Towards We-Government: Collective and participative approaches for addressing local policy challenges

Unified WeGovNow User Management and Integration Framework

INTEGRATION MANUAL AND STEP BY STEP GUIDE

Andreas Nitsche and Björn Swierczek

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 693514.

For information regarding OAuth2 we recommend the following specification documents:

- RFC 6749: "The OAuth 2.0 Authorization Framework”
- RFC 6750: "The OAuth 2.0 Authorization Framework: Bearer Token Usage"

Test platform for integration (WeGovNow specific)

LiquidFeedback with the UWUM server extension is available at

https://wegovnow.liquidfeedback.com/

The base URL of the API is

https://wegovnow.liquidfeedback.com/api/1/

All partners received invite codes for self-registration of user accounts. More codes are available on request.

UWUM API endpoints (WeGovNow specific)

The UWUM endpoints for OAuth2 and the integration framework are available at the following URLs:

https://wegovnow.liquidfeedback.com/api/1/authorization
https://wegovnow-cert.liquidfeedback.com/api/1/token ***
https://wegovnow.liquidfeedback.com/api/1/validate
https://wegovnow.liquidfeedback.com/api/1/navigation *
https://wegovnow.liquidfeedback.com/api/1/style **
https://wegovnow.liquidfeedback.com/api/1/client **

* The content provided by this endpoint is an example and subject to further improvements.

** These endpoints are subject to change.

*** Due to limitations in TLS and the way web browsers handle client certificate requests, we need a different hostname for client requests providing a client certificate. For requests made by a client application providing a client certificate please always use the following alternative API base URL:

https://wegovnow-cert.liquidfeedback.com/api/1/

Usage of scopes / screen name

When acting as WeGovNow UWUM client without data exchange with other WeGovNow applications, you will only need to request the "authentication" or the "identification" scope to be able to identify the user.

These scopes allow to retrieve some user related information (i.e. numerical ID, identification string, screen name) to identify the user. When using the "authentication" or the "identification" scope, the response of the /api/1/token endpoint can optionally include an additional data structure providing member information.

To request this optional "member" data structure, you need to set the parameter "include_member" to 1 or "true". Using the "identification" scope with the parameter "include_member" set to true, the response to the /api/1/token endpoint could look like as follows:


```
{
   "access_token": "UFYPzKrz7JHIKATI",
   "expires_in": 3600,
   "refresh_token": "5QM8OL7AbdabXusG",
   "token_type": "bearer",
   "member_id": 123,
   "member": {
      "id": 123,
      "name": "Johnny",
      "identification": "John Doe"
   }
}
```


The field "id" of the "member" object contains the static numerical ID of the user (equal to "member_id", i.e. redundant), the field "name" contains the screen name chosen by the user, the field "identification" contains the identification string set by the authority which identified the user as unique and eligible to use the WeGovNow application. In future there may be more fields according to the upcoming specification of the /api/1/member endpoint of LiquidFeedback.

The parameter "include_member" can also be used at the /api/1/validate and the /api/1/info endpoints.

When acting as WeGovNow application using user related data or services of other WeGovNow applications, you will need to request the appropriate scopes from UWUM for the types of actions you want to perform with other WeGovNow applications (e.g. if you want to post new
content to other applications, request the scope "post"; if you want to rate content in

other applications request the scope "rate"; ...)

When acting as WeGovNow resource server (i.e. when offering user related data or services to other WeGovNow applications) you need to check (via the /api/1/validate endpoint) the scopes of the access token you received from the requesting application (e.g. if another application tries to post content for a user, check for scope "post"; if another application tries to rate content, check for the scope "rate").

Handling of updated user related data / user's email addresses

When a WeGovNow application wants to send notification emails to users, it is not adequate to retrieve the email address only once from UWUM as the notification email can be changed by the user at any time. Such a change needs to be reflected by all applications using this email
address. Therefore an application needs to retrieve the current notification email address ***directly*** before using it, in fact again before every usage.

For that purpose, the newly introduced API endpoint GET /api/1/notify_email can be used (using an access token with the "notify_email" scope). To be able to retrieve the email address while the user is not currently logged in, it will be necessary to request the "notify_email_detached" scope when identifying the user and to store the received refresh token permanently. The suffix "_detached" requests a scope for detached usage, i.e. for usage even after the user logs out. Please note, when exchanging a refresh token for an access token after the user has been logged out, you must explicitly request the "*_detached" scope(s) you need, e.g. "notify_email_detached" using the scope parameter of the /api/1/token endpoint.

Similar situations can occur related to other member properties stored in one application but used in another one, e.g. the screen name. But these seem not to be as critical as to avoid using an outdated email address. Such properties could be cached for a limited time before retrieving them again from the application storing this property.

Sustainability, unregistered third-party clients and the future

Following these rules, even a complete new (non-registered, third-party) application can easily make use of the WeGovNow infrastructure. The application can request certain scopes from UWUM (which can be granted or declined by the user) and use the appropriate services of all other WeGovNow applications. Using the upcoming application and service discovery, this is also possible vice versa.

Scopes vs. user privileges

NB: The scope does NOT grant a privilege to a user, it just means an application can trigger an action within the scope ***if*** the user is authorized to perform the action. Example voting: an application needs the scope "vote" to cast a vote on behalf of the user but casting a vote will only work if the user has the necessary voting privileges.

You can think of this as a matrix of scopes and user privileges or (alternatively) as a logical AND conjunction. Scopes control that an application does not misuse user privileges: while the trusted WeGovNow applications can request certain scopes without user interaction, a non-trusted third party application/client would trigger a request for a confirmation by the user "Do you want to allow application X to cast votes on your behalf? [yes, one time / yes, permanently]" (compare to permissions for third party Twitter/Facebook clients and Android permissions).

The access token which is bound to a specific user is only used to authorize an action on behalf of the user. Whether a user is allowed to perform a certain action must be checked nonetheless. This is out of scope of UWUM but part of the "logic" of each application. Which means that UWUM authorises that indeed the one who calls is the right person BUT it's on every core component side to handle the "business logic" such as: (allow user to vote only once, mark an issue as solved, etc).

List of scopes

* authentication
Authenticate the current user by reading its
  - unique static ID (id)
  - current screen name (name)

* identification
Identify the current user by reading its unique ident string (identification).
Automatically implies scope "authentication".

* notify_email
Read the notification email address of the current user (notify_email)

* read_contents
Read any user generated content (w/o authorship, ratings and votes)

* read_authors
Read the author names of user generated content (author's static ID and screen name)

* read_ratings

Read rating scores by other users

* read_identities
Read the identities of other users (identification)

* read_profiles
Read the profiles of other users (e.g. phone number, self-description)

* post
Post new content

* rate
Rate user generated content (e.g. thumbs up/down, "+1", support an initiative, rate a suggestion)

* vote
Finally vote for/against user generated content in a decision (i.e. vote on an issue)

* profile
Read profile data of current user (e.g. phone number, self-description, ...)

* settings
Read current user's settings (e.g. notification settings, display contrast, ...)

* update_name
Modify user's screen name (name)

* update_notify_email
Modify user's notification email address (notify_email)

* update_profile
Modify profile data (e.g. phone number, self-description, ...)

* update_settings
Modify user settings (e.g. notification settings, display contrast, ...)

Any of these scopes can be suffixed with "_detached" to request the scope for usage without the need for the user to be logged in. This should only be used when it is really needed.

X.509 certificate for client identification

To create a trustworthy relationship between applications using UWUM and the central UWUM component, we will use X.509 certificates. Therefor, any official WeGovNow client is required to provide a valid X.509 certificate with each request made to the central UWUM service. For this purpose we kindly ask all technical partners to provide X.509 certificate signing requests to be signed by the UWUM Certificate Authority.

For more information on X.509 certificates and signing requests, please refer to the documentation of your preferred TLS software such as LibreSSL or OpenSSL.

Also see point 6 of the Integration Checklist.

Integration Checklist

We will support all technical partners during UWUM integration. We defined a number of steps we like to accomplish with every technical partner. In the following list, the term "client application" refers to the application to be integrated with UWUM.

1. Availability of application via IPv4
   The client application is available via a defined URL using IPv4.

2. Availability of application via IPv6.
   The client application is also available using IPv6.

3. Serving via HTTPS
   The client application service is encrypted via HTTPS.

4. Publicly trusted X.509 certificate for end users
   The client application server provides a publicly trusted X.509 certificate (see 3).

5. OAuth2 redirection endpoint defined
   The URL of the OAuth2 redirection endpoint of the client application has been determined and submitted to LiquidFeedback (FlexiGuided GmbH).
   - Endpoints need to be pre-registered for security reasons.
   - Multiple endpoints are possible (one is the default endpoint).
   - The endpoint can be selected by the redirect_uri parameter.
   - If redirect_uri is omitted the default endpoint will be used.

6. Certificate signing request for UWUM
   A private key for accessing the UWUM API has been created. A corresponding certificate signing request (CSR) has been submitted to LiquidFeedback (FlexiGuided GmbH).
   - Either co-signed X.509 end user certificate or a new certificate (only for communication between the application and UWUM).
   - Example for creating a CSR:
     openssl req -out wegovnow.infalia.com-uwum.csr -new -newkey rsa:2048 -nodes -keyout wegovnow.infalia.com-uwum.key

7. Certificate signed by UWUM Certificate Authority
   A signed certificate for the client application has been sent back to the technical partner.

8. Successful X.509 secured connection
   The client application has successfully established a secured connection with the UWUM server, e.g. using LiquidFeedback's /info API endpoint.

9. Authorization endpoint access
   The client application can redirect an end user to the UWUM authorization endpoint.

10. Authorization endpoint error response handling
    The client application is capable of receiving authorization errors through its OAuth2 endpoint.

11. Authorization endpoint error display
    The client application is able to display authorization error messages (see 10) to the end user.

12. Successful authorization request and user identification
    The client application made a successful authorization request, received a UWUM access token, and determined the end user ID.

13. Using access token for API calls to other components
    The client application has successfully performed a LiquidFeedback API call (e.g. to the /info API endpoint) using a previously obtained UWUM access token.

14. Accepting access token from other components
    The client application (acting as resource server) provides at least one API call which accepts a UWUM access token for authorization.

15. Access token verification
    The client application (acting as resource server) is capable of verifying the validity and scope of a UWUM access token passed from another component (see 14).

16. Access token verification errors
    The client application (acting as resource server) is capable of handling error responses during validation of UWUM tokens (see 15).

17. Accepting access tokens as "Authorization" header
    In conformance with RFC 6750 (Bearer Token Usage), the client application (acting as resource server) accepts UWUM access tokens through the HTTP request header field "Authorization".

18. Cross-origin resource sharing (CORS)
    The client application (acting as resource server) allows cross-origin resource sharing (CORS). See also https://www.w3.org/TR/cors/

19. HTTP Strict Transport Security (HSTS)
    The client application ensures secure access by using HTTP Strict Transport Security (HSTS) according to RFC 6797.

20. Cross-application navigation
    The UWUM navigation bar has been successfully integrated into the client application.
    - An additional format (RAW HTML instead of JSON) for the navigation endpoint was introduced by request of UniTo.
    - .../api/1/navigation?**format=raw_html**&access_token=your_access_token
    - see “display of current application”
    - see “enhanced login button (redirect to calling application)”
    - see “user specific menu bar for logged in users”

Navigation bar: display of (currently) active applications

Please add the parameter client_id=<your client id> to your navigation endpoint call. This enables visual highlighting of the navigation bar entry corresponding to your application. Thanks to Yiannis for the idea!

Navigation bar: Enhanced login button (redirect to calling application)

To make sure the user is redirected to the application from which he/she came from you can add the parameter login_url=<your login_url> to your navigation endpoint call.

The URL needs to be set as a link to the page on your site which initiates the OAuth2 login or alternatively you can directly provide the URL of the /api/1/authorization endpoint including the appropriate client_id, redirect_uri and state.

If you cache the result of the navigation endpoint and your login_url is volatile, please set a unique placeholder which can be replaced any time you deliver a page containing the navigation bar.

Examples of /api/1/navigation call (without proper encoding to enhance readability)

/api/1/navigation?client_id=wegovnow.infalia.com&login_url=/component/slogin/provider/uwum/auth

/api/1/navigation?
client_id=wegovnow.infalia.com
&login_url=https://wegovnow.liquidfeedback.com/api/1/authorization?
client_id=wegovnow.infalia.com
&redirect_uri=https://wegovnow.infalia.com/?option=com_slogin&task=check&plugin=uwum
&state=mysecretstate

/api/1/navigation?client_id=wegovnow.infalia.com&login_url=RANDOMPLACEHOLDER_134jn4hjnf9823

r23dd

Navigation bar: user specific menu bar for logged in users

Please don't forget to include the access token (parameter access_token) when calling the navigation bar for a logged in user. This way the navigation bar will display the screen name of the user and the link to the (future) user menu.

Example: .../api/1/navigation?access_token=your_access_token

CORS request to check if a user is logged in (without actually triggering a login)

There may be situations where you want to check whether a user is currently logged into WeGovNow without actually forcing the user's web browser to perform a login if no user was logged in. We evaluated multiple ways to solve this issue (including a "prompt=none" parameter
for the /api/1/authorization endpoint, similiar to the solution used by OpenID), but unfortunately it turned out that most ways
- are infeasible for 3rd-party clients,
- would lead to unwanted data exposure,
- have a bad response performance, or
- have a high implementational effort,

or a combination thereof.

We finally found a solution based on a CORS (Cross-Origin Resource Sharing) XMLHttpRequest done by the web browser. This solution works with any registered client but also with any 3rd-party client and prevents unwanted data exposure. However, since the request is done by the user's web browser, the answer is ***not*** authoritative and must only be used as a hint.

A returned member_id **MUST** still be validated via the regular OAuth2 procedure using the GET /api/1/authorization endpoint! In this case, the authorization endpoint will not show a login window (because the user is already logged in).

The call to the new API endpoint POST /api/1/session does not need any parameter (and SHOULD NOT add any request header) but may only be called directly by the web browser of the user and requires the "withCredentials" option of the XMLHttpRequest object to be set to true (see code example below). The result is a JSON object with the member_id attribute set to the static ID of the current user (or to null if there is no logged-in user or if a 3rd-party client is not authorized to obtain the login status):

```
{ "member_id": 123 }
```

or

```
{ "member_id": null }
```

Example JavaScript code:

```
function checkWeGovNowSession(callback) {
  var xhr = new XMLHttpRequest();
  var url = "https://wegovnow.liquidfeedback.com/api/1/session";
  xhr.open("POST", url, true);
  xhr.withCredentials = true; // sends UWUM cookies to UWUM (important)
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      if (xhr.status == 200) {
        var r = JSON.parse(xhr.responseText);
        callback(r.member_id);
      } else {
        // some error occured, add error handling
        callback(undefined);
      }
    }
  }
  xhr.send();
}

checkWeGovNowSession(function(result) {
  if (result === undefined) { // note === to distinguish undefined from null
    window.alert("Error during request")
  } else if (result) {
    window.alert("Web browser claims that a user with the following ID is logged in: " +
result);
  } else {
    window.alert("Web browser claims that no user is logged in.");
  }
});
```

We encourage you to test this function with several web browsers since the CORS feature might behave differently in regard to browser implementation. For those of you who are interested in the technical background, we refer to the following document: https://www.w3.org/TR/cors/

Single sign on with OAuth 2.0

1. application redirects user to central login page

User's web browser sends HTTPS request to UWUM:

GET <api_base>/**authorization**

?response_type=*code* — authorization code flow

&client_id=*app2.wegovnow.eu* — client id

&scope=*identification app1read app2read* — requested scope

&state=*6b8441515be47e72624597280c3cef24* — CSRF protection, random per session

Single sign on with OAuth 2.0

2. users logs in and is redirected back

- login
  - using credentials
  - using 3rd party service
    - e.g. Google, Facebook, Microsoft, ...
- if necessary, an account can be created before login
- afterwards, user is redirected back to application

Single sign on with OAuth 2.0

2. users logs in and is redirected back

User's web browser sends HTTPS request to application:

GET https://app2.wegovnow.eu/oauth_redir CSRF protection

?state=*6b8441515be47e72624597280c3cef24* temporary auth token

&code=*c48c479116ce7209b974577b36b5b48f*

state must be the same as in step 1!

Single sign on with OAuth 2.0

3. application converts token and checks identity

Application sends HTTPS request to UWUM:

POST <api_base>/**token**

?grant_type=*authorization_code* check and convert an auth token

&code=*c48c479116ce7209b974577b36b5b48f* temporary auth token

&client_id=*app2.wegovnow.eu* client id must match with step 1

Note: the TLS client certificate must be provided, in which case the server hostname differs.

Single sign on with OAuth 2.0

3. application converts token and checks identity


```
{
  "access_token": "08592bd818dace8159006ee640a499a",
  "token_type": "Bearer",
  "expires_in": 300,
  "refresh_token": "3564b6c8a18bf559474d2a544eb5b605",
  "member_id": "12345678"
}
```


access token for further API calls

lifetime in seconds

refresh token to get a new access token

the user which has been identified

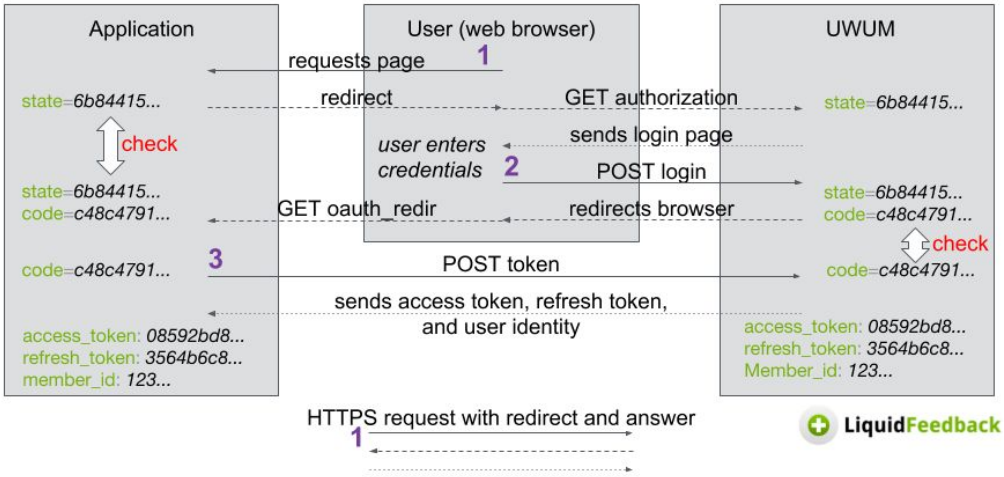
Single sign on with OAuth 2.0

1. application redirects user to login page
- GET authorization (browser → UWUM)

2. users logs in and is redirected back to application
- GET <oauth_endpoint> (browser → application)

3. application converts token and checks identity
- POST token (application → UWUM)

Single sign on with OAuth 2.0



Feature integration: Data embedding

POST <api_base>/**validate**
?access_token=*e8e4a54a0b6ef3c789842b155b18b4ab*

Received access token

```
{
   "scope": "identification app2read",
   "member_id": "12345678"
}
```

Confirmed scope

Confirmed user identity