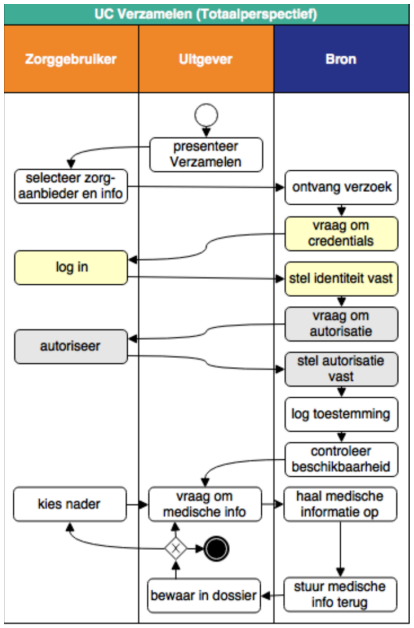
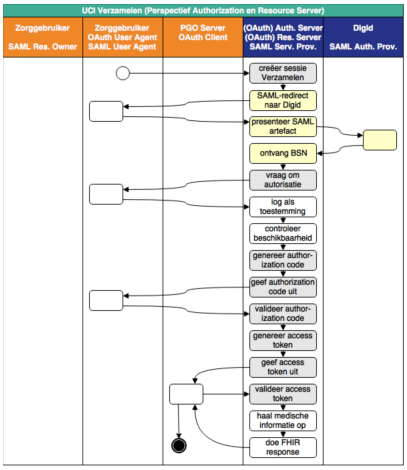
UCI ZA Node OAuth



Programmering talen: C#, Java, Node.js, Python, PHP
Complexiteit: *) +++++
Verwachte implementatie tijd: 8 u
Deployment: Library in een van Programmering talen
Testing: WS direct access?

Opdracht context:

1. De *PGO GW* start de flow door in de *User Agent* van de *Zorggebruiker* de mogelijkheid te presenteren om een bepaalde *Gegevensdienst* bij een zekere *Zorgaanbieder* te verzamelen. Uit de *Zorgaanbiederslijst* weet de *PGO GW* welke *Gegevensdiensten* voor een *Zorgaanbieder* beschikbaar zijn. In de local state-parameter geeft de *PGO GW* informatie mee aan de *ZA GW*, waaraan de *PGO GW* later, bij de redirect, precies weet bij welk verzoek de authorization code hoort.
2. De *Zorggebruiker* maakt zijn selectie en laat de *OAuth User Agent* een verzamel-verzoek sturen naar de *ZA GW*. Het adres van het authorization endpoint komt uit de *ZAL*. De redirect URI geeft aan waarnaartoe de *ZA GW* (als *OAuth Authorization Server*) de *OAuth User Agent* verderop moet redirecten (met de authorization code).
3. Daarop begint de *ZA GW* de OAuth-flow (in zijn rol als *OAuth Authorization Server*) door een sessie te creëren.
4. De *ZA GW* controleert alvast of de *Zorgaanbieder* voor de betreffende *Gegevensdienst* überhaupt gezondheidsinformatie van die *Persoon* beschikbaar heeft.
5. Zo ja, dan presenteert de *ZA GW* (nog steeds als *OAuth Authorization Server*) via de browser aan *Zorggebruiker* de vraag of laatstgenoemde hem toestaat de gevraagde persoonlijke gezondheidsinformatie aan de *PGO GW* (als *OAuth Client*) te sturen. Onder het flow-diagram staat gespecificeerd welke informatie, waarvandaan, de *OAuth Authorization Server* verwerkt in de aan *Zorggebruiker* voor te leggen autorisatievraag.



6. Bij akkoord logt de *ZA GW* dit als toestemming, genereert een authorization code en stuurt dit als ophaalbewijs, door middel van een browser redirect met de in stap 1 ontvangen redirect URI, naar de *PGO GW*. De *ZA GW* stuurt daarbij de local state-informatie mee die hij in de eerste stap van de *PGO GW* heeft gekregen. Laatstgenoemde herkent daaraan het verzoek waarmee hij de authorization code moet associëren.
7. De *PGO GW* vat niet alleen deze authorization code op als ophaalbewijs, maar leidt er ook van af dat de toestemming is gegeven en logt deze toestemming.
8. Met dit ophaalbewijs wendt de *PGO GW* zich weer tot de *ZA GW*, maar nu zonder tussenkomst van de *OAuth User Agent*, voor een access token.
9. Daarop genereert de *ZA GW* een access token en stuurt deze naar de *PGO GW*.
10. Nu is de *PGO GW* gereed om het verzoek om de gezondheidsinformatie naar de *ZA GW* te sturen. Het adres van het resource endpoint haalt hij uit de *ZAL*. Hij plaatst het access token in het bericht en zorgt ervoor dat in het bericht geen BSN is opgenomen.

Opdracht beschrijving:

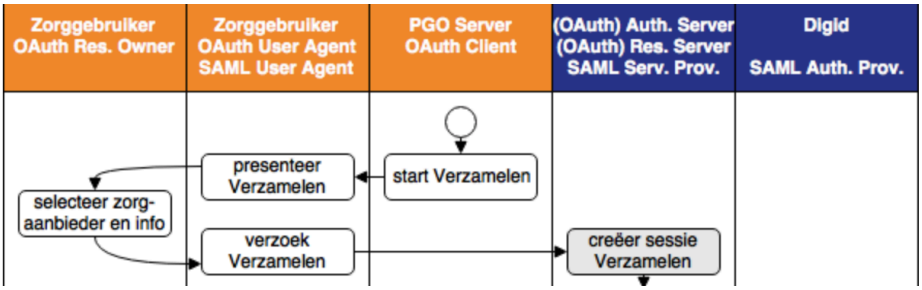
Test case 1:

UCI Verzamelen 1	*Preconditie:* *De Zorggebruiker maakt zijn selectie en laat de OAuth User Agent een verzamel-verzoek sturen naar de ZA GW. Het adres van het authorization endpoint komt uit de ZAL. De redirect URI geeft aan waarnaartoe de ZA GW (als OAuth Authorization Server) de OAuth User Agent verderop moet redirecten* Daarop begint de Authorization Server de OAuth-flow (in zijn rol als OAuth Authorization Server) door een sessie te creëren: 1. Tijdens de use case-implementatie UCI Verzamelen zet de Authorization Server, onmiddellijk na de authenticatie van de Zorggebruiker, de OAuth-autorisatie voort, volgens de standaard OAuth 2.0. 2. Voor zover er in het verkeer tussen PGO Server en Resource Server in de use case-implementatie UCI Verzamelen sprake is, in de payload, van een gegevenselement dat tot de identiteit van de Zorggebruiker herleid kan worden, gebruiken zij daarvoor niets anders dan de OAuth-gegevens die zij in hun respectievelijke OAuth Client en OAuth Resource Server moeten uitwisselen. PGO Server, Authorization Server en Resource Server treffen goed beveiligde voorzieningen waarmee zij hieruit waar nodig zelf de identiteit van de Zorggebruiker kunnen vaststellen. Voor zover sprake is van een informatie-element dat het BSN bevat, zal deze niet worden gebruikt of leeg blijven. 3. Van de vier soorten authorization grants die OAuth 2.0 biedt, beperken de OAuth-rollen zich tot alleen de eerste: Authorization Code. 4. De OAuth-rollen *Client* en *Resource Server* zullen slechts tokens van het type Bearer Token uitwisselen, conform RFC6750. 5. De OAuth-rollen Client, Authorization Server en Resource Server implementeren de op hen toepasselijke beveiligingsmaatregelen, voor zover zij passen bij het MedMij Afsprakenstelsel 6. De OAuth-rol Authorization Server genereert authorization codes en access tokens met een enkelvoudige scope die bepaald is door de op te vragen Gegevensdienst. 7. De OAuth-rol Authorization Server stelt van elke uitgegeven authorization code en elk uitgegeven access token de geldigheidsduur op exact 15 minuten (900 seconden). Zij geeft bovendien geen refresh tokens uit. 8. De OAuth-rol *Authorization Server* genereert authorization codes en access tokens volgens UUID. Daarbij wordt slechts gebruik gemaakt van UUID Version 4. Met betrekking tot zowel authorization codes als access tokens, draagt de OAuth-rol *Authorization Server* ervoor zorg dat nooit twee dezelfde geldige door haar uitgebrachte daarvan in omloop zijn. 9. De OAuth Client biedt een zekere authorization code maximaal eenmaal aan aan de Authorization Server ter verkrijging van een access token. De Authorization Server voert een authorization code af, wanneer het eenmaal is aangeboden ter verkrijging voor een access token. 10. De OAuth-rol Authorization Server draagt alleen een access token over aan een OAuth Client als de daartoe aangeboden authorization code aan diezelfde OAuth Client is afgegeven. 11. De OAuth-rol Client biedt aan de Authorization Servers slechts redirect URI's aan die volledig (full) zijn én verwijzen naar een HTTPS-beschermd endpoint. Authorization Servers redirecten niet naar een URI die niet aan deze eisen voldoet.	Conform OAuth 2.0

Exceptie handling

UCI Verzamelen 1	Authorization Server vindt het ontvangen verzoek ongeldig.	Authorization Server informeert Zorggebruiker over deze uitzondering. Zorggebruikerlaat PGO Server de flow afbreken.	conform OAuth 2.0-specificatie, par. 4.1.2.1, error code invalid_request, met in de error description de oorzaak
UCI Verzamelen 2	*Authorization Server* kan de identiteit van de *Zorggebruiker* niet vaststellen.	*Authorization Server* informeert *PGO Server* over deze uitzondering. *PGO Server* informeert daarop *Zorggebruiker* hierover .	conform OAuth 2.0-specificatie, par. 4.1.2.1, error code unauthorized_client
UCI Verzamelen 3	*Authorization Server*stelt vast dat van *Persoon* bij *Zorgaanbieder* geen gezondheidsinformatie voor die *Gegevensdienst*beschikbaar is.	*Authorization Server* informeert *PGO Server* over deze uitzondering. *PGO Server* informeert daarop *Zorggebruiker* hierover.	conform OAuth 2.0-specificatie, par. 4.1.2.1, error code access denied, met in de error description "No such resources."

Uitleg




```
URI - /oauth?response_type=code&client_id=CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=?&state=1233xxx
```

- `URI – provided from the definition in the ZAL`
- **client_id=CLIENT_ID** – PGO NODE Hostname
- **redirect_uri=REDIRECT_URI** - Indicates the URI to return the user to after authorization is complete, such as httpd://authorize

- **scope= 1… (zie https://afsprakenstelsel.medmij.nl/display/PUBLIC/Gegevenscatalogus)** - ONE scope value indicating which parts of the user's account you wish to access
- **state=1234zyx** - A random string generated by your application, which you'll verify later

***Note**: Noch in de authorization code, noch in het access token wordt betekenisvolle informatie opgenomen. Dat zorgt er ook voor dat er een minimale afhankelijkheid wordt gecreëerd tussen de PGO GW en de ZA GW, zodat principe P1 maximaal wordt nageleefd en interne complexiteit en implementatiekeuzes van de ZA GW niet doorschemeren in, of invloed uitoefenen op, de implementatie van de PGO GW.*

Het formaat van UUIDs wordt textueel vaak opgeschreven in 32 hexadecimalen, hier en daar gescheiden door streepjes, als volgt:
xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx
Hierbij worden de vier bits van M gebruikt voor het versienummer (in het MedMij Afsprakenstelsel dus: 4) en de eerste twee bits van N voor het zogenoemde variantnummer (altijd 1 in de betreffende RFC). Daarmee zien in het MedMij Afsprakenstelsel zowel de authorization code als het acces token eruit als:
xxxxxxxx-xxxx-4xxx-Nxxx-xxxxxxxxxxxx, met N = 10bb (in bits) en alle b en x random. Let wel, met random wordt niet bedoeld dat er geen eisen aan worden gesteld, maar juist dat eraan de eis wordt gesteld dat de waarden willekeurig worden gegenereerd.

- ***Test case 2:***

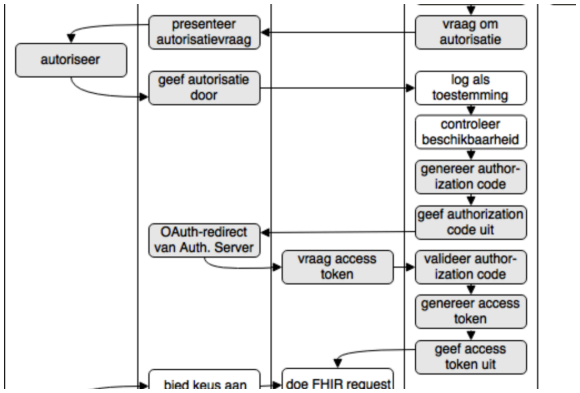
UCI Verzamelen 1	De *ZA GW* controleert alvast of de *Zorgaanbieder* voor de betreffende *Gegevensdienst* überhaupt gezondheidsinformatie van die *Persoon* beschikbaar heeft. Zo ja, dan presenteert de *ZA GW* (nog steeds als *OAuth Authorization Server*) via de browser aan *Zorggebruiker* de vraag of laatstgenoemde hem toestaat de gevraagde persoonlijke gezondheidsinformatie aan de *PGO GW* (als *OAuth Client*) te sturen. Onder het flow-diagram staat gespecificeerd welke informatie, waarvandaan, de *OAuth Authorization Server* verwerkt in de aan *Zorggebruiker* voor te leggen autorisatievraag. Bij akkoord logt de *ZA GW* dit als toestemming, genereert een authorization code en stuurt dit als ophaalbewijs, door middel van een browser redirect met de in stap 1 ontvangen redirect URI, naar de *PGO GW*. De *ZA GW* stuurt daarbij de local state-informatie mee die hij in de eerste stap van de *PGO GW* heeft gekregen. Laatstgenoemde herkent daaraan het verzoek waarmee hij de authorization code moet associëren.	Conform OAuth 2.0

Exceptie handling

UCI Verzamelen 4	De autorisatievraag wordt ontkennend beantwoord.	*ZA GW* logt de afwijzing eninformeert *PGO GW* hierover. *Uitgever* informeert daarop *Zorggebruiker* hierover.	conform OAuth 2.0-specificatie, par. 4.1.2.1, error code access denied, met in de error description "Authorization denied."
UCI Verzame	*ZA GW* kan de autorisatie	*ZA GW* informeert *PGO GW* over deze uitzondering. *PGO*	conform OAuth 2.0-specificatie, par.

len 5	niet vaststellen.	*GW* informeert daarop *Zorggebruiker* hierover.	4.1.2.1, error code access denied, met in de error description "Authorization failed."
UCI Verzame len 6	De validatie van de authorization code door*ZA GW* faalt.	*ZA GW* informeert *PGO GW* over deze uitzondering. *PGO GW* informeert daarop Zorggebruiker hierover.	conform OAuth 2.0-specificatie, par. 5.2, error code invalid_grant

Uitleg




```
https://REDIRECT_URI/cb?code=AUTH_CODE_HERE&state=1234zyx
```

- **code** - The server returns the authorization code in the query string
- **state** - The server returns the same state value that you passed

Token Exchange

PGO server exchanges het auth code voor een access token:

```
POST URI /token grant_type=authorization_code&code=AUTH_CODE_HERE&
redirect_uri=REDIRECT_URI& client_id=CLIENT_ID
```

- `URI – provided from the definition in the ZAL`
- **client_id=CLIENT_ID** – PGO NODE Hostname
- **redirect_uri=REDIRECT_URI** - Indicates the URI to return the user to after authorization is complete, such as httpd://authorize

Opdracht resources:

Template Autorisatievraag:
https://afsprakenstelsel.medmij.nl/pages/viewpage.action?pageId=14780378

Opdracht acceptatie:

Test case 1: Functionele API coverage

Test case 2: Unit Test executie voor elke API validatie

Test case 3: Integratie Test Met PGO OAuth Client