Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Implementation of a user interface for the monitoring of a refactoring bot

Hai Duy Dam

Course of Study:	Computer Science
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Marvin Wyrich, M.Sc.
Commenced:	September 24, 2019
Completed:	June 08, 2020

Abstract

The *Refactoring-Bot* automatically removes code smells by refactoring the source code and creates pull requests with the changes on GitHub. However, the process of configuring the bot isn't very user-friendly and it's not possible to monitor the bot comfortably.

In this paper, we introduce the implementation of a Graphical User Interface (GUI) for the *Refactoring-Bot* in which you can easily manage configurations to start refactorings with the Sonar-Qube Analysis Service and monitor the pull requests of your designated bot user in a specific GitHub repository. The GUI communicates with the Representational State Transfer (REST) Application Programming Interface (API) of the *Refactoring-Bot* to exchange all the data which are needed to present it to the user. Then we evaluate the usability of the GUI by conducting a usability test, asking the participants to perform specific tasks while we take notes their behaviour and interaction with the service to find possible usability issues. Our findings show that the implemented product is easy-to-use and provides a high usability which results in a satisfying experience.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 3

GUI Graphical User Interface. 3, 41

REST Representational State Transfer. 3

UCD User Centered Design. 14, 41

1 Introduction

1.1 Motivation

In the age of digitization, computers are an integral part of our everyday life. Since computers are used by all classes of society and all age groups it is important to make the usage of software applications as easy and intuitive as possible. Therefore the GUI plays a central role as it acts as an connection between the software project and the end-user by enabling him to interact with the computer using mouse, keyboard or other input devices. Most software tools and software programs nowadays have an implemented GUI so that an average user has a pleasant experience while using and interacting with the service. An advantage of a GUI is the big improvement in usability for most people, increasing their productivity and efficiency while using a software to get the job done and simplifying the learning process for users without any prior knowledge about computing language. One of the key aspects is that it should be visually intuitive and easy to understand which enables users to interact with the environment with help of graphics and visual indicators.

Usability plays a significant role in the design of a user interface.

> ”On the Web, usability is a necessary condition for survival. If a website is difficult to use, people leave. When the homepage fails to state what a company offers and what users can do on the site, people leave. If users get lost on a website, they leave. If a website’s information is hard to read or doesn’t answer users’ key questions, they leave.” - Jakob Nielsen, Web Usability Professional [Nie12]

In 2015, Huff Industrial Marketing, KoMarketing and BuyerZone [Huf15] released a report about B2B web usability which shows that 37% of users leave a website due to poor design or navigation. Loading time of a page plays an important role as well: 75% of the users would hit the ”Back”button before the page fully loads. 42% of the survey respondents define 10 seconds as slow loading, followed by 34% with 15 seconds. This report shows the negative impact of bad usability on the number of users and the overall user satisfaction.

Wyrich and Bogner [WB19] implemented the *Refactoring-Bot* to increase the efficiency and effectiveness of removal of code smells. This bot automatically refactors code in Java projects and interacts with the team via pull requests, where it presents the changes to a developer for asynchronous review on an established version control platform such as GitHub or GitLab. However, usability wasn’t the focus during the development of this bot. Currently, creating configurations and starting refactoring are handled over the provided Swagger UI. The process of configuring the bot is error-prone and tedious because the UI isn’t intuitive and user friendly for the average user. Additionally, comfortable monitoring of the bot isn’t possible at the moment but it’s necessary for the evaluation of the work of the bot.

1.2 Research objectives

The goal of this thesis is to implement a user friendly Graphical User Interface for the *Refactoring-Bot* which enables the user to create configurations to start refactorings with the SonarQube Analysis Service and monitor the communication and behaviour of the bot. After the refactoring, we need a method which takes the by the bot created pull requests on the GitHub repository as an input and presents them in an easily comprehensible way to the user To provide a satisfying GUI for the end-users we need to meet the requirements of a good usability for user interfaces and evaluate the solution according to these specified requirements.

1.3 Methodological approach

To implement a GUI based on an existing software, we followed User Centered Design (UCD) methodologies which involves conducting user interview, specification of the context of use and analyzing the user requirements. After gathering and analysing all the information above, we created and evaluated a paper prototype for the GUI. The implemented solution was evaluated by performing a usability test with intended users to help identifying existing problems of the current version and narrow down their location. The usability test also granted insight on whether the system would be able to fulfill user needs.

1.4 Structure of the work

In the following chapter we provide background knowledge about the **Refactoring-Bot** and introduce UCD process which in our case describes its workflow and benefits. In Chapter 3 provides a summary of related work on scientific literature related to GUI. The next Chapter 4 describes our approach method following UCD for analyzing the requirements and creating a prototype to evaluate its design. Chapter 5 will focus on the implementation where we explain the technical part. Afterwards in Chapter 6, is where we describe our evaluation of the usability of our implemented system and present the results. At last, we summarize our work and provide an outlook in Chapter 7.

2 Background

In this section, we clarify terminology which is used in the *Refactoring-Bot*. After that, we describe how the *Refactoring-Bot* works and how its architecture looks like. Then we define what usability is and why it's important. At last, we introduce the User Centered Design process which we follow in our methodology in Chapter 4.

2.1 GitHub

Git is one of the most popular version control system with almost 90% of developers voting Git as their preferred choice [1]. Version control systems store modifications of code in a central repository as revisions. Every developer can see the changes, download them and contribute. GitHub is a repository hosting web-platform which is based on Git where many developers store their projects on. From the official GitHub documentation, we list some of the terms we are going to use:

- ”A **repository** is like a folder for your project. Your project's repository contains all of your project's files and stores each file's revision history. You can also discuss and manage your project's work within the repository.” [2]
- ”**Pull requests** let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.” [3]
- ”A **fork** is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests.” [4]

2.2 Refactoring

Refactoring is one of the well-known techniques to change the source code with the aim of impoving the quality of the code structure. The benefit of refactoring is to make the internal code structure of a software more readable, easier to modify and to maintain. At the same time the external behavior shouldn't be altered. Code smells are symptoms to identify structural problems which negatively

[1] https://insights.stackoverflow.com/survey/2018#work-_-version-control

[2] https://help.github.com/en/github/creating-cloning-and-archiving-repositories/about-repositories

[3] https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

[4] https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-forks

impact design quality [Fow99]. A way to detect such code smells is the usage of static code analysis tools that helps the developers to gain insight on the location and type of existing code smells, which allows the developers to remove them effectively.

2.3 Software Bot

Despite their increase of usage and popularity over the years, there is no generally accepted definition of software bots. Most of the scientific literature and researches use *"bot"* to characterize a wide range of software programs which spans across multiple computer science areas. Therefore researchers and developers define bots according to their specific applications of software technologies. Lebeuf [Leb18] lists some key trends how people describe software bots:

- **Bots as Automation**: "One of more common ways of defining bots is as software programs that **automate** tasks".
- **Bots as Malicious**: Computer programs designed to perform a malicious action.
- **Bots as Human-Like**: The ability to act and be perceived **humanly**. Slack claims that bots are like a virtual team member.
- **Bots as Conversationalists**: The ability to **communicate using human language**.

After that, the author proposes a definition of software bots which is build upon the existing definitions in the previous list. From her point of view, software bots are a "new **interface paradigm**". They're the bridge that connects **users** with **software services** and provides the services to the user.

Wyrich and Bogner [WB19] define a software bot which includes the characteristics of the Refactoring as:

> For us, a bot is intelligent software that acts (to some extent) autonomously to achieve a defined goal and offers functionality for interaction. In addition, we agree with the description of Wessel et al. [WSS+18], who limit their work to bots " that have a user profile and play a role within the development team, executing well-defined tasks that complements other developers' work"

2.4 Refactoring-Bot

Wyrich and Bogner [WB19] from the Institute of Software Technology of the University of Stuttgart developed a bot that automatically refactors code to improve the source code quality and submits its changes to a version control platform as a pull request. The motivation behind this development was to solve an issue where manually removing code smells is error-prone and tedious [BDD+12]. The *Refactoring-Bot* gets described as an "Autonomous Bot for Automatic Source Code Refactoring" "that integrates into the team like a human developer via the existing version control platform" [WB19]. Figure 2.1 displays an overview of the bot in its environment. With a created configuration, it's possible to manually start the refactoring by calling the appropriate REST API via the provided Swagger UI. The bot creates a list of various kinds of code smells found by the static code analysis tool based on the rules that are setup for the mentioned code analysis tool. Editing the source code is done locally by automatically forking the repository on GitHub which allows the Refactoring-bot to work without restrictions on a copy. The static code analysis is performed by SonarQube which resolves following code smells: incorrect order of language modifiers, missing override annotation, commented-out code and unused method parameters. In the final step, it creates pull requests with the changes of the refactored code on the original GitHub repository for each correct finding. Afterwards, the developers and collaborators of the original GitHub repository can review these pull requests and either merge, close or leave a comment in these pull requests to communicate with the bot and make small corrections to the refactorings.

The *Refactoring-Bot* is a Spring Boot application, with other words it's using "Spring Framework which is an application framework and inversion of control container for the Java platform" [5]. Creating a configuration to setup the bot has to be done manually by inserting the necessary information and values in a JSON format. With a created configuration, it's possible to manually start the refactoring by calling the appropriate REST API via the provided Swagger UI. The bot creates a list of the found code smells by SonarQube and refactors all of them.

The created configurations are saved in a MySQL database along with a table of already refactored issues. With the provided Swagger UI the user can interact with its REST API resources to create, read, update, delete configurations, and perform refactorings as well.

The configuration process is error-prone because there is no explanation about the needed data or where to find them. To update an existing configuration there are more steps needed than necessary because the configuration ID has to be remembered and the whole previous configuration has to be copied and inserted again, even if only one attribute needs to be changed. Additionally, the user has to navigate between different controllers by scrolling through the page which results in more mental work by trying to remember information he needs.

The *Refactoring-Bot* can be found as an open source project on GitHub [6].

2.5 Usability

The official ISO 9241-11 [7] definition of usability is: "the extend to which a system, product or service can be used by specified users to archive specified uses to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Summarized, usability is a measure of the user experience with the associated system. It describes how easily a user can perform his tasks successfully and how satisfying and engaging it is to use.

Why is Usability important?

As a user, usability affects the experience and satisfaction to achieve his goal and makes the difference if a task can be performed accurately and completely [For]. From the developer's perspective, it determines if the system can fulfill the user needs and achieve its mission to serve the customer. Montero et al. [MGLV05] explain why websites with usability issues can confuse users which

[5] https://en.wikipedia.org/wiki/Spring_Framework

[6] https://github.com/Refactoring-Bot/Refactoring-Bot

[7] https://www.iso.org/standard/63500.html

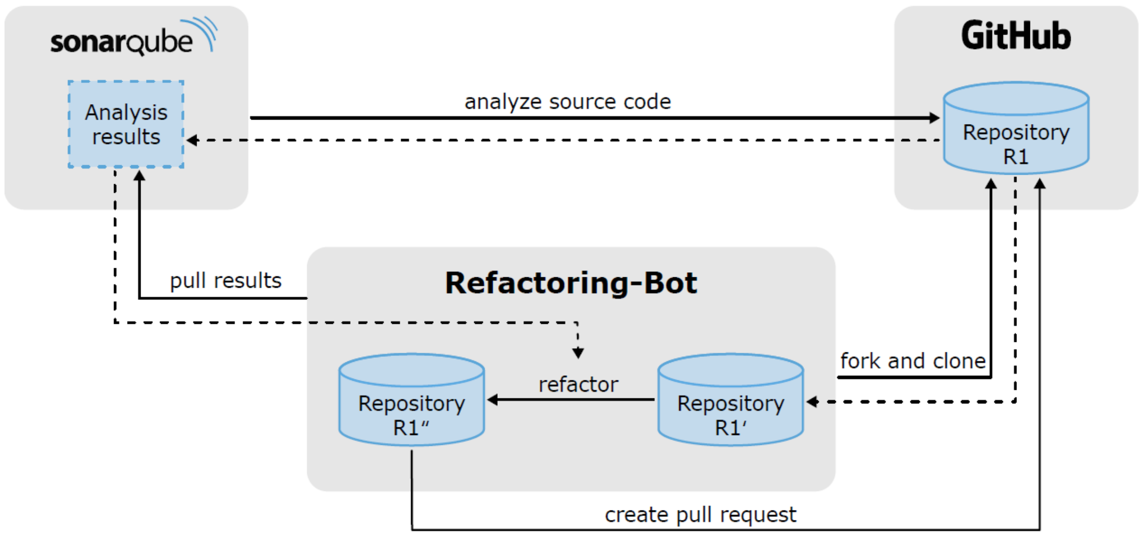


Figure 2.1: Overview of interactions between the Refactoring-Bot, a web-based hosting service for version control (GitHub), and a static analysis tool (SonarQube). [WB19]

results in a loss of revenue for the company. The quote of Nielsen [Nie12] in Chapter 1 describes the consequence of poor usability very well: the user can leave the website anytime with a press of a button when he's not satisfied with the system.

Usability has multiple components and can be defined by these five following key components which got determined by Jakob Nielsen [Nie94] [Nie12]:

1. **Learnability**: "The system should be easy to learn so that the user can rapidly start getting some work done with the system."

   "How easy is it for users to accomplish basic tasks the first time they encounter the design?"

2. **Efficiency**: "The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible."

   "Once users have learned the design, how quickly can they perform tasks?"

3. **Memorability**: "The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again."

   "When users return to the design after a period of not using it, how easily can they reestablish proficiency?"

4. **Errors**: "The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur."

   "How many errors do users make, how severe are these errors, and how easily can they recover from the errors?"

5. **Satisfaction**: ”The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.”

   ”How pleasant is it to use the design?”

2.6 User Centered Design

UCD is an iterative design process framework with the goal of developing a usable system by focusing on the users and their needs in every phase during the planning, designing and development of a product. In every stage of the design process the users are involved to get a greater level understanding of what they want and how they interact with each part of the system. Usability is the outcome of this design process.

The following methods are widely accepted as the general phases of UCD process [Usa][Fou]:

1. **Specify the context of use**: Identify the people who will use the product, what they will use it for, and under what conditions they will use it.
2. **Specify requirements**: Identify any business requirements or user goals that must be met for the product to be successful.
3. **Create design solutions**: This part of the process may be done in stages, building from a rough concept to a complete design.
4. **Evaluate designs**: Evaluation - ideally through usability testing with actual users - is as integral as quality testing is to good software development.

The exact methods for each phase are not specified and there are many variations of the UCD process.

The research of Vredenburg et al. [VMSC02] shows that UCD methods resulted in improvement of product usefulness and usability in general.

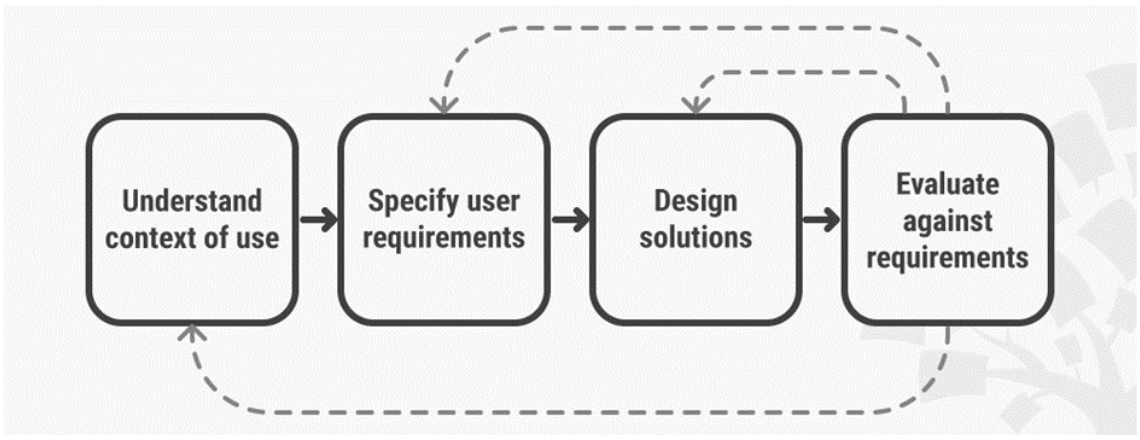

Understand context of use
Specify user requirements
Design solutions
Evaluate against requirements

Figure 2.2: User-centered design is an iterative process that focuses on an understanding of the users and their context in all stages of design and development. [Fou]

3 Related Work

Most of the scientific literature about **Graphical User Interface** are development, researches and implementation for specific software projects over a large range of issue-areas extending from computer science to medical researches.

For a Finnish ICT company, [Möl18] designed and developed a GUI for modifying chatbot configuration which was one of their products sold to customers to improve the customer experience over the web, resulting in higher customer service profitability. The problem that the developed system focused on was that the previous method to configure chatbots was time-consuming and error-prone.

To create a chatbot, a configuration file has to be written manually by the developers before the project can be started. For long projects this process was very tedious and typos in the configuration files resulted in time-consuming error finding during the development. Customers and users didn't have the options to modify the configuration files and had to contact the developers which negatively impacts the work efficiency for both parties as well. The frontend is an editor application which allows the users to drag-and-drop shape-looking components to visualize the chatbot flows, transforming the visual graph into a corresponding text represented configuration. For the technical architecture, as the main JS framework it uses Rappid Diagramming framework, along with modern JavaScript technologies like ECMAScript 6 and Typescript. The GUI communicates with the Java backend server that handles client requests, data persistence and user authentication. The server consist of a rest service, a connection to a MySQL database and the robot entity which is responsible for the logic. After the implementation the user experience got tested to get a better understanding where current usability issues lie that weren't considered during the development process. The aim of the test is to replicate a scenario in which the customer has to perform several tasks to create a chatbot flow by using the application. To make the test as authentic as possible only people without a technical background were chosen as test subjects. The results of this test were that the UI is clear and easy to learn but suffers from small issues like the user getting lost and did not know what the next steps are or that some visual elements are unclear.

Samuelsson and Nguyen [SN16] developed a dashboard as part of PinDown, an automatic debugging tool, developed by Verifyter. The goal of this thesis was to improve the usability by implementing a dashboard to monitor the information on a single page. Therefore, they investigated the previous implementation of the user interface and evaluated the usability with Lauesen's definition. One of the issues they mentioned was the need to switch panes whenever the user wanted to see the graphs. After spending some time with the tool, the finding showed that task efficiency and user satisfication could be improved. Through researches the frameworks for the frontend was decided. The libraries for the frontend are Bootstrap which supports responsive web page layout, D3.js to visualize data in graphs and NVD3 which is a library for prodividing re-usable charts for D3. For the backend, the decision was between Ruby on rails, Django and Play framework. The difference between these frameworks is the programming language used. In the end, Play got chosen as the web ramework because Samuelsson and Nguyen [SN16] were more familiar with Java and the current implementation of PinDown is Java based. To develope a dashboard, a process which consists of five stages was used as a guide and they followed the first three steps:

1. Requirement elicitation
2. Dashboard type selection
3. Dashboard design

Gatherhing feedback for the design of the dashboard components were done by using paper mock ups of the dashboard layout. A grid pattern layout with priority of the components in mind was chosen and was the reference layout for the digital prototype. To evaluate their design against the requirements and usability guidelines, a usability test which consisted of six tasks were conducted with three students. The goal was to test four usability factors which were: Ease of learning, task efficiency, subjective satisfaction and understandability. The conclussion was that the implemented user interface improved the usability compared to the previous product. However, the previously issue with switching panes still existed because the UI wasn not a dashboard.

4 Requirement Analysis

To develop and design a good GUI many aspects and factors have to be considered during the complex process to provide a product with high usability. We have to take into consideration who is going to interact with our system, what the context is and which goals a user try to achieve. These aspects are crucial to build a GUI that encourages an easy-to-understand and natural interaction between a user and a system. The best experience is when a user forgets that he is interacting with a device as he is trying to complete his tasks.

We followed UCD methodologies to identify the user needs and tried to involve the user in most of the design and development phases (see Section 2.6).

4.1 Stakeholder Interview

Gathering requirements is a necessary step for developing software projects. This procedure should be done early in the designing process to avoid changes which could result in increased costs in the future. Therefore we invited our stakeholder to an one-to-one interview to clarify the goal and define the success of the project. Interview is the most common technique for gathering requirements and for getting a better understanding of the user needs and his environment.

The interview was conducted in a meeting room at the stakeholder's workplace to avoid disturbance from the environment. We created a loose interview guide to control the flow of the interview and prepared some key questions to answer our research goals. The questions were open-ended to get the interviewee to keep talking. As a result it was possible to ask follow-up questions if more details were desired. This allowed us to explore the users intention and gave us insight of the users problems. These problems could be identified as undiscovered requirements. The interview lasted approximately an hour. During the interview we took notes of the interviewee's answers to analyze them after the interview and to convert them into requirements.

The key questions we wanted to be answered are listed below:

- What are the goals of this project?
- What problems do you run into?
- Why is it a problem?
- Who are the users of the system?
- What background knowledge do the users have?
- What can the users do with the system?
- Are there any technological limitations?

4.2 Requirement Specification

4.2.1 Previous Work

At the beginning of this thesis, a work-in-progress ”web frontend to conveniently configure and manage the Refactoring-Bot” was already existing. The *Refactoring-Bot Management UI* requires the *Refactoring Bot* to run in the background so it can communicate with its API. The framework for the GUI uses Vue.js, a Javascript framework for building UIs and single-page applications. The initial skeleton along with pages for the Git users well as for the configurations was already implemented that provides some API connections to the *Refactoring Bot*. However, most functions did not work or still were missing.

The repository for this project can be found on GitHub [1].

4.2.2 User Requirements

The following user requirements for the GUI have been obtained from the interview in Section 4.1.

- Users should be able to **add** an existing **Git Account** as a bot.
- Users should be able to **edit** an existing bot.
- Users should be able to **create** a **configuration** with an already added bot, so they do not need to reenter all the Git Account data for every new configuration.
- Users should be able to **edit** an existing configuration without entering all the previous data again.
- Users should be able to start the Refactoring Analysis.
- GUI should **display pull requests** created by the bot.
- GUI should **display communcation** and **interaction** between developers and bot
- Added users and created configurations should be **saved** in the **database**.
- Connection to the GitHub API to retrieve necessary data should be handled in the backend.

The use-case diagram 4.1 shows the basic functionalities of the GUI a user can interact with.

[1] https://github.com/Refactoring-Bot/Refactoring-Bot-UI

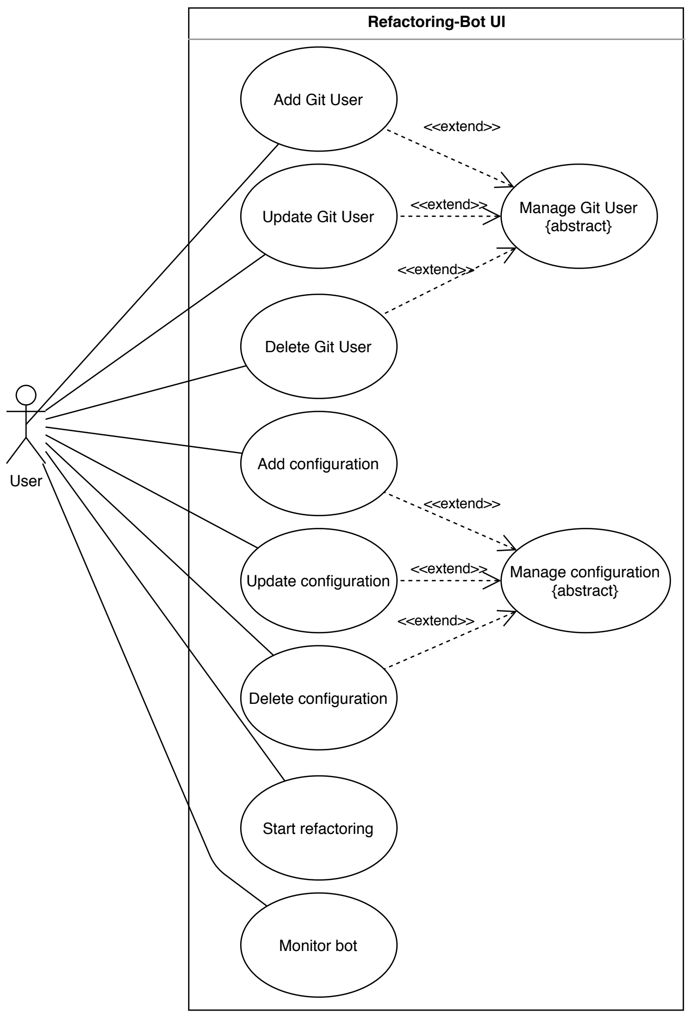


Figure 4.1: Use-Case: Basic functionalities of the Refactoring-Bot UI

4.2.3 Non-functional Requirements

- The frontend framework is built with Vue.js.
- The web page is accessible on all modern desktop browsers like Chrome, Firefox, Opera, Microsoft Edge on the latest version.
- The language for the GUI is displayed in English.
- **Efficiency**: The system is efficient to use and the user can easily and quickly accomplish his goals with a few or no user errors.
- **Intuitiveness**: The system is easy to use and learn for users without programming knowledge. Help and error messages are simple to understand and provide necessary information.

- **Consistency**: The system is consistent across all web pages. This includes page layout, button placement and color scheme. Experience and skills learned by using the system should be transferable to other parts of the site which increases the efficiency and makes the user more comfortable.
- Git User is a separate model which passes its unique ID to the configuration model.

4.3 Paper Prototype

Paper prototyping is a research methodology where designs and concept ideas can be tested to identify if they meet the user's needs and expectation before investing time and money into development. Feedback from end-users are valuable and testing the design and functionalities during the very early phase of development reveals usability issues. This method is a low-fidelity technique that enables developers to rapidly iterate their design ideas and make changes with minimum investment in time and cost.

We designed low fidelity paper prototypes based on the requirements we acquired in the previous Section 4.1 to comprehend how users interact with the system and which visual components they like to see on the monitoring page. One of the important goal of low fidelity prototypes is to check and test functionality instead of focusing on the visual appearance of the product [Ado]. For the navigation of the configurations we produced two different versions of the monitoring page. The first version of the prototype had a top navigation that resembled a horizontal tab navigation bar. The second version had a vertical navigation on the left side of the page where the configuration items were displayed below each other. Additionally, sample charts and components were placed on the monitoring page as well. For each web page we used a piece of paper to conduct a paper prototype testing.

In the end, our paper mock up screens consisted of five pages:
the first one was the home screen. Its main objective was to introduce the user to the system and give a short usage instructions. The next two pages were input forms for adding git users and creating configurations for the bot. The purpose of presenting these screens was to collect feedback about the form field layout and field labels. The last pages were the previously mentioned sketches for the monitoring page.

4.4 Evaluation of Prototype

For the paper prototype testing we invited our stakeholder to evaluate the setup process of the Refactoring-Bot and ask his opinion about the design decisions. To identify usability issues and discover not considered scenarios or unexpected behaviour, the user should interact with the paper prototypes like a simulation on a computer so that we can get an insight of the user needs as authentic as possible.

Before the test began, we explained the scenario and how paper prototype testing was conducted. Then we switched out the screens or gave a description what would happen, based on the user's actions. A pen acted as the mouse cursor and with this tool the user could fill out the forms as well. The two versions of the monitoring page were presented in sequence. At the beginning, we showed the version with the horizontal top navigation and next the version with the vertical left side navigation.

As a result of the testing we decided to implement the top navigation because of the visibility and item priority. The items are above the fold and therefore more visible when they are displayed in the top navigation. Furthermore, it is easier to find the navigation items because most of the time they are positioned right next to the header. In addition to the visibility, the placement of the items has an influence on the visual weight. Since our primary focus is at the top left corner or in other words at the primary optical area when we open a page, the leftmost items carry the most weight. Thus, the user perceives them as more important than the other items.

The suggestion of a timeline came up during the presentation of the components on the monitoring page. After a discussion, the idea got accepted and was written down.

5 Implementation

In this chapter, we describe the modification of the Refactoring-Bot and implementation of the GUI following the requirements we have specified in Chapter 4. In the first part, changes in the backend of the Refactoring-Bot are shown. Then, we talk about the implemented API to obtain the needed data from backend and GitHub to display them on the frontend. Lastly, we present the implementation of the GUI in Vue.js.

5.1 Refactoring-Bot Backend

To achieve the goal of refactoring code smells, the first step is to create a configuration for the Refactoring-Bot which requires following data in JSON format:

- **a GitHub user** who acts as an developer for the bot on the version control platform. The possible actions that the bot can perform are creating pull requests and communicate with other developers in the comment sections of his own pull requests. To act as a GitHub user via the GitHub API, following data are necessary:

Field	**Data Type**	**Description**
botName	String	GitHub username
botEmail	String	GitHub email address
botToken	String	Personal access token for GitHub API

Table 5.1: GitHub user data

- **a GitHub repository** where the source code of the project is located. To find the correct repository on GitHub, the visibility of the repository is set to public and the following parameters need to be provided:

Field	**Data Type**	**Description**
repoName	String	Repository name of project
repoOwner	String	Owner's username of repository
repoService	FileHoster	Filehoster system
maxAmountRequests	int	Maximum amount of pull requests which can be opened at the same time

Table 5.2: GitHub repository data

- **an Analysis service** which performs a static code analysis to identify code smells. It is possible to setup SonarQube rules to find specific types of code smells. The SonarQube analysis service needs following parameters:

Field	**Data Type**	**Description**
analysisService	AnalysisProvider	Analysis Provider Service
analysisServiceApiLink	String	Link to Analysis project API
analysisServiceProjectKey	String	Project key of Analysis project for API

Table 5.3: Analysis Service data

5.1.1 Git User Model

When the user created a configuration, the settings for the Git User had to be duplicated which increased the workload. Therefore, we added a separated Model which holds the necessary information of the user for the sake of convenience. The values are saved in an additional MySQL database table. Since we need to add an already existing GitHub user, the filehoster had to be defined before the configuration creating process. The reason why it is necessary to provide this parameter is to check if the user actually exists on the file hosting platform and pair him with the correct repository in case there will be more filehosters supported in the future.

By passing the unique *gitUserId* to the configuration service, the configuration is able to retrieve the user data from the database. This eliminates the need to reenter all the paramters when a configuration with the same user gets created.

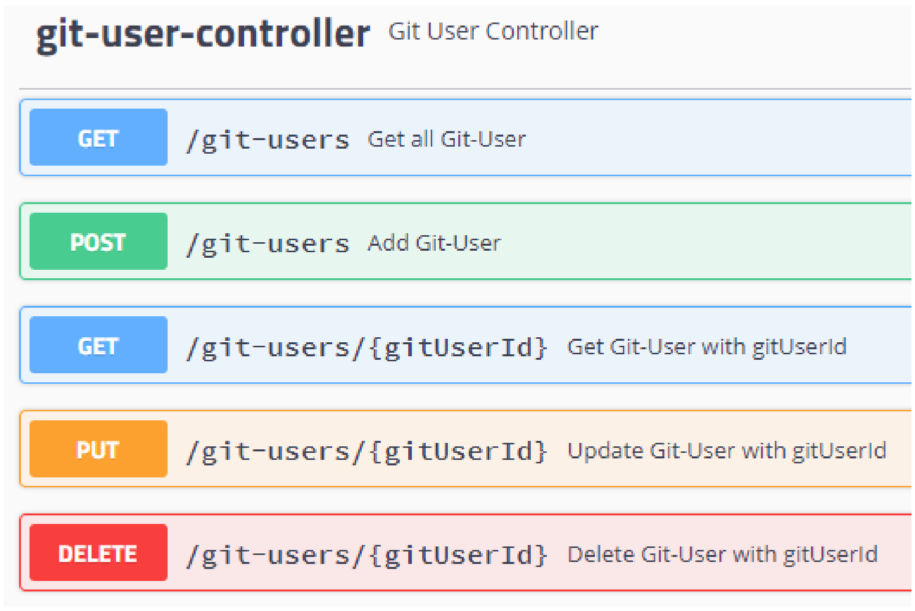
Along with the new files for the Git User class, an interface to communicate via SQL queries with the git user database table got implemented.

Furthermore, functions for the new User Model are required to manage the users. These are the basic functions for persistent storage, also called CRUD Operations, which is an acronym for CREATE, READ, UPDATE and DELETE. In our case, we added two READ functions for getting either a single user or all users from the database. The next one is the CREATE function which adds an existing GitHub User. Following, the UPDATE function to modify an added GitHub User. Both functions communicate with the GitHub API to validate the existence of the user account which is given as input. Last but not least, the DELETE function to remove an entry from the git user database table.

Figure 5.1 shows the User CRUD Interface as a REST Interface in Swagger UI.

With the newly introduced Model, the Configuration Model needed to be adapted to retrieve the user data from the database. To begin with, the attribute *gitUserId* was added to the configuration class along with the related get and set functions. After moving the attribute *repoService* to the User Model, the function for checking the existence of the GitHub User during the configuration creating process is no longer necessary, therefore it got removed to avoid a repeated API call to GitHub.

In order to assign the user data to a configuration, the *gitUserId* gets passed along with the other configuration parameters during the configuration's create and update process to search for the matching user in the database. If there is a valid entry, the controller takes these data and saves this configuration into the configuration database table. Otherwise, an error message is shown.



git-user-controller Git User Controller

GET /git-users Get all Git-User

POST /git-users Add Git-User

GET /git-users/{gitUserId} Get Git-User with gitUserId

PUT /git-users/{gitUserId} Update Git-User with gitUserId

DELETE /git-users/{gitUserId} Delete Git-User with gitUserId

Figure 5.1: Functions for Git User Model

Figure 5.2 shows the relations between the user and configurations.

5.1.2 GitHub REST API

To monitor the Refactoring-Bot, the events related to the assigned GitHub user and repository need to be retrieved from GitHub via REST API calls. At the current time, GitHub offers two stable versions of API[1]: REST API v3[2] and GraphQL v4[3]. The REST API v3 is used by default when requests to *https://api.github.com* are made.

For allowing the Refactoring-Bot to act as a GitHub user, authentication through GitHub API using the Personal access is necessary. We updated the authentication method to HTTP basic authentication which sends the Personal access token in the header instead of including it as a query parameter. The reason for this change is that authentication via query parameters will be discontinued by GitHub in the near future[4].

Our goal is to retrieve the pull requests created by the bot and comments that got created in these pull requests. After looking through the GitHub API documentation to find the API which relates to activities, the Events API[5] seems to be the correct place. This read-only API allows us to get the events of either a user or of a repository in JSON format. One important point that needs to be mentioned is that ”only events created within the past 90 days will be included in timelines”.

[1] https://developer.github.com/v3/versions/
[2] https://developer.github.com/v3/
[3] https://developer.github.com/v4/
[4] https://developer.github.com/changes/2020-02-10-deprecating-auth-through-query-param/
[5] https://developer.github.com/v3/activity/events/

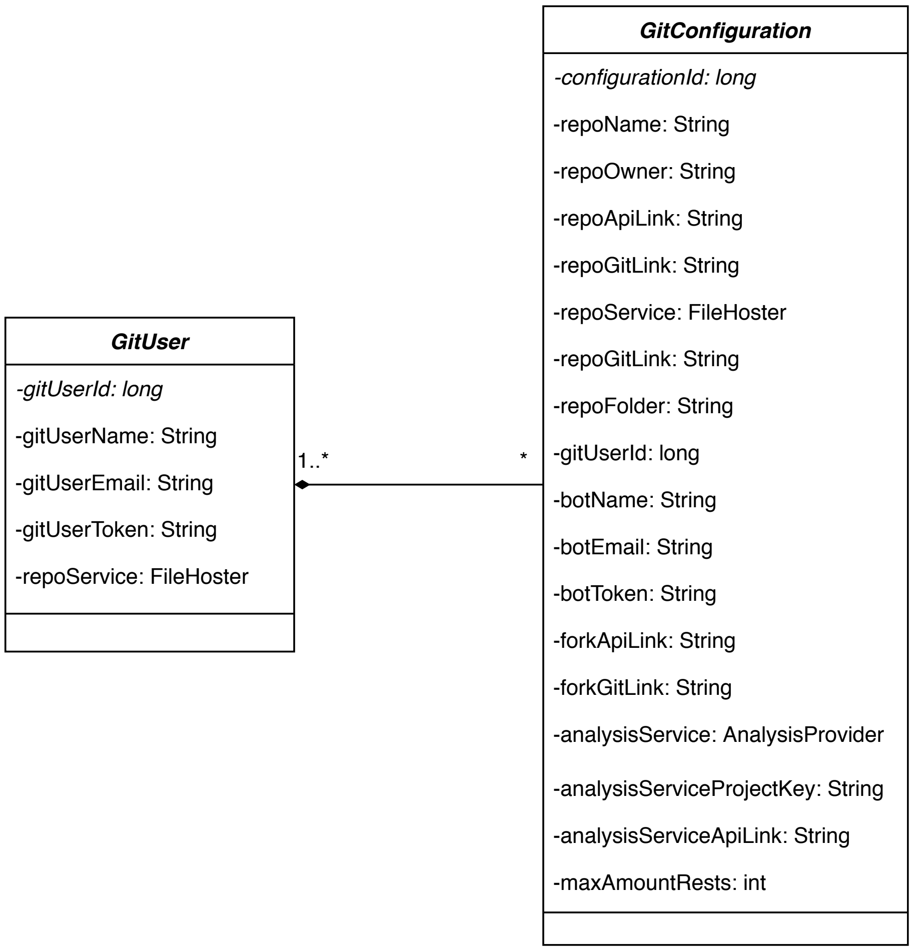


Figure 5.2: UML Class Diagram: Relation between User and Configuration model

Listing 5.1 ETags for REST

```
@Bean
public FilterRegistrationBean<ShallowEtagHeaderFilter> shallowEtagHeaderFilter() {
    FilterRegistrationBean<ShallowEtagHeaderFilter> filterRegistrationBean
      = new FilterRegistrationBean<>( new ShallowEtagHeaderFilter());
    filterRegistrationBean.addUrlPatterns("/git-users/*","/configurations/*");
    filterRegistrationBean.setName("etagFilter");
    return filterRegistrationBean;
}
```

However, there is a rate limit for how many requests you can make to the API per hour. For unauthorized users, it is limited to 60 requests. But for authenticated users the quota is up to 5000 requests per hour. Nevertheless, the Events API is optimized for polling with the ”ETag” header which does not decrease the number of allowed requests when the response of the API call has not changed. The HTTPStatus response is ”304 Not Modified” and the data are loaded from cache. Another reason why we implemented this feature was to save bandwidth as well because the server does not need to send a full response if the content has not changed (see Listing 5.1).

Listing 5.2 Get Events from a GitHub Repository

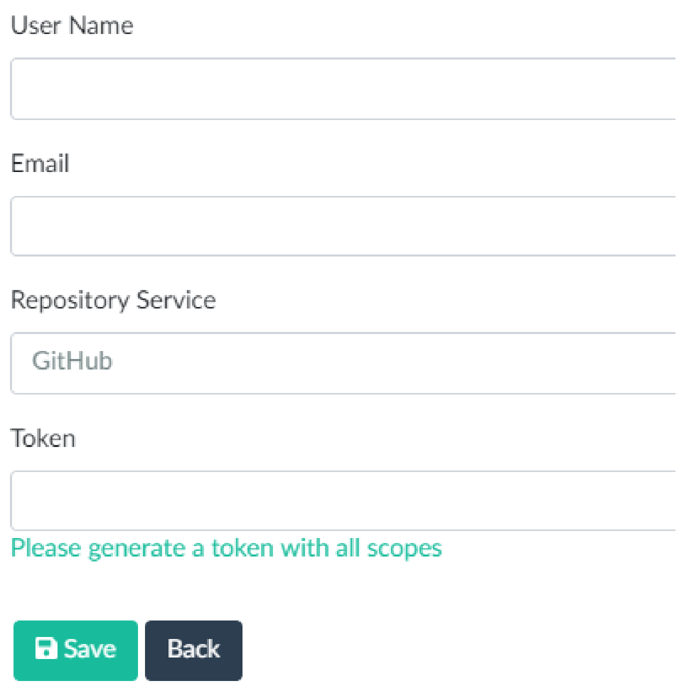
```
  public String getConfigEvents(GitConfiguration gitConfig)
      throws URISyntaxException, GitHubAPIException, IOException {

    // Create URI from user input
    URI eventUri = null;

    eventUri = createURIFromApiLink(GITHUB_DEFAULT_APILINK + "/repos/" + gitConfig.
getRepoOwner()+"/"+ gitConfig.getRepoName());


    // Build URI
    UriComponentsBuilder apiUriBuilder = UriComponentsBuilder.newInstance().scheme(eventUri.
getScheme()).host(eventUri.getHost())
        .path(eventUri.getPath()+ "/events");

    URI githubUri = apiUriBuilder.build().encode().toUri();

    RestTemplate rest = new RestTemplate();
    HttpHeaders headers = new HttpHeaders();
    headers.set("User-Agent", USER_AGENT);
    headers.setBearerAuth(gitConfig.getBotToken());
    HttpEntity<String> entity = new HttpEntity<>("parameters", headers);

    String json = null;
    try {
      // Send Request to the GitHub-API
      json = rest.exchange(githubUri, HttpMethod.GET, entity, String.class).getBody();
    } catch (RestClientException e) {
      logger.error(e.getMessage(), e);
      throw new GitHubAPIException("Could not get Events of the configuration from Github!", e
);
    }

    return json;
  }
```

The code in Listing 5.2 demonstrates the connection to the GitHub Events API with a configuration as the parameter using Bearer authentication.

An HTTP GET method is offered as a REST API interface in the backend to allow interactions for other applications to retrieve the events of a configuration as JSON format.

5.2 Refactoring-Bot GUI

The GUI for Refactoring-Bot is developed with Vue.js and Bootstrap which was specified as a user requirement in Chapter 4. Vue.js is a progressive JavaScript framework for building user interfaces and single-page applications [You]. Bootstrap is a CSS framework which helps us with the frontend


User Name

Email

Repository Service

GitHub

Token

Please generate a token with all scopes

Figure 5.3: Git User Details page

web development by providing design templates for interface components and a grid system layout [Tea]. For documentation of Bootstrap components in Vue.js, we referred to BootstrapVue[Boo]. In addition to these mentioned frameworks, the application is written in *HTML, LESS*, a dynamic preprocessor for CSS and *TypeScript*, a typed superset of JavaScript.

5.2.1 Git User and Configuration View

The first step was to adapt the already existing Git User and Configuration Views in the frontend to our Models in the backend. After fixing several bugs and issues, it was possible to create/add, read, update and delete GitHub users and configurations with the GUI.

With Usability in mind, we inserted default values for the *Repository Service* and *Analysis Service* input fields to make it more comfortable to setup a new user (Figure 5.3) and configuration (Figure 5.4). Additionally, we provided a description with a link for generating *Token* to authenticate with the GitHub API and a link which opened an example analysis project to retrieve the *Analysis Project Key*.

The call-to-action buttons are color-coded to guide the user to success [Usa]. Primary actions that should catch the attention of the user have more visual weights than secondary actions, for example like *cancel* or *back*. Thus, we utilized the property **variant** provided by Bootstrap to apply Bootstrap theme color variants to the components.

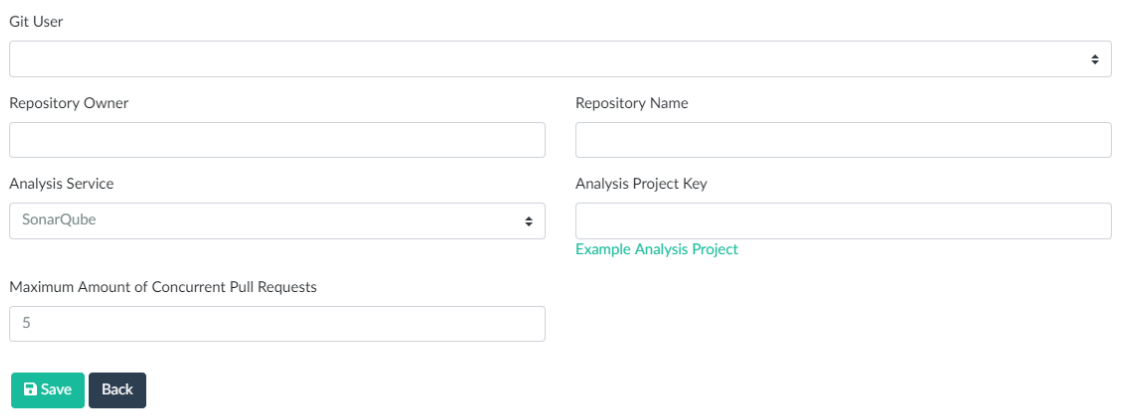


Figure 5.4: Configuration Details page

Listing 5.3 Axios GET method for Git User by ID

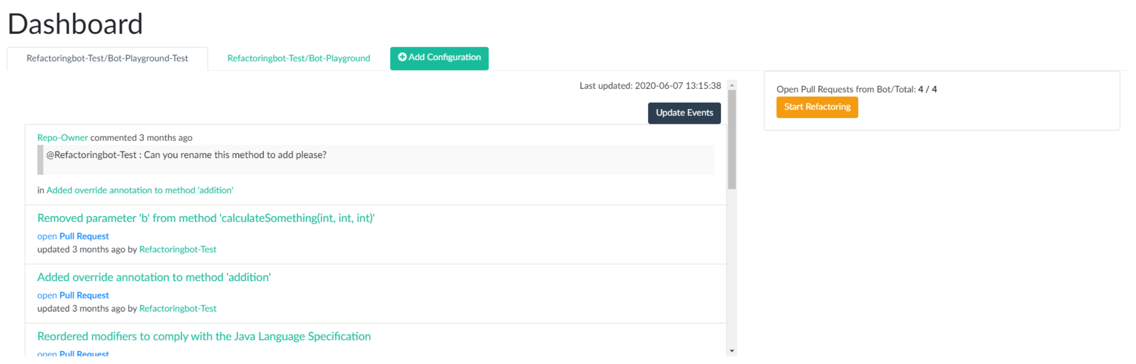
```
// Get git user by ID
public static async getGitUserById(id: string): Promise<IGitUser> {
return (await axios.get(`${this.api}/git-users/${id}`)).data;
}
```

The communication between the GUI and the Refactoring-Bot takes place via RESTful APIs which are consumed by Axios, an NPM package for making HTTP requests. The following code (5.3) shows how to create an HTTP GET request with axios.

5.2.2 Monitoring View

The activities of the Refactoring-Bot get displayed on a web page called **Dashboard**. On this site the user is able to monitor the pull requests which were created by the bot and observe the communication between the developers and the bot which happens in his pull requests. With the horizontal navigation tabs, switching between configurations is easily to do (Figure 5.5. Within the active tab, the refactoring with Analysis service for the chosen configuration can be manually started with the click of a button.

Vue Components are reusable instances which is an important feature of Vue.js. For this page, we created two components: the first one is for the timeline, the second one is for statistics. The advantages of using components are making the web page more modular and maintainable since each component maintains its own interface and data, therefore the initialized instances of a component are independent from each other. These components got implemented into the dashboard view which functions as the parent of those two. After a tab got chosen, it passes the current configuration as data to its children components which allows them to process the data and return them. On page load, the timeline component fetches the data from Refactoring-Bot and displays only pull requests and issue comment events from the assigned bot and specified repository.



Dashboard

Refactoringbot-Test/Bot-Playground-Test
Refactoringbot-Test/Bot-Playground
Add Configuration
Update Events
Start Refactoring

Last updated: 2020-06-07 13:15:38

Open Pull Requests from Bot/Total: **4 / 4**

Repo-Owner commented 3 months ago

@Refactoringbot-Test : Can you rename this method to add please?

in Added override annotation to method 'addition'

Removed parameter 'b' from method 'calculateSomething(int, int, int)'
open **Pull Request**
updated 3 months ago by Refactoringbot-Test

Added override annotation to method 'addition'
open **Pull Request**
updated 3 months ago by Refactoringbot-Test

Reordered modifiers to comply with the Java Language Specification
open **Pull Request**

Figure 5.5: Dashboard View

6 Evaluation

This chapter describes the usability tests conducted on the implemented GUI. Afterwards, the associated results obtained from the tests are presented. Lastly, we address how to improve the design.

6.1 Usability Testing

For usability testing, the product or service is evaluated by representative users. The participants try to complete typical tasks while observers watch, listen and take notes. The GUI was built with usability in mind for users who have little to no programming experience and the goal was to guide them through the process of creating a configuration so they can easily start refactoring their Java projects and comfortably monitor the behaviour and results of the Refactoring-Bot. The evaluation was performed to understand how user-friendly the system for the intended users was. How long does it take to complete a specified task and how satisfied were they with using the web page? Another point is to detect usability issues which were not considered during the development process. These mentioned aspects are part of the usability defined by Nielsen, see Section 2.5. In this usability test, only the following four out of five factors were considered: **Learnability, Efficiency, Errors** and **Satisfaction**. Since this is the first time that end-users interact with the newly implemented GUI, it was not possible to include **Memorability**.

Nielsen elaborated that only five participants are required to get the best results from usability testing because at a specific point the return is diminishing and ”you are wasting your time by observing the same findings repeatedly but not learning much new”, which is after the fifth participant [Nie]. We invited five participants which three of whom have experience with GitHub and the other two little to no experience. The location of the experiment was at the University in a closed room. The tasks were performed on a middle sized laptop. We decided to write the consent form and usability tasks in German because it is the native language of the participants which can be found in Appendix A and Appendix B. For one participant, we conducted the test remotely with a remote controlling software because of his schedule. Each session with the recruited participants didn't take longer than fourty five minutes. Including the preparation time for the usability tests, all sessions lasted for one hour.

Before conducting the tests, we let them sign the consent form and asked them kindly to think aloud during the sessions so we could understand their thoughts and actions. After introducing them to the topic and answering their questions, all participants were given the same five tasks listed on a paper sheet to absolve the tests. During the tests, we acted as the moderator and also took notes of the user's behaviour and thoughts. When they were struggling or didn't know what the next steps were, we tried not to influence or help them too much and instead asked them how they would proceed without any guidance or what they planned to do. After completing the tasks, we interviewed them to get feedback about the system and how satisfied they were with the design, functionalities and overall experience with the web page. They rated experience on a scale from one to five where one is the lowest score and five the highest in terms of satisfaction.

6.2 Usability Tasks

1. **Gehe zu dem Refactoring-Bot UI in dem geöffneten Web Browser. http://localhost:9000/**

   The purpose of this task was to see how they feel about the loading speed of the web page and let them read what was on the homepage.

2. **Du möchtest dein Java Projekt auf GitHub refactoren und Code Smells beseitigen. Erstelle dafür eine Refactoring Configuration.**

   The purpose of this task was to determine if the process of creating a configuration was user-friendly. How intuitive is the navigation menu? Are the labels easy to understand?

3. **Du möchtest einen Überblick haben, welche Code Smells dein Projekt hat und sie vom Refactoring-Bot beheben lassen. Starte die Refactor Analyse für deine erstellte Configuration.**

   The purpose of this task was to see if they were able to navigate to the dashboard page and find the "Start Refactoring" button to click on it. Additionally, we expected the participant to notice that the timeline didn't automatically refresh the timeline.

4. **Du bist interessiert, was in den Dateien geändert wurde. Finde die Änderungen zu dem erstellten Pull Request.**

   The purpose of this task was to determine if the participants will find out that they can interact with the pull requests in the timeline to get more information about the refactoring.

5. **Da du nun fertig bist und den Bot nicht mehr brauchst, möchtest du etwas aufräumen. Lösche sowohl deinen erstellten User als auch deine Configuration.**

   The purpose of this task was to see how easily the participants can find the edit and delete buttons and how intuitively they can identify the button icons.

6.3 Results of Usability Testing

We analyzed the notes we took during the sessions and the opinions from the interviews.

There are some common problems most users encountered during their session, which is listed here:

- The most criticized point was that the description for generating the GitHub API *Token* was too vague and the link, which opened the GitHub token page, was not perceived as a clickable link (Figure 6.1). Another problem which appeared was the position of this link. The text of the link wasn't read by some of the participants because it was below the input field. They thought it wasn't relevant or didn't belong to the *Token*.

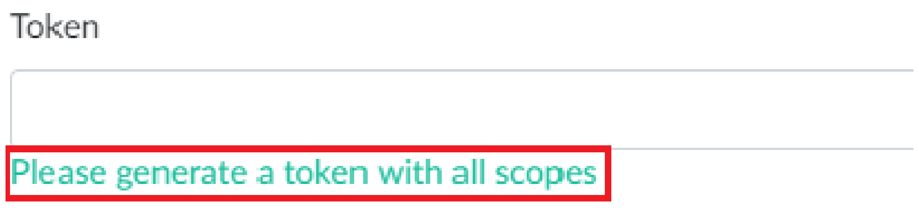


Figure 6.1: Not clearly recognizable link

- Also, the *Analysis Project Key* was difficult to find on the sonarcloud page because the link opened the Example Analysis Project, an external page with many charts and information, which was too overwhelming and brought up confusion.
- Another input field label which was not intuitive, was *Maximum Amount of Concurrent Pull Requests*. Even if the purpose of this attribute was comprehended, three participants hesitated to input a number.

Aside from the ambiguity of the technical terms, the overall feedback from the participants were rather positive. What all of them liked was the intuitive design and how user friendly the system was. They received feedback on the web page if an input was wrong or when they completed an action. The error messages displayed the reason why the action failed. Another design choice they were satisfied with was the consistent layout and styles on all pages. The layout was clear and most of the buttons and icons were recognizable. Navigating to different pages and components was done successfully by everyone without any obstructions.

On a scale from one to five, the first participant gave the system a rating of three regarding overall satisfaction. The following participants rated their satisfaction with the system with a four. The average rating of all five in total is three point eight out of five.

One of the suggestions worth mentioning was the feature to sort and search for pull requests. Another one was to add data visualization and display statistics like merged pull requests in percentage. The participants also expressed the desire to see which files got changed without the need to open the GitHub page. Besides these features, there were some small remarks regarding design improvements.

6.4 Improving the design

We gained valuable data from the usability test.

The first issue we would like to address is the *Token* input field. Since the reading pattern is from top to bottom, we think that moving the description text above the input field will make the user read it more carefully. Currently the user does not know where he can get the token from, because the token field captures his attention. This results in confusion of the user which can lead to frustration and therefore lower the satisfaction with the product. In addition, updating the description of link will improve the success rate.

For the *Analysis Project Key*, it makes sense to insert a default value to reduce the workload of the user. If the user doesn't have his own analysis project, he doesn't need to change the value of this input field. If he does have one, he probably knows his key is or where he can find it because he initially gave his project key a name to create an analysis project on sonarcloud.

The label for *Maximum Amount of Concurrent Pull Requests* will be changed to a simpler description to make it easier to understand. Furthermore, we will add a default value as well so the user does not have to think about it if he does not like change it.

Participant 4 made the valuable suggestion to add a confirmation window to the deleting process of a user or a configuration. This will prevent the user from accidentally deleting a user or a configuration which will save him a lot of frustration.

Participant 1 and 3 noticed that when a user gets deleted, the associated configuration still exists and made a suggestion to automatically delete it along with the user. We already had plans for this case but due to the time limit we were not able to implement a solution. A possible solution would be to automatically delete all configurations of the user as well. Another one would be to let them stay in the database where you can assign a new GitHub user to the configuration and update it. If a user which belongs to a configuration gets deleted, there should be a warning message as well.

Participant 4 mentioned that the footer design was too big, thus reducing the space to present valuable information seems reasonable.

7 Conclusion

In this work, we introduced a Graphical User Interface (GUI) for the Refactoring-Bot which provides a user-friendly interface to create and manage configurations. In addition, the monitoring of pull requests and the communication between developers and the bot should be comfortable. To develop a GUI with good usability we followed User Centered Design (UCD) methods and usability guidelines to meet the user requirements and increase the user's satisfaction. We described how we proceeded to gather and analyze the user requirements before we started the implementation. After we presented our design choices during the implementation, we conducted a usability test against the user requirements. The results obtained from the tests proved the GUI for the Refactoring-Bot to be clean and user-friendly. Except some errors and difficulties, the participants were satisfied with the product.

The drawback of the current implementation is the lack of features and statistics. To evaluate and analyze the work of the bot effectively, a dashboard with multiple components is required. Developing a dashboard would be out of scope of the range of this thesis.

7.1 Future Work

The next logical step would be to implement the solutions for the issues from the usability test to improve the efficiency for the configuration setup process. After that the next steps would be to add a component with all pull requests listed where you can sort, search and filter them. More features for data presentation and visualization can be added to improve the monitoring of the bot's work and Which would allow us to compare the results.

Bibliography

[Ado] Adobe. *Prototyping 101: The Difference between Low-Fidelity and High-Fidelity Prototypes and When to Use Each*. URL: https://theblog.adobe.com/prototyping-difference-low-fidelity-high-fidelity-prototypes-use/ (cit. on p. 26).

[BDD+12] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, O. Strollo. "When does a refactoring induce bugs? An empirical study". In: *Proceedings - 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation, SCAM 2012* (2012), pp. 104–113. DOI: 10.1109/SCAM.2012.20 (cit. on p. 16).

[Boo] BootstrapVue. *BootstrapVue*. URL: https://bootstrap-vue.org (cit. on p. 34).

[For] Foraker Labs. *Introduction to User-Centered Design*. URL: https://www.usabilityfirst.com/about-usability/introduction-to-user-centered-design/index.html (cit. on p. 17).

[Fou] I. D. Foundation. *User Centered Design*. URL: https://www.interaction-design.org/literature/topics/user-centered-design (cit. on p. 19).

[Fow99] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Reading, 1999 (cit. on p. 16).

[Huf15] D. Huff. "B2B Web Usability Report". In: (2015). URL: https://komarketing.com/files/b2b-web-usability-report-2015.pdf (cit. on p. 13).

[Leb18] C. R. Lebeuf. "A Taxonomy of Software Bots: Towards a Deeper Understanding of Software Bot Characteristics". PhD thesis. 2018. URL: https://dspace.library.uvic.ca//handle/1828/10004 (cit. on p. 16).

[MGLV05] F. Montero, P. González, M. Lozano, J. Vanderdonckt. "Quality Models for Automated Evaluation of Web Sites Usability and Accessibility". In: *Human-Computer Interaction - Interact 2005, Proceedings* 3585 (2005), pp. 37–43. ISSN: 0302-9743 (cit. on p. 17).

[Möl18] K. Möller. "Developing a graphical user interface for creating chatbot configurations". In: (2018), p. 33 (cit. on p. 21).

[Nie] J. Nielsen. *Why You need to Test with 5 User*. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (cit. on p. 37).

[Nie12] J. Nielsen. *Usability 101: Introduction to Usability*. 2012. URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (cit. on pp. 13, 18).

[Nie94] J. Nielsen. *Usability Engineering*. Interactive Technologies. Elsevier Science, 1994. ISBN: 9780080520292. URL: https://books.google.de/books?id=DBOowF7LqIQC (cit. on p. 18).

[SN16] E. Samuelsson, M. L. Nguyen. "Development of a dashboard as part of PinDown ' s new graphical user interface". In: (2016) (cit. on pp. 21, 22).

[Tea] B. Team. *Bootstrap*. URL: https://getbootstrap.com/ (cit. on p. 34).

[Usa] Usability.gov. *User-Centered Design Basics*. URL: https://www.usability.gov/what-and-why/user-centered-design.html (cit. on pp. 19, 34).

[VMSC02] K. Vredenburg, J.-Y. Mao, P. W. Smith, T. Carey. "A Survey of User-Centered Design Practice". In: CHI '02 (2002), pp. 471–478. DOI: 10.1145/503376.503460. URL: https://doi.org/10.1145/503376.503460 (cit. on p. 19).

[WB19] M. Wyrich, J. Bogner. "Towards an autonomous bot for automatic source code refactoring". In: *Proceedings - 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering, BotSE 2019* (2019), pp. 24–28. DOI: 10.1109/BotSE.2019.00015 (cit. on pp. 13, 16, 18).

[WSS+18] M. Wessel, B. M. de Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, M. A. Gerosa. "The Power of Bots: Characterizing and Understanding Bots in OSS Projects". In: *Proc. ACM Hum.-Comput. Interact.* 2.CSCW (Nov. 2018). DOI: 10.1145/3274451. URL: https://doi.org/10.1145/3274451 (cit. on p. 16).

[You] E. You. *Vue.js*. URL: https://vuejs.org/ (cit. on p. 33).

All links were last followed on June 08, 2020.

A Consent Form

Beschreibung

Im Rahmen meiner Bachelorarbeit werden Sie zur Teilnahme an einer Studie eingeladen, in der die Usability eines User Interfaces für das Konfigurieren und Monitoren eines Bots getestet wird. Dabei werden Sie gebeten, bestimmte Tasks durchzuführen und dabei Ihre Gedanken laut auszusprechen, während ich Ihr Verhalten und Interagieren mit dem System protokolliere.

Zeitaufwand

Ihre Teilnahme an der Studie wird einmalig 45 Minuten dauern. Bei eventuellen Rückfragen kontaktiere ich Sie anschließend, um diese zu klären.

Datenerhebung

Die erfassten Daten können Ihrem Namen zugeordnet werden und dürfen für wissenschaftliche Zwecke in aufbereiteter und anonymisierter Form von Dritten gelesen und wiederverwendet werden. Bei den erfassten Daten handelt es sich zum Teil um identifizierende Informationen wie beispielsweise bestimmte Verhaltensweise. Die identifizierenden Daten werden von keiner anderen Person als dem Studienleiter gelesen und ausgewertet. Die identifizierenden Daten werden nach der Auswertung gelöscht. Zweck der Datenerhebung ist es die Usability des Graphical User Interfaces für meine Bachelorarbeit zu evaluieren.

Rechte des Teilnehmers

Die Teilnahme an der Studie ist freiwillig. Sie haben das Recht, die Studie jederzeit abzubrechen oder gar nicht erst an ihr teilzunehmen. Aus einem Abbruch ergeben sich keine Nachteile für Sie.

Risiken und Vorteile

Mit der Teilnahme an der Studie entsteht kein Risiko, außer dass Sie mentale und physische Müdigkeit durch die Arbeitsbelastung erfahren. Die Vorteile, die wir aus diesem Usability Test erhalten, helfen uns möglicherweise die User Interface zu verbessern und dadurch die Zufriedenheit des Users mit dem Service zu steigern.

Studienleiter

Bei eventuellen Rückfragen, Bedenken oder Beschwerden, können Sie mich gerne kontaktieren.
Hai Duy Dam – st107877@stud.uni-stuttgart.de

Ich habe die Einverständniserklärung gelesen und verstanden. Ich erkläre hiermit meine freiwillige Teilnahme an dieser Studie.

B Usability Tasks

1. Gehe zu dem Refactoring-Bot UI in dem geöffneten Web Browser.

   http://localhost:9000/

2. Du möchtest dein Java Projekt auf GitHub refactoren und Code Smells beseitigen.

   Erstelle dafür eine Refactoring Configuration.

3. Du möchtest einen Überblick haben, welche Code Smells dein Projekt hat und sie vom Refactoring-Bot beheben lassen.

   Starte die Refactor Analyse für deine erstellte Configuration.

4. Du bist interessiert, was in den Dateien geändert wurde.

   Finde die Änderungen zu dem erstellten Pull Requests.

5. Da du nun fertig bist und den Bot nicht mehr brauchst, möchtest du etwas aufräumen.

   Lösche sowohl deinen erstellten User als auch deine Configuration.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Geislingen, 08.06.2020 Hai Duy Dam

place, date, signature