SLH
Lab #2

- This lab will be graded.
- You have to submit **your code** and a **report** answering the questions from Section 2.
- The quality of your code will be graded.
- Don't forget to **verify your inputs**.
- Your submission has to be in Rust.
- Test your code whenever you can.
- We provide you with a backend template and a fully implemented frontend. Note that we do server-side rendering.

1 Description

The goal of this project is to manage the authentication of a simple website. The authentication should be managed with JWT access and refresh tokens[1]. The only functionalities of the website are to register, login, and change password.

2 Questions

1. What is the purpose of a JWT access token? And what is the purpose of a JWT refresh token? Why do we have both?
2. Where are the access tokens stored? Where are the refresh tokens stored?
3. Is it a good idea to store them there? Is there a better solution?

3 Main tasks

1. You will have to complete the following files: `utils/jwt.rs`, `backend/handlers_*`
2. **Do not** modify `backend/router.rs` and `backend/models.rs`.
3. You are encouraged to add more modules into `utils`. If you do, add its name into the file `utils.rs` that describes the submodules.

[1]`https://datatracker.ietf.org/doc/html/rfc6749`

4. To simplify, we mock the database with files (very ugly). All the useful functions to interact with the database are in the file `database.rs`.

5. We will also simplify emails and not ask you to use a real SMTP server. Instead, we mock it with the following endpoint `http://127.0.0.1/email/alexandre.duc@heig-vd.ch`, for the email address alexandre.duc@heig-vd.ch. You will find a helper function to send an email in the file `email.rs`. You will also find in the same file a way to reach the endpoint used to validate emails.

6. The frontend is checking every minute whether the access token is valid and refreshes the access token every 2 minutes.

7. Create two functions to create and verify JWTs. The signature of the verification function is given in `utils/jwt.rs` as it is used by the more technical part of the backend.

8. Create two functions to hash and verify the hash of a password.

9. Complete the `backend/handlers_*` TODOs.

4 Explanations on the template

In the provided template, most of the server code is already implemented. The template relies on several libraries to support the web application. The main web library used is `axum`[2] for handling endpoints, `handlebars` for HTML templating, and `bincode` for saving the database.

4.1 axum

`axum` serves as the web application framework for implementing the backend. The server is started in `main.rs` and listens on `localhost:8080`.
The code defines three different routers in `src/backend/router.rs`:

```
fn unauth(state: AppState) -> Router {
    use crate::backend::handlers_unauth::*;
    Router::new()
        .route("/", get(home))
        % Additional routes...
}

fn access(state: AppState) -> Router {
    % Implementation...
}

fn refresh(state: AppState) -> Router {
    % Implementation...
}
```

These routers, `unauth`, `access`, and `refresh`, map to different handlers based on specific input requirements:

- `unauth()`: The request has no requirements, and any mapped route will be served.

[2]https://docs.rs/axum/latest/axum/index.html

- `access()`: The user must provide an access JWT; otherwise, the associated route will not be served.
- `refresh()`: The user must provide a refresh JWT; otherwise, the associated route will not be served.

4.1.1 Handlers

Handlers associated with the routers are functions with powerful functionality. They can easily retrieve various attributes of the request, including:

- `AccessUser`: The connected user.
- `Option<AccessUser>`: An optionally connected user.
- `Session`: The session of the user.
- `Json(parameters)`: A DTO (Data Transfer Object) of type T.
- `CookieJar`: The sent cookies, and more.

Additionally:

- Handlers require the `async` keyword. `https://rust-lang.github.io/async-book/`
- Handlers must return a struct that implements the `IntoResponse` trait.
- Authentication is checked by implementing the `FromRequestParts` trait.

4.2 handlebars

`handlebars`[3] is used for templating. The pages display different information based on whether the user is authenticated or not. It is important to note that direct usage of this crate should not be necessary as the code is already implemented in the template.

[3]`https://docs.rs/handlebars/latest/handlebars/`