

# *E6156 : Topics in Software Engineering Cloud and Microservice Applications*

*Donald F. Ferguson, [dff@cs.columbia.edu](mailto:dff@cs.columbia.edu)*

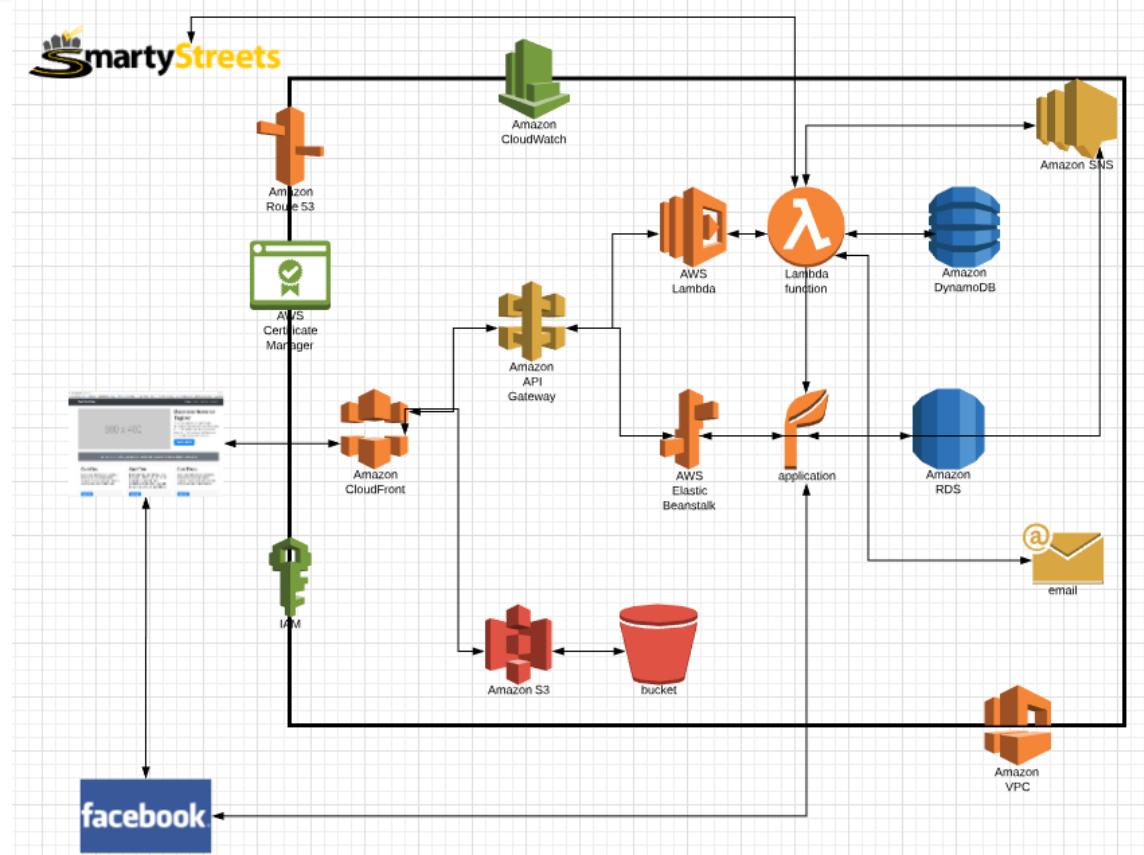
# *Contents*

# Contents

- Status
  - Reminder on overall project structure.
  - Semester roadmap.
- Email Verification Continued.
- OAuth2
  - Overview and concepts.
  - Facebook Login example.
- Composition and Orchestration
  - Composition concept.
  - Activiti example.
  - AWS Workflow, AWS Step Functions

# *Status*

# Overall Project Structure



# From Lecture 1 – Objectives

Build a simple multi-tenant cloud application in a small team

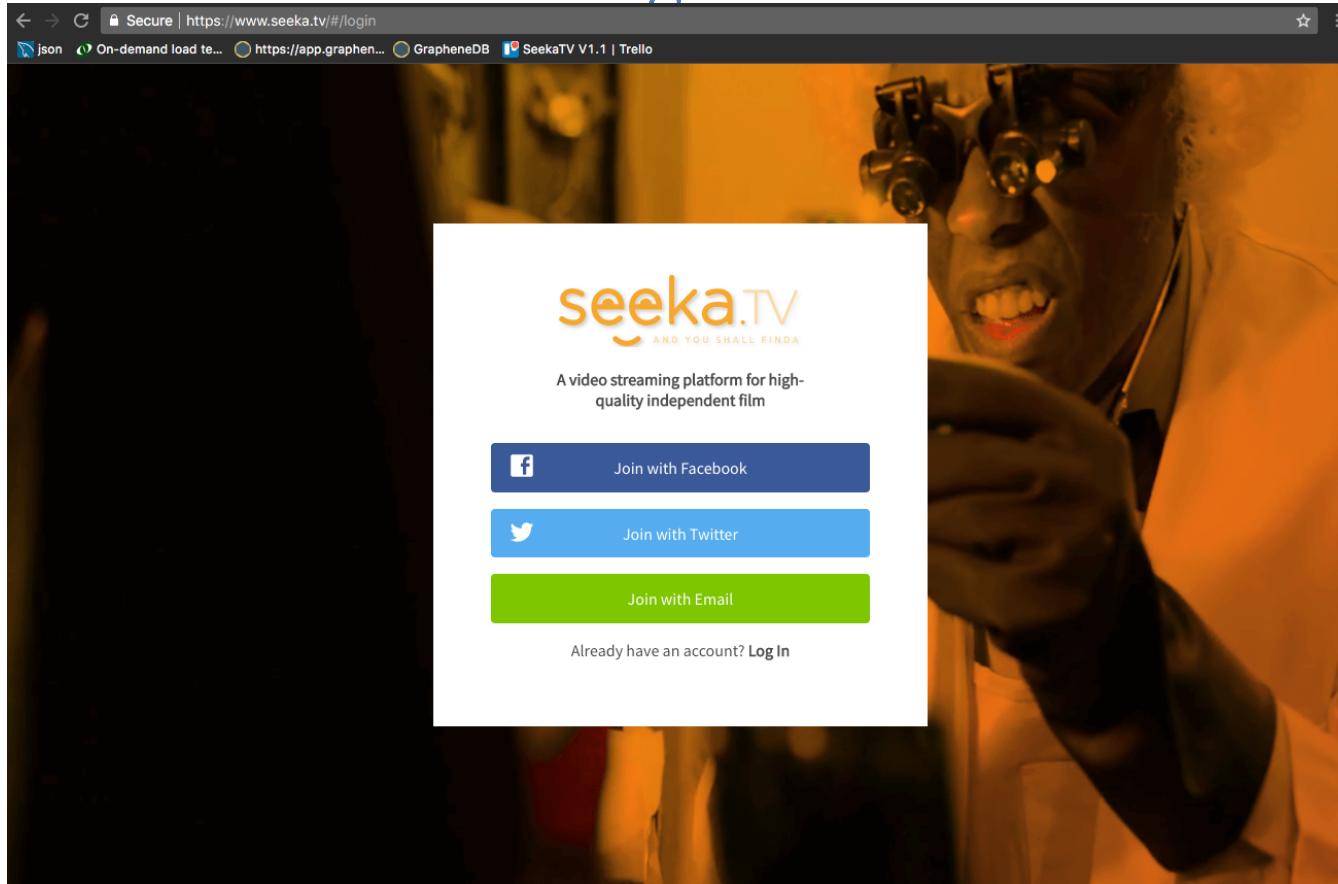
- A set of microservices.
- Using elements of cloud computing:
  - PaaS
  - FaaS
  - Logging
  - Serverless
  - Cloud Database(s)
  - **Security: Authentication, Authorization and Federation.**
  - Message Queues, **Events** and Event-Driven-Architecture
  - **Simple workflow and service orchestration.**
  - **Call cloud business APIs.**
  - Email, Mobile.
  - Other cool stuff.

# Status and Roadmap

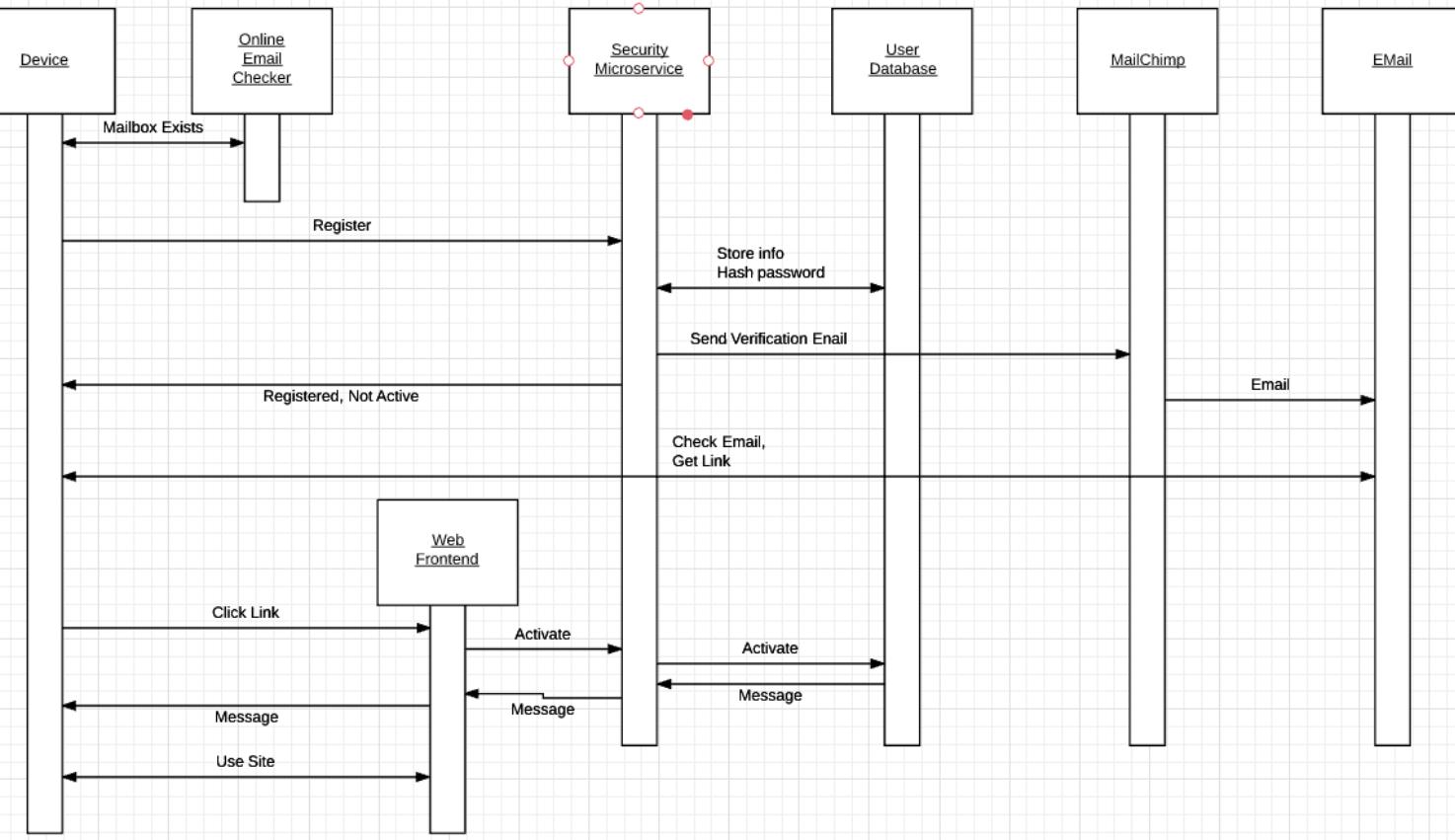
- We covered an overview of many concepts, but are behind on applying to a very basic, functioning application.
- Lecture Roadmap:
  - 09-Nov: Email verification, Facebook, Composition/Orchestration Concepts.
  - 16-Nov: Composition/Orchestration, Messaging – Details, Impl.
  - 23-Nov: Thanksgiving.
  - 30-Nov: Graph Database, other cloud databases.
  - 07-Dec: Misc. topics.
- Final project reviews: 10-Dec to 17-Dec.

# *Email Verification*

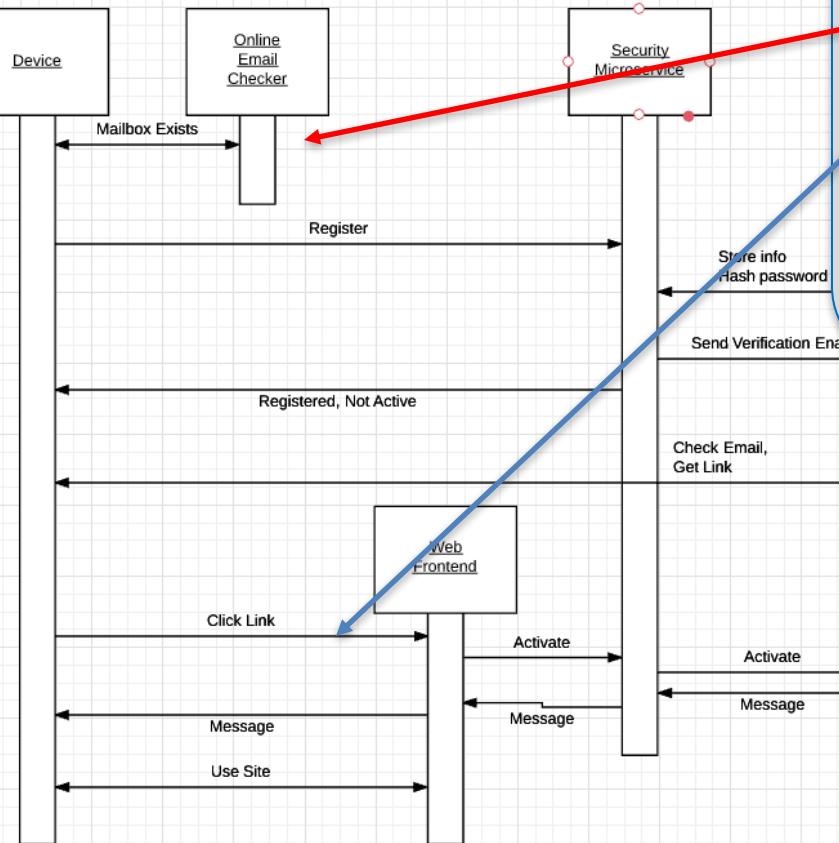
# Registration



# Email Registration – Seeka/Sparq Model



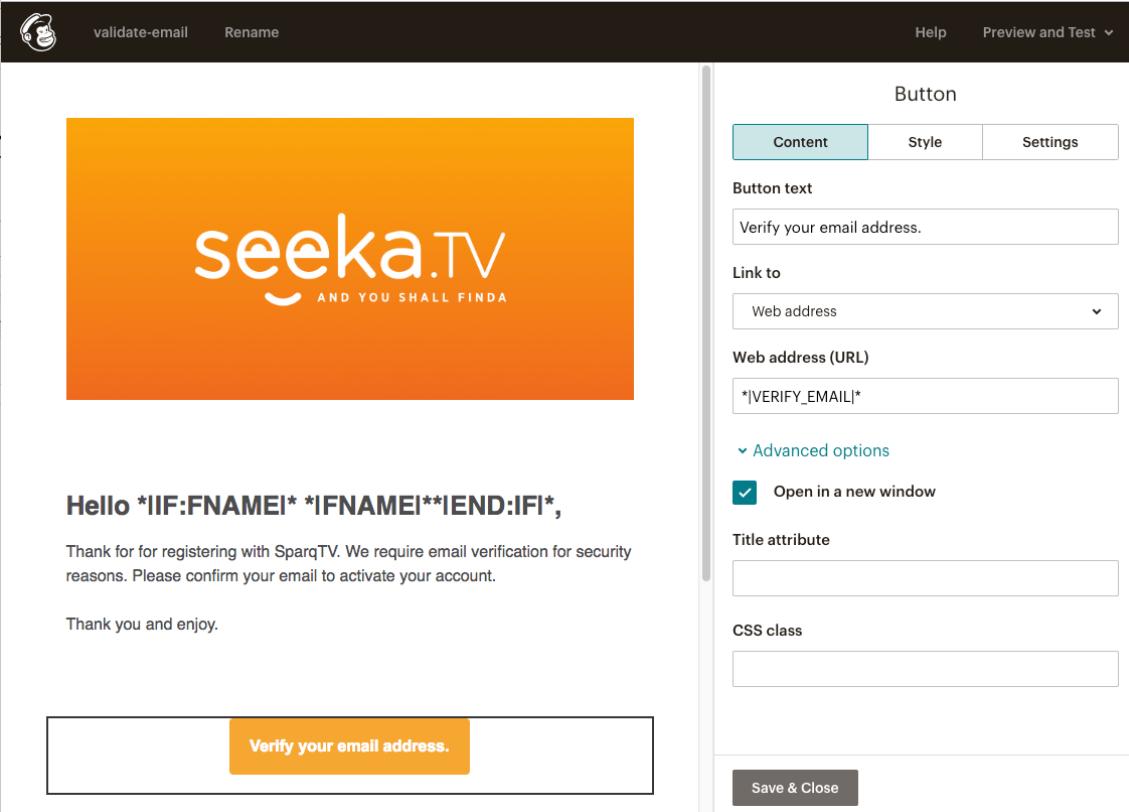
# Email Registration – S



Your implementation can be much simpler

- **Do not need to do initial verification.**
- **Clicking on verify link can go directly to Lambda function (or Beanstalk) to verify.**
- **User redirect to send to email confirmed page.**
- Use AWS SES and just a confirmed email account, not general registration.

# Email Ownership Verification – Seeka



The screenshot shows the Seeka.TV interface. On the left, there is a preview of an email template with the subject "Hello \*IIF:FNAMEI\* \*IFNAMEI\*\*IEND:IFI\*", a message body thanking the user for registering and asking them to verify their email, and a "Verify your email address." button. On the right, there is a configuration panel for a "Button" component. The "Content" tab is selected, showing "Button text" as "Verify your email address.", "Link to" as "Web address", and "Web address (URL)" as "\*|VERIFY\_EMAIL|\*". There is also an "Advanced options" section with a checked "Open in a new window" checkbox. The "Style" and "Settings" tabs are also present.

- The email is a template.
- The template
  - Is HTML
  - Has defined "merge tags" embedded in the document.
- Send email API:
  - Parameters for To:, Cc:, ...
  - Values to merge into email, e.g.
    - Name
    - Custom message
    - Verify URL for button
  - Verify email URL gets bound to the href for the button.

# Send Email Lambda – Seeka Example

```
'use strict';
console.log('Loading function');

var mandrill = require('mandrill-api/mandrill');
var mandrill_client = new mandrill.Mandrill('L5FjEL8ItFib4wIdQ0mlcw');

var theHandler = function(event, context, callback) {
  console.log('Received event:', JSON.stringify(event, null, 2));

  console.log("Calling sendTemplate");
  sendTemplate(event.template_name, event.toEmail, event.subject, event.fname, event.url);
  console.log("After sendTemplate");
  callback(null, "Email Sent!.");
};

exports.handler = theHandler;
```

# Send Email Lambda

```
var template_content = [];
var template_message = {
  "subject": "Welcome, and Verify Email",
  "from_email": "info@seeka.tv",
  "from_name": "SeekaTV",
  "to": [
    ],
  "headers": {
    "Reply-To": "info@seeka.tv"
  },
  "important": false,
  "track_opens": null,
  "track_clicks": null,
  "auto_text": null,
  "auto_html": null,
  "inline_css": null,
  "url_strip_qs": null,
  "preserve_recipients": null,
  "view_content_link": null,
  "bcc_address": "george.reese@seeka.tv",
  "tracking_domain": null,
  "signing_domain": null,
  "return_path_domain": null,
  "merge": true,
  "merge_language": "mailchimp",
  "global_merge_vars": [
    {
      "name": "merge1",
      "content": "merge1 content"
    }
  ],
  "merge_global": true
};
```

# Send Email Lambda

```
}],  
"merge_vars": [{  
    "rcpt": "donff2@aol.com",  
    "vars": [{  
        "name": "FNAME",  
        "content": "Donald"  
    }, {  
        "name": "VERIFY_EMAIL",  
        "content": "http://preview.seeka.tv"  
    }, {  
        "name": "PREFERENCES_LINK",  
        "content": "http://preview.seeka.tv"  
    }, {  
        "name": "EMAIL_ADDRESS",  
        "content": "info@seeka.tv"  
    }]  
},  
"tags": [  
    "email-verification"  
],  
"async": false,  
"ip_pool": "Main Pool",  
"send_at": "2016:09:03 00:00:00"  
;  
;
```

- This is a simple test/example.
- The URL for the button href would be the value of the “VERIFY\_EMAIL” variable.
- The obvious question is, “What is the URL?”

# Send Email Lambda

```
var sendTemplate = function(t_name, toEmail, subject, fname, url) {
  var async = false;
  var ip_pool = "Main Pool";
  var send_at = "2016:09:03 00:00:00";

  console.log("In sendTemplate.");
  console.log("")

  template_message.subject = subject;
  template_message.to = [
    {
      "email": toEmail,
      "type": "to",
      "subject": subject
    }
  ];
  template_message.merge_vars[0].rcpt = toEmail;
  template_message.merge_vars[0].vars[0].content = fname;
  template_message.merge_vars[0].vars[1].content = url;
  var template_name = t_name;

  console.log("Message = " + JSON.stringify(template_message));

  mandrill_client.messages.sendTemplate({
    "template_name": template_name,
    "template_content": template_content,
    "message": template_message,
    "async": async,
    "ip_pool": ip_pool,
    "send_at": send_at
  },
  function(result) {
    console.log("Mandrill result: " + JSON.stringify(result));
  },
  function(error) {
    console.log("Mandrill returned an error " + JSON.stringify(error));
  }
);
  console.log("Exiting.");
};


```

# Verification Email

```
o.js x JS env.js x JS social_media.js x JS security.js x JS logging.js x JS testcrypt.js x JS general_mail_url.js x JS homeController.js x
let jwt = require('jsonwebtoken');

// DO NOT EVER PUT SECRETS IN CODE.
let secret = "secret";

let generate_verify_email_url = function(info) {

    let result = jwt.sign(info, secret);
    result = "http://www.dff-cu.org/api/verify_email?m=" + result
    return result;
};

let info = {
    "nameLast": "Ferguson",
    "nameFirst": "Donald",
    "created": new Date(),
    "email": "someemail@some.org"
};

let verify_url = generate_verify_email_url(info)
console.log("URL = " + verify_url)
```

```
into
www x JS general_mail_url.js x
/usr/local/bin/node /Users/donaldferguson/Dropbox/ColumbiaCourse/Courses/Fall2018/E6156/Projects/CustomerV2/lib/general_mail_url.js
URL = http://www.dff-cu.org/api/verify_email?m=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyYW1TGFrzCI6IkZlcmdlcz9uIiwibmFtZUZpcnN0IjoiRG9uYWxkIiwiY3J1YXRlZCI6IjIwMTgtMTEtMDI
Process finished with exit code 0
```

# *Email Verification*

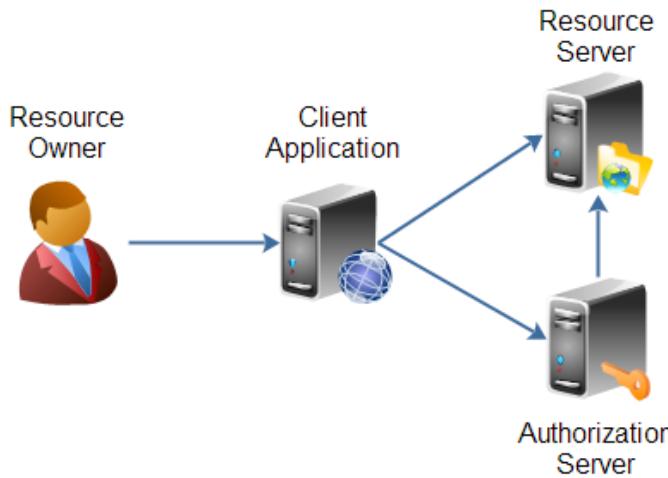
# Overview (<http://tutorials.jenkov.com/oauth2/index.html>)

- Resource Owner
  - Controls *access* to the “data.”
  - Facebook **user**, LinkedIn **user**, ...
- Resource Server
  - The website that holds/manages info.
  - Facebook, LinkedIn, ...
  - And provides access API.
- Client Application
  - “The product you implemented.”
  - Wants to read/update
    - “On your behalf”
    - The data the data that the Resource Server maintains, e.g. posts, status, tweets, ...
- Authorization Server
  - Grants/rejects authorization
  - Based on Resource Owner decisions.
  - Usually (logically) the same as Resource Server.

OAuth 2.0 defines the following roles of users and applications:

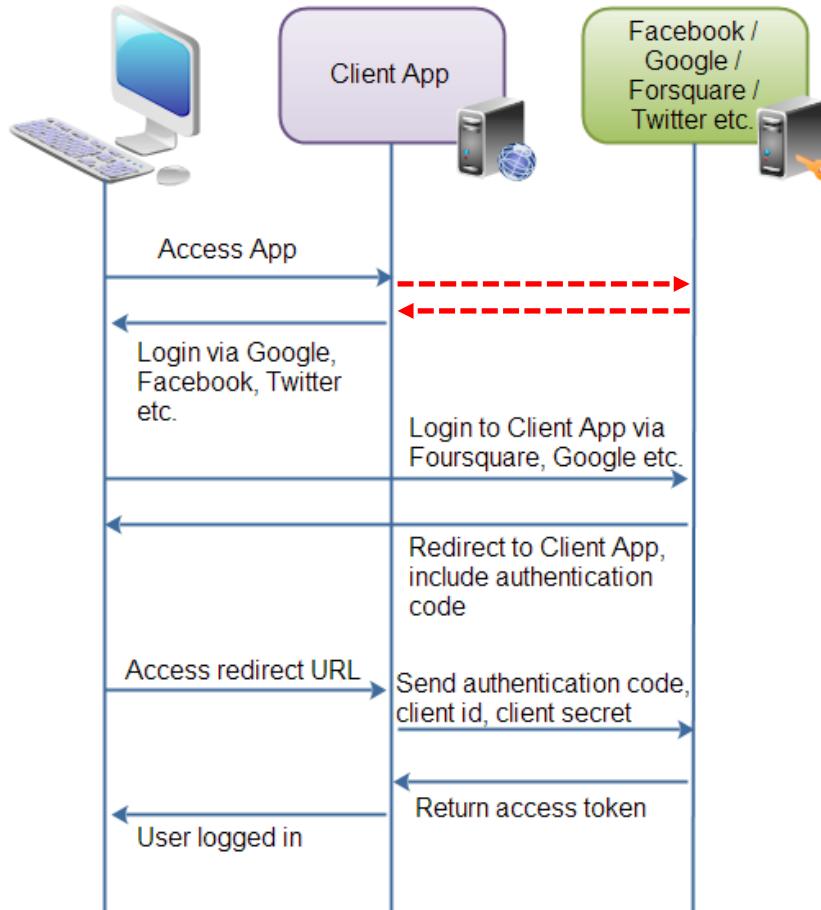
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:

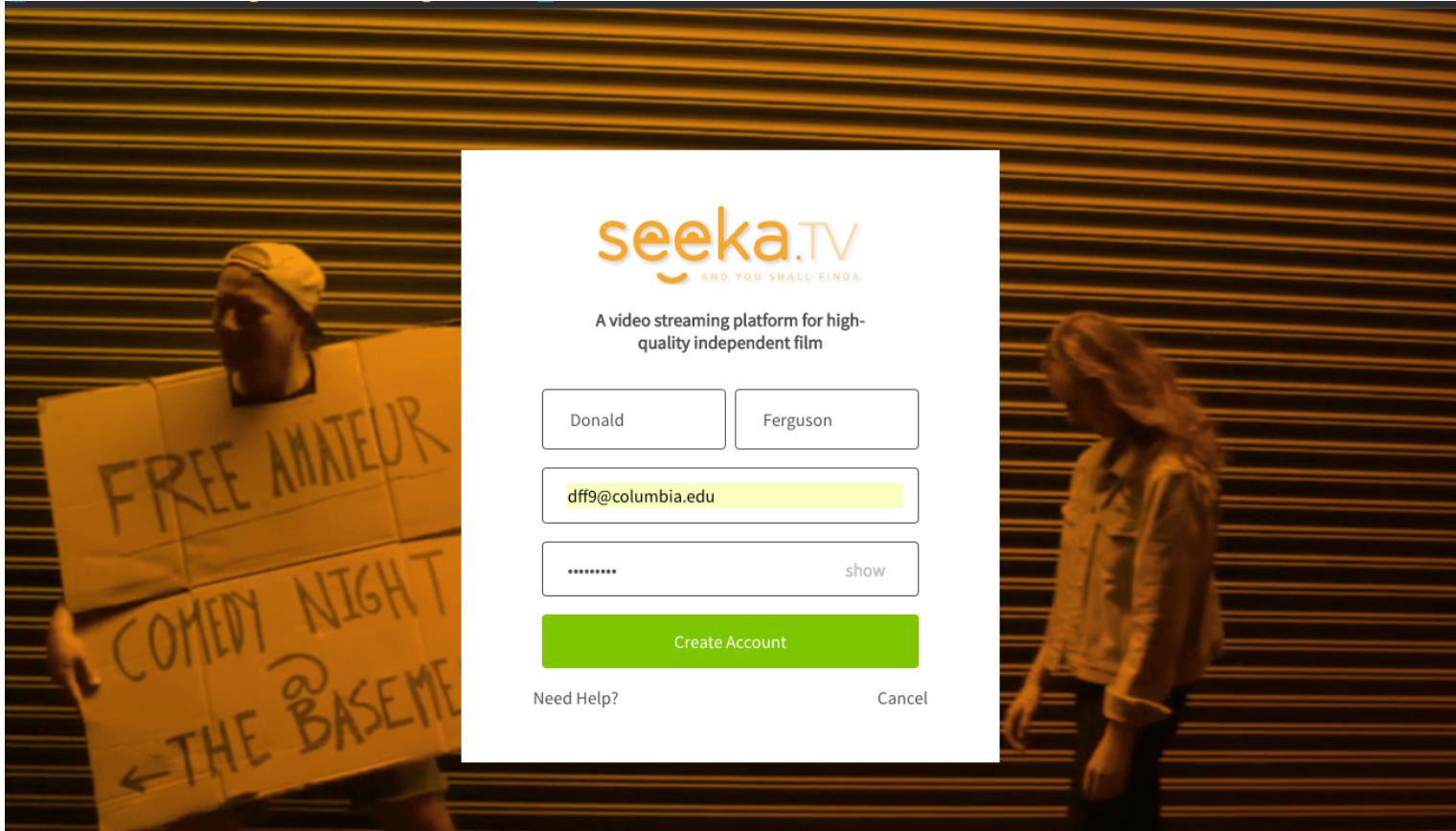


# Ro]

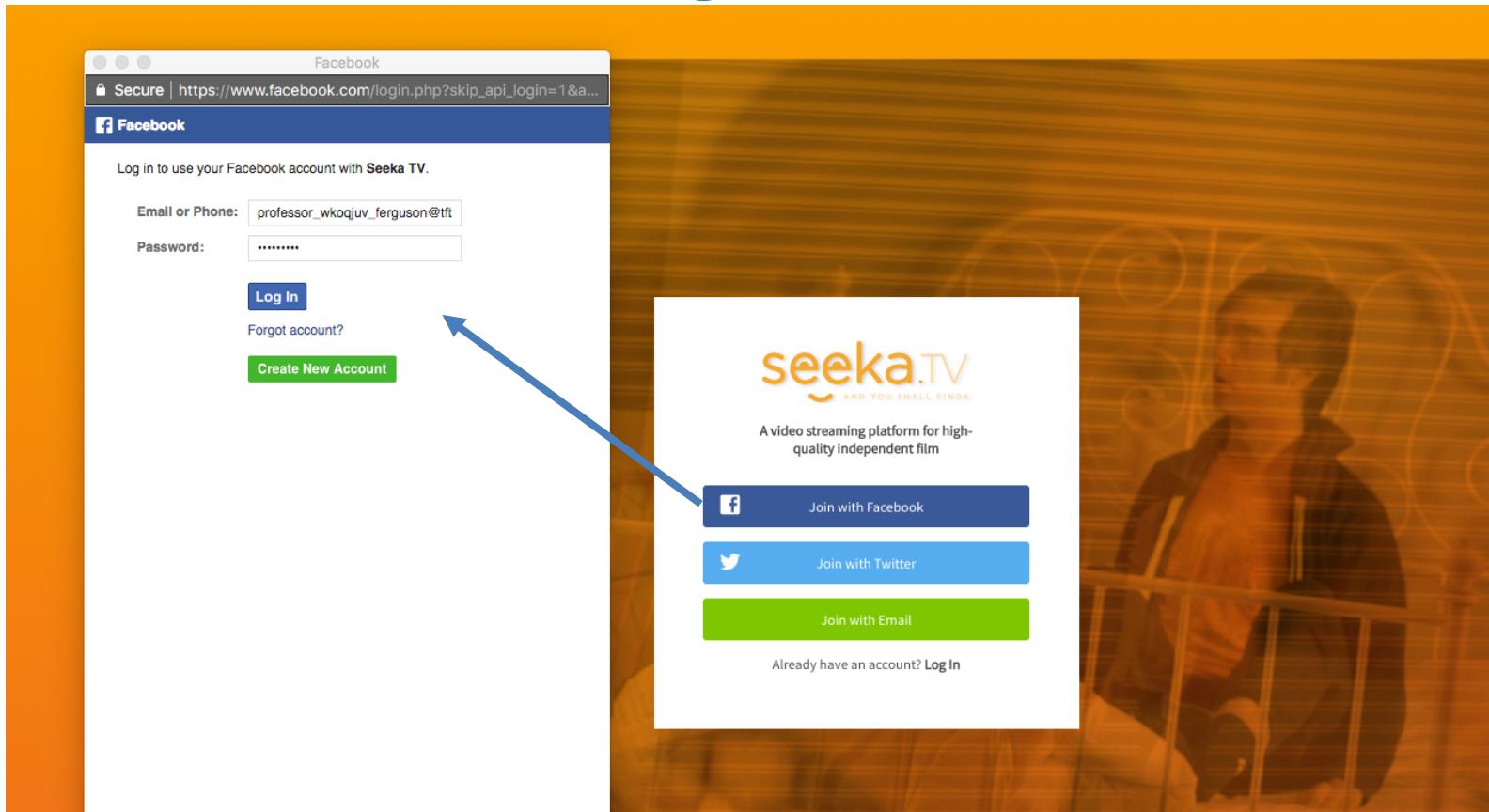
- User Clicks “Logon with XXX”
  - Redirect user to XXX
  - With Client App application ID.
  - And permissions app requests.
  - **Code MAY have to call Resource Server to get a token to include in redirect URL.**
- Browser redirected to XXX
  - Logon on to XXX prompt.
  - Followed by “do you grant permission to ...”
  - Clicking button drives a redirect back to Client App.  
URL contains a temporary token.
- User/Browser
  - Redirected to Client App URL with token.
  - Client App calls XXX API
  - Obtains access token.
  - Returns to User.
- Client App can use access token on API calls.



# Registration



# Registration



# Registration

Log in With Facebook

Secure | https://www.facebook.com/v2.6/dialog/oauth?channel=https%3A%2F%2Fsta...

Log in With Facebook

 ee

**Seeka TV** will receive:

your public profile, friend list, timeline posts, likes and email address. [?](#)

[Edit This](#)

[Continue as Professor](#)

[Cancel](#)

 This doesn't let the app post to Facebook

[App Terms](#) · [Privacy Policy](#)

 AND YOU SHALL FIND A

A video streaming platform for high-quality independent film

 [Join with Facebook](#)

 [Join with Twitter](#)

 [Join with Email](#)

Already have an account? [Log In](#)

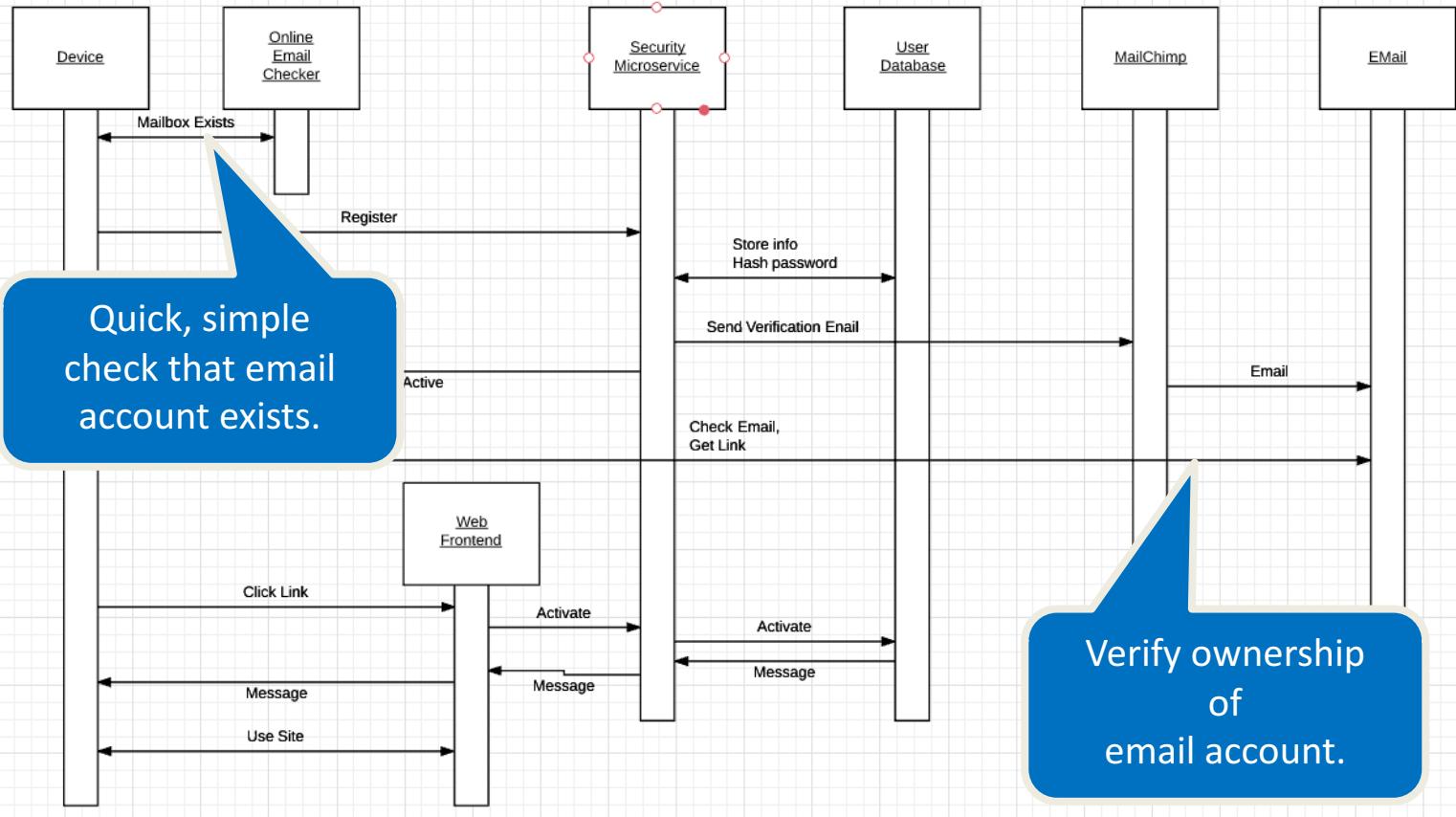
# Pros and Cons of Social Media Logon

(<https://econsultancy.com/blog/61911-the-pros-and-cons-of-a-facebook-login-on-ecommerce-sites>)

- Pros
    - One password for many services: Some people hate making up passwords
    - Familiarity: People trust Facebook and may not want to give you info.
    - Added convenience drives business: “Forced registration is a major cause of checkout abandonment, with [a quarter \(26%\) of respondents in a recent Econsultancy survey](#) stating that being forced to register would cause them to abandon a purchase.”
    - Sharing: Simplifies customers sharing their use of your site → drives awareness.
    - Access to profile data: Do not require users to enter a new profile.
    - Your site does not have to have code for password reset, locked account, etc.
  - Cons
    - Some users don’t want everything to be connected.
    - Some people don’t use Facebook.
    - Muddying your brand image.
-

# Email

# Email Registration



# Social Media Registration (Facebook)

# Define Application

APP ID: 3177 [REDACTED] [View Analytics](#)

Tools & Support Docs 

**Dashboard**

Settings  
Roles  
Alerts  
App Review

**PRODUCTS**

Facebook Login  
+ Add Product

**Dashboard**

**ColumbiaCourse** 

This app is in development mode and can only be used by app admins, developers and testers [?]

API Version [\[?\]](#) App ID  
v2.7 3177 [REDACTED]

App Secret  [Show](#)

 **Get Started with the Facebook SDK** x

Use our quick start guides to set up the Facebook SDK for your iOS or Android app, Facebook Web Game or website.

[Choose Platform](#)

**Facebook Analytics**

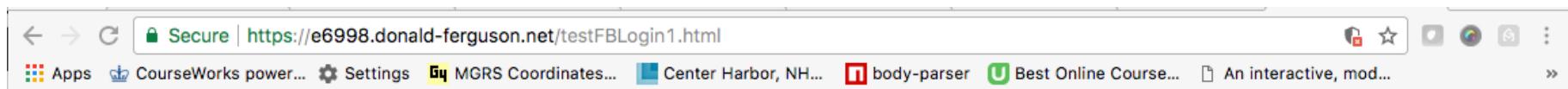
**Set up Analytics**

Analytics helps you grow your business and learn about the actions people take in your app. It only takes 5 minutes to set up.

[Try Demo](#) [View Quickstart Guide](#)

facebook for developers 

# Web Page



Secure | <https://e6998.donald-ferguson.net/testFBLogin1.html>

Apps CourseWorks power... Settings MGRS Coordinates... Center Harbor, NH... body-parser Best Online Course... An interactive, mod...

## COMSE6998-014: Microservice and Cloud Applications

This is without a doubt the totally coolest course on the Columbia syllabus, taught by an adjunct professor who is too cool for school. In this lecture you learn:

1. How to login to an application with Facebook.
2. General concepts around OAuth2
3. How this all fits into a larger security model.



Log In with Facebook

Full Name: {{name}}

email: {{email}}

security token: {{secret}}

```
<div ng-app="apiController" ng-controller="myCtrl">
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <script src="/apiController.js"></script>
  <script>

    // This function is called when someone finishes with the Login
    // Button. See the onlogin handler attached to it in the sample
    // code below.
    function checkLoginState() {
      FB.getLoginStatus(function(response) {
        statusChangeCallback(response);
      });
    }

    window.fbAsyncInit = function() {
      FB.init({
        appId      : '3177████████',
        cookie     : true,  // enable cookies to allow the server to access
                           // the session
        xfbml     : true, // parse social plugins on this page
        version   : 'v2.5' // use graph api version 2.5
      });

      // Now that we've initialized the JavaScript SDK, we call
      // FB.getLoginStatus(). This function gets the state of the
      // person visiting this page and can return one of three states to
      // the callback you provide. They can be:
      //
      // 1. Logged into your app ('connected')
      // 2. Logged into Facebook, but not your app ('not_authorized')
      // 3. Not logged into Facebook and can't tell if they are logged into
      //    your app or not.
      //
      // These three cases are handled in the callback function.

      FB.getLoginStatus(function(response) {
        statusChangeCallback(response);
      });
    };

    // Load the SDK asynchronously
    (function(d, s, id) {
      var js, fjs = d.getElementsByTagName(s)[0];
      if (d.getElementById(id)) return;
      js = d.createElement(s); js.id = id;
      js.src = "//connect.facebook.net/en_US/sdk.js";
      fjs.parentNode.insertBefore(js, fjs);
    }(document, 'script', 'facebook-jssdk'));
  </script>
```

# Code (II)

```
<h1 style="text-align: center;">COMSE6998-014: Microservice and Cloud Applications</h1>
<p>
  This is without a doubt the totally coolest course on the Columbia syllabus, taught by an adjunct professor who is too cool for school. In this lecture you
  learn:
  <ol>
    <li>How to login to an application with Facebook.
    <li>General concepts around OAuth2
    <li>How this all fits into a larger security model.
  </ol>
</p>

<!--
<fb:login-button scope="public_profile,email" onlogin="checkLoginState();">
</fb:login-button>
-->
<div ng-click="login()">
  <br>
  <span>Log In with Facebook</span>
</div>

<div>
  <p>
    Full Name: {{name}}<br>
    email: {{email}}<br>
    security token: {{secret}}<br>
  </p>
</div>
</div>
</body>
</html>
```

# Facebook Login

← → C i localhost:3000/testFBLogin1.html

CourseWorks power... Settings MGRS Coordinates... Center Harbor, NH... body-parser Best Online Courses... An interactive, mod... localhost:3000/spar... Benefits- D

## COMSE6998-014: Microservice and Cloud Applications

This is without a doubt the totally coolest course on the Columbia syllabus, taught by an adjunct professor who is too cool for school. In this lecture you learn:

1. How to login to an application with Facebook.
2. General concepts around OAuth2
3. How this all fits into a larger security model.

 Continue with Facebook

Log In with Facebook

Full Name: Professor Ferguson

email: professor\_rncgpb\_ferguson@tfbnw.net

security token:

EAAEgZBNkI0dcBAC05drNhSJKfC6b1owCaqUmoW3MsC2NFfHnTE4DioHT1jR28kPTvZCh63W

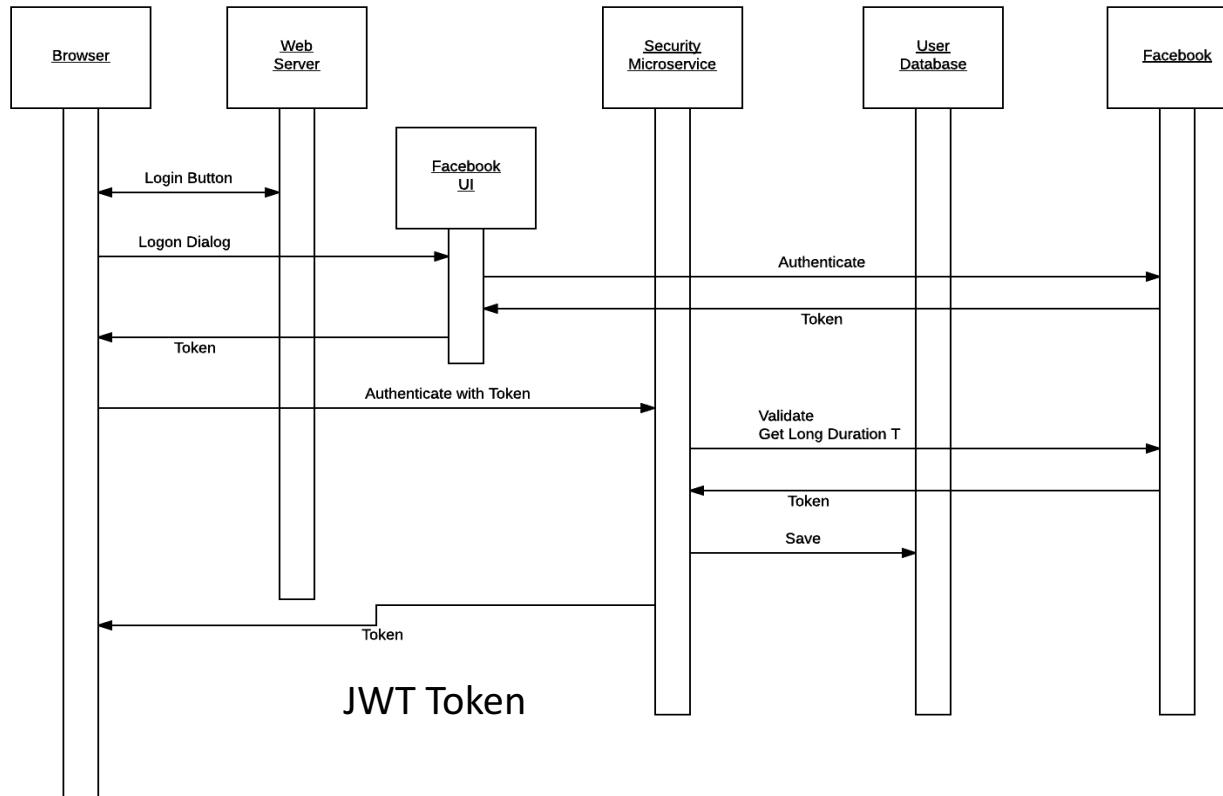
Elements Console Sources Network Performance Memory Application

top Filter Default levels ▾

```
About to call FB.login()
Leaving $scope.login()
In $scope.login(), $scope.status = undefined
About to call FB.login()
Leaving $scope.login()
In FB.login response, response = {"authResponse": {"accessToken": "EAAEgZBNkI0dcBAC05drNhSJKfC6b1owCaqUmoW3MsC2NFfHnTE4DioHT1jR28kPTvZCh63W", "id": "100000000000000", "name": "Professor Ferguson", "first_name": "Professor", "last_name": "Ferguson", "age_range": {"max": 20, "min": 18}, "email": "professor_rncgpb_ferguson@tfbnw.net", "gender": "male", "link": "https://www.facebook.com/app_scoped_user_id/100000000000000/"}}

Welcome! Fetching your information...
Leaving FB.login() response
In FB.api response, response = {
```

# Facebook Login



# Validate Token

```
public class FBConnector {  
  
    private static final Version FACEBOOK_APP_VERSION = Version.VERSION_2_6;  
  
    private static final org.slf4j.Logger logger = LoggerFactory.getLogger(FBConnector.class);  
  
    public static Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();  
  
    public static boolean validateUIDAndToken(String accessToken, String uid) {  
        boolean authenticated = false;  
        User theUser = null;  
  
        try {  
            theUser = FBConnector.fetchUserObjectAsJsonObject(accessToken, uid);  
  
            if (theUser != null) {  
                authenticated = true;  
            }  
        } catch (Exception e) {  
            System.out.println("Error = " + e.toString());  
            e.printStackTrace();  
        }  
  
        return authenticated;  
    }  
}
```

# Validate Token

```
public static User fetchUserObjectAsJsonObject(String accessToken, String uid) {
    User theUser = null;

    try {
        logger.debug("In FB Connector.");

        DefaultFacebookClient facebookClient23 = new DefaultFacebookClient(accessToken, FACEBOOK_APP_VERSION);

        logger.debug("Got Facebook client");
        logger.debug("Calling FB for ID = " + uid);

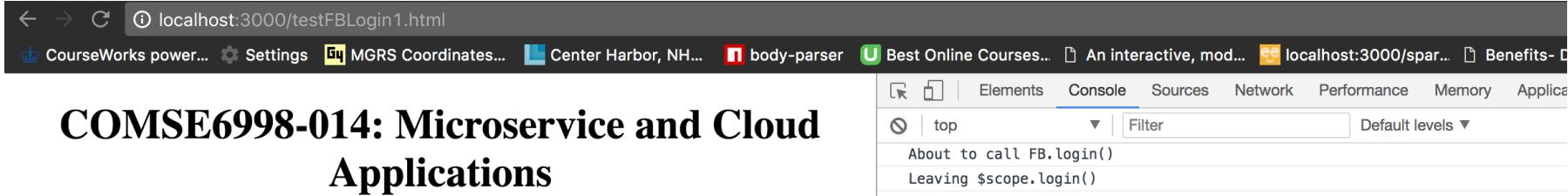
        // Make the API call
        theUser = facebookClient23.fetchObject(uid, User.class);

        // logger.debug("Back from FB call, result = " +
        // prettyGson.toJson(theUser));
        logger.debug("Last name = " + theUser.getLastName());

        // Based on the FB App ID and access token, we sometimes get the
        // last_name and first_name,
        // and sometimes just the full name, e.g. "Hermione Grainger."
        if ((theUser.getLastName() == null) || (theUser.getFirstName() == null)) {
            theUser = updateNames(theUser);
        }
    } catch (Exception e) {
        System.out.println("Error = " + e.toString());
        e.printStackTrace();
        theUser = null;
    }

    return theUser;
}
```

# Facebook Token



## COMSE6998-014: Microservice and Cloud Applications

This is without a doubt the totally coolest course on the Columbia syllabus, taught by an adjunct professor who is too cool for school. In this lecture you learn:

1. How to login to an application with Facebook.
2. General concepts around OAuth2
3. How this all fits into a larger security model.

 Continue with Facebook

Log In with Facebook

Full Name: Professor Ferguson  
email: professor\_rncgpb\_ferguson@tfbnw.net  
security token:  
EAAEgZBNkI0dcBAC05drNhSJKfC6b1owCaqUmoW3MsC2NFfHnTE4DioHT1jR28kPTvZCh63W

```
Elements Console Sources Network Performance Memory Application  
top Filter Default levels ▾  
About to call FB.login()  
Leaving $scope.login()  
In $scope.login(), $scope.status = undefined  
About to call FB.login()  
Leaving $scope.login()  
In FB.login response, response = {"authResponse":  
{"accessToken":"EAAEgZBNkI0dcBAC05drNhSJKfC6b1owCaqUmoW3MsC2NFfHnTE4DioHT1jR28kPTvZCh63W",  
"user": {"id": "100949004006465", "name": "Professor Ferguson", "first_name": "Professor", "last_name": "Ferguson", "age": 20, "email": "professor_rncgpb_ferguson@tfbnw.net", "gender": "male", "link": "https://www.facebook.com/app_scoped_user_id/100949004006465/"}  
,"status": "connected"}  
Welcome! Fetching your information....  
Leaving FB.login() response  
In FB.api response, response = {  
  "id": "100949004006465",  
  "name": "Professor Ferguson",  
  "first_name": "Professor",  
  "last_name": "Ferguson",  
  "age": 20,  
  "email": "professor_rncgpb_ferguson@tfbnw.net",  
  "gender": "male",  
  "link": "https://www.facebook.com/app_scoped_user_id/100949004006465/"}  
}
```

# Graph API Explorer Demo

facebook for developers

Products Docs Tools & Support News Videos

Search

Create App

## Graph API Explorer

Application: [?] Graph API Explorer ▾

Access Token:  [Get Token ▾](#)

 [GET](#) ▾ → /v2.10 ▾ /me?fields=birthday,devices,name  [Submit](#)

[Learn more about the Graph API syntax](#)

Node: me

+ Search for a field

### 1 Debug Message (Show)

```
{  
  "name": "Professor Ferguson",  
  "id": _____  
}
```

# OAuth

## Twitter and Others

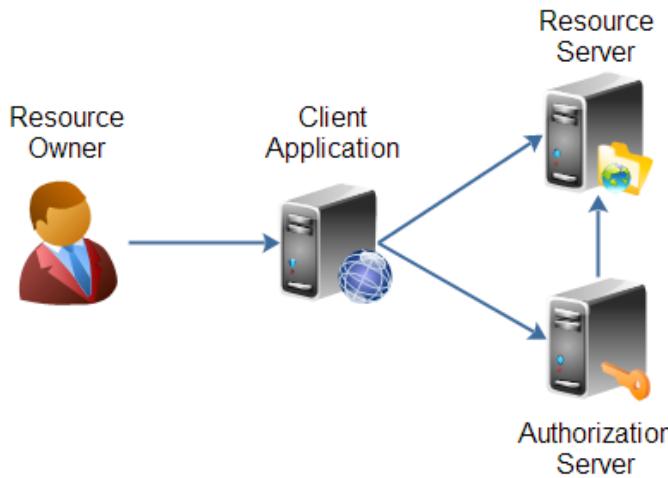
# Overview (<http://tutorials.jenkov.com/oauth2/index.html>)

- Resource Owner
  - Controls *access* to the “data.”
  - Facebook **user**, LinkedIn **user**, ...
- Resource Server
  - The website that holds/manages info.
  - Facebook, LinkedIn, ...
  - And provides access API.
- Client Application
  - “The product you implemented.”
  - Wants to read/update
    - “On your behalf”
    - The data the data that the Resource Server maintains, e.g. posts, status, tweets, ...
- Authorization Server
  - Grants/rejects authorization
  - Based on Resource Owner decisions.
  - Usually (logically) the same as Resource Server.

OAuth 2.0 defines the following roles of users and applications:

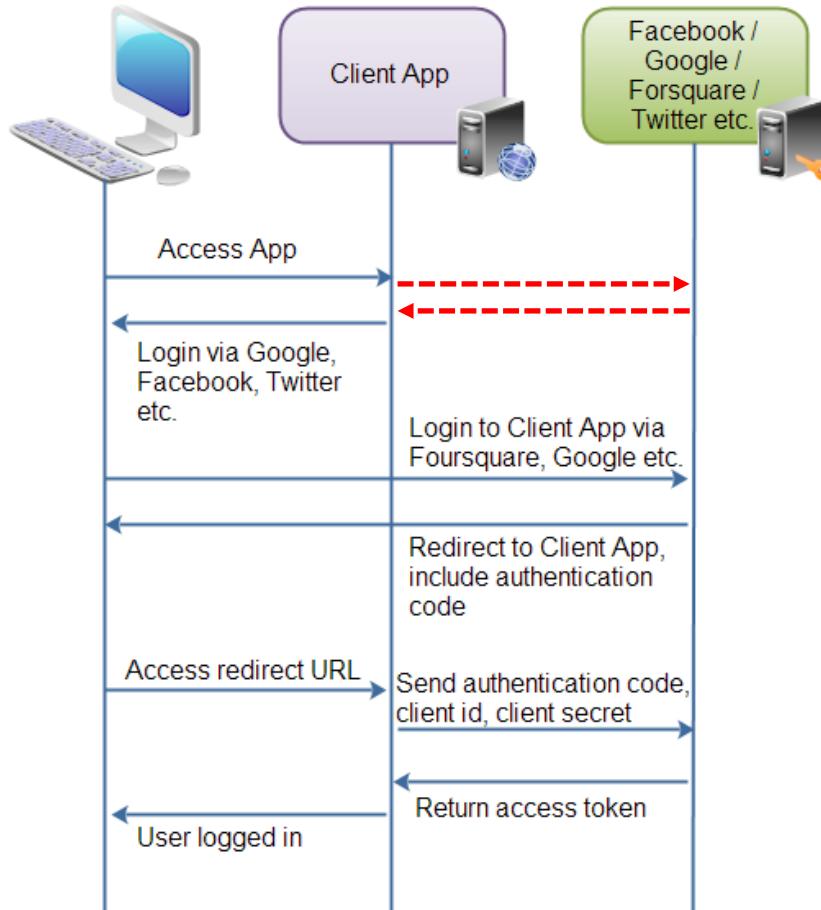
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:



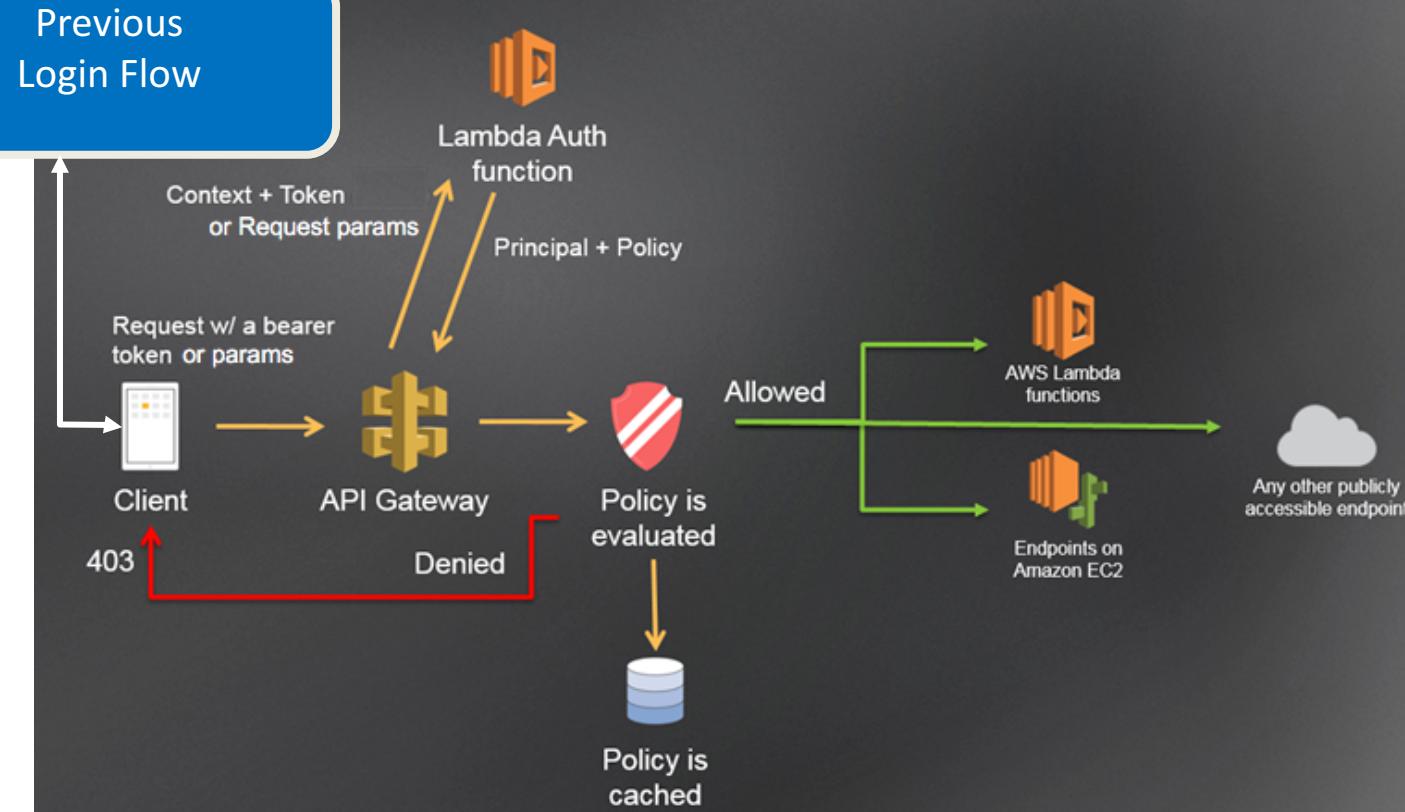
# Ro]

- User Clicks “Logon with XXX”
  - Redirect user to XXX
  - With Client App application ID.
  - And permissions app requests.
  - **Code MAY have to call Resource Server to get a token to include in redirect URL.**
- Browser redirected to XXX
  - Logon on to XXX prompt.
  - Followed by “do you grant permission to ...”
  - Clicking button drives a redirect back to Client App.  
URL contains a temporary token.
- User/Browser
  - Redirected to Client App URL with token.
  - Client App calls XXX API
  - Obtains access token.
  - Returns to User.
- Client App can use access token on API calls.



# Authorization

# Custom Authorizer



# Authorization

- 1<sup>st</sup> check occurs at perimeter in API GW Authorization
  - Based on (bearer token) JWT token
  - Is this role authorized to invoke this URL?
  - Implementation
    - Lambda function that ...
    - Checks an authorization table (role, URL)
    - Or the rights are encoded in the token.
- All down stream microservices
  - Receive the JWT token and perform their own authorization decisions.
  - Run with a set of AWS roles and permissions that determine if AWS allows access, which may include user role.