COMS E6998: **Microservices and Cloud Applications**

Lecture 7: Class Evaluation, OAuth2/FB/Twitter, AWS Security, 12 Factor Apps, Step Functions

Dr. Donald F. Ferguson
dff9@columbia.edu

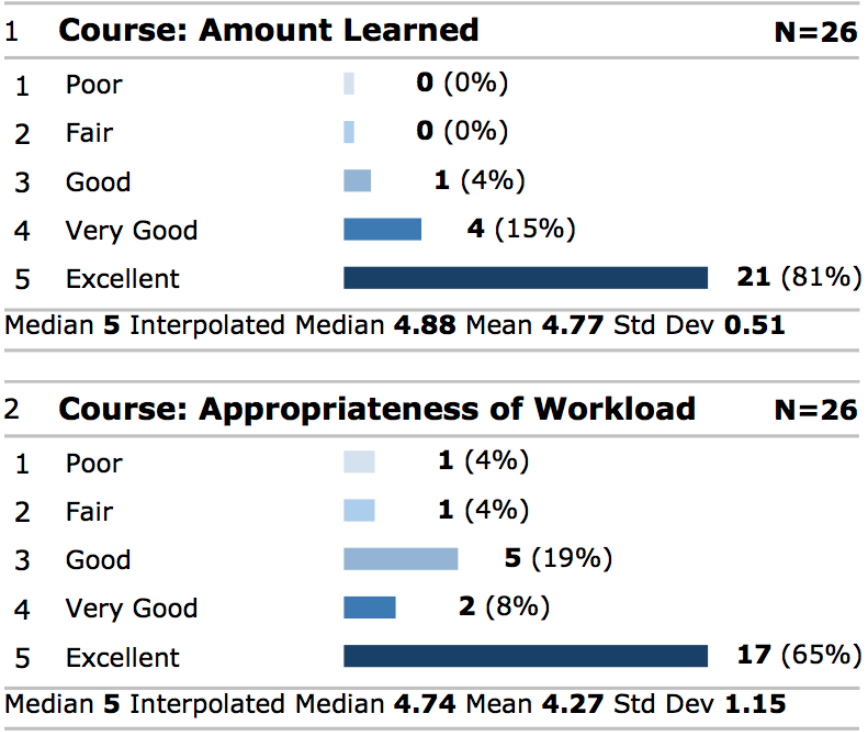
Contents

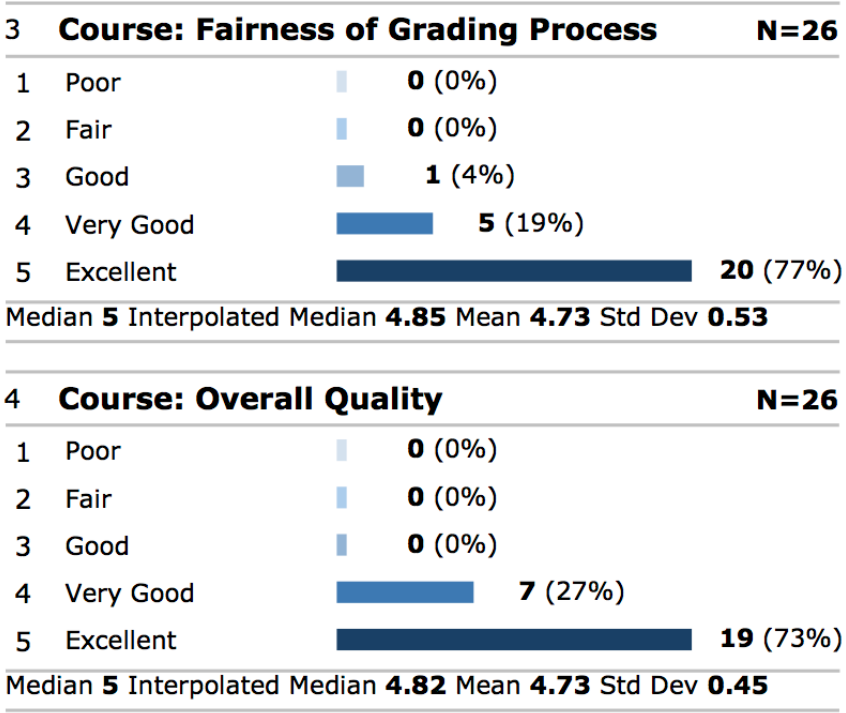
Contents

- Questions, Comments, Discussion?
- Midterm class evaluations
- Project 4: OAuth2 and security
- OAuth2
  - Facebook walkthrough with code.
  - OAuth2 flow.
  - Twitter example and code.
- Infrastructure and application security model (as an example)
  - Resources and operations.
  - Identities, roles, groups.
  - Customer GW Authorizer.
- ~~12 Factor Applications~~
- ~~Workflow/Orchestration and Step Functions~~
  - ~~Overview~~
  - ~~Simple examples~~

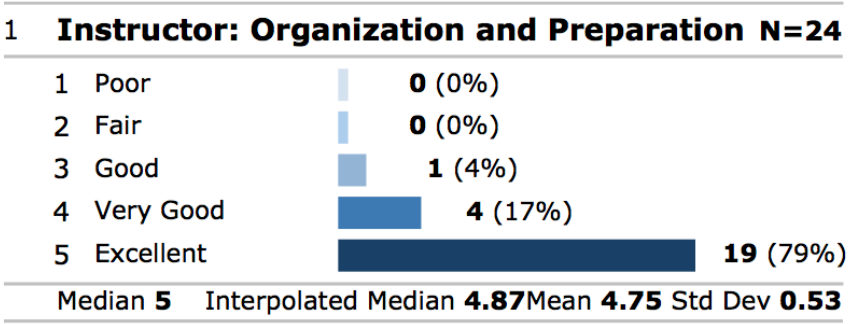
Midterm Class Evaluations

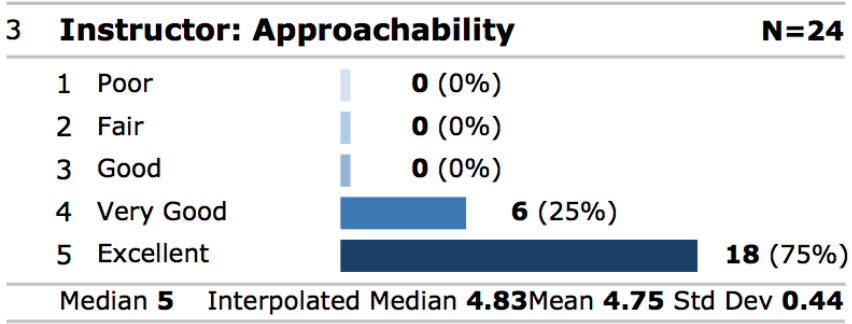
Summary – Course

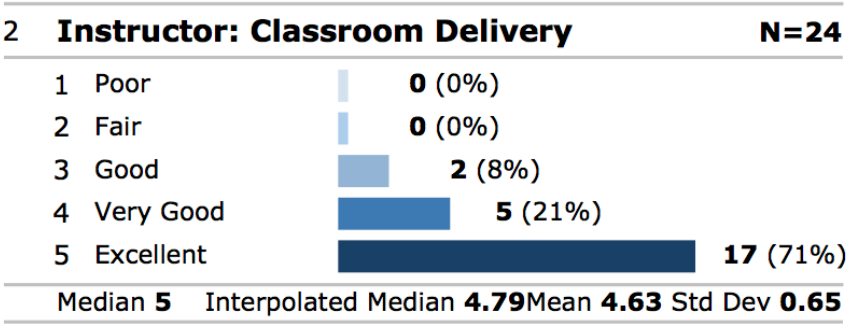


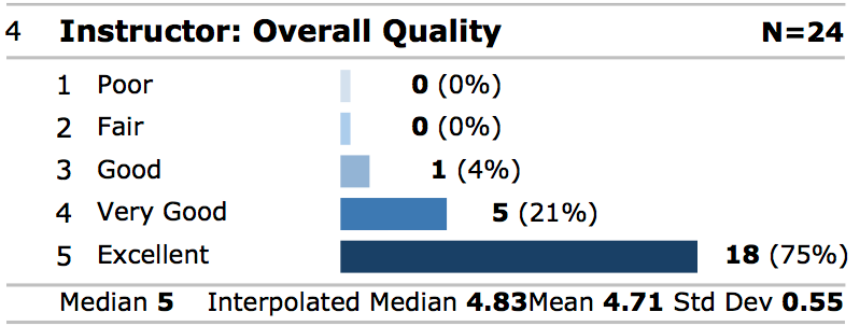


Summary – Instructor









Comments

Note: Did not include two positive comments.

Q1 Enter any additional comments here

- I think that there are too many versions of slides. Please only post the final slide versions so that students are less confused when going through them.
- Homework is more about dealing with difficulties of working with AWS than learning logic. Not very rewarding in my opinion.
- It would be great if you can make lectures slides more detailed and also assign some reference or reading materials.

Actions

- I will try to be more disciplined about the versions of slides.
- I will also try to identify references and reading.
- AWS complexity and nuisance factor is trickier
  - Imagine what it would be like if we were using several clouds.
  - I will try to provide templates to help you get started and more examples.

Project 4

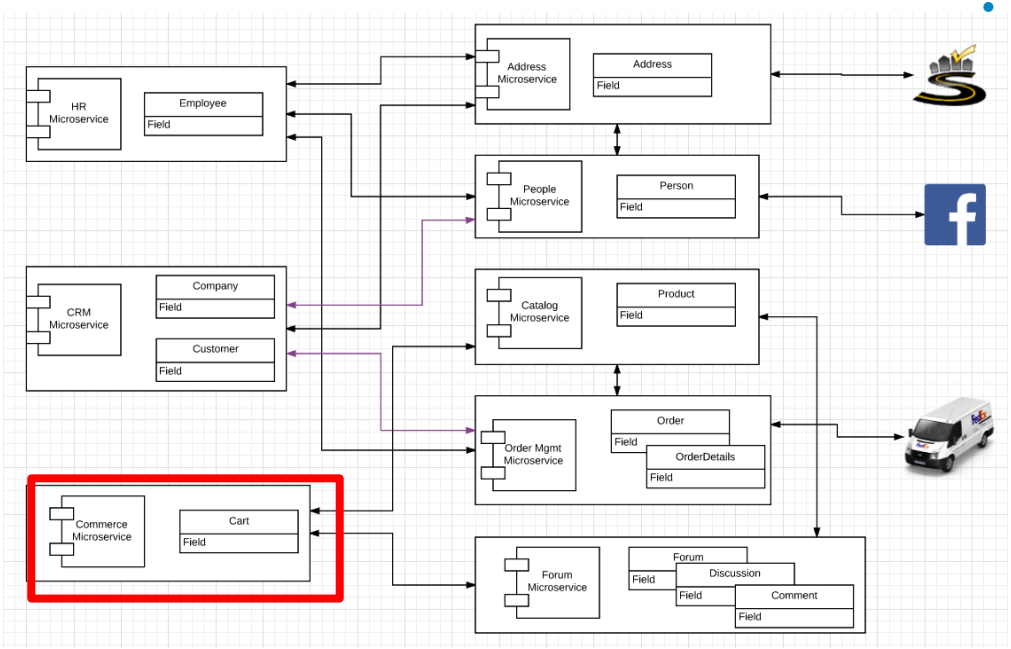
My Mousepad when Preparing Proj. 4 Spec.

Project Idea Inside

Reminder – (Some) Security Facets

- Authentication: " (…) is the act of confirming the truth of an attribute of a single piece of data claimed true by an entity. In contrast with identification, which refers to the act of stating or otherwise indicating a claim purportedly attesting to a person or thing's identity, authentication is the process of actually confirming that identity. "
- Authorization:
  - "The process of authorization is distinct from that of authentication. Whereas authentication is the process of verifying that 'you are who you say you are', authorization is the process of verifying that 'you are permitted to do what you are trying to do'.
  - "**Authorization** is the function of specifying access rights/privileges to resources related to information security and computer security in general and to access control in particular [1]. More formally, "to authorize" is to define an access policy."
- (Communication) Security
  - "**Transport Layer Security** (**TLS**) and its predecessor, **Secure Sockets Layer** (**SSL**), both frequently referred to as "SSL", are cryptographic protocols that provide communications security over a computer network."
    - "The connection is *private* (or *secure*) because symmetric cryptography is used to encrypt the data transmitted."
    - The identity of the communicating parties can be *authenticated* using public-key cryptography.
    - The connection ensures *integrity* because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.
  - **HTTPS** (also called **HTTP over Transport Layer Security** [**TLS**],[1] **HTTP over SSL**,[2] and **HTTP Secure**[3][4]) is a communications protocol for secure communication over a computer network which is widely used on the Internet. HTTPS consists of communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security, or its predecessor, Secure Sockets Layer. The main motivation for HTTPS is authentication of the visited website and protection of the privacy and integrity of the exchanged data.

Project 4 – Start/Expand Commerce Microservice



- Use Lambda function

Customers may

- Register with
- And subsequently logon with
- Facebook or Twitter

Just

- Implement register/logon
- We will later
  - Use Step Functions for composing microservices and APIs
  - Publish commerce actions to FB and Twitter.

Also, write

- A placeholder
- API Gateway Custom Authorizer
- Which we will later use to manager authorization to orders.

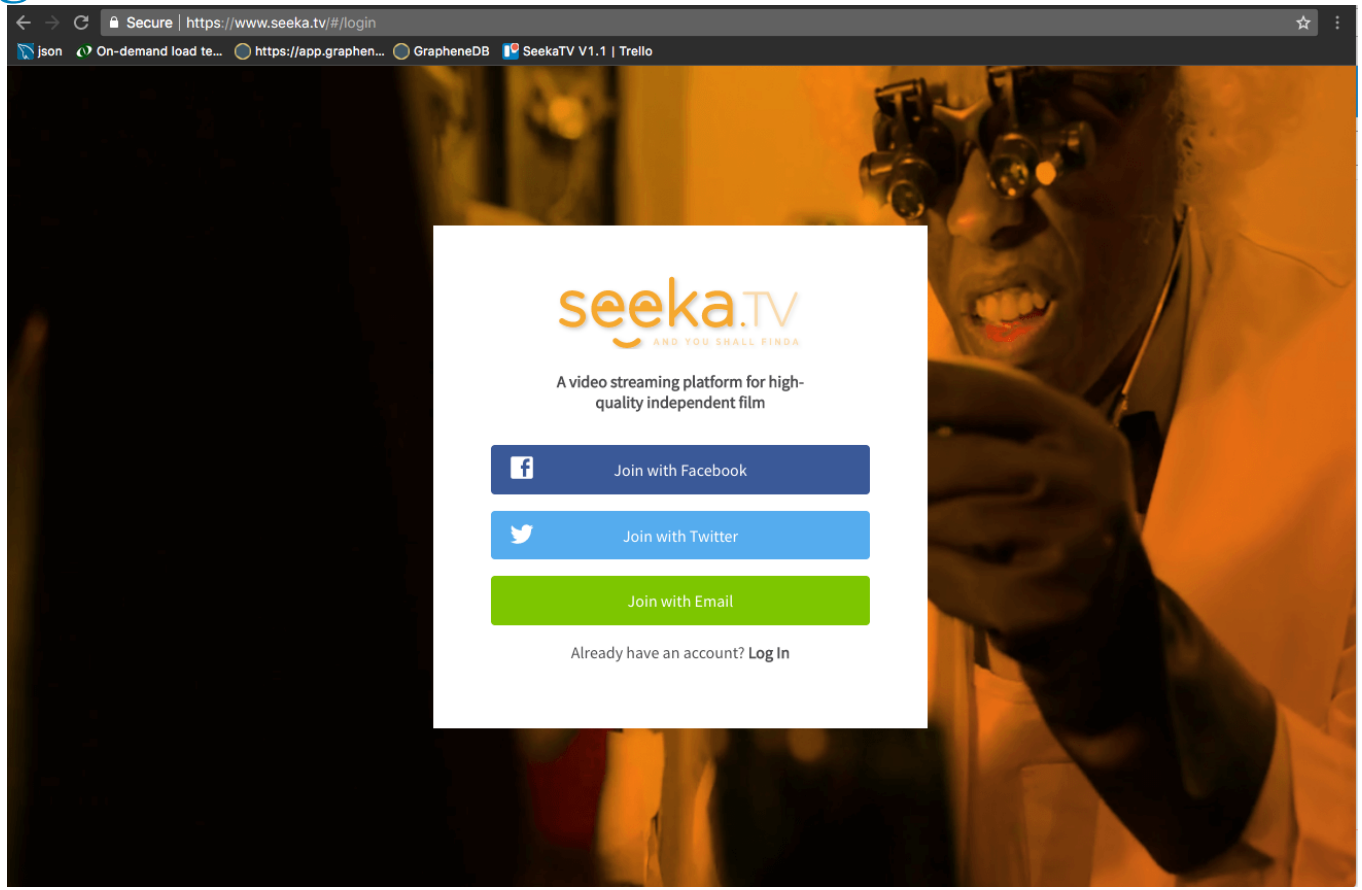
Seeka TV Demo

~~Email,~~ FB, Twitter

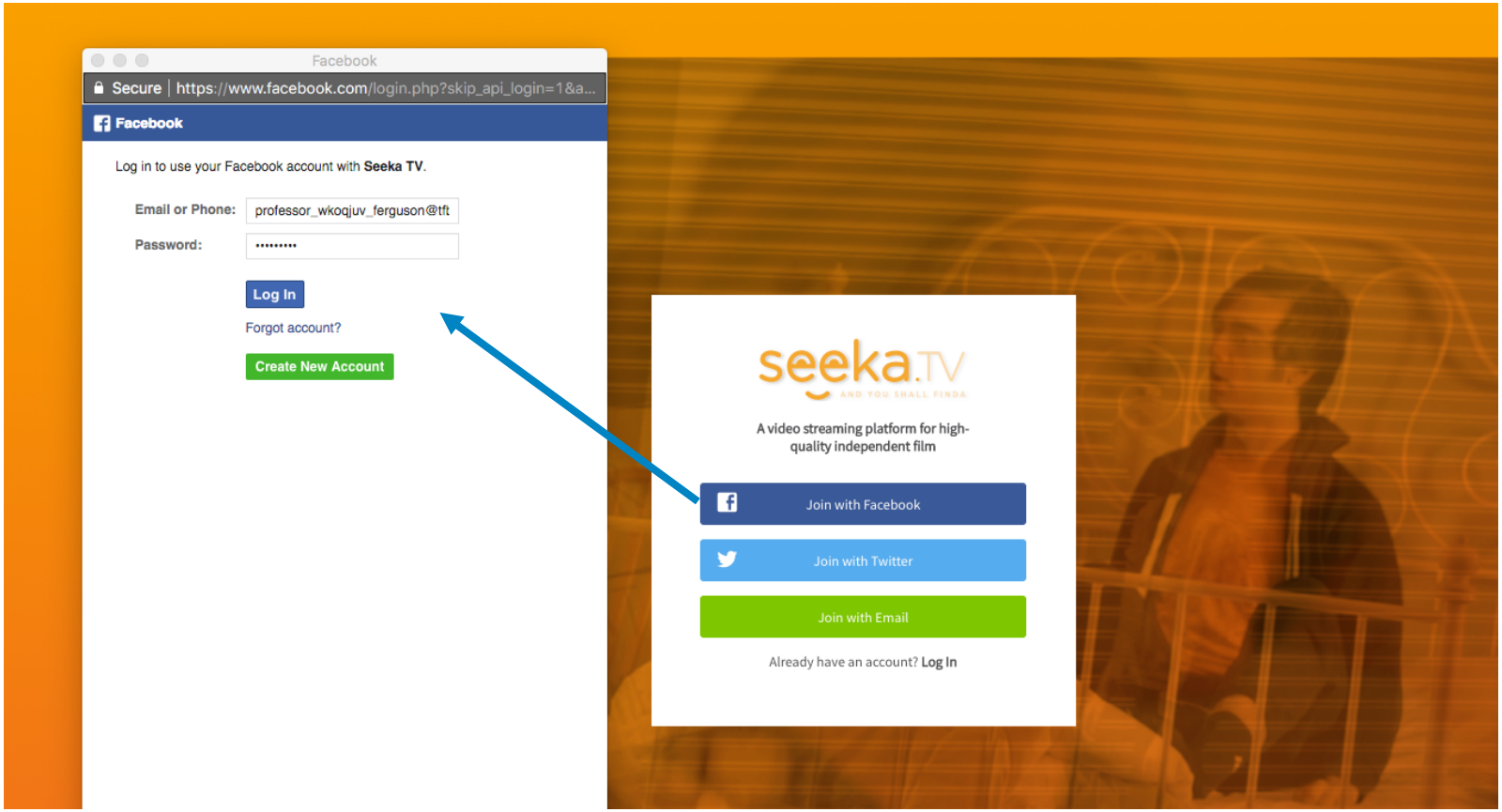
OAuth2

Demo

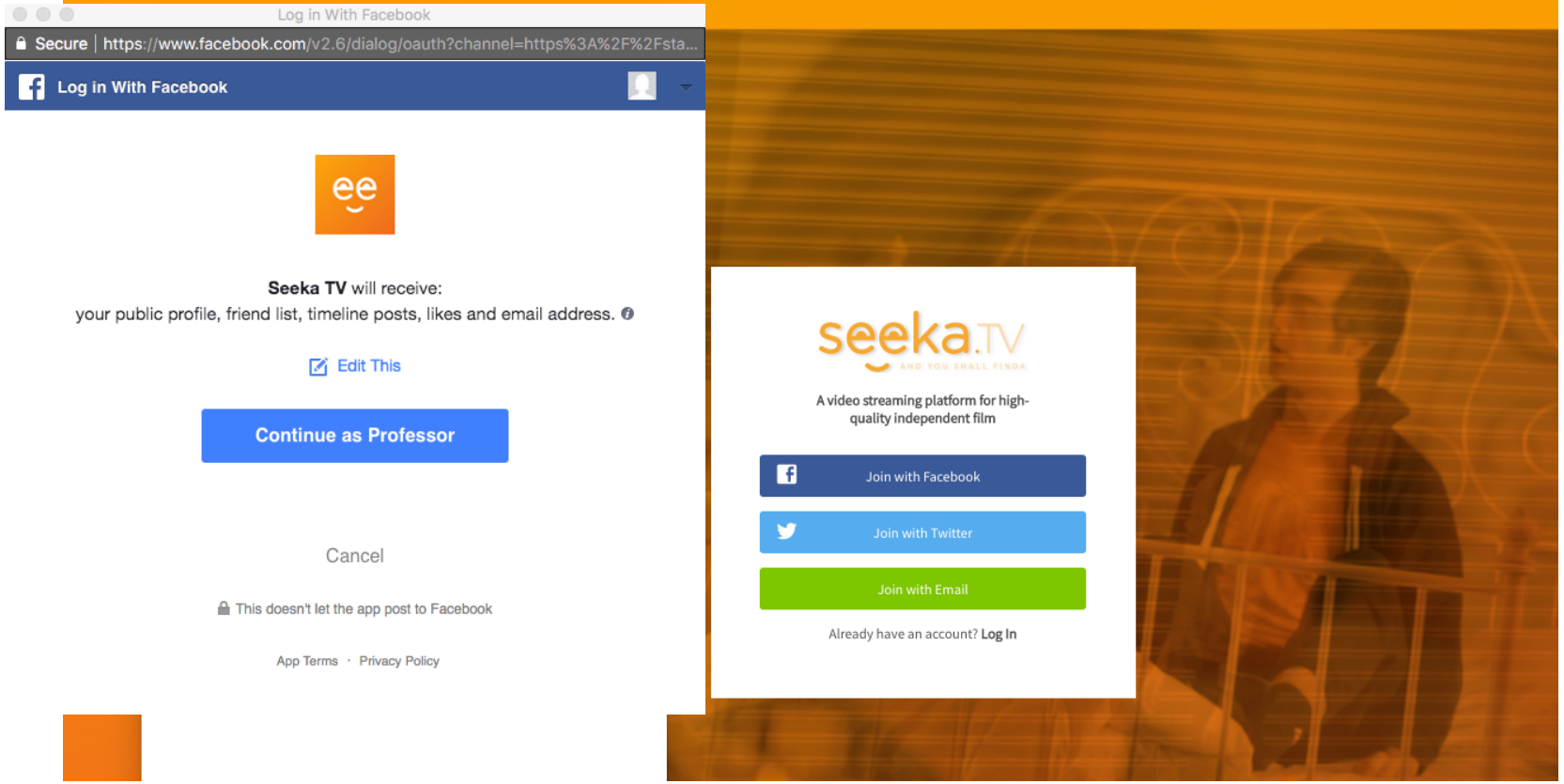
Registration



Registration



Registration



Pros and Cons of Social Media Logon

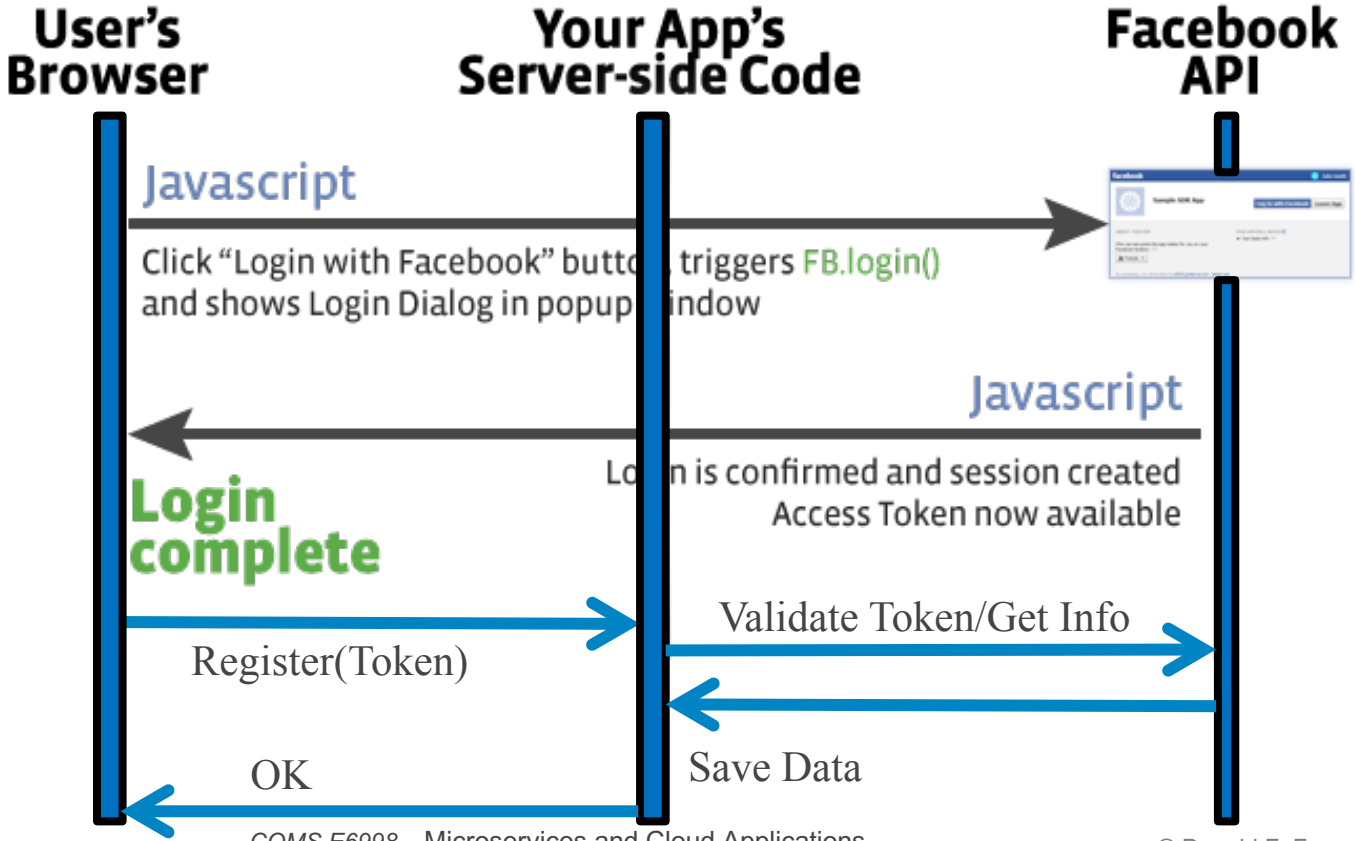
(https://econsultancy.com/blog/61911-the-pros-and-cons-of-a-facebook-login-on-ecommerce-sites)

- 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.

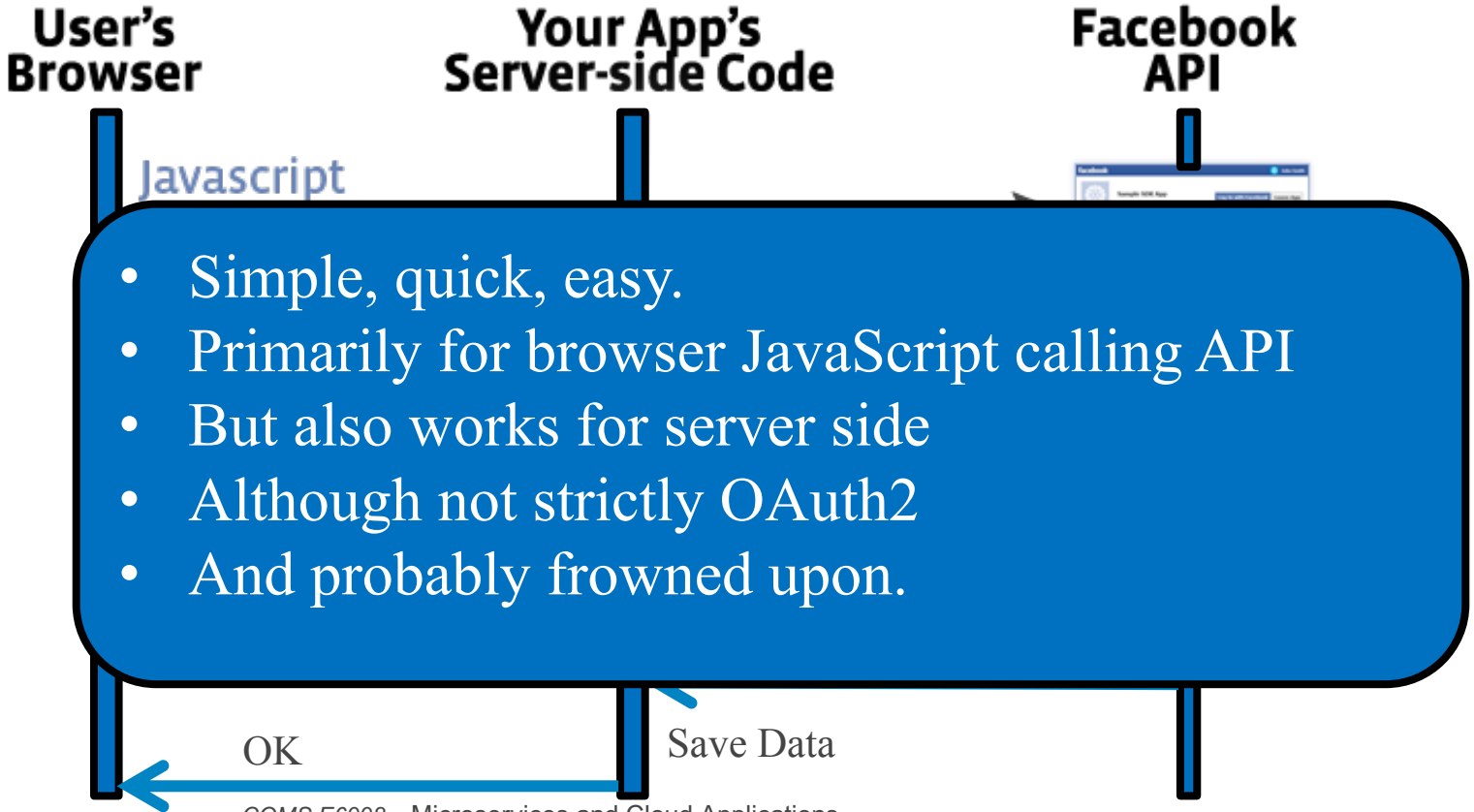
Social Media Registration (Facebook)

Frontend

Facebook



Facebook



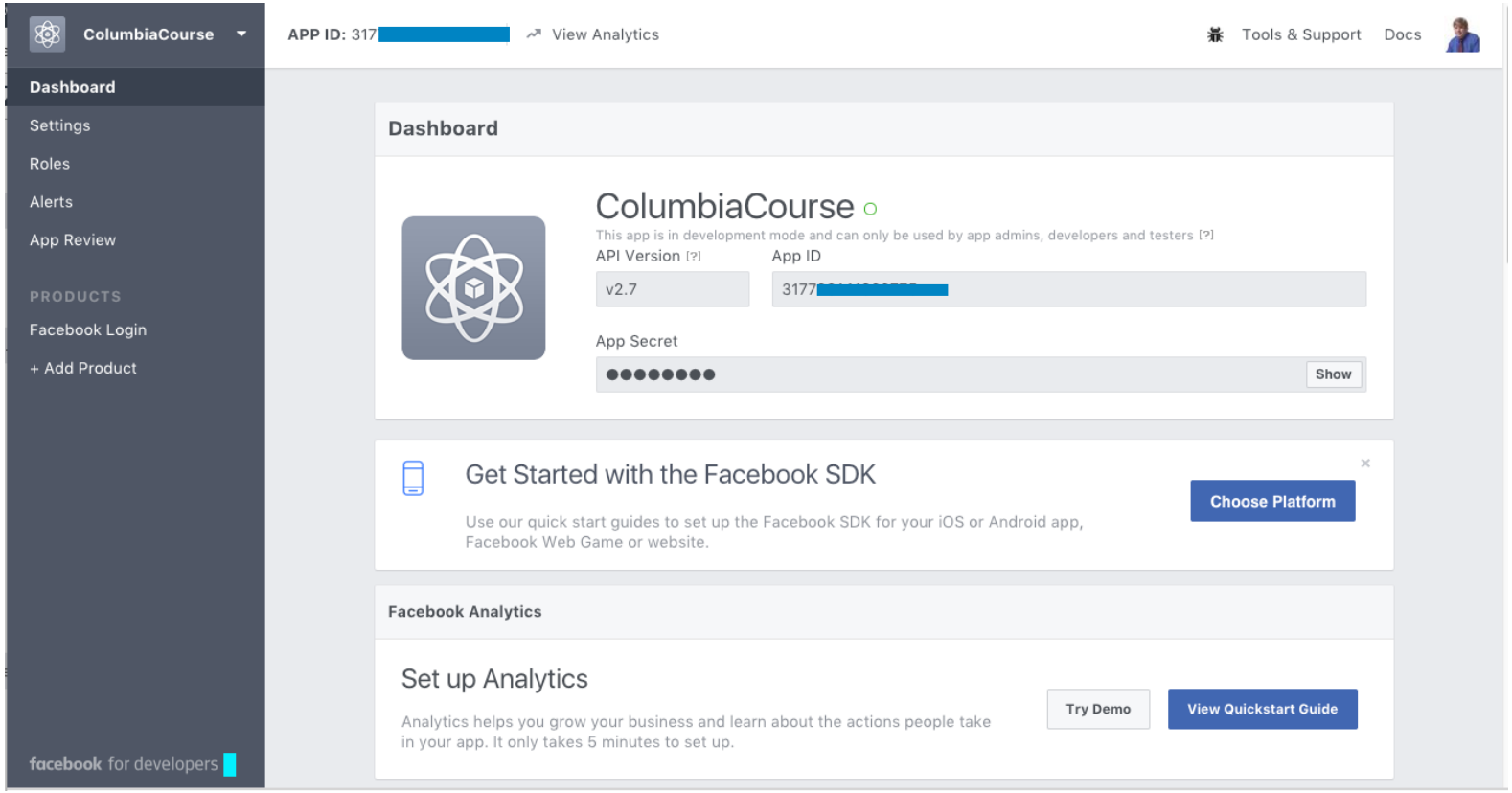
Facebook Login (https://developers.facebook.com/docs/facebook-login/web/)

To implement Facebook Login with the JavaScript SDK, you need a Facebook App ID before you start, which you can create and retrieve on the App Dashboard. You also need somewhere to host HTML files.

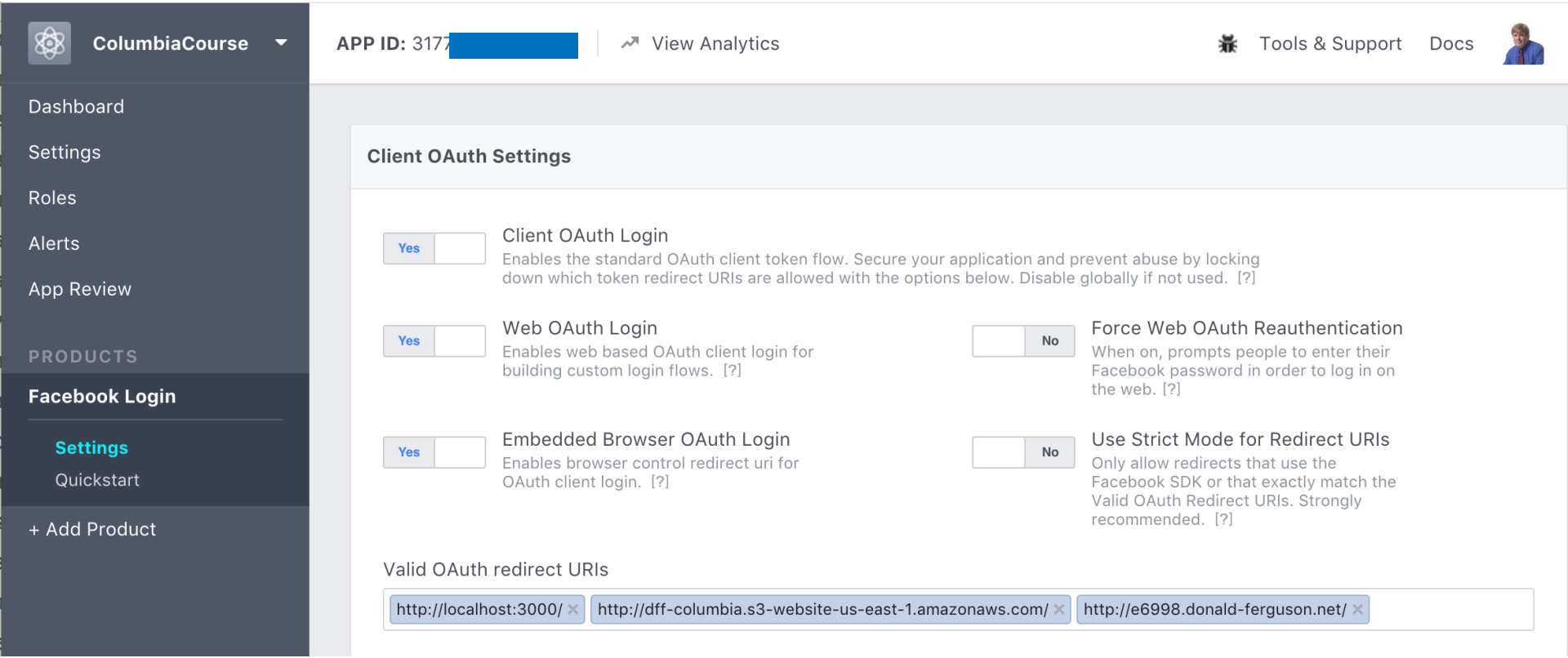
The Implement Login with the following steps:

1. **Choose Your App Settings** to allow email and SMS login, and choose security settings for your app.
2. **Enter Your Redirect URL** to take people to your seccessful-login page.
3. **Check the login status** to see if someone's already logged into your app. During this step, you also should check to see if someone has previously logged into your app, but is not currently logged in.
4. **Log people in**, either with the login button or with the login dialog, and ask for a set of data permissions.
5. **Log people out** to allow them to exit from your app.

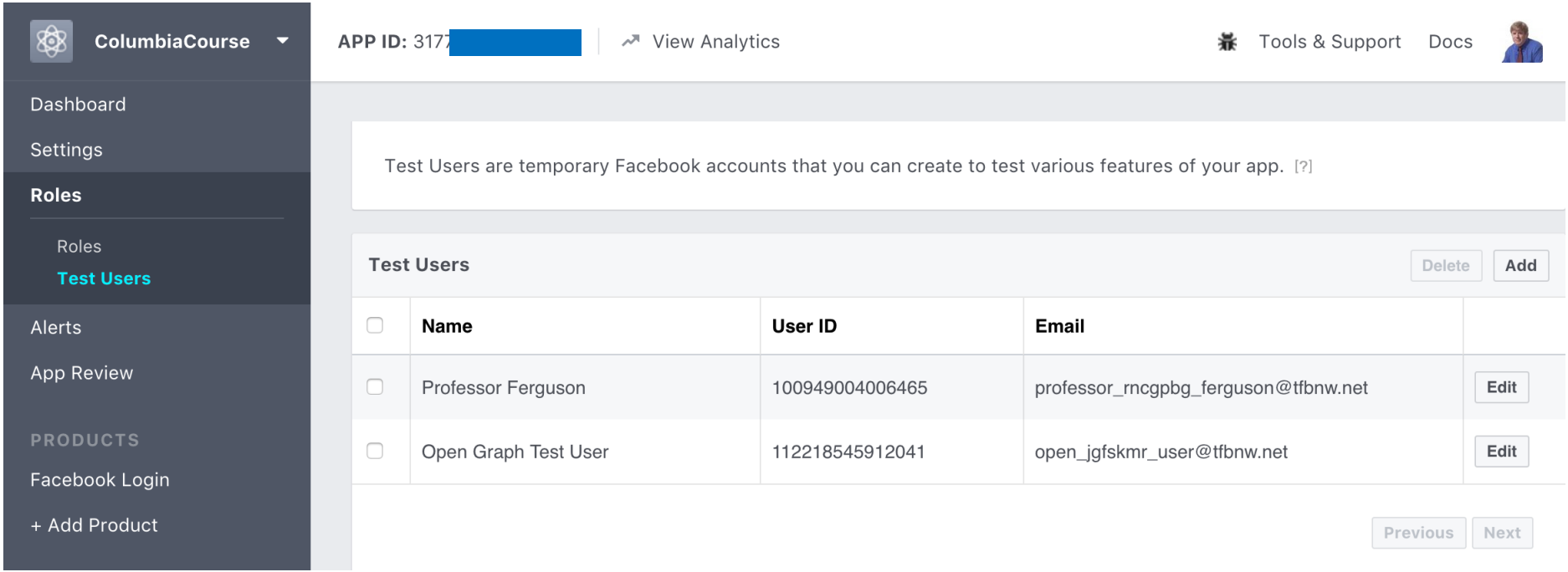
Define Application



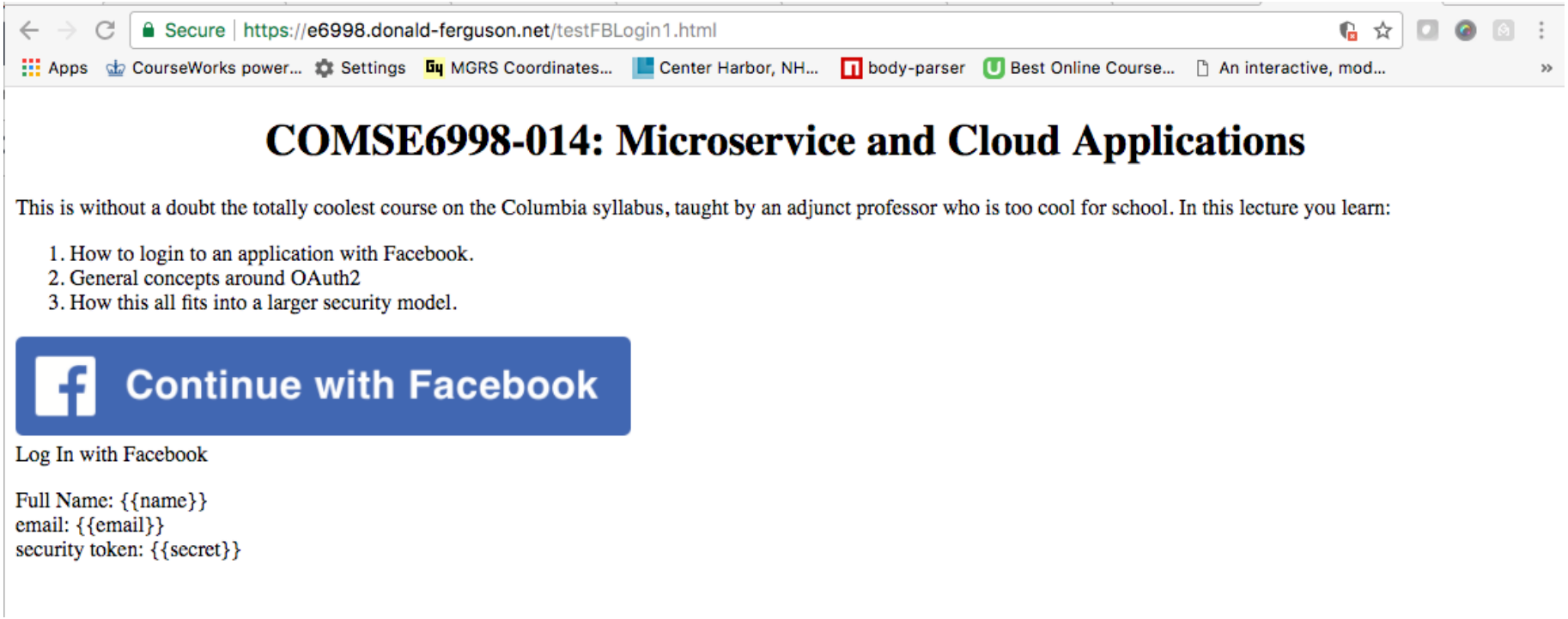
Define Application



Define Application



Web Page



Code (I)

```
<script src="/apiController.js"></script>
<script>

    window.fbAsyncInit = function() {
        FB.init({
            appId      : '3177[redacted]',
            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.

    };

    // 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) HTML

```
<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()">
    <img src="./facebook_login.png" style="width:40%;"><br>
    <span>Log In with Facebook</span>
</div>

<div>
    <p>
    Full Name: {{data.name}}<br>
    email: {{data.email}}<br>
    security token: {{secret}}<br>
    </p>

</div>
```

Code (III) Controller

Do not worry about this if test. The code does not do anything and is there because I copied the code.

```
$scope.login=function() {
    console.log("In $scope.login(), $scope.status = " + $scope.status);

    if (($scope.status == "unknown") || ($scope.status == "undefined") || ($scope.status === "not_authorized")){
        console.log("About to call FB.login()");
        FB.login(function(response) {
            console.log("In FB.login response, response = " + JSON.stringify(response));

            if (response.authResponse) {
                $scope.secret   =   response.authResponse.accessToken;
                console.log('Welcome!  Fetching your information.... ');
                FB.api('/me?fields=id,name,first_name,last_name,age_range,email,gender,link,location,about',
                    function(response) {
                        console.log("In FB.api response, response = " + JSON.stringify(response, null, 4));
                        console.log('Good to see you, ' + response.name + '.');
                        $scope.email    =   response.email;
                        $scope.name     =   response.name;
                        $scope.token    =   response.id;
                        $scope.gender       =   response.gender;
                        $scope.status       =   'connected';

                        $scope.data=response;

                        $scope.$apply();

                        registerWithServer(response).then(
                            function(rsp) {},
                            function(error) {});
                    }
                );
            } else {
                console.log('User cancelled login or did not fully authorize.');
            }
            console.log("Leaving FB.login() response");
        },
        {scope: 'email,user_likes,publish_actions,user_posts'});
    }
    console.log("Leaving $scope.login()");
};
```

Code (III) Controller

Your login/registration server call goes here.

In a “services” module.

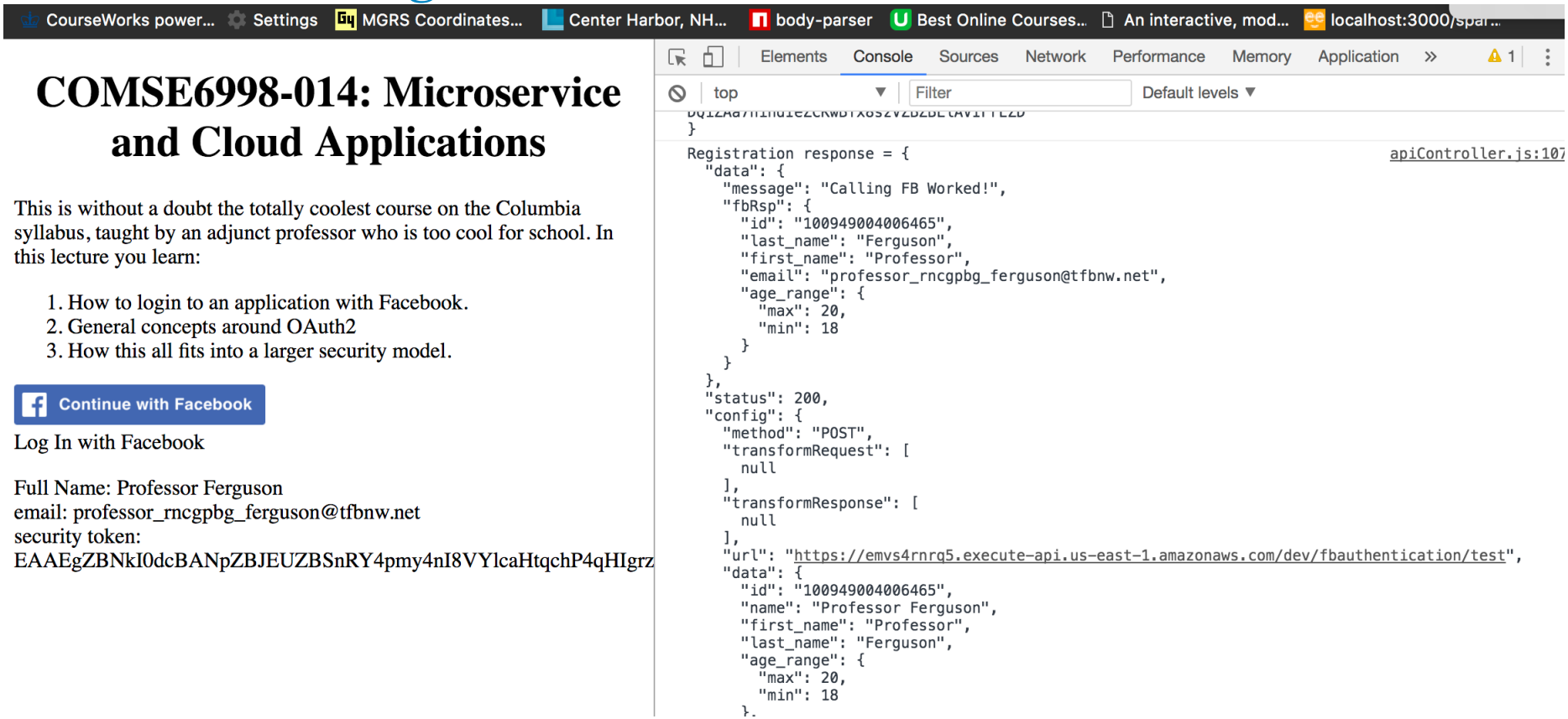
```
$scope.login=function() {
    console.log("In $scope.login(), $scope.status = " + $scope.status);

    if (($scope.status == "unknown") || ($scope.status == "undefined") || ($scope.status === "not_authorized")){
        console.log("About to call FB.login()");
        FB.login(function(response) {
            console.log("In FB.login response, response = " + JSON.stringify(response));

            if (response.authResponse) {
                $scope.secret   =   response.authResponse.accessToken;
                console.log('Welcome!  Fetching your information.... ');
                FB.api('/me?fields=id,name,first_name,last_name,age_range,email,gender,link,location,about',
                    function(response) {
                        console.log("In FB.api response, response = " + JSON.stringify(response, null, 4));
                        console.log('Good to see you, ' + response.name + '.');
                        $scope.email    =   response.email;
                        $scope.name     =   response.name;
                        $scope.token    =   response.id;
                        $scope.gender       =   response.gender;
                        $scope.status       =   'connected';

                        $scope.data=response;

                        $scope.$apply();

                        registerWithServer(response).then(
                            function(rsp) {},
                            function(error) {});
                    }
                );
            } else {
                console.log('User cancelled login or did not fully authorize.');
            }
            console.log("Leaving FB.login() response");
        },
        {scope: 'email,user_likes,publish_actions,user_posts'});
    }
    console.log("Leaving $scope.login()");
};
```

Sending Registration Request

```
var registerWithServer = function(response) {
    var url1 = 'https://emvs4rnrq5.execute-api.us-east-1.amazonaws.com/dev/fbauthentication/test';
    var url2 = 'http://e6998.donald-ferguson.net/fbauthentication/test';
    console.log("Sending body = " + JSON.stringify(response,null,2));
    $http.post(url1,
        response, {}).then(
            function(result) {
                console.log("Registration response = " + JSON.stringify(result,null,2));
            },
            function(error) {
                console.log("Registration = " + JSON.stringify(error))
            });
};
```

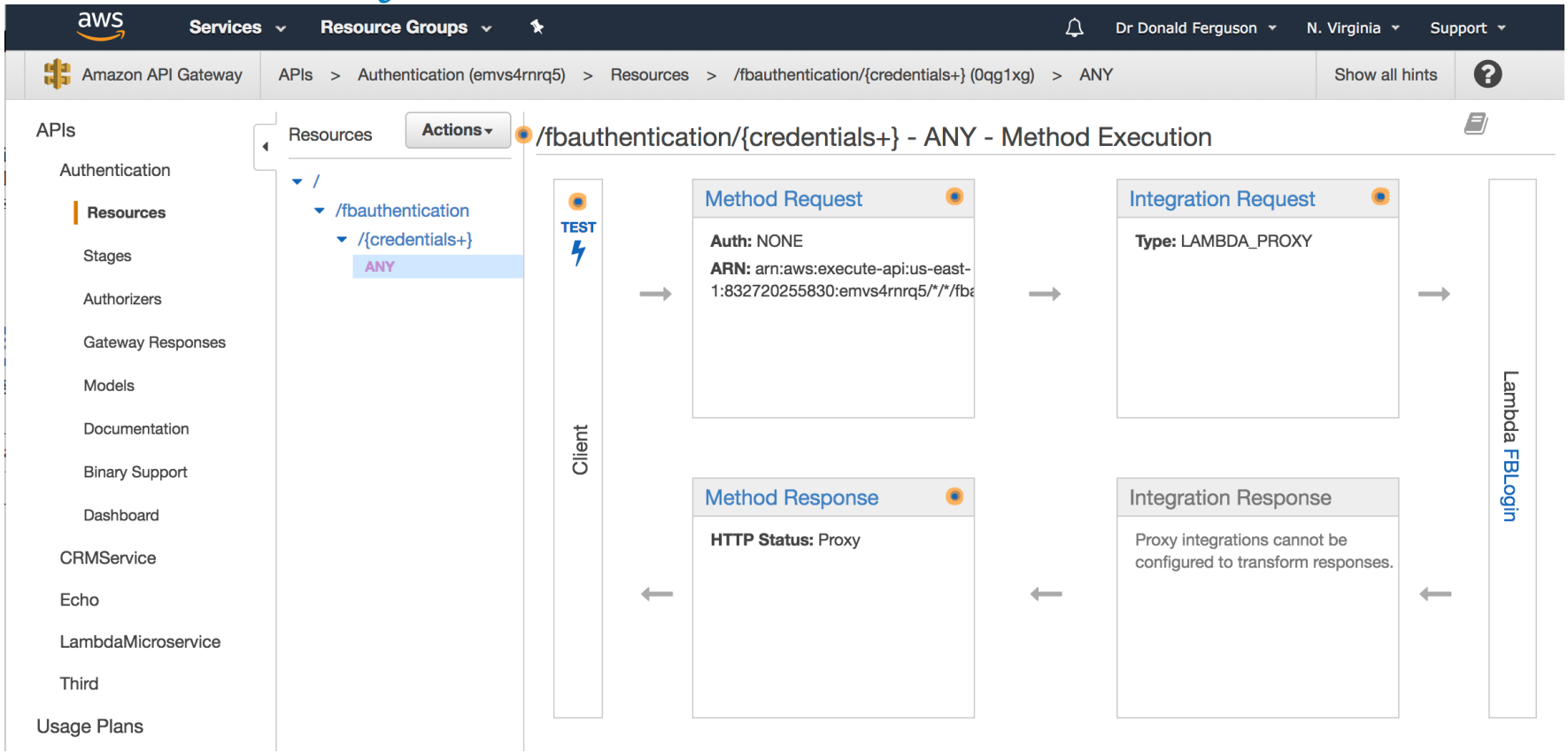
- I had some trouble with the CloudFront → API Gateway.
  Using direct GW URL for now.
- Also a little wonkiness around HTTP/HHTPS
- Had poor Internet connectivity this weekend and will figure out this week.

Facebook Login



Backend

API Gateway



Lambda Input (I)


```
{
    "resource": "/fbauthentication/{credentials+}",
    "path": "/fbauthentication/test",
    "httpMethod": "POST",
    "headers": {
        "Accept": "application/json, text/plain, */*",
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "en-US,en;q=0.8",
        "CloudFront-Forwarded-Proto": "https",
        "CloudFront-Is-Desktop-Viewer": "true",
        "CloudFront-Is-Mobile-Viewer": "false",
        "CloudFront-Is-SmartTV-Viewer": "false",
        "CloudFront-Is-Tablet-Viewer": "false",
        "CloudFront-Viewer-Country": "US",
        "content-type": "application/json;charset=UTF-8",
        "Host": "emvs4rnrq5.execute-api.us-east-1.amazonaws.com",
        "origin": "https://e6998.donald-ferguson.net",
        "Referer": "https://e6998.donald-ferguson.net/testFBLogin1.html",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36",
        "Via": "2.0 fb5794831ad522a7a9b0d102a91d1696.cloudfront.net (CloudFront)",
        "X-Amz-Cf-Id": "vnCHCZ_hLPiIBr54EGtawszL9YfKvYKw7iCbxo6OypkUmcV71rK95g==",
        "X-Amzn-Trace-Id": "Root=1-59f72256-53c0ee2256269fa006f8325f",
        "X-Forwarded-For": "107.14.54.0, 54.182.236.70",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    },
    "queryStringParameters": null,
    "pathParameters": {
        "credentials": "test"
    },
```

Lambda Input (II)


```
    "stageVariables": null,
    "requestContext": {
        "path": "/dev/fbauthentication/test",
        "accountId": "832720255830",
        "resourceId": "0qg1xg",
        "stage": "dev",
        "requestId": "406f32d4-bd72-11e7-996f-1512193281e3",
        "identity": {
            "cognitoIdentityPoolId": null,
            "accountId": null,
            "cognitoIdentityId": null,
            "caller": null,
            "apiKey": "",
            "sourceIp": "107.14.54.0",
            "accessKey": null,
            "cognitoAuthenticationType": null,
            "cognitoAuthenticationProvider": null,
            "userArn": null,
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36",
            "user": null
        },
        "resourcePath": "/fbauthentication/{credentials+}",
        "httpMethod": "POST",
        "apiId": "emvs4rnrq5"
    },
    "body": "{\"id\":\"100949004006465\",\"name\":\"Professor Ferguson\",\"first_name\":\"Professor\",\"last_name\":\"Ferguson\",\"age_range\":{\"max\":20,\"min\":18},
    "isBase64Encoded": false
}
```

Lambda Input (III)

```
2017-10-30T13:00:06.432Z 406ee3e4-bd72-11e7-8aaa-895af9389d1e Body =
{
    "id": "100949004006465",
    "name": "Professor Ferguson",
    "first_name": "Professor",
    "last_name": "Ferguson",
    "age_range": {
        "max": 20,
        "min": 18
    },
    "email": "professor_rncgpbg_ferguson@tfbnw.net",
    "gender": "male",
    "link": "https://www.facebook.com/app_scoped_user_id/100949004006465/",
    "secret": "EAAEgZBNkI0dcBANpZBJEUZBSnRY4pmy4nI8VYlcaHtqchP4qHIgrzViBiyQXuPvnbRimiqtD0OY1NEc1ZA75ccQ4UKbTAXHoAVKIWTFx3yBomcgrQiWDUHSZBkMw0WZCZCTDpj4xI4uZAwjuuYSZBs651(
}
```

▸	13:00:07	2017-10-30T13:00:07.016Z 406ee3e4-bd72-11e7-8aaa-895af9389d1e FB Login worked.
▾	13:00:07	2017-10-30T13:00:07.052Z 406ee3e4-bd72-11e7-8aaa-895af9389d1e { id: '100949004006465', last_name: 'Ferguson', first_name: 'Professor', email: 'professor_

```
2017-10-30T13:00:07.052Z 406ee3e4-bd72-11e7-8aaa-895af9389d1e { id: '100949004006465',
last_name: 'Ferguson',
first_name: 'Professor',
email: 'professor_rncgpbg_ferguson@tfbnw.net',
age_range: { max: 20, min: 18 } }
```

Lambda Function (I)

```
var FB = require('fb'),
    fb = new FB.Facebook({version: 'v2.7'});


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

    var body = JSON.parse(event.body);
    console.log("Body = " + JSON.stringify(body,null,2));

    const done = function(err, res) {

        callback(null, {
            statusCode: err ? '400' : '200',
            body: err ? err.message : JSON.stringify(res),
            headers: {
                'Content-Type': 'application/json',
                "X-Requested-With": '*',
                "Access-Control-Allow-Headers": 'Content-Type,X-Amz-Date,Authorization,X-Api-Key,x-requested-with,TENANT',
                "Access-Control-Allow-Origin": '*',
                "Access-Control-Allow-Methods": 'GET,POST,OPTIONS'

            },
        });
    };
```

Lambda Function (II)

- Call FB API and get basic info, which also validates token.
- Call FB to covert
  - 60 minute token
  - Into long lived token
- Save token and profile info in user DB (TBD)

```
// First test the incoming token to see if it is valid and we can get info.
FB.api('me',
    {
        fields: ['id', 'last_name', 'first_name', 'email', 'age_range'],
        access_token: body.secret
    },
    function (res) {
        console.log("FB Login returned. Response = " + JSON.stringify(res, null, 2));

        // Optimistic that I did not get an error. Exchange for a long-lived token
        FB.api('oauth/access_token', {
                client_id: '3177281[redacted]',
                client_secret: 'fab[redacted]',
                grant_type: 'fb_exchange_token',
                fb_exchange_token: body.secret
            },
            function (res) {
                if (!res || res.error) {
                    console.log(!res ? 'error occurred' : res.error);
                }
                console.log("FB exchange = " + JSON.stringify(res));

                var accessToken = res.access_token;
                var expires = res.expires ? res.expires : 0;
                var m = {
                    message: "Calling FB to exchange token worked.!",
                    fbRsp: res
                };
                done(null, m);
            });
    });
```

Execution Console Messages (I)

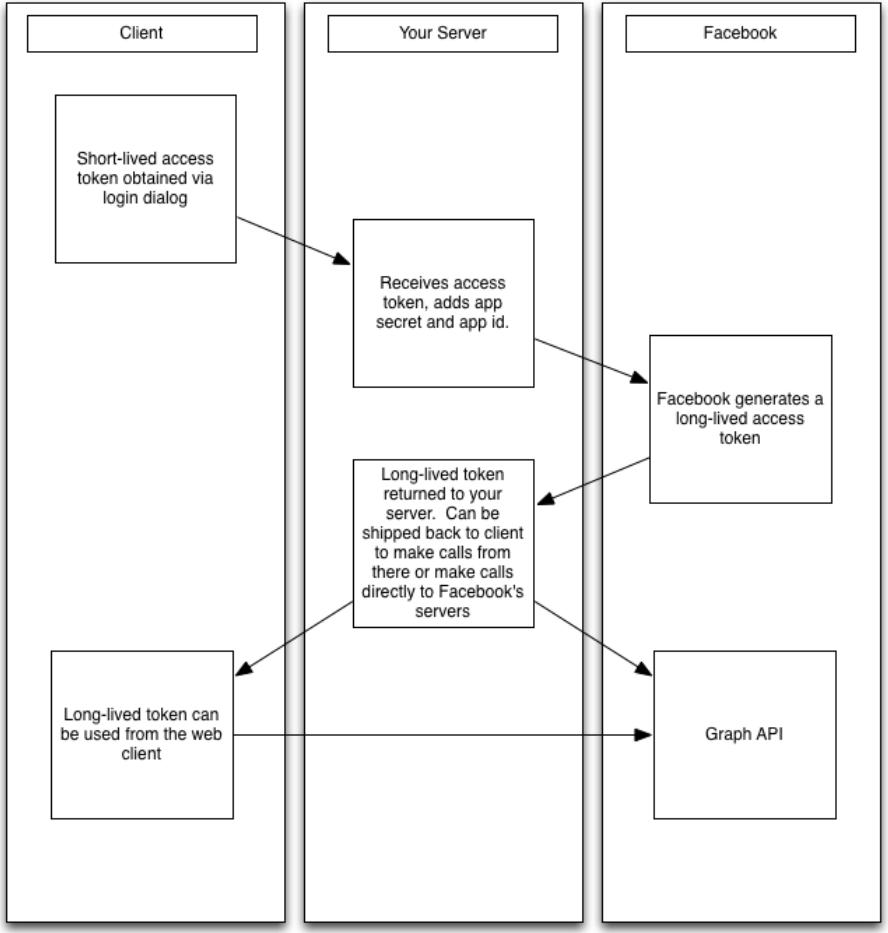
```
2017-10-30T14:36:20.678Z b227d34d-bd7f-11e7-b206-e5cd8a271e29 Body =
{
    "id": "100949004006465",
    "name": "Professor Ferguson",
    "first_name": "Professor",
    "last_name": "Ferguson",
    "age_range": {
        "max": 20,
        "min": 18
    },
    "email": "professor_rncgpbg_ferguson@tfbnw.net",
    "gender": "male",
    "link": "https://www.facebook.com/app_scoped_user_id/100949004006465/",
    "secret": "EAAEgZBNkI0dcBANHhI8liGBrFkTq0UCwVhD5S5ZBClRBXZByZBaNGZC0N7F2KstYQ4ZBB7McPmvUCSZAAV
}
```

```
▾ 14:36:21   2017-10-30T14:36:21.241Z b227d34d-bd7f-11e7-b206-e5cd8a271e29 FB Logi

2017-10-30T14:36:21.241Z b227d34d-bd7f-11e7-b206-e5cd8a271e29 FB Login returned. Response =
{
    "id": "100949004006465",
    "last_name": "Ferguson",
    "first_name": "Professor",
    "email": "professor_rncgpbg_ferguson@tfbnw.net",
    "age_range": {
        "max": 20,
        "min": 18
    }
}
```

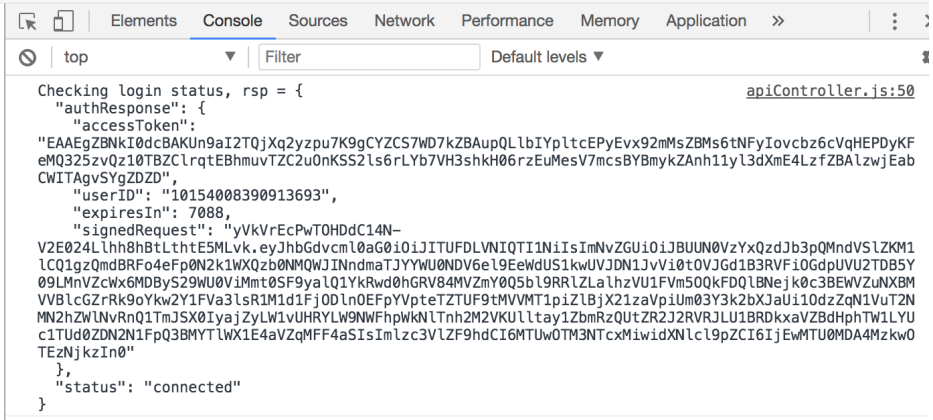
Execution Console Messages (II)

```
▼ 14:36:21          2017-10-30T14:36:21.476Z b227d34d-b
2017-10-30T14:36:21.476Z b227d34d-bd7f-11e7-b206-e5cd8a271e29 FB
{
    "access_token": "EAAEgZBNkI0dcBAPpz2R5GRpVkQFt82Y9vMdskB1ZB0
    "token_type": "bearer",
    "expires_in": 5183023
}
```

- Initial token is valid for 60 minutes.
- Long lived token above is good for 5183023/3600/60 = 60 days.
- Eliminates annoying re-login, even if customer already logged on.
- Server can refresh token as long as user has not removed application.



A Slight Modification



- Before showing login button, you can check the status wrt your application
  1. connected: User is logged onto Facebook and has previously authorized app.
  2. not_authorized: User is logged on but has not authorized application.
  3. unknown: User not logged on.

  Only show login button and drive dialog in cases (2) and (3).

Controller Code

```
$scope.checkStatus = function() {
    FB.getLoginStatus(function(response) {
        console.log("Checking login status, rsp = " + JSON.stringify(response,null,2));
        if (response.status === 'connected') {
            // the user is logged in and has authenticated your
            // app, and response.authResponse supplies
            // the user's ID, a valid access token, a signed
            // request, and the time the access token
            // and signed request each expire
            var uid = response.authResponse.userID;
            var accessToken = response.authResponse.accessToken;
        } else if (response.status === 'not_authorized') {
            // the user is logged in to Facebook,
            // but has not authenticated your app
        } else {
            // the user isn't logged in to Facebook.
        }
    });
};
```

- I just use the button to show triggering call.
- Can check login status silently on page load.

Social Media Registration (OAuth2 – full) (Twitter)

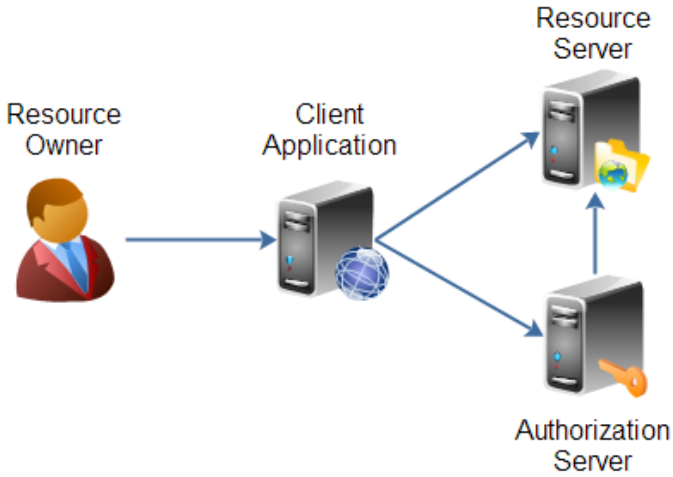
Overview (http://tutorials.jenkov.com/oauth2/index.html)

- 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:

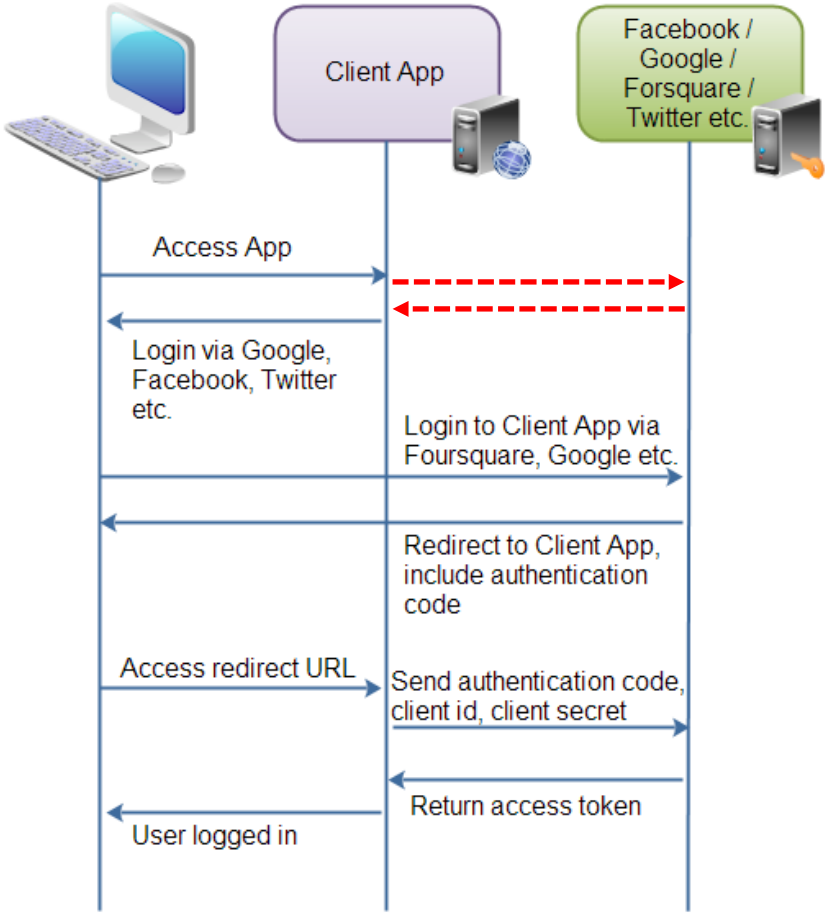
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:



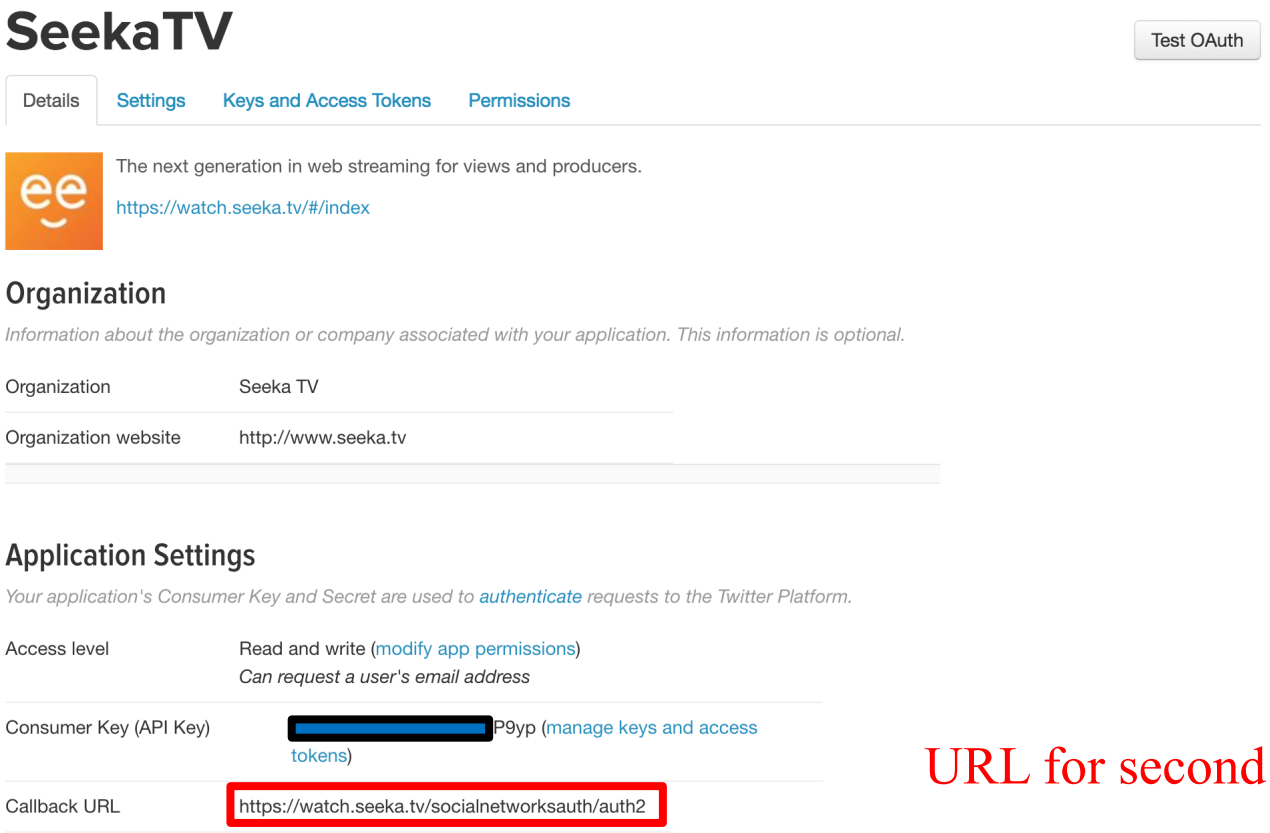
Roles/Flows

- 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.



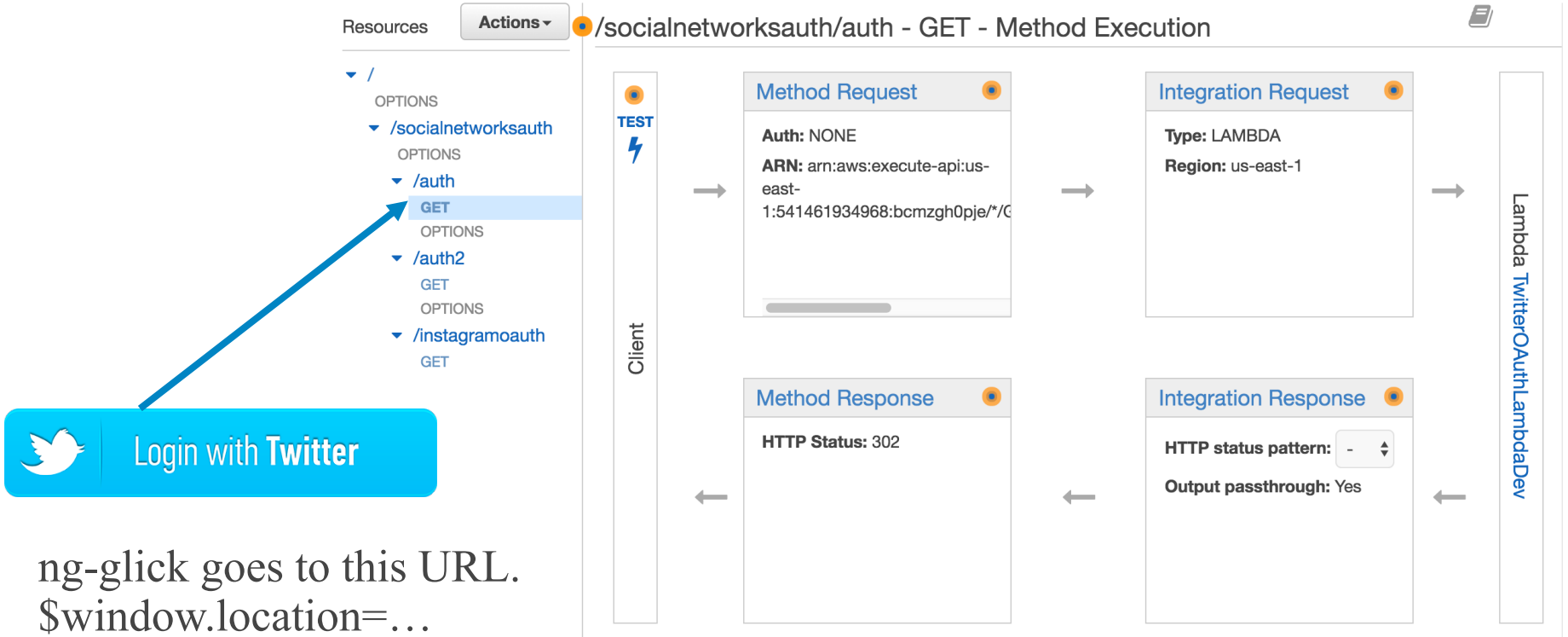
Twitter Example

Twitter Application Configuration



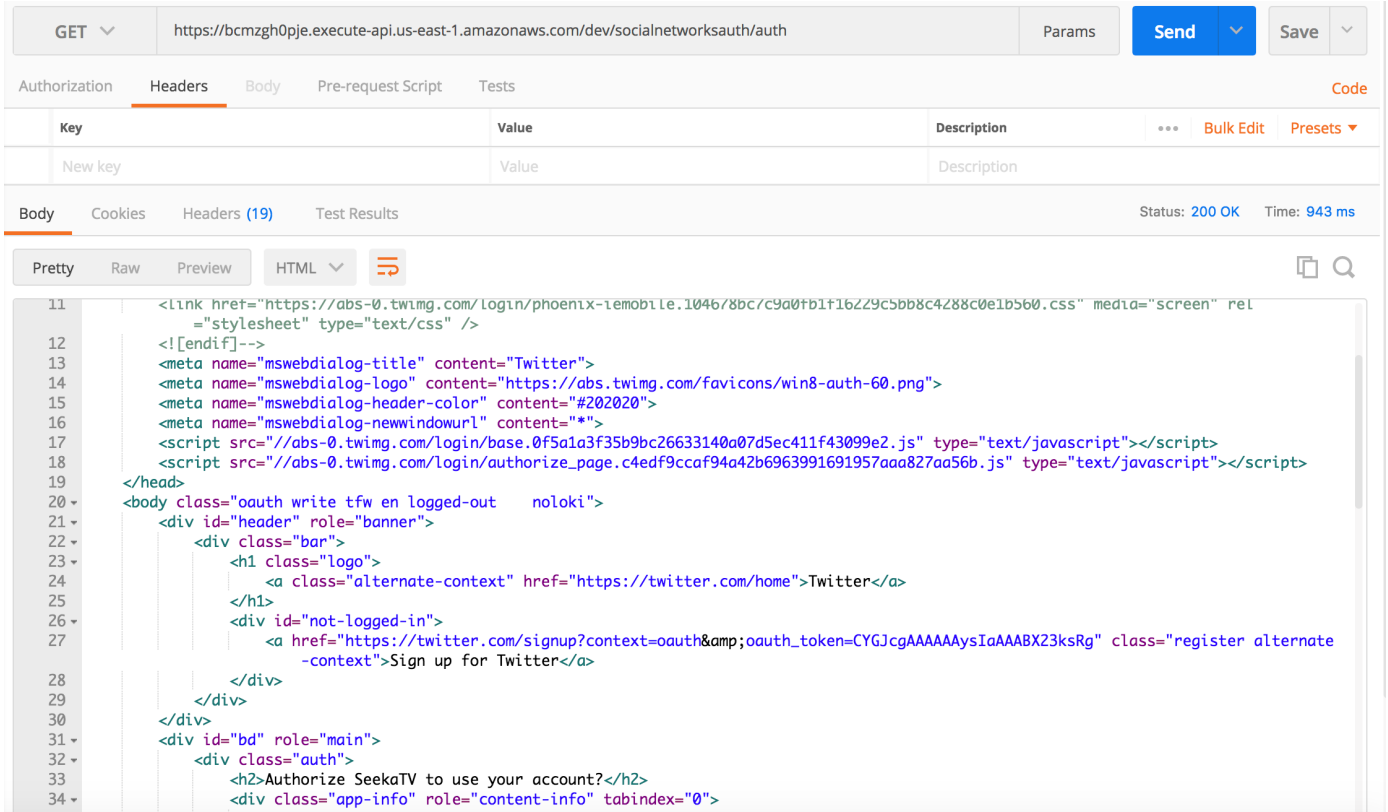
URL for second redirect.

Twitter OAuth2 – Phase 1



ng-glick goes to this URL.
$window.location=…

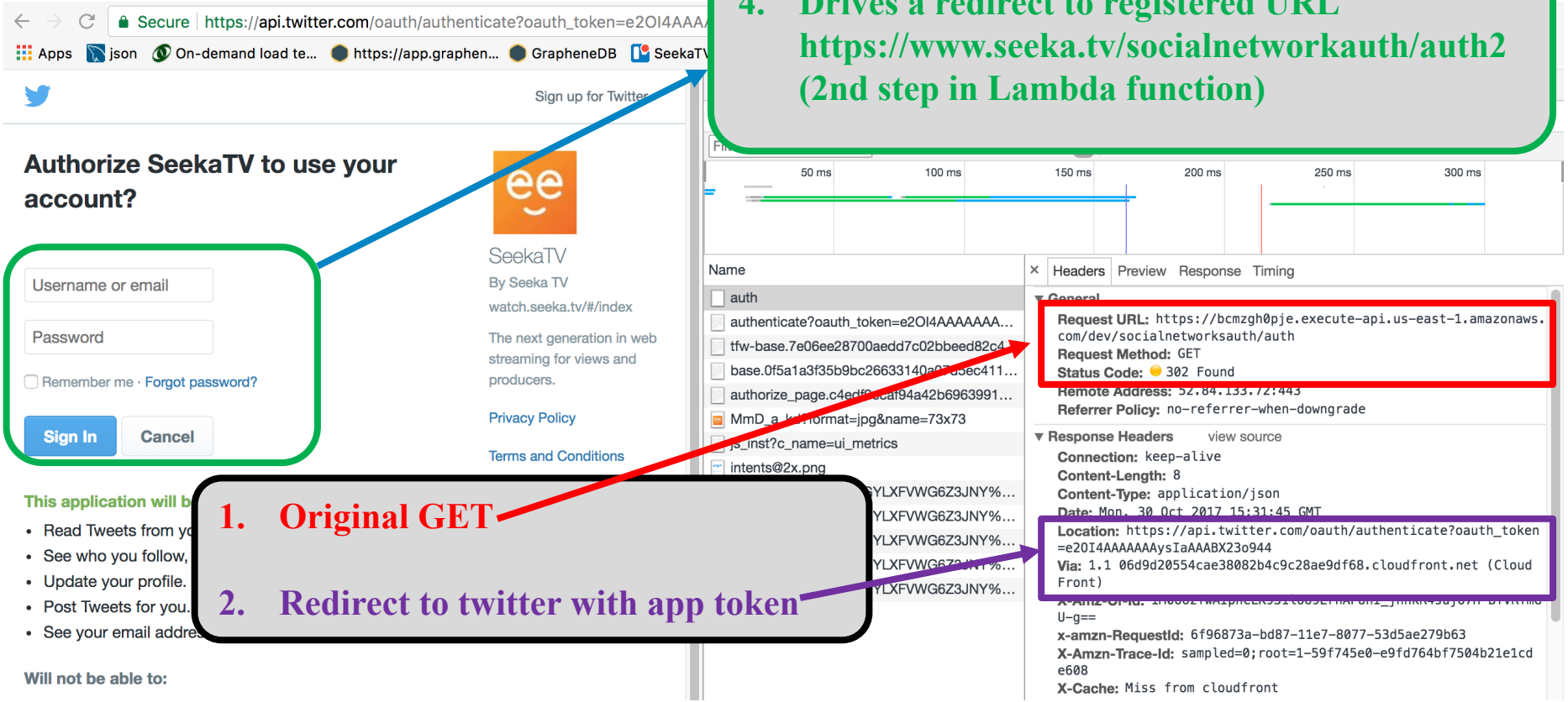
Calling API on Click



Redirected To Twitter Logon Page

A Little More Detail

3. **Successful login to Twitter**
4. **Drives a redirect to registered URL https://www.seeka.tv/socialnetworkauth/auth2 (2nd step in Lambda function)**



1. **Original GET**
2. **Redirect to twitter with app token**

2nd Step in Lambda (I)

```
public static HTTPRedirect twitterRedirect2(TwitterAuth accessToken) {
    long id;
    twitter4j.User u = null;
    AccessToken authorizationToken = null;
    String s = null;

    try {

        logger.debug("Attempting step 2 in Twitter Login. accessToken = " + JSONUtil.getInstance().toPrettyJSON(accessToken));

        TwitterConnector.initializeConnector();

        logger.debug("Attempting to load state from step 1.");
        SimpleRequestToken st = TwitterConnector.loadState(accessToken.oauth_token);

        if (st != null) {
            logger.debug("Got simple request token = " + JSONUtil.getInstance().toPrettyJSON(st));
            RequestToken rt = new RequestToken(st.token, st.secret);

            logger.debug("Attempting to get Authorization Token");
            authorizationToken = twitter.getOAuthAccessToken(rt, accessToken.oauth_verifier);
            logger.debug("Got Authorization Token = " + JSONUtil.getInstance().toPrettyJSON(authorizationToken));
            s = JSONUtil.getInstance().toSimpleJSON(authorizationToken);

            logger.debug("Trying to get info from Twitter.");
            id = twitter.getId();
            logger.debug("ID = " + id);
            u = twitter.showUser(id);
            logger.debug("User = " + prettyGson.toJson(u));
        }
    } catch (Exception e) {
        logger.debug("Exception = " + e);
        e.printStackTrace();
        u = null;
    }
```

Correlate with previous step to handle multiple user logins.

Call Twitter to convert to long life, final bearer token.

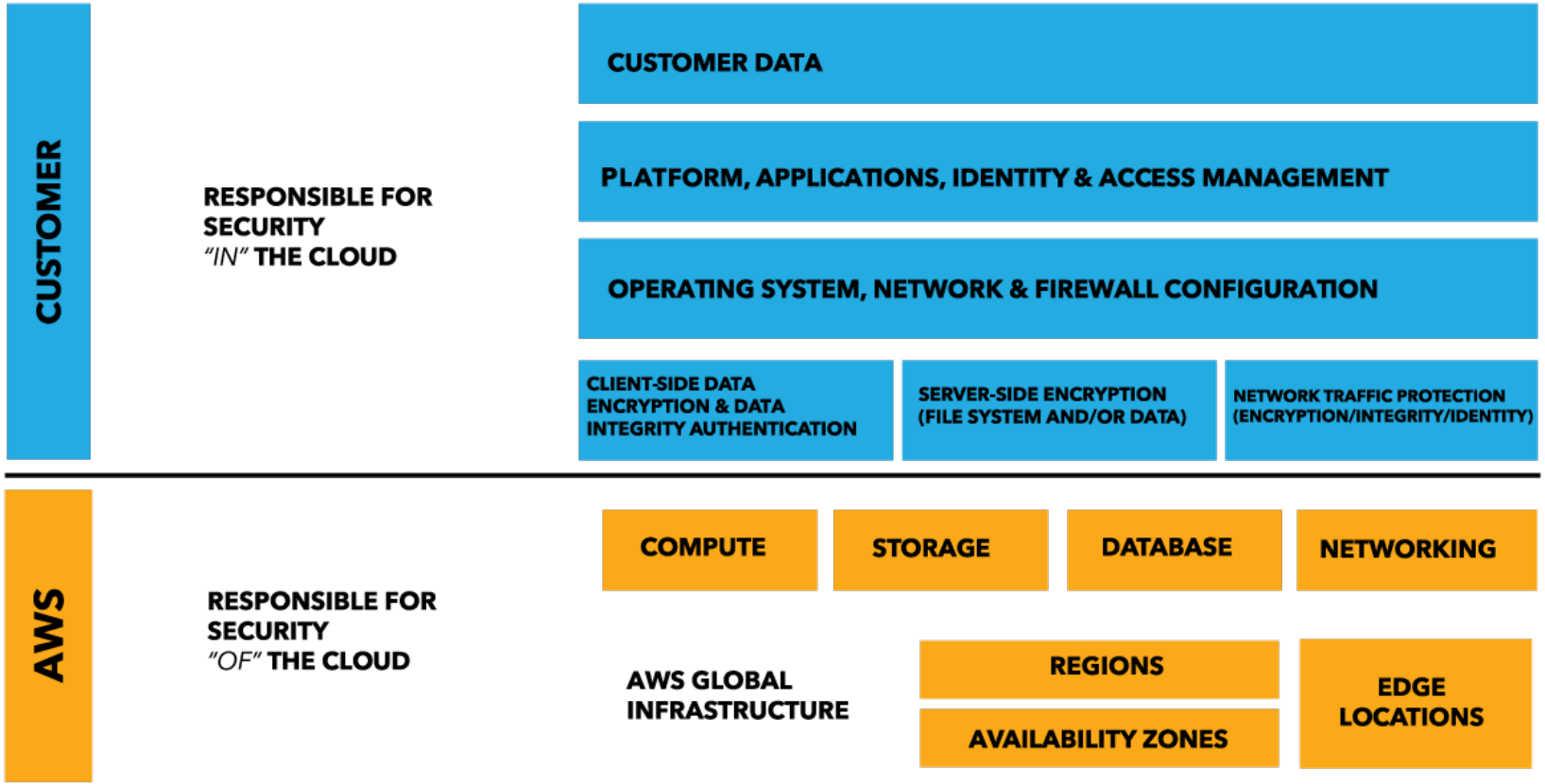
2nd Step in Lambda (I)

Generate redirect with user ID and info → Your apps' UI page.

```
HTTPRedirect r;
if (u != null) {
    r = new HTTPRedirect("https://" + SEEKA_TV_DOMAIN + "/#/twitterintegration?name=" + u.getName() + "&id="
            + u.getId() + "&token=" + authorizationToken.getToken() + "&tokenSecret="
            + authorizationToken.getTokenSecret() + "&result=true", HTTPRedirect.success);

    /*
     * Comment out the code above and uncomment this code if you want to test with UI code running on a local
     * server/test machine.

    r = new HTTPRedirect("http://" + "localhost:3000" + "/#/twitterintegration?name=" + u.getName() + "&id="
            + u.getId() + "&token=" + authorizationToken.getToken() + "&tokenSecret="
            + authorizationToken.getTokenSecret() + "&result=true", HTTPRedirect.success);
            */

} else {
    r = new HTTPRedirect(TwitterConnector.failureUrl, HTTPRedirect.success);

}
logger.debug("returning " + JSONUtil.getInstance().toPrettyJSON(r));
System.out.println("returning (println) redirect = " + JSONUtil.getInstance().toPrettyJSON(r));
return r;
```

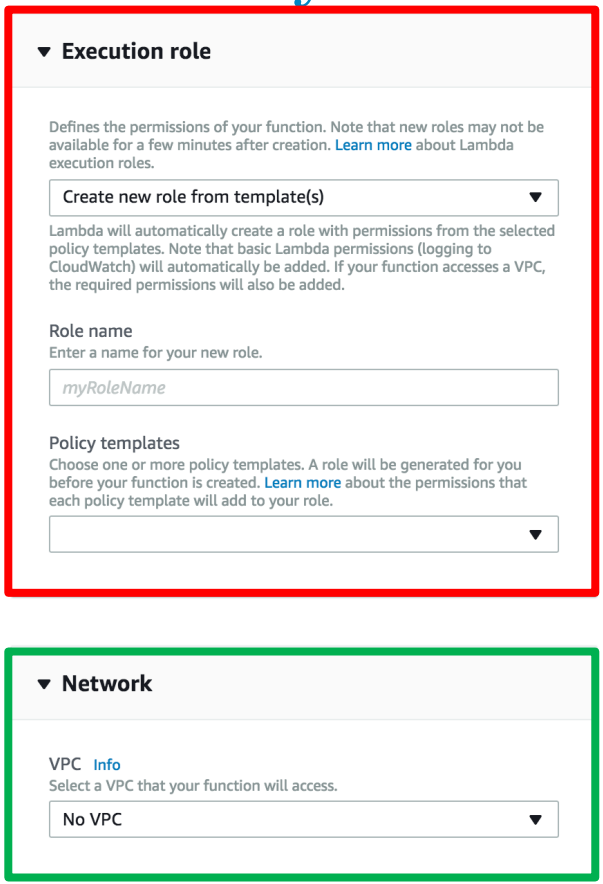
Authorization

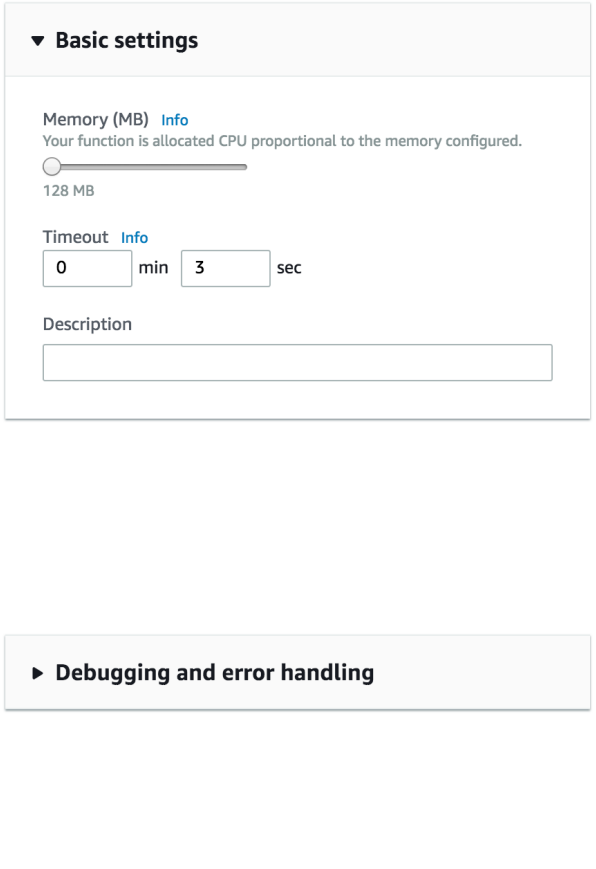
Division of Responsibility



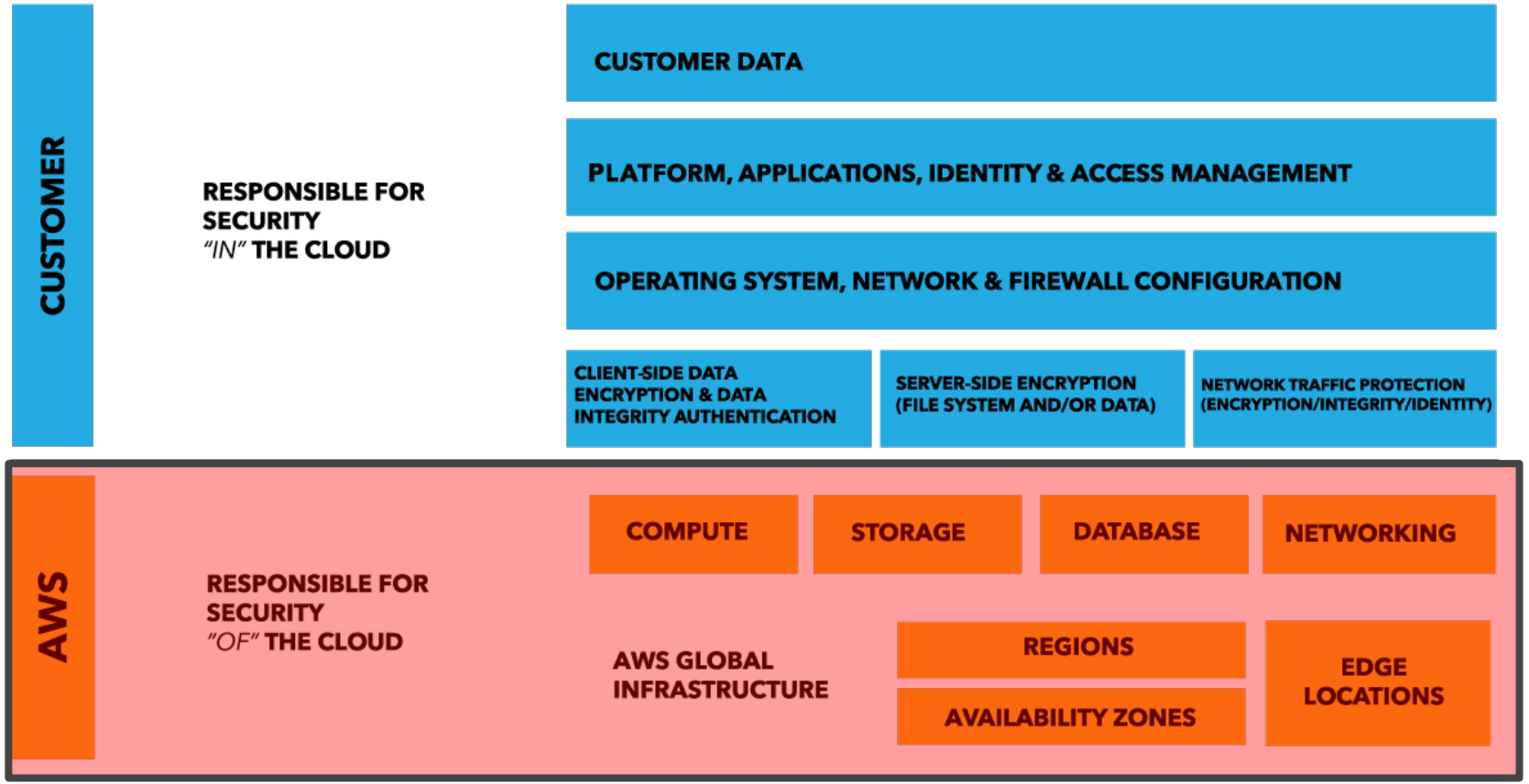
Division of Responsibility

- Resources
  - DynamoBD tables.
  - S3 buckets.
  - SNS topics.
  - etc.
- Have unique ARNs.
- Your elements
  - That "do something," e.g.
    - EC2 instance.
    - BeanStalk app.
    - Lambda functions.
  - Execute in a role (ID)
  - Have associated policies that specify allowed
    - Operations
    - On resources





Division of Responsibility



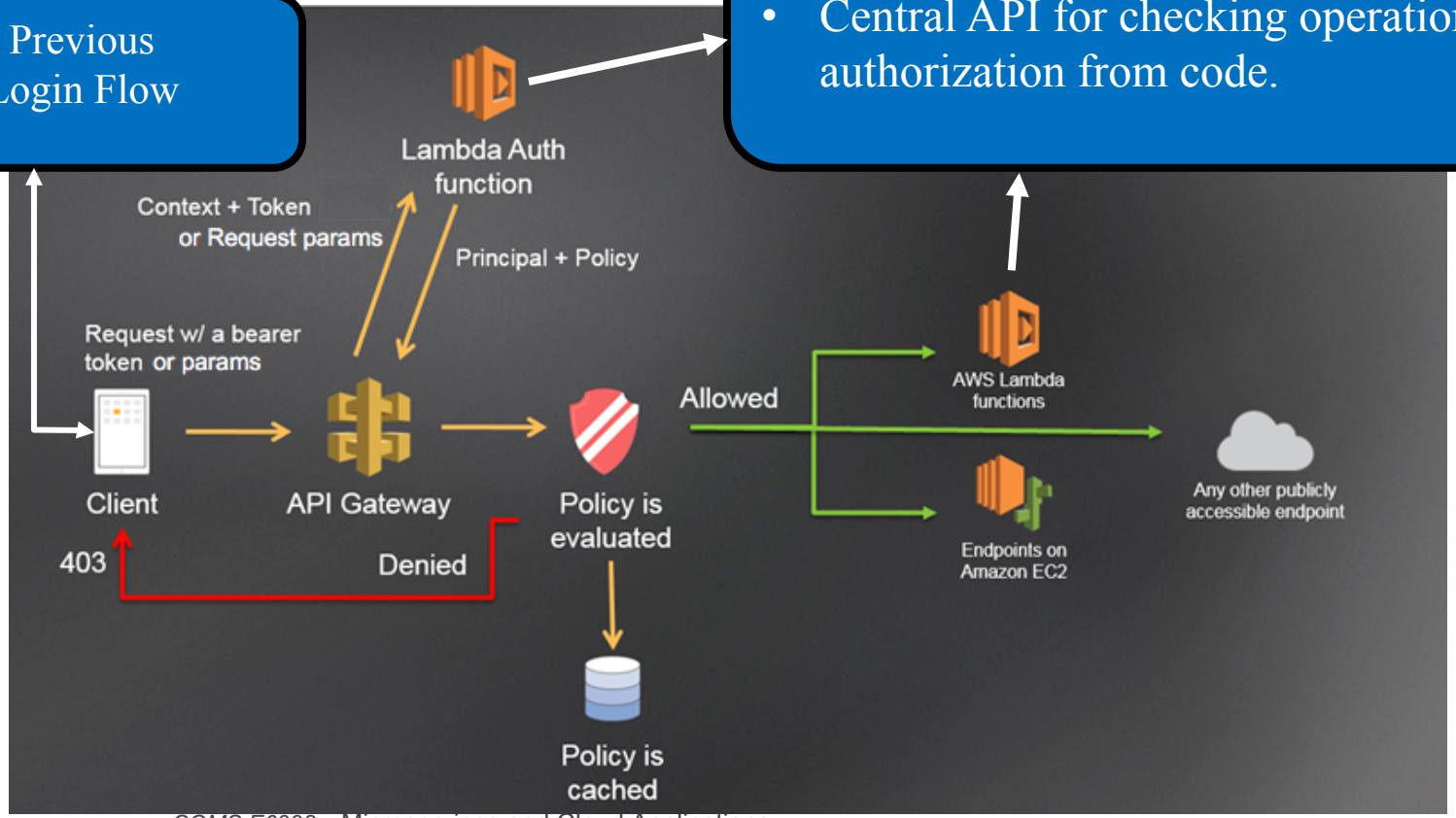
Division of Responsibility

- The application and supporting middleware and libraries are responsible for authorization of application-level resources, e.g.
  - Specific database tables or tuples.
  - Specific DynamoDB tables or documents.
  - Performing specific operations on URLs.
- Same basic model
  - Application defined “resource names,” e.g.
    - /customers/accounts/21
    - /customers/*
  - Authenticated users maps to named roles with associated policies, e.g.
    - {“Agent”, “/crm/agents/21/accounts/*”, {POST, DELETE, GET, PUT}}
    - {“Customer”, ”/commerce/customer/carts/*”, {POST, DELETE, GET, PUT}}
- Authorization checks
  - Explicit calls in application code, or
  - Handler/interceptor middleware similar to ETag and Idempotency checks.
- Security admin application that allows
  - Defining application resource templates and operations.
  - Creating and associating policies and roles.
  - Table for mapping IDs to roles and policies.

Authorization

Previous Login Flow

Security Microsevice
- CRUD resources ID, roles, policies
- Central API for checking operation authorization from code.



Authorization

- User logon to service/environment
  - Via email, Facebook, Twitter
  - Generates and returns a JWT token with identity, role and policy information.
  - Which flows in a customer header on also subsequent requests.
- API Gateway
  - Calls Custom Authorizer with headers, URL, body, query params.
  - Custom authorizer calls authorization microservice (2nd Lambda)
  - Which returns decision → Customer Authorizer → API Gateway.
- Other environments, e.g. Lambdas, BeanStalk
  - Also call authorization microservice
  - Passing resource, user info, operation
  - To determine if authorization allowed.

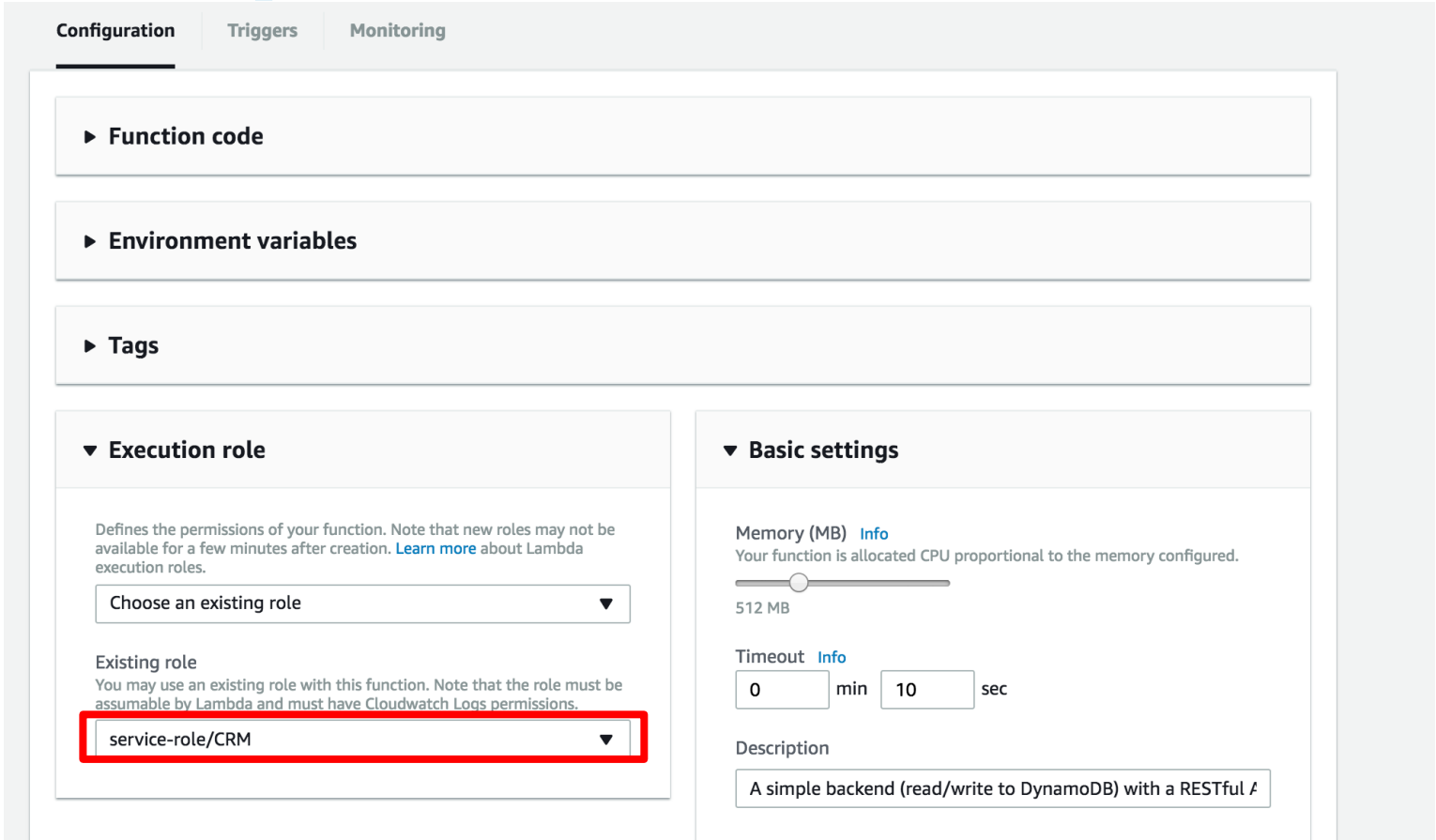
2nd part of project 4 will be fleshing out authorization to support 1st part of project 4, which is authentication/registration via social media.

Backup

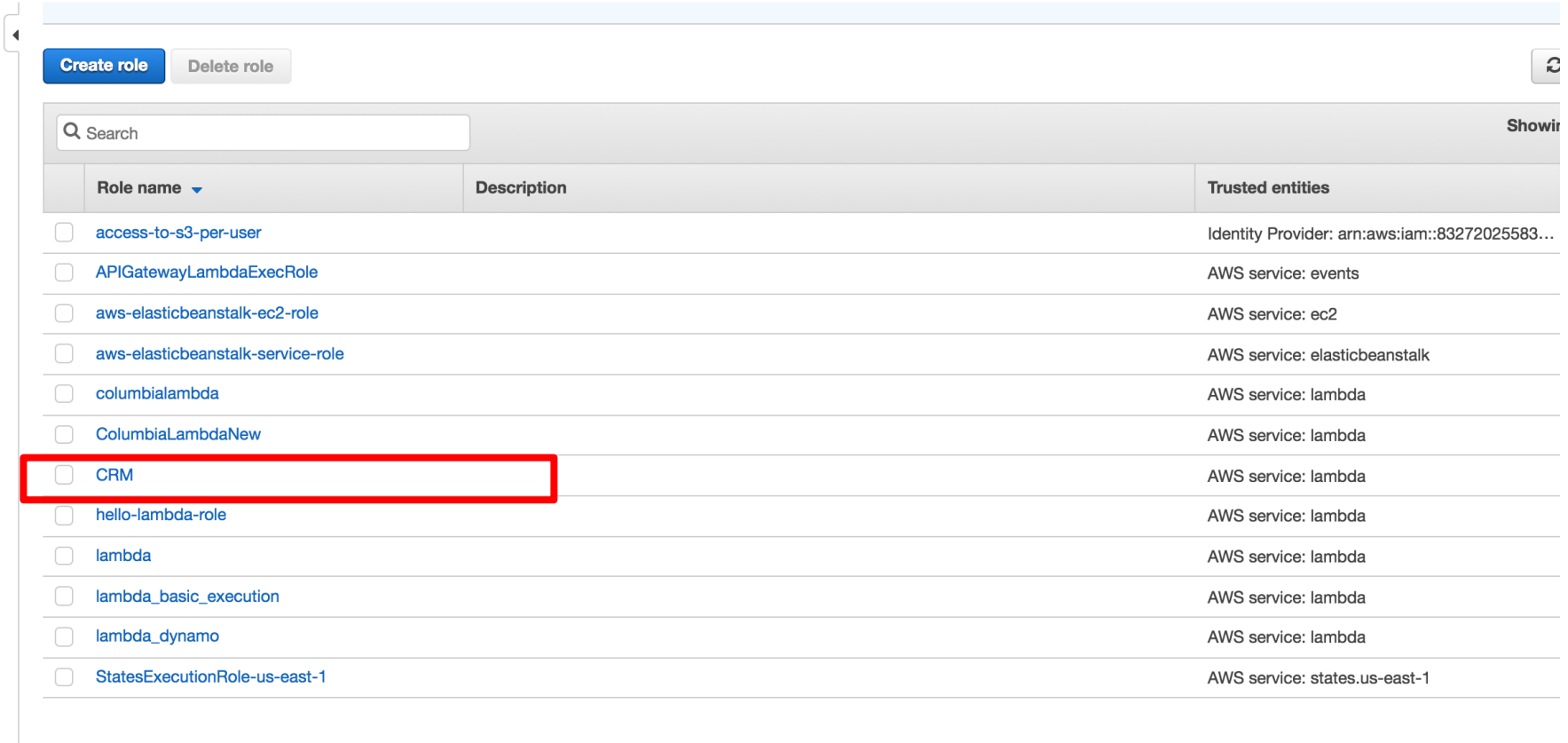
Authorization

- 1st 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.

An Example – Lambda Function Role



Role/Policies



Role/Policy