IHE SUISSE

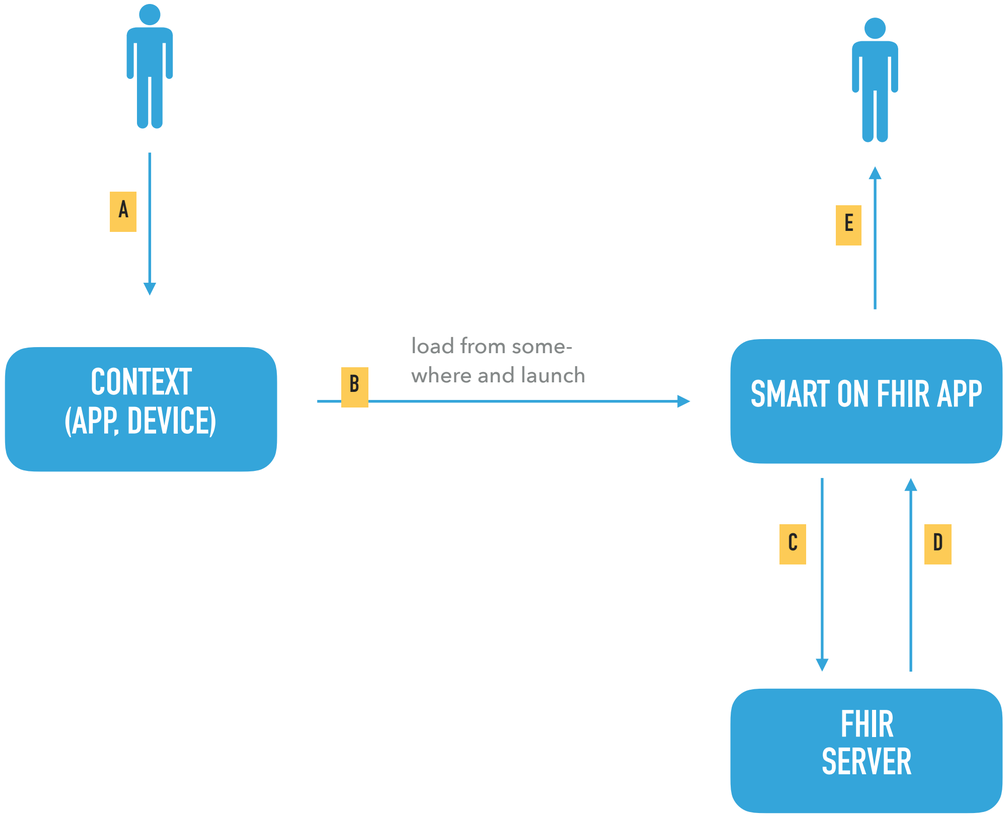
Integrating
the Healthcare
Enterprise

SMART ON FHIR

BASIC CONCEPTS

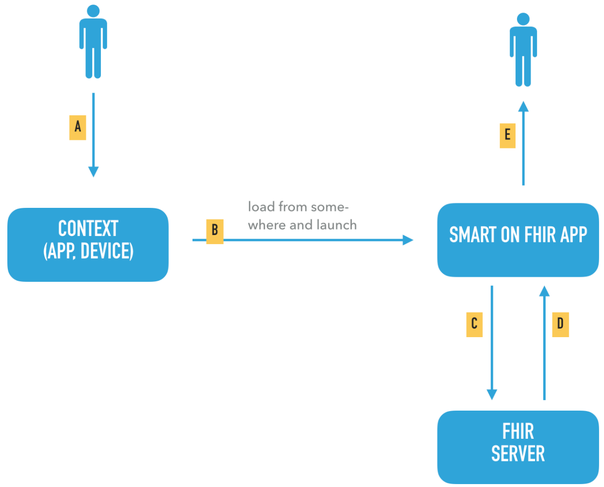
- SMART on FHIR is a specification for reusable health Apps using SMART® und FHIR technology.
  - FHIR: next level HL7 standard.
  - SMART® : Substitutable Medical Apps and Reusable Technology.
- To be used in modular medical information systems or patient apps.

BASIC CONCEPTS



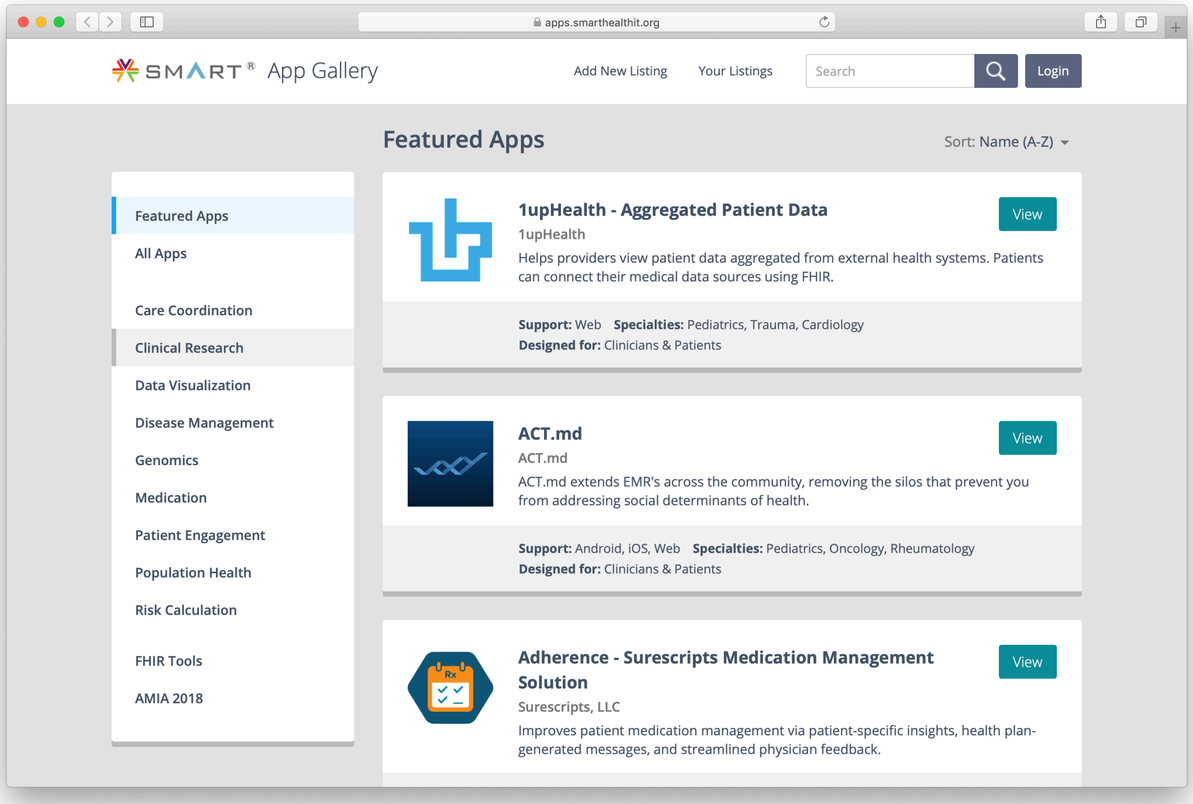
BASIC CONCEPTS

- Step A:
  - User selects a view in her information system, starts a web application or an app on her device.
- Step B:
  - The SMART on FHIR app is loaded from a (remote) server or from the device and launched.
- Step C:
  - The SMART on FHIR app launches and requests data from the FHIR server.
- Step D:
  - The FHIR server responds the data.
- Step E:
  - The SMART on FHIR app presents data to the user and provides actions, a.s.o.



SMART APP GALLERY

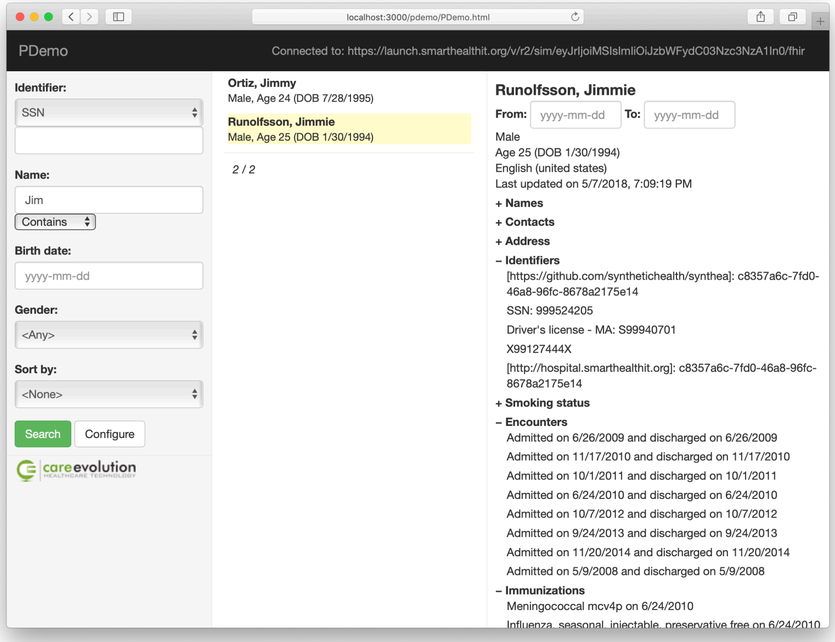
- There is a growing community providing SMART on FHIR apps.
- Most if not all open source and available under free of charge license on Github;
- See SMART App Gallery on https://apps.smarthealthit.org



P DEMO EXAMPLE

- P Demo:
  - An open source SMART on FHIR app published on Github.
  - Angular JS based single page Javascript application.
  - Embedded in a small web application for this demo.

```
<a href="/pdemo/PDemo.html?iss=https://
launch.smarthealthit.org/v/r2/sim/
eyJrIjoiMSIsImIiOiJzbWFydC03Nzc3NzA1In0/
fhir">
```



MY EDUCATIONAL DEMO

- Educational demo:
  - A Javascript server side Web Application app using the FHIR Client Library from smarthealthit.org.
  - Where the FHIR Client Library performs all the necessary steps for authorization.
  - And provides an API to access the resources on the FHIR server.

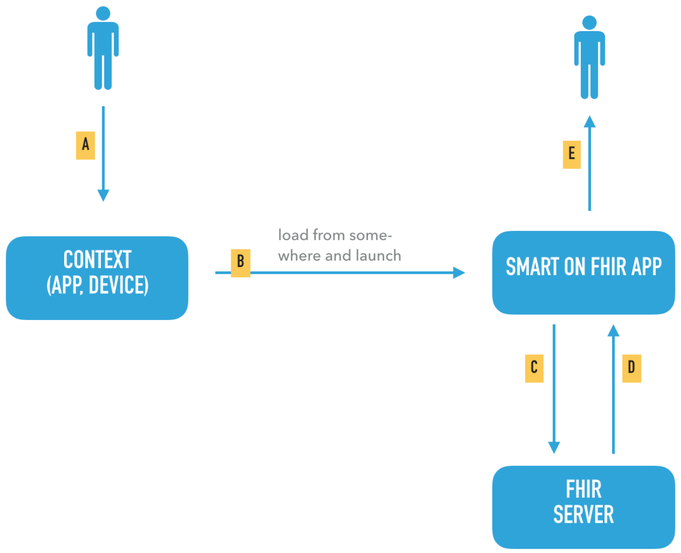
```
// The settings that we use to connect to our SMART on FHIR server
const smartSettings = {
  clientId: "my-client-id",
  redirectUri: "/fhir/app",
  scope: "launch/patient patient/*.read openid fhirUser",
  iss: "https://launch.smarthealthit.org/v/r2/sim/eyJrIjoiMSIsImIiOiJzbWFydC03Nzc3NzA1In0/fhir"
};

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('fhir', { title: 'FHIR' });
});

// LAUNCH:
router.get("/launch", (req, res, next) => {
  smart(req, res).authorize(smartSettings).catch(next);
});

// APP: The app lives at this redirect_uri. After waiting for "ready()", the
// client instance that can be used to query the fhir server.
router.get("/app", (req, res) => {
  // console.log(res);
  smart(req, res).ready().then(client => appHandler(client, res));
});

// FHIR handler
async function appHandler(client, res) {

  const data = await (
    client.patient.id ? client.patient.read() : client.request("Patient")
  );
  // TODO: data are the patients stored on server. Create a table view to display.
  // for now just send the JSON to the browser
  res.type("json").send(JSON.stringify(data, null, 1));
}
```

AUTHENTICATION & AUTHORIZATION

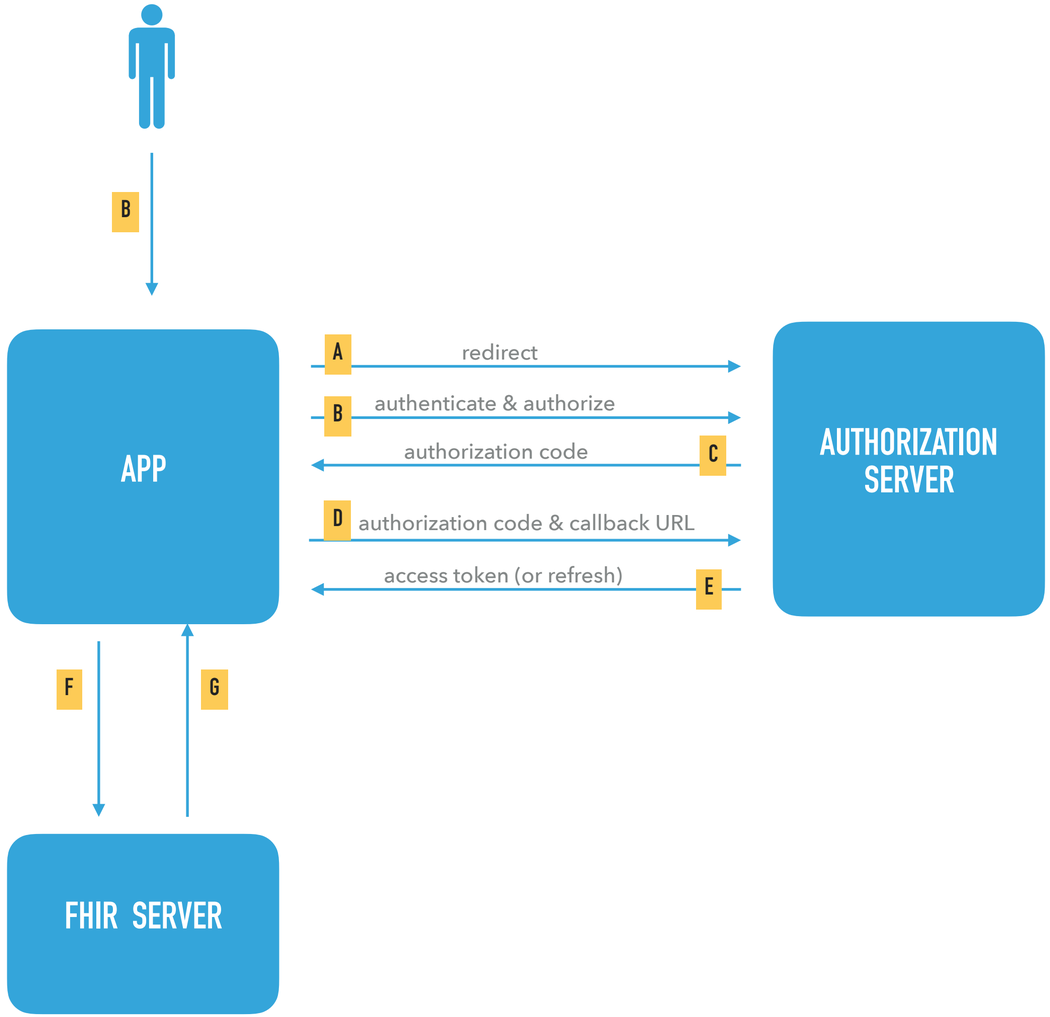
SMART ON FHIR

SMART AUTHORIZATION

- Accessing FHIR resources on a remote FHIR server is straight forward.
- Basic HTTP calls are all we need to retrieve or alter FHIR resources.
- The challenge solved by SMART in FHIR is to authorize the access to the resources on the remote FHIR server.

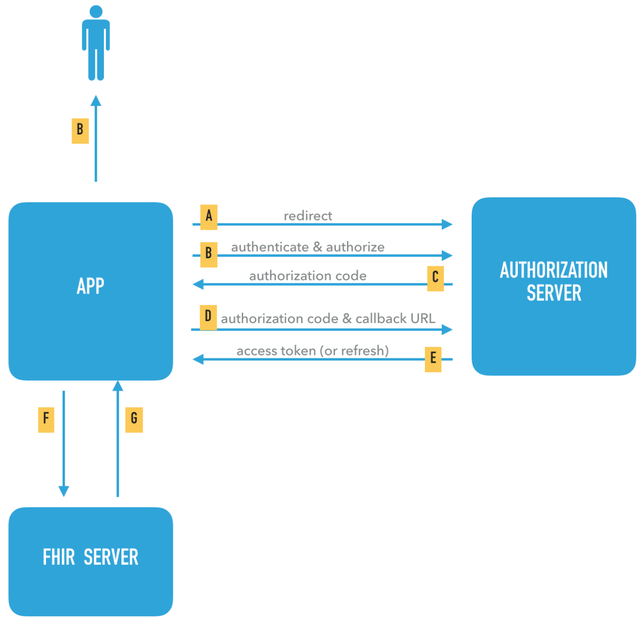


SMART AUTHORIZATION FLOW



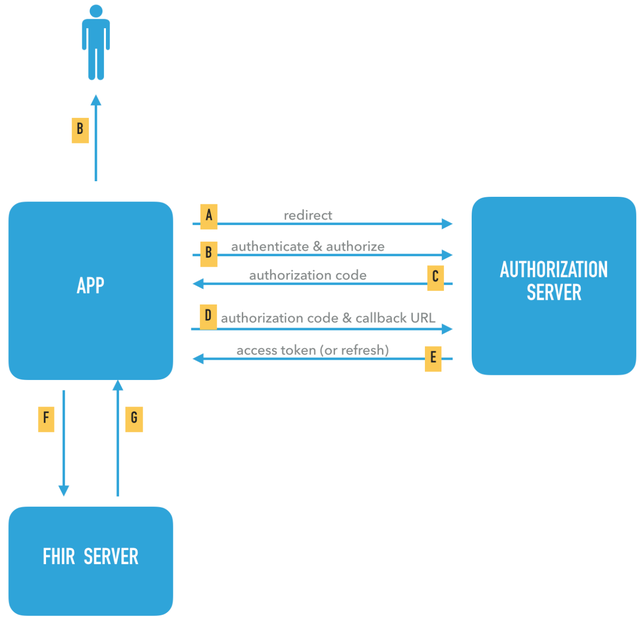
SMART AUTHORIZATION FLOW

- Launch:
  - When launching the app parameter are required to point to the authorization and resource endpoints.
- Step A:
  - The app redirects to the authorization server where the app authenticates with client ID and optional secret.
- Step B:
  - The user as resource owner authenticates at the authorization server and authorizes access to certain scopes (Optional).
- Step C:
  - The authorization server sends an authorization code to the app callback URL.



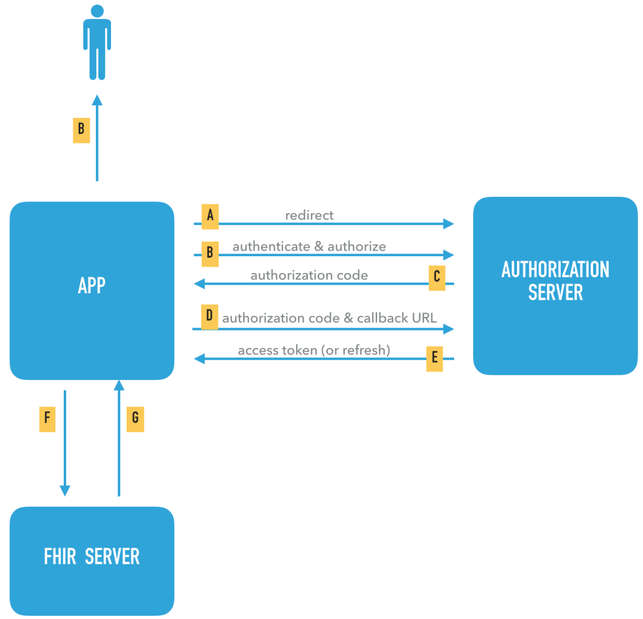
SMART AUTHORIZATION FLOW

- Step D:
  - The app sends the authorization code to the authorization server endpoint URL.
- Step E:
  - The authorization server evaluates the authorization code and sends an access token to the client callback URL.
- Step C:
  - The app requests access to the protected resources from the FHIR server.
- Step F:
  - The FHIR server evaluates the access token and returns the protected resources or performs the requested action.



SMART AUTHORIZATION FLOW

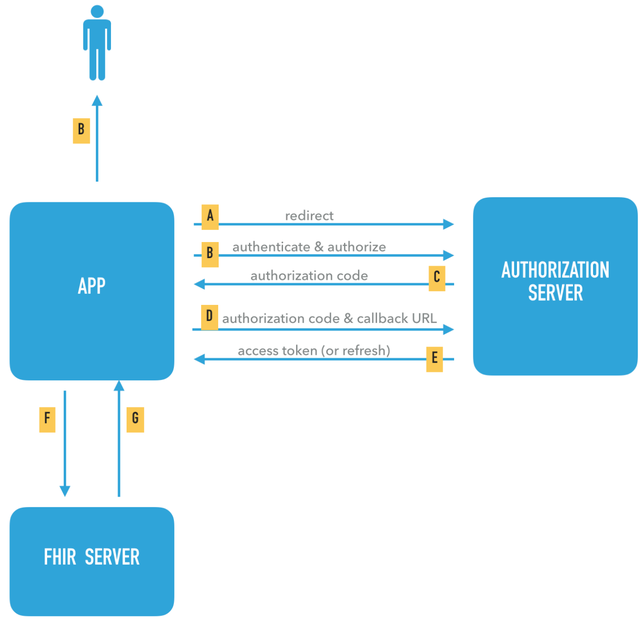
- Refresh
  - If a refresh token is returned along with the access token, the app may use this to request a new access token, with the same scope, once the access token expires.



PROTOCOLS

- SMART on FHIR implements specifications from the OAuth 2.0 RFC family.
- Beyond other:
  - HTTPS based transports and authorization codes from OAuth core.
  - JWT bearer token (RFC 7523).
  - Dynamic Client registration (RFC7591 & RFC7592).
  - MAC signatures (optional).
- SMART on FHIR additionally specifies the attributes used in the request and the JWT bearer token.

- **Note:** SMART on FHIR does not implement one of the standard flows defined in the OAuth 2 core specification.



SURVEY

OAUTH 2.0

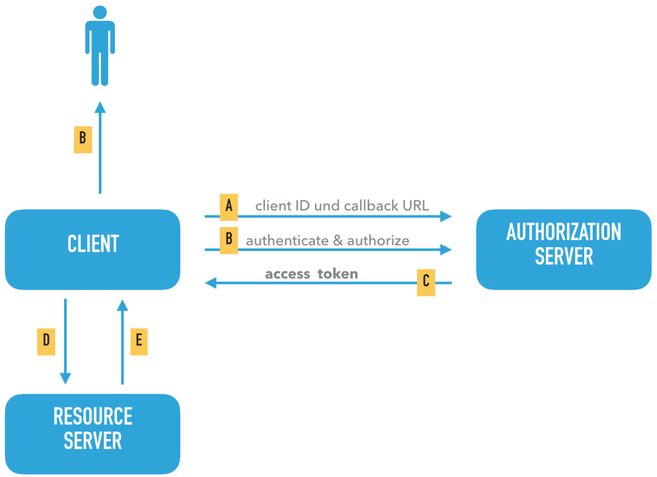
WHAT IS OAUTH?

- A family of specifications published as RFC for web based authorization (see https://tools.ietf.org/wg/oauth/).
- A framework for web based authorization with many options all published as RFC.

Draft name	Rev.	Dated	Status	Comments, Issues
Active:				
draft-ietf-oauth-access-token-jwt	-02	2019-07-24	Active	
draft-ietf-oauth-browser-based-apps	-04	2019-09-22	Active	
draft-ietf-oauth-incremental-authz	-02	2019-05-03	Active	
draft-ietf-oauth-reciprocal	-04	2019-08-01	Active	
draft-ietf-oauth-security-topics	-13	2019-07-08	Active	
Recently Expired:				
draft-ietf-oauth-pop-key-distribution	-07	2019-03-27	Expired	
IESG Processing:				
draft-ietf-oauth-jwsreq	-19	2019-06-11	IESG Evaluation::AD Followup	
draft-ietf-oauth-jwt-bcp	-07	2019-10-13	IESG Evaluation::AD Followup	
draft-ietf-oauth-jwt-introspection-response	-08	2019-09-20	IESG Evaluation::AD Followup	
RFC-Editor's Queue:				
draft-ietf-oauth-mtls	-17	2019-08-23	RFC Ed Queue	
draft-ietf-oauth-resource-indicators	-08	2019-09-11	RFC Ed Queue	
draft-ietf-oauth-token-exchange	-19	2019-07-21	RFC Ed Queue	
Published:				
Draft name	**Rev.**	**Dated**	**Status**	**Obsoleted by/*(Updated by)***
draft-ietf-oauth-amr-values	-08	2017-03-13	RFC 8176	
draft-ietf-oauth-assertions	-18	2014-10-21	RFC 7521	
draft-ietf-oauth-device-flow	-15	2019-03-11	RFC 8628	
draft-ietf-oauth-discovery	-10	2018-03-04	RFC 8414	
draft-ietf-oauth-dyn-reg-management	-15	2015-05-05	RFC 7592	
draft-ietf-oauth-dyn-reg	-30	2015-05-28	RFC 7591	
draft-ietf-oauth-introspection	-11	2015-07-04	RFC 7662	
draft-ietf-oauth-json-web-token	-32 ipr	2014-12-10	RFC 7519	*(RFC 7797)*
draft-ietf-oauth-jwt-bearer	-12	2014-11-12	RFC 7523	
draft-ietf-oauth-native-apps	-12	2017-06-09	RFC 8252	
draft-ietf-oauth-proof-of-possession	-11	2015-12-19	RFC 7800	
draft-ietf-oauth-revocation	-11	2013-07-13	RFC 7009	
draft-ietf-oauth-saml2-bearer	-23	2014-11-12	RFC 7522	
draft-ietf-oauth-spop	-15	2015-07-10	RFC 7636	
draft-ietf-oauth-urn-sub-ns	-06	2012-07-16	RFC 6755	
draft-ietf-oauth-v2-bearer	-23 ipr	2012-08-01	RFC 6750	
draft-ietf-oauth-v2-threatmodel	-08	2012-10-06	RFC 6819	
draft-ietf-oauth-v2	-31 ipr	2012-08-01	RFC 6749	*(RFC 8252)*

ROLES

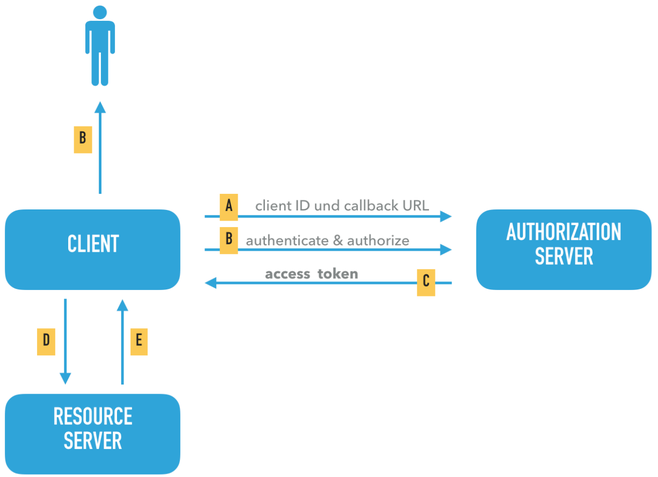
- Resource server (the API):
  - The server hosting user-owned resources that are protected by OAuth (FHIR server).
  - The resource server accepts and validates access tokens and grant the requests on behalf of the resource owner user.
  - The resource server does not necessarily need to know about applications.



ROLES

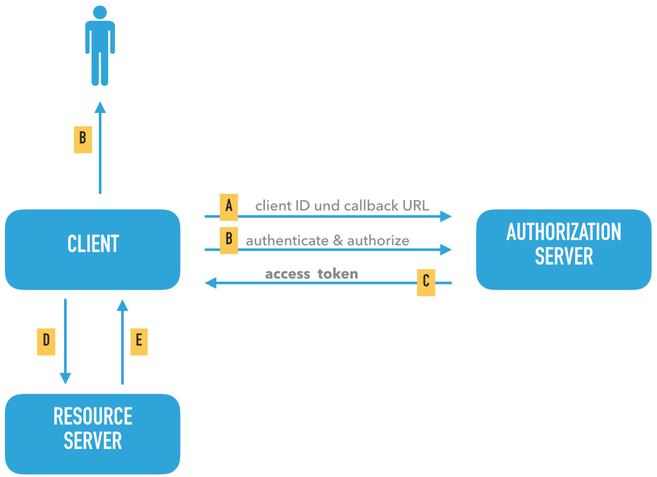
- Resource owner (the user):
  - The OAuth 2.0 spec refers to the user as the “resource owner.”
  - Resource owner grant access to their own data hosted on the resource server.

- Client (the third-party app):
  - An application making API requests to perform actions on protected resources on behalf of the resource owner.
  - The client will obtain permission by either directing the user to the authorization server, or by asserting permission directly with the authorization server without interaction by the user.



ROLES

- Authorization server:
  - The authorization endpoint is used to interact with the resource owner and obtain an authorization grant.
  - The authorization server gets consent from the resource owner and issues access tokens to clients for accessing protected resources.
  - The authorization server may therefore authenticate the resource owner.

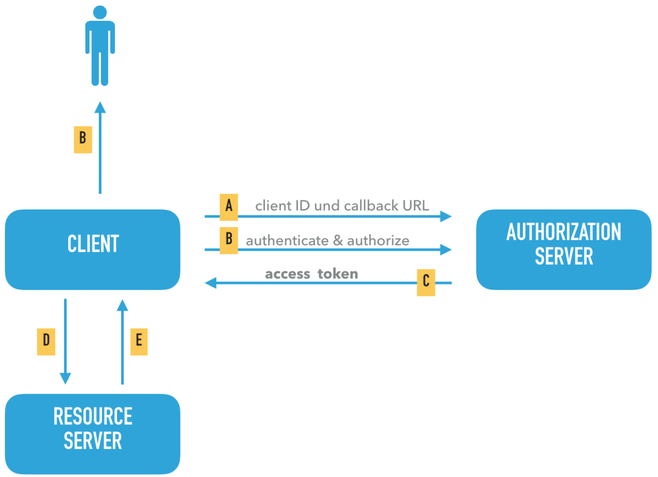


REMARKS

- Clients must be registered at the authorization server with it’s client credentials (client ID and secret), which are used to authenticate the technical system (Node Authentication).
- These credentials are critical in protecting the authenticity of requests when performing operations such as exchanging authorization codes for access tokens and refreshing access tokens.
- Currently this is weakened to support clients which cannot keep a secret like mobile apps or single page apps.

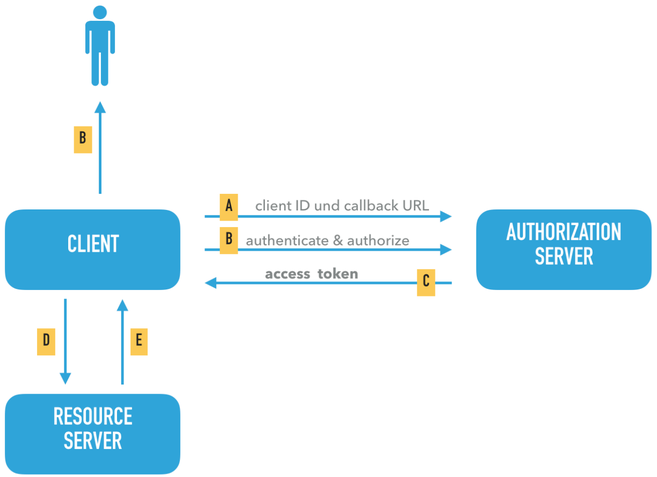
REMARKS

- Access token are credentials used to access the protected resources.
- Access token can have different formats, structures, and methods of utilization (e.g., cryptographic properties) based on the resource server security requirements.
- Access token attributes and the methods used to access protected resources are beyond the scope of the OAuth 2.0 specification and are defined by companion specifications.



REMARKS

- Signatures (Optional in SMART):
  - The MAC Access Authentication specification defines how clients can sign their OAuth 2.0 requests.
  - When connecting to OAuth-enabled APIs that require signatures, each API request must include a MAC signature in the Authorization header of the request.
  - Most OAuth 2.0 authorized APIs require only bearer tokens to make authorized requests.
  - Bearer tokens are a type of access token whereby simple possession of the token values provides access to protected resources.



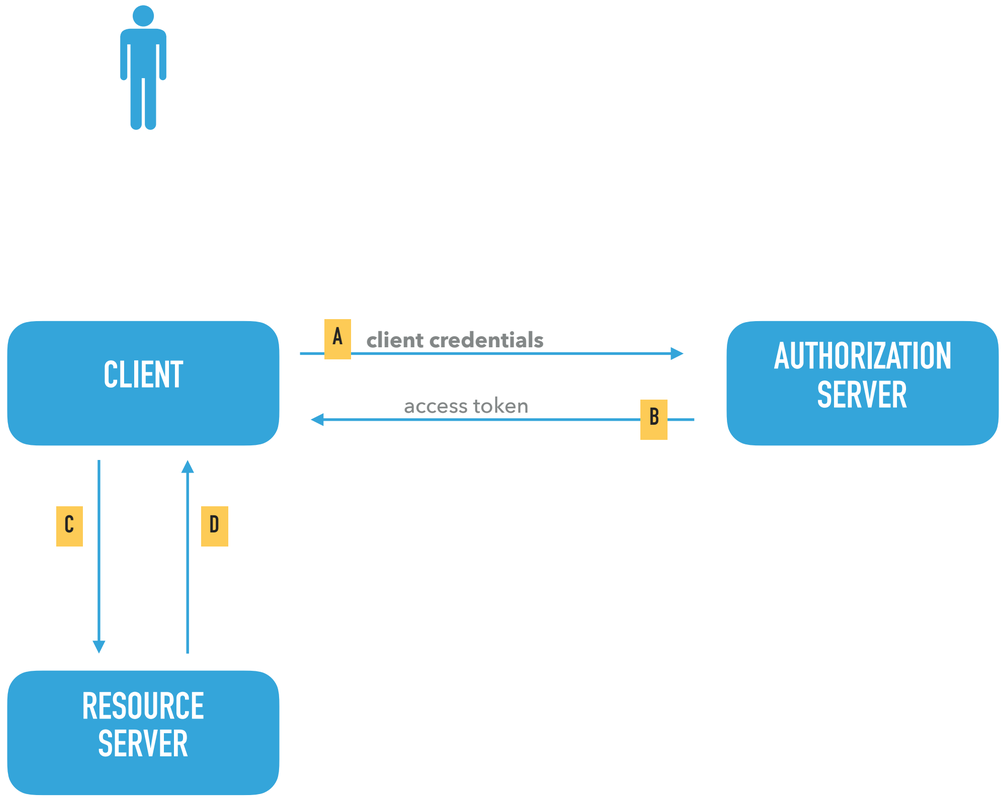
GRANT TYPES

OAUTH 2.0

OAUTH 2.0 GRANT TYPES

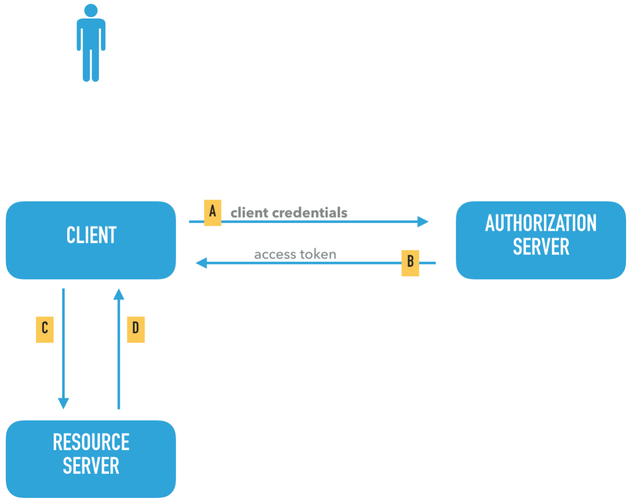
- Core Grant Types
  - Client Credentials
  - Authorization Code
  - Device Code
  - Refresh Token
- Extensions
  - Assertion Grant Type
- Legacy
  - Implicit Grant
  - Password Grant

CLIENT CREDENTIAL FLOW

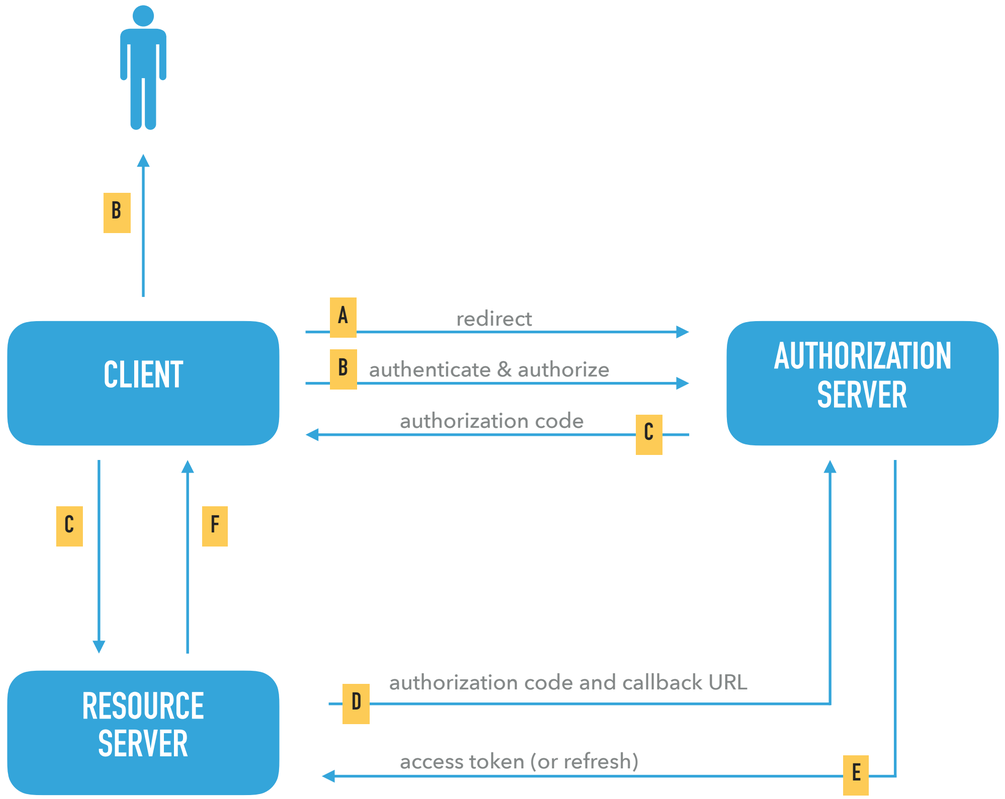


CLIENT CREDENTIAL FLOW

- Step A:
  - Client authenticates with client Id, secret and client credentials (or other forms of client authentication).
- Step B:
  - Authorisation server returns the access (or refresh token).
- Step C:
  - Client sends request to resource server with access token in the authorization header.
- Step D:
  - Resource server evaluates the access token and returns the protected resources.

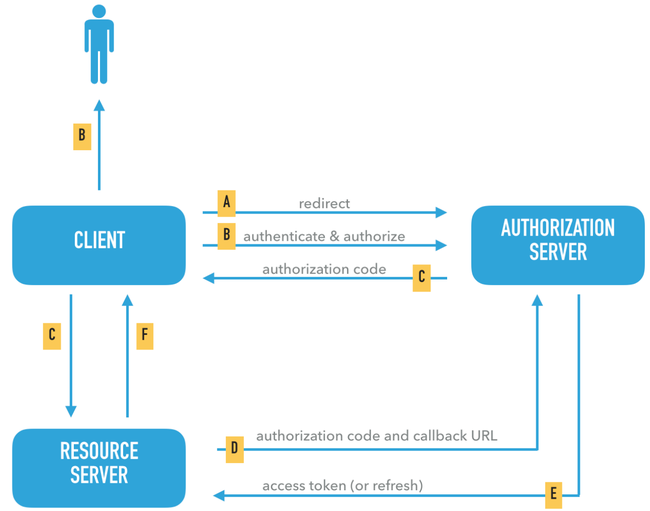


AUTHORIZATION CODE FLOW

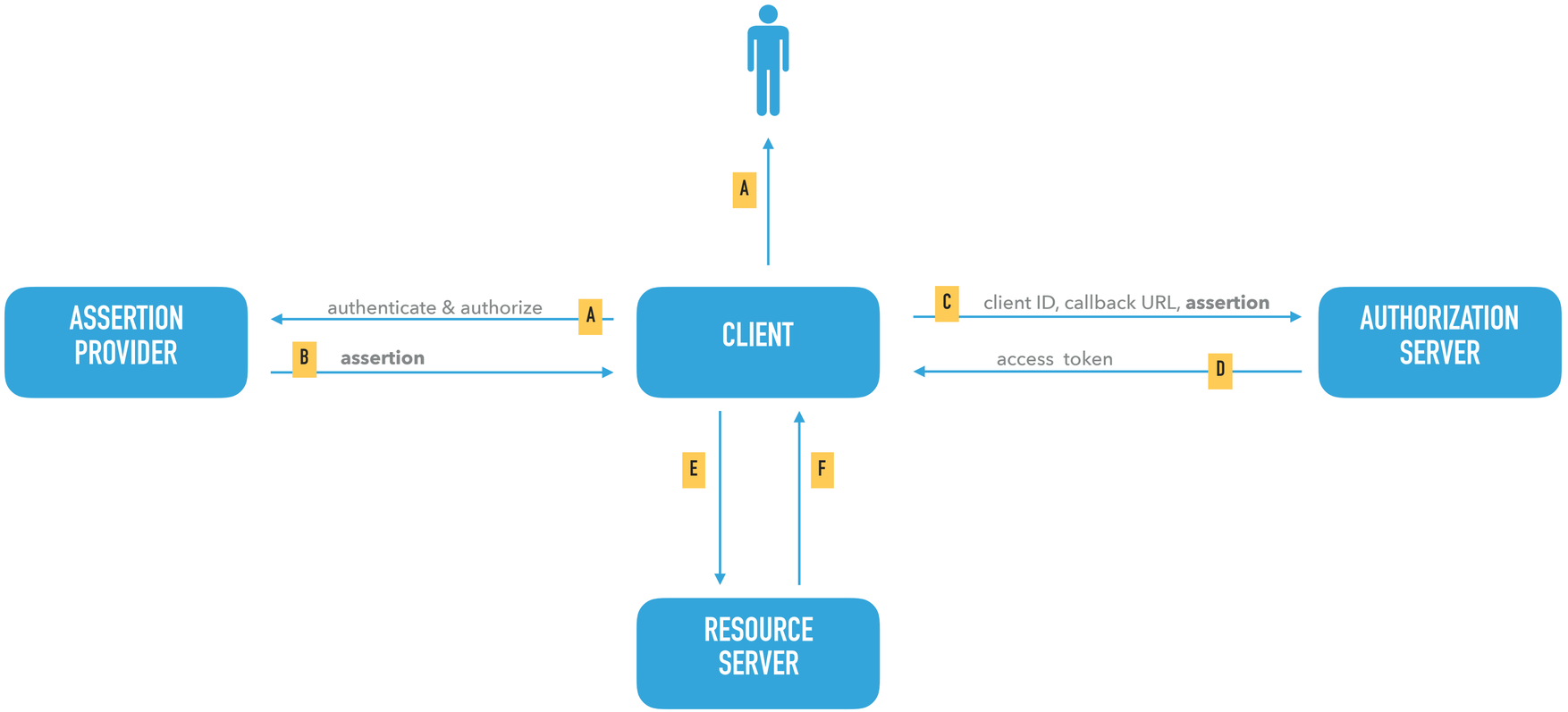


AUTHORIZATION CODE FLOW

- Step A:
  - Redirect the user to the authorisation server with client ID and secret.
- Step B:
  - Authorization server authenticates the resource owner.
  - Resource owner authorises client access to the required resources.
- Step C:
  - Authorisation server redirects to the client with the authorisation code.
  - Client requests protected resources from the client sending the in authorization code.
- Step D and E:
  - Resource server resolves the authorisation code to the access token at the authorisation server.
- Step F:
  - Resource server evaluates the access token and returns the protected resources.

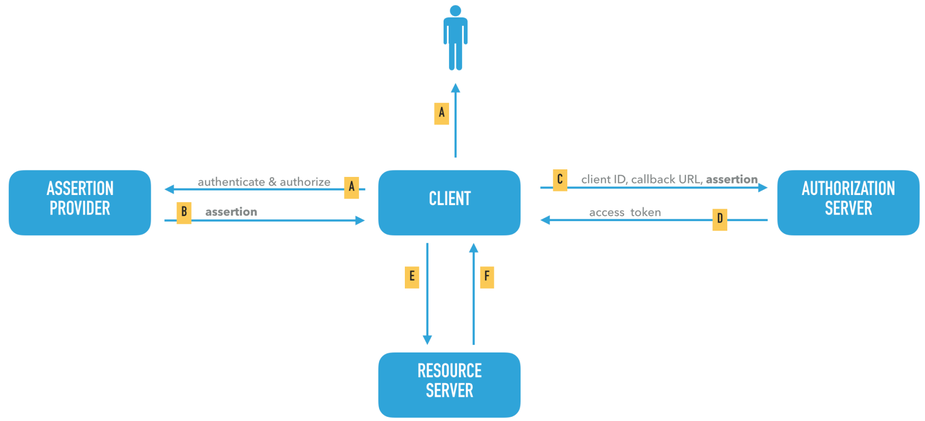


ASSERTION GRANT FLOW

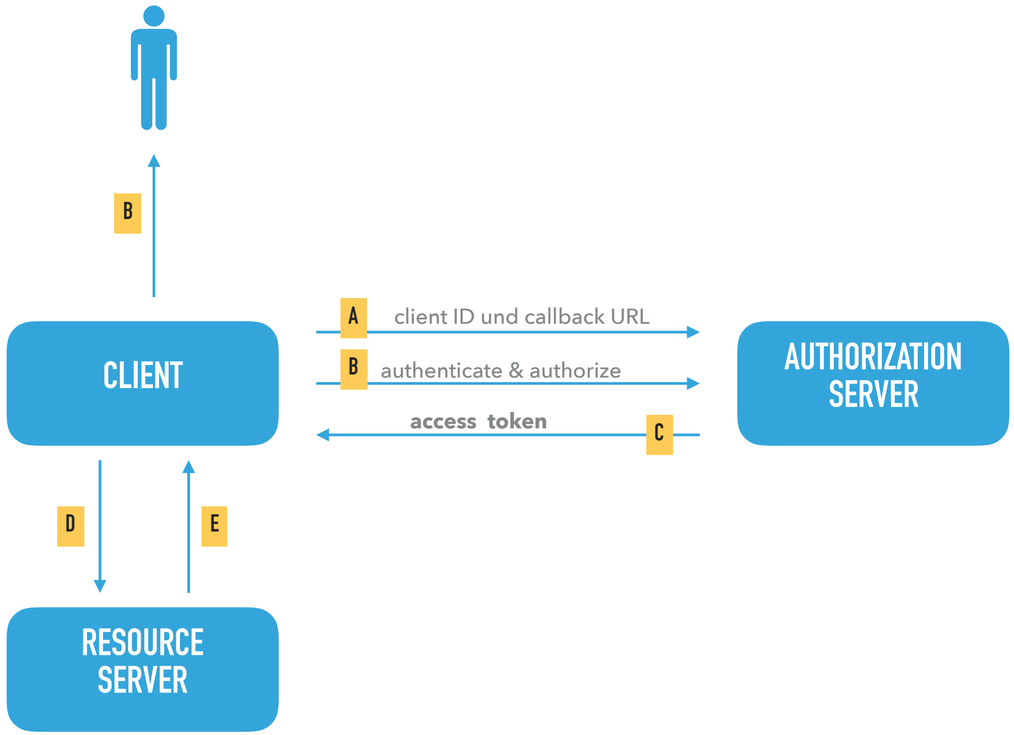


ASSERTION GRANT FLOW

- Step A:
  - Redirect to assertion provider server.
- Step B:
  - User authenticates at the assertion provider.
  - Assertion provider responds the assertion.
- Step C:
  - Redirect to authorization server with assertion.
- Step D
  - Authorization server responds the access token.
- Step E:
  - Client sends request to resource server with access token in authorization header.
- Step F:
  - Resource server evaluates the access token and returns the protected resources.

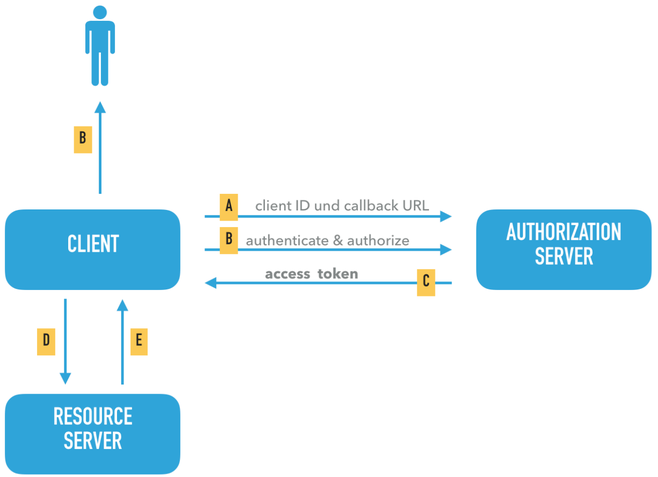


IMPLICIT GRANT FLOW

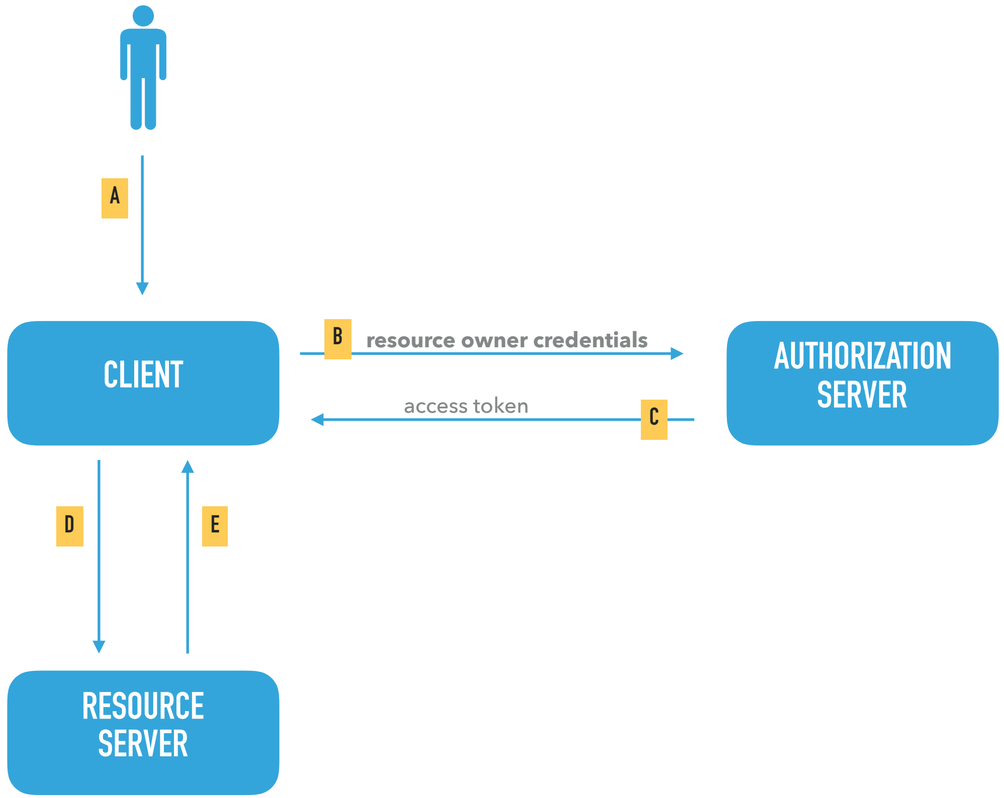


IMPLICIT GRANT FLOW

- Step A:
  - Redirect to authorisation server with client ID.
- Step B:
  - authorisation server authenticates the resource owner.
  - Resource owner authorises client access to the required resources.
- Step C:
  - Redirect to client with access token to resource server.
- Step D:
  - Client sends request to resource server with access token.
- Step E:
  - Resource server evaluates the access token and returns the protected resources.

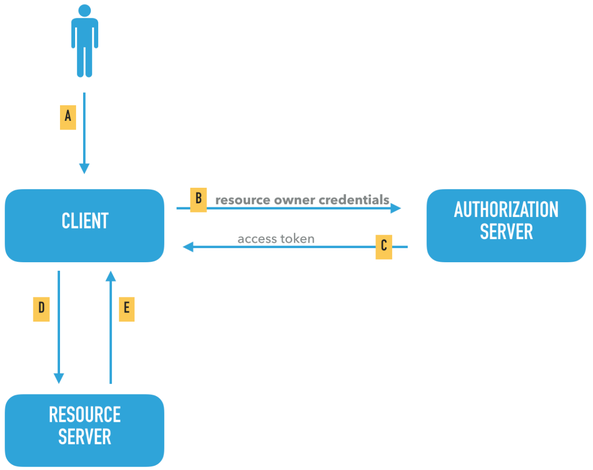


RESOURCE OWNER PASSWORD FLOW



RESOURCE OWNER PASSWORD FLOW

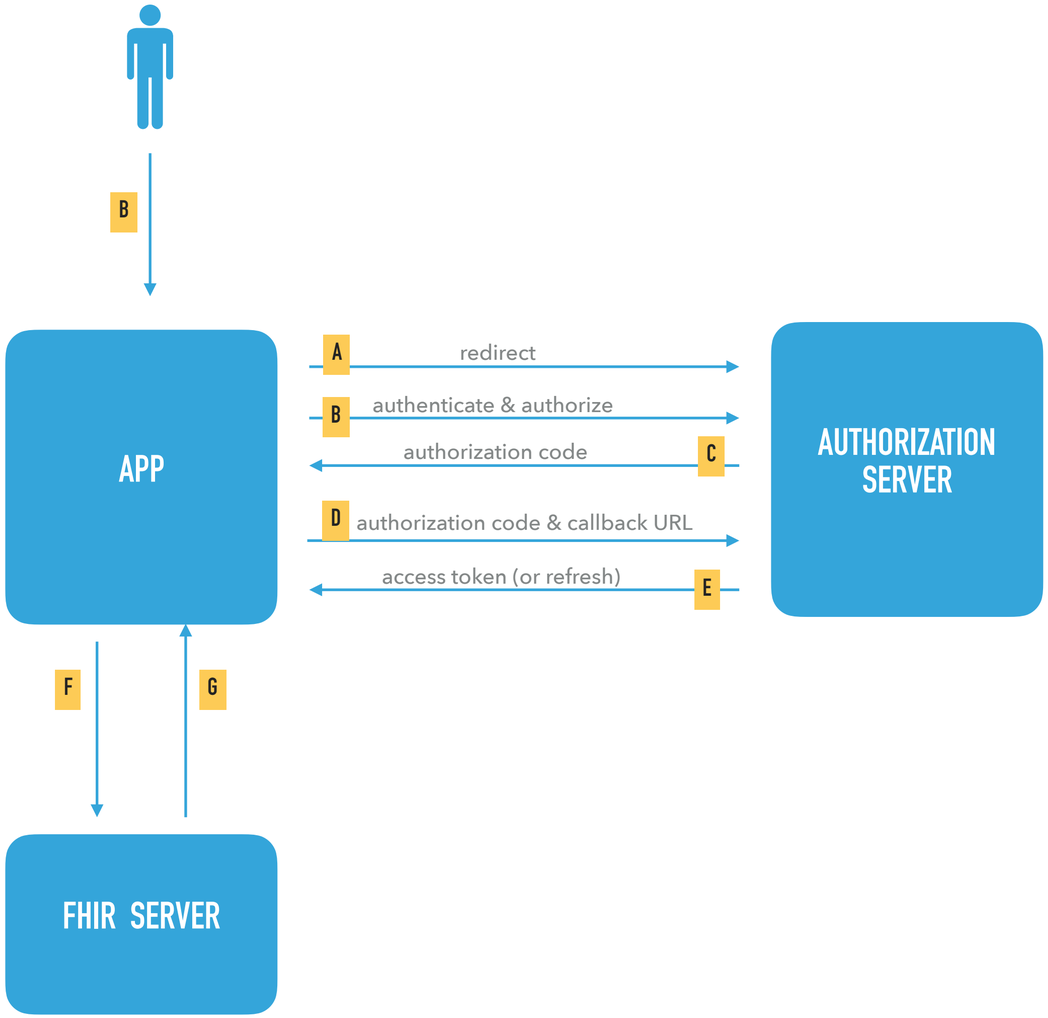
- Step A:
  - Resource owner present the client the password credentials (of the resource owner).
- Step B:
  - Clients sends resource owner password credentials to authorisation server.
- Step C:
  - Authorisation server returns the access (or refresh token).
- Step D:
  - Client sends request to resource server with access token in the authorization header.
- Step E:
  - Resource server evaluates the access token and returns the protected resources.



PROTOCOLS

SMART ON FHIR

SMART AUTHORIZATION FLOW

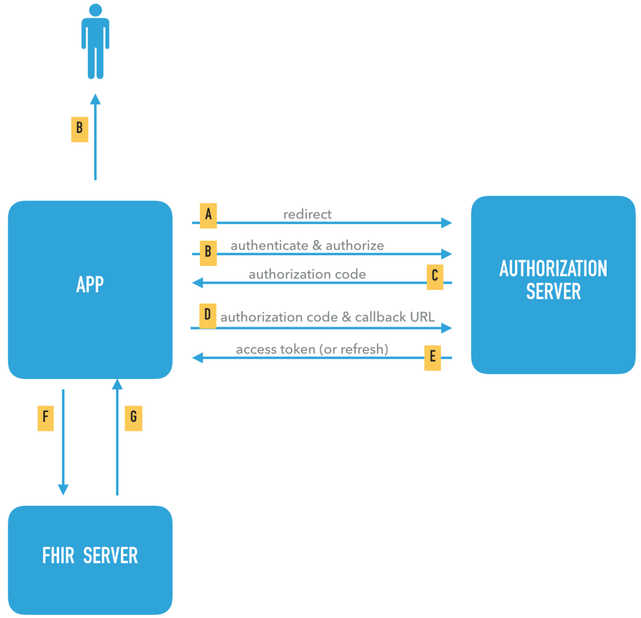


PROTOCOLS

- Step A

The client redirects to the authorization server with the required parameter:

```
GET authorize?response_type=code&
client_id=app-client-id&
http%3A%2F%2Flocalhost%3A9000%2Fcallback&
launch=xyz123&
scope=launch+patient%2FObservation.read+
         patient%2FPatient.read+openid+fhirUser&
state=98wrghuwuogerg97&
aud=https://ehr/fhir

HTTP/1.1 Host:localhost:9001

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:39.0)
Gecko/20100101 Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://localhost:9000/
Connection: keep-alive
```

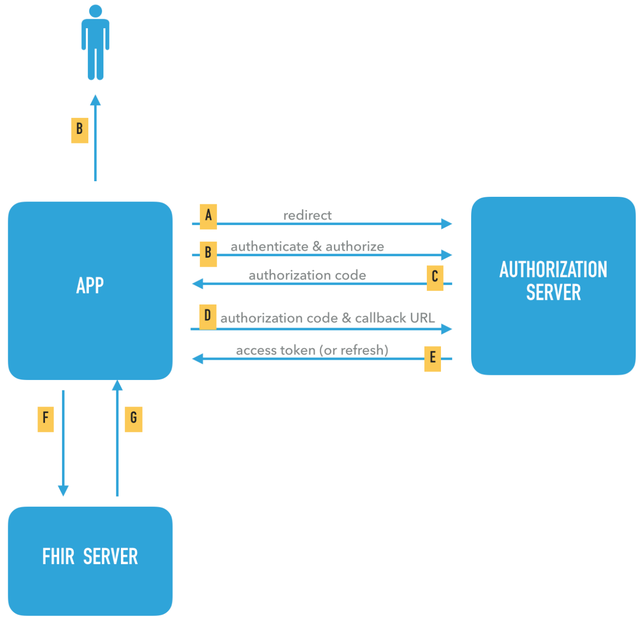


PROTOCOLS

- Step C

The authorization server performs a HTTP GET on the callback URL with the authorization code:

```
GET /callback?code=8V1pr0rJ&state=98wrghuwuogerg97

HTTP/1.1 Host: localhost:9000

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:39.0)
Gecko/20100101 Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://localhost:9001/authorize?response_type=code&
client_id=app-client-id&http%3A%2F%2Flocalhost%3A9000%2Fcallback&
launch=xyz123&scope=launch+patient%2FObservation.read+patient%2FPatient.read+openid+fhirUser&
state=98wrghuwuogerg97&aud=https://ehr/fhir
```

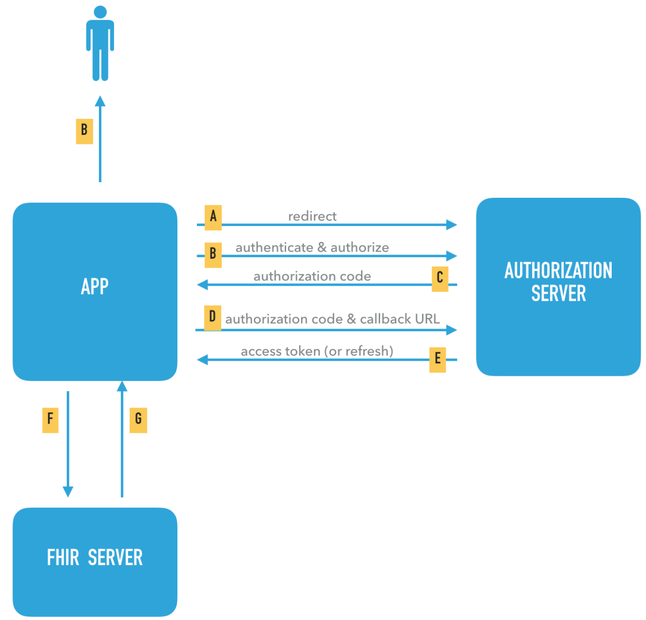


PROTOCOLS

- Step D:

The client performs an HTTP POST with parameter
as a form-encoded HTTP entity body, passing its
client_id and **client_secret** as an HTTP Basic authorization header.

```
POST /token
Host: localhost:9001
Accept: application/json
Content-type: application/x-www-form-encoded
Authorization: Basic bXktYXBwOm15LWFwcC1zZWNyZXQtMTIz

grant_type=authorization_code& redirect_uri=http%3A%2F%2Flocalhost%3A9000%2Fcallback&
code=98wrghuwuogerg97
```

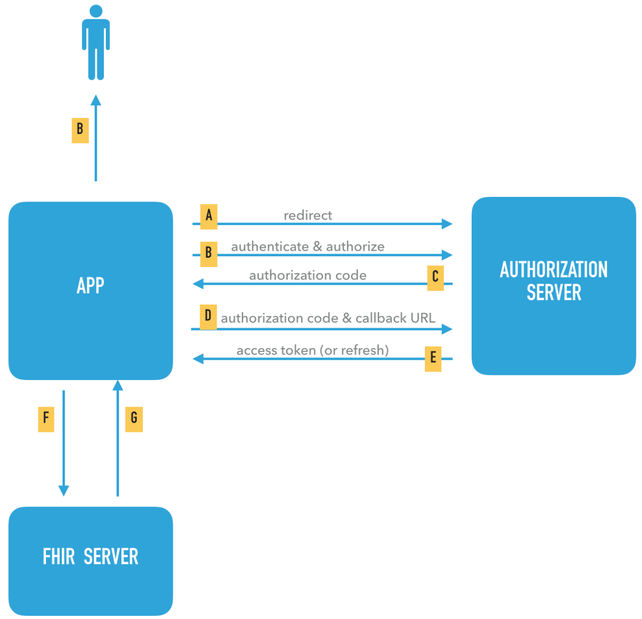


PROTOCOLS

- Step E

The authorization server response carries the access token:

```
HTTP 200 OK
Date: Fri, 31 Jul 2015 21:19:03 GMT
Content-type: application/json
{

"access_token": "i8hweunweunweofiwweoijewiwe",
"token_type": "bearer",
"expires_in": 3600,
"scope": "patient/Observation.read patient/Patient.read",
"intent": "client-ui-name",
"patient":  "123",
"encounter": "456"

}
```

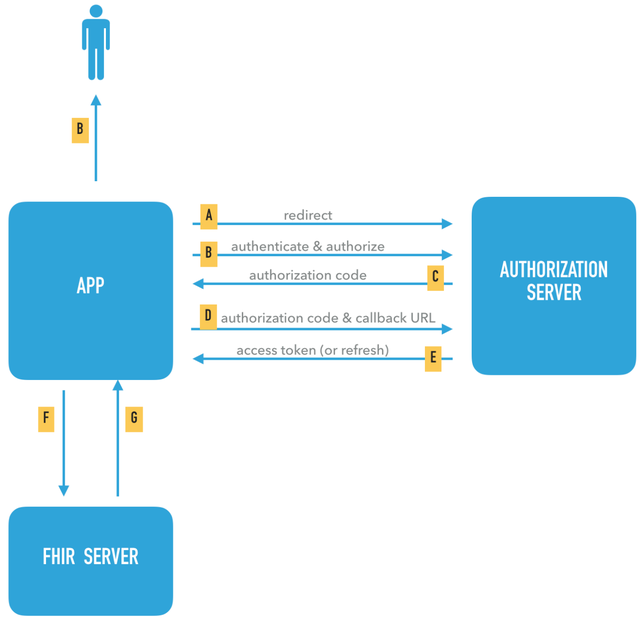


PROTOCOLS

- Step F

The client performs a HTTP request on the FHIR server URL with the access token in the authorization header:

```
GET /resource HTTP/1.1
Host: localhost:9002
Accept: application/json
Connection: keep-alive
Authorization: Bearer 987tghjkiu6trfghjuytrghj…
```



SUMMARY

- SMART on FHIR harmonises the way to authenticate and authorise apps using FHIR resources.
- There is a growing community providing free of charge apps that use SMART on FHIR and are this easy to integrate.
- The community also provides libraries for common used programming languages (Java, Ruby, ECMA Script, C#, …) which simplify implementing your own SMART on FHIR app.
- And even if you don’t want to use the libraries you can implement it on your own using specifications from the OAuth 2.0 family.

FINAL REMARKS

- With the SMART on FHIR authorization flow we trust the client app but not the runtime environment (mobile device, Web Browser, …).
- This trust model might be problematic in a health care environment and needs further investigation.
- SMART on FHIR currently does not support alternative flows like the Assertion Grant Flow. This might block the applicability in a more sensitive and restrictive health care environment like the Swiss EPR.

REFERENCES

- SMART App Gallery - https://apps.smarthealthit.org
- SMART launch framework - http://www.hl7.org/fhir/smart-app-launch/
- SMART launch scopes and context - http://www.hl7.org/fhir/smart-app-launch/scopes-and-launch-context/index.html
- SMART on FHIR - https://docs.smarthealthit.org
- OAuth 2.0 - https://oauth.net/2/
- OAuth RFC overview - https://tools.ietf.org/wg/oauth/
- Getting started with OAuth 2.0, O'Reilly media (2012)
- OAuth 2 in action, Manning Publications (2017)

SPEAKER

Martin Smock

Produktmanager Post CH

Entwicklung und Innovation

Vorstand IHE Suisse

Vendor Co-Chair IHE Europa

IHE SUISSE

Integrating the Healthcare Enterprise

QUESTIONS?