Interagency Wildland Fire Incident Response Portal

Access & Authentication Service

FAMAuth Developer Guide

Release 2.9

Rev #022

Table of Contents

Overview 5

FAMAuth Portal with eAuthentication, Login.gov and FAMAuth Admin 6

Get Started: Registering a Client Application 6

USDA eAuthentication Integration Overview 9

GSA Login.gov Integration Overview 10

Implementation: Login to the FAMAuth Portal 11

Part 1 - Authenticating with Service Provider (eAuthentication or Login.gov) 11

Part 2 – Client Web Application Integration – 6 Steps 12

STEP 1 – Invoke Client Application 12

STEP 2 – Browser Redirect to FAMAuth Authorization Server 12

STEP 3 – Authorization code returned from FAMAuth 13

STEP 4 – Authorization Code Posting for Acquiring Access Token 13

STEP 5 – The FAMAuth Response to Client Posting is an Access Token 14

STEP 6 - Issuing a GET with the Access Token for the authenticated user profile retrieval 16

STEP 7 – Browser Redirect to FAMAuth Admin (if using FAMAuth Admin) and Unauthorized User 18

Appendix A: Web App Examples Integrating with the Authentication Service 19

A. Java Client App Sample Code, Authenticating Users with FAMAuth 19

B. .NET Client App Sample Code, Authenticating Users with FAMAuth 33

C. Spring Boot Client App Sample Code, Authenticating Users with FAMAuth 44

Appendix B: User Experience with FAMAuth Admin 48

Frequently Asked Questions (FAQs) 49

Revision History

Rev #	Date	Author	Description
000	4/30/2021	EAD Team	Initial Revision
001	7/8/2021	Julie Olson	Incorporated changes from FAMAuth 2.0 release: Add instructions for applications using iNAP for authorization.
002	7/22/2021	Mike Roadifer	Updated FAMIT point of contact to Ryan Hunt.
003	9/14/2021	Julie Olson	Fixed iwfirp URLs from Dev to QA. Updated Login page snapshot.
004	10/2/2021	Dan Gropp	FAMAuth 2.1 Updated /me service responses, /token service screens shots, add information about the /meta service that can be used to auto configure an OAUTH2 application and added a FAQ to the appendix on how to book mark an application.
005	11/5/2021	Julie Olson	Updated the FAMAuth screen shots for 2.1 release.
006	11/22/2021	Mike Roadifer	Updated to consistently reference as FAMAuth. Corrected exchange between FAMAuth and client application as simply OAuth2. Added interaction with iNAP.
007	2/1/22	Julie Olson	Added FAMAuth/iNAP user experience flowchart in appendix B.
008	5/2/22	Julie Olson	Updated the FAMAuth screen shots for 2.2 release.
009	5/31/22	Dan Gropp	Added user_authenticated_by for Neil @AFF to the /me service in the 2.3 release.
010	8/16/22	Julie Olson	Updated the FAMAuth screen shots for 2.3 release.
011	10/27/22	Michelle Apicella	Updated FAMAuth screen shots for 2.4 release.
012	10/27/22	Dan Gropp	Added description of refresh token in FAQ.
013	10/28/22	Julie Olson	Updated FAMAuth/iNAP flow chart for iNAP 2.7 release.
014	12/13/22	Dan Gropp	Updated iNAP user authorization returned output for iNAP 2.8 release
015	11/6/23	Dan Gropp	Added how a client application can call FAMAuth's logout service.
016	12/7/23	Mike Roadifer	Updated for migration to famauth-qa.wildfire.gov.
017	12/7/23	Julie Olson	Updated screen shots for 2.8 release.
018	1/29/24	Dan Gropp	Added comments to describe appending a client application redirection URL parameter to the returned

			napRedirectURL
019	1/31/24	Julie Olson	Added an explanation of the benefit of appending the CLIENT_URL to the napRedirectURL. (see B rowser Redirect to iNAP if using iNAP and Unauthorized User section)
020	4/17/24	Julie Olson	Updated the User Experience with FAMAuth/iNAP flow chart for iNAP 2.12 changes.
021	5/9/24	Julie Olson	Updated with FAMAuth 2.9 changes.
022	6/3/23	Julie Olson	Updated with INAP 2.13 changes. (Rebrand)

Overview

The FAMAuth (Fire and Aviation Management Authentication and Authorization) Portal shown in the figure below, authenticates government application users via USDA eAuthentication and non-government users via Login.gov. Also, SAML federation bridging with eAuthentication may also be used to authenticate other agencies such as DOI.

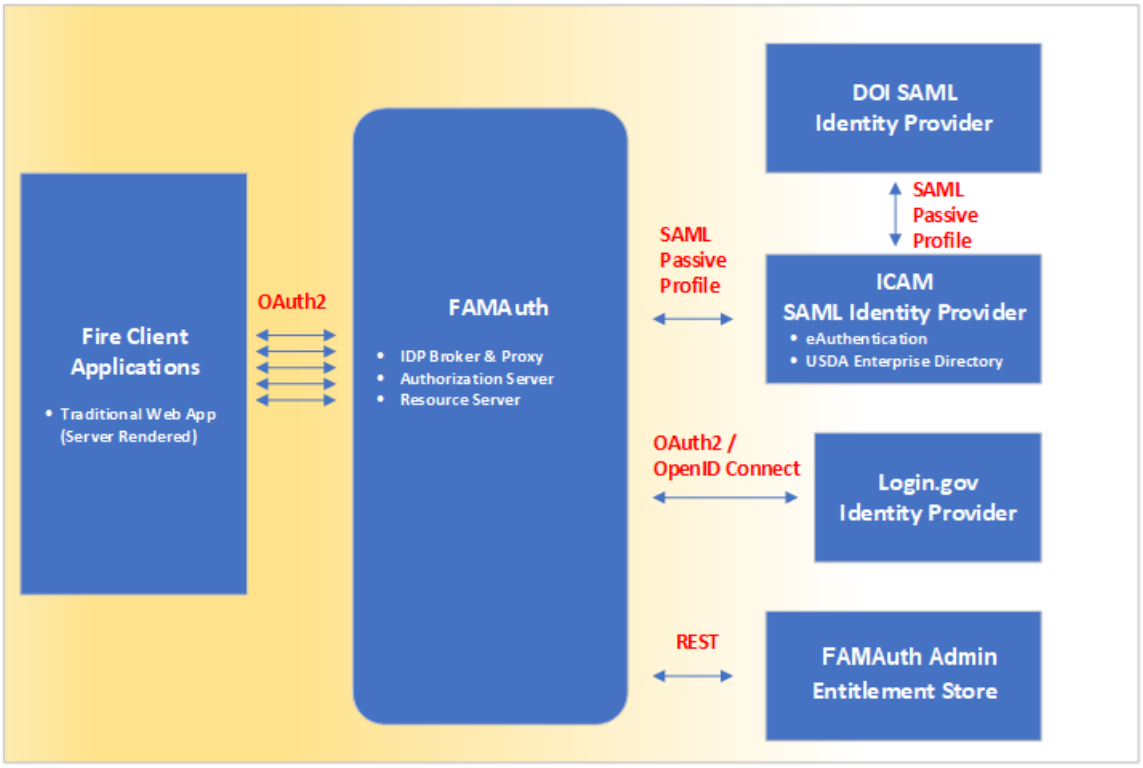
FAMAuth is used in this document as a reference to this application. FAMAuth is also sometimes referred to as IWFIRP (Interagency Wildland Fire Incident Response Portal).

FAMAuth currently supports the authentication of web server applications only.

Client applications that integrate with FAMAuth use the OAuth 2.0 protocol to validate that a user has authenticated in FAMAuth (either via eAuthentication or Login.gov). OAuth 2.0 is the industry standard protocol for delegated authorization in the modern web, which is supported by leading companies like Google, Facebook and Twitter. Client applications will first authorize with FAMAuth to ensure that the user is authenticated, and will then be provided an access token that can be used to return information about the user from FAMAuth. This guide is intended for developers so they can understand the workflow to be able to integrate a client application and work with the FAMAuth team to onboard a new client application to leverage the shared authentication services.

An optional component to consider when integrating with FAMAuth is FAMAuth Admin (formerly known as iNAP), which holds entitlements for users of Wildland Fire applications. Users can request access to applications and roles, that are then approved or denied by application representatives. The information captured is available to client applications that are integrated with FAMAuth. See Appendix B: User Experience with FAMAuth Admin for additional information.

FAMAuth Portal with eAuthentication, Login.gov and FAMAuth Admin



Get Started: Registering a Client Application

To have your application added to the FAMAuth portal and leverage the authentication service, please send a request to the FAMIT point of contact: Ryan Hunt – ryan.hunt2@usda.gov. Include the following information in the request:

- Short Name – Typically an acronym or abbreviation
- Long Name – Full name of the application
- Description – Brief description of the application
- Application URL – The URL for launching the application
- Agency – An agency abbreviation, such as FS
- Image (optional) – A thumbnail icon for the application
- Use FAMAuth Admin for application access and roles? – Yes or No

For example:

- Short Name: OIS QCTEST
- Long Name: Organization Information System
- Description: Unit identifier and organizational unit repository
- Application URL: https://ois-qt.nwcg.gov/ois/login
- Agency: FS
- Image:
- Use FAMAuth Admin for application access and roles? Yes

The FAMAuth team will use the information above to create a tile in the FAMAuth portal, and the following information will be sent to the application team:

- Client ID – example: OIS-QA1
- Client Secret – example: oissecret

- If using FAMAuth Admin
    - Application Abbreviation – example: OIS
    - Instance Abbreviation – example: TEST

Once the tile has been created, a FAM application developer may request to be set up/authorized to maintain the following properties using the FAMAuth UI.

- Short Name
- Long Name
- Description
- Application URL
- (Allowed) Redirect URLs

Below describes the flow after implementation is completed **(See the Implementation section for more details)**	
A user launches the FAMAuth portal in their browser using https://famauth-qa.wildfire.gov, selects an identity provider (Login.gov or eAuthentication) and authenticates. Upon successful authentication a session is established for the user in the FAMAuth portal and the tiles for all applications integrated with FAMAuth are displayed.	FAMAuth Team
The user clicks on the application tile rendered by FAMAuth and the Client App issues the redirection to the FAMAuth authorization service, initiating the OAuth2 exchange. https://famauth-qa.wildfire.gov/oauth/authorize?client_id=OIS-TEST&redirect_uri=https://ois-qt.nwcg.gov/ois/login&response_type=code&state=28IHK4	Client Developer
FAMAuth verifies that the user has a session in the FAMAuth portal. If not, the user is redirected to the identity provider selection page where they must authenticate (using Login.gov or eAuthentication). Upon validation of the user session (or authentication) FAMAuth returns an authorization code to the Client App. https://ois-qt.nwcg.gov/ois/login?code=FIHxR6&state=28IHK4	FAMAuth Team
The Client App uses its Basic Authorization credentials (Client_ID:Client_Secret) via a back channel, to POST and acquire an access token for a successfully authenticated user. **POST** https://famauth-qa.wildfire.gov/oauth/token HEADER: Authorization: Basic YWNZ…. Accept: application/json BODY: grant_type: authorization_code redirect_uri: https://ois-qt.nwcg.gov/ois/login code: FIHxR6	Client Developer
FAMAuth validates the Client_ID, Client_Secret, and authorization code, and returns an access token to the Client App.	FAMAuth Team

The access token is submitted via a back channel GET to retrieve the user profile. https://famauth-qa.wildfire.gov/me?access_token=<access token> Optionally include the application and instance abbreviations (from FAMAuth Admin) that user is being authorized for. https://famauth-qa.wildfire.gov/me?access_token=<access token>&application=OIS&instance=TEST	Client Developer

FAMAuth returns the user profile.
 	
The example provided here is for a user that authenticated via eAuthentication, and the client application is using FAMAuth Admin for access and roles. See Section 6 for other responses.
 	
{
"name": "{
\"usdaeauthid\":\"28552012122402152053535\",
\"usdafirstname\": \"JOHN\",
\"usdalastname\":\"SMITH\",
\"usdaemail\":\"john.smith@usda.gov\",
\"samaccountname\":\"null\",
\"approles\":\"OCIO_ENTVPN_FS-EMP^USDA_ICAMaaS-Federation_Customer_NFC^OCIO_CHD-Remedy_Record^OCIO_AgLearn_User\",
\"managername\":\"null\"}",
"email": "john.smith@usda.gov",
"sub": "28552012122402152053535",
This portion of the response is included if using FAMAuth Admin.
"authorized": "true",
"napRedirectURL": null,
"napUserProfile": "{
\"firstName\":\"John\",
\"middleName\":null,
\"lastName\":\"Smith\",
\"userName\":\"jsmith\",
\"napIdentifier\":\"1088661\",
\"eMail\":\"john.smith@usda.gov\",
\"primaryAffiliation\":\"Contractor/Vendor\",
\"seasonalFlag\":false,
\"dunsNumber\":null,
\"uei\":null,
\"inapAdminFlag\":true,
\"editProfileRedirectURL\":\"http://localhost/nap/oauth2/authorization/famauth?redirect_uri=%2FNAP%2F%3FACTION%3DfamHome%26TARGETPAGE%3DeditProfile%26EMAIL%3Dxxx%40usda.com\",
\"identityProviders\":[{
\"identityProviderID\":\"28552012122402152053535\",
\"identityProviderSystem\":\"EAUTH\",
\"identityProviderEmail\":\"john.smith@usda.gov\"}],
\"applicationRoles\":[{
\"applicationRoleName\":\"Reference Data Administrator\",
\"applicationRoleKey\":\"RDA\"}]}"
}
	FAMAuth Team

If authorizing with FAMAuth Admin and authorized=false, redirect to the URL value provided in the napRedirectURL key. This will display a page that will assist the user to do one or more of the following: 1. Request access [request a new user] or sign in with existing user 2. Edit their Profile and/or Request application access and roles 3. Accept Rules of Behavior 4. Receive instruction to restore their profile [if it has been removed] 5. Reactivate their user	Client Developer

USDA eAuthentication Integration Overview

For PIV authentication, FAMAuth is using the OAuth2 protocol to integrate with a client web application. The identity of a user is verified through the user accessing FAMAuth, acting as an authorization server. For eAuthentication, FAMAuth incorporates bridging of protocols to route the user to ICAM eAuthentication Identity Provider first, using the SAML Passive Profile protocol, where the user is authenticated. The user is returned to FAMAuth with the SAML assertion.

The user invokes the client application next that acquires an authorization code. The client application will post the authorization code to FAMAuth for an access token, used to acquire the user profile.

The user initially logs into the FAMAuth portal using the flow in yellow invoking the eAuthentication option. Next, when the user clicks on an application link, the Client Web Application OAuth2 integration flow in the blue box takes place.

For additional information refer to **Figure 1** below, detailing the FAMAuth Authentication Flow Diagram with SAML2 (eAuthication) and OAuth2 protocol chaining.

FAMAuth Authentication Service - Flow Diagram with SAML2 (eAuthentication) and OAuth2 Protocol Chaining

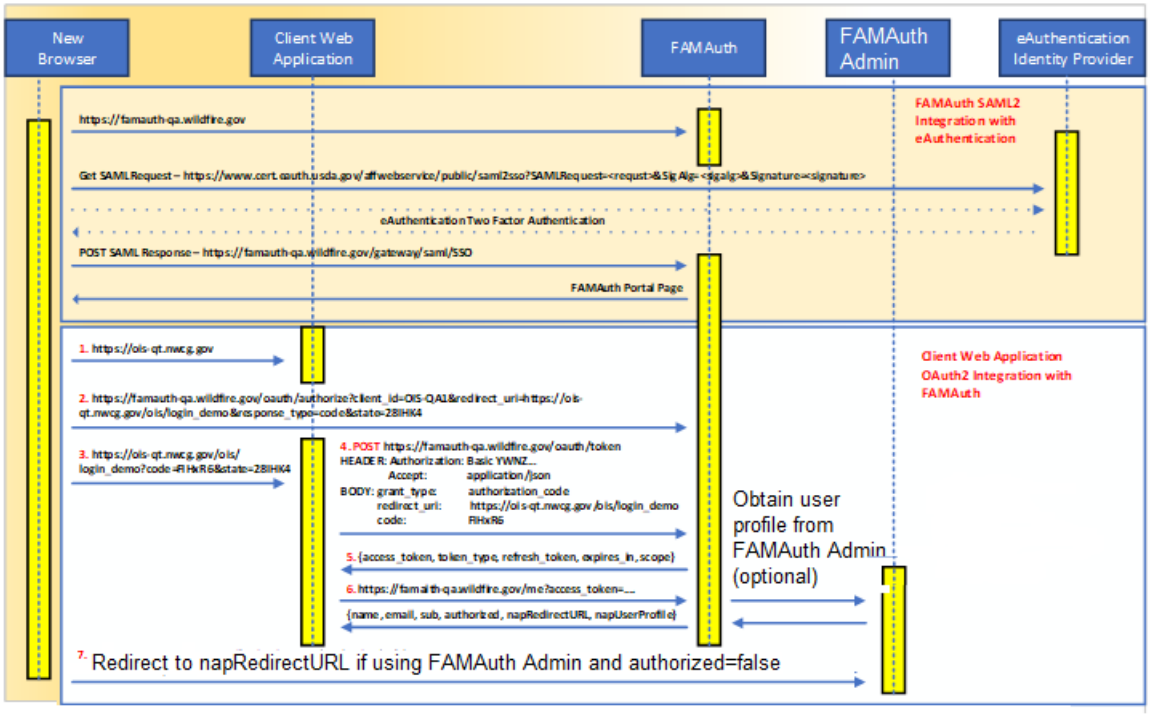


Figure 1 SAML2 to OAuth2 Protocol Chaining

*For the Client Web Application OAuth2 handshake Implementation details (steps 1-6), refer below to the Web Application Interface Description with OAuth2.

GSA Login.gov Integration Overview

For Login.gov authentication, FAMAuth is using the OAuth2 protocol to integrate with a client application (same as above). Instead of SAML Passing Profile in this case, FAMAuth will use OAuth2/OpenID connect to chain the two protocols together and authenticate the user with Login.gov. Once the user has been authenticated invoking FAMAuth first, the browser will be redirected back to the FAMAuth portal where the user can invoke an application tile to acquire an authorization code. The client application will post the authorization code to FAMAuth for an access token, used to acquire the user profile.

The user initially logs into the FAMAuth portal using the flow in yellow invoking the Login.gov option. Next, when the user clicks on an application tile (link), the Client Web Application OAuth2 integration flow in the blue box takes place. The client application will post the authorization code to FAMAuth for an access token, submitted to acquire a user profile.

For additional information refer to Figure 2 below, detailing the FAMAuth Authentication Flow Diagram with OAuth2/OpenID (Login.gov) and OAuth2 Protocol Chaining.

FAMAuth Authentication Service - Flow Diagram with OAuth2/OpenID (Login.gov) and OAuth2 Protocol Chaining

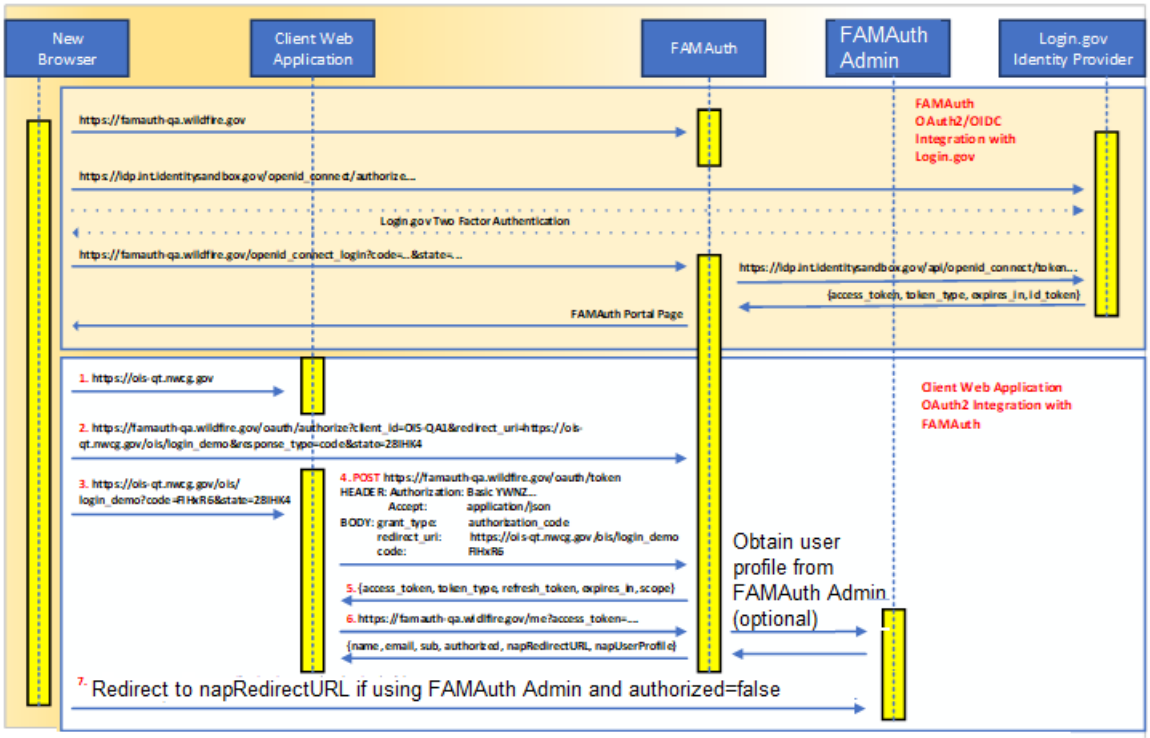


Figure 2 OAuth2/OpenID to OAuth2 Protocol Chaining

* For the Client Web Application OAuth2 handshake Implementation details (steps 1-6), refer below to the Web Application Interface Description with OAuth2.

Implementation: Login to the FAMAuth Portal

Part 1 - Authenticating with Service Provider (eAuthentication or Login.gov)

FAMAuth entails 2 parts. Part 1 is to login via eAuthentication or Login.gov from the Login Page. Upon successful authentication, the user is presented with the application page listing all available FAM Apps that have been added to FAMAuth portal. Below details the steps to exectue Part 1 of the flow, no development required at this point to view listed applications on the FAMAuth portal.

The user has to log first into FAMAuth by invoking: https://famauth-qa.wildfire.gov (FAMAuth QA Server) and selecting the LOGIN.GOV or E-AUTHENTICATION cards.

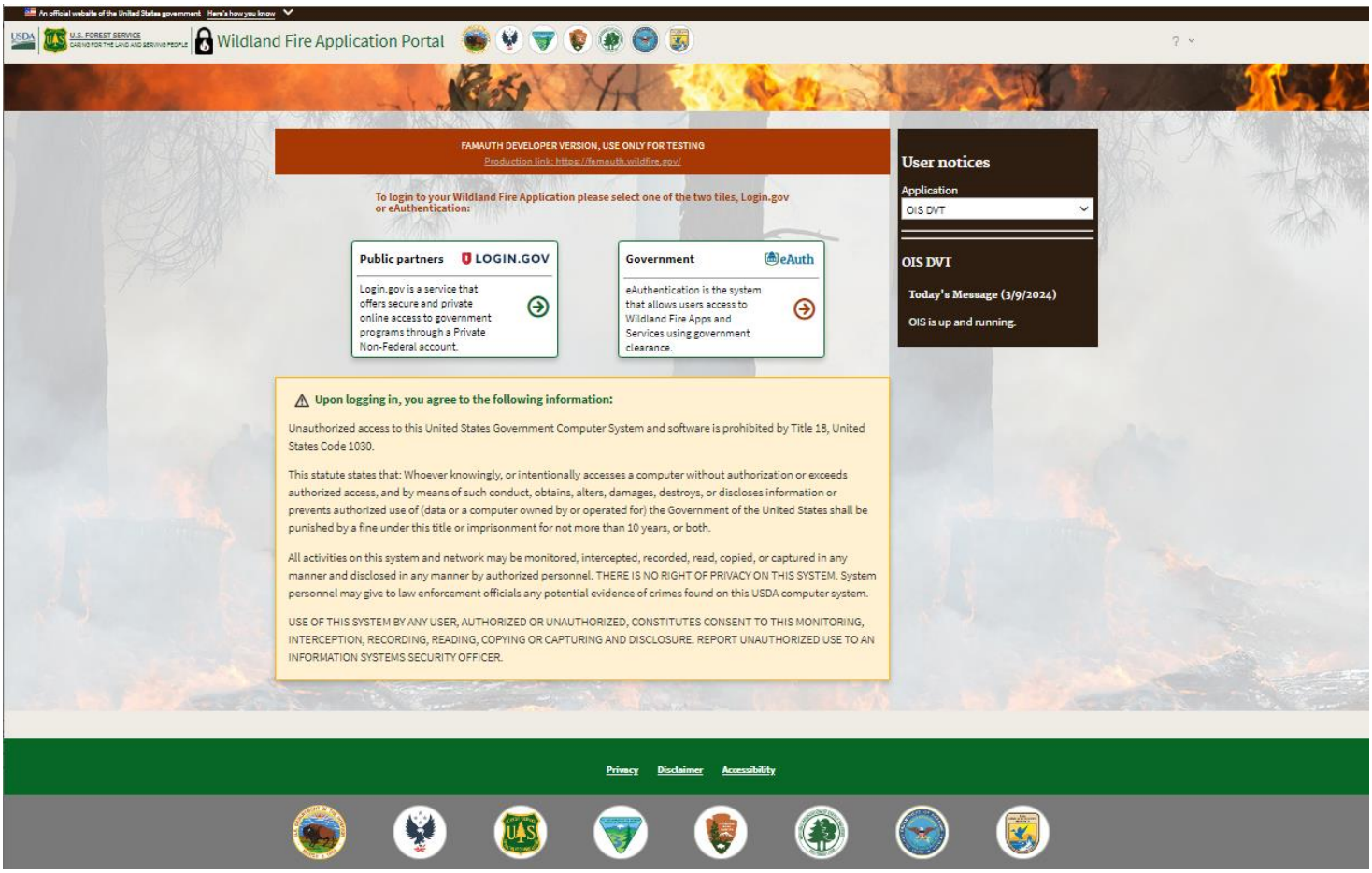


Figure 3 - Login Page

The user is authenticated using the OAuth2\OpenID integration with LOGIN.GOV or SAML2 integration with eAuthentication. These exchanges are depicted in the flow diagram above in the yellow boxes. Upon successful authentication the FAMAuth portal is shown and the user's email address from Login.gov or eAuthentication is displayed in the top right corner as shown below.

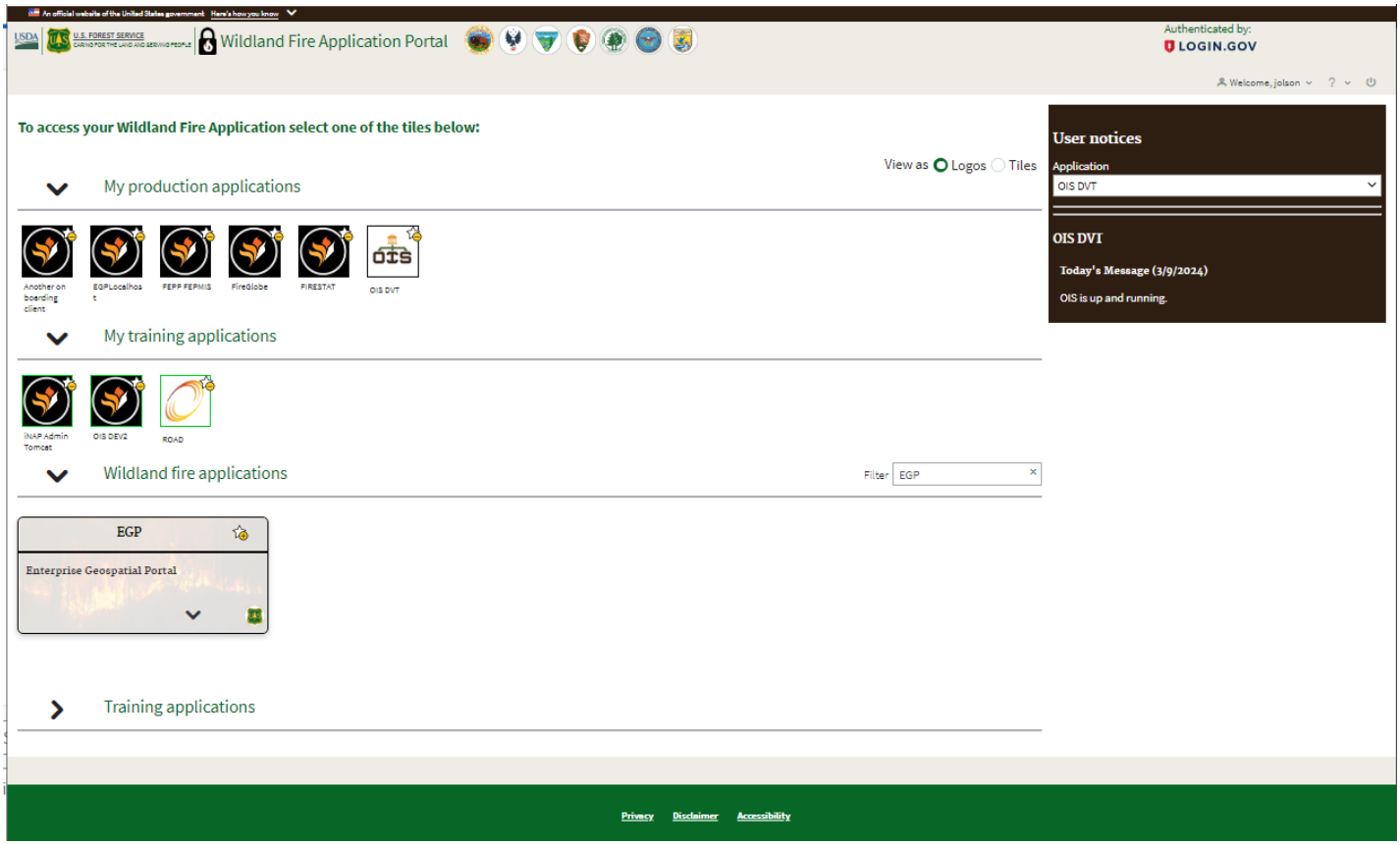


Figure 4 - Application Selection Page

Part 2 – Client Web Application Integration – 6 Steps

Part 2 applies only for applications that are listed on the FAMAuth portal. This flow is initiated upon clicking on the desired application access button associated with the application card. If you wish to add a new application to FAMAuth, see **Section Registering a Client App**.

Upon application addition with FAMAuth, the developer will need to implement the 6 steps described below that correspond to Figure 2. (See sample code in Appendix if needed for example implementation)

To obtain the authorization server meta data use the URL: https://famauth-qa.wildfire.gov/meta or https://iwfirp.nwcg.gov/meta. This data will help with your OAuth2 client setup for calling FAMAuth.

Next, follow below steps 1 to 7 below, referring to their depicted corresponding steps in the flow diagram blue boxes.

STEP 1 – Invoke Client Application

The user invokes his client web application by clicking on one of the cards of the presented applications in the FAMAuth portal. This will call the URL provided by the developer during the application registration process.

STEP 2 – Browser Redirect to FAMAuth Authorization Server

At this stage the registered client application redirects the client browser to FAMAuth, which is acting as an OAuth2 authorization server.

Here is the GET (302 redirect request) that needs to be coded in the Client application, to redirect the user to the FAMAuth authorization service:

https://famauth-qa.wildfire.gov/oauth/authorize?client_id=<my_client_id>& redirect_uri=https://<my_app_url>&response_type=code&state=<random_state>

Here is an actual GET (302 request) example.

https://famauth-qa.wildfire.gov/oauth/authorize?client_id=OIS-QA1&redirect_uri=https://ois-qt.nwcg.gov/ois/login_demo&response_type=code&state=28IHK4

The client application includes its **client id, response_type** (authorization code), **local state** (random state for preventing CSRF) and a **redirection URI** to which FAMAuth will send the browser back once the user has been authenticated (client_id and redirection URI are created during registration with FAMAuth).

STEP 3 – Authorization code returned from FAMAuth

If the user has been successfully authenticated via eAuthentication or Login.gov, FAMAuth responds to the client application with **https://<Client Web App URL>?code=<authorization_code>&state=<random_state>** containing the authorization code and the original state parameter. For example:

https://ois-qt.nwcg.gov/ois/login_demo?code=FIHxR6&state=28IHK4

No coding by the client application is required in Step 3.

STEP 4 – Authorization Code Posting for Acquiring Access Token

Upon obtaining the authorization code the client application code (need to be developed by the application team), posts in a backend channel the authorization code, redirect URI, client_id, client_secret to obtain an access_token for a successful response. The redirect URI sent to the token service must match the one that was sent to the authorize service.

POST authorization code https://famauth-qa.wildfire.gov/oauth/token?code=FIHxR6

The image below, Figure 5, depicts a Postman capture of a POST example

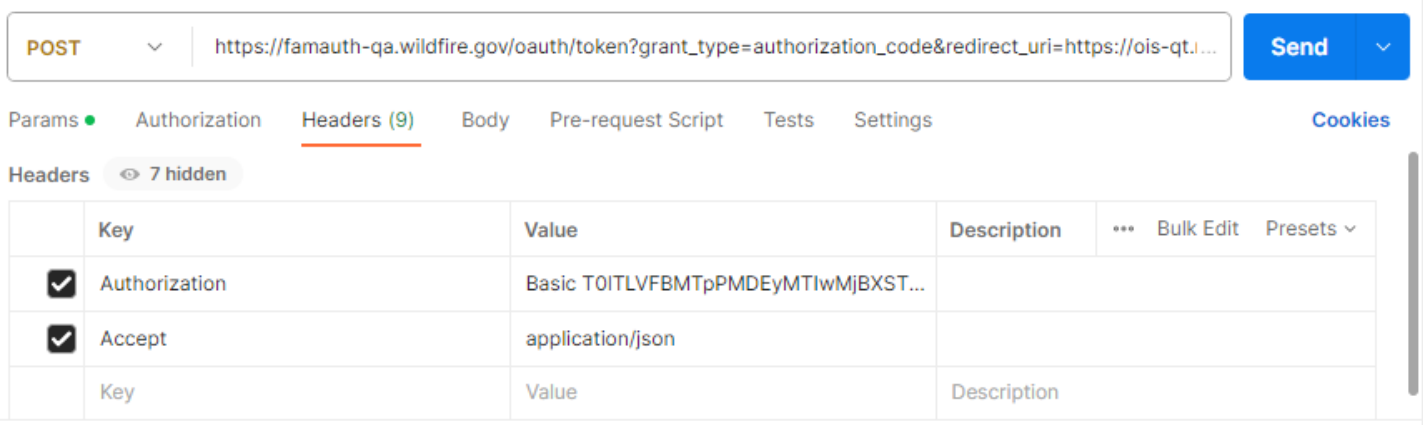


Figure 5 – Headers for TOKEN POST

Authorization: Basic YWNZ…. (**Note:** 'Basic '+ Base64 encoded 'CLIENT_ID:CLIENT_SECRET')
Accept: application/json

The image below, Figure 6, depicts the Postman capture of the POST example using query parameters, corresponding to the header above. Alternatively, these can be put in the body as form-data.

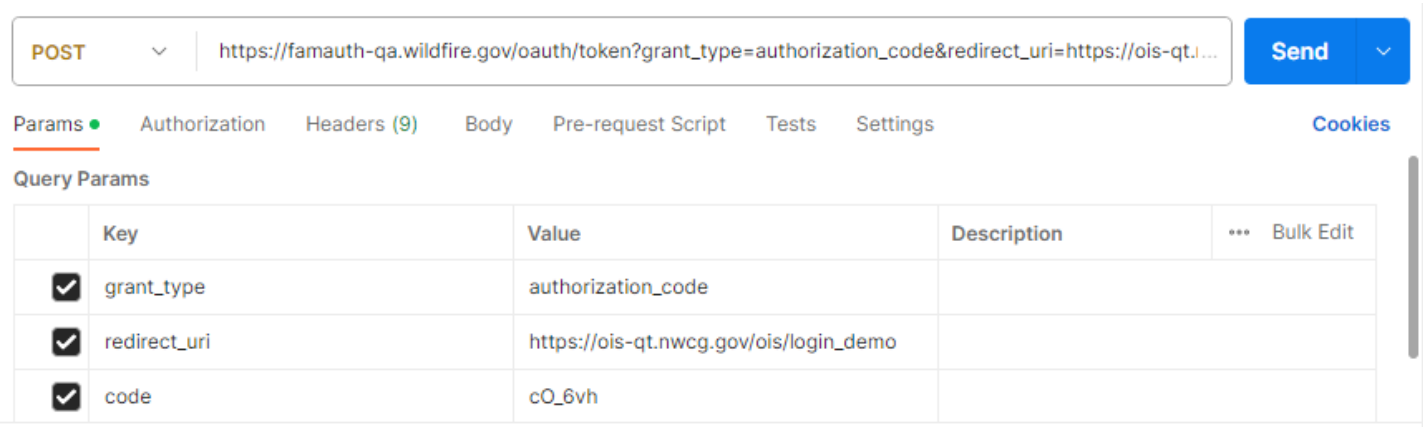


Figure 6 – Parameters for TOKEN POST

grant_type: authorization_code
redirect_uri: https://ois-qt.nwcg.gov/ois/login_demo
code: cO_6vh

STEP 5 – The FAMAuth Response to Client Posting is an Access Token

The complete roundtrip of the POST example with the returned response is shown in Figure 7 below.

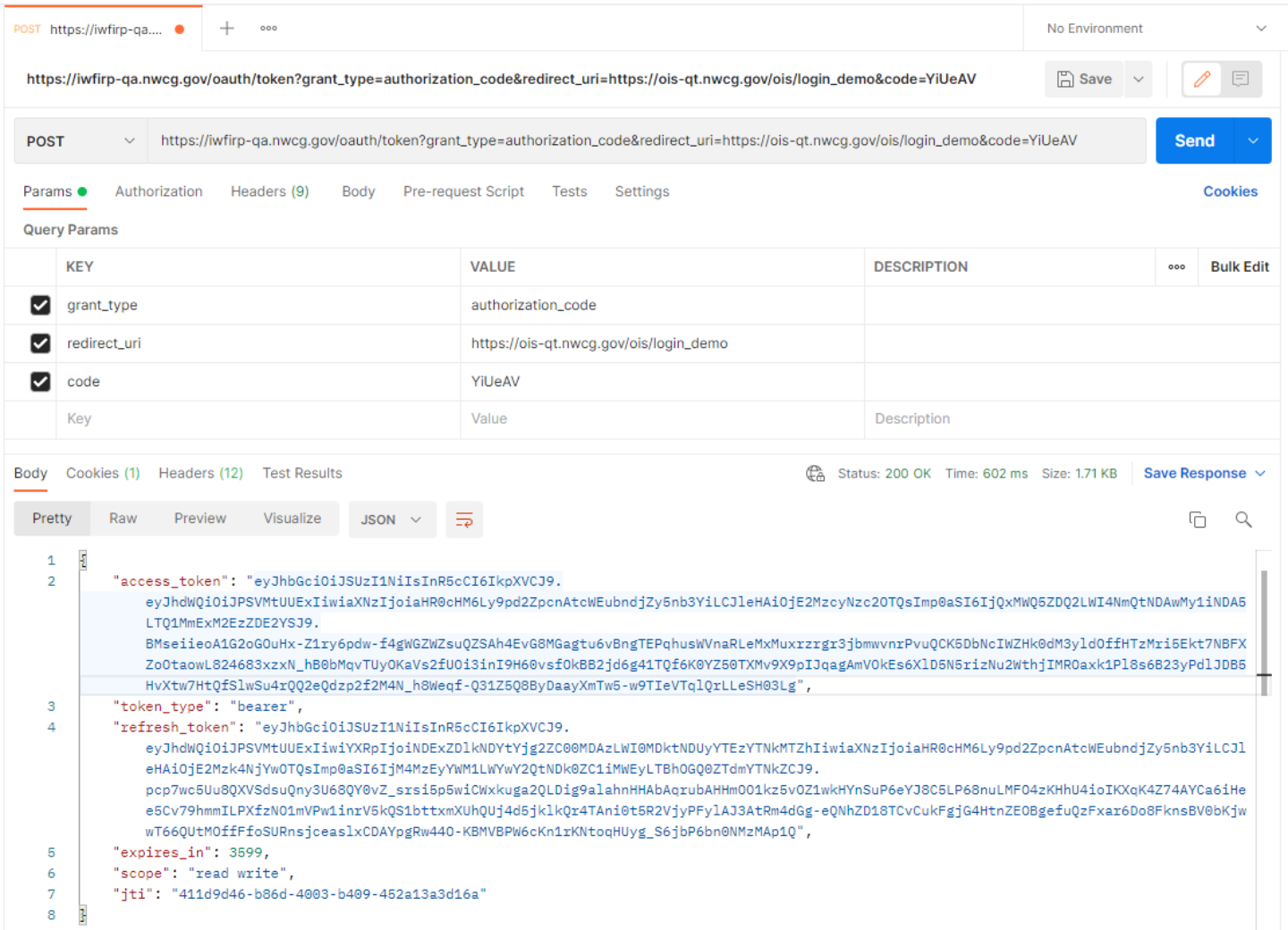


Figure 7 – Response for TOKEN POST

The response contains the following properties: {**access_token, token_type, refresh_token, expires_in, scope, jti**}.

{
"**access_token**": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJPSVMtUUExIiwiaXNzIjoiaHR0cHM6Ly9pd2ZpcnAtcWEubndjZy5nb3YiLCJleHAiOjE2MzcyNzc2OTQsImp0aSI6IjQxMWQ5ZDQ2LWI4NmQtNDAwMy1iNDA5LTQ1MmExM2EzZDE2YSJ9.BMseiieoA1G2oGOuHx-Z1ry6pdw-
f4gWGZWZsuQZSAh4EvG8Mgagtu6vBngTEPqhusWVnaRLeMxMuxrzrgr3jbmwvnrPvuQCK5DbNcIWZHk0dM3yldOffHTzMri5Ekt7NBFXZoOtaowL824683xzxN_hB0bMqvTUyOKaVs2fUOi3inI9H60vsfOkBB2jd6g41TQf6K0YZ50TXMv9X9pIJqagAmVOkEs6XlD5N5rizNu2WthjIMROaxk1Pl8s6B23yPdlJDB5HvXtw7HtQfSlwSu4rQQ2eQdzp2f2M4N_h8Weqf-Q31Z5Q8ByDaayXmTw5-w9TieVTqlQrLLeSH03Lg",
"**token_type**": "bearer",
"**refresh_token**": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJPSVMtUUExIiwiYXRpIjoiNDExZDlkNDYtYjg2ZC00MDAzLWI0MDktNDUyYTEzYTNkMTZhIiwiaXNzIjoiaHR0cHM6Ly9pd2ZpcnAtcWEubndjZy5nb3YiLCJleHAiOjE2Mzk4NjYwOTQsImp0aSI6IjM4MzEyYWM1LWYwY2QtNDk0ZC1iMWEyLTBhOGQ0ZTdmYTNkZCJ9.pcp7wc5Uu8QXVSdsuQny3U68QY0vZ_srsi5p5wiCWxkuga2QLDig9alahnHHAbAqrubAHHmOO1kz5vOZ1wkHYnSuP6eYJ8C5LP68nuLMFO4zKHhU4ioIKXqK4Z74AYCa6iHee5Cv79hmmILPXfzNO1mVPw1inrV5kQS1bttxmXUhQUj4d5jklkQr4Tani0t5R2VjyPFylAJ3AtRm4dGg-

eQNhZD18TcvCukFgjG4HtnZEOBgefuQzFxar6Do8FknsBV0bKjwwT66QUtMOffFfoSURnsjceaslxCDAYpgRw44O-
KBMVBPW6cKn1rKNtoqHUyg_S6jbP6bn0NmzMAp1Q",
"expires_in": 3599,
"scope": "read write",
"jti": "411d9d46-b86d-4003-b409-452a13a3d16a"
}

The **access_token** property is a JWT comprised of a header, payload and signature that can be validated either by your application using **https://famauth-qa.wildfire.gov/oauth/token_key** to obtain the public key, or by having FAMAuth decode and validate it by calling **https://famauth-qa.wildfire.gov/oauth/check_token?token=<access_token>**.

STEP 6 - Issuing a GET with the Access Token for the authenticated user profile retrieval

To obtain information about the authenticated user the client application calls **https://famauth-qa.wildfire.gov/me?access_token=<access_token>**. If using iNAP additional parameters of **application** and **instance** should also be included to return access and roles from iNAP.

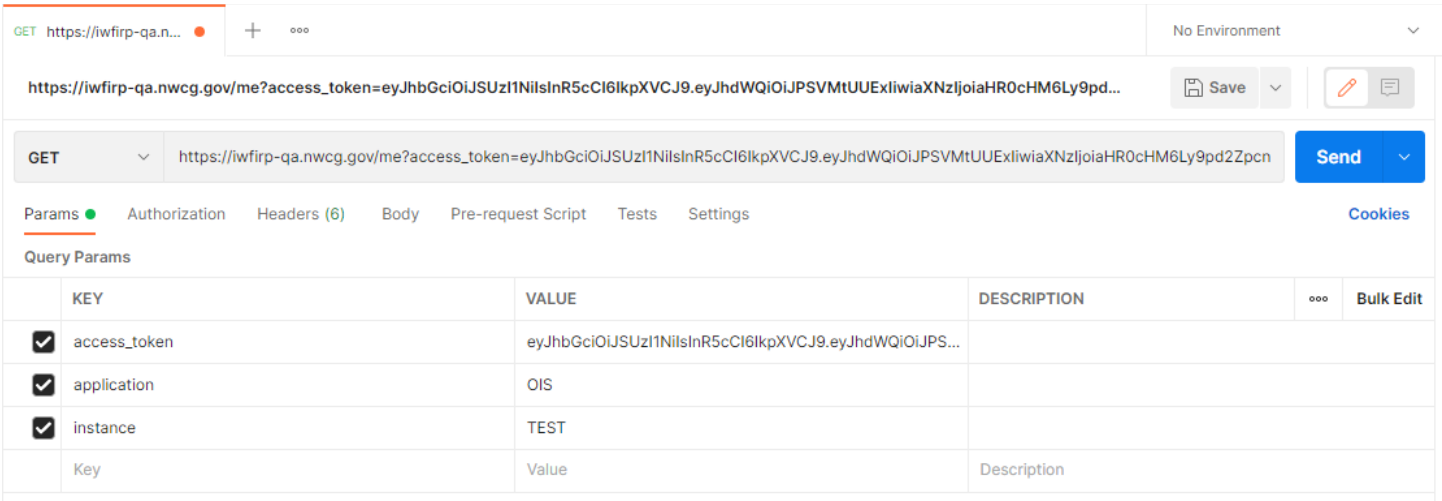


Figure 8 – GET User Information

The response contains the following properties: {**name, email, sub, user_authenticated_by, authorized, napRedirectURL, napUserProfile**}. Note that authorized, napRedirectURL and napUserProfile are only returned when the **application** and **instance** parameters are included.

Sample Response 1 - Response for eAuthentication and not using FAMAuth Admin:

{
"name": "{
\"usdaeauthid\":\"28552012122402152053535\",
\"usdafirstname\": \"JOHN\",
\"usdalastname\":\"SMITH\",
\"usdaemail\":\"john.smith@usda.gov\",
\"samaccountname\":\"null\",

```
        \"approles\":\"OCIO_ENTVPN_FS-EMP^USDA_ICAMaaS-Federation_Customer_NFC^OCIO_CHD-
Remedy_Record^OCIO_AgLearn_User\",
        \"managername\":\"null\"}",
"email": "john.smith@usda.gov",
"sub": "28552012122402152053535",
"user_authenticated_by": "e-authentication"
}
```


Sample 2 - Response for Login.gov and not using FAMAuth Admin:


```
{
"name": "{
        \"email\":\"john.smith@usda.gov\",
        \"uuid\":\"d789daf9-65b4-4c86-b6fc-5144d3eac3ac\"}",
"email": "john.smith@acme.com",
"sub": "d789daf9-65b4-4c86-b6fc-5144d3eac3ac",
"user_authenticated_by": "login.gov"
}
```


Sample 3 - Authorized response for Login.gov and using FAMAuth Admin:


```
{
"name": "{
        \"email\":\"john.smith@usda.gov\",
        \"uuid\":\"d789daf9-65b4-4c86-b6fc-5144d3eac3ac\"}",
"email": "john.smith@acme.com",
"sub": "d789daf9-65b4-4c86-b6fc-5144d3eac3ac",
"user_authenticated_by": "login.gov ",
"authorized": "true",
"napRedirectURL": null,
"napUserProfile": "{
        \"firstName\":\"John\",
        \"middleName\":null,
        \"lastName\":\"Smith\",
        \"userName\":\"jsmith\",
        \"napIdentifier\":\"1088661\",
        \"eMail\":\"john.smith@acme.com\",
        \"primaryAffiliation\":\"Contractor/Vendor\",
        \"seasonalFlag\":false,
        \"dunsNumber\":null,
        \"uei\":null,
        \"inapAdminFlag\":true,
        \"editProfileRedirectURL\":\"http://localhost/nap/oauth2/authorization/famauth?redirect_uri=%2FNAP%2F%3F
ACTION%3DfamHome%26TARGETPAGE%3DeditProfile%26EMAIL%3Dxxx%40usda.com\",
        \"identityProviders\":[{
                \"identityProviderID\":\" d789daf9-65b4-4c86-b6fc-5144d3eac3ac\",
                \"identityProviderSystem\":\"LOGIN.GOV\",
                \"identityProviderEmail\":\"john.smith@acme.com\"}],
        \"applicationRoles\":[{
```

```
                \"applicationRoleName\":\"Reference Data Administrator\",\"applicationRoleKey\":\"RDA\"},
                {\"applicationRoleName\":\"Unit ID Approver\",\"applicationRoleKey\":\"UIA\"},
                {\"applicationRoleName\":\"Account Manager\",\"applicationRoleKey\":\"ACM\"}]}",
}
```


Sample 4 - Unauthorized response for Login.gov and using FAMAuth Admin, where the client will need to redirect in the next step:


```
{
"name": "{
        \"email\":\"john.smith@usda.gov\",
        \"uuid\":\"d789daf9-65b4-4c86-b6fc-5144d3eac3ac\"}",
"email": "john.smith@acme.com",
"sub": "d789daf9-65b4-4c86-b6fc-5144d3eac3ac",
"user_authenticated_by": "login.gov ",
"authorized": "false",
"napRedirectURL": "https://nap-qt.nwcg.gov/NAP/?ACTION=login&IPID=d789daf9-65b4-4c86-b6fc-
5144d3eac3ac&IPSYSNAME=Login.gov&APP=OIS&INST=TEST&EMAIL=john.smith%40acme.com",
"napUserProfile": null",
}
```


STEP 7 – Browser Redirect to FAMAuth Admin (if using FAMAuth Admin) and Unauthorized User

If the client application is using FAMAuth Admin and the response from the previous step contains authorized = false, then the client application will redirect the user to the URL provided in the napRedirectURL. This will display a page that will assist the user to do one or more of the following:

1. Request access [request a new user] or link an existing user
2. Edit their Profile and/or Request application access and roles
3. Accept Rules of Behavior
4. Receive instruction to restore their profile [if it has been removed]
5. Reactivate their user

If the client application would like FAMAuth Admin to redirect back to the client application for the client application to retry the OAUTH2 flows to get the napUserProfile after the user accepts rules of behavior for example, the client application can append the CLIENT_URL parameter to the napRedirectURL. Retrying the OAUTH2 flows could be different per implementation of the client application. The benefit of providing the CLIENT_URL is so that the end user is more likely to successfully authorize, without the disruption of having to “Return to FAMAuth to continue.” (to click the tile again). See Appendix B – User Experience with FAMAuth Admin Flow Diagram. The Optional redirect arrows (in red) illustrate the flows when CLIENT_URL is appended.

Appendix A: Web App Examples Integrating with the Authentication Service

A. Java Client App Sample Code, Authenticating Users with FAMAuth

The following sample of a Java Servlet code (client app SIT209), authenticates SIT209 users and can be used as a reference to integrate java based apps with FAMAuth.

- client_id=FAMsit209

- client_secret=XXS31w73

- redirect_uri=http://localhost:8080/fam-web-was/SIT209/login

- response_type=code

Note that the user will have to login via the FAMAuth portal first and invoke this client app code from the portal page.

The client web application redirects the browser (302) to FAMAuth (step 2).

FAMAuth sends the first response to the client app containing the "code" with a 302 redirection (step 3).

The client app matches the returned "state" to the one sent and proceeds to get the access token with the back channel POST (step 4,5), followed by getting the user profile (step 6).

May refer to FAMAuth Service Flow Diagram with SAML2 Integration to eAuth and Client Web App Integration with OAuth2; can follow with steps 1-6:

(Step 1) User invokes the client app.

(Step 2) https://famauth-qa.wildfire.gov/oauth/authorize?client_id=FAMsit209&redirect_uri=http://localhost:8080/fam-web-was/SIT209/login/login&response_type=code&state=28IHK4

(Step 3) http://localhost:8080/fam-web-was/SIT209/login?code= FIHxR6&state=28IHK4

(Step 4) POST https://famauth-qa.wildfire.gov/oauth/token

HEADER:	Authorization:	Basic YWNZ….
	Accept:	application/json
BODY:	grant_type:	authorization_code

redirect_uri: http://localhost:8080/fam-web-was/SIT209/login/login

code: FIHxR6

(Step 5) The access_token returning from the POST is comprised of the following properties:

{Access_token, token_type, refresh_token, expires_in, scope}.

```
private static class AuthResponse {
                String access_token;
                String token_type;
                String refresh_token;
                String expires_in;
                String scope;
```

(Step 6) Issuing a GET with the access token for the authenticated user info retrieval.

https://famauth-qa.wildfire.gov/me?access_token

The sample code is comprised of the following classes:

```
package fam.security.oauth2.authentication.servlets;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Logger;
import fam.security.oauth2.authentication.utils.OauthUtils;
public class GetTokenServlet extends HttpServlet {
```

```
protected static final Logger logger =  Logger.getLogger(GetTokenServlet.class);
public void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
try {
String code = req.getParameter(OauthUtils.CODE);
String returnedState = req.getParameter(OauthUtils.STATE);
String state = (String) ((HttpServletRequest) req).getSession()
.getAttribute(OauthUtils.STATE);
if (returnedState != null && state != null
&& state.equals(returnedState)) {
OauthUtils oauthUtils = new OauthUtils();
String token = oauthUtils.getAuthorizationToken(code);
String cred = oauthUtils.getUserCredentials(token);
String sessionUserId = oauthUtils.getSessionUserId(cred);
resp.addCookie(createCookie(sessionUserId));
resp.sendRedirect(OauthUtils.REDIRECT_HOME_PAGE);
}
} catch (Exception e) {
e.printStackTrace();
throw new RuntimeException(e);
}
}
private Cookie createCookie(String sessionUserId) {
Cookie cookie = new Cookie(OauthUtils.CLIENT_COOKIE_NAME, sessionUserId);
cookie.setPath(OauthUtils.CLIENT_COOKIE_PATH);
cookie.setDomain(OauthUtils.CLIENT_COOKIE_DOMAIN);
return cookie;
}
```

}

This is the util class that does the heavy lifting. The first part is simply setting up the constants/values.

```
package fam.security.oauth2.authentication.utils;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.URI;

import java.security.KeyManagementException;

import java.security.NoSuchAlgorithmException;

import java.util.ArrayList;

import java.util.List;

import javax.net.ssl.SSLContext;

import org.apache.commons.lang.RandomStringUtils;

import org.apache.http.HttpEntity;

import org.apache.http.NameValuePair;

import org.apache.http.client.entity.EntityBuilder;

import org.apache.http.client.methods.CloseableHttpResponse;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.utils.URIBuilder;

import org.apache.http.conn.ssl.SSLConnectionSocketFactory;

import org.apache.http.conn.ssl.SSLContexts;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClients;

import org.apache.http.message.BasicNameValuePair;

import org.apache.http.params.BasicHttpParams;

import org.apache.http.params.HttpParams;

import org.codehaus.jackson.annotate.JsonIgnoreProperties;

import org.codehaus.jackson.annotate.JsonProperty;
```

```
import org.codehaus.jackson.map.ObjectMapper;
import com.ibm.ws.util.Base64;
import fam.framework.exceptions.FAMDataException;
import fam.framework.util.FAMPropertyConfigurator;
import fam.security.oauth2.authentication.db.Oauth2DB;

/**
 * Examples of properties to be set:

fam.security.oauth2.client.cookie.path=/
fam.security.oauth2.client.cookie.domain=localhost
fam.security.oauth2.url.client.redirect.home=http://localhost:8080/fam-web-was/SIT209/index.html
fam.security.oauth2.url.protocol=https
fam.security.oauth2.url.host=famauth-qa.wildfire.gov
fam.security.oauth2.url.host.path.authorize=/oauth/authorize
fam.security.oauth2.url.host.path.token=/oauth/token
fam.security.oauth2.url.client.id=FAMsit209
fam.security.oauth2.url.client.redirectURI.token=http://localhost:8080/fam-web-was/SIT209/login
fam.security.oauth2.url.responseType=code
fam.security.oauth2.url.client.secret=XXS31w73

 * @author Dan
 *
 */
public class OauthUtils {
public final static String CODE = "code";
public final static String STATE = "state";
public final static String OAUTH_REDIRECT_AUTHORIZE_URI;
```

```
public final static String CLIENT_COOKIE_PATH;

public final static String CLIENT_COOKIE_DOMAIN;

public final static String REDIRECT_HOME_PAGE;

public final static String STATE_PARM = "&state=";

public final static String CLIENT_COOKIE_NAME = "WSG_CLIENTID_COOKIE";

private final static String REDIRECT_URI = "redirect_uri";

private final static String ACCESS_TOKEN = "access_token";

private final static String CREDENTIALS_PATH = "/me";

private final static String GRANT_TYPE = "grant_type";

private final static String AUTHORIZATION_CODE = "authorization_code";

private final static String HEADER_ACCEPT = "Accept";

private final static String HEADER_ACCEPT_VALUE = "application/json";

private final static String HEADER_AUTHORIZATION = "Authorization";

private final static String HEADER_AUTHORIZATION_VALUE_BEARER = "bearer ";

private final static String PROTOCOL;

private final static String HOST;

private final static String CLIENT_ID_VALUE;

private final static String CLIENT_REDIRECT_URI_TOKEN_VALUE;

private final static String CLIENT_SECRET_VALUE;

private final static String OAUTH_SERVER_ACCESS_PATH_TOKEN;

private final static String HEADER_AUTHORIZATION_VALUE_BASIC;

private final static String RESPONSE_TYPE = "response_type";

private final static String CLIENT_ID = "client_id";

private final static String AND = "&";

private final static String EQ = "=";

private final static String QU = "?";

private final static String HOST_PATH_AUTHORIZE;

private final static String RESPONSE_TYPE_VALUE;
```

```
static {
FAMPropertyConfigurator.loadOverlays();
PROTOCOL = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.protocol");
HOST = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.host");
HOST_PATH_AUTHORIZE = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.host.path.authorize");
CLIENT_ID_VALUE = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.client.id");
CLIENT_REDIRECT_URI_TOKEN_VALUE = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.client.redirectURI.token");
RESPONSE_TYPE_VALUE = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.responseType");
CLIENT_SECRET_VALUE = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.client.secret");
OAUTH_SERVER_ACCESS_PATH_TOKEN = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.host.path.token");
CLIENT_COOKIE_PATH = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.client.cookie.path");
CLIENT_COOKIE_DOMAIN = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.client.cookie.domain");
REDIRECT_HOME_PAGE = FAMPropertyConfigurator
.getRequiredProperty("fam.security.oauth2.url.client.redirect.home");
StringBuilder sb = new StringBuilder();
sb.append(PROTOCOL).append("://").append(HOST)
.append(HOST_PATH_AUTHORIZE).append(QU).append(CLIENT_ID)
.append(EQ).append(CLIENT_ID_VALUE).append(AND)
```

```
.append(REDIRECT_URI).append(EQ)
.append(CLIENT_REDIRECT_URI_TOKEN_VALUE).append(AND)
.append(RESPONSE_TYPE).append(EQ).append(RESPONSE_TYPE_VALUE);
OAUTH_REDIRECT_AUTHORIZE_URI = sb.toString();
HEADER_AUTHORIZATION_VALUE_BASIC = "Basic "
+ Base64.encode((CLIENT_ID_VALUE + ":" + CLIENT_SECRET_VALUE)
.getBytes());
}
public static String generateRandomState() {
return RandomStringUtils.randomAlphanumeric(6);
}
public String getAuthorizationToken(String code) throws Exception {
List<NameValuePair> nameValuePairList = createNameValuePairList(code);
CloseableHttpResponse r = sendRequestPost(OauthUtils.HOST,
OauthUtils.OAUTH_SERVER_ACCESS_PATH_TOKEN, nameValuePairList);
HttpEntity responseEntity = r.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(
responseEntity.getContent()));
String line = new String();
line = reader.readLine();
responseEntity.consumeContent();
ObjectMapper mapper = new ObjectMapper();
AuthResponse atr = mapper.readValue(line, AuthResponse.class);
return atr.getAccess_token();
}
public String getUserCredentials(String token) throws Exception {
CloseableHttpResponse r = sendRequestGet(OauthUtils.HOST,
OauthUtils.CREDENTIALS_PATH, token);
```

```
HttpEntity responseEntity = r.getEntity();
BufferedReader reader = new BufferedReader(new InputStreamReader(
responseEntity.getContent()));
String line = new String();
line = reader.readLine();
ObjectMapper mapper = new ObjectMapper();
AuthResponse atr = mapper.readValue(line, AuthResponse.class);
responseEntity.consumeContent();
return atr.getName();
}
public String getSessionUserId(String cred) throws Exception {
Oauth2DB admin = new Oauth2DB();
return admin.getSessionUserId(cred);
}
private List<NameValuePair> createNameValuePairList(String code) {
List<NameValuePair> nameValuePairList = new ArrayList<NameValuePair>();
NameValuePair parm = new BasicNameValuePair(OauthUtils.GRANT_TYPE,
OauthUtils.AUTHORIZATION_CODE);
nameValuePairList.add(parm);
parm = new BasicNameValuePair(OauthUtils.REDIRECT_URI,
OauthUtils.CLIENT_REDIRECT_URI_TOKEN_VALUE);
nameValuePairList.add(parm);
parm = new BasicNameValuePair(OauthUtils.CODE, code);
nameValuePairList.add(parm);
return nameValuePairList;
}
private CloseableHttpResponse sendRequestPost(String host, String path,
List<NameValuePair> nameValuePairList) throws Exception {
```

```
URIBuilder uriBuilder = new URIBuilder().setScheme(OauthUtils.PROTOCOL)
.setHost(host);
URI uri = uriBuilder.setPath(path).build();
CloseableHttpClient httpclient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost(uri);
httpPost.addHeader(HEADER_ACCEPT, HEADER_ACCEPT_VALUE);
httpPost.addHeader(HEADER_AUTHORIZATION,
HEADER_AUTHORIZATION_VALUE_BASIC);
EntityBuilder entityBuilder = EntityBuilder.create();
entityBuilder.setParameters(nameValuePairList);
HttpEntity entity = entityBuilder.build();
httpPost.setEntity(entity);
return httpclient.execute(httpPost);
}
private CloseableHttpResponse sendRequestGet(String host, String path,
String token) throws Exception {
URIBuilder uriBuilder = new URIBuilder().setScheme(OauthUtils.PROTOCOL)
.setHost(host);
URI uri = uriBuilder.setPath(path).build();
CloseableHttpClient httpclient = getHttpClient();
HttpParams httpParameters = new BasicHttpParams();
httpParameters.setParameter(OauthUtils.ACCESS_TOKEN, token);
HttpGet httpGet = new HttpGet(uri);
httpGet.setParams(httpParameters);
httpGet.setHeader(OauthUtils.HEADER_AUTHORIZATION,
OauthUtils.HEADER_AUTHORIZATION_VALUE_BEARER + token);
httpGet.setHeader(OauthUtils.HEADER_ACCEPT,
OauthUtils.HEADER_ACCEPT_VALUE);
```

```
return httpclient.execute(httpGet);
}
private CloseableHttpClient getHttpClient() throws FAMDataException {
CloseableHttpClient httpClient = null;
SSLContext sslContext = null;
try {
sslContext = SSLContexts.custom().useTLS().build();
} catch (KeyManagementException e) {
String exceptionMsg = "KeyManagementException getting HttpClient: "
+ e.getMessage();
throw new FAMDataException(exceptionMsg);
} catch (NoSuchAlgorithmException e) {
String exceptionMsg = "NoSuchAlgorithmException getting HttpClient: "
+ e.getMessage();
throw new FAMDataException(exceptionMsg);
}
SSLConnectionSocketFactory sslConnectionSocketFactory = new SSLConnectionSocketFactory(
sslContext, new String[] { "TLSv1.2" }, null,
SSLConnectionSocketFactory.BROWSER_COMPATIBLE_HOSTNAME_VERIFIER);

httpClient = HttpClients.custom()
.setSSLSocketFactory(sslConnectionSocketFactory).build();
return httpClient;
}
@JsonIgnoreProperties(ignoreUnknown = true)
private static class AuthResponse {
String access_token;
String token_type;
```

```
String refresh_token;
String expires_in;
String scope;
String organization;
String jti;
String name;
@JsonProperty("access_token")
public String getAccess_token() {
return access_token;
}
@JsonProperty("access_token")
public void setAccess_token(String access_token) {
this.access_token = access_token;
}
@JsonProperty("token_type")
public String getToken_type() {
return token_type;
}
@JsonProperty("token_type")
public void setToken_type(String token_type) {
this.token_type = token_type;
}
@JsonProperty("refresh_token")
public String getRefresh_token() {
return refresh_token;
}
@JsonProperty("refresh_token")
public void setRefresh_token(String refresh_token) {
```

```
this.refresh_token = refresh_token;
}
@JsonProperty("expires_in")
public String getExpires_in() {
return expires_in;
}
@JsonProperty("expires_in")
public void setExpires_in(String expires_in) {
this.expires_in = expires_in;
}
@JsonProperty("scope")
public String getScope() {
return scope;
}
@JsonProperty("scope")
public void setScope(String scope) {
this.scope = scope;
}
@JsonProperty("organization")
public String getOrganization() {
return organization;
}
@JsonProperty("organization")
public void setOrganization(String organization) {
this.organization = organization;
}
@JsonProperty("cojtide")
public String getJti() {
```

```
return jti;
}
@JsonProperty("jti")
public void setJti(String jti) {
this.jti = jti;
}
@JsonProperty("name")
public String getName() {
return name;
}
@JsonProperty("name")
public void setName(String name) {
this.name = name;
}
}
}
```

B. .NET Client App Sample Code, Authenticating Users with FAMAuth

The following sample of a .NET code (client app acmenet), authenticates acmenet users, can be used as a reference to integrate .net based apps with FAMAuth.

- client_id= acmenet

- client_secret= acmesecretnet

- redirect_uri= http://fsxrnsx0128.wrk.fs.usda.gov/acme2

- response_type=code

Note that the user will have to login via the FAMAuth portal first and invoke this client app code from the portal page.

The client web application redirects the browser (302) to FAMAuth (step 2).

FAMAuth sends the first response to the client app containing the "code" with a 302 redirection (step 3).

The client app matches the returned "state" to the one sent and proceeds to get the access token with the back channel POST (step 4,5), followed by getting the user profile (step 6).

May refer to FAMAuth Service Flow Diagram with SAML2 Integration to eAuth and Client Web App Integration with OAuth2; can follow with steps 1-6:

(Step 1) User invokes the client app.

(Step 2) https://famauth-qa.wildfire.gov/oauth/authorize?client_id=acmene&redirect_uri= http://fsxrnsx0128.wrk.fs.usda.gov/acme2&response_type=code&state=28IHK4

(Step 3) http://localhost:8080/fam-web-was/SIT209/login?code= FIHxR6&state=28IHK4

(Step 4) POST https://famauth-qa.wildfire.gov/oauth/token

HEADER:	Authorization:	Basic YWNZ….
	Accept:	application/json
BODY:	grant_type:	authorization_code
	redirect_uri:	http://fsxrnsx0128.wrk.fs.usda.gov/acme2
	code:	FIHxR6

(Step 5) The access_token returning from the POST is comprised of the following properties:

{Access_token, token_type, refresh_token, expires_in, scope}.

```
private static class AuthResponse {
                String access_token;
                String token_type;
                String refresh_token;
                String expires_in;
                String scope;
```

(Step 6) Issuing a GET with the access token for the authenticated user info retrieval.

https://famauth-qa.wildfire.gov/me?access_token

fswebClientCore/fswebClientCore/*appsettings.json*

```
{
  "auth": {
    "oauth_clientId": "acmenet",
    "oauth_clientSecret": "acmesecretnet",
    "auth_code": "",
    "auth_server": "https://famauth-qa.wildfire.gov/",
    "REDIRECT_URL": "http://fsxrnsx0128.wrk.fs.usda.gov/acme2"
  },

  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

fswebClientCore/fswebClientCore/*fswebClientCore.csproj*

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
  </ItemGroup>

</Project>
```

fswebClientCore/fswebClientCore/*fswebClientCore.csproj.user*

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <NameOfLastUsedPublishProfile>FolderProfile</NameOfLastUsedPublishProfile>
  </PropertyGroup>
</Project>
```

https://code.fs.usda.gov/EAD/FS-Apps/tree/Sample_NET_Client/sample/dotNet/fswebClientCore

fswebClientCore/fswebClientCore/*Program.cs*

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace fswebClientCore
{
   public class Program
   {
      public static void Main(string[] args)
      {
         CreateWebHostBuilder(args).Build().Run();
      }

      public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
         WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
   }
}
```

fswebClientCore/fswebClientCore/*Startup.cs*

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

namespace fswebClientCore
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add services to the container.
      public void ConfigureServices(IServiceCollection services)
      {
         services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
         services.AddHttpClient();
      }
```

```
      // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
      public void Configure(IApplicationBuilder app, IHostingEnvironment env)
      {
        if (env.IsDevelopment())
        {
          app.UseDeveloperExceptionPage();
        }
        app.UseDeveloperExceptionPage();

        //          app.UseMvc();

        app.UseMvc(routes =>
        {
          routes.MapRoute("default", "{controller=Login}/{action=Login}");
        }
);

      }
    }
}
```

fswebClientCore/fswebClientCore/*appsettings.Development.json*

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
```

```
    "Microsoft": "Information"
  }
 }
}
```

fswebClientCore/Controllers/*LoginController.cs*

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace fswebClientCore.Controllers
{

    //.NET POCO object to convert from JSON object
    public class FAMUserAuth
    {
        [JsonProperty("access_token")]
        public string AccessToken { get; set; }

        [JsonProperty("token_type")]
        public string TokenType { get; set; }

        [JsonProperty("refresh_token")]
        public string RefreshToken { get; set; }

        [JsonProperty("expires_in")]
        public int ExpiresIn { get; set; }

        [JsonProperty("scope")]
        public string Scope { get; set; }

    }

    //.NET POCO object to convert from JSON Error object
    public class FAMAuthError
```

```
    {
        [JsonProperty("error")]
        public string Error { get; set; }
        [JsonProperty("error_description")]
        public string ErrorDescription { get; set; }

    }


    [Route("api/[controller]")]
    [ApiController]
    public class LoginController : ControllerBase
    {

        //Declarations
        private readonly IConfiguration configuration;
        private readonly IHttpClientFactory clientFactory;

        static string CLIENT_ID = "";
        static string CLIENT_SECRET = "";

        static string REDIRECT_URL = "";

        static string JWT_PRIVATE_KEY_PASSWORD = "";
        static string JWT_PUBLIC_KEY_ID = "";
        static string AUTH_CODE = "";
        static string AUTH_SERVER = "";

        //Contrsuctor
        public LoginController(IConfiguration configuration, IHttpClientFactory
clientFactory)
        {

            this.configuration = configuration;
            this.clientFactory = clientFactory;

            //Grabs the configuration values from the appsettings.json file
            //Set your configuration in the appsettings.json file
            CLIENT_ID = configuration.GetValue<string>("auth:oauth_clientId");
            CLIENT_SECRET = configuration.GetValue<string>("auth:oauth_clientSecret");

            AUTH_CODE = configuration.GetValue<string>("auth:auth_code");
            AUTH_SERVER = configuration.GetValue<string>("auth:auth_server");

            REDIRECT_URL = configuration.GetValue<string>("auth:REDIRECT_URL");

            JWT_PRIVATE_KEY_PASSWORD =
configuration.GetValue<string>("auth:boxPrivateKeyPassword");
            JWT_PUBLIC_KEY_ID = configuration.GetValue<string>("auth:boxPublicKeyId");

        }



        // GET api/values
```

```
        // Starting place / call that FAMAuth calls when click on the button on webpage
        // http://<server>/api/Login for this example
        [HttpGet]
        public IActionResult Login()
        {

            //Creates a random State
            Guid g = Guid.NewGuid();
            string state = Convert.ToBase64String(g.ToByteArray());
            state = state.Replace("=", "");
            state = state.Replace("+", "");
            //it has to be 6 characters
            state = state.Substring(0, 6);

            var type = "code";


            //Create the request
            //client_id = Client ID that is provided by the FAMAuth team
            //redirect_uri = URL used for callback after getting request
            //response_type = type of response , right now it's only code
            //state = session type like id but its only limit to 6 chars
            var request =
string.Format("{0}/oauth/authorize?client_id={1}&redirect_uri={2}&response_type={3}&state
={4}", AUTH_SERVER, CLIENT_ID, REDIRECT_URL, type, state);


            //Must called as a redirect , a HTTP GET will not work
            return Redirect(request);

            //After sending the request , the callback URL will be called with the Code

        }


        [Route("SignInCallback")]
        public IActionResult SignInCallback()
        {

            // Retrieve code and state from query string
            var qs =
Microsoft.AspNetCore.WebUtilities.QueryHelpers.ParseQuery(Request.QueryString.ToString())
;
            var code = qs.Where(q => q.Key == "code").FirstOrDefault().Value;
            var state = qs.Where(q => q.Key == "state").FirstOrDefault().Value;

            //64 Encodes the Clientid and ClientSecret provided by FAMAuth team in the
format of
            // <clientid> : <client Secret> as string
            string auth =
Convert.ToBase64String(Encoding.UTF8.GetBytes($"{CLIENT_ID}:{CLIENT_SECRET}"));

            //POCO object decalartion
            FAMUserAuth userAuth = null;
```

```
            //Declaration for end output display
            StringBuilder sb = new StringBuilder();

            // Send POST to make token request
            // Creates a WebClient
            using (var wb = new WebClient())
            {

                //Adds Http Headers to the POST
                //Accept , Json as result
                //Authorization , pass the base 64 encoded client id : client secret
string
                wb.Headers.Add("Accept", "application/json");
                wb.Headers.Add("Authorization", $"Basic {auth}");

                //build collection to post as body
                //code = value recieved by first call
                //grant type = for now , they only have authorization code
                //redirect url , does nothing at the moment
                var data = new NameValueCollection();
                data["code"] = code;
                data["grant_type"] = "authorization_code";
                data["redirect_uri"] = REDIRECT_URL;


                //Send the POST call to the server with the code from the first step and
additional parameters above
                var response =  wb.UploadValues($"{AUTH_SERVER}/oauth/token", "POST",
data);

                try
                {
                    //Convert the byts of information to string
                    var responseString =  Encoding.ASCII.GetString(response);

                    //build string to output debug info
                    sb.AppendLine("User Token Json");
                    sb.Append( wb.Encoding.GetString(response));

                    //Parses Json string into .NET Object
                    userAuth =
JsonConvert.DeserializeObject<FAMUserAuth>(responseString);
                }
                catch(WebException ex)
                {

                    return Content(ex.Message);

                }


                //Make another call to get the username
                //get username by token
                string _UserNameUri = AUTH_SERVER + "/me?access_token=" +
userAuth.AccessToken;
```

```
                var resp = wb.DownloadString(_UserNameUri);
                sb.AppendLine();
                sb.AppendLine();
                sb.AppendLine("User Name");
                sb.AppendLine(resp);



                //output example debug information
                return Content(sb.ToString());


                //What's Next?
                //With the information , username , you will use this information with
your application's authentication schema / authorization code.



            }

        }

    }


}
```

C. Spring Boot Client App Sample Code, Authenticating Users with FAMAuth

An example web client with Spring Boot follows. Refer to the Postman screen captures in the main document for other technology types of web application integration.

Creating a New Project

First create a Spring Boot application, going to http://start.spring.io and generate an empty project (choosing the "Web" dependency as starting points).

You can then import that project into your favorite IDE as maven.

Create a Client Application

A client web application is itself a web Application that may be created with Spring Boot. Here's an example:

ClientApplication.java
[source,java]

```
@EnableAutoConfiguration
@Configuration
@EnableOAuth2Sso
@RestController
public class ClientApplication {

  @RequestMapping("/")
  public String home(Principal user) {
    return "Hello " + user.getName();
  }

  public static void main(String[] args) {
    new SpringApplicationBuilder(ClientApplication.class)
       .properties ("spring.config.name=client").run(args);
  }

}
```

The ingredients of the client are a home page (just prints the user's name), and an explicit name for a configuration file (via `spring.config.name=client`). When we run this app it will look for a configuration file as follows:

.client.yml
[source,yaml]
----
server:
  port: 9999
  context-path: /client
security:
  oauth2:
    client:
      client-id: myfsapp
      client-secret: myfsappsecret
      access-token-uri: http://<FAMIT Service>:8080/oauth/token
      user-authorization-uri: http://FAMIT Service>:8080/oauth/authorize
    resource:
      user-info-uri: http://<FAMIT Service>:8080/me

The app will run on port 9999.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.example</groupId>
   <artifactId>social-auth-server</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>social-auth-server</name>
   <description>Demo project for Spring Boot</description>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.0.1.RELEASE</version>
      <relativePath /> <!-- lookup parent from repository -->
```

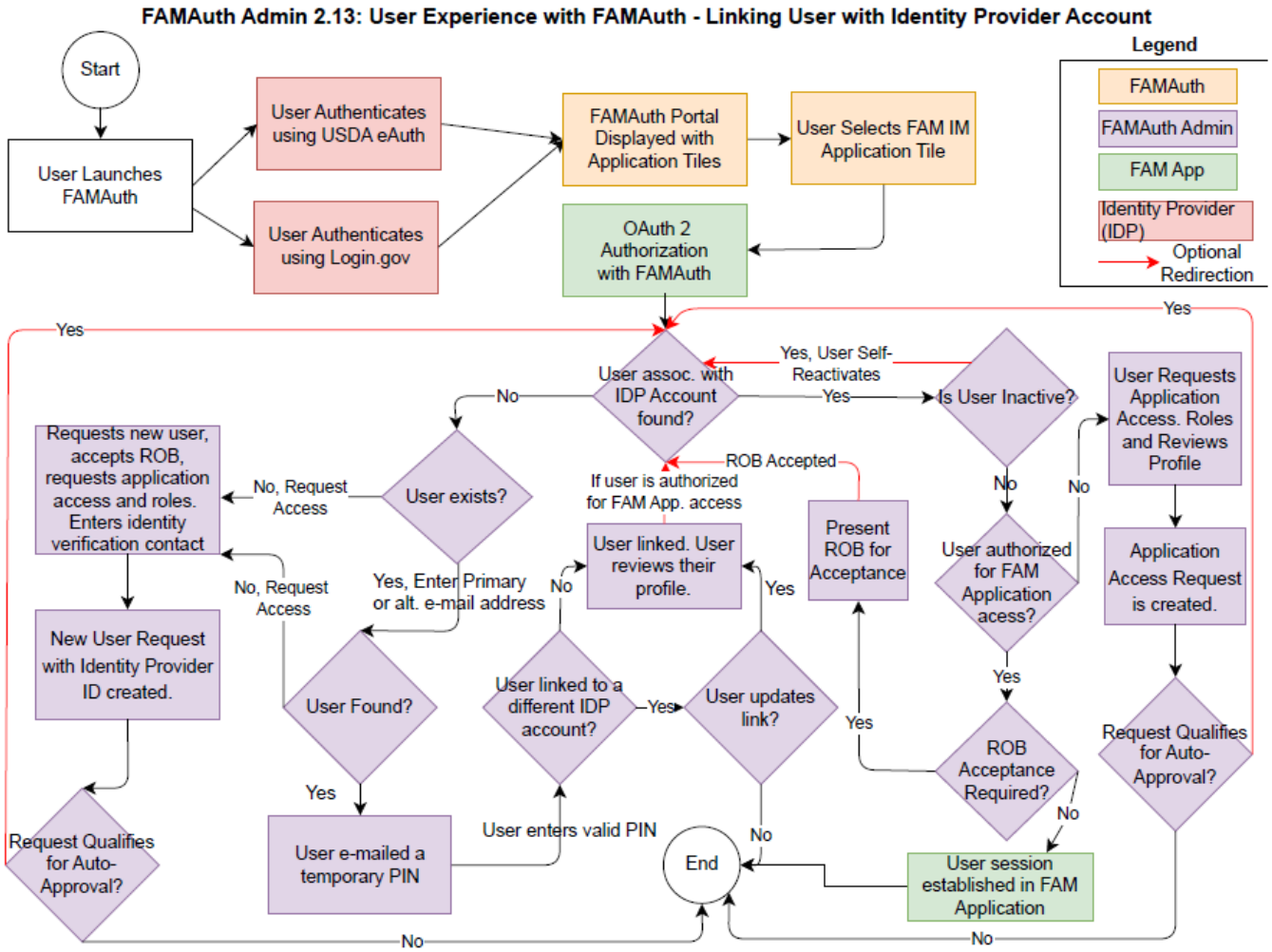
```
</parent>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <java.version>1.8</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
        <dependency>
            <groupId>org.springframework.security.oauth.boot</groupId>
            <artifactId>spring-security-oauth2-autoconfigure</artifactId>
            <version>2.0.0.RELEASE</version>
        </dependency>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>js-cookie</artifactId>
    <version>2.1.0</version>
  </dependency>
```

```
    <dependency>
      <groupId>org.webjars</groupId>
      <artifactId>jquery</artifactId>
      <version>2.1.1</version>
    </dependency>
    <dependency>
      <groupId>org.webjars</groupId>
      <artifactId>bootstrap</artifactId>
      <version>3.2.0</version>
    </dependency>
    <dependency>
      <groupId>org.webjars</groupId>
      <artifactId>webjars-locator-core</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-configuration-processor</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
  </build>
</project>
```

Appendix B: User Experience with FAMAuth Admin

The following flow chart depicts how new or existing FAMAuth admin users establish links with their EAuth and/or Login.gov accounts for purposes of authorization. Additionally the user may update their profile, and request application access and roles.



Frequently Asked Questions (FAQs)

What kind of code does a client app need to develop to be able to integrate with FAMAuth?

- There are currently three dominant and leading open web standards on the Internet for authentication and FAMAuth implements all three: OAuth2, OpenID and SAML. However, a client app will only have to develop code for the OAuth2 protocol to integrate with FAMAuth. FAMAuth itself uses SAML passive profile during the initial login process to authenticate a user with his/her PIV card or OAuth2 with OpenID to authenticate a user with Login.gov.

What application info is needed to integrate with FAMAuth?

- App short name
- App long name
- App description
- App logo
- App URI
- App region
- App state
- App product owner
- App business owner
- App point of contact
- App client ID
- App client secret
- App redirect_uri (to return the authorization code to the app

Can I invoke the client app URL directly without first invoking FAMAuth?

- No. You must first log in to FAMAuth at the following link (no coding required for this flow) https://famauth-qa.wildfire.gov. Then proceed with the eAuth ot Login.gov authentication selection.
- You may test your OAuth2 code by next pressing the access button associated with your application card.

What are the high-level flow steps for authenticating a client app user?

A. Client app user invokes FAMAuth at the following link; this is where the user is authenticated with his/her PIV card (using the SAML protocol) or Login.gov (Using OAuth2\OpenID protocol) by selecting an authentication alternative. The client app OAuth2 exchange takes place next.
   https://famauth-qa.wildfire.gov

B. When the user clicks on the application card rendered by FAMAuth, the Client App issues the redirection to FAMAuth, initiating the Oauth2 exchange.
   - https://famauth-qa.wildfire.gov/oauth/authorize?client_id= achme&
redirect_uri=https://< Client Web App>/login&response_type=code&state=28IHK4

C. FAMAuth returns an authorization code to the Client App upon successful user PIV or Login.gov authentication confirmation. https://<Client Web App>/login?code= FIHxR6&state=28IHK4.

D. The Client app uses its basic authorization credentials (Client_ID:Client_Secret) via a back host channel, to POST and acquire an access token for a successfully authenticated user.

POST https://famauth-qa.wildfire.gov/oauth/token

E. The access token received is being submitted next via the back-host channel with a GET to retrieve the user profile.

https://famauth-qa.wildfire.gov/me?access_token=.....

How does the initial GET app handshake take place?

- In STEP 2 – Browser Redirect to FAMAuth; a GET 302 redirect request is sent to the browser and not directly as a GET to the server, to point to FAMAuth, acting as the OAuth2 authorization server.

What is the role of the "state" and is it Unique?

- In STEP 2 https://famauth-qa.wildfire.gov /oauth/authorize?client_id=acme& redirect_uri=http://<myapp_server>/login&response_type=code&state=28IHK4, the uniqueness of the opaque value state is being enforced, preventing CSRF attacks during the access token retrieval flow. Make sure that a new random state value is being generated and matched against the returned state value (STEP 3) for every authentication request.

How does a client app authenticate with FAMAuth?

- In STEP 4 – Authorization Code Posting for Acquiring Access Token, the basic client application authorization validation against the client application stored credentials (Client_ID, Client_Secret) takes place with the header of the POST request. For "KEY" Authorization the "VALUE" is "Basic CLIENT_ID:CLIENT_SECRET". However, the actual string with values CLIENT_ID:CLIENT_SECRET has to be 64 encoded.

What is the authenticated user profile provided by FAMAuth from eAuth?

- { "usdafirstname": "TOM", "usdaemail": "tom.himonas@usda.gov", "usdalastname": "HIMONAS", "usdaeauthid": "1111111111111111", "samaccountname": "TOM.HIMONAS", "jti":"ec8432df-8c23-41e5-b483-f58ff8ade361" }

Login.gov provides only a user's email and uuid.

Does FAMAuth handle app role authorization?

- Yes, FAMAuth can return a user profile including access and roles from FAMAuth Admin. This is not required – an application can choose to store and maintain its own roles.

How are user application sessions managed?

- Once a user is authenticated with FAMAuth and returned to the App, it is up to the application to manage the user session. We do not remotely invalidate or expire a user's session.

Can I pass out a book mark to my application?

- Yes, give out the URL to your application that will initiate the call to the FAMAuth /oauth/authorize URL which is most likely the URL your applications FAMAuth tile opens a browser window in Step 2 above. When FAMAuth identifies the user is not authenticated by FAMAuth their browser will redirect to the FAMAuth login/landing page to choose Login.gov or E-Auth as their authentication provider. After the user is authenticated by one of the IDPs and comes back in to FAMAuth, FAMAuth will redirect the users browser back to the /oauth/authorize URL the users system sent originally to finish the OAUTH2 handshaking and the user will eventually redirect back to the application bookmarked.

Can I get a new access_token after it expires?

- Yes, FAMAuth has a reusable refresh_token that comes back with the first access_token received from the /token service, which means you will need to store it off for later use. The access tokens are valid for 30 minutes and the refresh tokens are valid for 12 hours, after they both expire you will not be able to call the /token or /me services without starting the OAUTH2 flow over again to reauthenticate and reauthorize. You will need basic authentication set to your apps client_id and client_secret, a grant_type of "refresh_token" and a refresh_token of the first refresh token received sent on the /token service call.

Is there a FAMAuth logout service?

- Yes, FAMAuth has a logout service that can be called. Applications can call the following service using the user_authenticated_by value returned by the /me service as the value to the logout service authenticatedBy parameter. https://famauth-qa.wildfire.gov/logout?authenticatedBy=<e-authentication/login.gov>