Raspirus docs

A simple hash-based virus scanner

Benjamin Demetz

Copyright © 2022 - 2023 Raspirus Project

Table of contents

Copyright © 2022 - 2023 Raspirus Project

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 virusscanner 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. - ClamAV: 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 ClamAV 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 continuosly. 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 always will 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 PotatoPC

Last update: April 20, 2023
Created: April 3, 2023

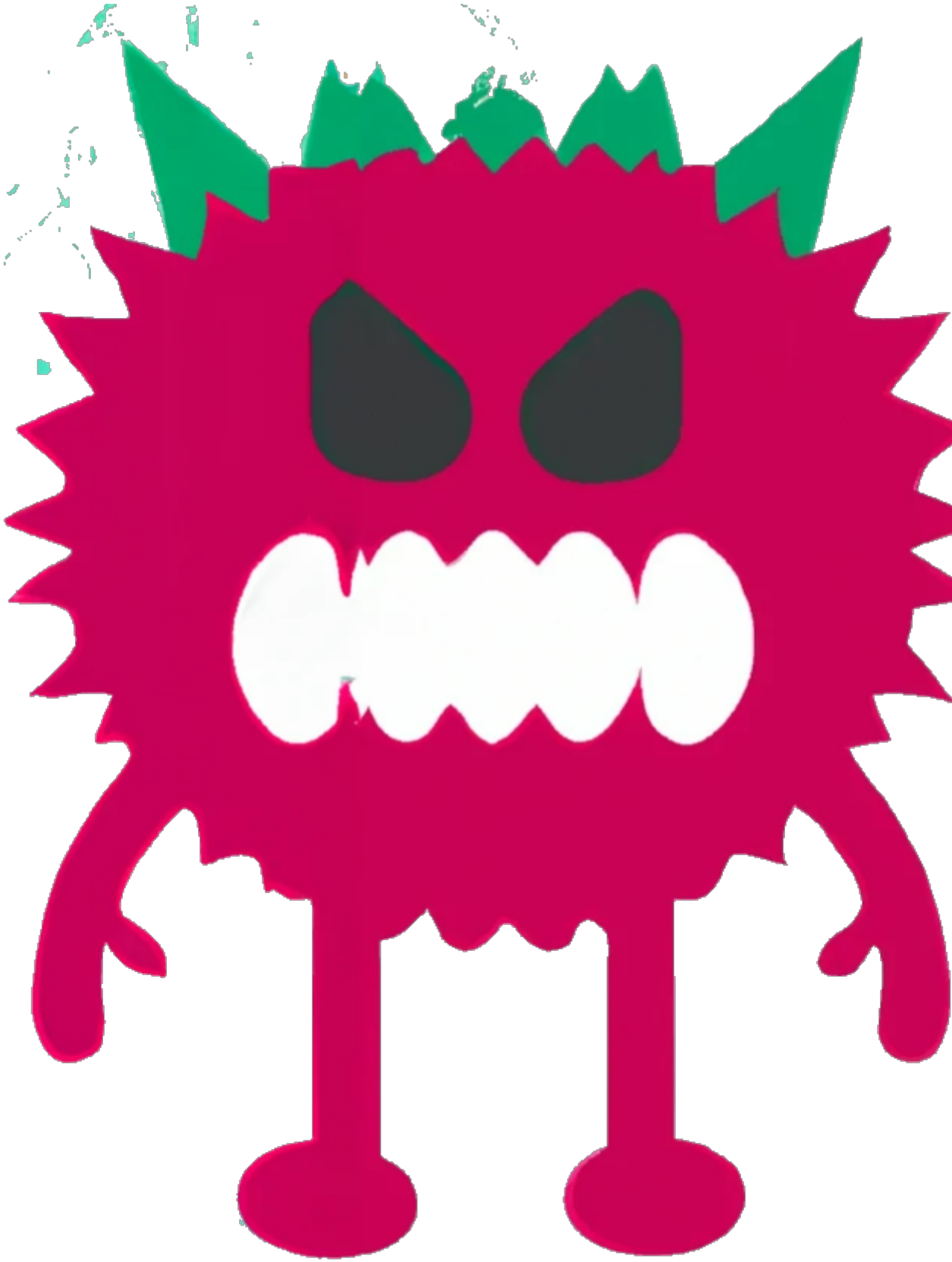
Copyright © 2022 - 2023 Raspirus Project

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 developemnt or usage. Maybe your error can be easily fixed and doesn't require a bug report.

Copyright © 2022 - 2023 Raspirus Project

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

Copyright © 2022 - 2023 Raspirus Project

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 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"
    ]
}
```

2.4 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

Copyright © 2022 - 2023 Raspirus Project

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 7, 2023
Created: April 3, 2023

3.3 Comments

Copyright © 2022 - 2023 Raspirus Project

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

Copyright © 2022 - 2023 Raspirus Project

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

Copyright © 2022 - 2023 Raspirus Project

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 7, 2023
Created: February 7, 2023

4.6 Comments

Copyright © 2022 - 2023 Raspirus Project

5. Usage

5.1 Tutorial

How to use this app. It is super easy

5.2 Case study

COMING SOON - Some metrics about the projects - Speed - Reliability - Functions, where to use this project

---

Last update: April 20, 2023
Created: April 3, 2023

5.3 Comments

Copyright © 2022 - 2023 Raspirus Project

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 7, 2023
Created: April 5, 2023

6.1.4 Comments

Copyright © 2022 - 2023 Raspirus Project

6.2 Code

6.2.1 COMING SOON

Last update: April 5, 2023
Created: April 3, 2023

6.2.2 Comments

Copyright © 2022 - 2023 Raspirus Project

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 7, 2023
Created: April 5, 2023

Copyright © 2022 - 2023 Raspirus Project

6.3.3 Comments

Copyright © 2022 - 2023 Raspirus Project

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 customisation 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 an 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 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 a 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 viceversa.

Copyright © 2022 - 2023 Raspirus Project

- 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 structurizing 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 ec´verything 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 duable. Therefore we setup 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;
```


Copyright © 2022 - 2023 Raspirus Project

```
    }
}
```

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 19, 2023
Created: April 5, 2023

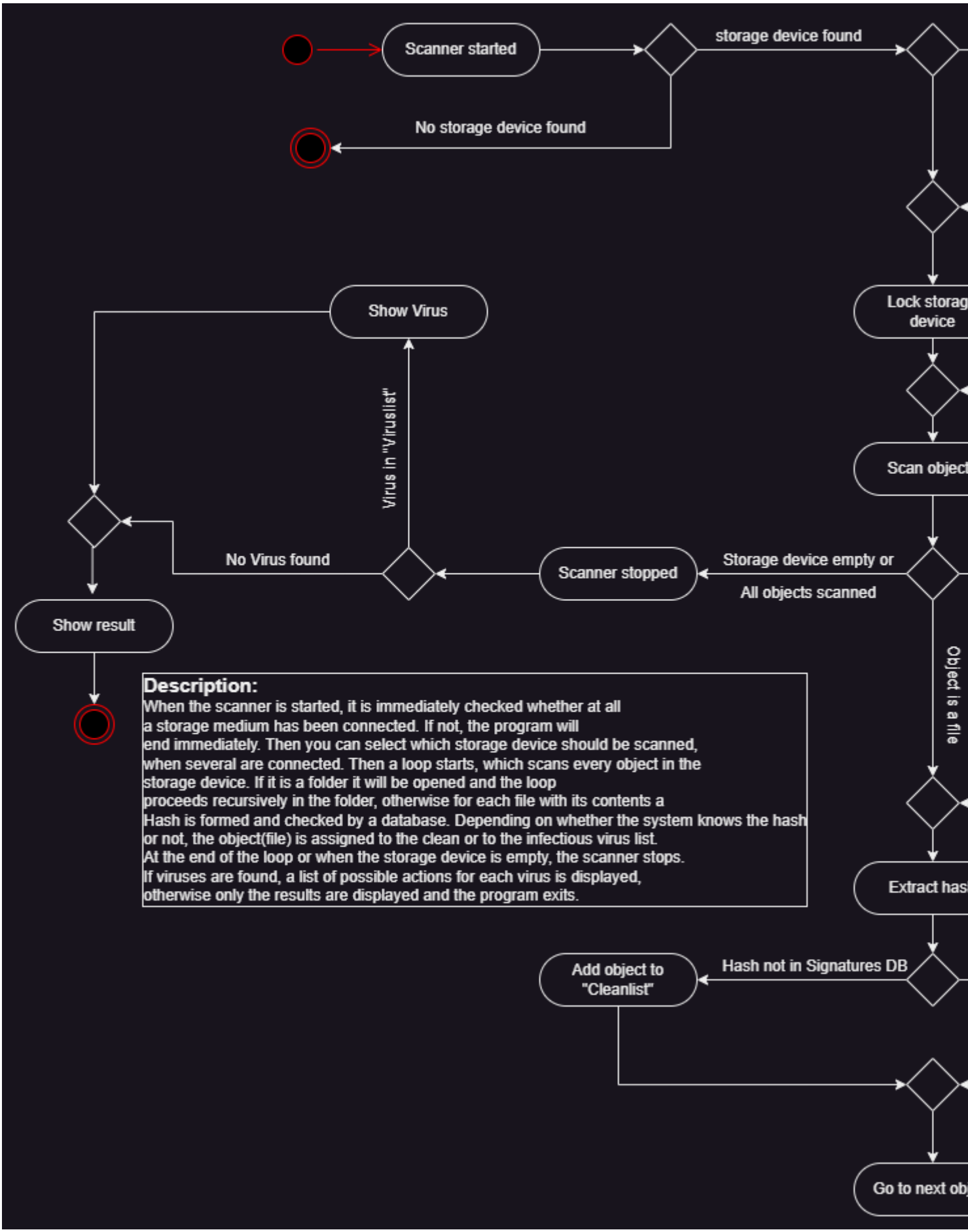
7.1.3 Comments

Copyright © 2022 - 2023 Raspirus Project

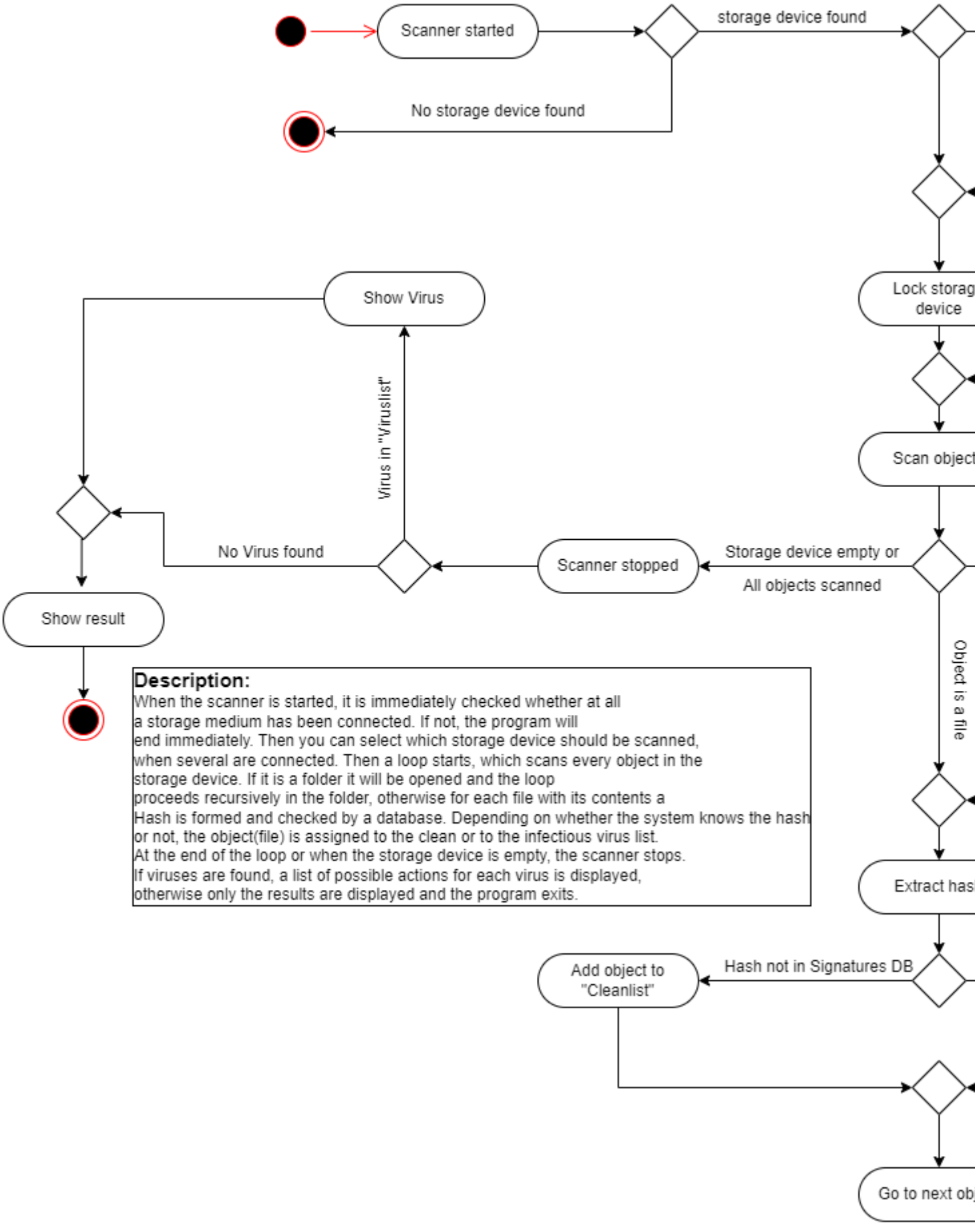
7.2 Backend

7.2.1 Planing

Copyright © 2022 - 2023 Raspirus Project



Copyright © 2022 - 2023 Raspirus Project



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.

Copyright © 2022 - 2023 Raspirus Project

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, comunicate the result to the frontend by emitting signals.

7.2.2 Database

7.2.3 Scanner

7.2.4 Logging

Last update: April 20, 2023
Created: April 3, 2023

7.2.5 Comments

Copyright © 2022 - 2023 Raspirus Project

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

7.3.4 Components

7.3.5 Localizing

7.3.6 Invoke Rust

Last update: April 20, 2023
Created: April 3, 2023

7.3.7 Comments

Copyright © 2022 - 2023 Raspirus Project