Raspirus docs

A simple hash-based virus scanner

Benjamin Demetz

Table of contents

1. Home

1.1 Welcome to Raspirus Docs

1.2 Contents:

1.3 Introduction

This repository contains all documentation of the Raspirus project. It is currently in development and therefore not too reliable.

Last update: April 12, 2023
Created: April 12, 2023

2. FAQ

2.1 How do I generate the documentation for this repository?

You can find the generated documentation in the rust folder and in case you want to generate your own, you can do so by using the `cargo doc` command. Here are some parameters you might want to use with it: - `--no-deps`: Ignores dependencies, only documents the code itself - `--release`: It is generally better than a debug - `--target-dir`: Where to output the docs All together, the command might look something like this: `\ cargo doc --no-deps --release --target-dir=/docs/generated/`

2.2 In VSCode, how do I set up Rust analyzer to work in non-standard directory structure?

The Rust analyzer plugin in Visual Studio Code tries to search for a Cargo.toml file in the current directory, or parent directory. But since we packed the entire application in the `app` directory, it's unable to find the file and therefore might not work. This is a big lost, as it doesn't tell you if your Rust files have correct syntax or not. To solve this issue, you can add an option to the plugin and specify the location of your Cargo.toml file. As stated in this comment, you need to add the following lines to the end of your plugin settings' JSON. Afterward, you will also need to restart the Analyzer for the modification to take effect.

```
{
    "rust-analyzer.linkedProjects": [
        "/home/matklad/tmp/hello/Cargo.toml"
    ]
}
```

2.3 More coming soon

...

Last update: April 12, 2023
Created: April 12, 2023

2.4 Comments

3. Guides

3.1 COMING SOON

3.2 Notes

3.2.1 Export to PDF:

- Follow instructions for your OS on here: https://github.com/orzih/mkdocs-with-pdf#requirements
- Install all dependencies using pip with `pip install -r requirements.txt`
- Run `mkdocs build`
- PDF is located in site/pdf/document.pdf

3.2.2 Translations

- Link: https://github.com/ultrabug/mkdocs-static-i18n

Last update: April 12, 2023
Created: April 12, 2023

3.3 Comments

4. Installation

This guide will help you in building the project on your own machine.

4.1 Table of contents

4.2 Introduction

For people that just want a working app, they can just head over to the Release page and download the executable for the correct platform. But if you are on a different Linux distribution, unsupported OS, or just want to compile the project on your own, this step-by-step guide will guide you.

4.3 Limitations

- Glibc can cause problems on Linux: https://tauri.app/v1/guides/building/linux#limitations
- You need to use 64-bit systems, else the app might crash because it's using memory improvements that only work there
- The app is meant to be run as a "I'm the only app running on this system" app. This is important regarding RAM usage, because if you have much RAM, it will use much RAM. And if you, for some reason, try to limit the initially available RAM, the app might crash because it doesn't have the promised amount of RAM. (A future version might have a toggle for this)

4.4 Step-by-step guide

Please read the whole guide once before starting to execute each step!

4.4.1 1. Download the repository

This step is very straightforward, just download the whole repository by clicking the green button on the homepage of this repository. Optionally, you can also download code specific to a Release by visiting the

Release page and download the `.zip` file in the assets. Another thing you might want to do is clone this repository

4.4.2 2. Install Rust

One of the requirements to compile the project is to have Rust installed. You can check if you have Rust installed on your machine with the command `rustc --version`, if this command fails, head over to the Rust website and follow the instructions for your OS. Also make sure that your Rust installation is up-to-date with the command `rustup update`.

4.4.3 3. Install NPM

NPM is needed because the frontend of the app works on JavaScript and is basically a website. To check if you already have Node.js installed, try executing the commands: `node -v` and `npm -v`. If any of them fail, or you find out you have an older version, head over to the NPM Website to install the latest version for your OS. If you are using a WSL, this guide might be useful to you.

4.4.4 4. Install Next.js

The frontend is built on JavaScript using a well-known framework named Next.js, it makes website development faster and more efficient. You will need to install this tool too to be able to build the application. But don't worry, you can do this easily with NPM: `npm install next@latest react@latest react-dom@latest eslint-config-next@latest`. This will install Next.js, React (which Next.js is based on) and ESLint. You can learn more about the installation process here.

4.4.5 5. Install Tauri

Tauri is the framework that connects the Rust backend with the Next.js frontend. It is an open-source project made by very friendly and welcoming people. Unfortunately, installing Tauri is not as straightforward as other processes. It is very OS dependent, and you will therefore make sure that you meet the Prerequisites before you start. Afterward, you can install Tauri suing cargo: `cargo install tauri-cli`. You could also use NPM to install it, but we will mainly work with cargo in this short guide. Check out their FAQ section to learn about why NPM might be better for you.

4.4.6 6. Install project dependencies

Firstly, we will install the node modules. To do this, head over to the directory that contains all the Raspirus code. Open the `app` directory, open a terminal in this location and execute the command: `npm install`. This might take a while, but it will download all the necessary modules. **Warning!:** On WSL you might get an `OpenSSL is missing` error, to fix this you need to edit the file `app/src-tauri/Cargo.toml` and add the following line: `openssl-sys = {version = "0.9.66", features = ["vendored"]}` in the `[dependencies]` section.

4.4.7 7. Build the project

Before you can completely build the project, there is one more thing you might want to check. To make sure that the Rust part of the project works fine, open the folder `app/src-tauri/` and execute the command `cargo build`. If this command succeeds, you can go back to the `app/` directory. If this command fails, please open an issue on this repository with as much information about the error as possible. If everything went well, you are now in the `app` directory, and you can safely execute the command `cargo tauri build`. This command will build the entire

application and display a path at the end of the process showing you where the executable is located. By default, you should be able to find it in the `app\src-tauri\target\release` folder.

4.5 Conclusion

This application is basically a website attached to some Rust code and packaged with the Tauri framework. It will therefore need a graphical overlay to start and display the website. This project is in constant development and therefore, if you find anything unusual, have some good ideas or find some errors, don't be afraid to open an issue on this repository and I will be happy to help you out.

Last update: April 12, 2023
Created: April 12, 2023

4.6 Comments

5. Usage

5.1 COMING SOON

Last update: April 12, 2023
Created: April 12, 2023

5.2 Comments

6. Contributing

6.1 Contributions

This project is entirely open-source and wouldn't be possible without your help. Every contribution even if just a little can help a lot. Opening an issue if you find a bug, fixing the bug by opening a pull-request, adding translations, adding features and much more are all types of contributions that help the project go forward.

6.1.1 Translations

The app itself is made with user-friendliness in mind and therefore doesn't have much text to translate as we want the experience to be fast without slowing down the user by forcing him/her to read lots of stuff. Nonetheless, this project still needs a lot of translators. Foremost we need translators for the docs that you are reading right now, but the app also needs translators. Translations help improve the user experience and make the project more widely available. You can find out more on the dedicated page.

6.1.2 Coding

The app is built using mainly two programming languages: Rust and JavaScript. JavaScript is used for the frontend to build a website using the Next.JS framework and make the app look nicer and be cross-platform. The backend on the other side is written in Rust because we value speed a lot when scanning files and therefore need a fast language. To make the two communicate we use the Tauri framework. Tauri is an open-source framework written in Rust, quite new in its field. In future there is also a plan to add a Python installer. If you have knowledge in any of these fields you are more than welcome to contribute to the project. You can obviously also use this project as an example to create your own project.

6.1.3 Sponsoring

COMING SOON

Last update: April 12, 2023
Created: April 12, 2023

6.1.4 Comments

6.2 Code

6.2.1 COMING SOON

Last update: April 12, 2023
Created: April 12, 2023

6.2.2 Comments

6.3 Translations

Translations are very important and can help other people understand the app easier and make it wider available. If you know a language outside of English and would like to add your translations you can do so in two ways: The first one is to translate the documentation, the second one is to translate the frontend strings.

6.3.1 Translating code

This is a work in progress, but essentially it will be done in one of two possible ways:

Translation file

There is a translations file in JSON format for each different language. So the app would load the strings it needs from that file. This is a bit easier to implement for the developers, but not very efficient or great for translators, as it involves forking the repository.

Translation service

We use an external translations service to translate the project. This option would allow translators to just translate the given strings and have an overview of how much has been translated and what still needs to be translated. This would be the preferred option, but the downside is the setup for this option. Since we don't know which service to use yet, this option will be added in the future when we have more concrete ideas and hopefully a bigger team. We will probably use Crowdin

6.3.2 Translating Docs

The docs you are reading right now also need translations and will require a much bigger effort than translating strings used on the frontend. The documentation is quite big and can therefore take quite some time to translate. As the option above, we could use the service here too, but that will be added in the future. If you want to start adding translations right away, I would suggest to do the following:

1. Fork this repository: You can follow a guide online on how to do it properly.
2. Enter the `docs` folder and, if the language you want to translate doesn't exist yet, add a new directory whose name is the name of the language you want to write translations for. For example, it would be `it` for italian, `de` for german and so on.
3. Then change to that directory and edit the Markdown files inside that directory. Make sure you keep the original folder structure.
4. After you have finished editing, save your files and upload them to your fork on GitHub. Basically make a GitHub commit
5. Now you can open a pull-request to the original repository and request that your changes get added to the main project.
6. Someone of the team will review the translations and if accepted add them to the main project.

In the future we hopefully have an external service that handles all this for you, so that you can focus on your translations without distractions. We will probably use Crowdin.

Last update: April 12, 2023
Created: April 12, 2023

6.3.3 Comments

7. Developers

7.1 Developers

The Raspirus project mainly uses two different programming languages and frameworks. One for the backend and one for the backend and one for the frontend. \ The frontend is written in JavaScript using the Next.JS framework. The backend on the other side is written in Rust. The communication between these two programming languages is made using the Tauri framework. Using this framework one can write functions in Rust and call them from the frontend easily.

Last update: April 12, 2023
Created: April 12, 2023

7.1.1 Comments

7.2 Backend

7.2.1 COMING SOON

Last update: April 12, 2023
Created: April 12, 2023

7.2.2 Comments

7.3 Frontend

7.3.1 COMING SOON

Last update: April 12, 2023
Created: April 12, 2023

7.3.2 Comments