Privacy-Aware Bulk Data Transfer Demonstration

Mohammad Jafari,
Kathleen Connor, John M. Davis, Christopher Shawn

Version 2.1
July 10, 2019
Table of Contents

1 INTRODUCTION.................................................................................................................................1
2 BULK EXPORT..........................................................................................................................................2
3 REQUIREMENTS..........................................................................................................................................3
   3.1 Explicit Bulk Permissions....................................................................................................................3
   3.2 Explicit Bulk Permission for FHIR Resource Types..............................................................................3
   3.3 Filtering Based on Security-Labels .....................................................................................................3
4 PROPOSED ARCHITECTURE ....................................................................................................................4
   4.1 Components .......................................................................................................................................4
   4.2 Flow ....................................................................................................................................................5
5 WALK-THROUGH EXAMPLES................................................................................................................7
   5.1 Rejected Request ...............................................................................................................................7
   5.2 Permitted Request ..............................................................................................................................7
   5.3 Permitted Request with Filters ..........................................................................................................9
6 DISCUSSION...........................................................................................................................................10
   6.1 JSON Scopes ....................................................................................................................................10
      6.1.1 Wildcard and Negative Scopes ....................................................................................................10
      6.1.2 Example .....................................................................................................................................10
   6.2 Rejecting vs. Filtering .......................................................................................................................11
   6.3 Authorization Proxy ..........................................................................................................................12
   6.4 Alternative Flows ..............................................................................................................................13
      6.4.1 Structured Access Tokens ............................................................................................................13
      6.4.2 Dedicated Bulk Clients and Long-Lived Tokens ........................................................................15
   6.5 Bulk API as a Standalone Service ......................................................................................................15
   6.6 Content Filtering ..............................................................................................................................16
7 ACRONYMS ..........................................................................................................................................18
8 REFERENCES ..........................................................................................................................................19
List of Figures

Figure 1: Overview of a FHIR Bulk Export Flow .................................................................................. 2
Figure 2: High-Level Flow of the Privacy-Aware Bulk Export Demonstration and the Components Involved.... 4
Figure 3: Privacy-Preserving Bulk Export Sequence Diagram .................................................................... 6
Figure 4: Rejected bulk export request due to insufficient scopes ............................................................... 8
Figure 5: Permitted bulk export request ....................................................................................................... 8
Figure 6: Permitted bulk export request with additional filters ...................................................................... 9
Figure 7: An example of JSON scopes used in the demo including the use wildcards and denoted scopes ....... 11
Figure 8: Implementing Authorization Service in the form of a reverse proxy .............................................. 12
Figure 9: Alternative Flow Using Structure Access Tokens without Introspection ....................................... 14
Figure 10: Alternative Flow using Dedicated Bulk Client with Long-Lived Access Token ............................ 15
Figure 11: FHIR Bulk Data Access API as a Standalone Service ................................................................. 16
Figure 12: Adding Support for Content Filtering to the Architecture ......................................................... 17
1 INTRODUCTION

This report discusses the technical details and lessons learned from the Privacy-Aware Bulk Data Transfer demonstration presented at the Health Level 7 (HL7) Fast Healthcare Interoperability Resources (FHIR) Connectathon in May 2019 in Montreal, Canada.

The demonstration featured a FHIR server fulfilling Bulk Export requests while enforcing privacy requirements by considering the scopes granted to the requesting Client by an OAuth 2.0 Authorized Server (AS).

Bulk Export is part of the emerging FHIR Bulk Data Transfer (BDT) specification [1], a draft specification for an Application Programming Interface (API) that enables clients to specify a broad set of FHIR resources and download them in batch as a single (or a few) file(s). Exported data can be sent to another FHIR server, a proprietary EHR system, a clinical data warehouse system, or other types of storage and processing systems.

This demonstration showed the interactions of the following standards and specifications:

- HL7 FHIR [2],
- HL7 FHIR security labeling specifications which are based on HL7 Healthcare Privacy and Security Classification System (HCS) [3],
- HL7 FHIR/SMART draft specifications for Bulk Data Transfer [1], and
- OAuth 2.0 Authorization Framework [4].

A separate report from the Security Framework team provides a more comprehensive discussion of the privacy-related requirements for Bulk Data Transfer in FHIR [5].
2 BULK EXPORT

Figure 1 shows an overview of the FHIR Bulk Export. The high-level use-case for a Bulk Export is for a Client to request to download a set of FHIR resources at once from a FHIR server. The server prepares and packages the requested resources and enables the Client to download them as a single file or a small number of files.

The flow starts when a Client sends a Bulk Export request to a FHIR server, at a designated Bulk Export endpoint using an HyperText Transfer Protocol (HTTP) GET request. This request includes the following parameters:

- **outputFormat**: output format for the exported resources. This enables the Client to choose the most suitable format for the consuming application.
- **since**: a moment in time to exclude older resources from the export. Only resources created or updated after this moment in time will be exported.
- **type**: to specify only certain types of resources.
- **typeFilter**: a FHIR search query to narrow down the exported resources further.

Since fulfilling a bulk request could be time consuming due to the potential large number of resources, these requests are handled in an asynchronous fashion. The server immediately sends back an HTTP 202 response which means the Client’s request was accepted but the response is not ready.

This initial response includes a Uniform Resource Locator (URL) address at which the final results will be available when ready. The Client can subsequently poll the new URL until the results are ready. If the results are not ready yet, the server responds with an HTTP 202 response, optionally with a header providing progress information, indicating that preparing the results is still in progress. Once the results are ready, the server responds with an HTTP 200 response which includes the location of all the files containing the results as well as any errors. The client can, then, download these files.

![Figure 1: Overview of a FHIR Bulk Export Flow](image-url)
3 REQUIREMENTS

While an extensive discussion of the requirements for Bulk Data Transfer is provided in a separate report [5], this section discusses a subset of those requirements which was the focus of this demonstration.

3.1 Explicit Bulk Permissions

The BDT service shall only accept requests from Clients which are explicitly authorized to make bulk requests.

This requirement emphasizes the distinction between regular read permissions granted to clients and the broader bulk export permissions. In theory, an export operation can be reduced to a large number of read operations; however, bulk requests are a different type of transaction which give the client additional power and capabilities and involve a higher risk to privacy if abused. Therefore, bulk access has been distinguished in some privacy regulations (e.g., the guidelines Dutch Data Protection Authority [6]).

3.2 Explicit Bulk Permission for FHIR Resource Types

The BDT service shall only accept requests from Clients which are explicitly authorized to make bulk requests for the requested FHIR resource types.

This requirement ensures that Client’s bulk access is controlled at the level of resource type. For example, a Client which is granted access to export only Immunization records, cannot request exporting Observations.

It is important from a practical perspective that a Client can be granted bulk export access to more than one type of resource, or potentially all resource types, in a single authorization instance, in order to avoid re-authorization overhead in cases where the Client needs to initiate different bulk exports for different resource types.

3.3 Filtering Based on Security-Labels

The BDT service shall filter the results of the export based on security labels, such that the Client only receives the resource for which it has sufficient clearance.

This requirement ensures that the Client’s security labeling clearance is considered in the export results. For example, a Client which has been granted authorization only to see non-sensitive resources, should not be able see any sensitive resources in the bulk export results.
4 PROPOSED ARCHITECTURE

Figure 2 and 3 show the high-level flow and the sequence diagram for the privacy-aware bulk export demonstration.

4.1 Components

The following components are involved in this flow:

- **Client**: The client which requests a bulk export. The client owns credential which enables it to communicate with an OAuth 2.0 Authorization Server to request for a token.

- **Authorization Server**: The OAuth 2.0 Authorization Server. In this demo, only the token and the introspection endpoints of the AS is used. The AS is assumed to be configured with policies which determines whether or not a Client should be granted an access token and the extent to which the Client can be authorized which are encoded in scopes.

- **FHIR Server**: a FHIR server capable of supporting the Bulk Export API.

- **FHIR Authorization Service (Authorization Proxy)**: The authorization component for the FHIR server. This service takes care of validating and introspecting Clients’ access tokens and ensuring whether or not the scopes granted to a Client are sufficient to fulfill its request. Moreover, it is capable of invoking other privacy services such as the Privacy Preserving Service (PPS).

Figure 2: High-Level Flow of the Privacy-Aware Bulk Export Demonstration and the Components Involved
4.2 Flow

The flow for fulfilling a bulk export request is as the following:

- **Acquiring Access Token**: The Client presents its credentials and required scopes in a request for Access Token to the AS. The AS evaluates this request against the applicable policies and if authorized, issues an Access Token. Depending on policies, the scopes associated with the issued Access Token may be all, or a subset of what the Client requested.

- **Client Bulk Request**: The client sends the bulk export request accompanied by the access token to the FHIR server.

- **Access Token Validation**: The Client’s request is captured by the Authorization Proxy which extracts the token and runs an introspection with the AS in order to determine the associated scopes. The request is rejected if the Client does not have explicit bulk export scopes, or the requested resource types are not in the authorized resource types based on the scopes.

- **Relayed Bulk Request**: If the Client is deemed authorized, the Authorization Proxy relays the request to the FHIR Bulk Data Transfer API. The Authorization Proxy may choose to modify this request by adding additional filters to the request, based on the scopes granted to the Client, for example, to filter out resources with security labels for which the Client has not been authorized.

- **Request Fulfillment**: The rest of the flow proceeds based on the Bulk Export API specification until the results are ready for the Client to download. The Authorization Proxy relays the responses back to the Client intact.
Figure 3: Privacy-Preserving Bulk Export Sequence Diagram
5 WALK-THROUGH EXAMPLES

In this section we discuss three sample requests to demonstrate the flow for the cases where the Client request is rejected, permitted as-is, and permitted with additional filters. The details are shown in Figures 4, 5, and 6.

5.1 Rejected Request

In this case, the Client has an access token associate with the permission to bulk export Immunization resources with the Normal confidentiality label.

- The Client requests a bulk export of all resources created after 2019-04-23.
- The Authorization Proxy invokes token introspection and receives the scopes associated with the Client’s access token.
- Since the requested resource types are beyond the Client’s authorized resource types this request is rejected.

Figure 4 shows more details of the request and the granted scopes for this case.

5.2 Permitted Request

In this case, the Client has an access token associated with the permission to bulk export Immunization resources with any confidentiality label.

- The Client requests a bulk export of all Immunization resources created after 2019-04-23.
- The Authorization Proxy invokes token introspection and receives the scopes associated with the Client’s access token.
- Since the requested resource types are consistent with the Client’s authorization, the request is permitted.
- The Authorization Proxy relays the request to the FHIR Bulk Data Transfer API.

Figure 5 shows more details of the request and the granted scopes for this case.
Figure 4: Rejected bulk export request due to insufficient scopes

Figure 5: Permitted bulk export request
5.3 Permitted Request with Filters

In this case, the Client owns an access token associated with the permission to bulk export resources of any types except resources with Restricted confidentiality label.

- The Client requests a bulk export of all resources created after 2019-04-23.
- The Authorization Proxy invokes token introspection and receives the scopes associated with the Client’s access token.
- Since the requested resource types are consistent with the granted permissions to the Client, the request is permitted.
- Since there is a restriction on the confidentiality of resources the Client can export, before relaying the request to the FHIR Bulk Data Transfer API, the Authorization Proxy modifies the Client’s query to add a filter for excluding any resources with Restricted confidentiality.

Figure 6 shows more details of the request and the granted scopes for this case.

```
[  
  {    
    "resource_set_id": {      
      "patientId": "*",       
      "resourceType": "*",     
      "securityLabel": "*"     
    },    
    "scopes": [     
      "bulk-export"    
    ]  
  },  
  {    
    "resource_set_id": {      
      "patientId": "*",       
      "resourceType": "*",     
      "securityLabel": [       
        {          
          "system": "http://terminology.hl7.org/ValueSet/v3-ConfidentialityClassification",          
          "code": "R"        
        }      
      ]    
    },    
    "deny": true,    
    "scopes": [     
      "bulk-export"    
    ]  
  }  
]
```

Permitted
Add a filter to client’s query to redact Restricted resources:
**?__security:not=R**

GET https://fhir-server/$export
?since=2019-04-23
&typeFilter=!*3F_security%3Anot%3DR

**Client’s Scopes:**
Permission to bulk export every resource except resources labeled as Restricted.

GET https://fhir-server/$export?since=2019-04-23

**Client’s Request:**
Bulk export every resource created since 2019-04-23.

Figure 6: Permitted bulk export request with additional filters
6 DISCUSSION

The highlights of this demo and the important lessons learned are discussed in this section. Note that some of these findings are broader than the scope of this project or even BDT, and are applicable to FHIR in general and other application areas.

6.1 JSON Scopes

In OAuth 2.0, the extent to which authorization is granted and the applicable constraints are specified by the scopes associated with the access token. The OAuth 2.0 specifications define scopes as plain strings, but opaque strings often lack the expressiveness required for modeling authorization details for a FHIR application, so, some proposals such as SMART of FHIR have expanded scope strings with a grammar which allows more expressiveness [7].

This demo uses JavaScript Object Notation (JSON) [8] for encoding complex scopes as shown in an example in Figure 7. JSON is a widely used and understood format which is based on a very simple grammar and is natively supported by most modern programming languages. This means that JSON scopes will be easy to understand by developers and parsing/encoding them will be less error-prone compared to custom grammars which require developing custom parsers and encoders. Moreover, JSON is extensible which means it can support both simple and more complex scopes with granular details (such as values from a vocabulary), and therefore, fit the requirements for a wide-range of applications.

6.1.1 Wildcard and Negative Scopes

For further flexibility, the JSON structure used in this demo, additionally supports use of wildcards and negation.

Wildcards allow matching for any value for a given attribute. For example, the JSON structure used for bulk export would require the use of a wildcard for the patient identifier in order to allow exporting resources belonging to any patients.

Negation allows stating that a scope or pattern of scopes have been denied. This allows making exceptions when broader scopes are granted to a Client. For example, when a Client is granted the scope to export any resources, an exception can be added in the form of a denied scope to exclude any resources labeled with Restricted confidentiality.

Using denied scopes to implement exceptions is particularly useful in the case of Client Credentials flow in OAuth (also known as the two-legged OAuth) in which the Client has to specifically request a set of scopes at the time of requesting an access token. Negative scopes enables the AS in these cases to grant the Client’s request by simply adding some exceptions to it based on policies. This is easier and computationally less costly to determine, compared to computing a comprehensive set of scopes which the Client is permitted to exercise. Moreover, it allows expressing the scopes in a more concise manner. For example, if there are 1000 possible scopes, instead of granting 999 scopes, the server can simply state that the Client is granted everything except one denied scope.

6.1.2 Example

Figure 7 shows an example of JSON scopes used in the demo. The access token in this case is associated with an array composed of two scopes. The first scope generally grants bulk export access to any resource type with any security labels belonging to any patients. The second scopes is a negative scope which denies bulk export of any resources of any types labeled with the security label R, which denotes confidentiality level of Restricted.
6.2 Rejecting vs. Filtering

The main function of the Authorization Service is to ensure that a Client can only access the information for which it is authorized. This can be accomplished in two ways:

- Rejecting the Client’s request when it does not have sufficient permissions for what it requests,
- Accept the Client’s request but filter any information the Client is not permitted to access from the response.

While a binary decision to either permit or reject a Client’s request is the simplest way for the Authorization Service to operate, in more complex cases, especially when a large amount of data is involved in the response, this is not desirable from a privacy and usability perspectives.

For example, consider a Client authorized for bulk exporting all resource types from a FHIR server except any resource marked as Restricted. When this Client requests to bulk-export all Immunization resources from a FHIR server which may contain some restricted Immunizations, rejecting the request is not only impractical. If the Client is not fully aware of the details of its access rights (which are often based on various applicable policies), it has to potentially go through a lengthy back-and-forth with the server and refine the query until it can receive the information. This can be frustrating and potentially impossible without sufficient clues about what may have caused the denial.

Moreover, this behavior by the Authorization Service can also be privacy-revealing, since it somehow signals the existence of restricted Immunizations to the Client. Such a Client can play with the query and by narrowing down the query parameters (e.g., time period) can infer further information about the sensitive information on the server.
Alternatively, the server can alter the results of the Client’s query to filter out the portions of the response which the Client is not authorized to access, and ultimately permit access to a more limited view of the requested data.

This, however, can have its disadvantages as well since silently altering the results of a Client’s query can be confusing and may lead to issues which are hard to detect. If the Client is explicitly told it is does not have sufficient authorization, it may be able to handle the error and remedy the problem by acquiring sufficient authorization. But when the results are quietly changed without an explicit error, issues may go unnoticed.

Thus, there is an important design decision for every Authorization Service to determine in what cases insufficient authorization should lead to rejecting the request and in what cases the request must be accepted while the results are modified.

The general principle is to use filtering only for individual data items and cases of fine-grained permission mismatch, while rejection should be used when the Client does not have sufficient authorization for large portions of the response due to a coarse-grained permission mismatch.

For this demo, the following design decision was made about the Authorization Service:

- If the Client’s authorization is not sufficient for bulk export, or for bulk export of any requested resource types, the request is rejected.
- Any resources for which the Client does not have the suitable security labeling clearance, are filtered out and removed from the results.

6.3 Authorization Proxy

In an OAuth ecosystem, the AS is in charge of making authorization decisions by identifying the Client, checking applicable policies, and issuing an access token. The FHIR server, however, still needs to implement the logic of inspecting and introspecting the access token, determining whether the associated scopes are sufficient for fulfilling the Client’s request, and ultimately permitting or rejecting the request and invoking or enforcing any applicable filtering. This logic is consolidated in an Authorization Service.

The Authorization Service is often implemented internally by the FHIR server but this demo used a more flexible architecture where the Authorization Service logic is implemented as a separate service outside of the FHIR server itself, as shown in Figure 8.

![Figure 8: Implementing Authorization Service in the form of a reverse proxy](image-url)
The Authorization Service in this architecture is a reverse proxy, a web component that sits in front of an existing web service, monitors the incoming requests and outgoing responses, and is capable of inspecting, altering, and blocking the traffic. The FHIR server is only accessible through the proxy and is unreachable otherwise.

The Authorization Service in this case works as the following:
- Receives the request from the Client.
- Extracts and inspects the accompanying access token and rejects the request right away if the access token is missing or not valid.
- Invokes the token introspection flow with the AS in order to verify the validity of the access token and obtain the associated scopes.
- If necessary, modifies the Client’s incoming request based on the scopes associated with its access token. This is, for example, to narrow down the query by adding more filters based on the Client’s authorization details.
- Receives the response from the FHIR server.
- Inspects the response by ensuring everything in the outgoing response is within the boundaries of the Client’s authorization. If necessary the Authorization Service can reject the request or modify the response at this stage.
- Forwards the (potentially modified) response to the Client.

While this architecture adds an additional performance overhead, it has the benefit of isolating the Authorization Service logic into a separate service which leads to less coupling between the authorization logic and the FHIR server. This is important in having a more flexible and configurable Authorization Service with a separate development and configuration plan and it enables the organization to use the Authorization Service with different implementations of FHIR servers.

6.4 Alternative Flows

The standard OAuth 2.0 flow relies on a general Clients obtaining an access token, in the form of an opaque string from an AS and presenting this token to the Resource Server’s Authorization Service, while the Resource Server uses token introspection to determine the scopes associated with the access token. In some cases, and for some applications, this flow might be inefficient which calls for alternative flows to simplify this process. These alternative flows are not necessarily simpler in every respect, so, their use is only justified when it fits the needs of the application based on the trade-offs discussed below.

6.4.1 Structured Access Tokens

Using self-sufficient structured access tokens, in the form of JSON Web Tokens (JWT) can eliminate the need for token introspection from the flow. The Authorization Service in this case can simple parse and validate the access token to determine the scopes associated with the token without making a call back to the AS. JWTs are signed and it is also possible to use encryption if the details of the scopes need to be kept secret from the Client. Figure 9 depicts this flow.

Note that while this flow saves the additional network call for introspection and reduces the coupling between the Authorization Service and the AS, it has the following disadvantages:
- Since the Authorization Service has to validate signatures on access tokens, there is an additional configuration and computational overhead on this service. This is a trade-off for eliminating the introspection network call to the AS which may not be suitable if these overheads are more costly than network calls (e.g., in smaller devices).

- The size of access tokens can be much larger because they will have to incorporate a payload including the scopes, signatures, expiry dates, and other attributes. Larger access tokens lead to larger request headers which could affect the performance in some use-cases, especially with slower network links between the Client and the FHIR Server.

- Cutting the live connection between the AS and the FHIR Server’s Authorization Service can make it more difficult to revoke an existing access token since the AS will not be able to instantaneously declare the token invalid in response to introspection calls. This makes the flow unsuitable for use-cases where revocations of access tokens are frequent and crucial.

Figure 9: Alternative Flow Using Structure Access Tokens without Introspection
6.4.2 Dedicated Bulk Clients and Long-Lived Tokens

Using dedicated Clients with locked-down scopes and long-lived tokens can further simplify the flow for bulk access as shown in Figure 10.

If a Client’s scopes are locked down at the time of client registration with the AS, the Client will not need to specify the scopes at the time of requesting a new access token. This mechanism can lead to registering dedicated Bulk Clients which can simplify the interactions with the AS for obtaining a new access token. A dedicated Bulk Client is granted bulk scopes and is only used for requesting bulk access to the FHIR server.

OAuth 2.0 access tokens are usually short-lived, and the Client needs to either use a refresh token to request a new access token, or re-do the entire request for access token from the AS. This can be a performance overhead, especially in cases where the interactions between the Client and the FHIR server are frequent. Although long-lived clients increase the risk of compromised tokens, they may be worthwhile in certain use-case where the Client has confidence about the secrecy of the token.

6.5 Bulk API as a Standalone Service

The Bulk Data Transfer draft specification define the bulk API as part of the FHIR API implemented for a single FHIR server. This demo expanded this architecture by relying on an independent Bulk Data Transfer service which can potentially be backed by more than one FHIR server. Figure 11 shows this architecture.

The Bulk API service is a reverse proxy which is backed by potentially several FHIR servers. This service receives Bulk API queries such as a Bulk Export request, breaks it down into FHIR API queries, in this case, search queries, and eventually collects and aggregate the results.

While this model brings about more flexibility, less coupling, and supports running bulk export against multiple FHIR servers, its performance may be worse than a bulk implementation that prepares export files directly from low-level internal FHIR server’s data stores (e.g., database).
6.6 Content Filtering

Filtering resource contents is a requirement in a lot of use-cases involving de-identification of the exported data (e.g., anonymization or pseudonymization). This requirement was not directly supported in the demo, however, an experimental parameter for the Bulk Export API was demonstrated in which the request for export could specify a simple JSON path for attributes to be removed from the exported results. In order to preserve the integrity of the FHIR resources, the value of the attribute is replaced with a suitable null flavor which indicates the value has been masked for privacy reasons.

While this is a preliminary step towards implementing content filtering, a more comprehensive implementation of content filtering requires the following steps:

- A standard vocabulary of applicable filtering algorithms which can be referenced in policies, scopes, and in the parameters passed to the Bulk Export API.
- Referencing the standard filtering algorithms in policies as obligations which specify additional conditions applicable to access.
- Defining suitable attributes in scopes granted to the client so that standard content filters can be referenced as conditions of a grant. In other words, the scopes need to include the syntactic mechanism for specifying that a granted scope is conditioned on the application of the filters on the contents of the resources.
- Incorporating content filters into scopes granted to a Client’s access token based on policies and communicating these via the introspection endpoint to the Authorization Service.
- Defining a standard parameter in the Bulk Export API for specifying application of content filters on the results.
- Adding content filters based on scopes as additional parameters to the Client’s query before relaying it to the FHIR Bulk API.
- Implementation of the standard content filtering algorithms by the Bulk API Service.

Figure 12 depicts these steps vis-à-vis the proposed architecture of this demo.
Figure 12: Adding Support for Content Filtering to the Architecture
### 7 ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AS</td>
<td>Authorization Server</td>
</tr>
<tr>
<td>BDT</td>
<td>Bulk Data Transfer</td>
</tr>
<tr>
<td>EHR</td>
<td>Electronic Health Record</td>
</tr>
<tr>
<td>FHIR</td>
<td>Fast Healthcare Interoperability Resources</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level 7</td>
</tr>
<tr>
<td>HSPC</td>
<td>Healthcare Services Platform Consortium</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>SMART</td>
<td>Substitutable Medical Applications &amp; Reusable Technologies</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
</tbody>
</table>
8 REFERENCES


