CERN-Solid Code Investigation

Proof of Concept and Prospects

by

Jan Schill
schi@itu.dk

Supervisors:

Philippe Bonnet (ITU)
phbo@itu.dk

Maria Dimou (CERN)
maria.dimou@cern.ch

A thesis presented for the degree of
Master of Science

IT UNIVERSITY OF COPENHAGEN

Computer Science
IT University of Copenhagen
Denmark
01.06.2021

Table of Contents

Introduction

1 Context

The Web was created in 1989 by Sir Tim Berners-Lee while working at the institution of European Organization for Nuclear Research (CERN) "[...] to allow people to work together by combining their knowledge in a web of hypertext documents" [1]. This brilliant idea has ever since grown as an essential part of our all lives [2]. While bringing a new platform for innovation into existence, a new level of oppression and surveillance that has never been seen before has facilitated without most even realizing. Data are harvested and analyzed to generate models describing and predicting human behavior far beyond what is morally tolerable. These actions permit the generation of large amounts of wealth as capitalistic and governmental bodies enjoy a great interest in these models to either generate more profit or more control. The escalation of attraction towards data has led to the construction of so called *data silos*, where data gatherers built attractive applications for users while locking them into their walled gardens to have the all data for themselves and then selling it.

Through whistle blowing, active journalism, and technical education a new wave of hope is arising in the ocean of data hunting. Not giving up on the Web, which has in its little over 30 years of lifetime proven its paramount act in state, society, and the economy, Sir Tim has an adjustment for the Web specified. With the help of several Web enthusiasts, companies, and even governments the idea of Solid was found.

Solid aims at giving the users back the control over their data to regain full data sovereignty. It does so by specifying through a number of technical reports a new way of building application in the Web that let the users control their data and in this way can decide who can see what. Just like the Web, which was also defined by technical reports and prototypes, it is for everyone and not limited by anything.

"Being the Web's birthplace, CERN remains a High Energy Physics laboratory; hence, its primary mission is to run an accelerator, its detectors, and the relevant experiments. Computing is of paramount importance for filtering, storing, distributing, accessing, analyzing the experimental data. Nevertheless, due to its large and distributed user base, CERN offers sophisticated solutions on all software application fronts. In terms of price and transparency, proprietary packages have been disappointing. Following the raising worldwide awareness of personal data ownership and sovereignty, CERN is interested in Solid." [2]

2 Goal

The work presented in this paper will conclude the *CERN-Solid Code Investigation* project [3]. Where the previous work done in [2] focused on analyzing the Solid ecosystem, this body of work will focus on the remaining points defined in the project description [3]. The remaining points include the development of two modules integrating Solid principles programmatically into existing software from CERN, these will be presented with their design choices and the challenges met. Further, these developments will be evaluated and analyzed based on the quality attributes security, performance, and usability. Followed by a section identifying the challenges with Solid in its current state and how traditional software, namely one from CERN called Indico, has issues adopting a *Solid-way* of building software. The last section will introduce possible proceedings for CERN as the goal of this project is to determine how CERN can be involved in the Solid ecosystem and the *fate of the Web*.

Related Work

This chapter will hold the parts of the project coming from the outside, all the relevant efforts happening in the Solid Community or CERN's involvement in the collaboration.

3 Background

The *CERN-Solid Code Investigation* project aims at linking CERN with Solid. Some of this linkage has been done in the form of a status quo check of the Solid ecosystem in [2]. This previous work and the efforts around the two proofs of concept (POCs) shall give insights into a symbiosis of CERN and Solid.

The following sections in this chapter will introduce the two systems and their significant subsystems, modules, libraries, or techniques.

4 Indico

Indico is one of CERN's most sophisticated software projects. It is an event management tool, giving users a means to organize complex meetings or conferences with an easy-to-use interface. It was started in 2002 as a *European project* and has been in production at CERN ever since. It is used daily to facilitate more than 600,000 events at CERN. It has helped others like the UN "to put in place an efficient registration and accreditation workflow that greatly reduced waiting times for everyone" at conferences with more than 180,000 participants in total [2].

4.1 Events

Being an event management tool at Indico's core lies the `Event` class. The Event's user interface is presented in 1, showing all sorts of attachable information.

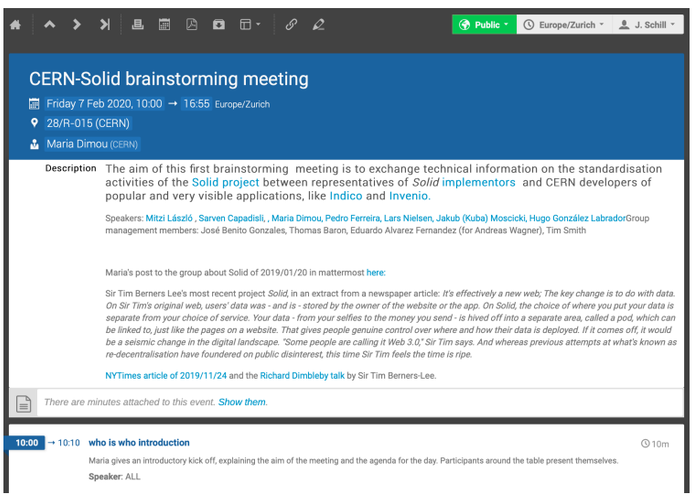


Fig. 1: User interface of an Indico event.

4.2 Storage Mechanisms

All events, all information in these events, file uploads, everything put into Indico are stored in a relational database on centralized servers. CERN's Indico instance is hosted on-premise in their own data centers. Another crucial feature of Indico is the permanent archival of event material and metadata [4]. It allows the lifetime access to all events hosted on the platform. As an example one can easily browse to the event "Big Data and Social Media" held by Vint Cerf in 2018 [5] and have access to description, recording, and slides, or even a presentation given in 2004 about "Practical Use of XML" [6].

4.2.1 EventSettingsProxy

A *proxy class* enables storage for event-specific settings. Commonly stored data types are contact email addresses for an event. In Indico these `EventSettingsProxy` are stored in a database table called `settings`. Adding a new setting for the POC it will receive a field in the row specific to the event.

4.3 Conferences

In Indico, an event can be simple meetings, lectures, or all kinds of presentations. Additionally, Indico allows the creation and management of conferences. Conferences are more complex events with several more features, including registration, call-for-abstracts, program definition, payments, and more [7]. The conference registration is especially interesting for this project as it is part of the POC.

4.3.1 Conference Registration

Once a conference is created, the conference manager needs to make a registration form to allow signups to the event. This form is created in Indico's backend and has default personal information fields but can be expanded as much as needed with free text fields. Payment for participation also needs to be enabled in this step. Meaning, as soon as a user fills in all the mandatory information and submits it, a successfully finished payment prompt will only complete the registration.

5 Solid

A initial introduction to Solid was given as part of a previous *research project* [2] in preparation for this thesis. In the *research project*, the Solid specification was, among other things, summarized and analyzed. This section will reiterate and study the subsequent experiment-relevant parts further.

5.1 Authentication With Solid

In the Solid ecosystem, agents identify themselves with their WebID and prove their ownership through the Solid OpenID Connect (Solid OIDC) protocol, which is a flavor of OpenID Connect (OIDC). For further explanation and flow diagrams through the authentication process, see either the work done in the previous report [2] or the Solid OIDC specification itself [8].

To help with the complex authentication flow of Solid OIDC a few libraries have been developed by different actors. Two libraries relevant for this project exist; their relevancy is discussed in the chapter 3 under the section 6.1.3.

Name	Solid Auth Fetcher	solid-client-authn
Repository URL	`github.com/solid/solid-auth-fetcher`	`github.com/inrupt/solid-client-authn-js`
Language	TypeScript (TS)	TS
Maintainer	Solid Community	Inrupt
Last updated	2021/03/05	2021/05/12

Table 1: Two Solid authentication libraries.

Solid Auth Fetcher is a fork of the `solid-client-authn` developed by Inrupt but has not seen much development in recent time. With the quickly evolving Solid ecosystem, it is vital to have working and up-to-date libraries to build applications in this ecosystem. Both libraries do not differ in their core functionality, and both support authenticating with the latest Solid servers, and thus the choice is not as important. Still, the frequency of commits in Inrupt's repository seems to indicate a more active development and a more reliable source when problems arise. This way, one does not rely on support from open-source developers, which is, per se, not a bad thing.

For this simple reason, the programming of the POC where Solid authentication was needed was enabled through Inrupt's `solid-client-authn`.

5.2 Reading and Writing Linked Data

Data in Solid is stored as Linked Data [9]. Resource Description Framework (RDF) is a framework for representing information of Linked Data in the Web [10]. The default file format so far implemented by the existing Solid servers is *Turtle* [11].

The graph-based data model from using RDF also requires additional computation as it is not natively supported with helper functions in JavaScript (JS), such as JavaScript Object Notation (JSON). The benefit of using JSON in JS is that a JSON object is automatically parsed as an object and can be operated on by using the dot notation to access attributes of the JSON data structure. For RDF, this is not the case for an into the program loaded Turtle resource.

Fortunately, existing libraries come to the rescue allowing such operations on the RDF-based data type. Again Inrupt and the Solid Community offer solutions. Again, Inrupt's solution was chosen for this development as it acts as a convenient wrapper to the bare bone Linked Data application programming interface (API) implemented in `rdflib.js` [12]. Even though working with RDF can be quickly done with just using `rdflib.js`, Inrupt's libraries tie nicely together and allow for instance a seamless passing of an authenticated session to its client library to enable authenticated requests to protected resources.

As mentioned before and looked at in detail in [2], the Turtle format is a graph data structure built-up with triplets—a triplet statement in its simplest form a sequence of (subject, predicate, object) terms [11].

Inrupt decided to call this construct a `Thing`, which is a data entity associated with a set of data or properties of this `Thing` [13]. A `SolidDataset` is a set of things [14].

The following code listing shows how the helper methods can be used to extract data from a Turtle file loaded by a request to the WebID profile document Uniform Resource Identifier (URI).

Listing 2.1: Basic usage of Inrupt's solid-client library.

```
1 // Import statements omitted for this demonstration
2 // 1
3 const myDataset = await getSolidDataset(
4   "https://janschill.net/profile/card"
5 );
6 // 2
```

```
const myProfile = getThing(
  myDataset,
  "https://janschill.net/profile/card#me"
);
// 3
const fn = getStringNoLocale(myProfile, VCARD.fn);
// fn => "Jan Schill"
```

1. **Fetching the Turtle resource at the given URI**: Notice here the WebID profile document URI is being loaded, which refers to the document describing the agent behind the WebID URI.
2. **Loading triplet statement into variable**: Now, the actual triplet statement containing information behind this specific agent is mapped to the `myProfile` variable.
3. **Reading a value from a triplet**: Every subject and predicate is a URI. The subject here is the loaded profile from the WebID URI and the to be extracted value from the statement matching on the subject and predicate. The predicate `VCARD.fn` when evaluated returns a URI, which on dereferencing describes what kind of value can be obtained from the object.

5.3 Authorization Through WAC

The as per Solid specification decided authorization mechanism is through Web Access Control (WAC) [15]. The majority of Solid servers use access control lists (ACLs) to manage the access modes on containers and resources. An ACL contains all the users and their permitted access rights to the resources on the pod. It may contain a *default*, or *public*, entry but must contain a reference to who the owner is to what the ACL is describing. ACLs are convenient as they allow easy and fast access to who has access to a specific resource. In contrast *capability tickets* are the preferred technique when wanting to find out what a specific user has access to [16].

5.4 Application Launcher

A so-called *application launcher* can mitigate a later presented discovery of a flaw in assigning unnecessary full access controls. The inconvenience lies in the fact when an application asks the client for what the allowed scope to its data pod is, most often, complete access control on the root container is needed – this is against the security principle or tactic called *least privilege*. The least privilege means to give actors of a system only exactly as much access they need and never more. The reason why this finds its way into the POCs will be presented in the chapter 3.

For now, an implementation shall be introduced showing how the application launcher can avoid this problem. When an application wants to create ACLs on a data pod, it needs the before-mentioned *control* access. This is the highest access mode in WAC with it files can be read, written, and ACLs can be modified – meaning the owner of files can be changed as an example. In the initial Solid OIDC flow, in one step the access scope for the application will be asked for. The asked for scope is for the root container of the data pod – a more fine-grained control to limit access on a specific child container does not exist. An application launcher is a Solid app that has full control of the pod and must be trusted. When a new application wants to make use of a data pod and needs control access but could, in theory, be limited to a specific container, the application launcher will create the appropriate ACL files to only allow precisely this. This works because all access modes are defined in ACL files, which can be dynamically created. The application will, with this launcher, now have complete control of the container it operates and stores data in but cannot reach outside of its container.

The source code for a standalone application launcher in JS exists here [17].

Investigation

6 Proof of Concepts

One main part of the investigation into the CERN-Solid collaboration is the development of a POC. The POC contains the creation of two independent software modules in an existing system from CERN. These software modules should show how it is to develop with the Solid principles in mind and to the Solid standard.

The goal of these modules is the symbiosis of decentralized stored data in a highly functional system without comprising its performance, security, or usability.

6.1 POC 1: Commenting Module for Events in Indico

The first POC is supposed to enrich the Indico system with some sort of Solid-based content. With the product owner and chief developer of Indico, the CERN-Solid project manager, and a Solid developer it was decided a commenting module for Indico events is an adequate solution to include data from an external storage entity namely a data pod. The ability to allow users of Indico to leave a comment on an event, which then lives in a data pod completely controlled by the author of the comment was concluded to be an attractive feature for Indico.

6.1.1 Architectural Analysis and Synthesis

In this section, the architectural analysis will be looked into. The scope of the system and its attributes will be defined based on the system description and the quality attributes (QAs) from the analysis of stakeholder needs.

System Description

The system aims at enabling commenting in web applications with decentralized storage on data pods. The system in the context of Indico intends to allow Indico users with a data pod to comment on specific Indico events. It thus needs to enable authentication with a Solid identity provider (IDP) to obtain and manage an authenticated session with a data pod. The module needs to provide an input field to be able to receive user input which can then be transformed to structured data in form of Linked Data and send to the data pod. On an invalid session or wrongly put input the module should tell the user in a appropriate manner. Comments need to be loaded from different data pods and be rendered into the Document Object Model (DOM). When a new comment is added the interface needs to be update accordingly.

Features

The core functionality of the module consists of the following three features:

1. A user with a Solid account can authenticate with their Solid IDP
2. A user can compose a text which is stored on their data pod
3. A user can see other users' comments

These features allow the development of a module giving users the ability to authenticate themselves using their WebID, storing the created data in form of comments in their own data pod, and also see decentralized stored data from other users.

Type of Users

There are two types of users in the system. The first group of users is the active authors of comments or observers of such. These will interact with the module by logging in to their data pod and compose comments. The other subset of users is the administrators of the application (Indico). This type of user is interested in keeping the tone of comments to the community guidelines and not allow any misuse.

Context Diagram

Other types of involved parties are the ones maintaining or developing the system and application. These are shown in the context diagram 2.

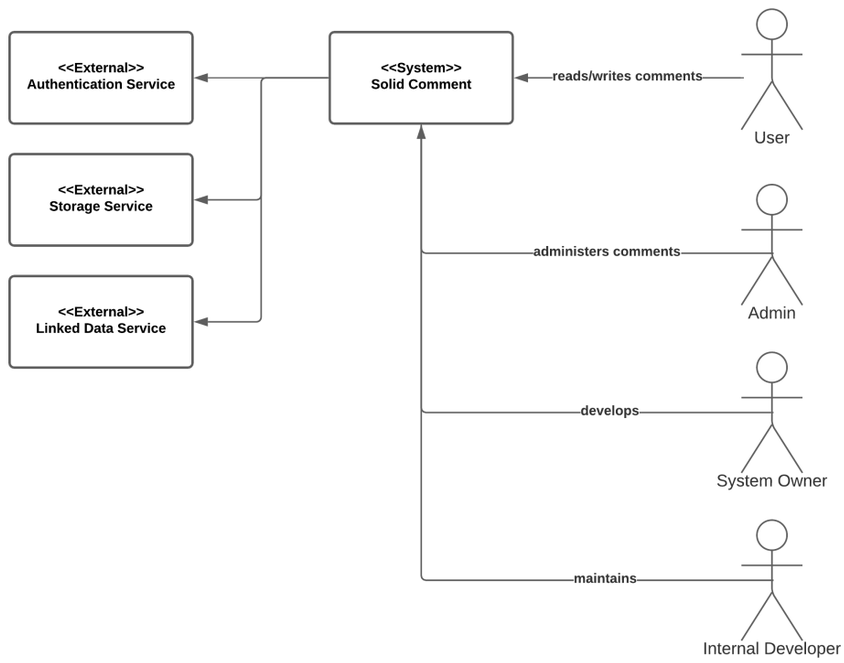


Fig. 2: Context diagram showing users and external services of the system.

Sequence Diagram

A sequence diagram brings a suitable overview for any software architecture, but especially useful for decentralized systems or those containing several separate services. It gives a clear understanding of each service's tasks and their relationship when passing messages around in the overall system.

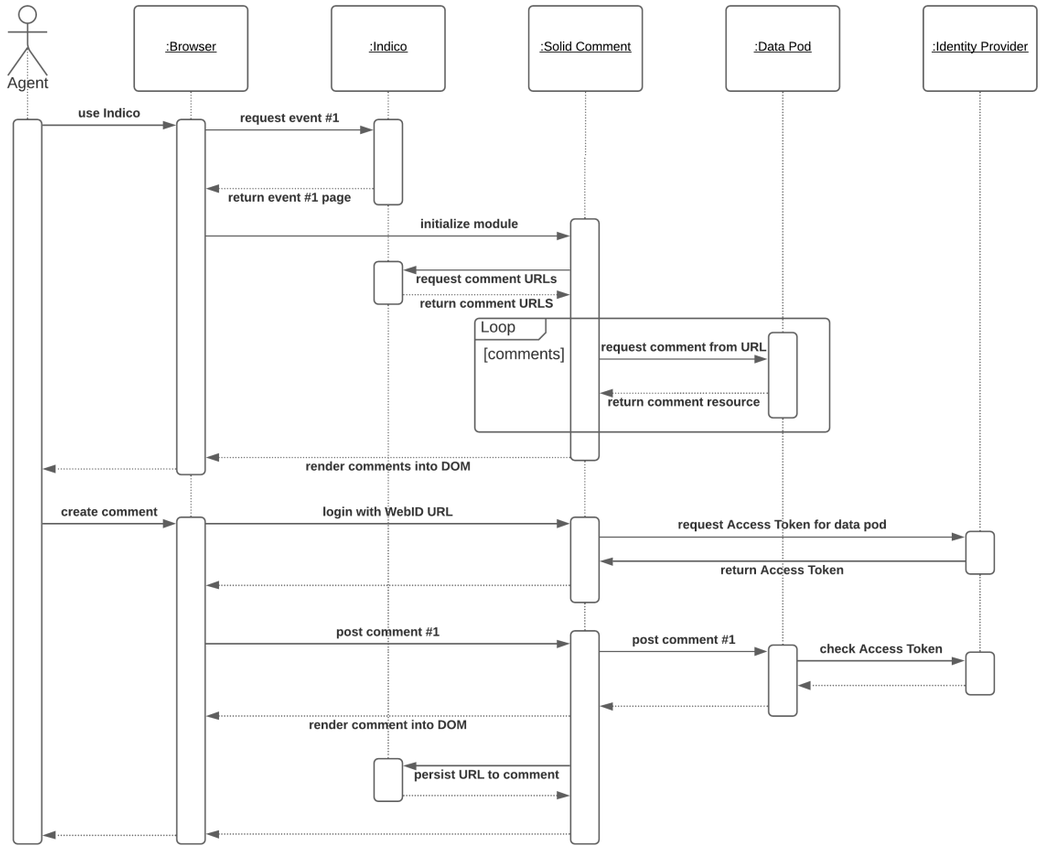


Fig. 3: Sequence diagram showing the sequential process through posting a comment.

Stakeholders

This section covers the various stakeholders in relation to the system. This both includes active users, but also various external and internal stakeholders who are impacted by the system or have an important say in the development process.

The **product owner** is responsible for the product, in this case, Indico. Usually, they are the ones planning repeated fixed time-boxes in which new features are developed or the existing software is maintained. They are constantly evaluating the state of the software and try to satisfy other stakeholders such as investors or the system users themselves. The main concerns of the product owner are to stay innovative while no compromising the quality of the existing software.

The **internal developer** is responsible for executing the decision made by, or together with, the product owner. The developer is also maintaining the software and is the one working with it on a daily basis and can, therefore, make assumptions about the evolution of the software. Their concerns lie in the maintainability of the software through its evolutionary cycle, as well as onboarding new developers.

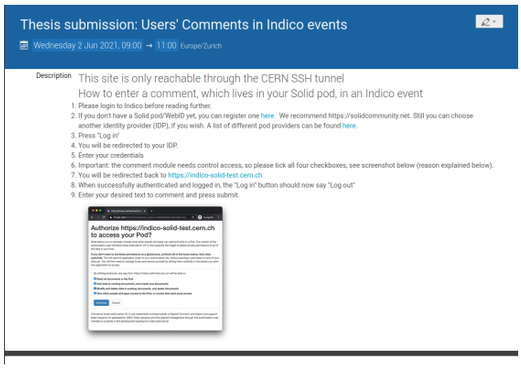
Drivers

The architectural drivers or QAs can be specified using "-illities". These QAs were defined with the stakeholders together in several meetings in which the existing QAs of the system, where the *comment* module would be embedded into, were drawn up. *Security* and *Performance* are the of utmost priority of the chief developer and product owner. For the collaboration manager having used Solid for a while it was of clear importance to her that the module is usable and requires as little effort as possible and causes the lowest amount of friction.

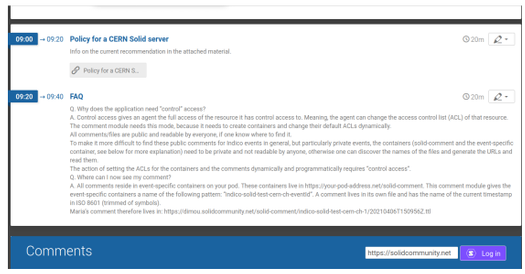
1. Security
2. Performance
3. Usability

6.1.2 User Interface

For various held meetings and presentations, a visual representation of the to-be-developed module helps to guide the audience through the process. For the comment module, a user interface was designed in graphical software to ease the explanation of the goals for the module. The existing Indico color scheme was adopted to blend into the system.



(a) User interface showing the comment module: Description.



(b) User interface showing the comment module: FAQ.

Fig. 4: User interface of Indico

6.1.3 Design

For the implementation of this module, several design decisions had to be made. From the fundamental choice of the module running on the client device or be computed on the server and then propagated to the client afterward or even with a microservice proxying all traffic to enable Solid without changing Indico. Other design challenges were around how to protect the resources holding the comment information. These resources reside on the external data pod and need to be fetched from the application and read by other agents. Can ACLs be configured to allow the specific use-case and other questions to come up and had to be considered?

Client- Versus Server-Side Versus Microservice

When an agent browses to a running instance of Indico most of the functionality is being prepared on the server hosting Indico. It retrieves the specific request, builds the Hypertext Markup Language (HTML), and sends it to the user. For Indico most of the functionality is built with Python and the web framework Flask. Sometimes functionality needs to be closer to the user, an example is a dynamic rendering of DOM elements. This is useful when new data needs to be shown right away without getting the blank white screen on a page reload. Indico does send JS, which is used for client-side features, but it focuses on keeping most of its features on the server.

To make the right decision if the module should be primarily developed for the client- or server-side or even as a microservice, a list of requirements to the module had to be defined. With the defined requirements in place, it

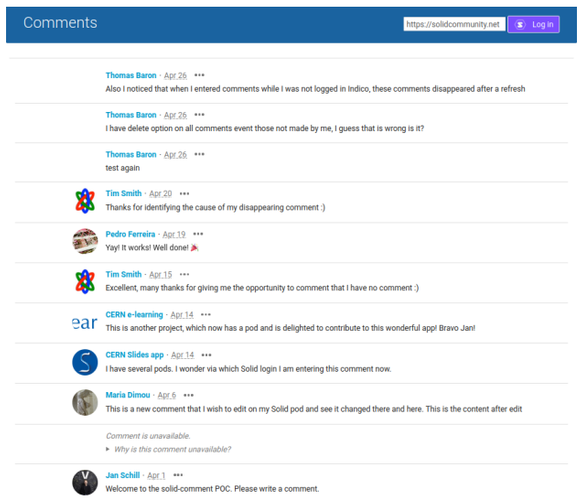


Fig. 5: User interface showing the comment module: Comments.

had to be figured out how much functionality can be extracted from existing libraries and how much needed to be implemented with the new module. Implementing existing functionality for a new programming language would defeat the POC's purpose of showing how an existing software could work with the Solid principles.

The rudimentary set of features to enable commenting for users in Indico while saving the data in a data pod includes:

1. Authentication with a Solid IDP
2. (Authenticated) Requests to a data pod
3. Parsing of structured data (Linked Data)

Client Approach

The module runs in the browser and is therefore written in JS. A programming language which compiles to JS, such as TS, is also possible. This means Indico remains mostly untouched but would have to serve the needed JS to the client on traffic to an event endpoint where the comment module is integrated.

Problem	Solution
Language	JS or TS
Framework	Native JS
Client	solid-client-js [18]
Authentication	solid-client-authn-browser [19]
RDF	solid-common-vocab-js [20], rdflib.js [21]

Table 2: Existing solutions to problems for a client approach.

The communication flow with the data pod and the module would happen primarily from the browser.

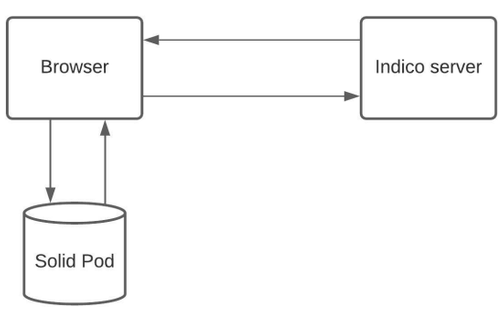


Fig. 6: Communication flow for a module developed on the client.

Microservice Approach

The microservice approach would allow developing the needed Solid logic on a separate service, which proxies all Solid related traffic through it and enables the Solid functionality. Most of the libraries from the client implementation can be used as well, as both developments would be written in JS. Only the authentication flow would work a bit differently.

Problem	Solution
Language	JS or TS
Framework	Node.js
Client	solid-client-js [18]
Authentication	solid-client-authn-node [22]
RDF	solid-common-vocab-js [20], rdflib.js [21]

Table 3: Existing solutions to problems for a microservice approach.

The microservice module would handle take all requests aimed at the data pod and make it compliant with the Solid server. It would provide the client with the proper Solid OIDC flow to attach the access token to all authenticated requests.

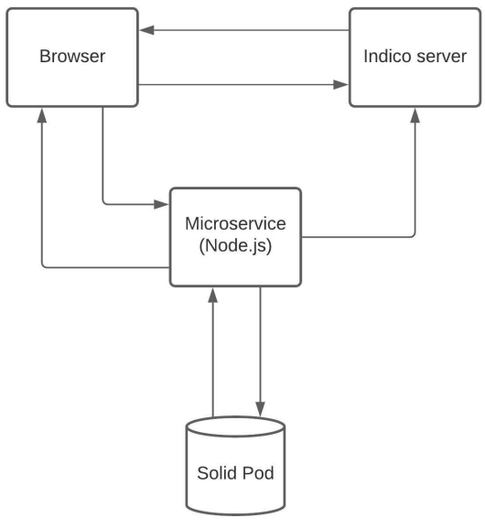


Fig. 7: Communication flow for a module developed as a microservice.

Server Approach

The goal of the server approach would be just like with the microservice approach to decouple the logic needed to work with Solid from the client and have it run on a server instance. The attractiveness for the server approach would be it could be fully integrated within Indico and be part of its Python codebase. The major drawbacks are no direct Solid libraries written in Python exist to allow a seamless integration into the ecosystem.

Problem	Solution
Language	Python
Framework	Flask
Client	-
Authentication	pyoidc [23] missing DPoP
RDF	solid-common-vocab-js [20], rdflib.js [21]

Table 4: Existing solutions to problems for a server approach.

The authentication library pyoidc allows authenticating with OIDC systems but is missing a mandatory feature called Demonstrating Proof-of-Possession (DPoP), which is needed to make requests to protected resources on

a data pod. DPoP is a technique to disallow replay attacks by attaching a DPoP token to the request indicating for what data pod the token can be used. This means when a request with a valid authentication token is made to a malicious pod and the adversary steals this authentication token, the authentication token is signed with a target to this malicious pod, which means the adversary cannot take this intercepted token and use it for another address, as it is only valid for requests to this malicious pod. The DPoP token therefore needs to newly generated on everything request [24].

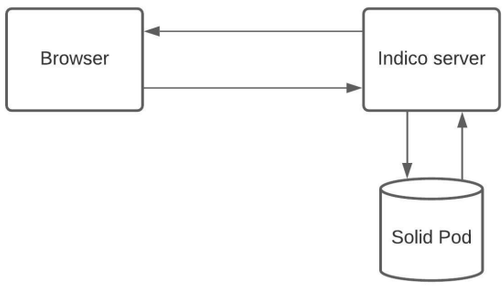


Fig. 8: Communication flow for a module developed on the server.

Comparison of the Different Approaches

Benefits from developing the module for the client:

- Necessary libraries exist (Major release for all basic Solid flows exist)
- Community support
- Programming effort for an minimum viable product (MVP) lowest
- Documentation on developing Solid apps in JS exist

Library	Description
solid-client	A client library for accessing data stored in Solid Pods.
solid-client-authn	A set of libraries for authenticating to Solid identity servers:solid-client-authn-browser for use in a browser.solid-client-authn-node for use in Node.js.
vocab-common-rdf	A library providing convenience objects for many RDF-related identifiers, such as the Person and familyName identifiers from the Schema.org vocabulary from Google, Microsoft and Yahoo!
vocab-solid-common	A library providing convenience objects for many Solid-related identifiers.
vocab-inrupt-common	A library providing convenience objects for Inrupt-related identifiers.

Table 5: Existing solutions to problems for a server approach.

Single Versus Multiple Resource(s) for Comments

Storing the comments in RDF can be done in two ways: storing it in one file as a graph with a list of comments, or creating a file for every comment.

When fetching the container with all the comment resources the request returns a Turtle file describing the container, but not the actual content of the contained resources. The misconception of receiving the content as well was thought to be a problem, as it was assumed an initial request to the container reading its child resources had to be made, and then for each resource a request needed to be built to retrieve the resource. This overhead was later disproved as the application where this module is embedded maintains the list of resources to be fetched. Therefore, no manual building of requests to those resources had to be done. The next paragraph also lays out how this design would not work with the protection of the resources.

(a) Comments being written in a single file resource on the data pod.

(b) Comments being written in multiple file resources on the data pod.

Fig. 9: Two different ways of saving the comment on the data pod.

Protection on Resource

Every container and resource in Solid is protected with WAC, which determines if specific agents, groups, or the world can have read, write, append, or control access. These control access modes are defined in ACL files. The Solid ACL inheritance algorithm looks for an ACL file attached to a specific resource, if it cannot find one it goes recursively up the file hierarchy and looks for ACLs on the containers. Indico allows two general types of protection *private* and *public* on its events. Public means open to everyone, no Indico account or any type of authorization is needed to see the event. Whereas private can be as fine-grained as only to specific agents or groups. A comment module is only valuable if the comments can be read by anyone and be written by authorized users.

In order for visitors of a private or public event in Indico to be able to see the comment, the comment's ACL needs to allow the public to read the resource. This can be achieved by using the *public* container, which comes with public-read by default on the Node Solid Server (NSS) or by creating a new container and setting the ACL with:

Listing 3.1: Setting default read for resources in container

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

# ... Definition for owner

<#example-container-name>
    a acl:Authorization;
    acl:agentClass foaf:Agent;
    acl:accessTo <./>;
    acl:mode acl:Read.
```

Every resource in this container is by definition readable by the public – if not otherwise stated in a more detailed resource ACL. The above definition even allows the reading of the container's content, meaning a request to the container would yield a list of resources in the container. This becomes unpleasant if the Indico event is private and the comments for this Indico event should not be read by the world, which is entirely possible when browsing to the location of a specific data pod and then looking into the public container.

To prevent a random agent to see the contents of a container, the container can be set to private, with the container's resources still be public. This would allow everyone provided they have the Uniform Resource Locator (URL) to browse to the public resource and read it, but not look into the resource's parent container. To achieve this behavior with ACL, the container needs to just define its owner and no specific rules for the public, as WAC comes with a default private access control. Each child resource needs to define an ACL now, allowing public read. The container's ACL would look like the following with just an owner defined:

Listing 3.2: Container ACL with owner defined.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.

<#owner>
    a acl:Authorization;
    acl:agent <https://janschill.net/profile/card#me>;
    acl:accessTo <./>;
    acl:default <./>;
    acl:mode acl:Read, acl:Write, acl:Control.
```

A child resource would allow public read with:

Listing 3.3: Giving read mode to child resources

```
@prefix : <#>.
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

# ... Definition for owner

:Read
    a acl:Authorization;
    acl:accessTo <test.txt>;
    acl:agentClass foaf:Agent;
    acl:mode n0:Read.
```

Another approach and the one implemented after iterating through the previous ones is to have the container's ACL resource define a default access mode for its child resources. This way one ACL only needs to be created on the container and all resources have proper access modes for public read and are not listed publicly in the container's description.

Listing 3.4: Giving default read mode to all child resources of container.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix target: <./>.

:ReadDefault
    a acl:Authorization;
    acl:default target:;
    acl:agentClass foaf:Agent;
    acl:mode acl:Read.
```

Preventing Unwanted Discovery for Resources

With the resources having proper access modes but being publicly readable a simple naming convention of taking the International Organization for Standardization (ISO) 8601 string and using it as a filename for the resources created on the data pod does not suffice – even though it is a good strategy when looking for a reliable naming convention to prevent duplication. Considering performance improvements such as pagination for future iterations of the module, which would require some sort of iterative indication, a combination of randomness, but also an order indicator it was settled for using universally unique identifier (UUID) plus the ISO 8601 string to form a filename.

Other ideas included hashing a random string with the timestamp to generate non-guessable filenames. The filename would need to use the same hash function to decipher the filename to figure out when the comment was generated. UUID is a reliable and easy-to-use system to generate *truly* globally unique strings.

Modification of Resource From Data Pod

When the users are in control of their data it means they can revoke access any time, or even change their data to their liking. This means when the comment is initially created through Indico and the commenting system's user interface it does not mean this comment needs to remain as is. Even if the interface of the system does not allow modification, a modification can still happen at the source of the storage of the comment.

Regular comment modules usually allow modifications of a submitted text throughout its existence. Twitter is an example where modifications are not allowed after posting [25].

When this aspect was discovered it was decided to allow this sort of behavior, as it seems to be natural in this context to be able to edit one's own comments.

Mitigation of Spam

Enabling user input in form of a comment module without application authentication is a gateway to spam. Even though authentication with a Solid IDP is necessary, it does not hinder a malicious actor to create a multitude of Solid accounts and spam into the application. In the first iteration of the module, only Solid authentication was integrated into the module and would therefore allow anyone with a Solid account to post comments and thus also spam the Indico event theoretically. In a second iteration of the module, another authentication layer was added to mitigate spam from outside Indico. For CERN's use-case, an authenticated Indico session was enough. This adds an extra step to the comment process and it is ensured only registered Indico users can comment.

Giving Application Full Control of Data Pod

To create or change programmatically ACLs requires *control* access to the container. By default NSS asks for the permissions when authenticating for the first time in the Solid OIDC flow between the *solid-comment* module and Solid IDP. The permissions granted to the application are on the root container of the data pod. Meaning, giving an application control access allows the application to read, write, change ACLs on the entire data pod. This is obviously troubling, as a simple application as a commenting module needs to have control access to set the needed ACLs for the containers and resources it creates.

The current implementation has not a built-in solution, but one way of solving it is the use of an application launcher, which is an application itself with full control access and then limits the access controls of the *solid-comment* module by creating a dedicated container for it and setting the needed ACL for this specific container only.

6.1.4 Integration with Indico

The need for integration with Indico is twofold: serving the module to the client and being the provider to the list of references to the comments that have been posted on a specific event.

Storing Reference to Comments in Indico

Indico operates using the relational database PostgreSQL [26]. When a comment gets posted and stored on an external data pod a reference to the location of the comment needs to be kept in order for Indico to pull the comment and render it in its frontend. Several possibilities are imaginable.

1. Create new EventComments table
2. Store in an Indico data pod
3. Use existing EventSettings table

One solution would be to create a new database table that associates with event using a foreign key and the event id and then stores every comment in its own row with all necessary meta information. The database schema would have to be updated in order to register the new table properly and then executed. A quicker alternative to this is the usage of an existing database table called `EventSettings`. `EventSettings` is for additional information for an event and is meant to be a lightweight key-value store for these extra data (often used with Indico plugins).

Another option would be to leave the database alone and embrace the Solid way of storing the comment references by deploying an own Indico Solid data pod. The data pod could hold a resource for every event and link to the data pods where the comments can be found. This solution is definitely the most aligned with the Solid principles and could have been an interesting challenge to develop, but it was identified to late to actually push through.

Enforce Authenticated Session For Posting Comments

It was established through the iterations of the design with the stakeholders that an authenticated Indico session is deemed necessary in addition to the already authenticated Solid IDP session with the data pod. The main motivation was to keep it as laborious as possible for spam to reach the system.

The awkwardness in validating for an existing Indico session and then be able to send requests to a protected endpoint comes with the great decoupling of the comment module and Indico, which is admirable, but also make this feature more challenging to implement. Indico uses a framework called Axios [27] for its Hypertext Transfer Protocol (HTTP) client request handling. In Indico Axios is configured with all necessary HTTP headers and when a session exists also equipped with the auth-token. The comment module can simply be extended to allow the usage of a custom HTTP client for sending requests to Indico. When the module is initialized the Axios client is already readily configured and needs to be passed over to the module. Both system are still decoupled and the comment module works with or without a custom HTTP client.

6.1.5 Evaluation

This section focuses on evaluating the system. This shall be done by iterating through the system's requirements from the stakeholders and how the architecture lives up to those expectations. The evaluation is done with the help of the architectural Software Quality Assurance (aSQA) framework to ensure continuous quality assessment and prioritizing with a lightweight technique [28]. aSQA contains seven process steps as shown in figure 10.

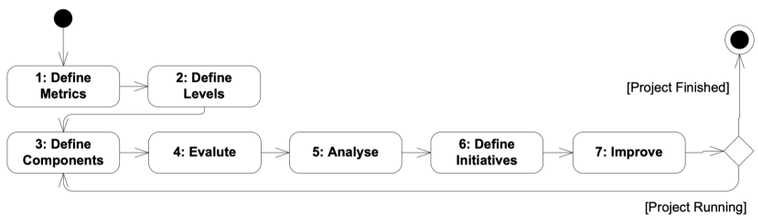


Fig. 10: aSQA Process Steps from [28]

1. **Define Metrics** to the components of the system. This can be done with QA scenarios (QASs), that may define the desired quality.
2. **Define Levels** to measure the state of the system
3. **Define Components** to scope the parts of the system that will be evaluated
4. **Evaluate** by taking the scenarios and looking at the architecture of the system. An evaluation matrix is used to define the following levels: (t)arget, (c)urrent, (health), (i)mportance, (f)ocus
5. **Analysis** will look at the areas where the value of focus is high

Metrics

The defined QAs are useful to know, but cannot be tested or often qualities of a system are hard to categorize with an adequate QA. A solution to the problem of untestability and overlapping concerns are QASs. QASs can be used to characterize QAs [29]. Based on the scenarios the evaluation will be carried out. The scenarios will always have the QAs in mind and are defined with the focus on those. The different scenarios are predicted to be the most common actions on the system and thus will stress the system and its architecture the most based on the QAs.

The previously picked QAs are as follows:

1. Security
2. Performance
3. Usability

Possible scenarios are centered around data not being available or altered or scenarios of malicious behavior from an adversary by injecting code to misuse the system in unintended ways:

1. As a user I expect after commenting to see my comment in Indico, but also in my data pod
2. As a user I expect after deleting my comment in my data pod that the comment is not shown in Indico
3. As a user I expect after I edit my comment in my data pod that the comment will be updated in Indico
4. As a user I expect to be able to revoke access to my data pod for the application
5. As a user I expect to see others' comments
6. As a user I expect to only give access to the necessary data

Levels

The levels are used to measure the state of the architecture of system. These can be freely chosen, but follow an ordinal scale ranging from 1 to 5 in the original paper and will therefore be adapted [28]. The levels will help to rate the quality of the system's architecture, where 1 is the lowest quality and 5 the highest rating.

Components

To evaluate the security of the chosen architecture of the system it makes sense to look at certain components of the system. For this commenting system it would be sensible to analyze the `auth` package and the modules that interface the user input and the system. The user input modules can be found in the `component` package.

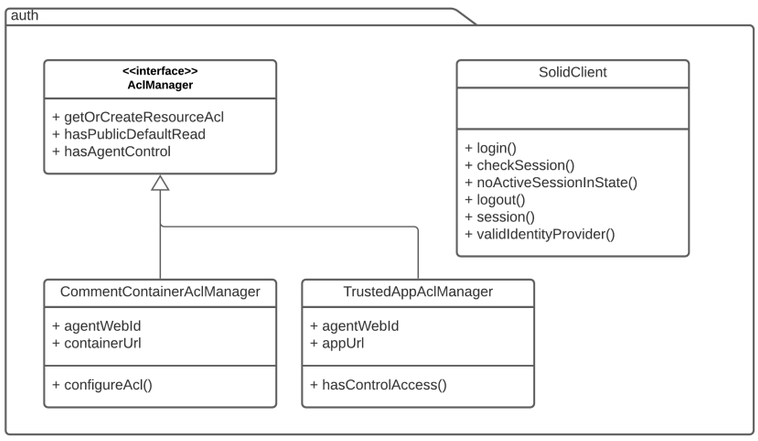


Fig. 11: The `auth` package with its classes.

The `auth` package holds classes handling the ACL logic necessary. The logic ranges from checking for existing specific authorization modes or even creating new ACL files for resources and updating existing resources. A careless architecture could result in either giving too much access control or allowing privilege escalation, where a flaw in the software can be used to elevate the access modes for an adversary.

The `auth` package further holds the `SolidClient` class. The `SolidClass` is a wrapper class for the external authentication functions, which bring the authentication and session management for the Solid ecosystem. Naturally, all authentication modules need to be inspected and made sure no vulnerabilities are let in.

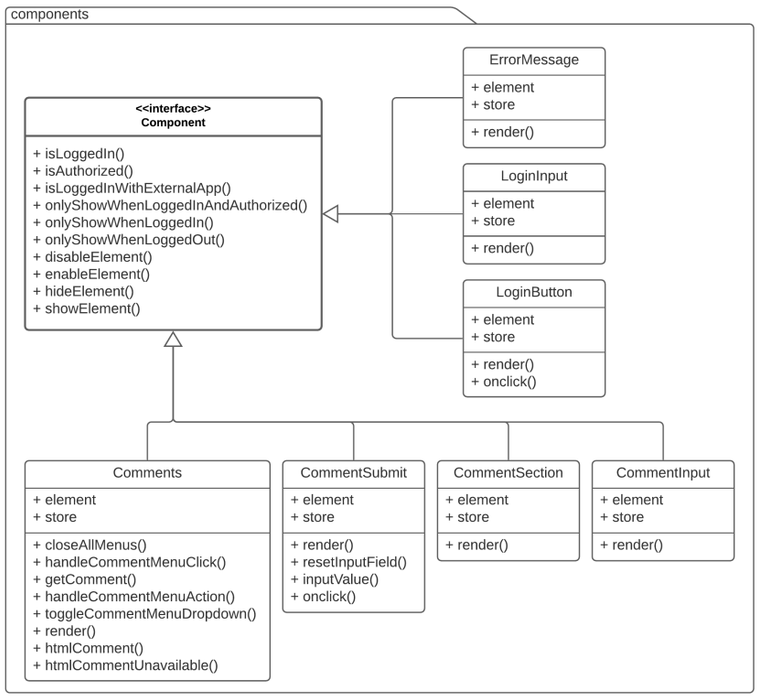


Fig. 12: The `components` package with its classes.

The second package, `components`, holds all elements in the user interface that have to interact with the state management module. These classes range from the input field for the WebID URI to retrieve the address of the user's IDP to the rendered comments.

Evaluate

- **Target**: What is the desired quality level?
- **Current**: What is the current level?
- **Health**: Where are the largest quality problems?
- **Importance**: How important is it to move from current to target level?
- **Focus**: What are the largest and most important quality problems?
- **Valid level values**: 1, 2, 3, 4, 5 (higher equals better)

Health and **focus** are not set by the evaluator but rather calculated from other levels.

$$health = 5 - max(0, (target - current))$$

$$focus = ceil((6 - health) * importance/5)$$

1. A user logs in with their WebID and posts a comment
2. An adversary posts a comment with a string containing a cross-site scripting (XSS) attack
3. High load of traffic is occurring with comments being posted within seconds
4. An adversary deploys software on their pod to serve varying resources based on the geolocation of the connecting Internet Protocol (IP) address

`auth`	T	C	H	I	F
Security	5	2	2	5	4
Performance	5	1	1	5	5
Availability	4	3	4	2	1
Usability	5	2	2	4	4

`components`	T	C	H	I	F
Security	5	3	3	5	3
Performance	5	1	1	5	5
Availability	4	3	4	2	1
Usability	5	3	3	3	2

Table 6: Evaluation of the two packages based on the QASs

6.1.6 Analysis

The following analysis section will explain how the values of the two tables from 6 came together and will give some more insights into how the modules are working and behaving in certain situations. After the explanation and analysis of this particular evaluation a broader analysis of the POC will be given, where critical areas in the architecture will be highlighted and some possibilities for future iterations of it.

auth and *components* packages

The `SolidClient` class is part of the `auth` component package, which handles the session management. When the comment module is loaded upon page load it is *booted* with the flow shown in 13. It can be observed that every time the module is loaded it has to do four round trips to the IDP and data pod. Once the application is booted, the list of comment URIs are passed to the application and it fetches all comments from their remote data pods. This behavior is linked to the `components` module, as it is responsible for rendering all these comments in the DOM. Further performance analyses in regards to the fetching of the comments is being done below in the paragraph 6.1.6. But when the comments are being fetched a second request happens that loads the WebID profile of the author of the comment. Resulting in $n * 2$ requests, where n is the number of comments. The exact flow of getting the comments is shown in figure 14.

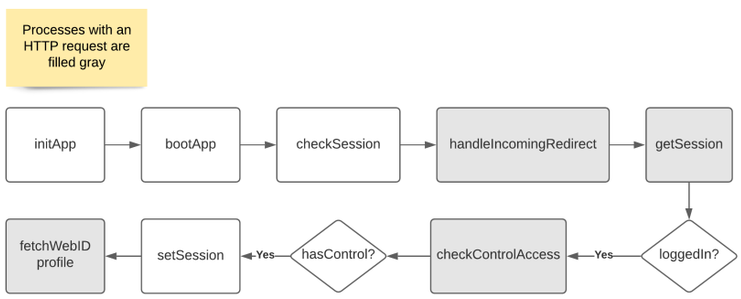


Fig. 13: Application boot flow. External HTTP requests are indicated by a filled grey box.

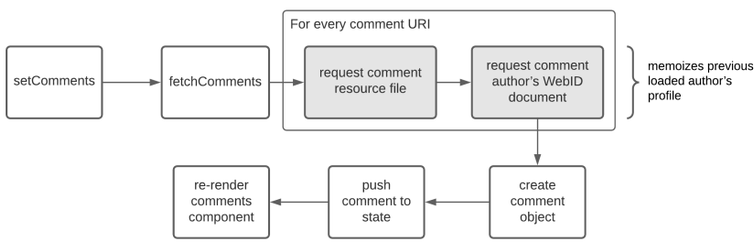


Fig. 14: Requesting comments from data pod. External HTTP requests are indicated by a filled grey box.

The application boot with successive comment fetching happens every time the system is loaded, which is in minimum four requests, when no comments are loaded and $n * 2 + 4$ requests with n comments.

In regards to *availability* for both looked at modules if a data pod is unavailable the `auth` module will simply not be able to do its checks on whether a session is present. The HTTP requests checking the session will respond with $40x$ HTTP status codes and simply disallow any comment posting. For the `components` module a dedicated message has been implemented which will render a hint on why a comment may not have loaded when a data pod is unreachable. The hint renders one long message guessing what has happened, but is not clever about inferring the reason.

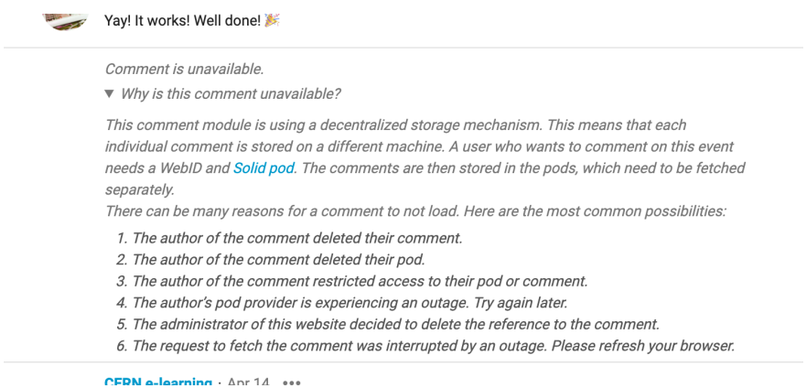


Fig. 15: Hint when a comment could not be loaded

Another striking value is the high focus for the usability in the `auth` component. This can be explained with the fact that it is not made clear to the user that *control* access to the root container of the data pod is needed in order for the application to function properly. Only when an authenticated session from the Solid IDP is present and the *control* access is lacking it will show in the interface. Firstly, this results in a bad user experience, but more over the fact that the user needs to gives full control over the data pod to the application is unfortunate. The end-user who is not a Solid professional will have a hard time understanding the reason and may step back from application that need this much control. The evaluation for the *security* resulted therefore also with a worse level than it could have scored. The security within the module is relatively stable with sanitization of inputs not allowing any malicious code to be executed when fetching resource from data pods.

Performance

The chosen technique of storing the comments is not as efficient as it could be. Every comment is stored as a self-contained Turtle file on the author's comment with Indico holding the URI to be able to fetch it on demand. As soon as the client is loading the module with the comments the client will have to make n requests, n being the number of comments.

The first improvement to this approach could be pagination. Pagination limits the number of initially loaded comments to a defined amount. This can be achieved by utilizing the date and time from the file name of the comment. Only the most recent n comments by time will have to be fetched. When the user clicks on a load more comments button, n more comments will be loaded. In the same area of improvement, the ways the comments are rendered can also be improved. The comments as of now will all be fetched and only when all requests are done it will be rendered in the DOM. If the comments are *separated* from each other and a comment is rendered as soon as it successfully fetched, it would speed up the time until a user could start seeing comments.

The second improvement could be a grouping of comments by the author. This way the number of requests would now be bound to the number of authors. In the worst case, where all comments are from different authors, it would not bring any improvement over the initial architecture. The design for this would be to change the storage from the multi-file approach to a single-file approach. On the data pod, one file would exist holding all comments from one author for one event. This file can be fetched with one request containing multiple comments.

Another enhancement can be attained by caching. In its current design, the comments are freshly fetched on each page load. Server-side HTTP caching is out of control for the clients, and completely relies on the Server implementations to do so. By Solid specification, HTTP caching is prescribed, but not vital [30], which means it *should* implement it, but does not have to. Regardless, the improvement would not be in the number of round trips the client has to complete, but rather in the time for the server to calculate the response it sends out [31]. A real improvement on consecutive page visits are won by client-side HTTP caching. Upon successive visits to a cached website, a previously fetched and in the browser's local storage stored copy of the response would be served [31]. The result would be immediately loaded, but might not serve the latest and up-to-date asset from the server. The *freshness* of the response is controlled using the `Cache-Control` header in either the request or response of the HTTP exchange between client and server [31].

More performance solutions shall be looked at with the other upcoming areas in this analysis section.

Security: Modification of Resources

In the case of comments being the resources that are being shared between data pod and application, it has been established a modification from outside the application is acceptable behavior. For a resource where a modification should be monitored or even forbidden. A few paths are conceivable.

Storing the comments on the author's data pod could be given up and instead be done by a trusted entity, e.g. the application embedding the comment module. It would result in CERN hosting one data pod for their Indico instance. Each event would create a container in this data pod with *append* access mode for the public. *append* allows an agent to add new resources to a container, but not modify or delete any [30]. Through this access mode a user can post comments freely, but cannot modify them at a later stage, as the data is now in Indico's data pod and the user is lacking the access control to edit existing resources. Having one data pod responsible for all comments also improves the performance by reducing the number of HTTP requests needed. Whereas before the requests were either bound to the number of authors or even by the number of comments, it is now only a single request to the event's container on the Indico data pod, to fetch a single file with all comments written in. A drawback to

this approach is the comments are again stored centralized in one storage, though because it is using a Solid server instead of a web server with unstructured data storage, the data is now structured and thus interoperable through *Linked Data*. Besides storing it in Indico's data pod, a copy of the comment could be stored in the author's data pod. The author acquires a version of the comment and Indico holds the single source of truth (SSOT).

One more option to control the modification on resources while not giving up on decentralized storage in the author's pods is to use versioning on the comments. Indico would when a user creates and submits a comment store a hashed value of the comment's content. When then serving the comments from the external data pods to visiting clients the received comments would be hashed and compared to the Indico stored hash value. It is important to associate the comment's URL with the hash to be able to make a proper comparison on the correct resources.

An important aspect of allowing the update of existing resources is to have an indication in the user interface notifying readers about a changed text. A simple *edited*-hint would suffice. When a user is browsing the comments and follows a conversation, which is met with an abrupt disruption of flow in the conversation, an updated comment could be the result and an indication would help to identify such a situation. To detect a change in the comment's content the same hashing function can be used as described before.

EventSettingsProxy Scalability Issues

The implementation with EventSettingsProxy is not scalable and a production-ready technique. Even though transactions are being used in the database and a dedicated table is being used to persist all related settings for the events, it can still happen that race conditions occur. As for example two requests read the event settings table and retrieve an entry with $[1,2]$ for the column holding the list of comment URLs. When request A now writes 3 and request B write 4 it generate two Structured Query Language (SQL) `UPDATE` statements, one with $[1,2,3]$ and one with $[1,2,4]$ where the second overwrites the first and only $[1,2,4]$ is persisted.

Justification for implementing it in this way is a faster *time-to-market* allowing a quicker development and thus testing of the prototype. It was never expected to test the prototype where these race conditions would occur. If a production-ready module should be developed, a separated database table would need to be created and every new comment URI would be added to the database by using SQL `INSERT` statement, which are safe against race conditions. A further benefit would be the ability to store more comment specific meta information such as signatures or hash values to be able to detect changes in pulled in comments.

Indico Solid Proxying

Indico could develop an API which sits between Indico and the Solid data pods to then for example make all the requests on behalf of the clients. A proxy as such would hide all the clients' IP addresses and would therefore disallow the logging of client IP addresses by a malicious data pod. A performance improvement is also foreseeable as the Indico proxy could make the requests to the different data pods and put them into cache where they would then be served from. By caching the request in this layer, the request do not have to be made by every client every time on page load. Loading resources before they are requested by clients and then serving clients from a cache is called cache warming [**cache-warming**]. Another benefit of doing the requests using an Indico proxy is also that a malicious data pod or just one that wants to do some kind of targeted advertisement, cannot identify the clients by their IP addresses and therefore not send different responses upon requests from different IP addresses based on geolocation.

6.2 POC 2: Auto-Complete Form Inputs for Conference Registration in Indico

The second prototype aims at connecting Solid with the conference registration module in Indico. When registering for a conference an HTML form is presented with fields previously defined by the conference manager, who deemed those fields necessary. A form always contains personal information of name and email address, but is not limited to it and can even range to more sensitive information such as copies of personal identification documents. Information of this type has perfect motivation to remain in the hand of the owner and not be stored in a remote data store, uncontrollable and unknown to the registrants.

Therefore, the initial aim was to extend the registration module to allow storage of sensible information in a data pod, where the user has full control and can handle the data to their own liking. This was decided to not be viable and the prototype changed to allow the extraction of data from a data pod and then use this data to map it to input fields of an Indico conference registration form – the functionality to then also store the data from the registration in the user's data pod and not in Indico was dropped. The reasons and comprehensive analysis will be shared after the architectural analysis, synthesis, and evaluation of the developed POC in the *design* 6.2.3 and *analysis* 6.2.6 sections of this chapter.

6.2.1 Architectural Analysis and Synthesis

System Description

The system pulls in a resource from a data pod in the RDF format Turtle, maps the received information to input fields in an HTML form and fills in missing inputs or asks the user if the values should be replaced.

Features

The core functionality of the module consists of the following features:

1. A user can provide a the URI to a public Turtle file
2. A user can choose to accept or reject values pulled in when values already exist

These features allow the development of a module giving users the ability to pull in their WebID profile document and use the information provided in it to populate a conference registration.

Type of Users

There is one type of user for this system, who is the user with a WebID profile and interested in using the existing information to fill in a form.

Context Diagram

Other types of involved parties are the ones maintaining or developing the system and application. These are shown in the context diagram 16.

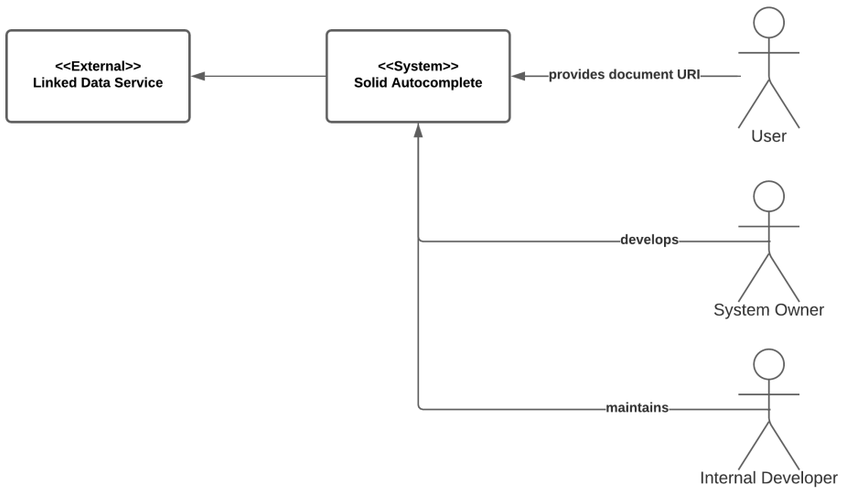


Fig. 16: Context diagram showing users and external services of the system.

Sequence Diagram

The following succeeding diagram shows the sequential flow through the system upon initialization and button press. The focus was laid in the diagram to show the request/response cycles in the infrastructure.

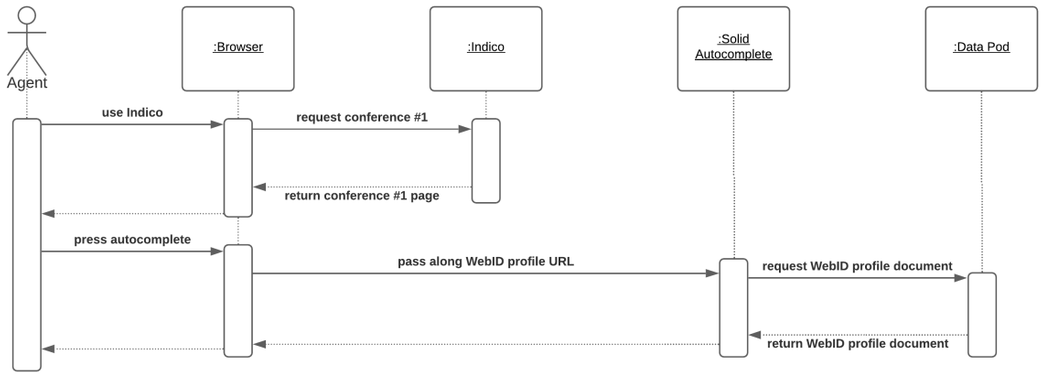


Fig. 17: Sequence diagram showing the sequential process through pulling in data from a data pod.

Stakeholders

Both modules were established with the same group of people and therefore carry the same set of stakeholders. The stakeholders defined in the section of the first POC can be found here [**poc1-stakeholders**].

Drivers

The drivers for this architecture endure the same as well. A system which operates on a form where sensitive data is exposed needs a high-level of security – under no circumstance should traffic be intercepted and leak any of such information. The performance for this module is also important, as conference tend to open their registration at a certain time, with high traffic spikes at those times of registration opening or announcements. The same reasoning as before applies to the *Usability* of the module. When a new software design is used, it should be made as simple as possible for the user to interact with it and ideally not even notice a difference to traditional practices.

1. Security
2. Performance
3. Usability

6.2.2 User Interface

The user interface for this module is

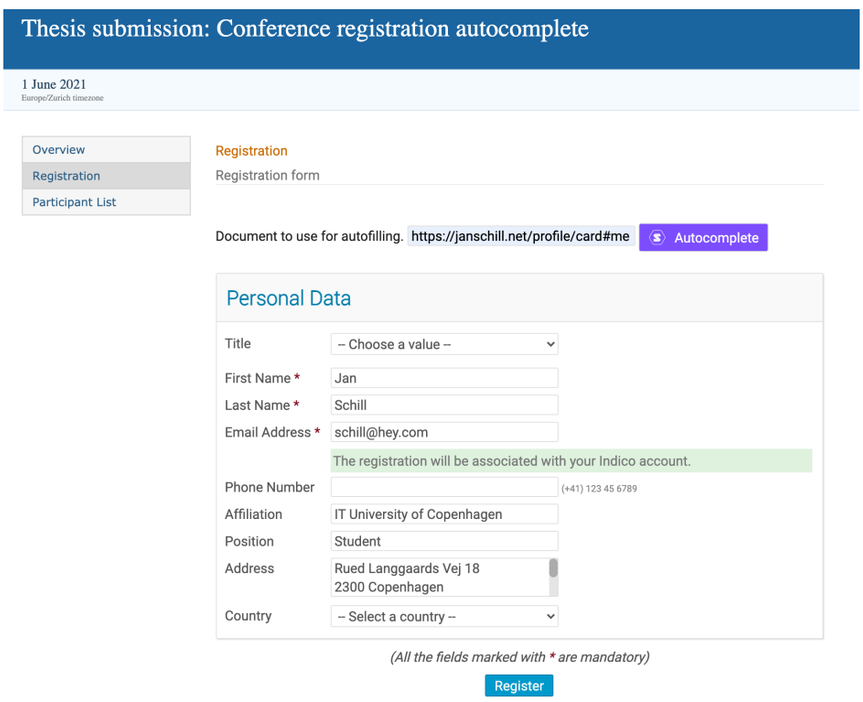


Fig. 18: User interface showing the comment module: Comments.

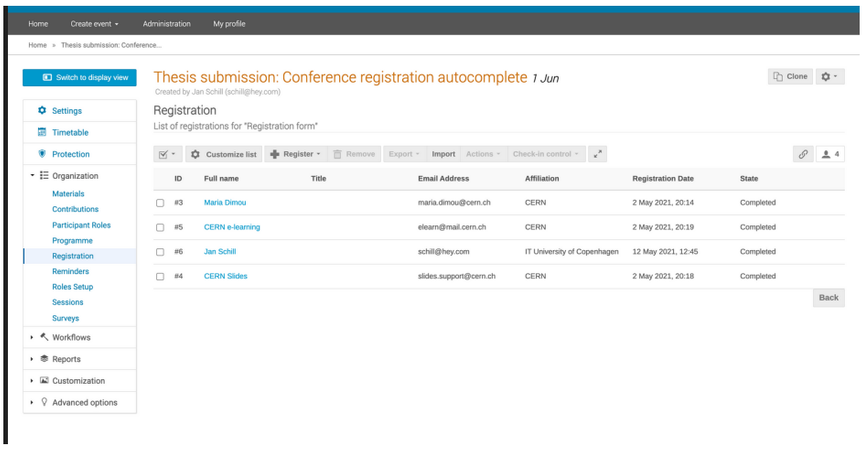


Fig. 19: User interface showing the comment module: Comments.

6.2.3 Design

The following section will shed light on the design decision made in the process of sketching out the architecture of the module and developing the system. It will introduce the different challenges and how they were overcome. Especially, it will give an explanation of what led to the change of core functionality of the module. In the succeeding section, the choices will be analyzed more explicitly.

Mapping Structured Data To HTML Inputs

A great benefit of using Linked Data is its semantic structure and the interoperability it brings when using in systems supporting it. This means the data are using public vocabularies to be described and thus can be reasoned about by machines and can be move from one system into another one and seamlessly integrated. Indico does not make use of any common RDF features and can therefore not extract any of those benefits. Because of this, no simple way of mapping the incoming data from the data pod to the inputs exist, but a few options are feasible.

1. Enrich the form with some kind of descriptive indicator
2. Parse and process existing values of attributes and labels

In the HTML specification an `autocomplete` attribute exist, which is used by browsers to prefill data from previously used values [32]. The value of the `autocomplete` key-value pair describes the input and what data is to be expected.

Listing 3.5: Autocomplete attribute on input field

```
<form method=post action="/">
  <p><label>Full name: <input type=text autocomplete=name></label>
  <p><label>Credit card number: <input type=text inputmode=numeric autocomplete=cc-number></label>
  <p><label>Expiry Date: <input type=month autocomplete=cc-exp></label>
  <p><input type=submit value="Submit">
</form>
```

In listing 3.5 the browser would suggest cached values when the user agent clicks into an input. This is the perfect indicator to *autofill* form inputs. Unfortunately, this attribute is not being utilized in Indico. Another and even better approach would be to annotate the input fields straight up with the vocabulary of for example Schema.org [33]. The mapping of the input fields and the fetched Turtle resource could then happen directly using the *predicates* of both structured formats. To change the AngularJS form and implement new features into Indico were already established not to be viable. Nevertheless, the `autocomplete` feature shall be kept in mind.

Without being able to extract from the `autocomplete` attribute within Indico it shall still be implemented and used, as other systems might use the attribute. Other means of extracting some kind of information from the form are the ID and name attributes or even the `TextNode` of the input label. The ID and name are often equipped with information about the input they are defined on. These values and additionally the value of the label node can be extracted and mapped against a dictionary of key-value pairs to find out what the input is supposed to be filled in with.

A possible mapping of address information could look like as in listing 3.6. The implementation shows a JS implementation where the key in the `address` object are the possibly extracted values from the different techniques described above and the value is in this example the vCARD [34] equivalent. vCARD is being used by the NSS to describe the information in the WebID profile document.

Listing 3.6: Dictionary to map extracted values with predicates from Turtle resource

```
const address = {
  address: 'full_address',
  country: VCARD.country_name.value,
  countryName: VCARD.country_name.value,
  region: VCARD.region.value,
  locality: VCARD.locality.value,
```

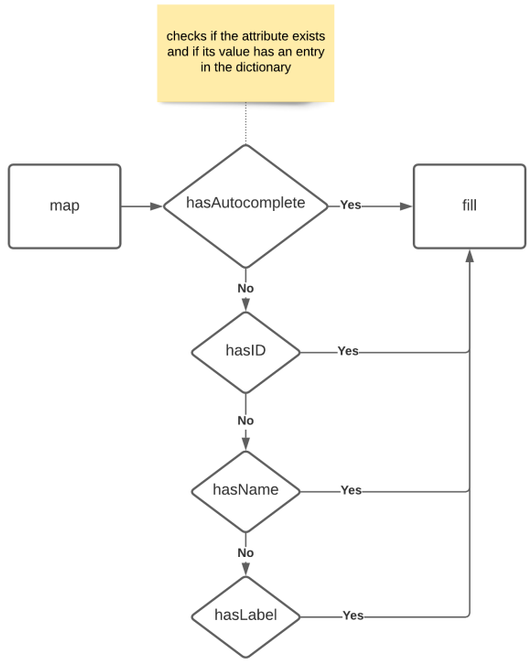


Fig. 20: The flow to find a source for dictionary mapping.

```
  city: VCARD.locality.value,
  streetAddress: VCARD.street_address.value,
  street: VCARD.street_address.value,
  postalCode: VCARD.postal_code.value
}
```

Listing 3.7 shows part of the WebID profile document where the address of an agent is described.

Listing 3.7: Extraction from WebID profile document showing address.

```
@prefix : <#>.
@prefix n: <http://www.w3.org/2006/vcard/ns#>.
# ...
:id1614172452178
  n:country-name "Denmark";
  n:locality "Copenhagen";
  n:postal-code "2300";
  n:region "Zealand";
  n:street-address "Rued Langgaards Vej 18".
# ...
:me n:hasAddress :id1614172452178;
# ...
```

6.2.4 Integration With Indico

A few challenges arose when integrating the module with Indico. A critical issue was when it was found out that the form is not rendered on the server and send as HTML in the response body as expected. Another challenge was when programmatically filling in the input fields of the inputs the frontend form validation would not detect a change in the input values and thus would think no input value is given.

Bind to Dynamically Created Form

Indico builds the registration form dynamically using a frontend library called AngularJS [35]. This introduces a challenge of interacting with the form using JS. Frontend libraries that either create new or modify existing DOM nodes need to wait until the complete DOM tree is loaded, as they bind to one DOM node from the initially served HTML document and then do their operations on this node. Meaning, AngularJS waits until the whole HTML document is parsed and rendered in the browser and then starts creating its form from scratch, which is then being rendered by the browser. The problem with this is in order to operate on the form, which is necessary for the module to be able to read the form inputs and labels and also to set their values, the JS code from this prototype needs to know when the form was successfully rendered. Three options are possible to achieve this.

1. Implement the autocomplete functionality in the existing AngularJS form code
2. Dispatch an event to notify the autocomplete module the form has been rendered
3. Use the `MutationObserver` to detect the form creation

Solutions 1 and 2 both involve the need to work with the AngularJS form, which is written in a legacy version and was recommended by an Indico developer to – if possible – be avoided. Indico developers also plan on removing AngularJS altogether and replace it with a more popular frontend framework. Therefore, option 3 was chosen even though the usage of a `MutationObserver` instance might add performance degradation to the page load [36]. The performance shall not be analyzed more carefully as it is not of relevance to prove the realization of the prototype.

The `MutationObserver` is initialized as soon as the DOM is rendered, it then takes a node as a target to observe and will register all modifications on this node: the creation of children nodes in it, updates to the node itself or any other modifications. To reduce computation the Indico code can be analyzed to find a suitable DOM element to have as a target node. This node needs to exist when the DOM is loaded and needs to contain the final form node. When looking at the final rendered document containing the AngularJS form one notices that the form has an ID, which can be used to observe its creation.

Listing 3.8: Observe function in Indico

```
function observeFormCreation() {
  // ID of the AngularJS form, its creation needs to be observed
  const formId = 'registrationForm';
  // Candidate to limit observing scope
  const $conferencePage = document.querySelector('.conference-page');
  const targetNode = $conferencePage;
  // Only observe nodes, not attributes
  const config = {attributes: false, childList: true, subtree: true};

  const callback = (mutationsList, observer) => {
    // Contains all mutations in the targetNode
    for (const mutation of mutationsList) {
      // Node has beed added or removed
      if (mutation.type === 'childList') {
        // Look at all added nodes
        for (const node of mutation.addedNodes) {
          // Look for the AngularJS form
          if (node.id === formId) {
            // Once found, stop observing
            observer.disconnect();
            // Initialize the autocomplete library
            const solidAutocomplete = new SolidAutocomplete({form: node});
            solidAutocomplete.createAutocompleteDomControls(node);
          }
        }
      }
    }
  };
  // Start observing
  const observer = new MutationObserver(callback);
  observer.observe(targetNode, config)
}
```

Detect Input Change for Frontend Validation

Indico deploys a frontend validation to make sure it receives proper values for the form's inputs. A basic validation is the check for a presence of values in required inputs. This way Indico can render a hint on the input fields with invalid values, such as if no input was given in a required email address field. It turns out Indico has `DOM Event Listeners` which detect if an input field is clicked in and if it is receiving inputs through a user typing in it. This validation implementation does not detect when changing the value of the `value` attribute of an input node, thus complaining when setting the values through JS.

Listing 3.9: Changing the value of an input node.

```
const formInputField = document.querySelector('.exampleFormInput')
formInputField.value = 'New value'
```

The example code in listing 3.9 would not be detected by Indico and would upon submission render a missing value hint.

Two ways of fixing the problem are conceivable.

1. Change Indico frontend validation to detect value change
2. Dispatch event when setting values in the autocomplete module

Changing the Indico frontend validation might turn out to be more time-consuming than anticipated and is therefore not viable, also it is unknown if a change is wanted by the Indico development team in the first place. The idea for a working implementation is to validate on submission of the form instead of validation when a change on the input is detected. As mentioned before the problem with the used `oninput` event is it does not detect when the value is set programmatically. If the validation is only happening when the values are tried to be sent to the server by submission the input values can be parsed and the correct value is detected – no matter how it was set.

The other and also picked solution is to trigger the `oninput` event that is being listened on by the Indico validation. The event dispatch needs to happen as soon the values are set within the module and because this happens in the module and the module is a self-contained module without any knowledge of Indico, the module should not be directly changed, but rather allow a callback function to be passed to it and then execute when appropriate.

Listing 3.10: Dispatching the `oninput` event within a callback.

```
function triggerInputEvent(inputs) {
  for (let i = 0; i < inputs.length; i++) {
    const element = inputs[i];
    [element, element.parentNode].forEach(node => {
      if ('createEvent' in document) {
        const evt = document.createEvent('HTMLEvents');
        evt.initEvent('input', false, true);
        node.dispatchEvent(evt);
      } else {
        node.fireEvent('oninput');
      }
    });
  }
}
```

6.2.5 Evaluation

Just like with the first POC from a section before 6.1, this POC shall be evaluated using the same motivation and framework. The stakeholders and QAs continue to be as is. An TODO:

Metrics

TODO:

Levels

The same levels 1 to 5 as in the first evaluation 6.1.5 will be used to rate the architecture of this system.

Components

TODO:

6.2.6 Analysis

The analysis of the second POC will focus on the original design idea and what led to a change of mind. It will do so by looking at the relevant modules in Indico that caused the direction for the POC to alter. It will also be analyzed, what role Solid's design played in the abandonment of the idea on storing the registration data in data pods and effectively giving up usage control.

* Gave up usage control * Why it didn't work, and **what is necessary to make it work** * usage control * question the choice of Indico usage control * versioning of tag of personal data * high-level constraints * implementation * indico limits this * if indico in this way or more generally

Iterations

TODO: 1st iteration, save data in pod 2nd iteration, only pull data from pod

TODO: Include these somehow:

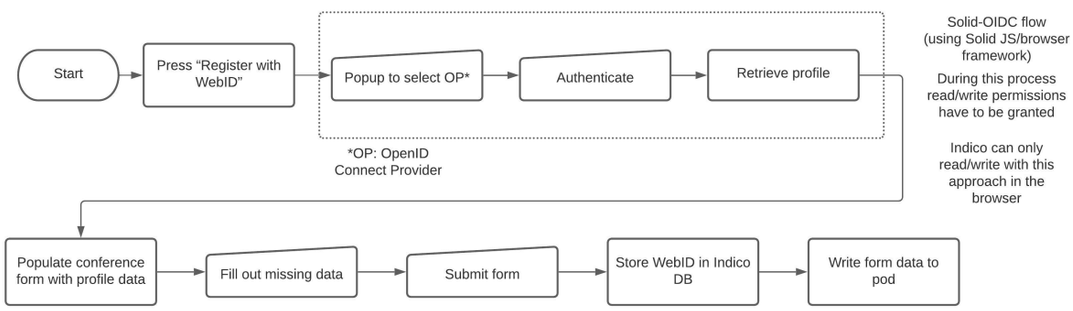


Fig. 21: TODO:

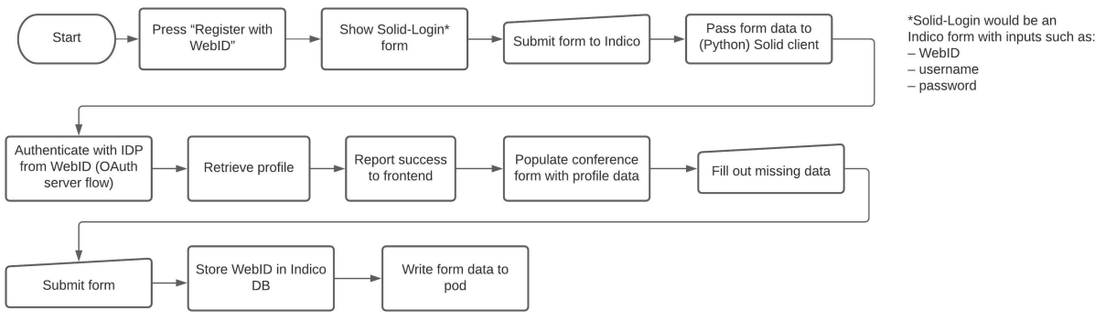


Fig. 22: TODO:

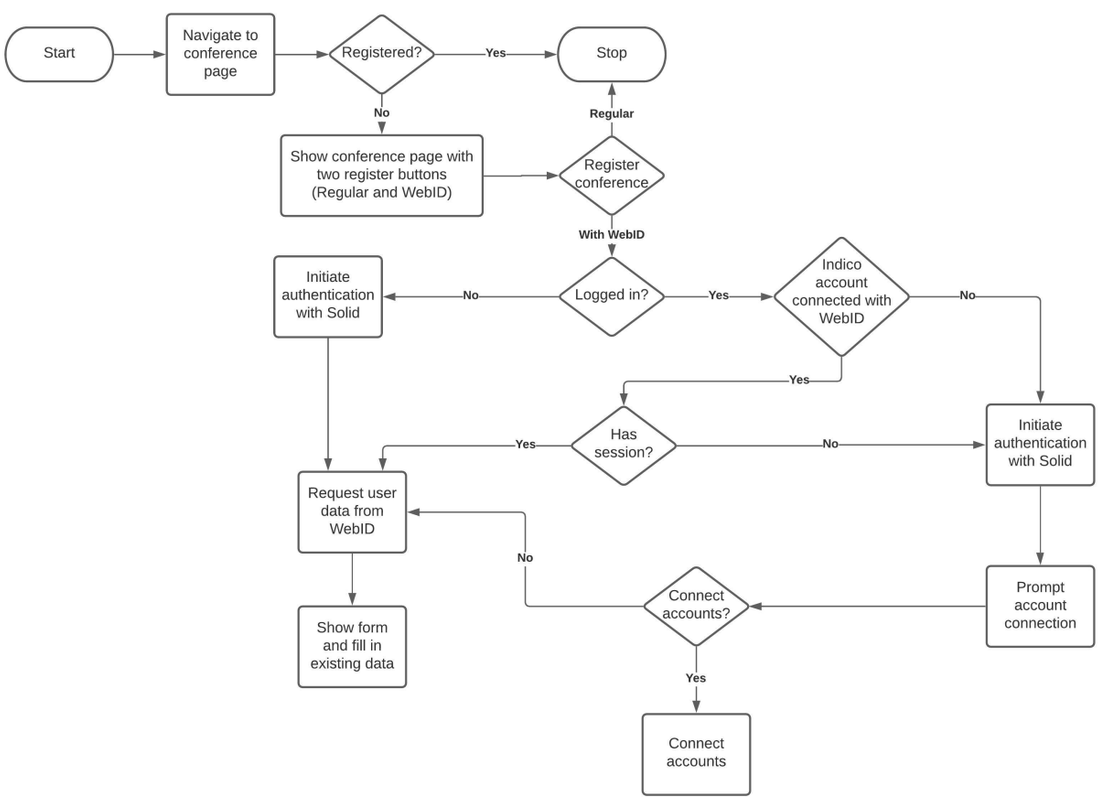


Fig. 23: TODO:

Modification of Resource From Data Pod

TODO:

Payment on Input Fields

TODO:

Performance of Large Conference

TODO:

Availability of Crucial User Data

TODO:
* performance? * Management part of Indico * Forward data between people * **Also serve data to people (admin of registration)** * Comments are more clear tied to comment?
* the flow of Solid (ref Tim from White Area) notifying for future conference * Indico does not have any RDF semantically structured data

6.3 Deployment of Indico Instance

As the development of the POCs should be tested in a running Indico instance by CERN personnel a deployment outside of the local development machine is necessary. A few prerequisites are needed to set up a staging environment. First is a proper CERN account and a running Secure Shell Protocol (SSH) tunnel to CERN's network. It is important to note the environment is also running behind CERN's firewall and can only access with the two mentioned prerequisites.

To deploy Indico with the two POCs an own hosting solution could have been chosen, but CERN has great documentation to an OpenStack virtual machine (VM) [37] workflow, which allows a straightforward deployment of Indico. OpenStack is set of open source cloud software components providing production-level infrastructure. CERN has a list of OpenStack images ready to be used for staging or production environments. A CentOS [38] distribution was chosen and the official Indico guideline to install Indico on Linux. A few caveats were met as the latest not yet release Indico version with Python 3 was used. This was decided with the Indico chief developer to test Indico and to use and have access to the latest developments. The differences to the Indico documentation which only supports previous Indico versions were not complex enough to be mentioned here, as they mostly meant installing different operating system (OS) packages.

Indico can be easily installed by building a Python wheel locally, which bundles all dependencies into a single file, and sending it to the OpenStack VM where it is then installed with a single `pip install` command. A few system calls were created from following the Indico documentation, which have to be restarted every time a new Indico version is installed.

Two additional parts are necessary to make Indico and the POC modules work properly together. A Transport Layer Security (TLS) certificate is necessary for the authentication flow with real data pods and also always a good practice to have enabled. Let's Encrypt [39] is a nonprofit Certificate Authority providing free TLS certificates and therefore a decent choice when in need for a certificate. Let's Encrypt certificates cannot be used in CERN's network as their ports are blocked by CERN's firewall. But again CERN has a setup solution for this providing their own self-signed certificates. These are pre-installed on CERN's work computers and thus trusted automatically when used in the staging instance.

The second improvement to the installation was to enable single sign-on (SSO) allowing authentication through CERN's authentication service. This means all users with a valid CERN account can authenticate with the freshly installed Indico instance and therefore provide an authenticated Indico session when needed. This was as easy as adding the domain of the Indico staging instance, installing CERN's Python Flask authentication library called *flask-multipass* [40] and appending some additional configuration to the Indico configuration file.

TODO: deployment diagram?

7 Challenges, Advantages, and Gaps of Existing Solid Solutions versus CERN Ones

* * Encryiption -> hosting on premise * Hosting on premise -> high involvement with the servers * Solid UI challenges * Commercial and open-source not perfectly hand in hand * Not many applications in Solid ecosystem * no sophisticated to say the least * the closest LiqidChat makes heavy use of an API

8 Proceedings in the CERN-Solid Collaboration

With the two prototypes developed, integrated into Indico, and deployed to a running instance and showing a working solution with Solid principles in Indico, how can CERN proceed with its collaboration attempt? In the following, the two fields of Solid specification implementations and Solid apps shall be looked at and debated what CERN's role in the future of Solid can be.

8.1 Solid Servers

The in [2] identified Solid implementations remain where they were when composing the research paper [2]. A lot of development has happened for the Community Solid Server (CSS), but it is still in a minor release version and therefore not yet deployed and switch out with NSS on the public solidcommunity.net domain. This is expected to happen as soon the developers trust the state of CSS to facilitate at least the functionality of NSS and the tooling is in place to transfer all existing data pods currently hosted on the web server. No publicly communicated date has been set in stone for this to occur but it can be assumed to happen this year or next. The Solid specification, on the other hand, has a clearly defined completion date of 30.06.21 [41]. Once the technical reports are completed – but of course remain in the state of a living standard – it can be expected to ease the development process in the Solid ecosystem, as no significant changes in the specification mean no critical new features need to be developed or supported by the server developments.

The NSS is maintained by a small team of unfunded open-source developers and can only fix critical bugs and keep the dependencies up-to-date. It is still the most used Solid implementation and recommended data pod solution in conjunction with one of the providers, namely solidcommunity.net [42].

CERN has a number of options in its proceedings with Solid servers in their current status.

1. Outstanding solution through solidcommunity.net
2. Integration with CERNBox
3. Sandboxed CSS
4. Develop own server solution

Outstanding Solution

The usage and testing of the POCs required a WebID and a data pod. Through extensive research and careful considerations, it was concluded and recommended to all POC participants to obtain the necessities through solidcommunity.net. The Solid Community is currently the most attractive data pod provider through its physical hosting in the United Kingdom (UK) and openness regarding data usage and usage of NSS the go-to open-source Solid server solution. With the exit of the UK from the European Union (EU) it seems uncertain what happens to those data residing in storage facilities in the country, which is by policy regulation for CERN not good enough, as all data need to reside in the EU, abide by principles from Operational Circular No. 11 (OC11) [43] and General Data Protection Regulation (GDPR) [44], and that all data are merely used to provide its services [45]. This is also a reason why a hosted instance through Inrupt [46] not acceptable. Not to mention the expected cost of such a service. An outstanding solution also gives away control over the running version of the Server implementation. As of now most data pod providers run the NSS, but might switch over to CSS – which is desirable but could bring new challenges. Even though a server implementation can only be called a Solid server when it adheres to the Solid specification, which can be tested by an Independent Solid Test Suite (ISTS) [47], which also means when the specification is followed, features should be equally supported and implementations should allow interoperability. This has been proven to be difficult when looking at the several browsers, which are all implementing the HTTP, HTML, and URI standards (and many more specifications) but are all behaving slightly differently. This might be due to missing resources to stay up to date with feature development, or contrasting interpretations of the specifications. These risks will always endure and hence bring challenges.

Integration With CERNBox

CERN uses CERNBox [48] to provide personal cloud storage to all CERN users to host and share files. The service is based of ownCloud [49] and hosted on CERN premise. In 2016 Nextcloud [50] was formed from a fork of the open-source core software of ownCloud. PDS Interop [51], a collective of open-source developers, has developed a Nextcloud [50] plugin to make the file hosting service Solid compatible. A Nextcloud server with the Solid plugin enabled passes currently the complete ISTS. The integration with a running Nextcloud instance is as easy as installing the plugin through the web interface of Nextcloud.

An integration with CERNBox seems to be a suitable candidate, but also brings high risk. Adding the Solid implementation into its own cloud storage infrastructure means the CERNBox administrators now also have to administer the Solid Nextcloud plugin and all the added complexity it enables. It would also be enabled for thousands of users adding even more area for possible deterioration. A possibility could be to only enable it for a subset of users to test the integration on trained personnel. These are all resources that have to be accounted for. Most importantly though it would have to be analyzed how the attack surface is increased by enabling a plugin that includes an extra sharing functionality.

Sandboxed CSS

A less risky solution would be to use a sandboxed Solid implementation, such as CSS. Once CSS is released under a major version it could be deployed to self-contained OpenStack VM instances and then run as a new file storage system. This might seem redundant, considering CERN is already running a cloud storage system with CERNBox, but the added risk is much lower to infiltrate an existing system. The new CSS deployments could solely be used for Solid related file storage and sharing and then incrementally be more integrated with existing CERN

solutions. Even though the complexity might not be as high as with the installation through the Nextcloud plugin in CERNBox it still requires administrative work to keep the deployments running and updating CSS regularly. Further, to even add more value and usability to this integration the CSS could be used with the CERN Authorization Service (CAS). Meaning, instead of identifying with an external IDP, the CERN personnel could authenticate with their existing CERN accounts using CAS, which would need to be extended to also provide WebIDs.

Open-source and community-owned software brings a lot of advantage but also needs to be tread lightly. Software as this always relies on independent developers to fix bugs, develop features, which will not happen as fast when using a product with service level agreements (SLAs). A motivator for CERN to get involved with the development of open-source Solid services.

Own Server Solution

Because Solid is an open standard and allows free development of own solutions it is always a possibility to develop a CERN Solid server from scratch. This is from all solutions the most ambitious one and least recommended, as it demands too many resources and a sophisticated solution through CSS is almost release, which would benefit from additional given resources.

8.2 Solid Applications

Uncertainties in the adoption of a CERN Solid server do not help to advocate the usage of Solid applications or even the development of new Solid services. The POCs have shown realistic use-cases for decentralized data storage. The architectural evaluation has shown the maturity, but also the flaws that would need to be tackled in order for them to be adopted into the production instance of Indico at CERN and then it also needs to be analyzed if those POCs would even find users or it would be just a dead feature. Indico has also been established to not be the perfect candidate for a complete Solid overtake in its architecture. Therefore, less complex software at CERN might be a better target for further Solid-based developments. One prospect is the CERN Slides' App [52], which is a web application recently developed as part of a student project and is in production-ready quality and planned to be used as a main slides maker for CERN users. A possibility for CERN to remain present in the Solid ecosystem is to further the development of the Slides' App and even make it Solid compatible. The final resource combining all slides into a single presentation could be stored in a data pod, and even further thought all the information on the slides could be connected to resources from data pods and thus leveraging a decentralized approach to data management. Everything would be connected, could be reused, and would be completely interoperable. Information could still come from non-RDF resources but could be transformed to RDF upon entry. Resulting in an attractive hybrid application, that could be used without the need of a data pod or WebID, but would still find users as it is a wanted software at CERN and would bring the decentralized resource management to the ones who want to use it.

8.3 Recommendation

A recommendation based on the previous two talked about fields of server implementation and application development for CERN is as follows.

The current efforts from CERN are rather limited in resources, general interest is present, but many factors hinder further active involvement with Solid. A public communication channel used to share milestones in the investigation is left unread. Announcements and call for testings of the POCs in CERN internal email groups reaching hundreds of CERN users are met with participation, compliments, and comments, but still small in number. A presentation given at the HEPiX [53] was well received by international Information Technology staff, from High Energy Physics and Nuclear Physics laboratories and institutes. So, a general curiosity exists, and a few enthusiasts are awaiting a Solid-like project to be supported by CERN, but it seems more generally that CERN is interested in an observing position as of now. This was concluded from the support given through this project's lifetime and the most recent decision of not funding through hiring for a fellowship position to work on a continuation of the CERN-Solid collaboration project.

The most likely future scenario is to work in collaboration with students from universities to work on further Solid related projects. By doing this CERN can stay closely connected to the Solid ecosystem and receive the latest developments in server implementations, specification evolution, and new exciting Solid apps. Once the Solid ecosystem has matured enough and has shown through other Solid app developments the possibilities and innovations with Solid, CERN can reevaluate its position and consider participating more actively by either Solid app development or hosting Solid pods for its personnel.

TODO: enrich Indico with RDF data

Conclusion

* Solid lives through the community, when only one company does the implementation it will not thrive * vicious cycle

References

[1] Tim Berners-Lee. *Longer Biography*. 2020. URL: https://www.w3.org/People/Berners-Lee/Longer.html. (Accessed: 08.05.2021).

[2] Jan Schill. *CERN-Solid Code Investigation: Specifications and Comparable Implementations*. English. WorkingPaper. IT University of Copenhagen, 2020.

[3] *CERN-Solid code investigation*. URL: https://it-student-projects.web.cern.ch/projects/cern-solid-code-investigation. (Accessed: 08.05.2021).

[4] CERN. *Indico*. 2020. URL: https://getindico.org. (Accessed: 11.12.2020).

[5] *Big Data and Social Media*. URL: https://indico.cern.ch/event/702278/. (Accessed: 08.05.2021).

[6] *Practical Use of XML*. URL: https://indico.cern.ch/event/420426/. (Accessed: 08.05.2021).

[7] CERN. *Learning Indico - Conference*. URL: https://learn.getindico.io/conferences/about/. (Accessed: 11.12.2020).

[8] Aaron Coburn (Inrupt), elf Pavlik, and Dmitri Zagidulin. *SOLID-OIDC*. Tech. rep. W3C, May 2021.

[9] Ashok Malhotra, Steve Speicher, and John Arwe. *Linked Data Platform 1.0*. W3C Recommendation. https://www.w3.org/TR/20ldp-20150226/. W3C, Feb. 2015.

[10] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. W3C, Feb. 2014.

[11] Eric Prud'hommeaux and Gavin Carothers. *RDF 1.1 Turtle*. W3C Recommendation. https://www.w3.org/TR/2014/REC-turtle-20140225/. W3C, Feb. 2014.

[12] *rdflib.js*. URL: https://github.com/linkeddata/rdflib.js. (Accessed: 08.05.2021).

[13] *Thing*. URL: https://docs.inrupt.com/developer-tools/javascript/client-libraries/reference/glossary/#term-Thing. (Accessed: 08.05.2021).

[14] *SolidDataset*. URL: https://docs.inrupt.com/developer-tools/javascript/client-libraries/reference/glossary/#term-SolidDataset. (Accessed: 08.05.2021).

[15] *Web Access Control (WAC)*. URL: https://solid.github.io/web-access-control-spec/. (Accessed: 08.05.2021).

[16] William Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. 3rd. USA: Prentice Hall Press, 2014. ISBN: 0133773922.

[17] *solid-app-launcher*. URL: https://github.com/ylebre/solid-app-launcher. (Accessed: 08.05.2021).

[18] *Solid JavaScript Client: solid-client*. URL: https://github.com/inrupt/solid-client-js/. (Accessed: 08.05.2021).

[19] *Solid JavaScript Authentication - solid-client-authn*. URL: https://github.com/inrupt/solid-client-authn-js/. (Accessed: 08.05.2021).

[20] *The Solid Common Vocab library for JavaScript*. URL: https://github.com/inrupt/solid-common-vocab-js. (Accessed: 08.05.2021).

[21] *rdflib.js*. URL: https://github.com/linkeddata/rdflib.js/. (Accessed: 08.05.2021).

[22] *Solid JavaScript Authentication - solid-client-authn*. URL: https://github.com/inrupt/solid-client-authn-js/. (Accessed: 08.05.2021).

[23] *A Python OpenID Connect implementation*. URL: https://github.com/OpenIDC/pyoidc/. (Accessed: 08.05.2021).

[24] D. Fett et al. *OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP)*. Tech. rep. Internet Engineering Task Force (IETF), Apr. 2021.

[25] *New user FAQ: Tweeting*. URL: https://help.twitter.com/en/new-user-faq. (Accessed: 08.05.2021).

[26] *PostgreSQL: The World's Most Advanced Open Source Relational Database*. URL: https://www.postgresql.org. (Accessed: 08.05.2021).

[27] *Promise based HTTP client for the browser and node.js*. URL: https://axios-http.com/. (Accessed: 08.05.2021).

[28] Henrik Bærbak Christensen, Klaus Marius Hansen, and Bo Lindstrøm. *aSQA: Architectural Software Quality Assurance: Software Architecture at Work - Technical Report 5:* English. WorkingPaper. Department of Computer Science, Aarhus University, 2010.

[29] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (3nd Edition)*. Addison-Wesley Professional, 2012.

[30] Sarven Capadisli et al. *The Solid Ecosystem*. Tech. rep. W3C Solid Community Group, Dec. 2020.

[31] R. Fielding, M. Nottingham, and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Caching*. Tech. rep. Internet Engineering Task Force (IETF), June 2014.

[32] WHATWG community. *HTML Standard*. Tech. rep. WHATWG community, Apr. 2021.

[33] *Schema.org*. URL: https://schema.org/. (Accessed: 08.05.2021).

[34] S. Perreault Viagenie. *vCard Format Specification*. Tech. rep. Internet Engineering Task Force (IETF), Aug. 2011.

[35] *AngularJS*. URL: https://angularjs.org/. (Accessed: 08.05.2021).

[36] WHATWG community. *DOM Standard*. Tech. rep. WHATWG community, Apr. 2021.

[37] *OpenStack*. URL: https://www.openstack.org/. (Accessed: 08.05.2021).

[38] *The CentOS Project*. URL: https://www.centos.org/. (Accessed: 08.05.2021).

[39] *Let's Encrypt*. URL: https://letsencrypt.org/. (Accessed: 08.05.2021).

[40] *Flask-Multipass*. URL: https://github.com/indico/flask-multipass. (Accessed: 08.05.2021).

[41] *Solid Technical Reports*. URL: https://solidproject.org/TR. (Accessed: 08.05.2021).

[42] *Solid Community*. URL: https://solidcommunity.net. (Accessed: 08.05.2021).

[43] CERN Human Resource Department. "THE PROCESSING OF PERSONAL DATA AT CERN". In: *OPERATIONAL CIRCULAR NO.11* (2019). URL: https://cds.cern.ch/record/2651311/files/Circ_Op_Angl_No11_Rev0.pdf.

[44] *General Data Protection Regulation*. URL: https://gdpr-info.eu. (Accessed: 08.05.2021).

[45] Maria Dimou. *Policy for a CERN Solid server*. URL: https://codimd.web.cern.ch/s/zl3yGAfYZ#. (Accessed: 08.05.2021).
[46] *Inrupt*. URL: https://inrupt.com/. (Accessed: 08.05.2021).
[47] *Independent Test Suite for Solid*. URL: https://github.com/solid/test-suite/. (Accessed: 08.05.2021).
[48] *CERNBox Service*. URL: https://information-technology.web.cern.ch/services/cernbox-service. (Accessed: 08.05.2021).
[49] *ownCloud*. URL: https://owncloud.com/. (Accessed: 08.05.2021).
[50] *Nextcloud*. URL: https://nextcloud.com/. (Accessed: 08.05.2021).
[51] *PDS Interop*. URL: https://pdsinterop.org/. (Accessed: 08.05.2021).
[52] *CERN Slides' App*. URL: https://slides.docs.cern.ch. (Accessed: 08.05.2021).
[53] *HEPiX Online*. URL: https://www.hepix.org/. (Accessed: 08.05.2021).