FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Linked Data Notifications and ActivityPub Client and Server
Student:	Bc. Antonín Karola
Supervisor:	RNDr. Jakub Klímek, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2019/20

Instructions

The student will get familiar with Linked Data, the RDF data model, the recent W3C Recommendations [1][2][3] and the Solid project [4], a recent activity of the inventor of the Web, Sir Tim Berners-Lee.
The student will implement a client and a server supporting decentralized messaging on the Web according to the Linked Data Notifications [2] and ActivityPub [3] W3C Recommendations in support of the Web re-decentralization.
The client part will be a new, user friendly messaging application.
Based on the analysis of existing Solid server implementations, the student will determine what is missing in the existing implementations for the given task.
The missing features will be implemented either as a new Solid server, or an existing implementation will be enhanced.
The client and the server will be documented, evaluated, tested and published as open-source on GitHub.
The tests will consist of unit tests and tests of compatibility with existing tools implementing the Recommendations.

References

[1] Linked Data Platform 1.0, W3C Recommendation, 2015, https://www.w3.org/TR/ldp/
[2] Linked Data Notifications, W3C Recommendation, 2017, https://www.w3.org/TR/ldn/
[3] ActivityPub, W3C Recommendation, 2018, https://www.w3.org/TR/activitypub/
[4] Solid, MIT, https://solid.mit.edu/

Ing. Michal Valenta, Ph.D.	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Head of Department	Dean

Prague December 18, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Linked Data Notifications and ActivityPub Client and Server

Bc. Antonín Karola

Katedra softwarového inženýrství
Supervisor: RNDr. Jakub Klímek, Ph.D.

July 30, 2020

Acknowledgements

I would like to thank my supervisor, RNDr. Jakub Klímek, Ph.D. for his guidence, valuable insight and patience.

Besides my supervisor, I would like to thank my family for their endless support, not just throughout my studies. Furthermore, big thanks to my friends for moral support and believing in me, especially Petr, Petra, Míša and Radim. My sincere thanks also goes to Nathaniel, Vratislav, Ema, Honza and Mike for their help.

Last but not least, thanks to IDC CEMA for their flexibility and meeting me halfway with my needs.

Declaration

In Prague on July 30, 2020 . . . .

Citation of this thesis

Karola, Antonín. *Linked Data Notifications and ActivityPub Client and Server.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

„Získejme web zpět!“ - Sir Tim Berners-Lee, tvůrce World Wide Webu.

Pro podporu re-decentralizace webu, principů otevřených dat a skutečného vlastnictví dat, tato práce buduje aplikace na základě technologií od Web Consortium (W3C).

Tato práce prozkoumává nejnovější W3C protokoly a doporučení: Linked Data (propojená data), RDF datový model, Linked Data Platform, Linked Data Notifications (LDN), ActivityPub (AP) a projekt Solid.

Pro důkaz použitelnosti těchto technologií byly vytvořeny a publikovány tři proof-of-concept aplikace (na ověření konceptu) - *inbox*, implementace Linked Data Notifications; *ldn-target-showcase*, implementace LDN Targetu; a *js-notification-poc*, implementace JavaScript Notification API a Push API.

Hlavním cílem této práce je vytvořit uživatelsky přívětivou webovou aplikaci podporující decentralizovanou komunikaci. Jako první je provedena analýza existujících LDN a AP aplikací. Na základě této analýzy je vybrán za server *solid-server* a je vytvořena nová aplikace *inbox-client*. Tento klient je otestován, zdokumentován a publikován jako open-source na GitHubu. Instance této aplikace je dále nasazena na web.

Klíčová slova webová aplikace, propojená data, návrh webové aplikace, implementace webové aplikace, Linked Data Platform, Linked Data Notifications, ActivityPub, Activity Streams, RDF, JavaScript, REST, JavaScript Notifications API, JavaScript Push API, node.js, Solid

Abstract

”Reclaim the web!” - Sir Tim Berners-Lee, the inventor of the World Wide Web.

To support Web re-decentralization, open data principles and true data ownership, this thesis builds applications on top of the Web Consortium (W3C) technologies.

This thesis investigates the current W3C protocols and recommendations: Linked Data, the RDF data model, the Linked Data Platform, Linked Data Notifications (LDN), ActivityPub (AP) and the Solid project.

To prove the applicability of W3C protocols and recommendations, three proof-of-concept applications are created and published - *inbox*, a Linked Data Notifications implementation; *ldn-target-showcase*, an LDN Target implementation; and *js-notification-poc*, a JavaScript Notifications API and Push API implementation.

The main goal of the thesis is to create a user-friendly web application supporting decentralized messaging. First, an analysis of existing LDN and AP applications is conducted. Based on this analysis, solid-server is chosen as a server and a new *inbox-client* application is created. The client is tested, documented and published as open-source on GitHub, and its instance deployed on the web.

Keywords Web application, Linked Data, Linked Data Platform, Linked Data Notifications, ActivityPub, Activity Streams, RDF, JavaScript, REST, JavaScript Notifications API, JavaScript Push API, node.js, Solid

Contents

List of Figures

Introduction

Internet was designed from the start as a decentralized network. It began as the military's ARPANET, where in the case of one network node being incapacitated, technologies like network packets and dynamic routing would still allow the rest of the nodes to communicate [7]. The internet infrastructure has since become very robust, and it is virtually impossible to take it down by disabling even multiple nodes.

Meanwhile, market monopolization has introduced a new problem - **web centralization** [8]. Companies like Google (with YouTube) and Facebook (owning Instagram) have made users dependent on them for information or entertainment [9]. As a result, the internet has become very centralized regarding ownership and services. Furthermore, web applications are incapable of intercommunication because they are closed systems (e.g. a Facebook user cannot comment on YouTube, YouTube cannot send you notifications to the app of your choice, etc.).

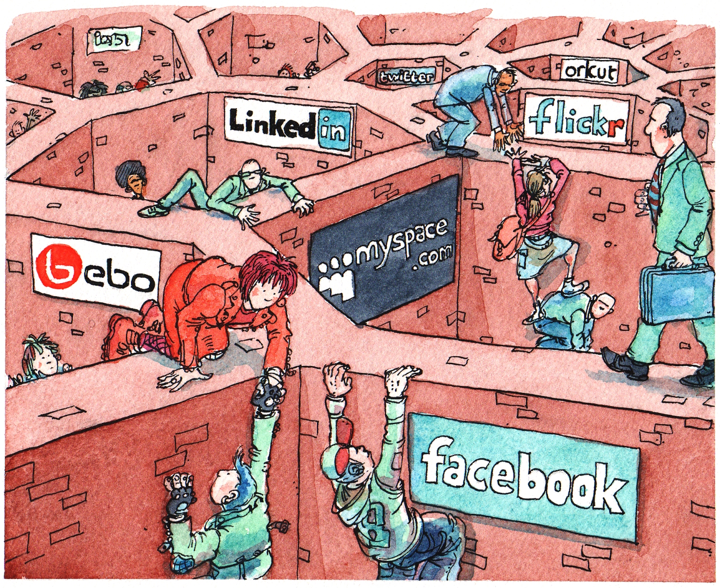
An important part of this problem is data ownership. For example, when you upload your photograph to the Facebook platform, you are granting Facebook many rights:

> "Specifically, when you share, post, or upload content that is covered by intellectual property rights ..., you grant us a non-exclusive, transferable, sub-licensable, royalty-free, and worldwide license to host, use, distribute, modify, run, copy, publicly perform or display, translate, and create derivative works of your content ... This means, for example, that if you share a photo on Facebook, you give us permission to store, copy, and share it with others ... This license will end when your content is deleted from our systems." [1]

The World Wide Web Consortium (W3C) together with the web creator Tim Berners-Lee are working to address these problems with technical solutions. They have proposed protocols such as Linked Data (LD), LD Platform

[1]From Facebook terms of use: `https://www.facebook.com/terms.php`

Figure 0.1: Example of centralized social networks [1]



(LDP) and Notifications (LDN), its extension ActivityPub (AP) and ActivityStreams (AS), so web developers can build their applications without these problems. Users would then can be able to choose e.g. their data provider and applications could intercommunicate.

We can see this development already taking place with social networks such as the decentralized platform mastodon [2] or AS video streaming service PeerTube [3]. In fact, a whole platform of interconnected, federated, open-sourced applications that are making use of the ActivityPub and other open protocols has emerged - *fediverse* [4].

Goals of this work

This work aims to provide a Linked Data Notifications and ActivityPub implementation that is easy to use and is not merely a proof-of-concept. The assignment divided the implementation into two parts - a client and a server.

The client should be a user-friendly messaging application that takes advantage of system notifications.

The server should be an AP and LDN compliant implementation, either an enhanced existing one or a new implementation.

[2]`https://joinmastodon.org/`

[3]`https://peertube.video/`

[4]Home page: `https://fediverse.party/`, about page: `https://fediverse.party/en/fediverse`

Both client and server should be documented, evaluated, tested and published as open-source on GitHub.

Chapter 1

State-of-the-art and available technology

This chapter introduces current technologies that are being used to address the centralization problem. It covers protocols developed by the World Wide Web Consortium (W3C), mainly by its Linked Data Platform Working Group and Social Web Working Group. These include Linked Data, Linked Data Platform and Notifications.

The main building block is Linked Data (LD) with its RDF representation. It allows resources to be more than just a heap of binary data. LD introduces data semantics, it gives data meaning and allows the resources to be interlinked. Furthermore, this enables data to be computer-readable and allows automated querying and processing of the data.

On top of Linked Data and RDF, W3C has developed protocols such as LD Platform (LDP) and LD Notifications (LDN). These protocols specify data formats and communications methods, so compliant applications can work together and exchange data. This allows e.g. various web applications to interchange notifications and messages. More specifically, with a LDN-compliant social network, a user could post e.g. comments on a video from another LDN-compliant video application.

1.1 Current technologies that address the centralization problem

World Wide Web Consortium (W3C) is trying to address the centralization problem with various technologies. The technologies described in this section were not necessarily created to address the centralization problem, but they are being used to do so.

1.1.1 RDF

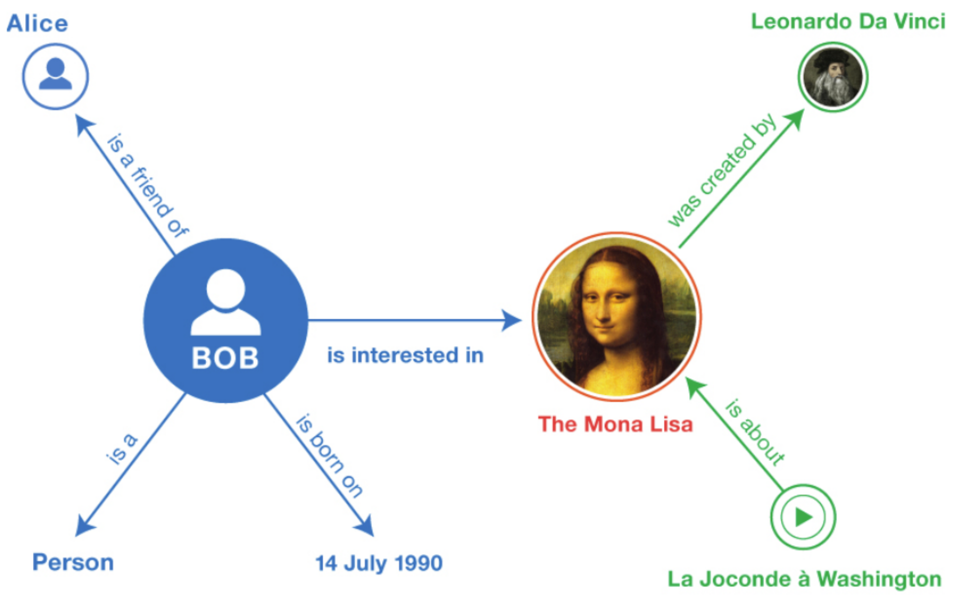
Resource Description Framework (RDF) is a standard graph data model created for data interchange on the web. It was created as a W3C specification [10] and is used for modelling information like web resources. RDF can be understood as a language for describing statements about things/entities. It consists of triples: subject + predicate + object:

Listing 1.1: RDF triple example

```
<subject> <predicate> <object>
```

These together create an oriented labeled multigraph, where subjects and objects represent nodes and predicates represent edges. Aditionally, RDF supports named graphs, thus creating quads: subject + predicate + object + graph label.

Figure 1.1: Example of a connected graph that can be represented with RDF. [2]



RDF supports a big variety of data serialization formats, Turtle/N-Triples being the most human-readable:

Listing 1.2: **RDF Turtle/N-Triples serialization example.** Turtle representation of the main subset of the graph at Figure 1.1

```
<http://example.org/#bob>
<http://perceive.net/schemas/relationship/isInterestedIn>
```

```
<http://example.org/#mona-lisa> .
```

In web applications, JSON-LD (JSON for Linking Data) [11] serialization is easier to use because JSON-LD is still valid JSON, which has robust support among web application technologies:

Listing 1.3: **JSON-LD serialization example.** JSON-LD representation of list of LDP notifications.

```
{
  "@context": "http://www.w3.org/ns/ldp#",
  "@id": "http://localhost:5001/API/notifications/",
  "@type": "ldp:Container",
  "ldp:contains": [
    {
      "@id": "http://localhost:5001/API/notifications/0"
    },
    {
      "@id": "http://localhost:5001/API/notifications/1"
    },
    {
      "@id": "http://localhost:5001/API/notifications/2"
    },
    {
      "@id": "http://localhost:5001/API/notifications/3"
    },
    {
      "@id": "http://localhost:5001/API/notifications/4"
    }
  ]
}
```

1.1.2 Linked Data

Linked Data are structured data that are interlinked. More specifically, it is a term defined by Tim Berners-Lee in his 2006 design note ”Linked Data” [12]. He outlines four basic principles of Linked Data:

1. use URI (IRI) to identify entities

2. use HTTP URI to access data

3. use RDF + SPARQL to retrieve useful information on entities

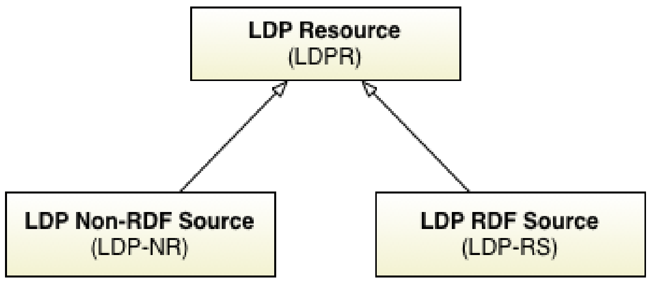
4. include links to other entities for discovery

1.1.3 Linked Data Platform

Linked Data Platform (LDP) is a W3C recommendation [13] from the 26 of February 2015, published by the Linked Data Platform Working Group[5]. It is a set of rules that applications must follow to exchange Linked Data resources.

LDP differentiates between a client and a server. They communicate using defined HTTP methods and exchange Linked Data in specific format, typically RDF. More specifically, LDP defines LDP Resource (LDPR) as a HTTP resource that conforms to the LDP patterns and conventions [3]. LDPR can be either RDF or a non-RDF resource (see Figure 1.2).

Figure 1.2: Structure of Linked Data Platform Resources [3].



Furthermore, LDP introduces an important concept for LDN - Linked Data Platform **Containers** (LDPC). Simply put, an LDP Container is an RDF resource where the subject is the container, the predicate is `ldp:contains`, and the object is the real data resource:
`<LDPC URI> <ldp:contains> <document-URI>`

Listing 1.4: **Simple LDP Container.** Example of a Linked Data Platform Basic Container in an RDF Turtle serialization format [14].

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.

<http://example.org/c1/>
   a ldp:BasicContainer;
   dcterms:title "A very simple container";
   ldp:contains <r1>, <r2>, <r3>.
```

The LDP Container concept is further extended in the Linked Data Notification protocol - LDN Inbox is based on LDP Basic Container.

[5] `https://www.w3.org/2012/ldp/wiki/Main_Page`

The following code is an example of a full LDP exchange. It represents an LDP-conformant client's request and the server's response.

Request to `http://example.org/container1/`:

Listing 1.5: **Example of LDP exchange - request.** Example of a full Linked Data Platform communication - client's request [14].

```
GET /container1/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation;
    include="http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

And response:

Listing 1.6: **Example of LDP exchange - response.** Example of a full Linked Data Platform communication - server's response [14].

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.

<http://example.org/container1/>
   a ldp:DirectContainer;
   dcterms:title "A Linked Data Platform Container of
    Acme Resources";
   ldp:membershipResource <http://example.org/container1/>;
   ldp:hasMemberRelation ldp:member;
   ldp:insertedContentRelation ldp:MemberSubject;
   dcterms:publisher <http://acme.com/>.
```

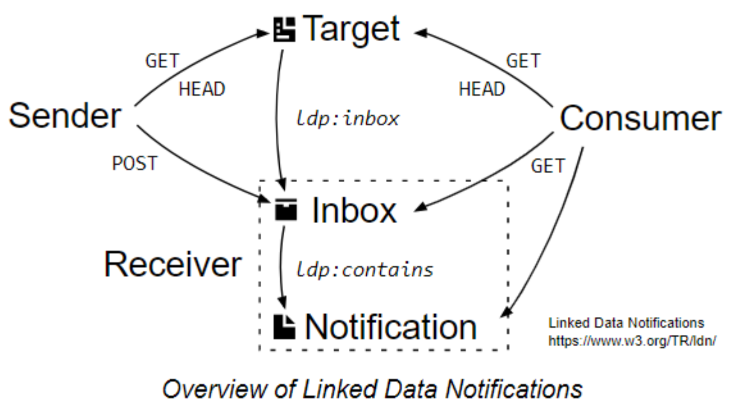
To summarize, LDP is introducing important concepts like LDP Containers, it defines communication roles (client/server), methods (HTTP) and formats (RDF). Together it represents an important building block for further applications like LDN.

1.1.4 Linked Data Notifications

Linked Data Notifications (LDN) is a W3C recommendation [4] from the Social Web Working Group. It is a subset of Linked Data Platform. LDN is a protocol that specifies generic notification format for sharing between various web applications.

It defines the following roles: target (for inbox discovery [15]), server with inbox = receiver and client = consumer/sender: Figure 1.3

Figure 1.3: **Overview of Linked Data Notifications.** LDN overview with distinct roles - Consumer, Sender and Receiver [4].


Target
GET
HEAD
Sender
POST
ldp:inbox
GET
HEAD
Consumer
GET
Inbox
Receiver
ldp:contains
Notification
Linked Data Notifications
https://www.w3.org/TR/ldn/
Overview of Linked Data Notifications

Consumer and **Sender** is typically one client web application. User A sends a notification using his application (LDN sender) to the user B's inbox on his receiver application. User B can then access the notification on his receiver using his application - LDN consumer. **Inbox** is an endpoint on the Receiver, to which the notifications are sent by the Sender and from which they are being accessed by the Consumer. Furthermore, LDN specifies a Target.

Target provides a way for a user to discover another person's Inbox [15]. There are two ways to present the Inbox's URL:

- as a response to an HTTP request using the `Link` header with `rel` value `http://www.w3.org/ns/ldp#inbox`,
- or as a predicate `<http://www.w3.org/ns/ldp#inbox>` in an RDF graph, where the subject is the requested resource and the object is the Inbox: `<http://localhost:3000/>` `<http://www.w3.org/ns/ldp#inbox>` `<http://localhost:5001/API/notifications/>`

The first option is a way to present the Inbox by the server's response headers, the second way can be embedded in the content's body, e.g. RDF, JSON-LD or even embedded in the HTML (e.g. on a blog post):

1. HTTP Link

   a) HEAD > `Link: <http://example.org/inbox/>; rel="http://www.w3.org/ns/ldp#inbox"`
   b) GET > `Link: <http://example.org/inbox/>; rel="http://www.w3.org/ns/ldp#inbox"`

2. RDF
   a) JSON with relation of type `http://www.w3.org/ns/ldp#inbox`
   b) HTML `<a>` tag with `rel="http://www.w3.org/ns/ldp#inbox"`
   c) HTML `<link>` tag with `rel="http://www.w3.org/ns/ldp#inbox"`
   d) HTML `<section>` tag with `property="http://www.w3.org/ns/ldp#inbox"`
   e) text/turtle with `<http://www.w3.org/ns/ldp#inbox>` relation

Simple discovery example:

Listing 1.7: **Example of LDN discovery.** Example of a Linked Data Notifications discovery of a Inbox using HTTP request to a LDN Target.

```
GET / HTTP/1.1
Accept: */*
Cache-Control: no-cache
Host: localhost:3000


HTTP/1.1 200 OK
Link: <http://localhost:5001/API/notifications/>;
    rel="http://www.w3.org/ns/ldp#inbox"
Content-Type: text/html; charset=utf-8
Content-Length: 249
Date: Sat, 18 Jul 2020 10:02:35 GMT

<!DOCTYPE html>
<html>
    ...
</html>
```

LDN also specifies the message format (RDF, preferably JSON-LD) and defines the application communication using HTTP protocol. Notifications can contain any data. The following HTTP dump is example of a simple LDN communication. The client, which is called ”consumer” in LDN, sends a GET request to the Receiver to access his notifications. The server with the LDN’s receiver role responds with RDF data (see Listing 1.8):

Listing 1.8: **Example of LDN exchange - request and response.** Example of a Linked Data Notifications communication - consumer’s request and receiver’s response.

```
GET /API/notifications/ HTTP/1.1
Host: localhost:5001
Accept: application/ld+json


HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Content-Type: application/ld+json; charset=utf-8
Content-Length: 390
Date: Sat, 18 Jul 2020 08:43:20 GMT
{
    "@context": "http://www.w3.org/ns/ldp#",
    "@id": "http://localhost:5001/API/notifications/",
    "@type": "ldp:Container",
    "ldp:contains": [
        {
          "@id": "http://localhost:5001/API/notifications/0"
        },
        ...
        {
          "@id": "http://localhost:5001/API/notifications/4"
        }
    ]
}
```

To summarize, LDN is a protocol for a universal notification exchange between LDN-compliant web applications. It uses RDF as data format and HTTP as communication protocol.

1.1.5 Activity Streams

Activity Streams 2.0 is a W3C data format specification [16]. It is basically a way of representing an activity in JSON. AS is specified with `application/activity+json` MIME media type.

Listing 1.9: **Basic AS example.** Very simple example of an Activity Streams data format.

```
{
  "@context": "https://www.w3.org/ns/activitystreams",
  "summary": "A note",
  "type": "Note",
  "content": "CTU FIT is awesome."
}
```

Using Activity Vocabulary[6], AS defines entities that are neccessary to represent an Activity. For example `summary`, `type` of an activity (e.g. ”Like”,

[6]https://www.w3.org/TR/activitystreams-vocabulary/

”Create”), `actor` and others. Furthermore, it defines collections, pagination and other useful constructs The five core objects are:

- Object
- Actor
- Activity
- Collection
- CollectionPage

In the following example, you can see an example of a `Person` adding an object of type `Article` to his blog, located at the `target: id` URL:

Listing 1.10: **Detailed AS example.** Example of an Activity Streams with additional details [16].

```
{
  "@context": "https://www.w3.org/ns/activitystreams",
  "summary": "Martin added an article to his blog",
  "type": "Add",
  "published": "2015-02-10T15:04:55Z",
  "actor": {
   "type": "Person",
   "id": "http://www.test.example/martin",
   "name": "Martin Smith",
   "url": "http://example.org/martin",
   "image": {
     "type": "Link",
     "href": "http://example.org/martin/image.jpg",
     "mediaType": "image/jpeg"
   }
  },
  "object" : {
   "id": "http://www.test.example/blog/abc123/xyz",
   "type": "Article",
   "url": "http://example.org/blog/2011/02/entry",
   "name": "Why I love Activity Streams"
  },
  "target" : {
   "id": "http://example.org/blog/",
   "type": "OrderedCollection",
   "name": "Martin's Blog"
  }
}
```

Activity Streams is used as the data format of the protocol ActivityPub.

1.1.6 ActivityPub

ActivityPub[7] is a protocol for decentralized social networks, which can also be extended to create all kinds of federated apps [17]. It is a W3C recommendation that provides two kinds of API:

- **client-server API** - protocol for AP clients for creating, updating and deleting content;
- **server-server API (federation protocol)** for delivering notifications and content between AP applications.

ActivityPub application can implement only one or both of them, based on the application's scope.

AP uses Activity Streams 2.0[8] as its data format. It adds couple of extra requirements. Only `Link` and `Object` entities are allowed. In the `Object` entity, `id` and `type` fields are required [17]. Additionally, the *Actor* object must have `inbox` and `outbox`. An `inbox` is a collection of all messages received by the Actor. Similarly, an `outbox` is a collection of messages produced by the Actor.

An *Actor* is not only a person/human user, but it can be any fitting object, such as a company, a website, software, city and others. Typically, it is one of the AC core types:

- Application
- Group
- Organization
- Person
- Service

However, it can also be of another type, made with ActivityStreams extension[9] (= type not defined by the Activity Vocabulary[10]). Furthermore, ActivityPub extends AS addressing with `to`, `bto`, `cc`, `bcc` and `audience` fields:

Listing 1.11: **AP example - Submitting an Activity to the Outbox.** Example of an ActivityPub Like with additional details [16].

```
POST /outbox/ HTTP/1.1
Host: dustycloud.org
Authorization: Bearer XXXXXXXXXXX
```

[7]https://www.w3.org/TR/activitypub/
[8]https://www.w3.org/TR/activitystreams-core/
[9]https://www.w3.org/TR/activitystreams-core/#extensibility
[10]https://www.w3.org/TR/activitystreams-vocabulary/

```
Content-Type: application/ld+json; profile="https://www.w3.org/ns/
    ↪ activitystreams"

{
  "@context": ["https://www.w3.org/ns/activitystreams",
               {"@language": "en"}],
  "type": "Like",
  "actor": "https://dustycloud.org/chris/",
  "name": "Chris liked 'Minimal ActivityPub update client'",
  "object": "https://rhiaro.co.uk/2016/05/minimal-activitypub",
  "to": ["https://rhiaro.co.uk/#amy",
         "https://dustycloud.org/followers",
         "https://rhiaro.co.uk/followers/"],
  "cc": "https://e14n.com/evan"
}
```

1.1.7 Solid

Solid is a set of open specifications, built on existing open standards like LDN and RDF, that describes how to build applications in such a way that users can conveniently switch between data storage providers and application providers. [18]

1.1.7.1 WebID

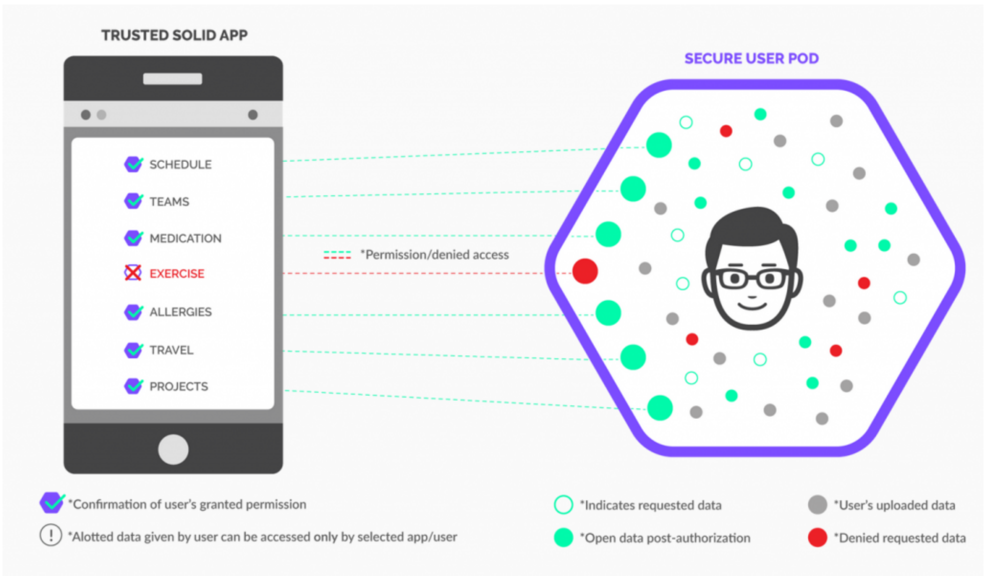
A WebID is a unique identifier of an agent (e.g. user, organization). It is an Internationalised Resource Identifier (IRI) and can be dereferenced as a FOAF profile document [19]. An example is `https://tonda.solid.community/profile/card#me`.

The owner can set sharing preferences of his WebID to the WebID of third parties [19]. In Solid, WebIDs are also used to manage access rights though Web Access Control [19].

1.1.7.2 Pod

"A Pod is where data is stored on the Web with Solid. A user may store their data in one Pod or several Pods, and applications read and write data into the Pod depending on the authorisations granted by the user or users associated to that Pod." [19] (see Figure 1.4).

Figure 1.4: **Illustration of a solid pod with application** [5]. An illustration of interaction of a web/mobile application with user's solid pod.


TRUSTED SOLID APP
SECURE USER POD
SCHEDULE
TEAMS
MEDICATION
EXERCISE
ALLERGIES
TRAVEL
PROJECTS
*Permission/denied access
*Confirmation of user's granted permission
*Alotted data given by user can be accessed only by selected app/user
*Indicates requested data
*Open data post-authorization
*User's uploaded data
*Denied requested data

Chapter 2

Analysis

This chapter deals with the analysis of the topics relevant to the thesis. First, based on the Linked Data Notification and ActivityPub protocols, the system actors are identified. Second, the system requirements are specified, distinguishing between functional and non-functional ones. Third, use cases are derived to further specify the desired system behavior.

With this system specification, an analysis of existing application was conducted in two phases. The first phase is a broad search for existing solutions with quick analysis to determine whether the application meets the basic requirements (sources, documentation are available) and is applicable for a more detailed analysis. The second phase goes into details of the system and studies if the application not only meets desired requirements but can support all the use cases.

Lastly, the analysis result is presented.

2.1 Requirements

First, requirements and use cases were specified for use in further analysis. Requirements cover required functionality on an abstract level. They are constructed based on the required technologies and represent boundaries of the system.

2.1.1 Actors

User roles can be divided into two roles as defined in LDN: consumer and sender (see Figure 1.3).

2.1.1.1 Consumer

Consumer is a person who can access his inbox and notifications. He understands what LDN and AP are and he wants to try communication using these protocols.

2.1.1.2 Sender

Sender is a person who can post notifications to other people's inboxes. He understands what LDN and AP are and he wants to try communication using these protocols.

2.1.2 List of requirements

2.1.2.1 Functional requirements

F1 Support LDN. Application supports communication using Linked Data Notification protocol.

F2 Support AP. Application supports communication using ActivityPub protocol.

F3 Support LDP inbox monitoring. Application supports monitoring of Linked Data Protocol inboxes that the user has access to.

F4 Support AP inbox monitoring. Application supports monitoring of ActivityPub inboxes that the user has access to.

F5 Support JSON-LD. Application is able to exchange data with another application using JSON-LD.

F6 Support system notifications. Application supports system notifications, such as pop-up information on incoming message to the monitored LDP inbox.

F7 Support WebID login. Users can authorize and authenticate using WebID [11].

F8 List of incoming messages. Application can show list of incoming messages.

F9 List of sent messages. Application can show list of sent messages.

F10 Show message content. Application can show message content.

F11 List of contacts. Application can show list of user contacts.

F12 Show contact detail. Application can show contact detail.

[11] https://www.w3.org/wiki/WebID

2.1.2.2 Non-functional requirements

N1 Web application. System is implemented as a web application.

N2 Git versioning. Application sources are versioned using Git VCS [12], publicly hosted on GitHub[13].

N3 Security. User can access only messages he has access to.

2.2 Use cases

A use case is a description of the specific ways a user interacts with a system. Use cases are a more specific view of system requirements.

2.2.1 List of use cases

2.2.1.1 Consumer

UC1 Start monitoring inbox. Consumer sets application so it monitors an LDP/AP inbox he has access to.

a) User logs in using WebID.
b) User clicks on action ”add inbox for monitoring”.
c) System shows form to add inbox.
d) User inputs IRI of a resource and submits.
e) System discovers resource’s inbox.
f) System starts monitoring messages coming to the inbox.

UC2 Stop monitoring inbox. User can turn off monitoring of an inbox he has previously selected for monitoring.

a) User logs in using WebID.
b) System shows list of monitored inboxes.
c) User chooses inbox to stop being monitored.
d) System stops monitoring incoming messages to the chosen inbox.

UC3 Read list of messages. Consumer can read a list of incoming messages that he has access to in a selected inbox.

a) User logs in using WebID.
b) System shows list of monitored inboxes.

[12] `https://git-scm.com/`
[13] `https://github.com/`

c) User chooses inbox.

d) System shows list of incoming messages.

UC4 Read detail of a received message. Consumer can read the complete content of a received message that he has access to.

a) User logs in using WebID.

b) User sees list of his incoming messages.

c) User can open and read the full content of the incoming message.

UC5 Receive a system notification on new message. Consumer gets a system notification, such as pop-up message, when he receives a new message to one of his monitored inboxes.

a) User logs in using WebID.

b) System receives a message for the user to a monitored inbox.

c) System shows a pop-up system notification to the user.

2.2.1.2 Sender

UC6 Send a message to a person/resource from contact list using LDN. Sender sends a message to an actor from contact list using Linked Data Notification protocol. The actor can be a person or some other resource (like company, website, etc.) identified by IRI.

a) User logs in using WebID.

b) User sees list of his contacts.

c) User clicks on the action ”send message”.

d) User enters a content of the message.

e) User submits the message.

f) System sends the message to the resource’s inbox.

UC7 Send a message to a person/resource from contact list using AP. Sender sends a message to an actor from contact list using ActivityPub protocol. The actor can be a person or some other resource (like company, website, etc.) identified by IRI.

a) User logs in using WebID.

b) User sees list of his contacts.

c) User clicks on the action ”send message”.

d) User enters a content of the message.

e) User submits the message.

f) System sends the message to the resource's inbox.

UC8 Send a message to an unknown person/resource using LDN. Sender sends a message to an actor that is not in his contact list using LDN protocol. The actor can be a person or some other resource (like company, website, etc.) identified by IRI.

a) User logs in using WebID.
b) User opens a new message form.
c) User enters the receiver's IRI.
d) User enters a content of the message.
e) User submits the message.
f) System sends the message to the resource's inbox.

UC9 Send a message to an unknown person/resource using AP. Sender sends a message to an actor that is not in his contact list using AP protocol. The actor can be a person or some other resource (like company, website, etc.) identified by IRI.

a) User logs in using WebID.
b) User opens a new message form.
c) User enters the receiver's IRI.
d) User enters a content of the message.
e) User submits the message.
f) System sends the message to the resource's inbox.

2.3 Analysis of existing solutions

An analysis of existing Solid server implementations was conducted to determine whether they could be used for the given task (a server supporting decentralized messaging). Applications were searched on the web using Google with terms like "solid server". Also, existing implementations linked at "LDN Test Reports and Summary" https://linkedresearch.org/ldn/tests/summary and https://solidproject.org/use-solid/apps were examined.

2.3.1 Criteria for analysis of existing solutions

The following criteria were considered:

- application has available sources;
- license permits extending the application;

- application has sufficient documentation on how to run it, or it is runnable without the need for documentation;
- application is live or with active development - sources without a commit within 1 year were excluded;
- application support for linked data, LDP, LDN, AP, RDF and system notification;
- application must be extensible with our cause - LDN/AP notifications. Applications without the possibility of being extended with our use cases were excluded.

Based on these criteria, three applications were selected for more detailed analysis (see section Detailed analysis below).

2.3.2 Overview of all analysed applications

2.3.2.1 SCTA inbox receiver

Only a simple LDN app to pass the AP test suite. No information available on how to run this. No extension possible.

Accessible at `https://github.com/scta/scta-inbox`.

2.3.2.2 Sloph/DIY Inbox

Sample/POC LDN demonstration in PHP. No extension possible. Made only to pass the AP test suite.

Accessible at `https://rhiaro.co.uk/2017/08/diy-ldn`.

2.3.2.3 gold

Reference Linked Data Platform server for the Solid platform. Not maintained anymore - latest commit on Oct 10, 2018.

Accessible at `https://github.com/linkeddata/gold`.

2.3.2.4 ldn-streams

Implementation of Linked Data Notifications for RDF streams. Not maintained anymore - latest commit on Jun 11, 2018.

Accessible at `https://github.com/jpcik/ldn-streams`.

2.3.2.5 Carbon LDP

Carbon LDP is ”an enterprise-class Linked Data Platform that helps artists and engineers create and extend web applications with ease” [20]. Based on its broad capabilities, it has been selected for a detailed analysis, see below.

Accessible at `https://carbonldp.com/`, `https://github.com/CarbonLDP`.

2.3.2.6 solid-server in Node

Solid-server is the main candidate for the possible solution. It is implemented in NodeJS on top of the file system. Solid supports Linked Data Platform, Web Access Control, WebID+TLS Authentication, real-time live updates (using WebSockets) and other features. It is available both as a NodeJS project and as a Docker container. Detailed analysis is available below.

Accessible at `https://github.com/solid/node-solid-server`.

2.3.2.7 pyldn

Pyldn is a lighweigth Linked Data Notifications (LDN) receiver implemented in python. As it does not include other LDN parts, it was not selected for further analysis.

Accessible at `https://github.com/albertmeronyo/pyldn`.

2.3.2.8 Virtuoso Universal Server

Commercial Data Virtualization platform. Sources not available.

Accessible at `https://virtuoso.openlinksw.com/#this`.

2.3.2.9 maytkso

”HTTP server and command-line RDF tool to get/send, serialise data.” [21] Written in one JS file, merely an LDN server/receiver. Not acceptable for extension.

Accessible at `https://github.com/csarven/mayktso`.

2.3.2.10 Apache Marmotta

Open platform for LDP, implemented as a Java web application. Has to be run on a JavaEE application server. Based on its capabilities, it has been selected for a detailed analysis below.

Accessible at `https://marmotta.apache.org/`.

2.3.2.11 IndieAnndroid/ blog-a-loosh

Indieweb blog platform. Developed for personal use and not applicable for our purpose.

Accessible at `https://github.com/Kongaloosh/blog-a-loosh`.

2.3.2.12 LDP-CoAP

LDP for the Constrained Application Protocol. Provided mapping of LDP over HTTP to CoAP (RFC 7252 Constrained Application Protocol [22]). Not applicable for our purpose.

Accessible at `http://sisinflab.poliba.it/swottools/ldp-coap/`, `https://github.com/sisinflab-swot/ldp-coap-framework`.

2.3.2.13 distbin.com

Application similar to pastebin - for easy copy, paste and share of text. Not useful for messaging. Not available anymore.

Accessible at `https://distbin.com/`.

2.3.2.14 Fedora Repository

Big repository system for management and dissemination of digital content (digital libraries and archives). It does not support use cases such as decentralized messaging for a common user.

Accessible at `https://wiki.duraspace.org/display/FF/Fedora+Repository+Home`.

2.3.2.15 SNS

Social network based on Solid build on core JavaScript. No sources available.

Accessible at `https://electrapro-pk.github.io/SNS/`.

2.3.2.16 Solidarity

Chat application written in node.js. It is an online chat with channels. No license information available.

Accessible at https://github.com/scenaristeur/solidarity, https://scenaristeur.github.io/solidarity/.

2.3.2.17 OChat

Simple chat application written with React. Not maintained anymore (last commit on Jul 25, 2019).

Accessible at https://github.com/jaxoncreed/o-chat, https://chat.o.team/.

2.3.2.18 Friend Requests Exploration

Exploration into how Solid could be used for sending friend requests. Not extensible.

Accessible at https://github.com/inrupt/friend-requests-exploration.

2.3.2.19 solid-inbox

Inbox for processing notifications. Single JS file app. Not maintained anymore.

Accessible at https://github.com/solid/solid-inbox.

2.3.3 Detailed analysis of selected applications

Applications selected in the previous analysis were subjected to a more detailed analysis based on the requirements and support of possible use cases. Table 2.1 is an overview of the analysis, details follow below.

2.3.3.1 solid-server

Solid is a project led by Prof. Tim Berners-Lee, inventor of the World Wide Web, taking place at MIT [23]. Solid-server is a server is implemented in NodeJS on top of the file system. It supports Linked Data Platform, Web Access Control, WebID+TLS Authentication, real-time live updates (using WebSockets) and other features. It is available both as a NodeJS project and as a Docker container.

It is still a developing prototype. The main instance is running at https://solid.community/, however it is designed so anybody can host their instance.

It is published under the free MIT license [24], so it available for extension. Source code and documentation is available at https://github.com/solid/node-solid-server.

Table 2.1: **Detailed analysis.** This table shows application support of requirements and use cases. Only applications selected for detailed analysis are shown. See requirements section and use cases section for details. ✓ means full support, ✗ means no support, - means that the support could not be verified or that it is not applicable in the application's scope.

requirement/	application name	
use-case ID	solid-server	Apache Marmotta
F1	✓	✓
F2	✓	✗
F3	✓	✓
F4	✓	✗
F5	✓	✓
F6	✓	-
F7	✓	✗
F8	✓	-
F9	✓	-
F10	✓	-
F11	✓	-
F12	✓	-
N1	✓	✓
N2	✓	✓
N3	✓	✓
UC1	✓	-
UC2	✓	-
UC3	✓	-
UC4	✓	-
UC5	✓	-
UC6	✓	-
UC7	✓	-
UC8	✓	-
UC9	✓	-

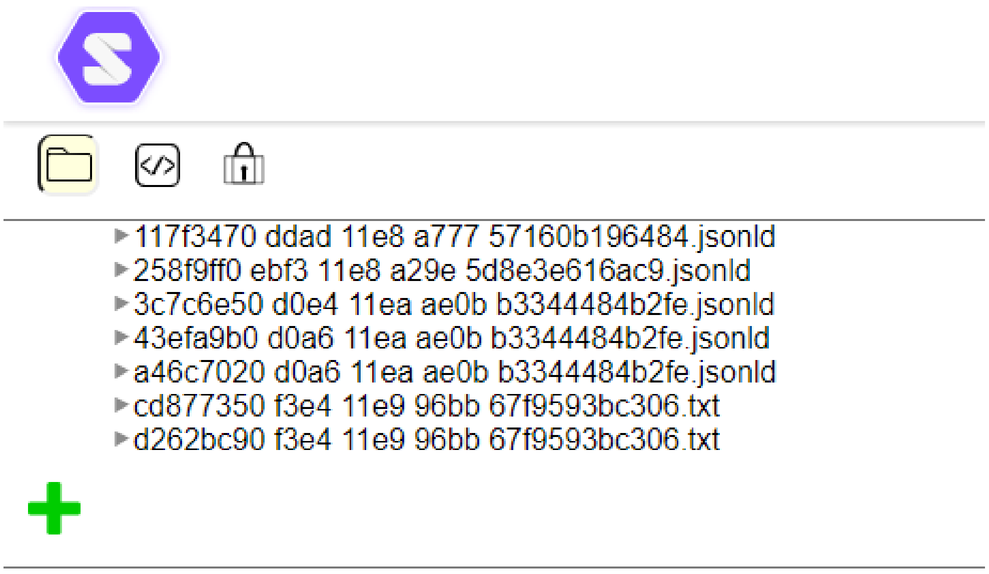
2.3.3.2 Solid inbox client

As a part of the public Solid server instance at `https://solid.community/`, there is an existing inbox client application. The problem is that its user interface (UI) is very un-intuitive and cumbersome. Figure 2.1 is the UI for the use case UC4 - read a list of inbox messages:

The following two screenshots Figure 2.2 and Figure 2.3 capture the solid UI for message detail - UC4.

As you can see, the current UI makes it impossible to access the detail content and the user is forced to use another solution (such as HTTP GET

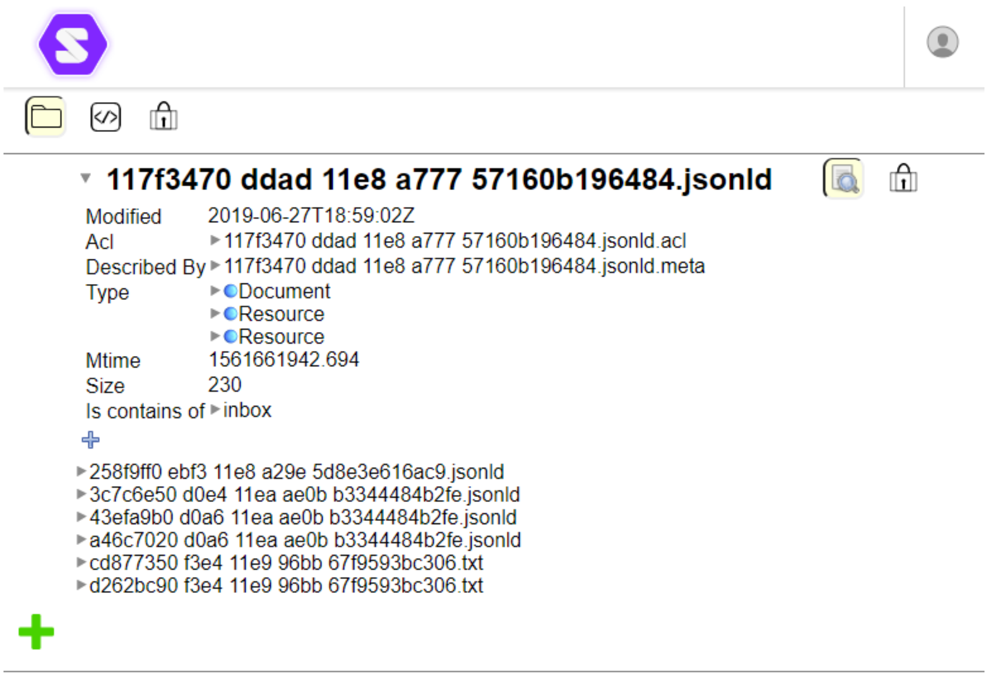
Figure 2.1: **Solid inbox - list of messages**. Screenshot of the current official inbox client application.



▸117f3470 ddad 11e8 a777 57160b196484.jsonld
▸258f9ff0 ebf3 11e8 a29e 5d8e3e616ac9.jsonld
▸3c7c6e50 d0e4 11ea ae0b b3344484b2fe.jsonld
▸43efa9b0 d0a6 11ea ae0b b3344484b2fe.jsonld
▸a46c7020 d0a6 11ea ae0b b3344484b2fe.jsonld
▸cd877350 f3e4 11e9 96bb 67f9593bc306.txt
▸d262bc90 f3e4 11e9 96bb 67f9593bc306.txt

the RDF document representation.

Figure 2.2: **Solid inbox - message detail**. Screenshot of the current official inbox client application.

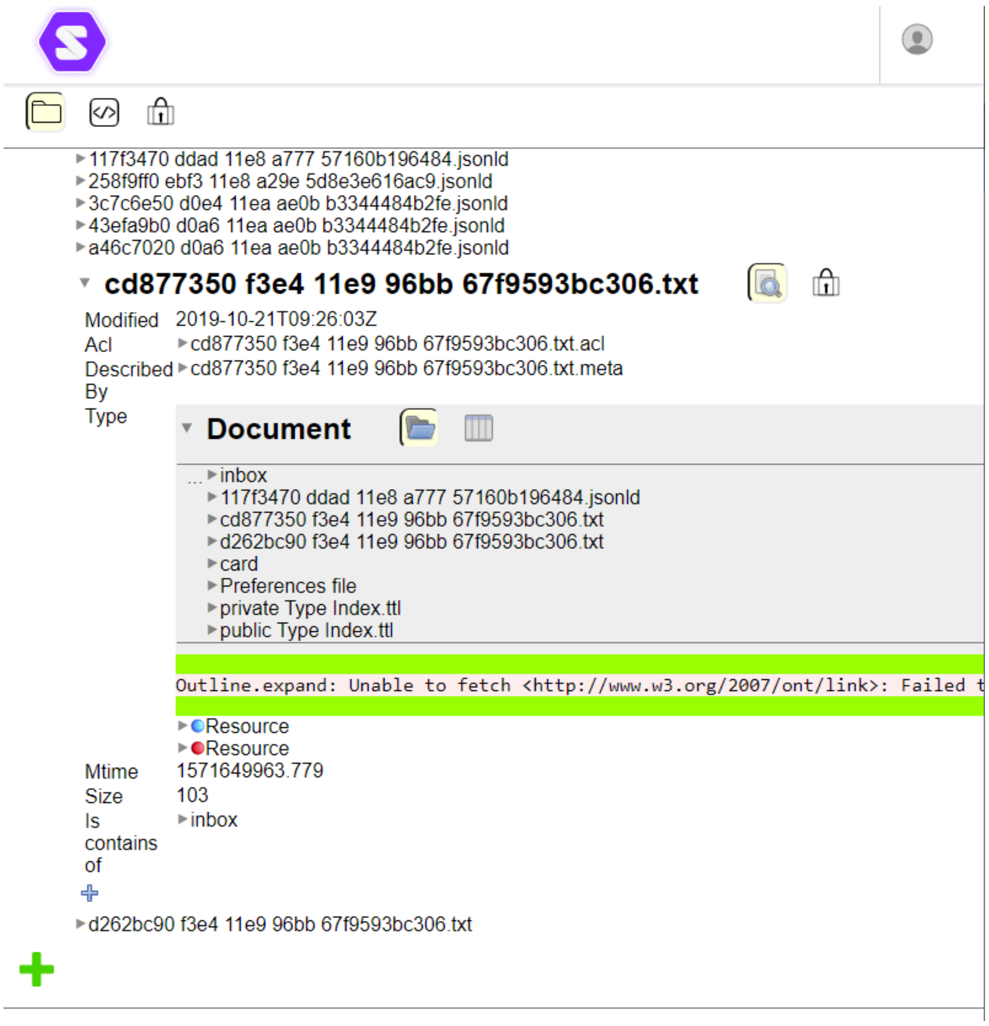


▾ 117f3470 ddad 11e8 a777 57160b196484.jsonld

Modified	2019-06-27T18:59:02Z
Acl	▸ 117f3470 ddad 11e8 a777 57160b196484.jsonld.acl
Described By	▸ 117f3470 ddad 11e8 a777 57160b196484.jsonld.meta
Type	▸ Document
	▸ Resource
	▸ Resource
Mtime	1561661942.694
Size	230
Is contains of	▸ inbox

▸ 258f9ff0 ebf3 11e8 a29e 5d8e3e616ac9.jsonld
▸ 3c7c6e50 d0e4 11ea ae0b b3344484b2fe.jsonld
▸ 43efa9b0 d0a6 11ea ae0b b3344484b2fe.jsonld
▸ a46c7020 d0a6 11ea ae0b b3344484b2fe.jsonld
▸ cd877350 f3e4 11e9 96bb 67f9593bc306.txt
▸ d262bc90 f3e4 11e9 96bb 67f9593bc306.txt

Figure 2.3: **Solid inbox - message content**. Screenshot of the current official inbox client application.



▸117f3470 ddad 11e8 a777 57160b196484.jsonld
▸258f9ff0 ebf3 11e8 a29e 5d8e3e616ac9.jsonld
▸3c7c6e50 d0e4 11ea ae0b b3344484b2fe.jsonld
▸43efa9b0 d0a6 11ea ae0b b3344484b2fe.jsonld
▸a46c7020 d0a6 11ea ae0b b3344484b2fe.jsonld

▾ **cd877350 f3e4 11e9 96bb 67f9593bc306.txt**

Modified	2019-10-21T09:26:03Z
Acl	▸cd877350 f3e4 11e9 96bb 67f9593bc306.txt.acl
Described By	▸cd877350 f3e4 11e9 96bb 67f9593bc306.txt.meta
Type	▾ **Document**

... ▸inbox
▸117f3470 ddad 11e8 a777 57160b196484.jsonld
▸cd877350 f3e4 11e9 96bb 67f9593bc306.txt
▸d262bc90 f3e4 11e9 96bb 67f9593bc306.txt
▸card
▸Preferences file
▸private Type Index.ttl
▸public Type Index.ttl

```
Outline.expand: Unable to fetch <http://www.w3.org/2007/ont/link>: Failed t
```

▸●Resource
▸●Resource

Mtime	1571649963.779
Size	103
Is contains of	▸inbox

▸d262bc90 f3e4 11e9 96bb 67f9593bc306.txt

2.3.3.3 Apache Marmotta

Apache Marmotta is an Open Platform for Linked Data [25]. Based on the Linked Media Framework project, it is an implementation of a Linked Data Platform. It is implemented as a Java Web Application [25].

Marmotta features Linked Data server for the Java EE stack, LDP, SPARQL and LDPath querying, caching and basic security mechanisms. The installation comes as a Java Web Archive (.war) file that has to be deployed on an application server (such as Tomcat) [25].

Marmotta is a robust, well-documented platform. However, based on the version list [26] and issue tracker [27], it does not appear to be under active development. The last version was published in June 2018 [26] and there are unresolved open issues from 2018 [27].

Apache Marmotta is published under the open-source Apache Software License, Version 2.0: `https://marmotta.apache.org/license.html`.

2.3.3.4 Carbon LDP

Carbon LDP is an enterprise server implementation of LDP. It aims to help engineers and artists to create web applications supporting LDP. It provides R/W access to RDF graph data using RESTful HTTP. Homepage: `https://carbonldp.com/`, GitHub page: `https://github.com/CarbonLDP`.

Carbon LDP supports REST API requests over HTTP. It supports JSON-LD, Turtle and RDF XML serializations. Data are stored in native RDF format - RDF triples in a NoSQL database. It also supports querying documents using SPARQL.

At first glance, Carbon LDP appears to be open source with commercial support (like e.g. Spring framework[14]). However, after a more thorough analysis Carbon LDP emerges as **a commercial product** without sources available and without an option for extensions. It does contain open source modules, such as a graph/document visualisation tool Workbench [15] and JavaScript SDK [16]. But **these modules do not meet requirements** for this theses and thus Carbon LDP is disqualified from further use.

2.4 Analysis results

In the analysis chapter, first, requirements for the solution were formulated. Use cases were derived from requirements to specify user actions. Based on the requirements and use cases, a broad analysis of existing solutions was conducted. Only solid-server and Apache Marmotta matched the first analysis and were selected for a more detailed review.

[14]`https://spring.io/`
[15]`https://github.com/CarbonLDP/carbonldp-workbench`
[16]`https://github.com/CarbonLDP/carbonldp-js-sdk`

After a more detailed analysis, the solid-server was selected as a sufficient solution to support the goals of the thesis. There is no need to implement a new server solution.

Chapter 3

Design

This chapter describes the architecture and technologies used for implementation. First, three prove-of-concept applications were developed. Second, based on the POC applications, an final client application was implemented.

3.1 inbox - LDN proof-of-concept

In this section, the design of the proof-of-concept (POC) application is described. First, the architecture is discussed, followed with a description of used technologies. This POC web application was created to investigate the LDN protocol. Its purpose is to test the architecture, technologies and the LDN, RDF libraries.

3.1.1 Architecture

Based on the actors of the LDN protocol, the application is divided into three modules (see Figure 1.3):

- **consumer** - LDN consumer + sender,
- **receiver** - LDN receiver,
- **target** - sample LDN target for inbox discovery.

The consumer is designed to communicate using REST API with any application compliant with the LDN specification. The receiver is a REST API server with in-memory DB implementation to receive and serve notifications using LDN protocol. The target is an implementation of the LDN target and works for the inbox discovery.

Both consumer and server are designed to have a back-end with HTML/CSS/JS front-end.

The consumer and receiver are an MVC architectures with RDF as a model and a separate service layer.

3.1.2 Technologies

After the analysis, JavaScript was chosen as the language for development. The main reasons are the requirement of a web client with desktop notification and the lack of back-end libraries. Only these four back-end libraries were discovered:

- **rdflib**[17] for Python,
- Java:
  - **Apache Jena**[18],
  - **Eclipse RDF4J**[19];
- **EasyRDF**[20] for PHP.

server For this application, the Node.js[21] server was used with the Express [22] web framework. Node.js is an asynchronous, event-driven JavaScript engine for back-end implementations. Express is a simple web framework built on top of Node.js.

front-end Front-end is made of HTML/CSS/JS pages. The Bootstrap framework was used to help with the UI design. Bootstrap is an open-source CSS/JS framework [28].

3.1.2.1 LD/RDF libraries

To read, write and manipulate Linked Data in RDF, a JavaScript library is needed. There are not many available libraries and the existing ones are still in development. For example, while working on this thesis, an issue[23] with the rdflib[24] was encountered - the library had troubles parsing JSON-LD.

The following LD/RDF libraries were considered and examined:

rdflib *rdflib.js* is an RDF JavaScript library. It supports R/W with RDF/XML and Turtle RDF serializations. It also supports reading of JSON-LD. Furthermore, it contains a fetch API to access RDF resources and local store with API to query the result.

[17] `https://rdflib.readthedocs.io/en/stable/`
[18] `https://jena.apache.org/`
[19] `https://rdf4j.org/`
[20] `https://www.easyrdf.org/`
[21] `https://nodejs.org/`
[22] `https://expressjs.com/`
[23] `https://github.com/linkeddata/rdflib.js/issues/364#issuecomment-546705383`
[24] `https://github.com/linkeddata/rdflib.js/`

Rdflib.js sources and documentation is accessible at `https://github.com/linkeddata/rdflib.js/`. It is also available as a npm[25] (JavaScript package manager) at `https://www.npmjs.com/package/rdflib`.

Tripledoc *Tripledoc* is an RDF JavaScript library to read, create and update documents on a Solid Pod [29]. It has a more intuitive and easy-to-understand interface than the rdflib, however fewer capabilities.

Please note that it was not available at the time of development of the inbox POC (first commit is Jul 17, 2019 [30]). For this reason it was not considered for this POC, but was later used in the inbox-client application. `https://vincenttunru.gitlab.io/tripledoc/`,

Shighl `https://github.com/scenaristeur/shighl`,

LDflex `https://github.com/LDflex/LDflex`.

The rdflib was chosen as the most mature technology. To access solid pods, the solid-auth-client[26] library is needed.

3.2 LDN-target

LDN-target is a simple web application that was separated from the original inbox POC. It was extracted to a new project and extended, so it is possible to document and deploy it separately. This way it can be a helpful contribution to the community.

3.2.1 Architecture

Based on the LDN target discovery protocol [15], the application has the following endpoints:

- HTTP HEAD that returns response with the Link header,
- HTTP GET that returns response with the Link header,
- HTTP GET that returns RDF content with serialization based on the HTTP content negotiation.

3.2.2 Technologies

The technology stack is the same as for the inbox POC.

[25]`https://docs.npmjs.com/about-npm/`
[26]`https://github.com/solid/solid-auth-client`

server For this application, the Node.js[27] server was used with the Express [28] web framework.

front-end Front-end is made of HTML pages with CSS styling.

3.3 js-notification-poc

js-notification-poc is an implementation of the JavaScript Notification API and Push API. Its development was intended to get familiar with the specifications and test the APIs. The Notification API part of the application is based on the API documentation [31], the second, Push API, is directly derived from an existing Push API example [6] (its source code available at `https://github.com/Spyna/push-notification-demo/`).

3.3.1 Architecture

This application is a web application. It is split to two conceptual parts, first being based on the JavaScript Notification API [31] and the second on the Push API [32]. Both are a simple web page, the whole application logic is written in JavaScript.

3.3.2 Technologies

The technology stack is the same as for the inbox POC.

server For this application, the Node.js[29] server was used with the Express [30] web framework.

front-end Front-end is made of HTML pages with CSS styling. The core functionality is written in JavaScript.

3.4 inbox-client

In this section, the design of the inbox-client is described. First, the architecture is discussed, followed with a description of used technologies.

[27]`https://nodejs.org/`
[28]`https://expressjs.com/`
[29]`https://nodejs.org/`
[30]`https://expressjs.com/`

3.4.1 Architecture

As for the inbox POC, JavaScript was used as the language for development. The main reasons are the requirement of a web client with desktop notification and the lack of back-end libraries.

The application logic is on the client side, with server used only for serving HTML/JS/CSS content. This is based on the findings from the development of the inbox POC section 3.1. The reason is that the solid pods require the solid-auth-client library for authentication. This is available only as a client-side browser library [33].

All data are stored in the solid pod and the client-side JavaScript is using the tripledoc library's fetch api to create authenticated HTTP requests to access the pod.

3.4.2 Technologies

The technology stack is the same as in the inbox POC (see subsection 3.1.2), with the main difference of using different LD/RDF library.

server For this application, the Node.js[31] server was used with the Express [32] web framework.

front-end Front-end is made of HTML/CSS/JS pages. The Bootstrap [28] framework was used to help with the UI design.

3.4.2.1 LD/RDF libraries

To read, write and manipulate Linked Data in RDF, a JavaScript library is needed. The *tripledoc* RDF JavaScript library [29] was chosen for this application. It has more intuitive and easy-to-understand interface than the rdflib used in the inbox POC.

To access solid pods, the solid-auth-client[33] library is required.

The *rdf-namespaces* library (`https://www.npmjs.com/package/rdf-namespaces` is used to help with RDF namespaces. This allows to use predefined constants like `rdfnamespaces.foaf.knows` instead of direct URLs like `http://xmlns.com/foaf/0.1/knows`.

[31]`https://nodejs.org/`

[32]`https://expressjs.com/`

[33]`https://github.com/solid/solid-auth-client`

Chapter 4

Implementation

This chapter covers the specifiv implementation details of all the developed applications. It covers both the initial proof-of-concept applications (section 3.1) and the following client application (section 3.4).

4.1 inbox - LDN proof-of-concept

First, a LDN proof-of-concept inbox application was created. The design and selection of technologies is described in section 3.1, here the implementation details are described.

The application flow is:

1. user visits consumer+sender application
2. the welcome page is shown: Figure 4.1
3. user inputs target URL or click on one from the last used list
4. consumer+sender module performs inbox discovery
5. if successful, consumer+sender reads messages from the inbox

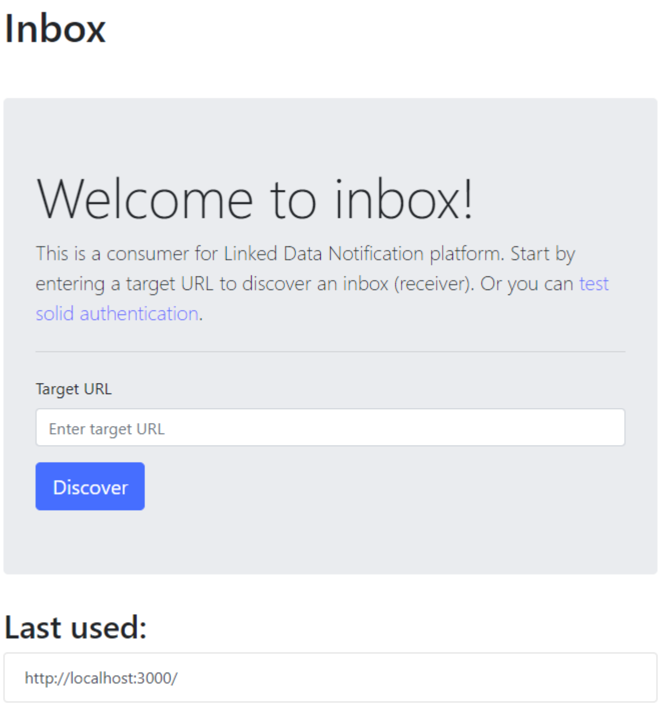
The inbox can be located anywhere. In this application, it is located at the receiver module.

4.1.1 consumer + sender

The main functionality is in the application services:

- `receiverServices.js`
  - `getNotifications(inboxUrl, callback)` retrieve all available notifications from the inbox/target, then execute callback

Figure 4.1: **inbox - index screen**. Screenshot of the index page


Inbox

Welcome to inbox!

This is a consumer for Linked Data Notification platform. Start by entering a target URL to discover an inbox (receiver). Or you can test solid authentication.

Target URL

Enter target URL

Discover

Last used:

http://localhost:3000/

  - `getNotificationById(id, callback)` retrieve a specific notification from the inbox/target by supplied id, then execute callback

- `targetService.js`
  - `discoverInboxAt(urlToExplore, callback)` attempt to discover inbox at the urlToExplore
  - `getInboxUrlFromRDF(urlToExplore, callback)` when no link header is available, try to get the inbox url from RDF content

4.1.2 receiver

The main functionality is in the application service:

- `notificationService.js`
  - `processMessage (notification)`
  - `createAllNotificationsResponse()` produce valid JSON-LD envelope for the notifications from DB
  - `getNotificationById()`

4.1.3 target

Target is a simple web page to try and showcase LDN discovery [15]. It is implemented as a node.js/express application that responds to HTTP requests.

As described at Figure 1.1.4, there are two options for discovery - using HTTP `Link` header[34], for both `HEAD` and `GET` methods, or by embedding RDF into the resource content.

For the first option (HTTP `Link` header), the application accepts both `GET` and `HEAD` HTTP requests:

```
const LINK_VALUE = '<' + INBOX_URL + '>;
rel="http://www.w3.org/ns/ldp#inbox"';

router.get('/', function (req, res, next) {
    res.set('Link', LINK_VALUE);
    res.render('index', {title: 'Inbox␣discovery␣demo'});
});

router.head('/', function (req, res, next) {
    res.set('Link', LINK_VALUE);
    res.status(200).end();
});
```

The second option is returning content based on the Accept header. It either returns RDF in JSON-LD or Turtle, or return HTML with embedded RDF:

```
// 2) RDF
router.get('/content', function (req, res, next) {
    // switch response based on Accept header and set the response content
    ↪ type accordingly
    res.format({
        // a) JSON with relation of type http://www.w3.org/ns/ldp#inbox
        'application/ld+json': function () {
            res.send(
                {
                    "@context": "http://www.w3.org/ns/ldp",
                    "@id": "https://tonda.solid.community/",
                    "inbox": "https://tonda.solid.community/inbox/"
                }
            )
        },

        // b) HTML <a> with rel="http://www.w3.org/ns/ldp#inbox"
        // c) HTML <link> with rel="http://www.w3.org/ns/ldp#inbox"
        // d) HTML <section> with property="http://www.w3.org/ns/ldp#inbox"
        'text/html': function () {
            res.render('contentWithRdf');
        },
```

[34] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link

```
        // e) text/turtle with <http://www.w3.org/ns/ldp#inbox> relation
        'text/turtle': function () {
            res.send("<https://tonda.solid.community/>␣<http://www.w3.org/
    ↪ ns/ldp#inbox>␣<https://tonda.solid.community/inbox/>␣.");
        },

        default: function () {
            // log the request and respond with 406
            res.status(406).send('Not␣Acceptable')
        }
    })
});
```

4.2 LDN-target POC

The implementation details are already described as a part of the subsection 4.1.3.

4.3 js-notification-poc

After the initial LDN proof-of-concept, another technology had to be analysed - system notifications. Since the JavaScript was chosen as the development language (see subsection 3.1.2), the JavaScript's notification API was examined in this JavaScript Notifications POC.

4.3.1 About JavaScript notification API

JavaScript notification API is divided into two conceptual, complementary parts. The first, Notification API[35], is a simple API to request permissions and create system notifications. The second, Push API[36], can be used to push notifications to web applications from a server.

The Push API uses 4 actors [6]:

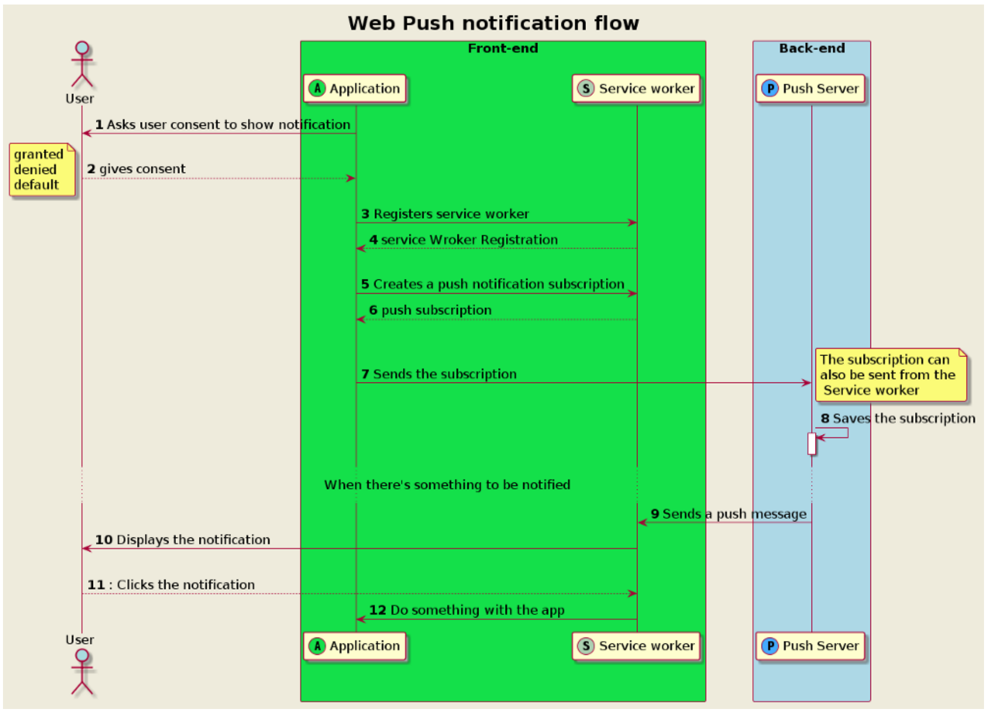
- **User** that wants to receive the notifications,
- **Application** that runs on the user-agent (typically the browser),
- **Service worker** that runs on the browser,
- **Push Server** that sends push messages to the service worker.

The whole interaction is illustrated in the Figure 4.2.

The service worker is also needed for Chrome on Android. However, as the time of writing, the Push API is still an experimental technology [32].

[35]https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API
[36]https://developer.mozilla.org/en-US/docs/Web/API/Push_API

Figure 4.2: **The push notification implementation flow.** JavaScript Push API overview with distinct actors - User, Application, Service worker and Push Server [6].


Web Push notification flow
Front-end
Back-end
User
Application
Service worker
Push Server
1 Asks user consent to show notification
granted
denied
default
2 gives consent
3 Registers service worker
4 service Wroker Registration
5 Creates a push notification subscription
6 push subscription
7 Sends the subscription
The subscription can also be sent from the Service worker
8 Saves the subscription
When there's something to be notified
9 Sends a push message
10 Displays the notification
11 : Clicks the notification
12 Do something with the app

4.3.2 POC description

This POC is a simple static page to test JavaScript notification api. It runs on node.js (express) server. The source code is available at `https://github.com/WhyINeedToFillUsername/js-notification-poc`. It includes installation and run documentation, as well as brief description.

The POC was created based on the MDN documentation. Quite simply, first the application first checks if the browser supports the Notification API:

```
// Let's check if the browser supports notifications
if (!('Notification' in window)) {
    const message = "This␣browser␣does␣not␣support␣notifications.";
    console.error(message);
} else {
// yes, supports, handle user's answer
}
```

After the user confirms that he wants to receive notifications, a notification is created with the following code:

```
function createSimpleNotification() {
      var img = '/images/fit.png';
      var text = 'Text␣of␣the␣notification.';
      var title = 'Hello␣there!␣Cool␣title!';
      const options = {
          body: text,
          icon: img,
          // A vibration pattern to run with the display of the notification.
          vibrate: [200, 100, 200],
          // An ID for a given notification that allows you to find, replace,
      ↪   or remove the notification using a script if necessary.
          tag: "new-product",
          image: img,

          // URL of an image to represent the notification when there is not
      ↪ enough space to display the notification itself
          badge:
          "https://spyna.it/icons/android-icon-192x192.png"
      };
      let notification = new Notification(title, options);
      // hook events, save reference etc.
  }
```

This creates a system notification with the requested parameters (the whole list is accessible at the MDN web docs: `https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerRegistration/showNotification#Parameters`).

4.3.2.1 Push API

The second part, Push API implementation, is directly derived from the *Push Notifications in JavaScript* article [6].

To make a full use of the JavaScript Notification + Push API, there are 6 steps [34]:

1. Check if Service Workers are supported
2. Check if the Push API is supported
3. Register a Service Worker
4. Request permission from the user
5. Subscribe the user and get the PushSubscription object
6. Send the PushSubscription object to your server

Also, to identify application and prevent spam for the clients, a VAPID key must be generated and used in the subscription. The web-push npm package was used for this purpose: `https://www.npmjs.com/package/web-push`.

Further implementation details are available in the *Push Notifications in JavaScript* article [6].

4.4 inbox-client

A final client inbox application was derived from the original proof-of-concept (section 3.1) application. The design and selection of technologies is described in section 3.4, here are the implementation details.

All the application's logic is in the front-end (browser) JavaScript. To structure the application, code is divided into separate modules based on their functionality. During the build, JS code is processed by the browserirfy plugin. A separate JS bundle file is created for each page. This way only used code is being loaded.

This application uses the solid-auth-client to authenticate and *tripledoc* RDF JavaScript library [29] to read and manipulate solid pod data.

4.4.1 Modules

Below is a list of the application JavaScript modules with description.

- `alerts.js` a helper module with `addAlert(level, message, autoDissmiss` function to create a user-friendly alerts

- `inbox.js` main module with the logic to add a watched inbox, retrieve notifications from it, show them to user and detect new messages. Every ten seconds, it loads notifications from the monitored inboxes and detects any new messages: `window.setInterval(loadNotifs, 1000 * 10);`

  - `loadMonitoredInboxesFromPod(webID)` retrieve monitored inboxes from the solid pod:
    1. fetch solid profile
    2. on the profile, get/create document that stores the watched inboxes list
    3. from the document, get all subjects of class schema.URL
    4. each has the url saved as type string; save them to memory and call `addInboxToShownList()`

- `inbox-detail.js` module for loading and formatting the inbox messages. Also contains methods for remove inbox from monitored list.

- `inbox-discover.js` module for LDP inbox discovery on the supplied LDP target

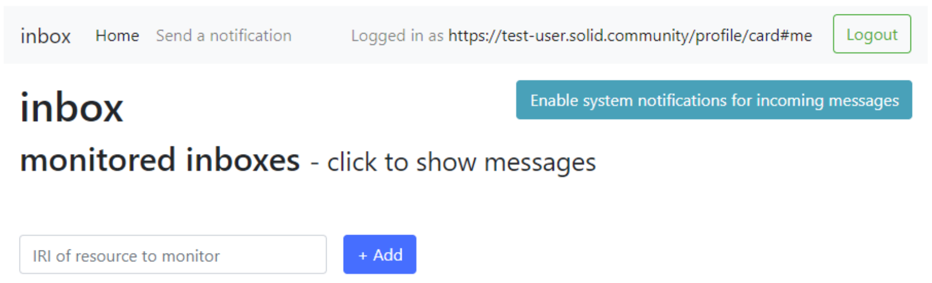
- `inbox-send.js` contains methods for sending the messages using LDN protocol (with AS format)

- `notifications.js` contains methods for handling system notifications using the JavaScript Notification API
- `pod.js` using the *tripledoc* library [29], this module contains methods for the communication and data retrieval from the solid pods.
  - `getFriends(webID)` retrieve a list of user contacts (predicate foaf.knows)
  - `getWatchedInboxesListDocument(profile)` retrieves the document where list of watched inboxes. If it does not exist in the client's profile, it is created by calling `initialiseWatchedInboxesList()` method.
  - `initialiseWatchedInboxesList(profile, typeIndex)` creates an empty document for storing the watched inboxes list
  - `addWatchedInbox(inbox, watchedInboxesListDoc)` stores a watched inbox into the supplied document
  - `removeWatchedInbox(inboxIRI, watchedInboxesListDoc)` removes a watched inbox from the supplied document
- `solid-login.js` contains logic for solid login, using the solid-auth-client library
- `solid-logout.js` functionality for the logout button

4.4.2 User interface

By using the Bootstrap CSS/JS framework [28], the UI is fully responsive. Below are few screenshots of the UI for illustration.

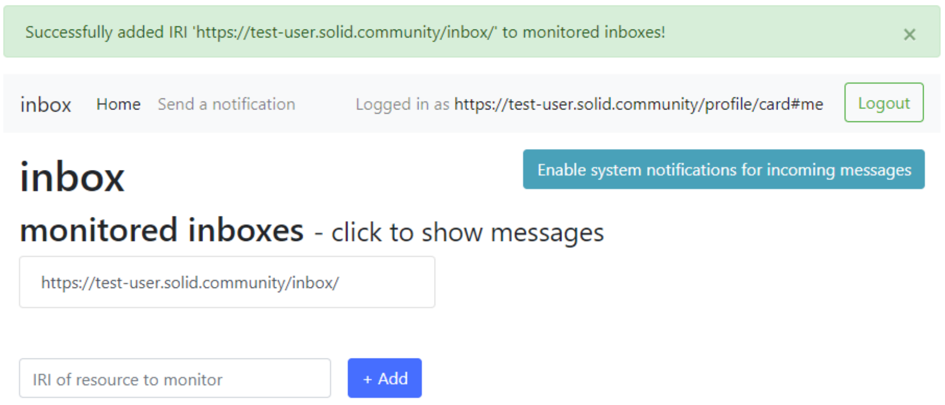
Figure 4.3: inbox-client screen 1 - watched inboxes


inbox
Home
Send a notification
Logged in as https://test-user.solid.community/profile/card#me
Logout
Enable system notifications for incoming messages
IRI of resource to monitor
+ Add

inbox

monitored inboxes - click to show messages

Figure 4.4: inbox-client screen 2 - added watched inbox


Successfully added IRI 'https://test-user.solid.community/inbox/' to monitored inboxes!

inbox
Home
Send a notification
Logout
Enable system notifications for incoming messages
+ Add
IRI of resource to monitor

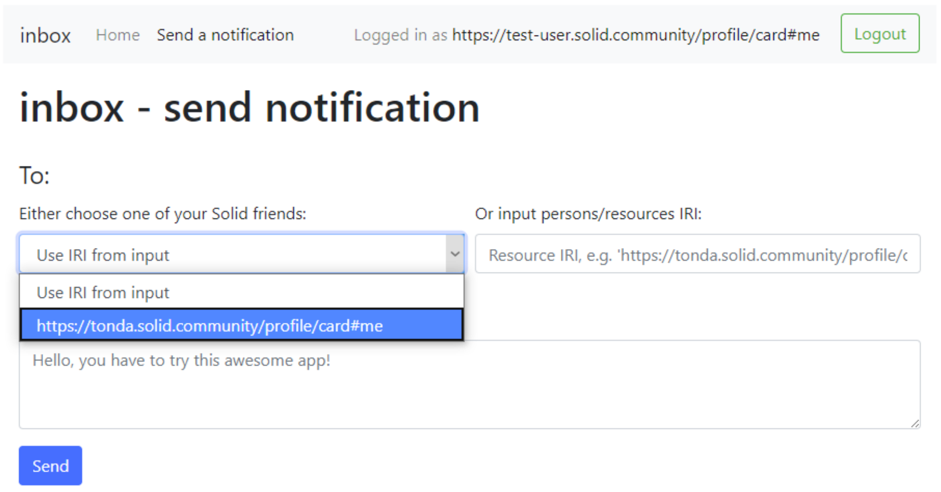
Logged in as https://test-user.solid.community/profile/card#me

inbox

monitored inboxes - click to show messages

https://test-user.solid.community/inbox/

Figure 4.5: inbox-client screen 3 - send message to a friend



Logged in as https://test-user.solid.community/profile/card#me

inbox - send notification

To:

Either choose one of your Solid friends:

Use IRI from input

Use IRI from input

https://tonda.solid.community/profile/card#me

Or input persons/resources IRI:

Resource IRI, e.g. 'https://tonda.solid.community/profile/c

Hello, you have to try this awesome app!

Chapter 5

Testing

This chapter describes how the developed applications were tested. First, the testing terms are defined. Second, information about testing each developed application is presented.

5.1 Terms definition

5.1.1 Unit tests

Unit tests are automated tests that verify behavior of the application's isolated methods. For example for a simple method `sum (a, b) {return a + b;}`, a unit test should verify the method's output. Unit tests should run with each application build. A build should fail when the tests don't pass. This way a developer can be informed about code changes that broke the desired behavior as soon as possible.

5.1.2 E2E tests

End-to-end (E2E) tests are automated comprehensive tests of the whole system. They should simulate behavior of a typical user of the tested system, they are usually based on the user scenarios. E2E tests of web applications are typically run with testing frameworks that control a testing instance of a browser. Principally they run longer than unit test, so they are usually run periodically and/or with each release candidate.

5.1.3 Usability testing

Usability testing is manual testing of the application. Its goal is to determine how usable for a typical user the system is. There are several types, such as cognitive walkthrough, heuristic evaluation and user testing [35].

As a part of this thesis, the cognitive walkthrough was conducted.

5.1.3.1 Cognitive walkthrough

Cognitive walkthrough testing is typically conducted by a developer/UX expert. This person uses user scenarios to walk through the application and uses his expertise to identify system's UX defects [35].

More specifically, the goal is to answer the following questions [36]

1. Will the user try and achieve the right outcome?
2. Will the user notice that the correct action is available to them?
3. Will the user associate the correct action with the outcome they expect to achieve?
4. If the correct action is performed; will the user see that progress is being made towards their intended outcome?

5.2 Application testing

5.2.1 inbox

Aim of this POC application was to investigate technologies and protocols. For this reason, application was tested mainly manually. To ensure LDN protocol compliance, the modules were successfully tested with the LDN test suite.

5.2.2 LDN-target

This POC application was tested manually together with the inbox POC. Furthermore, it was tested when discovering inbox in the inbox-client manuall testing.

5.2.3 js-notification

This POC application was subjected to manual testing.

5.2.4 inbox-client

As the main application of the thesis, most testing efforts were focused on the inbox-client application.

5.2.4.1 Unit tests

For this application, a unit testing framework research was conducted. From the candidates the Mocha testing framework (`https://mochajs.org/`) was selected. It is a JavaScript test framework running on Node.js.

However, during the test development it occurred that the inbox-client application has no backed services. Because of the used available solid and RDF libraries, all the application logic is on the front-end. The created front-end JavaScript methods are not suitable for unit tests - they either directly modify HTML DOM or rely on the solid session, which is created with the external solid-client-auth library.

The result is that the unit tests were considered, investigated, unit test frameworks researched. No suitable use was found for them and a decision was made not to use them for the application in its current design.

5.2.4.2 E2E tests

For the inbox-client, a E2E testing framework research was conducted. From the candidates, the Cypress E2E testing framework (`https://www.cypress.io/`) was selected as the most suitable framework [37]. It is an open source, JavaScript test framework running on Node.js and in a browser.

However a problem with the selected testing framework occurred. It does not support pop-up windows. And as of time of writing the thesis, a pop-up windows is the only way the solid-client-auth library is able to work. As a result, the testing framework is unable to log into the application This is an essential step in each UC and so the E2E tests are not part of the final solution.

5.2.4.3 Usability testing

The key part of the thesis is to prove that the solid inbox application can be user-friendly. To ensure this, a usability testing was conducted. Because the application's typical user is an experienced user, the cognitive walkthrough was selected as the most suitable usability testing method.

How the cognitive walkthrough was conducted The tester, an experienced front-end developer, followed the steps in the use cases section 2.2. At each step, the tester was trying to answer the questions defined at subsubsection 5.1.3.1. Afterwards, the tester summarized his findings and presented the results.

The complete testing records with the findings is available in the Appendix B.

Cognitive walkthrough summary The cognitive walkthrough confirmed the clear usage of the designed UI and the appropriate usability for the tested use cases. Only various minor UX issues were found and fixing them could increase the app's usability for a wider range of users.

Chapter 6

Documentation

This chapter provides documentation of all the created applications. It includes links to access source code, software prerequisites; installation, build and run steps for each.

6.1 Versioning - Git on GitHub

Source codes are versioned using Git on GitHub[37], published as open-source. GitHub is a public Git VCS with a web interface. Please note that the source codes at GitHub are subject to change. To access the code version that comes with this thesis, separate Git ”thesis-version” branches were made in each repository.

6.2 Deployment - Heroku

Applications were deployed using Heroku Cloud Platform: `https://www.heroku.com`. Please note that the platform is suspending applications that are inactive for a while, so the first time you open the application's page, you might experience a longer delay.

6.3 inbox

This section contains the inbox POC documentation.

6.3.1 Source code

Source code (with documentation) available at
`https://github.com/WhyINeedToFillUsername/inbox`

[37] `https://github.com/`

6.3.2 Requirements

You need to install node.js (with included npm).

6.3.3 Install

Run the following command in the **module's root folder**:

```
npm install
```

It installs all project dependencies, for details see `https://docs.npmjs.com/cli/install`.

6.3.4 Run

consumer listens on local port 8000, *receiver* on 3000. You can change that in their bin/www files. Run each module separately using following command in the **module's root folder**:

```
npm start
```

The *consumer* requires the *receiver* to be running.

6.3.5 Usage

You can try the *consumer* in your browser at `http://localhost:8000/`. When you post

```
{"@context": "https://www.w3.org/ns/activitystreams",
 "type": "Note",
 "to": ["https://chatty.example/ben/"],
 "attributedTo": "https://social.example/alyssa/",
 "content": "Say, do you think that Gary Webb and Jeffrey Epstein really
    ↪ killed themselves?"}
```

with `Content-Type: application/ld+json` to `http://localhost:5001/API/notifications`, the *receiver* will return 201 with location. You can then GET it at http://localhost:5001/API/notifications/xx. Or let the *consumer* display it in the browser at http://localhost:8000/notification/xx.

6.4 ldn-target-showcase

This section contains the LDN-target POC/showcase documentation. The app is a simple Linked Data Notification target implementation to test and showcase all discovery options. It is a node.js (express) app.

6.4.1 Source code

Source code (with documentation) available at
`https://github.com/WhyINeedToFillUsername/ldn-target-showcase`

6.4.2 Requirements

You need to install node.js (with included npm).

6.4.3 Install

Run the following command in the **root folder**:

```
npm install
```

It installs all project dependencies, for details see `https://docs.npmjs.com/cli/install`.

6.4.4 Run

Run with

```
npm start
```

Application is listening on the port 3000.

6.4.5 Usage

Open browser at `http://localhost:3000/`. You can see all the options to try out. Use e.g. curl or Postman to make HTTP request with various options.

For example this request:

```
GET /content HTTP/1.1
Accept: application/ld+json
Host: localhost:3000
```

gets this response:

```
HTTP/1.1 200 OK
Content-Type: application/ld+json; charset=utf-8
{
    "@context": "http://www.w3.org/ns/ldp",
    "@id": "https://tonda.solid.community/",
    "inbox": "https://tonda.solid.community/inbox/"
}
```

6.4.6 Live version

Application is deployed to: `https://ldn-target-showcase.herokuapp.com`.

6.5 js-notification-poc

js-notification-poc is an implementation of the JavaScript Notification API and Push API. It was developed to get familiar with the specifications and test the APIs.

6.5.1 Source code

Source code (with documentation) available at
`https://github.com/WhyINeedToFillUsername/js-notification-poc`

6.5.2 Requirements

You need to install node.js (with included npm).

6.5.3 Install

Run the following command in the **root folder**:

```
npm install
```

It installs all project dependencies, for details see `https://docs.npmjs.com/cli/install`.

6.5.4 Build

Run the following command in the **root folder**:

```
npm build
```

It uses browserirfy to build the project javascript bundle files.

6.5.5 Run

The node.js server is set to listen on local port 3001. You can change that in the bin/www files. Start it by this command:

```
npm start
```

Then go to http://localhost:3001/. Click on the "Enable notifications" button to request permission, and "Create notification" to show system notification.

Please note that request for notifications won't work in browser "private" mode.

6.5.6 Documentation

Code is self-explanatory with necessary comments. See `/public/javascripts/notifs.js` to how it works.

6.5.7 Live version

Application is deployed to: `https://js-notification-poc.herokuapp.com/`.

6.6 inbox-client

Documentation for inbox-client. The application is designed as a JavaScript client-side application with Node.js/express web framework back-end.

6.6.1 Source code

Source code (with documentation) available at
`https://github.com/WhyINeedToFillUsername/inbox-client`

6.6.2 Requirements

You need to install node.js (with included npm).

6.6.3 Install

Run the following command in the **root folder**:

```
npm install
```

It installs all project dependencies, for details see `https://docs.npmjs.com/cli/install`.

6.6.4 Build

Run the following command in the **root folder**:

```
npm build
```

It uses browserirfy to build the project javascript bundle files.

6.6.5 Run

The node.js server is set to listen on local port 3000. You can change that in the bin/www file. Start it by this command:

```
npm start
```

Then go to http://localhost:3000/. Click on the ”Enable notifications” button to request permission.

Please note that request for notifications won’t work in browser ”private” mode.

6.6.6 Usage

Open browser at `http://localhost:3000/`.

Please note that you have to add the running url ”http://localhost:3000” (exactly like this, with no trailing slash) to your trusted applications in your solid.community profile preferences with Read, Write and Append rights.

6.6.7 Documentation

All application logic is in JavaScript in the /modules folder.

6.6.8 Live version

Application is deployed to: `https://inbox-client.herokuapp.com/`.

Please note that you have to add ”https://inbox-client.herokuapp.com” (exactly like this, with no trailing slash) to your trusted applications in your solid.community profile preferences with Read, Write and Append rights.

Or you can use the following test user:

- **Username** ”test-user”
- **Password** ”SolidCommunity@2020”

Conclusion

In support of Web re-decentralization, the goal of this thesis was to get familiar with Linked Data, the RDF data model, the W3C Recommendations Linked Data Platform, Linked Data Notifications, ActivityPub, and the Solid project. Furthermore, the goal was to analyse current applications. Based on the analysis, an existing application was then to be enhanced or a new one implemented to produce a user-friendly messaging application.

First, the current technologies that address the centralization problems were studied and described.

Second, requirements for the result application were formulated. Actors in the system were identified and, from the requirements, a list of use cases was derived.

Based on the requirements and use cases, a thorough analysis of existing solutions and applications was conducted. The result is that the solid-server is a sufficient solution for the application server, but no user-friendly messaging client is available.

Next, the technologies to create the client application were researched and described in the Design section. Based on this research, multiple proof-of-concept applications were implemented to test these technologies for further use. The following proof-of-concept applications were created:

- **inbox** - Linked Data Notifications implementation
- **ldn-target-showcase** - LDN Target implementation
- **js-notification-poc** - JavaScript Notifications API and Push API implementation

inbox inbox is an implementation of the Linked Data Notifications protocol. The resulting application was successfully tested with LDN test suite. The source codes were published as open-source on GitHub.

ldn-target-showcase ldn-target-showcase is a simple POC implementation of the Linked Data Notification Target. Its goal is to showcase the LDN Discovery process to the community. The source code was published as open-source on GitHub and the application deployed for public access: `https://ldn-target-showcase.herokuapp.com`.

js-notification-poc js-notification-poc is an implementation of the JavaScript Notification API and Push API. It was developed to get familiar with the specifications and test the APIs. It can now serve as a technology showcase for the developer community. The source code was published as open-source on GitHub and the application deployed for public access: `https://js-notification-poc.herokuapp.com/`.

Based on the previous analysis and technology review, a resulting application was designed, implemented and tested:

inbox-client inbox-client is a new, user-friendly client application for Linked Data Notification and ActivityPub messaging. The application is designed as a JavaScript client-side application with Node.js back-end. It uses Solid WebID for user authentication and authorization, Solid Pods for data storage. The client allows its users to add their LDP inboxes for monitoring and get system notifications for new messages.

inbox-client has been tested, documented and published as open-source on GitHub. Furthermore, the application has been deployed for a real-world public use: `https://inbox-client.herokuapp.com/`.

Problems encountered

During the analysis and development, several major problems were encountered. The first problem occurred when developing the inbox POC. The RDF library rdflib.js had problems with JSON-LD serialization/deserialization. The problem was discussed on Gitter Solid Chat and reported on GitHub. In the inbox-client application, the problem was avoided by using another RDF library.

Another related problem is with the current technology and libraries for working with RDF data, Solid and Solid Pods. As the W3C recommendations for this area are relatively new, the technologies are still in rapid development and work-in-progress phases. Documentation is sparse and often insufficient.

The last major problem is with Solid rights management and the Solid Pod settings, as it is also in a WIP phase. Setting up user and application rights in the Solid settings are unreliable and not user-friendly. This affects user experience with the developed application and the application developers cannot directly influence this.

More specifically, when adding the new application to the trusted list in Solid preferences, one must not include a slash at the end of the submitted URL/IRI. This particular problem was resolved with the help of the Solid Gitter chat.

Future work

Future work should focus on further development of the inbox-client application. The main focus should be on better back-end capabilities. Because of the lack of server-side libraries for Solid, the complete logic has to be implemented on the front-end.

The future architecture could be an application with a modern JS framework (such as Angular or React) with a REST API back-end. This would allow for a better integration with existing applications like the social network Mastodon. To allow this integration, an HTTP signature for sending messages is required.

From the user-experience view, the client application should be extended with loading indicators for remote calls.

Also, future work should be done in cooperation with the developer/Solid community. This would allow further development of the existing libraries and tools.

Bibliography

[1] World Wide Web Consortium (W3C). *Social Network Silos [online].* [cit. 2020-07-12]. Available from: `https://www.w3.org/2010/Talks/0303-socialcloud-tbl/#(2)`

[2] World Wide Web Consortium (W3C) Working Group. *RDF 1.1 Primer [online].* [cit. 2020-07-17]. Available from: `https://www.w3.org/TR/rdf11-primer/#section-triple`

[3] World Wide Web Consortium (W3C), Linked Data Platform Working Group. *Linked Data Platform Resources [online].* [cit. 2020-07-17]. Available from: `https://www.w3.org/TR/ldp/#ldpr`

[4] World Wide Web Consortium (W3C). *Linked Data Notifications [online].* [cit. 2019-09-18]. Available from: `https://www.w3.org/TR/ldn/`

[5] Moody, G. *Tim Berners-Lee unveils next step for Solid, a decentralized Web platform with privacy built-in as standard [online].* [cit. 2020-07-22]. Available from: `https://www.privateinternetaccess.com/blog/tim-berners-lee-unveils-solid-a-decentralized-web-platform-with-privacy-built-in-as-standard/`

[6] Spyna, L. *Push Notifications in JavaScript? Yes, you can! [online].* [cit. 2020-07-24]. Available from: `https://itnext.io/an-introduction-to-web-push-notifications-a701783917ce`

[7] Abbate, J. E. *From ARPANET to Internet: A history of ARPA-sponsored computer networks, 1966-1988.* [cit. 2020-07-18].

[8] Hindman, M. *The Internet Trap: How the Digital Economy Builds Monopolies and Undermines Democracy (2018).* Princeton University Press, 2018, ISBN 9780691159263. Available from: `https://books.google.co.ke/books?id=hmmYDwAAQBAJ&printsec=frontcover&dq=monopoly+o`

f+the+internet&hl=en&sa=X&ved=0ahUKEwj6483N3ZvlAhWIh1wKHTTqAIkQ6AEIJzAA#v=onepage&q=monopoly%20&f=false

[9] Peng, Z. *Decentralized Internet [online]*. [cit. 2020-07-18]. Available from: https://www.cse.wustl.edu/~jain/cse570-19/ftp/decentrl/index.html

[10] World Wide Web Consortium (W3C). *Resource Description Framework (RDF) [online]*. [cit. 2020-03-29]. Available from: https://www.w3.org/RDF/

[11] W3C JSON-LD Working Group. *JSON for Linking Data [online]*. [cit. 2020-07-13]. Available from: https://json-ld.org/

[12] Berners-Lee, T. *Linked Data [online]*. World Wide Web Consortium (W3C), [cit. 2020-07-12]. Available from: https://www.w3.org/DesignIssues/LinkedData.html

[13] World Wide Web Consortium (W3C), Linked Data Platform Working Group. *Linked Data Platform [online]*. [cit. 2020-07-17]. Available from: https://www.w3.org/TR/ldp/

[14] World Wide Web Consortium (W3C), Linked Data Platform Working Group. *Linked Data Platform Containers [online]*. [cit. 2020-07-17]. Available from: https://www.w3.org/TR/ldp/#ldpc

[15] World Wide Web Consortium (W3C). *Linked Data Notifications - Discovery [online]*. [cit. 2020-07-24]. Available from: https://www.w3.org/TR/ldn/#discovery

[16] World Wide Web Consortium (W3C), Social Web Working Group. *Activity Streams 2.0 [online]*. [cit. 2020-07-20]. Available from: https://www.w3.org/TR/activitystreams-core/

[17] Castaño, A. *What is ActivityPub? [online]*. [cit. 2020-07-20]. Available from: https://alexcastano.com/what-is-activity-pub/

[18] *This Week in Solid 2019-12-12 [online]*. [cit. 2020-04-14]. Available from: https://solidproject.org/this-week-in-solid/2019-12-12

[19] *What is a Pod? [online]*. [cit. 2020-07-22]. Available from: https://solidproject.org/faqs#pod

[20] *About Carbon LDP implementation [online]*. [cit. 2020-06-07]. Available from: https://www.w3.org/wiki/LDP_Implementations#Carbon_LDP_.28Client_and_Server.29

[21] *maytkso, HTTP server and command-line RDF tool to get/send, serialise data. [online].* [cit. 2020-06-07]. Available from: `https://github.com/csarven/mayktso`

[22] Internet Engineering Task Force (IETF). *RFC 7252, The Constrained Application Protocol (CoAP) [online].* [cit. 2020-06-07]. Available from: `https://tools.ietf.org/html/rfc7252`

[23] *Solid [online].* [cit. 2020-07-29]. Available from: `https://solid.mit.edu/`

[24] *Solid license [online].* [cit. 2020-07-29]. Available from: `https://github.com/solid/node-solid-server/blob/master/LICENSE.md`

[25] The Apache Software Foundation. *Apache Marmotta [online].* [cit. 2020-07-29]. Available from: `https://marmotta.apache.org/`

[26] The Apache Software Foundation. *Download Apache Marmotta [online].* [cit. 2020-07-29]. Available from: `https://marmotta.apache.org/download.html`

[27] The Apache Software Foundation. *Apache Marmotta - Open issues [online].* [cit. 2020-07-29]. Available from: `https://issues.apache.org/jira/projects/MARMOTTA/issues/MARMOTTA-674?filter=allopenissues`

[28] *Bootstrap (front-end framework) [online].* [cit. 2020-07-24]. Available from: `https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)`

[29] Inrupt. *Tripledoc GitLab - project page [online].* [cit. 2020-07-24]. Available from: `https://gitlab.com/vincenttunru/tripledoc`

[30] Inrupt. *Tripledoc GitLab - first commit [online].* [cit. 2020-07-24]. Available from: `https://gitlab.com/vincenttunru/tripledoc/-/commit/802f3661920dddedf34120d4c07cacb8d4a49c94`

[31] Mozilla and individual contributors. *MDN web docs - Push API [online].* [cit. 2020-07-24]. Available from: `https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API`

[32] Mozilla and individual contributors. *MDN web docs - Push API [online].* [cit. 2020-07-24]. Available from: `https://developer.mozilla.org/en-US/docs/Web/API/Push_API`

[33] *Solid-auth-client GitHub [online].* [cit. 2020-07-29]. Available from: `https://github.com/solid/solid-auth-client`

[34] Copes, F. *The Push API Guide [online]*. [cit. 2020-07-24]. Available from: `https://flaviocopes.com/push-api/`

[35] Nielsen Norman Group. *Summary of Usability Inspection Methods [online]*. [cit. 2020-07-30]. Available from: `https://www.nngroup.com/articles/summary-of-usability-inspection-methods/`

[36] Interaction Design Foundation. *How to Conduct a Cognitive Walkthrough [online]*. [cit. 2020-07-30]. Available from: `https://www.interaction-design.org/literature/article/how-to-conduct-a-cognitive-walkthrough`

[37] SLANT TEAM. *JavaScript E2E test framework comparison [online]*. [cit. 2020-07-30]. Available from: `https://www.slant.co/versus/9648/20624/~nightwatch-js_vs_cypress`

Appendix A

Glossary

AP ActivityPub

AS Activity Streams

ČVUT České vysoké učení technické v Praze

GUI Graphical User Interface

IRI Internationalized Resource Identifier

JS JavaScript

LD Linked Data

LDN Linked Data Notifications

LDP Linked Data Platform

LDPC Linked Data Platform Container

LDPR Linked Data Platform Resources

NPM Node Package Manager

POC Proof of concept

R/W Read/Write

RDF Resource Description Framework

REST API Representational State Transfer Application Program Interface

SPARQL SPARQL Protocol and RDF Query Language

SSO Single Sign-On

URI Uniform Resource Identifier

UX	User Experience
VCS	Version Control System
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WIP	Work-In-Progress

Appendix B

Testing

This appendix contains detailed results of the cognitive walkthrough testing.

B.1 UC1 - start monitoring inbox

Consumer sets application so it monitors an LDP/AP inbox he has access to.

Table B.1: **Step A: User logs in using WebID.** User clicks on blue button with label Login and then the log in popup appears. User has to write/paste his WebID and to click the blue button with label Go.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		
Q2	Yes	Login button is the most visible element on the page.	There is missing direct sign-up button.
Q3	Yes	On the login screen the user can choose from three various login options.	
Q4	No	If the browser blocks the popup window the user will hardly see the issue.	The login process should not open a new tab. Instead it should open a modal window in the same tab (in the same DOM).

Table B.2: Step B: System shows form to add inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes	The “add inbox” is visible immediately after successful login.	
Q2	Yes	The Add button is the only blue element on the page, therefore the most visible element.	
Q3	Yes		The input field with the Add button could be positioned closer to the heading “monitored inboxes”.
Q4	Yes		

Table B.3: Step C: User inputs IRI of a resource and submits.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		The input field mentioned abbreviation IRI, which could not be clear for all users. There could be a help icon showing more information on a click/hover event.
Q2	Yes		
Q3	Yes		
Q4	Yes	The application shows confirmation/error notification after clicking on the Add button with describing the output of the action.	The notifications could be grouped if there are duplicates.

B.1.1 Use case summary

Possible UX issues were found on the login process, however it is an external functionality provided by the Solid framework. The process of adding new resource and getting proper feedback on the actions is clear for the end user.

Table B.4: Step D: System discovers resource's inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes	The user is informed via the confirmation notification.	
Q2	Yes	The user can easily see the list of the discovered and monitored resources on the same screen.	
Q3	Yes		
Q4	Yes		

Table B.5: Step E: System starts monitoring messages coming to the inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		
Q2	Yes	The user can easily see the list of the discovered and monitored resources on the same screen.	
Q3	Yes		
Q4	Yes		

B.2 UC2 - Stop monitoring inbox

User can turn off monitoring of an inbox he has previously selected for monitoring.

Step A: User logs in using WebID - already tested in UC1/A.

Table B.6: Step B: System shows list of monitored inboxes.

Question	Answer	Comment	Recommendation
Q1	Yes	The list of monitored inboxes is below the “monitored inboxes” heading.	
Q2	Yes		
Q3	Yes		
Q4	Yes		

Table B.7: Step C: User chooses inbox to stop being monitored.

Question	Answer	Comment	Recommendation
Q1	Yes	The resource card has on-hover effect changing the cursor icon to pointer which is helping user to understand the element as clickable.	
Q2	Yes		Text “click to show messages” could be different to not limit only to show messages, but resource’s detail in general.
Q3	Yes	The “Stop monitoring this inbox” button is clear and correctly colored with red color.	
Q4	Yes	The user will see the list of monitored resources without the deleted item.	There could be a notification confirming the successful deletion (would be beneficial in case of dozen resources in the lost)

Table B.8: Step D: System stops monitoring incoming messages to the chosen inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		
Q2	Yes		
Q3	Yes		
Q4	Yes	The user will see the list of monitored resources without the deleted item.	There could be a notification confirming the successful deletion (would be beneficial in case of dozen resources on the list of monitored resources).

B.3 UC3 - Read list of messages

Consumer can read a list of incoming messages that he has access to in a selected inbox.

Step A: User logs in using WebID - already tested in UC1/A.

Step B: System shows list of monitored inboxes - already tested in UC2/B.

Table B.9: Step C: User chooses inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes	The resource card has on-hover effect changing the cursor icon to pointer which is helping user to understand the element as clickable.	
Q2	Yes	Text “click to show messages” above the list of items clearly communicates the way to the desired action to read a list of messages.	
Q3	Yes		
Q4	Yes	After clicking on the inbox card, it shows its detail page.	

Table B.10: Step D: System shows list of incoming messages.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		In case of zero messages, there should be information explaining it, otherwise the user could get confused.
Q2	Yes	User can easily see the list of available messages under the “notifications” heading.	
Q3	Yes		
Q4	Yes		

B.4 UC4 - Read detail of a received message

Consumer can read the complete content of a received message that he has access to.

Step A: User logs in using WebID. Already tested in UC1.A.

Step B: User sees list of his incoming messages. Already tested in UC3.D.

Table B.11: Step C: User can open and read the full content of the incoming message.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes	The action is obvious thanks to the chevron-down icon on the right side of each message card.In addition the cursor is changed to a pointer on the hover event.	
Q2	Yes		
Q3	Yes		
Q4	Yes		The chevron-down icon could change to chevron-up when the message's detail is expanded to clearly communicate the possibility to close its detail.

B.5 UC5 - Receive a system notification on a new message

Consumer gets a system notification, such as a pop-up message, when he receives a new message to one of his monitored inboxes.

Step A: User logs in using WebID. Already tested in UC1.A.

Table B.12: Step B: System receives a message for the user to a monitored inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		
Q2	Yes		
Q3	Yes		
Q4	Yes	The steps correctness can be confirmed by the step C or checking the inbox detail (the list of the incoming message).	

Table B.13: Step C: System shows a pop-up system notification to the user.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes	The notification is shown on the top of the application as the rest of the system notifications.	
Q2	Yes	The notification contains a link to the inbox containing the message.	On clicking the notification's link the correct message could be auto expanded on the inbox page.
Q3	Yes		
Q4	Yes	The message has correct label (e.g. "New message in inbox `https://test-user2.solid.community/inbox/` !") and a correct informative color background (blue).	

B.5.1 Use case summary

The use case of receiving a system notification in case of a new message has a clear behavior.

B.6 UC6/7 - Send a message to a person/resource from contact list using LDN/AP

Sender sends a message to an actor from contact list using Linked Data Notification/ActivityPub protocol. The actor can be a person or some other resource (like company, website, etc.) identified by IRI.

Step A: User logs in using WebID. Already tested in UC1.A.

Table B.14: Step B: User sees list of his contacts.

Question	Answer	Comment	Recommendation
Q1	Yes		
Q2	Yes	The list is available on the “Send notification” page via select box.	For dozens of contacts the select box could include a text search supporting the search help functionality.
Q3	Yes		
Q4	Yes	After choosing the desired contact the user will clearly see that the option is selected.	

Table B.15: Step C: User clicks on the action ”send message”.

Question	Answer	Comment	Recommendation
Q1	Yes	The option is available as “Send a notification” in the main navigation.	
Q2	Yes		
Q3	Yes		
Q4	Yes	The user immediately sees the message editor after clicking the “Send a notification” option in the main navigation.	

B.6.1 Use case summary

From users perspective the process of sending a message to a known contact is smooth. The only UX issue is the complicated process of adding a new contact through the 3rd party form available at `https://taisukef.github.io/solid-addfriend/`.

Table B.16: Step D: User enters a content of the message.

Question	Answer	Comment	Recommendation
Q1	Yes		
Q2	Yes		
Q3	Yes		
Q4	Yes	The action consists of basic text writing into a textarea field. Submitting the empty message correctly informs about needs of writing any text into the "Notification content" textarea.	The character counter could have an extra UX value for the user. Plus there is not information about the maximum text length of the message.

Table B.17: Step E: User submits the message.

Question	Answer	Comment	Recommendation
Q1	Yes	The send button has a blue color and therefore is the most visible element in the form (correct behavior for the "Call to action" buttons).	
Q2	Yes		
Q3	Yes		
Q4	Yes	The application correctly shows the green system notification confirming about the successfully sending the message.	

Table B.18: Step F: System sends the message to the resource's inbox.

Question	**Answer**	**Comment**	**Recommendation**
Q1	Yes		
Q2	Yes		
Q3	Yes	The form content is reseted after the submit, which is a correct behavior allowing user to write another message without any extra action needed.	
Q4	Yes	The application correctly shows the green system notification confirming about the successfully sending the message.	

B.7 UC8/9 - Send a message to an unknown person/resource using LDN/AP

Sender sends a message to an actor that is not in his contact list using LDN/AP protocol. The actor can be a person or some other resource (like company, website, etc.) identified by IRI.

Step A: User logs in using WebID. Already tested in UC1.A.

Step B: User opens a new message form. Already tested in UC6/7.C.

Table B.19: Step C: User enters the receiver's IRI.

Question	Answer	Comment	Recommendation
Q1	Yes	The input field is clearly visible and correctly position after the "known contacts" field.	In case of wrong input the field should inform about the correct IRI syntax.
Q2	Yes		
Q3	Yes		
Q4	Yes		

Step D: User enters content of the message. Already tested in UC6/7.D.

Table B.20: Step E: User submits the message.

Question	Answer	Comment	Recommendation
Q1	Yes	The send button has a blue color and therefore is the most visible element in the form (correct behavior for the "Call to action" buttons).	
Q2	Yes		
Q3	Yes		
Q4	Yes	The application correctly shows the green system notification confirming about the successfully sending the message.	In case of not existing receiver's IRI, two system notifications are fired. One message explaining the issue would be sufficient.

Step F: System sends the message to the resource's inbox. Already tested in UC6/7.F.

B.7.1 Use case summary

Sending a message to an unknown person/resource using LDN/AP could be problematic in case if users does not know the syntax of the Internationalized Resource Identifiers. It could be easily fix by showing an explanation (or a link to the external resource) in case wrong IRI entered. Otherwise the use case has a smooth UX.

Appendix C

Attached medium content

```
apps ........................ source codes of the developed applications
  inbox ................................ inbox application source code
  ldn-target ..................... ldn-target application source code
  js-notification ............ js-notification application source code
  inbox-client .................. inbox-client application source code
text ...................................... diploma thesis document
  src ............................... document source code in LaTeX
  thesis.pdf ................................... document in PDF
readme.txt ............................ medium content description
```