SEC

Lab #1 - Input Validation Library

- This lab will be graded.
- The quality of your code will be graded.
- Your submission has to be in Rust.
- Your code should be tested and you should provide your tests.
- We provide you with a template so that you can focus on the essential
- You do **not** need to submit a report.
- Potentially useful crates: `regex`, `infer`, `read_input`, `uuid`

1 Goal

The objective of this lab is to create an input validation library allowing to validate the following inputs:

- URL validation
- File validation
- UUID validation (UUID version 5)

You are of course expected to write unit tests to confirm the proper behaviour of the validation functions. Don't forget your edge cases ! You will then write an *example program* using this library.

2 Validation directions

In this section we describe in more details what validations we ask you to perform.

2.1 URL validation

You are required to write your own regular expression(s) logic to syntactically validate URLs. It is **not** asked to validate semantically URLs.

We will consider a simplified domain name logic:

- Only ascii letters and full stops are allowed for **top-level domains** and they start with a full stop, be at least 3 characters long, and finish with an ascii letter.
- Only ascii letters, numbers, full stops and hyphens are allowed for **subdomains**.
- Only ascii letters and numbers are allowed for **protocol names**.

Your validation logic must support the *optional* presence of protocols in the URL (http, https, ftp, ...), the presence of *multiple sub-domains* (but at least one), as well as an *optional whitelist of authorised top-level* domains (.ch, .com, .org, ...).

The website `https://regex101.com/` may be useful to debug your regexes. Be careful and extensive with your testing for this validator, there are quite a few pitfalls with dynamic regex creation.

2.2 File Validation

You are required to validate that the file contents at a surface level match a given file type or group (image or video) by using information such as magic bytes and headers. To do so, you are allowed to use the `infer` library to extract mime type and extension information. You should also provide the option to validate that the file extension matches the provided filename, irrelevant of text case. You can **reject** all files that are not videos or images.

If needed, you can obtain sample image and video files from `https://file-examples.com/`

2.3 UUID Validation

You must write your own regex to validate that the provided UUID matches the expected format. You must write another function to validate that the contents of a file match the provided UUID. It should be noted that UUID v5 generation is deterministic.

The library `uuid` can help you performing this task.

3 Example Program

Once the library is created, you must provide an example program named `file_upload` located in the standard directory expected by cargo, such that it can be launched by running `cargo run --example file_upload` at the root of your library. It should be noted that all user input **MUST** be read using the `read_input` crate.

This program will provide a menu with 3 options :

- Upload a file by providing its path
- Verify if a file has already been uploaded
- Return the URL of a file

3.1 File upload

- You must not copy the file when uploaded, but simply store its UUID so that you can recover its file path from the UUID at a later time.
- You will only allow for image and video files to be uploaded.
- You must not allow uploaded files to be overwritten.

3.2 Verify File

The verification must be done based **on the UUID**, and not the file path.

3.3 File URL

The user will provide the UUID of a previously uploaded file. From that UUID, you will display an URL of the form `sec.upload/images/<local_path>` or `sec.upload/videos/<local_path>` based on the file type, replacing *"local_path"* with the stored path from when it was uploaded. You must NOT read the contents of the file at this point.

You are only required to display the URL, no need to actually serve the file.

4 Information

A template is provided so that you only need to focus on the validation, test and handler logic. You don't need to write a lab report, only your source code is to be turned in for corrections. When you submit your lab for correction, you **MUST only provide the source code**. You WILL be penalised if compiled binaries or target folders are found in your archive (If its size is any larger that 1 MB, make sure you're only including source code and cargo files).

5 Example Program Execution

```
Welcome to the super secure file upload tool !
Please select one of the following options to continue :
1 - Upload a file
2 - Verify file exists
3 - Get file URL
0 - Exit
Your input ? [0-3] 1
Please enter the path to an image or video file : Cargo.toml
Invalid file contents !
Please enter the path to an image or video file : files/file_example_JPG_100kB.jpg
File uploaded successfully, UUID : d0feb5d6-ce0a-5d47-8cb6-522c29d18eeb

Please select one of the following options to continue :
1 - Upload a file
2 - Verify file exists
3 - Get file URL
0 - Exit
```

```
Your input ? [0-3] 2
Please enter the UUID to check : d0feb5d6-ce0a-5d47-8cb6-522c29d18eeb
File d0feb5d6-ce0a-5d47-8cb6-522c29d18eeb exists, it is an image file.

Please select one of the following options to continue :
1 - Upload a file
2 - Verify file exists
3 - Get file URL
0 - Exit
Your input ? [0-3] 3
Please enter the UUID to get : d0feb5d6-ce0a-5d47-8cb6-522c29d18eeb
sec.upload/images/files/file_example_JPG_100kB.jpg

Please select one of the following options to continue :
1 - Upload a file
2 - Verify file exists
3 - Get file URL
0 - Exit
Your input ? [0-3] 0
Goodbye!
```