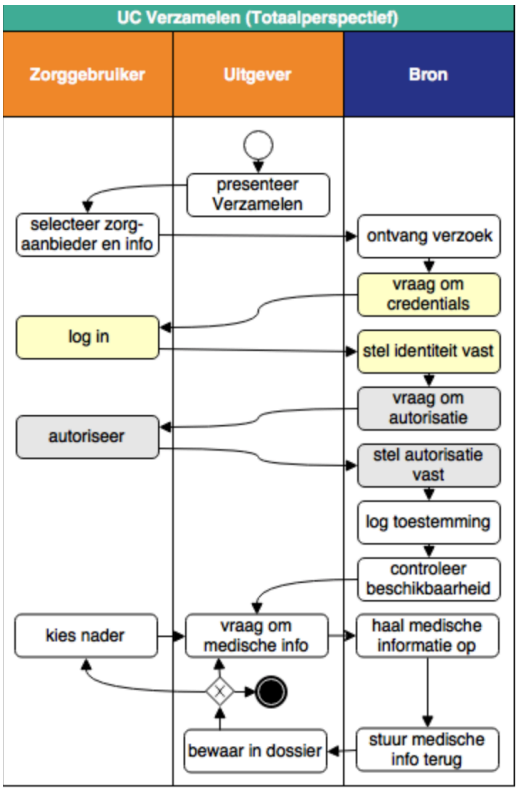
UCI PGO OAuth

Programmeertalen: C#, Java, Node.js, Python, PHP, Ruby,
Complexiteit: *) +++++
Verwachte implementatie tijd: 8-12 u
Oplevering: Coding Standard volgens Blauwdruk Design Template (zie Appendix)
Deployment: Library in de betreffende programmeertaal
Testing: Mockup testing



Figuur 1: OAuth MedMij - PGO Autorisatie

Uitvoering van de opdracht:

Deze opdracht "OAuth specifieke flow voor MedMij" is gedefineerd rondom de drie grijze blokken in figuur 1 hiernaast:

- Vraag om autorisatie
- Autorisatie proces
- Stel autorisatie vast

Er zijn twee kerninteracties binnen de usecase waar deze opdracht over gaat:

- Vraag om autorisatie (Test case 1)
- Stel autorisatie vast (Test case 2)

Document opbouw:

- Functionele Context vanuit stelsel: met de link naar https://afsprakenstelsel.medmij.nl
- De testcases voor de twee interacties samen met de exceptions: de test cases moeten direct binnen de library als onderdeel van de unit test zijn.

1. Opdracht context: (hier de gehele context om deze opdracht te kunnen uitvoeren – met het eind doel als focus op de 3 grijze blokken binnen Figuur 1)

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 (Figure 2 Vraag toestemming) staat gespecificeerd welke informatie, waarvandaan, de *OAuth Authorization Server* verwerkt in de aan *Zorggebruiker* voor te leggen autorisatievraag.

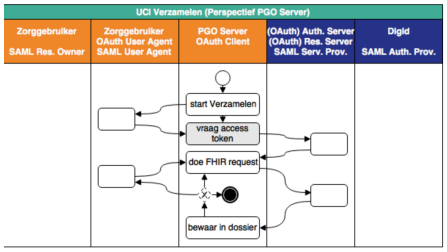


Figure 2 Vraag toestemming

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

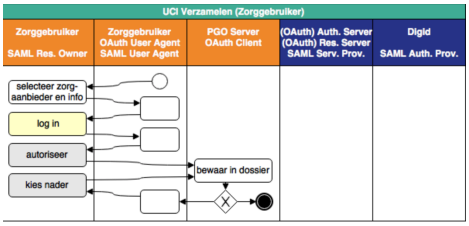


Figure 3 Autorisatie process

Figure 3 Autorisatie process

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.

2. Opdracht beschrijving (Techische informatie en Test cases voor acceptatie):

2.1 Test case 1: Vraag autorisatie

Het opvragen van de autorisatie zoals aangegeven in de Figuur 1: OAuth MedMij - PGO Autorisatie.

UCI Verzamelen 1	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	Conform OAuth 2.0

2.1.1 Exceptie handling

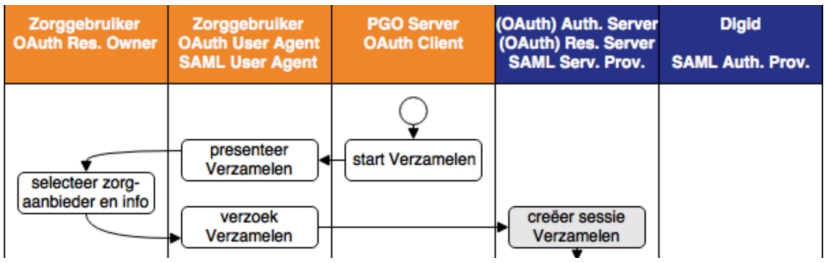
UCI Verzamelen 1	Authorization Server vindt het ontvangen verzoek ongeldig.	Authorization Server informeert Zorggebruiker over deze uitzondering. Zorggebruikerlaat PGO Server de flow afbreken.	conform OAuth 2.0-specificatie, par. 4.1.2.1, error code invalid_request, met in de error description de oorzaak

UCI Verzamelen 2	*Authorization Server* kan de identiteit van de *Zorggebruiker* niet vaststellen.	*Authorization Server* informeert *PGO Server* over deze	conform OAuth 2.0-specificatie, par. 4.1.2.1, error

		uitzondering. *PGO Server* informeert daarop *Zorggebruiker* hierover .	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."

2.1.2




```
URI voorbeeld - /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*
- ***Note**:* 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.

2.2 Test case 2: Stel autorisatie vast:

Het vast leggen van de autorisatie proces zoals aangegeven in de Figuur 1: OAuth MedMij - PGO Autorisatie.

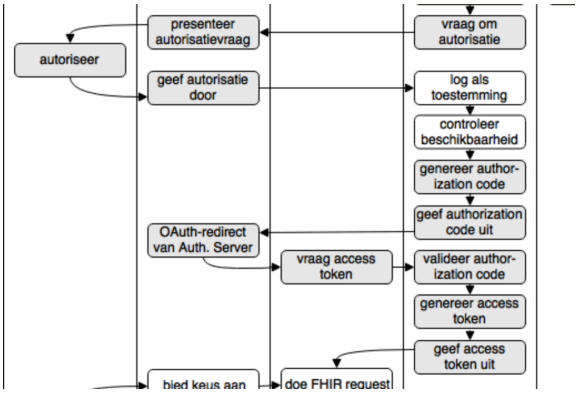
UCI	De *ZA GW* controleert alvast of de *Zorgaanbieder* voor de	Conform

Verzamelen 1	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.	OAuth 2.0

2.2.1 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 Verzamelen 5	*ZA GW* kan de autorisatie niet vaststellen.	*ZA GW* informeert *PGO GW* over deze uitzondering. *PGO GW* informeert daarop *Zorggebruiker* hierover.	conform OAuth 2.0-specificatie, par. 4.1.2.1, error code access denied, met in de error description "Authorization failed."
UCI Verzamelen 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

2.2.2 Uitleg




```
https://example-app.com/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

2.2.3 Token Exchange

PGO server exchanges het auth code voor een access token:

```
Voorbeeld - POST https://api.authorization-server.com/token
grant_type=authorization_code&   code=AUTH_CODE_HERE&
redirect_uri=REDIRECT_URI&   client_id=CLIENT_ID&
client_secret=CLIENT_SECRET
```

3. Opdracht acceptatie (Ter finale acceptatie van deze opdracht/library):

Acceptatie case 1: Functionele API coverage (documentatie van de code zal moeten de integratie binnen de gehele applicatie laten zien)

Acceptatie case 2: Unit Test executie voor elke API validatie (De library zal moeten de benoemde Test Cases ondersteunen)

Acceptatie case 3: Integratie Test Met PGO OAuth Client (Integratie Test uitvoering)