Individual Final Project BLJ

Creating a Notes App using Rust

Name: Matteo Luciano Siro Cipriani

Submission Date: June 18, 2025

Company: Soreco AG

Contents

1   Versioning

Version	Date	Time	Updates	Author
v0.1.0	03.06.2025	13:06	Started Documentation	Matteo Cipriani
v0.1.1	03.06.2025	14:27	Started Introduction	Matteo Cipriani
v0.1.2	03.06.2025	15:08	Started Listing Sources	Matteo Cipriani
v0.2.0	03.06.2025	17:11	Revamped Components Cable to be of LaTeX-style	Matteo Cipriani
v0.3.0	04.06.2025	10:22	Added Decision Matrix	Matteo Cipriani
v0.3.1	04.06.2025	10:24	Added Dailies Section	Matteo Cipriani
v0.3.2	04.06.2025	11:02	Added Dailies 1 & 2	Matteo Cipriani
v0.4.0	04.06.2025	11:37	Described egui Testing	Matteo Cipriani
v0.4.1	10.06.2025	09:34	Added Daily 3	Matteo Cipriani
v0.5.0	10.06.2025	11:18	Rough Description of Security Features	Matteo Cipriani
v0.5.1	10.06.2025	13:17	Refactored Descriptions	Matteo Cipriani
v0.6.0	10.06.2025	13:35	Initialized Glossary	Matteo Cipriani
v0.6.1	10.06.2025	14:52	Entered First Entries into Glossary	Matteo Cipriani
v0.7.0	10.06.2025	15:22	Began Gantt for Planned Schedule	Matteo Cipriani
v0.7.1	10.06.2025	16:40	Changed Gantt from SVG to PNG	Matteo Cipriani
v0.7.2	11.06.2025	09:01	Added Daily 4	Matteo Cipriani
v0.7.3	11.06.2025	16:35	Added Daily 5	Matteo Cipriani
v0.8.0	16.06.2025	13:51	Redesigned Glossary	Matteo Cipriani
v0.8.1	16.06.2025	16:24	Added Daily 6	Matteo Cipriani
v0.8.2	17.06.2025	14:29	Extended Glossary with new words	Matteo Cipriani
v0.8.3	17.06.2025	15:31	Redesigned versioning table	Matteo Cipriani

2 Introduction

2.1 Task Definition

At the end of the year, each apprentice who worked in the ZLI is required to do a project of their own choosing. They have to plan, execute and document an appropriate project over the span of 4 weeks, while working Monday - Wednesday (or Wednesday - Friday, depending on their school days). With this project, the apprentices can demonstrate what they have learned from the coaches during the last year, as all competences required to fulfill the project have been topics during this past year, some have been used very frequently, while others have only been discussed during 1 week.

2.2 Project Description

I chose to create a Notes App using Rust. I initially wanted to make a To-Do App, but as I have already done a To-Do App using Dart & Flutter as my Sportferienprojekt, I chose to go with something different. I want to try to write this project purely in Rust, to see how much of the language I have learned during the last year, and I can definitely learn new things from this project too. Because Rust is quite famous for being a really safe programming language, I want to try and implement one or two ways to encrypt and store the data safely.

2.3 Known Risks

I know that creating an application purely in Rust might be difficult, especially because Rust isn't really made to design, but to work. To implement a GUI, you have to use crates, which are known to sometimes be even more difficult than the standard Rust syntax itself. And Rust itself has a pretty steep learning curve too. Managing lifetimes, references, and borrowing can be complex, especially with dynamically changing data like note content. On top of that, Rust's error system (e.g., `Result` and `Option`) is safe but verbose, requiring you to explicitly handle many cases.
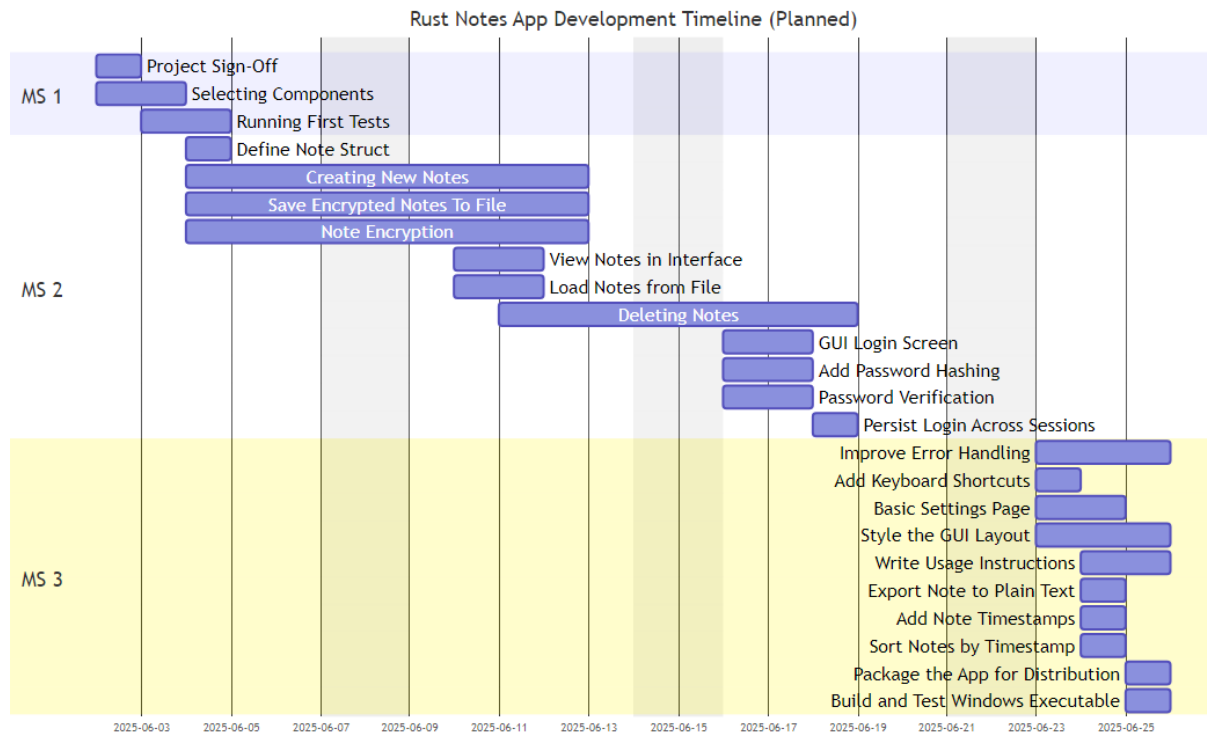
3 Planning

3.1 Schedule

Rust Notes App Development Timeline (Planned)

Figure 1: Schedule (Planned)

3.2 Decision Matrix

3.2.1 Basic Functionality Components (MS 2)

Feature	Recommended Crates	Implementation Suggestions
Password Verification	`argon2`, `bcrypt`, or `pbkdf2`	Use Argon2id for password hashing - it's modern and secure. Store the hash in a config file using `confy` or `config`. For verification, use the crate's verify function to compare entered password against stored hash.
GUI Framework	`egui`, `iced`, or `druid`	`egui` is lightweight and easy to use for beginners. `iced` offers a more Elm-like architecture. Both have good documentation and active communities.
Note Struct	`serde`, `chrono`, `uuid`	Use `chrono` for timestamps, `uuid` for unique IDs, and `serde` with the `derive` feature for serialization. Consider implementing `Display` and `Debug` traits.

Feature	Recommended Crates	Implementation Suggestions
Note Encryption	`aes-gcm`, `chacha20poly1305`, `orion`	ChaCha20-Poly1305 is modern and fast on all platforms. Use `ring` or `orion` for key derivation from password (PBKDF2 or Argon2). Store a random salt with each note.
File Storage	`serde_json`, `bincode`, `postcard`	`bincode` for efficient binary serialization or `serde_json` for human-readable storage. Use `directories-next` to find appropriate app data directory.
Loading Notes	Same as above + anyhow	Use `anyhow` or `thiserror` for error handling during file operations. Consider lazy loading for large note collections.
Note Viewing	GUI framework	Implement a split view with note list on left and content on right. Use virtual list if you expect many notes.
Deleting Notes	GUI framework + con-firm_dialog	Use context menus from your GUI framework. Consider soft deletion (marking as deleted) before permanent removal.
Session Persistence	`keyring`, `directories-next`	Store an encrypted token in the system keyring or in a hidden file in the app directory. Use `directories-next` to find appropriate locations.

3.2.2 Finalization & Polish Components (MS 3)

Feature	Recommended Crates	Implementation Suggestions
Error Handling	`thiserror`, `anyhow`, `log`	Define custom error types with `thiserror`. Use `log` with `env_logger` or `fern` for logging. Show user-friendly messages in UI while logging details.
GUI Styling	GUI framework theming	Most Rust GUI frameworks support theming. Use system colors or implement dark/light mode toggle. Consider accessibility (contrast, font sizes).
Timestamps	`chrono`, `time`	Store UTC timestamps internally, convert to local time for display. Format with `chrono`'s formatting options.
Note Sorting	Standard library	Use Rust's built-in sorting with custom comparators. Consider allowing multiple sort options (recent, alphabetical).
Plain Text Export	`std::fs`, GUI file dialog	Use native file dialogs from your GUI framework. Consider supporting multiple formats (txt, md, html).
Settings Page	GUI framework, `confy`	Store settings with `confy` which handles serialization. Create a modal dialog or separate tab for settings.
Windows Executable	`cargo-wix`, `cargo-bundle`	Use GitHub Actions for CI/CD to automate builds. Test on Windows VM before release.

Feature	Recommended Crates	Implementation Suggestions
App Packaging	`cargo-wix`, `cargo-bundle`	Create an installer with `cargo-wix` for Windows. Include a nice icon and proper metadata.
Documentation	`mdbook`	Write user docs in Markdown, possibly generate with `mdbook`. Include screenshots and keyboard shortcuts.
Keyboard Shortcuts	GUI framework	Most GUI frameworks have built-in shortcut handling. Map to actions using a configuration struct.

4   Main Content

4.1   Procedure and steps

4.1.1   Testing with `egui`

After reading up on a bit of the documentation, I tried to copy a simple "tutorial" app that is just a input field for a name and a slider for the age. Once I was done copying all the code and successfully ran the program for the first time, I tried to figure out how I can access variables that I referenced when initiating the app's components. I also played around with function calls, and where it is best to place them.

4.1.2   Encrypting and saving

During my experiments with `egui` and other crates that I'd use later in the project, I initially went for an approach that encrypts the notes in an unusual way:

1. Key Derivation: A random 16-byte salt is generated. Using Argon2id, a secure 32-byte key is derived from the user's password and the salt.
2. Encryption: A random 12-byte nonce is created. The plaintext data is encrypted using AES-256-GCM with the derived key and nonce, producing ciphertext and an authentication tag.
3. Metadata Attachment The salt, Argon2 password hash (as UTF-8), nonce, and ciphertext are bundled together along with structured metadata (e.g., as a JSON `EncryptedData` object).
4. Obfuscation and Finalization
    - A fake 'SQLite format 3' header is prepended.
    - A Unix timestamp (8 bytes) and 48 bytes of random padding are appended.
    - A SHA-256 checksum of the entire content is added for integrity.

4.2   Results

4.3   Function Descriptions

4.4   Work journal

4.5   Test plan

Figure 2: Results of First Tests with egui

5 Dailies

5.1 Day 1: 02.06.2025

On the first day, I first ran my idea by Reto really quickly, just to confirm wether I could do a project like this. After all, I didn't want to plan my project for the next few hours just for it to get turned down by him. After his confirmation, I began planning my project, utilizing GitHub Projects, just like I've already done for my other project that I realized while in the ZLI. Shortly after the lunch break, I presented my idea in a more detailed manner - every milestone and a few distinct goals that I wanted to reach. After getting the project signed off by Reto, I was able to begin my research on what I would actually need / use for this app. Very quickly, I found out about `egui` and `eframe`, which are 2 incredibly well documented crates that make it manageable to create a GUI for your app. As for safety, I chose to go with `argon2` for the password verification, while I decided to try out `chacha20poly1305` combined with `ring` or `orion` for the note encryption itself. I actually already got to test a bit with egui, where I tried to copy a basic application with name and age, that shows you how `egui` works and what is to expect when working with it.

5.2 Day 2: 03.06.2025

Day 2 was mainly focussed on the documentation, as I knew from my last project, that it would get incredibly difficult to write a good documentation just during the last week, as you forgot a lot of things already. I didn't want to create a documentation with Word, as I had quite a few problems with it the last time I tried it, so I did some research into Markdown-Documentations enhanced with LaTeX and found out, that it is actually a viable alternative to create your documentation with. While the installation of all the things I needed (or I needed to update) took quite some time, I think that I'll get that time back by not having to wrestle with the layout on each page every time I try to insert a picture. In the afternoon, I first began by describing the project's guidelines, my project description and the risks I knew about before beginning my project. I then added the list of sources that I have already used, which there were more of than I first thought.

5.3 Day 3: 04.06.2025

On day 3, I started implementing on implementing my basic features. I started by first designing a lightweight design for my app, with a simple GUI layout, so that the app wouldn't be hard to use. While I had setup the GUI fairly quickly, the saving / encryption process wouldn't be that fast. It took me way longer than expected to combine all my security features with each other, so that'd it actually be encrypted the way I described it in my project setup. But - after some trial and error, lots of documentation read and some help by v0, I got it to work. It now stores all data in the user's configuration directory. For windows, this would be `%APPDATA%\secure_notes\`, where it creates 3 files:

- `auth.hash`, which stores the password hash for authentication
- ~~`safety.meta`~~ `security.meta`, which contains security metadata (hardware fingerprints, timestamps)

    Changed `safety.meta` to `security.meta` for more accurate file names

- `notes.enc`, which is the encrypted note data itself

By implementing an encryption key which is bound to hardware characteristics like username, operating system, computer name, etc..., this creates a hardware fingerprint that makes the encrypted data only accessible on the same machine that it was encrypted on.

5.4 Day 4: 10.06.2025

The fourth day was also mainly focussed on documenting. I documented a few of my security features and how they work, and created the first entries into the glossary. I also created a Gantt diagram that shows how I initially planned my project to develop, to which I will add a Gantt of how it actually developed over the days once the project is finished. The app itself did not change much this day, as I had some catching up to do with my documentation. I only changed the UI a bit and also decreased the security of the app a little

bit - to a point where it still is theoretically safe for production, but the user doesn't have to wait 26 seconds every time they want to log in.

5.5   Day 5: 11.06.2025

The fifth day was overshadowed by a simple, yet embarrassing error that I made. When I was re-reading my code that I wrote over the past week, I noticed that I set the filename for security metadata to `safety.meta`, which I decided to change to `security.meta`, as that was a more fitting name for me. I also noticed that I was using `allocate_ui_at_rect()` to display my notes in the statusbar on the left, which I tried to refactor to `allocate_new_ui()` as `allocate_ui_at_rect()` is now deprecated and been replaced by `allocate_new_ui()`. When I ran the project for the first time again, the login just didn't work anymore. I got an error that my password was not matching, even though I entered the correct password. It took me an embarrassing amount of time to realize that it wasn't the new `allocate_new_ui()` approach that I took that was causing the error - which wouldn't have made any sense anyways since that only gets loaded once the login has already been completed - but the name change from `safety.meta` to `security.meta` that caused the error during login, as it searched for a file that didn't even exist. This is what caused the authentication error. I thusly had to delete all the files that have been created in `%APPDATA%\secure_notes\`, so that the program would think that it's a first time setup again, so I can create a new master password.

5.6   Day 6: 16.06.2025

Day 6 was shaped by some more documenting. Early in the morning, Reto reminded us that each of our repositories needed a README file, which I didn't yet create. So I had to work on that before I could do anything else. It took me a bit longer than expected because I wanted to create a clean and easy-to-read README file, that can be understood by anyone. I also added some parts where I'm not sure if I want to keep them in over longer periods of time, as they could grow to be untrue / not implemented.

5.7   Day 7: 17.06.2025

Day 7 was - unfortunately - quite an unproductive day. I tried my best at implementing user logins and getting the login to persist for *x* amount of time. However, I had some problems with storing all the user data and getting the right one to decrypt their password, and with showing them the right notes, that I had to call that mission off quite quickly. I then looked back at my documentation, and added some more words to my glossary. I then also went over my GitHub Roadmap again, for which I marked off the issues that I have now completed.

6 Appendix

6.1 Sources

- docs.rs
  - Basic Documentation for all of the crates used
  - Further linking to official websites / GitHub repositories with official examples / code snippets
  - Structs, Enums, Functions, Models & Type Aliases for each crate (if available)
- GitHub
  - Extensive Documentation about crates
  - Function snippets
  - Implementation Methods
- Rust's official website
  - Basic Questions about Rust's functionality
  - Further linking to community boards
- THE Rust Book
  - Basic Introduction to Rust
  - Easy explanations for some more complicated topics of Rust
- Rust By Example
  - Examples for some crucial features
    * Examples are editable & executable
  - Good "playground"
- The Cargo Book
  - Guide through Rust's package manager
  - Easy point to access Features, Commands and general infos about cargo

6.2 Glossary

6.2.1 Checksum

A small-sized hash or value used to verify the integrity of data. It ensures that data has not been tampered with or corrupted during storage or transmission.

6.2.2 CI/CD

Stands for Continuous Integration and Continuous Deployment/Delivery. In the context of a Rust secure notes app, CI/CD automates testing, building, and deploying updates to ensure code reliability and fast delivery.

6.2.3 Ciphertext

The encrypted form of data that cannot be read without decryption. In the app, notes are converted to ciphertext using an encryption key before being stored on disk.

6.2.4 Hash

A deterministic output of a hash function, producing a fixed-size value from arbitrary input. Hashes are used for verifying integrity, storing passwords securely, and comparing data without revealing the original input.

6.2.5 Key Derivation

A cryptographic process that generates a strong encryption key from a password or passphrase. Typically used with algorithms like PBKDF2, Argon2, or scrypt to protect against brute-force attacks.

6.2.6 Lightweight

Describes a program or library with minimal resource usage (e.g., memory, CPU). A lightweight secure notes app in Rust would be fast, efficient, and suitable for low-power or embedded environments.

6.2.7 Nonce

A "number used once" in cryptography to ensure that encryption results are unique each time. Used in encryption schemes like AES-GCM to prevent replay attacks and ensure data security.

6.2.8 Password Hashing

The process of converting a password into a fixed-size string (hash) using a cryptographic hash function. In a secure notes app, this is used to securely store and verify user passwords without keeping them in plain text.

6.2.9 Salt

A random value added to passwords before hashing to ensure unique hashes for identical passwords. This prevents precomputed hash attacks (e.g., rainbow tables).

6.2.10 Serialization

The process of converting data structures (e.g., Rust structs) into a format that can be stored or transmitted, such as JSON, TOML, or binary. Used in the app to save and load notes securely.

6.2.11 System Keyring

A secure storage mechanism provided by the operating system for storing secrets such as passwords or keys. The secure notes app can optionally use the system keyring to store encryption keys safely.

6.3   Code Snippets

```rust
struct MyApp {
    name: String,
    age: u32,
}

impl MyApp {
    fn new() -> Self {
        let mut app = Self {
            name: String::new(),
            age: 18,
        };
    }

    fn increment(&mut self) {
        if self.age < 120 {
            self.age += 1;
        }
    }

    fn decrement(&mut self) {
        if self.age > 0 {
            self.age -= 1;
        }
    }
}

egui::CentralPanel::default().show(ctx, |ui| {
    ui.add(egui::Slider::new(&mut self.age, 0..=120).text("years old"));

    ui.horizontal(|ui| {
        if ui.button("Increment").clicked() {
            self.increment();
        }

        if ui.button("Decrement").clicked() {
            self.decrement();
        }
    });
});
```

Figure 3: Learning how to place and call functions with the correct references

```rust
                              main.rs
fn main() → Result<(), eframe::Error> {
    let options = eframe::NativeOptions {
        viewport: egui::ViewportBuilder::default()
            .with_inner_size([800.0, 600.0])
            .with_title("Secure Notes"),
        ..Default::default()
    };

    eframe::run_native(
        "Secure Notes",
        options,
        Box::new(|_cc| Ok(Box::new(NotesApp::new()))),
    )
}
```

Figure 4: Initiating egui

```rust
egui::CentralPanel::default().show(ctx, |ui| {
    ui.heading("Encrypted Text Application");

    ui.horizontal(|ui| {
        let name_label = ui.label("Your name: ");
        let response = ui
            .text_edit_multiline(&mut self.name)
            .labelled_by(name_label.id);
    });

    ui.add(egui::Slider::new(&mut self.age, 0..=120).text("years old"));
    ui.label(format!(
        "Hello {}, you are {} years old!", self.name, self.age
    ));
});
```

Figure 5: Learning how to work with variables in egui