Evaluating Solid for Social Applications with Many Users

Martin Bruland

Master's Thesis
Department of Information Science and Media Studies
University of Bergen

June 1, 2022

Acknowledgements

Many thanks to my supervisor Csaba Veres, for consistent guidance and ideas throughout this work!

Martin Bruland
Bergen, 01.06.2022

Abstract

The world wide web has made it possible to access and share information anywhere and at any time. Commerce, entertainment and social networking are just a few examples of content that is accessible on the web. Solid is a project led by Tim Berners Lee that aims to provide a solution to the data privacy problem on the web, by separating data from the applications. This master thesis explores the limits of using the Solid platform in applications, that rely on data from multiple users at the same time. The retrieval and aggregation process of managing data from Solid PODS was benchmarked and compared to an alternative implementation, where copies of data from the users POD is stored in an RDF Triplestore, from which it is aggregated and retrieved.

Contents

List of Figures

Chapter 1

Introduction

"The web is for everyone and collectively we hold the power to change it. It wont be easy. But if we dream a little and work a lot, we can get the web we want." - Tim Berners Lee[2]

Over the past 30 years, the world wide web has evolved into a global information system, that is currently accessible to half of the earths population and with an ambitious goal to reach everyone by 2030 according to the Web Foundation.[3] The web has made it possible to access and share information with anyone, anywhere and at any time. As a result of this, the public is able to access most services from their government and different companies around the world, without leaving their homes. The value of this was especially tested under the covid-19 pandemic, where most countries had initiated country wide lockdowns and physical areas (stores, workplaces, events, etc) was either closed down or had limited access.

Many web companies are making money from selling large chunks of aggregated data to other companies. This data is typically in the shape of anonymous data profiles, where behavioural data has been collected from the users interaction with their website. These data profiles are often generalized by common characteristics among users, such as age, gender, location and interests. This type of business model are enabling companies to provide free web content to its users, in exchange for their consent to collect and sell the data. However an ethical concern arises when it becomes unclear what type of data has actually been collected about the users, the degree of its anonymity and how it is protected against malicious breaches and data thefts.[4; 5]

Because of these threats, the focus on data privacy has increased and especially as more and more services that requires transmission of sensitive user data has been made available online. To deal with this acknowledged issue, governments have implemented law binding regulations with the purpose of enforcing transparency about what data is collected, how it is stored and used. Not complying to these regulations is penalised by fines, which gives the companies incentive to prioritise their users privacy. The 'General Data Protection Regulation' (GDPR) in the EU and the 'California Consumer Privacy Act' (CCPA) in California, are two examples. These regulations give users the legal rights to be informed about what is collected and for it to be deleted if requested. According to a report from the European Union Commission about the results after

two years of the GDPR, the implementation has been successful. People have become more aware of data privacy and companies are improving their infrastructures to comply with the new guidelines.[6]

However. Regulations does not deal with the root of the problem, and the process of exercising these rights can be a tedious process for everyday web users. Not every privacy policy is easy to read due to factors such as difficult wording and the requirement of reading a numerous pages long document before using a website (which can often be a reason for uninformed consents). Another factor can be poor user experience that makes it difficult to find the privacy policy or to set the consent settings. A more fundamental problem is the bad architecture of how data is distributed on the web, as several websites often stores identical data about the same user. One could argue that the risk of data being stolen or misused increases with the number of datastores that contains the same data. Especially as the technologies and techniques used in data breaches often evolves in line with data management security. Finding solutions to these issues is a part of what drives the development of the technologies that will be used in the future web 3.0.

The Solid Project is one of these technologies and is the general motivation for this thesis. It is a project that is led by the inventor of the world wide web Tim Berners-Lee, and aims to return data privacy and ownership to the users by separating the data from the applications.[7]

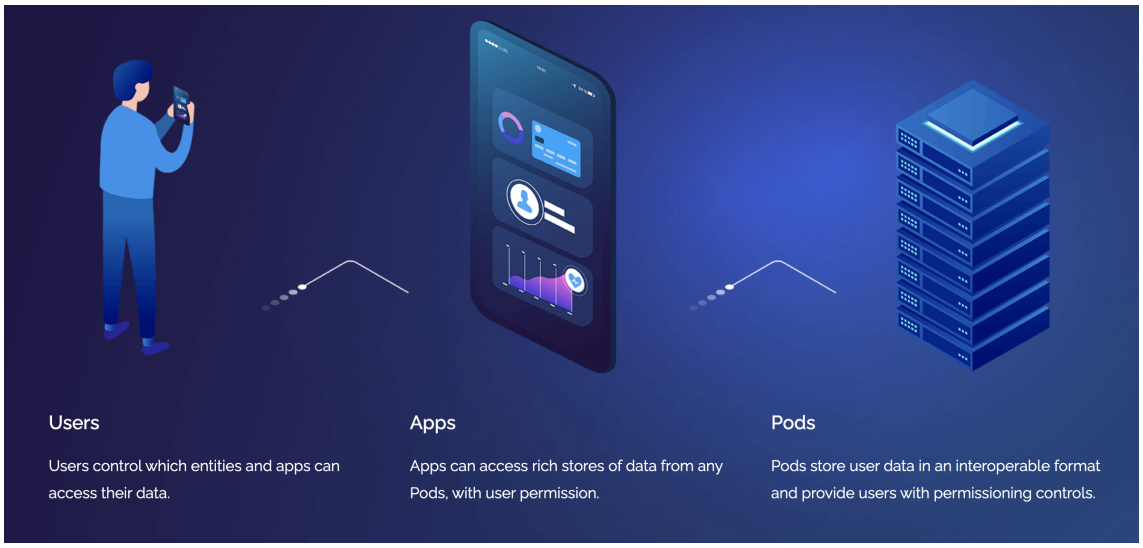


Figure 1.1: Visual representation of the Solid architecture by Inrupt.[1]

In short, Solid is a Protocol that provides a solution called Personal Online Datastore (POD), that allows users to store their own data and that Solid compliant applications are able to communicate with. This is used as an alternative to letting websites control where and how the users data is stored. Each user has complete ownership of the data that resides within the users POD, and applications would need to be granted access to the specific data that it wants to use. The users can revoke access to their data from within a POD Browser, and will stop the application from getting access to it.[8]

Implementing Solid in applications can be beneficial for both users and application providers. Storing data in one location instead of several individual datastores, would make it easier to keep it updated. This can give data more value to the applications that have access to it. By allowing multiple applications to reuse the same data (which is made possible through the interoperable Linked Data model), keeps the user from reentering it at every website. And by letting the user decide how the data is stored (either at a POD Provider or by hosting their own POD), puts them in greater control of the security measures placed on the data. This topic will be further explored in chapter 2.

1.1 Problem Space

Studies performed by Google have shown that users of the web are expecting content to load within 1-2 seconds, and that an increasing number of users will not continue to use the service the longer it takes to load.[9] This behaviour creates a performance requirement that the Solid platform must meet, in order for applications to be wanting to implement it.

When data is separated to each individual users Solid POD, an application will need to retrieve the data from each of these locations. Retrieving data from only one POD would very likely be performing at a satisfactory level and within the two second mark. But many applications will require to display data from several users. And since data is stored in different locations, it must be aggregated after it is retrieved. From this there is reason to assume that Solid applications with enough users will experience a scaling problem. This is an assumption that is debated within the Solid community forums.[10; 11]

1.2 Objectives

The goal of this thesis is first to identify what can be expected from the performance of the Solid architecture, and is investigated through the first research question. A bookmarking website (called Lexitags) that manages data through the Solid platform was created to facilitate this. The first research question is:

RQ 1: How does Solid perform as a data platform for Lexitags ?

> **1.A)** How is the retrieval time affected by increasing the number of users and bookmarks ?
>
> **1.B)** How is data aggregation impacted by increasing the number of users and bookmarks ?

Through the second research question, Solid is implemented in an alternative architecture, where copies of data from the users POD is stored in an RDF Triplestore, from which it is aggregated and retrieved. The goal of this is: 1) It explores an approach that can allow for faster aggregation and retrieval of trending data among users of a Solid application. 2) It allows for measuring and comparing the results found through the first research question, to a centralised datastore where all the data is stored in one location. This comparison was used as part of an evaluation of the alternative approach,

and is presented in chapter 7. The second research question is:

RQ 2: How does the performance compare to an alternative implementation where data from each Solid POD is stored in an RDF Triplestore?

1.3 Contribution

The results produced from the first research question shows how the latency of retrieving data from multiple Solid PODS is affected when the number of users increase. It also shows how performing the aggregation task in the client can add to the already increased loading time. Comparing the results to the RDF Triplestore gives a clearer picture of its performance. These results can be valuable to other developers who are considering to use Solid in their applications, in that it shows what to expect when it is implemented directly in the client application.

The second research question contributes to knowledge by creating and exploring an alternative implementation of Solid PODS, that allows for storing copies of the data in additional datastores. Specifically by documenting its advantages, limitations and performance in contrast to the direct client side implementation of Solid. This artifact can be relevant for developers and implementers, that are interested in alternative approaches in how to aggregate and retrieve data from a larger number PODS. Both the process and evaluation of the artifact can be of interest to those who seek to build other implementations for solving the same problem.

1.4 Thesis outline

The rest of this thesis is structured as follows:

Chapter 2 - Background: gives a theoretic overview about the world wide web, how Solid PODS work and its role in the next stage of the web. In addition to this, the tools that was used throughout the work of this thesis will be introduced.

Chapter 3 - Methodology: explains the methodologies that was used to investigate the research questions.

Chapter 4 - Methods: introduces the artifacts that was created and the experiments that they were used in.

Chapter 5 - Results: presents the results from performing the experiments, that measured the retrieval and aggregation time of both Solid PODS and the Triplestore.

Chapter 6 - Evaluation and Discussion: presents an evaluation and discussion of the artifacts, that is based on the performance and its overall architecture in regards to implementation in applications.

Chapter 7 - Conclusion and Future Work: summarises the findings and concludes this thesis. Reflections on future work is presented as well.

Chapter 2

Background

This chapter will first present the three stages of the world wide web (known as Web 1.0, Web 2.0 and Web 3.0) and how the Solid Protocol is Tim Berners Lees vision of what the future Web 3.0 can look like. It will then explain the technologies that makes up the Solid Protocol in more detail and how it works. Examples of existing adaptations and applications that has integrated Solid will then be introduced. At last an overview of the RDF Triplestore and the SPARQL Query Language is given, as well as related work and the tools that was used throughout this work.

2.1 The World Wide Web

The first version of the world wide web (which is commonly dubbed Web 1.0) and its three underlying technologies, was created in 1989 by Tim Berners Lee. Creating webpages was made possible through the HyperText Markup Language and made accessible to other users through the Uniform Resource Locator (URL) and the HyperText Transfer Protocol.[12] The web consisted of static web pages that contained information which could only be provided by the website administrators.

WEB 2.0
With Web 2.0, a new type of website that the average user could interact with emerged. Websites now offered comment sections, where its visitors could write text or even share images that was possible for other users to view and interact with. This possibility for interaction and user generated content gave rise to other forums, blogs and then social networks like MySpace and the ones we have today such as Facebook and Twitter.

The option to create an account with the websites became possible, and allowed for information about the users to be remembered. The users would typically need to provide their email address or a unique username, a password, and sometimes additional information such as their name, age, gender and address. As web security became better, companies could offer online stores, banking and government public services. This required storing credit card information and other sensitive data.

This data provided by the users are commonly stored in centralised datastores, that is controlled by the company. Controlling the data, allows the companies to decide how it is stored and protected, and its distance from the application that will use it. The com-

panies are thereby also in control of the applications performance and can more easily optimize it. Since the data is stored in these centralised datastores, it cant be used by other applications.

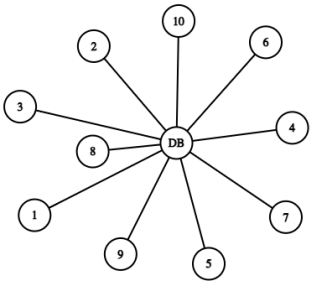


Figure 2.1: Visual representation of what a centralised datastore can look like.

For companies to understand how users interact with their website, they might collect behavioural data about their interactions. Such data could for example be used to measure how long it takes for a user to move around and navigate the page. The rise of smartphones made the web accessible everywhere, and its sensors additionally made it possible to collect other types of behavioural data about users. For example such as if one is left or right handed, how they hold their phone and the accurate location of where the service is used.

A common practice for web companies to be able to offer free content, is to create general data profiles from the collected data that is then sold to other third parties. Its not always clear what shape these data profiles are in or how detailed it is in terms of whether or not it can be used to identify an individual person. Sharing data to several third party actors can additionally increase the risk of it being misused by other entities.

As more and more sensitive and personal data is transmitted on the web, there has been an increasing concern about data privacy in regards to what is collected and how it is used, as well as measures against data breaches and theft. This has resulted in data privacy acts such as the the 'General Data Protection Regulation' (GDPR) in the EU. Acts similar to the GDPR is meant to regulate the web by giving its users the legal rights to be informed about what is collected and for it to be deleted if requested. Companies are fined if they do not have the necessary means to protect their users data, which creates an incentive to do just that. For example, Meta Inc. (Facebook) was recently fined 19 million dollars as a result of failing to prevent several data breaches in 2018, where access tokens from 50 million accounts was retrieved and could be used to gain access to their users accounts.[13; 14]

WEB 3.0

The current state of the world wide web is still in the Web 2.0 stage, but the technologies that will define the future Web 3.0 is already being developed, tested and used today. Web 3.0 is assumed to be a web where data will be owned and controlled by the

users instead of the companies, which allows them to control and affect how the data is secured and used. The returned ownership and control of data, allows the users themselves (instead of the companies) to monetize from selling their data. Storing your own data in a chosen datastore instead of a single centralised datastore that is controlled by a company is known as decentralization. This way of storing data makes it possible for any application to reuse the same data, as long as it is in a format that can be understood by the application. This can reduce the data duplication that is occurring on the web today, where companies are storing identical data about the users in their datastores. It also makes it much easier for the users to update their data whenever they get a new email, phone number or street address, which reduces inaccurate and outdated data. This in turn will be more valuable for the companies as well.

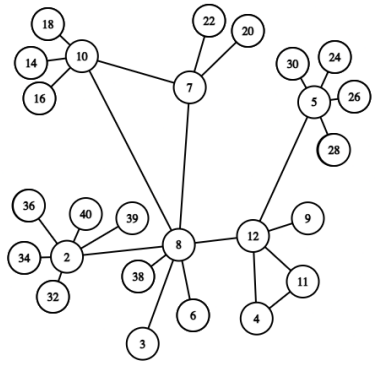


Figure 2.2: Visual representation of what a decentralised datastore can look like.

Web 3.0 was already envisioned by Tim Berners Lee in the 90s, which he called 'The Semantic Web'.[15] In this version of the web, data would have clear meaning and could be understood by humans and machines alike. The data would be structured as Linked Data in a giant web of data, that allowed it to be reused by any system or application. With this meaning the data would become more valuable, which opens up for more advanced artificial intelligence and machine learning algorithms, that can be used in new ways to help its users and give more accurate predictions.

Separating data into multiple datastores can introduce several challenges that can affect the applications performance and user experience, which needs to be dealt with through technological advancements. Separating data can for example introduce higher latency and inaccessible datastores due to system downtime. However it can also ensure that the system up-time increases, as measures could be integrated for it to work even if some of the data is inaccessible. This is in contrast to a centralised datastore where all the data could potentially be inaccessible. Separating data into different datastores can also make it difficult to perform complex user wide queries, that can sort and filter data based on what is needed to retrieve. Additionally it can for some companies be beneficial, in that it can reduce internal server and database administration work and costs. However it is very likely that a company would still need their own centralised datastores, for other types of data that they own or need to control for their service to work.

The Solid Project is Tim Berners Lees vision of what Web 3.0 could look like. It is a

web where data has meaning, where it can be stored anywhere and where it can be used by any application once access has been granted to it. The next section will present this topic in more detail.

2.2 The Solid Project

Solid (from Social Linked Data) is a project being led by the founder of the world wide web, Tim Berners Lee. It is a specification consisting of several different technologies that together makes it possible to separate users data from applications on the web. These technologies are based on open web standards that is already integrated elsewhere on the web, which makes Solid an interoperable platform that can be used across applications. Its architecture allows multiple applications to be given access and reuse the exact same data, without the user having to re-enter it at each service. The purpose of storing and managing data through the Solid Protocol is have a solution that better supports data privacy on the web, by returning the control of data to the users. The fundamental components that makes up the Solid platform is WebIDs, PODs, POD Servers, Web Access Control and Access Control List, Linked Data and URIs, and the HTTP Protocol.[16]

Separating data from the applications is achieved by storing the individual users data in the their own POD, which stands for Personal Online Datastore. A POD is its own separate datastore that can be accessed through a POD Server, that is either hosted by the users themselves or by a POD Provider. To identify and authorize users, Solid is using its own implementation of the OpenID Connect standard that is built on-top of the OAuth 2.0 standard, and which they have called Solid OpenID Connect.[17] This technology is implemented by Solid Identity Providers and allows for giving users a unique WebID, that is used to describe unique entities such as people, companies, applications, etc through an RDF Profile Document. Creating a new Solid WebID is typically available at each Pod Provider, but they often also let you create a POD that can be tied to an existing Solid WebID from another Solid Identity Provider. Some existing providers that offer both Solid WebIDs and PODS are: Inrupt.com [1], Inrupt.net [2], and SolidCommunity.net [3].

Any type of data and files can be stored in a POD with an interoperable format that can be used by all applications. Everything in the POD is identified by its own Unique Resource Identifier (URI), and connected through the Linked Data structure, where data can be described and connected as semantic triples by using the Resource Description Framework (RDF). A semantic triple consists of a subject, predicate and object, where the predicate is the connector between two resources. The predicate describes the relation that the subject has to the object. For example we can say that a folder contains a document by the following triple:

Subject: <http://example.org/folder>.
Predicate: <http://www.w3.org/ns/ldpcontains>.

[1] https://inrupt.com/
[2] https://inrupt.net/
[3] https://solidcommunity.net/

Object: <http://example.org/document>.

The predicate in this example originates from the public 'Linked Data Vocabulary' [4], which can be considered as a type of dictionary for describing items related to concepts within Linked Data. There are several other known vocabularies available that describe different concepts, and by using the same predicates or terms between applications makes the data interoperable among these applications. Since data in the PODs are described and given semantic meaning as part of the fundamental data flow, it additionally removes ambiguity and thereby makes it more valuable and understandable to both other users and automatic systems. Linking resources together forms a graph of data with nodes (subjects and objects) and edges (predicates). This connectivity makes it possible to follow the data in order to find out more about it. For example: data in one POD can be connected to data in another POD.

Whenever a user starts using a Solid application they need to login with their POD credentials and grant access to the data, in order for the application to be able to access it. Controlling access to data in the POD is made possible by the Web Access Control (WAC) system [5], which connects each resource in the POD to its own Access Control List (ACL) resource that contains the specific rules to which applications or individuals has access to the data. If an application does not have the necessary features to revoke its access to the POD, then different POD Browsers can be used to manage this. For example the 'Inrupt PodBrowser' [6].

Clients (websites, applications and servers) can communicate with PODS through the HTTP Protocol [7], as this is a requirement from the Solid specification to be supported by POD Servers. POD Servers must also support Linked Data Notifications that allows a client to push updates to a specific inbox that is connected to a resource in the users POD. When the inbox has received a new update, another client will get notified if it has subscribed to this inbox. A typical use case for this would be social data that gets updated regularly, for example a news feed with posts and comments. In Solid, each post or comment would be stored in the publishers own POD and connected through Linked Data. For applications to know when data has been updated in either POD, it can subscribe for updates in the resources Inbox.

2.2.1 Existing Adaptations and Applications

Inrupt Inc.[8] is a startup company also founded by Tim Berners Lee among others, that aims to create and provide the necessary tools to support and kickstart the Solid platform for usage in companies and governments. They have created developer tools such as libraries that can be integrated by websites to more easily communicate with Solid PODS and work with the RDF data format. They have created a high performance Enterprise Solid Server (ESS), that is used for hosting PODS and is commercially available. They also have a publicly available version of this POD Server, that anyone can

[4] https://www.w3.org/ns/ldp
[5] https://solidproject.org/TR/wac
[6] https://podbrowser.inrupt.com/
[7] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
[8] https://inrupt.com/

use with Solid compatible applications. Additionally they also have a POD Browser service, that can be used to manage the contents of a POD and access to its data.

Although Solid is not yet at the stage where it is broadly adopted and used on the web, there are multiple examples where it has been integrated and used in different pilot programs. Inrupt has made several partnerships with companies and the government of Flanders, to integrate and test the Solid platform in their services. These partners have integrated the ESS, to store their users data in parts of their service.

The National Health Service in the UK are for example testing it to store their users medical data.[18] Another example is as mentioned the Flanders government, that are using it to provide their citizens with a service that they call 'Citizen Profiles', that retrieves data from the citizens Solid POD. The users can give other companies or the government access to this profile or revoke it if needed. This also makes it easier and faster for the user to give updated information to those it is shared with.[19]

Several applications have also been created by different community developers, such as social chat applications, forums and blogs.[9]

2.3 RDF Triplestore and SPARQL

RDF Triplestores are used to store information in the shape of nodes and edges in a graph. RDF is short for Resource Description Framework, and is used for describing semantic triples, consisting of a subject, predicate and object. The subject and object are the nodes in the graph, and is connected to each other through the predicate, which is the edge. Semantic triples are the basic shape of Linked Data, and allows for any data resource to be linked to another. Each set of triples can either belong to its own distinct Named Graph or to a shared graph that is stored in the database.

SPARQL is short for "SPARQL Protocol and RDF Query Language", and is the standard language for querying Linked Data. It is used to retrieve or update data that is stored in the RDF Triplestore, and support different types of queries for this purpose. The SELECT query is used to retrieve data, and contains the "SELECT" and "WHERE" keywords, where the first specifies the variables to retrieve, and the latter contains a triple pattern, that will be matched against the data in the database. The pattern is consisting of a subject, predicate and object, where each of them can be replaced by a wildcard that is unknown to the query. Data that matches this pattern is retrieved from the database. Complex queries can be formed by using function keywords such as "UNION", "JOIN, "AND and "OR". SPARQL also supports functionality for sorting, filtering and grouping of data. The SPARQL UPDATE statements can be used to create, update or delete data in the database through keywords such as "INSERT" and "DELETE".

[9] https://solidproject.org/apps

2.4 Related Work

The experiments that was executed throughout this work falls into the category and field of performance testing, and specifically on different databases. The purpose of this is to find out how a system performs and what is causing the potential bottlenecks. Knowing this information allows for optimization and selection of the appropriate database. The artifact that was created for copying semantic triples from Solid PODS to an RDF Triplestore, could additionally categorise this work in the direction of data migration and data portability (specifically of RDF data). However that is out of the scope of this thesis.

2.5 Tools

The tools that was used throughout the work in this thesis is listed below.

Docker is a tool that allows for applications to be launched in containerized environments (known as Docker Containers), that works without additional configuration and installments of additional software. Docker Images are created from the code by defining a Docker file that specifies how the application should be configured. These configurations could for example specify networking options, hardware resource restrictions, etc. The Docker Images can be published and made available to others through its own repository on Docker Hub.[10]

K6 Open Source is a load testing tool that is used to execute performance testing on systems, to ensure that they are able to function as intended when the system is exposed to increased traffic. The tests are written in Javascript, but has a limited compatibility with NodeJS libraries and modules. The test results can be exported in different file formats such as CSV, JSON, TXT, XML or just as output in the console.[11]

Node Solid Server is a Solid server made in NodeJS (as its name suggests), that stores data in PODS on the Filesystem it runs on. It also features as an identity provider that gives the user a unique WebID. Two of the publicly available POD Providers (inrupt.net and solidcommunity.net) are using this software for their POD Servers.[12]

Other Tools that was used throughout this work is the OntoText GraphDB which is an RDF Database for storing semantic triples. The API that was created for communication with the RDF database was made using NodeJS and the LexiTags bookmarking website was written in ReactJS.

[10] https://www.docker.com/why-docker/
[11] https://k6.io/docs/
[12] https://github.com/nodeSolidServer/node-solid-server

Chapter 3

Methodology

This chapter will present relevant methods that was used to investigate the research questions presented in chapter 1. The topics are design science research, experimental research, performance testing and quantitative data analysis. How these methods was used in this project is explored in chapter 4.

3.1 Design-Science Research

Design science research is an approach where the goal is to produce new knowledge by building and evaluating an artefact, that will contribute to solving a problem. The requirements of what is considered a valuable artifact is disputed as this type of research is still evolving. Hevner et al. (2004) argues that an artifact produced through the process of design science research can either be a construct, model, method or an instantiation.[20]

- **Constructs** are conceptual objects intended for describing a phenomena.
- **Models** are also conceptual objects used for describing phenomenon's, but uses constructs and connections between them for the representation.
- **Methods** are used to guide others in how to do something that has already been achieved, and where the outcome can be replicated by following the same procedure.
- **Instantiations** are functional representations of a construct or model that is built through a method.

3.1.1 Evaluation of Design Science Research

The outcome of design science research must be a valuable contribution to new knowledge. Hevner et al. (2004) has defined seven guidelines for producing such contributions.[20] These are:

1. The outcome of the research must be a working artifact such as a construct, model, method or instantiation.
2. The outcome of the research must be of relevance and importance in making a contribution to a given problem.

3. The outcome of the research must be evaluated in terms of effectiveness and efficiency, to highlight the artifacts value.
4. The outcome of the research must be novel in how it contributes to new knowledge through the work accomplished.
5. The outcome of the research must be the result of a rigorous construction method, that ensures replicability.
6. The outcome of the research should be the result of a search process, where available resources are identified and applied to reach or get closer to a desired achievement.
7. The process and outcome of the research must be clearly communicated to its audience.

Another researcher Ron Weber (2017) argues that design science research can be evaluated through both process and artifact.[21] This is to be certain that high-quality design-science research has been performed, and that the contributions are of value. The entire process of design science research can be evaluated through three aspects, consisting of problem description, contribution to new knowledge and how the solution was found.

EVALUATION OF THE PROBLEM DESCRIPTION

This aspect can be evaluated by three criteria, where the first is how well the problem description is defining the nature and boundaries of the problem. This can be specified by describing what the problem is, why it is a problem, when and where the problem is happening, who is experiencing and who is affected by it. The second criteria is how broad or narrow the problem is defined and the third criteria is evaluated by looking at if the problem has been framed in a cunning way.

EVALUATION OF THE CONTRIBUTION AS NEW KNOWLEDGE

This aspect can be evaluated by exploring the artifacts novelty in solving a problem that is either old or new. Novel artifacts have the most value in contrast to artifacts that implements known solutions to known problems, which do not really provide any new contribution. Artifacts that solves a previously solved problem in a better or more efficient way, can also be considered valuable contributions to knowledge.

EVALUATION OF HOW THE SOLUTION WAS FOUND

This last aspect can be evaluated by analysing how the solution to the problem was found. This is done by following the steps of development from the initial state where the problem space is defined, through the path of finding working techniques that together qualifies as the end state and solution to the problem. The problem space and what qualifies as a solution should be described early on, to be able to decide whether or not the research contribution qualifies as a "problem solved".

3.2 Experimental Research

Experimental research is a method that investigates the cause and effect between variables. It is performed by manipulating independent variables in order to see what effect

it might have on another dependant variable. The primary goal of this type of research is to test if whether or not a hypothesis is correct. Kerry Tanner (2017) introduces the core concepts of experimental research as hypothesis, variables, reliability and validity, control groups and experimental group.[22] These concepts are summarised below.

HYPOTHESIS AND VARIABLES

A hypothesis is a statement that represents a theory that has not yet been tested and where the answer is not known. The theory can be a prediction about the effect that' something' will have on 'something else'. A variable can be considered as the 'something' that has an impact, for example a value or a state. In experimental research there are mainly three different types of variables that will naturally affect the research in some way.

1. The first is the dependant variable which is the main thing that is being experimented upon in the research and is the variable that we want to affect.

2. The independent variable is the thing that is manipulated in order to 'cause' the effect on the dependant variable.

3. Extraneous variables are unknown things that are somehow also affecting the research. These variables are very often unforeseen and can therefore be difficult to control or avoid. Such variables can also be hard to identify, however documenting these variables where possible is important in order to preserve the validity of the research.

RELIABILITY AND VALIDITY

Results produced by experimental research must be reliable and valid. To ensure that results are reliable the same tests should be performed several times and produce somewhat identical results. The validity of results is determined by how likely it is that the independent variable is affecting the dependant variable, and not some other extraneous variable. Changes in the subjects, instruments or any other part of the experiment, that occurs between each experiment can affect the dependant variable and return results that invalidates the experiment. Another important aspect that is affecting the validity is whether or not the results are useful to other parties or settings.

EXPERIMENTAL AND CONTROL GROUPS

In order to know what effect a independent variable has on a dependant variable, the subjects involved in the tests are separated into two separate groups. One group is called the control group and represents a baseline where the dependant variable has not been exposed to the effects of the independent variable. The other group is the experimental group and contains the subjects that will be affected by the independent variables. If participants are allowed to select groups themselves, it can sometimes affect the dependant variable. To avoid such errors, the groups should instead be formed by random assignation or by deliberate matching which can be based on a shared variable between subjects. The approach that is best suited for each type of test will typically be determined by the content of the tests.

3.2.1 Various Types

Experimental research is commonly categorized into three approaches: true experimental research, quasi-experimental and pre-experimental. Each of these have their own designs together with strengths and weaknesses, based on which situation it is used in.

TRUE-EXPERIMENTAL

True-experimental research is considered the best type of experimental research in regards to internal validity of the results. This is because of the strict constraints that it put on the method, such as the requirement of using randomisation to form groups and also the need to observe both a control group and a experimental group. By applying these constraints to the method, it reduces the possibility of it affecting the experiment as an extraneous variable. However depending on the contents of the experiment, these strict constraints can potentially limit the usefulness of the experiment by taking place in a unrealistic environment that is not applicable in the real world.

QUASI-EXPERIMENTAL

Not all experiments in experimental research is dependant on randomisation or has the opportunity to use it in order to form groups. Quasi-experimental research excludes this constraint on the experiment which opens for other alternative approaches.

PRE-EXPERIMENTAL

In pre-experimental research there is no requirement for randomisation or control groups. Experiments are performed by observing the effect of exposing the dependant variable of a group to an independent variable. Since the control group is not observed, there is nothing to compare the results with other than before and after. For some experiments this is enough to give the researcher the answers searched for, but for others it will not be a good option to use.

3.3 Performance Testing

Performance testing is a process that is used to identify and remove bottlenecks in the performance of a system, which is typically applied during the development in order to make sure that it is able to remain fast, scalable and stable under varying workloads. There are many sources that can cause poor performance. For example, it might take a long time before an application is ready to be interacted with, or that it takes a long time from a user performs an action to the application has completed the action. Another source of a performance problem is poor scalability, which is an applications ability to perform effectively with an increased number of users. For example, an application that does not scale could be performing well with just a few users, but whenever the number of users reaches a certain level it would become unbearably slow or even crash. Other sources might lie in the applications code or at the machinery it is running on. Unoptimized code or limited hardware can cause bottlenecks in the performance that can be fixed by identifying its source and optimizing whats causing it.[23] Performance testing can be divided into different types, based on what aspect is to be tested:

- **Load testing** is used to test if an application is able to perform as expected, with the expected amount of users.
- **Scalability testing** is used to test if an application is able to perform as expected, when there is a an increase in the number of users that goes beyond what is expected.
- **Stress testing** is used to test the limits of how high workload an application is able to process, with the goal of finding the breaking point.
- **Endurance testing** is a subset of stress testing, which is used to test that an application is able to do its intended workload over a longer period of time without crashing.
- **Spike testing** is another subset of stress testing, which is used to test that an application is able to handle a sudden increase in workload.
- **Volume testing** is used to test that an application is able to perform effectively with a larger database.

3.3.1 Metrics

Problems in performance can occur in different parts of the application, the datastore or in the hardware. And as a result of the many potential sources, there are different metrics that can be used to investigate. The only metric that will be measured in this thesis is the response time, which is the time taken between a request is sent by the user until a response is given by the application.

3.3.2 Methodology

Performance testing can be performed through varying designs and methods, but should always be performed in a structured manner, to get the best outcome. [24] Typical steps that can be included in performance testing is listed below.

THE GOAL

The first step of performance testing is to specify what the purpose and aim of the test is. These goals should include criteria of what should be considered a positive and negative result in the applications performance, in order to evaluate it.

THE TEST

The next step is to design the test to so that they reflect realistic scenarios, and that they are able to measure what is required. This can be achieved by identifying different use cases within the application that is tested. When the design is clear, the tests can be created.

THE ENVIRONMENT

The environment where the tests are executed should be configured and documented. This includes identifying information about the application, the hardware it is running on, the networking settings, additional tools that will be used or any other variables that can affect the results somehow.

EXECUTION AND ANALYSIS

The tests are then executed, observed and documented. When all tests have finished, the results are analysed, and if necessary the tests can be optimised and executed in several iterations.

3.4 Quantitative Data Analysis

Quantitative analysis is used when collected data can be characterised as numerical values that can be measured or counted. Numerical data can be of different types which is defined by the degree of information it carries.[25]

- **Nominal Data** are values that can be grouped together or categorized.
- **Ordinal Data** are values that follows a specific order.
- **Interval Data** which is an extension of Ordinal data that must also have an equal distance between each data point.
- **Ratio Data** which is a further extension of Interval data, additionally uses zero as a starting point for its values.

3.4.1 Data Preparation and Methods

Quantitative data analysis can be performed in two steps where the first step is to prepare the data, and the second is to perform the analysis.

DATA PREPARATION

Before analysing the collected data, its raw format often needs to be prepared before the techniques of either approach can be applied. For this task there are several methods that can be used, depending on what requirements is set by the techniques.

The first step of this process typically involves performing data validation on the collected data. The purpose of this task is to reduce the risk of errors in the data, such as missing values, wrong format or values that is obviously corrupted. Records that contains such errors should be excluded from the data or replaced by getting a newer correct version of the data if possible. The consequence of not cleaning the data are negative effects on the analysis by giving false observations and results that in the worst case invalidates the experiment.

The next step of the data preparation process is to transform the data into the format required by the technique. Raw data is not always numerical to begin with, and it is therefore necessary to encode such data into quantitative data. Categorical data such as gender, yes/no or 'static' answer alternatives can easily be transformed into numbers beginning from 0 and upwards. A dataset containing numerical values that is within a very large range, can be grouped into smaller sub-ranges to make it easier to work with. These sub-ranges can further be categorized by meaningful tags that represent the meaning of the values within.

Quantitative data that has been modified through the preparation process can provide a valuable abstraction from the actual collected values of the original dataset. However it can also remove important details and aspects that is valuable to the research. Information and insights about the dataset is therefore necessary before encoding the data into quantitative data, in order to find the right balance. When encoding data into quantitative data, different values can sometimes have identical meaning depending on the data collection methods. Open questions in a questionnaire is one common scenario where this might occur, where the participant are free to express themselves however they want. In larger datasets, automatic tools that use word extraction and stemming can be used to quickly categorize the initial values meaning into numerical values. However this being an automatic process with little human interaction, it is highly prone to errors.

METHODS

Techniques used in quantitative data analysis can be divided into the categories "Descriptive Analysis" and "Inferential Analysis". Descriptive methods is used to describe the data and inferential methods can be used to draw conclusions about its meaning.[25] The results produced through the experiments in this thesis, will use descriptive methods to make observations on the data. Some of these methods are listed below.

- **Mean** is the average of the set of values and is calculated by the addition of all values divided by the number of values. This method is best suited for a range of numerical values that is defined as interval, where each value has an equal distance between themselves.
- **Median** is the value at the center of the set of values and is found by splitting the range of values in equal halves. If the total number of values is an odd number, the values is found at the end of the first half. If the total number of values is an even number, it is found by adding the last number of the first half with the first number of the second half, divided by two. This method is best suited where the range of values contains an uneven distance between each values.
- **Mode** is the most occurring value in the range of values. This method is best suited for nominal data and can for example be used to identify the most frequent group of values.
- **Range** is the distance between the first and last value in a set of values.
- **Interquartile Range** is an alternative to range that is used to inspect the range at the center of all values. This is found by first splitting the set of values in four subsets and then comparing the first value of the second subset and the last value of the third subset.
- **Graphs** can be used to better visualise the values in a dataset, which can be helpful to explore and describe the data. Different types of graphs can be used depending on the type of data available. A few examples are variations of Scatter-Plot, Bar-Graphs and Diagrams.

Chapter 4

Methods

As introduced by the problem space in chapter 1, the current architecture of Solid requires the applications to request data from each individual users POD, instead of retrieving it from a single centralised datastore within the applications own landscape. From the concept of separating data there is reason to believe that this architecture will experience scaling problems due to increased latency, when used in applications that relies on data from multiple users at the same time (typically social networks). This is an assumption that is debated within the Solid community forums.[10; 11] Other concerns in these debates seems to also be the lack of ability to perform complex queries on data from multiple users (which is caused from data separation and storing data on the filesystem). As the Solid Specification is still at a very early stage of development, it is expected that POD Providers will offer different datastores as their underlying storage technologies in the future, that can support user wide queries within the individual POD Servers. Increased latency in data retrieval due to separating data can also be solved by other architectural implementations that can be developed and optimised by anyone.

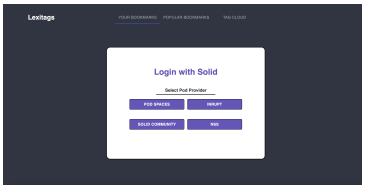
The goal of this thesis is first to identify what can be expected from the performance of the Solid architecture, and secondly to explore an alternative implementation of this technology with hopes of better performance in data aggregation and retrieval. The alternative implementation is integrated in a website that was created as an artifact through a design science research process, and can be divided into three parts consisting of: 1) The website itself with a direct Solid PODS integration, 2) The alternative Solid implementation, and 3) An additional script that was integrated in the website to measure the time it takes to put together data from multiple PODS. A second separate artifact was also created, and is a script that allows for measuring the data retrieval time from PODS and the Triplestore. The script is executed by the K6 performance testing application and its results will be used to evaluate the main artifact. As explored in chapter 3, Hevner et. al (2004) introduces 7 guidelines for what should be considered valuable contributions of design science research.[20] How this research is complying to these guidelines are presented in this chapter, where the first section will introduce the artifacts, what their purpose is for this research and how they were created through iterative stages. The next section explains how experimental research and performance testing was used to measure the retrieval and aggregation time in order to investigate the research questions.

4.1 The Artifacts

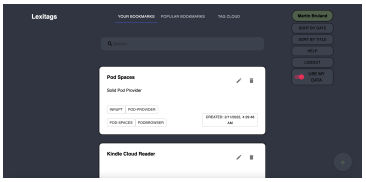
4.1.1 Artifact 1 : The LexiTags Website

LexiTags is a bookmarking website created by Prof. Csaba Veres, that uses semantic tagging to describe resources.[26] During the work of this thesis, this concept was used to create a new website with similar functionality. In this version of LexiTags, users can login with their Solid POD Provider and view their bookmarks, as well as create, update and remove existing ones. Bookmarks can be sorted by their title, date, tags or other keywords that appear within the title or description of a bookmark. Users can also display a list of the top 10 bookmarks between all users of the website. Public bookmarks can be explored through tags in a 'Word Cloud'. Selecting a tag from the word cloud will display the bookmarks from other users that has used this specific tag.

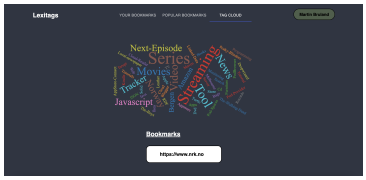
The website is written in ReactJS and allows for integrating and testing Solid as the backend, through two different approaches (Artifact 1.1 and Artifact 1.2). Each of these approaches and their individual purpose are further detailed in their own subsections below.



(a) LexiTags Website: The login screen for selecting POD Providers.



(b) LexiTags Website: The signed-in users personal bookmarks.



(c) LexiTags Website: The 'Tag Cloud', used for browsing bookmarks.

4.1.2 Artifact 1.1: LexiTags + Solid PODS

In the first version of the LexiTags website that was created, Solid was integrated through libraries created by Inrupt: the 'Solid-Client API' [1] and the 'Solid-Client-Authn-Browser API' [2]. When users are signing in, they are redirected to their POD Providers login screen where they must fill in their credentials. Once authentication is completed, they are returned to the LexiTags website, where they can start using the service. The data is transferred directly from their POD to the LexiTags client without any other in-between storage or server mechanisms.

For the website to know which PODS to look for trending items (such as the top 10 popular bookmarks), a list of users is kept in a POD owned by the application itself. When a new user starts using LexiTags, the website will first ask for access to create a new dataset within the users POD and then add the users WebID to the list of users. If a user decides to stop using the service, they have the option to opt out of using the application and to revoke the applications access to the users POD, which will make the data inaccessible to the application.

[1] https://docs.inrupt.com/developer-tools/api/javascript/solid-client/
[2] https://docs.inrupt.com/developer-tools/api/javascript/solid-client-authn-browser/

The purpose of integrating Solid as a backend directly in the LexiTags website (Artifact 1) was to create a real case scenario, that allows for realistically experience how long it takes to retrieve data directly from multiple PODS, without any middleware storage solutions. In order to find and display the popular bookmarks among users, the website needs to retrieve the dataset from every POD in the list of users and aggregate the results after everything is retrieved. This process exemplifies an actual use-case and the value of the first research question, as this would be an extremely common requirement of websites that have a social aspect and relies on data from multiple users.

This artifact was built through an iterative stage of development, where more features and improvements were added over time. Each of these iterative steps are listed below.

ITERATION 1

The first iteration of the development stage of this artifact consisted of making the user interface and adding the basic features such as displaying, searching and sorting bookmarks.

ITERATION 2

The second iteration integrated the libraries from Inrupt in order to login to the website with a POD Provider, as well as storing and retrieving bookmarks from the POD. In order to find and display popular bookmarks from all users of the website, a list of their WebID's were needed to be kept somewhere. For this purpose a dedicated POD was created to contain a dataset with these WebID's. A word cloud that displays the tags from all users bookmarks were integrated and used as a way to explore bookmarks among the users.

ITERATION 3

In order to let the users decide whether or not their data should be included to aggregate the popular bookmarks among all users, an opt-in/out feature was added. This would let the user add and remove themselves from the list of users. Additionally a drag and drop bookmarklet script was added in order to more easily create new bookmarks when visiting a website.

ITERATION 4

The last iteration consisted of cleaning the code and adding improvements to it.

4.1.3 Artifact 1.2: LexiTags + Solid PODS with Triplestore

In another version of LexiTags, Solid was integrated on the server-side through other libraries from Inrupt, such as the 'Solid-Client-Authn-Node API' [3] As mentioned in chapter 1, one of the reasons for making copies of the users data in an additional datastore, was to let the website more quickly retrieve and display the trending items among users. Instead of sending a data request to each individual users Solid POD and then aggregate the results once everything is retrieved, the website can instead ask the server to query the database for the most popular items. Data from each individual users POD

[3] https://docs.inrupt.com/developer-tools/api/javascript/solid-client-authn-node/

is stored in its own Named Graph within the Triplestore. As a result of the users data being stored in one place, it removes the need for the additional list of users stored in LexiTags own POD (as was required in Artifact 1.1). Whenever a user is updating their data (either through creating, editing or deleting data) through the LexiTags website, the server application will reiterate their POD and store the changes in the Triplestore. Arguments can be made that this way of storing a copy of the users data is against the nature of Solid. This is why the API includes features that allows a user to opt-in and out of this data collection process. The result of opting out is that the trending items will not include or be based on data from the given users Solid POD. Any existing data about the specific user would be removed from the database. Further details about the development of the API is presented later in this section.

The second reason for creating this artifact (The API and Triplestore) was to have a centralised datastore, where the retrieval time could be measured and compared against the results from retrieving data directly from Solid PODS. The purpose of integrating the endpoints in the LexiTags website (Artifact 1) was to experience the performance hands on the same way as with Artifact 1.1. It also allowed for observing it in a realistic environment, which resulted in valuable observations that was used in the evaluation of the artifact. They also affected the iterative stages in the development of the artifact, which are listed below:

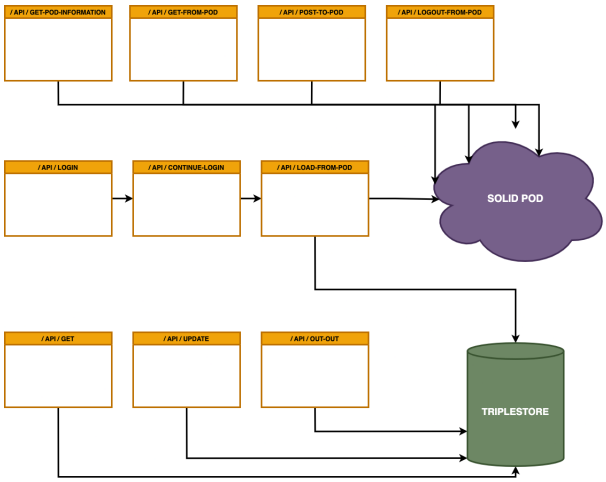


Figure 4.2: Visual representation of the API and the available Endpoints.

ITERATION 1

The goal of the first version of the artifact was to create the basic functionality, that allows a client to communicate with the Triplestore in order to store and retrieve data from it. Since the data would originate from the users Solid POD, an endpoint that redirects to their Pod Providers login page was made in order to gain access to the data. Once authentication is complete, the Pod Provider redirects the user back to a secondary endpoint that iterates through the authenticated POD and fetches each triple, which is then transformed into an INSERT query that is sent to the Triplestore. Any changes made on data in the Triplestore is also be made on the data in the users POD.

This current state of the API enables an application to load the users data into a Triplestore and retrieve it. Updates to the data are sent to both the Triplestore and to the users POD. However limitations to this model of data flow appeared quickly. 1) Sending data to two individual datastores can lead to data inconsistencies. 2) Making copies of the data could be considered against the Solid nature when the user is not able to have a say in it.

ITERATION 2

The first limitation described above regarding data inconsistencies can be solved in different ways, where some approaches are more simpler than others. This iteration chose to try out a simpler one, where updates are only sent directly to the users POD through an endpoint in the API, instead of updating the Triplestore as well. Whenever an update is made, the POD is reiterated and each triple is re-added to the Triplestore. While the downside to this approach is that it takes a longer time to perform, it ensures that whatever data is in the POD will also be in the Triplestore. This solution would only be optimal if the datastore insertion speed is rather high, which was not the case for this implementation of the Triplestore where more users resulted in slower insertions. Another endpoint was created that retrieves data directly from the users POD, which can be used when only needing data about the signed-in user. The Triplestore would thereby only be used for larger pieces of data that requires aggregation. To further extend the API, another endpoint was made that lets the user sign out from their POD.

ITERATION 3

The third iteration added a feature that allows a client to check if the Triplestore contains a Named Graph equal to the signed in users WebID and to remove its data if needed. This endpoint was added as a solution to the second limitation that was described in the first iteration. This enables an application to have features where the users can decide for themselves whether or not their data should be stored elsewhere.

ITERATION 4

During the fourth iteration the API was tested and implemented in the LexiTags website (Artifact 1), and thereby replacing the more direct POD communication that was integrated (Artifact 1.1). Having integrated the API allowed for testing and evaluation of the endpoints, in terms of fixing bugs and making improvements to the code.

ITERATION 5

Additional endpoints were added to the API in a separate iteration, which was used to automate the tasks needed for the preparation of the experiments. These endpoints allowed for the following tasks: 1) Creating any number of PODS. 2) Creating random datasets consisting of a any number of triples. 3) Inserting identical datasets into all PODS. 4) Copying the data from each POD in its own Named Graph in the Triplestore.

LIMITATIONS OF THE ARTIFACT

This artifact and its features should be considered a proof of concept, that represents one of many ways to use additional datastores in combination with Solid PODS. The limitations of this artifact are therefore many. One of these limitations includes the

challenge of knowing when data in each users POD has been updated elsewhere, and to retrieve a new copy when it does. An evaluation of this artifact is presented in chapter 6, and will include all the limitations that was found throughout this work, and potential approaches to resolve them.

4.1.4 Artifact 1.3: LexiTags + Script for Aggregation Time

In addition to the two implementations of Solid that was integrated in the LexiTags website, an additional script was created and integrated as well. The script facilitates for performing one of the experiments that will be introduced in the next section for this chapter, that measures the time it takes to put together data from multiple Solid PODS. This process is performed after all the data has been retrieved and does not include the retrieval time. More specifically, it is the duration of aggregating the retrieved data and transforming it into a format that can be used by the application. The reason for executing the script in the Lexitags website instead of the K6 application, is the lack of support for libraries such as Inrupts "Solid Client API" [4], which is used to read and modify the retrieved data.

4.1.5 Artifact 2 : Script for Retrieval Time

The other artifact that was created through this work are the scripts that is executed by the K6 application.[5] The scripts were used to measure the time it took to retrieve data directly from any number of Solid PODS and from the Named Graphs within the Triplestore. Each of the scripts consists of different groups that will be executed a given number of times. The groups are differentiated by the number of users to retrieve from and different datasets that differ in size. Each of the two scripts will produce a CSV file with the results from the experiment. The specific details about the groups in each script and their execution are further presented in the next section of this chapter that introduces the experiments.

The script that is measuring the retrieval time from the Solid PODS, is iterating through a given number of mock users and sending an individual request to each of these users POD where the dataset is located. When data has been retrieved from the current user, it continues on to the next. A code snippet from this script is displayed in figure B.1 (Appendix B).

The script that is measuring the retrieval time from the Triplestore, is sending a query to the API in Artifact 1.2 that specifies how many users (or Named Graphs) and which dataset to retrieve from. The endpoint in the API queries and retrieves this data from the Triplestore and returns it to the application when found. A code snippet from this script is displayed in figure B.2 (Appendix B). In order to see the benefits of aggregating data before the data is retrieved, a second query was sent to the Triplestore that only retrieved the top 10 items for each group. A code snippet of this script is displayed in figure B.3 (Appendix B).

[4] https://docs.inrupt.com/developer-tools/api/javascript/solid-client/
[5] https://k6.io/open-source/

4.2 The Experiments

According to a study performed by Google, users are expecting content to load within two seconds.[9] If the content is not displayed within this limit, the users are likely to grow impatient and leave the page, and in worst case they will not return to use it. As previously stated, concerns are made that the current architecture of Solid might not be able to scale when needed to retrieve data from a larger amount of users. If applications are going to want to implement Solid, they needs to meet the two second criteria. Otherwise they will risk loosing their customers and thereby revenue.

To investigate the performance of retrieving data directly from the PODS and from the alternative Solid implementation (Artifact 1.2), the K6 application was used in the experiments to benchmark the two approaches. The tool executed a script (Artifact 2), that can measure the time it takes to retrieve data from both datastores (Solid PODS and Triplestore). To investigate the effect of performing the data aggregation task on the client instead of the backend, a script (Artifact 1.3) was integrated in the LexiTags website (Artifact 1), that can measure how long it takes to put together the data and display it.

Measuring the time it takes to retrieve data from datastores is typically considered as performance testing. The aim of the experiments in this thesis was to see what happens to the retrieval time, when increasing the number of users up to a maximum of 10 000, and is therefore categorised as scalability testing since we don't know what to expect yet. The experiments was restricted to 10 000 users because of the time available to set everything up. The experiments was defined and structured by using terminology and techniques from both the field of performance testing and experimental research.

If an application wants to display the most popular items between all of its users, the traditional centralised database can aggregate and deliver this data quickly to the client. However in Solid the data is located in each individual users POD, which requires this task to be moved elsewhere such as the client. Because of this, the task must be performed after all the data is successfully retrieved from all of the PODS, and will therefore affect the overall loading time of such tasks within the application. Another experiment was therefore performed with the purpose of investigating how long this task can take, by measuring the data aggregation time (Artifact 1.3).

The experiments in this thesis is considered to be quasi-experimental, since a a true experiment strictly requires the control group and experimental group to be formed through randomisation. The groups of this experiment could have been formed through randomisation, but it would not have had any affect on the retrieval time, due to their nature. The purpose of the experiments is to see what happens to the retrieval and aggregation time, when the number of PODS and dataset size increases. The control and experimental groups in the experiments therefore consisted of PODS, which are all identical in that they are all individual datastores manufactured from mock data and containing identical datasets.

The validity of the experiments was ensured by 1) Documenting the environment in

which the experiments was performed in, 2) Documenting the variables that could affect the results produced by the experiments, and 3) Cleaning the environment before each experiment was executed, to avoid data to be cached or preloaded without intention. However the experiments also had threats to its validity, which will be introduced later.

The reliability of the experiments was ensured by performing each experiment 10 times and documenting the results. Caching of data was disabled where it was possible to do so, ie. the K6 application and the API in Artifact 1.2. The results produced by the K6 application, provided details about each of the steps in the individual requests, which made it possible to closer inspect any outlier values in order to see if there was a particular reason for it being further away from the rest of the iterations.

The replicability of the experiments was ensured by running the applications in a Docker [6] environment. This ensures that the experiments can be performed elsewhere with the same data, without having to configure or install anything other than Docker and each of the Images. Each Docker Container was also configured to only have access to a restricted amount of computer power (RAM, CPU). The applications was running locally on the same instance (Linux Ubuntu 20.04), hosted on a cloud computer from the Norwegian Research and Education Cloud.[7] Each of the specific docker containers was only running for the duration of the experiment it was used in. A complete list of the characteristics of the environment are displayed in Table 4.1 and Table 4.2.

Specifications	**Value**
Machine Host	Norwegian Research and Education Cloud
Operating System	Linux Ubuntu 20.04
Docker version	20.10.14
Disk Space	20 GB
Ram	16 GB
VCPU	4x 2GHz

Table 4.1: Specifications about the experiments environment.

Applications	**CPU**	**RAM**
K6 Open Source	1x 2Ghz	4 GB
Node Solid Server	1x 2Ghz	4 GB
Node Express Server	1x 2Ghz	4 GB
OntoText GraphDB	1x 2Ghz	4 GB
LexiTags Website	1x 2Ghz	4 GB

Table 4.2: Applications used in the experiments, with allocated hardware resources.

ENVIRONMENTAL THREATS TO VALIDITY

Since all experiments was running locally on the same instance, it can be a threat to the validity of the experiment. By not performing the experiments on the web, the

[6] https://www.docker.com/
[7] https://www.nrec.no/

results was not affected by extraneous variables that realistically would occur. However by not exposing the experiments to these variables, one could argue that the results would be cleaner, because of the difficulty to control the other extraneous variables. One example of this is the latency produced from unknown external services, which could give varying results depending on where the experiments are being executed from, and which paths the requests would take across the web. Another example is a realistic environment where the POD Providers would be hosted in different countries around the world and installed on hardware with varying performance. Another threat to validity could potentially be caused by the fact that the alternative implementation (Artifact 1.2), does not have any authentication or data-access mechanisms. By not having these security measures it can affect the results by making things faster then they would be in a realistic environment, where these measures would naturally be installed. Because all the PODS in these experiments were hosted on the same POD Server, the results does not show what would happen to the retrieval time if the PODS were hosted across several POD Servers (which would be more realistic). Based on these environmental threats to validity, it can be assumed that the results produced through these experiments will represent a best case scenario. If retrieving data directly from multiple PODS in this nicer environment is not performing, then it is unlikely that it will perform any better in a realistic environment. However it is worth mentioning that this performance is limited to the specific POD Server that was used in these experiments, which is the "Node Solid Server".

4.2.1 Experiment 1: Solid - Retrieval Time

The first experiment measured the time it took to retrieve different datasets from a varying number of PODS. This experiment covers two aspects: 1) what happens to the retrieval time when the number of PODS increases and 2) what happens to the retrieval time when the dataset size increases. This experiment is tied to the first research question that was introduced in chapter 1, specifically: "How does SOLID perform as a data platform for Lexitags ?".

PREPARATION

In order to measure the time it takes to retrieve data from any number of PODS, the following things were needed: 1) A POD Server, 2) Mock users, and 3) The datasets to retrieve. Creating thousands of fake users on a public POD Server would not be very popular, as it would be considered as an attack on their service. An instance of the "Node Solid Server" was installed in a Docker containerized environment, where 10 000 mock users was created automatically through additional functionality provided by the API in artifact 1.2.

Three datasets of varying size were first produced and inserted into each users POD, where each of these contained 1 item, 10 items, 100 items. A fourth dataset containing 1 000 items was created and inserted into the first 1000 users POD. The reason for not inserting the fourth dataset into all users POD, had to do with the time it took to do so because of its size.

The datasets that was inserted into each POD consisted of varying amounts of seman-

tic triples. For example: the dataset with one item had a total of 7 semantic triples. The first triple described the datasets type. The second triple connected the dataset with the item it contained. And each item had additionally 5 statements consisting of its type, date, title, description and url. The dataset with 10 items consisted of a total of 61 semantic triples. The dataset with 100 items consisted of a total of 601 semantic triples. And the dataset with 1 000 items consisted of a total of 7 001 semantic triples.

After the data was inserted into each POD, a Docker Image was created of the modified instance. The purpose and benefit of this was to keep a backup of the POD Server with data and to make it available on Docker Hub for any future necessities.

ENVIRONMENT AND EXECUTION

This experiment was using the docker containers "K6 Open Source" and the "Node Solid Server", and was performed by executing a script (Artifact 2) in the K6 application that retrieved data from the experimental groups described in Table 4.3.

Characteristic	Value
Dependent Variable	Retrieval ‖ăAggregation Time
Independent Variable 1	Number of Users
Independent Variable 2	Dataset Size
Group Size	Determined by I.V 1
Control Group 1	Users: 1
Control Group 2	Dataset: 1 Item
Experimental Group 1	Users: 10, 100, 1 000, 10 000
Experimental Group 2	Datasets: 10, 100, 1 000 Items
Model	4 x 5 Factorial

Table 4.3: Displays the characteristics of Experiment 1 and 3.

4.2.2 Experiment 2: Solid - Aggregation Time

The second experiment measured the time it takes to aggregate data in the client. The data was retrieved from each of the experimental groups. The purpose of this experiment was too see how performing the aggregation task in the client affects the total content loading time. When retrieving data from the Triplestore (Artifact 1.2), a query can specify exactly what to retrieve and thereby also the amount of data that is retrieved. The duration it takes to put together data from the Triplestore will therefore be consistent, which is not the case when retrieving data from Solid PODS where entire datasets of unknown sizes must be retrieved from each POD and then aggregated. The result of this is an overall content loading time that will be more unstable, depending on how much data is retrieved. The characteristics about the experiment is also displayed in Table 4.3.

PREPARATION

Additional input-fields and buttons was added to the LexiTags website in order to make it easier to specify which dataset to use and how many users to aggregate data from.

ENVIRONMENT AND EXECUTION

This experiment is using the docker containers "Node Solid Server", and the "LexiTags Website". The experiment was performed in the browser (Google Chrome) through an additional script (Artifact 1.3), that was integrated as part of the LexiTags website (Artifact 1). Because the script is executed as part of a website in the browser environment, the results can be highly prone to a multitude of extraneous variables that can be difficult to control and might therefore be problematic to accurately replicate. However it gives an indication of the time it takes to aggregate the data in the client instead of doing this elsewhere, which is a valuable insight in the overall performance. The extraneous variables can include the programming language, libraries, frameworks and APIs, coding approach, browser, client hardware, etc.

4.2.3 Experiment 3: Triplestore - Retrieval Time

The third experiment measured the time it takes to retrieve data from the Triplestore, where the requests are sent through the API (Artifact 1.2). This experiment covers three aspects: 1) what happens to the retrieval time when the number of named graphs increase. 2) what happens to the retrieval time when the amount of data to be retrieved from the triplestore increases. and 3) what happens to the retrieval time when data can be queried and aggregated before it is retrieved. This experiment is tied to the second research question that was introduced in chapter 1, specifically: "How does the performance compare to an alternative implementation where data from each Solid POD is stored in an RDF Triplestore?".

PREPARATION

In order to measure the retrieval time on the Triplestore, the data in each users POD was iterated, extracted and added into their own Named Graph in the Triplestore. The purpose of this was to store the same users with identical data in both datastores. The extracted data was stored in multiple files, that was added manually to the Triplestore. This approach was chosen in order to deal with the much higher insertion time, that was experienced when using INSERT statements through the API. The data created a total of 50 files with a total size of 2.61 GB. The number of files created was decided by the memory capabilities of the machine that extracted the data.

ENVIRONMENT AND EXECUTION

This experiment is using the docker containers "K6. Open Source", "Node Express Server" and "OntoText GraphDB". It was performed by executing a script (Artifact 2) in the K6 application, that 1) retrieved all data from the experimental groups described in Table 4.3 and 2) only retrieved the top 10 items among the same groups. The results of these experiments are presented in the next chapter.

Chapter 5

Results

Performing the experiments that measured the retrieval time from Solid PODS and the Triplestore, and the experiment that measured the time it took to aggregate data in the client, produced four separate datasets. A detailed overview of these datasets is presented in the first section of this chapter and can be skipped if necessary. The second section describes what data was extracted from each dataset and how it was prepared for further usage. The last section summarises the observations that can be made from the results.

5.1 The Datasets

5.1.1 Experiment 1 and 3: Retrieval Time

The output produced by the K6 application from executing the scripts that measured retrieval time (Artifact 2), is a CSV file that contains different data about the retrieval process from each individual POD in the experiment. The dataset produced from measuring the retrieval time directly from Solid PODS contains a number of 3 145 318 rows and 18 columns. The dataset from measuring the retrieval time of the Triplestore (Artifact 1.2), contains a number of 18 528 rows and 18 columns. The dataset from measuring the retrieval time of the Triplestore, when data had been aggregated to only retrieve the top 10 items, contains a number of 2065 rows and 18 columns. The most important columns in both of these datasets is the 'Timestamp', 'Group', 'Metric Name' and 'Metric Value'. Each of these are described below:

- **Timestamp** contains the date and time of when the request was sent and is displayed in a 16-digit Unix time format, that keeps track of the time by counting the number of seconds that has passed since January 1st, 1970.
- **Group** specifies the number of users to retrieve data from and which dataset to retrieve. As introduced in chapter 4, the experiment is performed by retrieving data from five sets of PODS or Named Graphs, which are based on the amount of mock users (1, 10, 100, 1 000, 10 000). There are also four different datasets of varying sizes based on the number of items (1, 10, 100, 1 000). The factorial combination of these gives a total of 20 different groups, where the retrieval time was measured on 19 of these.
- **Metric Name** specifies individual parts of each request, and can be used to investigate what might be causing delays in the duration of that particular request. These

individual parts are 'HTTP Reqs', ' Req Failed', 'Req Receiving', 'Req Waiting', 'Req Sending', Req TLS Handshake', 'Req Connecting', 'Req Blocked', 'Req Duration' and 'Group Duration'. Other metrics that is included but not used are 'VUS' and 'VUS Max', which are specifying the number of virtual users (value is set to 1). There are some differences in the values of the metrics between measuring the retrieval time directly from Solid PODS and from the Triplestore. For instance, querying the Triplestore through the API makes it impossible to measure the request duration of each Named Graph because all the data is retrieved in one response. This is also why the dataset from the Solid experiment is larger than the dataset from the Triplestore experiment. While the metric 'Req Duration' represents the duration of each individual request, the metric 'Group Duration' represents the duration it took to retrieve data from the whole group and is the 'metric of interest' to analyse further.

- **Metric Value** contains the duration it took to perform the given part of the request. The values of the group / request duration and each subpart of the request is measured as milliseconds.

- **Additional columns** that are also important to look at are the 'Error' 'Error Code' and 'Status. These columns tells if the request was successful or if it failed. If a request failed, the tests would need to be performed again after finding the cause. Other columns specify the type of HTTP Method, which HTTP Protocol, and the name of the URL where the request was sent. In the dataset with results from measuring the retrieval time directly from each Solid POD, the values were identical for each request. An HTTP GET method was sent to each POD using the HTTP/1.1 protocol. In the datasets with results from measuring the retrieval time from the Triplestore (Artifact 1.2), the values were also identical for each request. An HTTP POST method that contained a query was sent to the API Endpoint, using the the HTTP/1.1 protocol. The following columns doesn't provide any data in these tests and is therefore excluded: Check, Scenario, Service, Subproto, TLS Version, Extra Tags, Expected Response.

5.1.2 Experiment 2: Aggregation Time

The duration it took to aggregate data that was retrieved from the PODS in the client, was manually observed and documented into a table, that contains the time it took to perform the experiment 10 times (iterations) for each group. Table A.2 in appendix A, shows a descriptive summary of these results.

5.2 Data Preparation

The datasets that was produced by the K6 application, were filtered by the column 'Metric Name' to only display the metric 'Group Duration'. This returned a total of 190 rows that shows each of the 10 iterations that was measured for each of the 19 experimental groups. These rows were then sorted by the column 'Groups' and then by 'Timestamp', in order to display the rows by the order each request was sent. The name of each group with their metric values (ie: Group Duration) in all iterations, was then extracted into its own descriptive table that was used to find the range and

average values of each group. Additionally these values were formatted to only have two decimals, and the separator also needed to be changed from dot to comma. Table A.1, A.2, A.3 and A.4 (found in Appendix A), each contains descriptive summaries from the extracted values for each experiment.

5.3 Data Analysis

At this stage there are four different tables (Table A.1, A.2, A.3 and A.4) that show the retrieval and aggregation times from 10 iterations of each experimental group. These tables were further analysed using the same approach. Table A.1 shows the results from measuring the retrieval time directly from Solid PODS and Table A.2 shows the results from measuring the aggregation time after retrieving it from each POD. Table A.3 and Table A.4 shows the results of retrieving entire datasets and the top 10 items among all users in the Triplestore (Artifact 1.2).

To look for irregularities in the iterations of each group (such as spikes in the duration), the range (min, med, max) values were first documented for each group. The iteration values of each group was then displayed in their own graph, in order to visualise outlier variables that should be further investigated. This was achieved by looking at the different steps in the retrieval process, found in the original datasets produced by the K6 application. For the experiment measuring the aggregation time, unstable iterations should be expected due to the many potential extraneous variables previously mentioned.

The average value between the iterations were then calculated and documented for each group. The duration was measured in milliseconds, and was converted into seconds and minutes, in order to make it easier to interpret when dealing with higher retrieval times. The average values of each experiment was then extracted into their own 4 x 5 table, that was used to investigate factorial main effects and interactions. The columns in this table represents the different datasets that was retrieved and the rows represent the amount of users that was retrieved from. These tables (Table A.5, A.6, A.7, A.8 and A.9) are also found in Appendix A, and was used to generate the graphs displayed in this chapter.

5.3.1 Experiment 1: Solid - Retrieval Time

From observing Table A.1, it can be found that the first iteration in some of the groups has a much longer retrieval time than the rest of the group. This behaviour can be found most clearly in the four following groups: "1 Pod, Dataset 1 Item", "1 Pod, Dataset 1000 Items", "10 Pods, Dataset 1 Item" and "1 000 Pods, Dataset 100 Items". The other groups show more or less consistent results or results that are equally varying between all iterations in the group. This behaviour can be found in the table by comparing the min, med and max of each group, where the maximum values are further away from the median value than the minimum value.

Caching of data was disabled in the K6 application by default, and should therefore be ruled out as a reason for these spikes. The dataset produced by the application was

used to investigate to some degree if there was another apparent reason for these spikes. Since the document contains 3 145 318 rows, a number of random requests was first selected and looked at. All of these showed that the majority of the duration was spent on the "http-req-waiting" metric, which is the time it takes before receiving the first byte of data from the server. To continue the investigation, each row that displayed this metric was highlighted and all of the requests that was looked at exhibited this behaviour. It is uncertain if this waiting time was because of the server being busy with a previous request or if it was because of something else.

The factorial table for this experiment (Table A.5), displays the average retrieval time for all groups in measured in seconds. By looking at the first columns where the dataset with only one item was retrieved, it can be observed that the retrieval time is increasing somewhat linearly with the number of users. Retrieving the dataset from only one user took an average of 0,08 seconds, 10 users took 0,68 seconds and 100 users took 6,5 seconds. Increasing the number of users to 1 000 took an average of 65 seconds, followed by almost 11 minutes when increasing to 10 000 users.

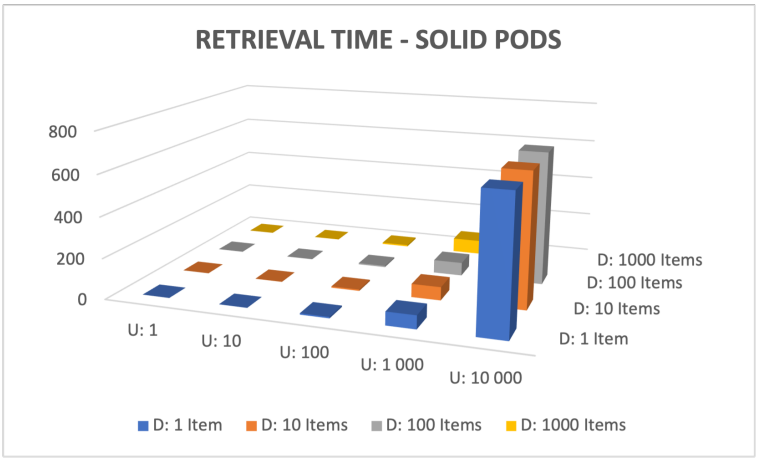


Figure 5.1: Duration of retrieving data directly from Solid PODS, measured in seconds.

Another observation that can be made from Table A.5 and the graph in Figure 5.1, is that the retrieval time does not seem at first to be affected by the increase in dataset size in any of the groups. For example, the duration of retrieving the dataset with 1 item from 10 000 PODS is almost identical to retrieving the dataset with 100 items from the same amount of PODS (just below 11 minutes), which is a strange behaviour. The file size of the dataset with 1 item is 4.3 kB and the size of the dataset with 100 items is 57kB. When retrieving these datasets from 10 000 users, the total size from each of these are 43MB and 570MB, which should normally affect the duration a lot more than it does. The most obvious explanation for this behavior is that both the Docker containers for the K6 application and the Node Solid Server is running locally on the same machine and is sharing the same network. This can be supported by the fact that it takes 1 second longer to retrieve 1 item from 10 000 PODS than 10 items. However retrieving the dataset with 1 000 items from 1 000 users, took 5 seconds longer than retrieving the other datasets from this amount of users. This 5 second difference is also occurring when retrieving the dataset with 100 items from 10 000 users. This likely indicates that the dataset size is also affecting the retrieval time just a little bit, even in the local envi-

ronment.

There is barely a minimum interaction between the number of users and dataset size in this experiment, that can only be observed in two of the groups (U:1 000, D: 1 000 Items and U:10 000, D: 1 00 Items). The interaction between these two independent variables would very likely increase in a realistic environment. The minimal effect that was observed is a limitation in the execution of this experiment. The results from this experiment is therefore representing a best case scenario in an optimal networking environment. Arguments can be made that these results are cleaner, since it is not exposed to extraneous variables that would likely produce varying results and be difficult to replicate. The results from both datastores can also be considered more "comparable", since they have a similar environment in that they both are running locally on Docker containers with equal access to hardware resources. And especially when it comes to the effect of their different architecture (the effect of data separation).

5.3.2 Experiment 2: Solid - Aggregation Time

The average duration of each group in Table A.6, naturally increases with the amount of data that is retrieved. Retrieving a dataset with 1 000 items from 1 000 users is a total of 1 million items (7 001 000 triples) that needs to be aggregated before it can be used by the website. Finding the top 10 items among these 1 million retrieved items, would take an average of 29,50 seconds and would be added to the total content loading time. The total time it would take for an application to both retrieve and aggregate data, is additionally displayed in Table A.7. The values are the summation of the results from measuring the retrieval and aggregation time in Table A.5 and A.6. From these tables it can be observed that the greatest impact on the total loading time occurs when the client needs to put together data from 100 000 items or more.

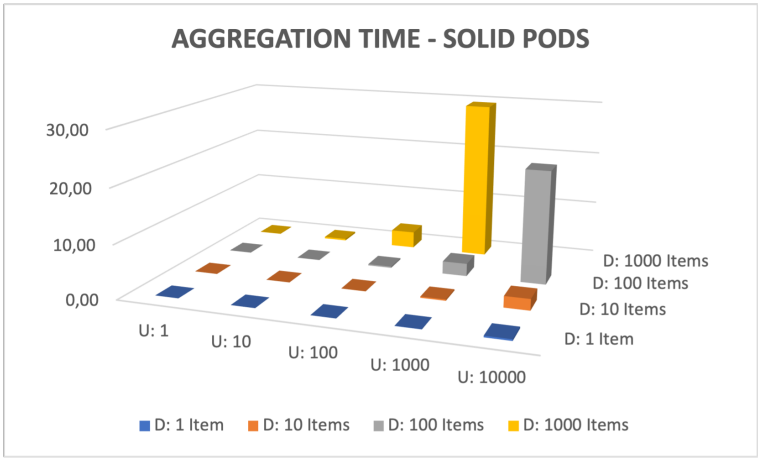


Figure 5.2: Duration of aggregating data in the client, measured in seconds.

5.3.3 Experiment 3: Triplestore - Retrieval Time

From observing the Table A.3 a similar behaviour to the spikes in retrieval time from Solid PODS can be found when retrieving data from the Triplestore. The first iterations of some groups has a much longer retrieval time than the rest of the group. One group

that stands out is the "Users: 1, Dataset: 1 Item", where the first iteration has an outlier maximum value of 21 seconds in contrast to the median value which is 30 milliseconds. This first group has the largest spread found in the results and is also the first request that was sent from the K6 application. This might indicate that the API or Triplestore in Artifact 1.2, was in some form of hibernation when the experiment was started. A similar behaviour is also especially present in the other groups where the dataset with one item is retrieved. However the range mainly differs in milliseconds and at most a couple of seconds when retrieving data from 10 000 users. The other groups either show somewhat even or equally varying durations among all iterations in the group.

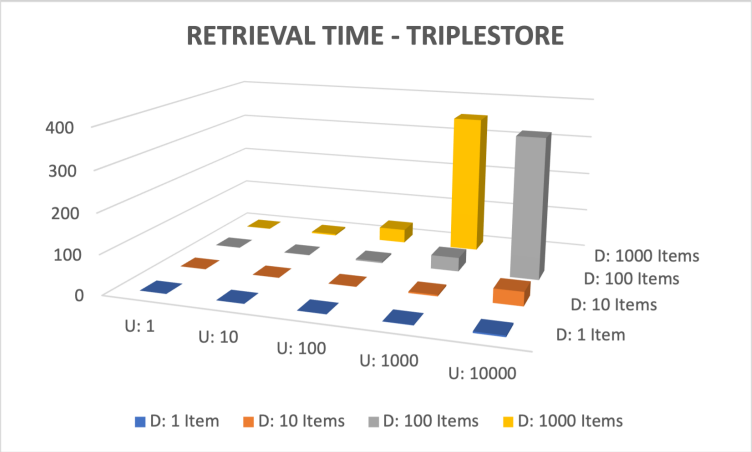


Figure 5.3: Duration of retrieving data from the Triplestore, measured in seconds.

As with the experiment that measures retrieval time from the PODS, the caching was disabled by default in the K6 application and also in the API. The dataset produced by the K6 application was used to investigate this behaviour. As with the retrieval from the PODS, the majority of the retrieval time was spent on waiting for the first byte of data. For example in the iteration that took 21 seconds (21047 ms), spent 21045 ms waiting for the first data.

If the iteration from the first group that took 21 seconds is excluded from the rest of the group (1 POD, Dataset 1 Item), then retrieving 1 item from 1 user took an average of 30 milliseconds and retrieving the same amount from 10 000 users (which is 10 000 items in total) took an average of 3,5 seconds. Increasing the retrieved number of items to 100 from 10 000 users (which is 1 000 000 items in total), took an average of 5,89 minutes.

By observing the average values of each group in Table A.8 and the graph in Figure 5.2, it does not seem like the retrieval time is affected by the number of users, but rather the number of items that is retrieved from the Triplestore in total. This can also be found by looking at the median values in Table A.3. For example when retrieving 1 000 items from 1 user, the median value was 0,36 seconds and 0,38 seconds when retrieving 1 item from 1 000 users. This behaviour is also found when comparing the duration of retrieving each of the four datasets from only one user, with the duration of retrieving the dataset with only one item from the following amount of users 1, 10, 100, 1 000. The total amount of retrieved data is identical and the retrieval time is around the same

range when compared.

When the the query was modified to only retrieve the top 10 items from each of the experiment groups, it drastically reduced the retrieval time. This is one of the big advantages of storing all the data in one location, compared to separating data into individual PODS. As can be observed in Table A.9, finding the top 10 items from the dataset with 1000 items among 1 000 users only took around two seconds.

As with the previous results from the Solid PODS retrieval time, since the Docker containers of the K6 applications and the API + Triplestore (Artifact 1.2) is running locally, these results are likely representing a best case scenario in an optimal networking environment. And it is therefore uncertain how the retrieval time would be affected once exposed to a realistic environment.

Chapter 6

Discussion and Evaluation

This chapter will first recap the research questions and discuss the main observations that was made from the results of each experiment in chapter 5. The points from this discussion will then be used in the evaluation of the solution that allows for storing copies of data from users POD in an RDF Triplestore. The artifact is evaluated by its performance compared to retrieving and aggregating data directly from the PODS, and by its architecture. It is also evaluated by the limitations that was found throughout this work, which will be listed with suggestions to resolve them. Potential uses cases of the artifact is also presented.

6.1 Research Questions: Summary and Discussion of Results

6.1.1 Experiment 1: Solid - Retrieval Time

The first research question: "How does SOLID perform as a data platform for Lexitags ?", was investigated through the first and second experiment. The purpose of this research question was to find out if the assumed scaling problem, actually occurs when needed to retrieve data from multiple PODS. The experiment measured the retrieval time of 19 groups, formed by the number of PODS to retrieve from and the size of the datasets.

The observations of performing this experiment was presented in chapter 5, and found that increasing the number of PODS was also increasing the retrieval time as expected. For example: retrieving the dataset with 1 item from only one POD took 0,08 seconds, and 1,09 minutes when retrieved from 1 000 PODS. When the number of PODS were increased to 10 000, it took 10,88 minutes to retrieve the data. The results also showed that the retrieval time was barely affected by increasing the dataset size (if at all), due to the local environment that the experiment was performed in. Arguments were made that this would likely change in a realistic environment, where the retrieval time would be exposed to the latency of transmitting the data across the web. Another limitation of this experiment was that the data was only retrieved from the PODS located on one POD Server. It is also likely that different POD Providers would perform differently depending on how it is configured and set up on the hardware it is running on. Because of these limitations the results is therefore representing a best case scenario in an an optimal networking environment, because of these missing aspects which are not covered by the experiment.

Retrieving data from only the logged-in users POD is performing very well as expected. As experienced through the LexiTags website (Artifact 1.1), reading and updating bookmarks in the users own POD is working smoothly without any visible delays. However retrieving data from all users POD, would be a common requirement for social applications that have features for displaying user generated content to all of its users. The problem occurs when an application needs to know what to display to the user. As have been previously mentioned, when separating data into several datastores, you loose the ability to perform complex user wide queries directly on the data. This requires data to be aggregated after all of it is retrieved, in order to be able to find similar or trending items among users, or show the data in its correct order. In the LexiTags website this caused a noticeable delay when displaying the popular bookmarks to the user, even with just a few users. This data consisted of bookmarks that only contained the URL of a website and the number of times it had been bookmarked. The list of popular bookmarks was created from data in each users POD, that was retrieved once the user logged into the website. If the website had 10 000 users it would take more than 10 minutes to display this content to the users, which would be a very poor user experience.

The long duration it takes to retrieve data from the increasing number of PODS, is a result of requesting the data from each POD individually. The current implementations of the existing POD Servers only offers POD wide queries and not Server wide. It is worth pointing out that this can change in the future, and especially if different underlying datastores is used instead of storing data in files on the filesystem. Functionality that allows a client to query data from several PODS hosted by the same POD Provider, would very likely increase the retrieval time considerably at each provider and also reduce the amount of data needed to be retrieved.

6.1.2 Experiment 2: Solid - Aggregation Time

In addition to measuring the retrieval time of Solid PODS, it was also necessary to see how performing the aggregation task in the client would affect the overall performance. The experiment was performed by putting together the top 10 items among a varying amount of data, that had been retrieved directly from several PODS. The results showed that finding the top 10 items from the datasets with 1 000 items that was retrieved from 1 000 users (which is a total of one million items or 7 001 000 triples), took about 30 seconds to achieve and would add to the total content loading time. Similarly putting together data from the dataset with 100 items that was retrieved from 10 000 users, which is the same amount of items, took 20 seconds to complete. This variance of 10 seconds for the same amount of work, can be due to a multitude of different extraneous variables. As mentioned in the description of the experiment in chapter 4, these variables includes the software, hardware and method used for the task. For example: it could be affected by the browser or machine doing other demanding tasks in the background. Additionally, the available resources on a users client that performs this task would also have an impact on how fast it is performed.

Applications that need to aggregate tons of data from all of its users, should avoid do-

ing this task in the client, as the retrieved data are stored and processed in the memory of the users computer. This task could potentially take up all available memory of the machine and crash it. Use cases that require aggregated data from all PODS, should therefore use other architectural solutions to integrate Solid, rather than letting the client do all the work at the time it is requested by the user. An obvious solution, would be to integrate Solid in a server application, that performs these larger retrieval tasks. For some applications this would also allow for caching the data a couple of times a day or whenever an action was triggered. This would additionally improve the content loading time as the data could already be retrieved and aggregated by the server, and ready to be displayed on the website.

The type of architectural solution that can be used in an application, will depend entirely on the type of data to be retrieved. For example, static data that doesn't change, is rarely interacted with or does not need to be updated constantly, could easily be cached and displayed. However for more dynamic data that is interacted with regularly, would require a different approach. The approach that was created throughout this work and presented in chapter 4 (Artifact 1.2), could store such dynamic data in the Triplestore.

6.1.3 Experiment 3: Triplestore - Retrieval Time

The second research question: "How does the performance compare to an alternative implementation where data from each Solid POD is stored in an RDF Triplestore?", was investigated through the third experiment that measured the retrieval time from the Triplestore. Through this research question, Solid was integrated in a server application (Artifact 1.2) instead of the client, with the aim of creating a solution that would improve the retrieval and aggregation of trending items among users. The purpose of this research question and the artifact was to find out how long it would take to retrieve the same amount of data from a triplestore, compared to retrieving it from the PODS. This comparison between the two datastores allowed for seeing how separating data would affect the retrieval process and the latency. The experiment measured the retrieval time of 19 groups, based on the number of Named Graphs to retrieve from and the amount of data to retrieve. An evaluation of the artifact is presented in the next section and will consist of its performance and architecture in regards to the aim and nature of decentralisation and Solid.

The observations of performing this experiment found that the retrieval time was not affected by increasing the number of Named Graphs to retrieve from in the Triplestore. For example retrieving the dataset with 1 item from only one Named Graph took 0,03 seconds and 0,52 seconds when retrieved from 1 000 Named Graphs. Increasing the number of Named Graphs to 10 000 took 3,70 seconds. Additionally, retrieving the dataset with 100 items from 10 000 Named Graphs, which is a total of one million items (7 001 000 triples), took 5,89 minutes. From these results it was found that the retrieval time was primarily affected by the amount of retrieved data. By modifying the query to only retrieve the top 10 items among this last group, reduced the retrieval time to only two seconds.

6.2 Evaluation of The Artifact

Chapter 4 introduced the artifacts that was produced throughout this work. The first artifact was divided into three sub parts consisting of: Artifact 1.1) the LexiTags website with a direct client side integration of Solid. Artifact 1.2) the LexiTags website with a server side integration of Solid that also uses a Triplestore. Artifact 1.3) the LexiTags website with an additional script that allowed for measuring the time it takes to put together data retrieved from several PODS. Artifact 2 consisted of the scripts that was executed by the K6 application to measure the retrieval time from Solid PODS and the Triplestore.ă

This section will mainly evaluate Artifact 1.2, which was produced through the second research question, as a means to investigate a potential solution to the scaling problem, that occurs when needed to retrieve data from a large number of PODS in order to display the required data. The other artifacts was created to provide comparative data that is used in the evaluation of Artifact 1.2. The artifact was evaluated through its performance, compared to retrieving data directly from each POD, and through its architecture in regards to the nature of Solid and decentralisation in general.

6.2.1 Benefits and Advantages

The performance of retrieving data from the Triplestore through the API was summarised in the previous section. It was found that the number of users were not affecting the retrieval time, which gives it a large advantage compared to retrieving data from Solid PODS, where the opposite behaviour was happening. It was found that the retrieval time was mainly affected by the amount of data to be retrieved. For example: the dataset with 100 items from 10 000 Named Graphs (which was a total of seven million triples) took about 6 minutes to retrieve, which was 5 minutes faster than retrieving the same amount of data from the same amount of PODS.

Even if the triplestore is clearly performing better, the duration of 6 minutes to retrieve data is still too long for a user to wait for data. This is where storing all the users data in one location has a major advantage, in that it can be queried and retrieved from the Triplestore in smaller chunks of data, which is in contrast to the PODS where data must be aggregated after it is retrieved. Because of this, it would be an unrealistic requirement from a website to retrieve seven million triples from the Triplestore in one go. If a website would like to display this much data within two seconds, they would typically integrate lazy loading techniques where data are retrieved and displayed to the user gradually as they continue to scroll or visits the next page. This creates a visually better user experience and reduces the amount of work on each part of the overall system.

6.2.2 Weaknesses and Limitations

LIMITATION 1 - Initial delay

The artifact performed better when used to retrieve data from several users, compared to the PODS. However, there are scenarios of using the artifact that could produce de-

lays caused by inserting and updating data in both the POD and Triplestore. The first type of scenario would be whenever a user starts using the service for the first time, where data from the users POD would need to be extracted and stored in the Triplestore before it can be used. The severity of this would depend on the type of data that should be copied and how it is used by the application. In some situations this would create an initial delay, where the application would be stuck on a loading screen for the duration it takes to do this task, which could take at least a couple of minutes. In other situations this task could be performed by the application in the background, where the user could start using the application at once. The duration of this process could vary between each user, as it would be determined by the amount of data to be retrieved from the POD.

At the more fundamental level when inserting the data in the preparation stage of the third experiment, it was experienced that the duration of inserting data from each users POD was increasing with the amount of data that was already stored in the triplestore. This behaviour is likely restricted to the specific Triplestore that was used and its default configurations, which can be optimised. This is also why the experiments does not include tasks for inserting or updating data, as this would become very specific to this implementation only.

When creating the artifact, two different approaches to the data flow was tried out. The first approach inserted data directly into both POD and Triplestore individually, when a user created new data or updated the existing ones. The second approach only inserted data into the POD, which was then retrieved from and inserted into the Triplestore. The first approach was obviously faster in that data became faster available, but it also introduced the risk of data becoming unsynchronized, if an update to the data in the POD was made elsewhere or if one of them failed. Both of these approaches were only intended as a minimal basis that allowed for exploring the artifact as a solution to the scaling problem. And because of the risk of data inconsistencies, the second approach was used instead to avoid this issue instead of the gained performance. However potential data inconsistencies of the first approach could be dealt with through a validation process where data in both datastores are compared. This would allow the application to insert data in both datastores and validate it in the background. The amount of occurrences where data is inconsistent would likely be most affected by frequency of the POD being updated elsewhere, which introduces the next limitation to this artifact which is knowing when the POD has been updated.

LIMITATION 2 - Maintaining updated version

Updating data directly in the POD or through other applications is one of the benefits of data decentralization and the of meaningful data in Solid. Since the artifact is not notified whenever there is a new version of the data in each users POD that can be retrieved and stored in the triplestore, it creates a limitation by the threat of outdated data. One solution to this problem would be for the server to have an open connection to each dataset in every users POD, that listens for changes and retrieves the data whenever there is a new version. The underlying technology that would make this possible, would depend on what is offered by the specific POD Servers, as there are different protocols that could be used for this. A common protocol that is typically used is the Web Socket

Protocol [1]. A challenge to this solution is that it is difficult to scale when you need too many concurrent connections, as could quickly become the case for this type of artifact. The hardware that the server (or artifact) is running on, has an upper limit of how many open connections it can use at once. This limit can can be extended either by upgrading or adding more machines, which quickly can become very expensive. Multiple cloud services offer server-less web-socket servers, that can reduce the complexity and can potentially also be cheaper. One example is the Amazon Web Services, which has a limit of 500 concurrent connections per second according to their documentation [27], which makes it possible to listen for changes in 1,8 million PODS per hour. The number of users of the application would determine whether or not web sockets should be used to keep the triplestore updated.

LIMITATION 3 - Deleting data on revoked access

One of the core concepts of the Solid Protocol is to allow users to revoke access to their data, that has been previously granted to applications. This action is typically triggered by the user from within their POD, by using a 'POD Browser' application. Since the artifact is storing the users data inside an additional datastore, this data should preferably be deleted whenever a user is revoking its access. This is partially solved in the artifact by providing an endpoint that can be called by an application, which removes all data tied to the user. However it can only be done through the application, and will not remove the data if the user revokes its access elsewhere. This could potentially be solved completely through web sockets, where the artifact could listen for changes in the Access Control List resource that contains the access rules for the particular dataset.

LIMITATION 4 - Lack of authentication and access controls

The next limitation of this artifact that was found is the obvious lack of authentication and access control in the API and Triplestore. The users need to sign into their POD in order to retrieve data from it. But once data has been inserted into the Triplestore, any user could potentially query and retrieve it through the API. An approach to solve this could be to restrict the endpoints to only retrieve data from the Triplestore, if the user is currently signed in with their POD Provider. An additional restriction is to only let the client have access to the Triplestore through HTTP GET methods, where the queries would only be generated by the API. These restrictions was not added in order to avoid complexities, when executing the experiment that measured the retrieval time on the Triplestore. Other measures would include the integration of more sophisticated technologies such as using the POD Identity Provider globally in the API. Another measure could be to adopt the existing rules from the Access Control Lists in the PODS, for each resource in the Triplestore.

LIMITATION 5 - Risk of breaking the nature of Solid

The artifact was produced as an approach to solve the latency issues, when needed to retrieve data from a larger amount of PODS, through storing a copy of the data in an additional centralised RDF Triplestore. This approach of storing copies of the users data introduces the perhaps most interesting limitation of the artifact, which has to do with the possibility of it breaking the concept of separating data from the applications

[1] https://websockets.spec.whatwg.org//

in Solid.

The Solid Protocol allows applications to use data directly from the users POD. It is not clearly stated if it is intended to be used entirely without additional datastores, even if its possible to do so. There would naturally be different opinions about whether or not data should be allowed to be copied. However once a user grants access to the data, there is no absolute guarantee that the applications wont copy it. In many scenarios it would be required to do this in order to provide a good user experience, to make recommendations to the users, or to generate new data about the user that the user also would benefit from, once shared with the users POD. Data decentralization itself is also not only about separating data into user controlled datastores, in order to protect users privacy. Its also about the benefits that can be gained from having the one source of truth, where data has meaning and can be used by several applications.

Another benefit is gained when users are allowed to decide where data is stored, as it can potentially become more secure by picking services with "state of the art" security measures. Additionally if data is stored privately, less data would potentially be gained from a breach, which can reduce the incentive for breachers. Copies of the data in several datastores can again of course increase the risk of it being stolen and misused, and thereby violates this benefit. But then again, if an application needs to keep a centralised copy of the data, in order for it to give the users the best possible user experience, then it is not unlikely that this will also be accepted among the users in contrast to a service that is not working smoothly. Through data privacy acts such as the GDPR, the user additionally has the legal rights for it to be removed or possibly ported into the POD.

Separating data from the applications, can return the ownership and control of personal data to the user. The question about ownership of data has created a debate of whether or not data can in fact be owned in the same way as an object or asset, and if so what type of data qualifies as personal data. There are many aspects to this debate where the arguments have roots in economy, legal, ethics, personality theories and technological practicalities. In an article[28] about this data ownership debate, Jurcys, et al (2021) argues that personal data does in fact meet all the requirements of an asset in property law through it being clearly defined, having economic value, and it being tradeable. They argue that data can be categorized as public, private and common property, depending on its sensitivity in the content, and the different actors that are involved. This would make an impact on what personal data can be owned and controlled by the user, in relation to governments, banks and other companies. They further argue that data can be divided into four layers consisting of: 1) Input Data, 2) Metadata, 3) Observable data, and 4) Derived data. While Input and Meta data is typically provided by the users, Observable and Derived data is generated by the applications from the data that was provided. The data that is generated by the companies should be shared and stored in the users own datastore, as this is also one of the legal requirements of privacy laws such as the GDPR, that allows users in the EU to request a copy of their data. They also argue that the copy of the data that is stored in the users datastore, will always be the updated version. And if the user revokes the access to this data, the companies and application will potentially very quickly have an outdated version. Because of this, they

argue that the most recent version of the data that is stored in the users datastore, should be considered as the asset that is owned by the user.

These arguments and ideas of data ownership can also be used to support why the artifact is not breaking the concepts of decentralizing data, which could also apply when using Solid in applications. The data in the users POD would be the data that is owned by the user, that represent an absolute origin of truth. When an application are requesting access to certain data that is stored in the users POD, it received a copy of the original data where its credibility can only be guaranteed for the duration it has access to it from the POD. Meaning that the copy of the data that is used by the application or in this case of the artifact where it is stored in the Triplestore, can quickly become outdated and irrelevant once access has been revoked. However this would naturally depend on the type of data that is copied as not all personal data is constantly changing or becoming outdated. The debate of data ownership is not yet settled and will affect what type of personal data can be claimed by the users and how it is managed on the decentralized web.

6.2.3 Example of Use Cases

The artifact could either be used to store a copy of all the data in a users POD or parts of it, depending on what is required by the application.

EXAMPLE 1

The artifact could be used to store a copy of the data from each users POD, that is only necessary to offer certain features within the application. One example could be trending items among all users in a social network application. In a Solid application whenever a user interacts with a post, the interaction would be stored in the POD of the user who made the interaction. For the applications to know which posts are the most popular, it would need to retrieve these interactions from each POD and then aggregate it. It would be required to perform this task each time a user created a new post or made a new interaction. To improve this, the application could retrieve data that is used alone directly from the users POD, and then use the artifact for faster aggregation and retrieval of the trending items.

EXAMPLE 2

The artifact could also be used to store a copy of the data from each users POD, that is necessary for the entire application to work. One example could be movie applications that offer recommendations based on viewer history, interactions, and details about the user. Keeping a copy of this data in a specific format in a centralized datastore that can quickly be retrieved from, would likely be required in order for it to be used by recommender-algorithms in other internal systems, that is part of the application.

Chapter 7

Conclusions and Future Work

The aim of this thesis has been to investigate the performance of using Solid in applications, that mainly require aggregated data from all of its users. An assumption was made in the introduction that when an application is needed to retrieve data from a larger amount of users that is stored in different locations, it would likely result in a high latency. This assumption is also debated within the "solid community forums", where concerns were also related to the lack of ability to query data between all users at once. This would require the data to be aggregated after all of it has been retrieved, and is therefore adding to the already increased latency. The actual affect of this was investigated through the first research question, specifically: "How does SOLID perform as a data platform for Lexitags ?"

Chapter 4 presented the artifacts and experiments that was used to look into this question, by measuring the retrieval and aggregation time of Solid PODS. The results of these experiments and observations that could be made from them, were then presented in chapter 5 and discussed further in chapter 6. It was found that the size of the retrieved dataset and the amount of data it contained, barely affected the retrieval time, and by at most 5 seconds when retrieving larger datasets from more than 1 000 users. The main variable that affected the retrieval time the most, were the amount of users that was retrieved from. Retrieving the smallest dataset consisting of one item from 10 000 users took at most 11 minutes, compared to milliseconds when only retrieving data from one user.

The time it takes to aggregate this retrieved data would additionally add to the total content loading time. At most it increases the loading time with 30 seconds. Arguments against retrieving and aggregating tons of data in the users client was made in chapter 6, due to the data being stored in the memory of the users machine. Applications that need to aggregate data from all users, should move this task to a dedicated server application instead or find alternative techniques to do this.

The results of retrieving and aggregating data from multiple Solid PODS, supported the assumptions and concerns that were made in advance. Using Solid in single user applications without needing to retrieve data from thousands of users performs at a satisfactory level. The severity of the benchmarks would therefore be determined by the requirements of the features within each application, and is mostly relevant to scenar-

ios where data to be displayed is derived from aggregating data among a larger amount of users.

This was further investigated through the second research question, specifically: "How does the performance compare to an alternative implementation where data from each Solid POD is stored in an RDF Triplestore?" Through this question an API that stored a copy of the data from a users POD in a centralised RDF Triplestore was created. The performance of retrieving and aggregating data directly from the Triplestore was measured and compared against the results from the PODS. As expected it was found that aggregating data and retrieving it from the Triplestore performed better than the PODS. The retrieval time was affected by the amount of data that was retrieved and not by the number of users / Named Graphs within the Triplestore. Retrieving one item from 10 000 users took at most 4 seconds, compared to the 11 minutes it took to retrieve it from the PODS. However increasing the number of items to 100 (7 001 000 triples) took 6 minutes when retrieved from the same amount of users. Arguments were made in chapter 6 that because data can be aggregated in the Triplestore before it is retrieved, it would rarely be necessary to retrieve this large amount of data in one go. For example aggregating the top 10 items from 10 000 users where the data was connected to the dataset consisting of 100 items, took around two seconds to retrieve from the triplestore, compared to the 11-12 minutes to do so from the PODS.

During the execution of the experiments it was found that the produced results is likely representing a best case scenario, where all the PODS were hosted on the same POD Server and where all the data was retrieved in a local environment on the same machine. Because of this the would not be exposed to the latency of transmitting the data across the web and therefore lacking the aspect. However the results from measuring both datastores are clearly showing the consequence of separating data when it needs to be aggregated.

Chapter 6 presented an evaluation of the API and Triplestore as a potential solution to increased performance, when needed to aggregate data from among all users of an application. Several limitations where listed together with suggestions for solving them. These limitations included the challenges of maintaining an updated copy of the data from all users POD and the removal of this data once access has been revoked by the user. The lack of authorization in the API and access control of the data in the Triplestore were also discussed. In order to gain the increased performance that can be offered by storing copies of data in the Triplestore, the application would have an initial delay the first time a user starts using it. This delay is caused by the duration it takes to extract the necessary data from the users POD and insert it in the Triplestore. Once the process is complete, the copy can be continued to be updated in the background of the application. Making copies of the users data in additional datastores can be considered to be breaking with the nature of Solid, as it is intended to completely return data ownership and control to the user, through data separation. Storing copies of the data several places on the web, could increase the risk of data breaches and misplacement.

The technologies used for transmitting data between applications and POD Servers will naturally evolve. And ultimately the need for additional datastores to store copies of

data originated from a users POD, will depend on the performance of these technologies. The value of this artifact lies in its superior performance to the PODS, where it has the benefit of aggregating the data before it is retrieved from the Triplestore. However the limitations of the artifact must be dealt with for it to be viable, either through the suggestions that was made or by other means. The debate of personal data ownership can affect what is considered as personal data, what can be owned by each individual, and how it is managed on the next stage of the web. This will likely decide whether or not storing copies of the data (through similar artifacts) breaks away from the concepts with Solid and decentralization in general.

7.1 Future Work

The artifact that was produced through the work of this thesis should be further developed, by resolving its limitations and optimising it for specific needs within different application use cases. As was discussed in chapter 6, the Web Socket Protocol can be used to solve several of the limitations but is expensive to scale. Moving the artifact to the cloud could potentially reduce the complexity of this task as well as the cost, as it can be easier to adjust the required hardware to the amount of users of the application at a time. Setting up different datastores for various application use cases can also become easier through cloud services.

This artifact is only one of many approaches that can be used in dealing with retrieval and aggregation of data, among all users of an application. There are likely other solutions that would be considered more Solid 'pure'. Some solutions might utilise a dedicated POD for the application itself. The applications POD could for example be used to store copies of aggregated data, instead of additional datastores. This could make it easier to control what is stored, and for it to be removed once access is revoked. For example in a social network application with posts and comments, it could be used to keep track of the users and their interactions. As an example: The POD could store the data and time of each post and count the number of interactions on each post. This data could be tied to the actual data describing the post, which is stored in the users POD. By using this 'metadata' about each post, it could potentially find the most recent or popular posts, which would reduce the need for aggregating all of the data after it is retrieved.

Appendix A

Tables

A.1 Descriptive Tables

The descriptive tables below are summaries of the original tables, that additionally contains the values of each of the 10 iterations and a graphical visualisation of these. Table A.1 shows the results of retrieving data directly from Solid PODS, and Table A.2 shows the time it took to aggregate the retrieved data. Table A.3 shows the results of retrieving increased data from the Triplestore, and Table A.4 shows the results of only retrieving the top 10 items among each experimental group. The original tables and datasets (CSV files) that was produced by the K6 load testing application can be shared on request.

Solid PODS - Retrieval Time

Groups		Min (s)	Median (s)	Max (s)		Average ms	Average s	Average m
1 Pod - Dataset 1		0,06	0,07	0,25		84,40	0,08	0,00
1 Pod - Dataset 10		0,05	0,06	0,08		61,96	0,06	0,00
1 Pod - Dataset 100		0,05	0,07	0,10		69,82	0,07	0,00
1 Pod - Dataset 1 000		0,06	0,06	0,18		74,64	0,07	0,00
10 Pods - Dataset 1		0,57	0,63	1,16		677,97	0,68	0,01
10 Pods - Dataset 10		0,54	0,64	0,78		648,67	0,65	0,01
10 Pods - Dataset 100		0,56	0,65	0,71		640,80	0,64	0,01
10 Pods - Dataset 1 000		0,58	0,66	0,85		681,67	0,68	0,01
100 Pods - Dataset 1		5,97	6,41	8,54		6546,72	6,55	0,11
100 Pods - Dataset 10		6,15	6,44	6,80		6481,24	6,48	0,11
100 Pods - Dataset 100		5,80	6,47	7,32		6482,07	6,48	0,11
100 Pods - Dataset 1 000		6,32	6,75	9,99		7012,66	7,01	0,12
1 000 Pods - Dataset 1		63,25	64,80	68,70		65112,60	65,11	1,09
1 000 Pods - Dataset 10		63,81	64,83	67,88		65164,69	65,16	1,09
1 000 Pods - Dataset 100		63,39	64,50	70,05		65278,24	65,28	1,09
1 000 Pods - Dataset 1 000		68,68	69,80	72,70		70347,33	70,35	1,17
10 000 Pods - Dataset 1		645,67	648,80	672,27		652724,06	652,72	10,88
10 000 Pods - Dataset 10		641,51	651,11	664,94		651743,23	651,74	10,86
10 000 Pods - Dataset 100		647,86	653,41	685,41		656619,84	656,62	10,94

Table A.1: Shows descriptive values of retrieving data directly from Solid Pods.

Solid PODS - Aggregation Time

Groups		Min	Median	Max		Average ms	Average s	Average m		Number of Items
1 Pod - Dataset 1		0,3	0,5	0,9		0,5	0,0	0,0		1,0
1 Pod - Dataset 10		0,6	0,8	2,2		1,0	0,0	0,0		10,0
1 Pod - Dataset 100		2,6	3,0	3,5		3,0	0,0	0,0		100,0
1 Pod - Dataset 1000		22,1	32,2	35,5		30,2	0,0	0,0		1000,0
10 Pods - Dataset 1		0,6	0,8	1,2		0,8	0,0	0,0		10,0
10 Pods - Dataset 10		2,3	2,9	3,3		2,9	0,0	0,0		100,0
10 Pods - Dataset 100		23,5	25,5	32,2		26,4	0,0	0,0		1000,0
10 Pods - Dataset 1000		287,0	311,0	357,5		314,4	0,3	0,0		10000,0
100 Pods - Dataset 1		2,9	3,4	5,3		3,9	0,0	0,0		100,0
100 Pods - Dataset 10		20,1	23,6	26,9		23,7	0,0	0,0		1000,0
100 Pods - Dataset 100		220,7	236,8	268,9		239,7	0,2	0,0		10000,0
100 Pods - Dataset 1000		2938,8	3119,3	3426,3		3138,7	3,1	0,1		100000,0
1 000 Pods - Dataset 1		26,7	30,6	37,4		31,8	0,0	0,0		1000,0
1 000 Pods - Dataset 10		218,0	231,9	247,6		233,0	0,2	0,0		10000,0
1 000 Pods - Dataset 100		2249,7	2358,7	2518,9		2374,6	2,4	0,0		100000,0
1 000 Pods - Dataset 1000		28157,3	29425,4	31073,3		29497,1	29,5	0,5		1000000,0
10 000 Pods - Dataset 1		275,8	288,2	326,0		292,9	0,3	0,0		10000,0
10 000 Pods - Dataset 10		1956,4	2066,5	2229,1		2087,1	2,1	0,0		100000,0
10 000 Pods - Dataset 100		19542,4	20187,5	29336,9		21082,8	21,1	0,4		1000000,0

Table A.2: Shows descriptive values of aggregating data from Solid Pods in the client.

Triplestore - Retrieval Time

Groups		Min (s)	Median (s)	Max (s)		Average ms	Average s	Average m
1 Pod - Dataset 1		0,02	0,03	21,05		2133,83	2,13	0,04
1 Pod - Dataset 10		0,02	0,02	0,06		35,11	0,04	0,00
1 Pod - Dataset 100		0,05	0,09	0,11		82,78	0,08	0,00
1 Pod - Dataset 1 000		0,35	0,36	0,56		383,87	0,38	0,01
10 Pods - Dataset 1		0,02	0,06	0,22		57,42	0,06	0,00
10 Pods - Dataset 10		0,04	0,05	0,18		73,85	0,07	0,00
10 Pods - Dataset 100		0,34	0,36	0,45		367,94	0,37	0,01
10 Pods - Dataset 1 000		3,47	3,56	3,79		3586,76	3,59	0,06
100 Pods - Dataset 1		0,04	0,06	0,73		131,24	0,13	0,00
100 Pods - Dataset 10		0,34	0,39	0,60		407,39	0,41	0,01
100 Pods - Dataset 100		3,44	3,57	3,78		3589,76	3,59	0,06
100 Pods - Dataset 1 000		35,02	35,40	35,70		35391,81	35,39	0,59
1 000 Pods - Dataset 1		0,33	0,38	1,83		516,38	0,52	0,01
1 000 Pods - Dataset 10		3,45	3,53	3,78		3552,56	3,55	0,06
1 000 Pods - Dataset 100		34,80	35,29	35,71		35245,87	35,25	0,59
1 000 Pods - Dataset 1 000		348,44	351,74	354,44		351550,29	351,55	5,86
10 000 Pods - Dataset 1		3,35	3,46	5,86		3701,80	3,70	0,06
10 000 Pods - Dataset 10		34,55	35,11	35,43		35059,13	35,06	0,58
10 000 Pods - Dataset 100		350,48	351,62	365,67		353532,46	353,53	5,89

Table A.3: Shows descriptive values of retrieving data from the Triplestore.

Triplestore - Retrieval Time of aggregated data

Group		Min (s)	Median (s)	Max (s)		Average ms	Average s	Average m
1 Pod - Dataset 1		0,01	0,01	0,06		22,10	0,02	0,00
1 Pod - Dataset 10		0,01	0,02	0,06		31,79	0,03	0,00
1 Pod - Dataset 100		0,01	0,01	0,06		29,73	0,03	0,00
1 Pod - Dataset 1 000		0,02	0,02	0,07		37,43	0,04	0,00
10 Pods - Dataset 1		0,01	0,01	0,06		23,56	0,02	0,00
10 Pods - Dataset 10		0,01	0,01	0,06		28,74	0,03	0,00
10 Pods - Dataset 100		0,01	0,02	0,06		28,86	0,03	0,00
10 Pods - Dataset 1 000		0,04	0,04	0,10		55,55	0,06	0,00
100 Pods - Dataset 1		0,01	0,02	0,06		32,44	0,03	0,00
100 Pods - Dataset 10		0,02	0,02	0,06		30,92	0,03	0,00
100 Pods - Dataset 100		0,04	0,05	0,12		60,78	0,06	0,00
100 Pods - Dataset 1 000		0,21	0,24	0,36		248,89	0,25	0,00
1 000 Pods - Dataset 1		0,04	0,05	0,12		61,92	0,06	0,00
1 000 Pods - Dataset 10		0,06	0,07	0,13		83,71	0,08	0,00
1 000 Pods - Dataset 100		0,23	0,27	0,43		295,50	0,30	0,00
1 000 Pods - Dataset 1 000		1,94	2,01	2,16		2019,54	2,02	0,03
10 000 Pods - Dataset 1		0,27	0,32	0,53		335,52	0,34	0,01
10 000 Pods - Dataset 10		0,47	0,51	0,93		569,74	0,57	0,01
10 000 Pods - Dataset 100		2,16	2,23	2,48		2288,58	2,29	0,04

Table A.4: Shows descriptive values of retrieving aggregated top 10 items from the Triplestore.

A.2 Factorial Tables

The factorial tables below are produced from the average values (measured in seconds) of the descriptive tables in the previous section.

Solid PODS - Retrieval Time

	D: 1 Item	D: 10 Items	D: 100 Items	D: 1 000 Items
U: 1	0,08	0,06	0,07	0,07
U: 10	0,68	0,65	0,64	0,68
U: 100	6,55	6,48	6,48	7,01
U: 1 000	65,11	65,16	65,28	70,35
U: 10 000	652,72	651,74	656,62	x

Table A.5: Shows the average duration of retrieving data directly from Solid Pods, measured in seconds.

Solid PODS - Aggregation Time

	D: 1 Item	D: 10 Items	D: 100 Items	D: 1000 Items
U: 1	0,00	0,00	0,00	0,03
U: 10	0,00	0,00	0,03	0,31
U: 100	0,00	0,02	0,24	3,14
U: 1000	0,03	0,23	2,37	29,50
U: 10000	0,29	2,09	21,08	x

Table A.6: Shows the average duration of aggregating data, measured in seconds. Groups with values of zero takes less than a second to complete.

Solid PODS - Retrieval + Aggregation Time

	D: 1 Item	D: 10 Items	D: 100 Items	D: 1000 Items
U: 1	0,08	0,06	0,07	0,10
U: 10	0,68	0,65	0,67	0,99
U: 100	6,55	6,50	6,72	10,15
U: 1 000	65,14	65,39	67,65	99,85
U: 10 000	653,01	653,83	677,70	x

Table A.7: Shows the average duration of retrieving and aggregating data, measured in milliseconds.

Triplestore - Retrieval Time

	D: 1 Item	D: 10 Items	D: 100 Items	D: 1 000 Items
U: 1	0,03	0,04	0,08	0,38
U: 10	0,06	0,07	0,37	3,59
U: 100	0,13	0,41	3,59	35,39
U: 1 000	0,52	3,55	35,25	351,55
U: 10 000	3,70	35,06	353,53	x

Table A.8: Shows the average duration of retrieving data from the Triplestore, measured in seconds.

Triplestore - Retrieval Time of aggregated data

	D: 1 Item	**D: 10 Items**	**D: 100 Items**	**D: 1000 Items**
U: 1	0,02	0,03	0,03	0,04
U: 10	0,02	0,03	0,03	0,06
U: 100	0,03	0,03	0,06	0,25
U: 1000	0,06	0,08	0,3	2,02
U: 10000	0,34	0,57	2,29	

Table A.9: Shows the average duration of retrieving aggregated data from the Triplestore, measured in seconds.

Appendix B

Code Snippets

The snippets of code below were used by the K6 application to retrieve data from Solid PODS and the RDF Triplestore. The entire scripts can be shared on request.

B.1 Experiment 1 - Solid PODS

```
function performExperiment(pods, dataset) {

  for (let index = 0; index < pods; index++) {

    const datasetURL = `https://user${index}.localhost:8443/public/${dataset}`

    http.get(datasetURL)
  }
}
```

B.2 Experiment 3 - Triplestore

All data

```

function performExperiment(pods, dataset) {

    const query = `

        PREFIX type: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        PREFIX container: <http://www.w3.org/ns/ldp#Container>
        PREFIX contains: <http://www.w3.org/ns/ldp#contains>
        PREFIX containsI: <http://www.schema.org/Contains>
        PREFIX person: <http://schema.org/Person>

        SELECT ?subject ?predicate ?object

          WHERE {

            {
                SELECT *
                WHERE {?webid type: person: .
                } LIMIT ${pods}
            }


            BIND (URI(REPLACE(STR(?webid),
            "/profile/card#me", "", "i")) AS ?podBase)

            BIND (URI(CONCAT(STR(?podBase),
            "/public/${dataset}")) AS ?dataset)
```

```
            ?dataset containsI: ?subject .
            ?subject ?predicate ?object .

        }
    `; // END OF QUERY.

      const data = JSON.stringify({ query: query });

      const params = {
        headers: { 'Content-Type': 'application/json' }
      };

      http.post(
          'http://localhost:3001/api/triplestore/get',
          data,
          params
      );

};
```

Top 10 Items

```

function performExperiment2(pods, dataset) {

  const topKItems = 10;

  const query = `
    PREFIX type: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    PREFIX container: <http://www.w3.org/ns/ldp#Container>
    PREFIX contains: <http://www.w3.org/ns/ldp#contains>
    PREFIX containsI: <http://www.schema.org/Contains>
    PREFIX person: <http://schema.org/Person>
    PREFIX url: <http://schema.org/Thing/url>

    SELECT ?object ?objectCount
      WHERE {

        {
          SELECT ?object (count(?object) as?objectCount)
          WHERE {

            {
                SELECT *
                WHERE {
                    ?webid type: person: .
                } LIMIT ${pods}
            }


              BIND (URI(REPLACE(STR(?webid), "/profile/card#me", "", "i")) AS ?podBase)

              BIND (URI(CONCAT(STR(?podBase), "/public/${dataset}")) AS ?dataset)

              ?dataset containsI: ?subject .
              ?subject url: ?object .

          } GROUP BY ?object

        }

      }
        GROUP BY ?bookmark ?object ?objectCount
        ORDER BY DESC(?objectCount)
        LIMIT ${topKItems}
  `;

  const data = { query: query };

```

```
48    const param = { headers: { 'Content-Type': 'application/json' }, timeout: 1000000 };
49
50    const response = http.post('http://localhost:3001/api/get', JSON.stringify(data),
        param);
51
52  };
```

Bibliography

[1] Inrupt accessed: 20052022 Solid URL https://inrupt.com/solid (document), 1.1

[2] Berners-Lee T accessed: 01022022 30 years on, whats next fortheweb? URL https://webfoundation.org/2019/03/web-birthday-30/ 1

[3] Foundation W accessed: 20022022 Leave no one behind: A people-centered approach to achieve meaningful connectivity URL https://webfoundation.org/2021/04/leave-no-one-behind-a-people-centered-approach-to-achieve-meaningful-conn 1

[4] Shoshana Z 2019 *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power* (PublicAffairs; 1st edition (January 15, 2019)) ISBN 978-1610395694 1

[5] James B 2020 *The System: Who Owns the Internet, and How It Owns Us* (Bloomsbury Publishing (20 Aug. 2020)) ISBN 978-1526607249 1

[6] Commision E U accessed: 20022022 Two years of the gdpr: Questions and answers URL https://ec.europa.eu/commission/presscorner/detail/en/qanda_20_1166 1

[7] Website S accessed: 22022022 The solid project URL https://solidproject.org/ 1

[8] Protocol S accessed: 22022022 The solid protocol URL https://solidproject.org/TR/protocol 1

[9] An D accessed: 17032022 Find out how you stack up to new industry benchmarks for mobile page speed URL https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/ 1.1, 4.2

[10] Rissacher L accessed: 20012022 Constructive criticism from an experienced developer URL https://forum.solidproject.org/t/constructive-criticism-from-an-experienced-developer/3521 1.1, 4

[11] Weiss T accessed: 30042022 General questions and clarifications on app development URL https://forum.solidproject.org/t/general-questions-and-clarifications-on-app-development/201/8 1.1, 4

[12] Consortium W W W accessed: 20052022 Tim berners-lee URL https://www.w3.org/People/Berners-Lee// 2.1

[13] Bodoni S accessed: 20052022 Meta slapped with $19 million fine for eu data law breaches URL https://www.bloomberg.com/news/articles/2022-03-15/meta-slapped-with-19-million-fine-for-eu-privacy-law-violations 2.1

[14] Wong J C accessed: 20052022 Facebook says nearly 50m users compromised in huge security breach URL https://www.theguardian.com/technology/2018/sep/28/facebook-50-million-user-accounts-security-berach 2.1

[15] Web T S accessed: 20052022 Tim berners-lee URL https://www.w3.org/2000/Talks/0906-xmlweb-tbl/text.htm 2.1

[16] Protocol S accessed: 17022022 Solid protocol specification URL https://solidproject.org/TR/protocol#reading-writing-resources 2.2

[17] Connect S O Solid-oidc URL https://solidproject.org/TR/oidc 2.2

[18] Inrupt accessed: 20052022 Proving the possible: Introducing the inrupt enterprise solid server URL https://inrupt.com/enterprise-server-release 2.2.1

[19] Inrupt accessed: 20052022 The flanders government and solid: an important milestone in flemish history URL https://inrupt.com/flanders-solid 2.2.1

[20] Hevner A, R A, March S, T S, Park, Park J, Ram and Sudha 2004 *Management Information Systems Quarterly* **28** 75– 3.1, 3.1.1, 4

[21] Weber R 2017 *Research Methods* (Chandos Publishing (Oxford) Ltd) chap 11 3.1.1

[22] Tanner K 2017 *Research Methods* (Chandos Publishing (Oxford) Ltd) chap 14 3.2

[23] Website L accessed: 20052022 Performance testing URL https://loadninja.com/performance-testing/ 3.3

[24] Hamilton T accessed: 20052022 Performance testing tutorial: What is, types, metrics example URL https://www.guru99.com/performance-testing.html 3.3.2

[25] Sheard J 2017 *Research Methods* (Chandos Publishing (Oxford) Ltd) chap 18 3.4, 3.4.1

[26] Veres C accessed: 17032022 Lexitags: An interlingua for the social semantic web URL https://www.academia.edu/52345256/LexiTags_An_Interlingua_for_the_Social_Semantic_Web 4.1.1

[27] Services A W accessed: 29052022 Amazon api gateway quotas and important notes URL https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html 6.2.2

[28] Paulius Jurcys e September 21, 2021 Ownership of user-held data: Why property law is the right approach URL https://jolt.law.harvard.edu/assets/digestImages/Paulius-Jurcys-Feb-19-article-PJ.pdf 6.2.2