Raspirus docs

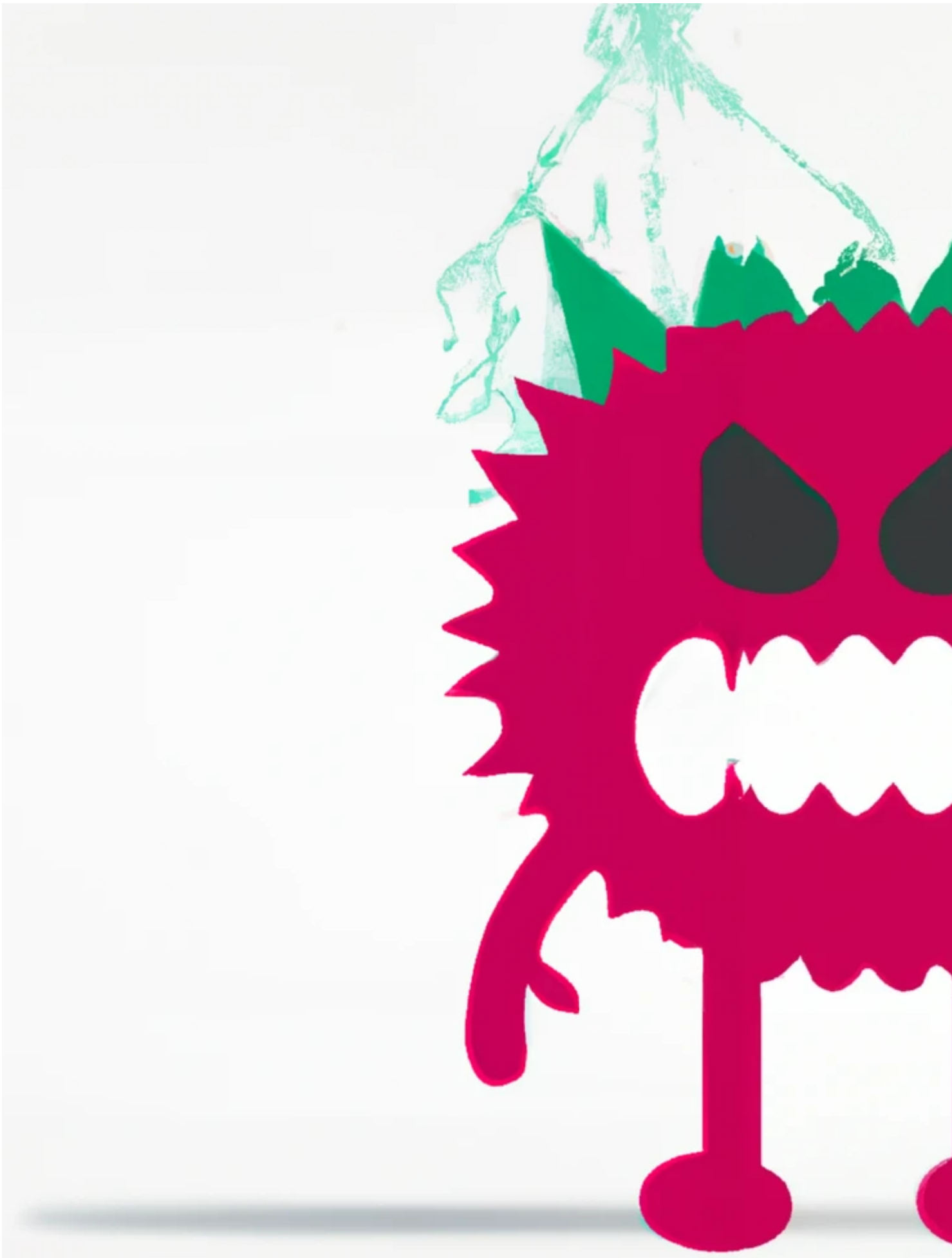
A simple hash-based virus scanner

Benjamin Demetz

Table of contents

1. Home

1.1 Welcome to the Raspirus Documentation



1.2 Table of Contents:

1.3 Introduction

Welcome to the Raspirus project documentation. This repository serves as a comprehensive guide to the Raspirus project. Please note that the project is currently under development, and while we strive for accuracy, some information may be subject to change.

1.4 Related Projects

Raspirus is a simple and efficient virus scanner, specifically designed for compatibility with single-board computers like the Raspberry Pi. While there are several similar projects available, Raspirus stands out for its unique features and advantages. Here are some notable comparisons:

- Clam AV: Clam AV is an open-source antivirus program that can also perform file and folder scanning similar to Raspirus. However, it is known to be resource-intensive and relatively slow. Due to its high memory requirements, it may not be suitable for deployment on low-spec systems like the Raspberry Pi. Nonetheless, Clam AV remains a powerful open-source tool.

- Windows Defender: Windows Defender is a comprehensive security program for Windows that continuously scans the entire system for threats. However, this continuous scanning process can significantly impact system performance. Additionally, Windows Defender is limited to the Windows operating system, whereas Raspirus is designed to be cross-platform, offering compatibility with various operating systems.

- Bitdefender: Bitdefender is a feature-rich commercial antivirus software solution. While it offers robust security capabilities, it comes at a cost. Raspirus, on the other hand, is committed to being a free and open-source project. We believe that security should be accessible to all users without the need for financial barriers. Furthermore, Raspirus aims to foster a collaborative community, leveraging the collective expertise of developers worldwide.

While there are numerous other antivirus solutions available, it's important to note that Raspirus does not seek to outperform or compete directly with other antivirus software. Instead, its primary focus is to excel at a specific task: comparing file hashes against a list of signatures.

Key features of Raspirus include: - Free and open-source nature - Transparent codebase for community review and scrutiny - Emphasis on community-driven growth and development - Cross-platform compatibility as a standard feature - Lightweight and fast performance, ensuring usability even on low-spec devices

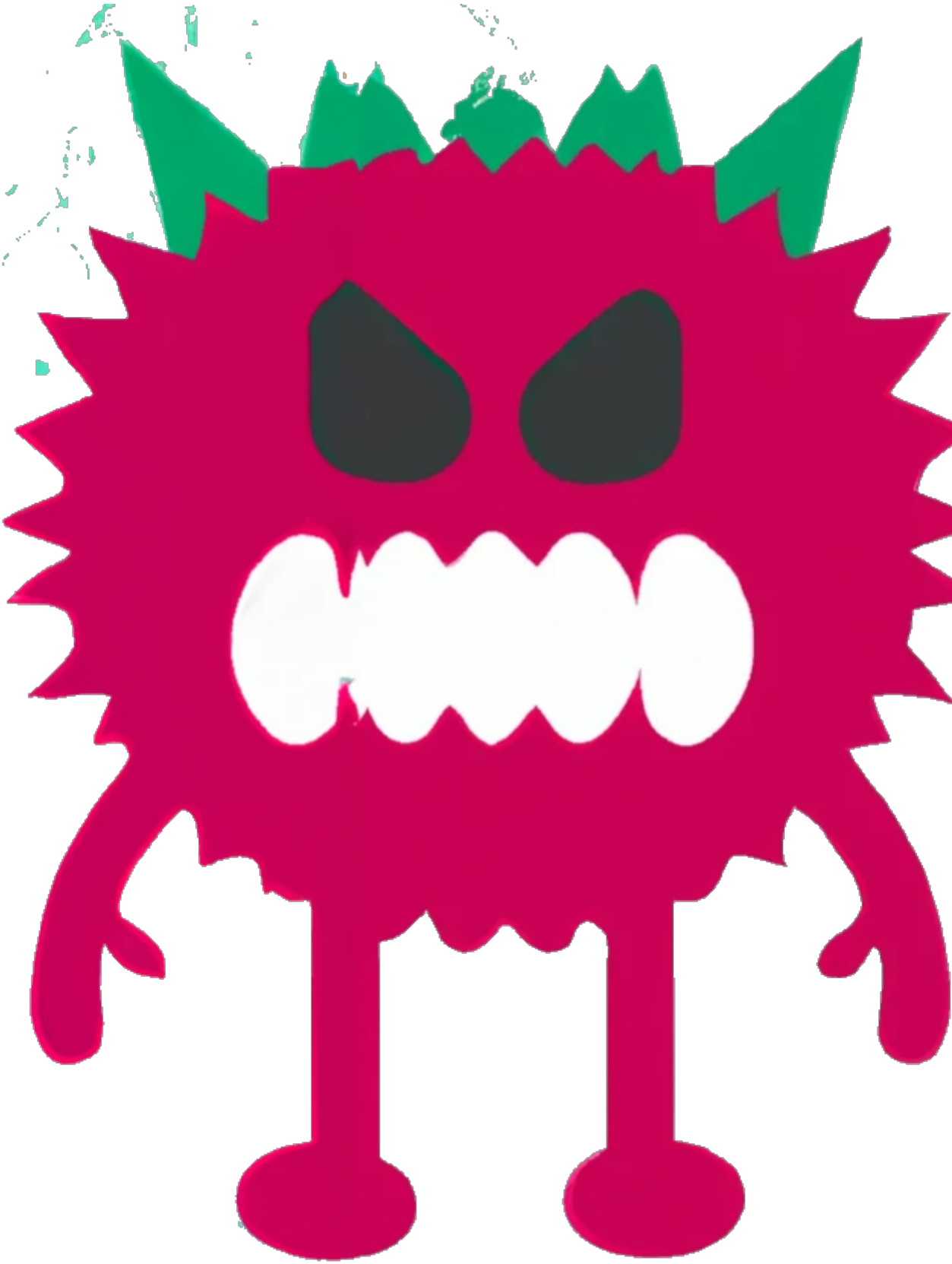
We invite you to explore the documentation further to gain a comprehensive understanding of the Raspirus project and its capabilities.

Last update: May 31, 2023
Created: April 3, 2023

2. FAQ

Welcome to our FAQ section, where we address some of the most frequently asked questions. If you encounter any errors or have queries during development or usage, this page can provide you with useful information. We continually update this section as new questions arise.

2.1 What is the project's icon?



The logo of the Raspirus app features a red monster named Stuart, who is designed to represent a virus-eating creature. The logo was generated using DALL-E, along with creative image editing and merging. Stuart is a friendly monster, except when he's hungry for viruses. You can find additional media and documents in the dedicated repository. Feel free to use these images to create your own artwork and share them in the discussion boards.

2.2 How can I generate the documentation for this repository?

The generated documentation can be found in the rust folder. If you want to generate your own documentation, you can use the `cargo doc` command. Here are some parameters you may find useful:

- `--no-deps`: Ignores dependencies and only documents the code itself.
- `--release`: Generates documentation optimized for release builds.
- `--target-dir`: Specifies the output directory for the documentation.

Putting it all together, the command might look like this:

```
cargo doc --no-deps --release --target-dir=/docs/generated/
```

2.3 How do I set up Rust Analyzer in VS Code to work with a non-standard directory structure?

The Rust Analyzer plugin in Visual Studio Code searches for a `Cargo.toml` file in the current directory or its parent directory. However, in our case, since the entire application is packed in the `app` directory, the plugin may not work as expected. To address this issue, you can add an option to the plugin settings and specify the location of your `Cargo.toml` file.

As mentioned in this comment, you can add the following lines to the end of your plugin settings JSON. Afterward, restart the Rust Analyzer for the modifications to take effect.

```
{
    "rust-analyzer.linkedProjects": [
        "/home/stuart/raspirus/raspirus/Cargo.toml"
    ]
}
```

2.4 The app crashes when updating the database

On Windows, it has been observed that the app crashes when attempting to update the database. We are aware of this issue and actively working to resolve it. The problem arises because the update function requires administrative privileges, which Windows does not automatically provide. To temporarily resolve this issue, you can execute the app with administrative privileges. Right-click on the app and select "Run as administrator" to launch it with the necessary privileges.

Last update: May 31, 2023
Created: April 5, 2023

2.5 Comments

3. Guides

Here you will find helpful guides on how to export the documentation to PDF format and how to contribute translations to this project. If you have any additional requests for guides, please leave a comment below!

3.1 Export to PDF

If you would like to have an offline version of this documentation in PDF format, you can follow these step-by-step instructions:

1. Clone this repository by following the instructions on GitHub.
2. Install the necessary requirements for the PDF conversion tool. You can find the requirements specific to your operating system here.
3. Navigate to the cloned directory and install the required dependencies. Please note that you will need Python3 and pip for this step. You can install the dependencies by running the following command: `pip install -r requirements.txt`.
4. Install mkdocs and execute the build command: `mkdocs build`.
5. If everything goes smoothly, the resulting PDF file should be located in the `site/pdf` directory with the name `document.pdf`.

Please be aware that the exported PDF may have some issues with images and iFrames, but the text should be readable and suitable for sharing offline.

3.2 Translations

We welcome contributions to translate this documentation into other languages. If you are interested in translating the document, you can find the necessary information and resources at the following link: Translate this document.

Your contributions are highly appreciated, and they will help make this documentation more accessible to a wider audience.

Last update: May 31, 2023
Created: April 3, 2023

3.3 Comments

4. Installation

This guide provides instructions on how to install the Raspirus application on your machine.

4.1 Using Executables

The quickest and simplest method to install the application is by using one of the executables available on the Release page. Once you have downloaded the executable, the installation process may vary depending on your system. For example, on Windows, you can double-click the executable and follow the installation instructions provided by the wizard. Once completed, the installation process is finished.

4.2 Using the Makefile (Debian-based Systems Only)

The project includes a Makefile that facilitates installation, updating, and testing. To install the application, you can clone the repository and execute the Makefile using the following command:

```
make install
```

4.3 Limitations

- Glibc can cause issues on Linux. Please refer to the limitations section of the Tauri documentation for more information.
- The application requires a 64-bit system. Running it on a different system architecture may result in crashes due to the utilization of memory improvements specific to 64-bit systems.

4.4 Step-by-Step Guide

Before executing each step, we recommend reading the entire guide thoroughly.

4.4.1 Step 1: Clone the Repository

If you have `git` installed, use the following command to clone the repository:

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

Alternatively, you can download the repository as a ZIP file from here and extract its contents. The GitHub guide provides further assistance on this process.

4.4.2 Step 2: Install Rust

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

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

For **Windows**: Download the Rust executable from the Rust website and follow the installation instructions.

4.4.3 Step 3: Install NPM

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

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

For **Windows**: Visit the NPM Website to download the NPM executable and proceed with the installation.

4.4.4 Step 4: Install Next.js

Use the following command with npm to install Next.js and its dependencies:

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

4.4.5 Step 5: Install Tauri

Tauri is the framework that connects the Rust backend with the Next.js frontend. Follow these steps to install Tauri:

1. Make sure you meet the Prerequisites specified by Tauri.
2. Install Tauri using cargo with the following command:

```
sh cargo install tauri-cli
```

Note: We primarily work with cargo in this guide. Refer to their FAQ section to learn about the alternative NPM installation.

4.4.6 Step 6: Install Project Dependencies

To install the project dependencies, follow these steps:

1. Navigate to the directory that contains the Raspirus code.
2. Open a terminal in the `Raspirus` directory.
3. Execute the command `npm install` to install the necessary node modules.
4. Note: If you encounter an "OpenSSL is missing" error on WSL, resolve it by editing the file `Raspirus/src-tauri/Cargo.toml`. Add the following line to the `[dependencies]` section: `openssl-sys = {version = "0.9.66", features = ["vendored"]}`.

4.4.7 Step 7: Build the Project

Before building the project, ensure the Rust part works correctly (optional):

1. Open the folder `Raspirus/src-tauri/`.
2. Run the command `cargo build`.
3. If the command succeeds, return to the `Raspirus/` directory.
4. If the command fails, please open an issue on this repository, providing detailed information about the error.

To build the entire application:

1. In the `Raspirus/` directory, execute the command `cargo tauri build`.
2. After the process completes, the executable file's path will be displayed.
3. By default, the executable is located in the `Raspirus\src-tauri\target\release` folder.

4.4.8 Step 8: Install the Raspirus Debian Package (for Debian-based Systems)

For Debian-based systems, an alternative installation method is available using the `.deb` package generated in the `Raspirus/src-tauri/target/release/bundle/deb` folder. Follow these steps to install the package:

1. Open a terminal in the `Raspirus/src-tauri/target/release/bundle/deb` directory.
2. Execute one of the following commands:
3. Using `apt`:

   ```sh
   sudo apt install ./raspirus-x.x.x_amd64.deb
   ```

4. Using `dpkg`:

   ```sh
   sudo dpkg -i raspirus-x.x.x_amd64.deb
   ```

Replace `x.x.x` with the actual version number of the Raspirus package.

4.4.9 Step 9: Execute the Application

After successfully building and installing the Raspirus application, you can execute it using the following command:

```
raspirus
```

Executing this command will launch the Raspirus application. Congratulations! You have now completed the installation process and can begin using the Raspirus app.

4.5 Conclusion

The Raspirus application combines a website interface with underlying Rust code, packaged using the Tauri framework. To launch and display the website, a graphical overlay is required. As the project is continually evolving, we encourage you to open an issue on the repository if you encounter any unusual behavior, have suggestions, or discover errors. We are here to assist you.

Last update: June 21, 2023
Created: February 7, 2023

4.6 Comments

5. Usage

5.1 Tutorial

Using the Raspirus app is straightforward and user-friendly. Once you have installed the application, it requires only a few simple steps to perform a scan. The app is designed with a focus on usability for touchscreens and does not require any input devices or keyboards to operate. Here's a step-by-step guide:

1. Open the app from the main page.
2. Insert a USB drive and select it, or click the folder icon to choose a local folder for scanning.
3. Carefully review and accept the user agreement.
4. Allow the app to perform its scanning process.
5. The results will be displayed, and you can initiate a new scan if necessary.

5.2 Case Study

Let's delve into some key metrics and specifications of the Raspirus app, evaluating its speed, reliability, and functionality. Additionally, we will outline the intended use of the application.

5.2.1 Specifications

- App size: 5MB
- RAM usage: Approximately 50MB
- Works offline, requiring an internet connection only for database file updates

5.2.2 Performance Metrics

The scanning speed of the app depends on three factors: - The storage medium (USB or local) used for scanning. - The CPU performance of the system running the app. - The file size of individual files being scanned.

For optimal scanning speed, it is recommended to utilize modern storage media with high read speeds, employ a capable CPU that is not overwhelmed with other applications, and focus on scanning smaller files. Scanning 100 files of 10GB each will generally be faster than scanning a single 10GB file, as the hashing function performs more efficiently on smaller files. For Raspberry Pis, CPU speed is of greater importance than RAM capacity, as the app only requires a few megabytes of memory.

5.2.3 Application Scope

Initially developed for Raspberry Pis with touchscreens, the Raspirus app offers easy setup and high security on Linux-based systems. Given that most viruses target Windows PCs, Linux-based Raspberry Pis are inherently less vulnerable. Additionally, by configuring a Raspberry Pi with a battery and enabling Kiosk mode, it is possible to create a portable Raspirus scanning box specifically tailored for scanning USB drives.

Thanks to Tauri's cross-compilation feature, the Raspirus app is now available for macOS, Windows, and Linux. While primarily developed and tested on Windows and Debian Linux, installation instructions for various operating systems can be found on the Installation page.

Last update: May 31, 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 Media Resources

In the Raspirus project, we recognize the importance of visual elements to enhance the user experience, provide clarity, and make documentation more engaging. We maintain a dedicated repository, Raspirus/media, which hosts various media resources, including images, artworks, diagrams, PDF documents, and more.

6.3.1 Accessing Media Resources

To access the media repository and explore the available resources, follow these steps:

1. Visit the Raspirus/media repository on GitHub.
2. Browse through the repository to discover images, artworks, diagrams, PDF documents, and other visual assets that may be relevant to your needs.

6.3.2 Utilizing Media Resources

Feel free to utilize the media resources from the Raspirus/media repository in your contributions to the project. Whether you're creating documentation, designing user interfaces, or adding visual elements to enhance communication, these resources can greatly assist in conveying information effectively.

6.3.3 Contributing Your Own Visual Assets

We highly encourage community members to contribute their own artwork, images, diagrams, or other visual content to the project. By doing so, you can help enhance the visual appeal of Raspirus and contribute to the overall project experience.

To contribute your own visual assets, follow these steps:

1. Fork the Raspirus/media repository to create a personal copy in your GitHub account.
2. Add your visual assets to the appropriate folders within the repository, ensuring proper organization and categorization.
3. Submit a pull request from your forked repository to the original Raspirus/media repository, indicating the changes you've made and providing a brief description of your contribution.

Once your pull request is reviewed and accepted by the team, your visual assets will be added to the media repository and made available to other contributors.

Feel free to customize and expand on this template to suit your specific needs. It's important to provide clear instructions, guidelines, and encouragement for contributors to utilize and contribute to the media resources effectively.

Last update: May 31, 2023
Created: May 31, 2023

6.3.4 Comments

6.4 Code Contributions

Contributions to the codebase are always welcome and highly valued in the development of Raspirus, a powerful virus scanner. If you're interested in contributing code, please take a moment to familiarize yourself with our Guidelines before opening a pull request. Additionally, we recommend exploring the coding practices and standards outlined in the Developer section for a comprehensive understanding of our project's structure and requirements.

6.4.1 Contributing to the Frontend

The frontend of Raspirus is built using the Next.js framework and leverages JavaScript for its implementation. To maintain a clean and efficient codebase, we prioritize modularizing components and ensuring that pages remain uncluttered with unnecessary code. Each page represents a distinct screen within the app and can be accessed either through manual routing or as a result of specific user actions, such as clicking a button or being redirected after the completion of a loading process.

When contributing to the frontend, consider the following guidelines:

- Emphasize clean and intuitive user interfaces, allowing users to navigate the app effortlessly.
- Utilize the Next.js structure effectively, leveraging the power of components for enhanced maintainability and reusability.
- While the frontend is currently developed exclusively in JavaScript, we remain open to migrating to TypeScript in the future.

Your contributions to the frontend will significantly impact the user experience of Raspirus, making it more accessible and user-friendly.

6.4.2 Adding to the Backend

The backend of Raspirus is implemented in Rust, a high-performance programming language renowned for its speed and reliability. By utilizing Rust, we achieve faster scanning speeds and enhance the overall robustness of the application. The backend interacts with a sizable database, locally storing hashes, and communicates responses via the Tauri API.

If you're interested in contributing to the backend, consider the following points:

- Familiarize yourself with Tauri, a framework that facilitates the integration of Rust and JavaScript, enabling seamless communication between the frontend and backend.
- Explore Rust's features and capabilities, which provide an efficient and secure foundation for the virus scanning process.
- Although learning Rust and Tauri might seem daunting at first, with a little practice, you can quickly grasp the concepts and contribute effectively to the backend development.

Contributing to the backend of Raspirus will help improve its scanning capabilities and ensure the reliability of the application.

We appreciate your interest in contributing code to Raspirus, and we look forward to your valuable contributions that will help make our project even better.

Last update: May 31, 2023
Created: April 3, 2023

6.4.3 Comments

6.5 Translators

Translations play a vital role in making the app accessible to users worldwide. If you are fluent in a language other than English and would like to contribute by providing translations, we welcome your assistance. There are two main areas where you can contribute: translating code and translating documentation.

6.5.1 Translating Code

Translating the app's frontend strings is an essential task that allows users to interact with the app in their native language. There are two possible ways to handle translations within the codebase:

Translation File

Each language has its own translation file in JSON format. The app loads the required strings from these files. While this approach is relatively simpler for developers to implement, it may not be the most efficient or user-friendly for translators, as it involves forking the repository and manually editing the JSON file.

Translation Service

To streamline the translation process and provide a better experience for translators, we plan to integrate an external translation service. This service would enable translators to focus solely on translating the strings and provide an overview of the progress. We are considering using Crowdin as the preferred translation service. However, the setup for this option is still being worked on, and it will be added in the future as the team expands and we have more concrete plans.

6.5.2 Translating Documentation

The documentation itself also requires translations to make it accessible to users who are more comfortable in languages other than English. Translating the documentation is a more significant undertaking compared to translating frontend strings. If you are eager to contribute translations to the documentation, follow these steps:

1. Fork this repository: You can find a comprehensive guide on how to fork a repository here.
2. Navigate to the `docs` folder in the forked repository. If the language you want to translate to doesn't exist yet, create a new directory with the name of the language you wish to translate into (e.g., `it` for Italian, `de` for German).
3. Enter the language directory and start editing the Markdown files while maintaining the original folder structure.
4. Once you have finished the translations, save your files and upload them to your forked repository. You can commit your changes following the GitHub commit guide.
5. Finally, open a pull request to the original repository, proposing your changes to be added to the main project.
6. A member of our team will review your translations and, if accepted, merge them into the main project.

We understand that this manual process might be cumbersome, but rest assured that in the future, we plan to integrate an external service like Crowdin. This service will simplify the translation workflow, allowing you to focus solely on your translations without any distractions.

We appreciate your contribution to making the app accessible to a global audience through your translations. Thank you for your support!

Last update: June 7, 2023
Created: April 5, 2023

6.5.3 Comments

7. Developers

7.1 Developers

The Raspirus project employs two distinct programming languages and frameworks: one for the frontend and one for the backend. The frontend utilizes JavaScript with the Next.js framework, while the backend is built with Rust. Communication between these languages is facilitated by the Tauri framework, enabling seamless integration of Rust functions in the frontend.

7.1.1 Technologies Used

- NPM: To achieve an aesthetically pleasing frontend, I explored native application development options initially. However, the customization capabilities fell short of my expectations. Consequently, I opted for a web-based frontend and selected Node.js over Deno due to my familiarity with Node.js.
- Next.js: Next.js, a frontend framework, proved to be easy to implement and incredibly powerful once set up. The project extensively leverages Next.js's capability to export the website as static HTML, which is crucial for the functioning of Tauri. Additionally, Next.js optimizes the application by automatically eliminating unnecessary components, resulting in a lightweight and fast application. Navigation with Next.js is nearly instantaneous.
- Rust: While C++ was my initial choice for the backend, my proficiency wasn't sufficient to develop the entire application in that language. I turned to Python, my preferred language and one in which I am proficient, for faster development. However, Python's speed limitations became a significant drawback. Fortunately, a friend helped rewrite the entire backend in Rust, boosting the application's speed by 100x. Rust's compilation-based approach and performance superiority over interpreted languages like Python made it an ideal choice. If you are interested in the previous Python implementation, you can refer to the dedicated repository.
- Tauri: Tauri plays a vital role in this project by bridging the Rust backend with the JavaScript frontend. This allows us to have an elegant frontend with Next.js and an impressively fast backend with Rust. Furthermore, Tauri simplifies the compilation of the application into installers or binaries for different systems like Windows, Linux, or macOS, greatly enhancing the installation experience.
- SweetAlert2: When presenting errors, warnings, or information to the user as pop-ups, the default JavaScript alert lacks visual appeal. SweetAlert2 significantly enhances the appearance of these pop-ups. It is easy to install and use. In our case, SweetAlert2 is mainly used for displaying errors or warnings, such as when the scanning process is abruptly interrupted.
- next-i18next: Although the application contains minimal text and relies on icons and colors for clarity, we decided to incorporate translations. This is achieved using the Next.js plugin called i18next, which facilitates easy translation management through .JSON files.
- Crowdin: Crowdin is a website that aids in translating open-source and private projects, and I am particularly grateful that it offers free pricing for open-source projects. Crowdin allows translators to focus solely on translations without the need to examine code. It also streamlines the translation synchronization process between Crowdin and GitHub.
- MkDocs: MkDocs, a Python framework, is utilized to generate project documentation. In fact, this very translation you are reading was created using MkDocs. It is user-friendly and facilitates document structuring and integration with plugins like

Crowdin. - Dependabot: Dependabot, a built-in feature in GitHub repositories, ensures that dependencies are up to date by automatically checking for updates. This helps maintain an up-to-date application with the latest patches and functionality. By improving dependencies, the application's performance and efficiency can be enhanced. Moreover, Dependabot notifies users of any security vulnerabilities in the utilized dependencies or crates, ensuring the application's security. - GitHub: GitHub was selected as the file hosting platform due to its popularity, user-friendly interface, and comprehensive feature set. Additionally, it offers free hosting. Although a company suggested hosting the project on their GitLab servers, GitHub's ease of use and collaboration capabilities prevailed over the complexities of utilizing a VPN to connect to a PC that connects to GitLab. GitHub offers numerous features, including project planning, milestones, issues, and actions. - CodeQL: CodeQL, another powerful tool from GitHub, scans the project for vulnerabilities in the code. It detects issues like cross-site scripting, improper handling of passwords and secrets, and inefficient coding practices. CodeQL ensures a higher level of code reliability. - CodeCov: Codecov, an external tool installable on GitHub, tracks test coverage. It provides insights into the effectiveness of written tests and their coverage across the entire project. Although testing the frontend is challenging as it often involves user interaction, testing the backend is feasible. Therefore, we integrated Codecov to monitor test coverage in our Rust backend.

7.1.2 Integration

Raspirus is a self-contained application, operating solely within its environment. There are no external APIs for retrieving information, nor is the application designed for integration with other apps. The primary user interaction occurs via a touchscreen interface, eliminating the need for text input fields, which can be frustrating on low-quality touchscreens like the one found on a Raspberry Pi.

Although API calls are made, they are internal to the application. For instance, in the `loading.js` file, we invoke the scanning function from the frontend to initiate the scanner in Rust:

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

The `start_scanner` function is called with the required parameters, and the resulting message is stored in the `message` constant. This basic Tauri API call enables seamless communication between the frontend and backend.

To send data from the backend to the frontend, Tauri employs the signal principle:

```
fn calculate_progress(&mut self, last_percentage: &mut f64, file_size: u64) {
    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;
    }
}
```

For further details on how this internal API system operates, refer to the official Tauri guide.

Last update: May 31, 2023
Created: April 5, 2023

7.1.3 Comments

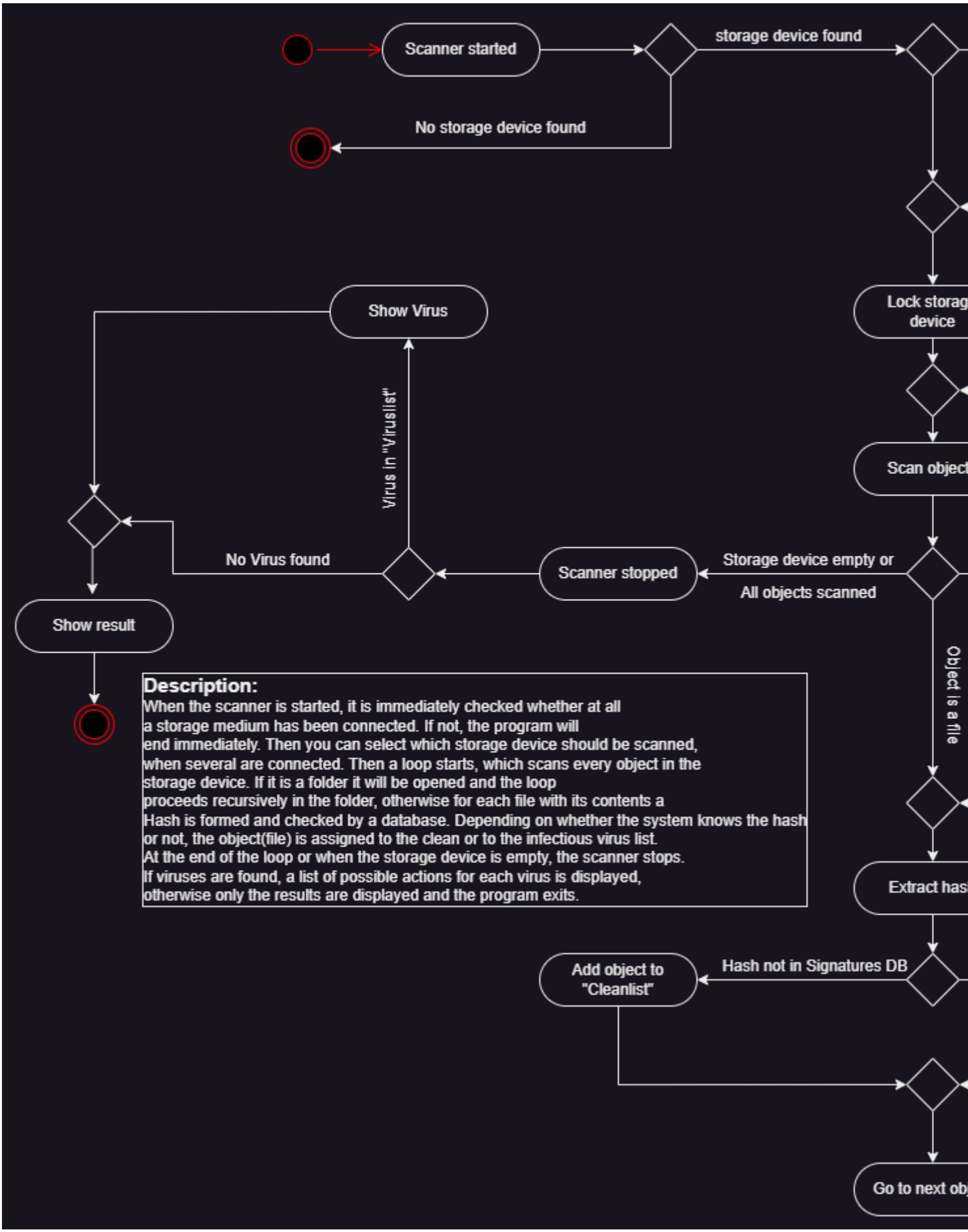
7.2 Backend

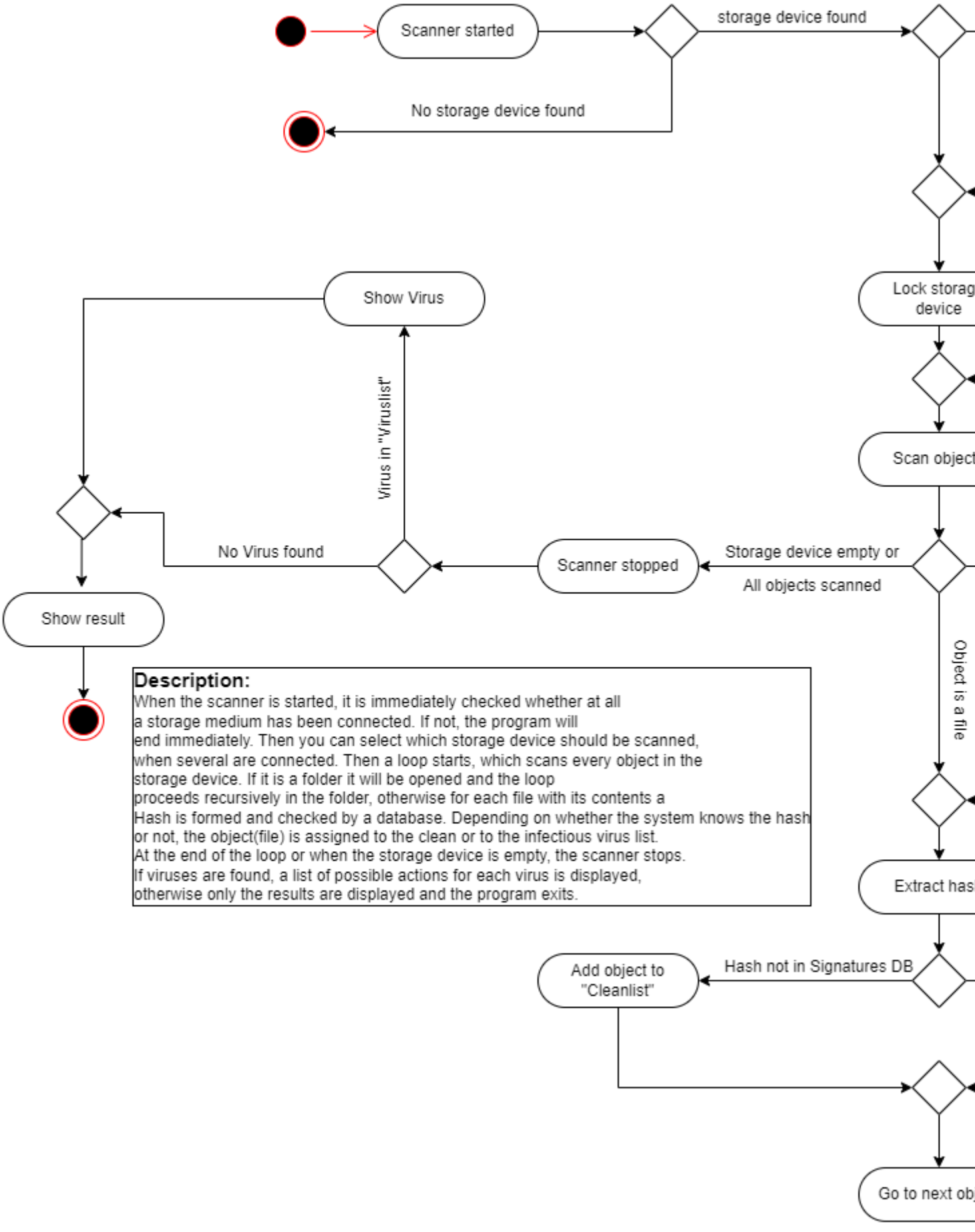
In the backend of the Raspirus app, several components work together to provide the desired functionality. This section outlines the planning, database structure, scanner, and logging aspects of the backend.

7.2.1 Planning

The activity diagrams below illustrate the logical flow of the app. While the exact flow may evolve over time, these diagrams represent the initial planning of the app. The backend follows a relatively straightforward design, utilizing if statements and a for-each loop. The complexity lies in managing the database, scanning files, creating hashes, and communicating the results to the frontend through signal emissions.

Activity Diagram:





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.

7.2.2 Database

The database consists of a single large file with two main columns. This file is generated at runtime and can be updated through a button on the GUI. At the time of writing, the database file is approximately 4 GB in size and can take up to an hour to generate. The first column contains all signatures as primary keys, while the second column contains the corresponding file names extracted from the Virusshare database file names.

Structure

The database structure is as follows:

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

Initialization

The following Rust code in the `main.rs` file 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, the code will attempt to create it and the necessary table. You can also provide your own file by placing it in the same folder as the executable. Just ensure that the table structure is the same.

7.2.3 Scanner

The scanner class is the core function of the app. It takes a directory as a parameter and starts scanning it. It recursively scans all folders and files by hashing them and checking the hash in the database for matches.

Ignoring Hashes (False Positives)

In the `file_scanner.rs` file, where the scanner is defined, there is an array that contains signatures to be ignored. Since Virusshare primarily adds hashes and rarely removes them, we need to handle filtering on the client-side. For example, if there is a hash for empty files, we added an exemption to prevent them from being flagged as dangerous:

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

Obfuscated Mode

To address privacy concerns, the scanner offers an "Obfuscated mode." When enabled, the app immediately fails upon finding a virus without scanning the entire folder

. It outputs red or green depending on the security of the scan, without revealing the path and names of the detected files.

7.2.4 Logging

The app incorporates a logger that tracks issues or warnings during execution. However, the logger only functions in `dev` or `debug` mode. When packing the app into an executable, the logger is stripped away for improved performance. Efforts are underway to reintroduce logging functionality in the future.

Last update: May 31, 2023
Created: April 3, 2023

7.2.5 Comments

7.3 Frontend

The frontend of the Raspirus app was planned using an external website called Figma. Although there have been some changes in styling and additional functions, the logical structure remains largely the same. Each screen in the app corresponds to a single page in Next.js. Additionally, the pop-ups in the original design have been replaced with ones from SweetAlertv2. You can explore the Figma project in the embedded iFrame below. If the iFrame doesn't load, you can access the project here.

7.3.1 Screenshots

(Screenshots are missing in the provided content.)

7.3.2 Pages

The entire project is structured into components and pages. The frontend of the app functions like a regular website, with routing and links between pages. There is a separate page for each important action in the app, such as scanning, settings, and information. All pages, except the scanning and home pages, have a button to return to the home page. In the case of this project, each page exists in duplicate due to an issue with the translation plugin. Therefore, the actual page resides inside the `[lang]` folder, while the pages outside of it are used for routing. It's important to note that there are special pages like `app.js` and `document.js` that have a specific structure. To learn more about these special pages, refer to the Next.js documentation on basic features/pages.

7.3.3 Components

Components are used to store repeated or cluttered code, helping keep the code of the pages clean and focused. Each component should have a single responsibility. For example, a component may display a single card on the settings page. Both components and pages are stored in separate directories at the root and can be imported. Components should be imported into pages, but not the other way around.

7.3.4 Localization

The frontend of the app is also translated into different languages. The language files can be found in the `public` directory, with each language having its own folder containing a JSON file. The JSON files follow a key-value structure, where the keys need to be consistent across all language files, while the values are the translations. When translating, it's important to review the context and consider any potential changes. The app utilizes Next.js' i18n feature for localization, which works well but can be complicated to set up with Tauri. If you want to add new languages, be sure to specify it in your pull request or issue, as some code adjustments may be necessary. For example, updating the language display on the main page requires manual changes and is not automated. Here's an example of how localization works on the frontend:

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

In this case, a SweetAlert pop-up is displayed with a title and message. The important part is the `t(key)` function, which retrieves the translation associated with the given key from the JSON file.

7.3.5 Invoking Rust

The frontend doesn't perform calculations itself, as that task is handled by the Rust backend. Rust is faster and has access to local files. To establish communication between the frontend and backend, Tauri is used. Tauri provides an API to call Rust functions and await results on the frontend, similar to a Promise. Here's 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");
  });
```

In this code snippet, the `invoke()` function from Tauri is used to call a Rust function, and then the frontend waits for the result or handles any potential errors.

Last update: May 31, 2023
Created: April 3, 2023

7.3.6 Comments