COMSE6998: Modern Serverless Cloud Applications

Lecture 5: TLDS, Pagination, OAuth2, Social Media

Dr. Donald F. Ferguson
Donald.F.Ferguson@gmail.com

Contents

- Introduction
  - TAs
  - Homework status
  - Q&A
- ~~Evolution and Motivation: Web apps → SOA/Microservices → Serverless~~
- Top-Level Design Specification
- New technical topics: Social Media Integration (Motivation) ➔ Topics
  - Pagination
  - OAuth2
  - Social Media Integration (Intro.)
  - ~~Introduction to Graph Databases~~

Introduction

TAs

- Kushwanth Ram Kesarla Shantharam
  - Just assigned.
  - Should be attending class today.

- Jingxiao Gu
  - Attends class.
  - Has announced office hours.
  - Active on Piazza.

- One more to be named, but have a good candidate.

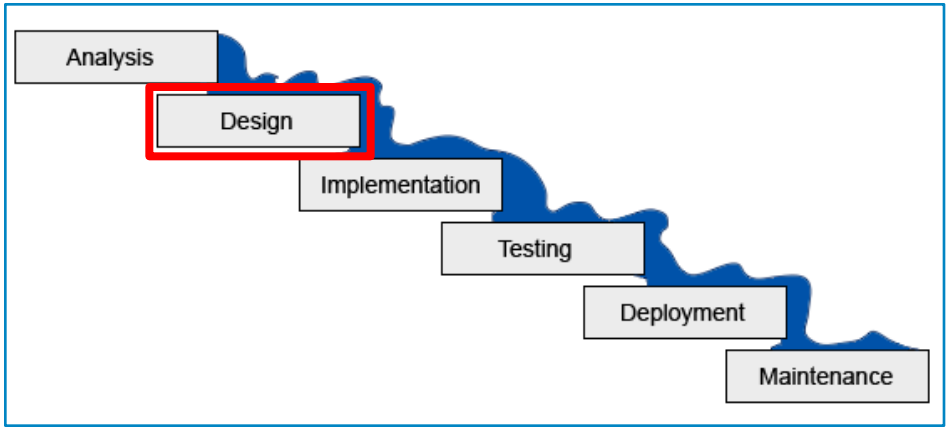
Homework Status

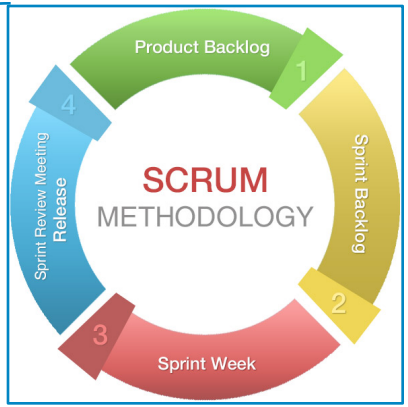
Q&A

Top-Level Design Specification

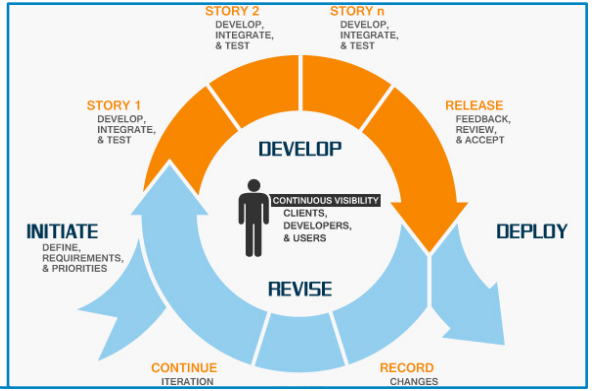
Agile Development

Traditional "Waterfall" Methodology





Design seems to be missing?



Observations

Weinberg's Second Law: “If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.”

Sr. Celestina: “If you cannot write it down, you do not understand it.”

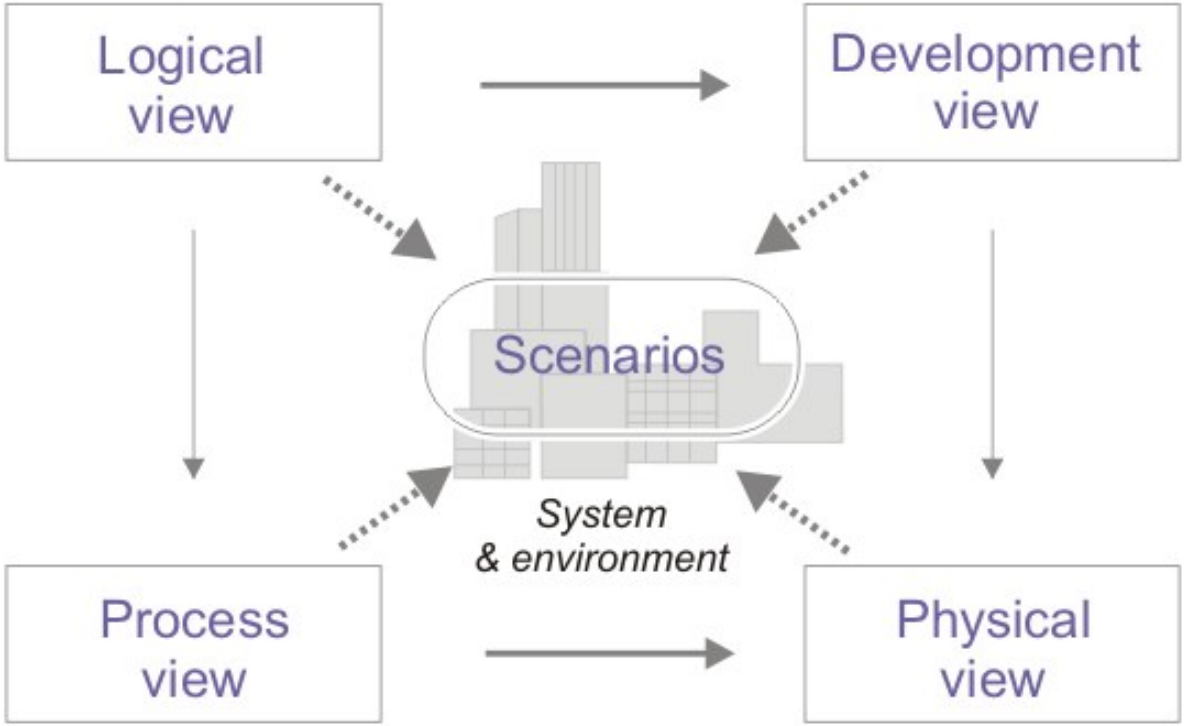
“it is a custom
More honor'd in the breach than the observance”
Hamlet Act 1, scene 4, 7–16

Agile Manifesto (http://agilemanifesto.org/):

- Individuals and interactions over processes and tools
- Working software over **comprehensive** documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

4+1 architectural view model

http://en.wikipedia.org/wiki/4%2B1_architectural_view_model



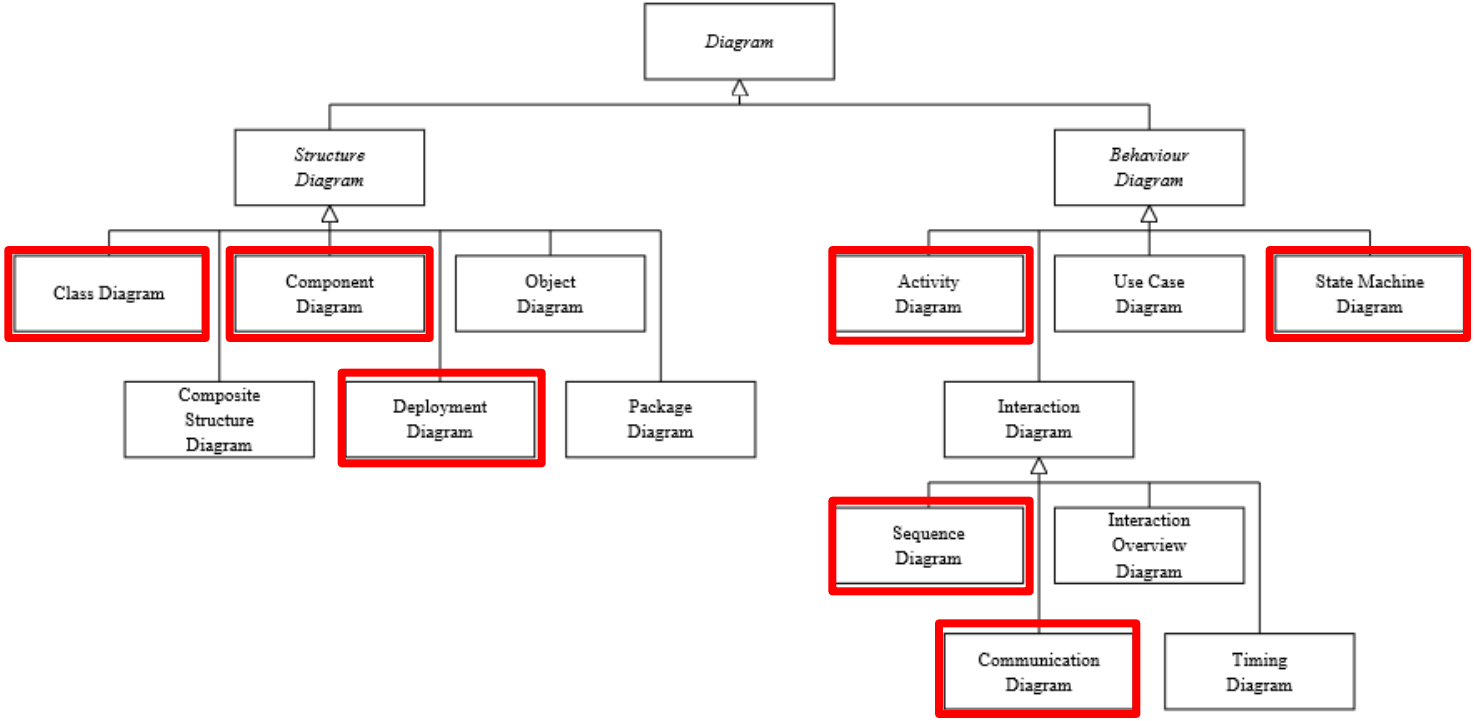
The Architecture Views

http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

- ***Logical view*** : The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include Class diagram, Communication diagram, Sequence diagram.[2]
- ***Development view*** : The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.[2]
- ***Process view*** : The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the Activity diagram.[2]
- ***Physical view*** : The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the Deployment diagram.[2]
- ***Scenarios*** : The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is **also known as use case view**

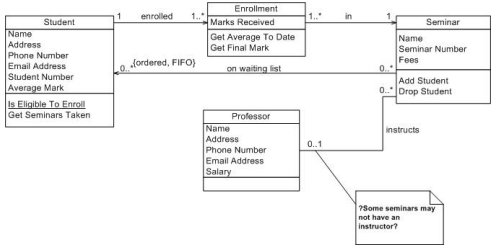
Some Diagrams

http://www.agilemodeling.com/essays/umlDiagrams.htm

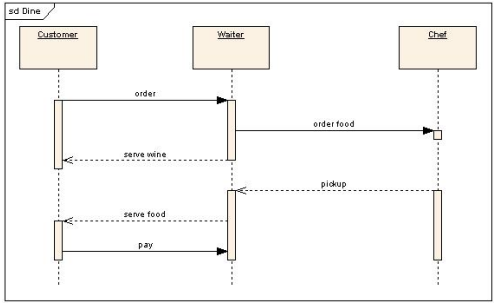


Some Diagrams

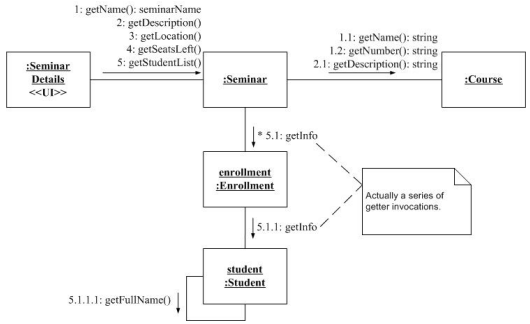
Class Diagram



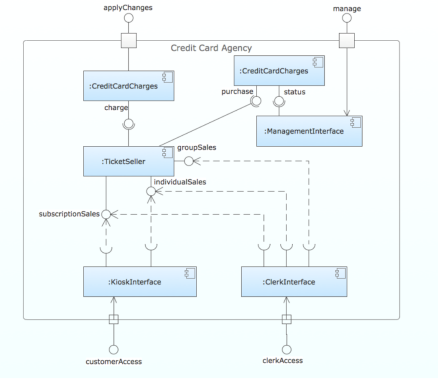
Sequence Diagram



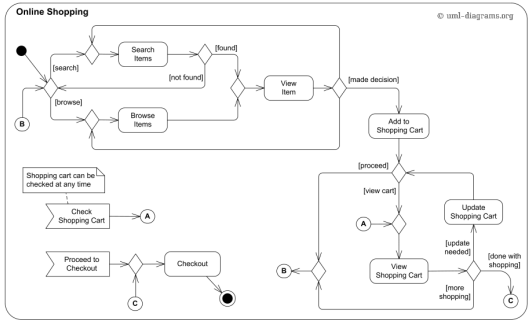
Communication Diagram



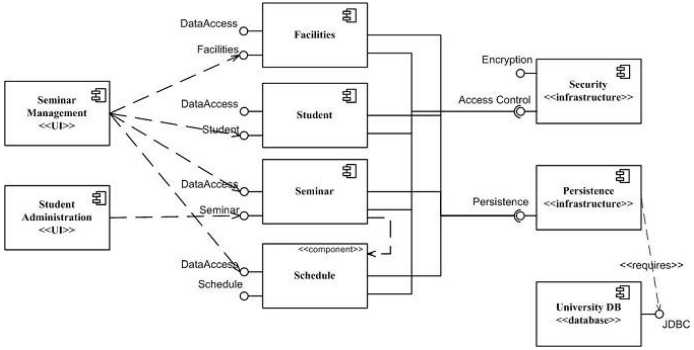
Component Diagram



Activity Diagram

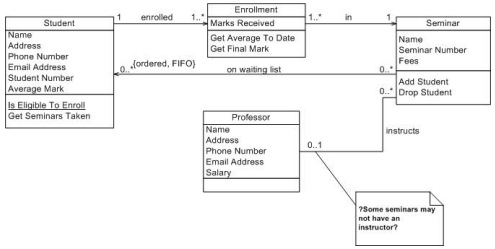


Component Diagram

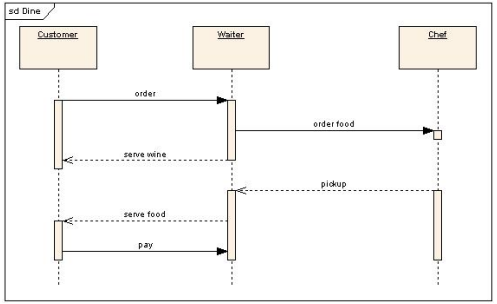


Use in Projects

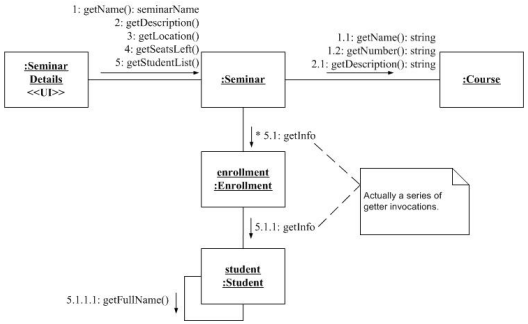
Class Diagram



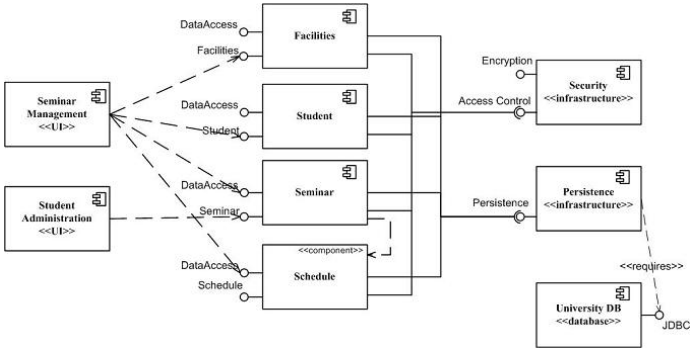
Sequence Diagram



Communication Diagram



Component Diagram



User Stories

- **User stories** are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template: (https://www.mountaingoatsoftware.com/agile/user-stories)
  - *As a <type of user>, I want <some goal> so that <some reason>.*
- Example user stories (http://www.agilemodeling.com/artifacts/userStory.htm)
  - Students can purchase monthly parking passes online.
  - Parking passes can be paid via credit cards.
  - Parking passes can be paid via PayPal.
  - Professors can input student marks.
  - Students can obtain their current seminar schedule.
  - Students can order official transcripts.
  - Students can only enroll in seminars for which they have prerequisites.
  - Transcripts will be available online via a standard browser

Agile Development and User Stories

2.3 User Stories

You can either embed a document that contains the user stories; provide a link to an online system where the stories can be reviewed. If neither of those are viable, you can include them here in a similar format to the table below.

Title	**<Insert title here>**
Description	<Describe use-case objectives here>
Actors/Roles Involved	
Pre-conditions	
Flow of Events	
Post-conditions	
Assumptions	
Limitations	

Documenting API Example

(We will use Swagger and Swagger Editor)

URI	https://mysite.com:3911/api/members/{id}
HTTP verb	**PUT**
Parameters	**id** : Card number of the member.
Body	**name** : Name of the member. **email** : Email adress of the member. **langage** : Langage used by member (Fr_CA ou En_US)

Sample body	{
	"name":"Mario Cardinal",
	"email":“mcardinal@mariocardinal.com",
	"language":"fr_CA"
	}

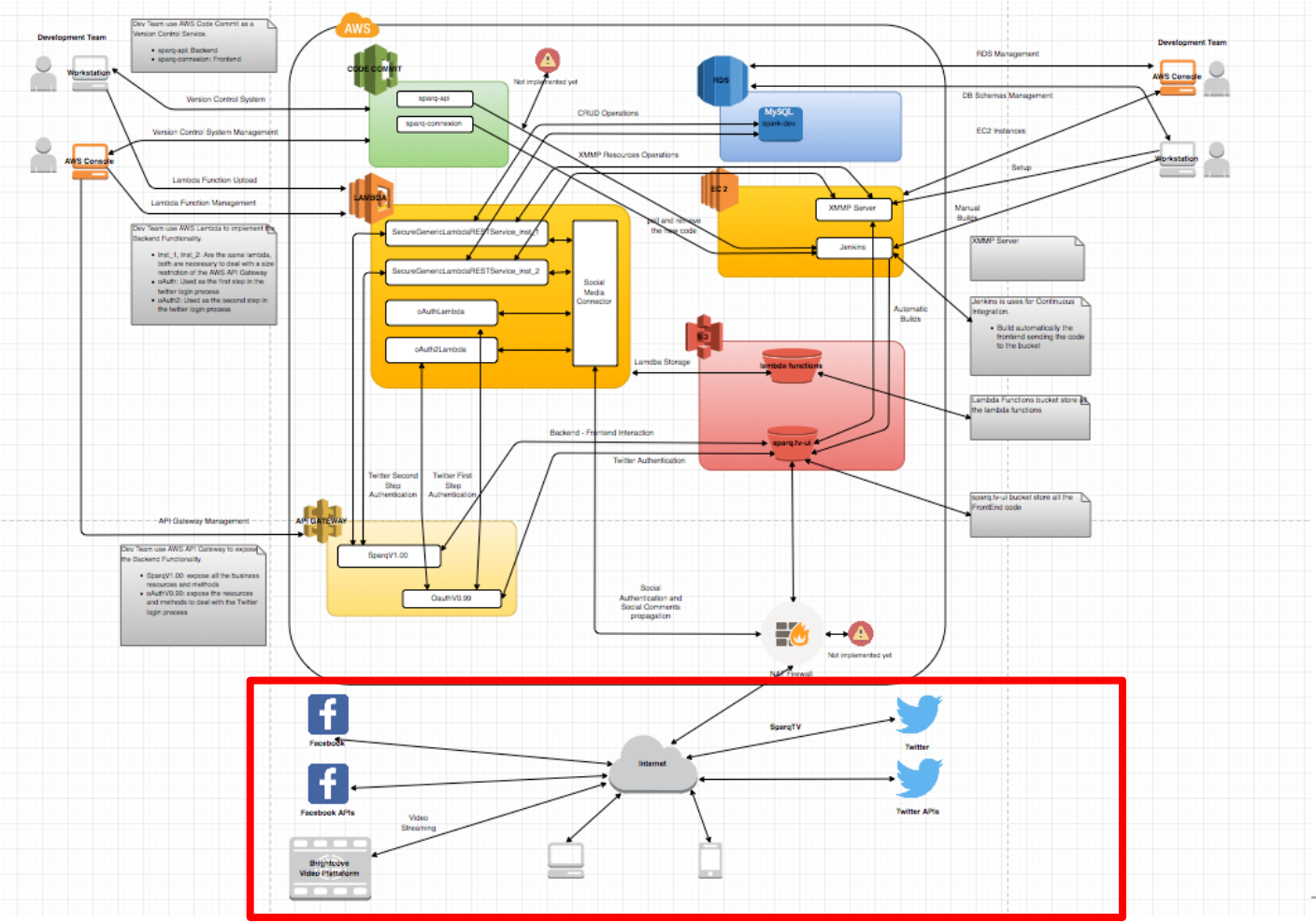
Success Response	Status Code: 204 No Content
Error Response	**Status Code**: 400 Bad Request, **Body**: {"Error Code":"..."} **Status Code**: 401 Unauthenticated, see **WWW-Authenticate** value in header **Status Code**: 403 Forbidden **Status Code**: 404 Not Found **Status Code**: 429 Too Many Requests, see **Retry-After** value in header **Status Code**: 500 Internal Server Error **Status Code**: 503 Service Unavailable
Error Code	**10**: Inactive member **20**: Denied access member **110**: Database issues, Retry later

Top-Level Design Specification

- Comments
  - I have presented a very *systematic* version of TLDS content.
  - All development projects should do *some flavor* of this approach, ideally simple and living/dynamic.
  - Just want to give you a feel for the types of things you can include in your TLDS, but you only need a really simple version.
  - Use common sense.
- For your TLDS
  - You do not need to use UML, but should use some of the concepts.
  - Some useful diagrams
    - Component diagram.
    - “Class” diagram documenting the logical datamodels/ resource models for components.
    - Some sequence diagrams showing end-to-end flows and Lambda functions in flows.
- Define and document
  - A couple of user stories.
  - Your APIs.

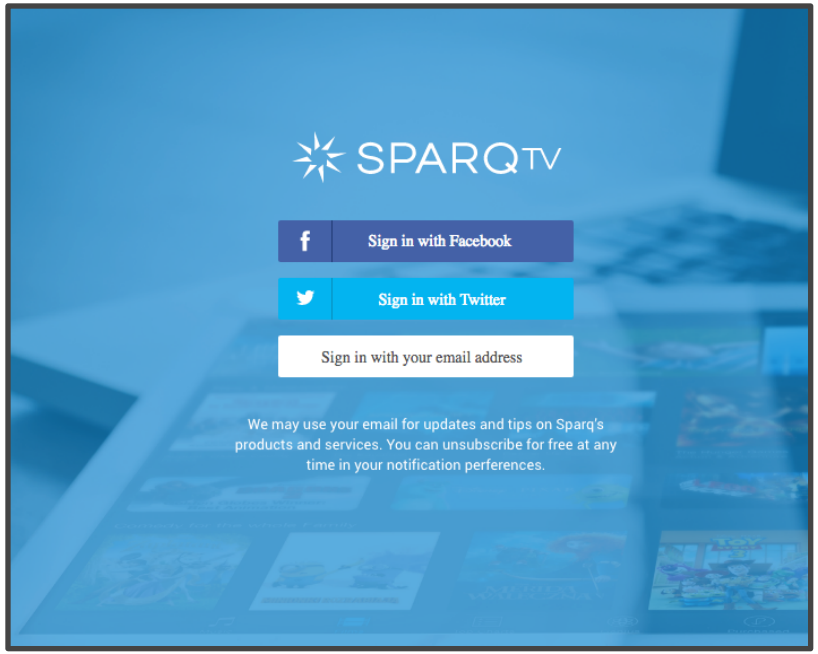
New Topics

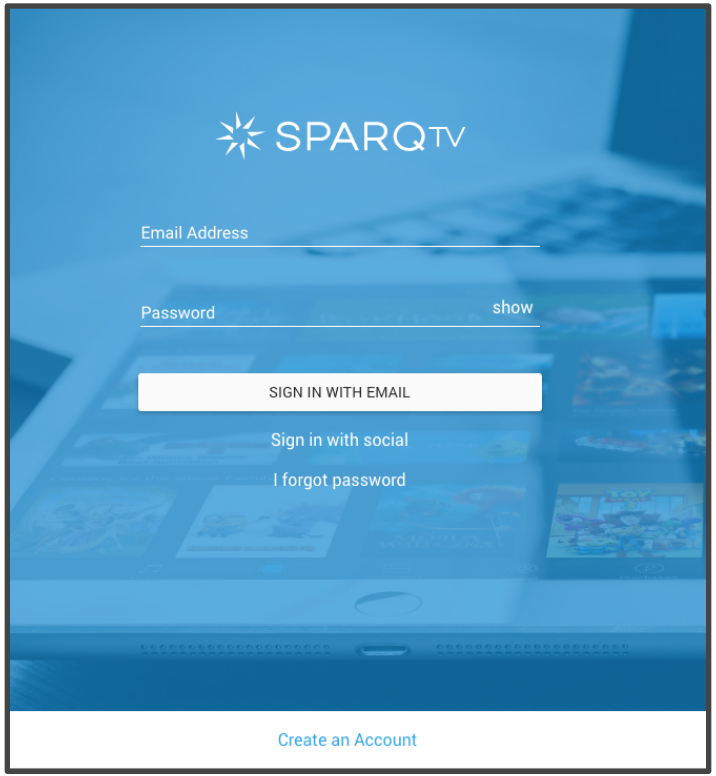
Social Media Integration (Motivation)



Register and Logon

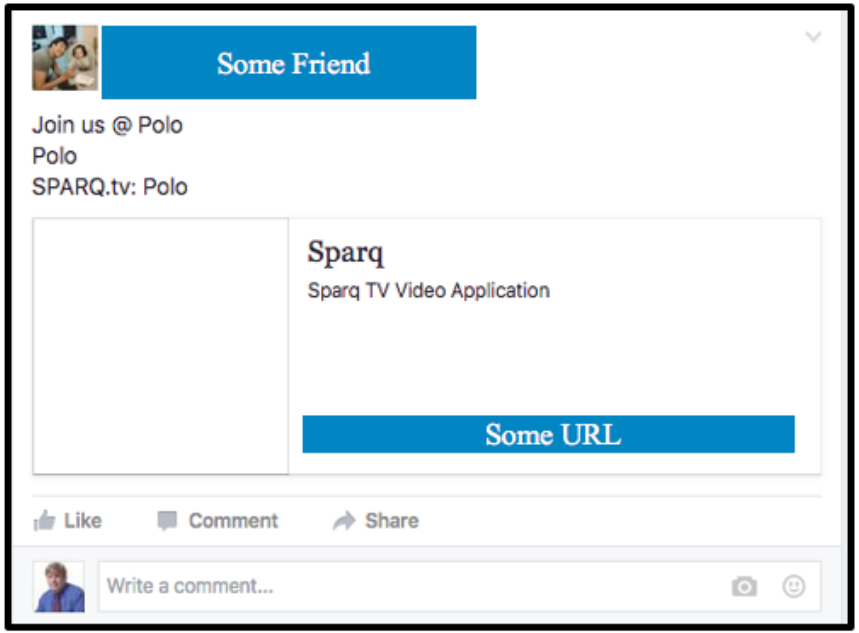
Simplifies and expedites logon and registering for your product.

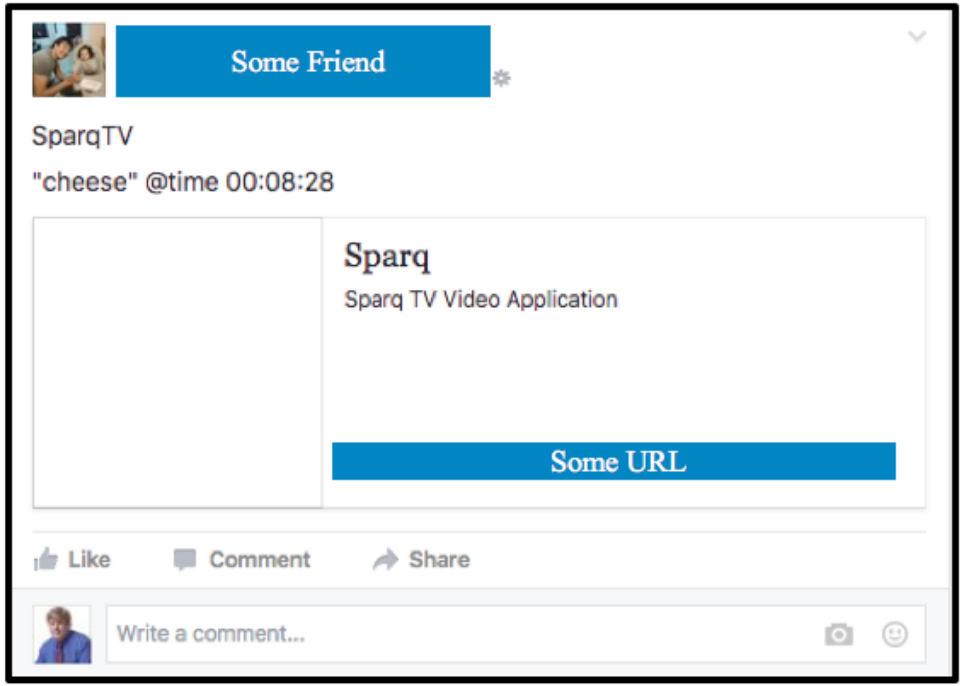




Federate Social Experience

Expands reach of your product to attract new users.
Simplifies your customers "sharing" what they are doing (no "cut and paste")





Graph API

Provides user insights to enable customized, optimized interactions and support.

Graph API Explorer

Application: [?] Graph API Explorer

Access Token: Access Token

Get Token

GET → /v2.7 / me?fields=id,name,email,devices,about,bio,education

Submit

Learn more about the Graph API syntax

Node: me

- id
- name
- email
- devices
- about
- bio
- education

+ Search for a field


```
{
  "id": "10153132236678693",
  "name": "Donald Ferguson",
  "email": "donff2@aol.com",
  "devices": [
    {
      "hardware": "iPhone",
      "os": "iOS"
    },
    {
      "hardware": "iPad",
      "os": "iOS"
    },
    {
      "os": "Android"
    }
  ],
  "education": [
    {
      "school": {
        "id": "110154005680773",
        "name": "Cardinal Spellman High School"
      },
      "type": "High School",
      "id": "10153471493923693"
    },
    {
      "school": {
        "id": "109136799104262",
        "name": "Cardinal Spellman High School"
      },
      "type": "High School",
      "year": {
        "id": "138792749476094",
        "name": "1978"
      },
```

Graph API

Simplifies inviting and finding friends to use/ already using the product.

Graph API Explorer

Application: [?] Graph API Explorer

Access Token: EAACEdEose0cBALvgi1GiZBJPdWvoaPS9ZCa3EyCpbpA46ZBeiUucBylg0fYDudUfNk1U0ZAQM1FxnZASMhkZB5MJcLxjt6mpZCi37Hpkv

Get Token

GET → /v2.7 / me/friends

Submit

Learn more about the Graph API syntax

Edge: me/friends

+ Search for a field

1 Debug Message (Show)


```
  },
  {
    "name": [redacted]
    "id": [redacted]
  },
  {
    "name": [redacted]
    "id": [redacted]
  },
  {
    "name": [redacted]
    "id": [redacted]
  },
  {
    "name": [redacted]",
    "id": [redacted]
  },
  {
    "name": [redacted]
    "id": [redacted]
  },
  {
    "name": [redacted]
    "id": [redacted]
  }
},
"paging": {
  "cursors": {
    "before": "QVFIUmtVRktsaXJRU3BpQ2tCVTRQS3BJVTY4T01paWpQdUp4ajZApRkdKdEZAIY01UOUSOb2taVTJYOWgxMjlKbFN1
    "after": "QVFIUjBXckJzcW00ZAm1iUlZAtVUU2LXpyanpGcC1aMGY5Nkgwa3NGUzVNTEN4LWxHNk1JdFNtcE1PVWZATQnBVcHdE
  }
},
"summary": {
  "total_count": 633
}
```

Graph API Permissions

Graph API Explorer

Application: [?] Graph API Explorer

Access Token: EAACEd ... Hpkv Get Token

GET → /v2.7 / m Submit

Edge: me/friends

+ Search for a field

...re about the Graph API syntax

Select Permissions

v2.7

User Data Permissions

- email
- publish_actions
- user_about_me
- user_birthday
- user_education_history
- user_friends
- user_games_activity
- user_hometown
- user_likes
- user_location
- user_photos
- user_posts
- user_relationship_details
- user_relationships
- user_religion_politics
- user_status
- user_tagged_places
- user_videos
- user_website
- user_work_history

Events, Groups & Pages

- ads_management
- ads_read
- business_management
- manage_pages
- pages_manage_cta
- publish_pages
- pages_messaging
- pages_messaging_payments
- pages_messaging_phone_number
- pages_messaging_subscriptions
- pages_show_list
- read_page_mailboxes
- rsvp_event
- user_events
- user_managed_groups
- pages_manage_instant_articles

Open Graph Actions

- user_actions.books
- user_actions.fitness
- user_actions.music
- user_actions.news
- user_actions.video

Other

- read_audience_network_insights
- read_custom_friendlists
- read_insights

Public profile included by default

Get Access Token | Clear | Cancel

Define an App

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

SparqTV

APP ID: 1694007880812098 | View Analytics | Tools & Support | Docs

- Dashboard
- Settings
- Roles
- Alerts
- App Review

PRODUCTS

- Facebook Login
- + Add Product

Dashboard

SparqTV

This app is public and available to all users

API Version: v2.5

App ID: App ID

App Secret: ••••••••• Show

Get Started with the Facebook SDK

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

Choose Platform

Facebook Analytics for Apps

Set up Analytics

Analytics for Apps 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 Login

Active Login Users | Trend

Monthly Acti... | Weekly Activ... | Daily Active ...

Pagination

Pagination/Iteration

- Consider the web method GET on
  - /customers?lastname=Smith
  - /tvshows?year=2016
- The result set could be thousands or millions of "records."
- The application should not/cannot return all records at once
  - Network/connection timeouts.
  - Overwhelm the client runtime with data and data processing.
  - Give the user a chance to realize
    - That they asked the "wrong question"
    - And submit a refined query.
  - etc.

Traditional Solution is Iterator/Cursor

GET /customers?name=bob&limit=10

… …

… …

GET /customers?”get next 10”

```
-- We must be in a transaction
BEGIN;
-- Open a cursor for a query
DECLARE medley_cur CURSOR FOR SELECT * FROM medley;
-- Retrieve ten rows
FETCH 10 FROM medley_cur;
-- ...
-- Retrieve ten more from where we left off
FETCH 10 FROM medley_cur;
-- All done
COMMIT;
```

Traditional Solution is Iterator/Cursor

GET /customers?name=bob&limit=10

GET /customers?"get next 10"

```
-- We must
BEGIN;
-- Open a c
DECLARE medley_cur  R SELECT * FROM medley;
-- Retrieve ten rows
FETCH 10 FROM medley_cur;
-- ...
-- Retrieve ten more from where we left off
FETCH 10 FROM medley_cur;
-- All done
COMMIT;
```

From "where we left off" is "state."

- From "where we left off …"
  - Requires client specific state held between requests, which violates
  - The REST tenet of "stateless."
- The benefits of stateless are
  - Not holding server resources (cursors, memory, ...) for millions of clients for several seconds/client.
  - Exploits the computing and storage of millions of client "computers."
  - And Lambda functions are STATELESS by definition. **So, you have to do it.**

Examples

- SQL
  - URLs
    - GET /customers?name=Bob&limit=10.
    - GET /customers?name=Bob&limit=10&offset=10
  - SQL
    - SELECT * FROM customers WHERE name='Bob' LIMIT 10;
    - SELECT * FROM customers WHERE name='Bob' LIMIT 10 OFFSET 10;
- DynamoDB –
  - SELECT * → AttributesToGet : […]
  - WHERE → KeyConditions: [...]
  - LIMIT → Limit:
  - OFFSET → ExclusiveStartKey ==
    - The **LastEvaluatedKey** of the last record in the **returned** previous query result set.
    - Start the query at the record after the ExclusiveStartKey

http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Query.html

Example

Next Request

```
var params = {
    TableName: 'table_name',
    IndexName: 'index_name', // optional (if querying an index)
    KeyConditions: { // indexed attributes to query
                     // must include the hash key value of the table or index
                     // with 'EQ' operator
        attribute_name: {
            ComparisonOperator: 'EQ', // (EQ | NE | IN | LE | LT | GE | GT | BETWEEN |
                                      //  NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WIT
H)
            AttributeValueList: [ { S: 'STRING_VALUE' }, ],
        },
        // more key conditions ...
    },
    ScanIndexForward: true, // optional (true | false) defines direction of Query in the index
    Limit: 0, // optional (limit the number of items to evaluate)
    ConsistentRead: false, // optional (true | false)
    Select: 'ALL_ATTRIBUTES', // optional (ALL_ATTRIBUTES | ALL_PROJECTED_ATTRIBUTES |
                              //           SPECIFIC_ATTRIBUTES | COUNT)
    AttributesToGet: [ // optional (list of specific attribute names to return)
        'attribute_name',
        // ... more attributes ...
    ],
    ExclusiveStartKey: { // optional (for pagination, returned by prior calls as LastEvaluatedKe
y)
        attribute_name: { S: 'STRING_VALUE' },
        // anotherKey: ...

    },
    ReturnConsumedCapacity: 'NONE', // optional (NONE | TOTAL | INDEXES)
};
dynamodb.query(params, function(err, data) {
    if (err) console.log(err); // an error occurred
    else console.log(data); // successful response
});
```

Previous Response

```
            "SS": [ "string" ]
        }
    }
],
"LastEvaluatedKey": {
    "string" : {
        "B": blob,
        "BOOL": boolean,
        "BS": [ blob ],
        "L": [
            "AttributeValue"
        ],
        "M": {
            "string" : "AttributeValue"
        },
        "N": "string",
        "NS": [ "string" ],
        "NULL": boolean,
        "S": "string",
        "SS": [ "string" ]
    }
},
"ScannedCount": number
}
```

Pagination –

The response to the client has info about “where you left off” and client returns it.

{“data”:
[{“user_id”:”42”, “name”:”Bob”,
“links”:[{“rel”:”self”, “href”:”http://api.example.com/users/42”}]},
{“user_id”:”22”, “name”:”Frank”,
“links”: [{“rel”:”self”, “href”:”http://api.example.com/users/22”}]},
{“user_id”:”125”, “name”: “Sally”,
“links”:[{“rel”:”self”, “href”:”http://api.example.com/users/125”}]}],

“links”:
[{“rel”:“first”, “href”:”http://api.example.com/users?offset=0&limit=3”},
{“rel”:“last”, “href”:”http://api.example.com/users?offset=55&limit=3”},
{“rel”:“previous”, “href”:”http://api.example.com/users?offset=3&limit=3”},
{“rel”:”next”, “href”:”http://api.example.com/users?offset=9&limit=3”}]}

Pagination –

The response to the client has info a

The response contains

- First, Last, Previous, Next
- With hrefs and URLs that are “opaque” strings.

So, the client is isolated

- From the specific DB in use.
- From forming database specific strings.

{“data”:
[{“user_id”:”42”, “name”:”Bob”,
“links”:[{“rel”:”self”, “href”:”http://a
{“user_id”:”22”, “name”:”Frank”,
“links”: [{“rel”:”self”, “href”:”http://api.exa ers/22 }]},
{“user_id”:”125”, “name”: “Sally”,
“links”:[{“rel”:”self”, “href”:”http://api.ex ple.com/users/125”}]}],

“links”:
[{“rel”:“first”, “href”:”http://api.example.com/users?offset=0&limit=3”},
{“rel”:“last”, “href”:”http://api.example.com/users?offset=55&limit=3”},
{“rel”:“previous”, “href”:”http://api.example.com/users?offset=3&limit=3”},
{“rel”:”next”, “href”:”http://api.example.com/users?offset=9&limit=3”}]}

OAuth2

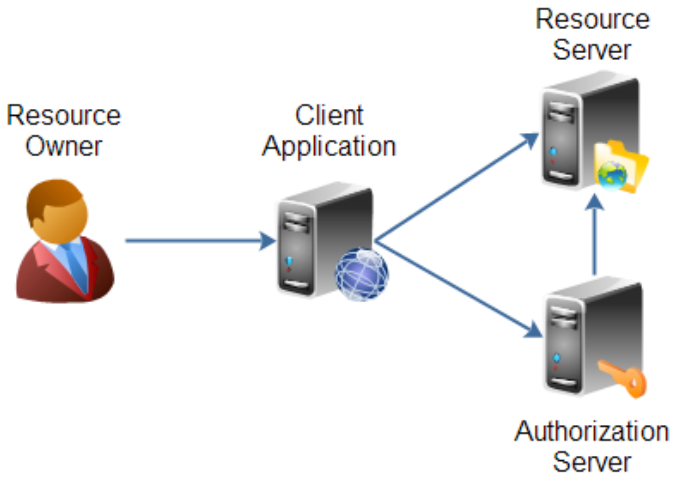
Overview (http://tutorials.jenkov.com/oauth2/index.html)

- 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:

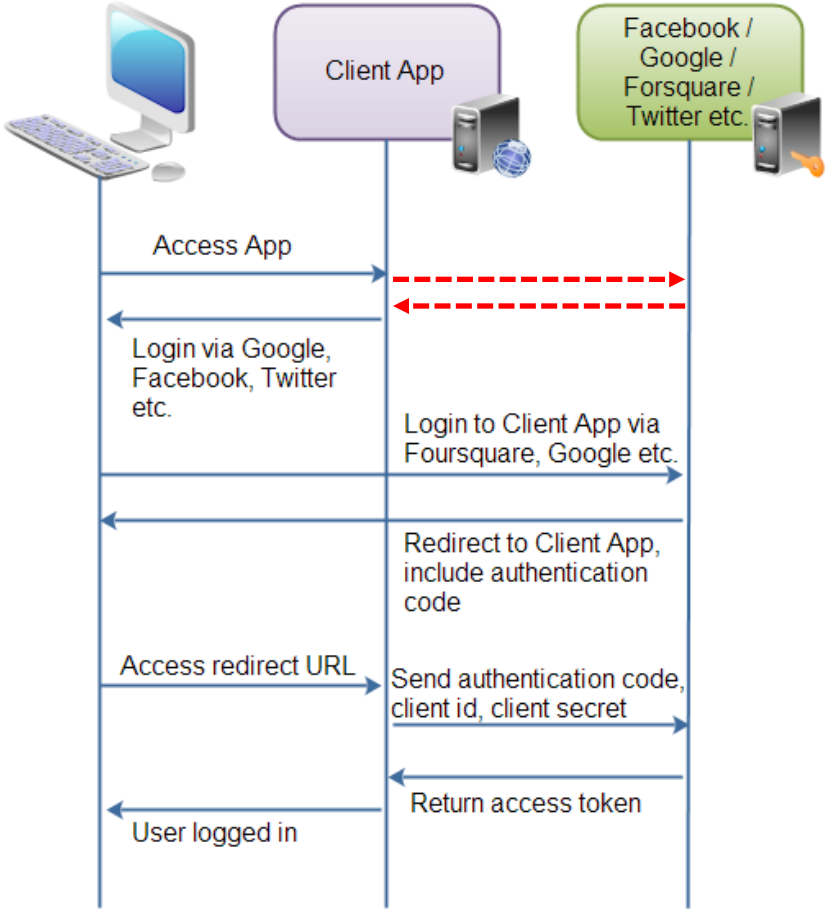
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

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.



Configuring Your Application at XXX

Details | Settings | Keys and Access Tokens | Permissions

Application Details

Name *

SparqTVTest

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

SparqTV Test Application

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your Product Site

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Redirect from Twitter

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Privacy Policy URL

The URL for your application or service's privacy policy. The URL will be shared with users authorizing this application.

Terms of Service URL

The URL for your application or service's terms of service. The URL will be shared with users authorizing this application.

☐ Enable Callback Locking (It is recommended to enable callback locking to ensure apps cannot overwrite the callback url)

☑ Allow this application to be used to Sign in with Twitter

Twitter Example – Step 1

Resources

Actions

/

OPTIONS

/auth

GET

OPTIONS

/auth2

GET

OPTIONS

/instagramoauth

GET

/auth - GET - Method Execution

TEST

Client

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-east-1:541461934968:bcmzgh0pje/*/GET/aut

Integration Request

Type: LAMBDA

Region: us-east-1

Lambda OAuthLambda

Method Response

HTTP Status: 302

Integration Response

HTTP status pattern: -

Output passthrough: Yes

SPARQTV

Sign in with Facebook

Sign in with Twitter

Sign in with your email address

We may use your email for updates and tips on Sparq's products and services. You can unsubscribe for free at any time in your notification perferences.

Twitter Example - Step 1 (Cont)

← Method Execution /auth - GET - Integration Response

First, declare response types using Method Response. Then, map the possible responses from the backend to this method's response types.

	Lambda Error Regex	Method response status	Output model	Default mapping	
▸	200	200		No	⊗
▾	-	302		Yes	⊗

Map the output from your Lambda function to the headers and output model of the 302 method response.

Lambda Error Regex default

Method response status 302

Cancel Save

▾ Header Mappings

Response header	Mapping value
Location	integration.response.body.location

▾ Body Mapping Templates

Content-Type	
application/json	⊖

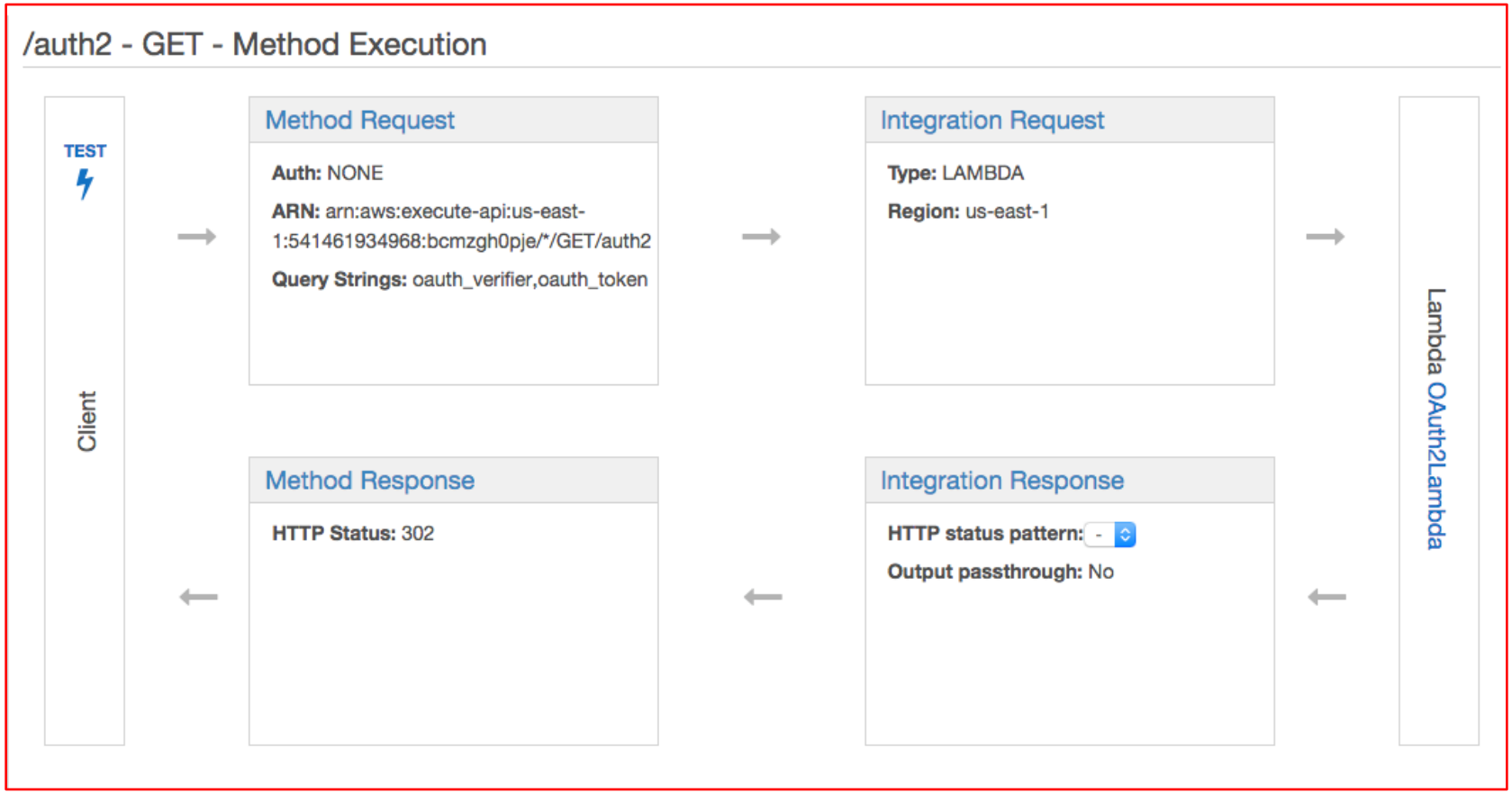
⊕ Add mapping template

application/json

Generate template:

```
1  #set($inputRoot = $input.path('$'))
2  {
3    $input.json('$.context.headers.redirect')
4  }
```

Step 2 – Same Basic Pattern



```java
public class TwitterConnector {

    private static final String S3_BUCKET =
    // Consumer Key (API Key)
    private static final String TWITTER_CONSUMER_KEY =
    // Consumer Secret (API Secret)
    private static final String TWITTER_CONSUMER_SECRET =
    // Info: Callback URL = S3_BUCKET + "/dev/auth2"

    /*
     * Interactions with Twitter require a RequestToken. There are two forms: 1.
     * Just information identifying the application. 2. Also including user
     * authentication information. This class represents the simple version.
     */
    public static class SimpleRequestToken {
        String token;
        String secret;

        public SimpleRequestToken(String t, String s) {
            this.token = t;
            this.secret = s;
        }
    }

    /*
     * This is the persistent OAuth2 credential that we get back from completing
     * the successful, second step in the Twitter login process. We save this
     * for subsequent interactions.
     *
     * TODO I am not sure if we should embed the class inside the connector.
     *
     */
    public class TwitterAuth {
        public String oauth_token;
        public String oauth_verifier;
    }
```

Base for our Web Site

Key (ID for App)

Password

```java
public class TwitterConnector {

    private static final String S3_BUCKET =
    // Consumer Key (API Key)
    private static final String TWITTER_CONSUMER_KEY =
    // Consumer Secret (API Secret)
    private static final String TWITTER_CONSUMER_SECRET =
    // Info: Callback URL = S3_BUCKET + "/dev/auth2"

    /*
     * Interactions with Twitter require a RequestToken. There are two forms: 1.
     * Just information identifying the application. 2. Also including user
     * authentication information. This class represents the simple
     */
    public static class SimpleRequestToken {
        String token;
        String secret;

        public SimpleRequestToken(String t, String s) {
            this.token = t;
            this.secret = s;
        }
    }

    /*
     * This is the persistent OAuth2 credential that we get back
     * the successful, second step in the Twitter login process.
     * for subsequent interactions.
     *
     * TODO I am not sure if we should embed the class inside the connector.
     *
     */
    public class TwitterAuth {
        public String oauth_token;
        public String oauth_verifier;
    }
```

Base for our Web Site

Key (ID for App)

Password

Hard coding these as strings is in the code is a bad idea.
But, I was lazy. Lazy is how security breaches occur.

```java
private static boolean initializeConnector(String access_id, String access_token) {

    // Turns out that every Lambda function request does not get a fresh
    // JVM. So, consecutive requests were reusing the wrong request token.
    // so for now we just reinitialize every time.
    // TODO Fix this later.
    //
    // if (TwitterConnector.connectorInitialized == false) {

    logger.debug("Initializing the Twitter Connector");

    try {
        TwitterConnector.setup();
        TwitterConnector.connectorInitialized = true;

        logger.debug("Classes initialized.");

        // TODO We need to move this configuration information to somewhere
        // else.
        //
        twitterProperties = new Properties();
        twitterProperties.setProperty("oauth.consumerKey", TWITTER_CONSUMER_KEY);
        twitterProperties.setProperty("oauth.consumerSecret", TWITTER_CONSUMER_SECRET);
        if (access_id != null) {
            twitterProperties.setProperty("oauth.accessToken", access_id);
        }
        if (access_token != null) {
            twitterProperties.setProperty("oauth.accessTokenSecret", access_token);
        }

        logger.debug("Initialized properties.");

        PropertyConfiguration pc = new PropertyConfiguration(twitterProperties);
        twitter = new TwitterFactory(pc).getInstance();
        logger.debug("Initialized Twitter4J.");

    } catch (Exception e) {
        logger.warn("Exception while initializing Twitter Connector, e = " + e);
        connectorInitialized = false;
        twitterProperties = null;
        twitter = null;
    }

    logger.debug("Successfully initialized Twitter Connector.");
    // }

    return connectorInitialized;
}
```

```java
public static HTTPRedirect twitterRedirect1() {

    HTTPRedirect result = null;
    UUID statusUUID = null;

    try {
        logger.debug("Step 1 in Twitter OAuth2 login.");

        TwitterConnector.initializeConnector();

        logger.debug("Attempting to get Twitter request token.");
        RequestToken requestToken = twitter.getOAuthRequestToken();

        // TODO Writing this info to a log file is probably a security
        // exposure.
        logger.debug("After getOAuthRequestToken = " + "token=" + requestToken.getToken() + ", " + "secret="
                + requestToken.getToken());

        // Save the request token, which we will need for the second phase.
        SimpleRequestToken st = new TwitterConnector.SimpleRequestToken(requestToken.getToken(),
                requestToken.getTokenSecret());
        statusUUID = saveState(st);

        if (statusUUID != null) {
            String authorizationUrl = requestToken.getAuthenticationURL();
            result = new HTTPRedirect(authorizationUrl, HTTPRedirect.success);
        } else {
            logger.warn("Could not save SimpleRequestToken in SessionStorage");
        }
    } catch (Exception e) {
        logger.warn("twitterRedirect1 failed with exception", e);
    }

    if (result == null) {
        logger.warn("Could not get redirect URL. Returning error URL.");
        result = new HTTPRedirect(failureUrl, HTTPRedirect.failure);
    }

    return result;

}
```

```java
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.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.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.toPrettyJSON(authorizationToken));
            s = JSONUtil.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;
    }

    HTTPRedirect r;
    if (u != null) {
        r = new HTTPRedirect("http://" + S3_BUCKET + "/#/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.toPrettyJSON(r));
    return r;

}
```

OAuth Process Allows Us to

- Get user profile info
  - Name
  - Email
  - Location
  - … …
- Get user Social network
  - User follows …
  - Follows user…
- Tweet on user's behalf when something happens in my application.

```java
public static int getRetweets(String access_id, String access_token, String post_id) {
    TwitterConnector.initializeConnector(access_id, access_token);
    Status result = null;
    int retweets = 0;
    try {
        long id = Long.parseLong(post_id);
        result = twitter.showStatus(id);
        retweets = result.getRetweetCount();
    } catch (Exception e) {
        e.printStackTrace();
    } catch (OutOfMemoryError e) {
        e.printStackTrace();
    }
    return retweets;
}
```

```java
public static List<Object> getFollowers(String access_id, String access_token) {
    TwitterConnector.initializeConnector(access_id, access_token);
    ArrayList<Object> followers = new ArrayList<>();
    PagableResponseList<User> all = null;
    long userId = Long.parseLong(access_id.substring(0, access_id.indexOf('-')));
    try {
        all = twitter.getFollowersList(userId, -1);
        followers.addAll(all);
        return followers;
    } catch (Exception e) {
        e.printStackTrace();
    } catch (OutOfMemoryError e) {
        e.printStackTrace();
    }
    return followers;
}
```