Contents

1 Introduction

Rust is a programming language that focusses on performance, security an reliability. The *Rust Project* is open source and dual-licensed under the Apache 2.0 and MIT license (Hoare and et al. 2019). Rust 1.0 the first stable version was announced in May 2015 (The Rust Core Team 2015). This release also marked the start of the *commitment to stability* which promises stability on future Rust stable releases (Turon and Matsakis 2014). This new commitment also introduced a 6-week release cycle as well as development channels for language users and early adapters (Turon and Matsakis 2014). The latest stable compiler version 1.51.0 has been released on 25 of March 2021 (The Rust Core Team 2021). Developers and teams within the project put high effort into open communication. This focussed is formalized in the official *Code of conduct* (The Rust Team 2021b). The language with its connected tools has attracted over 5900 individual contributes as of writing this (The Rust Team 2021a).

The Rust project develops several tools besides the compiler itself. These tools are seen as a vital part in automating parts of the development process and collaboration among teams. *Clippy* is the official linter for Rust and is being developed in the *rust-clippy* repository. The linter contains over 450 lints which span from complexity and style lints over to restriction lints which might be required by certificates (The Rust Clippy Developers 2021). Clippy is written in Rust itself and interfaces with the compiler directly. This direct connection enables the use of the existing lexer, parser and connected diagnostic tools and ensures that the project stays up to date with the latest compiler changes. Since 2018 Clippy is distributed as a component of the Rust installation itself (Lusby 2018).

1.1 Problem

Clippy, like the compiler, puts great effort into giving helpful diagnostic messages. These messages consist out of a message, a direct quotation of the suboptimal code fragment and a direct suggestion how the code quality can be improved. The first message of a lint additionally includes a note why which lint is enabled in this location and a reference to the lint documentation in Clippy's lint list. This makes Clippy's lint list the second point of contact for new users with the project itself. The initial load time of the lint list is noticeably slower than most other websites in the Rust eco system.

Another pain point of the website are some technical deficiencies. The development documentation of the Rust project includes some technical guidelines that the current lint list doesn't fully fulfill.

1.2 Research question

The described problem in 1.1 leads to the following research question: *How can the internet presentation of the lint list for the rust-clippy project be improved?*

1.3 Overall goal

The primary goal of this paper is to review the previously mentioned problems and to find solutions for them.

1.4 Methodical

The start of this paper will collect the formal and informal specifications that Clippy's lint list should optimally fulfill. The next part will analyze the current state of the website. This chapter will conclude with a list of aspects that should be improved as well as a selection which this research will focus on.

The previous preparation leads to the solution finding process. This process will start with a technical explanation of the aspect to determine why this specification is important. The next step tries to find a solution for the found issues under the given circumstances. Additionally, this section will include a practical test if the suggested change would improved the focussed problem.

This paper will end with a summary of the reviewed aspects and a conclusion as well as a suggestion what further work can be done on this topic.

2 Requirements

The goal of this work is to improve the impression of Clippy's lint list. This section of the document will set a list of requirements to focus on in further research for this paper. Requirements are usually split up into the following two groups (Sommerville 2010, p. 83ff):

- *Functional requirements* describe direct functionality and behavior that a system should provide. They can also be defined as negations, stating that a certain behavior should not happen. These requirements are usually documented in an abstract way to enable system users to understand them.
- *Non-functional requirements* are focussed on the characteristics of the system itself, an example might be the requirement to have a reliable and maintainable system. These requirements can include constraints that the system might have to take care of.

2.1 Implemented functionality

The topic of this assignment focusses on the perception and impression of Clippy's lint list as an entire system. The research question therefor puts a focus on non-functional requirements. The website itself is based on functional requirements and these should remain fulfilled even after the suggested changes. It is therefor important to note them in some way or form while not taking focus of the key point of this paper. All requirements that have been implemented previously will therefor be summarized in the following functional requirement: *The functionality of the website should not be impacted by the implementation of new measures to improve the impression or usage.* This requirement covers functionality like the search feature, filter options and theming. The implementation of all of these has been completed at the point of writing this. A more extensive specification of the underlying requirements can be retrieved from the rust-clippy issue tracker.

2.2 Non-functional requirements (Maybe: Requirements by the Infra team)

The Rust Infrastructure team has created a set of guidelines that static websites affiliated with the Rust project should fulfill. Clippy is an official Rust project and the website itself presents static content, the guidelines therefor apply to the website.

These guidelines contain a *formal specification* that states: *"The website must reach an A+ grade on the*

Mozilla Observatory." (The Rust Infrastructure team 2020). This specification is based on the requirement of security. The A+ grade ensures that all important headers have been set and that enhanced users privacy features should be enabled by the browser (The Rust Infrastructure team 2020). A secure website contributes to a trustful relation ship between the user and Clippy's lint list. It additionally improves the ranking in most search engines and therefor helps users to faster find the documentation they require.

The guidelines from the infrastructure team additionally contain some functional requirements that are not directly connected to the research question, they are also already fulfilled by the current setup. These are therefor included in the defined functional requirement.

2.3 Performance

The second major non-functional requirement this assignment will look at is performance.

- Search performance
- Load performance

3 Fulfillment of requirements

This chapter will measure the current fulfillment for the previously in 2 defined requirements. These results will then be used to identify the key aspects that need improvement. Additionally they will be used for comparison when testing suggestions.

3.1 Mozilla Observatory

Mozilla Observatory is a collection of tools that can analyze a website to determine which available security measures have been utilized by it (King and et al. 2018b). These security is focussed on values that can be set in the HTTP header to indicate that opt-in security options should be enabled by the browser (**TODO**). The Rust development documentation links to a free online interface[1] for the Mozilla Observatory that is provided by the Mozilla Foundation free of charge.

3.1.1 Scoring

The result of the analyzes is summarized in a single score with a corresponding grade. The score is calculated using a baseline each checked criteria can add bonus points or subtracted penalty. This implementation is used to give different weight to specific configurations. The significance of these modifiers are based on how important the analyzed aspect for security. Scores can range from a minimum of 0 to a maximum of 135. A score of 100 and above corresponds to the grade $A+$ (King and et al. 2018a).

The observatory documentation notes that all websites are graded equally, this means certain graded configurations might be unimportant for the specific use case (King and et al. 2020).

3.1.2 Measurement

3.2 Running benchmarks

1. Simple *Mozilla Observatory*

[1] <https://observatory.mozilla.org/>

2. Load times -> difficult
    - Comparing only on one device as this load time is significant and we want significant improvements
    - rustfmt's website shows that fast loading times are possible

3.3 Summary of benchmarks

Hello, was geht?

4 Analysis of benchmark results

Give hosting background IE the website is deployed using GH Pages etc...

4.1 Technical problems

- Explaining the grade C from *Mozilla Observatory*
- This should definitely include scientific sources to make this a valid paper
    - The examiner noted that the paper outline seems interesting but that I need to take care to include scientific sources
- Explanation why the listed security risks are security risks

4.2 Slow loading times (Browser debug tools)

Hello

5 Solutions for unfulfilled specifications

- Reading GH page documentation
- Maybe contacting support

Hello

6 Conclusion

Hello

References

Hoare, Graydon and et al. (2019-01). *COPYRIGHT*. URL: https://github.com/rust-lang/rust/blob/master/COPYRIGHT (visited on 2021-04-20).

King, April and et al. (2018a-01). *HTTP Observatory Scoring Methodology*. URL: https://github.com/mozilla/http-observatory/blob/fa38ab4/httpobs/docs/scoring.md (visited on 2021-04-24).

– (2018b-03). *Mozilla HTTP Observatory*. URL: https://github.com/mozilla/http-observatory/blob/1bb1566/README.md (visited on 2021-04-24).

– (2020-10). *Frequently Asked Questions*. URL: https://observatory.mozilla.org/faq/ (visited on 2021-04-24).

Lusby, Jane (2018-07). *Add clippy to the tools list #1461*. URL: https://github.com/rust-lang/rustup/pull/1461 (visited on 2021-04-17).

Sommerville, Ian (2010). *Software Engineering*. 9th edition. 501 Boylston Street, Suite 900, Boston, Massachusetts 02116: Pearson Education, Inc. ISBN: 978-0-13-703515-1.

The Rust Clippy Developers (2021-03). *Clippy*. URL: https://github.com/rust-lang/rust-clippy/blob/7fcd1/README.md (visited on 2021-04-17).

The Rust Core Team (2015-05). *Announcing Rust 1.0*. URL: https://blog.rust-lang.org/2015/05/15/Rust-1.0.html (visited on 2021-04-17).

– (2021-03). *Announcing Rust 1.51.0*. URL: https://blog.rust-lang.org/2021/03/25/Rust-1.51.0.html (visited on 2021-04-17).

The Rust Infrastructure team (2020-03). *Rust Infrastructure hosting for static websites*. URL: https://forge.rust-lang.org/infra/guidelines/static-websites.html (visited on 2021-04-24).

The Rust Team (2021a-04). *All-time Contributors*. URL: https://thanks.rust-lang.org/rust/all-time/ (visited on 2021-04-20).

– (2021b). *Code of conduct*. URL: https://www.rust-lang.org/policies/code-of-conduct (visited on 2021-04-20).

Turon, Aaron and Niko Matsakis (2014-10). *Stability as a Deliverable*. URL: https://blog.rust-lang.org/2014/10/30/Stability.html (visited on 2021-04-20).

7 Attachments