Paper AD10

Integrating Isolated Clinical Trial Systems with RESTful APIs: Enhancing Data Accessibility and Collaboration

Sumesh Kalappurakal, J&J, Raritan, USA
Nicolas Dupuis, J&J, Allschwil, Switzerland

ABSTRACT

Isolated clinical trial systems are often designed as independent software applications, which make it difficult for the same team using different systems or different team try to access or share data. However, RESTful APIs can be used to facilitate the integration of these systems, allowing for seamless data access between applications. RESTful APIs offer a standardized way to communicate between systems, making it easier for developers to create integrations using the right language, even if the systems were not originally designed for integration. By implementing RESTful APIs, clinical trial systems can become more connected, enabling teams to access data from multiple sources and improve collaboration.

INTRODUCTION

Clinical trial systems play a pivotal role in this pursuit, serving as the backbone for organizing, managing, and analyzing data generated during the research process. However, an underlying challenge persists - the isolation of these systems and the subsequent impediments to effective data sharing and collaboration within and across teams using different programming languages.

The traditional approach to designing clinical trial systems as standalone software applications has resulted in data silos and disjointed workflows. When different teams attempt to collaborate using diverse systems or when a single team operates multiple isolated systems, the barriers to accessing and sharing data become evident. This issue results in manual and repetitive actions and impacts research and development timelines.

Addressing this challenge requires a paradigm shift, one that embraces technological solutions capable of bridging the gap between disparate clinical trial systems. This paper explores the potential of Representational State Transfer (REST) architectural style and its application through RESTful APIs (Application Programming Interfaces) to foster seamless data access and collaboration in clinical trial environments. RESTful APIs offer a standardized method for systems to communicate, enabling even non-integrated systems to interact effectively. By facilitating the integration of these systems, RESTful APIs hold the promise of transforming clinical trial environments into interconnected hubs of knowledge, propelling research forward through enhanced data accessibility and collaborative opportunities.

This paper delves into the rationale behind the utilization of RESTful APIs in clinical trial systems, elaborating on their mechanisms, benefits, technical implementations, and the challenges they mitigate. Through comprehensive exploration and analysis, we aim to shed light on the transformational potential of RESTful APIs in creating a cohesive ecosystem that accelerates clinical research and cultivates collaboration across multidisciplinary teams. As we delve into each section, a holistic understanding of the power of RESTful APIs in enhancing connectivity and collaboration within clinical trial systems will unfold.

METHOD

To integrate two isolated systems using RESTful APIs, we need to get information about the following components of REST APIs.

ENDPOINT URL

The API provides specific URLs (endpoints) to interact with different resources or functionalities. Each endpoint corresponds to a specific action, such as retrieving data or submitting information.

HTTP METHODS (VERBS)

REST APIs utilize HTTP methods to indicate the type of action being performed. Common HTTP methods include:

GET: Retrieve data from the server.
POST: Send data to the server for processing.
PUT: Update or replace existing data on the server.
DELETE: Remove data from the server.

AUTHENTICATION: Most APIs require authentication to ensure secure access. This involves providing credentials (such as API keys, tokens, or usernames/passwords) to identify and authorize the user or application making the API request.

HEADERS: API requests often include headers, which provide additional information about the request. Headers can include metadata, content type, authorization details, and more.

REQUEST PAYLOAD (BODY): When sending data to the API, the request payload (or body) carries the information needed for the requested action. This is typically used with HTTP methods like POST and PUT.

QUERY PARAMETERS: API requests can include query parameters in the URL to refine the request or filter the response. Query parameters are often used with the GET method to specify criteria for data retrieval.

RESPONSE: Upon sending a request to the API, the server responds with data in a specific format, commonly JSON (JavaScript Object Notation). The response contains the requested data, or an acknowledgment of the action performed.

STATUS CODES: HTTP status codes are included in the response to indicate the request outcome. Common status codes include 200 (OK), 201 (Created), 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), and more.

The information can be obtained by following the approach outlined below:

1. Firstly, one should refer to the API Documentation. It is worth mentioning that many contemporary frameworks automatically generate one or more types of documentation, such as Swagger, Redoc.
2. If API doc endpoint is disabled, individual endpoints can be retrieved from the browser's built-in developer tools. It is essential to keep the Developer Tool open and carry out actions in the application as usual. This will enable the availability of corresponding endpoints for each action.

USE CASE 1

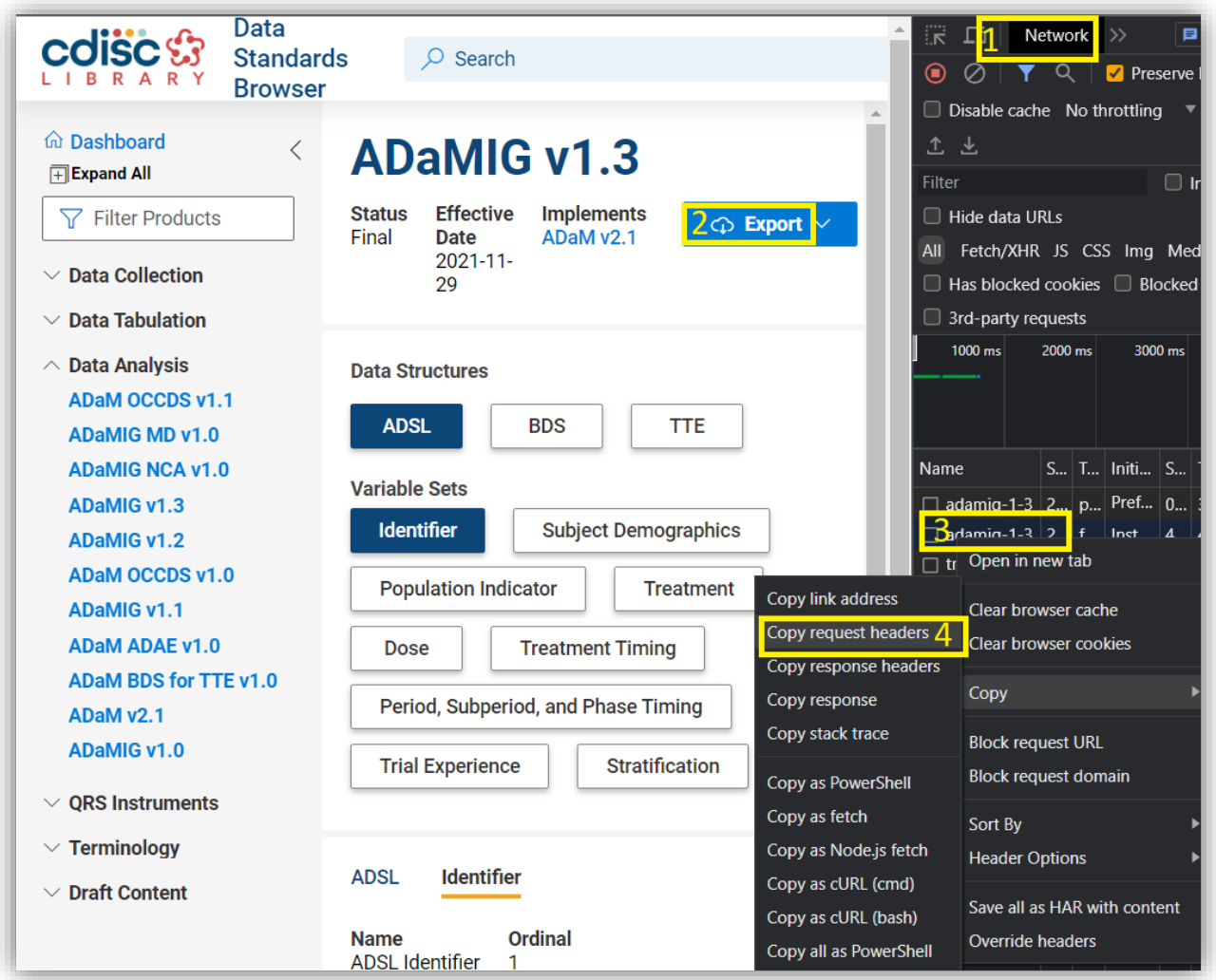
PROBLEM STATEMENT:

Our first use case is interaction between Statistical Computing Environment (SCE) and Metadata Repository (MDR). Clinical and Statistical Programming team are expected to work on metadata into MDR and generate datasets into SCE. To apply metadata into datasets both the systems do not provide any integration and programmers are expected to continuously export and import metadata from MDR to SCE.

SOLUTION:

Since MDR system is designed using modern web framework Programmers could find API end point used during download and call them in SCE directly. Once metadata is downloaded into file system remaining part could be done seamlessly and process stays the same here after. Let us break down solution with example of CDISC Library Metadata Browser. Even though CDISC Library is designed for data sharing across system but for being realistic we would use Download option available in Metadata Browser and try to mimic an isolated MDR system built using REST APIs.

- Navigate to CDISC Library Data Standards Browser: https://library.cdisc.org/browser/#/mdr/adam/adamig-1-3
- Open Developer tools from browser settings (try shortcut F12)
- Navigate to Network tab.
- Export metadata as Microsoft® Excel® (XLSX) while keeping Network tab open.
- Notice new entry Named as adamig-1-3.
- Right click on name adamig-1-3 and Copy → Copy request headers.
- Paste in a text editor like Notepad.
- You would be able to get all the details of REST API




```
GET /ui/api/mdr/adam/adamig-1-3 HTTP/1.1
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
Host: cdisc-library-apim.azure-api.net
Origin: https://library.cdisc.org
Referer: https://library.cdisc.org/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/116.0.0.0 Safari/537.36
accept: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
authorization: Bearer QifQ.MjczfQ.MZ--VTIambgpTO6tSJ0lq16jtbT-t4ZLXCA temporary.token
request-context: appId=cid-v1:e42f64bb-cef0-4634-9df0-b0df8509d035
request-id: |d09c658ab34743c1a1f41e2ea947532c.9ea678c805644d6c
sec-ch-ua: "Chromium";v="116", "Not)A;Brand";v="24", "Google Chrome";v="116"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
```

ENDPOINT URL

https://cdisc-library-apim.azure-api.net/ui/api/mdr/adam/adamig-1-3

HTTP METHODS (VERBS)

GET

AUTHENTICATION:

N/A, here as authentication is pass as temporary token

HEADERS:

- `accept: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`
  In this example above header is used to differentiate which type of file needs to be downloaded using the same endpoint url. It is also possible to have it part of query parameter then separate header might not be required in other case.

- `authorization: Bearer QifQ.MjczfQ.MZ--VTIambgpTO6tSJ0lq16jtbT-t4ZLXCA temporary.token`
  A permanent token or user id, password is recommended to avoid recreating and that is the only piece you might have to request in your isolated system. Or you might implement RPA to get the updated token into SCE before expiry time and does not require any change in MDR.

REQUEST PAYLOAD (BODY):

N/A for Get Request, usually when we need to edit or add any information then PAYLAOD is used.

QUERY PARAMETERS:

N/A for this example but it could be used to get metadata of specific study and dataset in MDR.

`https://mdr.companey.net/ui/api/adam/?study=xyz&data=adsl`

RESPONSE:

Downloaded Excel file.

STATUS CODES:

200, This information can be checked by copying the Response header of the Export operation described above. This could be particularly useful for error handling programmatically.

If we need to make a request and get a response programmatically and repeatedly then we could use programming languages used in SCE like SAS, R, Python etc., here SAS example is shared.

```
/*************************************************************************
Location to store output. If file needs to be stored at permanent location, then
A permanent library path could be used.
*************************************************************************/
%let workpth = %sysfunc(getoption(work));
filename out "&workpth./adamig13.xlsx";

/*************************************************************************
API call by using request details in proc http
*************************************************************************/
proc http out=out method="GET"
    url="https://cdisc-library-apim.azure-api.net/ui/api/mdr/adam/adamig-1-3";
  headers "accept"="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet";
  headers "authorization"="Bearer QifQ.MjczfQ.MZ--VTIambgpTO6tSJ0lq16jtbT-t4ZLXCA
temporary.token ";
run;


/*************************************************************************
Usual proc import of proc http response as Excel file and output as dataset
*************************************************************************/
proc import datafile=out out=adamig13  dbms=xlsx replace;
  sheet="Variables";
  getnames=yes;
run;
filename out clear;
```

Response output could be utilized further for dataset generation without any need for manual export and import. Once MDR is updated, subsequent run in SCE would get updated metadata in same fashion.

Total rows: 336 Total columns: 12

	VERSION	DATA_STRUCTURE_NAME	VARIABLE_GROUP	VARIABLE_...	VARIABLE_LABEL	TYPE
1	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	STUDYID	Study Identifier	Char
2	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	USUBJID	Unique Subject Identifier	Char
3	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	SUBJID	Subject Identifier for the Study	Char
4	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	SITEID	Study Site Identifier	Char
5	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	SITEGRy	Pooled Site Group y	Char
6	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	SITEGRyN	Pooled Site Group y (N)	Num
7	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	REGIONy	Geographic Region y	Char
8	ADaMIG v1.3	Subject-Level Analysis Dataset	Identifier	REGIONyN	Geographic Region y (N)	Num
9	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	AGE	Age	Num
10	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	AGEU	Age Units	Char
11	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	AGEGRy	Pooled Age Group y	Char
12	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	AGEGRyN	Pooled Age Group y (N)	Num
13	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	AAGE	Analysis Age	Num
14	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	SEX	Sex	Char
15	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	RACE	Race	Char
16	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	RACEGRy	Pooled Race Group y	Char
17	ADaMIG v1.3	Subject-Level Analysis Dataset	Subject Demographics	RACEGRyN	Pooled Race Group y (N)	Num
18	ADaMIG v1.3	Subject-Level Analysis Dataset	Population Indicator	FASFL	Full Analysis Set Population Flag	Char
19	ADaMIG v1.3	Subject-Level Analysis Dataset	Population Indicator	SAFFL	Safety Population Flag	Char
20	ADaMIG v1.3	Subject-Level Analysis Dataset	Population Indicator	ITTFL	Intent-To-Treat Population Flag	Char
21	ADaMIG v1.3	Subject-Level Analysis Dataset	Population Indicator	PPROTFL	Per-Protocol Population Flag	Char

USE CASE 2

PROBLEM STATEMENT:

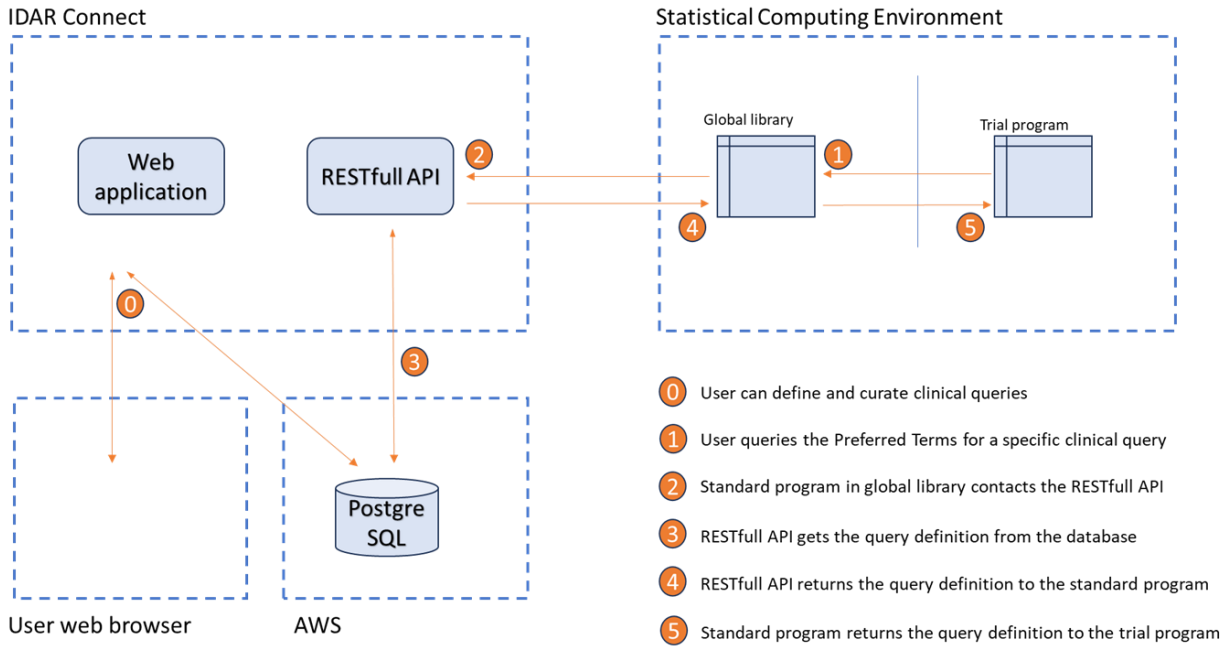
Our second use case is the interaction between our SCE and our web application deployment platform (IDAR Connect). IDAR Connect is where we deploy R and Python web applications. Our SCE users have a need for an application to manage clinical queries, such as Standard or Custom MedDRA Queries. An architecture with a RESTful API in between proves to be helpful.

SOLUTION:

A web application, deployed from IDAR Connect, is where users can define and curate clinical queries, using various workflows. The data is ultimately stored in a PostgreSQL database. In parallel, a RESTful API can query the database and return specific information. Finally, SCE users can query the database through the RESTful API. It's important to note that these 3 components (the web application, the RESTful API and the end-user program) can be written in different languages and not necessarily the same for each. In our case, the web app and the RESTful API were written in Python (Dash and Flask) and the SCE programs were tested with SAS, R or Python.

This solution was deemed appropriate for several reasons:

1) Allows full standardization and transparency.
2) A program in our SCE can pull external information (e.g. from IDAR Connect) but an external program cannot push information to our SCE, due to security reasons.
3) We can leverage the capabilities of each language as well as the skills of the developers/programmers.



The deployed API is easily created with an high-level package, Flask. Deployment is also made very easily using our RSConnect platform. Finally, making request to the API is also trivial, using for example the request package in Python.

CONCLUSION

The implementation of RESTful APIs has emerged as a transformative solution to address the challenges posed by isolated clinical trial systems. This paper has explored the critical role that RESTful APIs play in fostering connectivity, enabling data access, and promoting collaboration within the realm of medical research. Through a comprehensive analysis of design considerations, security measures, compatibility strategies, and real-world case studies, the potential of RESTful APIs to revolutionize clinical trial systems has been underscored.

By bridging the gap between disparate systems, RESTful APIs have the power to dissolve data silos and facilitate seamless data exchange, thereby accelerating timeline and increased productivity. Through standardized communication channels, RESTful APIs provide a pathway for programmers, statisticians, medical writers, and clinicians to access valuable insights from diverse sources, fostering interdisciplinary collaboration and driving the advancement of medical knowledge.

As clinical trial systems evolve towards greater integration through RESTful APIs, it is essential to acknowledge potential challenges and engage in ongoing efforts to ensure data security, privacy, and ethical considerations. Stakeholders must remain vigilant in upholding regulatory standards while harnessing the benefits of enhanced connectivity.

In conclusion, the integration of RESTful APIs within clinical trial systems marks a paradigm shift in clinical research and development. By breaking down barriers to data sharing, streamlining workflows, and fostering collaboration, RESTful APIs hold the promise of transforming the landscape of healthcare innovation. As the journey towards interconnected clinical trial environments continues, embracing RESTful APIs becomes a pivotal step in propelling clinical research to new heights and shaping a future characterized by increased efficiency, accelerated discoveries, and improved patient outcomes.

The seamless flow of data, insights, and ideas facilitated by RESTful APIs embodies a collective commitment to advancing science and medicine, driving humanity forward on the path to improved health and wellbeing.

REFERENCES

CDISC Library (Open); Data Standards Browser (Login Required)

ACKNOWLEDGMENTS

Sairam Gorthi
Karma Tarap
Pavan Prakash

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Lucky Singh
J&J
Bengaluru, India
LSingh4@ITS.JNJ.com

Johnson&Johnson
Innovative Medicine