Uppsala University
Operating Systems and Process-Oriented Programming

Juveler

A multi-platform media server

Group: Castafiore

Johan Andersson Östling, Kalle Nordgren, Isak Drevstad,
Josef Tiljander, Arvid Lorén, Edvin Illes, Simon Pislar

April-May 2023

Abstract

Today most media is stored digitally and on multiple devices. The problem of organizing, accessing, and synchronizing digital media from multiple different devices quickly becomes apparent once you attempt to do so using the built-in functionality of most of these devices.
Juveler aims to solve this problem as well as provide playback systems for the media-types it supports, which is currently only PDFs. Juveler provides a model for the users libraries, which can be stored locally or on a client provided server, and then accessed through the GUI (Graphical User Interface) to be played on Juveler or alternatively, a suitable media player. SurrealDB is used as the database, as well as for providing server functionality. The Tokio runtime library is used to support asynchronous functions, in order to run the parts of the system concurrently and in parallel.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem Description . . . 4

2 Related Work **4**
2.1 Media server . . . 4
2.2 eBook Managers . . . 5
2.3 Audio-book Manager . . . 6

3 User Interface **6**
3.1 Home view . . . 6
3.2 Library view . . . 7
3.3 Reader view . . . 8

4 Technical Background **8**
4.1 Rust . . . 8
4.2 Databases . . . 9
4.3 Async/Await . . . 9
4.4 Client-server . . . 10

5 System Architecture **10**
5.1 GUI (Graphical User Interface) . . . 12
5.2 Client . . . 13
5.3 Library . . . 13
5.4 Server . . . 13
5.5 Database Wrapper . . . 13
5.6 File I/O . . . 14

6 Concurrency Model **14**
6.1 Async/Await . . . 14
6.2 Threading . . . 15

7 Ethical Aspects **15**

8 Conclusion **15**

9 Summary **16**

10 Possible Continuations **16**
10.1 Filetypes 17
10.2 Mobile Support 17

1 Introduction

The product described in this paper is called Juveler. It is designed to provide a service and media server with which users can upload and download different types of multimedia, such as videos, audiobooks or pictures. This is structured as a backend written in Rust using the Tokio [22] library to perform and handle network requests. Since all the resources on the server needs to be stored, a database called SurrealDB is also used[20]. The user will then, either through a CLI (Command Line Interface) or GUI (Graphical User Interface), be able to play e.g. a video or listen to an audio track. This system offers versatility, modularization and feature-potential, with a wide base to expand upon. The frontend is written in Rust as well, therefore sharing of modules between the server and the client makes development easier and also implementation of shared functionality easier.

The purpose of creating a self hosted media server is to allow users to have freedom and control over their own media files. As discussed below, there are other systems that provide similar functionality to this one. The reason why this product is unique is that it gives the user freedom to store most of the commonly used file types, which is rare in systems of this kind.

1.1 Background

The written word and its medium has been a fundamental part of human civilization, with the earliest known occurrence widely considered to be the Epic of Gilgamesh, written on clay tablets. Since then the medium of the written word has gone through a number of radical transformations, with significant societal implications and its fair share of critics [13].

The 20th century saw the mass adoption of audio- and video based mediums, which coincided with a large decline in book reading in the later part of the 20th century and the early part of the 21th century. Between 1978 and 2002, the share of Americans who hasn't read a book during the past year went from 8% to 18% [23]. This decline is even more apparent from a generational perspective. In 2022, the oldest generation, the boomers [3] read an average of 9.54 books per year, while the youngest, Gen Z read an average of 3.52 books [17].

The normative merits of this transformation has been, and is widely debated to this day. This debate has been mainly focused on the fundamental merits of the mediums. One underrated aspect is the end users loss of control and autonomy as well fragmentation of the methods of media consumption. While an end user has total control and ownership of a physical book, digital forms of books, music, and visual content are controlled by a few large corporations. Amazon has a near monopoly on eBooks and audiobooks, Netflix and a few other corporations have an oligopoly on movie- and TV-streaming and Spotify has a near monopoly on music streaming [11][7].

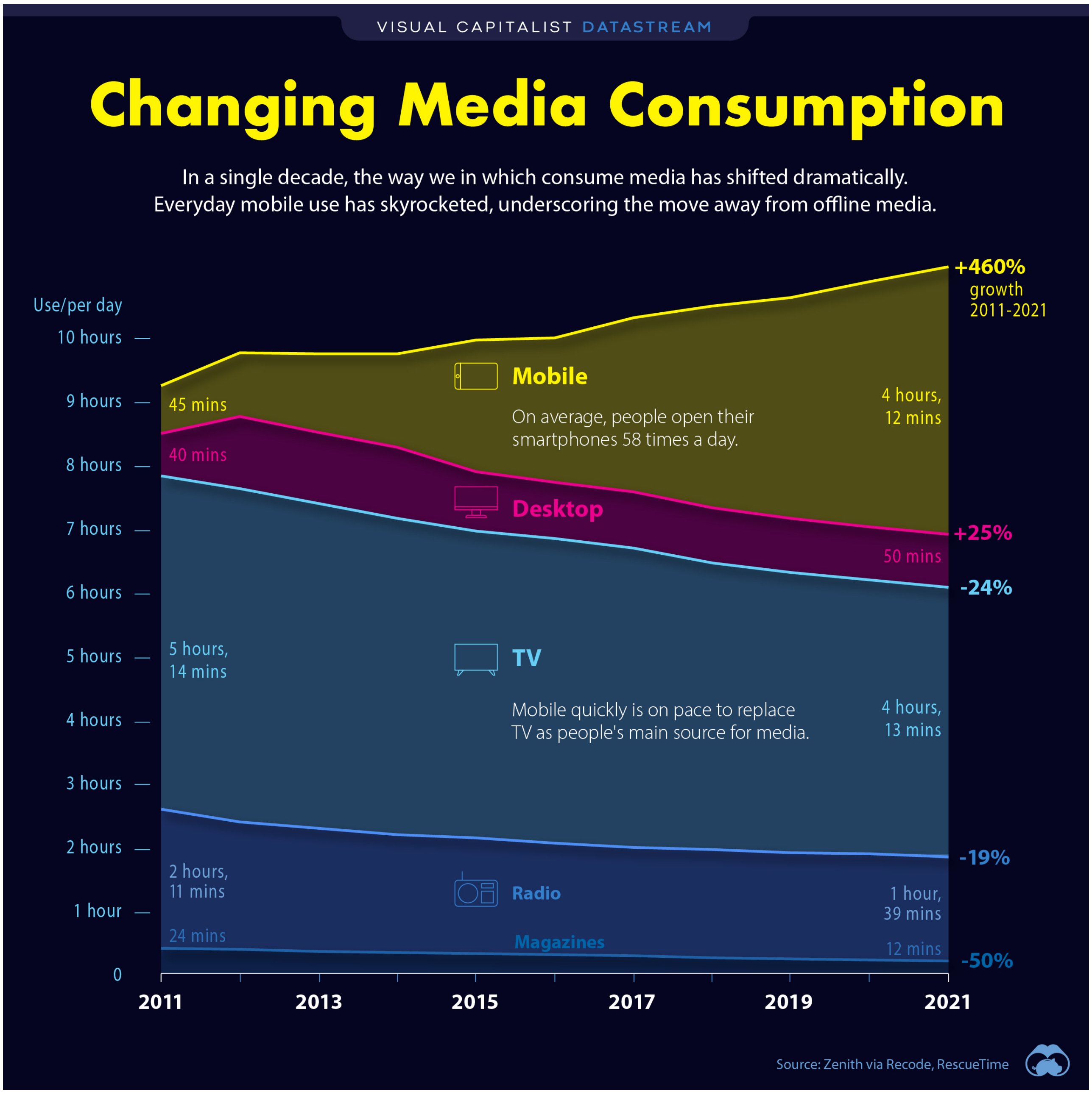


Figure 1: Changing media consumption [1]

As seen in figure 1, media consumption takes up a very large part of the average persons day and that digital consumption is a large and growing part. Thus improving this experience could have a significant positive societal impact.

1.2 Problem Description

This concentration is due in large part to the media ownership model where each company owns the media and then gives access to the end user. Another factor is the large amount of engineering effort required to create media consumption solutions, in combination with the low marginal cost associated with serving media.

A separation of providing media and the method of consuming it would have the potential effect to greatly alleviate this concentration and fragmentation. An end user could have one system for consuming media, instead of several today. This would greatly lessen the end users dependence on individual corporations, and enable smaller media producers to compete since they wouldn't have to spend the enormous effort to create solution to consume the media.

To compete with the established solutions the system would have to support a large number of media formats, be compatible with a large number of platforms and be able to sync consumption progress between all the end users devices. Since all the users media would be on one system, the system should have excellent features for organizing media, such as user defined tags, links and folders.

2 Related Work

There are of course existing services that provides users with the ability to store and play media of their own choice. Three similar products are Plex, Jellyfin and Calibre.

2.1 Media server

Plex is a comprehensive media player system built on a client-server model. At the core of Plex is the Plex Media Server (PMS), which takes charge of storing, organizing, and streaming media content. Users can access their media library through the Plex App, a playback client application available on multiple platforms such as Windows and macOS. While Plex primarily focuses on visual media like movies and TV series, it also offers support for music albums and audiobooks, catering to a broader range of multimedia consumption [16].

One notable aspect of Plex is its standard library of media, which is freely available to all users, regardless of whether they contribute their own content to the library or not. This means that users have a collection of content readily accessible for enjoyment, even if they haven't personally added any files to the library. This feature ensures that users can begin exploring and consuming media right away, without the need to curate their own library from scratch. It provides a convenient starting point for users who may not have their own content readily available or simply want to discover new media within the Plex ecosystem [14].

Jellyfin positions itself as "The Free Software Media System" and is an open-source project that operates under the GNU General Public License v2.0. This license grants individuals the freedom to modify, distribute, use privately, and even utilize the software for commercial purposes. In essence, Jellyfin offers a viable alternative to its proprietary counterpart, Plex. By deploying Jellyfin's server on one's system, users gain the ability to collect, manage, and stream media content. The software is a web application, which ensures comprehensive cross-platform compatibility, allowing users to enjoy its features on various operating systems [8].

One of the standout advantages of Jellyfin being an open-source solution is its inherent flexibility. Users have the freedom to enhance and expand the system by adding new features, thereby tailoring it to their specific requirements and preferences. This fosters a collaborative and innovative community that actively contributes to the continuous improvement of the software [8].

2.2 eBook Managers

Calibre is a comprehensive eBook manager that is available as a free and open-source software. It offers extensive support for a wide range of file formats commonly used for eBooks, such as epub and PDF files. One of the key features of Calibre is its ability to download and retrieve metadata [10] for eBooks. This includes information like author, title, cover image, publication date, and more. Users can easily access and modify this metadata, allowing them to organize their eBook collection effectively. Furthermore, Calibre provides options to create new metadata for eBooks, enabling users to personalize and customize the information associated with their books [4].

Calibre also boasts advanced search and filter functions, which enhance the users ability to locate specific eBooks within their collection. Users can leverage custom metadata fields to add additional tags or labels to their eBooks, making it easier to categorize and find content based on their preferences and organizational needs [4].

2.3 Audio-book Manager

Audiobookshelf is a self hosted, open-source audio-book manager that stores and displays the users audiobooks. While audiobooks are the main focus of Audiobookshelf, there is also support for eBooks and podcasts. The system is completely free and can be deployed through docker. It is also available on multiple platforms through mobile applications and a web application. When the user is done listening to an audio book, the system synchronizes with the server so that the user can continue listening right where they left off. Uploaded audiobooks are also available on all platforms due to the server making sure all clients are up to date. This gives the user the freedom to seamlessly enjoy their audiobooks no matter where they are or how they access their library [9].

3 User Interface

This section will describe what the user experience will be like for a user operating different parts of the system.

3.1 Home view

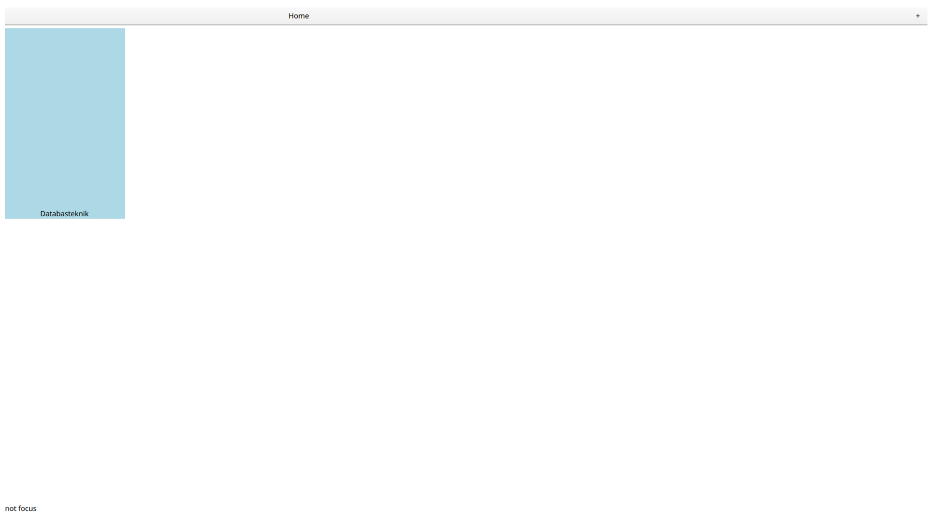


Figure 2: Home view

The program has three different views; home, library and PDF-reader. The default view is the home view (see figure 2), and this is the view shown upon launch. All of the libraries added by the user are to be shown in this view. Users can add a library by clicking on the plus icon in the top right corner and selecting a directory.

The home view is local to the user, meaning that it wont be synced between different computers. This means that user can have libraries from different sources. The application is designed like this because it lets users have a library that is stored locally on their own computer and a another one that is stored remotely on another computer at the same time.

3.2 Library view

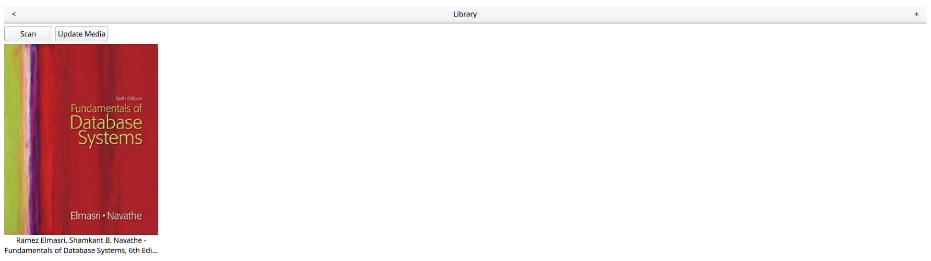
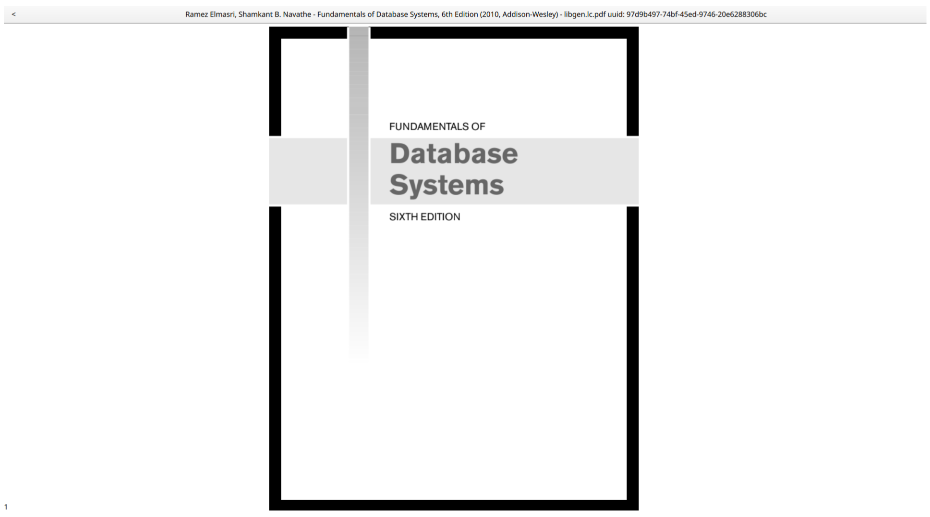


Figure 3: Library view

Users can enter the library view (see figure 3) by clicking on an a library in the home view. Once inside the library view all the files contained inside of the corresponding library are shown. In order to display the files users need to click the `Scan` button, this scans the metadata of the files and loads thumbnails of each file. Users can also refresh the UI to sync with the database by pressing the `Update Media` button.

The contents of each library are synced between different computers. For example, if a user scans the files of a library on one computer they can later press the `Update Media` button on another computer to load the scanned metadata. This is desired because users should be able to access their media files from multiple different locations. For example, the server can be ran on a desktop computer at home and when you are traveling you are able to access your media files from your laptop. This is one of the core features of the application.

3.3 Reader view



<
Ramez Elmasri, Shamkant B. Navathe - Fundamentals of Database Systems, 6th Edition (2010, Addison-Wesley) - libgen.lc.pdf uuid: 97d9b497-74bf-45ed-9746-20e6288306bc
FUNDAMENTALS OF
Database Systems
SIXTH EDITION
1

Figure 4: Reader view

When users click on a PDF within the library view, they can access the reader view (see figure 4). Once inside the reader view, users have the ability to navigate between pages of the PDF, allowing them to jump back and forth.

4 Technical Background

To give a better understanding of how the system is built and why certain choices was made in order to satisfy wanted properties, this section describes different technical aspects of the product.

4.1 Rust

Rust [18] is a modern, systems-level programming language that combines low-level control with high-level convenience and safety. Developed by Mozilla, Rust aims to provide a fast, secure, and reliable alternative to other systems programming languages like C and C++. Rust's main features include strong static typing, memory safety, and zero-cost abstractions, which make it ideal for building efficient and reliable software that runs on a variety of platforms. With its growing popularity and active community, Rust has become a go-to language for many developers working on high-performance systems or safety-critical applications.

4.2 Databases

Databases [5] are structured collections of data that provide reliable and efficient storage and management for large volumes of information. They are essential components in various applications and systems, serving everything from simple data storage needs to complex enterprise solutions.

Different data models are used in databases to define the logical structure and organization of data. The relational model is often used for structured data and relies on tables with rows and columns. Relational databases like MySQL and PostgreSQL follow this model and use SQL as the query language for data manipulation. In contrast, NoSQL databases offer more flexibility to handle unstructured and semi-structured data, allowing for scalability and performance optimization.

One specific type of NoSQL database is the graph database, which focuses on representing and querying data as a graph. Graph databases, such as SurrealDB [20], are particularly well-suited for managing highly interconnected data, such as social networks, recommendation systems, and knowledge graphs. These databases enable efficient traversal and analysis of relationships between data entities, providing powerful insights and query capabilities.

4.3 Async/Await

Async and await [2]are keywords used to implement asynchronous code for Rust. Async is used to define that a code is asynchronous and are going to return something in the future. Await is then used to pause the execution until the awaited operation completes, returning a Future which represents an ongoing computation that will eventually produce a value. To use async and await, an asynchronous runtime is needed.

4.4 Client-server

Client-server is a computer network architecture in which many remote computers, called clients, request and receive service from a centralized server - a host computer [6]. Client computers provide an interface for users to request services from the server and to display the result the server returns. The client-server model is suitable for when the client and server each have distinct routine work, as well as for resource sharing. Typically, the server offloads heavier computation from the clients, or manage a database which multiple clients interact with.

5 System Architecture

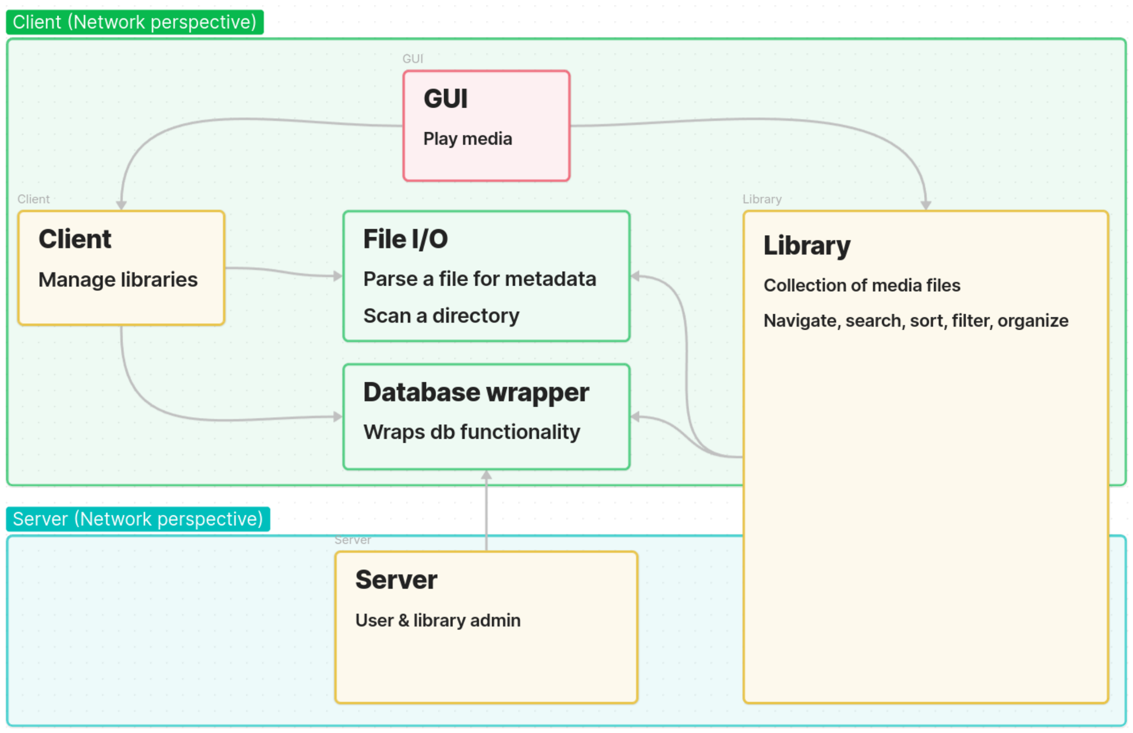


Figure 5: System architecture

The system architecture comprises six components, as illustrated in Figure 5 and 6. The process starts when the user launches the GUI application, which connects to the "Library" database concurrently with the UI. This database contains all the current media files. To upload files to the server, the user can select a local folder that is scanned recursively and parsed by the "File I/O"

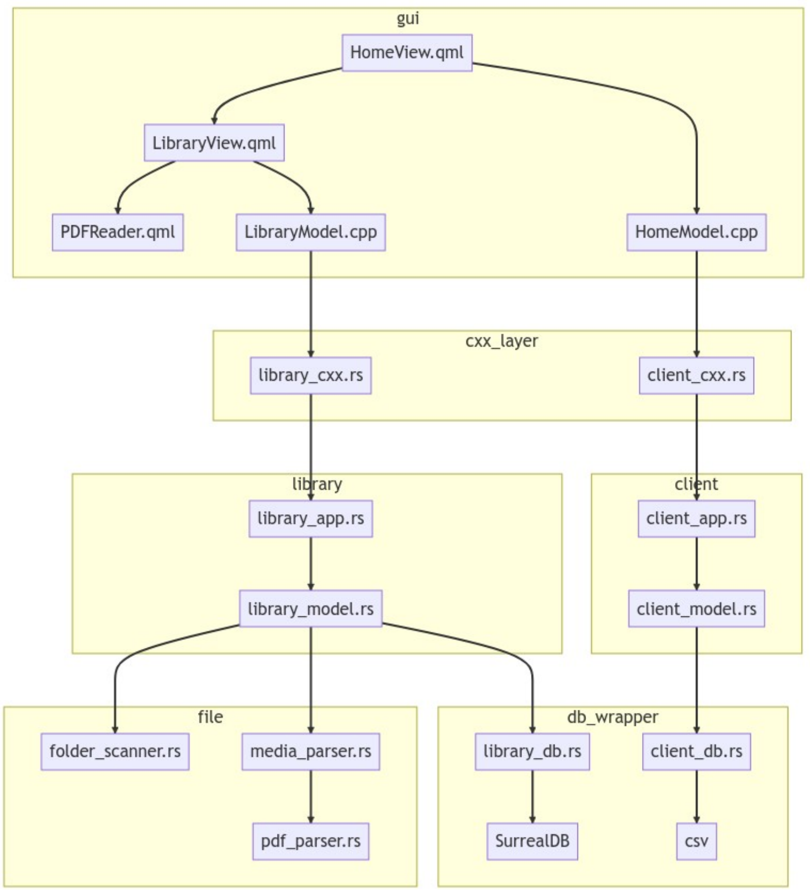


Figure 6: Granular module architecutre

component. The parsed metadata is then inserted into the library database, and the files are uploaded to the server through the "Database Wrapper" component.

Once uploaded, both the metadata and the files are available on the server and can be accessed at any time from the GUI. The user can also download media files hosted by the server, which are searchable and viewable directly from the GUI. These files will be stored temporarily in a directory on the users local machine. If the user wishes to add a file to their library, they can "move" it from the temporary directory to its corresponding directory in the GUI, which will also move the physical file to the appropriate location.

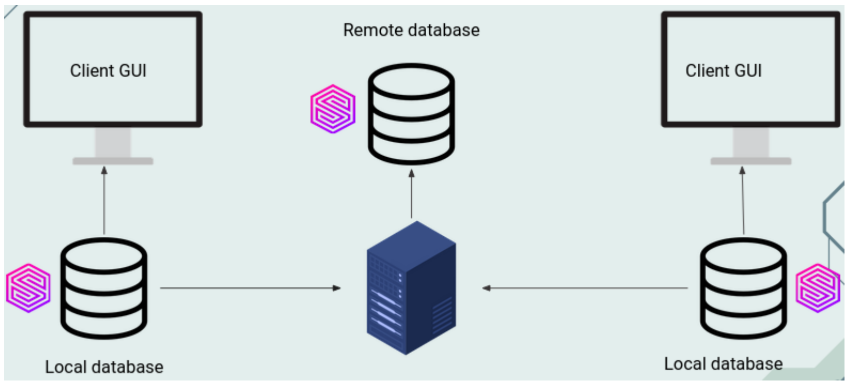


Figure 7: Networking structure

5.1 GUI (Graphical User Interface)

The graphical user interface (GUI) serves as a vital component, facilitating users' interaction with the system. It presents a visually appealing platform where various media options are conveniently accessible. These media options are typically accompanied by thumbnails, such as captivating eBook covers or fancy PDF front pages, enhancing the users browsing experience. Another important goal is to provide a fast, responsive and native-like experience for the user.

Moreover, the GUI should offer the flexibility to upload new media files. Whether it's a favorite book or an important PDF-document, the GUI encourages the user to save their content in the system for easier access.

Having these goals in mind, fast performance, flexibility and accessibility, Juveler came to the conclusion of using the cross-platform application development framework Qt[15].

Qt simplifies the process of developing desktop, mobile, and embedded applications. It offers pre-built components and supports multiple programming languages, including C++, Python, and JavaScript. Qt provides advanced features such as networking, database integration, and multimedia support, and has a comprehensive documentation and support community. With its cross-platform support, Qt enables developers to write code once and deploy it across multiple operating systems, making it a powerful and versatile tool for creating complex applications. However, it's worth noting that Qt does not natively support Rust, which is the language that Juveler is written in. To address this limitation, a crate called CXX [21] is utilized to establish a bridge between C++ and Rust, leveraging bindings to ensure compatibility.

5.2 Client

The client module serves the purpose of managing multiple libraries, offering operations for retrieving and creating libraries. This module interacts with the database wrapper module and the File I/O module by calling their respective functions. It was separated into its own module to enable seamless integration with various interactive user interfaces, such as a TUI (Terminal User Interface), CLI or a GUI.

5.3 Library

A library is a collection of media files. These can be of different supported file types. The files can also be organized in folders. These modules handle the internal operations on a library, which include but is not limited to: searching, adding files, deleting files, editing file metadata, and creating folders and sub-folders to organize those files. This module is also responsible for opening connections to a database represented by the library, and storing these in a hash map so that a server storing multiple users' libraries has quick access times to those libraries.

5.4 Server

This module is responsible for handling the connection to the server. The server will be used to sync the local copies of the database with the database on the server to ensure that the users files are kept safe and available from all locations.

5.5 Database Wrapper

The purpose of this module is to abstract away as much as possible about the database, so that it becomes easy for the other modules to interact with it. Programming in other modules should not be specific to what database is used, instead specific to what suits development.

The database selected to use is SurrealDB. SurrealDB is a new and modern cloud database library written in Rust. The server library supports many programming languages including Rust, Node.js, Deno, Python and Golang and the client side library supports Javascript at the moment. It has a simple

API (Application Programming Interface) with a total of 16 functions making it very trivial and consumer friendly. It provides many features such as relational, on-memory, in-disk and embedded. It has an SQL-style query language called SurrealQL. Juveler primarily leverages two key features offered by SurrealDB. Firstly, the schemaless capability enables the direct insertion of data structures with different fields into a table, eliminating the need for uniform fields/columns as required by traditional databases. This feature significantly reduced both development time and complexity from a systems perspective. Lastly, SurrealDB provides built-in security and authentication mechanisms. This functionality simplifies the process of setting up users, administrators, and access rules, ensuring the protection of database accesses.

5.6 File I/O

File I/O is where interactions with the computer's local files happen. When a file is first uploaded to the system, several pieces of information need to be retrieved, including the actual media file, metadata and the thumbnail which will display it. File I/O takes care of parsing a file to retrieve these things. This is kept separated from the other modules of the system because it is accessed by several other modules. This allows for reuse of code and contributes to modularity.

6 Concurrency Model

This section discusses how the concurrency model is implemented and what tools have been used to achieve this.

6.1 Async/Await

Because the product relies on IO-operations and clients need to make a load of operations to the database independent from each other, the functions needed for these operations need to be concurrent and asynchronous. To utilize asynchronous programming the product uses Tokio [22]. Tokio is an asynchronous run-time for the programming language Rust. Tokio is good for writing network applications but has a wide area of use, including everything from large network-based systems to embedded systems. It is both memory safe and thread safe, meaning that the developer does not have to worry about memory leaks and race conditions. Tokio also has a built in work scheduler that operates with minimal overhead resulting in efficient use of resources. Asynchronous versions of the standard library is included in Tokio, as well as a large ecosystem of other libraries.

6.2 Threading

The system incorporates multithreading by spawning a new thread for each file when scanning new media. This thread handles creating new thumbnails of four different sizes. This multithreading is used to make creating a new file not take too much time when scanning the directories by making it run concurrent.

7 Ethical Aspects

Similar services to Juveler, such as Plex and JellyFin, have gained widespread popularity as platforms for consuming pirated media, referring to media that has been illegally copied or distributed[19]. These services offer essential functionalities for hosting personal streaming services, utilizing private servers with extensive libraries of media files, regardless of their acquisition methods.

The availability of such tools and platforms has contributed to the proliferation of pirated media consumption. However, it is crucial to acknowledge that pirated media poses significant challenges for the original creators.

However, Juveler will not focus on implementing specific anti-piracy measures within its framework. Instead, it places the responsibility on the users to comply with the laws and regulations of their respective countries regarding pirated media. By adhering to legal and ethical standards, users can ensure that their actions align with the appropriate guidelines.

Regarding data security, Juveler takes advantage of the fact that SurrealDB has built-in security for protecting user info rmation and their files.

8 Conclusion

In conclusion, Juveler successfully provides tools for hosting a media server capable of organizing and synchronizing media-files. However, the goal to support many file types was not met, but should be fairly simple to develop. The use of Rust and its asynchronous run-time Tokio has allowed for development of a system that is both memory and thread safe. SurrealDB, a cloud database library written in Rust, provides security and authentication features, making it easy to set up users, admins, and rules to protect database accesses.

9 Summary

Juveler is a media server allowing users to upload and download various types of multimedia files, although it currently only supports PDFs. It utilizes a backend written in Rust using the Tokio library for asynchronous tasks and a database called SurrealDB for storage. Users interact with the system through a Graphical User Interface (GUI) to play videos, listen to audio tracks, and manage their media files. The frontend is developed using Qt together with C++ with the help of CXX, enabling shared functionality between the server and client modules. The purpose of Juveler is to provide users with a self-hosted media server, giving them freedom and control over their own media files.

The system architecture of Juveler consists of several components, including the GUI for user interaction, the client module for handling multiple libraries and file operations, the library module for searching and organizing media, the server module for database synchronization, the database wrapper for abstraction, and the file I/O module for managing local files.

Regarding ethical aspects, similar services like Plex and Jellyfin have been used for consuming pirated media. However, Juveler does not implement anti-piracy measures within its framework. Instead, it emphasizes user responsibility and compliance with relevant laws and regulations concerning pirated media.

In short, Juveler offers a robust solution for hosting and managing a media server, utilizing Rust, Tokio, SurrealDB, and a user-friendly interface. It prioritizes user control over media files and emphasizes legal and ethical user behavior.

10 Possible Continuations

There is multiple features of Juveler that has not been implemented yet. However, due to being heavily modularized, Juveler is easy to expand upon and change when required.

10.1 Filetypes

One possible continuation is to continue developing support for more file-types. This implementation of the system only supports two file types: epub and pdf. Some file types commonly supported by media servers similar to Juveler include: MP3, MP4, AVI, WAV. Supporting these media-file types would require creating new datatypes and a file-parser for each file type, as well as provide appropriate playback functionality.

10.2 Mobile Support

Currently the system is only supported on desktop, with no implemented support for mobile devices. Mobile support is very useful for a media application since that allows you to watch your media files whenever and wherever you want, despite not having access to a desktop computer. However, this continuation is not trivial to implement. Developing an application for mobile would require making a whole new UI and would probably require some adjustments of the back-end. For example, SurrealDB does not support mobile, meaning that the system would either have to find a way to bypass this or change the choice of database entirely.

Bibliography

Cited references

[1] Aran Ali. *How Media Consumption Has Changed Over the Last Decade (2011-2021)*. English. 2021. URL: https://www.visualcapitalist.com/how-media-consumption-has-changed-in-2021//.

[2] *async/.await Primer*. English. 2023. URL: https://rust-lang.github.io/async-book/01_getting_started/04_async_await_primer.html#asyncawait-primer/.

[3] *Boomer, definition*. 2023. URL: https://dictionary.cambridge.org/dictionary/english/boomer.

[4] *Calibre*. English. May 2023. URL: https://calibre-ebook.com/.

[5] *Databases*. English. 2023. URL: https://www.oracle.com/database/what-is-database/.

[6] The editors of Encyclopaedia of Britannica. English. In: May 2023. URL: https://www.britannica.com/technology/client-server-architecture.

[7] Eamonn Forde. *Spotify Comfortably Remains The Biggest Streaming Service Despite Its Market Share Being Eaten Into*. English. 2022. URL: https://www.forbes.com/sites/eamonnforde/2022/01/19/spotify-comfortably-remains-the-biggest-streaming-service-despite-its-market-share-being-eaten-into/.

[8] *Jellyfin*. English. Apr. 2023. URL: https://jellyfin.org/.

[9] Jeremy. *Audiobookshelf - A Self Hosted Audiobook and Podcast Server with Phone Apps*. 2023. URL: https://noted.lol/audiobookshelf-a-self-hosted-audiobook-and-podcast-server-with-phone-apps/.

[10] *Metadata, definition*. 2023. URL: https://www.merriam-webster.com/dictionary/metadata.

[11] Sneha Pandey. *Netflix's Market Share Decline Continues In 2023: Analysis Of Leading Streaming Platforms*. English. 2022. URL: https://www.similarweb.com/blog/insights/media-entertainment-news/streaming-q1-2023/.

[12] Husain Parvez. *Jellyfin Vs Plex - Battle of the Media Managers*. URL: https://www.rapidseedbox.com/blog/jellyfin-vs-plex.

[13] Plato. “Phaedrus”. In: *Complete Works*. Ed. by J. M. Cooper. Original work circa 399-347 BCE. Indianapolis, IN: Hackett, No Specific Year, pp. 551–552.

[14] *Plex*. 2023. URL: https://www.plex.tv/watch-free-register/.

[15] *Qt*. English. 2023. URL: https://www.qt.io/.

[16] Eric Ravenscraft. *What Is Plex?* English. 2023. URL: https://www.pcmag.com/news/what-is-plex.

[17] Nicholas Rizzo. *Over 50% of Americans haven't read a book in the past year*. English. 2022. URL: https://wordsrated.com/american-reading-habits-study/.

[18] *Rust*. English. 2023. URL: https://www.rust-lang.org/.

[19] Bijan Stephen. “Plex makes piracy just another streaming service”. English. In: (May 2023). URL: https://www.theverge.com/2019/7/23/20697751/piracy-plex-netflix-hulu-streaming-wars.

[20] *SurrealDB*. English. 2023. URL: https://surrealdb.com/.

[21] *The CXX Rust Crate*. 2023. URL: https://cxx.rs/.

[22] *Tokio*. English. 2023. URL: https://tokio.rs/.

[23] Jordan Weissmann. *The Decline of the American Book Lover*. English. 2014. URL: https://www.theatlantic.com/business/archive/2014/01/the-decline-of-the-american-book-lover/283222/.