

Home Lab Rust

Mars 2023



Developers
Group Dijon



Adrien Gras
Mathias Da Costa

Home LAB

rpass

Implémentation d'un gestionnaire de mots de passe CLI qui reprend les bases de la commande linux « pass »

...Avant de commencer



Allez chercher de la documentation

Rust est un langage très bien documenté avec une communauté très active, n'ayez pas peur d'aller chercher une solution sur internet !



Venez poser des questions

Un problème ? Rejoignez-nous sur le serveur Discord de la communauté du Developers Group Dijon et posez toutes vos questions



**[https://discord.gg/
Pp6pHUUBXd](https://discord.gg/Pp6pHUUBXd)**

Home LAB

rpas

- rpass est une réimplémentation « libre » du gestionnaire de mot de passe « pass » linux.
- Il permettra d'ajouter, de lister et de retirer des mots de passe du gestionnaire.
- Il permettra aussi d'insérer un mot de passe dans une suite de commandes.
- Enfin, il permettra de générer des mots de passe sécurisés.



Usage: rpass [OPTIONS] <COMMAND>

Commands:

list	List all the password stored in the DataStore
init	Initializes a new DataStore
add	Adds a new password to the DataStore
delete	Delete a given password from the DataStore
dump	Dumps a given password into standard output
generate	Generates a new strong password and stores it
help	Print this message or the help of the given subcommand(s)

Options:

-m, --master-password <MASTER_PASSWORD>	master password to unlock the DataStore
-h, --help	Print help
-V, --version	Print version

Setup

- Exécuter un `rustup update`.
- Se déplacer dans le dossier créé lors du starter lab.
 - Si vous ne l'avez pas, cloner le dépôt github : [git@github.com:developers-group-dijon/2023-codelab-rust.git](https://github.com/developers-group-dijon/2023-codelab-rust.git)
- Se rendre dans le dossier `starter_lab` puis `rpass`.
- Lancer VSCode dans le dossier.




```
rustup update
```

```
cd /srv/2023-codelab-rust/starter_lab/rpass
```


```
code .
```

Les structures de données : struct

- Une `struct` est une représentation d'un modèle de données. C'est « l'équivalent » d'une classe en POO (à quelques différences près).
- Une `struct` peut être agrémentée de comportements via des macros dérivatives qui vont ajouter des comportements à la `struct` au moment de la compilation.
- <https://doc.rust-lang.org/book/ch05-01-defining-structs.html>



```
struct User {  
    first_name: String,  
    last_name: String,  
    email: String,  
    /// ...  
}
```



```
#[derive(Serialize, Deserialize)]  
struct Payment {  
    currency: Currency,  
    amount: f32,  
    due_date: DateTime<Utc>,  
    message: String,  
}
```

Modules & visibilité

- En rust, un fichier = un module (espace de nom) qui va contenir des constantes, des fonctions, des traits, des enums, etc.
- Par défaut, tout composant d'un module est privé à ce module, c'est-à-dire qu'en dehors du fichier, le composant est inutilisable.
- Pour rendre un composant visible, il faut lui ajouter le mot clé `pub`.
- <https://doc.rust-lang.org/rust-by-example/mod.html>



```
// maths.rs
```

```
// sqrt est privée, invisible à  
// l'extérieur de maths.rs
```

```
fn sqrt(num: f32) -> f32 {}
```

```
// pow est taggé "pub" donc visible  
// à l'extérieur de maths.rs
```

```
pub fn pow(num: f32, exp: u32) -> f32 {}
```


Structuration de la CLI

- Ouvrez le fichier `src/cli.rs`.
- Constatez que la `struct Cli` est vide.
- Complétez la struct `Cli` pour y ajouter un champ `master_password` de type `Option<String>` et un champ `command` de type `Command`.
- Veillez à ce que ces deux champs soient visibles de l'extérieur.



Structuration de la CLI

- Ouvrez le fichier src/cli.rs.
- Constatez que la `struct Cli` est vide.
- Complétez la struct `Cli` pour y ajouter un champ `master_password` de type `Option<String>` et un champ `command` de type `Command`.
- Veillez à ce que ces deux champs soient visibles de l'extérieur.



Null = 😞, Option<T> = 😊

- Rust n'utilise pas le concept de `Null`.
- A la place il utilise `Option<T>` qui matérialise la possibilité qu'une valeur soit définie ou non, dans une énumération.
- C'est le même principe que les `Result`, ils doivent être traités obligatoirement.
- Cela permet d'être sûr et certain de traiter le/les bons cas d'utilisation.



```
enum Option<T> {  
    Some(T),  
    None  
}
```



```
let my_maybe_value = ...;  
  
// manière non "safe", peut causer un crash si  
// my_maybe_value est None (non défini)  
let value = my_maybe_value.unwrap();  
  
// ...si my_maybe_value est définie  
if let Some(value) = my_maybe_value {  
    // ...  
}  
  
// ...si my_maybe_value est non définie  
if my_maybe_value.is_none() {  
    // ...  
}  
  
// ...avec le pattern matching  
match my_maybe_value {  
    Some(value) => ...,  
    None => ...,  
}
```

Structuration de la CLI

- Ouvrez le fichier `src/cli.rs`.
- Constatez que la `struct Cli` est vide.
- Complétez la struct `Cli` pour y ajouter un champ `master_password` de type `Option<String>` et un champ `command` de type `Command`.
- Veillez à ce que ces deux champs soient visible de l'extérieur.



Structuration de la CLI



```
#[derive(Parser)]  
#[command(author, version, about, long_about = None)]  
pub struct Cli {  
    /// master password to unlock the DataStore  
    pub master_password: Option<String>,  
  
    /// sub-command to actually run a part of the program.  
    pub command: Command,  
}
```

Structuration de la CLI



```
[derive(Parser)]
#[command(author, version, about, long_about = None)]
pub struct Cli {
    /// master password to unlock the DataStore
    #[arg(short, long)]
    pub master_password: Option<String>,

    /// sub-command to actually run a part of the program.
    #[command(subcommand)]
    pub command: Command,
}
```

Lançons notre CLI

- Pour exécuter votre programme, il suffit de lancer un terminal, et d'exécuter `cargo run`.
- Pour l'instant, un `todo!()` bloque l'exécution de la suite de la CLI.
- Pour voir le résultat de notre structuration de la CLI, utilisez `cargo run -- --help`.
- Le `--` après `cargo run` permet de passer des arguments/options à la CLI.



```
rpass - A rust implementation of 'nix pass
```


```
Usage: rpass.exe
```

```
Options:
```

```
-h, --help      Print help  
-V, --version   Print version
```

Ajoutons les sous-commandes

- Complétez l'enum `Command` avec les possibilités suivantes :
 - `Init`
 - `Add`
 - `List`
 - `Delete {name: String}`
 - `Dump {name: String}`
 - `Generate`
- Vous pouvez ajouter des descriptions aux champs de `Cli` et `Command` pour le `-help` de la CLI.
- La documentation de code se fait avec un triple slash `///`.



```
pub enum MyEnum {  
    Choice1,  
    Choice2,  
    ChoiceWithString {my_string: String},  
}
```


Ajoutons les sous-commandes



```
#[derive(Subcommand, Clone, PartialEq)]
pub enum Command {
    /// List all the password stored in the DataStore
    List,
    /// Initializes a new DataStore
    Init,
    /// Adds a new password to the DataStore
    Add,
    /// Delete a given password from the DataStore
    Delete {
        /// name of the password to delete
        name: String,
    },
    /// Dumps a given password into standard output
    Dump {
        /// name of the password to dump
        name: String,
    },
    /// Generates a new strong password and stores it
    Generate
}
```

Relançons notre CLI

- Pour voir le résultat de notre structuration de la CLI, utilisez `cargo run -- --help`.



rpas - A rust implementation of 'nix pass

Usage: rpas.exe [OPTIONS] <COMMAND>

Commands:

list	List all the password stored in the DataStore
init	Initializes a new DataStore
add	Adds a new password to the DataStore
delete	Delete a given password from the DataStore
dump	Dumps a given password into standard output
generate	Generates a new strong password and stores it
help	Print this message or the help of the given subcommand(s)

Options:

-m, --master-password <MASTER_PASSWORD>	master password to unlock the DataStore
-h, --help	Print help
-V, --version	Print version

Passons aux outils de mots de passe

- Ouvrez le fichier `src/passwords.rs`.
- Implémentez la fonction `generate()` qui permet de générer un mot de passe.



```
/// Generates a safe password with the given length.  
///  
/// the return will be a `PasswordGenerationError::LengthTooLow`  
/// error if the length requested is under 8.  
pub fn generate(len: usize) -> Result<String> {...}
```

???



```
/// Generates a safe password with the given length.  
///  
/// the return will be a `PasswordGenerationError::LengthTooLow`  
/// error if the length requested is under 8.  
pub fn generate(len: usize) -> Result<String>
```



Simplification de la gestion des erreurs

- Nous allons utiliser les crates `thiserror` et `anyhow`.
- `thiserror` permet de créer des représentations de nos cas d'erreurs sous forme d'énumérations, et d'y annoter le message d'erreur correspondant.
- `anyhow` permet de simplifier les `Result<T, E>` en `Result<T>` et en gérant automatiquement l'erreur remontée pour nous.
- Il permet aussi d'ajouter une macro `bail!()` qui permet de retourner tout de suite une erreur en cas de problème, sans exécuter la suite.



```
/// possible errors upon password generation.
#[derive(Debug, Error)]
pub enum PasswordGenerationError {
    #[error("Length must be at least 8.")]
    LengthTooLow,
    #[error("Password generation error: {0}")]
    PasswordGenerationError(String),
}
```

Passons aux outils de mots de passe

- Ouvrez le fichier `src/passwords.rs`.
- Implémentez la fonction `generate()` qui permet de générer un mot de passe.



```
/// Generates a safe password with the given length.  
///  
/// the return will be a `PasswordGenerationError::LengthTooLow`  
/// error if the length requested is under 8.  
pub fn generate(len: usize) -> Result<String> {...}
```

Passons aux outils de mots de passe



```
pub fn generate(len: usize) -> Result<String> {  
    if len < 8 {  
        bail!(PasswordGenerationError::LengthTooLow);  
    }  
  
    let pg = PasswordGenerator { ... };  
  
    let generated = pg.generate_one();  
  
    if let Err(error) = generated {  
        bail!(PasswordGenerationError::PasswordGenerationError(  
            error.into()  
        ))  
    }  
  
    Ok(generated.unwrap())  
}
```

Vous vous souvenez de match ?

- Implémentez la fonction `format_password_strength()` qui permet de générer un mot de passe.
- La force d'un mot de passe est mesuré par un entier entre 0 et 4.
- <https://docs.rs/zxcvbn/2.2.2/zxcvbn/>



```
/// Outputs a String representing a password's strength (measured by Zxcvbn).  
///  
/// This will return an error if the generated score is invalid.  
pub fn format_password_strength(password: &str) -> Result<String> {...}
```


Vous vous souvenez de match ?



```
pub fn format_password_strength(password: &str) -> Result<String> {  
    let estimate = get_password_strength(password)?;  
  
    let message = match estimate {  
        0 => "👤 0/4 - You must change it !".to_string(),  
        1 => "❌ 1/4 - Nowhere near safe !".to_string(),  
        2 => "⚠️ 2/4 - Not safe !".to_string(),  
        3 => "✅ 3/4 - Safe".to_string(),  
        4 => "🔥 4/4 - Ultra safe".to_string(),  
        _ => bail!("Error while parsing password score !"),  
    });  
  
    Ok(message)  
}
```

???



```
pub fn format_password_strength(password: &str) -> Result<String> {  
    let estimate = get_password_strength(password)?;  
  
    let message = match estimate {  
        0 => "👤 0/4 - You must change it !".to_string(),  
        1 => "❌ 1/4 - Nowhere near safe !".to_string(),  
        2 => "⚠️ 2/4 - Not safe !".to_string(),  
        3 => "✅ 3/4 - Safe".to_string(),  
        4 => "🔥 4/4 - Ultra safe".to_string(),  
        _ => bail!("Error while parsing password score !"),  
    });  
  
    Ok(message)  
}
```



?, pour transmettre les erreurs au parent

- Le `question mark operator` permet de remonter l'erreur potentielle d'un bloc à son bloc parent.
- Le bloc parent peut alors le traiter, ou le remonter lui aussi, à condition de retourner un `Result`.
- Il ne peut être appliqué que sur des `Result` et des `Option`.



Et si on parlait... de tests unitaires !

- Rust contient déjà tous les outils pour lancer des tests unitaires sur votre code.
- Il suffit de déclarer un module de test avec la macro `#[cfg(test)]` dans votre code pour déclarer des tests unitaires.
- Ainsi, on peut placer dans le même fichier le code métier et les tests unitaires correspondants.
- Les tests se lancent avec la commande `cargo test`.

```
// votre_module.rs

// ...votre code métier

// déclaration du module de test nommé "tests"
#[cfg(test)]
mod tests {
    // import de toutes les dépendances du fichier actuel
    use super::*;

    // déclaration des fonctions de test unitaire

    #[test]
    fn test_something() {
        assert_eq!(some_function(), true);
    }

    #[test]
    fn test_something_going_wrong() {
        assert_eq!(danger_function().is_err());
    }
}
```

Mon premier test unitaire

- Utilisez ce modèle dans le fichier passwords.rs pour intégrer 2 tests unitaires pour tester la fonction `get_password_strength()` :
 - Un premier pour valider que le mot de passe « test » renvoie bien 0,
 - Un second pour valider qu'un mot de passe généré par la fonction `generate()` est un mot de passe fort (> 3).
- Vous pouvez tester en lançant la commande `cargo test`.

```
// votre_module.rs

// ...votre code métier

// déclaration du module de test nommé "tests"
#[cfg(test)]
mod tests {
    // import de toutes les dépendances du fichier actuel
    use super::*;

    // déclaration des fonctions de test unitaire

    #[test]
    fn test_something() {
        assert_eq!(some_function(), true);
    }

    #[test]
    fn test_something_going_wrong() {
        assert_eq!(danger_function().is_err());
    }
}
```

Mon premier test unitaire



```
// unit tests for this module.
#[cfg(test)]
mod tests {
    use super::*;

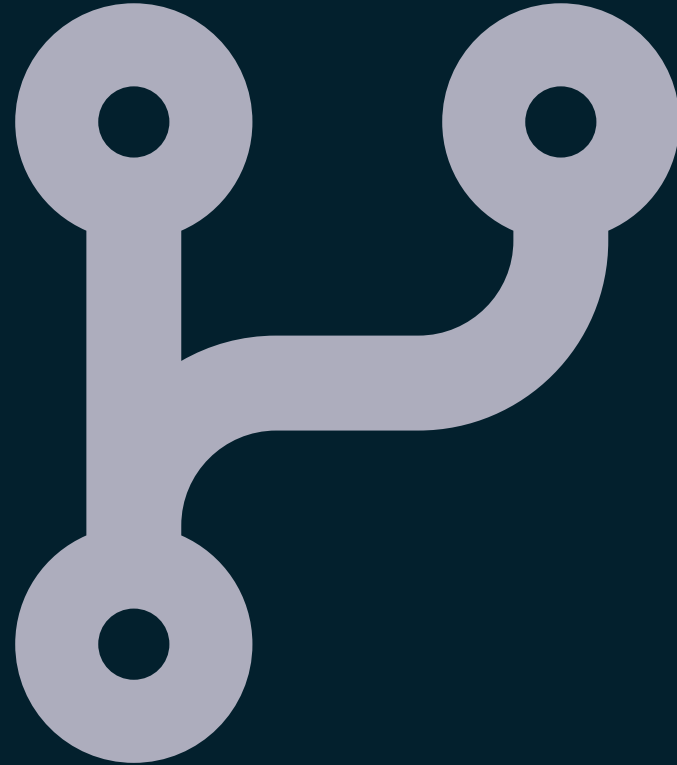
    /// test function for bad passwords
    #[test]
    fn assert_bad_password() {
        assert_eq!(0, get_password_strength("test").unwrap());
    }

    /// test function for strong passwords
    #[test]
    fn assert_strong_password() {
        let generated = generate(24).unwrap();

        assert_eq!(4, get_password_strength(&generated).unwrap());
    }
}
```

Commençons le code métier

- Ouvrez le fichier `src/middleware.rs` et implémentez la fonction `handle()`.
- Vous pouvez aller regarder les méthodes disponibles sur la struct `DataStore` ; soit par l'auto-complétion de votre IDE, soit dans le fichier `src/data_store.rs`.



Commençons le code métier



```
pub fn handle(cli: &Cli) -> Result<()> {  
    let master_password = if cli.master_password.is_none() {  
        require_master_password()?  
    } else {  
        cli.master_password.as_ref().unwrap().to_owned()  
    };  
  
    let data_store = DataStore::new();  
  
    if !data_store.is_initialized()? && !matches!(cli.command.clone(), Command::Init) {  
        bail!(HandlingError::NotInitialized);  
    }  
  
    let unlocked = match cli.command.clone() {  
        Command::List => list(data_store, &master_password)?,  
        Command::Init => init(data_store, &master_password)?,  
        Command::Add => add(data_store, &master_password)?,  
        Command::Delete { name } => delete(data_store, &name, &master_password)?,  
        Command::Dump { name } => dump(data_store, &name, &master_password)?,  
        Command::Generate => generate(data_store, &master_password)?,  
    };  
  
    unlocked.lock()?;  
  
    Ok(())  
}
```


Les choses sérieuses...

- Maintenant que nos sous-commandes vont être dispatchées correctement, il nous faut les implémenter.
- Pour pouvoir travailler sur le `DataStore`, il nous faut déjà le créer.
- Implémentez la fonction `init()` qui permettra de créer un `DataStore` sur votre système.



La struct `ConsoleIO`, une fois initialisé, vous permettra de créer des output stylisés en cas de succès de votre opération ;)

Les choses sérieuses...



```
fn init(data_store: DataStore, master_password: &str) -> Result<DataStore<Unlocked>> {  
    if data_store.is_initialized()? {  
        bail!(HandlingError::AlreadyInitialized);  
    }  
  
    data_store.initialize(master_password)?;  
    let opened = data_store.unlock(master_password)?;  
  
    let console = ConsoleIO::new();  
  
    console.success("Datastore initialized !");  
  
    Ok(opened)  
}
```

Notre première exécution !

- Ouvrez le fichier `src/main.rs`, et retirez le `todo!()` de la méthode `main()`.
- Lancez votre commande à l'aide de `cargo run - init`.

```
> Enter master password: *****  
[v SUCCESS] Datastore initialized !
```

Modèle de donnée



```
/// Representation of password data.
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct PasswordStore {
    /// A label (identifier) linked to a password.
    pub label: String,
    /// A login linked to a password (optionnal).
    pub login: Option<String>,
    /// The actual password.
    pub password: String,
    /// An URL linked to a password (optionnal).
    pub url: Option<String>,
    /// A comment linked to a password (optionnal).
    pub comment: Option<String>,
    /// The creation date for the password data.
    pub creation_date: DateTime<Utc>,
}
```

Ajoutons des données (1/3)

- Nous allons commencer à implémenter la méthode `add()`.
- Elle sera faite en plusieurs parties car elle est assez longue.
- Commencez par le bloc nommé « Partie 1 ».
- Vous pouvez tester votre code en lançant `cargo run - add`.



```
let my_var = console
    .input_text("Question:")
    .with_validator(required!())
    .prompt()?;
```



```
let password = console
    .input_password("Password:")
    .with_display_mode>PasswordDisplayMode::Masked()
    .with_validator(required!())
    .prompt()?;
```

Ajoutons des données (1/3)



```
let console = ConsoleIO::new();

let mut opened = data_store.unlock(master_password)?;

let label = console
    .input_text("Label/name for this password:")
    .with_validator(required!())
    .prompt()?;

if opened.get(&label).is_ok() {
    bail!(HandlingError::KeyAlreadyExists(label));
}

let url = console.ask_question_default("URL for this password:", "");
let login = console.ask_question_default("Login for this password:", "");

let password = console
    .input_password("Password:")
    .with_display_mode>PasswordDisplayMode::Masked)
    .with_validator(required!())
    .prompt()?;
```

Ajoutons des données (2/3)

- Nous allons implémenter une politique d'alerte si le mot de passe saisi est trop faible.
- Pour cela, nous allons utiliser les fonctions du module `passwords` vues précédemment.
- Implémentez la « PARTIE 2 » de la fonction `add()`.



Ajoutons des données (2/3)



```
let password_strength_label = passwords::format_password_strength(&password)?;  
console.writeln(&format!("Password strength: {password_strength_label}"));  
  
if passwords::get_password_strength(&password)? < 3 {  
    let confirmed = console.ask_confirm(  
        "Your password seems to be not safe enough, are you sure you want to store it as it is",  
    );  
  
    if !confirmed {  
        bail!(HandlingError::AdditionAborted);  
    }  
}
```


Ajoutons des données (3/3)

- Terminons notre méthode d'ajout de mot de passe dans le DataStore.
- Implémentez la « PARTIE 3 » de la fonction `add()`.



```
let my_data = MyStruct {  
    ...  
};
```

Ajoutons des données (3/3)



```
let comment = console.ask_question_default("Comment for this password:", "");

let data = PasswordStore {
  label: label.clone(),
  login: if login.is_empty() { None } else { Some(login) },
  password,
  url: if url.is_empty() { None } else { Some(url) },
  comment: if comment.is_empty() {
    None
  } else {
    Some(comment)
  },
  creation_date: Utc::now(),
};


opened.insert(&data)?;

console.success(&format!("{}", label) added !);

Ok(opened)
```

Ajoutons des données !

- Vous pouvez tester votre code en lançant `cargo run -- add`.

```
> Enter master password: *****
> Label/name for this password: my_new_password
> URL for this password:
> Login for this password: login
> Password: *****
Password strength:  0/4 - You must change it !
> Your password seems to be not safe enough, are you sure you want to store it as it is Yes
> Comment for this password:
[✓ SUCCESS] Password "my_new_password" added !
```

Lister nos données

- Implémentez la fonction `list()` qui permettra d'afficher vos données.
- Vous pouvez tester votre code en lançant `cargo run - list`.



```
let my_vec: HashMap<..., ...> = ...;  
  
for (key, value) in my_vec.iter() {  
    ...  
}
```

Lister nos données




```
for (_, data) in opened.data().iter() {  
    let url = sanitize_none_option_string(data.url.clone());  
    let login = sanitize_none_option_string(data.login.clone());  
    let comment = sanitize_none_option_string(data.comment.clone());  
  
    let local_time: DateTime<Local> = DateTime::from(data.creation_date);  
  
    lines.push(vec![  
        data.label.clone(),  
        url,  
        login,  
        comment,  
        local_time.format("%v %X").to_string(),  
        passwords::format_password_strength(&data.password)?,  
    ]);  
}  
  
let console = ConsoleIO::new();  
  
console.string_table(headers, lines);  
  
Ok(opened)
```

Lister nos données

- Vous pouvez tester votre code en lançant `cargo run - list`.

> Enter master password: *********

Label	Url	Login	Comment	Creation date	Password strength
my_new_password		login		8-Mar-2023 19:24:10	 0/4 - You must change it !

Effacer des données

- Implémentez la fonction `delete()` qui permettra d'effacer vos données.
- Vous pouvez tester votre code en lançant `cargo run -- delete <LABEL>`.



Effacer des données



```
let console = ConsoleIO::new();

let mut opened = data_store.unlock(master_password)?;

if opened.get(label).is_err() {
    bail!(HandlingError::KeyNotFound(label.into()));
}

let confirmed = console.ask_confirm(&format!(
    "Are you sure you want to delete entry \"{label}\""
));

if !confirmed {
    bail!(HandlingError::DeleteAborted);
}

opened.delete(label)?;

console.success(&format!("Entry \"{label}\" deleted !"));

Ok(opened)
```


Effacer des données

- Vous pouvez tester votre code en lançant `cargo run -- delete <LABEL>`.

```
> Enter master password: *****  
> Are you sure you want to delete entry "my_new_password" Yes  
[✓ SUCCESS] Entry "my_new_password" deleted !
```

Imprimer le mot de passe

- Implémentez la fonction `dump()` qui permettra d'imprimer le mot de passe sur la sortie standard.
- Vous pouvez tester votre code en lançant `cargo run -- -m <MASTER> dump <LABEL>`.



Imprimer le mot de passe



```
let console = ConsoleIO::new();

let opened = data_store.unlock(master_password)?;

if let Ok(data) = opened.get(label) {
    console.write(&data.password);
} else {
    bail!(HandlingError::KeyNotFound(label.into()));
}

Ok(opened)
```

Et si on générerait de beaux mots de passe ?

- Implémentez la fonction `generate()` qui permettra de générer des mots de passe sécurisés.
- Vous pouvez tester votre code en lançant `cargo run - generate`.



Et si on générerait de beaux mots de passe ?

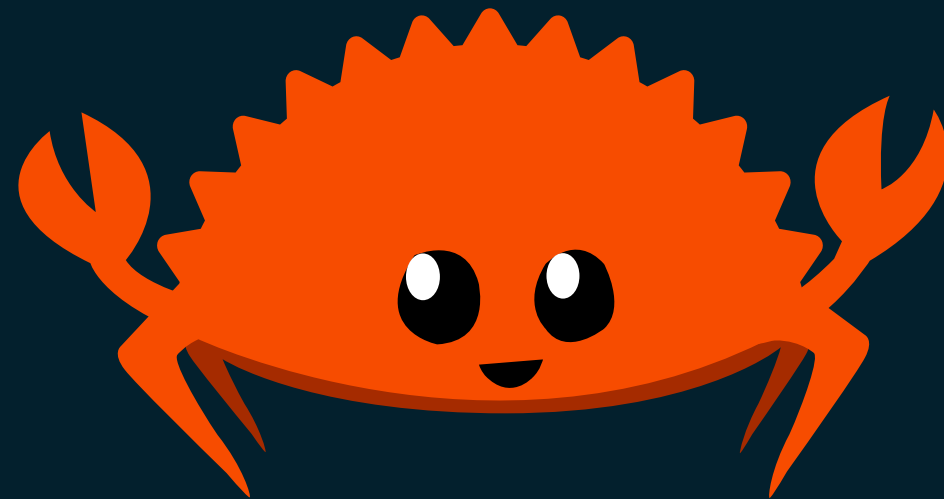


```
let console = ConsoleIO::new( );  
  
let opened = data_store.unlock(master_password)?;  
  
let generated = passwords::generate(24)?;  
  
console.success(&format!("Password generated: {generated}"));  
  
Ok(opened)
```

Et si on générerait de beaux mots de passe ?

- Vous pouvez tester votre code en lançant `cargo run -- generate`.

```
> Enter master password: ****  
[✓ SUCCESS] Password generated: [/3k(SjHouV26.GVVu6p?M>|
```



Félicitations

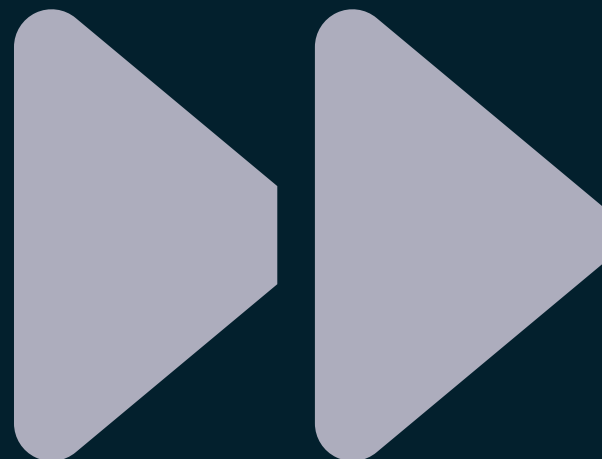
...et si je veux l'utiliser réellement ?

- Pour générer une version dite de « release » de l'application, il vous suffit de lancer la commande `cargo build --release`.
- Cela va construire une version sans code de debug et optimisée de votre application.
- Vous pourrez trouver l'exécutable dans le dossier `target/release`.
 - `rpas.exe` pour Windows
 - `rpas` pour linux



J'en veux plus !

- Vous pouvez améliorer cette application de cette manière :
 - Implémenter la destruction d'un `DataStore`.
 - Implémenter la possibilité d'avoir plusieurs `DataStore`.
 - Implémenter une « hiérarchie » ou des « dossiers » dans les `DataStore`.
 - Lier l'application à l'API `HaveIBeenPwnd` pour vérifier si vos identifiants/logins apparaissent dans des fuites de données.
 - Remplacer les Strings dans la `struct PasswordStore` par des `&str`.
 - Créer une interface graphique pour l'utilisation, avec `TauRi`, ou `Dioxus`.
 - Implémenter une version web SaaS avec `Rocket`.



Rejoignez la crab rave !

- Vous pouvez retrouver de quoi revoir et approfondir les notions de ce starter lab, et de quoi continuer à vous amuser avec Rust sur cette page.



Merci à tous !



Developers
Group Dijon

