FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**MASTER THESIS**

Jakub Matějík

# A platform for creating and sharing interactive educational materials based on data from knowledge graphs

Department of Software Engineering

Supervisor of the master thesis: doc. Mgr. Martin Nečaský, Ph.D.
Study programme: Computer Science
Study branch: ISDI

Prague 2023

I would like to thank Martin Nečaský and Jakub Klímek for the time they gave me in supervising and supporting this thesis. I would also like to thank my family and girlfriend who have always been there for me.

Title: A platform for creating and sharing interactive educational materials based on data from knowledge graphs

Author: Jakub Matějík

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., department

Abstract: In this thesis, we will explore the possibilities of Solid technology and the use of Solid for application implementation. This application will be used to create mind maps and social networking for teachers and students. In order to do this, we will analyze the features of Solid for designing a social network application and its features and suitable use cases. Based on this analysis and user requirements, we will design the architecture along with the operations to make them convenient for the user. Next, we will learn about the concepts in Linked Data and analyze knowledge bases for use in our application. The result is implementing a proof-of-concept application that allows the intuitive creation of mind maps and their sharing within classes.

Keywords: Solid DBpedia Education

# Contents

# Introduction

With the rapid development of digital technologies and the Internet, the number of applications and the associated generation of huge amounts of personal data has increased significantly in recent years. Therefore, there is a growing interest in privacy and data protection issues. Traditional centralized platforms such as Facebook have become a place where users' data is collected, analyzed, and sometimes sold to third parties without users' consent. These reports are worrying and raise concerns about privacy and control over personal data.

Therefore, an increasingly interesting alternative is the Solid[25] platform developed by Tim Berners-Lee, founder of the World Wide Web. Solid offers a new model for a decentralized web that emphasizes privacy, user control over data, and transparency. Solid seeks to shift data control back into users' hands, allowing them to determine what data they share and with whom.

One of the main benefits is privacy and enhanced data security. Users can control what data they share with each application in Solid and can change or revoke these permissions anytime. This makes Solid immune to the problems associated with data loss and unauthorized data sales that traditional centralized platforms suffer from.

As a decentralized platform emphasizing privacy and user control over data, Solid has great potential to bring innovation to the most disruptive areas.

Although modern technologies have become integral to our daily lives, their potential is often underutilized in Czech education[22].

Interactive technologies such as tablets, interactive whiteboards, or online learning platforms offer a wide range of possibilities for enriching learning. These technologies can increase student engagement and promote creativity and active learning.

Several factors may contribute to this under-use. One of the main reasons may be the need for more financial resources to purchase and maintain technological equipment. Schools often need more funds to purchase modern equipment and licenses for necessary software. In addition, teachers may need more training to become familiar with new technologies and learn how to incorporate them into their teaching effectively.

This lack of use of modern technology in teaching has consequences. Students may lose interest and motivation because they are taught outdated methods that do not match their behavior and learning needs. While interactive learning interests children and encourages active participation and engagement, traditional teaching methods can be restrictive and boring.

In this thesis, we focus on the emerging SOLID platform. We will first introduce this platform's basic concepts, analyze its strengths and weaknesses, and present them on different types of applications.

For this thesis, we will then focus on using SOLID in a proof-of-concept application that aims to create an interactive mind map application. This proof-of-concept application will exploit the potential of open knowledge bases.

Open knowledge bases are publicly available and freely usable databases that contain structured information. These databases provide the public and researchers easy access to facts, data, scientific studies, publications, articles, and other relevant information. We want to analyze the available options and propose one that could be used as an aid to the user of our application in creating teaching materials.

## Goals of this thesis

- Introduce concepts related to the SOLID platform. Then evaluate which knowledge base will be useful for offering similar entities.
- To propose the concept of data storage and sharing when using the SOLID platform for a social network.
- Implement a proof-of-concept application that allows users to create interactive mind maps.

## Non-goals of this thesis

- The thesis will not address any issues related to teaching in schools or changing the teaching method.
- We will use the knowledge base only as a source of data for querying and will not create a model for its use in recommendation.

# 1. Concepts

This section explains to the reader the terms we will use in this paper. First, we introduce the concepts of Linked Data, decentralization in the Internet environment, and data storage in order to have a foundation for understanding the SOLID technology itself. We then focus specifically on the SOLID platform. At the end of this chapter, we will introduce the concept of knowledge bases used for machine processing of encyclopedic data, which we intend to use in this thesis.

## 1.1 Linked data

First, let's focus in more detail on the concept of linked data. It is a fairly large set of technologies and protocols that are managed by the World Wide Web Consortium (W3C). Especially the Data Platform Working Group and the Social Web Working Group. These maintain the Linked Data(LD) and LD platform.

The concept of Linked Data[3] goes beyond simply putting data on the web. It involves creating meaningful connections between different pieces of data, enabling both humans and machines to explore and navigate this interconnected web of information. By leveraging linked data, one can discover related datasets when provided with a portion of the data.

Similar to the hypertext web, which is built upon interconnected HTML documents, the web of data is constructed using documents on the web. However, unlike hypertext web links that are embedded within HTML documents, data links are established between various entities described using RDF (Resource Description Framework). The use of URIs (Uniform Resource Identifiers) allows the identification of any type of object or concept, whether it is an HTML document or an RDF resource.

### 1.1.1 The Four Rules of Linked Data

These steps, often referred to as rules, define expectations for behavior rather than strict requirements. Not following these rules does not cause any harm but misses the opportunity to interconnect data. The unexpected reuse of information across the web adds value to the Semantic Web.

- **Identify things with URIs** Most practitioners in semantic web technology understand the importance of using universal URI symbols. Not using them excludes the project from being labeled as part of the Semantic Web.

- **Use HTTP URIs** While widely understood, there has been a tendency to invent new URI schemes outside the established Domain Name System (DNS), leading to fragmentation. Understanding that HTTP URIs are names and not addresses is crucial.

- **Serve information against URIs** Although mostly followed for ontologies, major datasets were not published on the web against a URI in 2006. Providing information about properties, classes, and relationships within the RDF, RDFS, and OWL ontologies is essential.

- **Make links elsewhere** To connect existing data into a web, linking is necessary. Just as it is considered good practice to link to related external material on hypertext websites, linking in the Semantic Web adds value to the information.

After this introduction to Linked Data, we will introduce the technologies that help structure data and create machine-readable connections between data.

### 1.1.2 RDF

RDF is a versatile framework for representing interconnected data on the web[26]. RDF organizes information based on relationships, serving as the foundation for the semantic web. It enables the standardized exchange of metadata, facilitating data integration from various sources.

RDF provides a standardized way to describe and structure data on the web. RDF is used in various fields such as ontologies, metadata descriptions, databases, etc. RDF enables efficient linking of data from different sources and simplifies the process of data sharing and interoperability on the web.

#### RDF form

An RDF statement consists of three components, referred to as a triple, as we can see in Figure 1.1.

- **Subject** Subject is a resource being described by the triple.
- **Predicate** Predicate describes the relationship between the subject and the object.
- **Object** Object is a resource that is related to the subject.

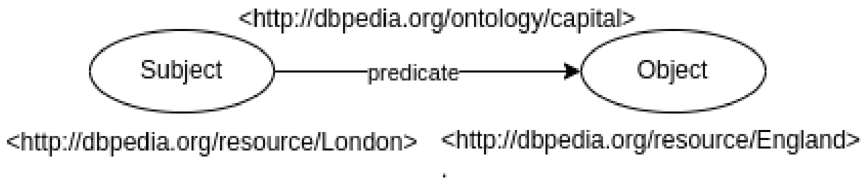

<http://dbpedia.org/ontology/capital>
Subject
predicate
Object
<http://dbpedia.org/resource/London>
<http://dbpedia.org/resource/England>

Figure 1.1: RDF triple

This triple represents the relationship between London, the subject of the ”capital” property, and England, the object.

Nowadays the World Wide Web Consortium (W3C) maintains the standards for RDF, including foundational concepts and specifications for different formats. Initially, RDF specification only included XML syntax. However, the standards for encoding RDF statements have expanded to incorporate three different syntaxes:

### Turtle

Considered the most widely used text syntax for RDF statements, Turtle is described by the W3C as a ”compact and natural text form.” It includes abbreviations for commonly used patterns, making it more concise and readable[2].

```
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbo: <http://dbpedia.org/ontology/>.

dbpedia:London dbo:capital dbpedia:England.
```

Listing 1.1: Turtle Code

### JSON-LD

This syntax utilizes the JSON (JavaScript Object Notation) format for RDF statements. It combines the flexibility of JSON with the semantic expressiveness of RDF, enabling easy integration with web APIs and data interchange between applications.

```
{
  "@context": {
    "dbpedia": "http://dbpedia.org/resource/",
    "dbo": "http://dbpedia.org/ontology/"
  },
  "@id": "dbpedia:London",
  "dbo:capital": "dbpedia:England"
}
```

Listing 1.2: JSON-LD Code

### N-Triples

Designed as a simplified text-based format, N-Triples is a subset of the Turtle syntax. It aims to enhance the ease of writing RDF statements for humans and facilitates the creation and parsing of RDF statements by programs. Its straightforward structure makes it more accessible for both humans and software.

## 1.1.3 SPARQL

SPARQL is a query language for searching and manipulating data in RDF format. SPARQL has a standardized syntax and semantics for querying. Queries are formulated in a structured language similar to SQL, allowing flexible pattern-based querying and data analysis. SPARQL allows complex queries, and in addition to basic queries, SPARQL provides advanced features such as aggregation, subqueries, negation, and the ability to create values using expressions[11].

Queries are formulated in a structured language similar to SQL, allowing flexible pattern-based data retrieval and analysis. The results can be manual, depending on what the user needs. It can be in classic tabular form, RDF charts, or others. SPARQL language is used in various fields such as artificial intelligence, bioinformatics, linguistics, ontology, web information retrieval, and many other applications where linked and structured data must be worked with efficiently.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?country ?capital
WHERE {
  ?country a dbo:Country ;
          dbo:capital ?capital ;
          dbo:continent dbpedia:Europe .
}
```

Listing 1.3: Example SPARQL query to retrieve the capitals of European countries

The provided SPARQL query retrieves the capitals of European countries from the DBpedia dataset. It uses the dbo:Country class and the dbo:capital property to match countries with their corresponding capitals. The query also includes the condition dbo:continent dbpedia:Europe to ensure that only European countries are selected.

## 1.2 Decentralization

Decentralization refers to the process of delegating decision-making authority to lower levels within an organizational hierarchy. In hierarchical structures with multiple levels, decentralization is measured based on its extent. The higher the frequency of decision-making at lower levels, the greater the level of decentralization within the organization. It is important to note that decentralization can vary across different types of decisions, with an organization being more centralized in some areas and more decentralized in others. The optimal level of decentralization for a specific decision occurs when the decision-making authority is assigned to the appropriate level within the hierarchy, resulting in the maximum organizational performance given the prevailing circumstances[7].

## 1.3 SOLID

In the context of this thesis, decentralization refers to the decentralization of data. One of the technologies that enable decentralization is SOLID. SOLID, which is the shortcut for Social Linked Data, is a specification that describes how data can be stored and shared on the web using Linked Data principles. These principles include using URIs to identify data, using RDF as a format for data description, using ontologies to define concepts and relationships between them, and using open standards for data communication and retrieval.

Leveraging this specification allows users to store their data in so-called ”pods,” digital containers that users can control and share as they see fit. Pods are assigned a unique URI identifier, which ensures unique data identification and enables easy connection with other data on the web.
The SOLID specification focuses on decentralizing user data to store and manage data in so-called ”Solid Pods.” These user-owned forms are where they can store their data and decide who can see or edit it. This decentralized approach gives users more control over their data while providing the server with better data security since it is not stored in a centralized location[23].
Another aspect of decentralization in Solid is the absence of a central data repository and a central authority to control that data. Users have complete control over their data and can store, manage and share it as they see fit without the intervention of a central authority.
Decentralization can have several advantages for data privacy:

1. **Control over data** Decentralized systems grant users greater data control as their information is stored under their ownership, enabling them to dictate who can access it. This increased control empowers users to manage their data more effectively.

2. **Reducing the risk of data leakage** Decentralization can reduce the risk of data leakage because data is stored in different places and not on one central server. This reduces the risk of hacking attacks and the impact of some data getting out.

3. **Protection from unauthorized tracking** Decentralization can also protect users from tracking by third parties such as governments and companies. If the data is decentralized, it is not easy to track users and their behavior on the web.

4. **Easier Data Linkage** Decentralized systems allow for more accessible data linkage because data is linked using standards such as RDF and Linked Data. This allows users to find and link data from different sources easily.

### 1.3.1 SolidPod

Solidpod is a decentralized platform for storing and sharing personal data as part of the Solid project by Tim Berners-Lee. Solidpod enables the storage and sharing of personal data in a decentralized repository, allowing users to control and decide with whom they share their data[25].
To access Solidpod, a Solid identity is needed, a digital identity that allows access to data on Solidpod. Suppose users do not already have a Solid identity; it is possible to create one at https://solidproject.org/get-a-solid-pod.
After creating Solid identity, it is possible to access data on a Solidpod using a dedicated client such as the Solid Community Pod Explorer or the Solid programming API to access data from own application. The Solid API provides a standard interface for reading and writing data to Solidpod and allows interaction with the application’s data[20].

### 1.3.2 WebID

Solid relies on the Web Identity and Discovery (WebID) specification, which defines the concepts of identity and authentication[29]. WebIDs are HTTP URIs used for user identification. Each Solid Pod has a WebID associated with it, and it is safeguarded by two authentication mechanisms: WebID-OIDC and WebID-TLS.
WebID-OIDC is a decentralized authentication protocol that extends the ”Open ID Connect (OIDC)” protocol. It involves user authentication using a username and password, and upon successful authentication, the identity provider issues an ID token for authorized access to the module. Alternatively, WebID-TLS offers an authentication method utilizing a TLS certificate to verify authenticity.
To utilize WebID, each Pod must have an RDF representation of the WebID profile that can be accessed by dereferencing the WebID URI. This profile document may contain additional user information like name, avatar, and email address and serves as a public document. Users can exclude personal data from the WebID Profile document if they wish to keep it private but including the ”foaf:name” predicate is mandatory.

## 1.4 Knowledge base

A knowledge base is a collection of information and knowledge organized and structured so that computer systems can easily access and process it. A knowledge base contains statements about the world and describes the relationships between different entities. These entity relationships allow computer systems to perform sophisticated tasks such as answering queries, retrieving information, and participating in natural language dialogue.
The knowledge base can be general in nature or domain-specific. Some knowledge bases are created automatically from large-scale sources such as Wikipedia, while others are created by the community or organizations specializing in a particular domain. Knowledge bases serve as an essential source of information for computer systems and enable automated processing and understanding of data.
Modern approaches consider the inherent incompleteness of knowledge bases and often combine structured resources with unstructured data. Knowledge bases, also called knowledge graphs, serve as a ”semantic backbone” and are used with lightweight subsumption ontologies that emphasize relationships between entities. There are various knowledge bases, including DBpedia, YAGO, and proprietary databases managed by prominent search providers such as Google, Microsoft, and Facebook[1].

# 2. Analysis

The main idea of this work is to create an application for creating educational materials with elements of a social network, thanks to which it would be possible to associate with other users in classes.
This chapter will first introduce the actor types that will access our application. Let us introduce their rights and allowed operations. After defining the actor, we will summarize the requirements of all system users, and the non-functional requirements for the application no longer depend on the user's requirements but are no less important.

## 2.1 Target group

The application will focus on the community of students and teachers who want to expand teaching and allow them to look at the teaching materials from a different perspective. Before we summarize user requirements in the form of user stories, let us suppose the individual actors who will appear in user stories.

## 2.2 Roles

First, let us imagine two roles that the application user can have and for which we will consider the following functional requirements.

#### User

Actor, who owns SolidPod and authenticates to the application using his SolidPod. As a user of the application I have rights to create mind maps and classes, send messages and set up a profile in this application.

#### Teacher

When I create my class as a user of the application, then I become a teacher within the class. As a teacher, I have the right to add and remove students and add class mind maps. As a teacher, I also see the test results of all students in the class.

#### Student

When I was added to the class as a student. As a result, I have limited rights in the classroom compared to the teacher. I only have the right to view the material and write the tests.

## 2.3 Nonfunctional requirements

In this section, we summarize the non-functional requirements for our application. We will emphasize implementing the application over a decentralized platform with non-functional requirements.

**Non-functional req. 1.1 - SOLID** The application for working with data uses the SOLID platform.

**Non-functional req. 1.2 - Select provider** The application must authenticate the user's Solid POD account via WebID-TLS and WebID-OIDC.

**Non-functional req. 1.3 - Web app** The goal is to create a web application to which the user will have easy access, i.e., even though the application is built on a platform, it should not limit users in any way compared to regular applications.

**Non-functional req. 1.4 - User-friendly** When designing the application's user interface, I want to focus on the most intuitive environment possible. Work in it should be easy for the user to understand, even without help or with less help. E.g., when completing the test, the difference between the results of the paper version and the test in the application should be minimal.

**Non-functional req. 1.5 - CI/CD** To make the application easy to deploy, a script will be prepared.

**Non-functional req. 1.6 - Compatibility** The application should be compatible with all currently most used browsers.

**Non-functional req. 1.7 - Mobile friendly** The application environment should also be friendly to mobile devices.

**Non-functional req. 1.8 - Compatibility with latest tools** The SOLID platform is still fresh, and the development of tools for working with this platform is still very dynamic. Therefore, we will want to use current tools.

**Non-functional req. 1.9 - User documentation** The application shall have a user documentation explaining its functionality.

### 2.3.1 User stories for users

First, we will go through user stories for ordinary users who visit the application without having a role within the created class.

**User story 1.1 - Homepage** As a user, after logging in to the application, I first want to see a list of my mind maps on the main screen that I have saved on my datapod because, most often, I want to log in to the application and view the mind map I created.

**User story 1.2 - Manage your mind maps** As a user, I want to manage my mind maps. I want to be able to rename or delete them.

**User story 1.3 - Visual form of mind map** As a user, I want the mind map to be simple and display individual entities as text in squares. I also want it to show the connection between entities.

**User story 1.4 - Viewing the mind map** As a user, I want the map to be interactive. To be able to scroll, zoom in and out.

**User story 1.5 - Searching** As a user who creates a mind map, I want to be able to display a prediction based on a keyword.

**User story 1.6 - Edit before adding** As a user, I want to be able to add an entity description from the knowledge base but be able to edit it before adding it.

**User story 1.7 - Searching** As a user, I want to have a search history and be able to go back in it.

**User story 1.8 - Custom nodes** As a user I want to add my own entity to the mind map.

**User story 1.9 - Edit nodes** As a user, I want to re-edit an existing entity.

**User story 1.10 - Similar entities** As a user, I want to get suggestions for a similar entity

**User story 1.11 - Color** As a user, I want to be able to change the color of each entity in the mind map.

**User story 1.12 - Remove node** As a user, I want to be able to remove an entity.

**User story 1.13 - Timeline view of the mind map** As a user I want to be able to display the time data of added entities as cards in chronological order.

**User story 1.14 - Aggregated list of events in periods** I also want to display all events as an aggregated list by century, decade, or year.

**User story 1.15 - Export mind map** As a user who created his mind map and would like to share it without needing the application. Therefore, I would like to export it in one of the classic image formats.

**User story 1.16 - Edit username** As a user, I want to be able to enter my username within the application.

**User story 1.17 - Create class** I want to create my class as a user. Then I will become a teacher in this class. I can add my students to this class.

**User story 1.18 - List of classes** As a user, I want to see an overview of all courses in which I am a teacher or student in one page.

**User story 1.19 - Messages** As a user, I want to see a list of my private chats.

### 2.3.2 User stories for teacher

First, let's go through the user stories for teachers. A teacher is a user who has created at least one class and accessed its main page. On this page he has different options from a student.

**User story 2.1 - Invitation to class** As a teacher in my course, I want to invite students. I want to do this by sending a link.

**User story 2.2 - Confirmation** I want to confirm assignments as a teacher.

**User story 2.3 - List of requests** As a teacher, I want to see all incoming requests to join a class on the page of all my classes.

**User story 2.4 - Class mind map** As a teacher I want to create mind maps in the classroom.

**User story 2.5 - Mind map test** As a teacher, I want to create a test from a mind map by hiding some of my chosen node names.

**User Story 2.6 - The class from the teacher's point of view** As a teacher, I want to have a clear view of all the tests performed in the classroom on the class page. Moreover, I also want to see a list of teaching materials and students.

**User story 2.7 - Messages** I want to create private chat with students in messages as a teacher.

**User story 2.8 - Announcements** As a teacher, I want to make announcements for all students.

**User story 2.9 - Names** If the student has a first and last name, I want to see it instead of the webId.

**User story 2.10 - Browser mode** As a teacher I want users to be able to view mind maps only in browser mode, so they can't change the mind map.

### 2.3.3 User stories for student

And last but not least, we will look at user stories from the user's perspective, who needs to have a comfortable preparation for the subject in the application.

**User story 3.1 - Class page from the student's point of view** As an example I want to see mind maps on the class page, my past test results and other students.

**User story 3.2 - Take a test** As a student, I want to be able to use mind maps in the classroom to do a test.

**User story 3.3 - Test results** I want to see the results of the tests I have already accomplished.

**User story 3.4 - Send message to teacher** When I am added to a new class, I want the chat with that teacher to appear among my other chats

**User story 3.5 - Class announcements** I need to see the teacher's messages posted in the class.

**User story 3.6 - Node detail** As a student, I want to see the detail for each added entity in browser mode.

## 2.4 Use cases

On the basis of the user stories we will introduce use-cases. We will want to cover all cases of user requests and on the other hand we will omit some that are almost identical, like getting a list of mind maps, a list of user's classes or a list of his private chats.

To know if we have missed a requirement and are considering it, we have created a Table 2.1 for requirements coverage.

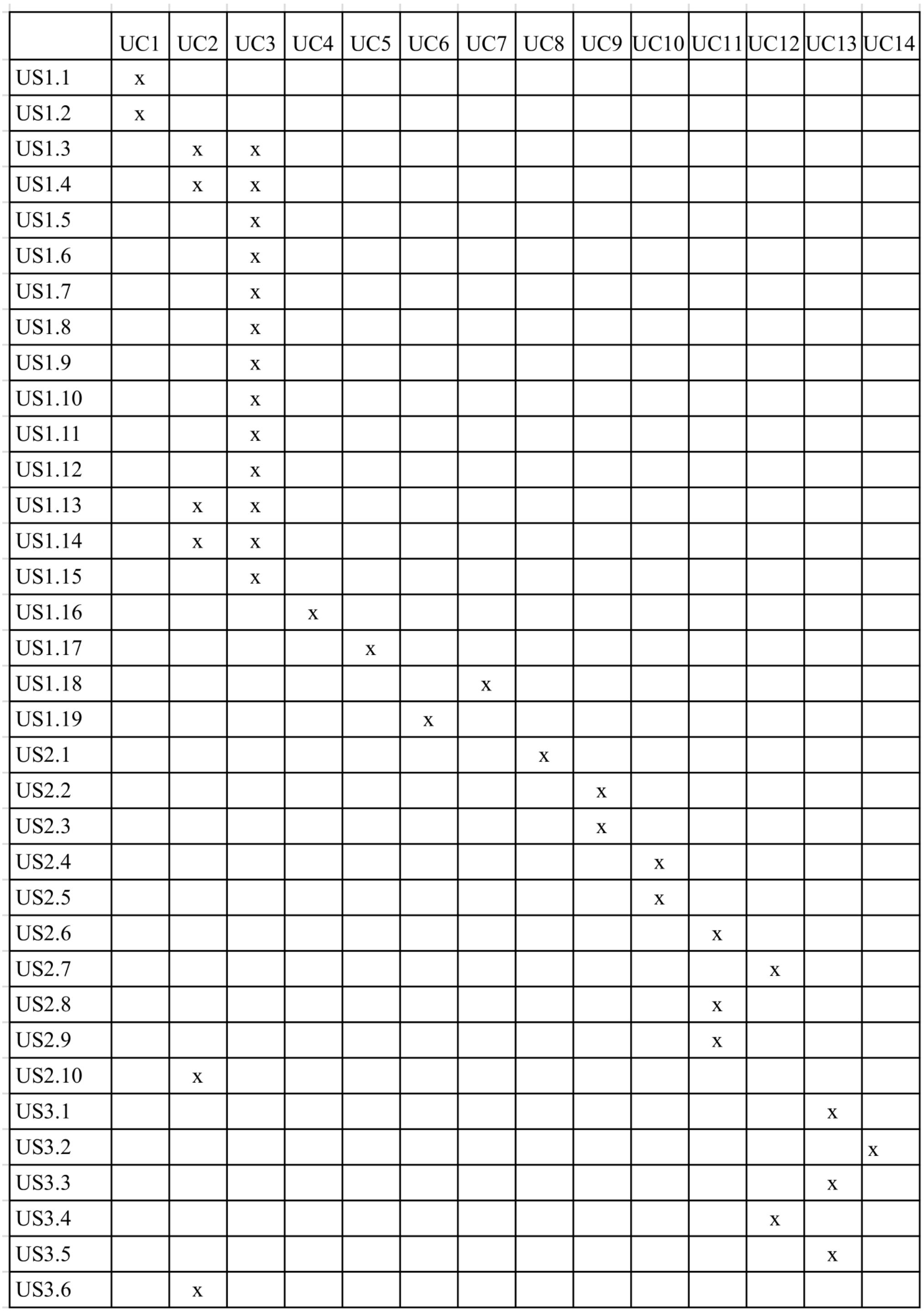
Here we introduce use-cases.

**1. Use case: List mind maps**

| | |
|---|---|
| Actor | User |
| Description | The user wants to list all his mind maps saved on his solid pod. |
| Input state | User is authenticated. |
| Scenario | User click a Home button in side menu and select Mind maps button. |
| Output state | User can see list of its maps. |

**2. Use case: Explore mind map**

| | |
|---|---|
| Actor | Student |
| Description | The student wants to explore classroom mind map. |
| Input state | Student is authenticated. |
| Scenario | 1. The student opens the menu and selects the Classes.<br>2. The student selects a class from the list of his classes.<br>3. The student chooses mind map to explore<br>4. The student can now in the mind map:<br>(a) Click on the node to see its detail.<br>(b) Pan and zoom through the mind map. |
| Output state | Student exploring mind map. |



| | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 | UC10 | UC11 | UC12 | UC13 | UC14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| US1.1 | x | | | | | | | | | | | | | |
| US1.2 | x | | | | | | | | | | | | | |
| US1.3 | | x | x | | | | | | | | | | | |
| US1.4 | | x | x | | | | | | | | | | | |
| US1.5 | | | x | | | | | | | | | | | |
| US1.6 | | | x | | | | | | | | | | | |
| US1.7 | | | x | | | | | | | | | | | |
| US1.8 | | | x | | | | | | | | | | | |
| US1.9 | | | x | | | | | | | | | | | |
| US1.10 | | | x | | | | | | | | | | | |
| US1.11 | | | x | | | | | | | | | | | |
| US1.12 | | | x | | | | | | | | | | | |
| US1.13 | | x | x | | | | | | | | | | | |
| US1.14 | | x | x | | | | | | | | | | | |
| US1.15 | | | x | | | | | | | | | | | |
| US1.16 | | | | x | | | | | | | | | | |
| US1.17 | | | | | x | | | | | | | | | |
| US1.18 | | | | | | | x | | | | | | | |
| US1.19 | | | | | | x | | | | | | | | |
| US2.1 | | | | | | | | x | | | | | | |
| US2.2 | | | | | | | | | x | | | | | |
| US2.3 | | | | | | | | | x | | | | | |
| US2.4 | | | | | | | | | | x | | | | |
| US2.5 | | | | | | | | | | x | | | | |
| US2.6 | | | | | | | | | | | x | | | |
| US2.7 | | | | | | | | | | | | x | | |
| US2.8 | | | | | | | | | | | x | | | |
| US2.9 | | | | | | | | | | | x | | | |
| US2.10 | | x | | | | | | | | | | | | |
| US3.1 | | | | | | | | | | | | | x | |
| US3.2 | | | | | | | | | | | | | | x |
| US3.3 | | | | | | | | | | | | | x | |
| US3.4 | | | | | | | | | | | | x | | |
| US3.5 | | | | | | | | | | | | | x | |
| US3.6 | | x | | | | | | | | | | | | |

Figure 2.1: Requirements coverage

**3. Use case: Edit mind map**

| | |
|---|---|
| Actor | User |
| Description | User wants to edit the mind map. |
| Input state | User is authenticated. |
| Scenario | 1. The user opens the menu and selects the Dashboard.<br>2. The user has two options how to find mind map for editing:<br>• The user click Create new mind map button on Mind map page, insert new name and click Create.<br>• The user chooses mind map from list of his mind maps to edit.<br>3. The user clicks the plus button on the bottom right to view the menu.<br>4. The user can now choose from the given operations:<br>(a) The user in the top search bar enters the search keyword and selects from the available options to add it to the mind map.<br>(b) After clicking on an entity in the mind map, the user presses the magnifying glass icon to find nearby entities of the same class.<br>(c) The user clicks on the Add custom button to add a custom entity, selects its title, desciption and presses the add button.<br>(d) After selecting an entity in the mind map by clicking on it, the user presses the palette button in the menu to change the color, in the opened window he selects the colors.<br>(e) Add connection to another node from selected node.<br>(f) To remove the node from mind map press the remove button.<br>5. The user save the changes by clicking button save on bottom left. |
| Output state | Mind map is saved with updates. |

**4. Use case: Update profile**

| | |
|---|---|
| Actor | User |
| Description | A user wants to view and edit their profile. |
| Input state | User is authenticated. |
| Scenario | 1. The user opens the menu and selects the profile.<br>2. The user can see their name and surname.<br>3. User enters updated data.<br>4. Clicking confirm will save the updated data. |
| Output state | User has updated data. |

**5. Use case: Create class**

| | |
|---|---|
| Actor | Teacher |
| Description | Teacher wants to create his new class. |
| Input state | Teacher is authenticated. |
| Scenario | 1. The user opens the menu and selects the Class.<br>2. Teacher clicks on the Create new class button.<br>3. Insert new class name.<br>4. Click on confirm to create a new class. |
| Output state | Teacher is redirected to the new class page. |

**6. Use case: Check messages**

| | |
|---|---|
| Actor | User |
| Description | User wants to check his messages. |
| Input state | User is authenticated. |
| Scenario | The user opens the menu and selects the Messages. |
| Output state | The user sees a list of all his conversations with other users with whom he is in classes. |

**7. Use case: Get classes**

| | |
|---|---|
| Actor | User |
| Description | The user wants to see which classes he is a member of. |
| Input state | User is authenticated. |
| Scenario | The user opens the menu and selects the Classes. |
| Output state | The user sees a list of all the classes he is a member of, including their names, and can go to their pages. |

**8. Use case: Copy invite link**

| | |
|---|---|
| Actor | Teacher |
| Description | Teacher wants to send his students a link to join the class. |
| Input state | Teacher is authenticated. |
| Scenario | 1. The user opens the menu and selects the Classes.<br>2. The teacher chooses the class to which he wants to invite students.<br>3. On the class page, click on the icon to copy the link to the class. |
| Output state | The teacher has a copied link in the clipboard that he can send to his students. |

**9. Use case: Send request**

| | |
|---|---|
| Actor | Student |
| Description | A student wants to send a request to be added to a class. |
| Input state | Student is authenticated and has a link to the class. |
| Scenario | 1. The user opens the menu and selects the Classes.<br>2. The student clicks on the button add a class.<br>3. Insert the received link to the class.<br>4. Click on the send request. |
| Output state | The student has sent a request for an assignment. |

**10. Use case: Add classroom material**

| | |
|---|---|
| Actor | Teacher |
| Description | Teacher wants to add new teaching material to the classroom. |
| Input state | Teacher is authenticated. |
| Scenario | 1. The user opens the menu and selects the Classes.<br>2. The teacher chooses the class to which he wants to add new mind map.<br>3. The teacher clicks on the add new button to create a new mind map for the class. |
| Output state | The teacher is redirected to the editor, where he can edit the map for the class (see use case Edit map). |

**11. Use case: Class message**

| | |
|---|---|
| Actor | The teacher |
| Description | Teacher wants to add a message to the class page for all students. |
| Input state | Sender is authenticated. |
| Scenario | 1. The user opens the menu and selects the Classes.<br>2. The teacher selects a class from the list of his classes.<br>3. The teacher selects the add class message button. |
| Output state | The announcement is added and can be seen by all students |

**12. Use case: Send private message**

| | |
|---|---|
| Actor | The student and the teacher, in this use case we will refer to them as the sender |
| Description | The sender wants to send a message to the receiver. |
| Input state | Sender is authenticated. |
| Scenario | 1. The user opens the menu and selects the Chats.<br>2. The student selects the recipient from the chat list.<br>3. On the private chat page, it writes the text of the message in the textfield at the bottom.<br>4. He clicks the submit button. |
| Output state | The message is sent to the recipient |

**13. Use case: Check class**

| | |
|---|---|
| Actor | Student |
| Description | As a student I want to check new events in the class. Whether there are new mind maps or new announcements from the teacher. |
| Input state | Student is authenticated. |
| Scenario | 1. The user opens the menu and selects the Classes. 2. The student selects a class from the list of his classes. |
| Output state | The student sees the class page together with all new information. |

**14. Use case: Do test**

| | |
|---|---|
| Actor | Student |
| Description | As a student I want to do the exam. |
| Input state | Student is authenticated. |
| Scenario | 1. The user opens the menu and selects the Classes. 2. The student selects a class from the list of his classes. 3. The student selects an exam. 4. The student fill all empty textfields. 5. The student click on button Done. |
| Output state | The student sees the result on class page. |

For simplicity we have the use cases and their use by individual actors shown in the picture. 2.2

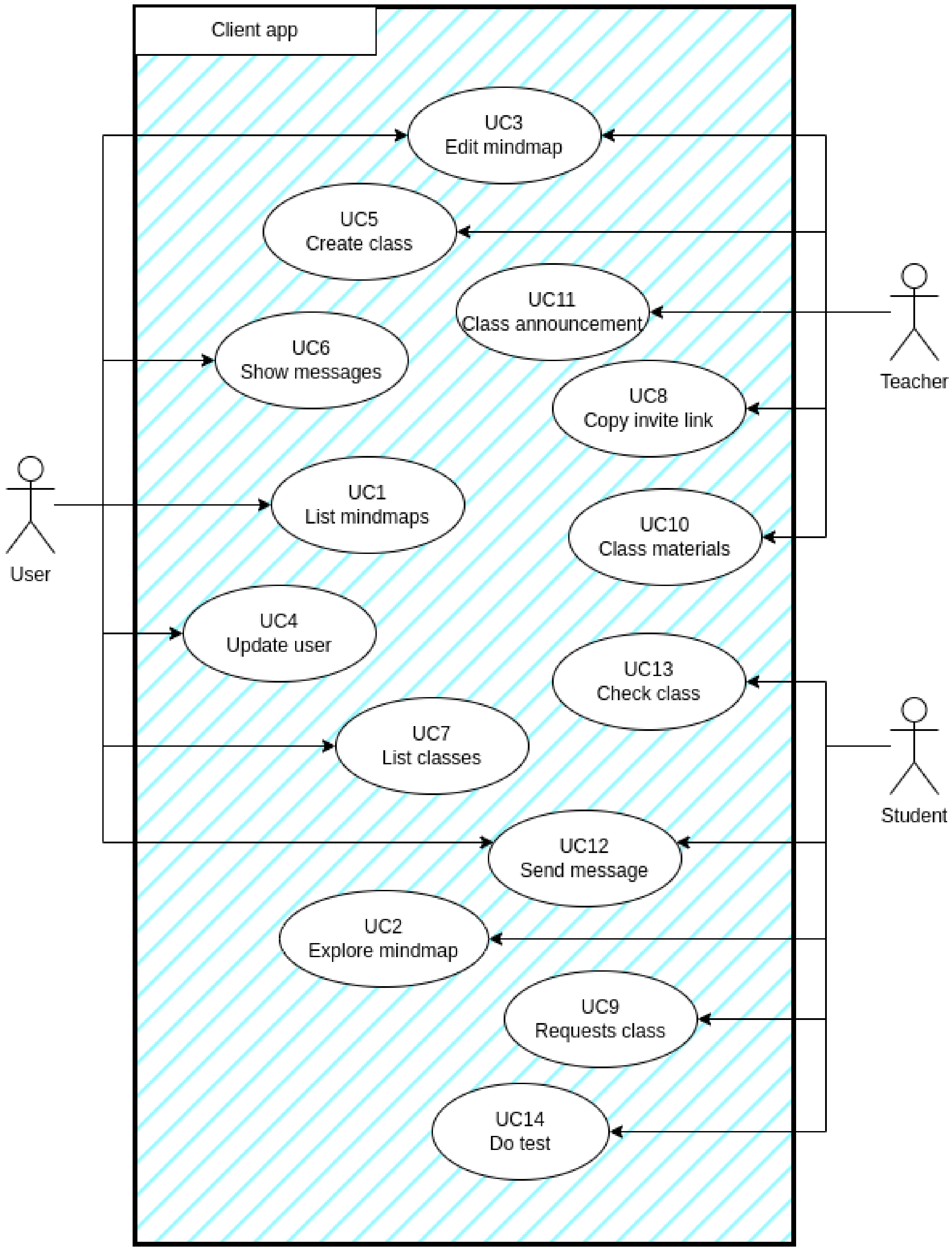


Figure 2.2: Use cases of our application with all actors

## 2.5 Knowledge bases

An essential part of our work will be to select a knowledge graph that will become the basis for creating mind maps. Therefore, we will introduce the adepts we will consider.

### 2.5.1 Wikidata

Wikidata, a project by the Wikimedia Foundation, is a freely accessible database of structured data[30]. It houses a vast collection of structured information that can be easily processed and linked to various other data sources. By adopting a uniform format for describing different types of entities, Wikidata simplifies machine processing and facilitates comparisons.
One of the key features of Wikidata is its query interface, which allows users to search for and retrieve information from the database using SPARQL. SPARQL, a data query language, enables advanced searching and linking of data from diverse sources. This empowers users to conduct complex queries and establish connections between different sets of data.
Wikidata has its ontology, where each entity and property is a wikidata entity(see fig. 2.3).

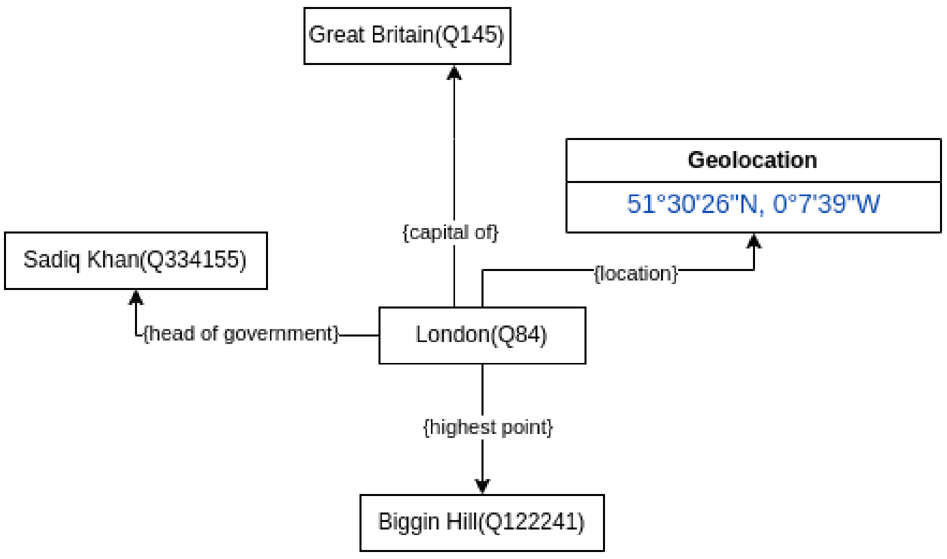


Figure 2.3: Wikidata onotology

### 2.5.2 DBpedia

DBpedia is a project that obtains structured data from the information in the English version of Wikipedia and represents it as a semantic web. The project was launched in 2007 and has since grown to include many language versions of Wikipedia and other similar data sources.
DBpedia processes Wikipedia content and extracts structured data from it, such as names of personalities, geographic coordinates, information about sports clubs, works of art, and much more. The data is stored in RDF format, which enables easy integration with other data on the Semantic Web.

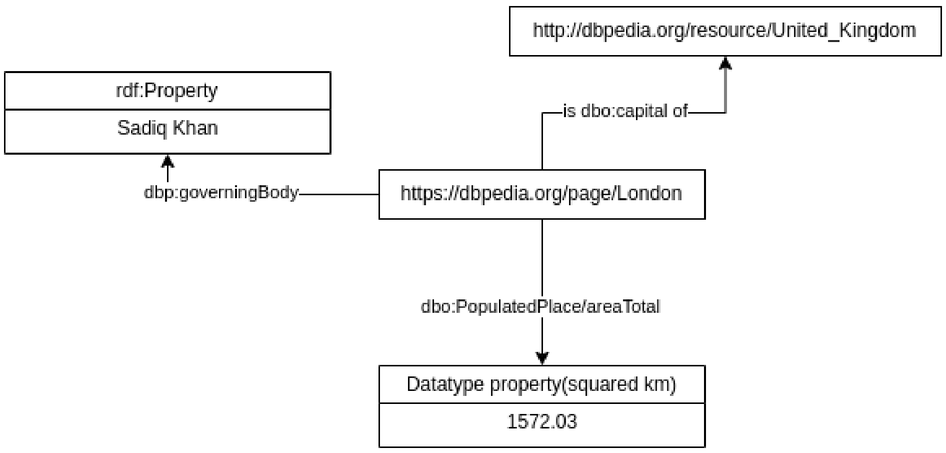

http://dbpedia.org/resource/United_Kingdom
rdf:Property
Sadiq Khan
is dbo:capital of
dbp:governingBody
https://dbpedia.org/page/London
dbo:PopulatedPlace/areaTotal
Datatype property(squared km)
1572.03

Figure 2.4: DBPedia ontology for London

DBpedia provides an interface for querying data using the SPARQL language, enabling easy search and retrieval of information. Data from DBpedia is also available in an open format for further processing and use.
DBpedia is an important data source for machine processing and artificial intelligence. It is used, for example, for creating chatbots, text classification, machine translation, sentiment analysis, and many other applications.
DBpedia uses both known dictionaries and its dictionary for its ontology(see fig. 2.4).

### 2.5.3 Yago

YAGO (Yet Another Great Ontology) is an extensive semantic database that describes the world using entities and their relationships. YAGO was founded in 2006 at the University of Jena in Germany and has since become one of the most important sources of structured data for machine processing and artificial intelligence.

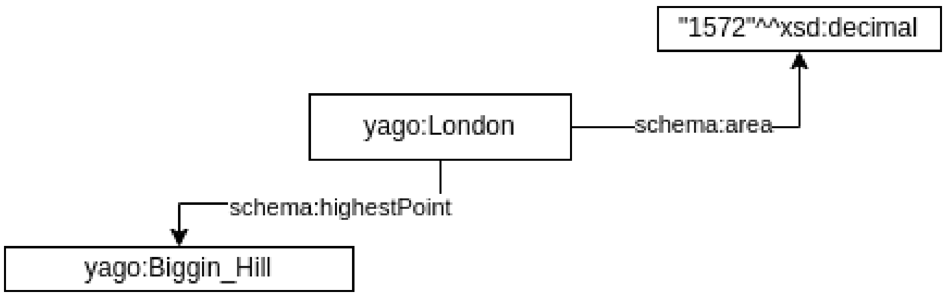


Figure 2.5: YAGO ontology

YAGO is built based on an ontology that describes relationships between entities and enables machine semantic data processing. The YAGO ontology was created by hand, which ensures high data quality and accuracy. YAGO also includes tools for searching and retrieving information from the database using the SPARQL language.
Of course, those above are not the only knowledge databases. They are the most widely used and widespread. In addition to them, there are also the following.

Geonames and FactGrid are worth mentioning. GeoNames for geographic data and FactGrid for historical and primarily German historical entities due to their origin at the University of Gottingen. Therefore, we will not consider these databases in the following analysis because we want to find a general one.
Like DBPedia, YAGO uses both known dictionaries and its own dictionary for its ontology(see fig. 2.5).

#### Geonames

Geonames is a geographic database that contains information about geographic locations from around the world. It provides data on geographic coordinates, place names, administrative divisions, geographic features, and other aspects. Geonames are often used to retrieve information about cities, villages, mountains, and rivers. The data is in a structured form and accessible through an API.

#### FactGrid

FactGrid is a knowledge database that focuses on historical and cultural data. It is developed based on the Digital Humanities project and stores and shares facts, information, and relationships between different entities. FactGrid focuses on historical persons, institutions, events, and other related data. The data in FactGrid is structured and linked, allowing researchers and historians to examine and analyze historical and cultural contexts.

### 2.5.4 Conclusion

For the final decision, we will present the criteria we require from the knowledge database.

1. **Extent of Data** For general use, we require that the knowledge database contains large amounts of data related to the most diverse areas.
2. **Wikipedia Link** Although we want to use the knowledge database mainly because of the relationships between entities, a very important aspect is a good description of individual entities. If the description is not directly linked to the entity, we would like each entity to have at least a link to some encyclopedia from which we can obtain a human-readable description.
3. **Language support** Last but not least, it is also important that the database does not become dead and does not provide new or un-updated information that does not correspond to reality after a specific moment.
4. **Schemas and SPARQL** An essential aspect in order to be able to use knowledge bases effectively is their schema and the way to obtain the information we want.
5. **Data Updates** DBpedia regularly updates its data from Wikipedia, while the community actively maintains Wikidata, and updates are more frequent.

6. **Categories** We want a database in which there are entities on one level and a superordinate category relationship. I can then get the other entities belonging to the selected category.
7. **Human readability** As the last feature, the human comprehensibility of the date we obtain from the database is very important

Each knowledge database we have presented is clearly evaluated in a table below.

| | Yago | WikiData | DBpedia |
|---|---|---|---|
| Extent of Data | yes | yes | yes |
| Wikipedia Link | yes | yes | yes |
| Language support | no | yes | yes |
| Schemas and SPARQL | yes | yes | yes |
| Data Updates | yes | yes | yes |
| Categories | no | no | yes |
| Human readability | no | no | yes |

Table 2.1: Knowledge databases comparison

We can see in the table(see 2.1) that all databases are very similar. According to the request for language support, we will only remove YAGO from the decision if we think. However, we have to evaluate the differences between Wikidata and DBPedia carefully.

### The main differences

DBpedia supports many languages because it is derived from different language versions of Wikipedia. Wikidata is designed to be multilingual and allows users to add and edit data in different languages.
Compared to DBpedia, Wikidata offers a broader possibility of categorizing the most diverse entities since the ontology is based on Wikibase, but this can be counterproductive for presenting such data to a person.
DBpedia is a database created by processing and structuring human data from Wikipedia. Thanks to this close connection with Wikipedia, DBpedia provides more natural and intuitive information for humans to understand.
That is why we chose the DBpedia database.

## 2.6 Conceptual model

Since our application is to be implemented on top of the SOLID platform, whose main idea is to decentralize user data storage, the main problem we will solve is the design of the user data storage architecture.
Based on the analysis of user stories, to be able to meet all the requirements, we will imagine the following data entities that will help us to model the problem

of creating teaching materials and also the conceptual representation of the class and their teacher and students, together with private chats between them.
Below is a list of all entities:

1. **Profile** - This entity represents information about the user. In app, we will use this entity to identify students and teachers by their names.
2. **Chat** - It will represent the entity of a private conversation between two users.
3. **Mind maps** - The mind map entity helps to model the mind map and to imagine how to represent the teaching material in the form of a mind map..
4. **Class** - It represents the entity of the class, which captures information about the class, their students, teachers and also includes the tests. It uses the Profile entity to get more information about the students and also uses the mind map entity to create class teaching materials.
5. **Request** - And the last entity represents the exchange of requirements between users, i.e. teachers and students.

We have displayed these entities in the Figure 2.6.

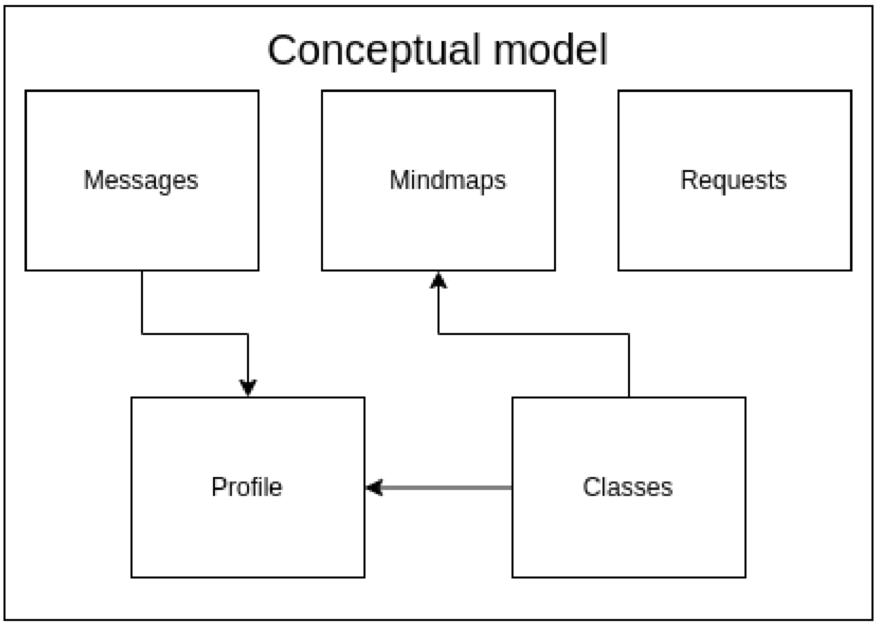


Figure 2.6: Conceptual model

# 3. Storage architecture

When designing the application, we encountered a fundamental problem. The main task of this paper is to present a proof-of-concept application for creating mind maps. In addition to creating mind maps, the application also provides a platform for sharing these materials, creating classes between teachers and students and communication between users, thus it serves as a social network .
The social network is an online service that allows the creation of a user profile upon registration, under which the service can be used mainly for communication, sharing information and multimedia files. Classic social networking applications such as Facebook, Instagram and others store user data on their storage.
In our case, the user does not create a profile on our storage but is authenticated within the application using his WebId linked to his Solid pod. On this Solid platform, a user has all his data stored on his Solid pod and has complete control over it.
The application does not have a central server but uses the SOLID platform. This is still not a standard situation from the point of view of today's applications. How to store such data? How to aggregate the data so that the user does not know the difference between a classic centralized service and a solution on a decentralized platform? How will the data users want to share with other users be distributed?
This chapter therefore focuses on the design of distributed user data storage on a decentralized storage platform. First, we analyze the possibilities of storing data not only on decentralized storage. And we will propose a decentralized data storage design for our application built on SOLID platform.

## 3.1 Data storage architectures

Although there have been proposals and practical solutions for decentralized data storage for some time, decentralized data storage has become more widely discussed, especially after Facebook's user data privacy problems. The latter even sold user data, and its use could significantly influence citizens in elections[17]. Even the original idea of the internet was to decentralize it, and perhaps that is why its 'father,' Tim Berners Lee, is trying again with a new platform. Therefore, in the following lines, we will introduce centralized and decentralized also called peer-to-peer solutions and look at the difference between the two. To clarify, when we talk about decentralization here, we mean it in terms of a storage owner and one central user entry point. Although Facebook user data is distributed worldwide due to its high availability and speed of delivery, Facebook still owns it.

### 3.1.1 Central storage

Centralized also called server-client data storage is a concept in which all data is stored in one central location, usually on a server or in the cloud, and users can access this data over a network(see Fig.3.7).
The first centralized data storage was probably developed for large computer

systems used for scientific and military purposes in the 1950s and 1960s. These systems allowed data to be stored and shared between users and processors.
However, their huge boom came in the late 1990s with the development of Internet services. Companies such as Amazon, Google, and Microsoft began offering cloud storage services that allowed users to store, manage and share their data on remote servers.
Today, centralized data storage is commonly used in various fields and industries. Businesses and organizations often use cloud services to store and share documents, files, and employee data. This model allows employees to access important information anywhere and from any device, promoting mobility and collaboration.

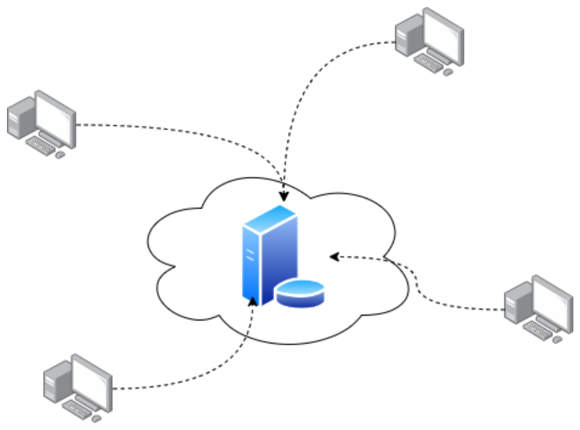


Figure 3.1: Central storage

### Advantages

1. **Control and management**
   The central server enables better control and management of data and services. Administrators have an overview of the system and can perform updates, backups, and monitoring in one place. A centralized approach also makes it easier to implement security measures.

2. **Security**
   A centralized server enables the introduction of comprehensive security measures and data access control. A central point of security makes it easy to protect against threats and monitor and implement security policies.

### Disadvantages

1. **Availability**
   The client-server model faces the problem of server availability, where the server must be online and accessible. Many factors can affect server availability, leading to the need for dedicated departments and industries (CDNs) to ensure high availability, which increases complexity and cost.

2. **High load**
   A common problem of client-server applications is high server load or unex-

pected demand. Anticipating and solving this problem is difficult and expensive. In the centralized model, the server must be able to handle client requests anytime. As the application grows in popularity, the number of clients requesting access to the server increases. Planning for unexpected demand is a big challenge because a single powerful client can monopolize server resources and disrupt other clients. Limiting client consumption levels can prevent disruption but leads to inefficient resource allocation where the server serves clients in a limited manner even when not overloaded. In an enterprise environment, high workloads often involve allocating more resources to servers, storage, and infrastructure. However, these additional resources are only needed during peak demand, resulting in wasted resources during off-peak hours. Planning for increased load often requires significant capital expenditures that may shift the bottleneck to another system component.

### 3.1.2 Decentralized

Peer-to-peer (P2P) data storage is an alternative concept to the centralized model, where data is not stored on one central server but spread among different computers, called peers, connected in a network(see Fig.3.2).
The history of peer-to-peer storage goes back to the early days of the Internet when the first peer-to-peer file-sharing networks began to develop. The technology was trendy in the 1990s and early 2000s when P2P protocols such as Napster, Kazaa, and BitTorrent allowed users to share music, movies, and other files directly without needing a central server. Now we are using decentralized concepts across emerging technology sectors such as blockchain, bitcoin, and other cryptocurrencies.
Today, P2P storage is used for file sharing, data storage, and backup. Various P2P cloud services allow users to store their data on different devices on the network[19].

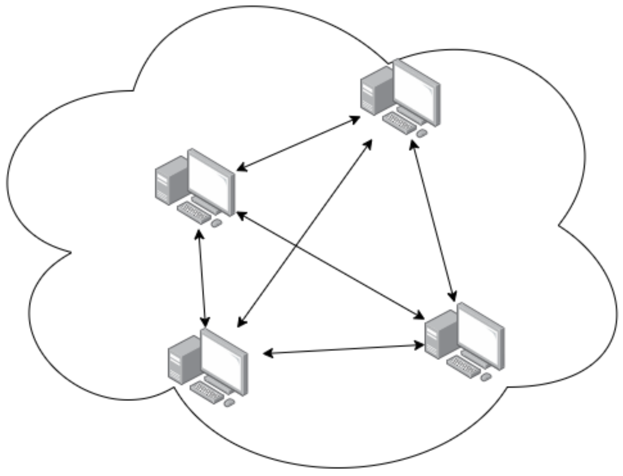


Figure 3.2: Peer-to-peer storage

**Advantages**

1. **Availability** By spreading data among peers, peer-to-peer storage is more resilient to outages and failures than centralized storage. If one peer fails, the others can still provide access to the data.

2. **High load** Peer-to-peer architectures turn each node into a server that can provide additional services. It has the property that each new user comes with additional capacity, which helps solve high-load problems organically. The problem of a robust client consuming all resources in the client-server model is an advantage of the peer-to-peer model, where that peer acts as a supernode and can service other peers at higher levels than the average node.

**Disadvantages**

1. **Data quality and consistency** In P2P networks, data is often spread between different nodes. This can lead to data consistency and quality issues. Inconsistent data quality and the inability to control and manage data accuracy can sometimes be a challenge.

## 3.2 Usage of peer-to-peer technologies

### File Sharing

Peer-to-peer networks allow users to share files directly with each other without the need for a central server. This means that any user who has a file can offer it to others, and they can download it directly. This approach facilitates fast and seamless file sharing between users.
BitTorrent, a peer-to-peer protocol that enables fast and efficient sharing of large files, is very popular. Each user who downloads a file also contributes to sharing it with other users. This we can see in Figure 3.3. This ensures that the file is more quickly available to everyone as it is offloaded to different users on the network.

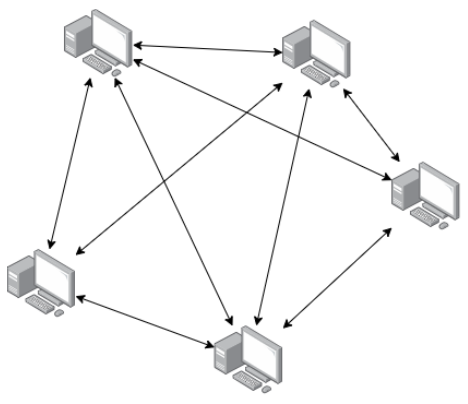


Figure 3.3: Peer-to-peer file sharing

### Cryptocurrencies

The blockchain technology underlying cryptocurrencies uses a P2P network to verify and transfer transactions. Every user on the network has a copy of the complete blockchain and works together to verify transactions and create new blocks(see Figure 3.4).

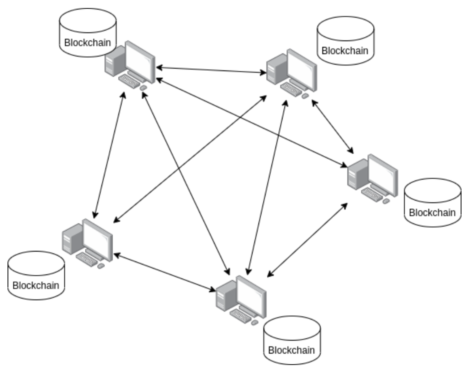


Figure 3.4: Blockchain network

### Internet of Things (IoT)

In the Internet of Things (IoT), devices can communicate with each other over P2P networks, and this enables data sharing and collaboration between different devices without the need for a central control point.

### Live streaming

P2P networks can be used to distribute streaming content such as videos and music. Users who stream content can simultaneously provide that content to other users on the network, reducing the load on central servers as we can see in Figure 3.5.

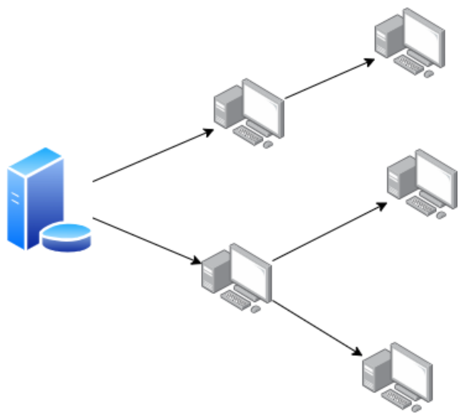


Figure 3.5: Peer-to-peer live streaming

## 3.3 Decentralized storage technologies

Although we want to take advantage of SOLID in this work, there are other technologies we can use for decentralized data storage.

### 3.3.1 Mastodon

Just as Facebook has had credibility problems, the same distrust has emerged with the centralized social sharing site Twitter. At the time, there was also much public talk about an alternative in the form of Mastodon. Mastodon is an open and decentralized social network and was created in 2016 as an alternative to traditional centralized social networks like Twitter or Facebook. The platform was inspired by decentralization, which means that data is not stored on a single central server but is distributed among many independent servers called instances. Due to its decentralized nature, Mastodon gives users more control over their data and more excellent resistance to censorship and unfair moderation. Each instance can have its rules and moderators, creating a diverse and varied user environment[12].

### 3.3.2 Diaspora

Diaspora is an open-source and decentralized social network that was first launched in 2010. It was developed as an alternative to centralized social networks, such as Facebook and Twitter, to give users more control over their data and privacy. Its solution is most similar to Solid.
The main idea behind Diaspora is that users can create their own ”pod” (similar to an ”instance” in Mastodon), which is a small standalone server where the user’s data is stored. These ”pods” are networked, allowing users to monitor and communicate with users from other pods.
Diaspora allows users to share posts, photos, videos, and other content with other users. Each user has control over who can see their posts and data. This gives users more privacy and security compared to centralized social networks, where one company collects and manages data.
Diaspora was created with an emphasis on open source and transparency. Anyone can contribute to the development and improvement of this platform, allowing the community of users to customize their own according to their needs[4].

### 3.3.3 IPFS

The InterPlanetary File System (IPFS) is an open-source, peer-to-peer protocol and network system designed for data distribution and storage in a decentralized environment. It was developed to replace traditional centralized web infrastructures, enabling fast, secure, and efficient data sharing between users.
The basic idea behind IPFS is to replace traditional Uniform Resource Locator (URL) addressing with unique hashes created based on the contents of the stored file. Data is uniquely identified based on content, not location on any particular server. This allows for easy dissemination of data between different nodes and provides resilience against outages and censorship.

IPFS users can store and share files, web pages, or content. Once a file is uploaded to IPFS, it acquires a unique hash and becomes part of the distributed networks. When someone requests access to this file using its hash, IPFS automatically searches for it and downloads it from the nearest available nodes[16].

### 3.3.4 Comparison of decentralized platforms

Mastodon offers accessible communication between users of different instances, allowing the creation of diverse and varied communities. Open source and decentralization make it transparent and censorship-resistant.
Diaspora is also a social network where users have complete control over their data and privacy thanks to their own "pod." Transparency and open source allow for community contribution and improvement of the platform. Compared to Mastodon, it can be more complex to run and is less widespread.
IPFS is a network protocol and data distribution and storage system. Its main idea is to replace traditional URL addressing with hashes, allowing data to be easily distributed between nodes and providing resilience against outages and censorship.
These platforms bring a revolutionary and innovative approach to data storage and sharing on the Internet. Introducing them as an alternative to Solid in certain aspects is undoubtedly interesting. Each of these platforms has its unique characteristics.

## 3.4 Data storage decentralization

The last chapter introduced the entities we will use in our solution. Since Solid is a decentralized storage system, it is necessary to clarify where the data entities of the application users will be stored. Decentralization of data storage does not only make sense for our reason because we want to decentralize user data aggregation, but here we will discuss more possibilities why companies decentralize data storage on their storage.

- **Availability**
  Decentralization increases the resilience of the system to outages. If one storage site fails, the others can continue to provide service. This can be especially important for mission-critical systems requiring continuous data access.

- **Scalability**
  Decentralization makes it easier to scale storage. Adding additional nodes or servers to the network can increase system capacity and performance. This is advantageous when data volume or system load grows.

- **Security**
  Decentralization can increase storage security. Attackers have a more difficult task because they must attack multiple sites simultaneously to gain data access. If data is scattered and encrypted in different locations, it is more difficult to compromise or steal it.

- **Privacy**
  Decentralized storage can bring higher privacy level. With centralized storage, the operator may have access to sensitive information. With Decentralization, data is dispersed among different entities, reducing the risk of misuse or unauthorized access.

Large corporations, such as Facebook or Google, also address decentralization. To maintain their position, these companies have to provide high availability of their services to ensure the availability of services, scalability for a growing number of users, and security.
These requirements are addressed by distributing data between nodes or servers through techniques such as data replication, fragmentation (sharding), or distributed file systems. Data replication involves storing copies of data in different locations, which provides redundancy and reduces the risk of data loss in the event of failure. Fragmentation divides data into smaller pieces stored on different servers, minimizing the impact of a single server failure on the entire system. Distributed file systems then enable efficient distribution and management of data across different nodes in the network. In addition to data distribution, redundant infrastructure and data replication across physical locations can also be used. This ensures that there are additional copies of the data if an outage or failure occurs at one location.
In addition to these techniques, the microservices architectural approach is used to maintain the scalability of a large system. It means partitioning an application into smaller, separate services that can be developed, deployed, and scaled independently. Each microservice is specialized for specific functionality a simplified diagram of which can be seen in Figure 3.6. Each microservice communicates with other services using defined interfaces, such as an Application Programming Interface (API)[9].

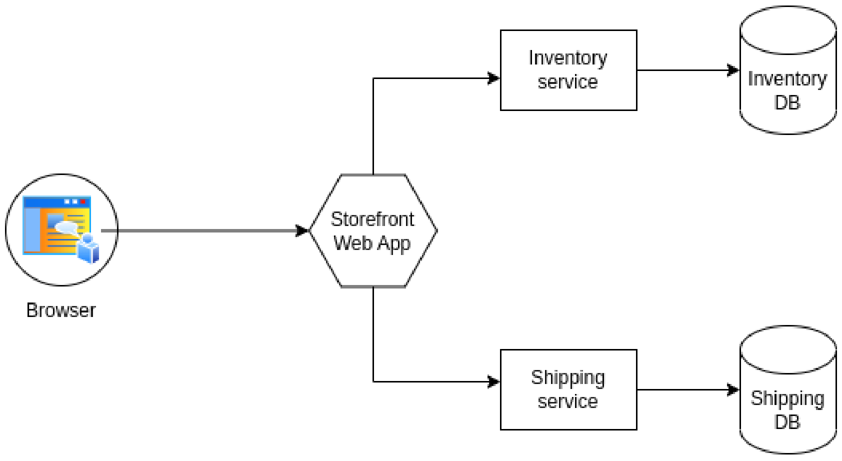


Figure 3.6: Microservices architecture

Nevertheless, what these companies do not take into account and do not adapt their storage solution to this is the privacy of user data. The user does not know where his data is stored, who has the right to access it, or whether it has become the target of some data leak in hackers' attacks on the company's infrastructure. For this reason, we will want to design the data distribution architecture of our application on a platform for which user data privacy is paramount. We will then

use this design as a proof-of-concept for our application.
In order to take into account all the possibilities, we will first present a centralized design in addition to our decentralized design. Then we will analyze two designs in more detail: the cluster architecture and the fully-decentralized architecture.

### 3.4.1 Centralized Architecture

In a central architecture, data is stored and managed on a single central server. All users access the data through this server. In the case of our application and the requirement that we want to use SOLID, all the node data would be hosted on one pod as we can see in Figure 3.7, and this pod would become the central component on which the whole application and its users depend.
This architecture would be the easiest to implement. There is no need to address the issue of access to resources. A server controls all communication with user data, including user data. Although this is a proven model that most of the applications we use currently run on, the fact that this server controls everything, including our data, is why we want to avoid this model.

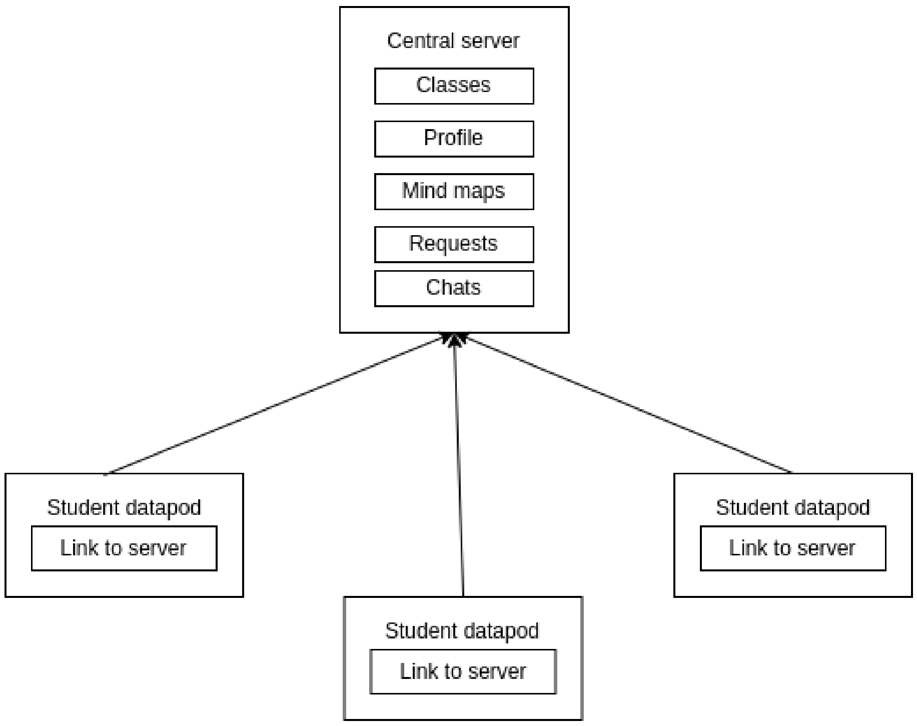


Figure 3.7: Centralized architecture

### 3.4.2 Cluster architecture

In this proposal, we propose not to have one central repository but to have its users and their Solid Pods form clusters like real-world classes. This is already a small enough subset to be considered a peer-to-peer decentralized model.
In this model, the teacher has the primary responsibility. He has all the data from his classes on his Solid pod, as shown in Figure 3.8). The student only has a link to the class on his Solid pod, which then gets the data from the teacher's Solid pod. The teacher has all the test results in the class, links to the student profiles, the class mind maps, and the communication between him and the students on his Solid pod.

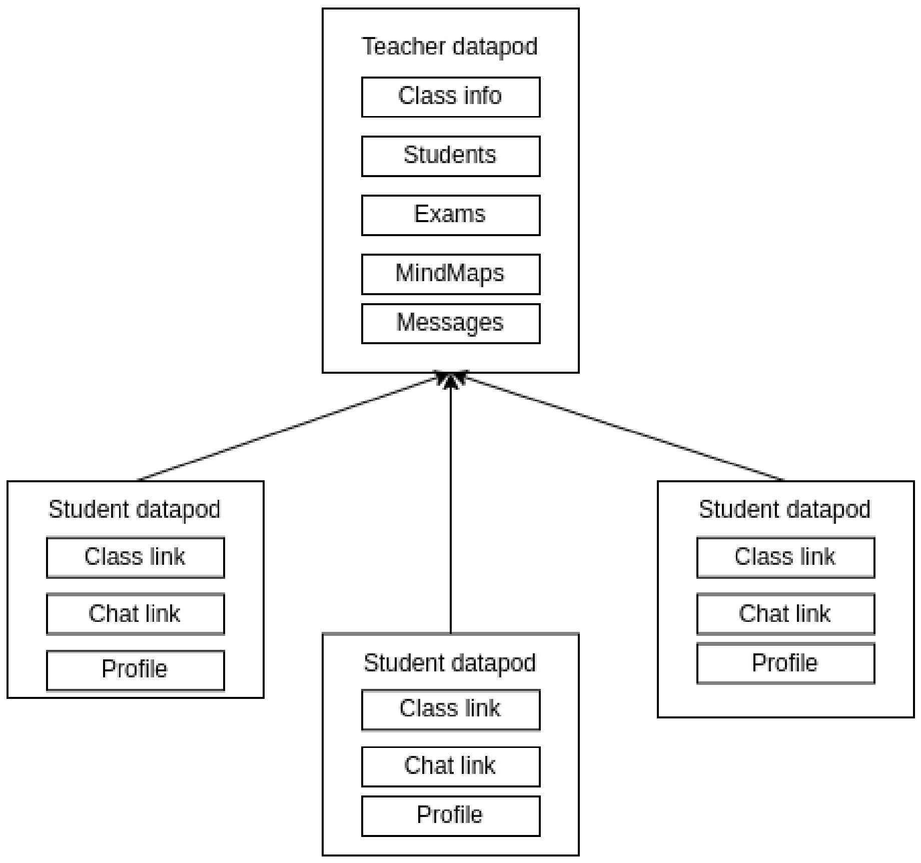


Figure 3.8: Cluster architecture

### 3.4.3 Fully-decentralized architecture

The last concept will be a fully decentralized model, where every social site user has the same rights. Data is distributed among all users as we can see in Figure 3.9. This would mean that, unlike the previous solution, in this solution, the student saves his test results on his Solid pod, as well as the messages that have been delivered to him. In order to read all the class data, we as a teacher have to read all the Solid files of all the students. The teacher in this model has only the mind maps of the class stored on his Solid pod.

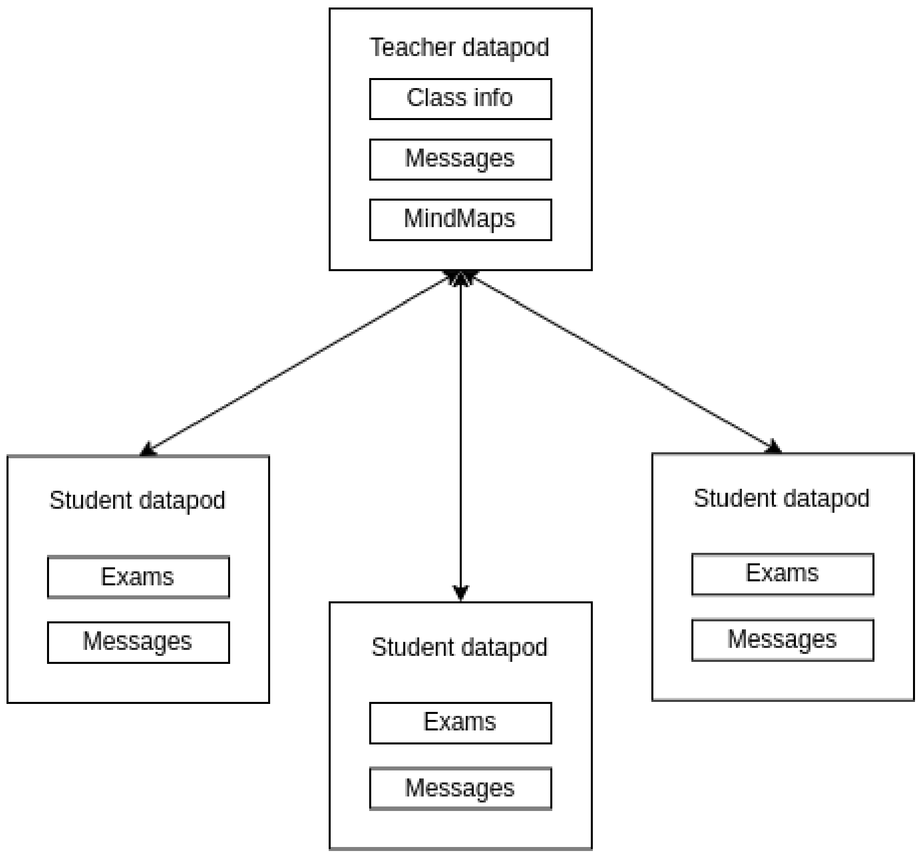


Figure 3.9: Fully-decentralized architecture

### 3.4.4 Comparison

After introducing the models, we will compare the two decentralized models. Because of the lack of privacy of our data, the centralized model may very soon be worth more than we can imagine now. Therefore, even if the model is implemented on top of a platform that allows decentralization, such a model would completely erase all the pros and, indeed, the reason for creating such a platform. When comparing Cluster and Fully-decentralized models, we will focus on evaluating the models in terms of availability, performance, and scalability. However, in our specific case, where we are designing a system for schools, most likely in primary schools, we will also examine the aspect of data reliability.

#### Performance

First, let us look at the designs from a performance perspective. Performance refers to the system's speed, efficiency, and capacity in processing requests and providing services. Higher performance means that the system can process a larger volume of data or requests in less time. Factors affecting system performance include the speed of data processing, the system's responsiveness to user actions, the performance limits of the hardware, and the efficiency of algorithms and programming. Optimizing hardware, software architecture, and algorithms, using high-performance databases, and performing code-level optimizations are often necessary to achieve high system performance. System performance is a key aspect that affects user satisfaction, response speed, and overall system efficiency.

In analyzing the performance perspective, we first briefly introduce the findings presented by Martin Bruland in his paper Evaluating Solid for Social Applications with Many Users[6]. In this work, he investigated the performance when reading and aggregating different sizes of Solid pools from different numbers of solid pods. We briefly summarize the results of this work for both reading and aggregating data:

- **Read** The retrieval time for the dataset with one item from a single POD was 0.08 seconds, while it took 1.09 minutes to retrieve the data from 1,000 PODs. When the number of PODs was further increased to 10,000, the retrieval time extended to 10.88 minutes. Interestingly, the results indicated that the size of the dataset had minimal (if any) impact on the retrieval time, primarily due to the local environment in which the experiment was conducted.

- **Aggregating** The experiment involved compiling the top 10 items from different datasets obtained directly from multiple PODs. The results demonstrated that retrieving the top 10 items from a dataset containing 1,000 items, which was retrieved from 1,000 users (totaling one million items or 7,001,000 triples), took approximately 30 seconds and contributed to the overall content loading time. Similarly, assembling data from a dataset of 100 items retrieved from 10,000 users, which had the same number of items, took 20 seconds to complete. The slight variation of 10 seconds in processing time for the same amount of work could be attributed to various external factors and variables.

Briefly, for reading data, these results could be summarized as the time to read all data increases significantly with the number of Solid Pods we read from, while the size of the data we read from individual Solid Pods has a negligible effect on the reading time.
Suppose we look at our concepts. Assuming a classic class of 20-30 students, like in elementary school, the difference between the cluster and the fully-decentralized model would be smaller. However, I have lectures for 300 students at universities; the difference would already be very noticeable in this case. So the cluster model is a clear choice in terms of performance.

### Availability

Before we relate the availability aspect to our two proposals, let us imagine what availability means. Availability is the degree to which a system is available and functional for users. It expresses the ability of the system to provide the required services and process requests without failures or unavailability. Higher availability means the system is stable, reliable, and has minimal downtime, contributing to a better user experience and minimizing interruptions. Availability is usually expressed as the percentage of time the system is fully functional and available. In system architecture, it is important to consider factors that affect availability, such as redundancy, fault tolerance, data backups, error management, and rapid recovery from system failure.
After defining the concept of availability, we look at our concepts from the perspective of this aspect. In the case of the Cluster model, the teacher would be

primarily responsible for ensuring data availability for the students. In the case of a fully-decentralized model, the students would also be responsible for data availability. This could be a problem in the real world for such a network. Students, especially in primary schools, who could be more tech-savvy might not ensure good data availability, and the teacher might not be able to access it. The cluster model would have the advantage in the real world that the teacher or the school would be responsible for the availability of the data, and in case of a problem, a solution would be provided.

#### Scalability

Scalability deals with the ability of a system, application, or service to handle increased load and growth in the number of users or volume of data. A scalable system can adapt to increased demands and maintain performance and availability under increasing load. There are two forms of scalability: vertical and horizontal. Vertical scalability involves adding compute or storage resources to a single node, while horizontal scalability involves adding additional nodes or servers.
From the point of view of scalability, the fully-decentralized model has the great advantage that in case of a large increase in users the load is evenly distributed among all users. The cluster model has a problem with handling all the requirements when there is a large increase in the number of users using one Solid pod. Such a moment comes as soon as the number of users exceeds 500.

#### Reliability

Unlike other types of social networks, we want to address the aspect of data reliability on the user's Solid Pods on our site. Even in real life, we can encounter a reversal of school results in the case of a student. A technologically savvy student could change the data on his Solid pod. If the teacher did not save the results of his students, he could then evaluate the student based on this fake data. Thus, it is also necessary to think about the entity of certain authorization of the results that students achieve in tests, and that can decide whether the results are correct.
In real life, such an entity is also the teacher, who is fully responsible for the correctness of the test results. In the cluster model, he would then take over this role. The student's results would be stored on its Solid pod, and the students could access the teacher's Solid pod only to read from it, keeping their results the same.

### 3.4.5 Conclusion

After analyzing both models from the perspective of several aspects, we summarize the results in this table, where we can easily see which model comes out better from the comparison.

|  | Cluster | Fully-decentralized |
|---|---|---|
| Availability | + | - |
| Performance | + | - |
| Scalability | - | + |
| Reliability | + | - |

Table 3.1: Comparison of solutions

According to this table, which summarizes our evaluation, the cluster is the model that best meets our requirements. In this model, the teacher has the main powers. So the teacher takes care of all the data of the class. He creates materials, manages them, has the results of the class stored and also has the private communication between him and the class stored.

## 3.5 Alternative usages

Above, we have analyzed one possible application of Solid technology. However, we would not like to stay only with it and will present other possible uses.

### 3.5.1 Another social network

Now, let us consider an alternative social network with different characteristics than ours if we had the task of implementing a social network that is more similar to the concept of today's social sites. A social network that includes thousands to millions of users who want to consume content not from a school class of 30 to 250 students, but from users worldwide.
In a nutshell, we could think of it as:

- We would again like to build this site on the Solid platform.
- The potential size of friends is in thousands, but we will stick to the average number of users, i.e., 500-1000.
- On the main page, we would receive aggregation from all users we have in friends.
- A user would provide his content from his Solid pod, which he would allow to be displayed by other users.

Of course, we will summarize the analysis of such a social site on fewer lines than in the case of our social site design.
First, we look at the performance aspect. A hierarchical variant would be preferable in the aggregation or performance when reading from the Solid friend's pool. As we have already learned above, the reading time increases exponentially as the number of users and their solid pools grows. With a massive number of friends, such aggregation is impossible from the point of view of user convenience. Waiting tens of seconds for even a simple aggregation could be better.

From the point of view of scalability, a fully-decentralized model would be preferable. If the network grows tremendously, and the data would always be stored on a cluster, this cluster would only be sufficient to handle data queries after a very short time. A fully-decentralized model would better handle such a growing workload.
Now let us look at availability. Unlike our social site, there would not be users yet to be of age or their parents who are not very good with technology. On this network, everyone would be responsible for their Solid pod. Otherwise, they would not be able to access this network, so it would be in the interest of every user to have their Solid pod with the best possible availability.
As we have evaluated the reliability aspect of our site, for the sake of the credibility of the signs, we can omit it from this social site. Then we could consider whether a Facebook profile should be linked to a citizen's ID card.

| | Cluster | Fully-decentralized |
|---|---|---|
| Availability | - | + |
| Performance | + | - |
| Scalability | - | + |

Table 3.2: Comparison of solutions for alternative social network

Therefore, we would opt for a fully decentralized solution for such a site. Zhodnoceni srovnani mame prehledne v tabulce 3.2 Fully decentralized offers a better option for a site with growth potential in terms of availability and scalability. In terms of performance, the problem would be primarily with data aggregation. However, as a rhetorical question, "Would not a network that did not constantly offer us ballast and paid content and only our friends' content in the feed be better than what we know today?"

### 3.5.2 Pros and cons

After we introduced Solid technology, we analyzed its use more closely, we probably have an idea of what this technology can do, but we also know its limits. We want to summarize these features here.
First of all, we will present the disadvantages:

- **Storage and scalability**
  Storing data in a decentralized way can sometimes cause problems related to data storage and scalability. Solid implementations may have problems handling large amounts of data or managing network traffic depending on the design and infrastructure.

- **Data Availability**
  Although Solid emphasizes user control over data, it is important to ensure that users do not lose access to data if a particular Pod or service goes down. Ensuring the portability and availability of data can be crucial for a long time.

- **Integration**
  Integrating Solid with existing systems and applications can require significant customization and adaptation. Compatibility issues may arise when attempting to migrate data or implement Solid in companies with existing systems.

And the benefits:

- **Date control** Users have complete control over their data in their Pods. They can determine who has access to their data and what data they want to share with other users or applications. This provides a higher level of privacy and security.

- **Interoperability**
  Data stored in Resource Description Framework (RDF) triples in Triplestore format and is easily linkable and interoperable with other applications and systems that support the RDF format. This allows easy data exchange between different platforms. Solid take enables easy data sharing and collaboration between users. Users can share and collaborate on files directly from their Pods, facilitating efficient collaboration.

- **Security**
  Solid emphasizes security and data encryption. Data stored in the Pod is encrypted and transmitted over secure channels, reducing the risk of data leakage and misuse.

After this summary of the main features of Solid technology, we could consider its use in different types of situations. Let us now imagine these situations and how our decision-making would turn out if we wanted to implement such a solution.

### 3.5.3 Personal storage

An exciting way to use it is as personal storage. For developers and technology enthusiasts, this technology is an interesting alternative to standard services like Google Drive, DropBox, or iCloud.
These services offer closed ecosystems, and users may be limited to exchanging data between different platforms. In addition, we have yet to learn who has access to the data and whether there is a copy elsewhere, so we should not store very sensitive data on such platforms.
These services' performance and availability are comparable to Solid, so it depends on everyone's preference. Currently, what speaks for classic iCloud storage is that this data can then be used on applications with which the platform works and on mobile devices; Solid has yet to offer such an ecosystem. For Solid, however, it speaks of complete privacy.

### 3.5.4 RDF storage

Since Solid is built on Linked Data principles and the data itself is stored in RDF format on Solid pods, it is suggested to use Solid as an RDF storage for more significant amounts of data. However, here we have to introduce another

interesting name in the world of Linked Data. Moreover, that is Triplestore, a special kind of database used for storing and managing data in RDF format. Triplestore has the advantage of being able to query over graph data efficiently and allows for the storage of linked data structures.
If we decide between these technologies to store a larger volume of RDF data and query it, Triplestore is the obvious choice. In terms of performance, Triplestore is orders of magnitude different[21].

### 3.5.5 Identity Management and Authentication

Apart from the above mentioned use cases, it is also possible that we will encounter Solid technology more frequently in verifying digital identity on watches. The EU authorities are looking into the possibility that Solid could one day provide a trusted data infrastructure for the European Union. Inrupt, Tim Berners Lee's company, has received significant investment, and his Solid Pods technology is starting to gain attention, even if it faces misunderstanding and apparent complexity[10].
Inrupt has found support in Belgium, where the government of Flanders has decided to use its platform to exchange data between citizens. However, interest in a trusted and secure data exchange platform spreads across Europe. Solid technology is also being flirted with in the UK. The Government Digital Service(GDS) is developing a new digital identification system to be the standard way of logging into the Gov. UK website. Called One Login, the system will require users to permit to access and share data held by various government departments. GDS has created a simple concept based on Solid technology to address potential data protection and privacy issues[18].

# 4. Design

After we have done the information gathering and analysis in the previous chapters, we will propose how our application will be implemented in this chapter. What components will be included? Then we will design a data model, introducing the RDF vocabulary, an essential part of the design for Linked data. Finally, we must represent flows in our application using a sequence diagram.

## 4.1 High-level architecture

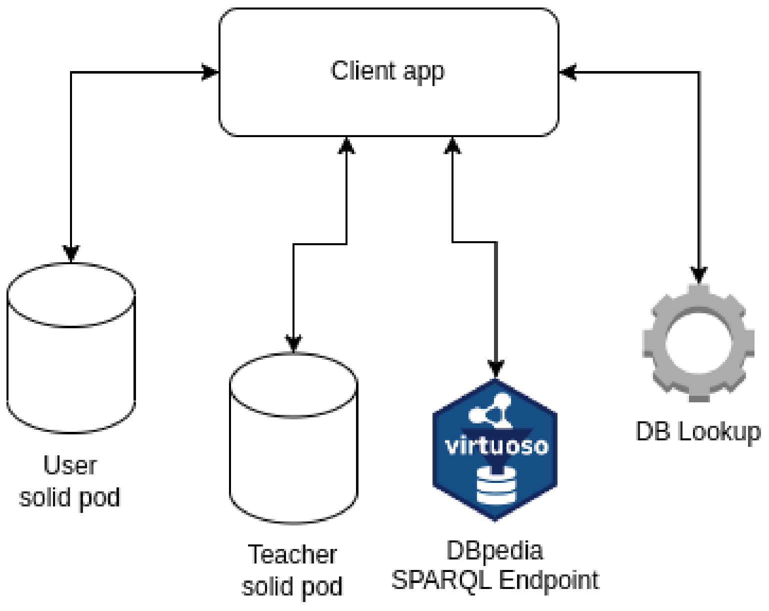


Figure 4.1: High-level architecture

Looking back at our analysis, we can see that the functional requirements set out well-defined boundaries for the application. Based on this reasoning, we can immediately define architecture. provided in this section shows more specific details on how all the external and internal components of the system are interacting(see Figure 4.1).

### Client application

Our web application provides a user interface for creating mind maps. One of the functions for creating and editing mind maps is recommending DBPedia entities based on keyword input. For this purpose, we use this service.

### Solid Pod

The solid pod was introduced in the first chapter.

### DBPedia SPARQL endpoint

When modifying a mind map, we require the user to be advised of a similar entity to the selected entity. The DBpedia SPARQL endpoint allows us to formulate and send SPARQL queries that retrieve specific information from the DBpedia database. The DBpedia SPARQL endpoint is a publicly accessible interface that allows us to execute SPARQL queries on the DBpedia database.

### DBPedia lookup

One of the functions of mind map creation and editing is recommending DBpedia entities based on keyword input. This service is used for this purpose. DBpedia Lookup is a service the DBpedia project provides that allows one to search and retrieve information about entities from the DBpedia database[8].

## 4.2 RDF vocabulary

Before moving on to the model design, we must introduce RDF Vocabularies.
Vocabularies are set of definitions of terms and relationships used within RDF graphs. Vocabularies typically specify basic concepts that apply to a particular domain or area of knowledge. Using these vocabularies, we can express structured information and link data from different sources on the Web.
When we want to represent a specific domain, objects in the domain, and their relationships, it is common to use already defined Vocabularies, called well-known vocabularies, for such a description.
These have the following main advantages:

- **Standardization** Well-known vocabularies are based on open and standardized specifications, ensuring that different systems and tools use the same concepts and relationships. This reduces the risk of inconsistencies and incoherence in the data.

- **Systems interoperability** Using well-known vocabularies allows different systems and applications to communicate and collaborate more easily. This creates an ecosystem in which data can move freely between different sources and applications

- **SEO** Some well-known vocabularies, such as Schema.org, are used by search engines and other tools to improve the interpretation of web content. This can increase the visibility of the site in search results.

Therefore, when describing our model, we will describe as many Trids and properties as possible using well-known vocabulary.
In the list, we will show the vocabularies we have used with their abbreviations, which we can then see in Figure 4.2:

1. https://schema.org - scheme
2. https://d-nb.info/standards/elementset/gnd - d-nb
3. https://dbpedia.org/ontology - dbpedia
4. http://purl.org/dc/terms - purl
5. http://www.w3.org/ns/dx/prof - w3

6. http://xmlns.com/foaf/0.1 - foaf
7. http://data.ign.fr/def/geometrie/20190212.htm - ign
8. http://vocab.gtfs.org/gtfs.ttl - gtfs

However, since our mind map domain and the social network around teachers and students are rare, we had to define our classes and properties. Since such a vocabulary needs highly stable URLs for proper functioning, that is why https://w3id.org/ exists. Its website provides a secure, permanent URL re-direction service for Web applications. The W3C Permanent Identifier Community Group runs this service.
We have chosen WikiMind as the name of our dictionary. The name have something to do with Wikipedia and Mind Maps. Moreover, that is exactly our case. Our dictionary is then redirected from address https://w3id.org/wikimind to its actual location https://raw.githubusercontent.com/matejikj/wikimind/main/wikimind.ttl.

## 4.3 Logical model

As we have introduced the conceptual model in the previous chapter(2.6), we will create a logical model(see Figure 4.2) on its basis, how we would like to work with individual entities in our application.
As we said in the previous chapter, we will divide the data into five entities: Profile, MindMap, Class, and Message. When creating the logical model, we will break down each of these entities and imagine the classes that we want to represent all the entity's properties.

### Profile

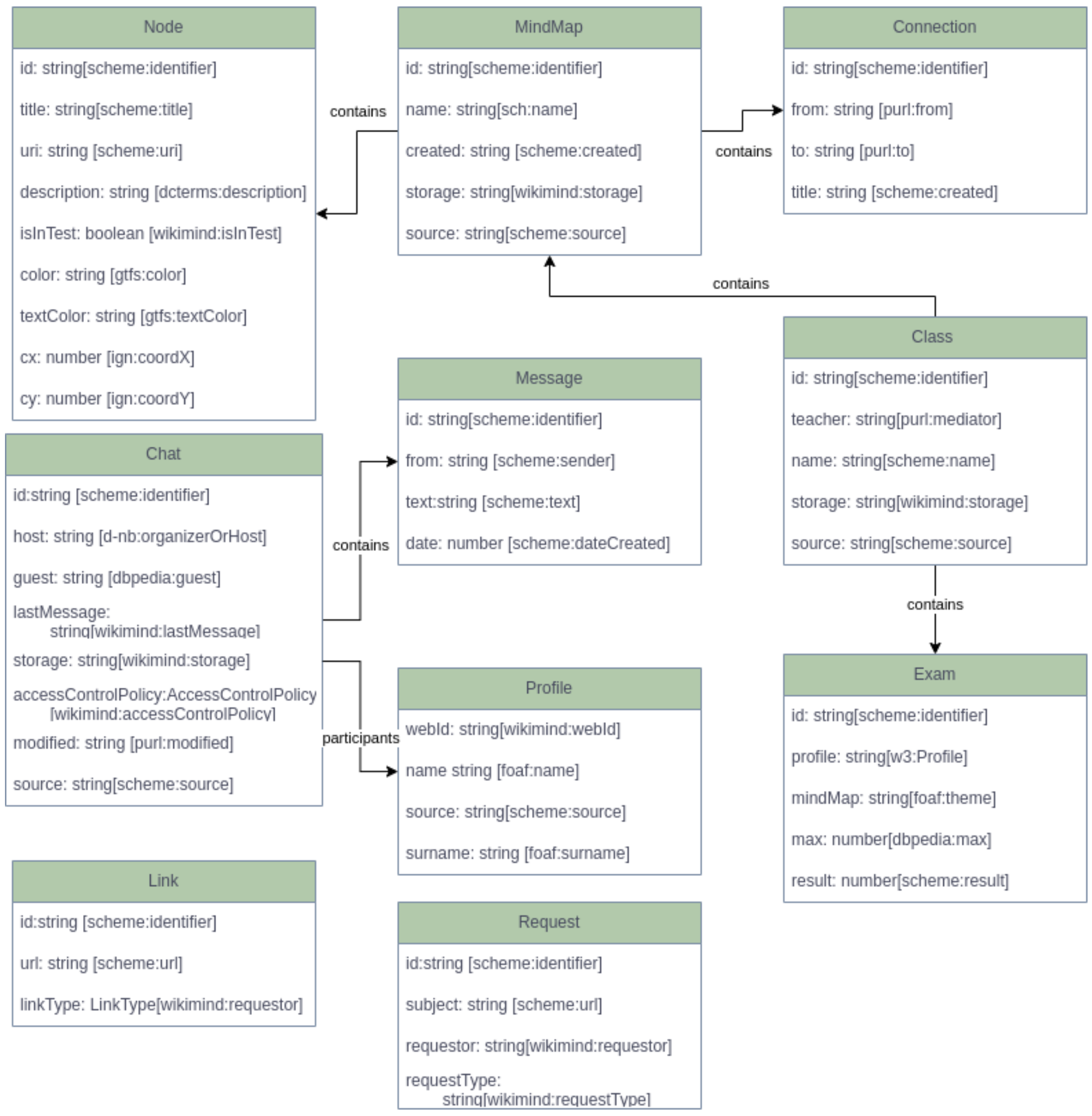
In the case of Profile, it is simple to start with. For the profile entity, we will need the **Profile** class, which will represent the user's entity in our application and will contain information about it (see fig. 4.2).

### Mind map

The Mind map is another entity for which we want to define individual classes. The mind map will be represented by the **MindMap** class to store information about the mind map itself. A mind map is supposed to contain uniform nodes and connections between them. So we need the **Node** class to represent an entity in the mind map that represents either a DBpedia entity or a user's custom entity. Another class is **Connection**, which carries information about the connection between nodes.

### Chat

Chat is an entity that represents the communication between the teacher and the student. We will represent a chat by the Chat class, which contains metadata about the conversation, such as the date of the last message or, but most importantly, information about the chat participants, their WebId, which we can use to get a profile of the users participating in the chat.



Node
id: string[scheme:identifier]
title: string[scheme:title]
uri: string [scheme:uri]
description: string [dcterms:description]
isInTest: boolean [wikimind:isInTest]
color: string [gtfs:color]
textColor: string [gtfs:textColor]
cx: number [ign:coordX]
cy: number [ign:coordY]
contains
MindMap
id: string[scheme:identifier]
name: string[sch:name]
created: string [scheme:created]
storage: string[wikimind:storage]
source: string[scheme:source]
contains
Connection
id: string[scheme:identifier]
from: string [purl:from]
to: string [purl:to]
title: string [scheme:created]
contains
Class
id: string[scheme:identifier]
teacher: string[purl:mediator]
name: string[scheme:name]
storage: string[wikimind:storage]
source: string[scheme:source]
Message
id: string[scheme:identifier]
from: string [scheme:sender]
text:string [scheme:text]
date: number [scheme:dateCreated]
Chat
id:string [scheme:identifier]
host: string [d-nb:organizerOrHost]
guest: string [dbpedia:guest]
lastMessage: string[wikimind:lastMessage]
storage: string[wikimind:storage]
accessControlPolicy:AccessControlPolicy [wikimind:accessControlPolicy]
modified: string [purl:modified]
source: string[scheme:source]
contains
participants
contains
Profile
webId: string[wikimind:webId]
name string [foaf:name]
source: string[scheme:source]
surname: string [foaf:surname]
Exam
id: string[scheme:identifier]
profile: string[w3:Profile]
mindMap: string[foaf:theme]
max: number[dbpedia:max]
result: number[scheme:result]
Link
id:string [scheme:identifier]
url: string [scheme:url]
linkType: LinkType[wikimind:requestor]
Request
id:string [scheme:identifier]
subject: string [scheme:url]
requestor: string[wikimind:requestor]
requestType: string[wikimind:requestType]

Figure 4.2: Logical model

## Class

The last entity we considered in the previous chapter is Class. we will introduce the **Class** class, which represents a class, containing information about its name, teacher, or last modified date. The members of the class are the students, and the mind maps are also used as class materials, both of these classes we have already created, so it remains to introduce the **Exam** class. This class represents the student's result in the test.

## Requests

Finally, we are left to represent the entity for managing requests and allowing access to classes. The **Request** model represents a class access request and contains information about the requester and the requested class.

#### Requests

Finally, we are left to represent the entity for managing requests and allowing access to classes. The **Request** model represents a class access request and contains information about the requester and request.

#### Link

There is also the Link entity, which we did not mention in the conceptual model because it is not a domain we would consider during the analysis. However, we must also define how we want to refer to other Linked data objects.

## 4.4 DBPedia queries

The focus of this work is not to provide a recommender model to an entity from the knowledge base but to choose one that will recommend suitable suggestions from a human perspective after analyzing knowledge bases. As we said, we will use the DBpedia knowledge base.
This knowledge base focuses on linking different pages and information from Wikipedia and allows users to find connections and links between different topics. DBPedia describes such relationships in its dictionary https://dbpedia.org/ontology.
We will show the real use of the example of a small village in the Zlin region of Bilovice. Its DBPedia page is here http://dbpedia.org/resource/Bílovice.
As a young student at primary school, I would like to do an essay on Bilovice and its surroundings. However, what to find? If I read Wikipedia, I can certainly find some data, but it would take me quite a bit of time to browse through all the entities mentioned on the Wikipedia page of this entity. If I had a handy DBPedia data viewer, I would want to know something about the area where this village is located. I can find some entities this through the **dbo:wikiPageWikiLink** relationship, which acts as a link between Wikipedia pages. Naturally, I can find Uherske Hradiste and Zlin as the nearest big cities. And to Zlín immediately Tomas Bata and to Uherske Hradiste the ethnographic area Slovacko. I would also like to explore the surrounding villages, what wikilinks they have. Furthermore, I will find the parent category for the surrounding villages through **is dcterms:subject of**. We can see all these relations in Figure 4.3.

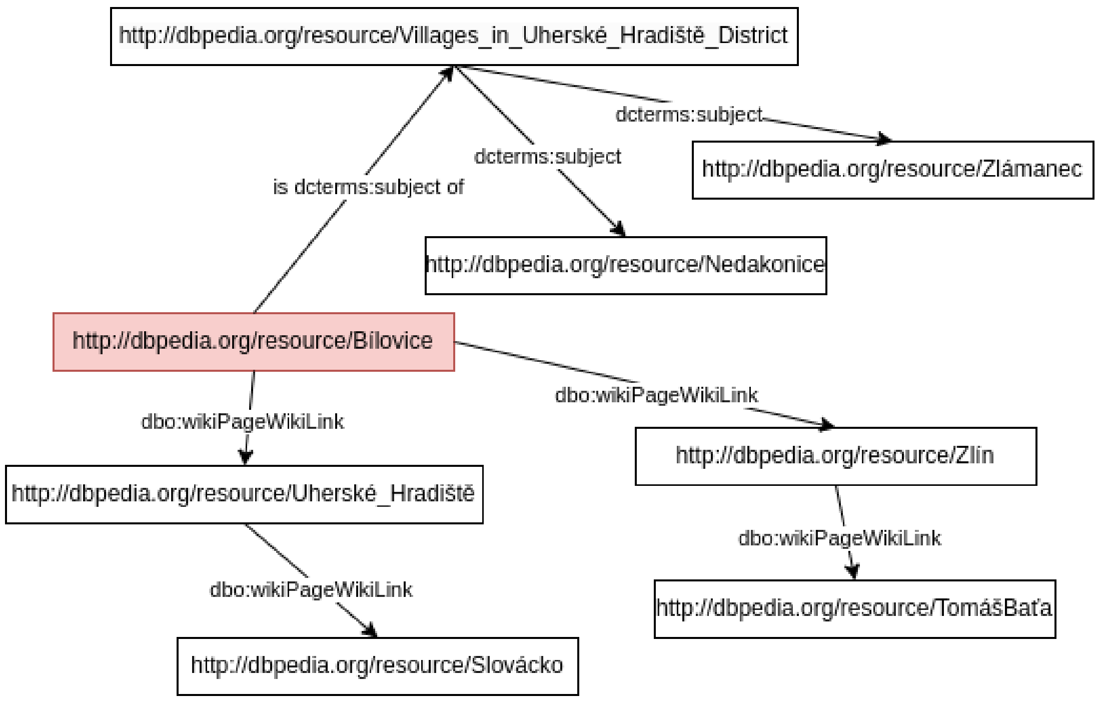

http://dbpedia.org/resource/Villages_in_Uherské_Hradiště_District
dcterms:subject
http://dbpedia.org/resource/Zlámanec
dcterms:subject
is dcterms:subject of
http://dbpedia.org/resource/Nedakonice
http://dbpedia.org/resource/Bílovice
dbo:wikiPageWikiLink
dbo:wikiPageWikiLink
http://dbpedia.org/resource/Zlín
http://dbpedia.org/resource/Uherské_Hradiště
dbo:wikiPageWikiLink
dbo:wikiPageWikiLink
http://dbpedia.org/resource/TomášBaťa
http://dbpedia.org/resource/Slovácko

Figure 4.3: DBPedia ontology

## 4.5 Operations

In this section, we will define the flow of operations that we will display as sequence diagrams. We will try to translate all use cases this way, but we will omit some because they are too similar to others. The flow of the mind map list is the same as the class list, of course with the corresponding containers on the solid floor.

### 4.5.1 SOLID authentication

First, look at the authentication flow, critical to accessing our application's functions. Without authentication or with its file, the user can only view the mind map in browser mode, to which he must know the URL address. Also, a significant part of this process takes place behind third-party functionalities. As the creators of this application, we only mediate the exchange of login credentials between the user and their solid pod provider.

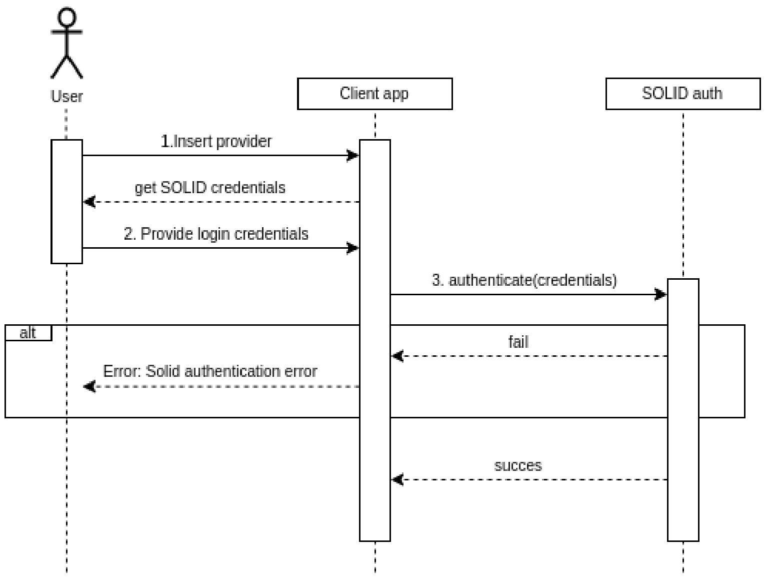

User
Client app
SOLID auth
1.Insert provider
get SOLID credentials
2. Provide login credentials
3. authenticate(credentials)
alt
fail
Error: Solid authentication error
succes

Figure 4.4: Sequence diagram: SOLID authentication

### 4.5.2 List mind maps

Now let us imagine the flow of getting the list of mind maps that the user has stored on his solid pod. The prerequisite for the actual retrieval is authentication, as with all subsequent operations, so we will not mention it further and will take it into account. This operation is the simplest of all. The user enters the application and clicks on the Mind Maps menu item to get a list of his mind maps. At that moment, the mind map service will send a request to the solid pod and then offer the user a list of his mind maps.
As we mentioned in the introduction of this section, this flow is very similar to the operations of getting the list of user classes and also the list of user conversations.

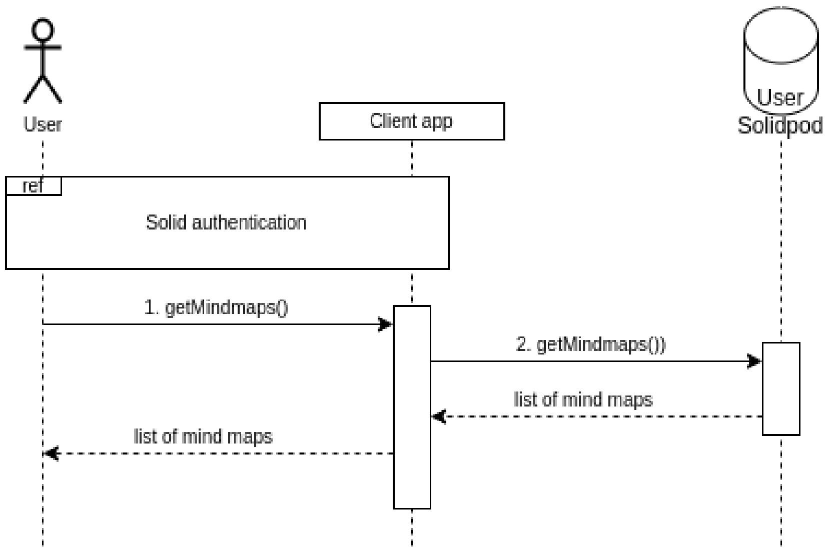


Figure 4.5: Sequence diagram: List mind maps

### 4.5.3 Create mind map

The next operation will be to create a new mind map. To create a new mind map, the user first enters the name of the new mind map and submits a request for its creation through the application. The mind map application service will create a new dataset with metadata about the mind map and a dataset to store the individual permissions in the mind map.

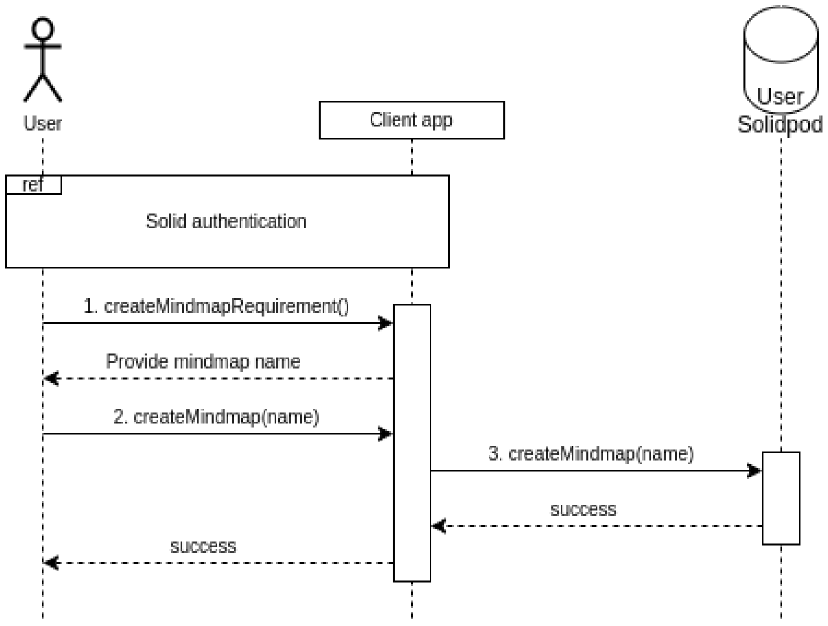


Figure 4.6: Sequence diagram: Create mind map

### 4.5.4 Edit mind map

The next operation for which we want to design a sequence diagram is editing the mind map. We can see its sequence in Figure 4.7. On the page where the user has a list of his mind maps, he chooses which one he wants to edit. So first, he loads it from his solid pod, and then he makes the edits, but during the editing, he only saves them locally in his browser, and only after clicking the save button the client sends the request to the solid pod to save the current state of the mind map.

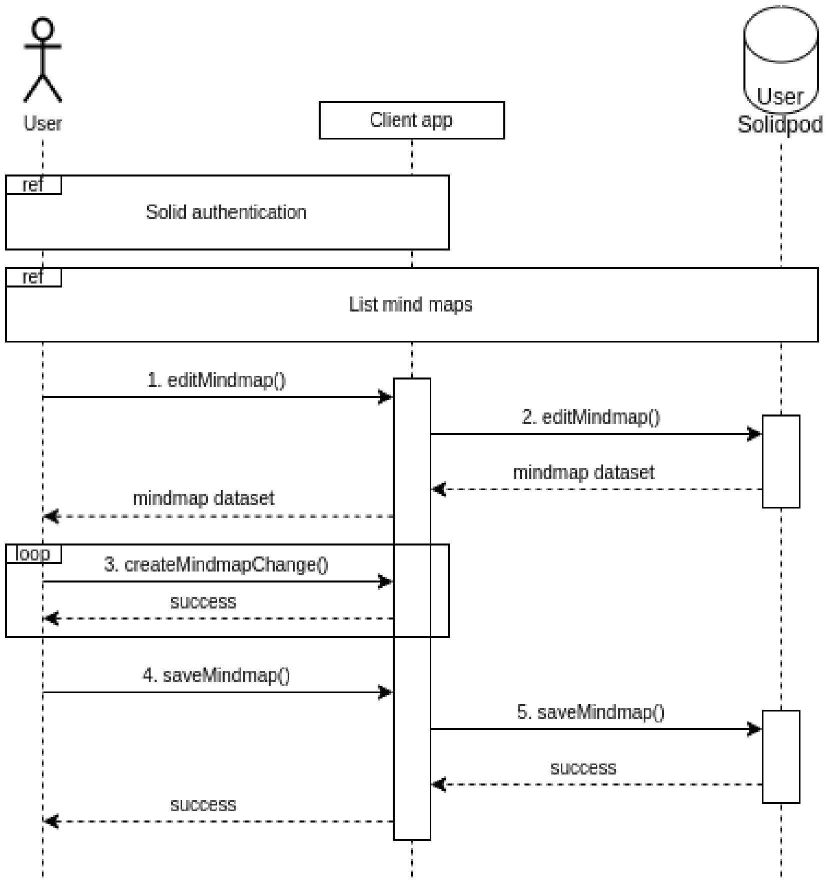


Figure 4.7: Sequence diagram: Edit mind map

### 4.5.5 Edit profile

To edit profile on the Profile page, the current profile is first downloaded from the user's solid pod. The user then makes edits, as in the case of editing mind maps, and selects the save button to send the updated profile data to their solid pod when finished as we can see in Figure 4.8.

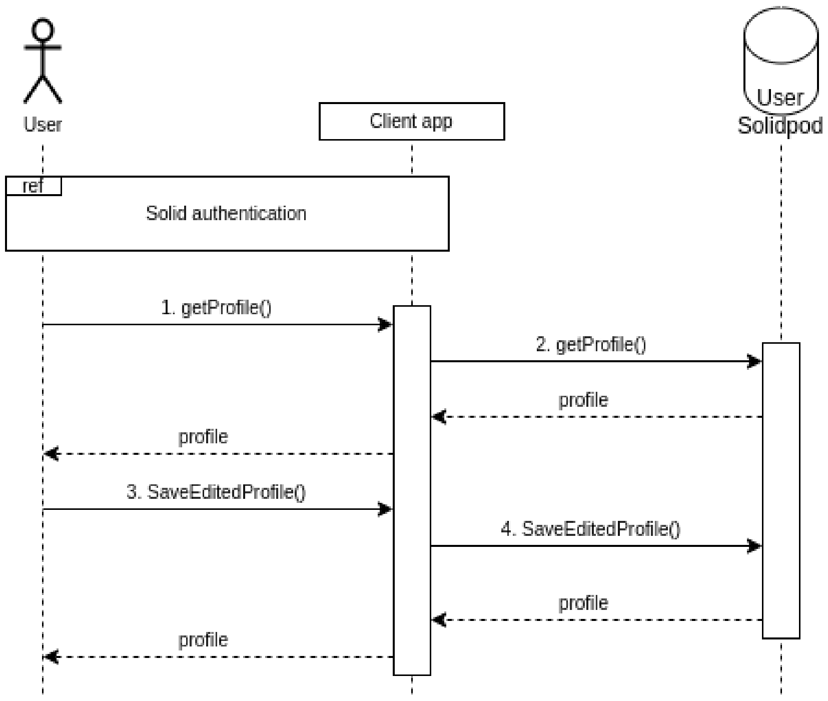


Figure 4.8: Sequence diagram: Edit profile

### 4.5.6 Create request

Now we move from editing data on our solid pod to communicating with the user's solid pod and other application users. The user inserts a link to the class he wants to enter. The client application correctly evaluates this request and sends the user's request to the solid pod of the teacher of the class(see Figure 4.9).

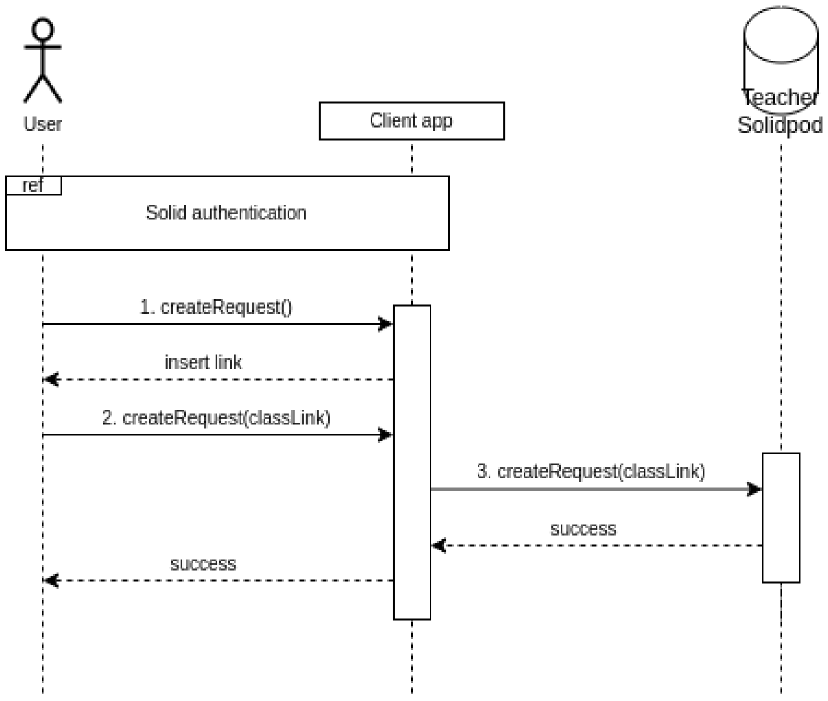


Figure 4.9: Sequence diagram: Create request

### 4.5.7 Allow access for student

This operation is probably the most trickiest of all as we can see in Figure 4.10. After the teacher accepts a student's request, several operations must be performed to guarantee the correct placement of the student in the class. First, a link to the student's profile must be stored in the class dataset. In addition, according to the request, we want to create a new chat between the teacher and the student. Thus, a new dataset for the chat is created on the teacher's solid pod. The link to this chat needs to be sent to the student again. After all this, we need to provide the new student with all the necessary access rights to the class materials.

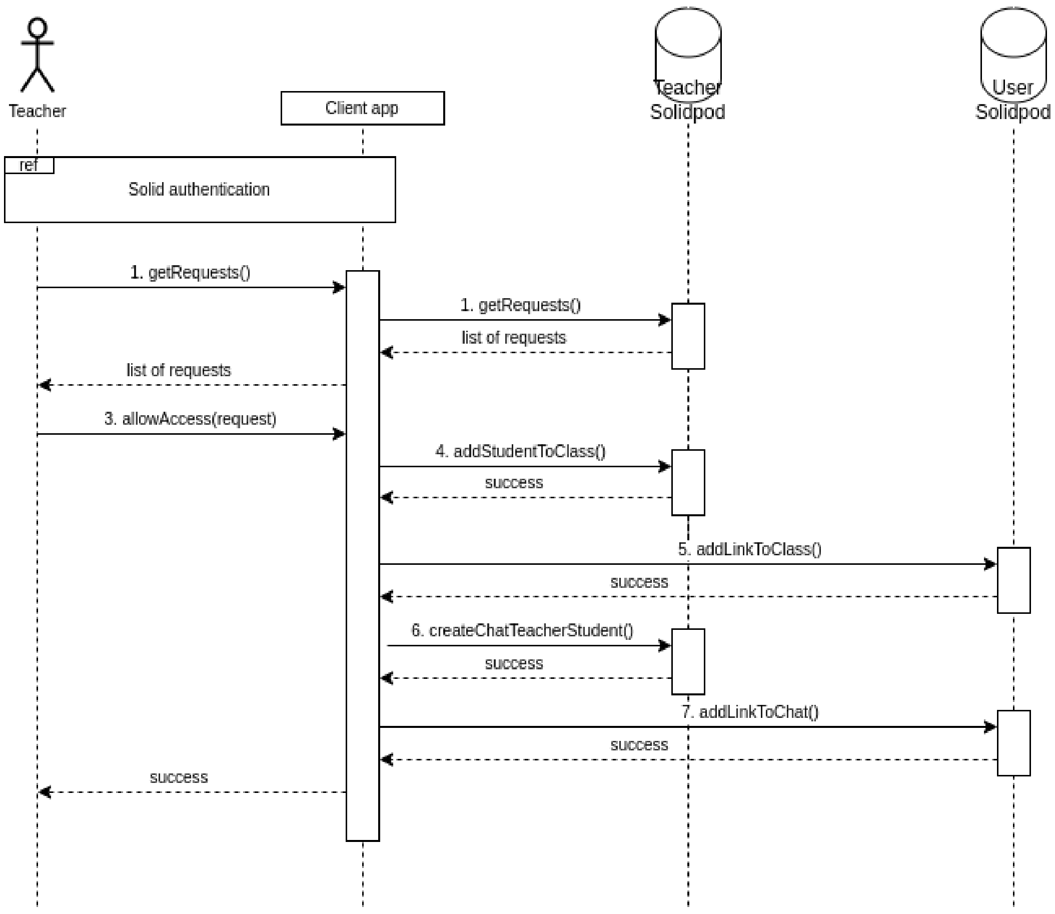


Figure 4.10: Sequence diagram: Allow access

### 4.5.8 Get class

Getting the class dataset is easier. To get all the data related to a class, which includes mind maps, class notifications, students' profiles, and their test results, we first need to get the class dataset, which contains information about the class and the location of all the other datasets that contain other data. Once we get the list of students from the class dataset, we need to get their profiles asynchronously(see Figure 4.11).

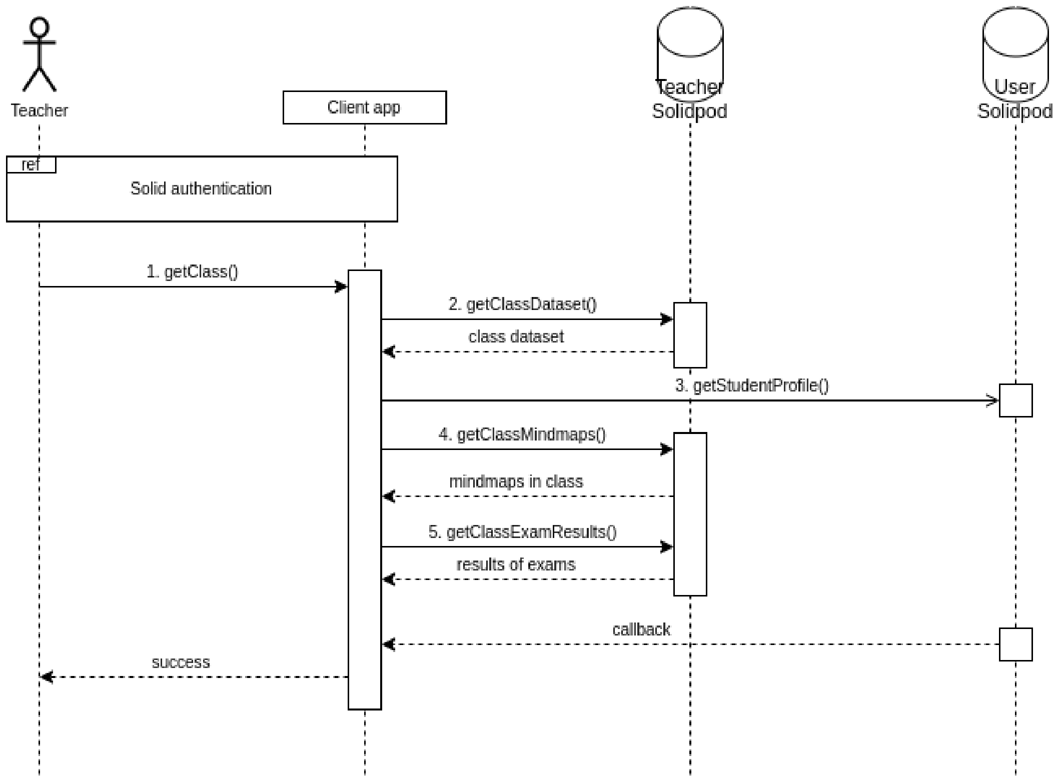


Figure 4.11: Sequence diagram: Get class

### 4.5.9 Do test

If a student wants to take an exam from a particular mind map, he must first choose a class. This includes all operations as in the previous case. The student selects the mind map for the test on the class page. We know precisely where this mind map is located from the class dataset. So we upload it, the student fills in the missing boxes, and then the test result is saved to the class dataset. This sequence we can see in Figure 4.12

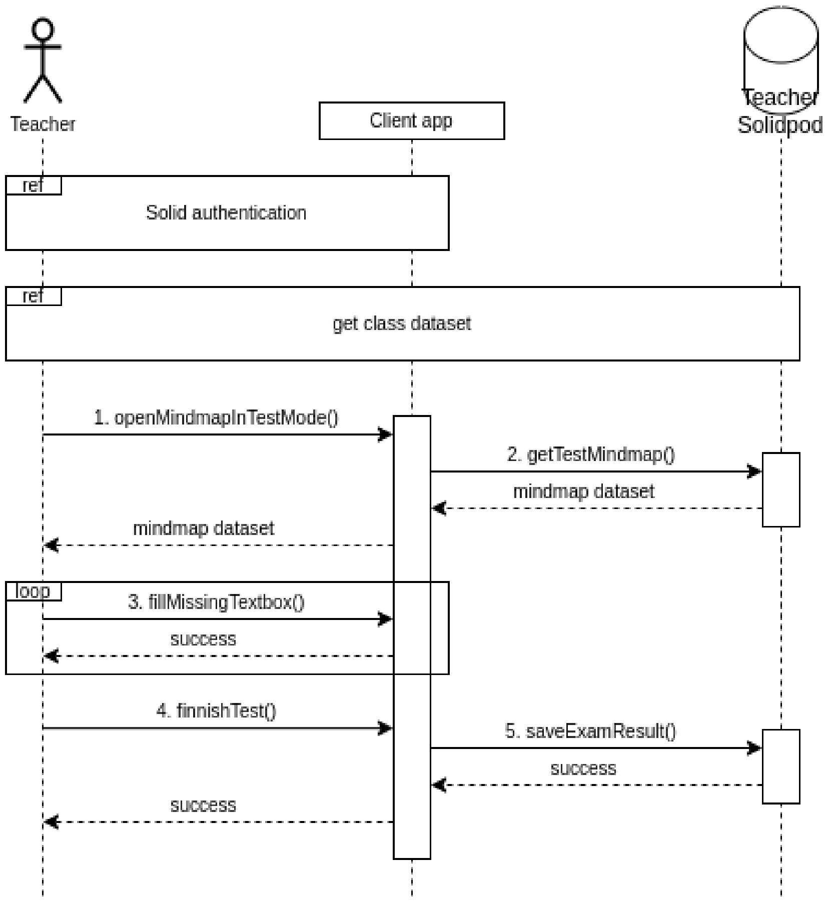


Figure 4.12: Sequence diagram: Test

# 5. Implementation

This chapter will first introduce the framework we will use to develop our web application. Then we will introduce the libraries that we will use to take full advantage of the potential of Solid.

## 5.1 Technologies

When developing web applications, it is important to have a suitable framework that provides structure and tools for efficient application development. Frameworks make developers' jobs easier by offering predefined functionality, architecture, and design patterns that can be used in application development.

### 5.1.1 Framework

Here is a list of frameworks we have considered:

1. **Vue.js** - Vue.js is a contemporary JavaScript framework designed for building user interfaces (UIs). It offers the flexibility to be used either for specific application components or as a comprehensive framework for complete web application development. Vue.js employs a user-friendly templating system, resembling HTML, which simplifies the process of defining UIs and enables the creation of dynamic and responsive user interfaces[28].

2. **Angular** - Angular is a JavaScript framework designed for crafting web applications. Developed by Google, it succeeds the previous AngularJS framework. With a declarative approach, Angular utilizes TypeScript, an extension of JavaScript enabling developers to write type-checked code, promoting clean, maintainable, and scalable application development. Notably, Angular's component-based architecture stands out as one of its primary features.

3. **React** - React is a popular open-source JavaScript library for creating user interfaces. Its main idea is a declarative approach to UI design that allows you to quickly and efficiently create interactive web applications. React uses virtual DOM to optimize performance and supports unidirectional data flow for efficient data management. The main building blocks of React are components, which are reusable blocks of code for defining different parts of the user interface.

#### Why React?

1. **Community** - React has a larger community and ecosystem than Vue.js and Angular. This means that more resources, libraries, and tools are available to developers, as well as more job opportunities for developers who specialize in React.

2. **Development** - React has been around the longest and is widely adopted by companies like Facebook, Airbnb, and Netflix. This maturity has led to a more extensive set of tools and resources.

3. **Performance** - React is known for its high performance, especially when rendering large amounts of data. React's virtual DOM system allows only the necessary UI parts to be updated, resulting in faster and more efficient rendering.

The React framework is based on the following technical foundations:

- **Virtual DOM** React uses a virtual DOM to minimize the number of changes to the real DOM. A virtual DOM is an in-memory copy of the real DOM where changes are made, and then only those changes are applied to the real DOM, significantly increasing application speed and performance.
- **JSX** JSX is an extension to JavaScript syntax that allows to insert HTML elements directly into JavaScript code. It means that React components are written in JSX, making it easier to create the UI and more readable.
- **Components** Components can be linked together in larger trees to create complex user interfaces.
- **Unidirectional data flow** This ensures that the data is always up-to-date and consistent throughout the application.
- **State Components** This allows for an interactive UI where the UI changes in response to user actions or data received from the API.

### 5.1.2 Typescript

React supports both JavaScript and TypeScript. Here are the two main reasons we chose Typescript:

- **Static type checking** TypeScript is a superset of JavaScript that adds static type checking. It is possible to add type annotations to code. TypeScript will check for correct types during development, which helps detect errors and reduces the risk of runtime bugs in the application.
- **Maintenance** Type annotations make TypeScript code more readable and well-documented. Types serve as code documentation and helping to developers to better understand the functionality of different parts of an application. This makes it easier to collaborate and maintain code over time.

## 5.2 Design pattern

Structuring your application correctly is the key to avoiding confusion and making future development easier while understanding your code. Organizing it efficiently allows you to create new features and keep track of your application effortlessly. Our application needed to solve the following problem when choosing a design pattern. On the one hand, we have a frontend in react, which mediates user

requests, e.g., Get a list of mind maps. On the other side, we have a library that sends the requests to the user's Solid pod. How can we connect these two things to make our code reusable and easily extensible? We choose the Repository-Service design pattern.
Figure 5.1 displays the Repository-Service pattern divides the application's business layer into two distinct layers.

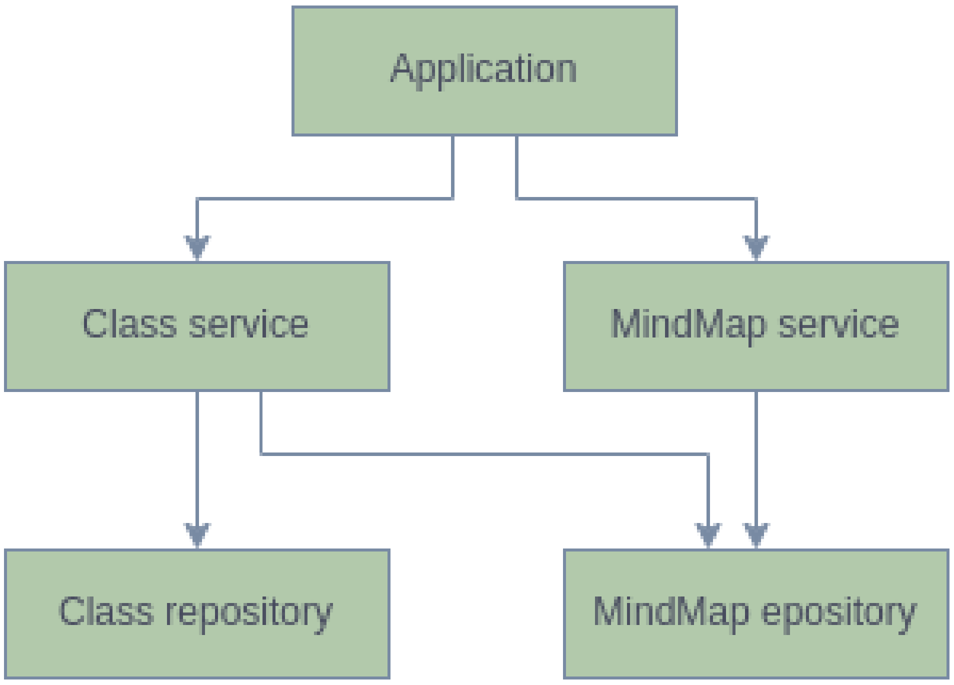


Figure 5.1: Service-repository pattern

### Repositories

On one hand, we have the Repositories, which handle data retrieval and storage from our data store. Each Repository is designed to work exclusively with a specific Model class. For example, if our models are MindMaps, Nodes, and Profiles, we would have separate repositories for each. The MindMapRepository would not directly interact with the NodeRepository, and vice versa.

### Services

On the other hand, we have the Services, which receive Repositories as dependencies. These Services can access multiple Repository classes and combine their data to create more complex business objects. Additionally, they provide an abstraction layer between the web application and the Repositories, promoting greater independence and flexibility for potential future changes.

## 5.3 Libraries

Developing an application that works with a solid pod is still not completely common, and there is not much material or recommendations to be found on the internet, so let us review the key decisions we had to address when choosing libraries and had to take into account during implementation.

### 5.3.1 Authentication

Authentication is the process of verifying the identity of an agent. To access private data on Solid Pods, you must authenticate as a user/agent who has been granted appropriate access to that data. Solid authentication is based on the Solid-OIDC specification. Solid-OIDC builds upon the OpenID Connect standards, which itself builds on the OAuth 2.0 authorization framework[24].

#### solid-client-authn-browser

There are several libraries for Solid authentication. We chose solid-client-authn-browser because its creator is Inrupt[13]. It has been in development for a long time, and to this day, its creator provides updates and support. When using this library, authentication is straightforward and proceeds as follows(see 5.2).

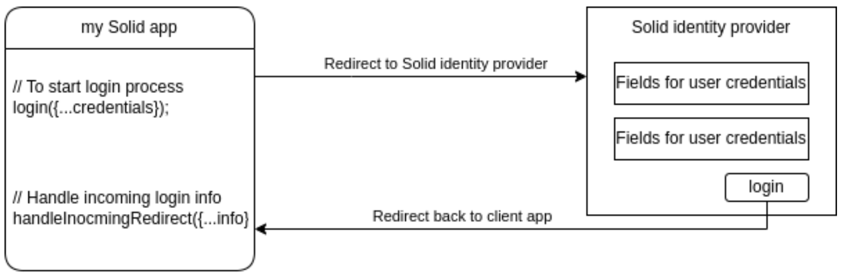

my Solid app
// To start login process
login({...credentials});
// Handle incoming login info
handleInocmingRedirect({...info}
Redirect to Solid identity provider
Solid identity provider
Fields for user credentials
Fields for user credentials
login
Redirect back to client app

Figure 5.2: Authentication flow

### 5.3.2 Library for data operations

When realistically implementing a data model for the SOLID platform, it is important to keep in mind that SOLID has a different design style compared to traditional relational or document databases. Therefore, let's review the basic principles of how data is represented on the SOLID platform.
The basic elements in SOLID are as follows:

- **Thing** Thing refers to a data entity. For example, if you want to store information about a course, you can have a Thing that represents a book for the course, and store various properties (such as title, author) for that book; the author itself can be another Thing with its own properties. The Thing is stored as part of the SolidDataset,
- **SolidDataset** contains a set of Things. This means that you don't save a Thing independently without a SolidDataset. You can organize SolidDatasets in a Container.
- **Container** Container can contain SolidDatasets and other resources, including other Containers. Containers are analogous to folders in a directory structure. For example, in your module you might have: A Container named football21/; football23/ contains another Container named leagues/;

leagues/ contains the SolidDataset that corresponds to the PremierLeague; and SerieA contains the Things data (such as statistics) for the season.

The SOLID platform focuses on the principles associated with Linked Data and the Web as a data environment. The SOLID data model focuses on the semantics of data and its structured description using ontologies. Ontologies define the relationships between entities and enable the representation of data in a semantic graph. This increases the flexibility and connectivity of the data.

**@inrupt/solid-client**

Again, this is a library from Inruptu. It provides developers with an abstract layer over RDF data and allows them to easily create, read, update and delete data in Solid pod. Using Solid Client, you can work with RDF graphs that represent the structure of the data in your Solid pod. The library provides functions for manipulating RDF triples, querying data, creating new entities, and managing access rights[14].
It is very simple to use. Let's demonstrate how to store a Profile class on our Solid pod. First we create a Profile thing, which we then want to save.

```
const profile: ThingLocal = buildThing(
    createThing({ name: ProfileId }))
        .addUrl(rdf_type, "wikimind#Profile")
        .addStringNoLocale(rdf.name, object.name)
        .addStringNoLocale(rdf.webId, object.webId)
        .addStringNoLocale(rdf.surname, object.surname)
        .addStringNoLocale(rdf.ownerPod, object.ownerPod)
        .build();
}
```

Listing 5.1: Creating of profile thing

Now we can save profile to Solid using this library.

```
const myDataset = await getSolidDataset(
    "mySolidPodUrl", { fetch });
const savedProfileSolidDataset = setThing(
    myDataset, profile);
await saveSolidDatasetAt(
    "mySolidPodUrl, savedProfileSolidDataset, { fetch });
```

Listing 5.2: Save profile thing with @inrupt/solid-client to Solid pod

### 5.3.3 SOLID servers

Before we introduce the other used tools, especially for working with access rights but also for receiving notifications about changes on the solid pod, let us introduce the possible types of solid pod because each has its specifics. Even though they meet the specifications of SOLID, they are not 100

- **Node Solid Server** Node Solid Server is a Node.js-based open-source implementation of Solid Server, adhering to Solid protocols and specifications. Easy to deploy on various infrastructures or cloud environments, Node Solid Server boasts an active developer community, providing regular updates and continuous improvement, ensuring robust support and the addition of new features. It is one of the most popular variants of Solid Server.

- **Community Solid Server**
  Community Solid Server is a Node.js-based open-source implementation of Solid Server that excels in fostering user data sharing and collaboration among Solid applications. It serves as a dedicated repository for Solid data and is specifically crafted to uphold the decentralized and community-driven ethos of the Solid platform.

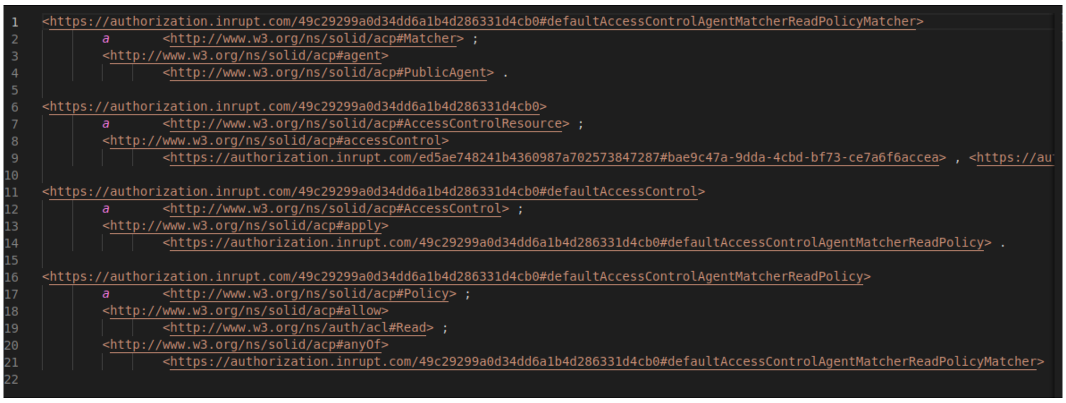
- **Inrupt Pod Server** Inrupt Pod Server is a commercial Solid server solution provided by Inrupt. Inrupt Pod Server provides storage and management of personal data on the Solid platform. Inrupt also focuses on providing tools and services to help users take better control of their data and gain more control over their digital identity.

### 5.3.4 Access control library

Access control/authorization determines which actions an Agent can perform on a Resource. For instance, an agent may have Read Access to a Resource, but not Write Access. Different Solid Servers can support different access control mechanisms; e.g., Access Control Policies (ACP) or Web Access Control (WAC).

#### Universal API library

We chose the Universal API library from Inrupt as the access policy library. Since we have multiple server types, we have multiple access policy solutions. This library offers a universal API for both mechanisms, so as a developer, I do not have to decide whether Solid should be under the ACP access control mechanism or the WAC access control mechanism[15]. This makes our work much more manageable. Both mechanisms are diametrically different, and the way of storing access information in both cases is very different. In fig.5.3, we can see the public access storage for the profile dataset with the ACP mechanism. On the other hand, the WAC method is much more information-poor, whose ttl. The file storing access rights for profile datasets can be seen in Fig. 5.4.



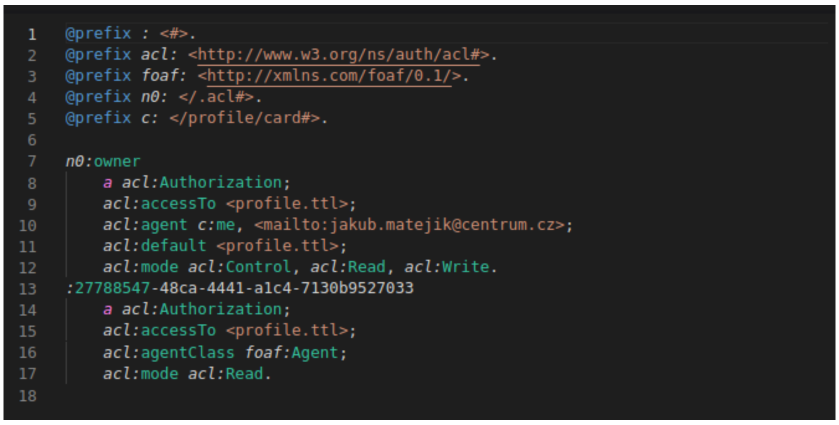
```
<https://authorization.inrupt.com/49c29299a0d34dd6a1b4d286331d4cb0#defaultAccessControlAgentMatcherReadPolicyMatcher>
        a       <http://www.w3.org/ns/solid/acp#Matcher> ;
        <http://www.w3.org/ns/solid/acp#agent>
                <http://www.w3.org/ns/solid/acp#PublicAgent> .

<https://authorization.inrupt.com/49c29299a0d34dd6a1b4d286331d4cb0>
        a       <http://www.w3.org/ns/solid/acp#AccessControlResource> ;
        <http://www.w3.org/ns/solid/acp#accessControl>
                <https://authorization.inrupt.com/ed5ae748241b4360987a702573847287#bae9c47a-9dda-4cbd-bf73-ce7a6f6accea> , <https://au

<https://authorization.inrupt.com/49c29299a0d34dd6a1b4d286331d4cb0#defaultAccessControl>
        a       <http://www.w3.org/ns/solid/acp#AccessControl> ;
        <http://www.w3.org/ns/solid/acp#apply>
                <https://authorization.inrupt.com/49c29299a0d34dd6a1b4d286331d4cb0#defaultAccessControlAgentMatcherReadPolicy> .

<https://authorization.inrupt.com/49c29299a0d34dd6a1b4d286331d4cb0#defaultAccessControlAgentMatcherReadPolicy>
        a       <http://www.w3.org/ns/solid/acp#Policy> ;
        <http://www.w3.org/ns/solid/acp#allow>
                <http://www.w3.org/ns/auth/acl#Read> ;
        <http://www.w3.org/ns/solid/acp#anyOf>
                <https://authorization.inrupt.com/49c29299a0d34dd6a1b4d286331d4cb0#defaultAccessControlAgentMatcherReadPolicyMatcher>
```

Figure 5.3: ACP policy RDF



```
@prefix : <#>.
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix n0: </.acl#>.
@prefix c: </profile/card#>.

n0:owner
    a acl:Authorization;
    acl:accessTo <profile.ttl>;
    acl:agent c:me, <mailto:jakub.matejik@centrum.cz>;
    acl:default <profile.ttl>;
    acl:mode acl:Control, acl:Read, acl:Write.
:27788547-48ca-4441-a1c4-7130b9527033
    a acl:Authorization;
    acl:accessTo <profile.ttl>;
    acl:agentClass foaf:Agent;
    acl:mode acl:Read.
```

Figure 5.4: AWC policy RDF

### 5.3.5 Notifications

Part of the application is also the sharing of messages between users. For user-friendliness, we wanted to implement a WebSocket, which subscribes to the data storage of the opened chat and serves updated messages to the user.
There is a Solid Notifications Protocol for this purpose. This defines an extensible HTTP-based framework that allows client applications to receive notifications of changes to HTTP resources. The goal of this protocol is to enable interaction patterns for both situations where the client maintains an open connection to receive notifications and situations where the client requests a notification and then disconnects. It should also be noted that all server types do not implement notifications in same way.

# 6. Testing and evaluation

The following section provides an overview of the test procedures. The chapter will start by describing the preliminary technologies used to implement automated testing and then introduce other aspects of the application that we have tested.

## 6.1 Technologies

When testing our application, we used the following technologies:

- **Jest** is a framework for testing JavaScript solutions. It is used to write and run automated tests for web applications, APIs, and other software systems. Jest is high-performance, simple to use, and provides a robust set of features for testing, including assertions, mocking code coverage, and asynchronous testing. Due to its popularity and active community, Jest offers broad support and extensibility.
- **tslint** is an extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors. It is widely supported across modern editors and build systems and can be customized with its own lint rules, configurations, and formatters.
- **eslint** is a static analysis tool for JavaScript code that identifies and detects errors, formatting issues, and inconsistencies in code. ESLint is often used in development environments and tools to integrate with development workflows, allowing quickly identifying and fixing code problems during development.
- **@testing-library/react** is a library for testing components in the React framework. The main principle of Testing Library/React is that tests should reflect the actual user behavior in the application. The tests focus on how the component behaves and interacts with the user. This ensures that the tests are robust and survive code changes. It simulates events such as clicks, text entries, or form submissions.

## 6.2 Unit and alpha testing

In order to verify the basic functionality during development at each code change or to quickly detect a bug in the code, we have created unit tests. The goal is to verify the correct functionality and behavior of each class, service, and module. We created unit tests to test the functionality of individual repositories, and services, correct querying, and create and read Things object from Solid pod. So for all the services, we need in our application.
As we have already introduced the Service-Repository pattern, the unit tests were well structured because, with this design pattern, the functionality of the individual repositories and services is nicely separated.
In order to verify the overall functionality of the application, we decided to choose Alpha testing, starting from the early stages of development. In our opinion,

Alpha testing suits the current state of Solid technology. Unfortunately, there are not many variants of Solid pod mock-ups for this technology yet. As a possibility, we tried Molid. However, it limited us in more complex testing because, as it says on the opening page of its documentation, it still lacks the following functionalities: authentication/authorization, link headers, metadata of container contents (sizes, modification dates, etc.), updating data (POST, PUT, PATCH), support for any file formats other than turtle and so on. That is why alpha testing seemed like the best option.
We did this by defining the application's intended functionality based on use cases and operations. With the help of these, however, creating the scenarios was relatively easy. Here is the basic one:

1. Sign in with solid pod. If you have no one, create one.
2. Create a first mind map.
3. Model domain in an empty mind map. Use colors and similarity search. Then save the mind map.
4. Display mind map entities on the timeline.
5. Edit profile.
6. Create first class.
7. Invite friends.
8. Accept their requests.
9. Create a mind map for them.
10. Send a message to one of them.

## 6.3 Performance testing

Since we mentioned performance for reading from the Solid pod by multiple users as one of the main aspects when designing our application on the Solid platform, we decided to verify these assumptions. Just as we mentioned in the thesis by Bruland[6], the author did a much more comprehensive analysis. This involved aggregating data with a very variable number of users and variable sizes of the processed datasets. In this thesis, we will test our created application only on the operations we need in our application. Thus, in the case of the class, it is reading the student's profiles, reading various large datasets of mind maps, which are for students as study materials, and also the test results.
However, what will be different, we will not perform this testing on laboratory conditions, but we will verify it on real solid pods of actual providers.
Here we will introduce the ones we will use in our testing:

1. **Inrupt Pod Spaces**
   These Solid Pods are solutions from the already mentioned Inrupt company, which is the company behind Tim Berners Lee. This is the only solution

based on the ESS server type. Since this solution is relatively new and still being experimented with by its publisher, it is not recommended for production apps. On many sites and forums, it is often mentioned that this solution has the best results in terms of speed of data delivery. For Europe, the data is physically stored in Germany.

2. **iGrant.io**
   This is a solution from Sweden and uses the NSS server type. This option was very similar to the ESS-type implementation during development. We did not notice any significant differences except for Access policies.

3. **redpencil.io**
   After the previous two types, we will introduce the remaining type here. This Solid pod is of type CSS. This seemed to be the least debugged type during development. Although it meets the Solid specification, there was a problem with getting the URL of the Solid pod with which the webId is associated. Additionally, it does not support notifications.

Setting up accounts with all hosts went without a problem. For testing purposes, we created 7 Solid pods at each provider. The teacher solid pod from which we read the class data is stored on the Inrupt solid pod.
After this preparation, we had to consider what cases to test. We decided to focus on the most complex part of our application, namely, reading data from the solid pod to load the class. So we will examine two cases.

#### Fetch multiple student profiles

In this case, we want to examine the time of reading all the data when we have different numbers of students and fetch their profiles from their solid pod. We will examine the results for 5, 10, and 20 students.
For five students, we have chosen two solid pods from Inruptu, two from Grant.io, and one from Redpencil; for ten students, three from Inrupt, four from Grant.io, and three red pencils. Finally, we used all of them simultaneously, one from Inrupt being a teacher. Results we can see in table 6.1.

#### Fetching multiple mind maps from one pod

In this second case, we did not have any students in the class, but we inserted five, ten, and twenty mind maps and tracked the time(see table 6.2) it took to load these datasets.

| | 5 | 10 | 20 |
|---|---|---|---|
| 1 | 1235 | 1954 | 2897 |
| 2 | 1687 | 1897 | 3125 |
| 3 | 1245 | 1788 | 4851 |
| 4 | 1145 | 2145 | 3568 |
| 5 | 982 | 2451 | 3122 |
| 6 | 1345 | 1645 | 2235 |
| 7 | 1258 | 1287 | 2789 |
| 8 | 1798 | 1653 | 2678 |
| 9 | 2468 | 1287 | 2540 |
| 10 | 1359 | 1789 | 2123 |
| max | 2468 | 2451 | 4851 |
| min | 982 | 1287 | 2123 |
| average | 1497.2 | 1840.4 | 3155.5 |

Table 6.1: Fetch time multiple student's profiles[ms]

| | 5 | 10 | 20 |
|---|---|---|---|
| 1 | 752 | 1235 | 1984 |
| 2 | 1210 | 1789 | 2458 |
| 3 | 1125 | 1687 | 2788 |
| 4 | 456 | 1875 | 1987 |
| 5 | 788 | 1235 | 3819 |
| 6 | 1241 | 1247 | 2235 |
| 7 | 877 | 1452 | 2145 |
| 8 | 1724 | 970 | 2475 |
| 9 | 1368 | 1120 | 1879 |
| 10 | 965 | 1324 | 2641 |
| max | 1724 | 1875 | 3819 |
| min | 456 | 970 | 1879 |
| average | 1008.4 | 1381.4 | 2500.1 |

Table 6.2: Fetch time multiple mind maps[ms]

**Results**

Based on the results of this test, we can see quite a noticeable difference between the case when we read one small dataset from multiple Solid Pods and when we read multiple datasets from one. When observed during this testing that the value of 2500ms is a breakpoint when the user gets the application running smoothly and when he already notices that it takes a while.

## 6.4 User evaluation

Since there was a strong requirement for user-friendliness, we focused more on this aspect during testing. We had the application evaluated by the user. In order to make the conditions the same and to come to some relevant conclusions, we gave the users the same tasks as in alpha testing.
After doing these tasks, I gave them a questionnaire about their immediate feelings about the app. If they had to, what review would they write about it?

Here are their feedbacks:

1. **user 1(male, 57 years)** ”The only thing that was so frustrating for me was creating a solid pod and then logging in with it. Can’t that be separated? I completed all the tasks without much problem.”
2. **User 2(female, 27 years old)** ”Very nice app. The environment is quite nice and it can do interesting things.”
3. **User 3(female, 48 years old)** Very nice app. I haven’t had any problems”
4. **User 4(female, 10 years)** ”Good. Why aren’t there pictures of everything like the cards?”
5. **User 5(male, 25 years)** ”Very good, only the Solid is not nice thing yet.”

We could make some judgements already from these short reviews, but in order to make some judgements, we had the user fill out a short SUS questionnaire.
System Usability Scale (SUS) is a questionnaire to find out how satisfied the customer was with the product. It measures its perceptions of its usefulness[27].
It contains the following questions:

1. I like to use the application often?
2. Isn’t the system too complicated for me?
3. Is it easy to use?
4. Do I need someone for advice on using the system?
5. Does the system have some great features?
6. Is the system too inconsistent?
7. Does the system have a fast learning curve for a lot of people?

8. Isn't the system too cumbersome for anyone?
9. Do I feel confident in using it?
10. Was it necessary to learn a lot of things to work with the system?

Here are the results of the questionnaire:

| | user1 | user2 | user3 | user4 | user5 |
|---|---|---|---|---|---|
| 1 | 4 | 4 | 5 | 3 | 4 |
| 2 | 1 | 1 | 2 | 1 | 2 |
| 3 | 4 | 5 | 4 | 5 | 4 |
| 4 | 3 | 1 | 3 | 1 | 1 |
| 5 | 5 | 5 | 5 | 5 | 3 |
| 6 | 2 | 2 | 3 | 2 | 2 |
| 7 | 5 | 3 | 3 | 3 | 4 |
| 8 | 2 | 2 | 1 | 1 | 3 |
| 9 | 4 | 5 | 4 | 3 | 5 |
| 10 | 1 | 1 | 1 | 1 | 1 |
| **SUS Score** | **82.5** | **87.5** | **77.5** | **82.5** | **77.5** |

Table 6.3: SUS results

It is often stated that a result above 70 is very good. When we asked the participants in more detail what the good feeling was based on when using the application, it was mainly based on creating the graph, which they found very good. The rest of the application is limited due to fetching data from the Solid pod.

# 7. Documentation

In the next chapter, we will introduce the documentation, first for developers and then for users of our application.

## 7.1 Developer documentation

Here is the abbreviated technical documentation, where we will summarize everything that might interest programmers to make it easier to get to know our project. We will present all the questions that programmers might have who would like to build on or be inspired by the design of this solution.

### 7.1.1 Repository

The application created in this thesis is an attachment **A.1**. It is also a repository on GitHub at https://github.com/matejikj/wikimind. The repository's structure can be seen in Figure 7.1.

Now let us present some common operations:

#### Add new type

First, we start by adding a new type that we want to work within the application that must be stored correctly on the Solid pod. The src/models/types folder contains the type definitions we work within the rest of the code. To write and read this type from Solid, we need some remapping. We do this by, for example, having a Linked data object NodeLDO in src/models/types to Node, which is then written to Solid. We must create a corresponding JSON file in definitions mapping each property type.

#### Add new service and repository

So we added a new type and defined its conversion to a Linked data object for storage on Solid pod. Now we want to read it and store it on a Solid pod. We will create a corresponding repository for our type for the most straightforward read-and-write operations. We have only the simplest data operations in the repository on the Solid pod.
If we want more complex operations, we add a service for our domain when we follow the Service/Repository pattern, in which we use the repository interface.

#### Add new page

We should add the React component to src/pages to add a new page and register the component in App.tsx to routes.

```
wikimind.......................................project folder
├─ docs..........................................Documentation
├─ public....................Directory with main HTML document
├─ src.........................................source folder
│  ├─ tests...............................................Tests
│  ├─ components..............................Small components
│  ├─ dbpedia.....................Services for DBPedia queries
│  ├─ definitions..............JSON definitions of RDF objects
│  ├─ models..............Types of entities and mapping to RDF
│  ├─ pages...........................................Apps pages
│  ├─ repository.............Repositories for Solid operations
│  ├─ service.....................Services for user operations
│  ├─ styles..........................................CSS styles
│  ├─ visualisation.............Components for mind map editor
│  ├─ App.tsx.......................Main component with router
│  └─ index.tsx..................Main entrance point for React
├─ package.json..........................Application packages
├─ tsconfig.json.........................Config forTypescript
└─ wikimind.ttl..........................Wikimind vocabulary
```

Figure 7.1: The structure of the Wikimind directory.

### 7.1.2 Scripts

The application contains scripts for creating its build and running it in development mode.

Before running scripts:

```
npm i
```

For running the app in the development mode:

```
npm start
```

For running the tests:

```
npm test
```

And to create the build:

```
npm run build
```

## 7.1.3 CI/CD

During the development process, we used AWS Amplify, a platform from Amazon that allows developers to quickly and easily develop and deploy web and mobile applications. It provides tools and services that make developing, testing, and deploying applications easy[5].

```
version: 1
applications:
  - frontend:
      phases:
        preBuild:
          commands:
            - npm ci
        build:
          commands:
            - npm run build
      artifacts:
        baseDirectory: build
        files:
          - '**/*'
      cache:
        paths:
          - node_modules/**/*
    appRoot: client
```

Listing 7.1: AWS flow definition

The configuration is very simple and can be executed with a few clicks. Amplify creates a YAML file for the configuration. This YAML file defines the configuration for the CI/CD process. It contains the "preBuild" and "build" phases, which run commands to install dependencies and compile the application. The resulting build is stored in the "build" directory as we can see in Figure 7.1.3. There is the link for the currently deployed version in the readme repository.

## 7.1.4 Automated documentation

TypeDoc is a documentation generator for TypeScript projects. It automatically generates API documentation from the TypeScript source code. The output of this tool can be seen in the docs folder. Here we can find structured documentation for our TypeScript classes, types, services, etc.

## 7.2 User documentation

When designing the user UI, the greatest demands were placed on simplicity and clarity, and the user should learn to work with the application in no time. However, the mind map editor could be more difficult to learn, so here is the complete user documentation.

### 7.2.1 Authentication

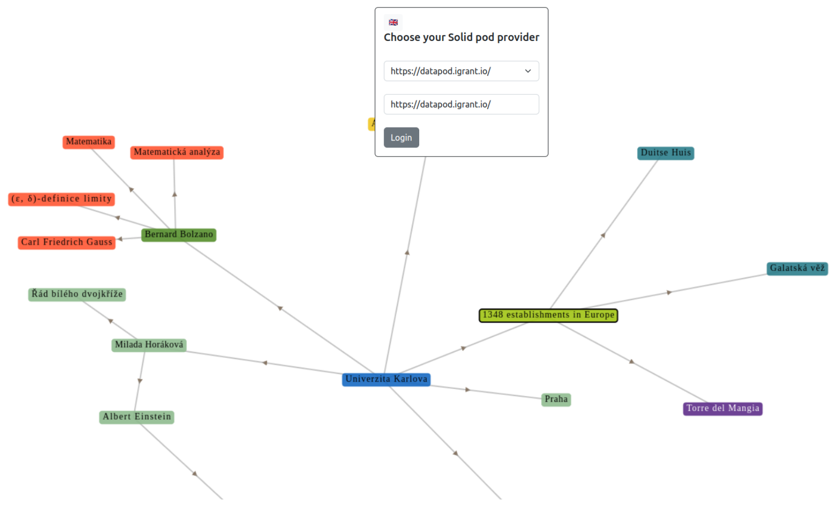


Figure 7.2: Wikimind - Login

The application allows only authenticated users to enter. Therefore, it is necessary to authenticate with Solid Pod on the login page first. On the login page, the user can choose from a list of pre-selected providers or insert a direct link to their provider(see Figure 7.2).

### 7.2.2 App layout

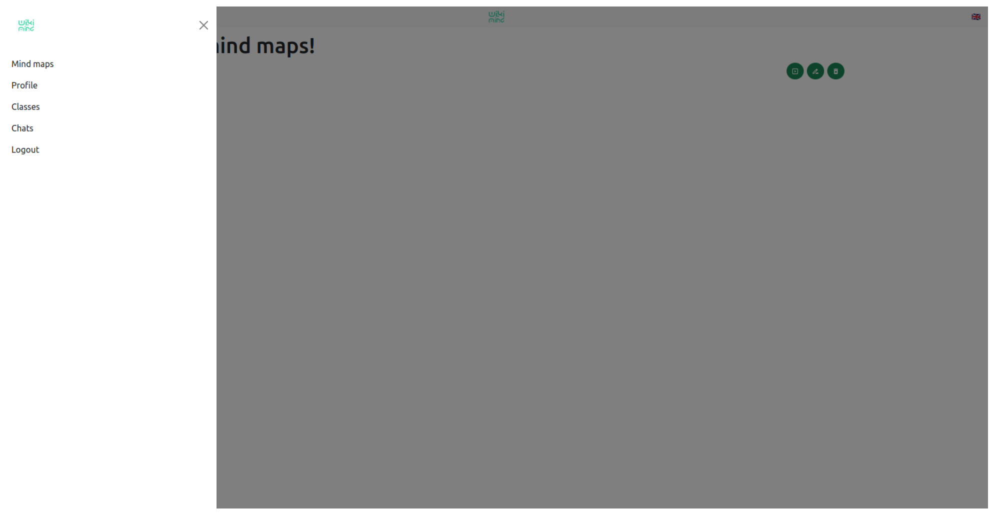


Figure 7.3: Wikimind - Menu

The application has a very simple structure. All pages can be accessed from the main menu. This can be expanded by clicking on the menu icon on the top left. After clicking, the menu will open, and the user can click the page user wants(see Figure 7.3).

On the top right, there is the option to change the localization of the application. It can be chosen between English and Czech. In case the language is changed after searching, the change will be reflected when the next query.

### 7.2.3 List of mind map

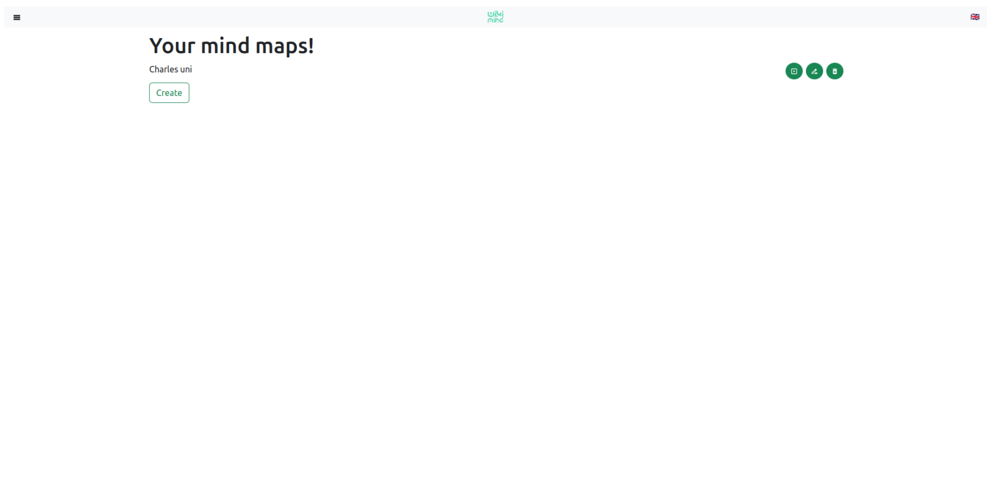


Figure 7.4: Wikimind - Mind maps list

The list of users' mind maps has been selected as the main page by default(see Figure 7.4). On this page, the user is shown all the mind maps he has stored on his solid pod, which are in Figure. (The class mind maps can be displayed only from the class).

In the list, each mind map can be displayed, renamed, or removed.

After clicking on rename, a dialog box for inserting a new name is displayed. Clicking on delete will remove the mind map.

Creating a new mind map on Solid pod is also possible by clicking the Create button. Once clicked, enter a name and click confirm and the application will be redirected to the new mind map editor.

## 7.2.4 Mind map editor

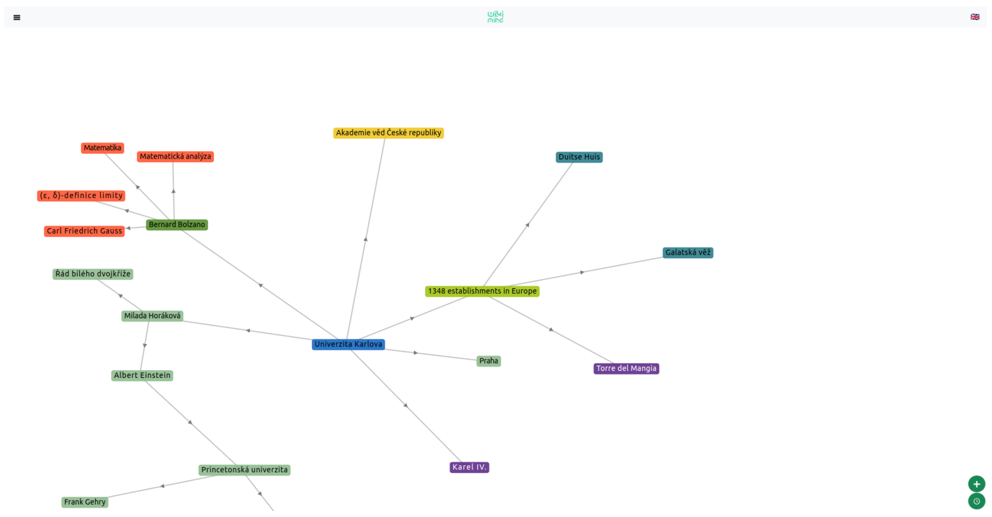


Figure 7.5: Wikimind - Editor start

If the user displays his mind map(see Figure 7.5), it will open in edit mode. This includes many options.

The screen of editor which can be seen in Figure 7.5 is empty after creating. The canvas can be zoomed in and zoomed out. To add a new entity, click on the Plus icon on the bottom right. This will display a new bar at the top, which allows to add new entities to the mind map.

More icons are displayed now.

After clicking on the first icon, the mind map is saved in PNG format.

The second icon is for saving the current state of the mind map.

The third icon Minus will hide the list again.

There is also a Watch icon.

This takes user to the timeline view. The timeline view will be described later.

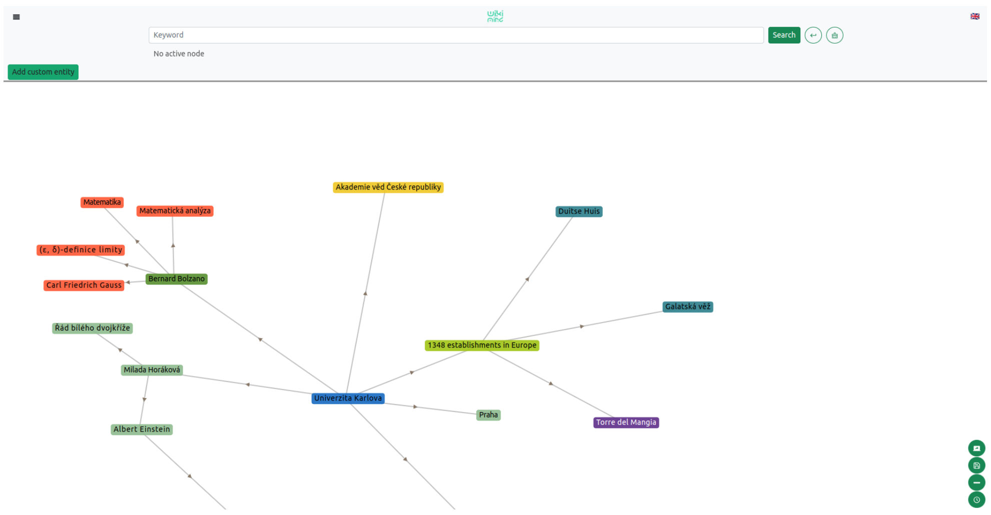


Figure 7.6: Wikimind - Top bar

On the left, under the search list, there is a button Add the custom entity to add a custom entity with a custom name(see Figure 7.6).

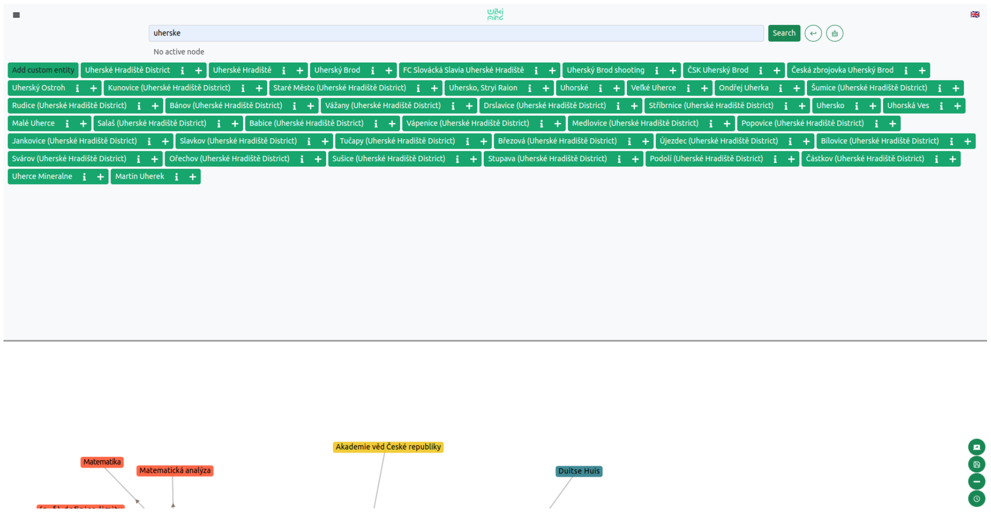
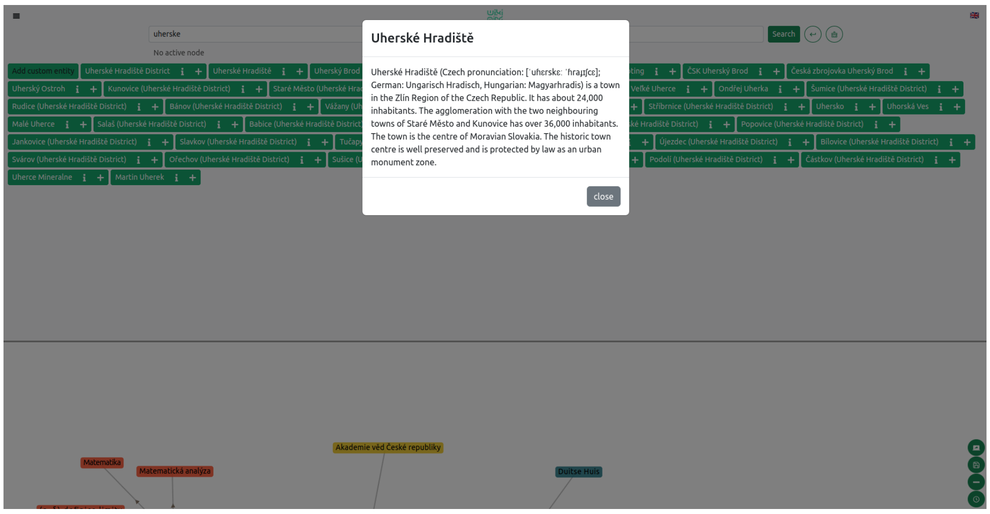

uherske
Search
No active node
Add custom entity
Uherské Hradiště District
Uherské Hradiště
Uherský Brod
FC Slovácká Slavia Uherské Hradiště
Uherský Brod shooting
ČSK Uherský Brod
Česká zbrojovka Uherský Brod
Uherský Ostroh
Kunovice (Uherské Hradiště District)
Staré Město (Uherské Hradiště District)
Uhersko, Stryi Raion
Uhorské
Veľké Uherce
Ondřej Uherka
Šumice (Uherské Hradiště District)
Rudice (Uherské Hradiště District)
Bánov (Uherské Hradiště District)
Vážany (Uherské Hradiště District)
Drslavice (Uherské Hradiště District)
Stříbrnice (Uherské Hradiště District)
Uhersko
Uhorská Ves
Malé Uherce
Salaš (Uherské Hradiště District)
Babice (Uherské Hradiště District)
Vápenice (Uherské Hradiště District)
Medlovice (Uherské Hradiště District)
Popovice (Uherské Hradiště District)
Jankovice (Uherské Hradiště District)
Slavkov (Uherské Hradiště District)
Tučapy (Uherské Hradiště District)
Březová (Uherské Hradiště District)
Újezdec (Uherské Hradiště District)
Bílovice (Uherské Hradiště District)
Svárov (Uherské Hradiště District)
Ořechov (Uherské Hradiště District)
Sušice (Uherské Hradiště District)
Stupava (Uherské Hradiště District)
Podolí (Uherské Hradiště District)
Částkov (Uherské Hradiště District)
Uherce Mineralne
Martin Uherek
Akademie věd České republiky
Matematika
Matematická analýza
Duitse Huis

Figure 7.7: Wikimind - Searching

At the top is a search bar, with the help of which the user can enter a keyword, with the help of which we can find the entities we want to search for(see Figure 7.7). To search, click on the Search button. To the right of the button are the Backspace and Clean icons.

After clicking on Backspace, we will return to the search if we have already searched for an entity.

After clicking on Clean, the list of entities according to the search is deleted.

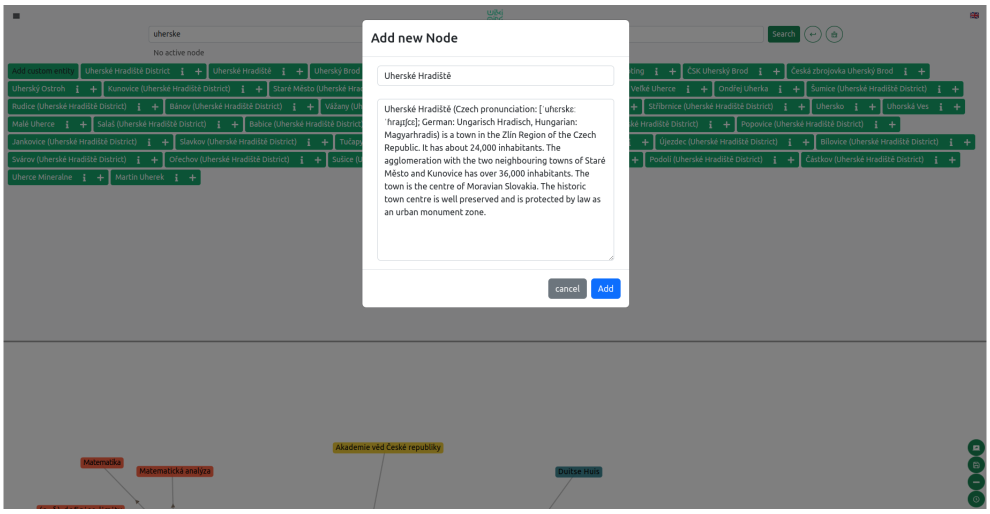


Uherské Hradiště

Uherské Hradiště (Czech pronunciation: [ˈuɦɛrskɛː ˈɦraɟɪʃcɛ]; German: Ungarisch Hradisch, Hungarian: Magyarhradis) is a town in the Zlín Region of the Czech Republic. It has about 24,000 inhabitants. The agglomeration with the two neighbouring towns of Staré Město and Kunovice has over 36,000 inhabitants. The town is the centre of Moravian Slovakia. The historic town centre is well preserved and is protected by law as an urban monument zone.

Figure 7.8: Wikimind - Recommendations detail

We can click on the information icon in the entity menu to see its detail(see Figure 7.8). To add it to the mind map, click on Plus icon.



Add new Node

Uherské Hradiště

Uherské Hradiště (Czech pronunciation: [ˈuɦɛrskɛː ˈɦraɟɪʃcɛ]; German: Ungarisch Hradisch, Hungarian: Magyarhradis) is a town in the Zlín Region of the Czech Republic. It has about 24,000 inhabitants. The agglomeration with the two neighbouring towns of Staré Město and Kunovice has over 36,000 inhabitants. The town is the centre of Moravian Slovakia. The historic town centre is well preserved and is protected by law as an urban monument zone.

Figure 7.9: Wikimind - Add node

After clicking on Plus, we will see the editor that pre-filled the data according to DBPedia, but we can edit it. To add it to the mind map, click on Add(see Figure 7.9).

If we want to avoid adding an entity directly but want to discover other entities with which it is connected, click on its name. This will display the suggested entities according to the selected one.

Also, at the top of the list, we see No active node information. If we click on any entity in the mind map, it will become active, and we will see a menu of icons.

1. **Information** Let us display its detail in the editor, where we can edit it.
2. **Question mark** This option is for the mind map in the class. If the icon is grey, the node is not in the test. If it is blue, it will be hidden in the test.

3. **Lens** Search for suggestions of similar entities according to the currently selected one.

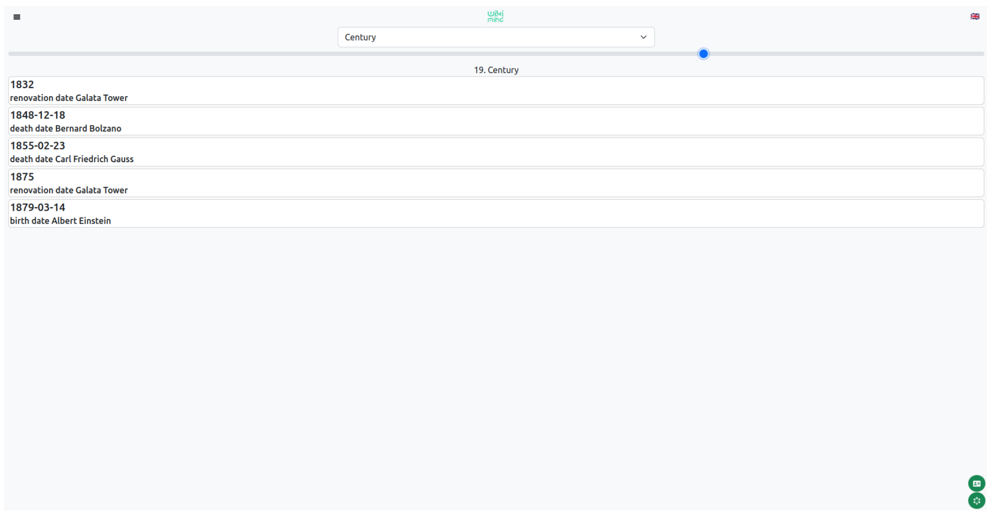
4. **Add connection** We can add connections between nodes. When clicked, the current selected one will be darkened, and when clicked on any other one, a connection will be created between them **Color palette** We change the color of nodes.

5. **Cancel** Deactivate the selected node.

6. **Trash** Delete the selected node.

In Canvas, we can move individual nodes.

Let us go back to the timeline option.



19. Century

1832
renovation date Galata Tower

1848-12-18
death date Bernard Bolzano

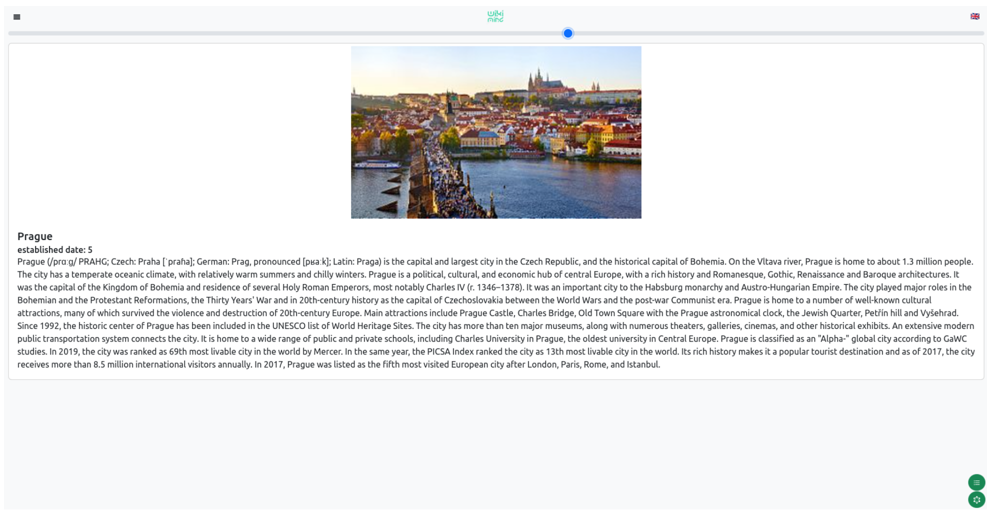
1855-02-23
death date Carl Friedrich Gauss

1875
renovation date Galata Tower

1879-03-14
birth date Albert Einstein

Figure 7.10: Wikimind - Timeline - list

When we switch to it, we first see all the timestamps that can be retrieved from the DBPedia for the entities in the mind map. We can see the events as a list aggregated according to the selected period and specific period, which we can change by scrolling as in Figure 7.10.

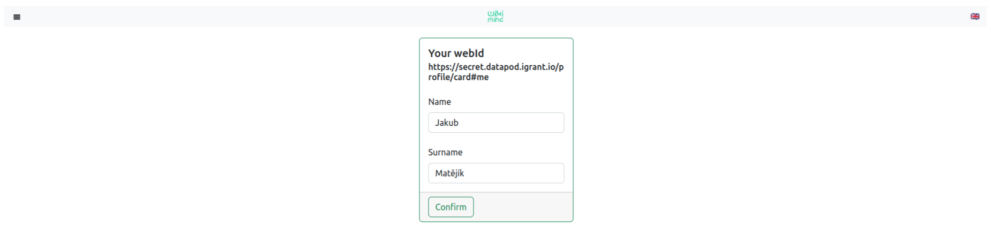


**Prague**
**established date: 5**
Prague (/prɑːɡ/ PRAHG; Czech: Praha [ˈpraɦa]; German: Prag, pronounced [pʁaːk]; Latin: Praga) is the capital and largest city in the Czech Republic, and the historical capital of Bohemia. On the Vltava river, Prague is home to about 1.3 million people. The city has a temperate oceanic climate, with relatively warm summers and chilly winters. Prague is a political, cultural, and economic hub of central Europe, with a rich history and Romanesque, Gothic, Renaissance and Baroque architectures. It was the capital of the Kingdom of Bohemia and residence of several Holy Roman Emperors, most notably Charles IV (r. 1346–1378). It was an important city to the Habsburg monarchy and Austro-Hungarian Empire. The city played major roles in the Bohemian and the Protestant Reformations, the Thirty Years' War and in 20th-century history as the capital of Czechoslovakia between the World Wars and the post-war Communist era. Prague is home to a number of well-known cultural attractions, many of which survived the violence and destruction of 20th-century Europe. Main attractions include Prague Castle, Charles Bridge, Old Town Square with the Prague astronomical clock, the Jewish Quarter, Petřín hill and Vyšehrad. Since 1992, the historic center of Prague has been included in the UNESCO list of World Heritage Sites. The city has more than ten major museums, along with numerous theaters, galleries, cinemas, and other historical exhibits. An extensive modern public transportation system connects the city. It is home to a wide range of public and private schools, including Charles University in Prague, the oldest university in Central Europe. Prague is classified as an "Alpha-" global city according to GaWC studies. In 2019, the city was ranked as 69th most livable city in the world by Mercer. In the same year, the PICSA Index ranked the city as 13th most livable city in the world. Its rich history makes it a popular tourist destination and as of 2017, the city receives more than 8.5 million international visitors annually. In 2017, Prague was listed as the fifth most visited European city after London, Paris, Rome, and Istanbul.

Figure 7.11: Wikimind - Timeline - cards

We can also click on the Card button in the timeline. Then the events will be displayed as cards in chronological order. We can always see the name of the entity, the event only in English, and the description of the entity, in which the event may appear as in Figure 7.11.

### 7.2.5 Profile



**Your webId**
**https://secret.datapod.igrant.io/profile/card#me**

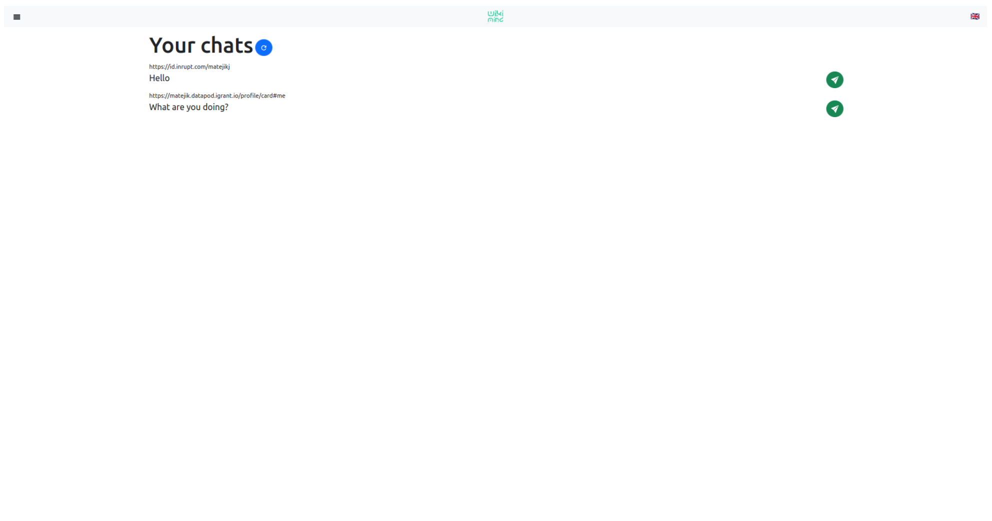
Name: Jakub

Surname: Matějík

Figure 7.12: Wikimind - Profile

If you select Profile from the menu, you will be taken to the user data editor. You can change your first and last name and then click Confirm as we can see in Figure 7.12.

### 7.2.6 Chats



Your chats

https://id.inrupt.com/matejikj

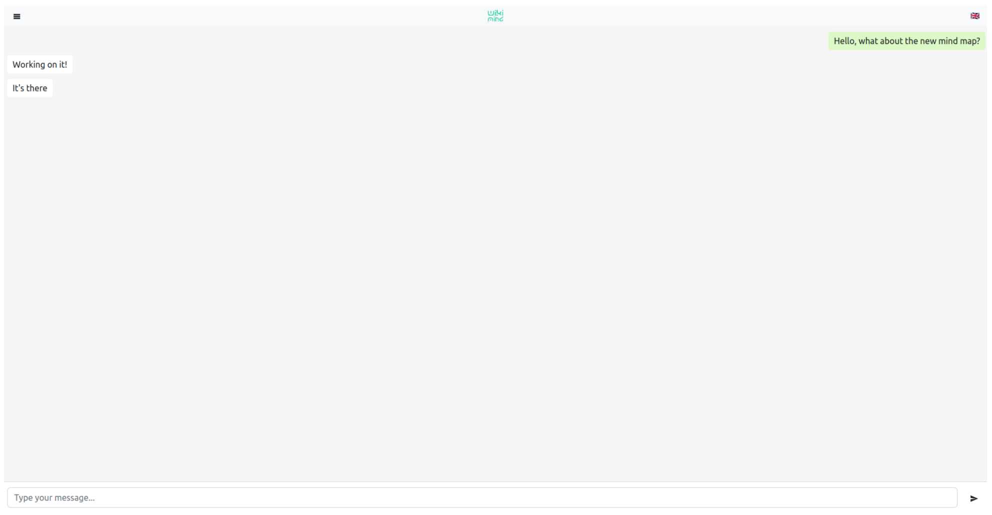
Hello

https://matejik.datapod.igrant.io/profile/card#me

What are you doing?

Figure 7.13: Wikimind - User chats

In the Figure 7.13 you can see the list of chats. There we can select a private chat with the user we want to send a message to.



Hello, what about the new mind map?

Working on it!

It's there

Figure 7.14: Wikimind - Private chat

Here we have a private chat in Figure 7.14 with the user. Below we can insert a message, and after clicking on the send icon, the message will be sent. If another user sends us a message, it will be displayed thanks to Solid notifications, without having to update the page.

### 7.2.7 Classes

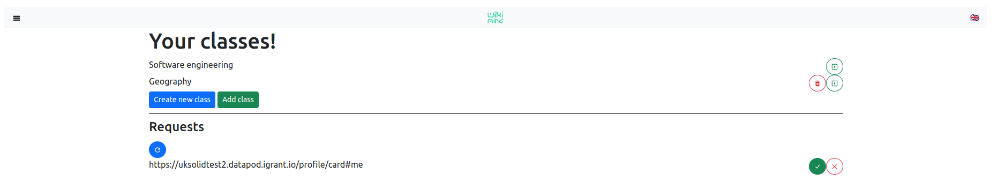


Figure 7.15: Wikimind - Class list

In Figure 7.15 we can see a list of all classes in which we are teachers or students. To enter a class, click the button to the left of the class name.

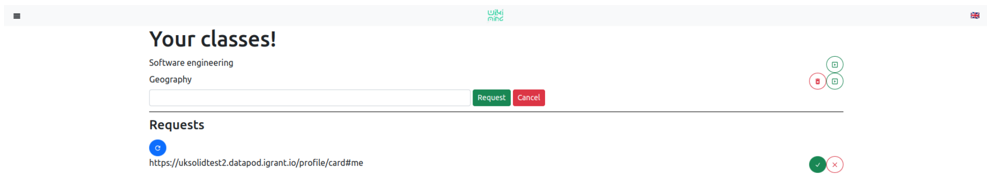
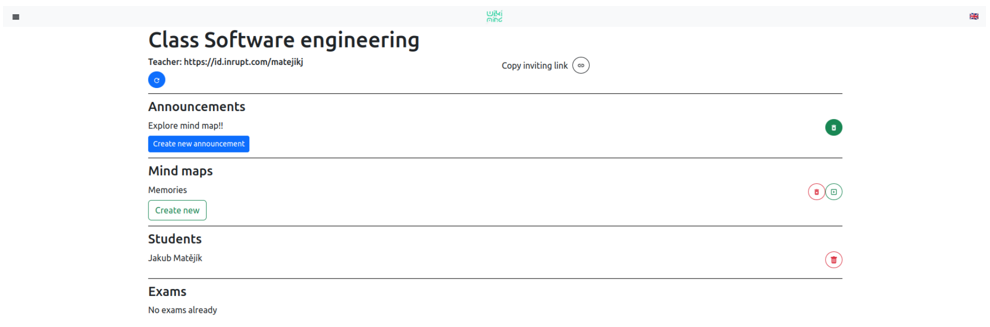


Figure 7.16: Wikimind - Class list requests

At the bottom of the class list page in Figure 7.16 we can see the list of requirements. There we can either accept the request to add the student to our class or reject it.
On the page we can see two buttons. One is to add a new class and the other is to create a request to add another teacher to the class.


Class Software engineering

Teacher: https://id.inrupt.com/matejikj

Copy inviting link

Announcements

Explore mind map!!

Create new announcement

Mind maps

Memories

Create new

Students

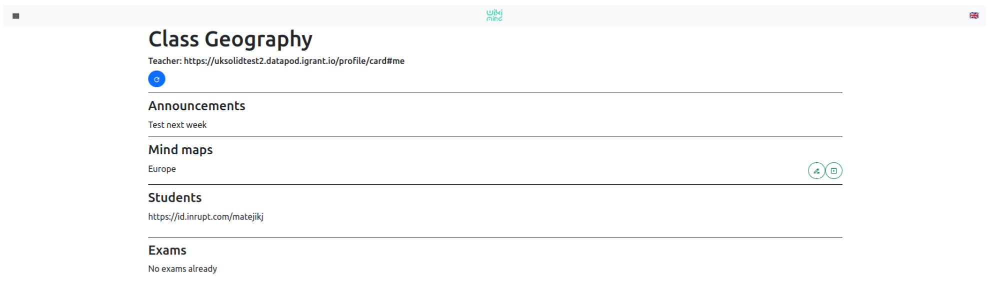
Jakub Matějík

Exams

No exams already

Figure 7.17: Wikimind - Class as a teacher

In Figure 7.17 we have shown the class from the teacher's point of view. In the upper part, we have a button to copy the link to the class. We can send it to students to request membership in our class.
Below that is the Refresh button for the actualization of the class page. Then we have a list of class announcements below. As a teacher, I can add new announcements.
Below that is a list of mind maps. The teacher can also create a new class mind map. In the editor, choose which entity will be hidden in the test.
Below is a list of students and list of their test results.


Class Geography

Teacher: https://uksolidtest2.datapod.igrant.io/profile/card#me

Announcements

Test next week

Mind maps

Europe

Students

https://id.inrupt.com/matejikj

Exams

No exams already

Figure 7.18: Wikimind - Class as a student

We still have to show the student's point of view. We can see it in the Figure 7.18. We can see that the student needs help to copy the link and create an announcement. He can only see the list of other students and the results.

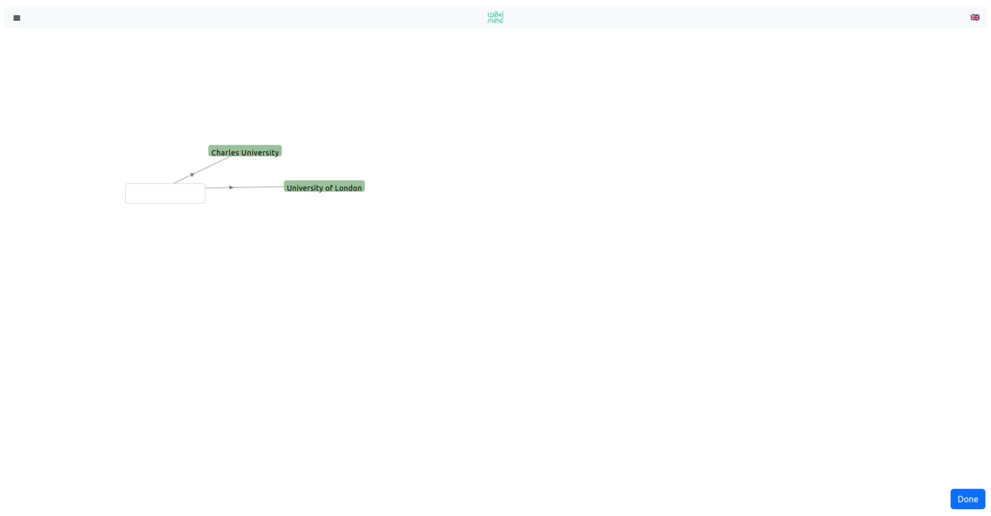


Figure 7.19: Wikimind - Test

Finally, here we have the test display. We can see in Figure 7.19 that we have some entities hidden. Student fill the empty boxes and press done.

# Conclusion

In this work we introduced a few concepts from Linked Data to get acquainted with the issue. Then we defined the exact requirements for the application and created use cases based on them.
One of the following tasks was to define the requirements that we expect from the knowledge base. We presented suitable candidates and chose the one. Since Solid technology is still relatively new and applications that would use it are scarce, it was necessary to analyze how to build our application on this technology carefully. Specifically, the distribution of data between users because each user has his data stored on his Solid pod.
The result of this analysis and the design of the solution is an application that allows the user to create mind maps. Using the links from the knowledge database for the mind map creation is possible. The user can explore the given topic and all its possible links and discover new information that he did not know before. Users can manage its materials on their Solid pod. In addition to such storage with an editor, he can create his class to add students and share mind maps with them. The application also allows messaging over Solid technology.
In order to make this application more useful later, this work also includes programming documentation, which can make it easier to build on this work or introduce other programmers to a model of how to design an application on the Solid platform. This work can serve someone who wants to get acquainted with Solid for the first time.
After this summary of information, I would like to present my subjective opinions on Solid technology and especially my feelings about working with this technology, because it is not quite common yet.
What surprised me the most was that there was quite a problem with the ambiguous implementation of the Solid server specification during the show. During the development, I encountered the most problems with CSS. Inrupt is a private company that develops its own, not open-source, server. Of course, they have to follow the given Solid specifications. However, they set new specifications by their libraries, assuming particular RDF objects in the description of the user's Solid pod. E.g., pim:storage problem, where getting the Url of a Solid pod depends on pim:storage, the NSS server implementation already fulfils this, but CSS does not.
Getting into this platform initially took much work compared to other technologies. Not because Solid is complicated, but the problem is the lack of material to explore. The platform is still relatively new, and basically, the only place to meet the community is forum. Most questions are answered by people from the team developing libraries or servers for this technology, which is excellent for communicating with a group of enthusiasts. Additionally, they are happy to communicate with people with similar interests and have no problem giving time for your questions or evaluating your solutions.
What I like a lot about this technology is its usability for creating applications from the perspective of a young programmer and the initial investment. If I have some cool idea for a certain application that does not need a central server but works with user data, I do not have to pay for any computing capacity. I do not

need to store user data, they have it on their Solid Pods, and I am only interested in the application working correctly. I create the application and then deploy it; other users can immediately start discovering it.

# Bibliography

[1] Krisztian Balog. “Populating Knowledge Bases”. In: Oct. 2018, pp. 189–222. ISBN: 978-3-319-93933-9. DOI: 10.1007/978-3-319-93935-3_6.

[2] Dave Beckett. *Turtle - Terse RDF Triple Language*. Tech. rep. W3C Recommendation. Mar. 2013. URL: https://www.w3.org/TR/turtle/.

[3] Tim Berners-Lee. *Linked Data*. World Wide Web Consortium (W3C). 2006. URL: https://www.w3.org/DesignIssues/LinkedData.html.

[4] Ames Bielenberg et al. “The growth of Diaspora - A decentralized online social network in the wild”. In: *2012 Proceedings IEEE INFOCOM Workshops*. 2012, pp. 13–18. DOI: 10.1109/INFCOMW.2012.6193476.

[5] AWS Mobile Blog. “Complete Guide to Full-Stack CI/CD Workflows with AWS Amplify”. In: *AWS Mobile Blog* (Month of publication Year of publication). Accessed on [Insert Access Date]. URL: https://aws.amazon.com/blogs/mobile/complete-guide-to-full-stack-ci-cd-workflows-with-aws-amplify/.

[6] Martin Bruland. “Evaluating Solid for Social Applications with Many Users”. Master thesis. The University of Bergen, June 2022. URL: https://hdl.handle.net/11250/3002311.

[7] Myong-Hun Chang. “Decentralization”. In: (Oct. 2006).

[8] DBpedia. *DBpedia Lookup*. Accessed on: July 20, 2023.

[9] Omar Al-Debagy and Peter Martinek. “A Comparative Review of Microservices and Monolithic Architectures”. In: Nov. 2018, pp. 000149–000154. DOI: 10.1109/CINTI.2018.8928192.

[10] diginomica.com. “Could Sir Tim Berners-Lee One Day Unite Europe with a Shared Data Platform”. In: *Diginomica* (2023). URL: https://diginomica.com/could-sir-tim-berners-lee-one-day-unite-europe-shared-data-platform.

[11] Eric Prud’hommeaux and Andy Seaborne. *SPARQL 1.1 Query Language*. Tech. rep. W3C Recommendation. Mar. 2013. URL: https://www.w3.org/TR/sparql11-query/.

[12] Lorenzo Gabrielli et al. “Challenges in the Decentralised Web: The Mastodon Case”. In: *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*. ACM. 2019, pp. 89–94.

[13] *Inrupt JavaScript Client Libraries - Authentication*. https://docs.inrupt.com/developer-tools/javascript/client-libraries/authentication/. Accessed: ¡accessed-date¿.

[14] *Inrupt Solid JavaScript Client Libraries Documentation*. https://docs.inrupt.com/developer-tools/api/javascript/solid-client/. Accessed: ¡accessed-date¿.

[15] *Inrupt Solid JavaScript Client Libraries: Access Control Documentation*. https://docs.inrupt.com/developer-tools/javascript/client-libraries/access-control/. Accessed: ¡accessed-date¿.

[16] *IPFS.tech.* https://ipfs.tech/. Accessed on [Insert Access Date].

[17] Shiona McCallum. "Meta settles Cambridge Analytica scandal case for 725m". In: *BBC News* (2022). URL: https://www.bbc.com/news/technology-64075067.

[18] Author Name. "Title of the Article". In: *Computer Weekly* (2023). URL: https://www.computerweekly.com/news/252506983/UK-government-turns-to-Tim-Berners-Lee-startup-for-digital-identity-plan.

[19] Resilio. *What's the Difference Between Peer-to-Peer and Client-Server?* Resilio. URL: https://www.resilio.com/blog/whats-the-difference-between-peer-to-peer-and-client-server.

[20] Michal Richter. "Application supporting re-decentralization of the Web: Photo manager". Czech Technical University in Prague, 2020. URL: https://dspace.cvut.cz/handle/10467/82580.

[21] Kurt Rohloff et al. "An Evaluation of Triple-Store Technologies for Large Data Stores". In: vol. 4806. Nov. 2007, pp. 1105–1114. ISBN: 978-3-540-76889-0. DOI: 10.1007/978-3-540-76890-6_38.

[22] Špomocník RVP. *CSI O ICT VE ŠKOLÁCH: Zajišťování nedostatečné počítače, zastaralé přípojení, omezené situace kritická.* URL: https://spomocnik.rvp.cz/clanek/21625/CSI-O-ICT-VE-SKOLACH-ZAJISTENI-NEDOSTATECNE-POCITACE-ZASTARALE-PRIPOJENI-OMEZENE-SITUACE-KRITICKA.html.

[23] Andrei Vlad Sambra et al. "Solid : A Platform for Decentralized Social Applications Based on Linked Data". In: 2016.

[24] *Solid OpenID Connect (OIDC) Specification.* https://solidproject.org/TR/oidc. Accessed: ¡accessed-date¿.

[25] Solid Project. *Solid Project.* URL: https://solidproject.org.

[26] TechTarget. *Resource Description Framework (RDF).* TechTarget. URL: https://www.techtarget.com/searchapparchitecture/definition/Resource-Description-Framework-RDF.

[27] Usability.gov. *System Usability Scale.* https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. Accessed: [Insert Date].

[28] Vue.js. *Vue.js Documentation.* Accessed: [Insert Date]. Year of publication. URL: https://vuejs.org/.

[29] W3C Wiki. *WebID.* URL: https://www.w3.org/wiki/WebID.

[30] Wikidata. *Wikidata: Main Page.* Wikimedia Foundation. URL: https://www.wikidata.org/wiki/Wikidata:Main_Page (visited on 07/08/2023).

# List of Figures

# List of Tables

# List of Abbreviations

**WebId** - WebID is a URI that refers to a person.
**RDF** - Resource Description Framework.
**Wikimind** - We use Wikimind as the name of our vocabulary and the application's name.

# A. Attachments

## A.1 Wikimind

The main attachment to this thesis is file wikimind.zip, containing a client app that was implemented for this thesis.
This attachment contains all the source code for the application in the src folder. There is also documentation for all the classes in the docs folder. Last but not least there is also the wikimind.ttl dictionary.