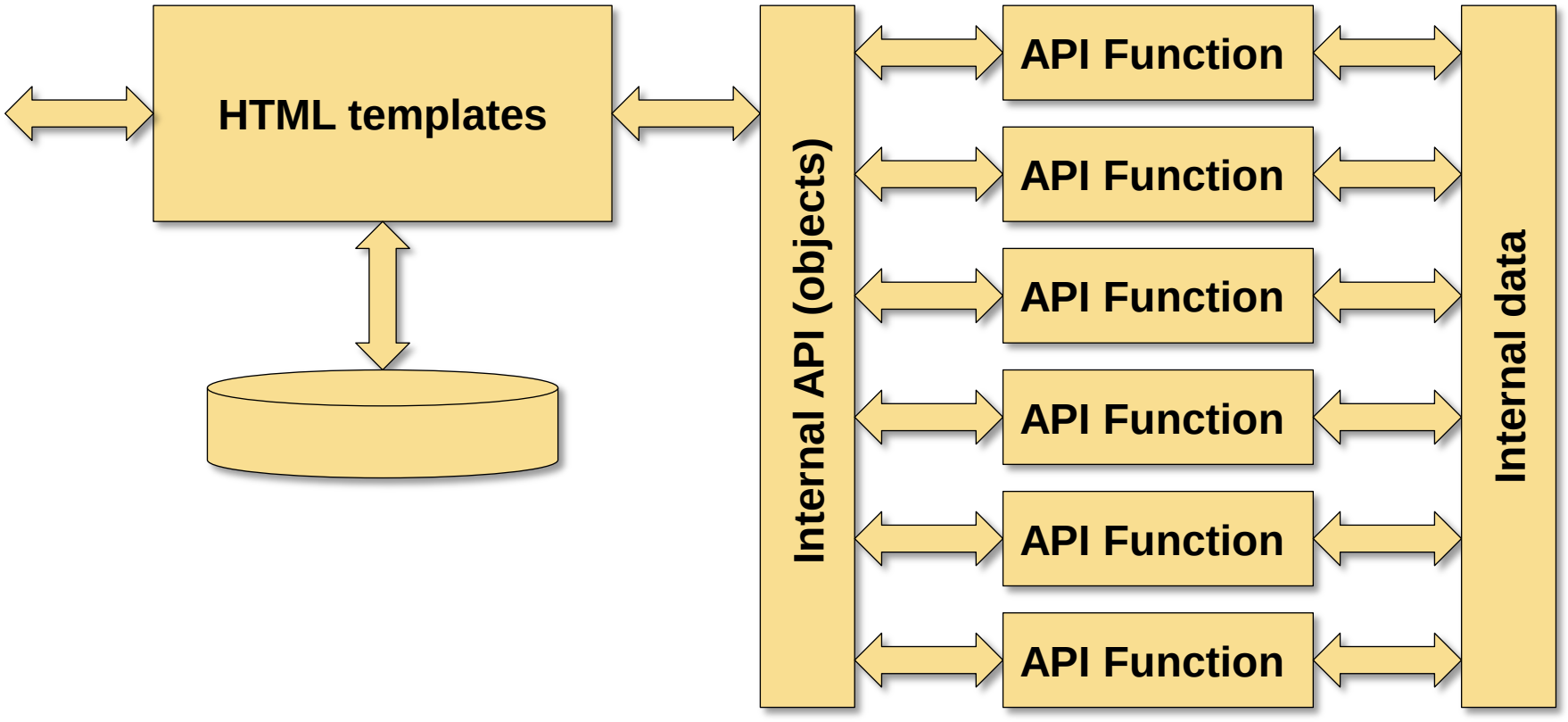
# HTTP API



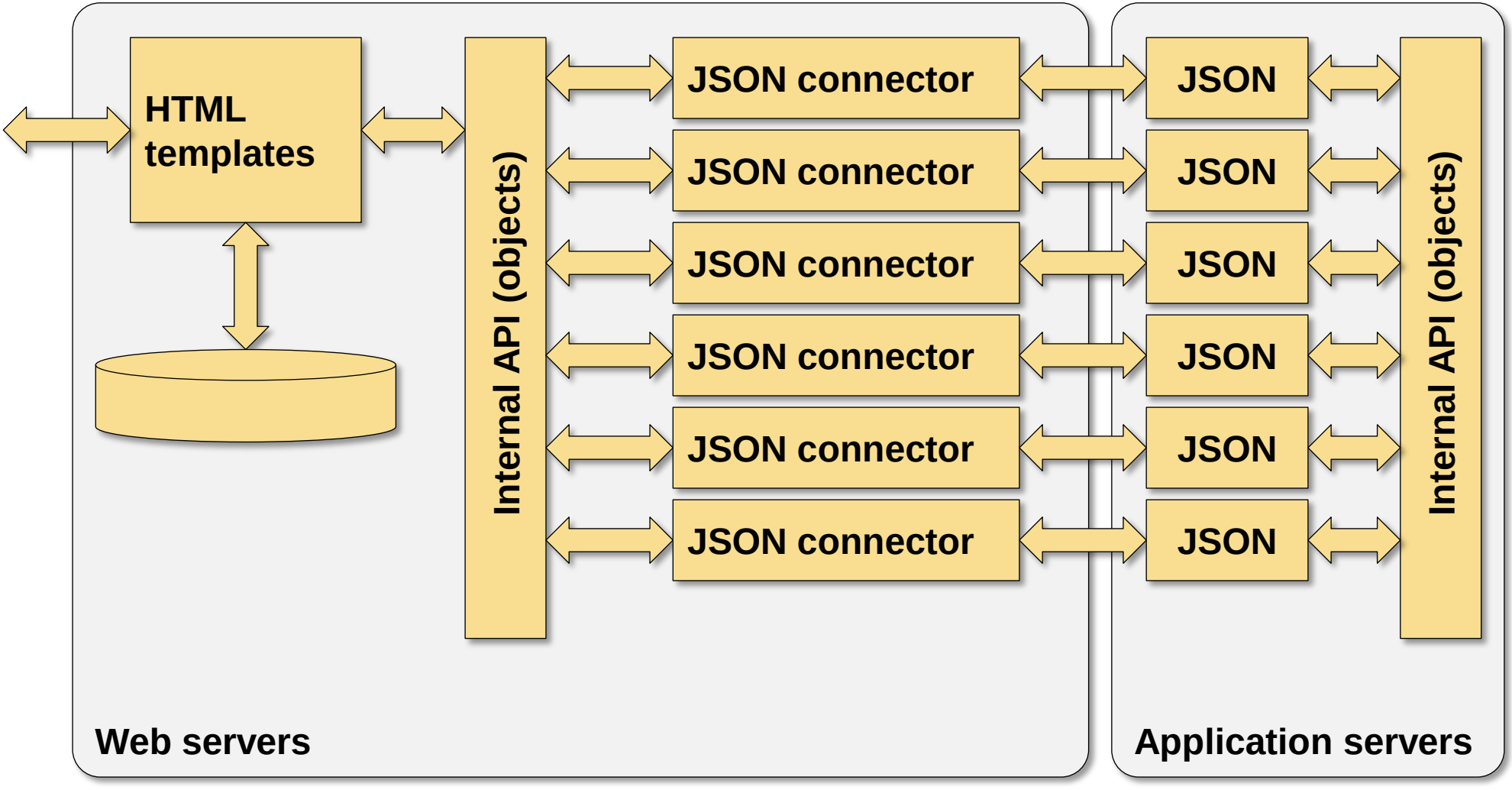
# Client-Side User Interface



# Server-Side HTML Generation



# Server-Side HTML Generation With Scale-Out Design



# Questions?