

On My Disk integration: Milestone 2, Deliverable 1

Porting existing PeARS pipeline to On My Disk

The On My Disk and PeARS teams

This technical report is the first of a series to be published in the context of the NGI Search Project ‘On My Disk: search integration’. The goal of the project is to add search capabilities to the On My Disk private cloud solution by integrating the open-source PeARS search engine to the existing codebase. The present document provides a list of requirements for this integration, focusing on the challenges of porting a Web search system to a file storage domain. It does not cover aspects related to the training evaluation of the underlying machine learning models, which will be discussed in a separate document.

Introduction

On my Disk¹ (henceforth OMD) is a personal cloud storage system which lets users host files locally on their private device, while also providing secure access to selected third parties. It is addressed to the growing number of individuals and enterprises who are concerned about the privacy of their data.

PeARS² is a search engine that can be run locally inside a browser. It allows individuals to index and search Web pages, and to share their indexed data with others, bypassing centralized search engines. It is addressed to privacy-conscious users who wish to keep their search queries private and avoid bubbling or inappropriate advertisement.

In the context of the NGI Search programme³, OMD and PeARS are partnering to provide a decentralized search system over personal cloud data. The joint project will provide OMD users with a way to search documents seamlessly across networked devices. The cooperation distinguishes itself from the environmentally-damaging energy consumption of other existing solutions, such as blockchain-based systems. Both the OMD and PeARS code stacks have been tested on single board computers and their respective teams are actively researching green networking solutions.

The integration of PeARS into OMD brings with it several technical challenges. In particular, the type of data stored by OMD customers is expected to differ significantly from standard Web documents and call for different indexing and search processes. The purpose of this document is to find out how much adaptation will be required to port the PeARS pipeline to OMD, and to highlight any additional requirement for the integration of the search functionality to the existing OMD codebase.

Architecture

The OMD and PeARS codebases are to be run in separate Docker containers, with PeARS providing an API for indexing and search. OMD supplies cloud storage functionalities which make specific demands on the behaviour of an associated search engine. In the following sections,

¹ <https://onmydisk.com/>

² <https://pearsproject.org/>

³ <https://www.ngi.eu/ngi-projects/ngi-search/>

```

<omd_index>
  <doc url='sample2.txt' contentType='text/plain'>
    <title>Letter to grandma</title>
    <description>Telling grandma about the roses.</description>
  </doc>
  <doc url='sample3.txt' contentType='text/plain'>
    <title>Theory of Everything</title>
    <description>The 347th draft of my theory of everything.</description>
  </doc>
  <doc url='sample4.txt' contentType='text/plain'>
    <title>Tolstoi</title>
  </doc>
  <doc url='Cornerstone_BobMarley.mp3' contentType='audio/mpeg'>
    <title>Corner stone</title>
    <artist>Bob Marley</artist>
    <album>Corner stone album</album>
  </doc>
</omd_index>

```

Figure 1

An automatically generated XML file listing the documents to index in a particular user directory. This directory contains three .txt files and a music file, with different meta-data information.


we describe in more details the modifications that must be made to the PeARS system to satisfy such demands.

Requirement 1: API for indexing and search

Currently, a call to PeARS's search function returns a results page on a Flask interface. For the OMD use case, we require results in json format. The indexer itself is not API-ready. Requirements are as follows:

- OMD should be able to call the PeARS indexer, supplying it with the URL of a user's 'home' directory, that PeARS will recursively crawl.
- When calling PeARS, OMD will automatically generate XML files in each of the user's folders, listing the files that should be indexed in the course of the PeARS crawl.
- Upon receiving the OMD call, PeARS will look for the XML file in the provided directory and start its crawl, parsing each new XML it finds to identify the files to be indexed.

A toy example of an XML file is shown in Fig. 1

 **R1:** express PeARS results in .json format. Implement functions for recursive indexing of OMD content. Provide suitable API endpoints for search and indexing.

Requirement 2: Processing different data types

The current version of PeARS was built for the Web and only caters for indexing / searching .html and .pdf files. The OMD users store many different file types, including images and videos. We have two main requirements:

- Convert various text formats (e.g. .pdf, .docx, .rtf) to .txt for indexing.
- Use the meta-data associated with non-text files to provide indexable information in .txt format.

Here below is an example of the way the meta-data of a music file could be converted into text-based information in the OMD-generated XML files:

```
<omd_index>
  <doc url='Cornerstone_BobMarley.mp3' contentType='audio/mpeg'>
    <title>Corner stone</title>
    <artist>Bob Marley</artist>
    <album>Corner stone album</album>
  </doc>
</omd_index>
```

When encountering this information in the XML, PeARS turns the URL into a pseudo-document containing the information in the title, artist and album tags, and indexes it as if it were a text document.

☞ **R2:** OMD to provide a dedicated XML format containing meta-data on the content of a user's directory. PeARS to implement crawl function and indexing for different file types.

Requirement 3: Catering for file modifications

Another difference between Web search and the present setting is that OMD users will be performing standard modifications to their files and folders: creating, modifying, moving, renaming, deleting. Each of these operations should be associated with an appropriate function in the PeARS indexer.

We recall that PeARS stores its index as a combination of a database and Python numpy matrices. For each document, the database records URL, title and snippet of the document, as well as a thematic 'pod' name and an identifier pointing at a position in that pod's numpy matrix. The matrix stores the mathematical representation of the document, as generated by the indexer. Each row in a matrix corresponds to a single document in the index.

When an operation is performed on a file, both the database and the matrices may require an update. For instance, moving or renaming will require an update in the URL and/or title fields of the database, while deletion necessitates a complex operation to remove a row in the relevant pod's matrix, and an update of all database-to-matrix pointers.

☞ **R3:** PeARS to provide dedicated functions for file manipulation, callable over the API.

Requirement 4: Securing the system

The current PeARS system is meant to be run locally and therefore not adapted to the use case where it provides an API over a network. In particular, we must ensure that a malicious actor cannot use the PeARS search endpoint to discover the private files stored on a user's device. On the other hand, we should make public files discoverable, resulting in two main search modes: 'anonymous' (the user should only be able to search public files) and 'signed in' (the user can search both public files and their own devices).

We show a proposed authorization flow between OMD and PeARS in Fig. 2.

☞ **R4:** PeARS to implement anonymous and signed-in modes. OMD to provide API returning user info and session tokens to PeARS.

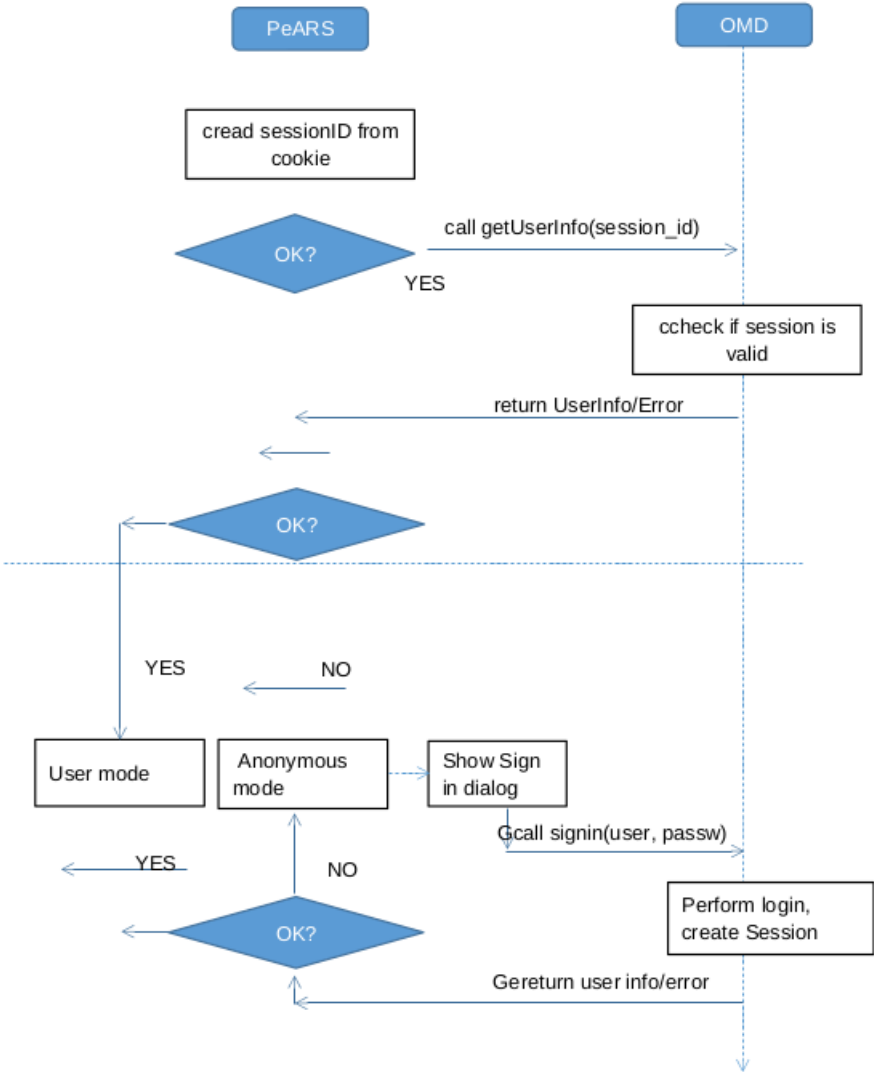


Figure 2
The authorization flow between OMD and PeARS

