Server Side
Data Access, Authentication

IF3110 – Web-based Application Development
School of Electical Engineering and Informatics
Institut Teknologi Bandung

Objective

- Students understand the role of data access on web application
- Students understand the concepts of authentication and relevant technologies.

What we have learned so far

- HTTP Protocol
- State-handling
- Form handling
- Templating

Typical Web App Processing

Source:

Shklar, L.
Web Application Architecture: Principles, Protocols and Practices
Wiley Publishing, Inc., **2003**

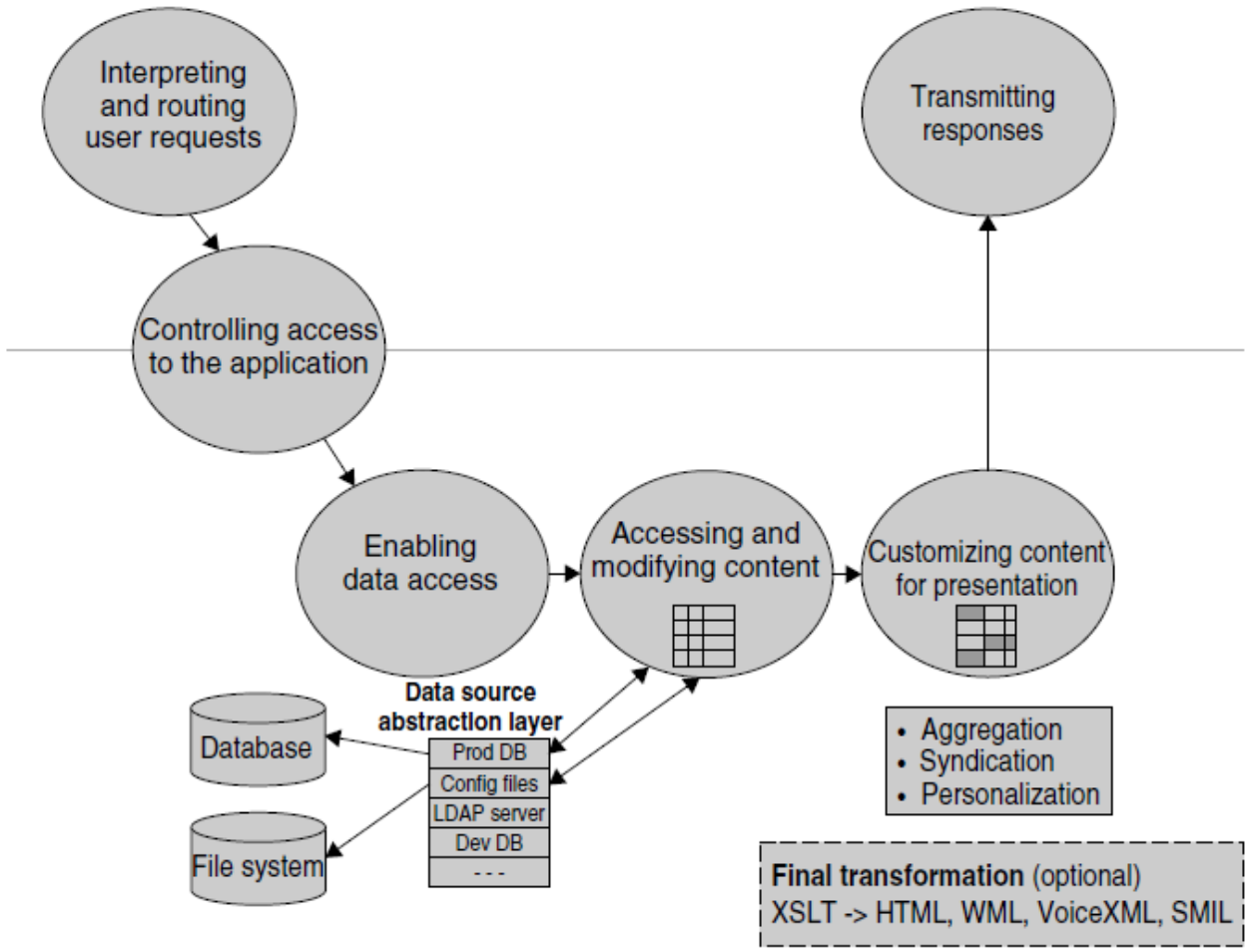


Figure 8.3 Processing flow in a typical Web application (Above the grey line—Web server; below the grey line—Web application)

Reference

- Learning PHP, David Sklar, Oreilly

Data Access

- Why do we need to have a data access mechanisms
  - Remember your users (and their attributes)
  - Which users do these transactions
  - The “stuff” that our site manages/sells

- Why the following mechanisms are insufficient
  - URL parameters
  - Cookies
  - Session attributes

Possible Mechanisms

- Database
- File

Advanced

- Services
- Object Storage
- Memcache

Database

Things commonly done

- Connecting to

```
$db = new PDO
      ('mysql:host=db.example.com;dbname=dbname',
      'johndoe','secretpwd');
```

  - Setting attributes (if needed)
  - Handling errors (e.g., database failure, network failure)
- Putting Data into
  - Data modification
- Retrieving Data

Things commonly done

- Putting Data into

```
try {
     $db = new PDO('sqlite:/tmp/restaurant.db’);
     $db->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
     $affectedRows = $db->exec("INSERT INTO dishes
     (dish_name, price, is_spicy) VALUES ('Sesame Seed
          Puff', 2.50, 0)");
} catch (PDOException $e) {
     print "Couldn't insert a row: " . $e->getMessage();
}
```

What it does?

```
try {
     $db = new PDO('sqlite:/tmp/restaurant.db’);
} catch (PDOException $e) {
     print "Couldn't connect: " . $e->getMessage(); }
     $result = $db->exec("INSERT INTO dishes (dish_size,
dish_name, price, is_spicy) VALUES ('large', 'Sesame Seed
Puff', 2.50, 0)");
if (false === $result) {
     $error = $db->errorInfo();
     print "Couldn't insert!\n";
     print "SQL Error={$error[0]}, DB Error={$error[1]},
Message={$error[2]}\n"; }
```

Things commonly done

- Data Modification

```
$rows = $db->exec("UPDATE dishes SET is_spicy = 1 WHERE
dish_name = 'Eggplant with Chili Sauce'");
```

- Data Deletion

```
if ($make_things_cheaper) {
     $rows = $db->exec("DELETE FROM dishes WHERE price >
19.95");
} else { // or, remove all dishes
     $rows = $db->exec("DELETE FROM dishes");
}
```

- `$rows` is number of affected rows

Things commonly done

- Retrieving Data

```
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch()) {
     print "$row[dish_name], $row[price] \n";
}
```

- What is the difference from

```
$q = $db->query('SELECT dish_name, price FROM dishes’);
$rows = $q->fetchAll();
```

Which one is better?

Things commonly done

- Retrieving Data **as OBJECT**

```
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch(PDO::FETCH_OBJ)) {
    print "{$row->dish_name} has price {$row->price} \n";
}
```

Things commonly done… more

- Data Insertion from Request Attributes (e.g., $_POST)

```
$stmt = $db->exec('INSERT INTO dishes
      (dish_name) VALUES (…)');
```

- Be careful …for injection, imagine
  - $_POST[dish_name] = 'Fried Rice'
  - $_POST[dish_name] = 'Uncle Wong's Fried Rice')

```
$stmt = $db->prepare('INSERT INTO dishes
      (dish_name) VALUES (?)');
$stmt->execute(array($_POST['dish_name']));
```

  - Not using `exec()` or `query()` but using `prepare()` `execute()`

Things commonly done… more

- Putting data in from a web form
  1. Read request parameters as inputs
  2. Validate and process the inputs
  3. Sanitize the inputs (before being SQL parameters)
  4. Prepare the SQL Statements (DML)
  5. Put the inputs as the parameters of the prepared statement
  6. Execute

Things commonly done… more

- Presenting data in a web from database
  1. Read request parameters as inputs
  2. Validate and sanitize the inputs
  3. Prepare the SQL Statements (Query)
  4. Put the inputs as the parameters of the prepared statement
  5. Execute
  6. Put select resultset into a template of the web form

Things commonly done… more (retrieval)

```
$stmt = $db->prepare($sql);
$stmt->execute(array($input['min_price'],
$input['max_price']));
$dishes = $stmt->fetchAll();
if (count($dishes) == 0) {
    print 'No dishes matched.';
} else {
    print '<table>';
    print '<tr><th>Dish Name</th><th>Price</th><th>Spicy?
</th></tr>';
    foreach ($dishes as $dish) {
        if ($dish->is_spicy == 1) {
            $spicy = 'Yes'; } else { $spicy = 'No'; }
        printf('<tr><td>%s</td><td>$%.02f</td>
    <td>%s</td></tr>', htmlentities($dish->dish_name),
$dish->price, $spicy); }}
```

Remarks

- Often you only have a single instance of SQL DBMS
  - Chocking point to the Web App Performance, Single Point of Failure
- Be careful with SQL wildcard
  - Lead to slow query
  - Unnecessary big result sets
- Never trust other inputs
  - Sanitize before store in Database
  - Sanitize/escape before printing to client (after retrieval from database)
- Open and Close Database Connection wisely

File

Things commonly done

- Reading a File
- Writing a File
- Working with CSV

Read a File

```
$page = file_get_contents('page-template.html’);
$page = str_replace('{page_title}', 'Welcome', $page);
if (date('H' >= 12)) {
     $page = str_replace('{color}', 'blue', $page);
} else { $page = str_replace('{color}', 'green', $page); }

$page = str_replace('{name}', $_SESSION['username'],
$page);
print $page;
```

- What it does?
- Handle the error/exception wisely

Write a File

```
$page = file_get_contents('page-template.html’);
$page = str_replace('{page_title}', 'Welcome', $page);
if (date('H' >= 12)) {
     $page = str_replace('{color}', 'blue', $page);
} else { $page = str_replace('{color}', 'green', $page); }

$page = str_replace('{name}', $_SESSION['username'],
$page);
file_put_contents('page.html', $page);
```

- What it does?
- Handle the error/exception wisely

Read a File

```
$fh = fopen('people.txt','rb');
while ((! feof($fh)) && ($line = fgets($fh))) {
     $line = trim($line);
     $info = explode('|', $line);
     print '<li><a href="mailto:' . $info[0] . '">' .
$info[1] ."</li>\n";
}
fclose($fh);
```

- `rb` is a flag to read from beginning; return false if doesn't exist

Working with CSV

- How to put data in CSV into Database?
  1. Open the CSV
  2. Prepare the SQL statement
  3. Parse CSV line into array to be SQL Parameters
  4. Close file (and close DB Connection if necessary)

Remarks

- Handle file errors correctly (open, permission, space, etc.)

```
$page = file_get_contents('page-template.html’);
if ($page === false) {
     print "Couldn't load template: $php_errormsg";
} else { // ... process tem
```

```
$_POST['user’] =
‘/usr/local/data/../../../etc/passwd'
```

- Sanitized supplied filenames
  - Filename

```
$user = str_replace('/', '', $_POST['user']);
$user = str_replace('..', '', $user);
```

```
$user = $_POST['user'];
if (is_readable("/usr/local/data/$user")) {
   print 'User profile ' . htmlentities($user) .': <br/>’;
     print file_get_contents("/usr/local/data/$user"); }
```

Object Storage

- It is a computer data storage that manage data as objects, as opposed to file system, data blocks, or records
- Common flows for Upload
  - WebApp requests to Object Storage for Singed URL to upload a file
  - Client uploads the file via the Signed URL
  - Signed URL is only valid for a short of time
- Flow Download
  - WebApp requests to Object Storage for Singed URL to download a file
  - Client shows/downloads the file via the signed URL

Authentication

Authentication

- Identity Verification
  - Verify the claims

- How can Bob be sure that he is communicating with Alice?

- Three General Ways:
  - Something you ***know (i.e., Passwords)***
  - Something you ***have (i.e., Tokens)***
  - Something you ***are (i.e., Biometrics)***

Why do we need Authentication

- Web App uses an HTTP (Stateless Protocol)
- A web app is commonly used by many users
  - I am Yudis and I want a new book
  - Í am Riza, a lecture, and I need to access the lectures that I teach
  - I am Catur and I need to upload new materials

Where to store

- Cookie
- Session Attribute
- HTTP Header

How a web app do Authentication

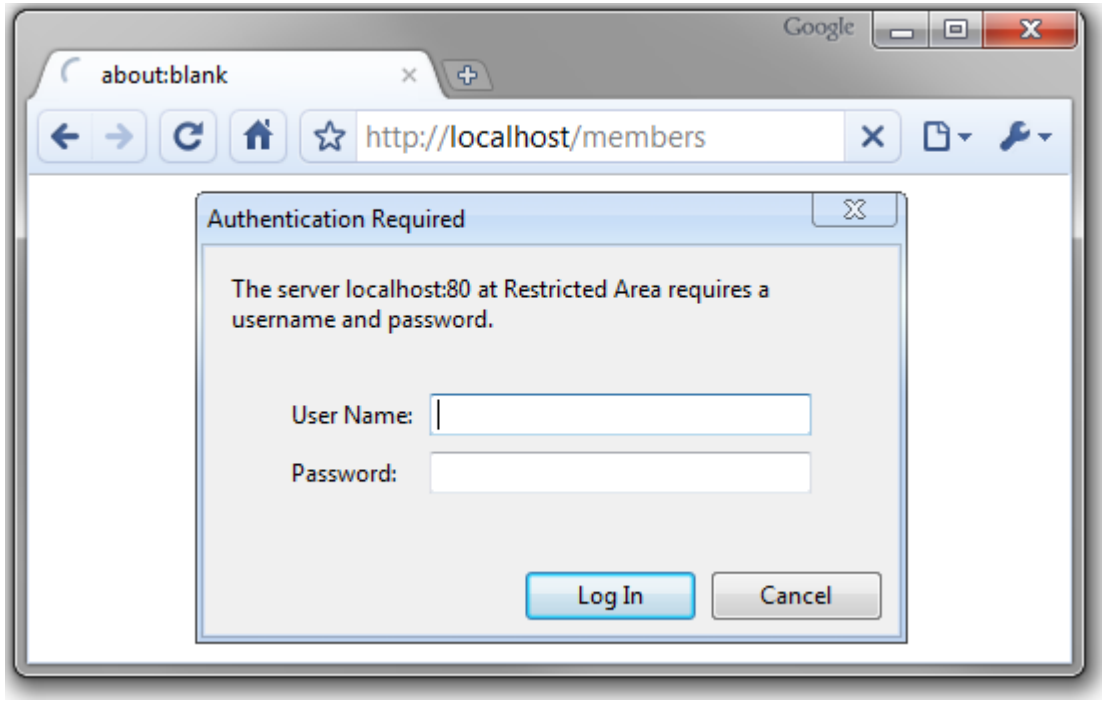
- Backed by internal user database
  - HTTP Basic Auth
  - Database-backed Auth
- External services
  - LDAP
  - OpenID+Connect (Open ID, OAuth2)
  - SAML
  - CAS
  - …

Common flow

1. Display web form (or Dialog in the case HTTP Basic Auth)
2. Enter the credential
3. Checking the form submission
4. (if the submitted credential is correct) adding the username in the session
5. ….browsing…
6. Remove the username from the session when the user logout/timeout

HTTP Basic Authentication

- PHP will define the params
  - $_SERVER['PHP_AUTH_USER']
  - $_SERVER[' PHP_AUTH_PW']
  - $_SERVER['PHP_AUTH_DIGEST']



Database-backed Authentication [simplified]

- Checking the credential in Database

```
session_start();
$password_ok = false;
$input['username'] = $_POST['username'] ?? '';
$submitted_password = $_POST['password'] ?? '';
$stmt = $db->prepare('SELECT password FROM users WHERE
username = ?');
$stmt->execute($input['username']); $row = $stmt->fetch();
if ($row) { $password_ok =
     password_verify($submitted_password, $row[0]); }
if (! $password_ok) { $errors[] = 'Please enter a valid
     username and password.'; }
else { $_SESSION['valid_user'] = $input['username'] }
```

Database-backed Authentication [simplified]

- If success

```
session_start();
if (isset($_SESSION['valid_user'])) {
     do_html_menu();
     //contents
     do_html_footer();
} else {
     //redirect to login page
}
```

- End user session

```
session_start();
unset($_SESSION['valid_user']
$res = session_destroy();
```

Database-backed Authentication [simplified]

- Insert/Update Password
  - Update query in SQL with prepared statement
  - Never store the password in plaintext
    - Hash – sha1, sha2
    - Hash+Salt – password_hash()
- Reset password
  - ?

Common Protocols on Web

Using LDAP for Authentication

- LDAP – Lightweight Directory Access Protocol
  - A “database” for user directory

```
$ldap = ldap_connect("ldap://ldap.mydomain.com") or die('Could
not connect to LDAP server.');
ldap_set_option($ldap, LDAP_OPT_PROTOCOL_VERSION, 3);
ldap_set_option($ldap, LDAP_OPT_REFERRALS, 0);
$bind = @ldap_bind($ldap, $ldapuser, $ldappass);
if ($bind) {
      $filter="(sAMAccountName=$username)";
      $result = ldap_search($ldap,"dc=MYDOMAIN,dc=COM",$filter);
      ldap_sort($ldap,$result,"sn");
      $info = ldap_get_entries($ldap, $result);
      if($info['count'] > 0)
      // exists entries
```

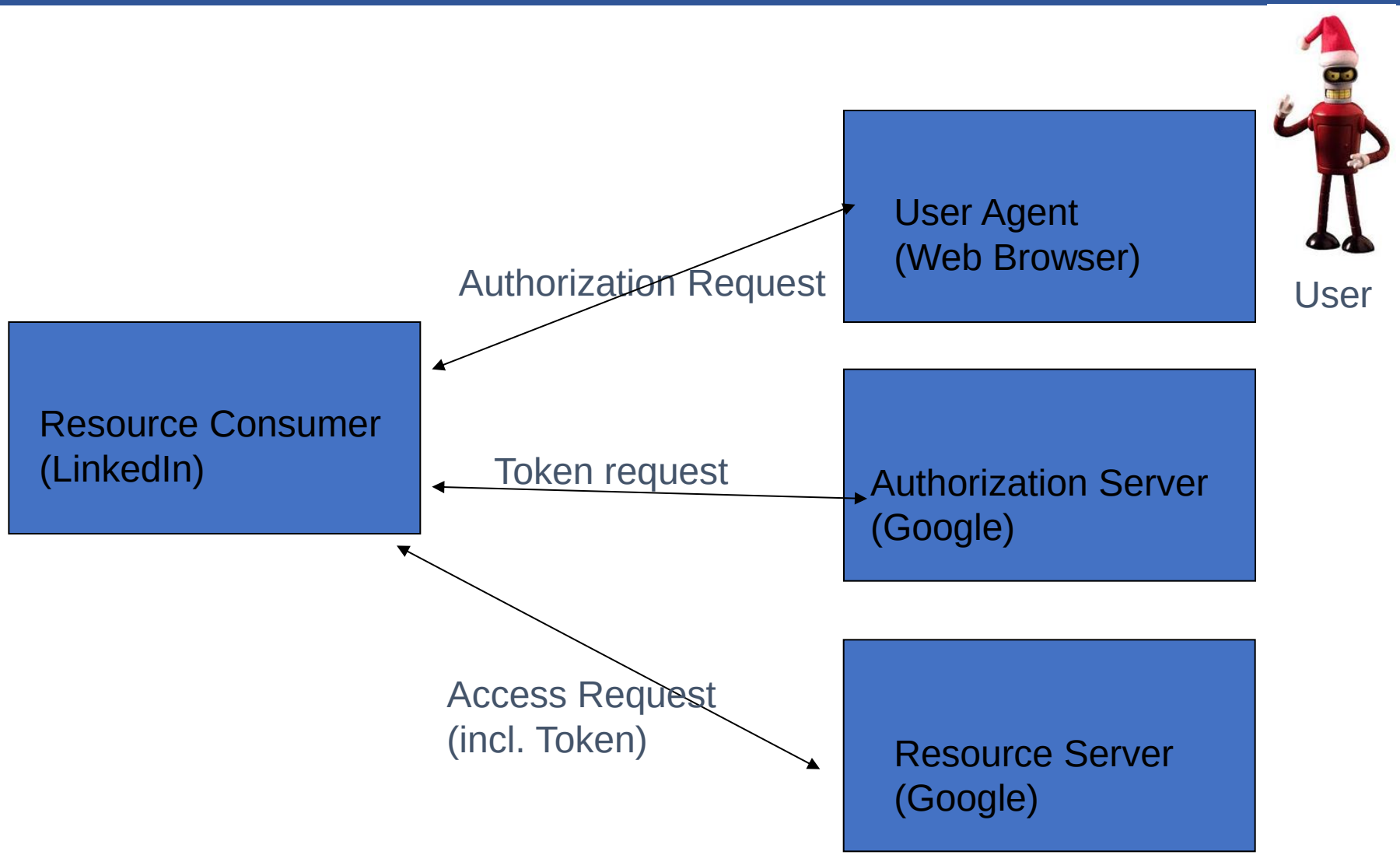
OAuth - Features

- A third party app can access user's data stored at service provider without requiring username and password.
- Delegated authorization protocol
- Explicit user consent is mandatory.
- Light-weight*
- Use Case
  - Website X can access your protected data at API Y
    - All without sharing your password off-site
    - especially when there isn't one like with OpenID
- Approach
  - Signed HTTP Requests
  - Safe, Password-less Token Exchange

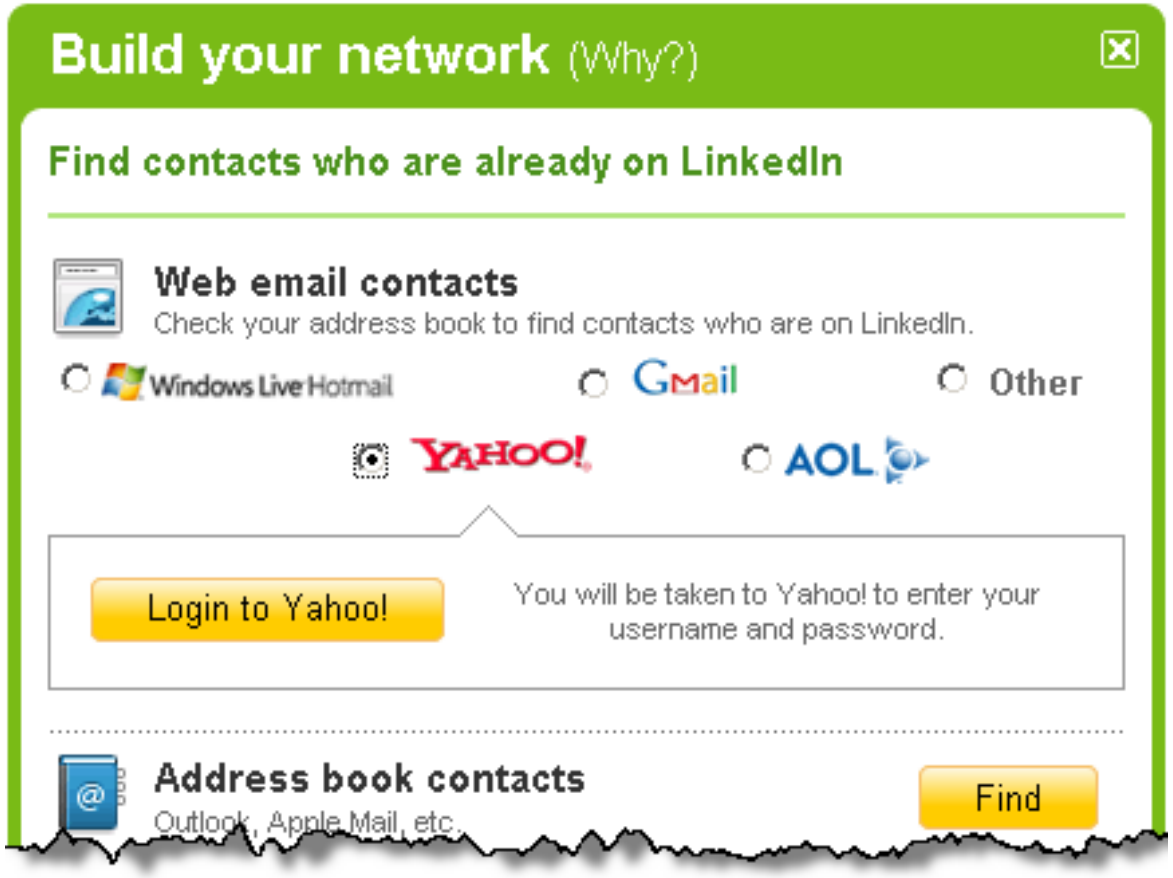
Three things

- Actors
  - User
  - Service Provider
  - Consumer
- Token
  - Access Token
  - Request Token
  - Consumer Key
- URLs
  - Request Token Issuer
  - Authorization Page
  - Access Token Exchanger

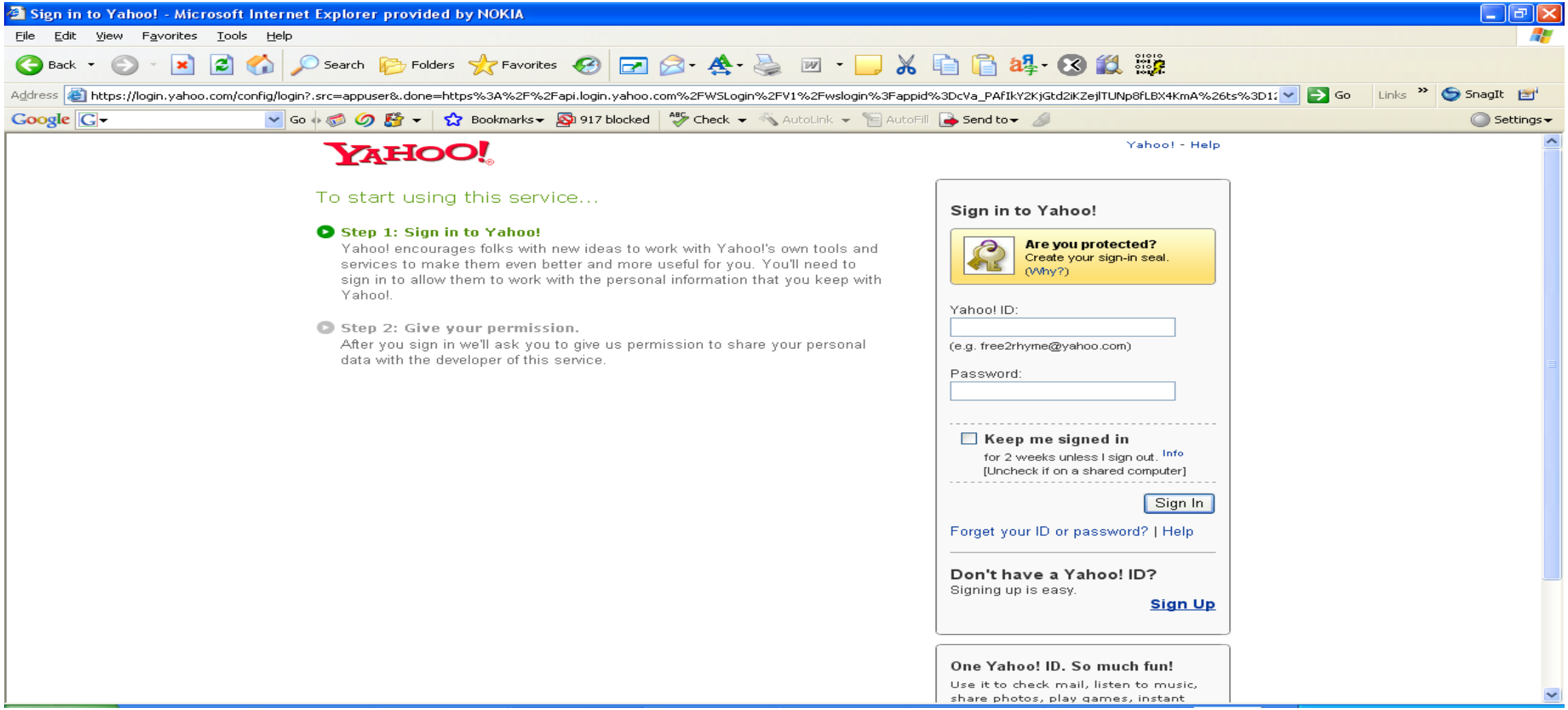
Entities


User Agent
(Web Browser)
User
Authorization Request
Resource Consumer
(LinkedIn)
Token request
Authorization Server
(Google)
Access Request
(incl. Token)
Resource Server
(Google)

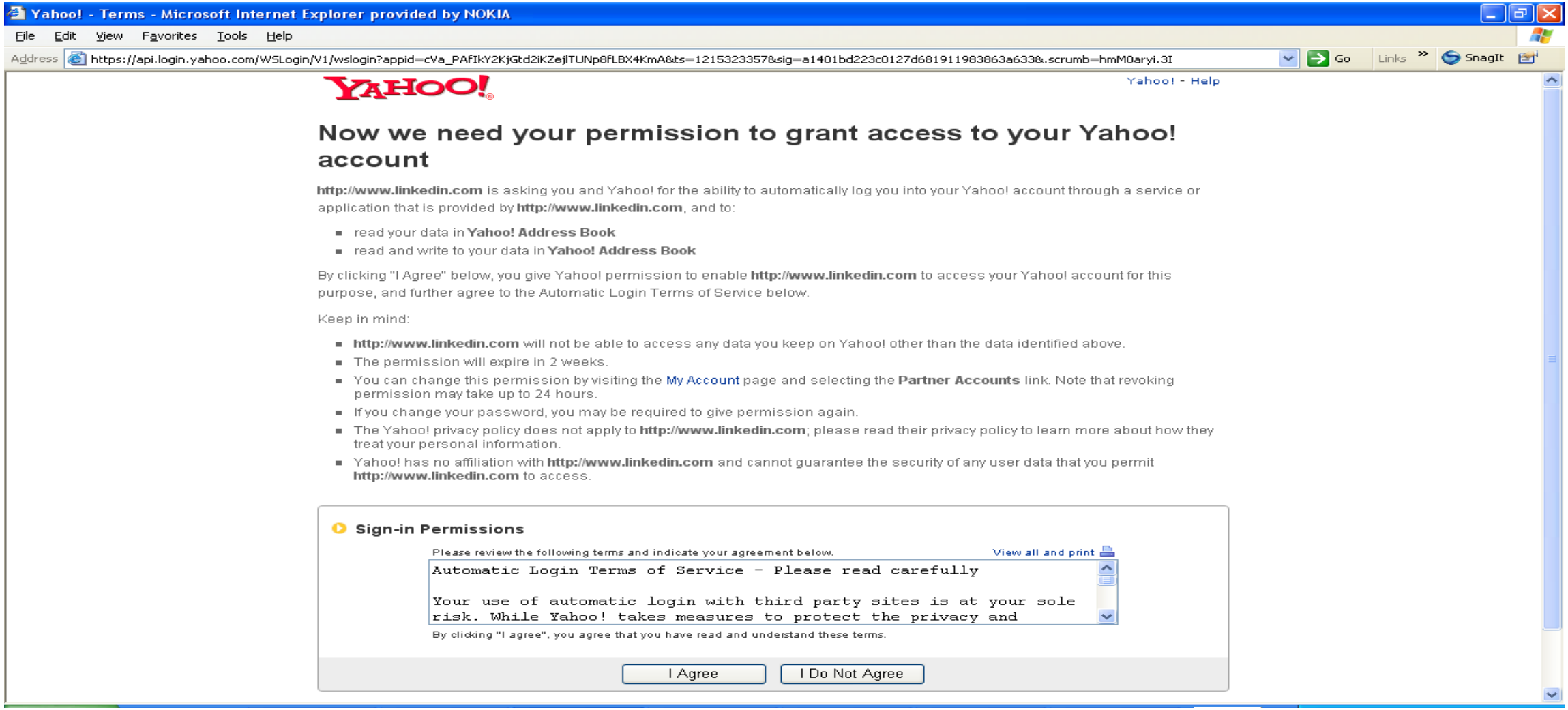
User navigates to Resource Client



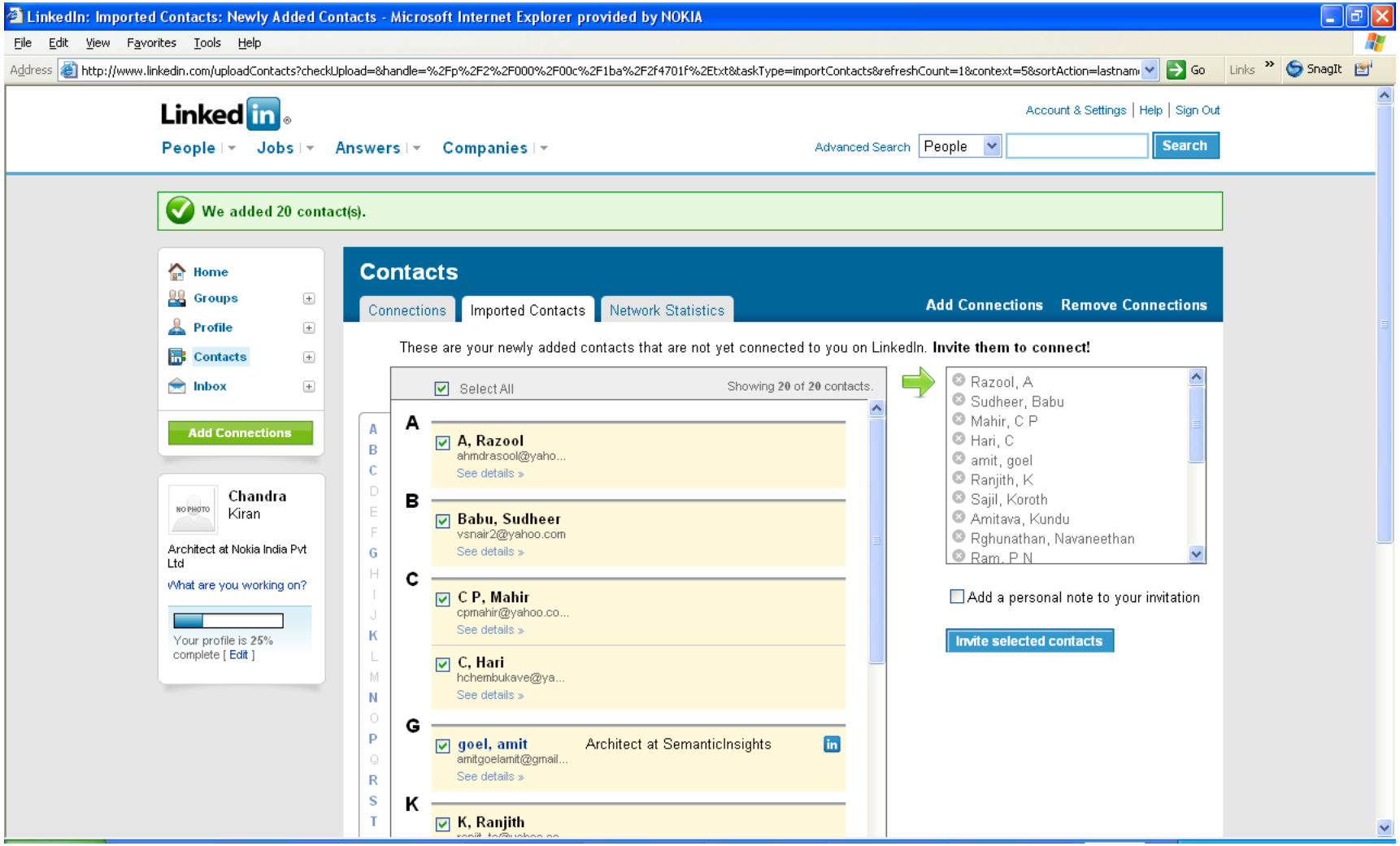
User authenticated by Authorization Server

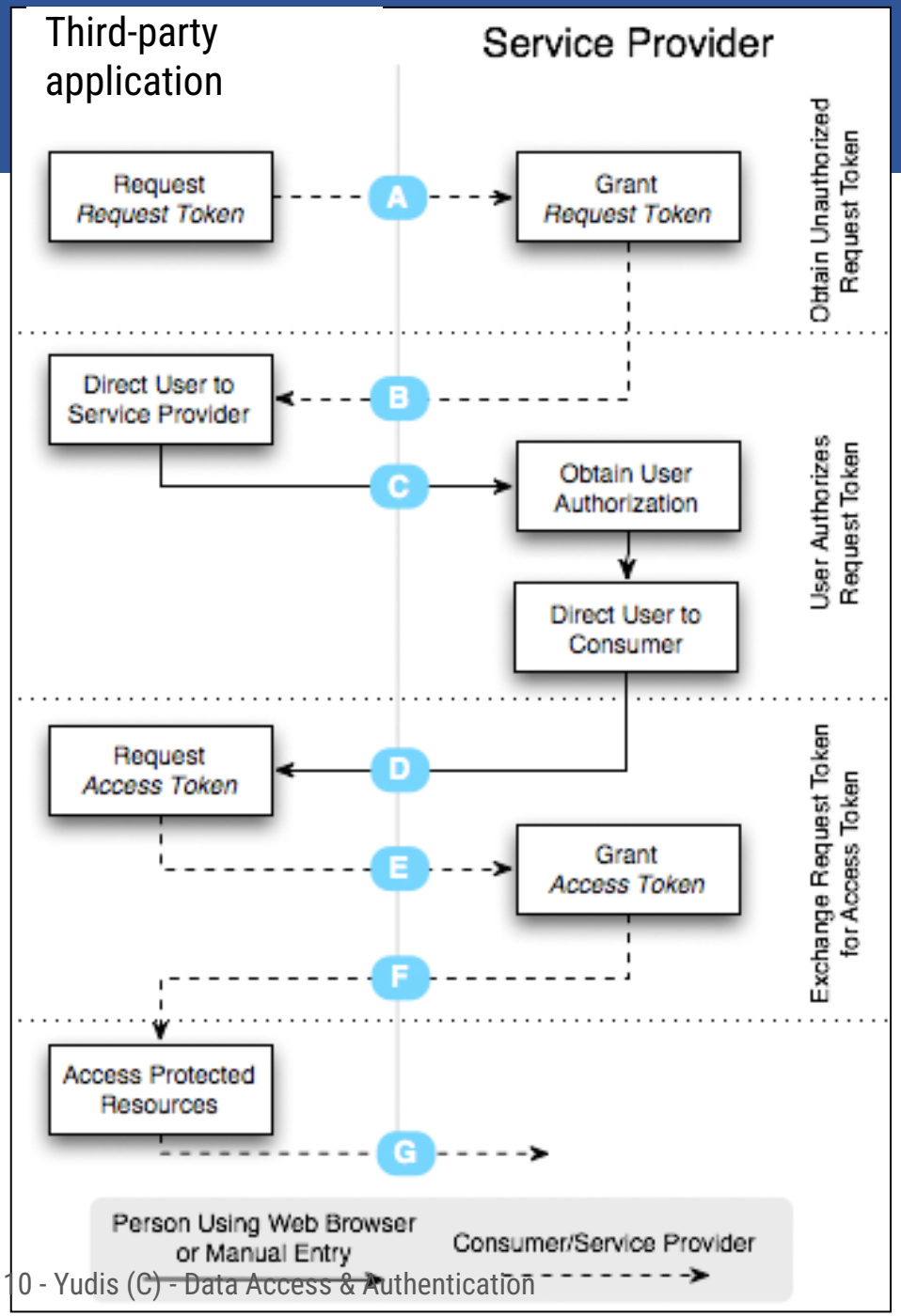


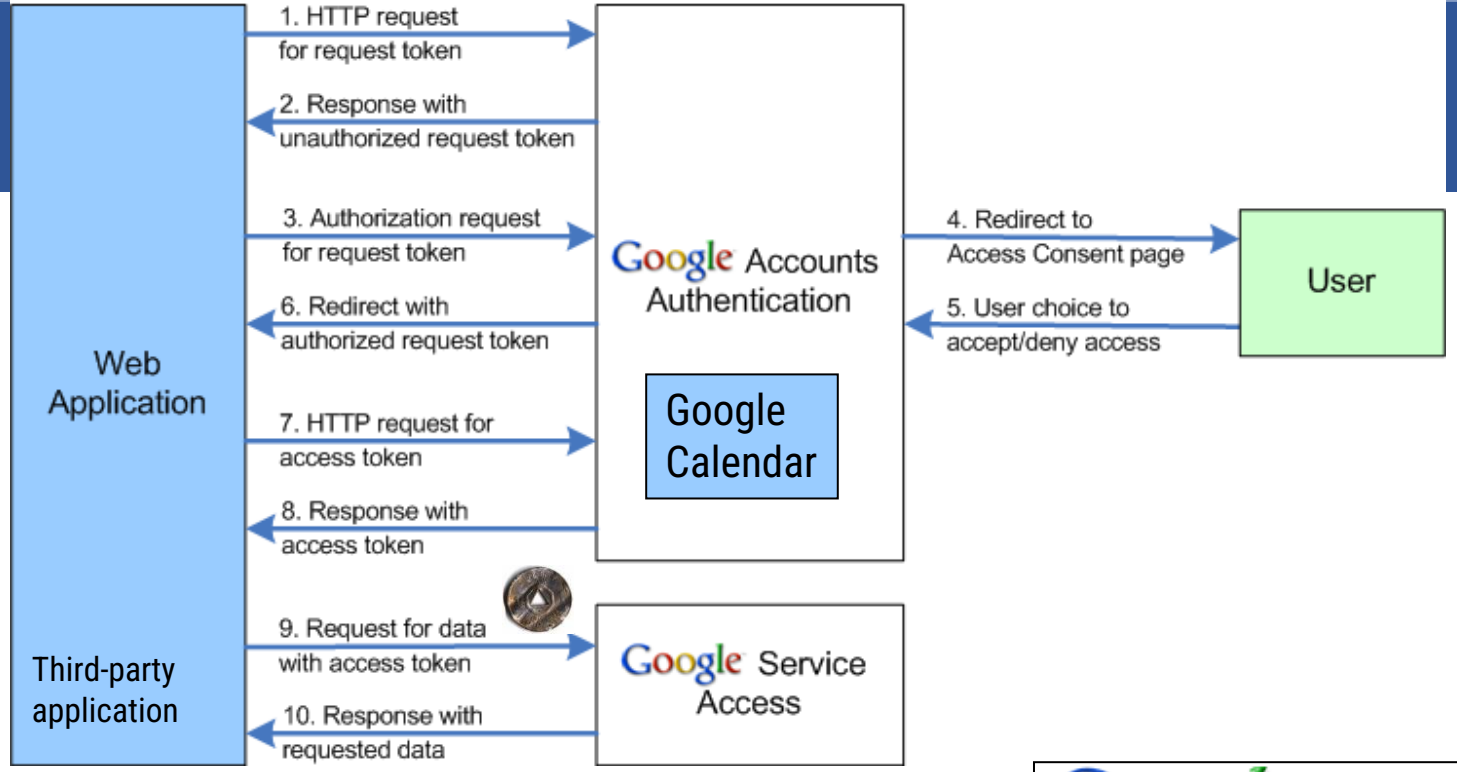
User authorizes Resource Consumer to access Resource Server



Resource Client calls the Resource Server API

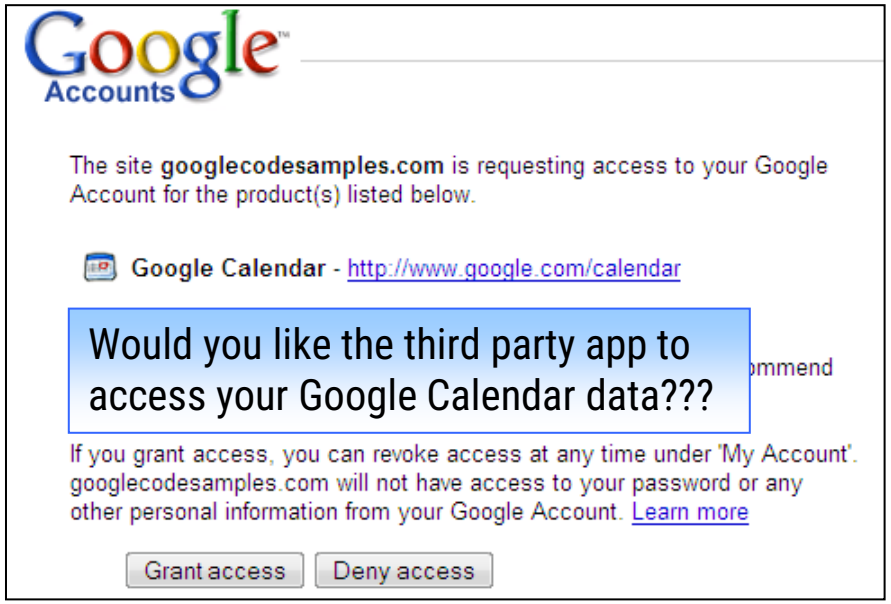



Third-party application
Service Provider
Request Request Token
A
Grant Request Token
Obtain Unauthorized Request Token
Direct User to Service Provider
B
C
Obtain User Authorization
Direct User to Consumer
User Authorizes Request Token
Request Access Token
D
E
Grant Access Token
F
Exchange Request Token for Access Token
Access Protected Resources
G
Person Using Web Browser or Manual Entry
Consumer/Service Provider



Your google calendar data is:

```
<pre><?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:ope
<id>http://www.google.com/calendar/feeds/default/ou
<updated>2009-06-29T16:44:00.229Z</updated>
<category scheme="http://schemas.google.com/g/2005#
<title>Guo Zhenhua's Calendar List</title>
<link rel="alternate" type="text/html" href="http:,
<link rel="http://schemas.google.com/g/2005#feed" t
<link rel="http://schemas.google.com/g/2005#post" t
<link rel="self" type="application/atom+xml" href="
<author>
  <name>Guo Zhenhua</name>
  <email>jenvor@gmail.com</email>
</author>
<generator version="1.0" uri="http://www.google.com
```



OAuth - Drawbacks

- Delegation granularity
- Error handling
- Token expiration vs revocation

JWT

- JSON Web Tokens
  - jwt.io
  - Example


```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
```


HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Remarks

- Auto log-off (session destroy) after x minutes idle
  - `session.gc_maxlifetime`
  - `session.cookie_lifetime`
- Beware cookie can be accessible from JS
  - `HttpOnly (session.cookie_httponly)`