CERN-Solid code investigation

Jan Schill

IT University of Copenhagen, Copenhagen, Denmark
schi@itu.dk

Table of Contents

1 Introduction

The CERN-Solid code investigation is a project that originated at CERN. Its goal is

1. to review the maturity of the Solid specifications.
2. to evaluate existing Solid implementations.
3. *to implement a proof of concept (POC) with Solid principles in CERN software.*
4. *to compare Solid principles with design principles already at CERN.*
5. *to document challenges, advantages or gaps of upcoming Solid solutions versus existing CERN ones.*
6. *to determine the proceedings for the CERN-Solid collaboration.*

The results for items 1 and 2 are shared in this document – with some additional investigative work into how to build simple applications communicating within the Solid ecosystem. The remaining milestones in this roadmap will be worked on in the upcoming Master's thesis.

Chapter 2 and 3 focus on Solid, with an introduction to Solid and its existential reasons and a brief overview of Solid's already vast ecosystem. Chapter 4 and 5 introduce and present an overview of CERN, its applications, and motivations. Chapter 6 gives a summary and review of the specifications describing Solid. Chapter 7 looks at existing Solid implementations and evaluates their status. A conclusion on the findings so far is given in chapter 8.

2 Introduction Solid

The Web was created in 1989 by Tim Berners-Lee while working at CERN "[...] to allow people to work together by combining their knowledge in a web of hypertext documents" [1]. This brilliant idea has ever since grown as an essential part of our all lives. While it has given a new platform for all types of innovation, it has also evolved away from the initial idea of sharing knowledge freely. A new term has been coined describing the phenomena of isolating data from the public by creating the so-called *data silos*. The data in these silos is then only available to the organization controlling the application. Many problems reside with this, like the actual content creator not owning their data or full access. Another drawback is that the application owners decide what interfaces are publicly accessible, therefore not allowing users to efficiently migrate their data. This centralized approach results in one user having to provide the same information to different applications: username, name, age, and others, depending on the domain. The same problem applies to traditional web applications when authenticating their users. Usually, applications will do the authentication themselves, but some initiatives decentralize this authentication called single sign-on (SSO).

Solid is aiming at solving these problems by standardizing an ecosystem where data is stored on data pods chosen and fully controlled by the users/agents, where they can decide who has access to what data; Linked Data is utilized to create interoperable data for seamless migration between applications and pods; the user authenticates with one identity provider (IDP) to use multiple Solid applications with one username and password combination.

3 Overview of Solid

In Solid data is stored on personal and through the Web-accessible storages, these are called *data pods*. Data pods are personal in users configuring the access control to the data on their pods themselves. Web-accessible because the pods can be connected to as long as a connection to the Web exists, and the proper access controls are given. Users or agents can freely choose from their favorite pod provider where they would like to store their data. As of writing this there are two major providers online, inrupt.net[2] and solidcommunity.net[3]. These providers are also used as IDPs to enable decentralized authentication, which shall be looked at more closely in a moment. The data pod storage architecture follows the Linked Data server specifications. It enables hierarchical resource discovery in a RESTful manner, where the path of the Uniform Resource Identifier (URI) gives information about the relation of its underlying data. Up until a URI does not end with a `/` character, every path segment resembles a container. A container is the collection of multiple resources. Resources are the data items stored on a pod. Every container holds information of the access control in the form of an access control list (ACL) and information of what resources it contains. Both of these resources are returned in an Resource Description Framework (RDF) compliant format, mostly Turtle. The data pod differentiates between two resource types: RDF and binary/text. RDF is a framework to represent data on the Web. The basic structure follows a graph representation, where two nodes, the subject and object, are connected by an edge, the predicate. This structure is called a *triple*. In RDF, nodes and edges elevate the benefits of URIs, more specifically Internationalized Resource Identifier (IRI) – which are a generalization of URIs, offering more Unicode characters – by either using them as globally unique identifiers or globally unique and reusable property names. This method allows interoperable data by reusing schemas with agreed-upon vocabulary to describe data and can be used to obtain more information by dereferencing the IRI.

Listing 1.1. Simple example of Linked Data with Turtle.

```
1 @prefix jan: <https://janschill.solidcommunity.net/profile/card> .
2 @prefix schema: <http://schema.org/> .
3
4 jan:me schema:familyName "Schill" .
```

The other crucial part of Solid is decentralized authentication. It is realized with WebIDs. It works so that users need a globally unique identifier—a WebID URI—which can be used with every Solid application to identify an agent. These WebIDs are handed out by IDPs, which in most cases are also data pod providers. A WebID encompasses a profile document, describing the person in more detail who is the referent of the WebID URI.

Listing 1.2. WebID URI, and WebID Profile Document URI.

```
1 https://janschill.solidcommunity.net/profile/card#me
2
3 https://janschill.solidcommunity.net/profile/card
```

Solid OpenID Connect (Solid OIDC) is the standard that is being used to authenticate within the Solid ecosystem. It is based on OAuth2/OpenID Connect. A more thorough look into Solid OIDC will be given in section 6 "Evaluation of the Specifications".

Today, several implementations exist that all do parts of the described solutions, but there is no existing complete solution. The Node Solid Server (NSS) is the original implementation and is the closest to completion measured by passing Solid Test Suite cases. It is being deployed onto both providers *inrupt.net* and *solidcommunity.net*, acting as a data pod and IDP. The Community Solid Server (CSS) is a new project aiming at replacing the NSS to become the new official open-source implementation of the Solid specifications. It is currently in beta version and actively developed. The Enterprise Solid Server (ESS) is Inrupt's commercial closed-source solution launched late this year 2020.

A lot of different libraries are built to enable development in the ecosystem. A subset as an example are client-side libraries for authentication with data pods; reading and writing RDF based resources; an SDK for React development. Additionally, efforts are put into the development of a Solid operating system (SolidOS), which can be deployed onto the data pod and supports the browsing of one's pod, editing files, parsing, and showing the data in a meaningful manner and other useful additions, such as Solid Panes, which is a set of Solid-compatible apps. These are useful for the richness of SolidOS. The core idea is to enable Solid's operating system an interface to the diverse group of linked data on a data pod. One core example is that it aims to allow a sensible representation of objects, extending to an address book showing all user contacts.

My Citizen Profile The Flanders' government in Belgium has committed to providing Solid data pods for all its six million citizens. Besides the obvious but quite honorable motivations of giving the population of Flanders control over their data, the goal was to have one storage endpoint for many governmental applications and, therefore, no need to duplicate information with different applications.

National Health Service in UK The National Health Service (NHS) in the United Kingdom (UK) is redesigning their systems with decentralized ideologies. It uses the ESS to give each citizen one health record, combine all different health data, provide ownership to their data, and improve health outcomes. Janeiro Digital[4] is the company implementing it for the NHS. They face many issues around data interoperability, as health data is complex and therefore making it work with the old and new systems in a meaningful manner is a challenge. In the Solid Data Interoperability Panel[5] all challenges, thoughts, and possible solutions are documented.

4 Introduction CERN

CERN, or the European Organization for Nuclear Research, is the largest particle physics laboratory globally, with its main site in Switzerland. It maintains the world's highest energy accelerator, the Large Hadron Collider, and houses a broad scientific program. With staff members, users, collaborating scientists, and students from all around the globe, it accounts for 12,500.

Significant challenges arise when managing this number of scientists and their experiments. CERN has closed this gap by creating and maintaining several reliable software projects and a robust infrastructure. Several relevant open-source applications shall be introduced in the next section.

5 Overview of CERN

The following open-source software systems have proven to be of excellent operational quality while serving tens of thousands of users with a wide array of functionality.

CERN has met in the past difficulties to fulfill some core workflows in the authentication and authorization in their infrastructure expected by users of their platform. The new **CERN Authorization Service** is a centralized authentication and authorization service with components such as single sign-on, group management API, account management, and computing resource lifecycles [6].

Invenio consists of three different parts. The first one is the **Framework** and provides a package for building large scale digital repositories while having scalability, security, and long-term preservation of data as its primary goals. The other two parts RDM and ILS, are not yet released but actively developed with planned releases in the year 2021.

ILS is an integrated library system allowing cataloging with bibliographic structure records, a circulation workflow, and much more in a modern user interface.

RDM stands for research data management and aims at opening a platform for researchers to share and preserve their research results.

Zenodo is a small layer on top of the Invenio Framework. The goal with Invenio RDM is to build a common RDM-platform from which everyone can profit. Invenio RDM will be based on Zenodo, and once done, Zenodo will be migrated over [7].

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 [8].

Indico is actively worked on by a team of six developers from CERN. Its open-source approach allows external participation. It is written in Python and is currently on version 2.7 and plans to move onto Python 3.x soon coupled with the release of Indico 3.0 scheduled to be released at the end of this year. Indico version 2.3 was released earlier this year—during the summer of 2020—updating the software with multiple features with the community's help, the team managing Indico at the United Nations Office at Geneva, and funding from IEEE.

Indico is the POC candidate after this research project followed Master's thesis. It is a suitable contender for applying the Solid principles as it is one of CERN's most reliable applications with a long history of operation. It does not carry any incentives in, for example, its conference registration module. This part is responsible for administering the storage (and other necessary parts) of the given data from a conference's attendee. The host of a conference decides what information is necessary to register. The information can go as far as being digital copies of physical identifications. This scenario qualifies Indico for being an ideal use-case to apply the Solid principle of decentralized storage on a data pod owned by the attendee.

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.

6 Evaluation of the Specifications

Initially reviewed the document: Editor's Draft, 13 November 2020
Revisited and partially updated: Editor's Draft, 4 December 2020

The Solid Ecosystem[9] is a by the Solid editorial team[10] published technical report. It is the official rewrite of the informal Solid specification[11], which was initially used to define the architecture of Solid servers and clients. This rewrite is still incomplete and being worked on continuously.

6.1 Summary

The Solid Ecosystem combines a set of carefully selected specifications that were adopted or newly defined to bring together an architecture that aligns Solid's principles and values. These components are loosely coupled, can therefore evolve as independently as possible to ensure flexibility and robustness [9].

The primary specification starts by describing how a data pod and a Solid app should be implemented using the Hypertext Transfer Protocol (HTTP) protocol.

A data pod is a web server that responds to HTTP requests and returns HTTP responses. When the server supports a storage mechanism, it needs to provide "a space of URIs in which data can be accessed" [9]. Solid uses containment. Containment is the relationship binding between a container, a Linked Data Platform Container (LDPC), and its resources, Linked Data Platform Resource (LDPR). The lifecycles of the LDPRs are limited by the lifecycle of its LDPC, as a resource cannot be stored without a container [9]. The storage is in the file system's hierarchy system, the root container for all the system's resources. An LDPC maintains a list of containment triples, which have the form of (LDPC URI, ldp:contains, document-URI) and list all the by the LDPC created documents.

"There is a 1-1 correspondence between containment triples and relative reference with the pathname hierarchy" [9]. This comment[12] showcases how the information of a containment triple is presented in the resource file of a container and how the URI to the resources would look.

Listing 1.3. Resource file of a container and the path to the resource.

```
1 $ curl http://example.org/container/
2
3 <> ldp:contains <resource> .
4
5 http://example.org/container/resource
```

A Solid app is a client that is sending requests to a data pod. It should be able to read and write depending on the access control to a data pod.

The URI plays an essential role in the Solid Ecosystem. It is being used to identify users with WebID, with resources in the Linked Data Platform (LDP) and more generally give information about the data pod's hierarchy of stored information.

A container resource is an organizing concept in the LDP [13]. It stores linked documents or information resources, which handle clients' requests for their creation, modification, and traversal of the linked documents [13].

An auxiliary resource exists to give additional information, like configuration, processing, or interpretation about a Solid resource, for example: "A container linked to an auxiliary resource that includes access control statements for that container and the resources that belong to it" [9]. Another more intuitive example is the need for an auxiliary resource when handling binary JPEG images. The auxiliary resource will be linked to the image and include information about it.
The ACL in Solid is realized with Web Access Control (WAC). The section for WAC is not yet written in the Solid specification but shall be given a short introduction.

WAC is similar to access control schemes used in file systems. URIs reference files, users, and groups. WebIDs identify users, in particular. Its functionality is cross-domain and can therefore have an ACL resource – holding the permissions for an agent – on domain A while setting the permissions for a file on domain B. The supported modes of operation are read, write, append, and control. Read and write are self-explanatory, whereas append and control introduce two exciting modes. Append allows the agent to add files to a container without reading or writing any containers' files. The idea of an email inbox can be compared to this functionality. Control means that the agent with this permission has access to the ACL resource and can modify it.

As mentioned, Solid follows the specifications of the LDP to define its storage mechanism. In LDP resource representation is realized with RDF. Therefore, all resources that are created are LDPR and in the Turtle format.

A WebID is an HTTP URI that denotes an agent on the Web. It is used as the primary agent identification in the Solid Ecosystem.

When making requests to a Solid server to create a resource on the server, HTTP `POST`, `PUT` or `PATCH` can be used. If the client wants to associate a specific URI with a resource, `PUT` or `PATCH` needs to be used. With the HTTP method `GET`, `HEAD OPTIONS` information about a resource can be requested. To remove a resource from the server the `DELETE` method can be used. A server must create all intermediate containers and containment triples according to `PUT` and `PATCH` requests.
On a `POST` request to `/`, the server needs to create a resource under `/slug`.
On a `POST` request to `/slug/`, the server needs to create a container for `/`.

Authentication in the Solid Ecosystem is supported in two ways. Solid OIDC is the Solid specific implementation of the widely used OpenID Connect (OIDC). The alternative, but not from the specification preferred method, is WebID-TLS. OIDC shall be focused on as it is the default for authentication in Solid. It enables the decentralized authentication and SSO mechanism needed for Solid. As previously mentioned OIDC is based from OAuth v2.0[14] and OpenID Conncect[15]. OAuth v2.0 is a protocol used for authorizing communicating parties. Its convenience in providing authorized access to resources while hiding most of its complexity from the client developer has made it a popular industry standard. A small layer was developed on top of the protocol called OIDC to allow authentication of resource owners.

Basic Flow of OpenID Connect

1 Initial Request The user or resource owner (RO) starts using the client or application and make an initial request to a private resource.

2 Client → Authorization Server The client sends a request to the authorization server (AS). The body of the request includes `Client ID`, `Redirect URI`, `Response Type`, `Scope`.

3 Provider Selection and Discovery The RO is now prompted with a selection of his preferred data pod or IDP to sign in with.

4 Authentication The RO is redirected to his preferred choice and authenticates by preferred choice – most commonly a username and password. *Optional optimization*: a cookie can be set for this domain to save a session for future visits.

5 Consent Form The RO is shown a consent form based on the `Scope` claim from the client and can choose what permissions to grant for the client.

6 Redirect The RO is now send to the `Redirect URI` from the previously sent request from the client with a temporarily `Auth Code`.

7 Client → Authorization Server The client sends another direct request with `Client ID`, `Client Secret`, and `Auth Code` to the AS.

8 Client ← Authorization Server The AS verifies the sent information and responses with an `Access Token` and `ID Token`, this `ID Token` is to identify the RO.

9 Client → Resource Server The client finally requests the resource with the `Acccess Token` attached, which is then verified between resource server (RS) and AS. Once verified, the resource is sent back to the client, and the process is complete.

Differences from Classic OpenID Connect Where in OAuth a key with permissions is granted, in OIDC a *badge* with additional basic information of the resource owner is provided. In Solid OIDC this changes to be a token including the WebID of the owner, since Solid OIDC uses the WebID URI as an identifier rather than the *issuer* and *subject* claims [16]. Using a WebID here introduces a new problem, as OIDC requires the ID to be unique for a given provider, an adversary could tamper with the WebID claim in the ID token and claim to be someone else. An additional step to verify what provider is approved by the owner of the WebID.

1. Alice using the IDP `alice.example` logs into `bob.example`. Alice's WebID `https://alice.example/#i` is included in the ID Token from `alice.example`
2. The adversary logs into `bob.example` using the IDP under their control and tampers the WebID claim in the ID Token to also be `https://alice.example/#i`.
3. The recipient `bob.example`, has no way of knowing what IDP is to be trusted with the associated WebID

Therefore, Solid OIDC introduces an extra step called "WebID Provider Confirmation" in the end to verify WebID and provider [16].

6.2 Comments

The Solid Ecosystem does an excellent job in the claims from the beginning. It does not go into best practices on building a Solid server or client but solely focuses on the precise definition of what Solid is when looked at technically. Other documents like Linked Data Primer[17] and Best Practices[18] are written to describe common patterns in the development with Linked Data. These documents would also be of value for the Solid Ecosystem. Further, the review process seems sophisticated and lively in its discussion. Contributions to the specifications are heavily discussed using the GitHub issue and pull request features, but also chat platforms like Gitter[19]. A review of such a contribution follows strict regulations. A contribution is encouraged to come with a sophisticated explanation of why this change is appropriate. Each topic within the specifications has editors assigned to them. Because Solid is open-source and therefore benefits from all parties' active contribution, it is highly recommended to participate in its development.

Clearly stating that the Solid Ecosystem document has its purpose in defining the implementation requirements for a data pod and makes suggestions to other documents that do a thorough job on speaking out use-cases and best-practices is an excellent structural decision.

No Justification For the Usage Of Linked Data Even though it might not be the proper place to explain the reasons for choosing specific technologies like Linked Data—as those discussions happen before to defining the technologies in the documentation—but it seems some clarifications why Linked Data as a technology is being used for data representation might be valuable beyond just stating that is used because of "resource discovery and lifecycle management" [9].

Limited Information on Solid Client Section 2.1.2 "Required Client-Side Implementation", in the specifications goes into the requirements for a Solid client implementation and is limited in its details. It only states it needs to be an `HTTP/1.1` client, must implement the HTTP Authentication framework[20] and the `Content-Type` HTTP header for `PUT`, `PATCH`, and `POST` requests. From a commit[21] to the Solid specification repository, it can be assumed that a section for client implementation was planned but reprioritized and delinked from the main document. A lot of Solid clients exist. Of course, the Solid ecosystem – as stated in the beginning – is not a document for best-practices. It would be highly beneficial to have such documents explicitly giving useful implementation details for developers. Comparing this specification again with the LDP specifications enjoying the presence of such supporting documents. To compare this specification again with the LDP specification enjoying such supporting documents. Section 7 "Evaluation of Solid Implementations", will look more closely at existing solutions on the server and client-side.

Incomplete Supporting Documents The Solid Ecosystem uses its specifications and external supporting specifications and capitalizes on sophisticated technologies like the HTTP. But it also references some technologies that have not been around for as long as HTTP, like WebID. WebID, in itself, is also defined in an incomplete technical report. It being incomplete as well creates a chain of uncertainty towards their definitions. In the case of WebID, it might not be crucial, as it is a straightforward specification. However, the supporting document of Solid OIDC is also a reasonably new specification, where only time and implementations tell its readiness. Suppose a missing section in the Solid Ecosystem links to an external specification. In that case, one could use that document as a source of truth, but if it is also incomplete, the risk of building something that becomes inaccurate increases.

Users Have Too Much Control WAC allows the owner of a pod to configure his access controls. With Solid gaining more popularity, the user base grows with it and the diversity in technical proficiency. Having full control over the ACLs a minor mistake in giving a malicious person root access could yield catastrophic results. Therefore, to make a data pod more user friendly, this should be addressed. Proposals such as acccess control policiess (ACPs) are being discussed and wanted in the specifications but are not written and merged in yet[22].

Complexity Even though the document does a great job of detailing specific areas, it is still demanding to follow with only limited web technologies knowledge. The complexity can be justified by the document's incomplete status and its complex nature with many areas that need to be studied. One example of this is the concept of Linked Data and all its components. It cannot be assumed of the Solid Ecosystem to explain all of its linked concepts – as it would render the document redundantly convoluted – but the fact remains that it is challenging to follow.

Incomplete Draft Since the specifications are work in progress and even some crucial *sub-specifications*, like WAC existing draft[23], are not even started, makes a review challenging as the documents are subject to additions, removals, or changes. Even though it can be assumed that the general direction of its underlying principles does not change, an application developed to the rules of today's Solid rules could result in the same application not conforming to tomorrow's set of rules.

Edit: as of lately 05.12.2020, the WAC section has been added.

6.3 Conclusion

The specifications are to be seen as *almost complete*; the incomplete sections were either finished or removed. There are still some areas of improvement, but so far, nothing major. However, what is wholly left out so far is that the specifications are the published document and all the work surrounding it: Actual implementations of the specification tested against the specification, applications that consume Solid servers, and practice the Solid principles with the help of Solid servers. To call the specification ready would therefore not help anyone. It is now up to the development to seek shortages in the definitions in the specs.

A rough estimate from a Solid developer and spec writer is the second quarter of 2021 to be the time when a *call for implementations* will be officially made. Enough confidence in the specs will be gathered until it has enough stability not to introduce breaking-changes and be technically and ethically mature.

7 Evaluation of the Solid Implementations

The ecosystem of Solid is already diverse in existing implementations. Attempts to transfer the Solid specifications into software have been carried out with different programming languages and completion levels. These servers or data pods have different goals in mind, and even though a server needs to adhere to the specifications, it does not make them the same. In the following, various existing Solid servers shall be examined. In the second part of this section, libraries for development in the ecosystem and actual developed Solid applications will be evaluated.

7.1 Solid Servers

A Solid server is a web server enabling storage through data pods. It may optionally offer IDP implementation as well [24]. In Solid, a server only needs to enable the authentication through Solid OIDC, which requires an IDP, if the user controls this IDP through the usage of an existing Solid server that is hosted on their infrastructure or they are using an identity-as-a-service vendor is up to them [9].

Node Solid Server The original Solid server was developed at the Massachusetts Institute of Technology (MIT) by Ph.D. students. This server is still the only server passing most test cases of the Solid Test Suite[25], which is a set of checks developed to test an implementation against the Solid specifications. The Test Suite for Solid is also still in development and continuously extended by more tests for the different categories of a Solid server. This server is completely open-source, written in JavaScript with the help of the web framework Node.js[26] and is commonly referred to as NSS. NSS implements a pod server and and IDP, meaning users can register a WebID, create a data pod and authenticate with it. `https://inrupt.net/` and `https://solidcommunity.net/` [27] are currently the two domains hosting the NSS and allowing users to register and use these services.

Because NSS was started as a research project, the codebase was subject to many experiments. These experiments were sometimes successful and improved the server experience by implementing useful functionality, but sometimes it would also introduce vulnerabilities or not yield the expected outcomes. These implementations were often not wholly well-designed or made self-contained, resulting in code that was hard to remove and therefore just left in the project. This negligence increased its complexity to a level where it is difficult to find an enthusiastic developer to maintain the implementation.

Community Solid Server The CSS is the from the Solid community-driven development of new open-source software to provide a way for everyone to host a data pod. It aims to allow developers to create new Solid apps and test them against a working implementation of the Solid specifications while ensuring no legacy code from older experiments influences the testing, such as in NSS.

Another significant feature of CSS is its modular architecture. Because Solid is just in the beginning and there is still a lot of different ideas, and a road map full of features for the future, CSS tries to enable by high cohesion and loose coupling in its modules a highly flexible platform for easy integration of experiments and new ideas. That can be implemented without altering existing code by plugging the experiment in as a self-contained module, which can just as easily be removed again. The modularity allows flexibility and prevents mistakes NSS experienced.

These design choices encouraged a rewrite of a Solid open-source server. Inrupt is sponsoring this development with two imec researchers and one developer. On 3 December 2020, the first beta version of CSS was released, marking a significant milestone in open-source Solid servers' journey. Developers working in the Solid ecosystem are encouraged by the core developers of CSS to switch over to CSS when developing new applications. The switch prepares the new applications to work with in the future available open-source servers and allows spotting bugs or features that have not been accounted for and therefore help with the progress of CSS.

One of the most incredible benefits of the development of CSS now is the Solid specifications are in a much more mature state they were when the development of NSS started. In-fact no such specifications existed, and Solid's experimental nature harmed the quality of the software.

It can be developed against a mature specification and has a test-suite continually checking if the development of CSS adheres to the specifications. These checks substantially increase the code's quality because it does not depend on manual checks if it is still on track with the specifications. The Solid Test-Suite is a sophisticated collection of test cases to ensure an implementation complies with the specifications. The test-suite is not yet complete and still lacks in the category of access control policies. It is not crucially consequential, as it is the alternative to the Solid preferred WAC, which is, on the other hand, well covered.

The CSS language of choice is TypeScript (TS)[28]. TS is a statically typed programming language bringing strict types to the dynamic language of JavaScript (JS). The TS compiler transpiles TS source code into JS source code.

A developer's estimate was given that by the second quarter in 2021, CSS could be production-ready.

Enterprise Solid Server Inrupt, the American-based company Tim Berners-Lee, cofounded develops the ESS. It is a commercial and closed-source alternative based on Trellis[29]. Trellis is a platform to build scalable Linked Data applications in Java. In November 2020, Inrupt released the first major version, 1.0. Besides developing a Solid server behind closed doors, they are also active in the open-source community, having developed applications like a PodBrowser[30], allowing the browsing of one's data in a pod or a set of libraries helping developers get started with the development of Solid applications.

Not much more of the implementation of the ESS can be evaluated. Inrupt does offer a practical journey for new customers, where access is given to the server with introductions to the open-source developer tools or a well-defined and in great detail outlined roadmap containing the design of a proof of concept, proof of value, pilot stage, and ready for production with a service level agreement (SLA)—considering the untrustworthiness of the NSS and that any open-source solution of a Solid server will not come with any guarantees of bug fixes within a timely manner.

PHP Solid Server The standalone PHP Solid Server is a project from PDS Interop[31] funded by the NLnet foundation[**nlnet**]. It is actively maintained and passes a good amount of cases from the Solid Test Suite, even more than the NSS in the area of CRUD. Besides the standalone version PDF Interop is also developing a plugin[32] for Nextcloud[33]. Nextcloud is an open-source collection of client-server software enabling file hosting services. The plugin makes Nextcloud compatible with Solid.

Hosting a Solid Server Inrupt and the Solid Community currently use the NSS to offer free data pods for development, experiments, and to get familiar with Solid. NSS was also used to set up an own instance on the `janschill.de` domain.

This write-down mostly follows this guide[34] from the official Solid website, the documentation in the repository[35] of the NSS.

Web Server Before installing the NSS, a physical web server, preferably running a Linux distribution, is needed. A domain should be configured at the DNS hosting and domain name registration service that holds the domain to point to this web server. The domain that will be used in this example is `janschill.de`.

Digital Wildcard Certificates NSS uses instead of a subdirectory approach a subdomain one to create the space for an isolated user pod. Using subdomains for this means a new user registers and gets a pod location at the address https://username.janschill.de/ and not https://janschill.de/username/. This design decision was made, and there has been some [discussion](https://github.com/solid/node-solid-server/issues/1349) about moving or allowing the setting of the latter. There are benefits and drawbacks to these approaches that shall not be discussed in this context. One drawback of this needs to be addressed – as it is essential for this setup. It is the need for wildcard certificates[36]). Wildcard certificates are only a drawback if a developer has not heard about this concept or has never set up digital certificates in general, as the process is quite straightforward. In short, a wildcard certificate allows a certificate to be used with multiple subdomains and is created with `certbot`, a program offered by Let's Encrypt, as follows:

Listing 1.4. Certification issuing with certbot.

```
# Install certbot
apt install certbot
# Issue certificate
certbot certonly \
--manual \
--preferred-challenges=dns \
--email schill@hey.com \
--server https://acme-v02.api.letsencrypt.org/directory \
--agree-tos \
-d janschill.de -d *.janschill.de
```

This command shows that a certificate for the ‘janschill.de‘ domain is created and for the wildcard ‘*.janschill.de‘ domain. It means any string in front of the ‘janschill.de‘ domain, separated by a period, is allowed and will have a valid certificate.

- ‘certonly‘ obtains or renews a certificate, but does not install it (it does not edit any of the server’s configuration – this will be done manually in the next step).
- ‘manual‘ will make the process of obtaining the certificate interactive.
- ‘preferred-challenges=dns‘ this is a challenge that needs to be completed to get certificates. Let’s Encrypt will not allow HTTP-01 challenges for wildcard certificates. Therefore, DNS is set for the DNS-01 challenge ([Challenge types](https://letsencrypt.org/docs/challenge-types/)).
- ‘email‘ which will be used to send important notifications.
- ‘server‘ the address ‘certbot‘ will connect to.
- ‘agree-tos‘ agree to the server’s Subscriber Agreement.

DNS-01 challenge asks to prove the control of the DNS for the specified domain. The challenge is invoked by placing a TXT record with a defined value under the domain name. Let’s Encrypt will then verify the key and value (TXT record) by querying the DNS system. Make sure the certificate directory has the correct permissions set.

“For historical reasons, the containing directories are created with permissions of 0700, meaning that certificates are accessible only to servers that run as the root user. If you will never downgrade to an older version of Certbot, then you can safely fix this using chmod 0755 /etc/letsencrypt/live,archive” [37]

Listing 1.5. Set permissions.

```
chmod -R 755 /etc/letsencrypt/live
```

Why are digital certificates needed in the first place? The Solid specifications say that: “A data pod SHOULD use TLS connections through the HTTPS URI scheme in order to secure the communication between clients and servers” [9]. Therefore, the NSS makes it mandatory to provide the location of a valid certificate when started.

Reverse Proxy A reverse proxy allows a server to run multiple services on the same port. It does so by forwarding the initial request on the host and port to the machine’s configured local service. Solid has WebID-TLS implemented as one of its authentication mechanisms. A reverse proxy – when not configured correctly – does not permit the usage of this, as the client when performing the handshake with the server also [sends its certificate](https://blog.cloudflare.com/introducing-tls-client-auth/#handshakeswithtlsclientauth), which means with the usage of a reverse proxy that performs the handshake, the certificate is not sent to the Solid server, denying the possibility of authenticating correctly. A solution is the correct configuration of the reverse proxy. [This document](https://github.com/solid/node-solid-server/wiki/Running-Solid-behind-a-reverse-proxy) introduces this issue and a few solutions to it. Therefore, the same Nginx configuration with the necessary steps to set up can be found [here](https://solidproject.org/for-developers/pod-server/nginx):

1. Open the default configuration after installing Nginx

Listing 1.6. Installing and editing Nginx.

```
sudo apt update
sudo apt install nginx
vi /etc/nginx/sites-available/default
```

2. Configuration for the reverse proxy

Listing 1.7. Nginx configuration file.

```
server {
  listen 0.0.0.0:80;
  listen [::]:80;
  server_name janschill.de;
  server_tokens off;

  return 301 https://$http_host$request_uri;

  access_log /var/log/nginx/solid_access.log;
  error_log /var/log/nginx/solid_error.log;
}

server {
  listen *:443 ssl;
  listen [::]:443 ssl;
  server_name janschill.de;
  server_tokens off;

  access_log /var/log/nginx/solid_ssl_access.log;
  error_log /var/log/nginx/solid_ssl_error.log;

  ssl_certificate /etc/letsencrypt/live/janschill.de/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/janschill.de/privkey.pem;

  root /var/www/janschill.de;
```

```
26
27    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains";
28
29    location / {
30      proxy_pass https://localhost:8443;
31
32      gzip off;
33      proxy_redirect off;
34
35      proxy_read_timeout 300;
36      proxy_connect_timeout 300;
37      proxy_redirect off;
38
39      proxy_http_version 1.1;
40
41      proxy_set_header Host $http_host;
42      proxy_set_header X-Real-IP $remote_addr;
43      proxy_set_header X-Forwarded-Ssl on;
44      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
45      proxy_set_header X-Forwarded-Proto $scheme;
46    }
47  }
```

As per default, the server's logs will be written in `/var/log/nginx/solid_*.log` files.

An additional exciting part of the configuration is that it sets the *Strict-Transport-Security* header. This header instructs the user's browser to use HTTP Strict Transport Security (HSTS) – meaning that it should use HTTP Secure (HTTPS) for every request. It is beneficial as requests that are addressed to http://janschill.de, or just janschill.de will usually connect on HTTP to the server and then get redirected, leaving an open window for a man-in-the-middle attack. HSTS solves this by instructing the browser on the first visit to use HTTPS when connecting to janschill.de. HSTS is not perfect, as it still needs one initial request even to be able to cache the 'Strict-Transport-Security' header ([Source](https://www.nginx.com/blog/http-strict-transport-security-hsts-and-nginx/)).

3. Restart the Nginx server

Listing 1.8. Restart Nginx server.

```
1  systemctl restart nginx
```

Node Solid Server 1. Install `npm` and `solid-server`

Listing 1.9. Install NSS.

```
1  sudo apt update
2  sudo apt install nodejs npm
3  npm install -g solid-server
```

2. Initialize `solid-server`

Listing 1.10. Configure NSS.

```
1  solid init
2  # Path to the folder you want to serve. Default is (./data)
3  /var/www/janschill.de/data
4  # SSL port to run on. Default is (8443)
5  8443
6  # Solid server uri (with protocol, hostname and port)
7  https://janschill.de
8  # Enable WebID authentication
9  Yes
10 # Serve Solid on URL path
11 /
12 # Path to the config directory (for example: /etc/solid-server) (./config)
13 /var/www/janschill.de/config
14 # Path to the config file (for example: ./config.json) (./config.json)
15 /var/www/janschill.de/config.json
16 # Path to the server metadata db directory (for users/apps etc) (./.db)
17 /var/www/janschill.de/.db
18 # Path to the SSL private key in PEM format
19 /etc/letsencrypt/live/janschill.de/privkey.pem
20 # Path to the SSL certificate key in PEM format
21 /etc/letsencrypt/live/janschill.de/fullchain.pem
22 # Enable multi-user mode
23 Yes
24 # Do you want to set up an email service (y/N)
25 N
26 # A name for your server (not required)
27 janschill.de
```

```
# A description of your server (not required)

# A logo (not required)

# Do you want to enforce Terms & Conditions for your service (y/N)
N
# Do you want to disable password strength checking (y/N)
N
# The support email you provide for your users (not required)
```

The configuration directories must exist and have correct user permissions.

Listing 1.11. Configure directories for file creation from NSS.

```
# Create directories
mkdir -p /var/www/janschill.de/config
mkdir /var/www/janschill.de/data
mkdir /var/www/janschill.de/.db
# Give permission
chown -R 1000:1000 /var/www/janschill.de/
```

Within the directory, start up the server.

Listing 1.12. Start Solid server.

```
# Change directory
cd /var/www/janschill.de
# Start server
solid start
```

Difficulties In the beginning, the thought of using Docker seemed tempting. Installing all dependencies in isolated environments gives the benefit of having all configurations as code. A Dockerfile holds all commands that are needed to set up an Nginx reverse proxy, for example.

Because this setup needs multiple running services (Nginx reverse proxy, certification issuing, the Solid server) that all need to communicate to each other, the Docker configuration can get quickly out of control and not offer a one-click solution anymore. Docker Compose tackles this problem by offering a configuration file to easily define how these different services/containers should be connected. To not reinvent the wheel and spend too much time configuring an Nginx reverse proxy, well-established Docker images can be used. Existing solutions exist and can be used to set up an NSS.

Unfortunately, problems occurred when the Docker images were tried; for example, the wildcard certificates were not distributed correctly. Due to time constraints and the additional overhead of dealing with these additional issues, Docker was abandoned.

7.2 Solid Client

In the process of understanding Solid and all the connected technologies, a tiny Solid app was written. It is a Node.js application, which, when compiled and bundled, runs in the browser. The functionality goes from authenticating with an IDP to reading from and writing to a data pod. The authentication was implemented with `solid-auth-client`[38] which is a browser library for allowing the authentication flow from application to the server. To handle RDF typed data the library `rdflib`[39] was used. It links a local RDF graph with a remote one. It uses the `fetch` function from JavaScript to load the requested file from a remote store and sends it back when instructed.

Listing 1.13. Basic usage of the two libraries to load authenticated requests from a data pod.

```
// 1 Importing external libraries
import { fetch } from 'solid-auth-client';
const hellowWorldURL = 'https://janschill.solidcommunity.net/public/hello-world.ttl';
const $rdf = require('rdflib');

// 2 Setting up local store
const store = $rdf.graph();
// 3 Patching browser fetch function
const fetcher = new $rdf.Fetcher(store, {
  fetch: async (url, options) => {
    return await fetch(url, options)
  }
});
// 4 Creating linked resource
const subject = store.sym(`${hellowWorldURL}#this`);
const predicate = store.sym('https://example.org/message');
const object = store.literal('Hello World');
const document = subject.doc();
```

```
19
20 async function main(){
21   // 5 Login using solid-auth-client
22   // 6 Adding resource to local store
23   store.add(subject, predicate, object, document);
24   // 7 Loading updated local store to remote server
25   await fetcher.putBack(document);
26 }
27 main();
```

1 Import External Libraries The two libraries help with the intricate work of authentication with a Solid data pod and all the needed work with RDF data.

2 Setting Up Local Store A local RDF store or graph is created and can be used to add, remove, or update the resource in it. At the moment, it is not synchronized with any remote graph and is empty.

3 Patch Browser Fetch Function The `Fetcher` is a helper to connect with a remote store. The local store is passed in and its `fetch` method patched to use the one from the `solid-auth-client`, which allows authenticated requests with a data pod.

4 Creating a Linked Resource In this step, a LDPR is created. It uses the previously defined URI, which will be later used to find the remote resource to update.

5 Login Using solid-auth-client In this step a session needs to be established with the RS holding the to be updated resource under the `helloWorldURL`. This part is omitted as it is a client-side flow, where at this part the user would be prompted to authenticate with their AS. The session, which is persisted in a cookie is automatically picked up by the `fetch` function from `solid-auth-client`.

6 Adding Resource To Local Store This step add the created resource to the local store.

7 Updating Remote Store with Local Store The local store is now uploaded to the remote server and update the resource specified.

Lessons Learned

Globbing Searching with glob patterns is the process of using wildcard characters to find matching files on the partially existing characters [40].

Listing 1.14. Listing contents of directories in root ending with r.

```
1 $ ls /*r
2 /usr:
3 bin lib ...
4
5 /var:
6 audit db ...
```

Solid data persistence is realized with RDF formatted files and can be requested by giving the exact URI of the desired file. When designing this application's initial architecture, the idea was to persist every comment in its file. The problem with this is, as a request to the container containing all the comment files would only respond with the RDF file describing the container, but not the information of all the contained resource files.

Listing 1.15. RDF describing a container. All prefixes have been omitted for readability.

```
1  # Prefixes go here
2  pub:
3      a ldp:BasicContainer, ldp:Container;
4      ldp:contains <test-file-1.ttl>, <test-file-2.ttl>;
5      st:size 4096.
6  <test-file-1.ttl>
7      a oct:Resource, ldp:Resource;
8      st:size 85.
9  <test-file-2.ttl>
10     a oct:Resource, ldp:Resource;
11     st:size 523.
```

A solution – but vigorously discussed by the spec writers and implementers – is a request to the container appended with a wildcard character.

Listing 1.16. Glob pattern in GET request to request container contents.

```
1 $ curl https://janschill.solidcommunity.net/public/*
```

Another possibility is to use the information from the request to the container and then use consecutive requests to fetch all files contained in the container. The drawback of this approach is an increase in requests to the server.

Lastly and the chosen path is writing all single comments into one file. By requesting this file, all comments can be loaded into the local graph and iterated over. Writing to the graph and then putting it back into the remote data pod works flawlessly.

RDF Schemas One challenge arose when the JS data structures had to be persisted and transformed to RDF/Turtle. Large numbers of RDF Schemas (RDFSs) exist to this day. Finding correct schemas with suitable vocabulary to describe the ontologies one is working with is troublesome. It will most likely get better with more experience, but one concern is that this might frighten beginners trying to build Solid applications.

7.3 Conclusion

The NSS is a decent foundation to get started in the realm of Solid. Setting up the server and using it is straightforward. It has been running with occasional usage on the domain for two months without problems. No significant bugs were discovered in the process so far, but it must be said the server never got pressured into a heavy load.

The CSS has not been used for any personal experiments so far. It promises a lot for the future of Solid in open-source. The architecture and quality of the code seem to be well-thought-out. Defining a clear goal in the beginning and making considerations in the architecture of the implementations, having access to people that developed on the NSS to acquire learned lessons, working with a *most complete* specification and Test Suite to allow constant testing against the specification make this an opportune candidate for future work.

The ESS is an appealing product, as it is the first professionally and closed-source server currently available for production usage. The guaranteed reliability is an attractive solution for demanding customers but not suitable for everyone as it is also a paid solution.

The Solid ecosystem is vibrant and in full motion. Server implementations are being worked on, Solid applications are starting to appear here and there, and everything seems to move in a direction where everything will come together. Even though it might seem no perfect solution exists, there is still potential as Solid's idea allows interoperability as one of the key concepts. Interoperability in this sense means if a Solid application in the form of a proof of concept is developed while using the CSS as a data pod, everything from the data pod on CSS could be easily migrated to another server, and the applications would still work. Of course, it would be naive to assume a flawless migration from one implementation to another. Different server implementations will never be 100% equal, and theory should be validated by actual practice. Therefore, claims about a flawless migration between servers are much safer to be done after testing. The specifications are so important, and only the future will tell how well-defined and robust they are in the current stage, but a foundation can be laid.

8 Conclusion

The CSS shows much potential for what is to come for Solid in the open-source community. ESS is the first example of how corporations with the needed service support could adopt Solid into their business. The NHS is a good example and an exciting project to follow. The NSS internals are better mastered at this point for this report
a. because it is the oldest,
b. because it is open to and successfully passing the tests and
c. because it is used in practice by the janschill.de pod, which is hosted there. Hence, like with all current implementations, one should follow the specs' evolution, seek advice in the Solid chat in Gitter and be ready to contribute and or change solutions as things evolve rapidly.

Acronyms

ACL access control list. 3, 6, 7
ACP acccess control policies. 7
AS authorization server. 6, 7, 13

CSS Community Solid Server. 4, 8, 9, 14

ESS Enterprise Solid Server. 4, 9, 14

HSTS HTTP Strict Transport Security. 11
HTTP Hypertext Transfer Protocol. 5–7, 11, 15
HTTPS HTTP Secure. 11

IDP identity provider. 3, 4, 6–8, 12
IRI Internationalized Resource Identifier. 3

JS JavaScript. 9, 14

LDP Linked Data Platform. 6, 7
LDPC Linked Data Platform Container. 5
LDPR Linked Data Platform Resource. 5, 6, 13

MIT Massachusetts Institute of Technology. 8

NHS National Health Service. 4, 14
NSS Node Solid Server. 4, 8–12, 14

OIDC OpenID Connect. 6, 7

POC proof of concept. 3, 5

RDF Resource Description Framework. 3, 4, 6, 12–15
RDFS RDF Schema. 14
RO resource owner. 6, 7
RS resource server. 7, 13

SLA service level agreement. 9
Solid OIDC Solid OpenID Connect. 4, 6–8
SolidOS Solid operating system. 4
SSO single sign-on. 3, 6

TS TypeScript. 9

UK United Kingdom. 4
URI Uniform Resource Identifier. 3–7, 13

WAC Web Access Control. 6–9

References

[1] Tim Berners-Lee. *Longer Biography*. 2020. URL: https://www.w3.org/People/Berners-Lee/Longer.html. (Accessed: 11.12.2020).
[2] *Inrupt website*. URL: https://inrupt.net. (Accessed: 11.12.2020).
[3] *Solid Community website*. URL: https://solidcommunity.net. (Accessed: 11.12.2020).
[4] *Janeiro Digital website*. URL: https://www.janeirodigital.com. (Accessed: 11.12.2020).
[5] *Solid Data Interoperability Panel repository*. URL: https://github.com/solid/data-interoperability-panel. (Accessed: 11.12.2020).
[6] CERN. *CERN Authorization Service*. 2020. URL: https://auth.docs.cern.ch. (Accessed: 11.12.2020).
[7] Lars Holm Nielsen. *InvenioRDM: a turn-key open source research data management platform*. 2019. URL: https://inveniosoftware.org/blog/2019-04-29-rdm/. (Accessed: 11.12.2020).
[8] CERN. *Indico*. 2020. URL: https://getindico.org. (Accessed: 11.12.2020).
[9] Tim Berners-Lee et al. *The Solid Ecosystem*. Tech. rep. W3C Solid Community Group, Dec. 2020.
[10] *Solid Specifications Panel repository*. URL: https://github.com/solid/process/blob/master/panels.md. (Accessed: 11.12.2020).
[11] *Solid Specifications repository*. URL: https://github.com/solid/solid-spec/. (Accessed: 11.12.2020).
[12] *Solid Specifications Containment Issue*. URL: https://github.com/solid/specification/issues/98%5C#issuecomment-547506617. (Accessed: 11.12.2020).
[13] Ashok Malhotra, Steve Speicher, and John Arwe. *Linked Data Platform 1.0*. W3C Recommendation. https://www.w3.org/TR/2015/REC-ldp-20150226/. W3C, Feb. 2015.
[14] *OAuth v2.0 website*. URL: https://oauth.net/2/. (Accessed: 11.12.2020).
[15] *OpenID Connect website*. URL: http://openid.net/connect/. (Accessed: 11.12.2020).
[16] Adam Migus and Ricky White. *WebID-OIDC Authentication Spec*. 2019. URL: https://github.com/solid/webid-oidc-spec. (Accessed: 11.12.2020).
[17] Roger Menday and Nandana Mihindukulasooriya. *Linked Data Platform 1.0 Primer*. Tech. rep. W3C, Apr. 2015.
[18] Cody Burleson, Miguel Esteban Gutiérrez, and Nandana Mihindukulasooriya. *Linked Data Platform Best Practices and Guidelines*. Tech. rep. W3C, Aug. 2014.
[19] *Gitter.im website*. URL: https://gitter.im/. (Accessed: 11.12.2020).
[20] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. Tech. rep. IETF, June 2014.
[21] *Solid specifications commit removing client section*. URL: https://github.com/solid/specification/commit/d387e332f3bbc9af8e7ad596fa742530262a76a9/. (Accessed: 11.12.2020).
[22] *Solid ACP proposal*. URL: https://github.com/solid/authorization-panel/blob/2d80b870dd0f71ae1d89a2dda908proposals/acp/index.md. (Accessed: 11.12.2020).
[23] *WebAccessControl wiki website*. URL: https://www.w3.org/wiki/WebAccessControl. (Accessed: 11.12.2020).
[24] Adam Migus and Ricky White. *SOLID-OIDC*. Tech. rep. W3C, Dec. 2020.
[25] *Solid Test Suite repository*. URL: https://github.com/solid/test-suite/. (Accessed: 11.12.2020).
[26] *Node.js website*. URL: http://nodejs.org/. (Accessed: 11.12.2020).
[27] *Solid Get A Pod website*. URL: https://solidproject.org//users/get-a-pod. (Accessed: 11.12.2020).
[28] *TypeScript website*. URL: https://www.typescriptlang.org. (Accessed: 11.12.2020).
[29] *Trellis website*. URL: https://www.trellisldp.org. (Accessed: 11.12.2020).
[30] *PodBrowser website*. URL: https://inrupt.com/products/podbrowser/. (Accessed: 11.12.2020).
[31] *PDSInterop website*. URL: https://pdsinterop.org/. (Accessed: 11.12.2020).
[32] *Solid Plugin for Nextcloud repository*. URL: https://github.com/pdsinterop/solid-nextcloud. (Accessed: 11.12.2020).
[33] *Nextcloud website*. URL: https://nextcloud.com/. (Accessed: 11.12.2020).
[34] *Pod Server tutorial website*. URL: https://solidproject.org/for-developers/pod-server/. (Accessed: 11.12.2020).
[35] *Node Solid Server website*. URL: https://github.com/solid/node-solid-server/. (Accessed: 11.12.2020).
[36] *SSL website*. URL: https://www.ssl.com/faqs/what-is-a-wildcard-ssl-certificate/. (Accessed: 11.12.2020).
[37] Certbot. *Where are my certificates?* 2018. URL: https://certbot.eff.org/docs/using.html#where-are-my-certificates. (Accessed: 11.12.2020).
[38] *Solid Auth Client repository*. URL: https://github.com/solid/solid-auth-client. (Accessed: 11.12.2020).
[39] *RDFlib.js repository*. URL: https://github.com/linkeddata/rdflib.js. (Accessed: 11.12.2020).
[40] techopedia. *Globbing*. 2020. URL: https://www.techopedia.com/definition/14392/globbing. (Accessed: 11.12.2020).