Raspirus docs

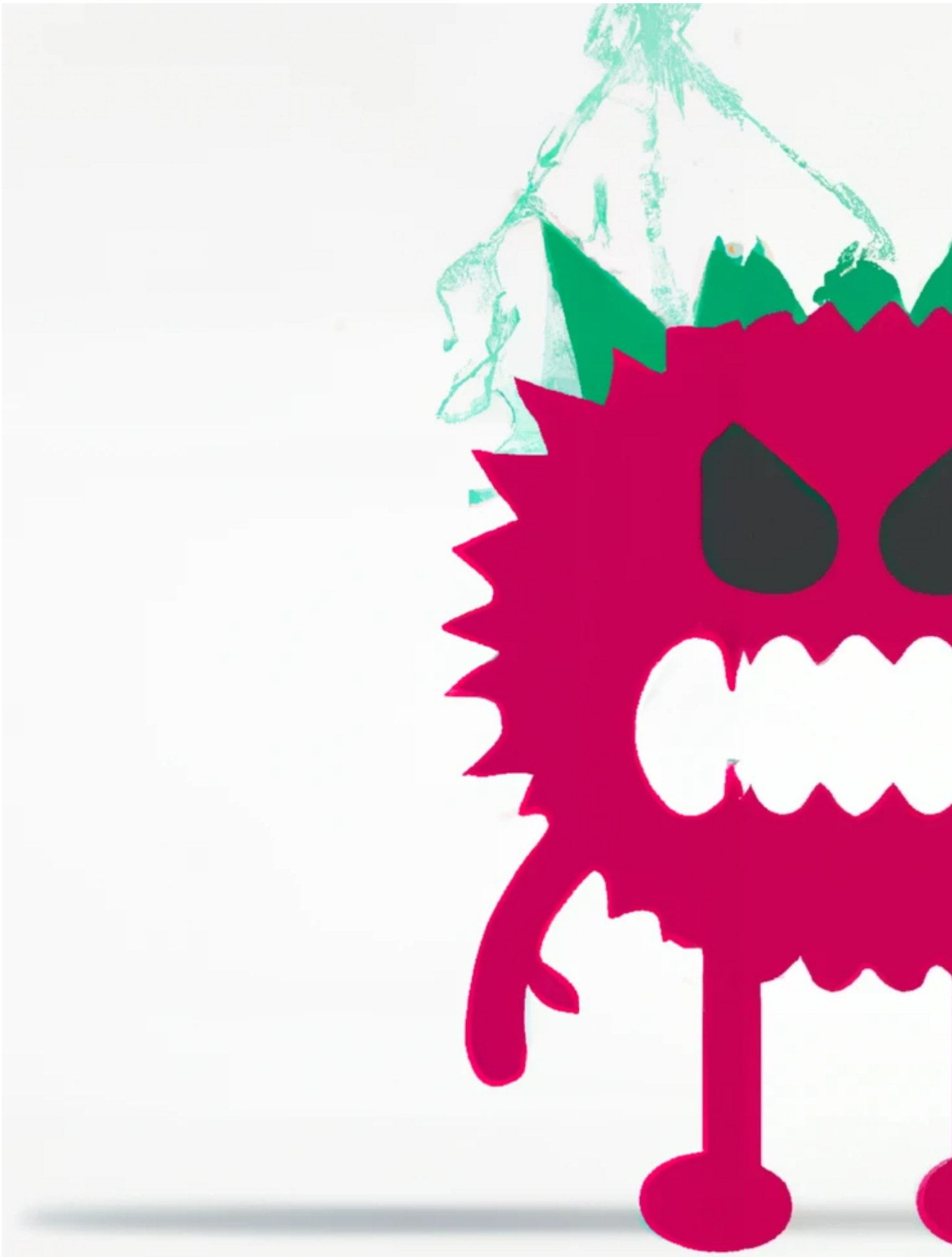
A simple hash-based virus scanner

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.

1.4 Related projects

Raspirus is a simple virus scanner, and there are many similar projects out there. I will list a few of them I know and why I decided not to use them for my project, or why Raspirus is better.

- Clam AV: This program is an open-source Antivirus, but it can also be used like Raspirus to scan single files or folders. It surely is more accurate when it comes to search for viruses, but it is very resource intensive and a bit slow. Therefore, it is not suited for single-board computers like the Raspberry Pi, as it doesn't have enough RAM. Nonetheless, Clam AV is a great open-source tool.
- Windows Defender: This program from Windows doesn't just scan files and folders on-demand, but scans the entire system continuously. This slows down the entire system. Furthermore, it is also Windows-only, while Raspirus is cross-platform.
- Bitdefender: This is another great Antivirus software, but it comes with a cost. I think that security should be a must, not something you need to pay for. But if you are willing to pay a bit, this piece of software really has it all, and as far as I know it is even cross-platform.

There surely are many others that I could compare here, but Raspirus is not aimed at beating other Antivirus software. Its aim is to do one thing, and do it as best as possible: Compare Hashes of a file with a list of signatures.

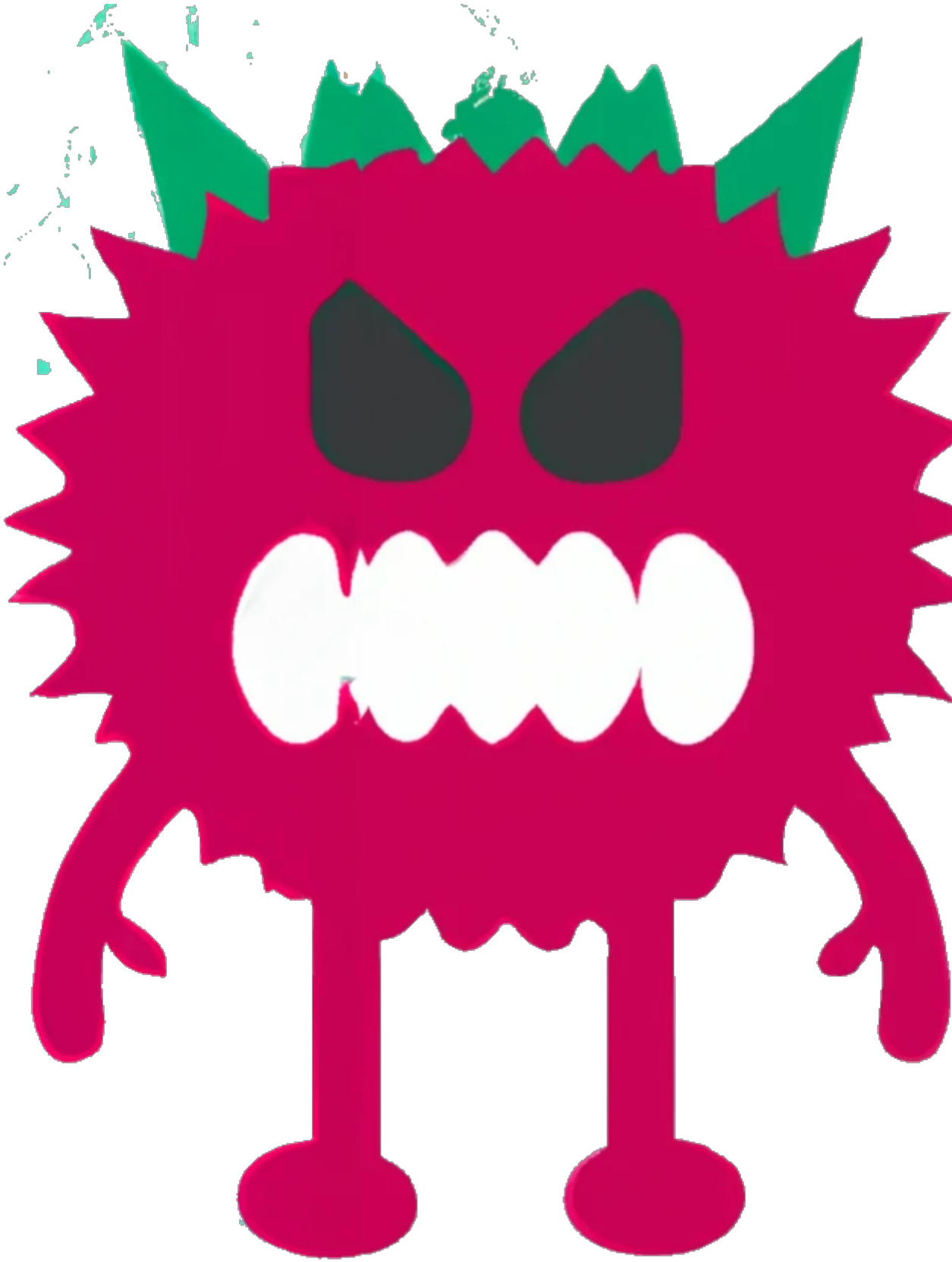
- This project is free and will always stay free.
- It is open-source, so you can look at the code and judge for yourself if you trust the app or not.
- Community helps grow the project, and many smart minds surely can achieve a lot if they work together
- Cross-platform as a standard: Raspirus should work everywhere
- Lightweight and fast, usable even on your Potato PC

Last update: April 21, 2023
Created: April 3, 2023

2. FAQ

In this page, we will answer your most asked questions. As more questions arise, we will expand this page to include more answers. This page is very useful if you encounter any errors during development or usage. Maybe your error can be easily fixed and doesn't require a bug report.

2.1 What is the Icon of the project?



In case you didn't notice yet, this is the logo of the Raspirus app. It was generated with DALL-E and some creative image editing and merging. It should represent a red monster that eats viruses. His name is Stuart by

the way, and don't worry, he is a very kind monster, except for when he is hungry, then you better feed him viruses. You can find more media and documents in the dedicated repository. You are free to use these images to create your own art and showcase them in the discussion boards

2.2 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/`

2.3 In VS Code, 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"
    ]
}
```

2.4 Updating database crashes app

On Windows, it seems like the app crashes when the user tries to update the database. We are aware of this issue and working to fix it. The issue arises because the function needs administrative privileges, which Windows isn't providing. To fix this issue for now, simply execute the app in administration mode, aka. With admin privileges. You can do so by right-clicking the app and clicking `Run as administrator`.

Last update: April 20, 2023
Created: April 5, 2023

2.5 Comments

3. Guides

Here are some guides on how to export the docs to PDF format, or how to add translations to this project. If anything is missing, make sure to post a comment below and request a guide!

3.1 Export to PDF

This documentation can be exported in PDF format for offline sharing. The exported PDF has some issues with Images and iFrames, but the text is readable and can be shared. Here step by step: 1. Clone this repository, you can find instructions on how to it on GitHub 2. Install the requirements for the PDF conversion tool, you can find them for your OS here 3. Change to the cloned directory and install the required dependencies. Note: You will need Python3 and pip for this. You can install the dependencies with the command: `pip install -r requirements.txt` 4. Install mkdocs and run the build command: `mkdocs build` 5. If everything works as expected, your PDF should be located in `site/pdf/document.pdf`

3.2 Translations

- Translate this document: https://github.com/ultrabug/mkdocs-static-i18n

---

Last update: April 21, 2023
Created: April 3, 2023

3.3 Comments

4. Installation

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

4.1 Use an executable

The fastest and easiest way to install the app on your system is using one of the executables provided in the Release page. Once downloaded, it will depend on your system on how to run it, but on Windows for example you can easily double-klick on the executable and follow the installation instructions from the wizard. And that's it, you are done.

4.2 Use the Makefile (Debian based systems only)

The project also features a Makefile for easier installation, update and testing. You can therefore simply clone the repository and execute the Makefile with the command:

```
make install
```

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

4.4 Step-by-step guide

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

4.4.1 1. Clone the repository

With `git` installed, do:

```
git clone https://github.com/Raspirus/Raspirus.git
```

Else, download the repository from here and extract the `.zip` file. The GitHub guide might help you.

4.4.2 2. Install Rust

On **Linux** (Debian, Ubuntu):

```
curl https://sh.rustup.rs -sSf | sh -s -- --help
```

On **Windows**: Download the executable from the Rust website and install it.

4.4.3 3. Install NPM

On **LInux** (Debian, Ubuntu):

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash - &&\
sudo apt-get install -y nodejs
```

On **Windows**: Head over to the NPM Website and download the executable to install it.

4.4.4 4. Install Next.js

Execute the command with npm:

```
npm install next@latest react@latest react-dom@latest eslint-config-next@latest
```

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: May 10, 2023
Created: February 7, 2023

4.6 Comments

5. Usage

5.1 Tutorial

Using the app is really easy. Once you have it installed it is just as easy as pressing a couple of buttons. The entire app is built around the concept of being used on a bad touchscreen, so it needs no input or keyboard to work. **Step by step:** 1. Open the app on the main page 2. Insert an USB and select it, or click the folder icon to select a local folder to scan 3. Accept the user agreement after you have read it carefully 4. Wait for the app do to it's magic 5. Result is shown and you can now start a new scan if needed

5.2 Case study

Here are some metrics of the app, how fast it is, how reliable and what it can actually do. It also describes how the app is intended to be used.

5.2.1 Specification

- App size: 5MB
- RAM usage: ~ 50MB
- Works offline, only requires connection to Internet to update the database file

5.2.2 Metrics

The speed in which the app is able to scan highly depends on three things: - The USB or Storage medium used - The CPU of the system the app runs on. - The size of the single files that need to be scanned

For optimal scanning speed, you should use a modern storage medium with high read speeds, a modern CPU that is not overloaded with other apps and files that are low in size. It is faster to scan 100 files of 10GB than 1 file of 10GB. The reason being that the Hash funktion is faster on smaller files. For Raspberry Pis, don't look at RAM, as the app only requires a couple MB. Instead, look at CPU speed, as that's more important.

5.2.3 Application

The app was originally intended to use only on Raspberry Pis with a touchscreen. The Raspberry Pi with Linux on it is easy to setup and very secure. Most of the viruses target Windows PC, and therefore have no effect on Linux. Furthermore, if you add a battery to your Raspberry Pi and set the app in Kiosk mode, you can create some sort of Raspirus box that can be carried around and is focused on scanning USB sticks. Thanks to Tauri and it's cross-compilation feature, Raspirus is available for all major systems: macOS, Windows, Linux. The app is primarly being developed and tested on Windows and Debian Linux. You can find installation instructions in the Installation page.

Last update: May 10, 2023
Created: April 3, 2023

5.3 Comments

6. Contributing

6.1 Contributing to Raspirus

First off, thanks for taking the time to contribute! ❤️

All types of contributions are encouraged and valued. See the Table of Contents for different ways to help and details about how this project handles them. Please make sure to read the relevant section before making your contribution. It will make it a lot easier for us maintainers and smooth out the experience for all involved. The community looks forward to your contributions. 🎉

And if you like the project, but just don't have time to contribute, that's fine. There are other easy ways to support the project and show your appreciation, which we would also be very happy about: - Star the project - Tweet about it - Refer this project in your project's readme - Mention the project at local meetups and tell your friends/colleagues

6.1.1 Table of Contents

6.1.2 Code of Conduct

This project and everyone participating in it is governed by the Raspirus Code of Conduct. By participating, you are expected to uphold this code. Please report unacceptable behaviour to demetzbenjamin@duck.com.

6.1.3 I Have a Question

If you want to ask a question, we assume that you have read the available Documentation.

Before you ask a question, it is best to search for existing Issues that might help you. In case you have found a suitable issue and still need clarification, you can write your question in this issue. It is also advisable to search the internet for answers first.

If you then still feel the need to ask a question and need clarification, we recommend the following:

- Open an Issue.
- Provide as much context as you can about what you're running into.
- Provide project and platform versions (nodejs, npm, etc), depending on what seems relevant.
- If the question is about the docs, write a comment at the bottom of the said page.

We will then take care of the issue as soon as possible.

6.1.4 I Want To Contribute

Legal Notice

When contributing to this project, you must agree that you have authored 100% of the content, that you have the necessary rights to the content and that the content you contribute may be provided under the project licence.

Reporting Bugs

Before Submitting a Bug Report

A good bug report shouldn't leave others needing to chase you up for more information. Therefore, we ask you to investigate carefully, collect information and describe the issue in detail in your report. Please complete the following steps in advance to help us fix any potential bug as fast as possible.

- Make sure that you are using the latest version.
- Determine if your bug is really a bug and not an error on your side e.g. using incompatible environment components/versions (Make sure that you have read the documentation. If you are looking for support, you might want to check this section).
- To see if other users have experienced (and potentially already solved) the same issue you are having, check if there is not already a bug report existing for your bug or error in the bug tracker.
- Also make sure to search the internet (including Stack Overflow) to see if users outside of the GitHub community have discussed the issue.
- Collect information about the bug:
- Stack trace (Traceback)
- OS, Platform and Version (Windows, Linux, macOS, x86, ARM)
- Version of the interpreter, compiler, SDK, runtime environment, package manager, depending on what seems relevant.
- Possibly your input and the output
- Can you reliably reproduce the issue? And can you also reproduce it with older versions?

How Do I Submit a Good Bug Report?

You must never report security related issues, vulnerabilities or bugs including sensitive information to the issue tracker, or elsewhere in public. Instead sensitive bugs must be sent by email to demetzbenjamin@duck.com.

We use GitHub issues to track bugs and errors. If you run into an issue with the project:

- Open an Issue. (Since we can't be sure at this point whether it is a bug or not, we ask you not to talk about a bug yet and not to label the issue.)
- Explain the behaviour you would expect and the actual behaviour.
- Please provide as much context as possible and describe the *reproduction steps* that someone else can follow to recreate the issue on their own. This usually includes your code. For good bug reports you should isolate the problem and create a reduced test case.
- Provide the information you collected in the previous section.

Once it's filed:

- The project team will label the issue accordingly.
- A team member will try to reproduce the issue with your provided steps. If there are no reproduction steps or no obvious way to reproduce the issue, the team will ask you for those steps and mark the issue as `needs-repro`. Bugs with the `needs-repro` tag will not be addressed until they are reproduced.
- If the team is able to reproduce the issue, it will be marked `needs-fix`, as well as possibly other tags (such as `critical`), and the issue will be left to be implemented by someone.

Suggesting Enhancements

This section guides you through submitting an enhancement suggestion for Raspirus, **including completely new features and minor improvements to existing functionality**. Following these guidelines will help maintainers and the community to understand your suggestion and find related suggestions.

Before Submitting an Enhancement

- Make sure that you are using the latest version.
- Read the documentation carefully and find out if the functionality is already covered, maybe by an individual configuration.
- Perform a search to see if the enhancement has already been suggested. If it has, add a comment to the existing issue instead of opening a new one.
- Find out whether your idea fits with the scope and aims of the project. It's up to you to make a strong case to convince the project's developers of the merits of this feature. Keep in mind that we want features that will be useful to the majority of our users and not just a small subset. If you're just targeting a minority of users, consider writing an add-on/plugin library.

How Do I Submit a Good Enhancement Suggestion?

Enhancement suggestions are tracked as GitHub issues.

- Use a **clear and descriptive title** for the issue to identify the suggestion.
- Provide a **step-by-step description of the suggested enhancement** in as many details as possible.
- **Describe the current behaviour** and **explain which behaviour you expected to see instead** and why. At this point you can also tell which alternatives do not work for you.
- You may want to **include screenshots and animated GIFs** which help you demonstrate the steps or point out the part which the suggestion is related to. You can use this tool to record GIFs on macOS and Windows, and this tool or this tool on Linux.
- **Explain why this enhancement would be useful** to most Raspirus users. You may also want to point out the other projects that solved it better and which could serve as inspiration.

6.1.5 Attribution

This guide is based on the **contributing-gen**. Make your own!

Last update: April 28, 2023
Created: April 5, 2023

6.2 Code of Conduct - Raspirus

6.2.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

6.2.2 Our Standards

Examples of behaviour that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologising to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behaviour include:

- The use of sexualised language or imagery, and sexual attention or advances
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

6.2.3 Our Responsibilities

Project maintainers are responsible for clarifying and enforcing our standards of acceptable behaviour and will take appropriate and fair corrective action in response to any instances of unacceptable behaviour.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviours that they deem inappropriate, threatening, offensive, or harmful.

6.2.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

6.2.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behaviour may be reported to the community leaders responsible for enforcement at demetzbenjamin@duck.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

6.2.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4 and 2.0, and was generated by contributing-gen.

Last update: April 21, 2023
Created: April 21, 2023

6.3 Code

Code are always welcome and strongly needed. Make sure you read the Guidelines before opening your own pull-request. Another good idea is to read about the coding in the Developer section

6.3.1 Contributing to the Frontend

The Frontend uses the basic Next.js structure and tries to put as much stuff in components as possible. The pages should look clean and not cluttered with unnecessary code. Each page represents a screen of the app and are accessed via manual routing. User either accesses a page by clicking a button or by getting redirected there after a loading process finishes. The frontend is entirely in Javascript and there are currently no plans to migrate to Typescript.

6.3.2 Adding to the Backend

The backend is written in Rust for faster scanning speeds and reliability. It uses a big database to locally store the hashes and sends responses over the Tauri API. If you want to contribute here it might be useful to have a look at how Tauri and Rust work. It is not complicated and can be learned quickly with a bit of practice.

Last update: May 10, 2023
Created: April 3, 2023

6.3.3 Comments

6.4 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.

6.4.1 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

6.4.2 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 doing 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: May 10, 2023
Created: April 5, 2023

6.4.3 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.

7.1.1 Technologies used

- NPM: I wanted the frontend to be beautiful, and at first I tried creating an actual native application, but that just didn't have the customization I was searching for. Therefore, I decided to write a Web-frontend, and for that I could choose between Node.js or Deno. I ended up using Node.js as it's the one I am mostly familiar with.
- Next.js: Next.js is a frontend framework that is fairly easy to implement, and once setup can be very powerful. The most useful feature used in this project is the export of the website in static HTML. This is essential for Tauri to function. Another advantage of Next.js is that it optimizes itself, by automatically stripping away everything unnecessary, this makes the app lightweight and fast. Navigation with Next.js is practically instant.
- Rust: Rust was not my first choice for the backend. At first, I started with C++, but I was not skilled enough to create the entire app with that language, so I turned to Python. Python is way slower than C++, but it's my favorite language and the one I know best, so I was able to develop faster. Nonetheless, the lack of speed with Python was a big drawback. Luckily, a friend of mine helped me out and rewrote the entire backend in Rust, basically speeding the app up by 100x. What previously took a couple of minutes, now took mere seconds. Rust is a compiled language and faster than Python, which is an interpreted language. If you want to test the old Python system, there is an entire repository for that.
- Tauri: Tauri plays a very important part in this project, as it is a framework that allows connecting the Rust backend to a JavaScript frontend. This essentially allowed us to have a beautiful frontend in Next.js and an impressively fast backend in Rust. Furthermore, Tauri compiles the application for different systems, like Windows, Linux or macOS by creating installers or binaries. This improved the installation of the app.
- SweetAlertv2: When errors occur, or there are warnings and information to display as a pop-up to the user, the simple JavaScript alert just doesn't look that good. SweetAlert improves the look of these pop-ups by a lot. It's fairly easy to install and to use. In our case, it is mainly used to display errors or warnings, for example when the scanning process is suddenly interrupted.
- next-i18next: The app doesn't have much text, and with the icons and colors it should be fairly simple to understand what you need to do. Nonetheless, we decided to add translations to the app. This is done using a Next.js plugin named i18next, that allows to translate the project easily with .JSON files.
- Crowdin: Crowdin is a website that collects translations from users and helps translate open-source and private projects. I am especially grateful that for open-source projects, the price for this service is free. It is useful for translators to focus on translations, without the need to look at code. It also helps developers, as it automatically syncs new translations with GitHub and vice versa.

- MkDocs: MkDocs is a Python framework used to create documentation for projects. In fact, it is the framework used to create this specific translation. It is really easy to use and makes structuralizing docs or integrating plugins like Crowdin fairly easily.
- Dependabot: Dependabot comes with every GitHub repository, and can be activated fairly easily. Its job is to check dependencies for updates and therefore to keep the entire app up-to-date. This also ensures that the app has the latest patches and functions. By improving its dependencies, the app might become faster and more efficient. Furthermore, Dependabot also informs the user about Security vulnerabilities of used dependencies or crates, a very useful feature to maintain the app secure.
- GitHub: GitHub has been chosen as the file hosting platform mainly because it is popular, easy to use and has everything needed. Plus it's free. A company also suggested hosting the project on their GitLab servers, but in the end the project remained on GitHub as it is easier to use and to collaborate than to use a VPN to connect to a PC that then connects to GitLab. GitHub also has a lot of features: Project planing, Milestones, Issues, Actions, ...
- CodeQL: CodeQL is another powerful tool from GitHub. It scans the entire project and checks for vulnerabilities in the code. For example, if there is cross-site-scripting happening somewhere, or we are not hiding passwords and secrets correctly, or we are coding inefficiently. This is very useful, as it assures a certain reliability of the code.
- CodeCov: Codecov is an external tool that can be installed on GitHub, and it tracks test coverage. It basically tells you how efficient the tests you just wrote actually are, and if they cover the entire project. Testing the frontend is hard, as we would likely need to test user interaction, but testing the backend is doable. Therefore, we set up Codecov to check our Rust backend for test coverage.

7.1.2 Integration

Raspirus is a standalone app and everything happens inside the app. What I mean by this is that there are mo APIs for the outside to get information from, and the app is also not meant to be used by other apps. The main usage of the app is through a touchscreen, that's why there are no input fields where a user has to type text, as that is a quite frustrating experience on bad touchscreen, as the one from a Raspberry Pi. Actually, there are API calls, but they happen inside the app. In fact, the Next.js frontend calls the backend through the Tauri API.

For example, in the `loading.js` file, we call the scanning function from the frontend to tell Rust to start the scanner:

```
const message = await invoke("start_scanner", {
    path: scan_path,
    dbfile: db_location,
    obfuscated: obfuscatedMode,
});
```

We basically start the `start_scanner` function and give it the needed parameters. Then we save the result in the `message` constant. This is a basic Tauri API call.

To instead communicate from the backend back to the frontend, Tauri uses the principle of signals:

```
fn calculate_progress(&mut self, last_percentage: &mut f64, file_size: u64) {
    self.scanned_size = self.scanned_size + file_size;
    let scanned_percentage = (self.scanned_size as f64 / self.folder_size as f64 * 100.0).round();
    info!("Scanned: {}%", scanned_percentage);
    if scanned_percentage != *last_percentage {
        self.tauri_window.emit_all("progress", TauriEvent {message: scanned_percentage.to_string()}).unwrap();
        *last_percentage = scanned_percentage;
```

```
    }
}
```

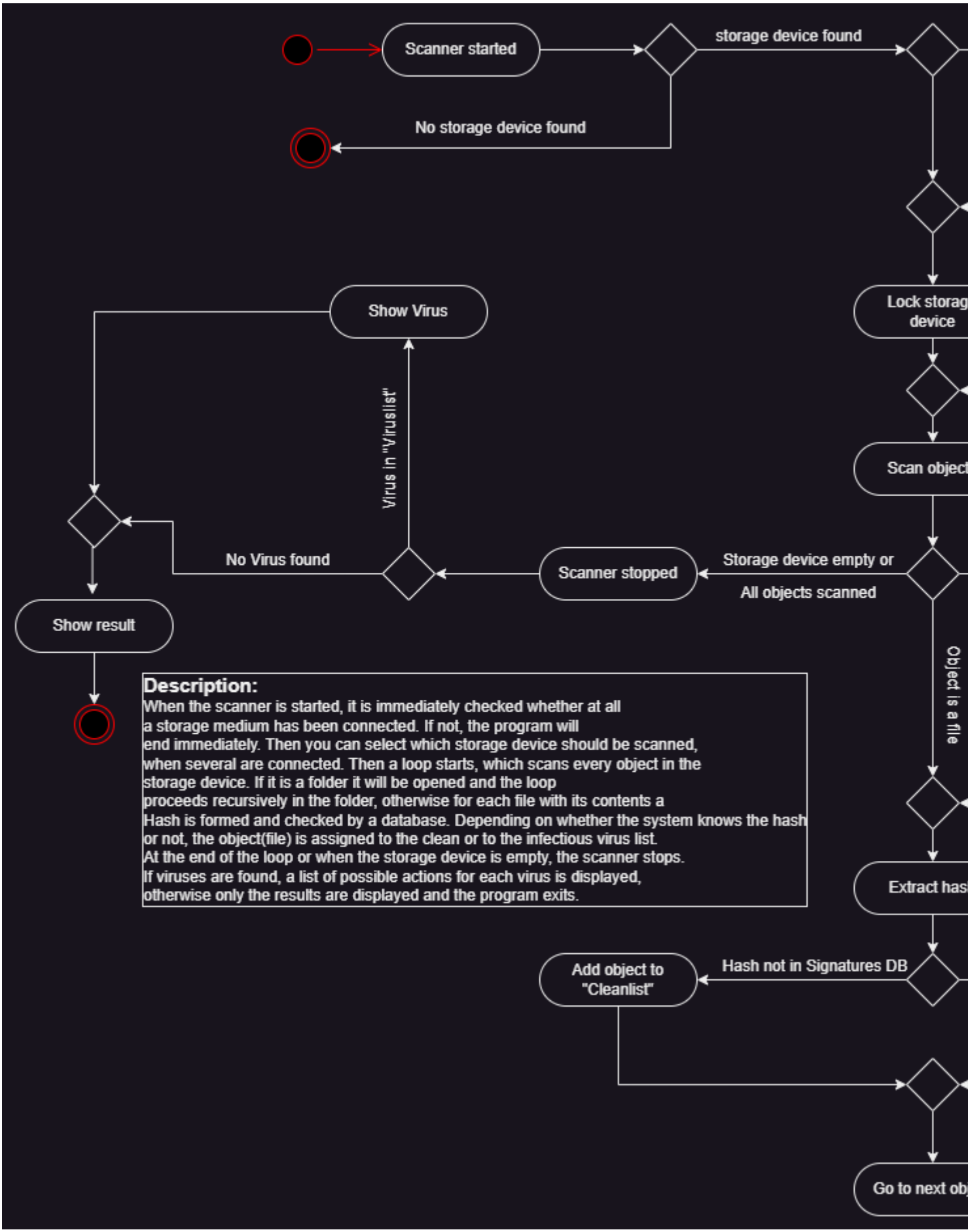
If you want to know more about how this internal API system exactly works, you can refer to the official Tauri guide.

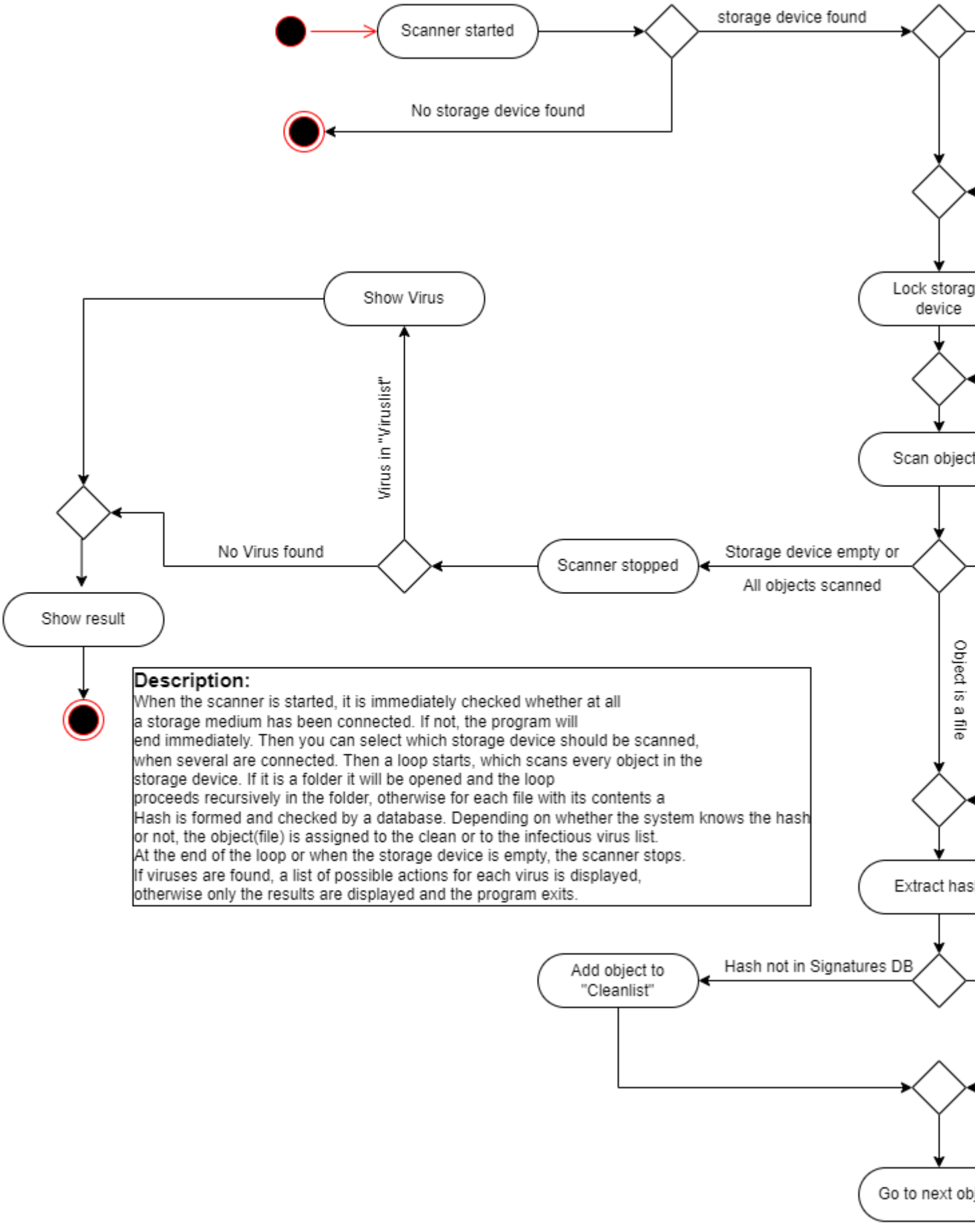
Last update: April 20, 2023
Created: April 5, 2023

7.1.3 Comments

7.2 Backend

7.2.1 Planing





Description:
When the scanner is started, it is immediately checked whether at all
a storage medium has been connected. If not, the program will
end immediately. Then you can select which storage device should be scanned,
when several are connected. Then a loop starts, which scans every object in the
storage device. If it is a folder it will be opened and the loop
proceeds recursively in the folder, otherwise for each file with its contents a
Hash is formed and checked by a database. Depending on whether the system knows the hash
or not, the object(file) is assigned to the clean or to the infectious virus list.
At the end of the loop or when the storage device is empty, the scanner stops.
If viruses are found, a list of possible actions for each virus is displayed,
otherwise only the results are displayed and the program exits.

This diagram describes how the logical flow of the app work. The exact flow can change in the future, but this was the initial planing of the app. As you can see, it is rather simple with just a couple if statements and a for-each loop. The backend is actually not that hard to understand, the complicated part is the more in-depth part of how to manage the database, scan files, create the hash and most importantly, communicate the result to the frontend by emitting signals.

7.2.2 Database

The Database consists of a single big file with mainly two columns. This file is generated at runtime and can be updated with a button on the GUI. At the time of writing, this file is approximately 4 GB in size and can take up to an hour to generate. The database contains in one column all signatures as a primary key, and in the second column the name of the file they got extracted from, based on the Virusshare database file names.

Structure

Here is how the database may look like:

MD5 Hash	FileNr
....	....
40610db6af6eaf2391b7a169e2540de9	00219
64a613db4aa368108e6d4c15ef7f6454	00219
....	....

Initialisation

Here is the Rust code in the main.rs file that creates the Database:

```
let mut use_db = "signatures.db".to_owned();
match dbfile {
    Some(fpath) => {
        if Path::new(&fpath).to_owned().exists() && Path::new(&fpath).to_owned().is_file() {
            info!("Using specific DB path {}", fpath);
            use_db = fpath.to_owned();
        } else {
            info!("Falling back to default DB file (signatures.db)");
        }
    }
    None => {
        info!("Path is None; Falling back to default DB file (signatures.db)");
    }
};
```

If the specified `signatures.db` file doesn't exist, it will automatically attempt to create on and even create the necessary table. You can technically also provide your own file, just put it in the same folder as the executable. Also make sure that the table is the same.

7.2.3 Scanner

The scanner class is the main function of the entire app. It takes a directory as parameter and starts scanning it. It will scan all folders and files recursively by hashing it and checking the hash in the database.

Ignoring hashes (false positives)

In the `file_scanner.rs` file, where the scanner is declared, we also have an Array that contains signatures that should be ignored. Virusshare is a databse where Hashes are mostly added, not removed, and therefore we need to do our filtering on the client-side. For example there was a hash for empty files, but in our opinion empty file should not be flagged as dangerous, so we added an excemption:

```
let false_pos: Vec<String> = vec!["7dea362b3fac8e00956a4952a3d4f474".to_owned()];
```

Obfuscated mode

Sometimes, for privacy reasons, you might not want to display the path and names of the found files, therefore the scanner provides a so-called 'Obfuscated mode'. Setting this to true will make the app fail immediately as soon as it finds a vairus, without scanning the entire folder, and output red or green depending on the security of the scan.

7.2.4 Logging

The app also has a logger that keeps track of issues or warnings during the execution of the app. Sadly this only works in `dev` or `debug` mode. When packing ito an executable the logger is lost as it gets stripped away for better performance. We are nonetheless working on bringing it back to life in the future.

Last update: April 28, 2023
Created: April 3, 2023

7.2.5 Comments

7.3 Frontend

7.3.1 Planing

The planing of the Raspirus frontend happened on an external Website called Figma. The styling changed a bit and there are some more functions now, but the logical structure is roughly the same. Each screen you see is a single page on Next.js. Furthermore, the pop-ups have been replaced with ones from SweetAlertv2. You can explore the Figma project in the iFrame below, else if nothing is showing up, you can look at the project here.

7.3.2 Screenshots

7.3.3 Pages

The entire project is structurized in components and pages. The frontend is basically a normal website, so it has routing, links and so on. We have one page for each important action of the app. There is one for the scanning process, for the settings, for the informations, ... Pages link to each other and apart from the scanning and home page, they all have a button to return to the home-page. In our case we have all pages double, this has to do with an issue in the translation plugin. So now the actualy page resides inside the [lang] folder and the pages outside of it are simply used for routing. Note that there are some special pages, like the app.js and the document.js page that have a special structure. To learn more about them, visit the Next.js page about this topic

7.3.4 Components

Components are used to store repeated or cluttered code. This helps in keeping the code of the pages clean and move the focus of certain code outside of them. Each component should only fulfill a single task. For example it should display a single card in the settings page. Both components and pages are stored in different directories at root and can be imported. Only import components into pages, not the other way around.

7.3.5 Localizing

The frontend is also translated into different languages. The languages can be found in the public directory, where each langauge has its own folder with a json file. The JSON files are built on a key-value basis. The keys always need to be the same on all JSON files and the value gets translated. Make sure to look at the app before blindly translating, as some things might change depending of the context. For the translations, we used the Nextjs feature i18n, which works fine but is a bit complicated to setup with Tauri. If you want to add new languages, make sure to specify it in your pull-request or in the issue, as we also need tu update some of the code to cover the changes. For example we need to update the button on the main page to display the new language, this is not automatic. Here is how localizing looks like on the frontend:

```
Swal.fire(t("usb_list_error"), t("usb_list_error_msg"), "error");
```

In this case we are calling a SweetAlert with a title and a message. The important part is the `t(key)` function. The function will return the text associated with the given string key, if it is able to find it in the JSOn file.

7.3.6 Invoke Rust

The frontend doesn't do any calculations, because that's what we have the Rust backend for. Rust is much faster, and also has the ability to actually access local files. But to let the frontend and the backend communicate with each other we need Tauri. Tauri offers us an API to call functions from Rust and wait for a result on the frontend. It basically works like a Promise. Here an example:

```
invoke("list_usb_drives", {})
  .then((output) => {
    setDictionary(JSON.parse(output));
    setTimeout(() => {
      refreshButton.classList.remove(styles.refreshStart);
    }, 3000);
  })
  .catch((error) => {
    console.error(error);
    refreshButton.classList.remove(styles.refreshStart);
    Swal.fire(t("usb_list_error"), t("usb_list_error_msg"), "error");
  });
```

As you can see, we use a Tauri function called `invoke()` that calls a Rust function and then awaits for it result or catches a possible error.

Last update: April 28, 2023
Created: April 3, 2023

7.3.7 Comments