Security

Dictionary terms

- **Base permission** - basic permission, e.g. `READ`, `CREATE`, `UPDATE`, `DELETE`
- **Group permission** - group permission could contain base permissions. Group is related to some domain object, e.g. `IDENTITY`, `ROLE`
- **Authority** - Group + base permission, e.g. `IDENTITY\_READ`, `ROLE\_CREATE`. Authority is granted to identity by authentication.

API authentication

API access requires the user to be authenticated, excluding a few public endpoints. We can divide the sign in into two parts:

- authentication - the user proves his identity
- authorization - the user has access to given resource

Authentication

Authentication is implemented through a request filterchain. The filters must always follow specified behavior:

- if credentials are OK, continue to authorization
- if credentials do not match, pass request to another filter in chain

In reality there is only one authentication servlet filter - `AuthenticationFilter`. Others filters are Spring beans implementing `IdmAuthenticationFilter` interface. An exception in filters is the `ExtendExpirationFilter`, which is another servlet filter handling the extension of expiration date of JWT tokens. This filter also controls possible exceptions in authentication flow.

API endpoints

There are 2 endpoints for successful user login:

- /authentication - POST, public - authentication, returns JWT token
- /authentication/remote-auth - GET, secured - authentication against external service, returns JWT token and expects the user to be successfully authenticated by one of filters

And one endpoint for user logout:

- /authentication - DELETE - logout with used JWT token (⇒ disable used JWT token)

In case of successful request on the endpoint, the token is passed in HTTP response header. Respose header is called `CIDMST`.

Implemented authentication filters

Filter do not check authorization or user's permissions, just validate the request data, transform these and pass them to Authentication Managers. There are following filters in IdM core (the *core* module):

- BasicIdMAuthenticationFilter
  - HTTP Basic authentication, login, and password are base64 encoded in HTTP header "Authorization" prefixed by "Basic ", according to RFC 2617
  - internal API transforms the login and password and passes them to Authorization Manager
  - enabled by default
- JwtIdMAuthenticationFilter
  - validates IdM JWT token (read as "jot")
  - tokens without expiration are taken as valid
  - HMAC256
  - enabled by default
- SsoIdmAuthenticationFilter
  - used mainly for AD SSO authentication, expects configured web server in front of IdM which handles Kerberos authentication
  - processes the result of the authentication in the HTTP header "REMOTE_USER"
  - **disabled by default**, can be enabled by the configuration property `idm.sec.core.authentication-filter.core-sso-authentication-filter.enabled=true`
- *external authentication server*
  - CzechIdM is ready to authenticate users based on external resource
  - external filters are based on the same principles mentioned above
  - primary purpose is to issue the internal IdM JWT token to user using the remote-auth endpoint
  - filter behaves quite permissively - if user accesses other endpoints then those for authentication, the filter signs the user in for each request and completes the request (if the user is authorized)
- CasIdmAuthenticationFilter
  - @since 10.9.0
  - Used for SSO against CAS
  - CAS need to be configured correctly - see tutorial TODO
  - By default this filter is disabled
  - **Don't override this property on FE, otherwise you can end up without option how to login (if CAS is broken). Enable it only directly in application.properties**
  - Can be enabled by configuration property `idm.pub.core.cas.sso.enabled=true`

Authorization and JWT token

User authorization is checked on the API endpoint layer and enforced by Spring Security. The content of IdM JWT:

- currentUsername - effective user's login
- originalUsername - logged user's login
- currentIdentityId - effective user's ID
- originalIdentityId - logged user's ID

- exp - token expiration date
- iat - issued at date

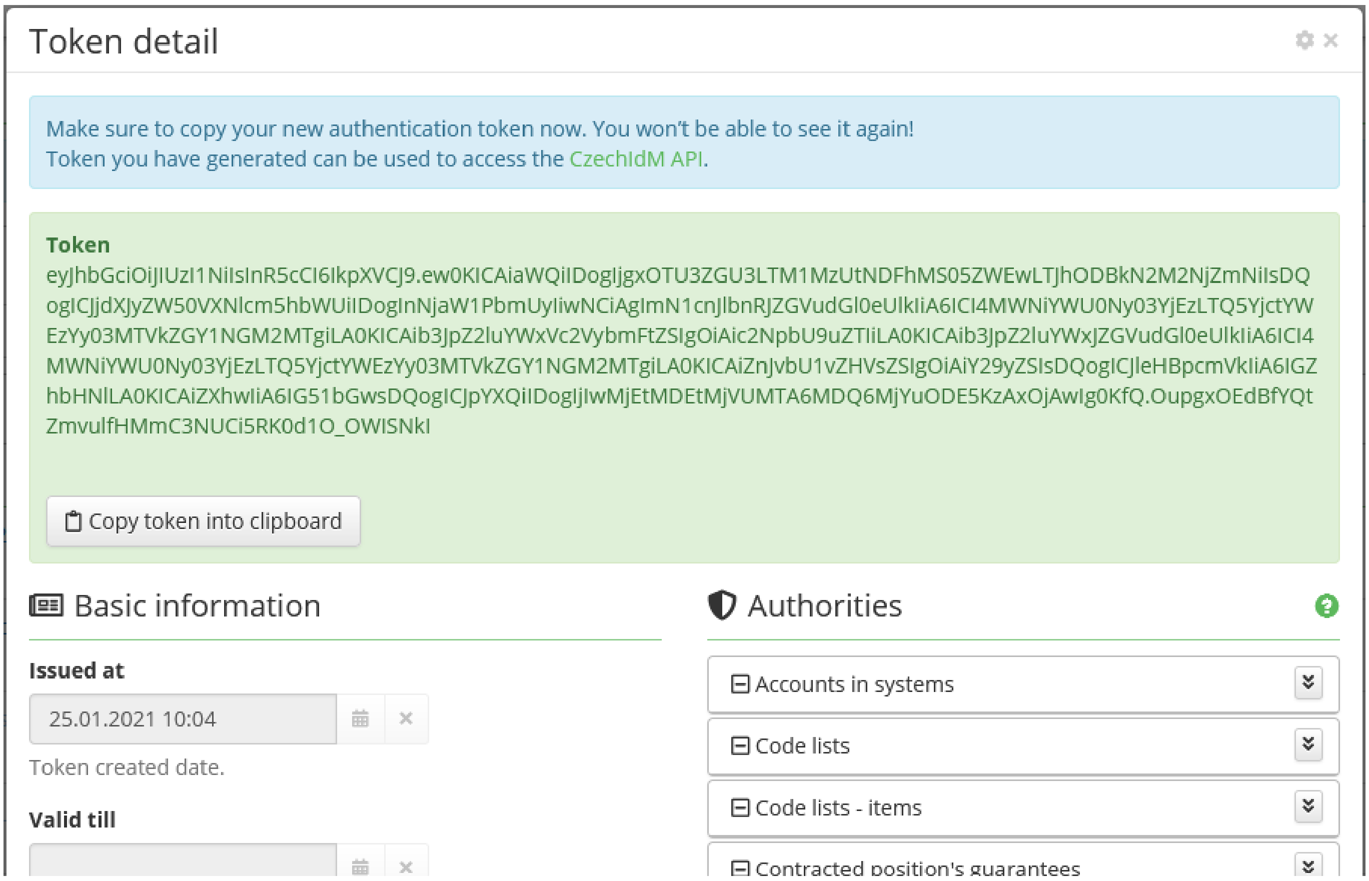
All IdM JWT tokens are signed using HMAC256 algorithm. The symmetric encryption key is configuration property of CzechIdM, stored as "idm.sec.security.jwt.secret.token". Default token expiration time is 10 hours (idm.sec.security.jwt.expirationTimeout). JWT tokens are persisted in database (`IdmToken` entity) with assigned authorities. When identity is logged out, token is disabled. Disabled and expired tokens are purged periodically by internal scheduled task.

Backend of CzechIdM supports immediate detection of user's authorization change. Each modification type is implemented as application event processor, for further details please check the source code and tests :) When user's authorization changes, then persisted tokens, which user owns, are disabled ⇒ user is logged out. Types of modifications:

- removal of role, which carries application permissions ⇒ user losses some permission.
- disabling the user
- role's permissions change - revokes tokens of all users which have the role assigned

How to use JWT token

Token can be used instead of a basic authentication to authenticate to the CzechIdM API. Token can be obtainted by authentication endpoints above or can be generated from application (token agenda was added @since 10.8.0 version):


Token detail

Make sure to copy your new authentication token now. You won't be able to see it again!
Token you have generated can be used to access the CzechIdM API.

Token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ew0KICAiaWQiIDogIjgxOTU3ZGU3LTM1MzUtNDFhMS05ZWEwLTJhODBkN2M2NjZmNiIsDQogICJjdXJyZW50VXNlcm5hbWUiIDogInNjaW1PbmUyIiwNCiAgImN1cnJlbnRJZGVudGl0eUlkIiA6ICI4MWNiYWU0Ny03YjEzLTQ5YjctYWEzYy03MTVkZGY1NGM2MTgiLA0KICAib3JpZ2luYWxVc2VybmFtZSIgOiAic2NpbU9uZTIiLA0KICAib3JpZ2luYWxJZGVudGl0eUlkIiA6ICI4MWNiYWU0Ny03YjEzLTQ5YjctYWEzYy03MTVkZGY1NGM2MTgiLA0KICAiZnJvbU1vZHVsZSIgOiAiY29yZSIsDQogICJleHBpcmVkIiA6IGZhbHNlLA0KICAiZXhwIiA6IG51bGwsDQogICJpYXQiIDogIjIwMjEtMDEtMjVUMTA6MDQ6MjYuODE5KzAxOjAwIg0KfQ.OupgxOEdBfYQtZmvulfHMmC3NUCi5RK0d1O_OWlSNkI

Issued at
25.01.2021 10:04
Token created date.

Valid till

Use token agenda for generate system tokens, those can be used for system to system communication. Token expiration can be set optionally, but is not required. Token will grant the same authorities and permissions as token owner.

If you have a valid token, you can use it, for example to get all identities:

```
curl -X GET "http://localhost:8080/idm-backend/api/v1/identities" -H
"accept: */*" -H  "CIDMST:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ew0KICAiaWQiIDogIjMwNmJlMWMyLTNmZTItNDZ
lMy1hYzkwLTExZmI1NTA1ZDYwZSIsDQogICJjdXJyZW50VXNlcm5hbWUiIDogImFkbWluIiwNCiA
gImN1cnJlbnRJZGVudGl0eUlkIiA6ICJkNDE0YTA5MS0zODE1LTQzOWQtOWVjNi1iZWQ4MTlmZWE
0MzkiLA0KICAib3JpZ2luYWxVc2VybmFtZSIgOiAiYWRtaW4iLA0KICAib3JpZ2luYWxJZGVudGl
0eUlkIiA6ICJkNDE0YTA5MS0zODE1LTQzOWQtOWVjNi1iZWQ4MTlmZWE0MzkiLA0KICAiZnJvbU1
vZHVsZSIgOiAiY29yZSIsDQogICJleHBpcmVkIiA6IGZhbHNlLA0KICAiZXhwIiA6IG51bGwsDQo
gICJpYXQiIDogIjIwMjEtMDEtMjVUMTA6MDk6NTYuODk1KzAxOjAwIg0KfQ.4duJJb5pJDfJvXmd
k259gW92fkjzqEr8ya9VqVJ3jCo"
```

TokenManager

`TokenManager` is used for creating IdM tokens. Token can be created for different purpose (token type). When token is used, can be disabled (e.g. logout, one time usage).

Token expiration extension

By default both frontend (FE) and backend (BE) automatically extend the expiration of IdM JWT. In case of successful GET request on API, the token expiration is extended on BE and passed in HTTP response (watch out for CORS, has to be explicitly allowed, since the HTTP header is called "CIDMST"). FE (`RestApiService.js`) automatically disassembles the response looking for auth. token. If token is present, it is set as new token which shall be used for new API requests (done in `App.js`). Expiration is extended in one minute window - when token has expiration at least one minute older than new one (prevent to persist token every api call).

Developer tip: do not trust the console, especially while working with promises:

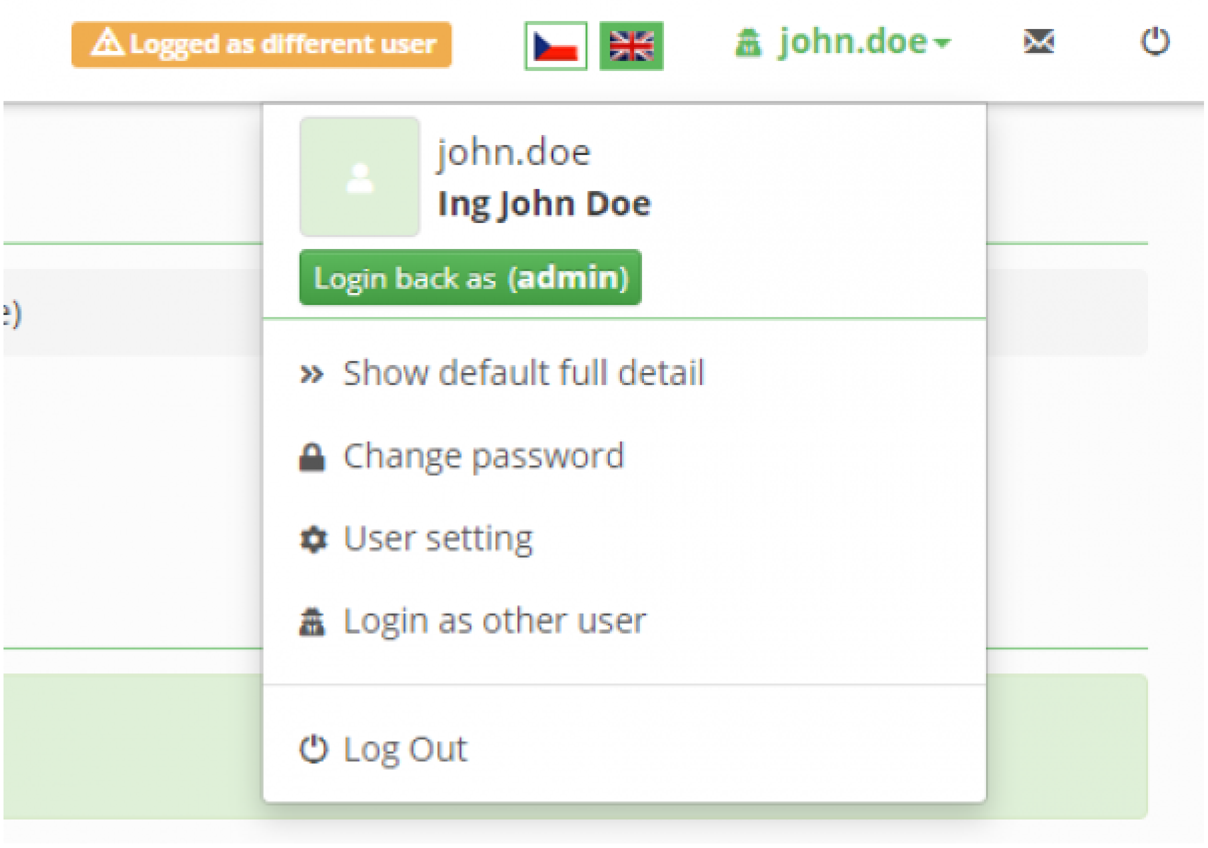
```
static printHeadersOK(response) {
  console.log(response.headers.get('CIDMST')); prints header value
}

static printHeadersWRONG(response) {
  console.log(response.headers); // prints '{}', object looks empty
}
```

Switch user - login as other user

@since 10.5.0

User with authority `IDENTITY_SWITCHUSER` can login as different user. Feature is usable for checking application behavior under selected user (e.g. helpdesk). Logged user needs to have permission SWITCHUSER to select target user. Feature is available from user main menu in application - any other active user can be selected and currently logged user will be logged as selected used.

Currently logged user token (see above) is filled with selected user authorities - original user is preserved as token owner. Return back (logout) to originally logged user is available from user main menu too.



Enforce password change

@since 10.5.0

When password is reseted (or generated), then password change can be enforced, after the first password usage (after the first user login). User is redirected to password change form and password has to be changed before user can continue to work with application.

Flag for enforce password change can be filled by project processors (see password change, reset and generate events) or by GUI (flag added to password detail).

SSO

SSO support on FE is implemented by issuing the internal IdM JWT without the need to type in user's credentials. This is done by sending a "prerequest" onto secured /authentication/remote-auth endpoint, accessible only to authenticated users. The authentication itself must be done in one of auth filters.

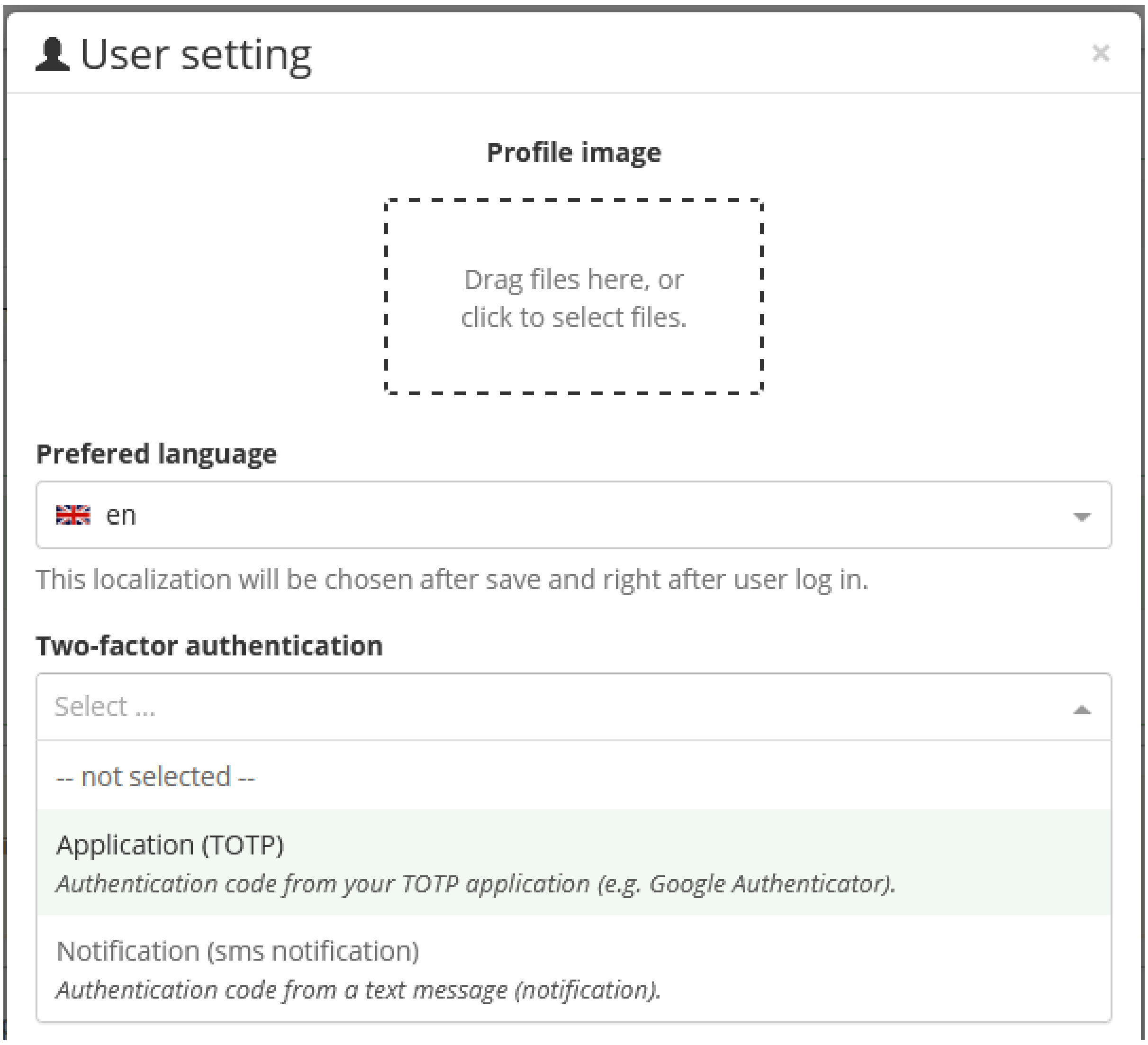
We expect either HTTP header or Cookie to be the bearer of the remote authentication data. When the user accesses IdM, FE first tries to sign him in using the `remote-auth`. The `remote-auth` attempt is done only on first access to CzechIdM, token expiration, authority change or user login.

SSO authentication to AD domain can be configured with the help of SsoIdmAuthenticationFilter and Apache web server, which is installed in front of IdM. For more details about SSO configuration see SSO install guide.

Two-factor authentication

@since 10.7.0

Feature is available from user profile (user main menu) in application. Two-factor authentication is optional and can be enabled from user profile:

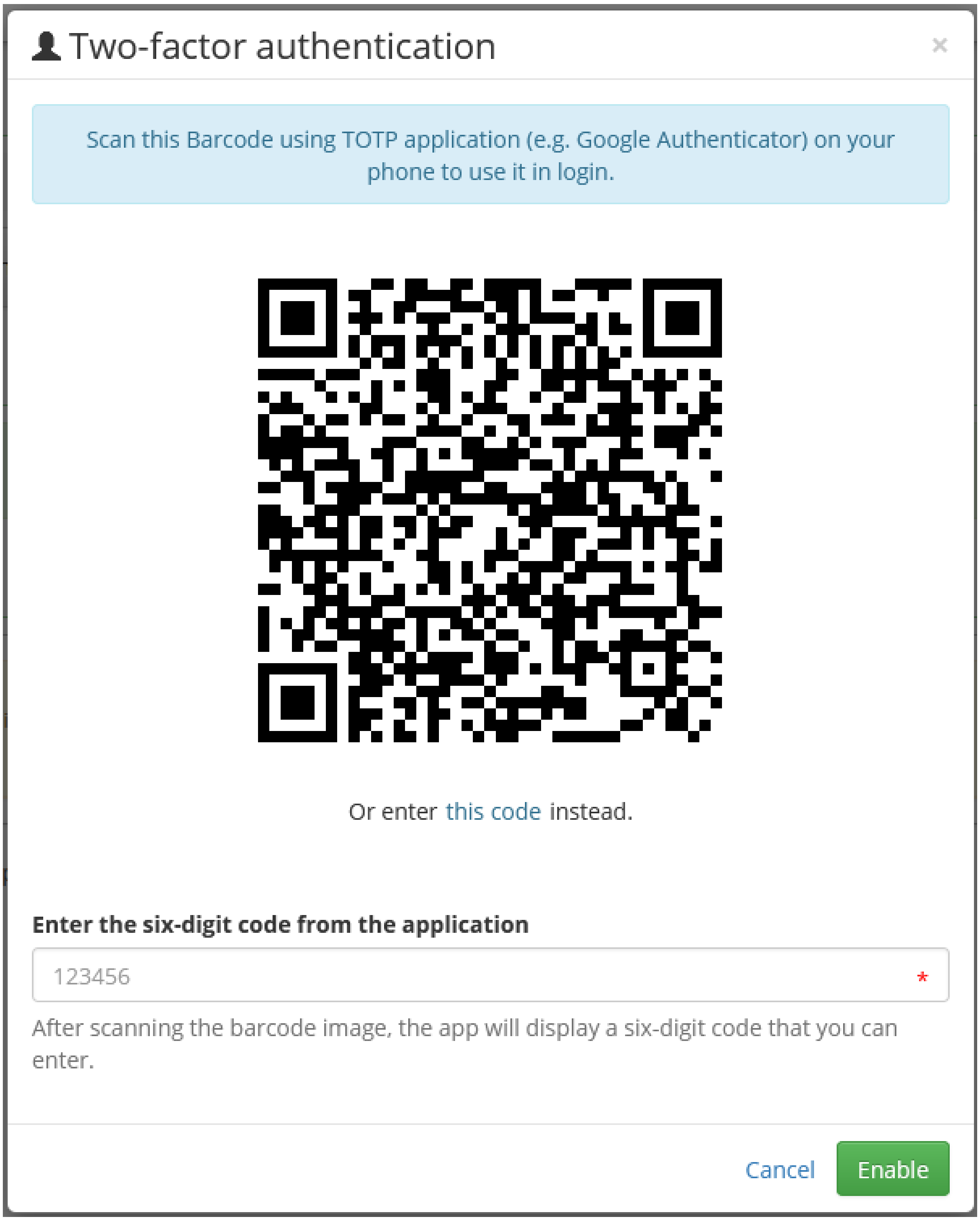

User setting
×
Profile image
Drag files here, or click to select files.
Prefered language
en
This localization will be chosen after save and right after user log in.
Two-factor authentication
Select ...
-- not selected --
Application (TOTP)
Authentication code from your TOTP application (e.g. Google Authenticator).
Notification (sms notification)
Authentication code from a text message (notification).

After user sign in to CzechIdM using username and password, user will be prompted to provide an authentication code from a text message (notification) or from TOTP application. CzechIdM will only ask to provide two-factor authentication code again if user have logged out, is using a new device, or user session expires.

Supported methods to get authentication code:

* **Application** - authentication code is generated by TOTP application (Google Authenticator, TOTP Authenticator, FreeOTP etc.). * **Sms notification** - authentication code is send by standard IdM notification. **Sms notifications have to be supported** to use this method (sms notification sender needs to be provided).

The first authentication code obtained by selected method is needed to enable using two-factor authentication:



Two-factor authentication

Scan this Barcode using TOTP application (e.g. Google Authenticator) on your phone to use it in login.

Or enter this code instead.

Enter the six-digit code from the application

123456 *

After scanning the barcode image, the app will display a six-digit code that you can enter.

HTTP Basic authentication cannot be used with two-factor authentication enabled.

SSO (e.g. remote user) can be used with two-factor authentication enabled. After user is logged in by sso, user will be prompted to provide an authentication code from a text message (notification) or from TOTP application.

Password change is enforced after two-factor authentication code is entered. After password is changed, then user will be prompted to provide an authentication code again to sign in.

Recovery codes are not provided. E.g. when user lost his device with configured TOTP aplication, then helpdesk user can help him to reset his authentication method with switch user feature usage.

Configuration

Basic configuration is available by application properties.

Advanced: Is possible to change default SHA1 hashing algoritm (~provide different hashing algorithm as spring bean) and default 6-digit verification code length (~change frontend localization) programatically.

CAS authentication

@since 10.9.0

This feature is disabled by default. If you want to enable it, see configuration properties application

You need to have working CAS - You are able to log in directly to CAS. Cas will return username of authenticated user. This username for CAS must be same as username in IdM.

Other then that, you need to configure CAS. Mandatory CAS configuration is below cas.properties

```
...
cas.client.validator-type=CAS30
# ticket is validated multiple times, because auth filter is called multiple times. This should be no security problem, because the token is still usable only for 10 sec.
cas.ticket.st.number-of-uses=25

# redirectiong after logout will be supported
cas.logout.followServiceRedirects=true
...
```

idm-200.json

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "https://HOSTNAME/idm.+",
  "name" : "IdM",
  "id" : 200,
  "evaluationOrder" : 1
}
```

When all configuration is done. You will be able to log into IdM **ONLY** via CAS. **The authentication flow is**: Access IdM URL → if you are not logged in → redirect to CAS → log in → redirect back to IdM → You are authenticated in IdM Access IdM URL → If you are logged in (jwt token is valid) → You are authenticated in IdM **Logout:** Click on logout button in IdM → Log out of IdM → CAS logout URL is called → logout from CAS → redirect to IdM → Redirect back to CAS because you are not authenticated **Expired jwt token:** Logged out of IdM → redirect to /login → CAS is conntacted → authentication flow is applied now

OIDC authentication

@since 13.1.0

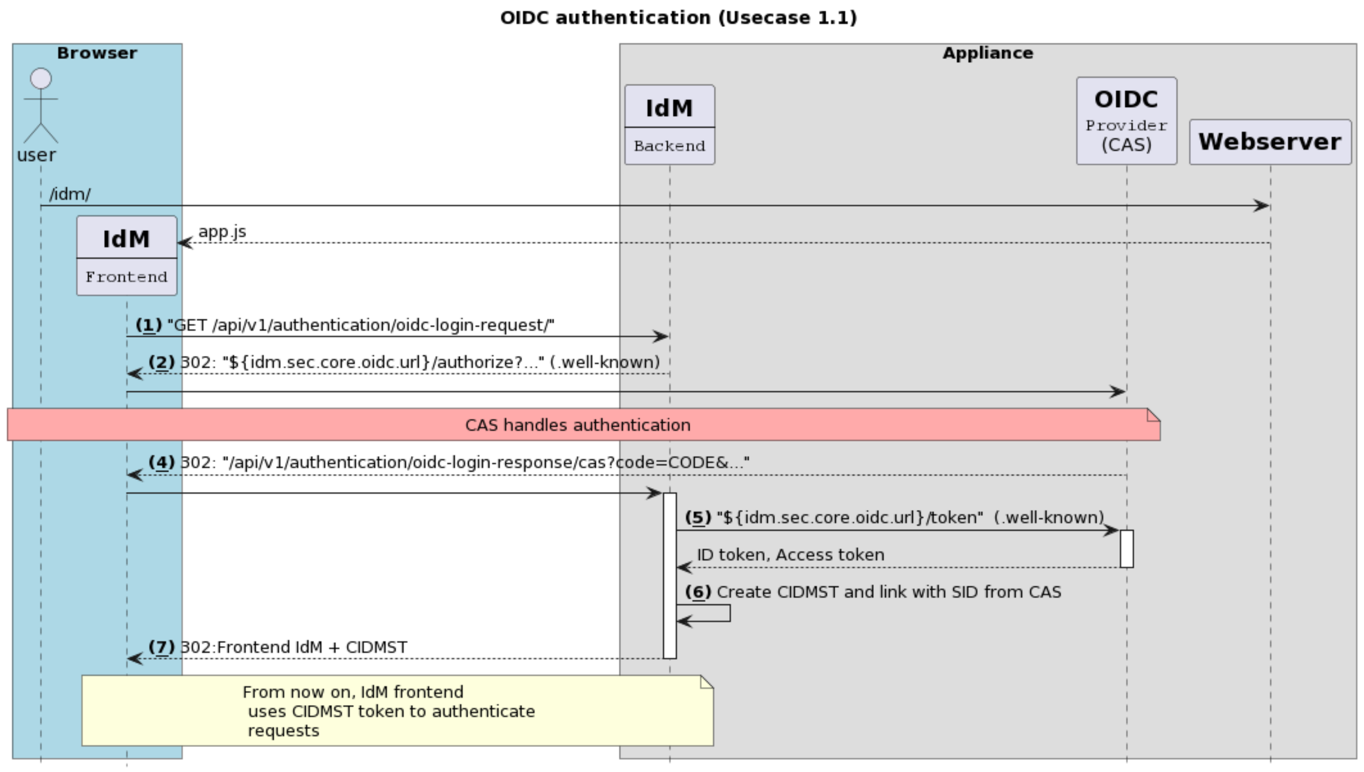
This feature is disabled by default. If you want to enable it, see configuration properties application

CAS Service for OIDC configuration:

idm-oidc-201.json

```
{
  "@class" : "org.apereo.cas.services.OidcRegisteredService",
  "clientId" : "client",
  "clientSecret": "secret",
  "serviceId" : "redirectUrl",
  "name" : "CzechIdM OIDC",
  "id" : 201,
  "evaluationOrder" : 1,
  "scopes": ["java.util.HashSet", ["openid"]],
  "supportedResponseTypes": ["java.util.HashSet", ["code"]],
  "logoutType": "BACK_CHANNEL",
  "logoutUrl": "logoutUrl",
  "redirectUrl": "redirectUrl"
}
```

The authentication flow from front


OIDC authentication (Usecase 1.1)
Browser
Appliance
user
IdM Frontend
IdM Backend
OIDC Provider (CAS)
Webserver
/idm/
app.js
(1) "GET /api/v1/authentication/oidc-login-request/"
(2) 302: "${idm.sec.core.oidc.url}/authorize?..." (.well-known)
CAS handles authentication
(4) 302: "/api/v1/authentication/oidc-login-response/cas?code=CODE&..."
(5) "${idm.sec.core.oidc.url}/token" (.well-known)
ID token, Access token
(6) Create CIDMST and link with SID from CAS
(7) 302:Frontend IdM + CIDMST
From now on, IdM frontend uses CIDMST token to authenticate requests

1) User isn't authenticated so frotend redirects user to api endpoint for OIDC login.

2) IDM redirects user to OIDC providers login page (adress from .well-known endpoint on OIDC providers or can be overwritten in config)

3) User login on OIDC providers page

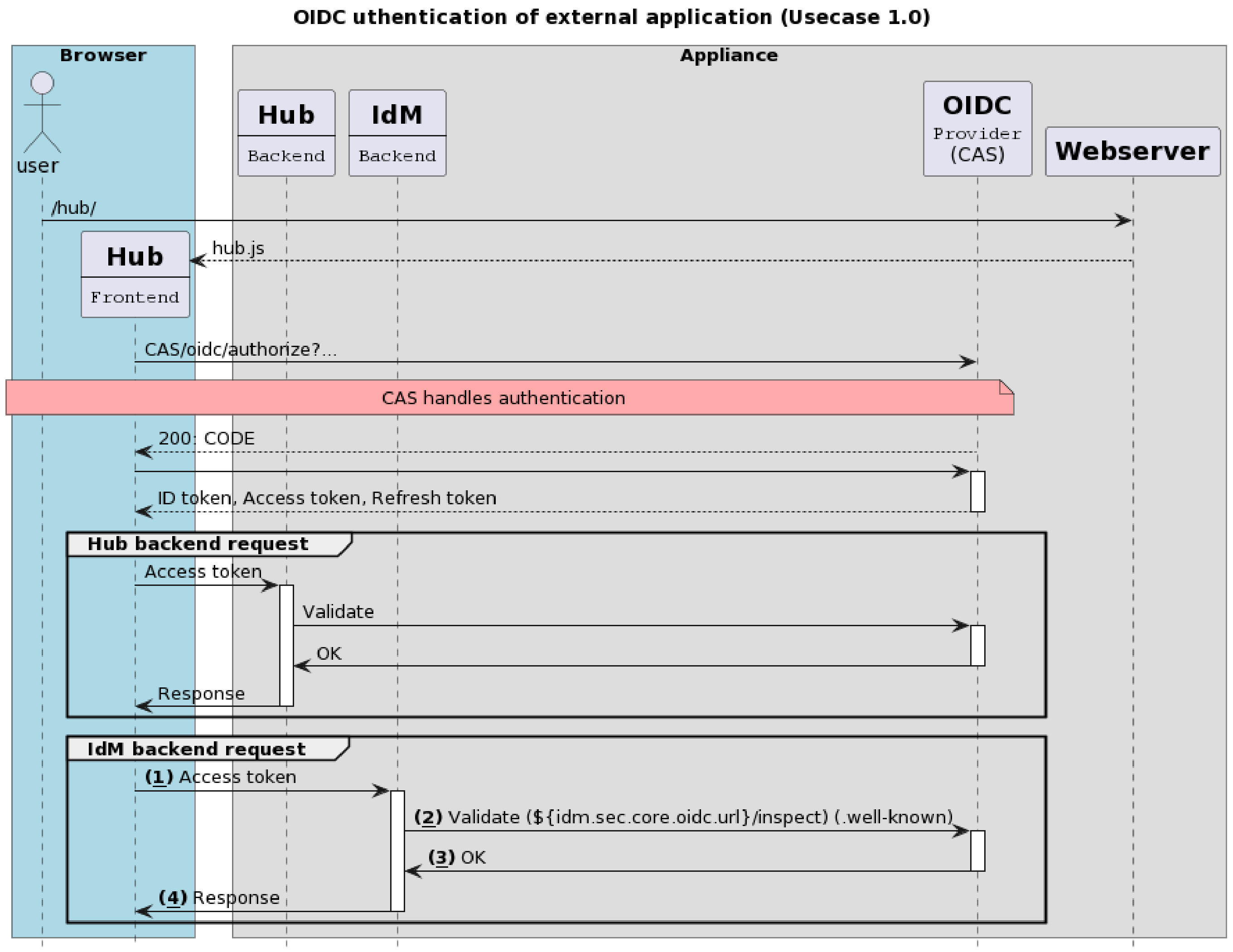
4) User gets redirected back to IDM with "code"

5) IDM uses "code" to get from OIDC provider, ID token and Access token (adress from .well-known endpoint on OIDC providers or can be overwritten in config)

6) After validation IDM creates CIDMST token (with externalID set to SID)

7) IDM redirects to front with CIDMST

The authentication flow from external application (hub, ...)



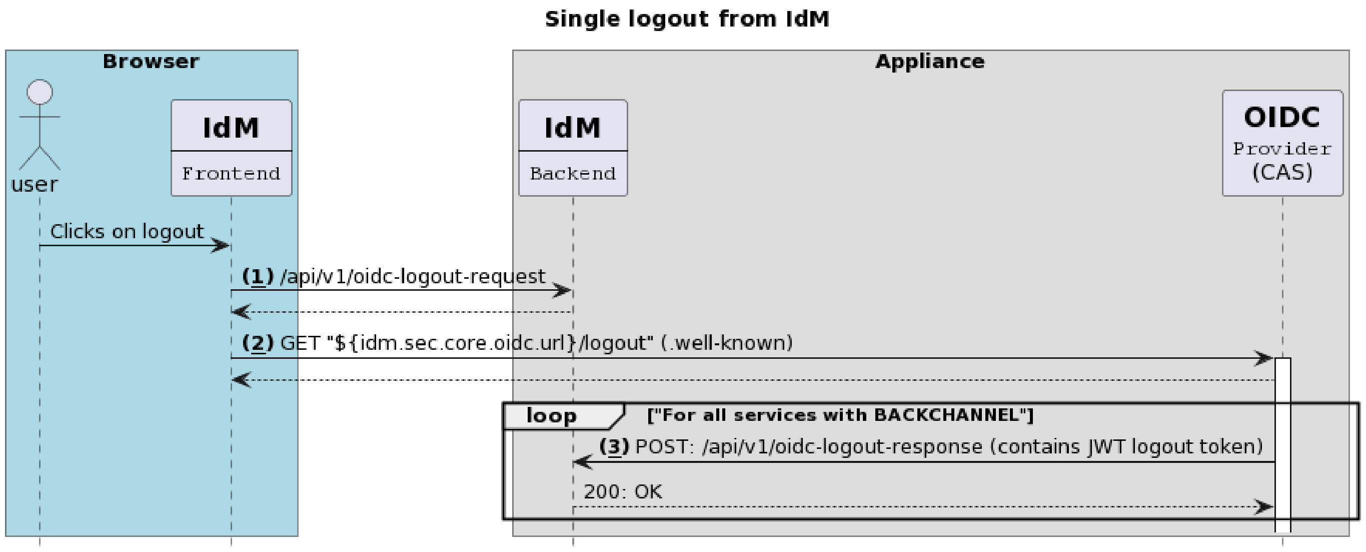
1) Authentication happens on external application

2) When accessing a resource, external application presents access token (in header "Authorization" and value starts with "Bearer") (adress from .well-known endpoint on OIDC providers or can be overwritten in config)

3) IDM validates token on OIDC provider

4) If user has access to resource returns it

Single logout flow from IDM


Single logout from IdM
Browser
Appliance
user
IdM Frontend
IdM Backend
OIDC Provider (CAS)
Clicks on logout
(1) /api/v1/oidc-logout-request
(2) GET "${idm.sec.core.oidc.url}/logout" (.well-known)
loop ["For all services with BACKCHANNEL"]
(3) POST: /api/v1/oidc-logout-response (contains JWT logout token)
200: OK

1) User clicks on logout button so front redirects him to api endpoint for OIDC logout.

2) IDM redirect him to logout endpoint on OIDC provider (adress from .well-known endpoint on OIDC providers or can be overwritten in config)

3) OIDC provider calls IMD and IDM invalidates all tokens with SID (from JWT token)