Work report on Unified WeGovNow User Management (UWUM) development

Jan Behrens, Axel Kistner, Andreas Nitsche, Björn Swierczek

2016-12-12

1 Presentation of UWUM in Berlin

A first draft of UWUM has been presented in the kick-off meeting "Connecting The Bits" on April 14, 2016 in Berlin. The overall idea was to build a single-sign-on (SSO) solution on OAuth 2.0's Authorization Code[1] flow.

For access tokens, the use of bearer tokens[2] was proposed. Furthermore, it was agreed on that TLS is to be used to secure all communication between UWUM and other components.

In addition to single-sign-on, UWUM's capabilities were planned to include:

- a style endpoint, which allows applications to retrieve style information (e.g. a color scheme),
- a navigation endpoint, which allows applications to incorporate a common nagivation bar into their user interfaces, and
- a service discovery endpoint, which allows applications to retrieve a list of other applications within the system and their capabilities/protocols.

This way, WeGovNow is designed to be a modular system that may be extended with different services which are all connected through UWUM.

It was agreed that UWUM will be implemented by LiquidFeedback such that it is possible to use synergetic effects between the necessary creation of an API for LiquidFeedback and the newly created features required by UWUM.

[1]See `https://tools.ietf.org/html/rfc6749#section-1.3.1` for a short overview on the Authorization Code flow and `https://tools.ietf.org/html/rfc6749#section-4.1` for a detailed description.

[2]`https://tools.ietf.org/html/rfc6750`

2 Authentication and Authorization

For reasons of interoperability and security, we aimed to create an implementation that is fully compliant with RFC 6749.[3] In this section, the extensions necessary in addition to that document will be explained below. All functionality has been implemented by the time of publishing this work report except where otherwise noted.

2.1 Roles

RFC 6749 defines several roles in subsection 1.1.[4] The UWUM component as implemented by LiquidFeedback takes the role of the "authorization server". Other WeGovNow components will take the role of "clients" but may also act as "resource server" for other components.

2.2 Choice of protocol flow

UWUM requires the Authorization Code flow[1] for secure user authentication, i.e. when used for single-sign-on (SSO). (Note that subsection 10.16 in RFC 6749 explains why the Implicit flow[5] as defined by OAuth 2.0 is *not* suitable for secure user authentication.[6])

The Implicit flow[5] is still supported for clients which only require authorization but do not rely on secure user authentication (e.g. pure JavaScript clients which access other components but do not store themselves any resources which would need to be protected by SSO).

2.3 Types of clients

RFC 6749 distinguishes between "confidential clients" (which are capable of secure client authentication, e.g. by maintaining confidentiality of their client credentials) and "public clients" (which are incapable of secure client authentication). UWUM requires all clients which use OAuth 2.0's Authorization Code[1] flow (and thus receive long-lasting refresh tokens) to be capable of secure authentication; i.e. every use of the token endpoint (see subsections 2.7 and 2.8) will require client authentication (except when an access token scope downgrade

[3]https://tools.ietf.org/html/rfc6749

[4]https://tools.ietf.org/html/rfc6749#section-1.1

[5]See https://tools.ietf.org/html/rfc6749#section-1.3.2 for a short overview on the Implicit flow and https://tools.ietf.org/html/rfc6749#section-4.2 for a detailed description.

[6]https://tools.ietf.org/html/rfc6749#section-10.16

is performed, see subsection 2.14). The use of "public clients" is only supported for those clients which utilize the Implicit[5] flow because these clients will not handle any long-lasting tokens.

2.4 Client registration

Client registration is mentioned in section 2 of RFC 6749, even though the standard explicitly states that "the means through which the client registers with the authorization server are beyond the scope of [the] specification".[56] UWUM provides two methods of client registration:

- registering clients through the municipality (or their technical administration) or an organization running a particular installation of WeGovNow,
- registration of any other ("dynamic") client on a per-user basis by each user who wishes to use that client to access WeGovNow (machine accessibility).

These two registration methods are described in the following two subsections respectively.

2.4.1 Clients approved by the municipality

Clients approved by the municipality authenticate through TLS (X.509) certificates which are signed by the municipality or a certificate authority acting on their behalf. For example, the operator of the UWUM server could issue a certificate to the operator of each respective client. Furthermore, the operator of the UWUM server configures a list of automatically granted access scopes[7] for the particular client (not every client has the same automatically granted access scopes, e.g. some clients might not require voting rights). Any other access scope may be granted on a per-user basis by the respective end-user or be disallowed by the municipality for a particular client (through white or black lists).

This results in the following information being stored per client:

- name of client,
- OAuth 2.0 client identifier (`client_id`),
- redirect URI(s)[8],
- common name (CN) of the TLS certificate,

[7]`https://tools.ietf.org/html/rfc6749#section-3.3`

[8]See `https://tools.ietf.org/html/rfc6749#section-3.1.2` for redirection URIs. One redirect URI is the default redirect URI, other redirect URIs may be selected through the `redirect_uri` parameter, see: `https://tools.ietf.org/html/rfc6749#section-4.1.1`

- automatically granted scopes[7],
- white list of scopes (optional),
- black list of scopes (optional, i.e. may be empty).

2.4.2 Dynamic clients

For the sake of machine accessibility, it would be nice to allow unregistered clients. Unfortunately, OAuth 2.0 requires some sort of client registration (at least) for the following security reasons:

- allowing capability to authenticate a client,[9] in order
  - to avoid refresh token abuse by a third party in case of accidentially exposed refresh tokens,[10]
  - to avoid authorization code abuse (which could expose access and refresh tokens to a malicious 3rd party) in case of exposed authorization codes,[11]
- restriction of choice of the redirect URI[12], in order
  - to avoid redirection URI manipulation,[13]
  - to avoid open redirector attacks.[14]

In order to be able to provide an open platform, however, it should still be possible to use clients which have not been explicitly approved by the operator of the WeGovNow platform. Assuming there will be more than one WeGovNow installation (e.g. run by different municipalities, each operating their own system), this is necessary in order to enable third parties to provide generic clients that can be used by *any* WeGovNow platform, even those not known to the operator of the client.

Consequently, registration of these clients should happen dynamically without further human interaction.[15] This requires to automatically establish a channel

[9] See `https://tools.ietf.org/html/rfc6749#section-2.3` and `https://tools.ietf.org/html/rfc6749#section-10.1`

[10] `https://tools.ietf.org/html/rfc6749#section-10.4`

[11] `https://tools.ietf.org/html/rfc6749#section-10.5`

[12] `https://tools.ietf.org/html/rfc6749#section-3.1.2`

[13] `https://tools.ietf.org/html/rfc6749#section-10.6`

[14] `https://tools.ietf.org/html/rfc6749#section-10.15`

[15] We assume that every user of WeGovNow is legally entitled to use any client of his or her choice to access his or her data and to perform actions. In cases where a particular operator of LiquidFeedback (e.g. a municipality) wants to decline this right, the use of dynamic clients could be disabled.

of trust between the client and the UWUM server through secure authentication. UWUM relies on the following mechanism to archive secure authentication of a dynamic client:

- a dynamic client is only referenced by its domain, and
- at the choice of each client, registration is performed either
  - by adding a certain entry to the domain's DNS zone[16] or
  - temporarily through a REST API call to the UWUM server with a client-side TLS (X.509) certificate issued to the respective domain and signed by a publicly trusted certificate authority (e.g. "Let's Encrypt")[17].

Taking into account that it cannot be outruled that TLS certificates could accidentially be exposed to a malicious 3rd party and considering that there might be at least one publicly trusted CA which is vulnerable to a state-level attack,[18] we restrict the redirection URI[12] to the following static path on the web server's root level:

```
/liquidfeedback_client_redirection_endpoint
```

This repels any attempts of "authorization code redirection URI manipulation" as explaiend in subsection 10.6 of section 10 ("Security considerations") of RFC 6749 ("The OAuth 2.0 Authorization Framework")[13] even in cases where dynamic client registration could be forged.

Any client that cannot follow the above redirection URI convention must be registered by the municipality or organization running a particular installation of WeGovNow (see subsection 2.4.1).

As an additional security mechanism, the dynamic registration is always done for a set of access token scopes[7] to be used with a particular OAuth 2.0 flow. Thus a client's redirection endpoint registered for the Authorization Code flow cannot be used by the Implicit flow or vice versa unless the registration is broadened accordingly.

[16]A TXT DNS resource record needs to be added to the subdomain "`_liquidfeedback_client`" of the respective domain which must include a so-called magic string (namely "`dynamic client v1`") as first entry.

[17]The operator of LiquidFeedback is therefore required to decide on a list of trusted CA's. Many operating systems already ship with such a list of root certificates.

[18]Note that similar security considerations also apply to DNS and the risk of DNS cache poisoning or similar attack vectors. This could, however, be fixed by DNSSEC such that future versions of UWUM might lift the described restrictions for domains which are cryptographically secured.

The operator (e.g. a municipality) may still decide to disallow the use of non-approved (dynamic) clients completely. This would, however, limit machine accessibility and render the platform less open for extensions and unforseen use cases. An appropriate configuration option will be provided which can also be used to limit the access token scope of dynamic clients (using a white or black list).

Unless dynamic clients are entirely disabled, an additional security warning will be displayed to the user when authorizing such a client. The user will be requested to verify that:

- the client domain is trustworthy,
- the client domain is used to host a legit application to access LiquidFeedback,
- the spelling of the domain name (whose client is going to be authorized) is correct,
- the granted scope of access (access token scope) is intended by the user.

Clients which want to avoid these warnings must be approved by the municipality or organization that is operating the LiquidFeedback system (see subsection 2.4.1).

2.5 Access token types

As previously mentioned, bearer tokens[2] as defined in RFC 6750 will be used as access tokens. Therefore, the access token type ("`token_type`")[19] returned by UWUM is always set to "`bearer`".

2.6 Access token scopes

The following set of generic[20] access token scopes[7] has been specified:

`authentication`: Authenticate the current user by reading its unique static ID and current screen name.

[19] `https://tools.ietf.org/html/rfc6749#section-7.1`

[20] Application specific scopes could be introduced if they turn out to be necessary in the future. It would also be thinkable for dynamic clients acting as a resource server to provide a set of application specific scopes as part of their registration. Further security analysis would be required for such an extension. See also subsection 5.8 for considerations on generic versus application specific scopes.

`identification`: Identify the current user by reading its unique identification string. Automatically implies scope "`authentication`".

`notify_email`: Read the notification e-mail address of the current user.

`read_contents`: Read any user generated content (without authorship, ratings and votes).

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

`read_ratings`: Read ratings (see scope "`rate`" below) by other users.

`read_identities`: Read the identities (identification strings) of other users.

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

`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 (e.g. vote on an issue in LiquidFeedback)

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

`settings`: Read current user's settings (e.g. notification settings, display contrast, etc).

`update_name`: Modify user's screen name.

`update_notify_email`: Modify user's notification e-mail address.

`update_profile`: Modify profile data (e.g. phone number, self-description, etc).

`update_settings`: Modify user settings (e.g. notification settings, display contrast, etc).

Note that any of these scopes can also be suffixed with "`_detached`" to request the scope for usage also when the user is not logged in (which will be explained in subsection 2.9).

2.7 User authentication (single-sign-on)

OAuth 2.0 by itself is not suitable for user authentication. Both the Authorization Code flow[1] and the Implicit flow[5] can be extended to provice user authentication and thus allow to implement a single-sign-on (SSO) system. Because the Implicit flow would require additional security mechanisms to be implemented at client side (where bad implementations result in security vulnerabilities),[6] UWUM extends the Authorization Code flow for the purpose of implementing an SSO solution as described in the following.

In order to protect against authorization code substitution attacks, the UWUM server checks the OAuth 2.0 client identity before accepting an authorization code.[21] This is both a requirement stated in subsection 4.1.3 of RFC 6749 ("The OAuth 2.0 Authorization Framework")[22] and a recommended countermeasure to avoid authorization code substitution attacks in subsection 4.4.1.13 of RFC 6819 ("OAuth 2.0 Threat Model and Security Considerations")[23].

The Access Token Response[24] of the OAuth 2.0 Authorization Code flow gets extended with the field "`member_id`" which returns the LiquidFeedback member ID of the signed-in user. OAuth 2.0 clients not aware of this extension are requested to ignore this field as stated in subsection 5.1 of RFC 6749.[25] Nonetheless, these clients may still pass the returned access token to the validate endpoint (see next section) in order to determine the `member_id` of the user who has logged in.

2.8 Endpoints

RFC 6749 defines two endpoint URIs at the authorization server side: the "authorization endpoint"[26] and the "token endpoint"[27]. These are defined as follows:

- `https://`*server_name*`/api/1/authorization` (GET)
- `https://`*server_name*`/api/1/token` (POST)[28]

Note that a base path may be appended to the *server_name* component if applicable.

[21]Note that, if the client is authenticating with the UWUM server, the `client_id` parameter can be ommitted by the client when accessing the token endpoint (see next footnote).

[22]`https://tools.ietf.org/html/rfc6749#section-4.1.3`

[23]`https://tools.ietf.org/html/rfc6819#section-4.4.1.13`

[24]`https://tools.ietf.org/html/rfc6749#section-4.1.4`

[25]`https://tools.ietf.org/html/rfc6749#section-5.1`

[26]`https://tools.ietf.org/html/rfc6749#section-3.1`

[27]`https://tools.ietf.org/html/rfc6749#section-3.2`

[28]The server name for the token endpoint may differ for those requests where TLS client certificates are used. See subsection 5.2 for explanation.

RFC 6749 does not specify any method for a resource server to "ensure that an access token presented to it by a given client was issued to that client by the authorization server".[29] Therefore, an additional validation endpoint has to be specified:

- `https://`*server_name*`/api/1/validate` (POST)

The validation endpoint does not require any parameters except the access token (bearer token) to be passed using the mechanisms described in section 2 of RFC 6750.[30] It returns a JSON object with the following fields:

- `scope`: a space separated list of scopes[7] associated with the access token (with any "`_detached`" suffix stripped off, see next subsection 2.9),
- `member_id`: an integer set to the id of the user who logged in,
- `logged_in`: a boolean set to false if the user has meanwhile logged out.

Note that the scope of an access token may change when the user logs out. This is explained in the following subsection 2.9. Subsection 2.12 will pick up the issue of user logout again.

There may be situations where an OAuth 2.0 client wants to check whether a user is currently logged in without actually forcing the user's web browser to perform a login if no user was logged in. To provide this functionality, a 4th endpoint (also out of scope of the OAuth 2.0 specification) is added at the authorization server side:

- `https://`*server_name*`/api/1/session` (POST)

This "session" endpoint can be accessed directly by a user's web browser (through a script performing a CORS[31] HTTP request with credentials). Its usage is further explained in subsection 2.10.

2.9 Binding lifetime of access and refresh tokens to a users web session by default

Access tokens have an expiry time after which they will be invalidated.[32] In addition to the maximum access token lifetime returned in the Access Token Response,[24] UWUM additionally limits the lifetime of both access tokens and

[29]See section 10.3 of the RFC: `https://tools.ietf.org/html/rfc6749#section-10.3`
[30]`https://tools.ietf.org/html/rfc6750#section-2`
[31]Cross-origin resource sharing, see `https://www.w3.org/TR/cors/`
[32]`https://tools.ietf.org/html/rfc6749#section-5.1`

refresh tokens to the user's web session at the LiquidFeedback (UWUM) server by default (i.e. if the user logs out, the access tokens and refresh tokens will be immediately invalidated).

Some clients, however, require access longer than the user's login session. For this purpose, access token scopes[7] with the suffix "`_detached`" may be requested (e.g. "`vote_detached`" instead of "`vote`"). Whether an application may request these scopes (as well as which scopes may be requested for detached access) depends on the configuration for the particular client, or – in case of dynamic clients – on the configuration for all dynamic clients. An access or refresh token that contains only detached scopes will not be invalidated on user logout. Access tokens, however, will still be invalidated when their expiry time (as denoted by the "`expires_in`" field in the Access Token Response[24]) has elapsed, in which case a refresh token must be used to obtain a new access token. Access and refresh tokens which contain both detached and non-detached scopes will only have their non-detached scopes removed on user logout instead of being invalidated completely.

Other than the behavior described above, the "`_detached`" scopes behave as any other scope for the authorization[26] and token[27] endpoint. Only the validation endpoint ("`api/1/validate`") will strip the suffix "`_detached`" from the scope field in its response because it doesn't matter for a validating resource server whether a scope has been granted detached from a web session or not.[33]

Even if the token lifetime is bound to the web session (i.e. when only non-detached scopes are requested), a user's logged in web browser may still automatically re-authorize the client whenever he or she is logged in at UWUM and visits the client's website. If such a client was authorized by the user, the permission can be revoked by the user at any time using a designated configuration dialog provided by the UWUM server.

2.10 Checking user login without triggering a login

An interactive UWUM client application may want to determine whether a user is logged in without actually triggering a login. OAuth 2.0 does not provide such a mechnaism on its own.[34] UWUM therefore provides an additional "session" endpoint (`https://`*server_name*`/api/1/session`, see subsection 2.8) to allow

[33]Neither RFC 6749 nor RFC 6750 are violated because the authorization and token endpoint treat detached scopes like any other scope and a validation endpoint is not covered by these RFCs.

[34]Also, extending the authorization endpoint by accepting a "prompt" parameter as done by OpenID Connect is not feasible for user-registered clients because non-logged-in users could be redirected to malicious clients registered by other users, making the system susceptible to open redirector phishing attacks. See subsection 5.5

web applications to gather information about the current login status of a user without actually triggering any (interactive) login or permission grant procedure. This endpoint is directly accessed by the user's web browser through an XMLHttpRequest (XHR) call while setting the "`withCredentials`" option of the XMLHttpRequest object to true.

The call does not need any parameters and should not have any additional request headers set[35]. It returns a JSON object with the "`member_id`" attribute set to the ID of the current user (or to null if there is no logged-in user or if a user-registered client is not authorized to obtain the login status). Since the request is done by the user's web browser, the answer is *not* authoritative for the UWUM client and must only be used as a hint. **A returned user ID MUST still be confirmed via the regular OAuth 2.0 procedure using the authorization endpoint!** In this case, the authorization endpoint will not show a login window (because the user is already logged in).[36]

2.11 Caching the login state

A successful user authentication could be cached in the session store of the UWUM client (usually at the web server side in conjunction with a cookie). This, however, can create confusion for the user because he or she might show up as being logged into the system after having logged out or vice versa. A possible solution is to use the "`session`" endpoint as discussed in the previous subsection 2.10 through a JavaScript which then notifies the server side of the UWUM client by redirecting the web browser if a reconfirmation of the user's login status is necessary.

In either case, UWUM clients should reconfirm that the user has not logged out at least immediately before any state changing request (e.g. posting, rating, voting, etc.) by using the validation endpoint (see subsection 2.8). This check cannot be done directy by the web browser due to security reasons (as also explained in the previous subsection 2.10).

[35]Not setting additional request headers avoids CORS pre-flight requests, see `https://www.w3.org/TR/2014/REC-cors-20140116/#cross-origin-request-with-preflight-0`

[36]There is a chance for a race-condition if the user simultaneously logs out. This could be solved by returning an authorization code through a CORS call. However, implementation of such a protocol is out of scope for WeGovNow and would require further security analysis.

2.12 Logout

2.12.1 Checking for logout

As explained in subsection 2.9, an access or refresh token is automatically invalidated on logout if only non-detached scopes have been requested. For all other cases, the "`logged_in`" boolean field returned by the validation endpoint (see subsection 2.8) may be used to detect a logout by the user.

The "session" endpoint (as further explained in subsection 2.10) may also be used to check whether a user might have logged out (without consuming much resources on the server-side of the UWUM client).[37] Note, however, that a request from the web browser to the session endpoint is not suitable for the UWUM client application to validate that a user is really logged in or to securely confirm that his or her session has really ended (see subsection 2.10).

2.12.2 Performing logout

Depending on design criteria, logout could be performed either

- through a direct link in the UWUM navigation bar or
- through a link in the UWUM navigation bar which leads to a user page where there is a second link for the actual logout procedure.

Technical implementation requirements differ for these two cases. In the first case, the logout is performed in the context of any UWUM client; while in the second case, the final logout link or button can be displayed in the context of a web page returned by the UWUM server (which is a different origin). Due to protection against cross-site-request-forgery (CSRF), an appropriate access token or dedicated logout token would need to be part of the link in the first case (the case of using a direct link for logout). In this case, an appropriate OAuth 2.0 access token scope would need to be added to avoid unwanted exposure of the logout token (or an access token with respective scope).

A decision on this issue has not been taken yet; user interface design considerations and technical security considerations should determine which of the discussed two approaches is more suitable. Also refer to subsection 5.10, which discusses certain design limitations due to privilege separation.

[37] A future extension of UWUM could also allow UWUM clients (or their JavaScript components at the web browser side) to issue a request which is held open by the UWUM server for a set amount of time in order to allow pushing a change of the user's login status just-in-time (see also subsection 5.4).

2.13 Requesting several access token scopes at once

To avoid unnecessary delays, a client may (as an extension to RFC 6749) request several access token scopes[7] (i.e. sets of access ranges) at once by using the parameters "`scope1`", "`scope2`", etc. in the Authorization Request. The corresponding result parameter "`access_token`" will have "1", "2", etc. appended to its name (e.g. "access_token1" etc.). Note that counting must start with "1". It is, however, allowed to include an optional non-numbered "`scope`" parameter in addition to "`scope1`", "`scope2`", etc. The result parameters "`token_type`" and "`expires_in`" are never numbered or duplicated due to size limitations in the Implicit flow (maximum URL length) but always relate to all returned access tokens.

The described behavior of this subsection is not part of OAuth 2.0. Using this extension is entirely optional for the client.

2.14 Downgrading access token scopes

As an extension to RFC 6749, the token endpoint has been extended in such a way that it can be used to downgrade access token scopes. This feature is important for meta-APIs because according to RFC 6749, the only way to obtain a new access token without the user's web browser is to provide a refresh token to the token endpoint.[38] Refresh tokens, however, are bound to a particular client and must not be shared by the client with any other party but the authorization server.[39]

A meta-API might receive an access token with a broader scope[7] than the scope necessary for calls made by the meta-API provider to another resource server. Using a greater scope than necessary for calls to resource servers, however, weakens the overall security of the system. In order to allow meta-API providers to downgrade the scope prior to using the access token, the token endpoint[27] accepts the string "`access_token`" as value for the "`grant_type`" parameter, which will tell the UWUM server that an access token (and not an authorization code or refresh token) is being presented to receive a new access token with a downgraded scope. The access token has to be provided according to the rules stated in section 2 of RFC 6750,[30] and one or more scopes must be requested through the "`scope`", "`scope1`", etc. parameters (see subsection 2.13 for details on requesting several scopes at once). Client authentication is not required. The old access token with the broader scope will not be invalidated and may still be used in future requests (e.g. to receive another access token with a different scope).

[38]`https://tools.ietf.org/html/rfc6749#section-6`

[39]`https://tools.ietf.org/html/rfc6749#section-10.4`

For security reasons, downgrading an access token scope will never extend the token lifetime, i.e. the returned access token will have the same remaining maximum lifetime than the access token presented to the token endpoint.[40]

2.15 Additional measures to prevent refresh token abuse

Conforming with section 10.4 of RFC 6749,[41] the UWUM server (LiquidFeedback) ensures that refresh tokens are bound to the client they have been issued to. As also suggested in subsection 10.4 of RFC 6749, further means to restrict refresh token abuse are implemented. Refresh tokens are replaced periodically and using a refresh token invalidates the corresponding scope[7] of all other previously issued refresh tokens, with the exception that refresh tokens which are still bound to a logged in user are unaffected.[42] An additional grace period avoids problems due to race conditions or aborted connections. This approach is similar to the example given in subsection 10.4 of RFC 6749 while being resistant against accidental race-conditions or connection aborts and allowing for a more flexible usage (e.g. different subsystems of the same client may store different refresh tokens indepenently).

2.16 Required CORS support for resource servers

Because RFC 6750 requires bearer tokens[2] to be accepted through the HTTP header "`Authorization`",[43] and because the "`Authorization`" header is not in the list of "simple response headers" as defined by the W3C recommendation on cross-origin resource sharing,[44] it is inevitable for all resource servers to support cross-origin resource sharing (CORS) with the respective "`Access-Control-Allow-Headers`" option[45] set to be able to fulfill the requirements of RFC 6750. Every UWUM component acting as a resource server should therefore enable and configure CORS accordingly. See `https://www.w3.org/TR/cors/` for details.

[40]This is the reason why client authentication would not grant any extra security here and thusly can be omitted.

[41]`https://tools.ietf.org/html/rfc6749#section-10.4`

[42]This is implemented by downgrading "_detached" scopes to their corresponding non-detached scopes.

[43]`https://tools.ietf.org/html/rfc6750#section-2.1`

[44]`https://www.w3.org/TR/cors/#terminology`

[45]`https://www.w3.org/TR/cors/#access-control-allow-headers-response-header`

2.17 HSTS

We recommend to use HTTP Strict Transport Security (HSTS)[46] for all WeGov-Now components to increase security.

3 Additional endpoints for integration

Beyond user authentication and authorization, three more API endpoints are being defined for backend and UI integration:

- a "`navigation`" endpoint to incorporate a navigation bar,
- a "`style`" endpoint to retrieve style information, and
- a "`client`" endpoint for applicaton and service discovery.

Prototypes for the navigation and style endpoint have been implemented; the client endpoint for application and service discovery is currently only a stub.

3.1 Navigation endpoint

In order to integrate all WeGovNow applications in such a way that they look and feel like a single application, all WeGovNow applications share a common navigation bar. The "`navigation`" endpoint of the UWUM server returns this navigation bar to be included by each WeGovNow application. This way, modifications to the navigation bar can made at a central place without the need to change every single application.

Either a login button or the user name with a link to a user page (where logout is possible) is included in the navigation bar, depending on whether an access token is provided when calling the endpoint.[47] For the login button, an alternative URL may be provided by the caller of the navigation endpoint. This login URL may either be the authorization endpoint of the UWUM server with an appropriate "`state`" HTTP GET parameter included (note that the value must be percent-encoded[48]) or an URL provided by the UWUM client which initiates the OAuth 2.0 authorization and authentication procedure as described in section 2 of this document. Alternatively a unique placeholder (e.g. a GUID)

[46] `https://tools.ietf.org/html/rfc6797`

[47] Also a dynamic popup menu is thinkable. However, issues with JavaScript and privilege separation in case of animated submenus according to Material Design require further consideration. Refer to subsection 5.10 in that matter.

[48] `https://tools.ietf.org/html/rfc3986#section-2.1`

can be passed as login URL to allow caching of the rendered navigation bar and replacing the login URL locally at the UWUM client.[49]

Whether a structured JSON document or a pre-rendered HTML snippet is returned can be selected by another parameter passed to the navigation endpoint. A pre-rendered HTML snipped may be either returned encapsulated in a JSON response or raw for usage with the HTML5 include tag.

When the "`client_id`" parameter is provided to the navigation endpoint, the corresponding client tab gets highlighted (or marked as active in case of the structured JSON document response).

It is planned to collapse the navigation bar on small screens. This feature might interfere with application specific menus; refer to subsection 5.12 for that matter.

3.2 Style endpoint

The style endpoint provides basic color definitions for a primary and an accent color as 8-bit RGB triplet to be able to customize the unified visual look of all WeGovNow applications for a particular installation by central configuration. Additional colors can be derived from these two base colors. If the UWUM server gets configured with colors from the Material Design color palette, the corresponding Material Design color name of the primary and the accent color is also provided.

3.3 Endpoint for application and service discovery

The endpoint "`client`" is supposed to return a list of all system applications and, if an access token is provided, a list of all registered dynamic clients for the corresponding user. Implementation of this endpoint will require storing the base URL of all system applications at the UWUM server.

Further discussion with OntoMap is required for specification and implementation of this endpoint.

4 Test platform

A test platform has been created in mid September to start integration with the other consortium partners.

[49]Note that the characters "<", ">", "`&`", as well as the quotation mark character should be avoided in a placeholder string because these characters would get HTML entity encoded as described in subsection 8.1.4 of the HTML5 standard, see: `https://www.w3.org/TR/html5/syntax.html#character-references`

4.1 Benchmarks

The following benchmarks for integration have been defined, whose fulfillments have been published in the weekly status reports.

Client URLs established A client application (resource server and/or relying party) has been installed and its base URL and redirection endpoint[50] has been communicated to the consortium.

SSL key and certificate for end-users A private key and a publicly trusted SSL certificate has been created for the end-user web interface and SSL connections to that interface have been successfully tested.

Certificate signing request (CSR) for UWUM API A private key for accessing the UWUM API and a corresponding certificate signing request (CSR) has been created and submitted to FlexiGuided GmbH (LiquidFeedback).

SSL certificate for UWUM API A signed certificate for the UWUM API client key has been sent back to the consortium member and their client application has successfully established a secured connection with the UWUM server.

Authorization endpoint accessed The client application can redirect an end-user to the UWUM authorization endpoint.[51]

Authorization endpoint error response handling The client application is capable of receiving authorization errors[52] through its redirection endpoint[50] and displaying it to the end-user.

Access token request (including end-user identification) The client application has successfully received an authorization code and identified the end-user through an access token request.[53]

Access token request error handling The client application is capable of properly processing errors during the access token request.[54]

Using access tokens for API calls to other components The client application has successfully used an access token to perform a LiquidFeedback API call.

[50]`https://tools.ietf.org/html/rfc6749#section-3.1.2`
[51]`https://tools.ietf.org/html/rfc6749#section-4.1.1`
[52]`https://tools.ietf.org/html/rfc6749#section-4.1.2.1`
[53]`https://tools.ietf.org/html/rfc6749#section-4.1.3`
[54]`https://tools.ietf.org/html/rfc6749#section-5.2`

Access token verification The client application is capable of verifying the validity and scope of an access token.

Accepting access tokens from other components The client application provides at least one API call where an access token is used for authorization.

Accepting access tokens as “Authorization” header In conformance with RFC 6750 (Bearer Token Usage), the client application (resource server) accepts access tokens through the authorization request header field.[55]

Cross-origin resource sharing The client application allows cross-origin resource sharing (CORS) as described in subsection 2.16 of this document.

Cross-application navigation The UWUM navigation bar has been successfully integrated into the client application.

IPv6 IPv6 capabilities have been tested.

5 Technical challenges

In this section, we will describe obstacles encountered during implementation and during integration with the consortium partners as well as respective solutions.

5.1 Third party clients (non-registered clients vs. dynamic registration)

OAuth 2.0 demands client registration but does not specify how such client registration is to be implemented.

> “Before initiating the protocol, the client registers with the authorization server. The means through which the client registers with the authorization server are beyond the scope of this specification but typically involve end-user interaction with an HTML registration form.”[56]

Manual client registration, however, is only suitable for a service-centered approach where a software provides only a single service (e.g. Facebook, Google, Twitter, etc). An open source solution, however, could be installed at several sites by different service providers. It is therefore not sufficient to register a client

[55] `https://tools.ietf.org/html/rfc6750#section-2.1`

[56] Ed. D. Hardt: The OAuth 2.0 Authorization Framework, October 2012. Section 2 (Client Registration), `https://tools.ietf.org/html/rfc6749#section-2`

at a single service provider if this client shall be usable for any service provider using the UWUM server software.

One possible solution would be the creation of a central (i.e. world-wide) UWUM client registry. Such central client registry, however, could be a single point of failure and would empower a central authority to control usage of the UWUM protocol (e.g. it would be possible to block certain clients). We consider this approach contrary to the concepts of open source and open data.

Therefore, we implemented a dynamic client registration protocol that keeps implementational complexity at a minimum while providing good security properties which outperforms many other solutions for client registration due to requiring direct access to the DNS zone of the domain (for adding a TXT record) or credentials (a publicly trusted TLS certificate with corresponding key) that should be accessible only by the domain owner. Dynamic client registration is described in subsection 2.4.2 of this document.

5.2 TLS client side certificates and web browser behavior

Web server software often offers three different settings for handling TLS client certificates:

- client-side certificates disabled,
- optional client-side certificate,
- mandatory client-side certificate.

Often these settings can be made only on a per-domain basis (i.e. for each virtual host). Furthermore, enabling client-side certificates (even if set to "optional") will cause web browsers to show up a dialoge when accessing pages on that domain.

For these reasons, a separate hostname has to be used for API endpoints when a TLS client-side certificate is to be provided (which affects the token endpoint[27]). The UWUM server will have to provide a configuration endpoint where dynamic clients may retrieve a deviant domain for the token endpoint; and dynamic UWUM clients (see subsection 2.4.2) will have to query this configuration endpoint prior to using the token endpoint).

5.3 Multi-domain certificates

TLS certificates may be issued for more than one domain using the "Subject Alternative Name" (SAN) extension. The current implementation of LiquidFeedback, however, relies on an HTTP reverse proxy to include the distinguished name (DN)[57] of the certificate in a designated HTTP header. Some reverse proxy software, namely "NGINX" which is recommended for use with LiquidFeedback, does not properly support transmitting a domain list from the SAN extension. In case of "NGINX", header line folding[58] is used to pass multiple domain names from a TLS certificate to the respective backend (e.g. LiquidFeedback). Header line folding, however, has recently been deprecated by RFC 7230,[59] and it is not supported by LiquidFeedback (and not even by "NGINX" for incoming requests). The problem of header line folding in the context of multi-domain TLS certificates has also been discussed in the "NGINX" issue tracker under ticket #857.[60] The issue is currently not classified as bug[61] and it is unclear when a patch will be incorporated into the software.

For the technical difficulties explained above, we refrained from supporting multi-domain certificates at this stage. In case of UWUM clients approved by the municipality or operator of LiquidFeedback, this shouldn't be a problem anyway because the certificate authority will be under the control of the operator, such that it is easy to create a certificate using the DN/CN property. For dynamically registered clients, an alternative mechanism using DNS TXT records is available (see subsection 2.4.2).

If multi-domain certificates are supported in the future, it is vital that the token endpoint requires the "`client_id`" parameter to be set for all clients authenticating with such a multi-domain certificate. This way, code substitution attacks[23] can be repelled. (Note that RFC 6749 requires the "`client_id`" parameter to be set only if the client is not authenticating with the authorization server;[22] but this does not work for multi-domain certificates.)

5.4 Outdated logins

While a successful OAuth 2.0 authorization procedure (using the Authorization Code flow[1]) can be used to confirm that a user is logged in at the particular time of the Access Token Response[24], an UWUM client obviously can't assume that the login will be still valid at any later time.

UWUM currently provides two methods to check if a user has logged out; these are explained in subsection 2.12.1 of this work report. Considerations in regard to purposeful caching of the user's login status are found in subsection 2.11.

Even if no caching of the login status is performed, there is still the possibility that a user opens WeGovNow with two different browser windows or browser tabs.

[57]The DN contains a single domain as CN (common name).

[58]`https://tools.ietf.org/html/rfc2616#section-2.2`

[59]`https://tools.ietf.org/html/rfc7230#appendix-A.2`

[60]`https://trac.nginx.org/nginx/ticket/857`

[61]`https://trac.nginx.org/nginx/ticket/857#comment:2`

He or she might then log out in one window and afterwards switch to the other window where the logout has not been noticed yet, which creates confusion for the user. A possible solution is to regularly check if the user has logged out by utilizing the cross-origin-resource-sharing (CORS) XML-HttpRequest (XHR) as explained in subsection 2.10.

Regular requests to detect logouts, however, cause unnecessary resource consumption for all involved components. A better approach would be to have a permanent TCP connection between the web browser and the UWUM server (or alternatively between the UWUM client and the UWUM server if at the same time there is a permanent connection between the web browser and the UWUM client). There are different technologies thinkable for this approach. One method is to keep an XML-HttpRequest open for a set amount of time during which the server is capable of sending a message directly to the web browser. (The request has to be repeated after timeout or after a message has been received, whichever happens first.) Another technique would be to use WebSockets. None of these additional techniques have been implemented yet.

5.5 Susceptibility to open redirector phishing attacks when allowing login checks through web browser redirection

Subsection 2.10 mentioned that an interactive UWUM client application may want to determine whether a user is logged in without actually triggering a login. OAuth 2.0 does not provide such a mechnaism on its own, and our research concluded that any form of redirection-based mechanism for providing this functionality[62] would be susceptible to open redirector phishing attacks as described in subsection 4.2.4 of RFC 6819 ("OAuth 2.0 Threat Model and Security Considerations")[63] as long as third parties are capable of registering a malicious client with a corresponding redirection URI[12] that is under the control of the third party.

The previously mentioned subsection of the threat model and security considerations document (RFC 6819) suggests client registration with redirect URI registration (and avoiding redirects to any non-registered redirect URI)[64] as only countermeasure for this threat. However, this countermeasure only works when manual client registration (and manual approval through the operator of the UWUM server) is mandatory. It particularly fails if dynamic client registration (e.g. as described in subsections 2.4.2 and 5.1 of this work report) is allowed.

[62]e.g. accepting a "prompt" parameter as done by OpenID Connect, see `http://openid.net/specs/openid-connect-core-1_0.html#AuthRequest`

[63]`https://tools.ietf.org/html/rfc6819#section-4.2.4`

[64]`https://tools.ietf.org/html/rfc6819#section-5.2.3.5`

Luckily, the technique of cross-origin resource sharing (CORS) allowed for the development of an alternative to the redirect-based approach. Subsection 2.10 explains the mechanism.

5.6 Handling of updated user related data (e.g. user's e-mail addresses)

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

For that purpose, another API endpoint `/api/1/notify_email` (GET) can be used (using an access token with the "`notify_email`" scope). To be able to retrieve the e-mail 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.[65]

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 e-mail address. Such properties could be cached for a limited time before retrieving them again from the application storing this property.

5.7 Race conditions with refresh token rotation

As suggested in subsection 10.4 of RFC 6749,[41] refresh token rotation is employed to provide better security properties (e.g. in case of exposed refresh tokens and client certificates, or in case of the existence of a single compromized certificate authority which would render authentication of dynamic clients insecure).

Unfortunately, RFC 6749 does not specify how old refresh tokens are invalidated. Section 6 of RFC 6749 only says that[66]

- the authorization server MAY issue a new refresh token, in which case
- the client MUST discard the old refresh token and replace it with the new refresh token, and

[65]Note that when exchanging a refresh token for an access token after the user has been logged out, an UWUM client must also explicitly request the "`*_detached`" scope(s) it needs, e.g. "`notify_email_detached`" using the scope parameter of the `/api/1/token` endpoint.

[66]`https://tools.ietf.org/html/rfc6749#section-6`

- the authorization server MAY revoke the old refresh token.

Always revoking the old refresh token after transmission can have a bad effect on system stability, considering that responses might be interrupted. Furthermore, multiple backends of an UWUM client could simultaneously access the token endpoint. Such legit accesses by two legit backends of the same client would need to be distinguished from accesses by a legit client and a malicious third party who obtained a copy of a refresh token.

Subsection 2.15 explains the mechanisms employed by the UWUM server to mitigate the risk of refresh token abuse while solving the problems stated above.

5.8 Creating a set of suitable access token scopes

A useful set of access token scopes[7] is a vital aspect of privilege separation. From a security point of view, scopes should be as fine-graded as possible, particularly there should be different scopes for different applications (e.g. an application that wishes to rate user contributions in application X does not need an access token that allows to rate user contributions in application Y). Extensibility, on the other hand, would be complicated if access token scopes always refer to a single application (i.e. a single resource server in this context). Furthermore, it is a goal that the WeGovNow platform looks and feels like a single integrated application. When users grant access scopes to third party clients, such application-based scopes would be difficult to understand for the user, which by itself can have bad influence on the overall system security.

We therefore decided to provide a set of generic access token scopes as listed in subsection 2.6. For future extensions, see footnote 20.

5.9 Misconceptions regarding scopes vs. user privileges

Scopes must not be mistaken for user privileges. I.e. a 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. For example, 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.

Programmers of UWUM clients must keep these differences in mind and execute an action only if both the scope and the users privileges are sufficent for the respective action.

5.10 JavaScript integration and privilege separation

Dynamically sharing JavaScript code between UWUM clients or between the UWUM server and an UWUM client violates privilege separation because it would enable one component to execute code in the security context of another origin. For example, one application 'A' could send a harmful JavaScript to be included in a web page returned by another application 'B' which then discloses the session cookie for application 'B' to application 'A'.

For this reason, the common navigation bar as returned by the navigation endpoint (see subsection 3.1) currently does not include any JavaScript code. UWUM clients may therefore even consider to sanitize the returned HTML code in such a way that any JavaScript is removed or rejected.

Interface design decisions, however, might suggest to use JavaScript for the navigation bar. Material design, for example, requires popup-menus to be animated, which cannot be done with CSS alone. Another reason for JavaScript might be dynamic modifications of the navigation bar (e.g. collapsing the navigation bar to a menu icon) depending on the screen size or the device of the user. Also other integration techniques might suggest the use of JavaScript.

An alternative to dynamically provided JavaScripts by the UWUM server would be a common library to be included locally by each WeGovNow component. Whenever this library is updated, administrators of each component can look over it before incorporating it. While this approach provides proper privilege separation, its downside would be the administrative overhead.

At least in regard to the navigation bar, it would eventually need to be decided whether

- there will be no JavaScript used by the navigation bar,
- the UWUM server will dynamically return JavaScript code for the navigation bar, or
- each WeGovNow component needs to include a pre-distributed JavaScript.

5.11 Logout through navigation bar

The common WeGovNow navigation bar (as returned by the "`navigation`" endpoint, see subsection 3.1) should also include a possibility to logout. Due to protection against cross-site-request-forgery (CSRF) and because the navigation bar will be included in responses from different web servers (different "origins"), a simple logout link does not work. Subsection 2.12.2 deals with different approaches to this problem.

5.12 Collapsing navigation bar and application menu

In case of mobile devices, it may be desirable to collapse the navigation bar to a single menu icon displayed in the corner of the screen. Despite the technical problems in regard to JavaScript (which are discussed in subsection 5.10), there is also a challenge in regard to a potentially existent second menu bar which is provided by the particular application currently selected.

It could be difficult for the user if two menu icons are being displayed (i.e. a meta-menu, which covers the entries of the navigation bar, and an application specific menu). A potential solution could be to combine both menus into a single one. In this case, however, the considerations of subsection 5.10 still apply.

5.13 UWUM clients without user interface

In addition to UWUM clients having a user interface, there are also WeGovNow applications thinkable which do not have any (end-)user interface. This includes both meta-API providers as well as other service components. In the context of WeGovNow, one meta-API provider could be OntoMap.

The current UWUM specification enables the development of meta-APIs because access tokens are not bound to a particular UWUM client and can be downgraded in regard to their access token scope (of which the latter is important for security, see subsection 2.14). Thus, a meta-API can simply require its callers to provide a valid access token which then can either be used directly or downgraded for further requests performed by the meta-API provider to other resource servers.

Nonetheless, the mechanisms described in this work report still require privileges that are bound to a particular user. For UWUM clients requiring access privileges that are not tied to a particular user (e.g. clients which aggregate data of all users and publish that information), the Client Credentials Grant[67] should be implemented.

5.14 Client authentication for resource servers

While UWUM enables (a) its clients to authenticate users and (b) resource servers to verify user authorization (both explained in section 2), it does not enable resource servers to authenticate clients. Such client authentication might be required by applications that want to establish a trusted channel to another application independently of user authorization.[68]

[67] `https://tools.ietf.org/html/rfc6749#section-4.4`

[68] An example could be OntoMap logging actions executed at other applications (which are then reported to OntoMap by the respective application with client authentication enabled).

Unfortunately, neither OAuth 2.0 nor UWUM enable applications to verify the identity of another application. Even if OAuth 2.0 uses client authentication for a variety of reasons[69] for the authorization endpoint[26], it doesn't provide such an authentication method to other applications. Extending the OAuth 2.0 work flow in this matter (e.g. by returning the `client_id` when an access token is presented to the validation endpoint[70]) would rise some issues:

- Tieing an access token (a bearer token[2] in case of UWUM) to a particular client does not make sense in case of applications that behave both as an OAuth 2.0 resource server and as a client (e.g. meta-API providers or applications which provide an API and have to perform further API calls to complete a requested action). Also, client impersonation would be possible. To give an example: if a received access token is tied to a particular client A, and if application A uses this access token to perform an action at application B, then application B would be able to impersonate application A. Furthermore, application B couldn't use the access token to authenticate as application B when performing further requests at the API of another application C.

- Using a custom scope to identify the origin of a request (e.g. a scope "`I_am_appX`") would also enable client impersonation (e.g. any application who receives an access token with the scope "`I_am_appX`" could then impersonate application X). An alternative could be to use scopes that reflect better the particular action to be performed, e.g. a scope "`write_appXs_log_at_appY`". It is self-evident that this would increase the number of scopes drastically (possibly quadradically), which, in turn, might create a maintenance/configuration mess. Other than that, there is another problem with using scopes for client authentication: following RFC 6750, there can only be one bearer token per request.[2] If a client needs to use a received access token for an API call at another component, then this access token could not be used to authenticate that client because it won't have the necessary scope. One possible solution could be to allow adding scopes to an existing access token or extend RFC 6750 in such a way that multiple access tokens could be used per request. All those solutions, however, go far beyond OAuth 2.0 and would require extra implementation work for all consortium partners. In the end, the created solution wouldn't be OAuth 2.0 anymore.

[69] See beginning of subsection 2.4.2 of this report.

[70] See subsection2.8 for an explanation of the validation endpoint.

The straight-forward way of authenticating clients is to use the existing mechanism already employed by all UWUM clients: TLS client-side certificates. This, however, requires TLS client certificate checking by each resource server that needs to authenticate other clients.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693514.