Building a database in Rust

# Project Aims & Objectives

Rust ( <https://www.rust-lang.org/en-US/> ) is a relatively new  
programming language developed by Mozilla. It is intended to allow low  
level programming in a "safe" way -- there should be none of the  
memory errors, undefined behaviour, and race conditions that often  
arise in other low level languages like C and C++.  
  
Rust enables safe low level programming by making the notion of  
"lifetime" explicit in programs. A lifetime tracks the parts of a  
program that have access to a piece of memory, preventing errors such  
as "use after free" and accessing memory on stack frames that have  
been deallocated. Lifetimes also enable race free concurrency.  
  
The objective of this project is to use Rust to implement a simple database  
server in order to gain experience in how Rust's features help or  
hinder safe systems programming.

# Related Work

# Project Specification

# Project Plan w/ Progress Summary

## Part I: Learning Rust

My initial reflex was to find resources for learning Rust in the university library. I was successful since I found the book: [“Beginning Rust” by Carlo Milanesi.](https://books.google.co.uk/books?vid=ISBN9781484234686) I got it as an online resource through the library’s online search engine. For the first couple of weeks, m focus was on reading that book. For my second meeting with my supervisor I had produced a simple String manipulation program. Firstly, the user is asked on the terminal what it is they are wanting to do among the following options:

* Remove a character from a String
* Turn the entire String into uppercase
* Change the case (upper to lower or vice versa) of a single character in the String
* Split the text by some character or sequence of characters

Then, the program would ask the user to input a String and depending on the option that was chosen, it would prompt the user for a character or an option to choose from e.g. when changing case, what the desired case is.  
Lastly, the user would see the output of the operation and they would be asked if they wanted to save the result into a text file (I did this to experiment with file access).  
I mainly made use of Iterators and corresponding functions that work on them e.g. map, fold, collect and closures to make my program.

Over the next couple of weeks, I finished reading the book. The last couple of chapters explained the concept of borrowing and Lifetimes. I was still confused after reading the book, so I looked for more information online. Among what I found, this video stood out: <https://www.youtube.com/watch?v=QoEX-Vu-R6k>  
The video helped me understand better why the usage & implementation of Lifetimes is required. It also helped me understand the syntax better. Unfortunately, at this point I still felt it wasn’t something I would understand properly unless I put it into practice myself.

I decided to put the research into Lifetimes on hold as I had agreed with my supervisor that I would aim to produce a simple server by our next meeting. So, I started learning about the I/O libraries in Rust and then came across Tokio. My supervisor had mentioned previously that I would probably have to make use of this library so I decided to look into it. By following the tutorials in the Tokio documentation I made a simple “Echo” server. This program, when you connect through a client like “Telnet” on Windows will immediately send back anything that is sent to it i.e. if you press “h” on your keyboard, you would receive “h” back immediately so your terminal would display “hh”. This implementation of Echo did not satisfy me, I decided I wanted to implement a version where the sent data would only be returned upon a newline being received i.e. when the user presses the enter key.

After a bit of digging, I found out about the Tokio Codec library which is used to apply certain modifications to data (as I presently understand it). The objects in the library work on the “Streams” (Input) and “Sinks” (Output) objects in Tokio’s I/O library. In the Codecs library one can find the “LinesCodec” object which splits data by using the newline (“\r\n” on Windows) character(s). Thanks to this I could now implement an Echo server that would only send back “h” if one pressed the Enter key. Therefore, unlike my previous version, now a user could type “hello” then the Enter key and get “hello” back.

I was still feeling a bit confused about how Tokio worked, especially when it came to its asynchronous logic. So, in the hopes to get more knowledgeable about Tokio, I came across this video of a lecture from “RustFest” done in Zurich in 2017: <https://www.youtube.com/watch?v=4QZ0-vIIFug>  
I feel like this video helped me learn about the thoughts that came behind the development of Tokio and why it was designed the way it is i.e. what issues arised when asynchronicity was implemented into other languages and how Tokio could be developed while taking those issues into account. For example, when passing “Future” objects in between threads when dealing with concurrency.  
I feel I now understand a bit more about the inner workings of Tokio, which I hope will help me when using the library myself.

At my next meeting with my supervisor I presented to him the progress I had made and the programs I had developed. We agreed that the next step for me to take would be for me to implement a server that keeps the same state between sockets i.e. clients. For example, that one connected client can increase the value of a variable and that another connected client can at the “same” time read the value in this variable and modify it. Clearly, the main challenge of this program is dealing with race conditions.

After some Googling, I came across [a project tutorial](https://doc.rust-lang.org/book/ch20-00-final-project-a-web-server.html) on the Rust Language book website where one builds and HTTP server starting from a simple single-threaded one and building upon it to get a more a more complex multi-threaded one. This interested me and thought I could learn enough from it that I could then transfer the knowledge to my Tokio Echo server.

When I completed the tutorial, I had implemented a simple HTTP server that was multi-threaded, I learned a lot about implementing one’s own Thread Pool and working with Workers that could receive jobs. I also learned about sharing resources within threads by using Atomic Reference Counters (Arc) and Mutexes to be able to lock a variable before modifying it so that race conditions cannot occur. This last piece of knowledge was especially valuable since this is what I could apply to my Rust Echo server.

Using the knowledge I gained, I implemented a counter variable into my “Echo” server using the Arc and Mutex Objects so that when multiple clients connected over a system like “telnet” one could modify the value and another could read it and the changes the former made were reflected in the latter’s terminal. Therefore, I managed to implement a Shared-State variable into my Rust server which would no longer be an “Echo” server but simply a server where a variable can be manipulated by multiple clients.  
For now, the knowledge I gained on thread pools and workers is not required for the Tokio server since Asynchronous logic takes care of handling multiple clients (connections).

## Part II: Learning about Database Design

Now that I felt more confident with my Rust skills, I felt it was time to start thinking about how I’m going to build a Database using Rust.  
I know very little about database design, therefore that is where I decided to research next.

Firstly, I came across [this lecture](https://archive.fosdem.org/2018/schedule/event/rust_distributed_kv_store/) from the 2018 “FOSDEM” event which according to [their website](https://fosdem.org) is a “[…] free event for software developers to meet, share ideas and collaborate.” It is held in Brussels.  
The lecture, given by Siddon Tang talked about using Rust to Build a Distributed Transactional Key-Value Database. Unfortunately, even though this lecture gave me knowledge about what tools and libraries are available to get a database up and running in Rust, it was too high-level in terms of it’s thinking. It seems the lecture is aimed more what a business could do rather than what I’m looking for which is to get into the low-level technical details of database design so I can build one myself from scratch. Thankfully it did offer a clue as to what I could research next: it was mentioned that for the database’s key-value storage engine, the Rust library “RocksDB” could be used. I thought that maybe I could see how this library is implemented so that I can gain more of the low-level knowledge I’m looking for.

# Development Methodology, Design, Implementation, Testing and Evaluation + Proposed Technologies