POWER2DM

"Predictive model-based decision support for diabetes patient empowerment"

Research and Innovation Project
PHC 28 – 2015: Self-management of health and disease and decision support systems based on predictive computer modelling used by the patient him or herself

POWER2DM D4.10 (or D4.6.1b)

Privacy/Security Enablers for POWER2DM Services II

Due Date:	31.10.2017
Actual Submission Date:	31.10.2017
Project Dates:	Project Start Date: February 01, 2016
	Project End Date: July 31, 2019
	Project Duration: 42 months
Deliverable Leader:	SRDC

Project co-funded by the European Commission within H2020 Programme (20015-2016)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document History:

Version	Date	Changes	From	Review
0.1	06.10.2017	Initial Version (Section 3)	SRDC	
0.2	13.10.2017	Section 4 Demonstration of functionalities	SRDC	
0.3	23.10.2017	Update the summary section (Section 2)	SRDC	
1.0	30.10.2017	Latest compiled version	SRDC	ALL

Contributors (Benef.) Tuncay Namlı (SRDC)
Ozan Köse(SRDC)

Responsible Author Tuncay Namlı **Email** tuncay@srdc.com.tr

POWER2DM Consortium Partners

Abbv	Participant Organization Name	Country
TNO	Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek	Netherlands
IDK	Institute of Diabetes “Gerhardt Katsch” Karlsburg	Germany
SRDC	SRDC Yazilim Arastirma ve Gelistirme ve Danismanlik Ticaret Limited Sirketi	Turkey
LUMC	Leiden University Medical Center	Netherlands
SAS	SAS Servicio Andaluz de Salud	Spain
SRFG	Salzburg Research Forschungs Gesellschaft	Austria
PD	PrimeData	Netherlands
iHealth	iHealth EU	France

Table of Contents

1 Introduction

1.1 Purpose & Scope

The purpose of deliverable D4.10 is to provide final privacy and security mechanisms applied in POWER2DM system as well as to demonstrate them. This covers the authentication, authorization, auditing and other security mechanisms to protect the **security and privacy of patients' medical and identity data**. POWER2DM will consist of several services that exchange patient's data among them and visualize them to the users (practitioners and patient) in several phases. In order to ensure strict privacy and security requirements, they should implement certain processes and cryptographic protocol based on the **specified security and privacy policies**. This deliverable will enable these common mechanisms (enablers) and define how the software specifications and standards are implemented.

D4.9 provides an initial overview of the architecture and specifications/standards used to provide the security and privacy mechanisms. This deliverable, D4.10 focuses on implementation details and demonstration of functionalities. On the other hand, within the project, some of the requirements changed due to requests from piloting partners and therefore we have updated some mechanisms (e.g. Section 2.2 User identification). Therefore, in Section 2, we provide a new summary of the privacy and security mechanisms to be applied in POWER2DM (Updated version of same section in D4.9). Section **Error! Reference source not found.** provides further details of the implementation of sub components and Section 4 provides snapshots and demonstration of the functionality provided by the Core Services component.

1.2 References

- D4.9 (D4.6.1a) Privacy Security Enablers I
- D1.3 Conceptual Design

2 Summary of Privacy and Security Mechanisms

2.1 Summary of Information Collection and Use

POWER2DM SMSS collects a range of information about patients to evaluate the medical, contextual and psychological situation of the patient to provide the necessary self-management support for patient and support the care management process in shared-decision making encounters. Although, the details of the information to be collected in POWER2DM Care Program will be provided in D5.3 "POWER2DM Evaluation Campaign Protocol ", anyone can consult the D4.1 "Personal Data Model and Service API" to understand the general coverage. In this section, the following list shows the categorization of information collection in terms of security and privacy perspective;

- **User Identity Data**: After the discussions with piloting partners, the decision was to limit the identity data to be collected from patient as far as possible. The only identity data elements are as follows;
    - **email:** This is required due to management of patient account (e.g. password lost, etc) and used to be a username for patient

- **Personal Lifestyle and Health Data**: This is the set of personal health records collected via medical devices, collected from patient via POWER2DM SMSS Applications or 3rd party lifestyle applications, and data entries of care providers during shared-decision making. This includes daily observations, clinical test results, problems, barriers, goals, etc. This set also includes the following information; which can be considered as identity data but not provide much information about the identity of patient within the POWER2DM patient population.
    - **age:** Age is required in POWER2DM risk score calculation algorithms and also for POWER2DM care and collected as a birth year instead of whole birth date to preserve privacy.
    - **gender:** Gender is required in POWER2DM risk score calculation algorithms and also within POWER2DM care program
    - **ethnicity:** Ethnicity is required for POWERDM's risk score calculation algorithms

  All these data will be protected separately from patient's main identity data (email and the study number) and maintained in pseudynomized way.
- **Application Usage Behaviour and Contextual Data:** This is the set of analytic information related with how patient use the POWER2DM SMSS applications and how they react to interventions, as well as the contextual data like location (home, office, etc.), interruptibility, activity (walking, transportation, etc.) that can be derived from the on-board mobile phone sensors.
- **Patient Consent, Privacy Policies and Auditing Data:** This is the set of privacy policies and consent authored by patient and the audit logs describing all the operations done in POWER2DM (e.g. accessing to data, login to application, updating data, etc)

In terms of **information sharing and disclosure**, POWER2DM have the following general policy for these categories;

- For **"User Account/Identity Data"**, the default POWER2DM policy is to disclose the information to only the owner (patient). The information is only used for authentication and account management mechanisms.
- For **"Personal Lifestyle and Health Data"**, the information is disclosed only to authorized users according to the specified privacy policy in POWER2DM. Mostly the information is used by the POWER2DM SMSS to display the content back to the patient himself or perform some analytics/algorithms to make some deductions and make actions accordingly (interventions; reminders, motivations, etc.). On the other side, information is also used to support the shared-decision making process for diabetes care management. Therefore, to assure the realization of POWER2DM care process needs at least some permission for the corresponding care providers. Conflicting rules that will prevent the realization of POWER2DM care process will not be allowed.
- For **"Application Usage Behaviour and Contextual Data",** the information is only used by POWER2DM SMSS internally and will not be disclosed to any user other than patient himself/herself. The SMSS use the information in its internal algorithms and decision flows.

2.2 Identification of Users

Identification of users (patients and care providers) and systems involved in the POWER2DM processes is important in terms of privacy and security.

For patient identification, POWER2DM will manage the following identifiers:

- **Study Number:** This is the identifier assigned to the patient by the practitioner while registering patient to the system. Similarly, the identifier will be used by practitioners to identify and confirm patient is the right patient. This is bound to the user account data (email) in Core Service. Study number will not be disclosed to any client, and will only be used during patient selection procedure where practitioner select the patient among all his/her patients.

- **Pseudonymized data store identifier:** This is the random unique identifier assigned to patient by Personal Data Store (PDS). PDS share this value with Core Service during patient registration and Core Service maps this value to patient's identity data (email, study number) internally. Core Service embed this value (encrypted) into access tokens in authorization process and when the client uses the access token to access the resources in PDS, PDS use the value to match to the correct patient's records.

For care providers, there will be random unique identifier assigned by PDS and this will be bound to care provider's identity data in Core Services.

2.3 Authentication

POWER2DM deals with both user authentication and client authentication.

For user authentication, **OpenID Connect 1.0**[1] protocol will be implemented where the "Core Services" component will provide the protocol's endpoints (Authorization Endpoint, etc. see Section **Error! Reference source not found.** for details) and web application (**single sign-on page**) to authenticate the user. POWER2DM front end components will use the endpoints provided by the Core Service to be sure that the user is authenticated. User authentication is done by username and password in default, however, users can select 2-factor authentication via SMS to their mobile phone in addition to username/password.

For client (system) authentication, only Confidential Clients are authenticated (as it is theoretically impossible to authenticate Public Clients) and **SSL client authentication with x509 certificates** will be used. For each component, an X509 certificate will be generated which is signed by a root POWER2DM certificate.

2.4 Authorization and Access Control

In POWER2DM, there will be two types of data access;

i. a client system accessing on behalf of a user to a specific patient's data
    a. (Public Client)
        i. e.g. POWER2DM Shared-decision Making Web Application accessing directly to PDS for patient's personal records on behalf of a care provider to visualize the data to care-provider
        ii. e.g. POWER2DM SMSS Mobile App accessing PDS on behalf of patient himself/herself
    b. (Confidential Client) e.g. POWER2DM Action Plan Engine accessing the records of patient on behalf of him/her to analyse them and provide feedback to patient for his performance in the weekly review of action plan

ii. a confidential client accessing data for internal analysis and algorithm execution
    a. POWER2DM Communication Engine accessing the data of each patient (anonymously) to perform daily analytics to plan for the interventions

For the first alternative (data access on behalf of a user), a **delegated access control mechanism** will be implemented in POWER2DM. Access control mechanism will be based on **"Role Based Access Control"** where the policy can allow or deny access of a specific role **in granularity of record types** (e.g. Blood Glucose Measurements, Goals, Personal Values, etc.). The role and record type hierarchies will be defined in line with the record types stored in PDS and the implementations will be configurable in terms of new role or record type definitions. For POWER2DM pilots, privacy policy will be as follows (more details will be given in Section 3);

[1] http://openid.net/specs/openid-connect-core-1_0.html

- Users who has assigned to "care manager" or "care supporter" role (during patient registration) for the patient can access the Personal Lifestyle and Health Data of the patient
- Patient himself having assigned to "self care manager" can access all his own data.
- Patient may assign "self care supporter" role to anyone (friends or relatives) and these users can see the patient Personal Lifestyle and Health Data

Core Services component will provide an OAuth 2.0[2] complaint Authorization Service to manage the authorization requests and map the decisions from PDP to permissions (OAuth scopes) bound to the issued access token (JWT token) to the client. The PDS will verify the access token and decide on the authorization decision based on the bound permissions (scopes).

For the second alternative, specified Confidential Clients (POWER2DM Backend components) will be authorized to access all the data as they can reach only to pseudonymised data and cannot identify whose records are they.

2.5 Communication Security

Communication among POWER2DM backend components (which can be deployed on different machines) will be protected by TLS v1.2[3] protocol with mutual authentication. The implementations will conform to the IHE Audit Trail and Node Authentication (ATNA)[4] specification.

Communication between a POWER2DM frontend component and POWER2DM backend component will be protected by TLS v1.2 protocol.

2.6 Auditing

Auditing is an important concept for non-repudiation and being transparent to patients who are accessing his/her personal health data. For the auditing mechanism, FHIR compliant auditing mechanism will be built in which all audits are stored in a secure audit repository by the FHIR Audit Event format by FHIR services. Personal Data Store and Core Services will log all the access requests and data disclosures in this repository.

In addition, a web interface will be provided for patient to show the list of accesses to his/her personal health records.

3 Summary of Implementation

POWER2DM security and privacy management architecture consists of three main sub-components; **Onauth Server** which represents the Core services and **Onauth Manager** which represents the Security and privacy management UI and **Audit Repository**.

- **Onauth Server** provides services for user registration, privacy policy management and endpoints defined in **OpenID Connect Core 1.0**[5] standard to perform authentication and authorization (Authorization Endpoint, Token Endpoint, etc.).

- **Onauth Manager** is a web application for representing the functionalities of **Onauth Server** with following user screens; single sign on UI, consent/approval UI, policy management UI, client registration UI, user registration UI and audit viewer UI.

[2] https://tools.ietf.org/html/rfc6749
[3] https://www.ietf.org/rfc/rfc5246.txt
[4] http://wiki.ihe.net/index.php/Audit_Trail_and_Node_Authentication
[5] http://openid.net/specs/openid-connect-core-1_0.html

- **Audit Repository** is a FHIR repository that has the ability to only manage FHIR AuditEvent resources which are send by other POWER2DM components.

3.1 Onauth Server

3.1.1 Configuring Onauth Server for POWER2DM

Onauth Server could be initiated with default policy definitions and set of credentials. In order to configure **Onauth Server** for POWER2DM, a default privacy policy and set of credentials have been defined. Rest of this section focuses on the details of these configurations and definitions.

3.1.1.1 Realm

Each realm represents a different domain in **Onauth Server** environment, every realm has their own set of rules and users. Realm definition consists of a name for representational purposes and a unique realm ID for identifying the realm and its components. Users in **Onauth Server** can only be a member of single realm and they are treated with respect to the rules and concepts defined by their realm. POWER2DM is also a realm in **Onauth Server** environment, following is the realm definition of POWER2DM realm;

```
{
  "id": "power2dm",
  "name": "Power2dm Realm"
}
```

3.1.1.2 Groups

Groups in **Onauth Server** represents the rules and concepts defined for specific subset of users in the realm. Groups are defined under a realm and groups may define modified rules and concepts by extending the base definitions defined by their realm. Creating groups and assigning a to user to a group is decided by the administration; being a member of a group is not mandatory in **Onauth Server** environment. Users that belong to a group inherits the rules and concepts defined by their group. Group definitions consists of a name for representational purposes, a unique group ID for identifying the group and the ID of the realm that group belongs to.

In POWER2DM, for testing purposes, a group is defined for each partner to prevent unwanted modifications on their test users. Following is the group definition of a LUMC group;

```
{
   "id": "lumc",
   "name": "Leiden University Medical Center",
   "realmId": "power2dm"
}
```

3.1.1.3 Roles

Authorization of the users in **Onauth Server** environment are done based on the roles of the users. Each role defines access to some set of permissions and having that role authorizes user for permission that role have. Role definitions contain following fields;

id	Unique identifier of the role
name	Human readable name of the role
realm_id	ID of the role that this role belongs to
isFunctional	Flag that indicated if rules is functional or structural

Following is the definition of the role in POWER2DM;

```
{
    "id": "patient",
    "name": "Patient",
    "realm_id": "power2dm",
    "isFunctional": false
}
```

POWER2DM realm defines two types of role; structural and functional roles. Structural roles identify type and category of user with respect to the POWER2DM realm (e.g. realm administrator, physician). Users could be assigned to more than one structural role and being assigned to multiple roles authorizes them for accessing all the permissions that set of roles define. Besides administrative roles (e.g. realm_admin, group_admin) which are defined for all realms, POWER2DM realm defines three more structural roles;

practitioner	A Practitioner providing care with support of POWER2DM
nurse	A Nurse supporting care in POWER2DM
patient	A Patient in POWER2DM

Functional roles identify type and category of user with respect to the specific patient. Every patient in POWER2DM realm has a care team that maps functional roles to the users. So, users' permissions may vary from patient to patient with respect to the functional role of the user. For POWER2DM realm, four functional roles are defined;

care-manager	Assigned to Practitioners that manage the care of the patient in POWER2DM
care-supporter	Assigned to Nurses that supports the care of the patient in POWER2DM
self-care-manager	Assigned to patient himself as the manager of the self-management. Patient may assign this role to a relative to give a full access to the self-management system.
self-care-supporter	Patient may assign this role to relatives or friends to give him a partial access to the self-management system.

3.1.1.4 Rules

Rules define the access of a role on specific protected resource. There is a rule definition in **Onauth Server** for each relation between a role and a resource set. Rule definitions contain following fields;

resourceSetId	Indicates which resource is affected by this rule.
roleId	Indicates which role can access this rule.
policyId	ID of the policy that this rule belongs to.
permissions	Only required for smart scopes values (see 2.1.2 for scope values). It defines which permissions are given for resource set (read or write)

Following is the rule definition that identifies the relation between care-manager and Observation resource;

```
{
    "resourceSetId": "Observation",
    "roleId": "care-manager",
    “policy_id”: “power2dm_privacy_policy”
    "permissions": {
        "read": 1,
        "write": 1,
        "isSmartScope": true
    }
}
```

3.1.1.5 Privacy Policy Definition

Privacy Policy definition consists of set of rules and options that completely identifies the authorization profile of realm. Every realm within **Onauth Server** has a base privacy policy that defines the mapping between roles and resources. In addition, groups and patients within a realm can create their policies by extending the base privacy policy of their realm. If a group defines its own policy, that policy is considered as the base privacy policy for its users. Privacy policy definition contains following fields;

id	Unique identifier of the policy.
name	Human readable name of the policy.
description	Simple description that explains the purpose of policy.
authorId	Identifier of the user who defined the policy.
realm_id	Identifier of the realm that policy belongs to.
group_id	Identifier of the group (If the policy defined for the group) that policy belongs to.
patient_access	Option that are used to decide which set of patients a practitioner can access. Options are; • **realm**: Practitioners can access every patient in their realm • **group**: Practitioners can access every patient in their group • **care-team:** Patients are only accessed by the practitioners within their care-team.
isBase	Flag that indicates if the policy is a base policy
isActive	Flag that indicates if the policy is the active policy of the author
rules	Rules that defines the relation between roles and resources.

See Appendix A for the complete base privacy definition of POWER2DM.

3.1.2 Scope Values

Onauth Server binds scope values to access tokens to decide which token can access which protected resources. Scopes values are generated with respect to the rule definitions that are defined for user's role. For **Onauth Manager** and PDS the following scopes are defined;

openid	Informs the Authorization Server that the Client is making an **OpenID Connect**[6] Authentication request.
profile	This scope value requests access to the End-User's default profile information (claims), which are: given_name, family_name, username, picture, gender, birthdate,zoneinfo, locale, pds_resource_type, pdm_user_type, pdm_care_givers, pdm_organization (See Section 2.1.3 for the definition of user claims).
Email	This scope value requests access to the email and email_verified claims
address	This scope value requests access to the address claim.

[6] http://openid.net/specs/openid-connect-core-1_0.html

patient/resourceType.permission	This scope value requests access to the patient's protected resources on PDS. The resource type could be any FHIR resource type or (*) (which means all resource types the user authorized for) and the permission could be read, write or (*) (which means both read and write access). E.g.: patient/Observation.read means read all observations about patient, patient/*.read means read all available data about patient. This type of scopes are defined by "Health Relationship Trust Profile for Fast Healthcare Interoperability Resources (FHIR) OAuth 2.0 Scopes"[7]
patient/UserInfo.permission	OPTIONAL. This scope value is used to access the UserInfo of patient (only given_name, family_name, picture, gender, birthdate). Users also may get this scope with patient/*.*.
fhir/patient	This scope is defined for confidential clients that are authorized with client credentials and that needs offline access to PDS (without any user, for internal data processing). They can request this scope to access all the data in PDS.
offline_access	This scope value requests that an **OAuth 2.0 Refresh Token**[8] be issued that can be used to obtain an Access Token that grants access to the End-User's UserInfo Endpoint even when the End-User is not present (not logged in).

3.1.3 User Claims

Fields of a user's profile information stored at **Onauth Server** are protected with claim values. Some scopes defined in **Onauth Server** enables access to some set of claim values. For example, being authorized for *profile* scope value authorizes access token for accessing; given_name, family_name, username, picture, gender, birthdate, zone_info, locale, pds_resource_type, pdm_user_type, pdm_care_givers, pdm_organization claim values. **Onauth Server** supports following standard claim values;

username	Unique username of the user which is used for login.
password	User password.
given_name	Name of the user.
family_name	Surname/Family name of the user.
middle_name	Middle name(s) of the user.
gender	Gender of the user.
birthdate	Birthdate of the use (in ISO date format e.g. 1966-03-03).
email	Email of the user.
address	JSON Object representing the address of the user (See Section 5.1.1 of OpenID **Connect Core 1.0**[9] for the details of address claim).
picture	URL of user's profile picture.
zone_info	User's time zone.
locale	Language tag (e.g. en-US) indicating the language of user.

Following extra claims are defined in **Onauth Server** for POWER2DM;

pdm_organization	**Id of the organization that the user is assigned to. For patients, this may be the Healthcare Organization that patient is getting the care. For physicians and nurses, this will be the Healthcare Organization they are working for.**
roles	Set of structural roles assigned to the user. See Section 2.1.1.3 for the description of roles.

[7] http://openid.net/specs/openid-heart-fhir-oauth2-1_0-2017-05-31.html
[8] https://tools.ietf.org/html/rfc6749
[9] http://openid.net/specs/openid-connect-core-1_0.html

care_team	Only used for patients. JSON object representing the care team assigned to the patient. Each parameter name represents a functional role and its value (array of string) provides the identifiers of users assigned to that functional role for patient. See Section 3.1.1.3 for the description of roles. The following is an example care_team claim;
	 `"care_team”: {`
	 `"care-manager": ["cad2a677-bc63-4d19-ae03-545fc872dd3”],`
	 `"care-supporter": [],`
	 `"self-care-manager": ["36d8b11b-1c68-421d-bc89-cc7a92471e61"],`
	 `"self-care-supporter": []`
	 `}`

3.1.4 User Registration

Onauth Server also provides endpoints for creating and updating users. The endpoint only accepts requests if a valid access token that is authorized for required scopes is provided at the Authorization header. Following sections explains the details of those endpoints;

3.1.4.1 Create/Invite Endpoint

To perform user registration, the authenticated end user (that will perform the user registration) should be authorized for the at least one *user_registration/{role}* scope (e.g. a user with *user_registration/patient* can register users with *patient* role). In POWER2DM realm, users who have *practitioner* or *nurse* role can register users with *patient* role and users who have *group_admin* role can register users either with *practitioner* or *nurse* role.

Onauth Server enables the authenticated users to register user either by providing all the required information or by only inviting him through an email address. In POWER2DM pilots, for patient registration, invitation mechanism will be used.

The schema and details of registration request is defined below. Note that, for patient registration, not all the parameters will be used.

```
POST power2dm/onauth/api/userinfo
Host: app.srdc.com.tr
Authorization: Bearer access_token
{
  username: String,
  password: String,
  family_name: String,
  given_name: String,
  middle_name: String,
  picture: String,
  gender: String,
  birthdate: String,
  pdm_user_type: String,

  zone_info: String,

  email: String,

  address: Address Claim object,

  locale: String,

  roles: List[String]

}
```

username	REQUIRED. Preferred username of the user which will be used for login. For patients, this will be the Study Number assigned by the practitioner that registers the patient.
password	OPTIONAL. User password
given_name	OPTIONAL. Name of the user. (for non-patient users)
family_name	OPTIONAL. Surname/Family name of the user. (for non-patient users)
middle_name	OPTIONAL. Middle name(s) of the user. (for non-patient users)
gender	OPTIONAL Gender of the user. (REQUIRED for patient)
birthdate	OPTIPNAL. Birth year of the user (e.g. 1966). (REQUIRED for patient)
email	OPTIONAL. Email of the user. (REQUIRED for patient)
address	OPTIONAL. JSON Object representing the address of the user (See Section 5.1.1 of **OpenID Connect Core 1.0**[10] for the details of address claim).
picture	OPTIONAL. URL of user's profile picture.
zone_info	OPTIONAL. User's time zone.
locale	OPTIONAL. Language tag (e.g. en-US) indicating the language of user.
roles	REQUIRED. Set of roles defined for the user.

For performing registration, the parameters defined above should be wrapped in a JSON object and sent to the UserInfo endpoint with Http POST request with access token that is authorized for registration at the Authorization header. Following is an example request for registering a patient;

```
POST power2dm/onauth/api/userinfo

Host: app.srdc.com.tr

Authorization: Bearer SlAV32hkKG

Content-Type: application/json

{

  "username": "patient_sas1",

  "gender": "female",

  "birthdate": "1966",

  "email": xyz@gmail.com

  "roles": ["patient"]

}
```

[10] http://openid.net/specs/openid-connect-core-1_0.html

Upon successful registration request, **Onauth Server** responds with 201 CREATED with stored UserInfo of the registered user, returned UserInfo may contain additional fields. The following is a non-normative example of a registration response;

```
HTTP/1.1 201 CREATED
Content-Type: application/json
Cache-Control: no-cache, no-store
Pragma: no-cache

{
  "sub":"36d8b11b-1c68-421d-bc89-cc7a92471e61",
  "preferred_username":"patient_sas1",
  "gender": "female",
  "care_team": {
    "care-manager": ["cad2a677-bc63-4d19-ae03-545fc872dd33"],
    "care-supporter": [],
    "self-care-manager": ["36d8b11b-1c68-421d-bc89-cc7a92471e61"],
    "self-care-supporter": []
  },
  "pdm_organization":"sas",
  "roles": ["patient"]
}
```

Here as you can see, an identifier (data store identifier) is assigned to patient which is given in “sub” element. Furthermore, the registrar is assigned to “care manager” role for patient where the patient is assigned for self-care-manager role automatically.

An unsuccessful create response would be 401 Unauthorized or 400 Bad Request with either FHIR Operation Outcome (which means there was a problem at PDS) or a message that explain the error. The following is a non-normative example of a registration response (selected username is already taken);

```
HTTP/1.1 400 Bad Request
Content-Type: text/plain
Cache-Control: no-cache, no-store
Pragma: no-cache
{
  “error”: “username_taken”
  “error_desc”: “Username is already taken. Please select another username”
}
```

3.1.4.2 Update Endpoint

Update operations (update of user profile) could only be performed by the owner of the UserInfo. In order to perform update operation, user must be authorized for the user_info_update scope. Parameters of the update operation are the same for create operation except username, password, role and pdm_user_type could not be updated from this endpoint. Those parameters will be ignored even if they exist in request body. The schema of update request is defined below;

```
PUT power2dm/onauth/api/userinfo

Host: app.srdc.com.tr

Authorization: Bearer access_token

{

  family_name: String,

  given_name: String,

  middle_name: String,

  picture: String,

  gender: String,

  birthdate: String,

  zone_info: String,

  email: String,

  address: Address Claim object,

  locale: String,

 }
```

For updating profile information, the parameters defined above should be wrapped in a JSON object and sent to the UserInfo endpoint with Http PUT request. Following is an example request for updating profile information,

```
POST power2dm/onauth/api/userinfo

Host: app.srdc.com.tr

Authorization: Bearer SlAV32hkKG

Content-Type: application/json

{

  "family_name": "Jane",

  "given_name": "Doe",

  "gender": "female",

  "birthdate": "1966-10-10",

}
```

Upon successful update request, **Onauth Server** responds with 200 OK with updated UserInfo at the body. The following is a non-normative example of an update response;

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store
Pragma: no-cache

{
  "sub":"36d8b11b-1c68-421d-bc89-cc7a92471e61",
  "given_name": "Doe",
  "family_name": "Jane",
  "preferred_username":"patient_sas1",
  "gender": "female",
  "pdm_user_type": "patient",
  "care_team": {
    "care-manager": ["cad2a677-bc63-4d19-ae03-545fc872dd33"],
    "care-supporter": [],
    "self-care-manager": ["36d8b11b-1c68-421d-bc89-cc7a92471e61"],
    "self-care-supporter": []
  },
  "pdm_organization":"sas"
}
```

An unsuccessful update response would be 401 Unauthorized or 400 Bad Request with a message that explains the error. The following is a non-normative example of a error response (request without name parameter):

```
HTTP/1.1 400 Bad Request
Content-Type: text/plain
Cache-Control: no-cache, no-store
Pragma: no-cache
The request content was malformed:
No usable value for name
```

3.1.5 Client Registration

In order to use services that are defined in this documentation, client applications must register themselves to **Onauth Server**. In the POWER2DM setup, all the POWER2DM components will be registered as clients to the Onauth server. **Onauth Server** uses "Dynamic Client Registration" process explained in **OpenID Connect Dynamic Client Registration 1.0**[11]. Following fields are important for

[11] http://openid.net/specs/openid-connect-registration-1_0.html

POWER2DM to register a new client (for the full list of fields and their details see section 2 of **OpenID Connect Dynamic Client Registration 1.0**[12]);

redirect_uris	REQUIRED. Array of Redirection URI values used by the client. One of these registered Redirection URI values MUST exactly match the *redirect_uri* parameter value used in each Authorization Request.
grant_types	OPTIONAL. List of the grant types client is declaring that it will restrict itself to using. The grant type values used by **Onauth Server** are: • authorization_code: It is used for authorization process using code. • client_credentials: It is used for authorization with client credentials. • refresh_token: It is used for refreshing access token
client_name	RECOMMENDED. Name of the Client to be presented to the end-user. If present, client name is displayed to end-user during approval process. e.g. POWER2DM SMSS Mobile Application
logo_uri	RECOMMENDED. URL that references a logo for the Client application. If present, the logo is displayed to the End-User during approval. The value of this field must point to a valid image file.
client_uri	OPTIONAL. URL to the home page of the client application. The value of this field must point to a valid Web page. If present, this URL is displayed to the end-user during approval.
contacts	OPTIONAL. Array of e-mail addresses of people responsible for this client application. If present, contacts are displayed to the end-user during approval.
policy_uri	OPTIONAL. URL that the Relying Party Client provides to the End-User to read about the how the profile data will be used
tos_uri	OPTIONAL. URL that the client application provides to the end-user to read about its terms of service. The value of this field must point to a valid web page. If present, this URL is displayed to the end-user during approval.
token_endpoint_auth_method	OPTIONAL. Requested Client Authentication method for the Token Endpoint. The defined options are *client_secret_basic* and *none*. If omitted, the default is *client_secret_basic*. Public apps should select this option as *none*.

For performing registration, the parameters defined above should be wrapped in a JSON object and sent to the Client endpoint with Http POST request. Following is an example request for registering a client;

```
POST power2dm/onauth/api/register

Host: app.srdc.com.tr

Content-Type: application/json

{

  "redirect_uris": [

    "http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/authorize-cb"

  ],

  "grant_types": [
```

[12] http://openid.net/specs/openid-connect-registration-1_0.html

```
    “authorization_code"
  ],
  "client_name": "SRDC Sample Client",
  "logo_uri": "http://www.srdc.com.tr/wp-content/uploads/2014/12/srdc-wp.png",
  "client_uri": "http://dummyclient.com",
  "contacts": [
    "tuncay@srdc.com.tr"
  ],
  "token_endpoint_auth_method": "none"
}
```

Upon successful registration request, **Onauth Server** responds with 201 CREATED with stored client metadata of the registered client application. Returned client metadata includes unique *client_id* and *client_secret* of the client. Public applications which select *token_endpoint_auth_method* as *none* may ignore client secret but private applications should keep their secret safe. The following is a non-normative example of a registration response;

```
HTTP/1.1 201 Created
Content-Type: application/json


{
  "client_id": "fdb74fa7-cb60-423f-87ee-89dba4c490c1",
  "client_secret": "r247v76ngrnga8d7hdojmmaj9m",
  "redirect_uris": [
    "http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/authorize-cb"
  ],
  "client_name": "SRDC Dummy Client",
  "client_uri": "http://dummyclient.com",
  "logo_uri": "http://www.srdc.com.tr/wp-content/uploads/2014/12/srdc-wp.png",
  "contacts": [
    "tuncay@srdc.com.tr"
  ],
  "token_endpoint_auth_method": "none",
  "scope": "profile openid address email patient user fhir/patient",
  "grant_types": [
    "authorization_code"
  ],
  "response_types": [
    "code"
  ],
  "application_type": "web",
  "request_object_encryption_enc": "A128CBC-HS256",
  "user_info_encrypted_response_enc": "A128CBC-HS256",
  "id_token_signed_response_alg": "RS256",
  "id_token_encrypted_response_enc": "A128CBC-HS256",
  "client_secret_expires_at": 0,
  "require_auth_time": false
}
```

An unsuccessful registration response would be 400 Bad Request with a message that explains the error. The following is a non-normative example of an error response (request without redirect_uri field):

```
HTTP/1.1 400 Bad Request
Content-Type: application/json


{
  "errorCode": "invalid_redirect_uri",
  "errorDesc": "At least 1 redirect_uri should be provided.",
}
```

3.2 Authentication & Authorization

Authentication and authorization of POWER2DM is managed by single sign on policy. Every client application should request user authorization from **Onauth Server** to get an access token. Applications can use that token to access users' protected information. Authentication and authorization mechanism of POWER2DM is implemented with respect to the rules and concepts defined in **OpenID Connect Core 1.0**[13]. Briefly, a client application should follow these steps to get an access token from the server (see Figure 1);

1. Client application prepares an authorization request and sends it to the authorize endpoint of **Onauth Server**,
2. **Onauth Server** authenticates the end user, obtains his/her consent for the client application,
3. **Onauth Server** redirects the end user back to the client application with *code*,
4. Client application sends *code* to the token endpoint of **Onauth Server,**
5. **Onauth Server** responds with the access token and id token,
6. Client uses the access token received from **Onauth Server to** access to the protected resources of the user.

Remaining part of this section will focus on the details of steps defined above.

[13] http://openid.net/specs/openid-connect-core-1_0.html

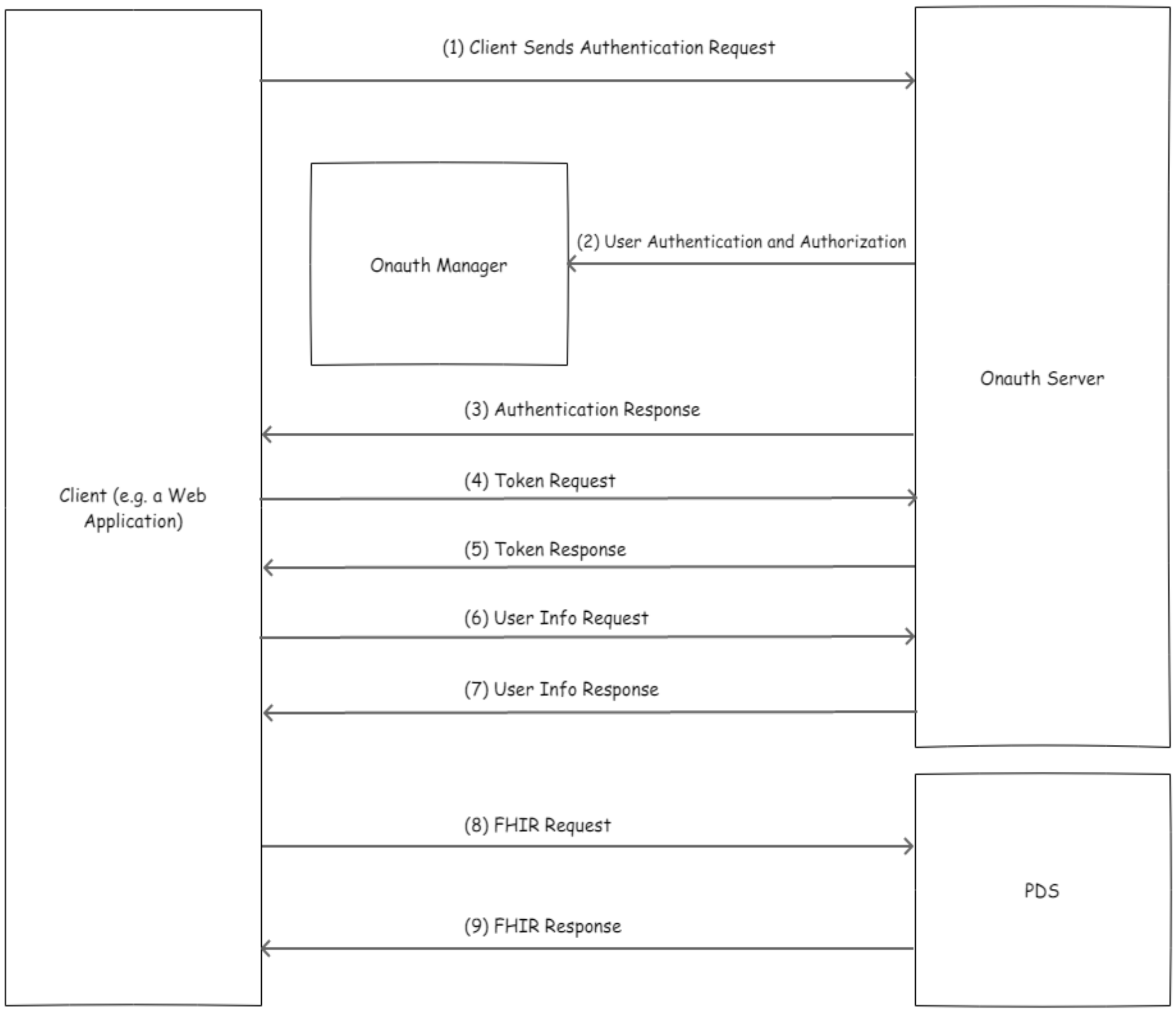


Figure 1 Authentication and authorization flow

3.2.1 Client Sends Authentication Request

Client application initiates authentication flow by making a HTTP Get request to the authorize endpoint of **Onauth Server** with following parameters;

response_type	This value must be *code*. This type of call requests an Access Token and an ID Token be returned from the Token Endpoint in exchange for the code value returned from the Authorization Endpoint.
client_id	Client Identifier valid at the Authorization Server.
scope	OpenID Connect authentication requests MUST contain the *openid* scope value. See Section 2.1.2 for more about the scope values defined for POWER2DM.
redirect_uri	Redirection Uri which the response will be sent. This Uri must match one of the Redirection Uri values of client.
state	Opaque value used to maintain state between the request and the call back. Typically, Cross-Site Request Forgery (CSRF, XSRF) mitigation is done by cryptographically binding the value of this parameter with a browser cookie.

If parameters provided by the client is correct, user is responded with 302 Found and redirected to the **Onauth Manager** for authentication. Erroneous requests are processed differently based on the error generated; If the client application provided an incorrect *client_id* or *redirect_uri*, user is redirected **Onauth Manager** with the proper error message, If the *client_id* and *redirect_uri* is correct then the user redirected to the client application's redirect URI address with proper error message. An example successful authentication request and response would be;

```
GET http://app.srdc.com.tr/power2dm/onauth/api/authorize?
  response_type=code
  &client_id=sample_client_id
  &redirect_uri=http%3A%2F%2Fapp.srdc.com.tr%2Fpower2dm%2F
                onauth%2Fsample-smart-client%2Fauth_callback.html
  &scope=openid%20profile%20patient/*.*%20offline_access
  &state=af0ifjsldkj
```

```
HTTP/1.1 302 Found
Location:
  http://app.srdc.com.tr/power2dm/onauth/onauthmanager/login?
     scope=profile%2Bopenid%2Bpatient%2F*.*%2Foffline_access
     &redirect_uri=http%3A%2F%2Fapp.srdc.com.tr%2Fpower2dm%2F
                   onauth%2Fsample-smart-client%2Fauth_callback.html
     &client_id=sample_client_id
     &response_type=code
```

An error on *client_id* or *redirect_uri* parameters would result in;

```
HTTP/1.1 302 Found
Location:
  http://app.srdc.com.tr/power2dm/onauth/onauthmanager/information?
    error=unauthorized_client
    &errorDesc=Invalid+client_id+parameter
```

3.2.2 End User Authentication

Upon successful authentication request by the client application, **Onauth Server** redirects user to the **Onauth Manager** and authentication process continues as follows;

- If the user does not already have an authenticated session, user is redirected login screen and asked for his/her credentials (see Figure 3).
- User is asked whether he/she accepts or denies client application's request for private information (see Figure 4).
- If End user is not a patient (e.g. physician), user is asked to select a patient from list of related patients (i.e. patients of the physician, see Figure 5). Only "Study Numbers" of patients will be shown to the user for the selection.

3.2.3 Client Receives Code

If user gives his/her consent to the client application, **Onauth Server** issues a *code* and delivers it with the *state* parameter to the client application. Client applications are responsible for checking whether the *state* parameter matches with the *state* parameter they've sent in authentication request. On contrary, if user denies the client application's request, user is still redirected to the client application but this time with proper error message.

For both cases, parameters to be delivered to the client application are added as query parameters to the URI of the client application using the *application/x-www-form-urlencoded* format. An example approved and denied authorization request would be;

```
HTTP/1.1 302 Found
Location:
  http://app.srdc.com.tr%/power2dm/onauth/sample-smart-client/auth_callb
ack.html?
    code=SplxlOBeZQQYbYS6WxSbI
    &state=af0ifjsldkj
```

```
HTTP/1.1 302 Found
Location:
 http://app.srdc.com.tr/power2dm/onauth/onauthmanager/information?
   error=access_denied
   &errorDesc=User+denied+application+access
   &state=af0ifjsldkj
```

3.2.4 Client Exchanges Code with Token

After getting the authorization code from the query parameters, client application makes a token request by presenting its authorization grant (in the form of an *authorization_code*) to the Token Endpoint of **Onauth Server**. In addition, clients should provide the same *redirect_uri* parameter that they used while performing the authentication request.

Token Endpoint of **Onauth Server** requires client authentication depending on the type of client which is making the token request. For public clients (e.g. User-agent based web applications, native mobile applications), client authentication using *client_id* and *client_secret* is not required but they must provide their *client_id* as a parameter. But confidential clients (who can store a secret safely) are required to authenticate themselves using HTTP Basic which is an authentication scheme which is defined at the Section 2.3.1 of the OAuth 2.0 Authorization Framework[14].

Requests to the Token Endpoint is made with HTTP Post request with the required parameters are added to the body of the request using *application/x-www-form-urlencoded* format. An example token request by a confidential and public client application would be;

```
POST power2dm/onauth/api/token HTTP/1.1
Host: app.srdc.com.tr
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded


grant_type=authorization_code
```

[14] https://tools.ietf.org/html/rfc6749

```
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=http%3A%2F%2Fyet.another.secure.server%2Fcb
```

```
POST power2dm/onauth/api/token HTTP/1.1 HTTP/1.1
Host: app.srdc.com.tr
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&client_id=fca8d49c-3f53-4886-8f7c-0cdf8a9d9a09
&redirect_uri=http%3A%2F%2Fapp.srdc.com.tr%2Fpower2dm%2F
                onauth%2Fsample-smart-client%2Fauth_callback.html
```

3.2.5 Client Receives Token

Upon successful token request, **Onauth Server** responds with the access token object in JSON format. Token response contains following fields;

access_token	Access token for accessing protected resources. This token can be used for UserInfo Endpoint and PDS.
Scope	Scopes that this access token is authorized for.
id_token	See Section 2 in **OpenID Connect 1.0**[15] for the details of ID token (JWT token that contains Claims about the Authentication event for the End user).
refresh_token	Refresh token. See *offline_access* scope at Section 2.1.2.
token_type	**OAuth 2.0 Token Type**[16] value. The value is always *Bearer*.
refresh_token	OPTIONAL. This scope value is used to access the UserInfo of patient (only given_name, family_name, picture, gender, birthdate). Users also may get this scope with patient/*.*.
expires_in	Expiration time of the access token in seconds since the response was generated.
patient	If the patient scope is present at the requested scopes, **Onauth Server** returns the patient identifier of patient that this access token is authorized for (If end user is patient himself, this is his own patient identifier. Otherwise, this is the patient identifier of the selected patient).

If token request results with an error, then proper error code and error description is returned to the client application in JSON format. The following is an example for successful token response;

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store
Pragma: no-cache


{
```

[15] http://openid.net/specs/openid-connect-core-1_0.html
[16] https://tools.ietf.org/html/rfc6749

```
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "scope": "patient/Observation.write patient/UserInfo.read…",
  "refresh_token": "8cjhj43d5893jic4dg4c22bh10b8i1",
  "expires_in": 3600,
  "id_token": "eyJraWQ....bePk2ob2tJFJ....iZz5Og",
  "patient": "00d545e8-7c86-4c16-84bc-7397f6e8741e"
}
```

The following is an example for erroneous token response;

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Cache-Control: no-cache, no-store
Pragma: no-cache

{
  "error": "invalid_grant"
  "errorDesc": "Only authorization_code is supported."
}
```

3.2.6 Client Uses Token to Access Information

Client applications can use access token received from **Onauth Server** to access UserInfo Endpoint and Pds resources by inserting the access token to the authorization header of the HTTP requests. Following sections exemplify such a request made to the secure endpoints.

3.2.6.1 Client Accesses User Info

An example UserInfo request and response;

```
GET power2dm/onauth/api/token HTTP/1.1
Host: app.srdc.com.tr
Authorization: Bearer SlAV32hkKG
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store
Pragma: no-cache

{
  "sub": "294d92b6-7610-40a3-a3f2-44443486d1f2"
  "name": "Martha",
  "surname": "Cook",
  "preferred_username": "martha_cook",
  "pdm_user_type": "patient",
  "care_team”: {
    "care-manager":"cad2a677-bc63-4d19-ae03-545fc872dd33"],
    "care-supporter":],
    "self-care-manager":"36d8b11b-1c68-421d-bc89-cc7a92471e61"],
    "self-care-supporter":[]
  },
  "pdm_organization": "lumc"
}
```

3.2.6.2 Client Accesses Patient’s User Info

Client applications also can access to the basic profile information of the patient in context (e.g. a physician authorized the client application for a patient). Following is the example request made to the token endpoint for a patient;

```
GET power2dm/onauth/api/userinfo/forbidden_patient HTTP/1.1
Host: app.srdc.com.tr
Authorization: Bearer SlAV32hkKG
```

Such a request may fail if the client tries to access to a patient profile other than the patient in context. Following is the example to error response of such a request;

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Cache-Control: no-cache, no-store
Pragma: no-cache

{
  “error”: “access_denied”
   errorDesc”: “Token is not authorized to access this patient.”
}
```

3.2.6.3 Client Access PDS Resources

PDS checks the reliability of access tokens by introspecting them from **Onauth Server.** Rather than reliability, token introspection also gives information to the PDS about the token. For example, even though a client application tries to run a query on all Observation records, PDS limits the query only to the patients that the access token is authorized for. Following is such a search request and response for Observation records;

```
GET /power2dm/pds-secure/Observation
Host: app.srdc.com.tr
```

```
Authorization: Bearer SlAV32hkKG
```

```
HTTP/1.1 200 OK
Content-Type: application/json
{
 "resourceType": "Bundle",
 "id": "ffc3a75a-32a5-435a-9597-e2c8c7b64c95",
 "type": "searchset",
 "total": 1,
 "link": [
  {
    "relation": "self",
    "url": "http://app.srdc.com.tr/power2dm/pds-secure/Observation?patie
nt=00d545e8-7c86-4c16-84bc-7397f6e8741e"
  }
 ],
 "entry": [
  {
   "fullUrl": "http://app.srdc.com.tr/power2dm/pds-secure/Observation/14
a94454-410a-4e93-8ee5-691460c7bba5",
   "resource": {
    "resourceType": "Observation"…
   }
  }
 ]
}
```

3.2.7 Refreshing Access Tokens

Client applications can use the *expires_in* field from the authorization response to determine when its access token will expire. After an access token expires, it may be possible to request an updated token without user intervention, if the app asked for a refresh token via the *offline_access* scope and the server supplied a *refresh_token* in the authorization response. To obtain a new access token, the app issues an HTTP POST to the Token Endpoint, with content-type *application/x-www-form-urlencoded*

For confidential clients, an Authorization header using HTTP Basic authentication is required, where the username is the app's *client_id* and the password is the app's *client_secret*. For public clients, authentication is not possible (and thus not required) but *client_id* must be provided.

Client makes a Token Request by presenting its Authorization Grant (in the form of an Refresh Token) to the Token Endpoint using the *refresh_token* value for *grant_type* parameter with refresh_token value. In addition, *scope* parameter could also be defined and if present, scope value must be a strict sub-set of the scopes granted in the original request (no new permissions can be obtained at refresh time). A missing value indicates a request for the same scopes which are granted in the original request.

The following is a non-normative example of a Token Request made by confidential client (with line wraps for the display purposes only):

```
POST power2dm/onauth/api/token HTTP/1.1
Host: app.srdc.com.tr
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&refresh_token=8cjhj43d5893jic4dg4c22bh10b8i1
```

The following is a non-normative example of a Token Request made by public client (with line wraps for the display purposes only):

```
POST power2dm/onauth/api/token HTTP/1.1
Host: app.srdc.com.tr
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&refresh_token=8cjhj43d5893jic4dg4c22bh10b8i1
&client_id=fca8d49c-3f53-4886-8f7c-0cdf8a9d9a09
```

A successful *refresh_token* response is same with the *access_token* response except *id_token* may not be present at the body. If a new refresh token is present at the response, clients should replace the old refresh token with the new one.

3.2.8 Authorization with Client Credentials

POWER2DM also has some applications (confidential clients) that require access to the private information but have not any UI elements (i.e. Communication Engine). Such client applications may request access tokens by authenticating themselves with HTTP Basic scheme. To obtain an access token using client credentials, client makes a request to the Token Endpoint using client*_credentials* value for the *grant_type* parameter. Only certain clients are given such scope. The following is an example of such a token request and response;

```
POST power2dm /onauth/api/token
Host: app.srdc.com.tr
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=client_credentials
&scope=fhir/patient
```

```
HTTP/1.1 200 OK
```

```
Cache-Control: no-cache, no-store

Pragma: no-cache

{

  "access_token": "jfieej3hic5gj55ai50cded6ag3b5j",

  "token_type": "Bearer",

  "scope": [

    "fhir/patient"

  ],

  "expires_in": 3600

}
```

3.3 Auditing

Audit Server implements FHIR STU3 (3.0.1) specification[17] for FHIR Create, Read, VRead, History and Search operations for FHIR Audit Event resource type[18]. POWER2DM PDS uses "FHIR Create" service to send the audits for each data access/update to PDS. Onauth Server generates and send audits for user authentications. The following record shows a PDS generated audit for a query to PDS. AuditEvent records basically includes the following information;

- **Type of event** (type, subtype): This gives the type of event that audit is generated for. The given example indicates that the audit is for a FHIR search event (data search).
- **Time of event** (recorded)
- **Outcome of event** (outcome, outcomeDesc)**:** Indicates whether the operation is successful or not. e.g. If search operation is successfully completed
- **Agents of the event** (agent): This defines the parties (applications) that performs the operation. In this example, it is seen that the search is performed from a specific IP address (**95.9.71.171**) by the practitioner given with id (4b19f8a6-9f3f-40c0-988f-2e7894ee7416). The other agent is the performed of the search which is the PDS itself.
- **Entity of the event** (entity): This part provides the entities that the operation is performed on. In the example, the first one indicates the search is performed for Patient with id (7fd4c6c5-9aa3-4bb1-9bd0-33d4e7ebdabf) and the second entity provides the query (encoded in Base64) that is performed on Observations of patient.

```
{
    "resourceType": "AuditEvent",
    "type": {
        "system": "http://hl7.org/fhir/audit-event-type",
        "code": "rest"
    },
    "subtype": [
        {
            "system": "http://hl7.org/fhir/restful-interaction",
            "code": "search-type"
        }
    ],
    "action": "R",
    "recorded": "2017-10-25T14:10:42.535+03:00",
    "outcome": "0",
    "outcomeDesc": "200 OK: OK",
    "agent": [
```

[17] http://hl7.org/fhir/index.html
[18] http://hl7.org/fhir/auditevent.html

```
        {
            "role": [
                {
                    "coding": [
                        {
                            "system" : "http://nema.org/dicom/dicm",
                            "code" : "110153"
                        }
                    ]
                }
            ],
            "requestor": true,
            "network": {
                "address": "95.9.71.171",
                "type": "2"
            },
           "reference": {
                 "reference": "Practitioner/4b19f8a6-9f3f-40c0-988f-2e7894ee7416"
            }
        },
        {
            "role": [
                {
                    "coding": [
                        {
                            "system": "http://nema.org/dicom/dicm",
                            "code": "110152"
                        }
                    ]
                }
            ],
            "name": "POWER2DM Personal Health Data Store",
            "requestor": false,
            "network": {
                "address": "http://127.0.0.1:8080/pds",
                "type": "2"
            }
        }
    ],
    "source": {
        "identifier": {
            "value": "http://127.0.0.1:8080/pds"
        }
    },
    "entity": [
         {
            "type": {
                "system": "http://hl7.org/fhir/audit-entity-type",
                "code": "1"
            },
            "role": {
                "system": "http://hl7.org/fhir/object-role",
                "code": "1"
            },
            "reference": { "reference": "Patient/7fd4c6c5-9aa3-4bb1-9bd0-33d4e7ebdabf"}
        },
        {
            "type": {
                "system": "http://hl7.org/fhir/resource-types",
                "code": "Observation"
            },
            "role": {
                "system": "http://hl7.org/fhir/object-role",
                "code": "24"
            },
            "query": "aHR0cDovLzEyNy4wLjAuMTo4MDgwL3Bkcy9QYXRpZW50"
        }
    ]
}
```

4 Demonstration of Functionalities

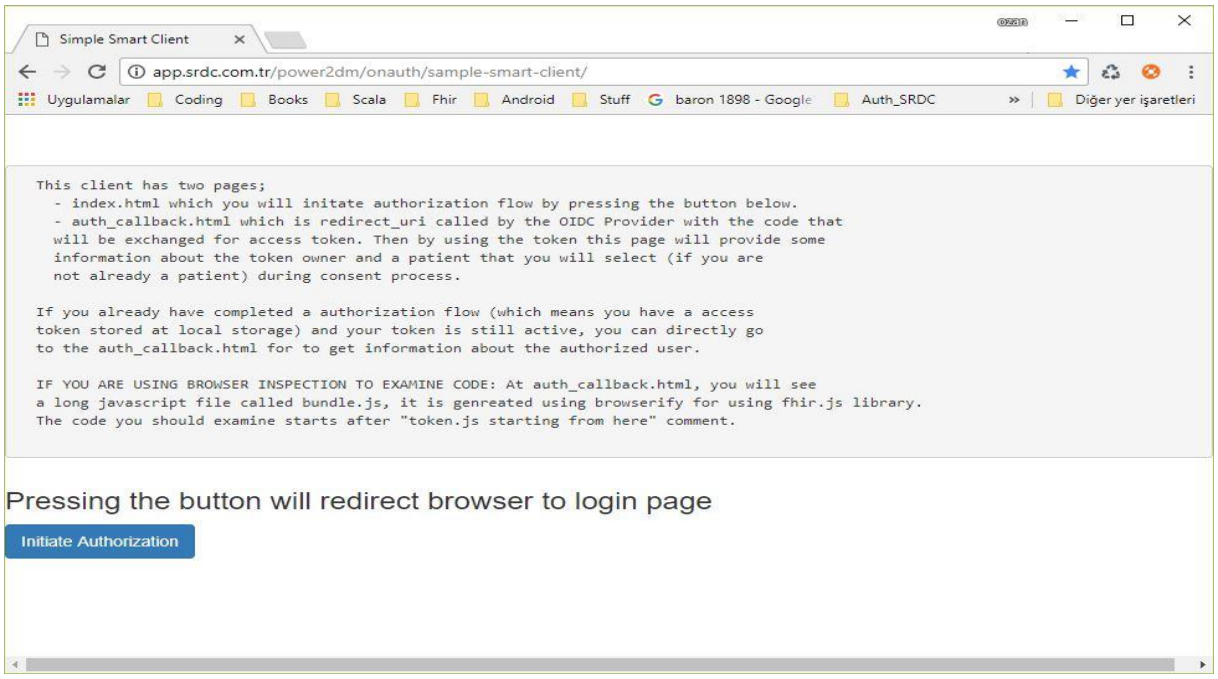
Onauth Server and **Onauth Manager** are deployed on our cloud for demonstration and for testing environment for other partners. **Onauth Server** is accessible on http://app.srdc.com.tr/power2dm/onauth/api and **Onauth Manager** is accessible on http://app.srdc.com.tr/power2dm/onauth/onauth-manager.

4.1 Authorization & Authentication

This section demonstrates the example **OpenID Connect Authorization Code Flow**[19] using the **Onauth Manager** and a sample client application which is developed only for demonstration purposes and to be an example for other POWER2DM components. You can access the sample client which is deployed for demonstration purposes on http://app.srdc.com.tr/power2dm/onauth/sample-smart-client.

4.1.1 Single Sign On

When you open the sample client, first screen you will see is the "Initiate Authorization" button with some explanations;



```
This client has two pages;
  - index.html which you will initate authorization flow by pressing the button below.
  - auth_callback.html which is redirect_uri called by the OIDC Provider with the code that
  will be exchanged for access token. Then by using the token this page will provide some
  information about the token owner and a patient that you will select (if you are
  not already a patient) during consent process.

If you already have completed a authorization flow (which means you have a access
token stored at local storage) and your token is still active, you can directly go
to the auth_callback.html for to get information about the authorized user.

IF YOU ARE USING BROWSER INSPECTION TO EXAMINE CODE: At auth_callback.html, you will see
a long javascript file called bundle.js, it is genreated using browserify for using fhir.js library.
The code you should examine starts after "token.js starting from here" comment.
```

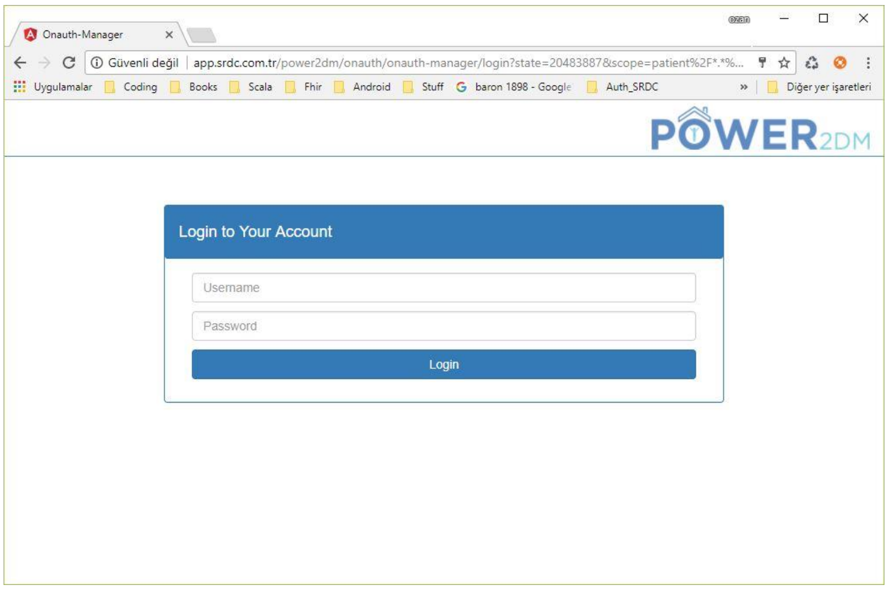
Pressing the button will redirect browser to login page

Figure 2 Client sends authorization request

Clicking "Initiate Authorization" button will send authorization request to the authorize endpoint of **Onauth Server** as explained in Section 3.2.1.1. On successful request, **Onauth Server** redirects user to the log in page of the **Onauth Manager**.

Username/Password authentication is the default authentication mechanism for all users. However, users can select another alternative which is 2-factor authentication in which case they should also register their phone number to their account. If that is the case, after confirming username and password, Onauth server sends a verification code via SMS to the user and user is expected to enter this code in the next step.

[19] http://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth



Güvenli değil | app.srdc.com.tr/power2dm/onauth/onauth-manager/login?state=20483887&scope=patient%2F*.*%...

Login to Your Account

Username

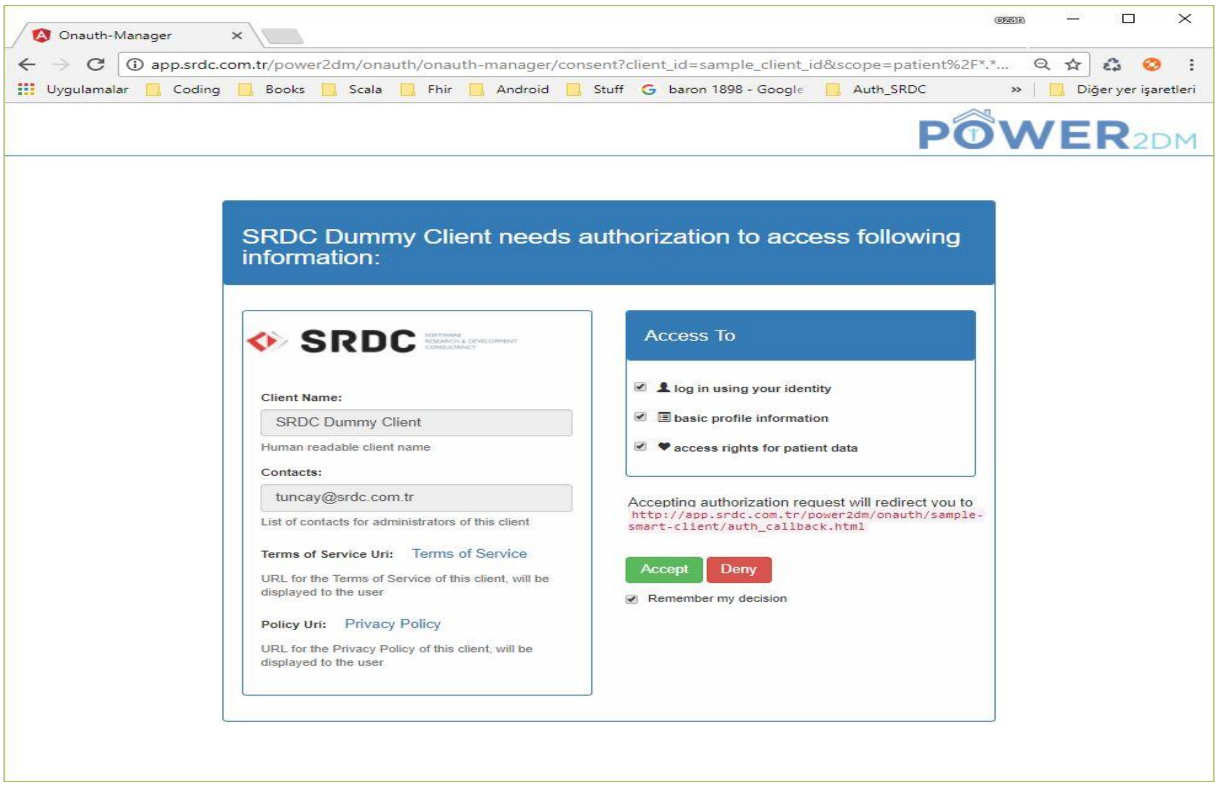
Password

Login

Figure 3 User is redirected to Sign In page

4.1.2 User Consent

Upon successful sign on, user is redirected to the consent page where he/she is asked to authorize the client to act on behalf of him/her as shown in Figure 4. Here the user can check the registered data for client and which data client will access to on behalf of him.



app.srdc.com.tr/power2dm/onauth/onauth-manager/consent?client_id=sample_client_id&scope=patient%2F*.*...

SRDC Dummy Client needs authorization to access following information:

Client Name: SRDC Dummy Client

Human readable client name

Contacts: tuncay@srdc.com.tr

List of contacts for administrators of this client

Terms of Service Uri: Terms of Service

URL for the Terms of Service of this client, will be displayed to the user

Policy Uri: Privacy Policy

URL for the Privacy Policy of this client, will be displayed to the user

Access To

- ✓ log in using your identity
- ✓ basic profile information
- ✓ access rights for patient data

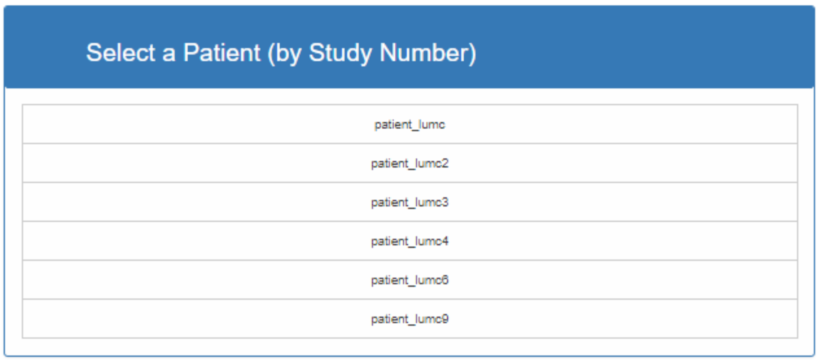
Accepting authorization request will redirect you to `http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/auth_callback.html`

Accept Deny

✓ Remember my decision

Figure 4 User is asked for consent

In the next step, if user is not a patient, he/she is asked to select a patient to set the authorization context to a specific patient. In other words, the Shared Decision-Making Application, used by care providers never access to the identity data of patients as the patient selection is done via the patient selection screen as shown in Figure 5 internally.



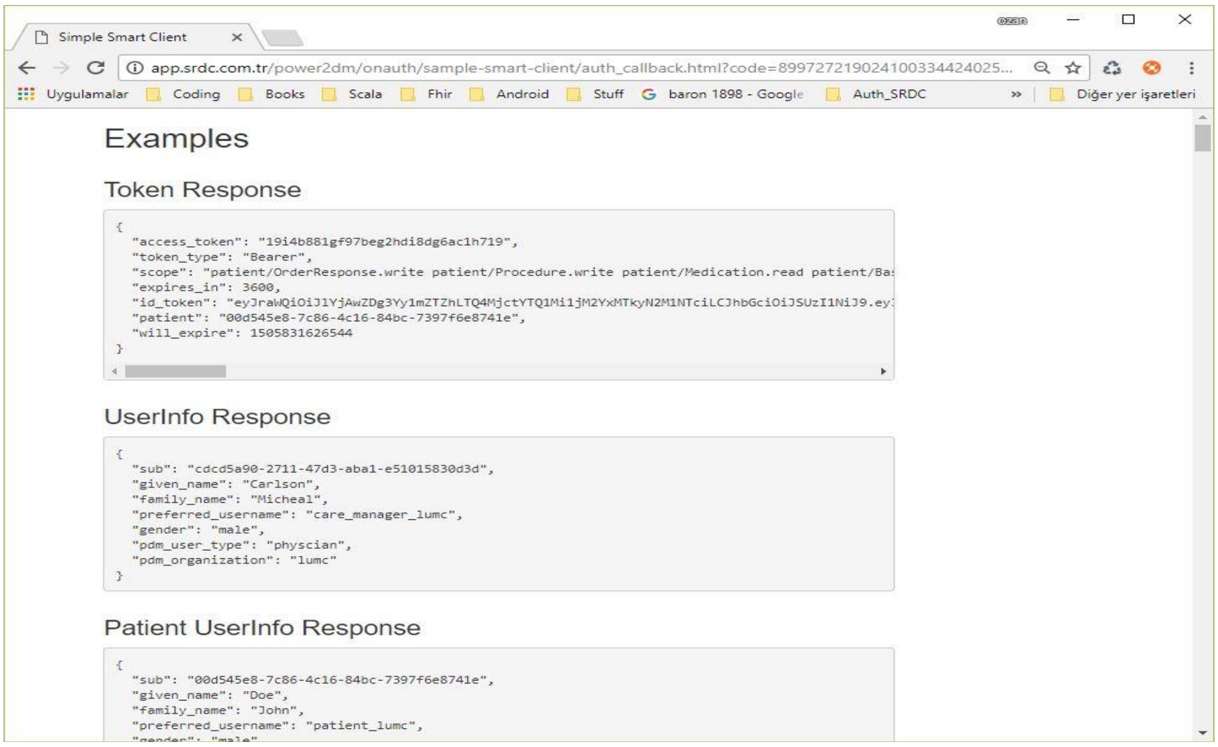
Select a Patient (by Study Number)

patient_lumc
patient_lumc2
patient_lumc3
patient_lumc4
patient_lumc6
patient_lumc9

Figure 5 User selects patient

4.1.3 Requests with Sample Client

After user approved the client, he/she is directed back to the client with the code and client exchanges code for an access token as explained in Section 2.2.1.3. Below, Figure 6, is the snapshot of sample client after exchanging code with the token;



Examples

Token Response

```
{
  "access_token": "19i4b881gf97beg2hdi8dg6ac1h719",
  "token_type": "Bearer",
  "scope": "patient/OrderResponse.write patient/Procedure.write patient/Medication.read patient/Ba
  "expires_in": 3600,
  "id_token": "eyJraWQiOiJ1YjAwZDg3Yy1mZTZhLTQ4MjctYTQ1Mi1jM2YxMTkyN2M1NTciLCJhbGciOiJSUzI1NiJ9.ey
  "patient": "00d545e8-7c86-4c16-84bc-7397f6e8741e",
  "will_expire": 1505831626544
}
```

UserInfo Response

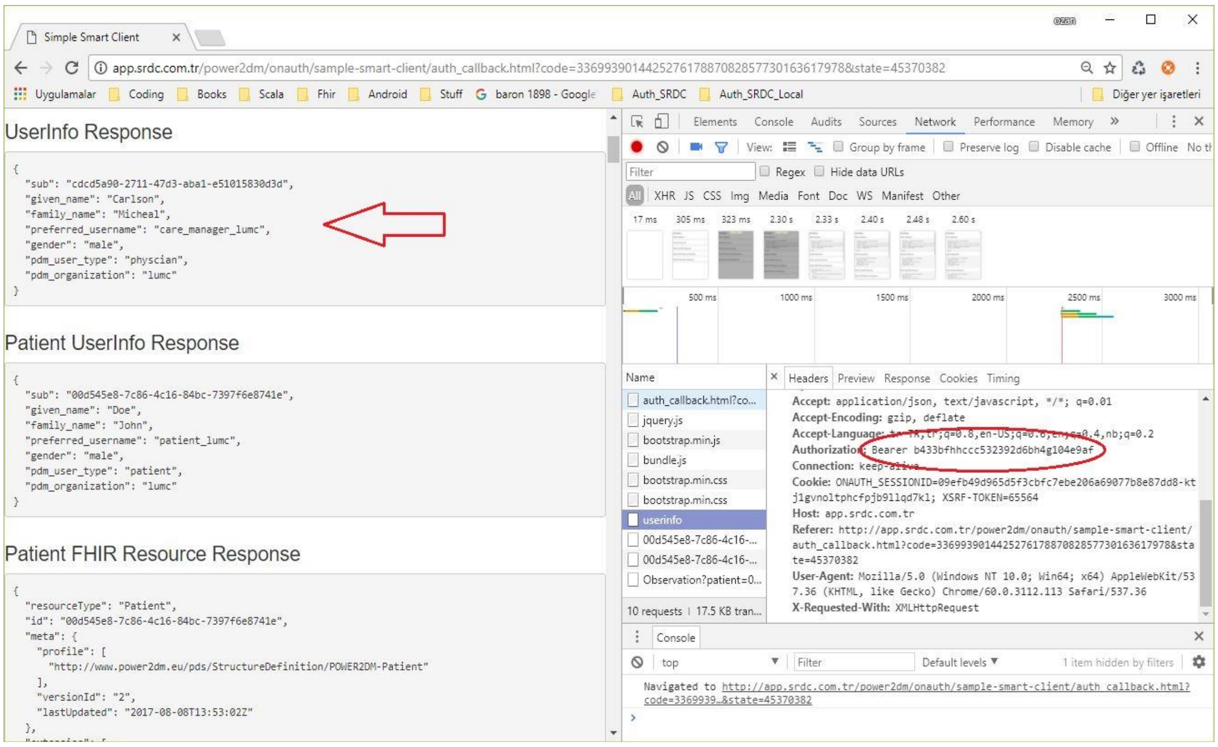
```
{
  "sub": "cdcd5a90-2711-47d3-aba1-e51015830d3d",
  "given_name": "Carlson",
  "family_name": "Micheal",
  "preferred_username": "care_manager_lumc",
  "gender": "male",
  "pdm_user_type": "physcian",
  "pdm_organization": "lumc"
}
```

Patient UserInfo Response

```
{
  "sub": "00d545e8-7c86-4c16-84bc-7397f6e8741e",
  "given_name": "Doe",
  "family_name": "John",
  "preferred_username": "patient_lumc",
  "gender": "male"
```

Figure 6 Client exchanges code for token

Having an access token, client can make requests to the secure endpoints by passing access token in authorization header. You can find example requests below in Figure 7 and Figure 8.


app.srdc.com.tr/power2dm/onauth/sample-smart-client/auth_callback.html?code=336993901442527617887082857730163617978&state=45370382

UserInfo Response

```
{
  "sub": "cdcd5a90-2711-47d3-aba1-e51015830d3d",
  "given_name": "Carlson",
  "family_name": "Micheal",
  "preferred_username": "care_manager_lumc",
  "gender": "male",
  "pdm_user_type": "physcian",
  "pdm_organization": "lumc"
}
```

Patient UserInfo Response

```
{
  "sub": "00d545e8-7c86-4c16-84bc-7397f6e8741e",
  "given_name": "Doe",
  "family_name": "John",
  "preferred_username": "patient_lumc",
  "gender": "male",
  "pdm_user_type": "patient",
  "pdm_organization": "lumc"
}
```

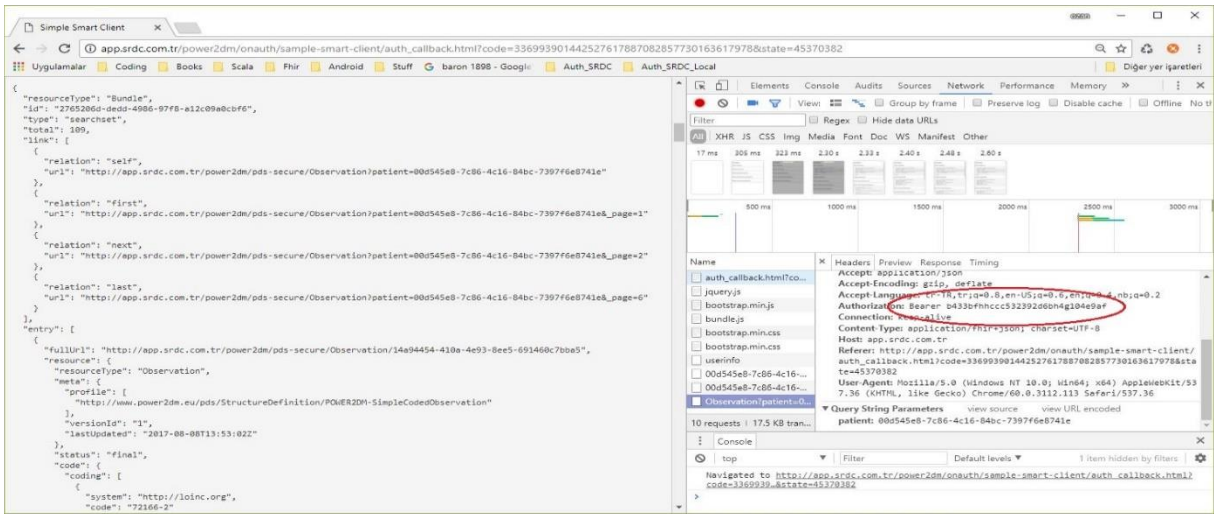
Patient FHIR Resource Response

```
{
  "resourceType": "Patient",
  "id": "00d545e8-7c86-4c16-84bc-7397f6e8741e",
  "meta": {
    "profile": [
      "http://www.power2dm.eu/pds/StructureDefinition/POWER2DM-Patient"
    ],
    "versionId": "2",
    "lastUpdated": "2017-08-08T13:53:02Z"
  },
```

```
Accept: application/json, text/javascript, */*; q=0.01
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.8,en-US;q=0.6,en;q=0.4,nb;q=0.2
Authorization: Bearer b433bfhhccc532392d6bh4g104e9af
Connection: keep-alive
Cookie: ONAUTH_SESSIONID=09efb49d965d5f3cbfc7ebe206a69077b8e87dd8-kt
j1gvnoltphcfpjb91lqd7kl; XSRF-TOKEN=65564
Host: app.srdc.com.tr
Referer: http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/
auth_callback.html?code=336993901442527617887082857730163617978&sta
te=45370382
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
7.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
X-Requested-With: XMLHttpRequest
```

```
Navigated to http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/auth_callback.html?
code=3369939…&state=45370382
```

Figure 7 Client fetches user info


app.srdc.com.tr/power2dm/onauth/sample-smart-client/auth_callback.html?code=336993901442527617887082857730163617978&state=45370382

```
{
  "resourceType": "Bundle",
  "id": "2765206d-dedd-4986-97f8-a12c09a0cbf6",
  "type": "searchset",
  "total": 109,
  "link": [
    {
      "relation": "self",
      "url": "http://app.srdc.com.tr/power2dm/pds-secure/Observation?patient=00d545e8-7c86-4c16-84bc-7397f6e8741e"
    },
    {
      "relation": "first",
      "url": "http://app.srdc.com.tr/power2dm/pds-secure/Observation?patient=00d545e8-7c86-4c16-84bc-7397f6e8741e&_page=1"
    },
    {
      "relation": "next",
      "url": "http://app.srdc.com.tr/power2dm/pds-secure/Observation?patient=00d545e8-7c86-4c16-84bc-7397f6e8741e&_page=2"
    },
    {
      "relation": "last",
      "url": "http://app.srdc.com.tr/power2dm/pds-secure/Observation?patient=00d545e8-7c86-4c16-84bc-7397f6e8741e&_page=6"
    }
  ],
  "entry": [
    {
      "fullUrl": "http://app.srdc.com.tr/power2dm/pds-secure/Observation/14a94454-410a-4e93-8ee5-691460c7bba5",
      "resource": {
        "resourceType": "Observation",
        "meta": {
          "profile": [
            "http://www.power2dm.eu/pds/StructureDefinition/POWER2DM-SimpleCodedObservation"
          ],
          "versionId": "1",
          "lastUpdated": "2017-08-08T13:53:02Z"
        },
        "status": "final",
        "code": {
          "coding": [
            {
              "system": "http://loinc.org",
              "code": "72166-2"
```

```
Accept: application/json
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.8,en-US;q=0.6,en;q=0.4,nb;q=0.2
Authorization: Bearer b433bfhhccc532392d6bh4g104e9af
Connection: keep-alive
Content-Type: application/fhir+json; charset=UTF-8
Host: app.srdc.com.tr
Referer: http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/
auth_callback.html?code=336993901442527617887082857730163617978&sta
te=45370382
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
7.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
▼ Query String Parameters    view source    view URL encoded
patient: 00d545e8-7c86-4c16-84bc-7397f6e8741e
```

```
Navigated to http://app.srdc.com.tr/power2dm/onauth/sample-smart-client/auth_callback.html?
code=3369939…&state=45370382
```

Figure 8 Client fetches patient's observations

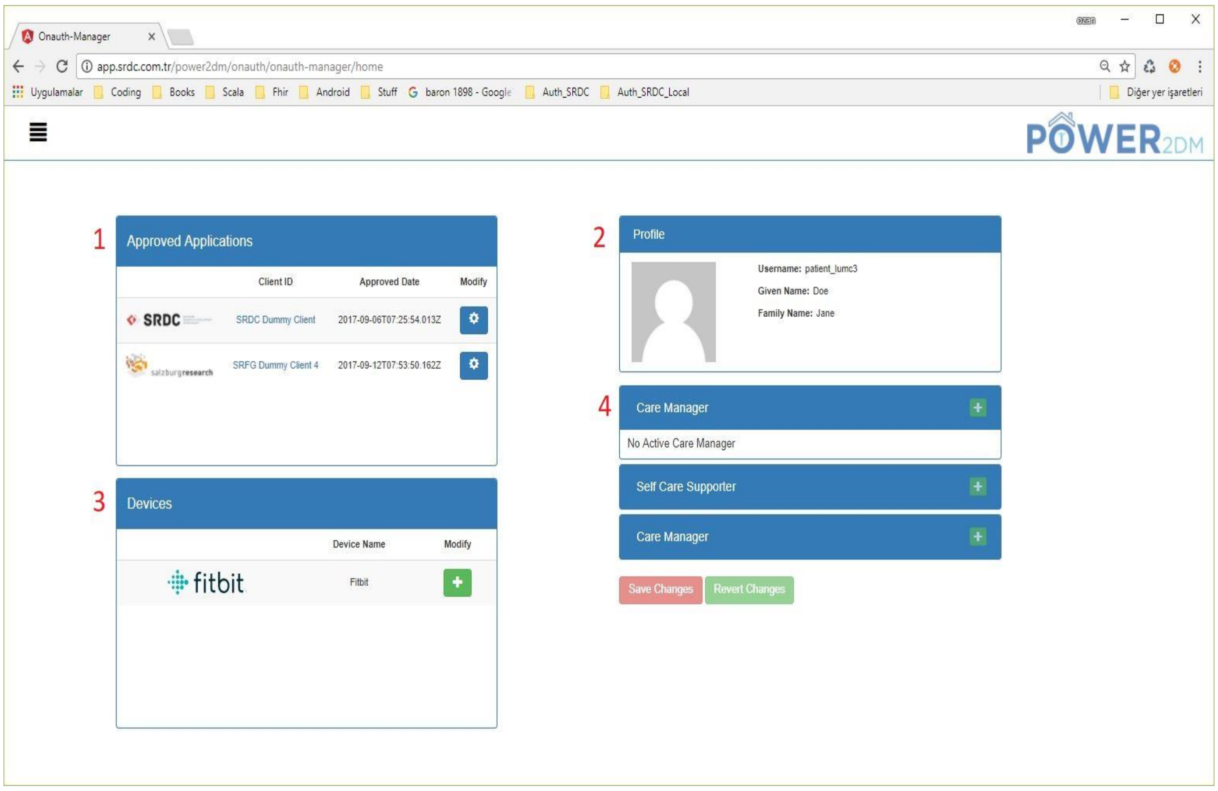
4.2 Onauth Manager

4.2.1 Home Screen

4.2.1.1 Patient

Patient home screen consists of four different components (see Figure 9);

- Component-1: List of approved applications of the user; users can view details of each approved client and remove their approval.
- Component-2: Basic profile information of the user.
- Component-3: List of devices or 3rd party data providers that user may authorize POWER2DM to access patient's own data; users can authorize/deny POWER2DM for the device/application.
- Component-4: Care team of the user; user may modify the care team based on his/her permissions, but he cannot unassign care managers.



Approved Applications

	Client ID	Approved Date	Modify
SRDC	SRDC Dummy Client	2017-09-06T07:25:54.013Z	
salzburgresearch	SRFG Dummy Client 4	2017-09-12T07:53:50.162Z	

Profile

Username: patient_lumc3
Given Name: Doe
Family Name: Jane

Devices

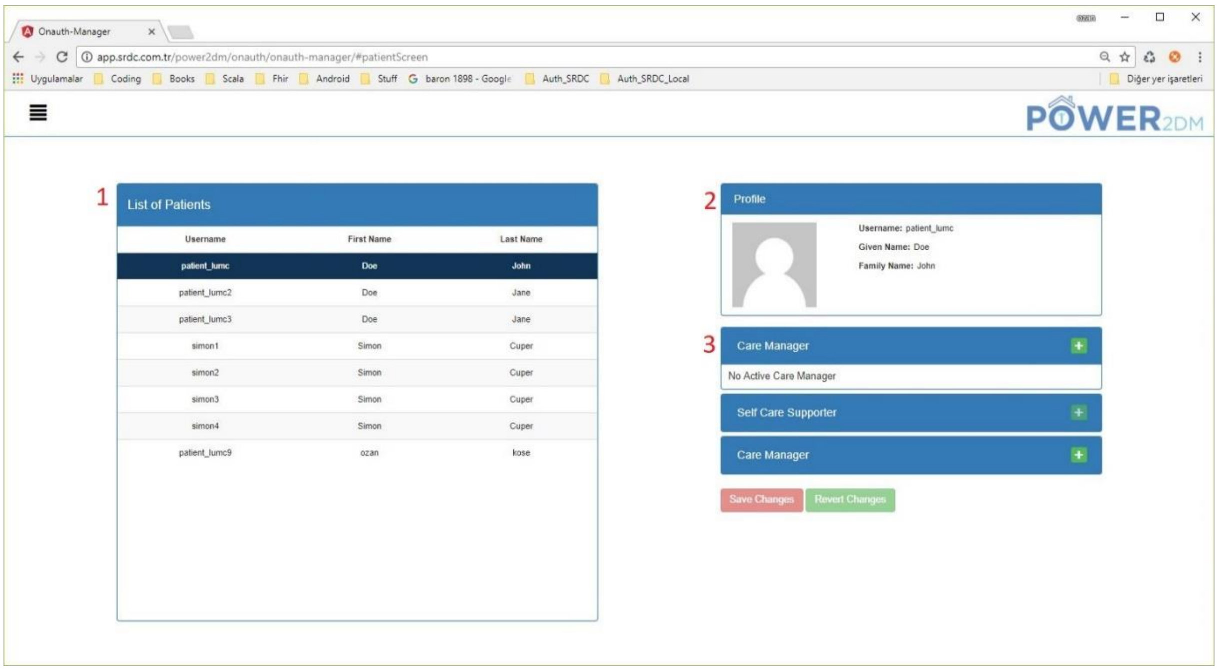
	Device Name	Modify
fitbit	Fitbit	

Figure 9 Patient home screen

4.2.1.2 Care Provider

Home screen for care provider consists of 3 different components (see Figure 10);

- Component-1: List of patients which the user is authorized for; user may switch between patients to see the details.
- Component-2: Basic profile information of the selected patient.
- Component-3: Care team of the selected patient; user may modify patient's care team based on his/her permissions. He can add new care managers or care supporters.



List of Patients

Username	First Name	Last Name
patient_lumc	Doe	John
patient_lumc2	Doe	Jane
patient_lumc3	Doe	Jane
simon1	Simon	Cuper
simon2	Simon	Cuper
simon3	Simon	Cuper
simon4	Simon	Cuper
patient_lumc9	ozan	kose

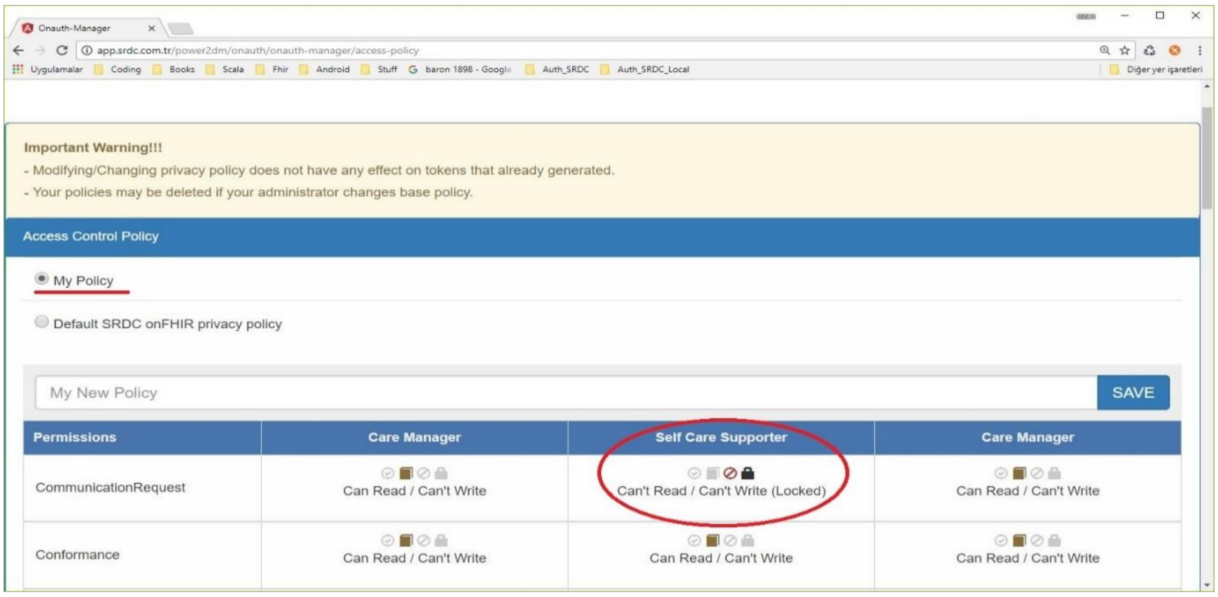
Profile

Username: patient_lumc
Given Name: Doe
Family Name: John

Figure 10 Care Provider home screen

4.2.2 Policy Management

Patients can create several policies of their own by extending the base policy defined for POWER2DM. Policy management screen contains a table where each column represents a functional role, each row represents a resource set and intersection of a row and column represents that role's permission for the resource set. In addition, patients also can apply, delete and update an existing policy. Rules that are marked as locked couldn't be modified by the patients.



Important Warning!!!
- Modifying/Changing privacy policy does not have any effect on tokens that already generated.
- Your policies may be deleted if your administrator changes base policy.

Access Control Policy

● My Policy

○ Default SRDC onFHIR privacy policy

My New Policy SAVE

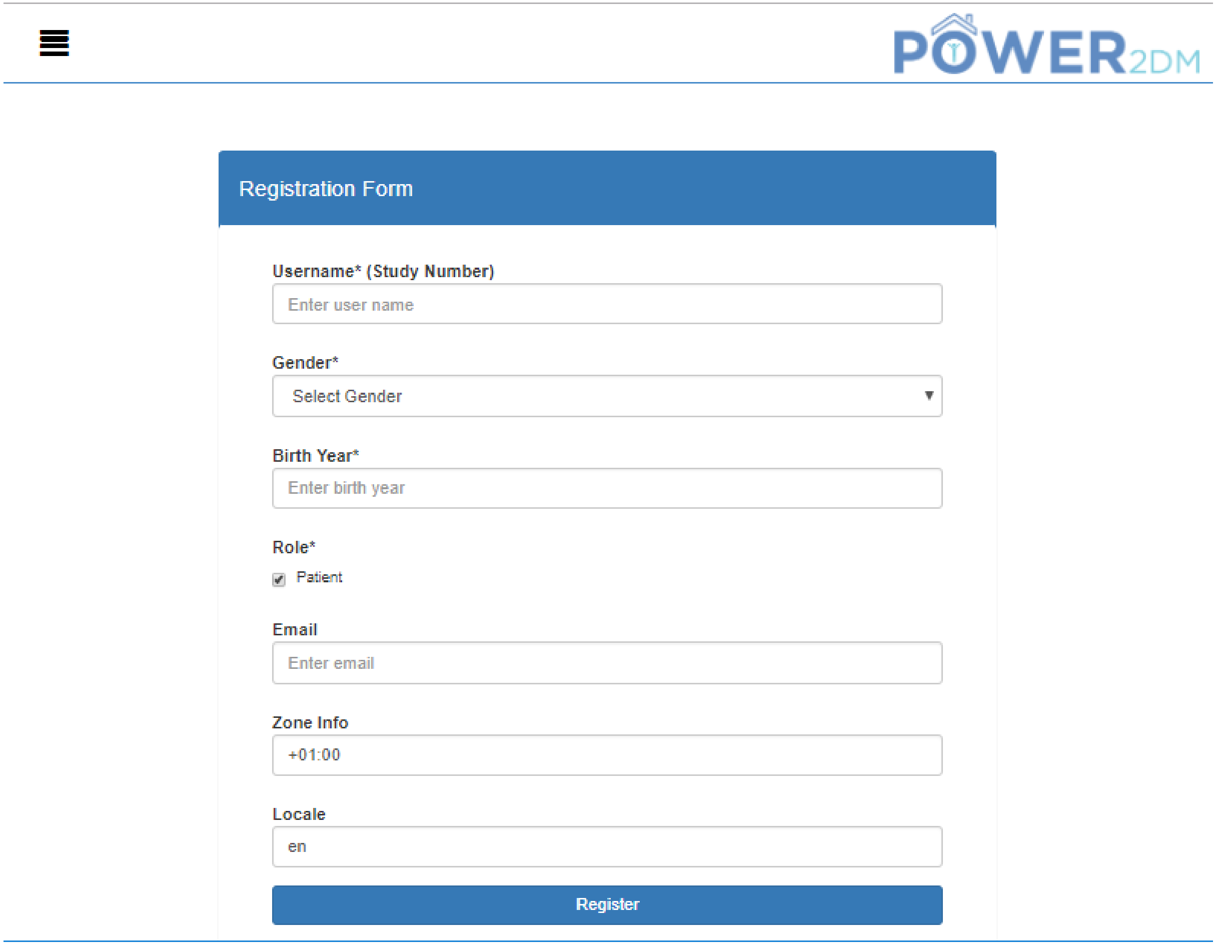
Permissions	Care Manager	Self Care Supporter	Care Manager
CommunicationRequest	Can Read / Can't Write	Can't Read / Can't Write (Locked)	Can Read / Can't Write
Conformance	Can Read / Can't Write	Can Read / Can't Write	Can Read / Can't Write

Figure 11 Policy management screen

Although, this interface provides detailed policy management to patient, after discussions with piloting partners, it is decided to go with a single POWER2DM access control policy that simplifies the procedure and POWER2DM care which will be defined by POWER2DM realm manager from the same interface.

4.2.3 User Registration

Users can register new users if they have permission for user registration by using the user registration form provided by this screen. Type of user and permissions that newly registered user will get are dependent on the roles are set during registration process. Below, you can find an example patient registration performed by a practitioner;



Registration Form

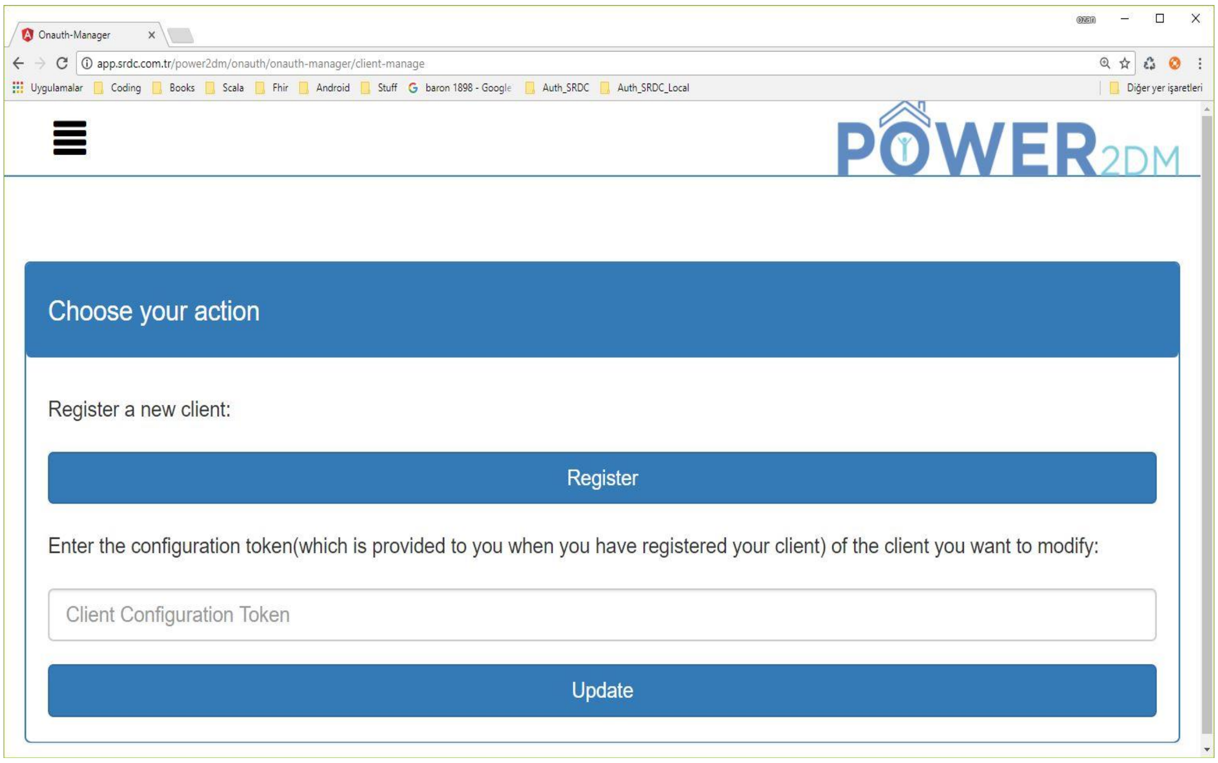
Username* (Study Number)	Enter user name
Gender*	Select Gender
Birth Year*	Enter birth year
Role*	[✓ Patient]
Email	Enter email
Zone Info	+01:00
Locale	en

Figure 12 Patient Registration form

Upon successful registration, practitioner who registered this new patient with username "example_new_user" is added to the care team of the newly registered patient.

4.2.4 Client Registration

In order to use **OpenID Connect 1.0** endpoints defined in this document, clients should be registered to **Onauth Server**. Users can register a new client or modify an existing client application using the client registration form provided by this screen;



Choose your action

Register a new client:

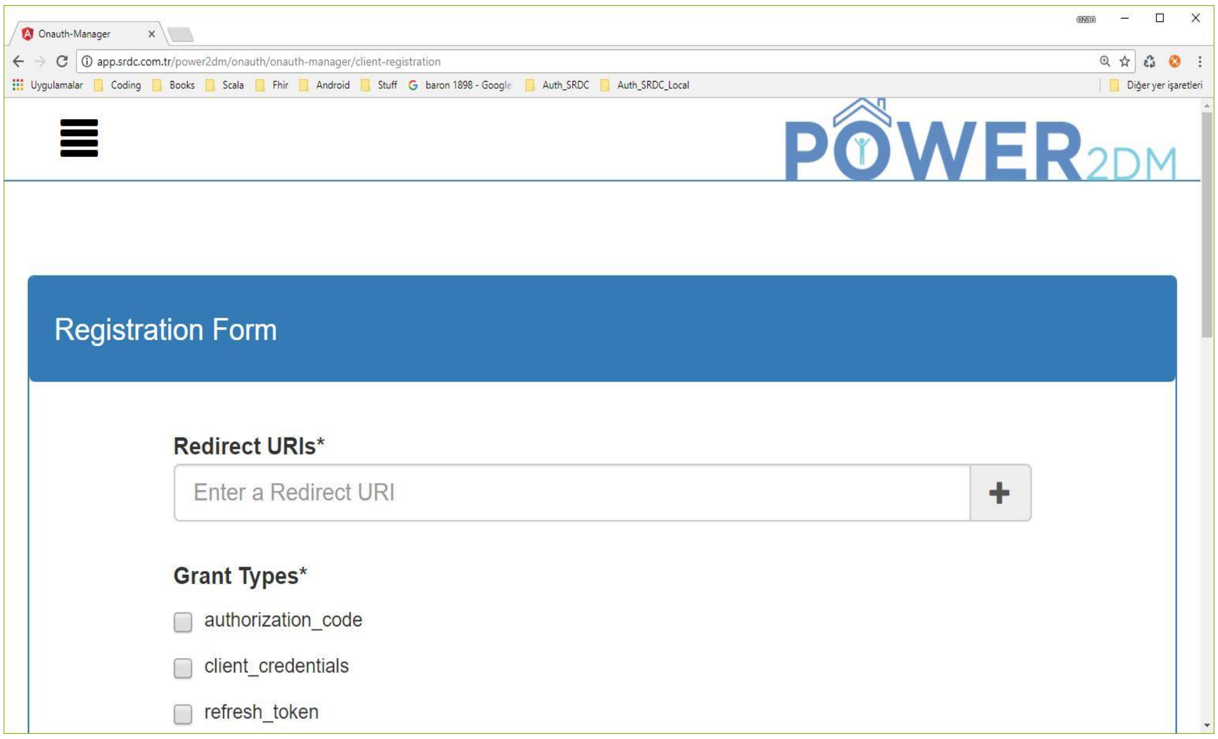
Register

Enter the configuration token(which is provided to you when you have registered your client) of the client you want to modify:

Client Configuration Token

Update

Figure 13 Client registration menu



Registration Form

Redirect URIs*

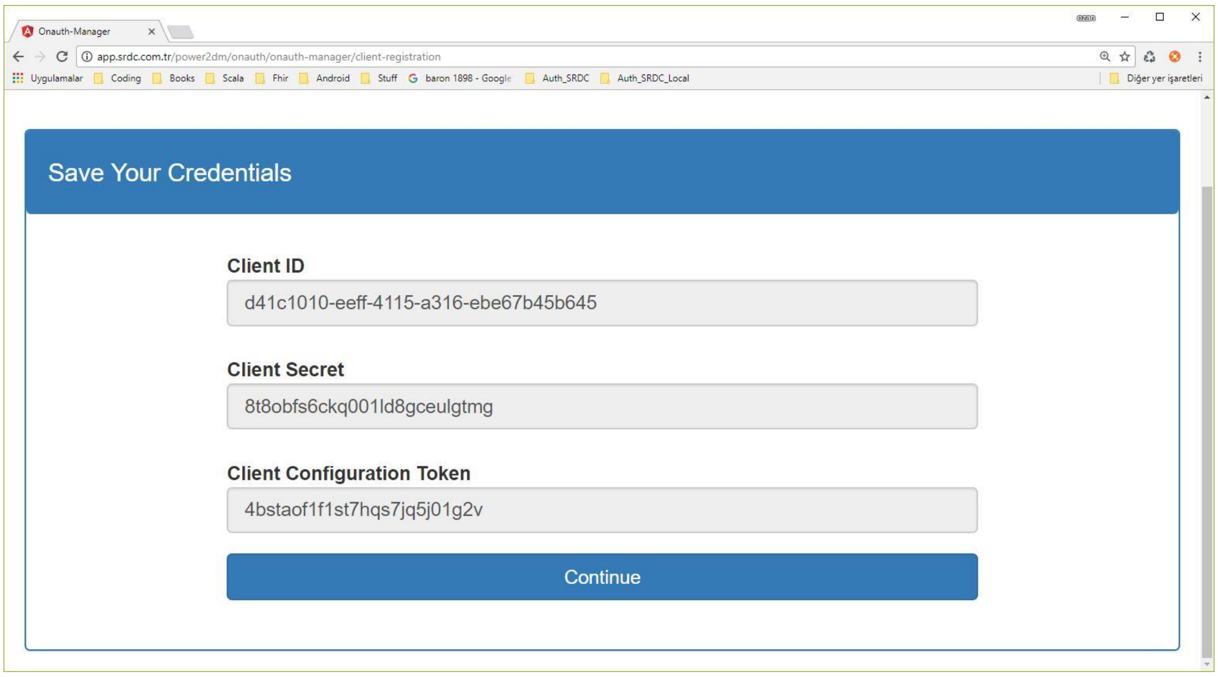
Enter a Redirect URI

Grant Types*

- [ ] authorization_code
- [ ] client_credentials
- [ ] refresh_token

Figure 14 Registration form for clients

Upon successful registration, users are prompted with new screen that shows the credentials of the newly registered client;



Save Your Credentials

Client ID
d41c1010-eeff-4115-a316-ebe67b45b645

Client Secret
8t8obfs6ckq001ld8gceulgtmg

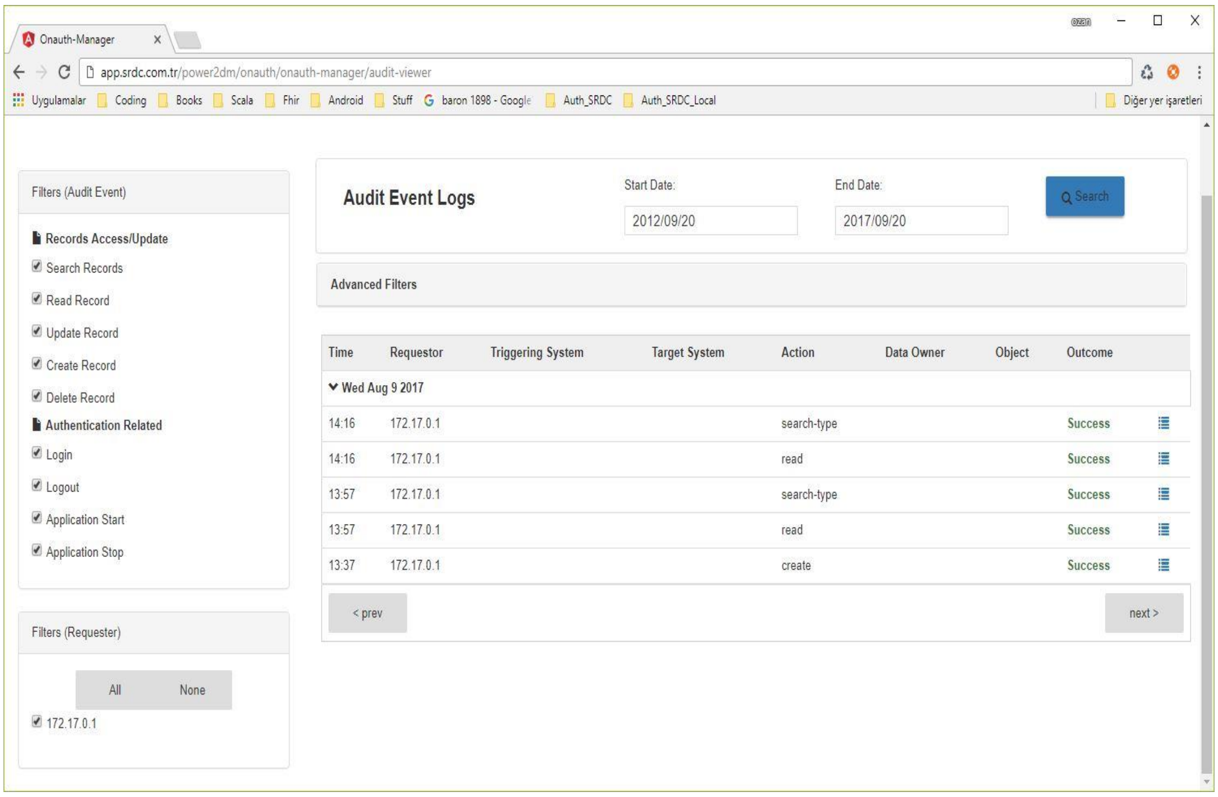
Client Configuration Token
4bstaof1f1st7hqs7jq5j01g2v

Continue

Figure 15 Successful client registration

4.2.5 Audit Viewer

Patients can see audit records about them from the audit viewer as shown in Figure 16. Audit viewer lists audit records on a table and patients can view details of any record by selecting it from table. Records on the table could be filtered by using the filters defined on the left-hand side of the screen.



Filters (Audit Event)

Records Access/Update
- ☑ Search Records
- ☑ Read Record
- ☑ Update Record
- ☑ Create Record
- ☑ Delete Record

Authentication Related
- ☑ Login
- ☑ Logout
- ☑ Application Start
- ☑ Application Stop

Filters (Requester)

All None

- ☑ 172.17.0.1

Audit Event Logs

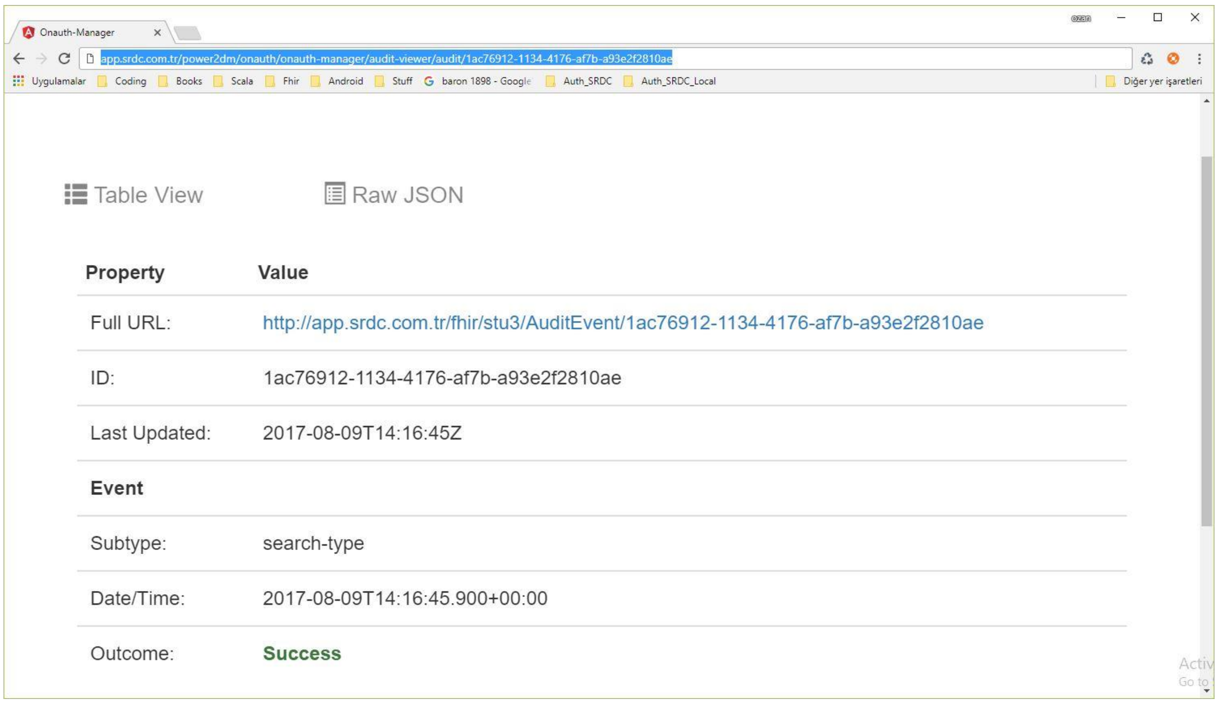
Start Date: 2012/09/20
End Date: 2017/09/20
Search

Advanced Filters

Time	Requestor	Triggering System	Target System	Action	Data Owner	Object	Outcome
Wed Aug 9 2017							
14:16	172.17.0.1			search-type			Success
14:16	172.17.0.1			read			Success
13:57	172.17.0.1			search-type			Success
13:57	172.17.0.1			read			Success
13:37	172.17.0.1			create			Success

< prev next >

Figure 16 Audit view for patients



Property	Value
Full URL:	http://app.srdc.com.tr/fhir/stu3/AuditEvent/1ac76912-1134-4176-af7b-a93e2f2810ae
ID:	1ac76912-1134-4176-af7b-a93e2f2810ae
Last Updated:	2017-08-09T14:16:45Z
Event	
Subtype:	search-type
Date/Time:	2017-08-09T14:16:45.900+00:00
Outcome:	**Success**

Figure 17 Patient audit detail

5 Appendices

5.1 Appendix A – POWER2DM Privacy Policy

```
{
  "id": "power2dm_privacy_policy",
  "name": "Default SRDC onFHIR privacy policy",
  "description": "Simple privacy policy for testing purposes",
  "authorId": "power2dm_admin",
  "lastUpdateTime": "2017-04-04T20:30:30Z",
  "resourceSetId": "sample-resource-set-Id",
  "realm_id": "power2dm",
  "patient_access": "care_team",
  "isBase": true,
  "isActive": true,
  "rules": {
    "Patient": {
      "care-manager": {
        "read": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
    "Device": {
      "care-manager": {
        "read": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
   "Composition": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      }
    },
     "Condition": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      }
    },
     "Goal": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "ProcedureRequest": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "MedicationOrder": {
```

```
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "Appointment": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "write": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "Encounter": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      }
    },
     "Observation": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "Procedure": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "RiskAssessment": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "QuestionnaireResponse": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      }
    },
     "MedicationAdministration": {
      "care-manager": {
        "read": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      },
      "self-care-supporter": {
        "read": 1
      }
    },
     "Order": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      }
    },
     "OrderResponse": {
      "care-manager": {
        "write": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      }
    },
      "CommunicationRequest": {
      "care-manager": {
        "read": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      }
    },
     "Basic": {
      "care-manager": {
        "read": 1


      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "write": 1
      }
    },
     "careteam/self-care-supporter": {
      "care-manager": {
        "read": 1,
        "isSmartScope": false
      },
      "care-supporter": {
        "read": 1,
        "isSmartScope": false
      },
      "self-care-manager": {
        "write": 1,
        "isSmartScope": false
      },
      "self-care-supporter": {
        "read": 1,
        "isSmartScope": false
      }
    },
     "careteam/care-manager": {
      "care-manager": {
        "write": 1,
        "isSmartScope": false
      },
      "care-supporter": {
        "write": 1,
        "isSmartScope": false
      },
      "self-care-manager": {
        "read": 1,
        "isSmartScope": false
      },
      "self-care-supporter": {
        "read": 1,
        "isSmartScope": false
      }
```

```
    },
   "careteam/care-supporter": {
      "care-manager": {
        "write": 1,
        "isSmartScope": false
      },
      "care-supporter": {
        "write": 1,
        "isSmartScope": false
      },
      "self-care-manager": {
        "read": 1,
        "isSmartScope": false
      },
      "self-care-supporter": {
        "read": 1,
        "isSmartScope": false
      }
    },
     "Practitioner": {
      "care-manager": {
        "read": 1
      },
      "care-supporter": {
        "read": 1
      },
      "self-care-manager": {
        "read": 1
      },
      "self-care-supporter": {
        "read": 1
      },
      "group_admin": {
        "read": 1
      },
      "client_admin": {
        "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },
        "Organization": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },
        "Medication": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },
        "Parameters": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },
      "Conformance": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },

      "StructureDefinition": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },

      "SearchParameter": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },

      "ValueSet": {
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        },
        "self-care-supporter": {
          "read": 1
        },
        "group_admin": {
          "read": 1
        },
        "client_admin": {
          "read": 1
        },
        "realm_admin": {
          "write": 1
        }
      },
      "AuditEvent": {
        "realm_admin": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        }
      },
      "UserInfo": {
        "group_admin": {
          "read": 1
        },
        "care-manager": {
          "read": 1
        },
        "care-supporter": {
          "read": 1
        },
        "self-care-manager": {
          "read": 1
        }
      },
      "user_registration/patient": {
        "practitioner": {
          "isSmartScope": false
        },
        "group_admin": {
          "isSmartScope": false
        }
      },
      "list_users/practitioner": {
        "group_admin": {
          "isSmartScope": false
        },
        "practitioner": {
          "isSmartScope": false
        },
        "nurse": {
          "isSmartScope": false
        }
      },
      "list_users/nurse": {
        "group_admin": {
          "isSmartScope": false
        },
        "practitioner": {
          "isSmartScope": false
        },
        "nurse": {
          "isSmartScope": false
        }
      },
      "list_users/patient": {
        "group_admin": {
          "isSmartScope": false
        },
        "practitioner": {
          "isSmartScope": false
        },
        "nurse": {
          "isSmartScope": false
        },
        "patient": {
          "isSmartScope": false
        }
      },
      "user_registration/practitioner": {
        "group_admin": {
          "isSmartScope": false
        }
      },
      "user_registration/nurse": {
        "group_admin": {
          "isSmartScope": false
        }
      }
    }
  }
```