



Unabhängige Treuhandstelle

UNIVERSITÄTSMEDIZIN GREIFSWALD

Stand: Dez. 2024

Absicherung der THS-Schnittstellen mit **Keycloak**-basierter Authentifizierung

Mit dem Herbstrelease 2022 können nun **alle** THS-Schnittstellen (**WEB**-Oberfläche, **FHIR**-Gateway und **SOAP**-Webservices) je Endpunkt und somit je Werkzeug (**E-PIX**, **gICS** und **gPAS**) mit **Keycloak**-basierter (und damit **OIDC**-konformer) Authentifizierung abgesichert werden.

Dies ist eine Anleitung für die Installation, die Einrichtung, den Test und die Benutzung eines **Keycloak**-Servers für die **THS-Tools**, sowie für die Konfiguration und Benutzung der drei Schnittstellen der Werkzeuge mit **Keycloak**-basierter Authentifizierung:

- **Einrichtung von Keycloak für die THS-Tools**
 - 0. Installation und Start von Keycloak
 - 1. Realm `ttp` hinzufügen
 - 2. Clients einrichten
 - 3. Rollen anlegen
 - 4. User anlegen und Rollen zuweisen
 - 5. Test und Benutzung von Keycloak
 - 6. Export des `ttp`-Realms
- **Konfiguration der THS-Tools**
 - Anpassung der Rollennamen
 - Verbindungseinstellungen
 - Aktivierung der Authentifizierung
 - Web-Oberfläche
 - FHIR-Gateway
 - SOAP-Webservices
- **Test und Benutzung der Schnittstellen mit Authentifizierung**
 - Web-Oberfläche
 - Bedeutung der Admin-Rolle im Web
 - FHIR-Gateway
 - Bedeutung der Admin-Rolle im FHIR-Gateway
 - Variante 1: Verzicht auf Angabe einer Admin-Rolle
 - Variante 2: Angabe einer gültigen Admin-Rolle
 - SOAP-Webservices
 - Bedeutung der Admin-Rolle in SOAP-Webservices

- [Einbettung von glCS-Formularen in externe Anwendungen](#)
- [Credits](#)
- [License](#)

Einrichtung von **Keycloak** für die **THS-Tools**

0. Installation und Start von **Keycloak**

Keycloak wird als Container-Image für **Docker** angeboten. Wenn nicht schon geschehen, dann muss also zuerst **Docker** installiert werden, z.B. **Docker Desktop**.

Gemäß dieser [Anleitung](#) kann dann **Keycloak** in **Docker** installiert werden. (Das Beispiel-**Realm** `myrealm` braucht dabei natürlich nicht angelegt zu werden.)

Wenn die **THS-Tools** und der **Keycloak**-Server auf der gleichen Maschine laufen sollen, ist zu beachten, dass beiden Anwendungen verschiedene Ports zugewiesen werden. Wenn durch die **THS-Tools** z.B. der Port `8080` schon belegt ist, müsste **Keycloak** einen anderen Port nutzen, z.B. `8081`, was beim Start des Containers beachtet werden muss:

```
docker run -p 8081:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin
quay.io/keycloak/keycloak:19.0.3 start-dev
```

In der Standardinstallation von **Keycloak** darf der zweite Port in `-p 8081:8080` nicht verändert werden, über den ersten Port (im weiteren `{KEYCLOAK_PORT}` genannt) wird **Keycloak** auf dem Dockerhost angesprochen.

Zum Einrichten von **Keycloak** startet man dessen Admin-Konsole unter

http://{KEYCLOAK_HOST}:{KEYCLOAK_PORT}/admin

und meldet sich als Benutzer `admin` mit dem zuvor vergebenen Keycloak-Admin-Passwort an.

Achtung: Ab **Keycloak** v17 wird standardmäßig **kein** `/auth`-Prefix mehr in den URLs verwendet, vorher enthielten alle URLs diesen Prefix: `http://{KEYCLOAK_HOST}:{KEYCLOAK_PORT}/auth/...`

1. **Realm** `ttp` hinzufügen

Die Admin-Konsole startet im **Realm Master**, erkennbar links oben. Dieses ist nur zur Verwaltung von **Keycloak** selbst bestimmt. Öffnet man die Kombobox mit den **Realms**, dann kann man mit **Create Realm**, ein neues **Realm** erstellen. Das **Realm** für alle **THS-Tools** bekommt z.B. den Namen `ttp`.

2. Clients einrichten

Für die tokenbasierte Authentifizierung benutzt man am besten gesicherte (confidential) Clients. Dabei steht es einem frei, **nur einen Client** (z.B. `ths`) für alle Arten von Zugriff einzurichten, oder feinkörniger **für jede Zugriffsart einen eigenen Client** zu verwenden (z.B. `web`, `soap`, `fhir`). Als authentifizierter Client (bei Kenntnis des sogenannten Client-Secrets) darf man dann bei **Keycloak** Benutzertokens erfragen (`token request`) und prüfen lassen (`token introspection`).

Um Clients anzulegen, navigiert man am linken Rand zu **Clients**, und kommt so in den entsprechenden Bereich. Mit dem Button **Create Client** erzeugt man einen neuen Client, wählt als **Client Type** **OpenID Connect** und vergibt die gewünschte **Client ID**. Im nächsten Schritt muss für diesen Client **Client authentication** und **Service accounts roles** aktiviert werden - so erhält man nach **Save** einen **confidential** Client:


[Clients](#) > [Create client](#)


Create client

Clients are applications and services that can request authentication of a user.


1 General Settings


2 **Capability config**


Client authentication  On


Authorization  Off


Authentication flow Standard flow 

Direct access grants 

Implicit flow 






Service accounts roles 

OAuth 2.0 Device Authorization Grant 

OIDC CIBA Grant 

Im nächsten Schritt werden die Zugriffseinstellungen für den Client konfiguriert. Wichtig sind die **Root Url**, die **Home URL** und die **Valid Redirect URIs**. Für den Anfang oder zum Testen können erstere auch leer bleiben und für die Redirect URI kann ein ***** gesetzt werden.



Access settings

Root URL Home URL Valid redirect URIs [+ Add valid redirect URIs](#)Valid post logout
redirect URIs [+ Add valid post logout redirect URIs](#)Web origins [+ Add web origins](#)Admin URL 

Im Unterbereich **Credentials** (für diesen Client) findet man das zugehörige **Client secret** (z.B. `5TQNIeAOr0mVbx3rUaUWPa1yvRVbVfG`). Das wird im weiteren Verlauf noch benötigt und muss deshalb kopiert werden.


[Clients](#) > [Client details](#)**X** [OpenID Connect](#)

Clients are applications and services that can request authentication of a user.

[Settings](#)[Keys](#)[Credentials](#)[Roles](#)[Client scopes](#)[Service accounts roles](#)[Sessions](#)[Advanced](#)Client Authenticator Client Id and Secret [Save](#)

Client secret

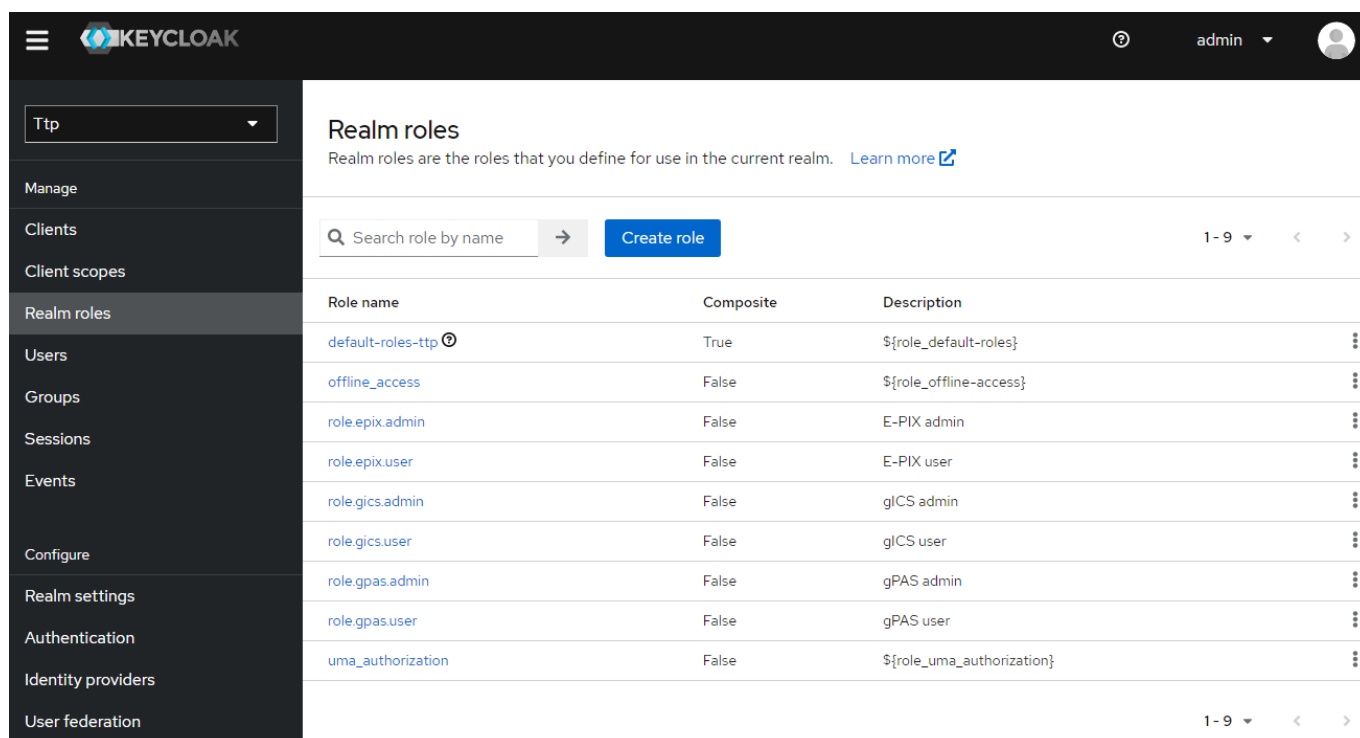
5TQNIeAOr0mVbx3rUaUWPalyvRVbVfG

[Regenerate](#)Registration access
token [Regenerate](#)

Diese Prozedur wiederholt man für alle einzurichtenden Clients.

3. Rollen anlegen

Im einfachsten Fall arbeitet man mit **Realm**-Rollen:



The screenshot shows the Keycloak Admin Console interface. The left sidebar contains navigation options: Manage, Clients, Client scopes, Realm roles (selected), Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'Realm roles' and includes a search bar, a 'Create role' button, and a table of roles.

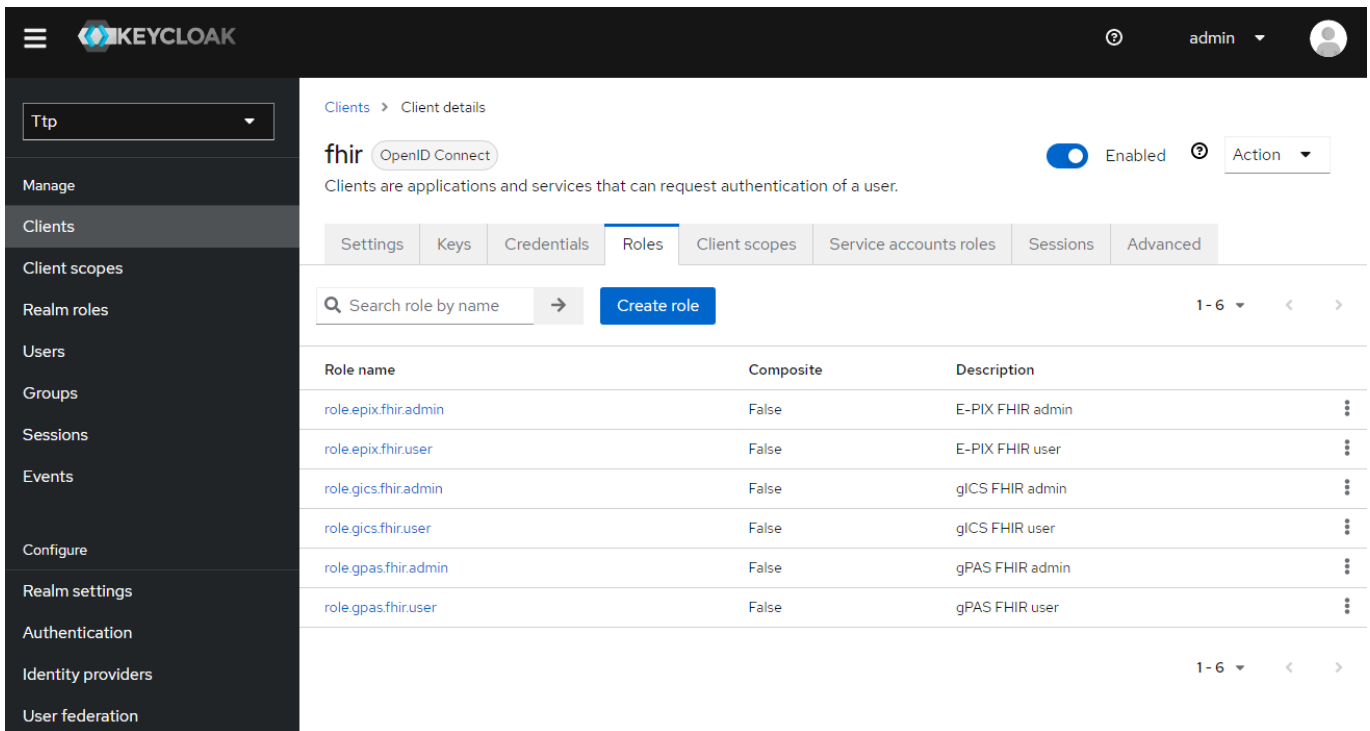
Role name	Composite	Description
default-roles-ttp	True	`\${role_default-roles}`
offline_access	False	`\${role_offline-access}`
role.epix.admin	False	E-PIX admin
role.epix.user	False	E-PIX user
role.gics.admin	False	gICS admin
role.gics.user	False	gICS user
role.gpas.admin	False	gPAS admin
role.gpas.user	False	gPAS user
uma_authorization	False	`\${role_uma_authorization}`

Die Namen der Rollen sind standardmäßig nach einem einfachen Schema aufgebaut und für alle Clients bzw. Schnittstellen (**WEB**, **FHIR** und **SOAP**) gleich:

Komponente	Admin-Rolle	User-Rolle
gICS	<code>role.gics.admin</code>	<code>role.gics.user</code>
gPAS	<code>role.gpas.admin</code>	<code>role.gpas.user</code>
E-PIX	<code>role.epix.admin</code>	<code>role.epix.user</code>
Dispatcher	<code>role.dispatcher.admin</code>	<code>role.dispatcher.user</code>

Allerdings können die Namen der Rollen für die Absicherung der Zugriffe über **FHIR** und **SOAP** frei konfiguriert werden (die für **WEB** sind festgelegt).

Die Anpassbarkeit der Rollennamen erlaubt den Zugriff auf die **THS-Tools** über die verschiedenen Schnittstellen feinkörnig differenziert zu konfigurieren, insbesondere bei Verwendung clientbasierter Rollen:



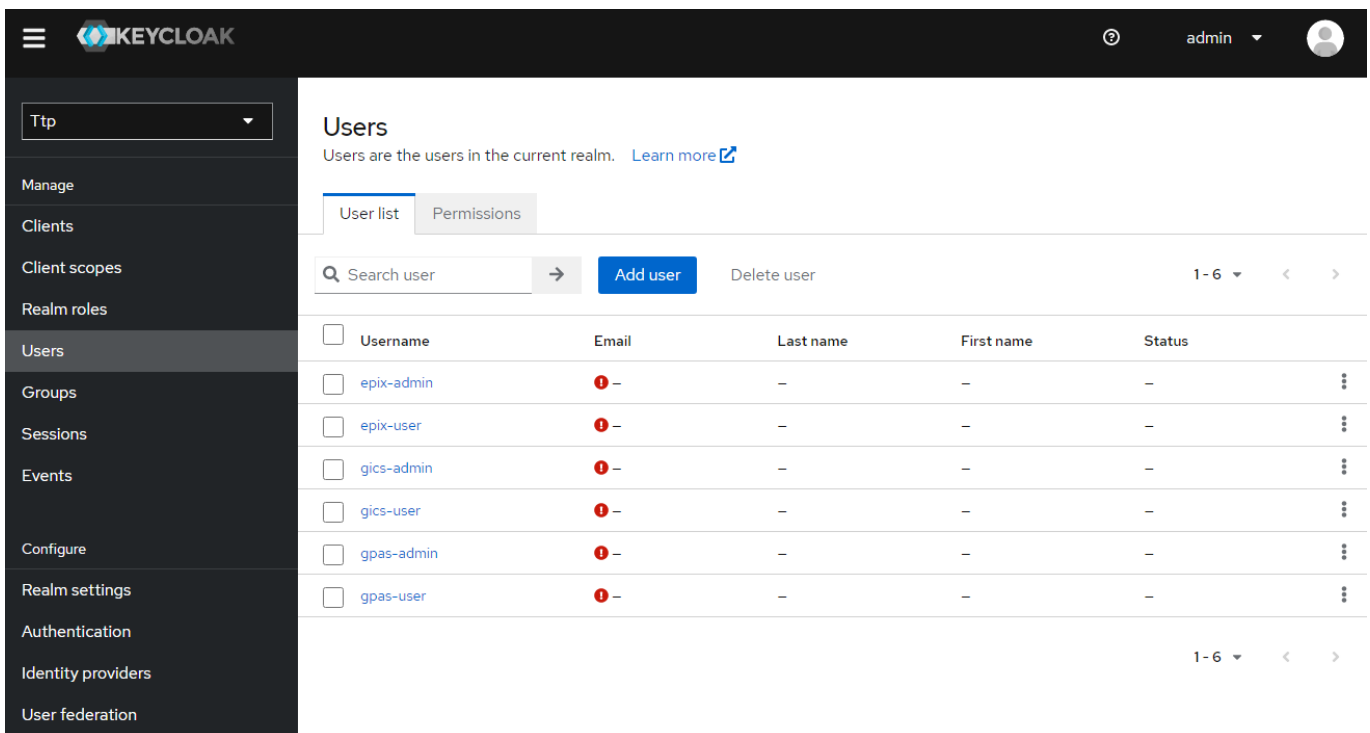
Keycloak Admin Console - Client details for 'fhir' (OpenID Connect). The 'Roles' tab is active, showing a list of roles. The roles are:

Role name	Composite	Description
role.epix.fhir.admin	False	E-PIX FHIR admin
role.epix.fhir.user	False	E-PIX FHIR user
role.gics.fhir.admin	False	gICS FHIR admin
role.gics.fhir.user	False	gICS FHIR user
role.gpas.fhir.admin	False	gPAS FHIR admin
role.gpas.fhir.user	False	gPAS FHIR user

Wie die Rollennamen in den **THS-Tools** konfiguriert werden können, wird später im [Abschnitt über deren Konfiguration](#) beschrieben.

4. User anlegen und Rollen zuweisen

Nun müssen noch je Werkzeug zwei Nutzer für den Zugriff auf diese angelegt werden, je ein (einfacher) Benutzer und ein Administrator:

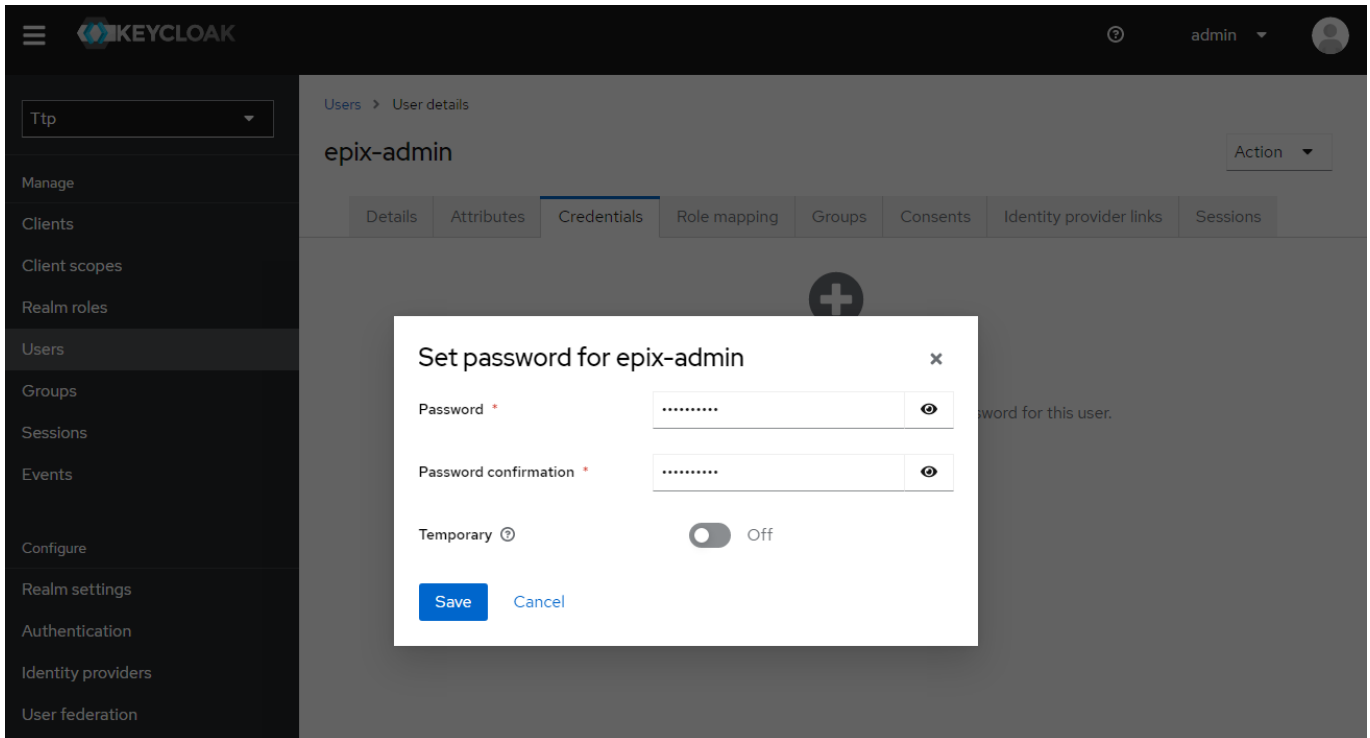


Keycloak Admin Console - Users configuration page. The 'User list' tab is active, showing a list of users. The users are:

Username	Email	Last name	First name	Status
epix-admin	!	-	-	-
epix-user	!	-	-	-
gics-admin	!	-	-	-
gics-user	!	-	-	-
gpas-admin	!	-	-	-
gpas-user	!	-	-	-

Dabei muss das Feld **Required User Action** leer bleiben, lediglich der jeweilige **Username** und optionale Informationen werden eingetragen.

Anschließend vergibt man für die Nutzer unter **Credentials** nicht-temporäre Passwörter (die man sich natürlich merkt):



Und schließlich ordnet man den Nutzern unter **Role Mappings** die entsprechenden Rollen zu (auf jeden Fall die **Realm**-Rollen und ggf. auch die clientbasierten Rollen), z.B.:

Werkzeug	Username	Assigned Role
gICS	gics-user	role.gics.user, ...
gICS	gics-admin	role.gics.user, role.gics.admin, ...
gPAS	gpas-user	role.gpas.user, ...
gPAS	gpas-admin	role.gpas.user, role.gpas.admin, ...
E-PIX	epix-user	role.e-pix.user, ...
E-PIX	epix-admin	role.epix.user, role.epix.admin, ...
Dispatcher	disp-user	role.dispatcher.user, ...
Dispatcher	disp-admin	role.dispatcher.user, role.dispatcher.admin, ...

The screenshot shows the Keycloak user details page for the user 'epix-admin'. The 'Role mapping' tab is selected, displaying a table of roles assigned to the user. The table has columns for 'Name', 'Inherited', and 'Description'. The roles listed are:

Name	Inherited	Description
default-roles-ttp	False	`\${role_default-roles}`
role.epix.user	False	E-PIX user
role.epix.admin	False	E-PIX admin
fhir role.epix.fhir.admin	False	E-PIX FHIR admin
fhir role.epix.fhir.user	False	E-PIX FHIR user

Das Zuordnen von clientbasierten Rollen ist etwas versteckt, dazu muss im Assign-Dialog **Filter by clients** ausgewählt werden.

The screenshot shows the 'Assign roles to epix-admin account' dialog box. The 'Filter by roles' dropdown is open, and 'Filter by clients' is selected. The dialog displays a table of roles available for assignment:

Name	Description
offline_access	`\${role_offline-access}`
role.gics.admin	gICS admin
role.gics.user	gICS user
role.gpas.admin	gPAS admin
role.gpas.user	gPAS user
uma_authorization	`\${role_uma_authorization}`

5. Test und Benutzung von Keycloak

Für den Zugriff auf die durch **Keycloak** abgesicherten Komponenten über **FHIR** oder **SOAP** benötigt man neben den Client-Secrets auch einen **Access-Token**. Den bekommt man mit einem **POST**-Request vom **Keycloak-Server** (`token-request-endpoint`):

```
POST /realms/ttp/protocol/openid-connect/token HTTP/1.1
Host: {KEYCLOAK_HOST}:{KEYCLOAK_PORT}
Content-Type: application/x-www-form-urlencoded
Content-Length: 128
```



```
username={USERNAME}&password={PASSWORD}&grant_type=password&client_id=
{CLIENT_ID}&client_secret={CLIENT_SECRET}
```

Achtung: ab **Keycloak** v17 ohne `/auth`-Prefix!

Hierbei wird, wie man sieht, auch das früher schon kopierte **Secret** des Clients benötigt. Die Variablen sind entsprechend zu ersetzen: beispielsweise für **gICS** wären

- der `{USERNAME}` dann `gics-user`,
- das `{PASSWORD}`, das für `gics-user` vergebene Passwort,
- die `{CLIENT_ID}` (z.B. `fhir`)
- und das `{CLIENT_SECRET}`, das zuvor gemerkte **Secret** des Clients.

Mit `curl` könnte das etwa so aussehen:

```
curl -d 'username={USERNAME}&password={PASSWORD}&grant_type=password&client_id=
{CLIENT_ID}&client_secret={CLIENT_SECRET}' \
      http://{KEYCLOAK_HOST}:{KEYCLOAK_PORT}/realms/ttp/protocol/openid-
connect/token
```

Achtung: ab **Keycloak** v17 ohne `/auth`-Prefix!

...und die (hier etwas aufgehübschte) Antwort mit dem **Access-Token** darauf wäre:

```
{
  "access_token" :
  "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXLTJ5IiwiaWF0IjoiYTA6IC01M3drWmNua0h...",
  "expires_in" : 300,
  "refresh_expires_in" : 1800,
  "refresh_token" :
  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ5IiwiaWF0IjoiYTA6IC01M3drWmNua0h...",
  "token_type" : "Bearer",
  "not-before-policy" : 0,
  "session_state" : "a343bc8f-7616-4deb-a635-293a91556b96",
  "scope" : "email profile"
}
```

Das **Access-Token** (später `{ACCESS_TOKEN}` genannt) wird bei allen **FHIR**- und **SOAP**-Requests als sogenanntes **Bearer-Token** im **Authorization-Header** benötigt.

6. Export des Realms

An dieser Stelle empfiehlt es sich, die Konfiguration des **Realms** `ttp` in eine **JSON**-Datei zu exportieren, um sie bei Bedarf wieder importieren zu können. Zu beachten ist dabei allerdings, dass die angelegten Nutzer, deren Konfigurationen und die **Client secrets** nicht mit exportiert werden. Sie müssen nach einem Import neu erstellt bzw. neu generiert werden.

Ein Export des (beispielhaften) **ttp-Realms** mit den Clients **ths**, **web**, **fhir** und **soap** (inklusive jeweiliger Client-Roles) ist im Dockerpaket enthalten und kann nach Import als Basis verwendet und nach Belieben angepasst werden.

Konfiguration der **THS-Tools**

Detaillierte Beschreibungen der Konfiguration der Authorisierung und deren **Keycloak**-bezogenen Details für die **docker-compose**-Pakete der jeweiligen Werkzeuge finden sich in der separaten ReadMe-Datei (**README_<TOOL>.md**). Hier folgt deshalb nur ein kurzer Überblick:

Anpassung der Rollennamen

Wie gesagt, die Namen der Rollen für **FHIR** und **SOAP** sind frei konfigurierbar, was besonders sinnvoll bei der Verwendung clientbasierter Rollen ist. Die Konfiguration der Rollennamen kann über Docker-ENV-Variablen erfolgen:

Standardname der Rolle	ENV-Variable in Docker für FHIR	ENV-Variable in Docker für SOAP
<code>role.epix.user</code>	<code>TTP_FHIR_KEYCLOAK_ROLE_EPIX_USER</code>	<code>TTP_SOAP_KEYCLOAK_ROLE_EPIX_USER</code>
<code>role.epix.admin</code>	<code>TTP_FHIR_KEYCLOAK_ROLE_EPIX_ADMIN</code>	<code>TTP_SOAP_KEYCLOAK_ROLE_EPIX_ADMIN</code>
<code>role.gics.user</code>	<code>TTP_FHIR_KEYCLOAK_ROLE_GICS_USER</code>	<code>TTP_SOAP_KEYCLOAK_ROLE_GICS_USER</code>
<code>role.gics.admin</code>	<code>TTP_FHIR_KEYCLOAK_ROLE_GICS_ADMIN</code>	<code>TTP_SOAP_KEYCLOAK_ROLE_GICS_ADMIN</code>
<code>role.gpas.user</code>	<code>TTP_FHIR_KEYCLOAK_ROLE_GPAS_USER</code>	<code>TTP_SOAP_KEYCLOAK_ROLE_GPAS_USER</code>
<code>role.gpas.admin</code>	<code>TTP_FHIR_KEYCLOAK_ROLE_GPAS_ADMIN</code>	<code>TTP_SOAP_KEYCLOAK_ROLE_GPAS_ADMIN</code>

Die Verwendung clientbasierter Rollen kann man (mit **true** oder **false**) auch über Docker-ENV-Variablen steuern:

```
TTP_KEYCLOAK_USE_RESOURCE_ROLE_MAPPINGS=true      # für alle Schnittstellen
TTP_FHIR_KEYCLOAK_USE_RESOURCE_ROLE_MAPPINGS=true # nur für FHIR
TTP_SOAP_KEYCLOAK_USE_RESOURCE_ROLE_MAPPINGS=true # nur für SOAP
TTP_WEB_KEYCLOAK_USE_RESOURCE_ROLE_MAPPINGS=true  # nur für WEB
```

Verbindungseinstellungen

Die Docker-ENV-Variablen für die grundsätzlichen Einstellungen für die Verbindung zum **Keycloak**-Server :

```
TTP_KEYCLOAK_REALM=ttp
TTP_KEYCLOAK_CLIENT_ID={CLIENT_ID}
TTP_KEYCLOAK_SERVER_URL=http://{KEYCLOAK_HOST}:{KEYCLOAK_PORT}
TTP_KEYCLOAK_SSL_REQUIRED=none # oder external, all
TTP_KEYCLOAK_CLIENT_SECRET={CLIENT_SECRET}
```

```
TTP_KEYCLOAK_USE_RESOURCE_ROLE_MAPPINGS=false  
TTP_KEYCLOAK_CONFIDENTIAL_PORT=0 # ohne confidential port oder z.B. 8443
```

Die Verbindungsparameter können auch schnittstellenspezifisch eingestellt werden, insbesondere die Einstellungen für die `{CLIENT_ID}` und das `{CLIENT_SECRET}` sind dabei von Bedeutung:

```
TTP_WEB_KEYCLOAK_CLIENT_ID=web  
TTP_WEB_KEYCLOAK_CLIENT_SECRET={CLIENT_SECRET_WEB}  
TTP_FHIR_KEYCLOAK_CLIENT_ID=fhir  
TTP_FHIR_KEYCLOAK_CLIENT_SECRET={CLIENT_SECRET_FHIR}  
TTP_SOAP_KEYCLOAK_CLIENT_ID=soap  
TTP_SOAP_KEYCLOAK_CLIENT_SECRET={CLIENT_SECRET_SOAP}
```

Weitere schnittstellenspezifische Variablennamen und Details befinden sich im jeweiligen `README_<TOOL>.md`.

Aktivierung der Authentifizierung

Auch die Aktivierung erfolgt über Docker-ENV-Variablen:

WEB

```
TTP_EPIX_WEB_AUTH_MODE=keycloak  
TTP_GICS_WEB_AUTH_MODE=keycloak  
TTP_GPAS_WEB_AUTH_MODE=keycloak
```

SOAP

```
TTP_EPIX_SOAP_KEYCLOAK_ENABLE=true  
TTP_GICS_SOAP_KEYCLOAK_ENABLE=true  
TTP_GPAS_SOAP_KEYCLOAK_ENABLE=true
```

FHIR

```
TTP_FHIR_KEYCLOAK_ENABLE=true
```

Test und Benutzung der Schnittstellen mit Authentifizierung

Web-Oberfläche

Test bzw. Benutzung der mit **Keycloak** abgesicherten Web-Oberfläche ist denkbar einfach: Der Benutzer wird an einen Login-Dialog des **Keycloak**-Servers weitergeleitet, wo er seine Credentials (Benutzername und

Passwort) eingibt. Entsprechend der Rollen des verwendeten Benutzers ist er dann als **admin** oder **user** autorisiert.

Anmerkung: Anders als das **FHIR-Gateway** und die **SOAP-Webservices** kann die **Web-Oberfläche** alternativ **gRAS**-basiert abgesichert werden. (Details wieder dazu im [README_<TOOL>.md](#))

Bedeutung der Admin-Rolle im Web

Ein detaillierte Beschreibung der Aktionen im Web, die als **user** erlaubt sind und jener, für die die **admin**-Rolle benötigt wird, findet sich im jeweiligen Handbuch des Werkzeugs.

FHIR-Gateway

Für die Benutzung des **FHIR-Gateway**, nun mit Authentifizierung über **Keycloak**, muss das aus dem Token-Request entnommene **Access-Token** als Authorization-Header **Authorization: Bearer {ACCESS_TOKEN}** bei jedem Request an das **FHIR-Gateway** im Header mitgegeben werden. Um die korrekte Funktion zu testen, kann man nun in einem **GET**-Request die Metadaten der **FHIR**-Schnittstelle (z.B. für **gICS**) anfordern:

```
GET /ttp-fhir/fhir/gics/metadata HTTP/1.1
Host: {GICS_HOST}:{GICS_PORT}
Authorization: Bearer {ACCESS_TOKEN}
```

Mit **curl** könnte das für **gICS** beispielsweise so aussehen:

```
curl -X GET http://{GICS_HOST}:{GICS_PORT}/ttp-fhir/fhir/gics/metadata \
--header "Authorization: Bearer {ACCESS_TOKEN}"
```

Die erwartete (hier etwas aufgehübschte) Antwort darauf sollte etwa so aussehen:

```
{
  "resourceType": "CapabilityStatement",
  "status": "active",
  "date": "2021-06-03T20:54:47+02:00",
  "publisher": "Not provided",
  "kind": "instance",
  "software": {
    "name": "HAPI FHIR Server",
    "version": "5.0.0"
  },
  "implementation": {
    "description": "HAPI FHIR",
    "url": "http://{GICS_HOST}:{GICS_PORT}/ttp-fhir/fhir/gics"
  },
  "fhirVersion": "4.0.1",
  "format": [
    "application/fhir+xml",
    "application/fhir+json"
  ]
}
```

```
],
"rest": [
  {
    "mode": "server",
    "resource": [
      ...
    ],
    "operation": [
      {
        "name": "allConsentsForDomain",
        "definition": "http://{GICS_HOST}:{GICS_PORT}/ttp-
fhir/fhir/gics/OperationDefinition/-s-allConsentsForDomain"
      },
      ...
    ]
  }
]
}
```

Wenn dann der **GET**-Request ohne gültigen Authorization-Header `Authorization: Bearer {ACCESS_TOKEN}`:

```
curl -X GET http://{GICS_HOST}:{GICS_PORT}/ttp-fhir/fhir/gics/metadata
```

auch noch einen Fehler liefert:

```
Unauthorised Access to Protected Resource (No OAuth Token supplied in
Authorization Header)
```

dann bedeutet das, dass die **Keycloak**-basierte Authentifizierung für **FHIR** wie gewünscht funktioniert.

Bedeutung der Admin-Rolle im FHIR-Gateway

Anmerkung: `admin`-Rollen im **FHIR-Gateway** sind für **gICS**, **gPAS** und **E-PIX** vorbereitet, werden derzeit nur von **gICS** verwendet

Die Angabe einer `admin`-Rolle wird derzeit nur bei der Generierung von **FHIR-ConsentPatient-Resources** berücksichtigt. Da ein Consent durchaus mehrere `SignerIds` besitzen kann und diese Informationen nicht jedermann zugänglich sein sollten, kann der Umfang der im FHIR Consent Patient exportierten Identifier auf diese Weise reglementiert werden.

Variante 1: Verzicht auf Angabe einer Admin-Rolle

Ist keine `admin`-Rolle konfiguriert oder ist im Token keine oder eine fehlerhafte `admin`-Rolle angegeben, wird als Identifier der *ConsentPatient-Resource* **nur die SafeSignerId** verwendet. Die **FHIR UUID** des *ConsentPatient* entspricht somit der **FHIR UUID** der *SafeSignerId* (Details dazu im [gics-Handbuch](#))

Variante 2: Angabe einer gültigen Admin-Rolle

Ist in **Keycloak** eine **admin**-Rolle konfiguriert und verweist das im Request verwendete Token korrekt auf diese **admin**-Rolle, enthält die *ConsentPatient-Resource* **ALLE für den Consent relevanten SignerIds** als separate Identifier. Die **FHIR UUID** des *ConsentPatient* entspricht der **FHIR UUID** der *SafeSignerId* (Details dazu im [gics-Handbuch](#)).

SOAP-Webservices

Analog zum **FHIR-Gateway** muss auch für die Benutzung von **SOAP-Webservices** mit Authentifizierung über **Keycloak**, das aus dem Token-Request entnommene **Access-Token** als Authorization-Header

Authorization: Bearer {ACCESS_TOKEN} bei jedem Request an die **SOAP**-Schnittstelle im Header übergeben werden werden. Um die korrekte Funktion zu testen, kann man nun in einem **POST**-Request beispielsweise die *SignerIdTypes* des **gICS** erfragen:

```
POST /gics/gicsService HTTP/1.1
Host: {GICS_HOST}:{GICS_PORT}
Authorization: Bearer {ACCESS_TOKEN}
Content-Type: application/xml
Content-Length: 268

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cm2="http://cm2.ttp.ganimed.icmvc.emau.org/">
  <soapenv:Header/>
  <soapenv:Body>
    <cm2:listSignerIdTypes>
      <domainName>MII</domainName>
    </cm2:listSignerIdTypes>
  </soapenv:Body>
</soapenv:Envelope>
```

Mit **curl** könnte das für **gICS** beispielsweise so aussehen:

```
curl -X POST http://{GICS_HOST}:{GICS_PORT}/gics/gicsService \
  --header "Authorization: Bearer {ACCESS_TOKEN}" \
  --data \
  <soapenv:Envelope \
    xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" \
    xmlns:cm2=\"http://cm2.ttp.ganimed.icmvc.emau.org/\"> \
  <soapenv:Header/> \
  <soapenv:Body> \
    <cm2:listSignerIdTypes> \
      <domainName>MII</domainName> \
    </cm2:listSignerIdTypes> \
  </soapenv:Body> \
</soapenv:Envelope> \
```

Die erwartete (hier etwas aufgehübschte) Antwort darauf sollte etwa so aussehen:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:listSignerIdTypesResponse
xmlns:ns2="http://cm2.ttp.ganimed.icmvc.emau.org/">
      <return>
        <signerIdTypes>
          <fhirID>2e040fc3-0e91-4148-a8bd-0ff2d9b25982</fhirID>
          <createTimestamp>2022-02-
01T10:13:27.789+01:00</createTimestamp>
          <name>Pseudonym</name>
          <updateTimestamp>2022-02-
01T10:13:27.789+01:00</updateTimestamp>
        </signerIdTypes>
      </return>
    </ns2:listSignerIdTypesResponse>
  </soap:Body>
</soap:Envelope>
```

Wenn dann der gleiche **POST**-Request ohne gültigen Authorization-Header auch noch einen Fehler liefert:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Unauthorised Access to Protected Resource (No OAuth Token
supplied in Authorization Header)</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

dann bedeutet das, dass die **Keycloak**-basierte Authentifizierung für **SOAP** wie gewünscht funktioniert.

Bedeutung der Admin-Rolle in SOAP-Webservices

Die Werkzeuge bieten mehrere **SOAP**-Schnittstellen, in der Regel (mindestens) eine für Routearbeiten und eine für Konfiguration und Management. Standardmäßig sind erstere für (einfache) Benutzer zugänglich, letztere hingegen nur für Administratoren.

Dies kann aber über Docker-ENV-Variablen konfiguriert werden (hier mit den Standardeinstellungen):

```
TTP_EPIX_SOAP_ROLE_USER_SERVICES=/epix/epixService,/epix/epixServiceWithNotificati
on
TTP_EPIX_SOAP_ROLE_ADMIN_SERVICES=/epix/epixManagementService
TTP_GICS_SOAP_ROLE_USER_SERVICES=/gics/gicsService,/gics/gicsServiceWithNotificati
on
TTP_GICS_SOAP_ROLE_ADMIN_SERVICES=/gics/gicsManagementService,/gics/gicsFhirServic
```

```
e
TTP_GPAS_SOAP_ROLE_USER_SERVICES=/gpas/gpasService,/gpas/gpasServiceWithNotificati
on
TTP_GPAS_SOAP_ROLE_ADMIN_SERVICES=/gpas/DomainService
```

Einbettung von gICS-Formularen in externe Anwendungen

Seit Version 2024.3.0 können gICS-Formulare in externe Anwendungen eingebettet werden. Für eine mit **Keycloak** abgesicherte Web-Oberfläche ist automatisch auch der Zugriff auf diese Formulare abgesichert: der Benutzer externer Anwendungen mit eingebetteten gICS-Formularen wird an einen Login-Dialog des **Keycloak**-Servers weitergeleitet, wo er seine Credentials (Benutzername und Passwort) eingibt.

Für den Zugriff auf die entsprechenden Endpunkte, die sich alle im Kontext `html/embedded` befinden, benötigt der Nutzer jedoch die Rolle `role.gics.embedded`, die für diesen im **Keycloak**-Server entsprechend konfiguriert sein muss.

Credits 'Keycloak for for TTP-Tools and TTP-FHIR-Gateway'

Implementation and documentation: P. Penndorf, F.-M. Moser, M. Bialke, R. Schuldt, A. Blumentritt

Additional Information

The gICS was developed by the University Medicine Greifswald and published in 2014 as part of the [MOSAIC-Project](#) (funded by the DFG HO 1937/2-1). Selected functionalities of gICS were developed as part of the following research projects:

- MAGIC (funded by the DFG HO 1937/5-1)
- MIRACUM (funded by the German Federal Ministry of Education and Research 01ZZ1801M)
- NUM-CODEX (funded by the German Federal Ministry of Education and Research 01KX2021)

Credits

Concept and implementation: L. Geidel

Web-Client: A. Blumentritt, M. Bialke, F.M.Moser

Docker: R. Schuldt

TTP-FHIR Gateway für gICS: M. Bialke, P. Penndorf, L. Geidel, S. Lang, F.M. Moser

License

License: AGPLv3, <https://www.gnu.org/licenses/agpl-3.0.en.html>

Copyright: 2014 - 2024 University Medicine Greifswald

Contact: <https://www.ths-greifswald.de/kontakt/>

Publications

- <https://doi.org/10.1186/s12911-022-02081-4>
- <https://rdcu.be/b5Yck>
- <https://rdcu.be/6LJd>

- <https://dx.doi.org/10.3414/ME14-01-0133>
- <https://dx.doi.org/10.1186/s12967-015-0545-6>

Supported languages

German, English