CERN-Solid Code Investigation

Proof of Concept and Prospects

by

Jan Schill
schi@itu.dk

Supervisors:

Philippe Bonnet (ITU)
phbo@itu.dk

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

A thesis presented for the degree of
Master of Science

IT UNIVERSITY OF COPENHAGEN

Computer Science
IT University of Copenhagen
Denmark
01.06.2021

Table of Contents

Introduction

Related Work

This chapter will give place to the parts of the project coming from the outside, all the relevant efforts happening in the Solid Community, or European Organization for Nuclear Research (CERN)'s involvement in the collaboration.

1 Background

The *CERN-Solid Code Investigation* project aims at linking CERN with Solid. Some of this linkage has been done in form of a status quo check of the Solid ecosystem [1]. This previous work and special thanks to Sir Tim Berners-Lee for also making a recommendation on the scope of the two proof of concepts (POCs) to test symbiosis of the two entities, CERN and Solid.

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

2 Indico

Indico is one of CERN's most sophisticated software projects. It is an event management tool, giving users a tool 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 [1].

2.1 Events

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

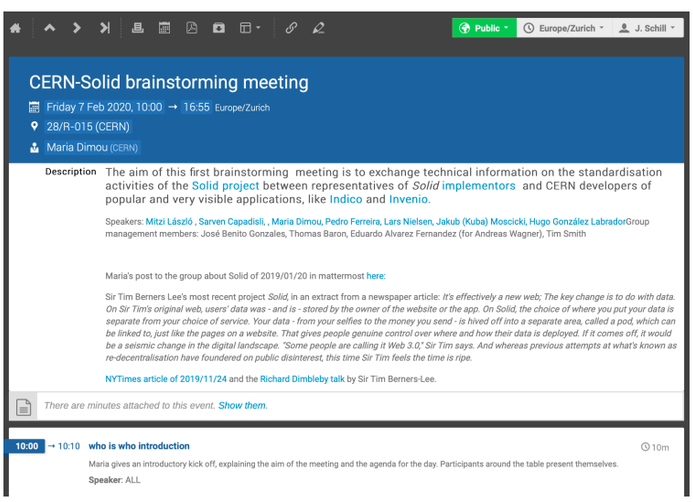


Fig. 1: User interface of an Indico event.

2.2 Storage Mechanisms

All events, all information in these events, file uploads, everything put into Indico is 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 [2]. It allows the lifetime access to all events hosted on the platform. As an example one can easily browse to the event "Big Data and Social Media" held by Vint Cerf in 2018 [3] and have access to description, recording, and slides, or even a presentation given in 2004 about "Practical Use of XML" [4].

2.2.1 EventSettingsProxy

A *proxy class* enabling 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`. When adding a new setting for the POC it will receive a field in the row specific to the event.

2.3 Conferences

In Indico an event can be a simple meetings, lectures, or all kinds of presentations. Additionally, Indico allows creation and management of conferences. Conferences are more complex events with several more features, including registration, call-for-abstracts, program definition, payments and more [5]. The conference registration is especially interesting for this project as it is part of the POC.

2.3.1 Conference Registration

Once a conference is created the conference manager needs to create 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.

3 Solid

A prior introduction to Solid was given as part of a previous *research project* [1] 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.

3.1 Authentication With Solid

In the Solid ecosystem, agents identify themselves with their WebID and proof their ownership through the Solid OpenID Connect (Solid OIDC) protocol, which is a flavor of OpenID Connect (OIDC). For further explanation and flow diagrams through the authentication process see either the work done in the previous report [1] or the Solid OIDC specification itself [6].

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 chapter 3 under the section 1.1.3.

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

Table 1: Two Solid authentication libraries.

Solid Auth Fetcher is a fork of the `solid-client-authn` developed by Inrupt but has not seen much development in recent time. With the quickly evolving Solid ecosystem, it is important to have working and up-to-date libraries to be able to build applications in this ecosystem. Both libraries do not differ in their core functionality and both support authenticating with the latest Solid servers and thus the choice is not as important. Still, the frequency of commits in Inrupt's repository seems to indicate a more active development and thus 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, all programming where Solid authentication was needed was enabled through Inrupt's `solid-client-authn`.

3.2 Reading and Writing Linked Data

Data in Solid is stored as Linked Data [7]. Resource Description Framework (RDF) is a framework for representing information in form of Linked Data in the Web [8]. The default file format so far implemented by the existing Solid servers is *Turtle* [9].

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

Fortunately, existing libraries come to rescue allowing such operations on the RDF-based data type. Again Inrupt and the Solid Community offer solutions. Again, Inrupt's solution was chosen for this development as it acts as convenient wrapper to the bare bone Linked Data application programming interface (API) implemented in `rdflib.js` [10]. Even though working with RDF can be easily done with just using `rdflib.js`, Inrupt's libraries tie nicely together and allow for instance an 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 [1] the Turtle format is a graph data structure build up with triplets. A triplet statement in its simplest form a sequence of (subject, predicate, object) terms [9].

Inrupt decided to call this construct a `Thing`, which is a data entity associated with a set of data or properties of this `Thing` [**thing**]. A `SolidDataset` is a set of things [11].

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

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

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

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

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

3.3 Authorization Through WAC

The as per Solid specification decided authorization mechanism is through Web Access Control (WAC) [12]. The majority of Solid servers use access control lists (ACLs) to manage the access modes on containers and resources.

TODO: more here

3.4 Application Launcher

A later presented discovery of a flaw in the assigning of unnecessary powerful access controls can be mitigated by a so-called *application launcher*. The inconvenience lies in the fact when an application asks the client for what the allowed scope to its data pod is, most often full access control on the root container is needed – this is against the security principle or tactic called *least privilege*. 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 chapter 3.

For now, an implementation shall be introduced showing how this problem can be avoided. When an application wants to create ACLs on a data pod it needs the before mentioned *control* access. This is the highest access mode in WAC with it files can read, written, and ACLs can be modified – meaning the owner of files can be changed as an example. In the initial Solid OIDC flow when asked for scope these controls will be decided 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, which has full control on the pod and thus must be trusted. When a new application wants to make use of a data pod and needs control access, but could, in theory, be limited to a specific container, the application launcher will create the appropriate ACL files to only allow exactly this. This works because all access modes are defined in ACL files, which can be dynamically created. The application will with this launcher now have full control of the container it operates and stores data in, but cannot reach outside of its container.

The source code for a standalone application launcher in JS exists here [13].

Investigation

1 Proof of Concepts

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

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

1.1 POC 1: Commenting Module for Events in Indico

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

1.1.1 Architectural Analysis and Synthesis

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

System Description

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

Features

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

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

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

Type of Users

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

Context Diagram

Other types of involved parties are the ones maintaining or developing the system and application. These are shown in the context diagram 11.

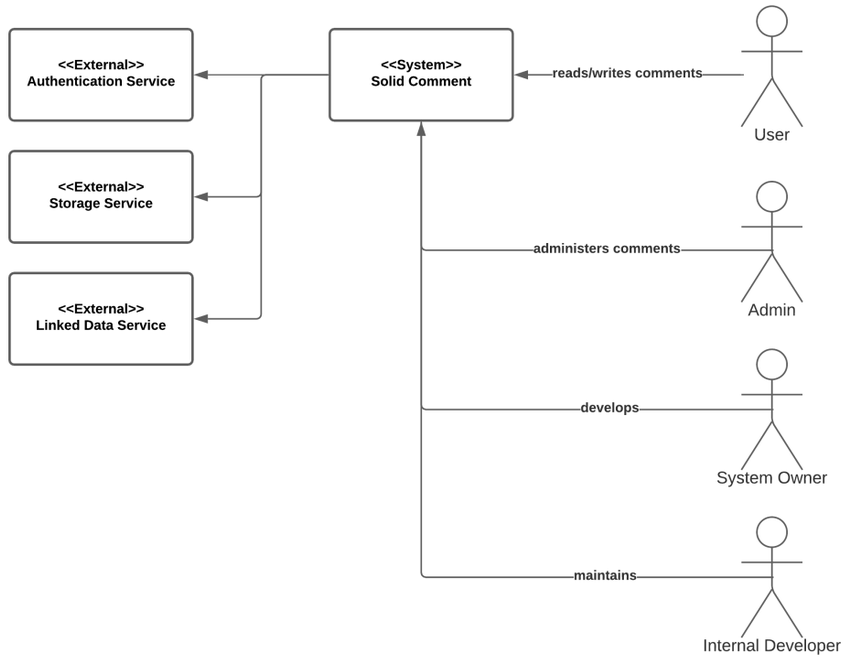


Fig. 2: Context diagram showing users and external services of system.

Sequence Diagram

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

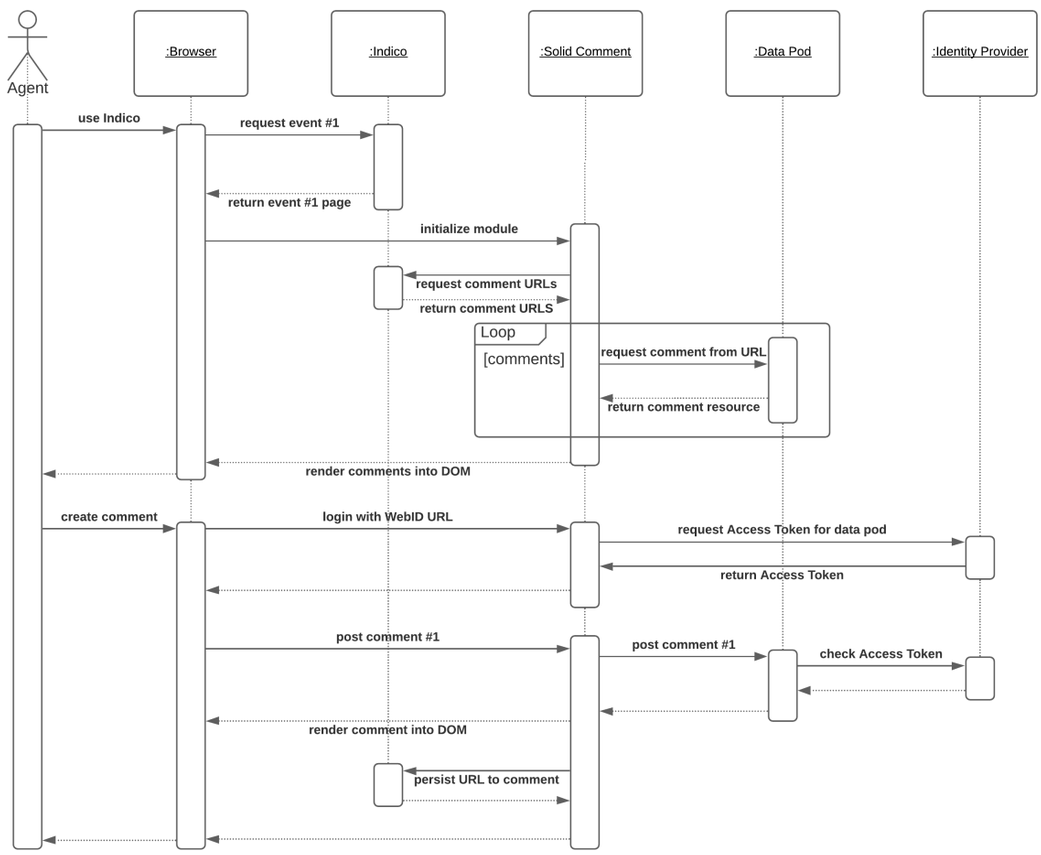


Fig. 3: Sequence diagram showing the sequential process through posting a comment.

Stakeholders

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

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

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

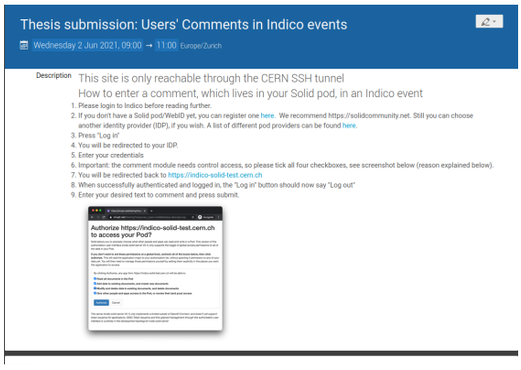
Drivers

The architectural driver or quality attributes can be specified using "-illities"

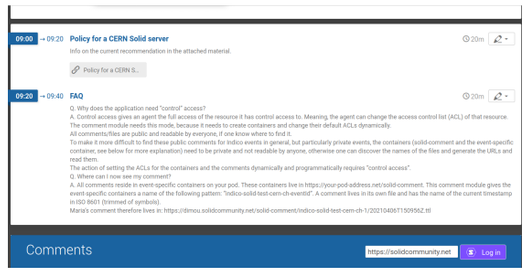
1. Security
2. Performance
3. Usability

1.1.2 Screen Design

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



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



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

Fig. 4: User interface of Indico

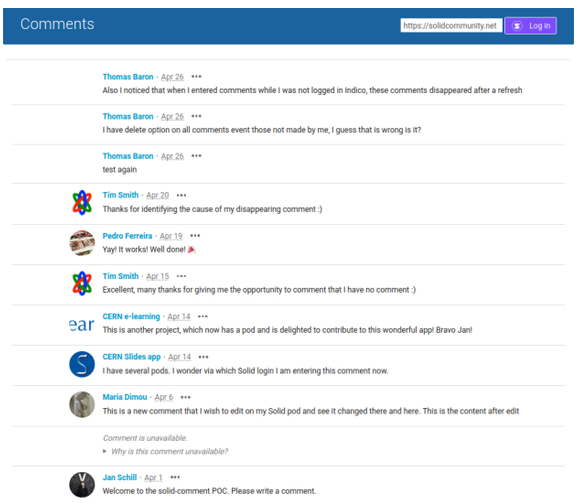


Fig. 5: User interface showing the comment module: Comments.

1.1.3 Design

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

Client- Versus Server-Side Versus Microservice

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

To make the right decision if the module should be primarily developed for the client- or server-side or even as a microservice, a list of requirements to the module had to be defined. With the defined requirements in place, it had to be figured out how much functionality can be extracted from existing libraries and how much needed to be implemented with the new module. Implementing existing functionality for a new programming language would defeat the POC's purpose of showing how an existing software could work with the Solid principles.

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

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

Client Approach

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

Problem	Solution
Language	JS or TS
Framework	Native JS
Client	solid-client-js [14]
Authentication	solid-client-authn-browser [15]
RDF	solid-common-vocab-js [16], rdflib.js [17]

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

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

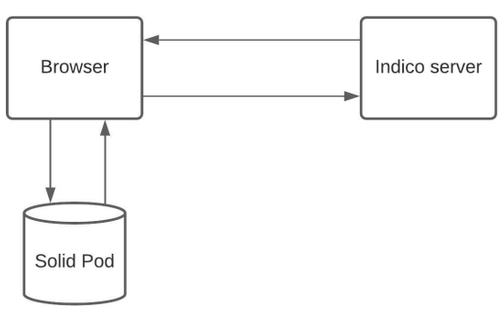


Fig. 6: Communication flow for a module developed on the client.

Microservice Approach

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

Problem	Solution
Language	JS or TS
Framework	Node.js
Client	solid-client-js [14]
Authentication	solid-client-authn-node [18]
RDF	solid-common-vocab-js [16], rdflib.js [17]

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

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

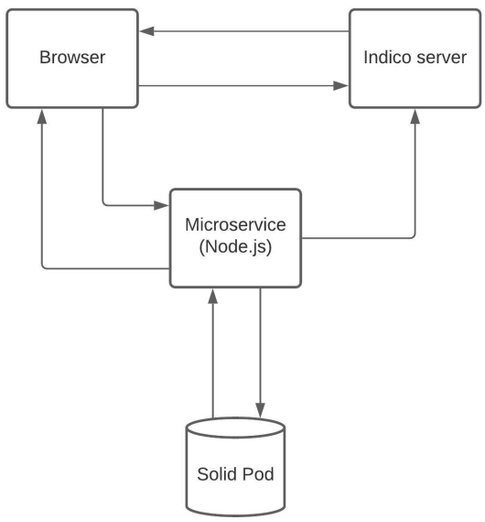


Fig. 7: Communication flow for a module developed as a microservice.

Server Approach

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

Problem	Solution
Language	Python
Framework	Flask
Client	-
Authentication	pyoidc [19] missing DPoP
RDF	solid-common-vocab-js [16], rdflib.js [17]

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

The authentication library pyoidc allows authenticating with OIDC systems but is missing a mandatory feature called Demonstrating Proof-of-Possession (DPoP), which is needed to make requests to protected resources on a data pod. TODO: say more about DPOP.

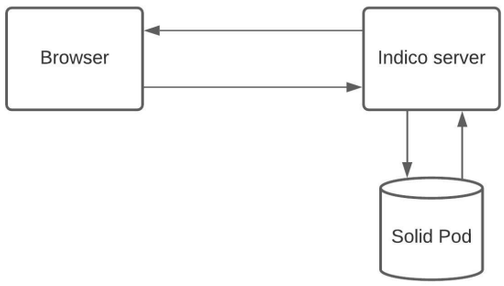


Fig. 8: Communication flow for a module developed on the server.

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.

Storage Location of the Comments

TODO:

Single Versus Multiple Resource(s) for Comments
Storing the comments in RDF can be done in two ways: storing it in one file as a graph with a list of comments, or creating a file for every comment.

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

(a) TODO: caption (b) TODO: caption

Fig. 9: TODO: two approaches

Protection on Resource

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

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

Listing 3.1: TODO: Label caption

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

# ... Definition for owner

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

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

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

Listing 3.2: TODO: Label caption

```
@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: TODO: Label caption

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

# ... Definition for owner

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

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

Listing 3.4: TODO: Label caption

```
1 @prefix acl: <http://www.w3.org/ns/auth/acl#>.
2 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
3 @prefix target: <./>.
4
5 :ReadDefault
6     a acl:Authorization;
7     acl:default target:;
8     acl:agentClass foaf:Agent;
9     acl:mode acl:Read.
```

Preventing Unwanted Discovery for Resources

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

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

Modification of Resource From Data Pod

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

Regular comment modules usually allow modifications of a submitted text throughout its existence. Twitter is an example where modifications are not allowed after posting [20].

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

Mitigation of Spam

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

Giving Application Full Control of Data Pod

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

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

1.1.4 Integration with Indico

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

Storing Reference to Comments in Indico
Indico operates using the relational database PostgreSQL [21]. When a comment gets posted and stored on an external data pod a reference to the location of the comment needs to be kept in order for Indico to pull the comment and render it in its frontend. Several possibilities are imaginable.

Enforce Authenticated Session For Posting Comments

1.1.5 Evaluation

This section focuses on evaluating the module. This shall be done by iterating through the module's requirements from the stakeholders and how the architecture lives up to those expectations. The evaluation is done with the help of the architectural Software Quality Assurance (aSQA) framework to ensure continuous quality assessment and prioritizing with a lightweight technique [22]. aSQA contains seven process steps as shown in figure 10.

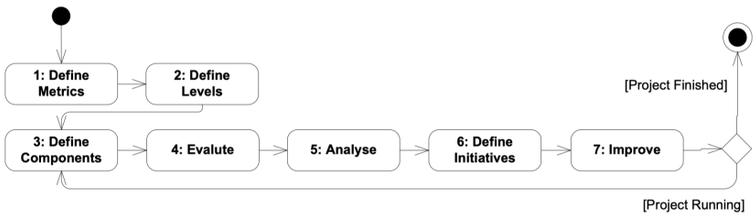


Fig. 10: aSQA Process Steps from [22]

Metrics

Scenario-based metrics were chosen to measure the quality attributes. The different scenarios are predicted to be the most common actions on the system and thus will stress the system and its architecture the most based on the quality attributes.

The iterate over the previously picked quality attributes:

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. A user who has commented deletes their comment in their data pod
2. A user who has commented deletes their data pod
3. An adversary uses a script to generate a large number of comments
4. An adversary creates a comment with a cross-site scripting (XSS) attack
5. An adversary serves different resources based on client Internet Protocol (IP) address

Levels

TODO:

Components

TODO:

Scenarios

TODO:

1.1.6 Analysis

TODO:

Performance

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

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

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

Another enhancement can be attained by caching. In its current design, the comments are freshly fetched on each page load. Server-side Hypertext Transfer Protocol (HTTP) caching is out of control for the clients, and completely relies on the Server implementations to do so. By Solid specification, HTTP caching is prescribed, but not vital [23], 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 [24]. A real improvement on consecutive page visits are won by client-side HTTP caching. Upon successive visits to a cached website, a previously fetched and in the browser's local storage stored copy of the response would be served [24]. The result would be immediately loaded, but might not serve the latest and up-to-date asset from the server. The *freshness* of the response is controlled using the `Cache-Control` header in either the request or response of the HTTP exchange between client and server [24].

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

Modification of Resources

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

Storing the comments on the author's data pod could be given up and instead be done by a trusted entity, e.g. the application embedding the comment module. It would result in CERN hosting one data pod for their Indico instance. Each event would create a container in this data pod with *append* access mode for the public. *append* allows an agent to add new resources to a container, but not modify or delete any [23]. Through this access mode a user can post comments freely, but cannot modify them at a later stage, as the data is now in Indico's data pod and the user is lacking the access control to edit existing resources. Having one data pod responsible for all comments also improves the performance by reducing the number of HTTP requests needed. Whereas before the requests were either bound to the number of authors or even by the number of comments, it is now only a single request to the event's container on the Indico data pod, to fetch a single file with all comments written in. A drawback to this approach is the comments are again stored centralized in one storage, though because it is using a Solid server instead of a web server with unstructured data storage, the data is now structured and thus interoperable through *Linked Data*. Besides storing it in Indico's data pod, a copy of the comment could be stored in the author's data pod. The author acquires a version of the comment and Indico holds the single source of truth (SSOT).

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

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

EventSettingsProxy Scalability Issues

TODO: EventProxy is not scalable, when comments are sent at the same time could be overwritten

Indico Solid Proxying

TODO: log IP and blog on spam; extra layer api to cache comments, performance and security against IP logging; one pod sends different content to users depending on their location

* Performance of petabytes? * trustworthiness of the data

1.2 POC 2: Auto-Complete for Conference Registration in Indico

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

Therefore, the initial aim was to extend the registration module to allow storage of sensible information in a data pod, where the user has full control and can handle the data to their own liking. This was decided to not be viable and the prototype was changed to allow the extraction of data from a data pod to be used in the registration process within Indico – the functionality to then only store the data from the registration in the user's data pod was dropped. The reasons and comprehensive analysis will be shared after the design and evaluation of the second POC in this chapter.

1.2.1 Architectural Analysis and Synthesis

TODO:

System Description

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

Features

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

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

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

Type of Users

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

Context Diagram

Other types of involved parties are the ones maintaining or developing the system and application. These are shown in the context diagram 11.

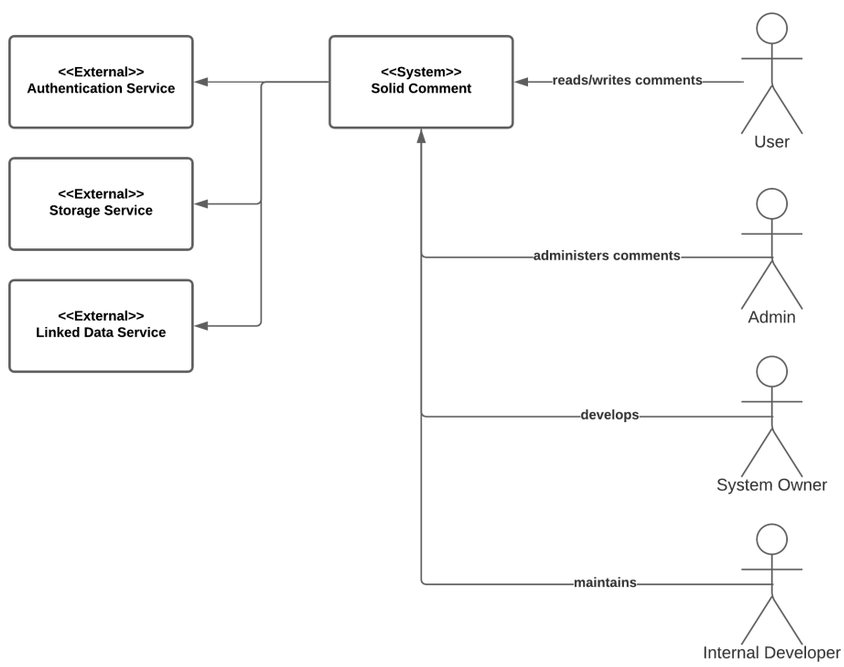


Fig. 11: Context diagram showing users and external services of system.

Sequence Diagram

TODO:

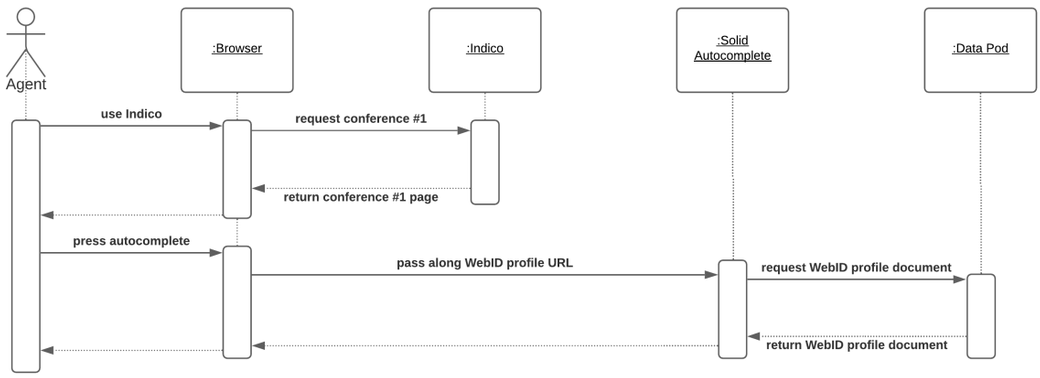


Fig. 12: Sequence diagram showing the sequential process through posting a comment.

Stakeholders

TODO:

Drivers

TODO:

1.2.2 Design

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

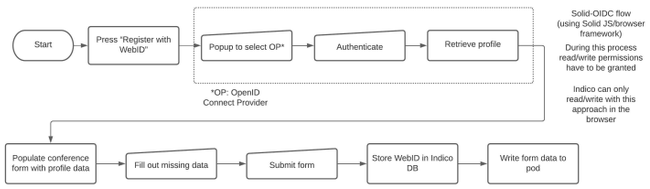


Fig. 13: TODO:

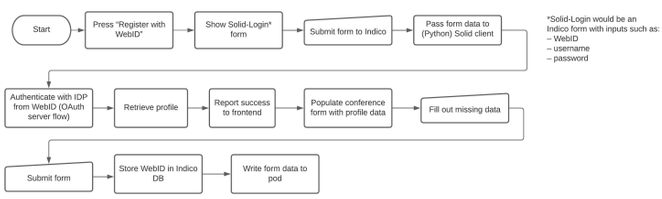


Fig. 14: TODO:

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

Modification of Resource From Data Pod

TODO:

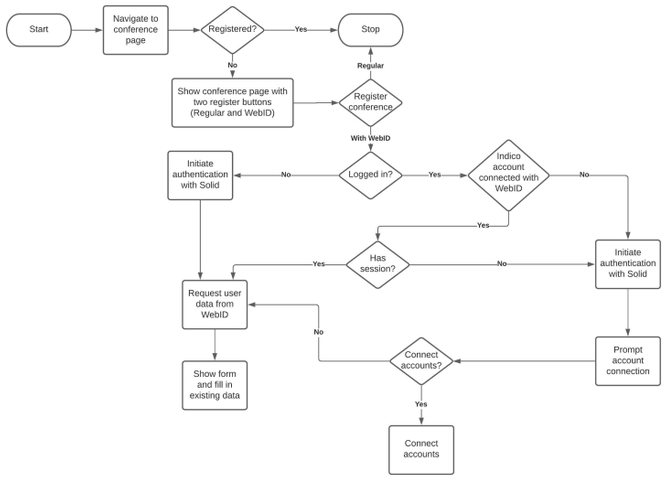


Fig. 15: TODO:

Payment on Input Fields

TODO:

Performance of Large Conference

TODO:

Availability of Crucial User Data

TODO:

1.2.3 Integration With Indico

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

Bind to Dynamically Created Form

Indico builds the registration form dynamically using a frontend library called AngularJS [25]. This introduces a challenge of interacting with the form using JS. Frontend libraries that either create new or modify existing DOM nodes need to wait until the complete DOM tree is loaded, as they bind to one DOM node from the initially served HTML document and then do their operations on this node. Meaning, AngularJS waits until the whole HTML document is parsed and rendered in the browser and then starts creating its form from scratch, which is then being rendered by the browser. The problem with this is in order to operate on the form, which is necessary for the module to be able to read the form inputs and labels and also to set their values, the JS code from this prototype needs to know when the form was successfully rendered. Three options are possible to achieve this.

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

Solutions 1 and 2 both involve the need to work with the AngularJS form, which is written in a legacy version and was recommended by an Indico developer to – if possible – be avoided. Indico developers also plan on removing AngularJS all together and replace it by a more popular frontend framework. Therefore, option 3 was chosen even though the usage of a `MutationObserver` instance might add performance degradation to the page load [26]. The performance shall not be analyzed more carefully as it is not of relevance to proof the realization of the prototype.

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

Listing 3.5: Observe function in Indico

```
1 function observeFormCreation() {
```

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

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

Detect Input Change for Frontend Validation

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

Listing 3.6: Changing the value of an input node.

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

The example code in listing 3.6 would not be detected by Indico and would upon submission render a missing value hint.

Two ways of fixing the problem are conceivable.

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

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

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

Listing 3.7: 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');
      }
```

```
    });
  }
}
```

1.2.4 Evaluation

TODO:

Metrics

TODO:

Levels

TODO:

Components

TODO:

1.2.5 Analysis

* performance? * Management part of Indico * Forward data between people * **Also serve data to people (admin of registration)** * Comments are more clear tied to comment?

* the flow of Solid (ref Tim from White Area) notifying for future conference

1.3 Deployment of Indico Instance

2 Comparison of Solid and Indico Design Principles

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

4 Proceedings in the CERN-Solid Collaboration

4.1 Servers

4.1.1 Solution

4.1.2 Hosting

4.2 Applications

Conclusion

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

References

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

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

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

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

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

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

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

[8] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

[22] 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.

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

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

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

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