CERN-Solid Code Investigation

Proof of Concept, Challenges, and Continuation

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

Course code: KISPECI1SE

IT UNIVERSITY OF COPENHAGEN

Computer Science
IT University of Copenhagen
Denmark
01.06.2021

Abstract. Tim Berners-Lee invented the Web to distribute information freely at CERN. The undeniable growth and dependence have created a data exploitation dilemma. Solid is a set of technical reports, an ecosystem, and a movement to regain data sovereignty and empower the users of the Web. CERN defined a project to investigate Solid and how CERN can be involved. The work presented in this paper is tackling part of this project, where the Solid principles will be applied to software at CERN in an experimental approach. Relevant software at CERN and in Solid had been identified and analyzed. Two Solid apps were developed, evaluated, and analyzed. The work concluded that Solid development is challenging in fields of *Performance* and *Usability*. When every user stores their data, it is difficult for traditional applications to retrieve the data from hundreds of users promptly. Flows of granting access to sensitive information are also still complex and confusing. Solid is maturing but still requires a lot of work and expertise from many professions. CERN is in an ideal situation to extend its collaboration by integrating the decentralized storage into their infrastructure and migrating or creating new applications with Solid principles in mind.

Acknowledgment

I want to express my sincere gratitude to my supervisor from CERN, **Maria Dimou**. Over the last nine months, I have learned beyond the content of this thesis; I would be incapable of putting the insights from these attainments into writing. I have enjoyed our weekly supervision meetings, which had often drifted from the planned agenda into exchanges over life beyond our solar system, the philosophy of ancient Greeks, intuitionism in mathematics, challenges in learning the German language, political decisions in tumultuous times, and much more. Thank you for teaching me so much; I will miss the weekly chat undoubtedly, but I am sure it is no goodbye.

I had the great pleasure to work with **Michiel De Jong** for the longevity of this project. He has given me every needed resource to understand Solid and be part of the community. His deep understanding of the Web, political values, and immense motivation to build purposeful software for everyone is admirable and has become a role model to me.

Adrian Mönnich, chief developer of Indico at CERN, provided me with answers to every possible question I could have asked in the context of Indico, technicalities at CERN, or general advice in Web engineering. Thank you for listening and giving me all the information I needed for this project.

And one more expression of gratitude towards **Sir Tim Berners-Lee** for giving input on the possible scope developing the prototypes, showing curiosity towards my implementations, and providing feedback several times. It has been a great honor working so closely and on a project as impactful as Solid.

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 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 it. Data are harvested and analyzed to generate models describing and predicting human behavior far beyond what is morally tolerable. These actions permit large amounts of wealth as capitalistic and governmental bodies enjoy a great interest in these models to generate more profit or more control. The escalation of attraction towards data has led to the construction of so-called *data silos*. Data gatherers built attractive applications for users while locking them into their walled gardens to have all the data for themselves and then sell it.

Through whistle-blowing, active journalism, and technical education, a new wave of hope arises 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, they founded the idea of Solid.

Solid aims to return data control to the users. To achieve full data sovereignty, it specifies a new way of building applications on the Web through several technical reports. That lets the users control their data and can decide who can see what. Like the Web, which was also defined by technical reports and prototypes, it is for everyone and is 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

This paper tackles a portion of a project defined at CERN. **The project in its entirety investigates how two architectural contrasting systems can be combined or how a decentralized software ecosystem can integrate into a centralized one**. The two systems are Indico, which acts in this context as the traditional and centralized software monolith, and Solid, the decentralized ecosystem, which decouples authentication, application, and data from each other. The first part of this *CERN-Solid Code Investigation* project [3] has been done in a previous *research project* [2], where the two entities CERN and Solid were introduced, and an overview of both ecosystems was given. Further, the paper identified software at CERN and generally analyzed it based on their quality and compatibility for integration with Solid. The paper also looked at the Solid ecosystem and how the concept of data sovereignty is technically translated.

The following steps in the project are presented in this body of work. In an experimental approach, two Solid modules are developed and embedded into CERN's Indico system. The adopted design of the modules is critically put into perspective with the many possibilities met, and its architecture is thoroughly evaluated. Challenges, advantages, and gaps in programming and integration, and existing solutions are described and documented. A possible continuation of how CERN can be involved in the Solid ecosystem and the fate of the Web is debated and a recommendation put in place.

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 [3] aims at linking CERN with Solid. This linkage has been done, forming 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

Indico being an event management tool, one of its core entities is the *Event*. The *Event's* user interface (UI) is presented in 1, showing all sorts of attachable information. It gives a place for all relevant information surrounding an event.

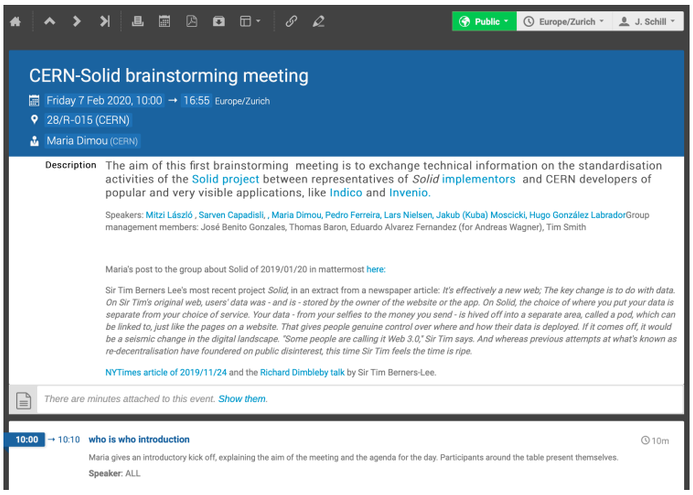


Fig. 1: UI of an Indico event.

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 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, in-

cluding 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

An 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, 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 recently. 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 are similar 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 indicates 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 Linked Data information on the Web [10]. The default file format implemented by the existing Solid servers is *Turtle* [11].

Graph-based data models such as RDF require libraries to work with them, as the extraction of data out of a graph model is not as trivial as with JavaScript Object Notation (JSON), where the dot notation can easily pull out a value of a key.

Fortunately, existing libraries come to the rescue allowing such operations on the RDF-based data type. Again Inrupt and the Solid Community offer solutions. For the same reasons, it was decided to use Inrupt's solution 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 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`, 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 to use helper methods to extract data from a Turtle file loaded by sending a request to the WebID profile document Uniform Resource Identifier (URI).

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

```
// Import statements omitted for this demonstration
// 1.
const myDataset = await getSolidDataset(
  "https://janschill.net/profile/card"
);
// 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 that the WebID profile document URI is being loaded, referring to the document describing the agent behind the WebID URI.
2. **Loading triplet statement into a 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. When evaluated, the predicate `VCARD.fn` returns a URI, which on dereferencing describes what kind of value can be obtained from the object.

5.3 Authorization Through WAC

The Solid architects chose Web Access Control (WAC) [15] as the authorization mechanism on the Solid data pod. 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 include 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 finding out what a particular user has access to [16].

5.4 Application Launcher

The Solid authentication flow has a design inconvenience, that only can only scope to the root container when handing out the access control to an asking application. The flow is initiated when a client wants to use an application and this application needs to be trusted to interact with the data pod, most often, a the *control* mode 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 an *application launcher* can avoid this problem. When an application wants to create ACLs on a data pod, it needs the before-mentioned *control* access. *Control* access is the highest access mode in WAC. An agent with *control* access can read, write, and modify ACL resources – meaning the owner of files can be changed as an example. In the initial Solid OIDC flow, one step asks for the application's allowed access scope. 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 use 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. Launching an application like 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 JavaScript (JS) exists here [17].

Investigation

6 Proof of Concepts

One central 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 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 Solid-based content. The product owner and chief developer of Indico, the CERN-Solid project manager, and a Solid developer 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 wholly 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 system's scope and 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 requires an input field to receive user input, which can then be transformed to structured data in Linked Data and sent to the data pod. On an invalid session or wrong input, the module should appropriately tell the user to either log in or try again. 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 updated 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 that gives users the ability to authenticate themselves using their WebID, store the created data in the shape of comments in their data pod, and 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.

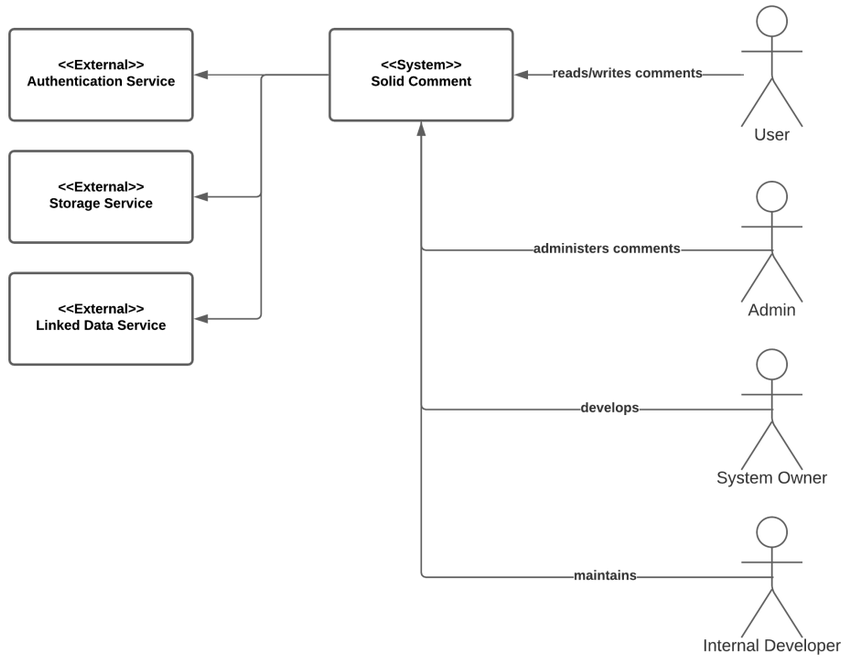


Fig. 2: Context diagram showing users and external services of the system.

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 design.

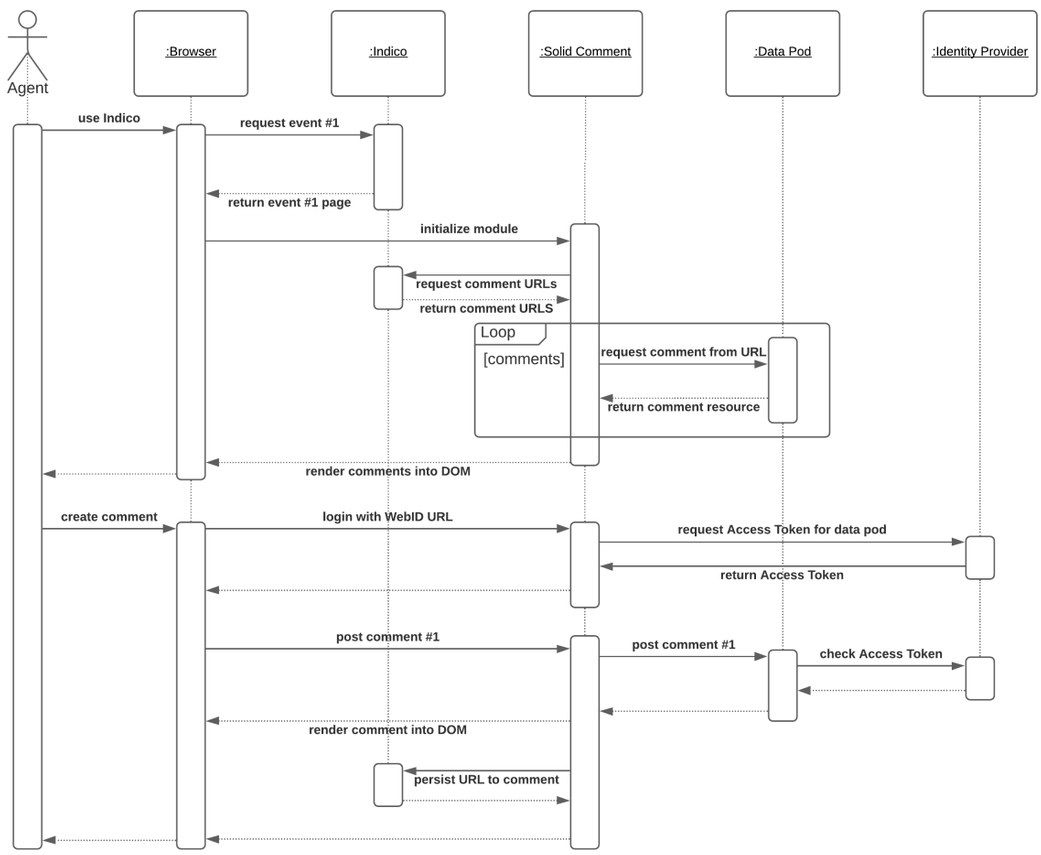


Fig. 3: Sequence diagram showing the sequential process through posting a comment.

Stakeholders

This section covers the various stakeholders concerning the system. The stakeholders include active users and various external and internal stakeholders impacted by the system or who have a significant say in the development process.

The **product owner** is responsible for the product, in this case, Indico. Usually, they plan repeated fixed timeboxes 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 not 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 daily and can make assumptions about the evolution of the software. Their concerns lie in the maintainability of the software through its evolutionary cycle and onboarding new developers.

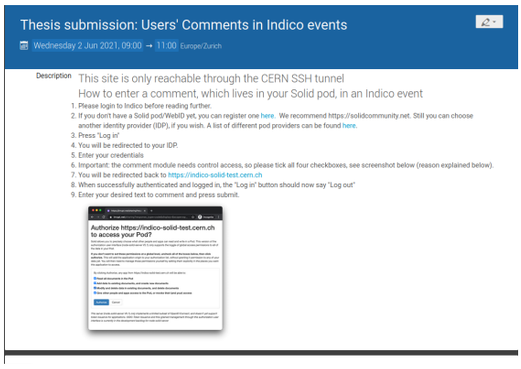
Drivers

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 of utmost priority to the chief developer and product owner. For the collaboration manager, having used Solid for a while, it was clear that the module is usable, requires as little effort as possible, and causes the lowest friction. Making the *Usability* also an important QA to focus on. Therefore, the following QAs will be the driving motivation:

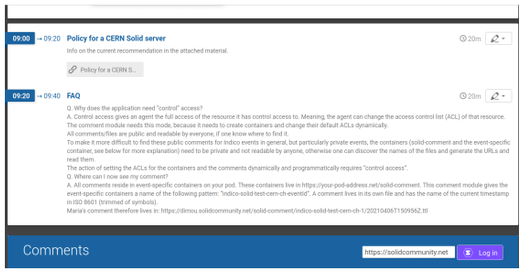
1. Security
2. Performance
3. Usability

6.1.2 User Interface

For various held meetings and presentations, a visual representation of the to-be-developed module helps guide the audience through the process. For the comment module, a UI 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) UI showing the comment module: Description.



(b) UI showing the comment module: FAQ.

Fig. 4: UI 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.

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. Dynamic rendering is useful when new data is shown right away without a 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 components 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 specified requirements in place, it

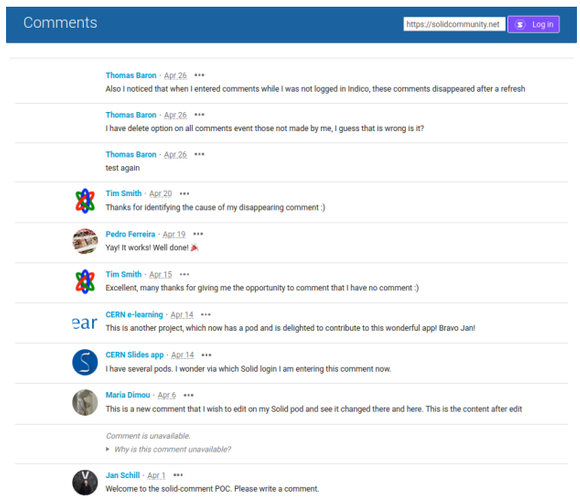


Fig. 5: UI showing the comment module: Comments.

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 that compiles to JS, such as TS, is also possible. A browser solution means Indico remains primarily 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 [12]

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.

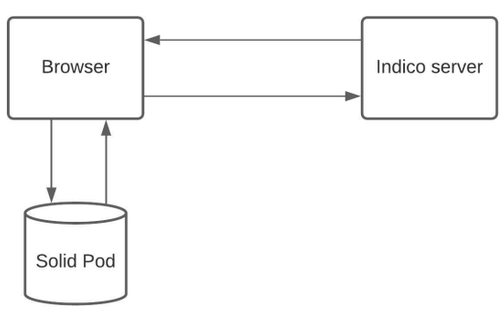


Fig. 6: Communication flow for a module developed on the client.

Microservice Approach

The microservice approach would allow developing the needed Solid logic on a separate service, which proxies all Solid related traffic 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 [21]
RDF	solid-common-vocab-js [20], rdflib.js [12]

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

The microservice module would handle 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.

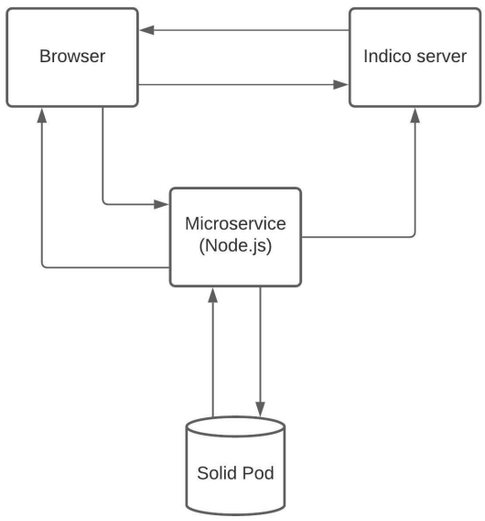


Fig. 7: Communication flow for a module developed as a microservice.

Server Approach

The goal of the server approach would be just like with the microservice system 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 that it could be fully integrated within Indico and be part of its Python codebase. Significant drawbacks are that no Solid libraries written in Python exist to allow a seamless integration into the ecosystem.

Problem	Solution
Language	Python
Framework	Flask
Client	-
Authentication	pyoidc [22] missing DPoP
RDF	solid-common-vocab-js [20], rdflib.js [12]

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 request protected resources on a data pod.

DPoP is a technique to disallow replay attacks by attaching a DPoP token to the request indicating what data pod the token can use. In a replay attack, an adversary intercepts the authentication token of an agent and uses the token to make authenticated requests to protected resources. With the usage of DPoP tokens, the authentication token is signed with a target address. The target address defines the only valid address for which the token can be used. Meaning, when a request is made to a malicious pod, the interceptable token in the Hypertext Transfer Protocol (HTTP) header can only be used to make requests to this data pod and not to any other pods. The DPoP token therefore needs to be newly generated on every request [23].

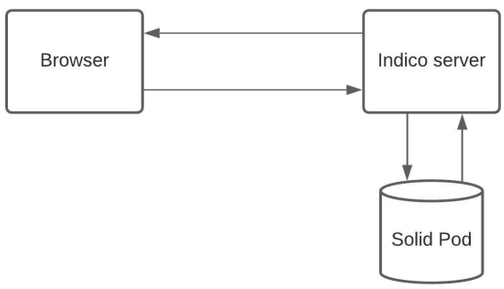


Fig. 8: Communication flow for a module developed on the server.

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: holding 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 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 following paragraph also lays out how this design would not work with the protection of the resources.

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

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

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

Protection on Resource

Every container and resource in Solid is protected with WAC, determining 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 authorization is needed to see the event. At the same time, private can be as fine-grained as only to specific agents or groups. A comment module is only valuable if anyone can read and be written by authorized users.

For visitors of a private or public event in Indico to see the comment, the comment's ACL needs to allow the public to read the resource. Public read 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 description even allows reading the container's content, meaning a request to the container would yield a list of resources in the container. A publicly readable list becomes unpleasant if the Indico event is private. The comments for this Indico event should not be read by the world, which is entirely possible when browsing to a specific data pod location and then looking into the public container.

An ACL defining private access to a container can prevent the problem of a random agent seeing a container's content, with the container's resources still public. The private access mode would allow everyone, provided they have the Uniform Resource Locator (URL), to read a resource in a private container but not look into the resource's parent container. To achieve this behavior with ACL, the container needs to 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 specified:

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 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 proper access modes for the resources in place, but still publicly readable. Anyone with the URI to the resource could read it. 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, because the resource names could be guessed or iterated – 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 would require some iterative indication in the filename. A combination of randomness and an order indicator is a good solution. It was settled for using universally unique identifier (UUID) prepended with an ISO 8601 string to form a filename.

Other ideas include 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 control their data, they can revoke access or even change their data to their liking. The ability to modify data from within the data pod means the application cannot be guaranteed the same comment that was initially created and needs to be aware that data can be changed at any time. Even if the system's interface does not allow modification, a modification can still happen at the source of the storage of the comment.

Regular comment modules usually allow updates of a submitted text throughout its existence. Twitter is an example where an update is not allowed after posting [24].

When the development discovered this aspect, the relevant stakeholders and developer decided to allow this behavior, as it seems natural to edit one's comments.

Mitigation of Spam

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

Giving Application Full Control of Data Pod

An agent or application requires *control* access to the container to create or change ACLs programmatically in this 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. Giving an application control access allows the application to read, write, change ACLs on the entire data pod. As a simple application, this is troubling as a commenting module needs to have control access to set the necessary ACLs for the containers and resources it creates.

The current implementation has not a built-in solution. Still, one way of solving it is using an application launcher, an application itself with complete control access, and then limiting the access controls of the solid-comment module by creating a dedicated container for it and setting the needed ACL for it 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 posted on a specific event.

Storing Reference to Comments in Indico

Indico operates using the relational database PostgreSQL [25]. When a comment gets posted and stored on an external data pod, a reference to the location of the comment needs to be kept for Indico to pull the comment and render it in its frontend. Several possibilities are imaginable.

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

One solution would be to create a new database table that associates with the event using a foreign key and the event id and then stores every comment in its row with all necessary meta information. The database schema would have to be updated 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 Indico Solid data pod. The data pod could hold a resource for every event and link to the data pods where the comments reside. This solution is the most aligned with the Solid principles and could have been an interesting challenge to develop, but it was identified too late to 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 primary motivation was to keep it as laborious as possible for spam to reach the system.

The awkwardness of validating a current Indico session and then sending requests to a protected endpoint comes with the great decoupling of the comment module and Indico, which is admirable and makes this feature more challenging to implement. Indico uses a framework called Axios [26] for its 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 be extended to allow a custom HTTP client to send 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 systems 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. The evaluation shall be done by iterating through the stakeholders' requirements and how the architecture lives up to those expectations. The assessment is done with the help of the architectural Software Quality Assurance (aSQA) framework to ensure continuous quality assessment and prioritizing with a lightweight technique [27]. aSQA contains seven process steps, as shown in figure 10.

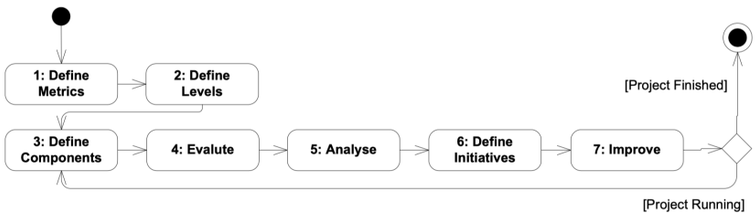


Fig. 10: aSQA Process Steps from [27]

1. **Define Metrics** to the components of the system. This can be done with QA scenarios (QASs), which 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 helpful 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 is QASs. QASs can be used to characterize QAs [28]. 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 will stress the design 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 only to give access to the necessary data

Levels

The levels are used to measure the state of the architecture of the system. These can be freely chosen but follow an ordinal scale ranging from 1 to 5 in the original paper and adapted [27]. The levels will help rate the quality of the system's architecture, where one is the lowest quality and five the highest rating.

Components

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

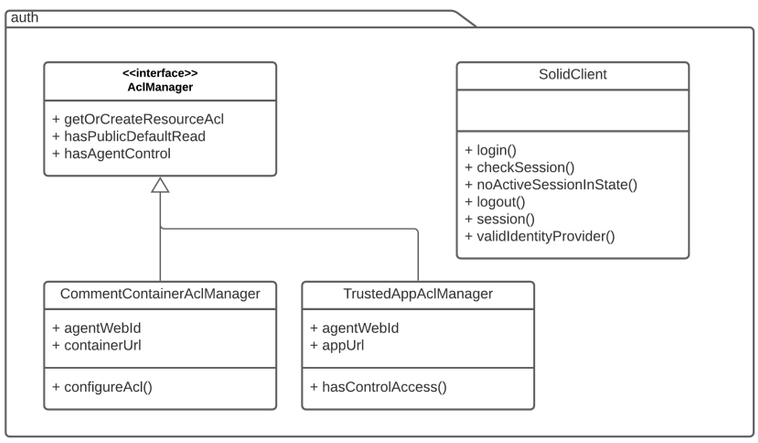


Fig. 11: The `auth` package with its classes.

The `auth` package holds classes handling the ACL logic necessary. The logic ranges from checking for existing specific authorization modes or creating new ACL files for resources and updating existing resources. A careless architecture could either give too much access control or allow 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.

The second package, `components` 12, holds all elements in the UI 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 most considerable quality problems?
- **Importance**: How important is it to move from current to target level?
- **Focus**: What are the most significant and most essential quality problems?
- **Valid level values**: 1, 2, 3, 4, 5 (higher equals better)

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

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

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

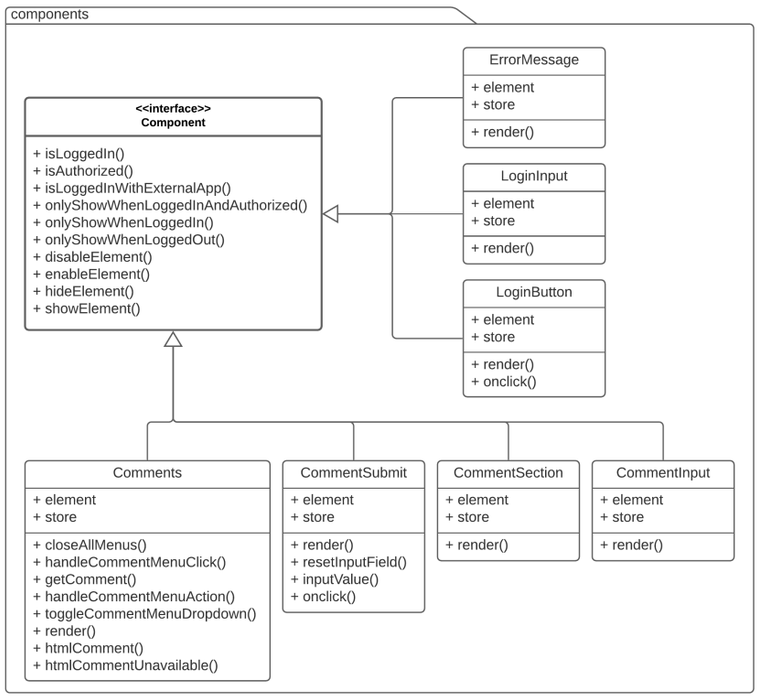


Fig. 12: The `components` package with its classes.

1. A user logs in with their WebID and posts a comment
2. An adversary posts a comment with a string containing an 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, that highlight critical areas in the architecture 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 boots with the flow shown in 13. Every time the module is loaded, it has to make four round trips to the IDP and data pod. Once the application is booted, the list of comment URIs is 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 regarding the fetching of the comments are being done below in paragraph 6.1.6. However, when the comments are being fetched, a second request loads the author's WebID profile. Resulting in $n * 2$ requests, where n is the number of comments. The exact flow of getting the comments is shown in figure 14.

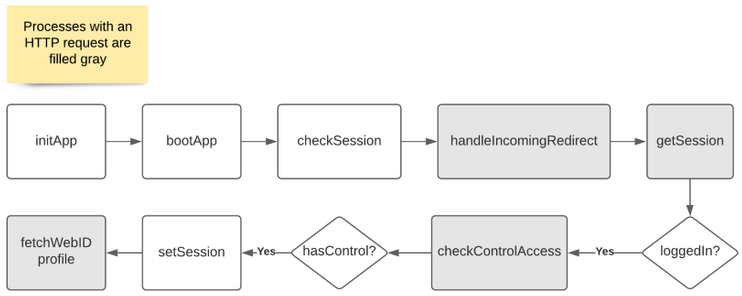


Fig. 13: Application boot flow. A filled grey box indicates external HTTP requests.

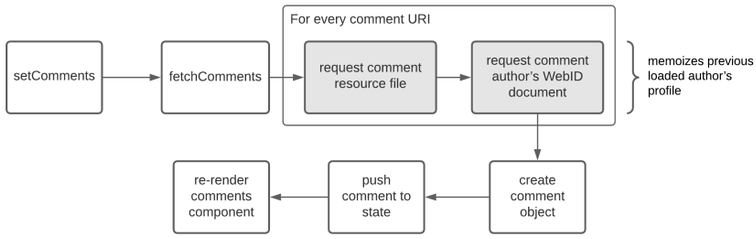


Fig. 14: Requesting comments from data pod. A filled grey box indicates external HTTP requests.

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

Regarding *Availability* for both modules, if a data pod is unavailable, the `auth` module will not be able to do check whether a session is present. The HTTP requests containing the session will respond with $40x$ HTTP status codes and disallow any comment posting. A dedicated message has been implemented for the `components` module, which will hint at 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.

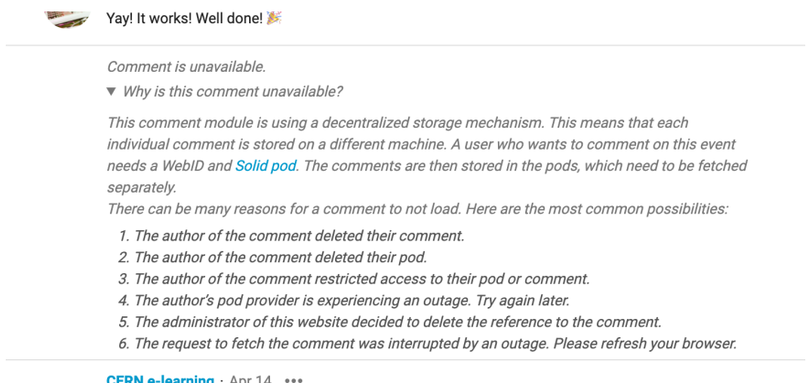


Fig. 15: Hint when a comment could not be loaded

Another striking value is the high focus on *Usability* in the `auth` component. This can be explained by the fact that it is not apparent to the user that *control* access to the root container of the data pod is needed for the application to function correctly. Only when an authenticated session from the Solid IDP is present, and the *control* access lacks will it show in the interface. Firstly, this results in a lousy user experience, but moreover, 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 an application that needs this much control. Therefore, the evaluation for the security resulted, 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 a 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 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 will all be fetched, and only when all requests are made will the application render the comments in the DOM. If the comments are *separated* from each other, and a comment is rendered as soon as it is successfully fetched, it would speed up until a user could start seeing comments.

An additional 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 improve 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 relies on the Server implementations. By Solid specification, HTTP caching is prescribed but not vital [29], 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 [30]. A real improvement on consecutive page visits is won by client-side HTTP caching. Upon successive visits to a cached website, a previously fetched and browser's local storage stored copy of the response would be served [30]. The result would be immediately loaded but might not serve the latest and up-to-date asset from the server. The response's *freshness* is controlled using the `Cache-Control` header in either the request or response of the HTTP exchange between client and server [30].

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 that 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 [29]. A user can post comments freely through this access mode but cannot change them later, as the data is now in Indico's data pod, and the user lacks 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. A drawback to this approach is that the comments are again stored centralized in one storage. However, because it uses a Solid server instead of a web server with unstructured data storage, the data is now structured and interoperable through *Linked Data*. Besides storing it in Indico's data pod, the author's data pod could keep a copy of the comment as well. 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 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 essential to associate the comment's URL with the hash to make a proper comparison of the correct resources.

A vital aspect of allowing the update of existing resources is to indicate the UI 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 discussion, an updated comment could be the reason. An indication would help to identify such a situation. The same hashing function can detect a change in the comment's content.

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. 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 generates 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 testing of the prototype. It was never expected to test the prototype where these race conditions would occur. For a production-ready module, a separated database table is necessary. Every new comment URI would be added to the database using SQL `INSERT` statements, 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 detect changes in pulled-in comments.

Indico Solid Proxying

Indico could develop an API that sits between Indico and the Solid data pods to make all the requests on behalf of the clients. A proxy would hide all the clients' IP addresses and 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 cache the response and serve them from one request. By caching the requests to the data pods in this layer, a client only needs to make a single request to the Indico's proxy cache. Loading resources before they are requested by clients and then serving clients from a cache is called *cache warming* [31]. 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 targeted advertisement, cannot identify the clients by their IP addresses and therefore not send different responses upon requests from separate IP addresses based on geolocation.

This proxy equals the *microservice* architecture described in 6.1.3.

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 the 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 owner's hand 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 the storage of practical information in a data pod. The user has full control and can handle the data to their liking. The stakeholders and developer decided that storage in a data pod was not viable for the prototype. Consequently, the prototype changed to allow the extraction of data from a data pod and then use it to map it to input fields of an Indico conference registration form – the functionality to then store the data from the registration in the user's data pod, 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 *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, fills in missing inputs, or asks the user if it should replace the existing values.

Features

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

1. A user can provide 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 to populate a conference registration.

Type of Users

There is one type of user for this system: 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.

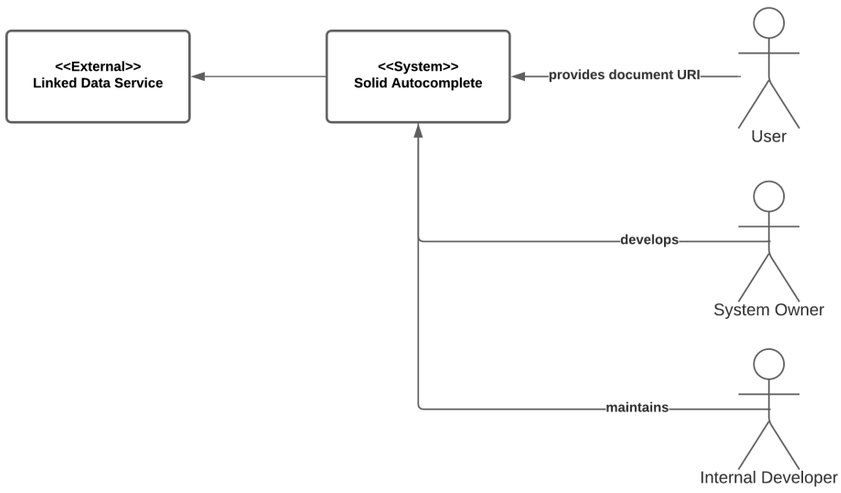


Fig. 16: Context diagram showing users and external services of the system.

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.

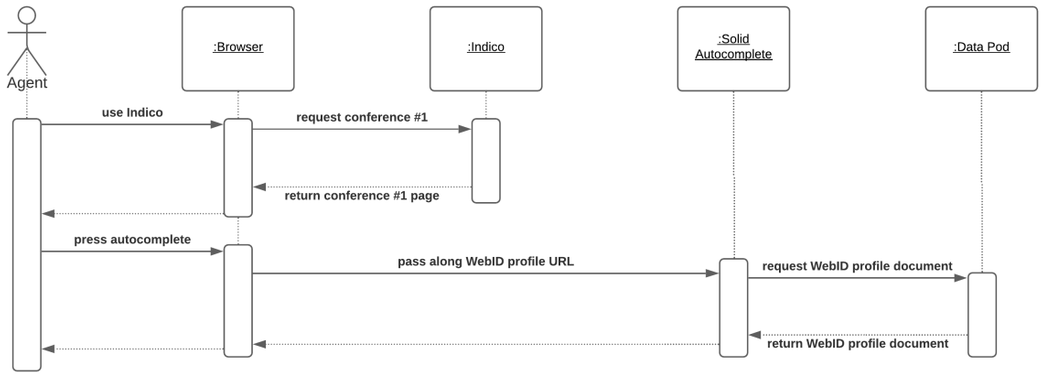


Fig. 17: Sequence diagram showing the sequential process through pulling in data from a data pod.

Stakeholders

Both modules were established with the same group of people and therefore carried the same set of stakeholders. The stakeholders defined in the section of the first POC can be found here 6.1.1.

Drivers

The drivers for this architecture endure the same as well. A system that 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 essential, as conferences tend to open their registration at a particular 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. Software design should be as simple as possible for the user to interact with, and ideally, a user should not even notice a difference from traditional practices.

1. Security
2. Performance
3. Usability

6.2.2 User Interface

The UI for this module is moderately limited as it only requires an input field for the user to provide the URI of where to find an RDF document. The RDF document fills in the registration form, an action button, and then a design for when two values on input are probable.

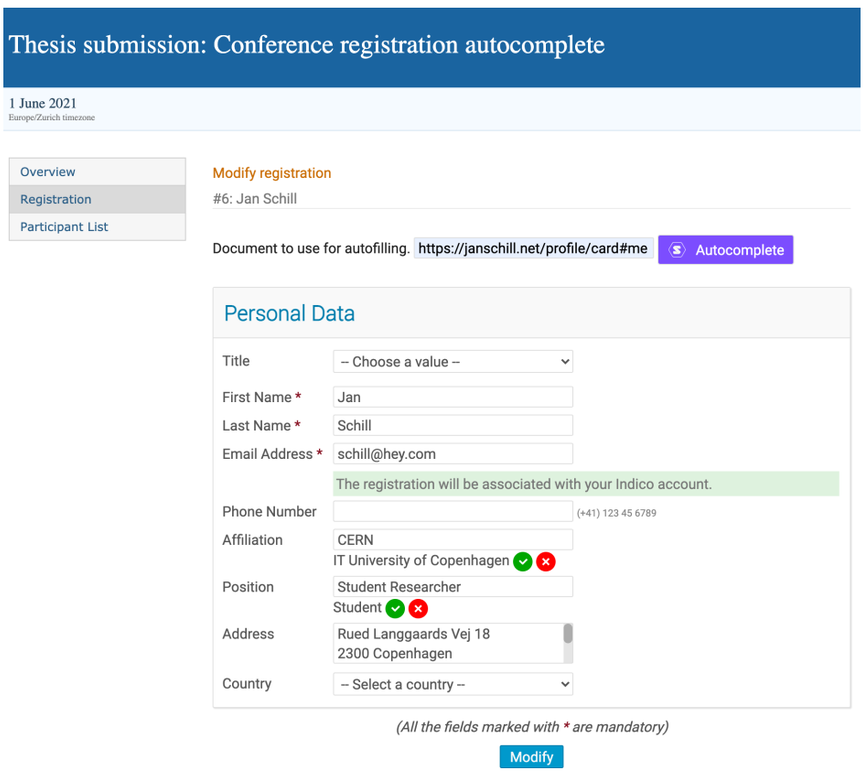


Fig. 18: UI showing the autocomplete module.

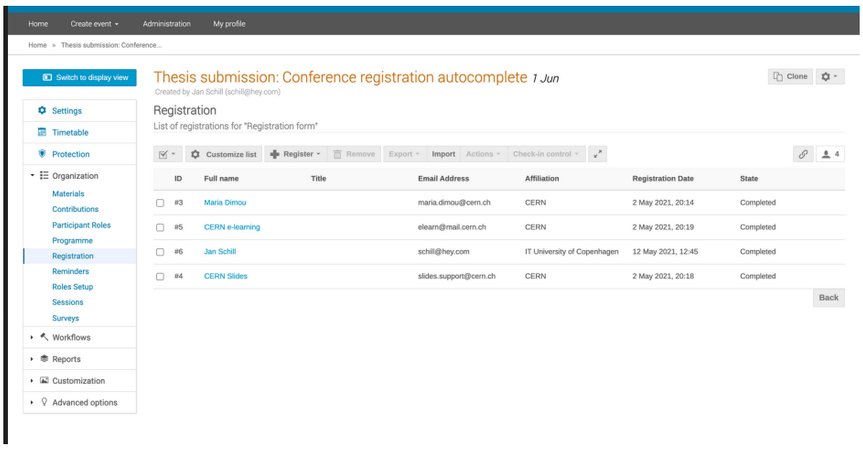


Fig. 19: The Indico control panel showing registered users.

6.2.3 Design

The following section will shed light on the design decision made in sketching out the module's architecture and developing the system. It will introduce the different challenges and how they were overcome. Primarily, it will explain what led to the change of core functionality of the module. In the next 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. Linked Data means the data uses shared vocabularies to be described and reasoned about by machines and can be moved from one system into another and seamlessly integrated. Indico does not use any standard RDF features and can therefore not extract any of those benefits. Because of this, no straightforward 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 descriptive indicator
2. Parse and process existing values of attributes and labels

In the HTML specification, an `autocomplete` attribute exists, which browsers use 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 suggests cached values when the user clicks into an input. This functionality 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 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 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 the label node's value can be extracted and mapped against a dictionary of key-value pairs to determine what the input is supposed to be filled.

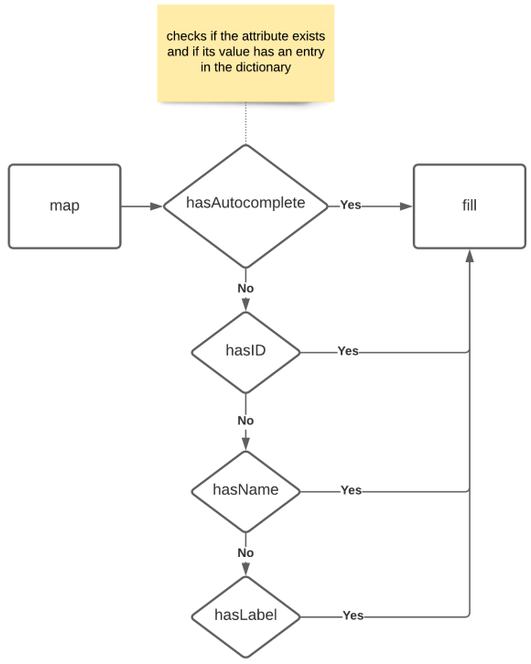


Fig. 20: The flow to find a source for dictionary mapping.

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 is the possibly extracted values from the different techniques described above. The value is in this example the vCARD [34] equivalent. NSS uses vCARD to convey 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,
  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 is that the form is not being rendered on the server and then served 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]. A dynamically rendered form introduces the challenge of interacting with it using JS. Frontend libraries that either create new or modify existing DOM nodes need to wait until the complete DOM tree is loaded. 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 to operate on the form, which is necessary for the module to read the form inputs and labels and set their values. The JS code from this prototype needs to know when the browser rendered the form successfully. 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 a `MutationObserver` instance might add performance degradation to the page load [36]. The performance shall not be analyzed more carefully as it is not relevant to prove the prototype's realization.

The `MutationObserver` is initialized as soon as the DOM is rendered; it then takes a node as a target to observe and registers all modifications on this node: the creation of children nodes in it, updates to the node itself, or any other modifications. In order to reduce computation on nodes that do not need to be observed, a suitable DOM element is needed. This node needs to exist when the DOM is loaded and needs to contain the final form node. The finally rendered HTML form has an ID. The `MutationObserver` can use this ID to grab the target node and scope the observations.

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. Basic validation checks for the presence of values in required inputs. This way, Indico can render a hint on the input fields with invalid values, for example, when no input was given. 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 notice 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 render a missing value hint upon submission.

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 be more time-consuming than anticipated and is therefore not viable; it is also unknown if the Indico development team wants a change 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. The problem with the used `oninput` event is that it does not see 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 listened to by the Indico validation. The event dispatch needs to happen as soon the values are set within the module, and because this occurs in the module and the module is a self-contained module without any knowledge of Indico, the module should not be directly changed but instead, 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

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 are motivated for the same reasons as before.

Metrics

The previously picked QAs are as follows:

1. Security
2. Performance
3. Usability

The QASs to make the QAs measurable allowing the analysis of health in the system's architecture:

1. As a user, I expect to see the information from my WebID profile document when using the module
2. As a user, I expect to have the choice of two available values
3. As a user, I expect to be incapable of pulling in protected resources without authenticating myself

Levels

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

Components

This module is compared to the other module, relatively small, and does not contain multiple packages. It forms through the three major classes `Picker`, `Mapper`, and `Filler`. These three classes find all relevant input fields from the application; secondly, maps the input fields with the available data fetched using the `DataFetcher` class; thirdly, it fills the available information through the mapping into the input fields.

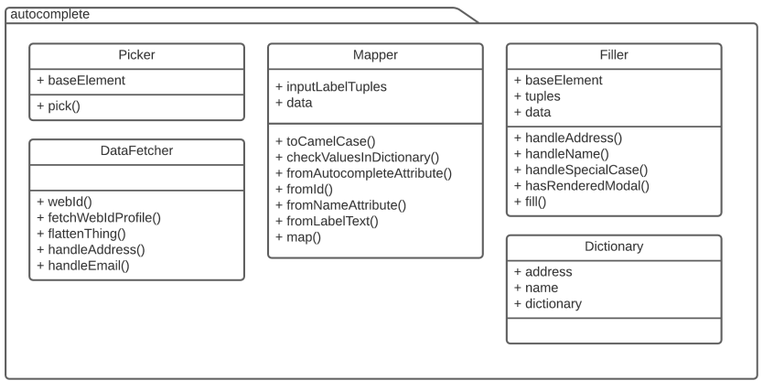


Fig. 21: The classes in the autocomplete module.

Evaluate

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

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

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

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

1. A user enters their WebID URI and presses *autocomplete*
2. A user enters someone else's WebID URI and presses *autocomplete*
3. A user enters a URI with a protected resource and presses *autocomplete*
4. An adversary enters a URI to a malicious document

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

Table 7: Evaluation of the autocomplete module based on the QASs

6.2.6 Analysis

The analysis of the second POC will explain the evaluation matrix 7 and then 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. The analysis will also look at what role Solid's design played in the abandonment of storing the registration data in data pods and effectively giving up usage control.

Because the module does not use any authentication libraries to allow users to log in or store any data in a data pod, it simply fetches public documents. The user interacting with the module decides what to fetch; no risk is involved. Regarding pulling in malicious documents from a trustworthy thought source, the module is embedded into Indico, which has validations and makes sure no malicious code is injected into their system. The performance of the module is also in good shape. Only one request happens when the system is doing its tasks. The dictionary for the mapping of values uses the JS `Object` data type. It allows constant $\mathcal{O}(1)$ look-ups of values. See listing 3.6 to see an example data store of values. One area of improvement for the performance is the `MutationObserver` that was explained in paragraph 6.2.4. Indico throws an event when all dynamically created DOM elements have been rendered, perfect for the module's binding to the Indico form. It was only later noticed that this event exists.

Iterations

As mentioned before, this module went through two iterations of the design. Until creating the architecture of the actual module, it was undiscovered that the development of a module that sits on the registration form and allows decentralized storage is not viable. Figure 22 shows the same flow through the conference registration with Solid. It checks if the user wants to register regularly and let Indico have and handle the registration data or if the user wants to use their WebID and thus store the data in their data pod for maximum usage control.

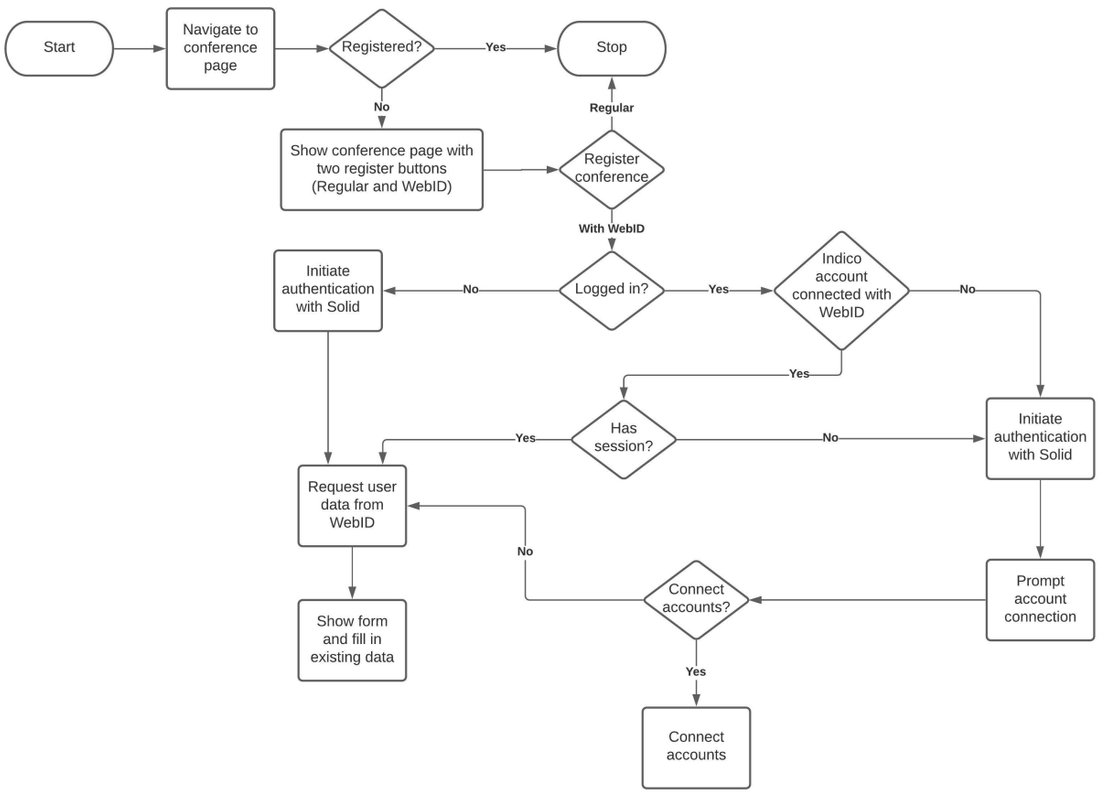


Fig. 22: Flow through the registration flow using Solid account.

A few challenges were unidentified at this point but will be named and explained now.

Availability or Modification of Crucial Data From Data Pod

As also realized in the first POC, when the application is decoupled from the data and receives it upon request, the application hands over the control. This shift of power is a core concept in Solid but problematic for Indico. The use-case of registration data differs with the handling of comments. Comments are often desired to be changed, and primarily comments are just presented in the UI, and no other interaction or processing happens. Registration data is much more complex and requires a lot of processing, such as submitting abstracts. *Call for abstracts* is a workflow in Indico, where conference speakers need to upload an abstract about the conference talk. Another is the requirement for processing in the field of payments in the registration. A registration form can offer users an option to book accommodation, for example. These fields are called *billable* fields and are hooked into a checkout process. In Indico's management view of a conference, an administrator can check the status of these actions.

1. What happens now if this data is not kept in Indico anymore?
2. How is Indico validating that a registrant has paid their bookings if all registration information is kept outside of Indico?
3. What happens to generated registration tickets or receipts when a user changes the name on their data pod?

Indico allows the editing of the registration up until a set modification deadline; after it, the registration is frozen, and registrants cannot update input fields. It is necessary to have time to process the registrations, such as the booking of accommodation. The classic setup where every user has their data pod to store all this registration data would not work without additional functionality. A few things are imaginable.

– Store a copy of the registration data in Indico and freeze the copy on the modification deadline.
– Only allow decentralized storage on data where a modification would not matter.

Storing a copy of the data is an interesting idea. The freezing of the data would require Indico to also save the data in their data stores. If information is modified directly on the data pod, Indico is not necessarily notified by the change. If the user who changed the data on their data pod wants Indico to use the latest version for their copy, Indico would have to be either notified so it can fetch the new resource from the data pod; the change needs to

happen through Indico's web interface, so Indico can send the request to update it in the data pod, but also in their storage; Indico sends a request to all data pods on modification deadline day and updates the stored version. The latter solution needs validation so that existing is not replaced with corrupt or invalid data, and it is also questionable what happens when the data pod is unavailable. Still, the data shows a resignation from the conference or anything similar. How is it ensured that Indico and the user's intentions are aligned? The two systems need to be interconnected, and it needs to be clear when updates happen—availability and flow of communication or critical aspects for this success.

Communication Flow

Not to cut off the previous issue, but to continue and nurture the flow aspect in Solid to describe another issue. One crucial feature of conferences and registration is reaching out to people to let them know about next year's conference. A user would ideally only consent to the conference where they had actively signed up. Hence, next year's conference would not be able to read the participant's list and connect with them, and even if the list would be visible, how does the new conference know how to ask for consent to reach out to them or do akin actions. Solid offers an *inbox* and *notifications* [37], which allow anyone in the Solid ecosystem to connect and send a *message* to a WebID. The reached-out agent gets a notification as soon as a new item is appended into their inbox. This proactive initiation is not ideal, as the initiator does not know what data they can ask. Only when the owner actively says: "You are allowed to read this resource A and modify my resource B." does the application know about the existence of the data.

Solid-Inbox seems to be a decent solution for the use-case of Indico reaching out to previous participants. Another suggestion would be to escalate the access scope for Indico and conferences and the agent. Meaning, when a user signs up for a conference, the registration asks for consent to allow future contacting. The user could get overwhelmed when the application asks for too many different reasons to get consent. The *Usability* is at stake, and it could result in a similar situation, such as with *cookie consents. Cookie consents* are one of the main reasons browsing the web has become an unpleasant experience, due to websites flooding their visitors with complicated and obnoxious popups that need to be clicked.

Performance of Large Conference

The identical performance issue from the first POC 6.1 would arise in the management part of the conference. Storing the source of truth for hundreds of registrations without a local copy in a remote data pod would mean they would have to be fetched. Even with caching improvements, this would still be problematic and would not be feasible – pods can be down or stored on slow web servers. To make this usable in Indico, a proxy is compulsory. The proxy would temporarily store the information and make it available in a single request for multiple conference managers on separate machines. Warming up the cache by requesting resources from hundreds of data pods would still take a long time. The key takeaway here is that Indico is also serving data to users. Indico is not only taking in the registration data and then processes it and enables some feature for a single user who is only interacting with his data. On the contrary, Indico also serves a subset of users, the conference managers, data. The progress of serving the data from others is the critical facet.

Another possibility to improve the performance of the data retrieval is to look at the data pod providers. Most likely, users will not host their data pod from a computer at home but rather from a cloud provider. At the time of writing the report, a handful of such pod providers exist. Suppose the Solid server implementation running on the server of the pod provider would do optimizations, such as offering an endpoint where it would allow in one HTTP request to be asked for the data of multiple users. In that case, the server could bundle the data of such users and serve it to the application in one request-response cycle.

Modification of User Data by Management

Sometimes the conference management users may want or need to change data from the registrants. It is an interesting question if this should ever be allowed, no matter the extent of the modification on the data. An Indico developer explained a scenario when formatting names might be off, such as inputs being entirely upper- or lowercase. In this case, it is just aesthetics, but changing is understandable to ease data processing. Else, a user could provide input with offensive language – it could, of course, be removed, but a modification or moderation over the data is helpful. A solution to these problems could be to parse, filter, and format the data within Indico. Indico could have a formatting function whenever a name is displayed or processed for the case with unformatted names. The Solid approach to this problem is to ask the user for consent to change the data. Asking for consent to change data is more accessible than the case where the application did not know what to ask for. In this example, the application knows whom to ask and what to ask.

6.3 Deployment of Indico Instance

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

Indico, with the two POCs integrated, can be deployed using a hosted solution. CERN has excellent documentation to an OpenStack virtual machine (VM) [38] workflow, which allows a straightforward deployment of Indico. OpenStack is a 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 [39] distribution was chosen and the official Indico guideline to install Indico on Linux. The latest, not yet released, version of Indico uses Python 3 and brings a few caveats, but was nevertheless used for deployment. The Indico chief developer recommended this to test Indico and to use and have access to the latest features. The Indico documentation's differences that only support 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. It bundles all dependencies into a single file and sends it to the OpenStack VM, to install with a single `pip install` command. The Indico documentation requires a few system calls to be created, which must be restarted every time a new Indico version is installed.

Two additional parts are necessary to make Indico and the POC modules work correctly together. A Transport Layer Security (TLS) certificate is needed for the authentication flow with real data pods and also always a good practice to have enabled. Let's Encrypt [40] is a nonprofit Certificate Authority providing free TLS certificates and, therefore, a decent choice when in need of a certificate. Let's Encrypt certificates cannot be used in CERN's network as CERN's firewall blocks their ports. Nevertheless, again, CERN has a setup solution for this providing their self-signed certificates. These are pre-installed on CERN's work computers and thus trusted 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. SSO enables all users with a valid CERN account to authenticate with the freshly installed Indico instance and provide an authenticated Indico session when needed. Activating SSO is as easy as adding the domain of the Indico staging instance, installing CERN's Python Flask authentication library called *flask-multipass* [41] appending some additional configuration to the Indico configuration file.

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

7.1 Lack of Solid Applications

The relatively young lifetime of Solid, the only recent matured specifications, the rather unorthodox way of developing applications, vague documentation on how to build for the ecosystem, and sheer lack of motivation to develop a solution for a small group of enthusiasts are significant reasons for the lack of sophisticated Solid applications so far. It is hard for non-developers to engage and help in an ecosystem so young. Of course, many experts are needed to shape Solid and data sovereignty, but it takes developers to build applications that everyone can use. Without such applications, Solid mainly remains theoretical or relies on developers to show the signs of progress and opportunities. The *causality dilemma* can be blamed for this. It takes developers to build the software, but no one will build solutions if no one engages because of lacking solutions. However, the future is looking bright as the specifications have an approximate completion date for the summer of 2021.

7.2 Encryption at Rest

Just like HTTP is specified without an encryption layer, so is Solid. The boards of the multiple Solid panels [42] decided that encryption in Solid is not a problem to be solved for now. HTTP brings encryption with HTTP Secure (HTTPS) and ensures that all communications over HTTP are encrypted in the Solid ecosystem. For encryption at rest, meaning at the place where the data is stored and retrieved, no encryption is intended. The standards around Solid describe how the access to information is contracted, but if this data is encrypted is up to the storage mechanism, which is up to the provider of the data pod [43].

CERN has a high interest in data sovereignty and requires all their external services to store the collected data on European Union (EU) soil, where the General Data Protection Regulation (GDPR) support the data authors. For the cases where it is impossible, CERN has its own data centers to store data. In a scenario where CERN seeks a pod provider to enable Solid for their users, and the provider cannot guarantee GDPR compliance or encrypts data in the data center, it is unlikely for CERN to adopt such a solution.

A hosting on-premise solution also comes with challenges. The involvement with Solid changes from user to the maintainer, as a self-hosted solution, means it needs to be ensured services are operating as expected at all times. These tasks always demand many resources, especially with outside solutions, where an in-house engineer team cannot fix bugs, instead rely on other developers to fix them. Open-source solutions are especially of high risk, since they do not come with any service level agreements (SLAs).

7.3 User Interface

When a Solid server supports storage one mechanism can be file-based system. Modern computers ship with graphical OSs to offer a good user experience when interacting with the file system on such computers. A Solid server is a place for users to manage their data requires a good user experience and a graphical OS. The great benefit of using Linked Data to enable interoperability for Solid applications has led to many solutions for connecting and using a data pod. One system is called Solid Operating System (SolidOS) [44] and acts as the OS for Solid servers. SolidOS has many great features that allow the direct use of Linked Data on one's pod, but it comes with a challenging UI. The UI is powerful but filled with bugs and counter-intuitive interactions. For example, a user is often prompted to drag and drop a WebID URI to add the agent as a friend instead of allowing a regular text input to either copy and paste or type the WebID URI.

SolidOS is currently the best and most sophisticated solution by features; hence it comes shipped with NSS deployments on most pod providers. These UI complications make it extremely strenuous even for professionals to use one's pod and even more ambitious to onboard newcomers.

7.4 Commercial and Open-Source Solutions

What started with Sir Tim Berners-Lee as an idea and initial documentation of the idea has evolved dramatically. Today, Solid is an open standard driven by the Solid community. Everybody who wants to participate, help, steer the direction of the movement is welcome and can join at any time. Web enthusiasts do the majority of work in their spare time. Sometimes foundations or governments reward the developer's work, such as the EU, which has funded several Solid-related projects. A significant number of startups have also spotted the potential for Solid and now hold large teams. The most notable is Sir Tim's co-founded company Inrupt [45], which focuses entirely on Solid. These companies are invaluable and necessary for the success of Solid, but they also bring some complications. In recent progressions, discrepancies were faced, which can be good and healthy to debate openly about solutions and find the best one, but differences can likewise cause discomfort. Especially when decisions are made without involving the community or using their market dominance to steer the movement against planned and defined goals. Solid being so lively with a community full of opinions can lead to a multitude of contrasting Server implementation even though it is described by technical reports and backed by a test-suite offering automatic auditing of implementations. Incidentally, harming the open-source idea for the same reason as mentioned in section 7.1, where open-source developers fear to drain their time by developing an idea, which gets then replaced or made redundant by a more resourceful entity.

8 Continuation 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 switched out with NSS on the public solidcommunity.net domain. The migration 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. On the other hand, the Solid specification, has a clearly defined completion date of 30.06.21 [46]. 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 [47].

CERN has several options in its continuation 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. CERN, by policy, opts for data storage in European locations based on Operational Circular No. 11 (OC11) [48]] and GDPR [49], and that all data are merely used to provide its services [50]. Storage locations outside Europe can be possible only if there is a clear justification officially approved by the CERN Office of Data Protection (ODP). These data protection laws are also why a hosted instance through Inrupt [45] is unacceptable, as they are hosted with Amazon Web Services (AWS) on United States (US) ground. 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. Most data pod providers run the NSS, but might switch over to CSS – which is desirable but could bring new challenges. An Independent Solid Test Suite (ISTS) [51] can test if a Solid server implementation follows the Solid specification. If the implementation adheres to the specification, all features described in those technical reports should be supported and allow interoperability between servers. Feature agreements and adhering to specifications have been proven to be problematic in the past 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. Variations might be due to missing resources to stay updated with feature development or contrasting interpretations of the specifications. These risks will always endure and hence bring challenges.

Integration With CERNBox

CERN uses CERNBox [52] to provide personal cloud storage to all CERN users to host and share files. The service is based on ownCloud [53] and hosted on CERN premise. In 2016 Nextcloud [54] was formed from a fork of the open-source core software of ownCloud. PDS Interop [55], a collective of open-source developers, has developed a Nextcloud [54] plugin to make the file hosting service Solid compatible. A Nextcloud server with the Solid plugin enabled currently passes 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 option, provided resources are attributed for taking care of operational requirements and security preoccupations. Adding the Solid implementation into its cloud storage infrastructure means the CERNBox administrators now also have to administer the Solid Nextcloud plugin. System administrators should study in advance the performance when exposing the system to thousands of CERN users. A possibility could be only to enable it for a subset of users to test the integration. Resources for these steps would have to be planned. Most importantly, the attack surface increased by enabling a plugin that includes an extra sharing functionality would have to be analyzed.

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. Running a file system 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 add more value and usability to this integration, the CSS could use CERN's existing authentication service called CERN Authorization Service (CAS). Meaning, instead of identifying with an external IDP, the CERN users could authenticate with their CERN accounts using CAS, which would need to be also extended to provide WebIDs.

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

Own Server Solution

Because Solid is an open standard and allows free development of its solutions, developing a CERN Solid server from scratch is always possible. Implementing the Solid specification is from all solutions the most ambitious one and least recommended. It demands too many resources and a sophisticated solution through CSS is almost released, benefiting from additional resources.

8.2 Solid Applications

Uncertainties in adopting a CERN Solid server do not help 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 and the flaws that would need to be tackled for them to be adopted into the production instance of Indico at CERN. 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 not to 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 [56], a web application recently developed as part of a student project and is in production-ready quality and planned to be used as a leading slides maker for CERN users. A possibility for CERN to remain present in the Solid ecosystem is to develop the Slides' App further and even make it Solid compatible. The final resource combining all slides into a single presentation could be stored in a data pod. Even further, all the information on the slides could be connected to resources from data pods, 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. A design like this results in an attractive hybrid application that allows usage without a data pod or WebID but would still find users as it is a wanted software at CERN and would bring decentralized resource management to those who wish to use it.

Another minor idea in application development is based on the second POC the `autocomplete` module. Application developers could enrich their applications by using more structured data concepts in their code. As an example, attach the HTML `autocomplete` attribute to known input fields. Add more common input fields to allow the pre-definition of semantic structure on them. This way, the forms can be reasoned about easier by machines, but this effort should not be limited to HTML forms and can be extended to all sorts of fields where Schema.org has a description.

8.3 Recommendation

Based on the previous, a recommendation about 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. Announcements and calls 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 [57]. A presentation given at the HEPiX [58] was well received by international Information Technology staff from High Energy Physics and Nuclear Physics laboratories and institutes. So, a general curiosity exists, and enthusiasts are awaiting a Solid-like project to be supported by CERN. Still, it seems more generally that CERN is interested in an observing position as of now. University students are currently sought to work on the follow-up CERN-Solid project [56]. Student projects will be beneficial for CERN to understand the Solid ecosystem further and receive the latest developments in server implementations, specification evolution, and new exciting Solid apps. It will also benefit the students for their studies and experience in dealing with complex projects and environments. 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 users.

Conclusion

Decentralizing a centralized system is complex, and even developing and integrating small-scale Solid modules that harmonize with sophisticated software is. The in this paper shown work has shown in an experimental approach how the Solid ecosystem can enrich CERN software, its challenges and has laid out a possible future for the collaboration of the two.

The paper presented two working Solid apps embedded into Indico and has shown how data can reside in user's control while still be served purposeful to others. The applications have also shown how existing information in one's data pod can be used in other systems. These two insights from storing data outside of Indico and enabling information to flow back into the system has shown admirable features for the future of the Web: control through external data pods and reuse thanks to interoperable data formats of Linked Data. The third principle of Solid, decentralized authentication was also enabled and necessary for the development, but had a lesser impact due to restricting the prototype to only Indico and not guiding testing participants to other applications. Further, a decentralized SSO service is already in use at CERN.

The implementation of Solid comes with imperfections. The detailed analysis and evaluation of the Solid apps have shown those flaws in the synergy between Indico and Solid principles. The flaws being mainly performance regressions in the retrieval and distribution of data from several external data pods. Where in centralized systems the data is stored in one place and can be processed and prepared for clients neatly, in a decentralized architecture several requests are required to retrieve the data first. The prototypes' design was as decoupled as possible from Indico to not infiltrate Indico's functionality and allow other systems to integrate the modules as well. The decoupling has proven that the two systems are too distant and implementations need to be closer to Indico. A more imminent solution means Indico is required to change too. The data flow from the data pod back in to Indico, as demonstrated in the second implementation of the POC, is challenging by reason of lacking any semantic structure whatsoever in Indico.

The POCs and analyses has displayed that decentralization brings many challenges for the development of applications. However, as Tim Berners-Lee once said: "Data is a precious thing and will last longer than the systems themselves." it is sensible to decouple applications from data to allow innovation not through owning data, but by building applications without relying on having the data.

Solid as an ecosystem is active, maturing, and should be closely followed. Startups are forming worldwide and innovate by developing applications for the Solid ecosystem. Governments in several countries show their interest by investing and funding Solid-related projects to enable its citizens data pods and opportunity for data control.

The ecosystem is still inexperienced due to the scarcity of available applications for end-users to use. An atrocious UI for the currently most popular data pod implementation hinders newcomers from test Solid for private use. Whereas the UI of the implementation can be fixed by using more apparent interactions with the system, Solid still has many usability challenges to account for. Applications and agents need workflows for sharing data so seamless and without being in doubt for a second when handling their sensitive personal information. The limited number of applications and developers actively participating makes Solid still very experimental and theoretical. A planned *call-for-implementation* by the Solid technical writers will encounter these challenges and find solutions for the interesting performance complications found in the POC development. There is no doubt that in the following years, the Solid implementations and their use will skyrocket. However, it will take more organizations, companies, and individuals to initiate a breakthrough until then. The products developed need to be clever, and their benefits need to be visible to the end-user. When the end-user cannot distinguish a decentralized from a centralized solution, and no frictions are visible on use, Solid will be a success.

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: 08.05.2021).

[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: 08.05.2021).

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

[9] A. Malhotra, S. Speicher, and J. Arwe. *Linked Data Platform 1.0*. W3C Recommendation. https://www.w3.org/TR/2015/REC-ldp-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] *Solid JavaScript Authentication - solid-client-authn*. URL: https://github.com/inrupt/solid-client-authn-js/. (Accessed: 08.05.2021).

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

[23] 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.

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

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

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

[27] 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.

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

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

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

[31] *What is Cache Warming?* URL: https://www.section.io/blog/what-is-cache-warming/. (Accessed: 08.05.2021).

[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] *solid-inbox*. URL: https://github.com/solid/solid-inbox. (Accessed: 08.05.2021).

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

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

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

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

[42] *Solid Panels*. URL: https://github.com/solid/process/blob/35396c6e5a1afce674361259f63ef1b485cf6d96/panels.md. (Accessed: 08.05.2021).

[43] *Frequently Asked Questions*. URL: https://solidproject.org/faqs. (Accessed: 08.05.2021).

[44] *An Operating System for Solid*. URL: https://github.com/solid/solidos. (Accessed: 08.05.2021).

[45] *Inrupt*. URL: https://inrupt.com/. (Accessed: 08.05.2021).

[46] *Solid Technical Reports*. URL: https://solidproject.org/TR/. (Accessed: 08.05.2021).
[47] *Solid Community*. URL: https://solidcommunity.net. (Accessed: 08.05.2021).
[48] 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.
[49] *General Data Protection Regulation*. URL: https://gdpr-info.eu. (Accessed: 08.05.2021).
[50] Maria Dimou. *Policy for a CERN Solid server*. URL: https://codimd.web.cern.ch/s/zl3yGAfYZ#. (Accessed: 08.05.2021).
[51] *Independent Test Suite for Solid*. URL: https://github.com/solid/test-suite/. (Accessed: 08.05.2021).
[52] *CERNBox Service*. URL: https://information-technology.web.cern.ch/services/cernbox-service. (Accessed: 08.05.2021).
[53] *ownCloud*. URL: https://owncloud.com/. (Accessed: 08.05.2021).
[54] *Nextcloud*. URL: https://nextcloud.com/. (Accessed: 08.05.2021).
[55] *PDS Interop*. URL: https://pdsinterop.org/. (Accessed: 08.05.2021).
[56] *CERN-Solid and the Slides' app*. URL: https://it-student-projects.web.cern.ch/projects/cern-solid-and-slides-app. (Accessed: 08.05.2021).
[57] *CERN-Solid*. URL: http://solid.cern.ch/. (Accessed: 08.05.2021).
[58] *HEPiX Online*. URL: https://indico.cern.ch/event/995485/contributions/4256479/. (Accessed: 08.05.2021).