Diogo Alexandre Santos Amores

UNIVERSIDADE D COIMBRA

1 2 9 0

UNIVERSIDADE D
COIMBRA

Diogo Alexandre Santos Amores

Mining GitLab Repositories for Software Development Activities

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Mário Zenha-Rela and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2021

Acknowledgements

Firstly, I would like to thank my family for all the support they have given me throughout these years and for always believing in me.

To my advisor, Mário Zenha-Rela, thank you for your availability, solid advice and especially, for helping me find motivation when it sometimes lacked to see this through the end.

And finally, to my friends, from the ones I made when I first entered university to the ones who were already there, thank you for making me a better person and enriching my life. I would not be who I am today without you.

Faculty of Sciences and Technology

Department of Informatics Engineering

Mining GitLab Repositories for Software Development Activities

Diogo Alexandre Santos Amores

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Mário Zenha-Rela and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2021

1 2 9 0

UNIVERSIDADE Đ
COIMBRA

This page is intentionally left blank.

Abstract

With the growing complexity of modern societies, organizations become increasingly conscious of the improvements made possible by Process Management to improve workflows and optimize resources. Mining Software Repositories, initially an obscure research topic, has grown to provide invaluable contributions to the field of process management, made only possible by the exponential increase in the usage of software repositories, available event data, and open-source projects.

In particular, software repositories such as GitLab and GitHub contain vast data that is considered of particular interest to researchers in this field. Mining Software Repositories aims to shift these repositories from purely storage spaces to actual tools that help in shaping and improving the development process. The data these repositories contain provide information about the different states of the project during development such as the history of every change made to every file and the responsible contributors. Due to this, these repositories present an excellent starting point for project analysis and provide an opportunity to better understand how a project is being conducted, if established methodologies, schedules, and if work distribution is comparable to the envisioned execution.

The primary goal of this thesis is to mine these software repositories and extract information that can be useful for monitoring software projects. This information can prove extremely valuable in order to improve the effectiveness of teams and identify issues early in order to improve the quality of the produced software. It can also provide insight into the development practices of individual team members.

To achieve this goal, a tool that uses git-based repositories (such as GitHub and GitLab) as the main source to extract event logs was designed and built by the author. The aim of this tool is to extract this data and establish a method to categorize the different contents of these repositories into software development activities (such as Code or Test). After this categorization, multiple aspects of the log will be analyzed and displayed to the user via a user-friendly visual interface.

Keywords

Business Management, Process Management, Mining Software Repositories, Software Development, event logs, workflow, performance, efficiency, GitLab, GitHub, Process Mining

This page is intentionally left blank.

Resumo

Com a crescente complexidade da sociedade moderna, as organizações tornam-se cada vez mais conscientes das melhorias potenciadas por Process Management para melhor fluxos de trabalho e otimizar recursos. Mining Software Repositories, inicialmente um tópico de pesquisa obscuro, cresceu para fornecer contribuições inestimáveis no campo de process management. Estas contribuições foram possibilitadas pelo aumento no uso de repositórios de software, dados de evento disponíveis e projetos open-source.

Em particular, repositórios de software como GitLab e GitHub contêm uma quantidade vasta de dados que são considerados de particular interesse para cientistas neste campo. Mining Software Repositories visa alterar a função puramente de armazenação destes repositórios para ferramentas que possam ajudar a moldar e melhorar o processo de desenvolvimento de software. Os dados que estes repositórios contêm fornecem informação sobre os diferentes estados de um projeto durante o desenvolvimento, como o histórico de alterações feitas em cada ficheiro bem como os autores responsáveis. Devido a isto, estes repositórios apresentam um excelente ponto de partida para a análise de projetos e fornecem uma oportunidade para melhor entender como um projeto está a ser conduzido, se as metodologias estabelecidas, planos e distribuição de trabalho são comparáveis à execução prevista.

O objetivo principal da presente tese é explorar estes repositórios de software de modo a extrair informação que pode ser útil para monitorizar projetos de software. Este tipo de informação pode ser extremamente valiosa de modo a melhorar a eficácia de equipas de desenvolvimento e identificar eventuais problemas com a devida antecedência, a fim de melhorar a qualidade do software produzido. Estes dados podem também fornecer informações relativas às práticas de desenvolvimento de membros individuais de uma equipa.

Para atingir este objetivo, uma ferramente que usa repositórios git (como GitHub e GitLab) como fonte principal para extrair logs de evento foi desenvolvida pelo autor. O objetivo desta ferramenta é extrair estes dados e estabelecer um método para categorizar os diferentes conteúdos do repositório em atividades de desenvolvimento de software (como Code ou Test). Após esta categorização, vários aspetos dos logs serão analisados e exibidos ao usuário através de uma interface visual simples e acessível.

Palavras-Chave

Business Management, Process Management, Mining Software Repositories, Desenvolvimento de Software, logs de evento, workflow, desempenho, eficácia, GitLab, GitHub

This page is intentionally left blank.

Contents

This page is intentionally left blank.

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

This document presents the 'Mining GitLab repositories for software development activities' thesis performed as part of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Mário Zenha-Rela and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

This first chapter of the thesis serves as an introduction to the project, the context and motivations as well as a basic overview of the structure that will be followed throughout the document.

1.1 Context and Motivation

As businesses become increasingly aware of the need for Process management (or Business process management) as a means to better understand a business' practices in order to improve methodologies and the outcome of their work, it becomes inherently obvious that understanding business processes as a whole, how to improve said processes, how to make them more agile and efficient is the root to success for any business to strive in an increasingly competitive market.

Business process management (BPM) is a discipline that uses various methods to discover, model, analyze, measure, improve and optimize business processes. A business process coordinates the behavior of people, systems, information, and things to produce business outcomes in support of a business strategy. Processes can be structured and repeatable, or unstructured and variable [13] and should allow a correlation and understanding of the company's workflow in order for tasks to be efficiently and consistently executed.

As per the work of this thesis, the focal point will be monitoring, and visualization of these processes, as applied to the software development activity. Monitoring of processes can prove difficult for businesses, especially the larger they become due to an increasing lack of centralization of the tools developers use. Tools such as GitLab or GitHub are seen as common usage to address this problem, offering options such as version control, storage of file changes, and multiple logged traces related to the whole life cycle of software development. However, these tools fall short when it comes to actually get insights, namely through visualization of this data which becomes necessary in order to understand the work being done throughout the development and identifying potential roadblocks or deviations of projects even on a small scale. Manual approaches are incredibly vexing or most of the times practically impossible.

The information available in these tools is incredibly useful for analysis purposes and can be deemed as a essential into obtaining more information about a specific project and understanding the different activities done by developers during the course of a project. This will be the starting point of this thesis which can be associated to the research area of Mining Software Repositories. This field has been gathering increased interest over the years has it focuses on analyzing the data present in these tools as they provide an useful opportunity into looking at the overall history of a software project and aid in the monitoring of complex systems without interfering with actual development activities and deadlines [27].

1.2 Goals

The main goal of this thesis is to develop an MVP (Minimum Value Product) of a software tool that we named ProjectScanner, that visualizes information extracted from project repositories.

To accomplish this, we shall extract available logs from these repositories and categorize them into activities by using a default configuration and by enabling the user to create their specific configuration to suit individual projects.

After extracting and categorizing the logs into activities, we will then look into the data obtainable from these logs in order to understand the kind of information we want do display to the user. The information displayed aims to provide a more inside look into the development and helping understand the work of developers and their relationship with the different activities in order to establish if internal methodologies, schedules and work distribution are being properly executed.

Since the extraction of logs from these sources can be associated with the Mining Software Repositories research field, we contextualize what we aim to achieve by introducing this research topic.

As a starting point, we looked into Mining Software Repositories and in specific, the project ActiVCS which we considered very aligned to what we are trying to accomplish in the thesis. This will be the focus of Chapter 2.

There are also plans to further integrate this artifact with other ongoing and future projects of the same nature. This means that the architecture needs to be simple, modular, and easily scalable to be fully integrated into a future system. One of these integrations will be applying Process Mining techniques to the collected project dataset in order to further explore and improve development processes. This was initially the main idea behind the thesis but we ultimately decided to shift priorities and developed ProjectScanner as a baseline work that is useful on its own that provides the groundwork for further Process Mining analysis to be integrated in the future.

1.3 Thesis Structure

In this section, an overview on the focus of this thesis will be presented as well as a brief summary of what each subsequent chapter of the document will entail.

- **Chapter 2 - Background:** The Background Chapter introduces the context of the problem we aim to aid with the overview of the Problem Specification, an introduction to Mining Software Repositories and the analysis a similar artifact to the one we developed.
- **Chapter 3 - Work Plan:** The Work Plan Chapter will focus on the work plan for the First and Second semester. The Gantt Diagrams split per work milestone and a brief description of what was accomplished will be provided.
- **Chapter 4 - Exploration Phase:** The Exploration Phase Chapter will introduce the exploratory work done throughout the development of the ProjectScanner, describing and justifying the decisions that were made during development. Along with this, key topics introducing Process Mining will also be presented here as it contains exploratory material into an earlier approach of the thesis theme.
- **Chapter 5 - System Specification:** The System Specification Chapter introduces relevant Diagrams for the System Architecture, the chapter will provide a template for the overall system diagram and then expand into the specifications for the software being developed for this thesis. After this, the different components relevant to development will be introduced. These entail the User Stories, Use Cases, Functional Requirements, Non-Functional Requirements, Mockups, Navigation Diagram and the Entity-Relationship Diagram.
- **Chapter 6 - Development:** The Development Chapter introduces the organization methodology used throughout development along with the implemented requirements. This chapter also features the Project Structure of the ProjectScanner along with an explanation of key components within it.
- **Chapter 7 - Project Scanner Overview:** The Project Scanner Overview Chapter will introduce the ProjectScanner and all the implemented requirements.
- **Chapter 8 - Conclusion and Further Work:** For this final chapter we will discuss the conclusion topics of the thesis and provide some essential points into future development of the ProjectScanner.

For the next chapter, the objective is to describe in more detail the background work done prior to the development of the ProjectScanner.

This page is intentionally left blank.

Chapter 2

Background

This chapter will feature an introduction to the main problem we wish to address with the development of the ProjectScanner tool. After this introduction, an overview into Mining Software repositories will be presented in order to better contextualize this problem. Finally, we will introduce a project of similar nature to ours, named ActiVCS and analyze the key functionalities behind this artifact.

2.1 Problem Specification

Project Development involves most of the time complicated processes that relate to the coordination of multiple people working towards a specific goal. As mentioned in section 1.1, monitoring these specific development processes is extremely important to ensure that development follows established goals and meets certain criteria.

However, it can prove difficult to have access to the data to monitor and the specific metrics associated with this data as there's a lack of tools that can aggregate and present it in a simple and efficient way to provide actual conclusions either to on-going or already finished projects (*post-mortem analysis*). Software is intangible, making it difficult to correctly establish specific variables that influence the development, leading to bugs, inconsistencies and budget or time limit bypasses [29].

This entails the main objective we wish to aid by developing the ProjectScanner tool. Ensuring the understanding of the different states of projects can be extremely useful in order to provide clarity of the development process and in being able to distinguish the kinds of development activities that were done by each member of a collaborative team. This establishes the possibility to understand how a project is being conducted, if internal methodologies are being followed and the actual work being developed across time. This can lead teams in a good direction when it comes to identifying timely solutions to problems that can be hard to pinpoint without centralization and adequate presentation of the relevant data produced in software development.

Data that can be considered relevant can usually be found in the configuration management systems that teams use to manage and store development software as a result of collaborative work. Software Repositories such as GitLab and GitHub are two of the most popular examples of systems of this nature. These systems actively store data that can be utilized to aid in addressing this problem. This data includes information such as the history related to every change that was made to a file as well as the individual member or members behind it. For this reason, these systems serve as the most promising starting point into project analysis.

The extraction of information from software repositories and the different analysis that can be done with this data is addressed by the researah field 'Mining Software Repositories'. As we established the main issues with monitoring software projects and concluded that data from software repositories provided a good starting point into this problem, the next step into the research was looking into Mining Software Repositories and in specific, an artifact developed in this field of study.

2.2 Mining Software Repositories

2.2.1 Introduction

Mining Software Repositories (MSR) is a research field that has been made popular by the increased use of software repositories as means to store and manage source code. MSR focuses on extracting and analyzing data available in these repositories to uncover interesting, useful, and actionable information about software systems and projects [23].

Even though software repositories are available for most large software projects and are usually standard use within software development teams, the data these repositories contain had not been seen as valuable means to improve software projects until relatively recently. This can be briefly explained by some key issues. Companies in most cases were not very willing to openly provide repository data as it contained very sensitive information about the software systems being developed. This means that varied data for analysis was ultimately scarce. Another issue considered by researchers is the complexity behind the actual data extraction as most system repositories usually do not consider automated data extraction and the mining of its contents, making the support to get this information extremely limited. Usually, researchers are more interested in getting access to already extracted data that is easy to process [16].

With the increasing occurrence of public repositories, a new option to explore MSR became possible and the paradigm began to change as this concept evolved and many high quality software emerged from projects of this nature [27]. These public and open source repositories finally provided valuable data available to be extracted and easily shared among researchers throughout the world. As this field began gathering popularity, it became easier to have access to tools and information that aid in the extraction of the data from these sources. Mining Software Repositories became a renowned and important field within the area of Empirical Software Engineering, originating its own Conference in 2008 (International Conference on Mining Software Repositories) after four years of being the International Conference on Software Engineering's largest workshop [16] and is consistently having many contributions and and high quality papers.

2.2.2 Software Repositories

Software Repositories are obviously the primary object of study within the Mining Software Repositories field of study. As mentioned in 2.2.1, MSR focuses on the extraction and analysis of data from these repositories in order to aid in better software development.

Software repositories can be considered any online of offline instance described as a storage space for data such as source code, software packages, metadata, software defects and other relevant information related to software and its development [3]. Most software repositories also provide additional functionalities such as access control, versioning and branching. These repositories may also offer many built-in security systems such as anti-malware and authentication systems [35] in order to provide safe development storing spaces.

These repositories can be inserted into the following categories:

- **Historical Repositories:** These types of repositories aggregate large amounts of data for record keeping. Some popular examples are:
  - **Source Control Repositories** such as CVS, Mercurial, GitHub and GitLab store files and include metadata related to every change done to the repository. These changes can target files or other contents and are usually represented by commit objects and references to these objects [34];
  - **Bug Repositories (or Bug Tracking Systems)** such as JIRA, Bugzilla or Wrike, focus on maintaining a record on bugs and issues of software during multiple development cycles in one centralized platform. These records usually contain information such as the current behaviour versus the expected, the author and the possible developers who might be assigned to fix the bug/issue [31];
  - **Archived Communications** such as messages, emails and platforms such as Discord, Slack or Microsoft Teams store discussions related with workplace chat, files and application integration [33].
- **Run Time Repositories:** Such as Deployment Logs contain information related to software deployment. These can range from software updates, installation or even software errors or abrupt terminations of the system.
- **Code Repositories:** Such as Google Code and Sourceforge.net actively store source code of a vast quantity of projects.

2.2.3 Application of Mining Software Repositories

The information available in Software Repositories provide excellent and valuable datasets which can be directly applied to aid collaborative teams to understand and manage complex projects without taking time away from other important activities. Research has proven that interesting and actionable results can be delivered from the mining of these repositories, allowing managers and developers alike a new understanding into their systems, resulting in an increase of the software quality and efficiency with less applied cost [7][28].

Mining Software Repositories aims to shift these repositories from purely storage spaces to tools that actually shape and improve the development process. Information in Historical Repositories can potentially be used to capture dependencies between different project components such as functions, documentation files or configuration files, this information can then be used by developers to propagate different changes that relate to specific code dependencies [11]. Other interesting application is establishing direct correlations between concrete development activities (such as Code or Testing) and the different components present in these repositories as well. These correlations can then be used to analyse the evolution and effort of each activity over development cycles [26]. Bug Repositories and Archived Communications can be used to establish a direct connection between these sources and the actual source code being developed. This can help in the reporting of bugs or issues by better by determining which portion of the code is related to a specific bug or issue, reducing significantly the maintenance effort and its cost [7]. Run Time Repositories can be useful in identifying critical execution anomalies by determining usage patterns of different project artifacts across deployments and Code Repositories can be used to identify accurate library usage patterns by looking at library usage across a plethora of projects [16].

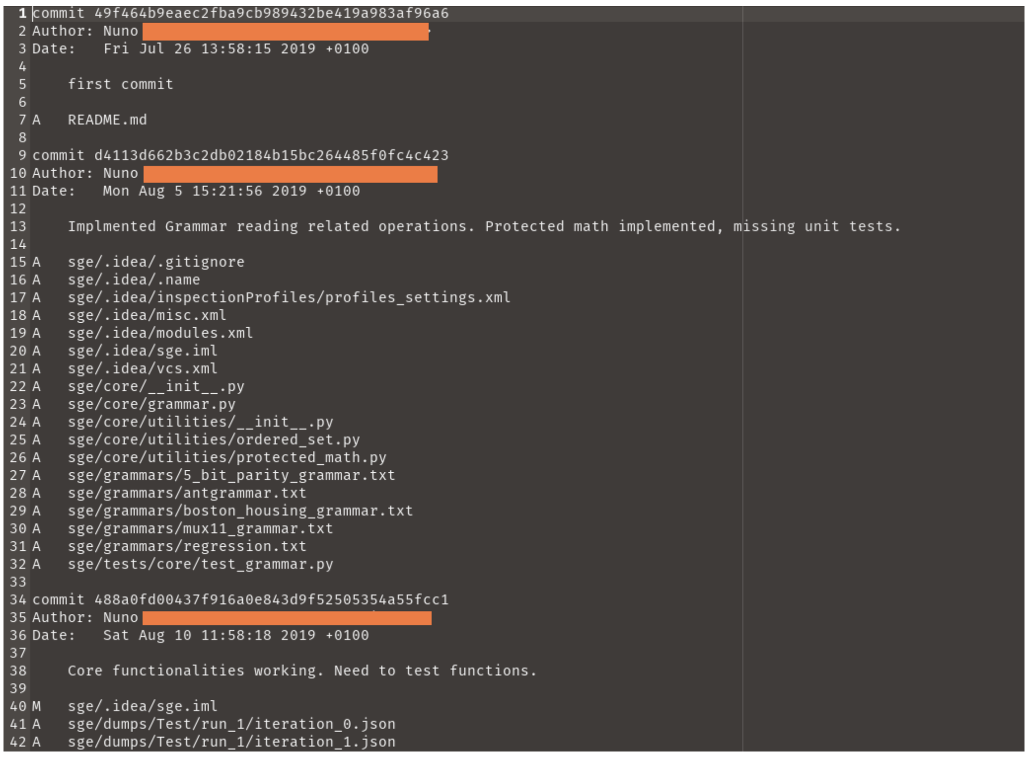
One of the many interesting facts we found during this research was the very close proximity between Mining Software Repositories and the previous topic that this thesis focused on, Process Mining. This topic will be introduced as a part of the Exploratory Phase in 4.2. Mining Software Repositories represents an extremely important field to the application of Process Mining. This can be explained by the fact that in order to explore Process Mining, the required data needs to be extracted from these sources and correctly parsed into a format that the Process Mining algorithms can accept. This ultimately became one of the primary reasons for the shift in priorities in the thesis theme. With the goal of extracting data from repositories, it came as a natural conclusion to us to explore the data we could directly obtain from these sources before moving on to apply Process Mining techniques to it.

As for the next point into the research, the focus was on a specific artifact named ActiVCS. This artifact deals with the extraction and analysis of logs from Historical Repositories and served as crucial point for us into understanding one of the direct applications of MSR.

2.2.4 ActiVCS

ActiVCS is an artifact developed with the intent to analyze event logs extracted from Version Control Systems (a more general term to Source Control Repositories explained in 2.2.2), and use them to discover process activities to better understand the work of developers. While researching for possible ways to interpret the different kind of data we could get with logs this artifact proved useful in establishing some baseline work and solidifying our own ideas.

The first step into using ActiVCS was feeding it a log it could use to extract information. This log focuses on the commits that were pushed to a repository and is structured as presented in 2.1.



```
commit 49f464b9eaec2fba9cb989432be419a983af96a6
Author: Nuno [redacted]
Date:   Fri Jul 26 13:58:15 2019 +0100

    first commit

A   README.md

commit d4113d662b3c2db02184b15bc264485f0fc4c423
Author: Nuno [redacted]
Date:   Mon Aug 5 15:21:56 2019 +0100

    Implmented Grammar reading related operations. Protected math implemented, missing unit tests.

A   sge/.idea/.gitignore
A   sge/.idea/.name
A   sge/.idea/inspectionProfiles/profiles_settings.xml
A   sge/.idea/misc.xml
A   sge/.idea/modules.xml
A   sge/.idea/sge.iml
A   sge/.idea/vcs.xml
A   sge/core/__init__.py
A   sge/core/grammar.py
A   sge/core/utilities/__init__.py
A   sge/core/utilities/ordered_set.py
A   sge/core/utilities/protected_math.py
A   sge/grammars/5_bit_parity_grammar.txt
A   sge/grammars/antgrammar.txt
A   sge/grammars/boston_housing_grammar.txt
A   sge/grammars/mux11_grammar.txt
A   sge/grammars/regression.txt
A   sge/tests/core/test_grammar.py

commit 488a0fd00437f916a0e843d9f52505354a55fcc1
Author: Nuno [redacted]
Date:   Sat Aug 10 11:58:18 2019 +0100

    Core functionalities working. Need to test functions.

M   sge/.idea/sge.iml
A   sge/dumps/Test/run_1/iteration_0.json
A   sge/dumps/Test/run_1/iteration_1.json
```

Figure 2.1: Sample Log File used in ActiVCS

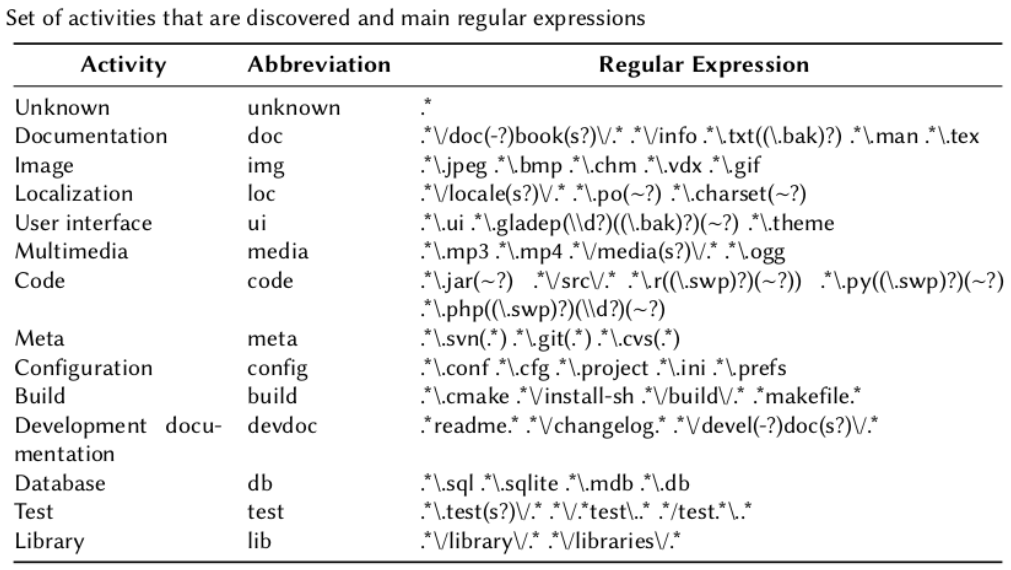
ActiVCS specifies four main steps after getting this log. These steps entail [26]:

- **Preprocess VCS:** In this step information within the log is extracted. ActiVCS considers the following entities from the contents of the log: Project, User, Commit, File and Edit. *Project* represents the repository project the log was extracted from. *User* represents the user who performs the Commit. *Commit* represents the status of the repository at a given state and contains a revision number, timestamp and the author of the Commit. *File* relates to a file contained in the Commit and is represented by its full path (e.g, *sge/.idea/.gitignore*). *Edit* is used to contain the number of lines added and removed from a file.

- **Classify Activities:** In this step, the information is classified into different development activities. To accomplish this, the full path of the files present in the commits is used as the means to categorize activities. ActiVCS then applies regular expressions to the these file paths. The file paths are tested against the Regular Expressions and categorized according to the contents of the full path and the extension of the file. (e.g, a file with *test* in its full path would be considered a Test activity regardless of the extension it had). Figure 2.2 presents the activity set considered by ActiVCS, these activity types are based on [30].

- **Compute Key Performance Indicators (KPIs):** ActiVCS then computes a set of KPIs based on [30]. Key Perfomance Indicators (KPIs) are measurable metrics with the objective of evaluating effectiveness of businesses in achieving specific goals [19]. Figure 2.3 presents the considered KPI sets. ActiVCS divides these KPIs into basic (absolute and relative) and specialization metrics. Basic metrics feature statistics such as frequency counts related to the number of times each user works on a file and specialization metrics aim to measure the imbalance of work between different commit authors.

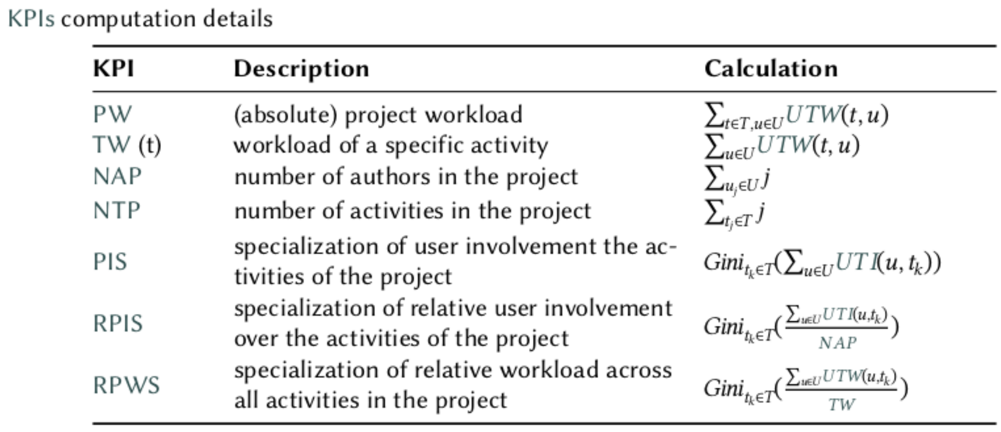
- **Visualize results:** The last step is visualizing the results. ActiVCS considers being informative to project managers extremely important. Due to this, the results are graphically displayed. Figures 2.4 and 2.5 present the different interfaces of ActiVCS.



Set of activities that are discovered and main regular expressions

Activity	Abbreviation	Regular Expression
Unknown	unknown	`.*`
Documentation	doc	`.*\/doc(-?)book(s?)\/.* .*\/info .*\.txt((\.bak)?) .*\.man .*\.tex`
Image	img	`.*\.jpeg .*\.bmp .*\.chm .*\.vdx .*\.gif`
Localization	loc	`.*\/locale(s?)\/.* .*\.po(~?) .*\.charset(~?)`
User interface	ui	`.*\.ui .*\.gladep(\\d?)((\.bak)?)(~?) .*\.theme`
Multimedia	media	`.*\.mp3 .*\.mp4 .*\/media(s?)\/.* .*\.ogg`
Code	code	`.*\.jar(~?) .*\/src\/.* .*\.r((\.swp)?)(~?)) .*\.py((\.swp)?)(~?) .*\.php((\.swp)?)(\\d?)(~?)`
Meta	meta	`.*\.svn(.*) .*\.git(.*) .*\.cvs(.*)`
Configuration	config	`.*\.conf .*\.cfg .*\.project .*\.ini .*\.prefs`
Build	build	`.*\.cmake .*\/install-sh .*\/build\/.* .*makefile.*`
Development documentation	devdoc	`.*readme.* .*\/changelog.* .*\/devel(-?)doc(s?)\/.*`
Database	db	`.*\.sql .*\.sqlite .*\.mdb .*\.db`
Test	test	`.*\.test(s?)\/.* .*\/.*test\..* .*/test.*\..*`
Library	lib	`.*\/library\/.* .*\/libraries\/.*`

Figure 2.2: ActiVCS's Activity Set extracted from [26]



KPIs computation details

KPI	Description	Calculation
PW	(absolute) project workload	$\sum_{t\in T, u\in U} UTW(t,u)$
TW (t)	workload of a specific activity	$\sum_{u\in U} UTW(t,u)$
NAP	number of authors in the project	$\sum_{u_j\in U} j$
NTP	number of activities in the project	$\sum_{t_j\in T} j$
PIS	specialization of user involvement the activities of the project	$Gini_{t_k\in T}(\sum_{u\in U} UTI(u,t_k))$
RPIS	specialization of relative user involvement over the activities of the project	$Gini_{t_k\in T}(\frac{\sum_{u\in U} UTI(u,t_k)}{NAP})$
RPWS	specialization of relative workload across all activities in the project	$Gini_{t_k\in T}(\frac{\sum_{u\in U} UTW(u,t_k)}{TW})$

Figure 2.3: ActiVCS's KPI Set extracted from [26]

In figure 2.3, **UTW** refers to *User Activity Workload* and is computed using the number of files relative to an activity t that a user u edits over the course of a project. **UTI** indicates *User Activity Involvement* and is 1 if an user u has edited a file related with an activity w at least once and 0 otherwise [26].

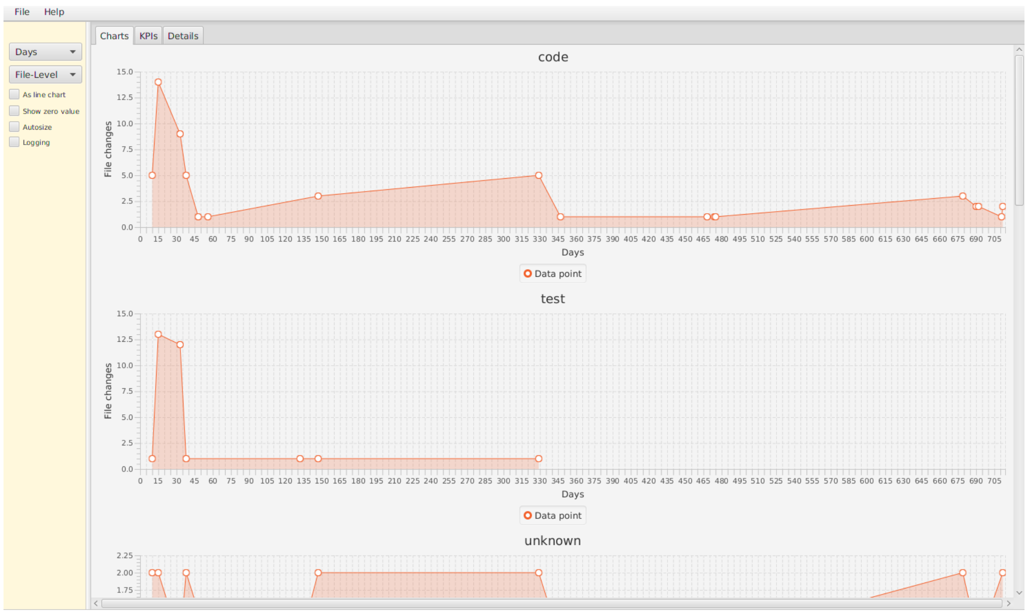


Figure 2.4: ActiVCS's Chart Visualization of activity at File Level

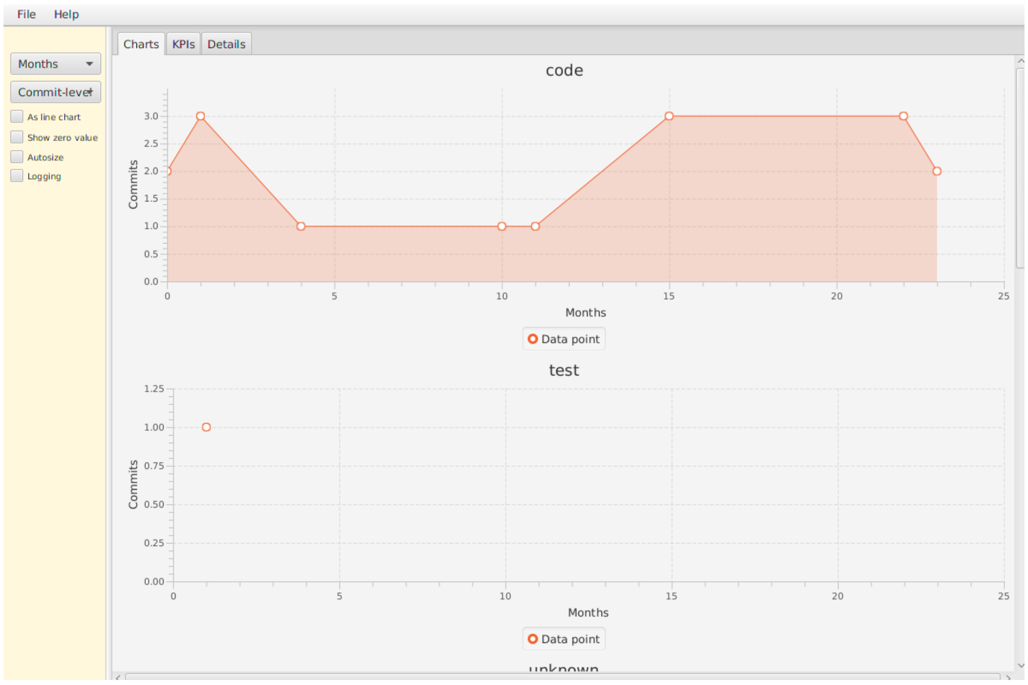


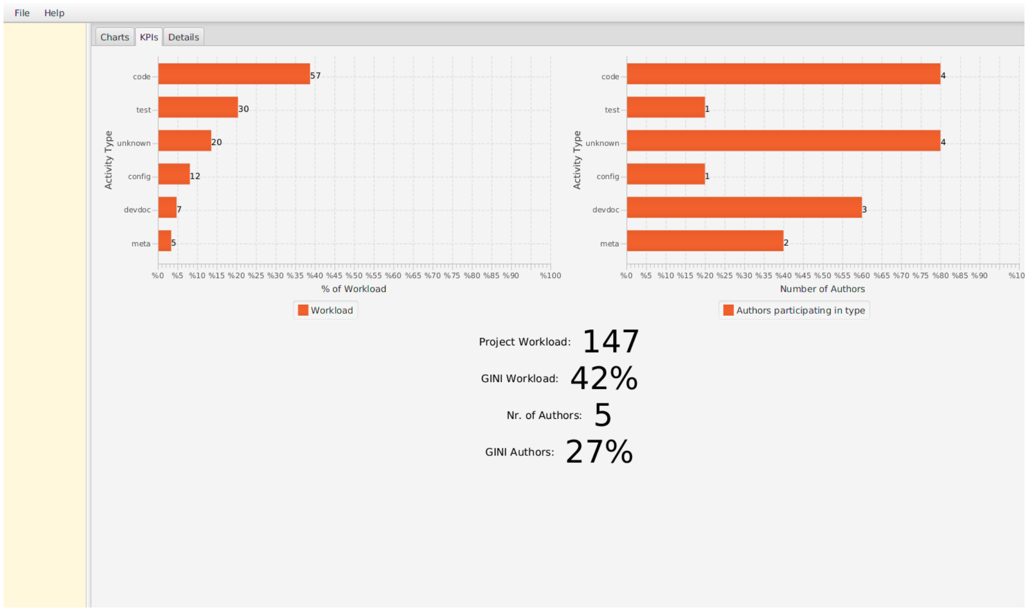
Figure 2.5: ActiVCS's Chart Visualization of Commit activity

Figure 2.4 and 2.5 present the visualization of the data fetched after manually creating and loading a log file from a VSC repository using ActiVSC. The information from the log is sorted into different activities with each activity displaying a chart that visualizes specific information.

Figure 2.4 provides information about File Changes in the repository according to the type of activity across time. Using as an example the Code activity in the same figure, we can visualize how the number of file changes across the development period shaped the workflow of the Project. This can prove useful to identify critical points in development cycles. Figure 2.5 follows the same logic but focuses instead on the number of Commits associated with each category and their relative evolution across the timeline.

For instance, we can establish looking at the evolution shown in the two charts in 2.4 that the Test and the Code activity started at mainly the same time. The peak in changes for these two activities were in the beginning of the development of the Project, where both activities had similar levels of changes. Code had a decline in File Changes followed by another peak and eventually stabilizing across time. The Test Activity stabilized as well but much earlier than the Code Activity (which is not a good indicator of quality focus).

By specifically looking at 2.5, we can conclude an interesting fact. Only one Commit was fully associated with the Test category, meaning that throughout the development of this Project, multiple Commits associated with other activities also had aspects related with the Test Activity (otherwise, there would be no curve of File Changes detailed in 2.4. This can mean that the majority of the developers of this Project were working on multiple activities at once and there wasn't an inherit focus on the Test activity by a singular user (this might also indicate the adoption of a test-driven-development approach). The files associated with the Code category also had an increased amount of effort across time when related to the test files. Depending on the nature of the project and how important these tests are to the development of the project, we can eventually correlate eventual faults in the code to specific Commits that were pushed after the last Test file change.



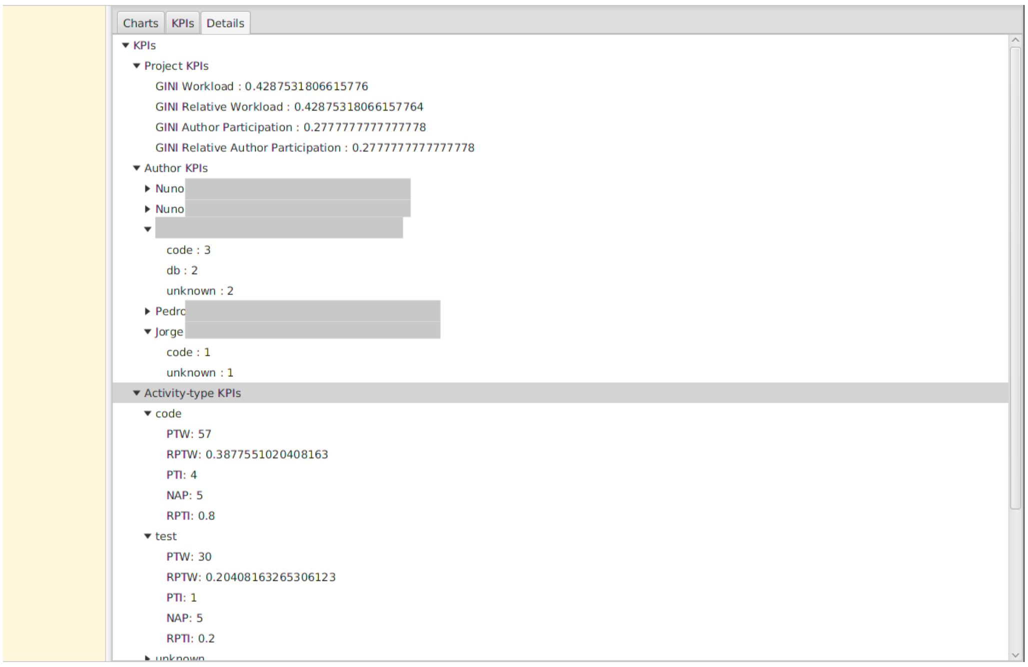
Project Workload: 147

GINI Workload: 42%

Nr. of Authors: 5

GINI Authors: 27%

Figure 2.6: ActiVCS's displayed information on the KPIs



```
▾ KPIs
  ▾ Project KPIs
      GINI Workload : 0.4287531806615776
      GINI Relative Workload : 0.42875318066157764
      GINI Author Participation : 0.2777777777777778
      GINI Relative Author Participation : 0.2777777777777778
  ▾ Author KPIs
    ▸ Nuno [redacted]
    ▸ Nuno [redacted]
    ▾ [redacted]
        code : 3
        db : 2
        unknown : 2
    ▸ Pedro [redacted]
    ▾ Jorge [redacted]
        code : 1
        unknown : 1
  ▾ Activity-type KPIs
    ▾ code
        PTW: 57
        RPTW: 0.3877551020408163
        PTI: 4
        NAP: 5
        RPTI: 0.8
    ▾ test
        PTW: 30
        RPTW: 0.20408163265306123
        PTI: 1
        NAP: 5
        RPTI: 0.2
    ▸ unknown
```

Figure 2.7: ActiVCS's displayed information on the KPIs

Figures 2.6 and 2.7 present the associated information related with the KPI set that ActiVCS implemented into the artifact.

Figure 2.6 features a bar-chart relating each activity type with the TW (workload of a specific activity) KPI and a bar-chart that relates the number of authors that were involved in a specific Activity Type. Below the bar-charts, the specific values of other KPIs are displayed and contain:

- **Project Workload:** Contains the computed value of the *PW* KPI in 2.3, the higher the value, the higher the workload. This KPI can be useful in comparing the different PW among projects.
- **GINI Workload:** General imbalance of the workload. The higher the number, the higher the imbalance of the workload within activity types. As with the PW KPI, this KPI can be useful for comparing workloads among projects.
- **Number of Authors:** Number of Authors that were involved in the project.
- **GINI Authors:** Imbalance related to the Authors. The higher the number, the more imbalance there was between the workload of authors. As with the PW and GINI Workload KPIs, this KPI can be useful for comparing workloads among the authors in different projects.

Figure 2.7 contains more KPI related information. This information is split into Project KPIs, Author KPIs and Activity-type KPIs and feature similar information to Figure 2.6. Author KPIs display the workload of each author split by Activity type and Activity-type KPIs feature information related to the KPIs in 2.3.

With this introduction of ActiVCS, we can already establish different ways in which we can explore the logs to visualize different information from using MSR. Some aspects of ActiVCS will be kept in mind when developing ProjectScanner and this introduction served as a good stepping stone into using MSR and interacting with the available data in a mostly visual and engaging way that can be considered useful for developers and managers as means to better understand the data they are working with.

Hopefully the reader is also aroused to the significant value that this information can provide to the team and project managers, not only during the post-mortem reflection on a project, but also during its active development.

For the next chapter, before featuring all the of the exploratory work that went into the development of ProjectScanner, it's important to introduce the Work Plan now that an overview into the main topics was discussed. The Work Plan chapter will detail how the preliminary work was split into tasks and the specific timeline for each.

Chapter 3

Work Plan

This chapter presents the exploratory work displaying the Gantt diagram of the preliminary work done up until the current point in the thesis and detailing how each task was split throughout the semesters.

3.1 First Semester

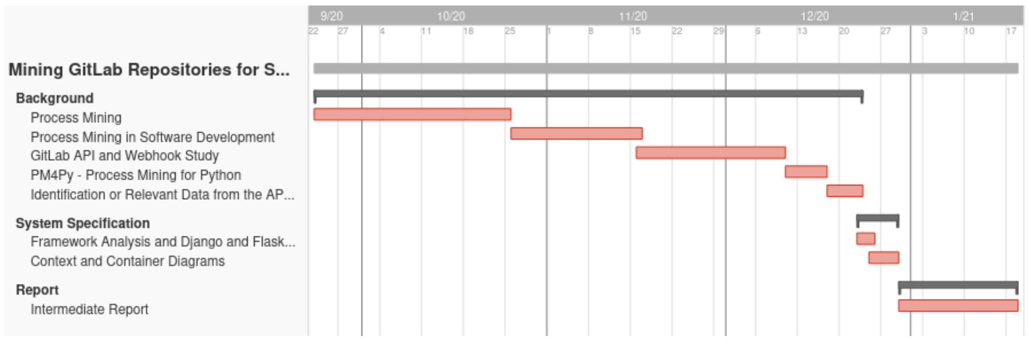


Figure 3.1: Gantt Diagram for the First Semester

For the first semester, the first step into the thesis was the research into Process Mining as a whole, which was a complete new field that the author had no previous experience with. Due to this, specification into the focus of the thesis was necessary and in order to complement the Process Mining study and apply it to the main theme of the project, an analysis into Process Mining in Software Development started.

After the learning of the core concepts was completed, a study into the API and Webhook of GitLab was necessary in order to understand implementation specifications and how to directly apply the GitLab tools to process mining. Following this was an overview into the analysis tool that would produce the process models using process mining algorithms and the relevant data to construct the dataset was surveyed.

After the Background was complete, the knowledge presented allowed a more precise look into possible frameworks for the implementation of the project. Following the decision, the context and container diagrams of the system were built.

3.2 Second Semester

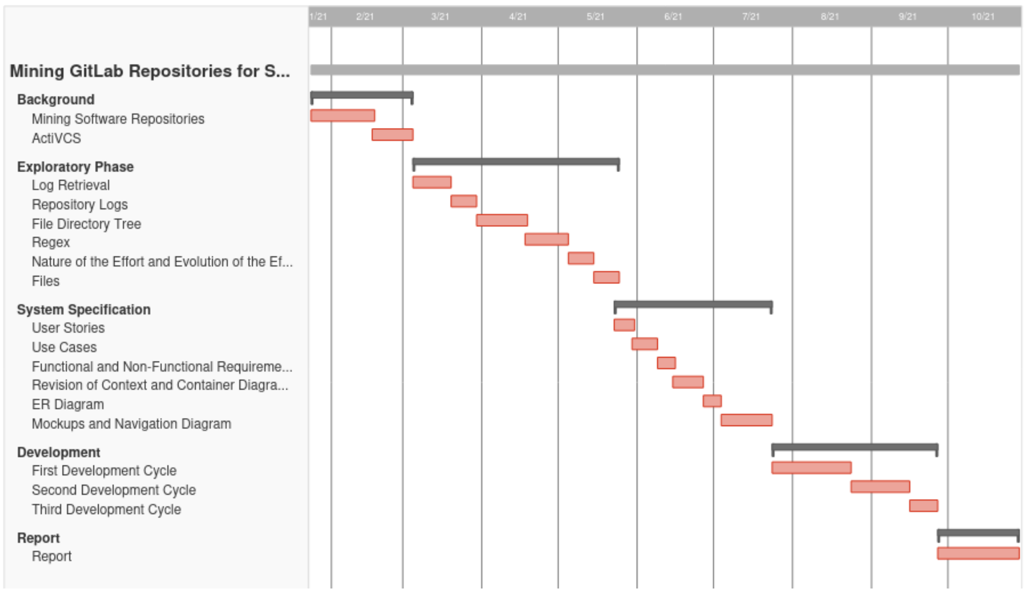


Figure 3.2: Gantt Diagram for the First Semester

For the second semester, as our understanding of the nature of the tool evolved, we shifted focus into researching Mining Software Repositories. The first step consisted in researching this field and searching for existing tools that worked with the data from software repositories. Then, a study of our own was conducted in order to build our own application. The exploratory nature of this thesis meant that there was no defined structure to follow, this meant that topics would be explored as they appeared in research and were deemed relevant for future discussion.

After the exploratory phase was completed, system specification was revised from the first semester and further enhanced with development work of the ProjectScanner. This meant User Stories, Use Cases, Functional/Non-Functional Requirements, ER Diagram, Mockups and Navigation Diagrams all had to be discussed and built in order to establish the necessary steps for a smooth development proccess. The Container Diagrams were updated as some of our previous decisions had to be adapted to the new direction of the ProjectScanner.

Development cycles were established in order to guide development and finally, the final report for the thesis concluded.

3.3 Methodology

Since this is an exploratory research structure work was structured in one 'week timeboxes': after discussing with the supervisor the goals for the week, that goal becomes the focus of the thesis work. At the end of each week, the work performed was presented and discussed. After discussion, the next step was devised and executed the following week.

When the development cycle began, prior checkup meetings would be decided. Each meeting would present some of the established software requirements and report the progress or problems into them. These steps would guide the route of development and decide on the next checkup meeting.

This chapter allowed an outlook of the contents of the report and how each task was assigned in the timeline leading to the report. The planning for both semesters is specified and the methodology allows an understanding of how of how the goals of the thesis were achieved.

In the next chapter, the focus will be on documenting and detailing all the preliminary research that helped shape the ProjectScanner and the development phase.

This page is intentionally left blank.

Chapter 4

Exploratory Phase

This chapter presents the exploratory work that went into the development of ProjectScanner. This will include choices in the many technologies present in the project as well as the baseline logic behind each implementation decision.

The initial idea behind the Thesis had Process Mining and an increased focus in GitLab in mind for development. Due to this, the chapter will also present the work done and the technologies explored into this topic as a possible proof of concept for building upon the ProcessScanner. While these topics will be split into their own sections, portions of sections about Process Mining will be relevant to ProjectScanner and may be referenced as means to better explain the decisions made and contextualize the reader.

4.1 Project Scanner

4.1.1 Framework

When discussing frameworks, there was not any inherit restrictions from the start aside from the fact that the produced software should be a web service. With this in mind, the options were narrowed to frameworks we had already experience working on, decision which could improve the quality of the developed code and the speed of development due to fact that no time would need to be allocated to learn the framework and programming language.

Our initial approach was constrained by the adoption of PM4Py when selecting frameworks. PM4Py is a python library that allows the extraction of process models from datasets using Process Mining technologies. An overview into PM4Py can be found in 4.4 below.

With the objective of implementing Process Mining analysis as mentioned in 1.2, it seemed sensible to keep this library in mind and choose a python based frameworks. While PM4Py could be used with other languages a lot of work can be reduced by using the same programming language. To add to this point, python based frameworks are excellent options for building web services with the author already having experience with two of the more common and popular ones, **Django** and **Flask**.

4.1.2 Overview of Django and Flask

Django is an open source high-level python web framework built for web development in mind. Django helps write code that is complete, following the "batteries included" philosophy [10], featuring a vastly versatile standard library that encapsulates most development needs without the need to resort to outside libraries. It is also easily scalable due to its "shared-nothing" architecture [10], meaning that every part of the Django architecture is independent from each other and can easily be replaced for scalability. Another great advantage is that it's incredibly secure, featuring default protections against many common security attacks such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking [10].

Django also features an automatic admin interface, allowing simple readability of a database as well as the performance of basic database functions directly within this interface.

Django follows the MTV (Model-Template-View) and has its own ORM (object-relational mapping) allowing for database query code to be written using Django's model system, making it much simpler to handle database functionalities.

Flask is a python micro web framework for web development. The main difference between Django and Flask is that Flask is more minimalist in nature in order to be much lighter, modular and simpler. It doesn't offer most of the options Django provides by default such as the ORM or MTV. Flask provides the flexibility to make the developer decide what they need and focus on it, the standard web development library is very straightforward and offers more overall customization than Django, allowing much more room for experimentation.

The ProjectScanner could certainly be developed in either Django or Flask, the decision ultimately was based on ease of development, since our focus is on supporting research, not on a final product. While the Flask simplicity could offer some benefits and allow room to experiment different things for the application, Django offers a set of really useful features for the development of the ProjectScanner that are available from the start, namely the ORM and Django Admin interface and the secure database options. ORM and Admin interface are still possible with Flask, however these require some effort configuring, making it hard to choose Flask over Django.

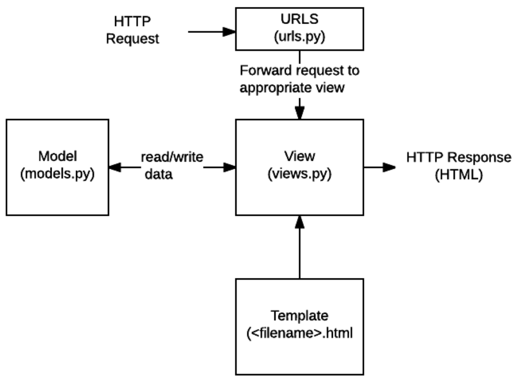
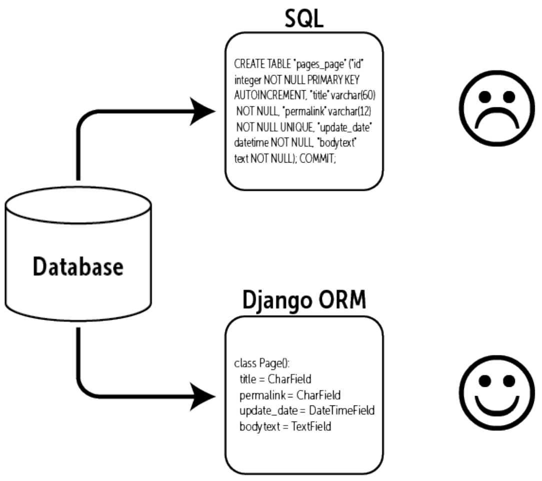


Figure 4.1: Django MTV (extracted from [10]).



```
CREATE TABLE "pages_page" ("id"
integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "title" varchar(60)
 NOT NULL, "permalink" varchar(12)
 NOT NULL UNIQUE, "update_date"
datetime NOT NULL, "bodytext"
text NOT NULL); COMMIT;
```

```
class Page():
  title = CharField
  permalink = CharField
  update_date = DateTimeField
  bodytext = TextField
```

Figure 4.2: Django ORM (extracted from [4]).

4.1.3 Log Retrieval

One of the first steps into building the ProjectScanner was establishing how to get the repository logs for analysis. A key aspect we considered when fetching the logs included an automatic process behind it. This meant that the platform would take care of the details behind getting the logs in order to make ProjectScanner user friendly and streamline some of the aspects relating to getting the actual data.

The initial research into Process Mining lead to an overview into the GitLab API and the GitLab Webhook as baselines for the implementation. This research will be mentioned further into this chapter in sections 4.3.1 and 4.3.2 as contextualization but the main aspect into consideration is that Process Mining greatly benefits from a stream of constant real-time event logs, making the adoption of Webhook a promising approach to explore in the future with the API aiding in getting more specific information when needed. This was not a requirement with the ProjectScanner due to it being focused on the readily-available information within the logs. Nonetheless, we decided to that we would specify a requirement that would allow automation in the updating of the logs to accommodate this exploration in the future as well as aiding in easily keeping the logs of the ProjectScanner relevant.

The next step was deciding on the tool to help with extraction. One of the other advantages of choosing python-based frameworks is the vast community support on libraries. We easily came across the **Gitpy** library. This library is a python wrapper for Git and allows functions such as log extraction and creating automatic commits in python[15]. As such, we used the Gitpy library to perform the streamline proccess we discussed earlier. Gitpy would allows us to:

- Extract the required logs from Git based repositories.
- Streamline this extraction by requiring only the repository URL instead of the logs.
- Allow log updating, guaranteeing that the logs contain their most up to date version.

The standard interaction of ProjectScanner with this library can be described as follows:

1. An URL to a Git-based repository is manually provided by the user (in the case

of private projects, a different kind of URL must be provided, one of the possible approaches is mentioned in 4.3.2).

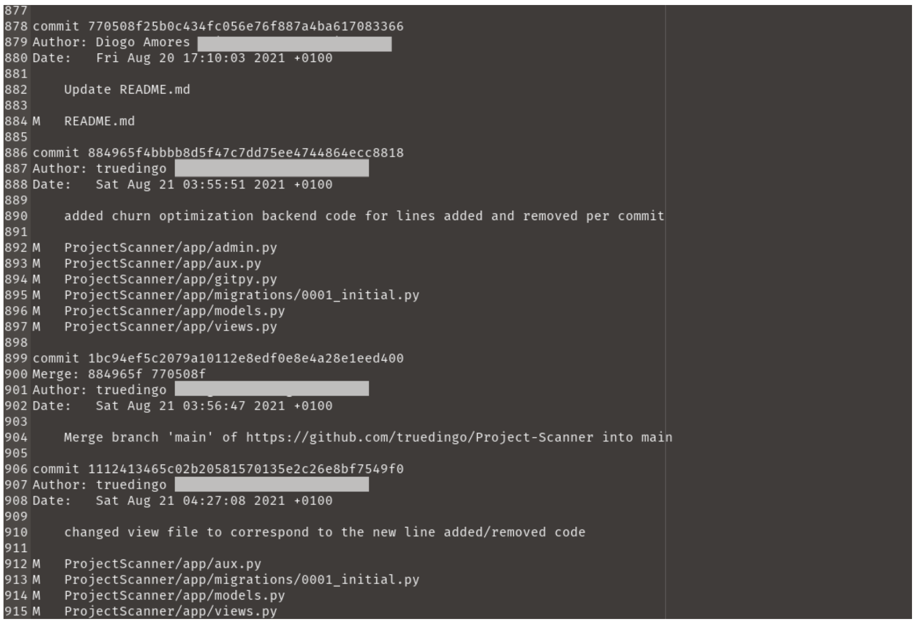
2. This URL is then used with GitPy to create a local (temporary) clone of the project.
3. The required logs are extracted.
4. The local clone of the project is deleted.

As a final note into this topic, we found very late into development the existence of a framework named **PyDriller**. This framework helps developers analyze Git repositories. It functions similarly to Gitpy but provides some increased ease of use by providing methods to get specific information within the logs. This would essentially remove the need to develop a parser for the logs. However, this parser was already implemented and was working as intended. Due to this, we decided to maintain our usage of the **Gitpy** library but document a simpler alternative in case this step needed to be revisited in the future.

4.1.4 Repository Logs

After establishing all the main steps of interaction with the retrieval of logs, we focused on analysing the actual content of the logs in order to understand what data these logs contain and the overall relevancy of its contents to what we aim to achieve with the development of the ProjectScanner.

ProjectScanner works with two types of logs, we will be referring to them from this point onward as the **Commit Log** and the **Lines Log**.

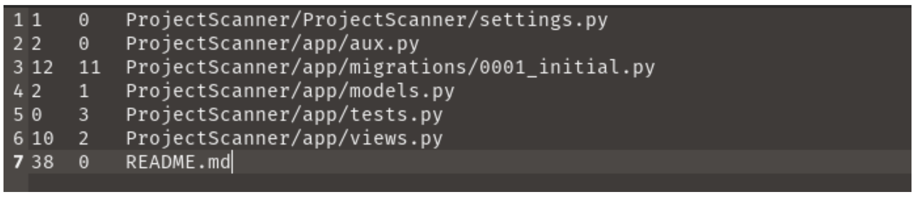


```

commit 770508f25b0c434fc056e76f887a4ba617083366
Author: Diogo Amores [redacted]
Date:   Fri Aug 20 17:10:03 2021 +0100

    Update README.md

M   README.md

commit 884965f4bbbb8d5f47c7dd75ee4744864ecc8818
Author: truedingo [redacted]
Date:   Sat Aug 21 03:55:51 2021 +0100

    added churn optimization backend code for lines added and removed per commit

M   ProjectScanner/app/admin.py
M   ProjectScanner/app/aux.py
M   ProjectScanner/app/gitpy.py
M   ProjectScanner/app/migrations/0001_initial.py
M   ProjectScanner/app/models.py
M   ProjectScanner/app/views.py

commit 1bc94ef5c2079a10112e8edf0e8e4a28e1eed400
Merge: 884965f 770508f
Author: truedingo [redacted]
Date:   Sat Aug 21 03:56:47 2021 +0100

    Merge branch 'main' of https://github.com/truedingo/Project-Scanner into main

commit 1112413465c02b20581570135e2c26e8bf7549f0
Author: truedingo [redacted]
Date:   Sat Aug 21 04:27:08 2021 +0100

    changed view file to correspond to the new line added/removed code

M   ProjectScanner/app/aux.py
M   ProjectScanner/app/migrations/0001_initial.py
M   ProjectScanner/app/models.py
M   ProjectScanner/app/views.py
```

Figure 4.3: Commit Log

Figures 4.3 and 4.4 represent two examples of the Commit Log and Lines Log that the ProjectScanner interacts with. The Commit Log presents an accurate history of every change made to the repository since creation and is structured as follows:

- **Commit ID:** The unique identifier associated with every commit in a repository. It can be used to access other repository information that is not available with the logging we are using.



```
1   0   ProjectScanner/ProjectScanner/settings.py
2   0   ProjectScanner/app/aux.py
12  11  ProjectScanner/app/migrations/0001_initial.py
2   1   ProjectScanner/app/models.py
0   3   ProjectScanner/app/tests.py
10  2   ProjectScanner/app/views.py
38  0   README.md
```

Figure 4.4: Lines Log

- **Merge ID:** Unique identifier relating to a branch merge within the repository. As with the Commit ID, it can also be used to access additional merge specific information.
- **Author:** Identification of the author who pushed the commit.
- **Commit Message:** A text message of variable length. The main objective is describing the nature of the changes made with the commit.
- **Commit Changes:** Identifies all the files that were changed in any way by the current commit. Files can have one of three associated changes:
  - **Added (A):** Indicates that the file was added to the repository;
  - **Modified (M):** Indicates that the file was modified in some way, this entails any change to the number of lines the file originally had;
  - **Deleted (D):** Indicates that the file was removed from the repository.

  In the Commit Log, these changes are displayed firstly by the nature of the change (A, M or D), followed by the full file path of the affected file in the repository. This field is also of variable length.

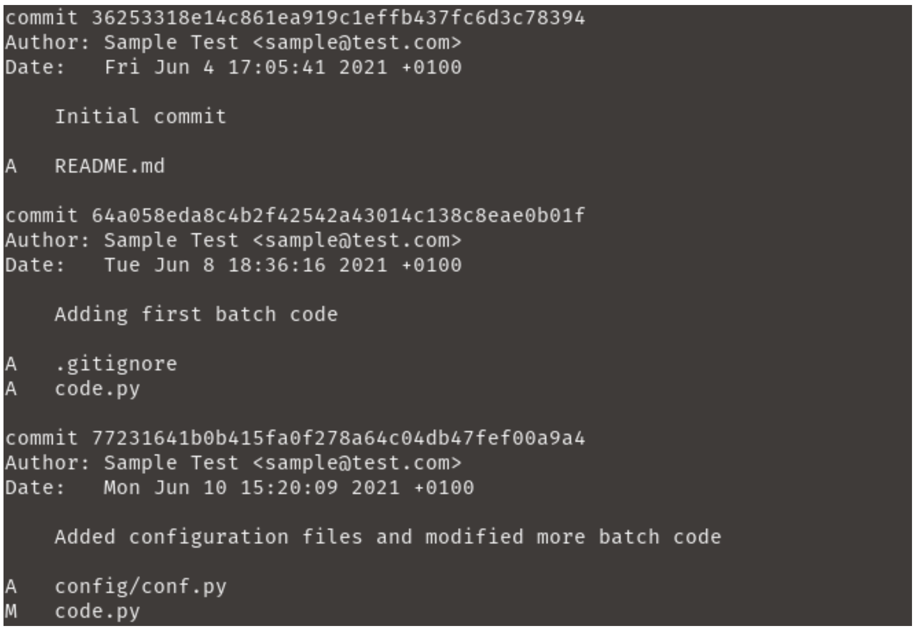
The Line Log was introduced later in development as we found it relevant for the user to be able to check the history of lines added and removed from each file. The Lines Log requires a commit ID, meaning that the Lines Log in 4.4 represents the information on the number of added and removed lines related to the affected files of a specific commit. It is also of variable length and is structured as follows:

[*added lines*] [*removed lines*] [*full path of the affected file*]

4.1.5 File Directory Tree

The File Directory Tree was one of the first features to the ProjectScanner that we discussed during meetings. As we explored this feature, it also began to shape the development of the ProjectScanner as new feature concepts began to arise from building upon this idea.

After going through different Commit Logs, we realised that we could reconstruct the contents of the repository at any point in time. To accomplish this, we needed to traverse the Commits in chronological order as we could look at the files that were added as a result of each commit in order to recreate the history of the repository at any specific time.



```
commit 36253318e14c861ea919c1effb437fc6d3c78394
Author: Sample Test <sample@test.com>
Date:   Fri Jun 4 17:05:41 2021 +0100

    Initial commit

A   README.md

commit 64a058eda8c4b2f42542a43014c138c8eae0b01f
Author: Sample Test <sample@test.com>
Date:   Tue Jun 8 18:36:16 2021 +0100

    Adding first batch code

A   .gitignore
A   code.py

commit 77231641b0b415fa0f278a64c04db47fef00a9a4
Author: Sample Test <sample@test.com>
Date:   Mon Jun 10 15:20:09 2021 +0100

    Added configuration files and modified more batch code

A   config/conf.py
M   code.py
```

Figure 4.5: Sample Commit Log

Using the three commits in 4.5, the File Directory Tree per Commit would be represented the following way:

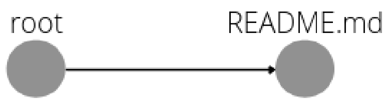


Figure 4.6: File Directory Tree after the First Commit

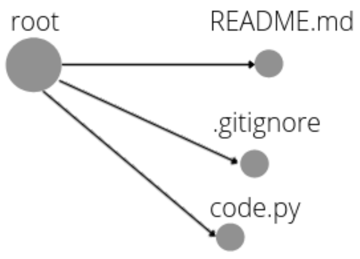


Figure 4.7: File Directory Tree after the Second Commit

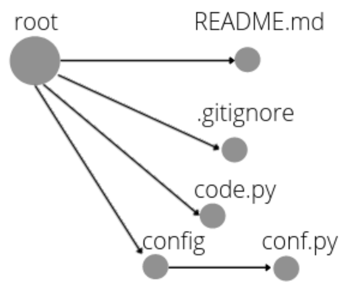


Figure 4.8: File Directory Tree after the Third Commit

We found this process of visualizing the evolution of the repository across time extremely interesting and also concluded on some ways it could prove useful. While platforms such as GitHub and GitLab allow the user to consult this information, it takes multiple actions to get successive outlooks. The insight this provides is not very meaningful as it is built mostly for browsing through the many different folders and files of a project and not with a perspective of monitoring this process as development advances.

Having the File Directory in the repository presented as a tree could allow developers and managers to have an increased outlook at how a project is being structured across development, allowing for more control over the project. Teams who apply *agile* methodologies, focusing on short development cycles and rapid production can greatly benefit from an outlook like this. With the rapid development of software and files, it can be hard to correctly keep track of project evolution. The File Directory Tree can help in this scenario by displaying exactly what happened between commits and showcasing the actual project structure across time.

We expanded further into this concept as we wanted to be able to get more information about the individual files within the tree. This is where we established the use of an additional log to the ProjectScanner, the **Lines Log** covered in 4.1.4. With the Lines Log, we discussed the possibility to color code the files based on the number of lines being added/removed at that point in time, similarly to an heat map. While the number of lines added/removed does not necessarily correlate with the effort being applied on a certain file (due to refactors or slight code organization changes), it can serve as decent indicator into what files are being worked on the most at a certain time and can help in shaping decisions and pinpointing eventual problems. We decided to name this type of tree the CHURN File Directory Tree.

The next point of discussion was how to effectively display the File Directory Tree to the user. The first option we explored was the python library **anytree**, this library provides simple data tree structures as well as built-in methods for tree creation and allows the generated trees to be exported to many formats.

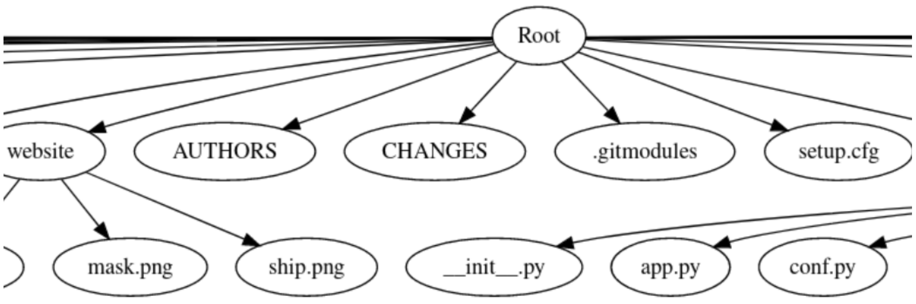


Figure 4.9: File Directory Tree using *anytree*

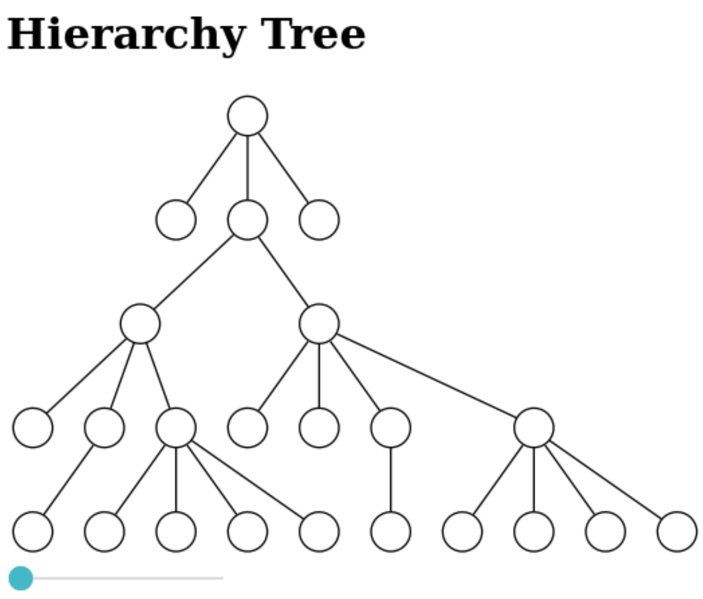


Figure 4.10: Variation of the File Directory Tree using *anytree*

Figure 4.9 and 4.10 present the first variations of the File Directory Tree using anytree. Unfortunately, we had problems implementing the color coding. The way we were exporting the Tree also did not help as we were using an image to display the tree on the front-end. This would also compromise any possible interactive elements that could be added to the dynamic of the File Directory Tree.

Due to this, we focused on finding an alternative solution. The solution would be the usage of **D3.js**. D3.js is a JavaScript library for manipulating documents based on data [8] and there exists a large community support of this library. One of the tools we found incredibly useful was the **D3.js graph gallery**, which provides vast amounts of charts which are easily editable, allowing for extra customization and interactivity which we did not find possible with our initial approach.

After some research, we decided to display the File Directory Tree as a dendogram to the user. Dendograms are diagrams used to display trees and focus on representing hierarchical structure which we found fitting for the purpose of the File Directory Tree.

Figure 4.11: Sample Dendograms (image extracted from [12])

As far as implementation details, we used the basic Dendogram template available on the D3.js Graph Gallery to present the Dendogram. The D3.js dendogram requires the data to be specified in JSON and in a specific format. We initially tried to use the **anytree** generated tree exported to JSON and adapt the formatting to work with the D3.js dendogram template but problems arose. We eventually decided to construct the tree from scratch using python, as this would allow us complete control over the formatting and make the integration with the template much easier. The approach was based on using python dictionaries to construct the tree. The file paths available from commits were split by the "/" token, which would allow us to know their exact placement within the directory hierarchy. For instance, a file path *src/code/code.py* would be divided in three dictionaries, where *code* would be a dictionary that had as a parent the *src* dictionary and the *code.py* dictionary would have as a parent the *code.py* dictionary. The number of lines added/removed per file would be used as an indicator to assign a specific color to that file. This lead to 4.12 as one of the first instances of our implementation of the File Directory Tree.

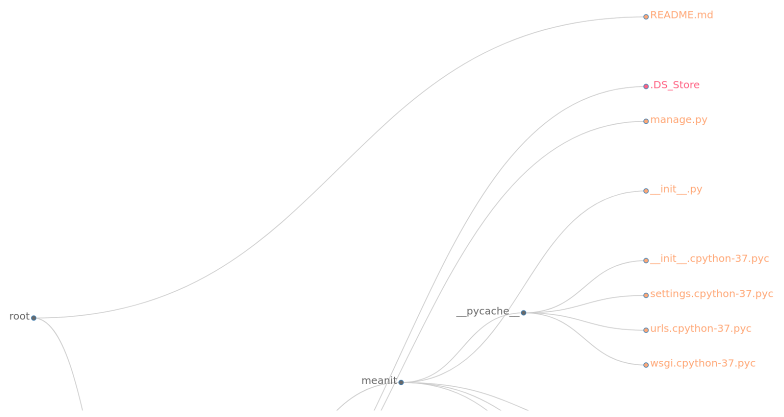


Figure 4.12: Implemented D3 Dendogram with color coding

4.1.6 Regex

As the development continued, we noticed a similarity in purpose of our tool when compared to the tool we discussed in 2.2.4. The File Directory Tree feature aligned with the monitoring aspect of contents of repositories that ActiVCS builds upon. With this in mind, we searched for relationships we could find between our own tool being developed and ActiVCS. This lead to the 'Regex' feature.

Software Repositories such as GitLab and GitHub offer no way to distinguish between actual development activities (such as Code or Testing), making them process unaware. Looking at our own approach with the File Directory Tree, there is also no concrete way to tell apart specific files and what software activity could be related to said file. ActiVCS provides a solution to this problem by establishing a direct relationship between the filetype or filepath of the file and the development activity that could be involved when modifying it.

We concluded that a similar approach would greatly increase the usefulness of the ProjectScanner and one of the immediate uses we had for this feature was applying the different development activities to the File Directory Tree using color coding. This would establish a concrete way to distinguish software development activities, which is useful for monitoring and built directly into our File Directory Tree feature.

For the implementation of this feature, we used ActiVCS as a reference. In order to classify the different contents of the repository into activities, the full file paths of the files available from the Commit Log are categorized after their extraction. To categorize the files, these file paths are tested with a set or regular expressions, each set of regular expressions is associated with a development activity. For the majority of the testing, we used the same set of regular expressions and activities to the one ActiVCS uses. We slightly tweaked the Regular Expressions as the testing into the Regex advanced. A sample of the activities and some regular expression rules is referenced in figure 2.2.

This implementation is not flawless as it's extremely difficult to correctly categorize a nearly unlimited range of file paths, whose activities are largely dependant on the programming language being used, the framework, among many other types or variables. Software Development is also not a process that follows a strict set of rules as there are an extended amount of ways to run projects, there is also no guarantee that all projects use the activities that we initially specified. Due to this, we decided to allow the ProjectScanner to use custom Regex Configurations. The objective with custom Regex Configurations is that a power user with some degree of familiarity of how our tool works and some knowledge of regular expressions can craft a custom Regex Configuration that can best represent the reality of the project and the development process, allowing for more tailored monitoring and conclusions that can be directly applied to the project at hand.

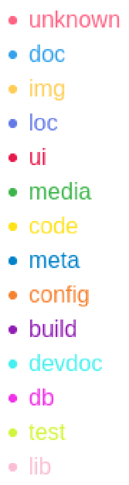


Figure 4.13: Activities (example)

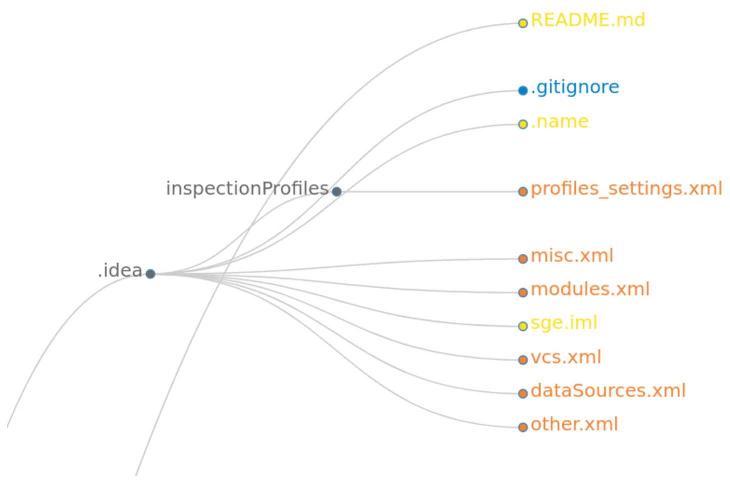


Figure 4.14: Snippet of a D3 Dendogram with color coding related to the Activity Type

4.1.7 Nature of the Effort and Evolution of the Effort

For the next step into the research of features for the ProjectScanner we decided to focus on other information present in the Commit Log. Up until this point the major point of focus within the Commit Log was the full path of the Files that were present with each commit. As such, with the total number of commits and the authors, we can display the evolution of each activity type across the whole timeline of the project. This is also an aspect of similarity with between the ProjectScanner and ActiVCS which can be utilized to compare the actual state of the project to the idealized one, which can be useful during development and after development is complete.

Other points we considered relevant to the usefulness of the ProjectScanner for monitoring, now that we were able to discern development activities from the Commit Log was assigning development roles to the various members of a project and displaying how the various activities weighed in into the overall effort of that specific developer. We named these features **Evolution of the Effort** and **Nature of the Effort** respectively.

We also noticed that we could apply this concept to a project scope, an individual scope and a team scope. This would mean that we could better apply the information the ProjectScanner presents to a wide variety of scenarios. For instance, if a Project is split into different teams, it could be presented as useful to the teams to be able to check the Nature of the Effort and the Evolution of the Effort of the team members. The File Directory Tree could also be applied in this way.

With this in mind, we created the first set of requirements and split them into different categories, this would later be used as a starting point into the Navigation Diagram.

Category	**Scope**	**Features**
Project	Project	Nature of the Effort Evolution of the Effort
Social	Individual Team Members	Nature of the Effort Evolution of the Effort File Directory Tree per Commit
Artifact	Project	File Directory Tree File Directory Tree per Commit

Table 4.1: Categories and Scope of the different features of the ProjectScanner

The following point of focus was in finding the best way to present the Nature of the Effort and the Evolution of the Effort. For this, we decided to use **Chart.js** to display the charts. Chart.js is a chart library which provides simple and flexible charting for developers [6]. Chart.js provides a vast amount of templates for many chart types which are easily customized to fit the needs of developers. The templates we used with the ProjectScanner was the Line Chart and Stacked Bar Chart for the Evolution of the Effort, for the Nature of the Effort we used the Pie Chart Template. The final result applied to our data will be presented in Chapter 7.

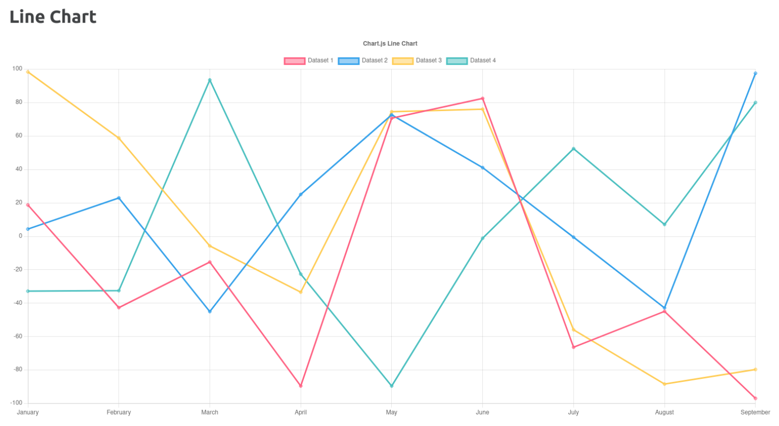


Figure 4.15: Template of the Line Chart (extracted from [6])

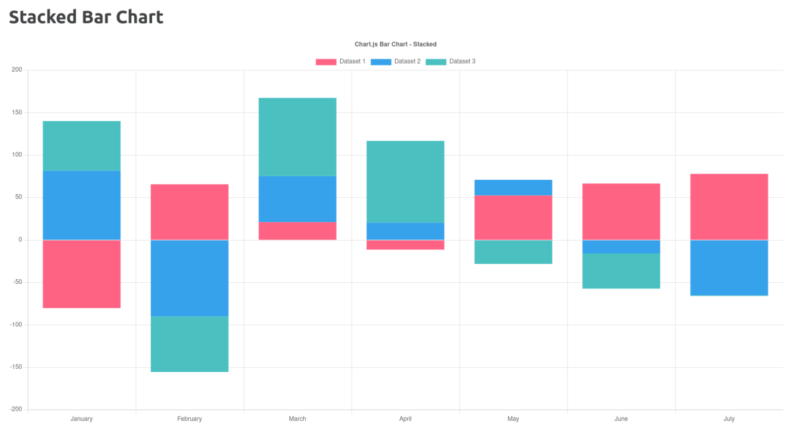


Figure 4.16: Template of the Stacked Bar Chart (extracted from [6])

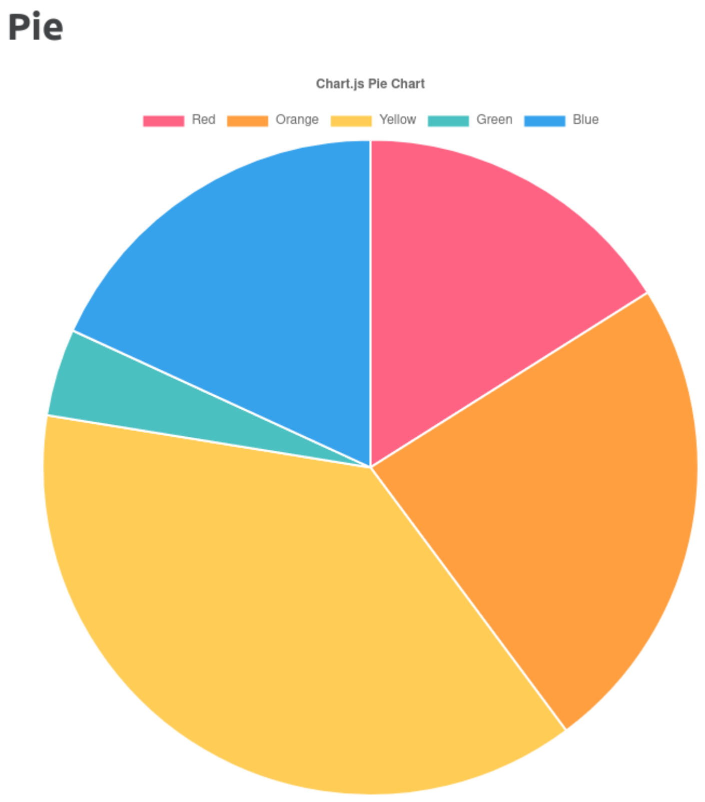


Figure 4.17: Template of the Pie Chart (extracted from [6])

4.1.8 Files

The final step into the research of the ProjectScanner focused on the files present in the commit. Our ideas for the final features centered around aspects of the Project that we considered useful to have ease of access to and would provide helpful information to our monitoring tool.

Activities of the Files

The ProjectScanner will feature a full list of the files of the repository sorted by the development activities considered by the selected Regex. We found this useful as this makes it easier to search for a specific file.

Most Modified Files

ProjectScanner will display a list of all the files in descending order. To accomplish this, we introduced the concept of **Modification**.

A Modification is simply the number of times a file was changed by a commit, we considered initially using the Lines Log to get the number of added/removed lines. However, as we mentioned in 4.1.5, this number does not necessarily correlate with the actual effort that is being applied to that file. Using the Lines Log with the File Directory Tree makes sense from a perspective of scope. The File Directory Tree displays a view per commit and as such, having the number of lines added/removed is a much better indicator of what files are being most worked on at that specific commit. Most Modified Files is centered around a project scope and represents every change that occurred throughout the timeline of the project.

Due to this, we considered Modifications to better accurately present the effort that went into specific files by correlating this effort with the number of commits that changed the file. Most Modified Files can be used to better understand which files had a critical role in development and eventually establish certain critical points and faults in the project.

Modified Files by the Most People

This feature is similar to the Most Modified Files but the files are listed in descending order according to the number of people that were involved in the change of certain file. We consider this feature to have similar applications to the Most Modified Files.

View File

We considered important to allow users of the ProjectScanner to view information specific to each file without having to consult other sources. While Software Repositories such as GitLab and GitHub offer information such as the number of contributors and date when the changes occured to an individual file, we decided to include some of this information in the ProjectScanner as well in order to centralize all the information.

View File will contain the author of the file, an history of the added/removed lines across time using the Line Chart template that Chart.js provides and the date and author of each modification.

Modification History

Modification History will focus on the number of Modifications of a specific file across time, with an indication of the author of the modifications. This will be displayed using the Stacked Bar Chart template that Chart.js provides.

Manipulated Files

For this final feature, we decided to allow the users to consult the actual files that an individual user modified. For this, we will use the Line Chart Template provided by Chart.js. It will indicate the number of modifications across time with a tool tip that will display the files involved.

After this step, we updated Table 4.1 to reflect these changes and help with shaping the full requirement set and Navigation Diagram in the future.

Category	Scope	Features
Project	Project	Nature of the Effort Evolution of the Effort
Social	Individual Team Members	Nature of the Effort Evolution of the Effort File Directory Tree per Commit Manipulated Files
Artifact	Project	File Directory Tree File Directory Tree per Commit Activities of the Files Most Modified Files Modified Files by the Most People View File Modification History

Table 4.2: Categories and Scope of the different features of the ProjectScanner

4.2 Process Mining

4.2.1 Process Mining in Software Development

One of the first steps that were necessary in order to understand how to develop a process mining software was to understand exactly what process mining entailed and how it could be directly applied to Software Development.

Process mining represents one of the multiple areas in the field of process management and is focused on the discovery, monitoring and improvement of current processes by analyzing event logs of the development of a project [5]. The objective is in extracting the process models used by teams when developing. These process models are built using datasets that are constructed with the event logs of the systems development teams utilize when developing software.

Using these datasets, a plethora of information can be retrieved and analyzed using process mining techniques in order to enhance efficiency, identify trends, patterns and discover problems in workflow [5]. This can be extremely hard to track and correct using raw data, especially the more complex the software and the respective team are but also due to eventual bias in the engineers reviewing these processes.

Process mining delivers crucial information for development teams by focusing on practical results that are a mirror of what is happening during development. These results can be directly applied to the team's processes, improving or reworking them and leading to higher quality software.

Process Mining Phases

The task of mining processes can be divided by phases. While these are not linear for all cases there's a set structure that applies and it can be condensed into three vital phases as follows [22][9].

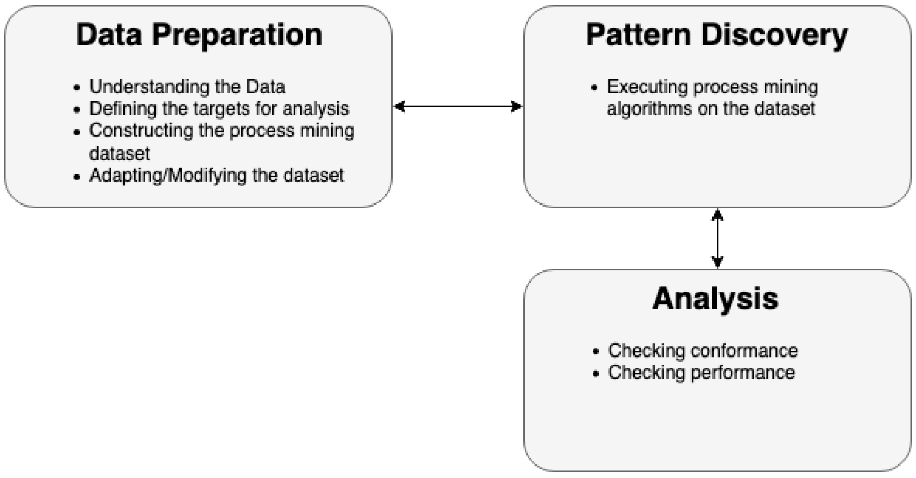
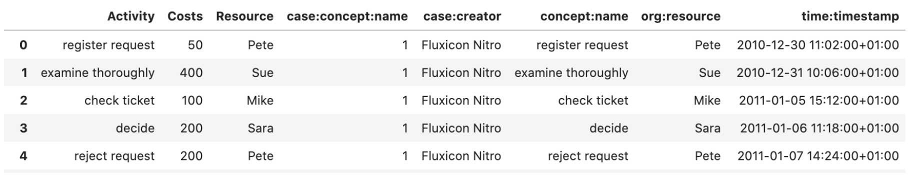

Data Preparation
• Understanding the Data
• Defining the targets for analysis
• Constructing the process mining dataset
• Adapting/Modifying the dataset
Pattern Discovery
• Executing process mining algorithms on the dataset
Analysis
• Checking conformance
• Checking performance

Figure 4.18: Process Mining Phases (adapted from [22])

4.2.2 Data Preparation

Data Preparation can be described in simpler terms as the learning period of Process Mining. The main goal is firstly, understanding what kind of data is necessary for analysis and what it represents within the context of the process mining. For instance, if the goal with a certain mining process is identifying the processes behind *pull requests* for a team of developers, the first step would be in identifying what set of data would be relevant for this information and how to obtain it. A possible solution for Data Preparation would begin with extracting event logs from the team's repository of *pull requests*, *commits* and *issues* in order to build the process models in the next phase.

After defining the and extracting the data it is necessary to arrange it into an appropriate dataset. This usually means parsing the event logs into a format that will be accepted by the process mining algorithms that will be applied to the dataset.



	Activity	Costs	Resource	case:concept:name	case:creator	concept:name	org:resource	time:timestamp
0	register request	50	Pete	1	Fluxicon Nitro	register request	Pete	2010-12-30 11:02:00+01:00
1	examine thoroughly	400	Sue	1	Fluxicon Nitro	examine thoroughly	Sue	2010-12-31 10:06:00+01:00
2	check ticket	100	Mike	1	Fluxicon Nitro	check ticket	Mike	2011-01-05 15:12:00+01:00
3	decide	200	Sara	1	Fluxicon Nitro	decide	Sara	2011-01-06 11:18:00+01:00
4	reject request	200	Pete	1	Fluxicon Nitro	reject request	Pete	2011-01-07 14:24:00+01:00

Figure 4.19: Dataset Example (extracted from [21])

It is common that the Data Preparation phase is revisited multiple times, as it is unlikely that the produced dataset is complete within the first iteration and with the results of upcoming phases there can be a need to return to the Data Preparation phase to re-evaluate.

4.2.3 Process Discovery

Process Discovery is the phase where process models are extracted using the dataset in *Data Preparation* by applying it to process mining algorithms.

There exists a wide variety of process mining algorithms available for building process models, these techniques mostly focus on building process models using different kinds of graphs such as Petri Nets. The algorithms won't be mentioned in this section, however, they will be briefly touched upon in an upcoming section of this report on the tool that the project will be using for applying the algorithms.

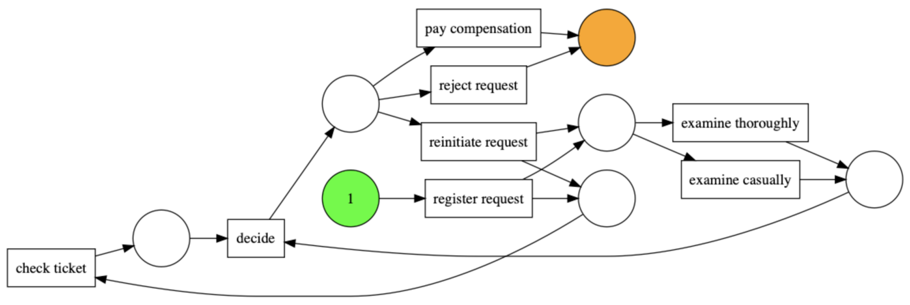


Figure 4.20: Example Process Model with a Petri Net (extracted from [21])

Like the Data Preparation phase, Pattern Discovery can be revisited multiple times based on the results and analysis of the datasets in order to identify the processes that best reflect the reality that is being modelled.

4.2.4 Analysis

The Analysis phase is the culmination of this first step in process mining. The process models and results obtained in *Pattern Discovery* are analyzed. This step usually includes checking for conformance, meaning establishing and analyzing the differences between the process models generated and what is actually happening in reality and performance checking, focusing on detecting bottlenecks, referring to timestamps and other relevant information between events in process models in order to draw conclusions on execution and eventual deviations from the models obtained [21].

As with the above phases, the Analysis phase may also need to be repeated and improved according to the results and analysis gathered.

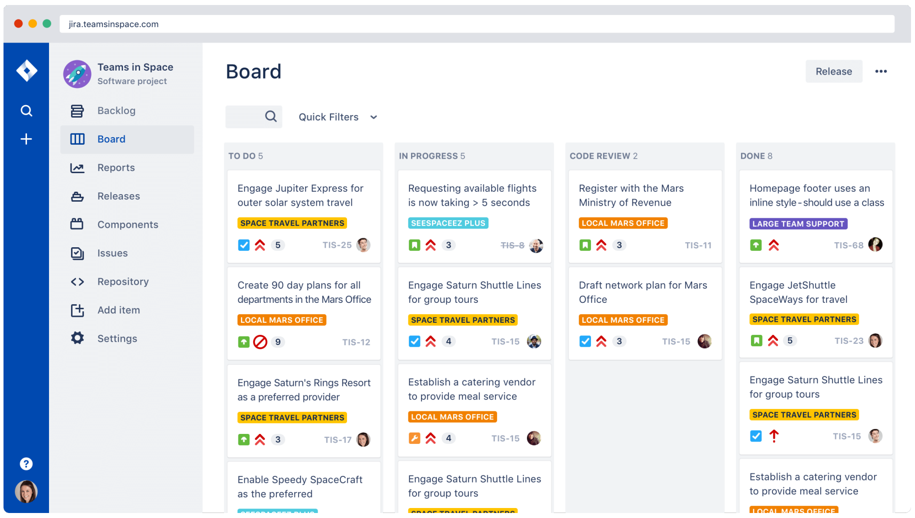
4.2.5 JIRA

JIRA is an issue tracking and project management product developed by Atlassian, it initially started out as a bug and issue tracker but evolved into a work management tool usable for a variety of cases, namely, agile teams, project management teams, software development teams and others [2].

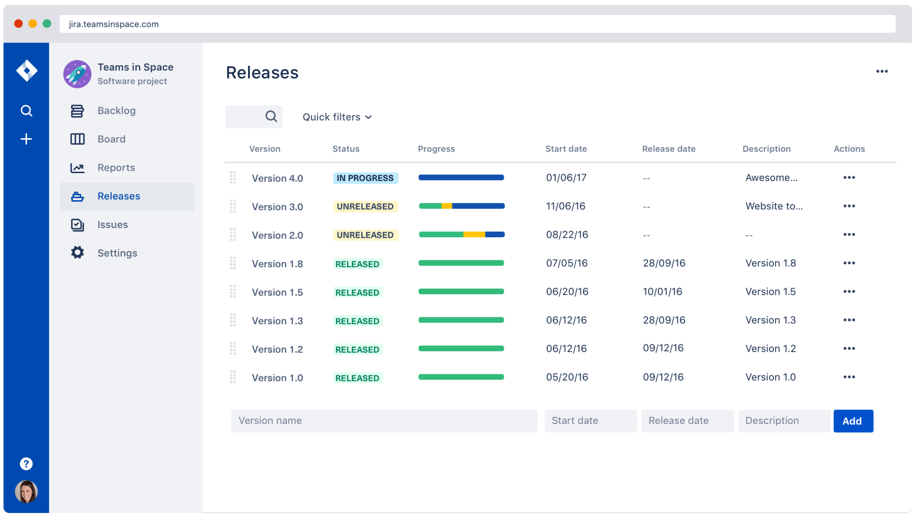
JIRA essentially functions similarly to other very popular tools such as *Trello* or *Wrike*, allowing users to track their projects, assign different tasks to each team member, prioritize, filter tasks, open issues and offers integration with other Atlassian products such *Bitbucket* and *Confluence*.

4.2.6 Webhooks

Webhooks (also commonly referred to as "Reverse APIs" or "web callbacks") are lightweight solutions to fetch real-time data. Unlike a standard API request implementation, using webhooks avoids having to constantly request the API information by utilizing a "triggering system" meaning that one can get real-time data every time something happens (with APIs, the request needs to be handled by the user making it much harder to get



TO DO 5	IN PROGRESS 5	CODE REVIEW 2	DONE 8
Engage Jupiter Express for outer solar system travel · SPACE TRAVEL PARTNERS · 5 · TIS-25	Requesting available flights is now taking > 5 seconds · SEESPACEEZ PLUS · 3 · ~~TIS-8~~	Register with the Mars Ministry of Revenue · LOCAL MARS OFFICE · 3 · TIS-11	Homepage footer uses an inline style - should use a class · LARGE TEAM SUPPORT · TIS-68
Create 90 day plans for all departments in the Mars Office · LOCAL MARS OFFICE · 9 · TIS-12	Engage Saturn Shuttle Lines for group tours · SPACE TRAVEL PARTNERS · 4 · TIS-15	Draft network plan for Mars Office · LOCAL MARS OFFICE · 3 · TIS-15	Engage JetShuttle SpaceWays for travel · SPACE TRAVEL PARTNERS · 5 · TIS-23
Engage Saturn's Rings Resort as a preferred provider · SPACE TRAVEL PARTNERS · 3 · TIS-17	Establish a catering vendor to provide meal service · LOCAL MARS OFFICE · 4 · TIS-15		Engage Saturn Shuttle Lines for group tours · SPACE TRAVEL PARTNERS · TIS-15
Enable Speedy SpaceCraft as the preferred	Engage Saturn Shuttle Lines for group tours		Establish a catering vendor to provide meal service



Version	Status	Progress	Start date	Release date	Description	Actions
Version 4.0	IN PROGRESS		01/06/17	--	Awesome...	•••
Version 3.0	UNRELEASED		11/06/16	--	Website to...	•••
Version 2.0	UNRELEASED		08/22/16	--	--	•••
Version 1.8	RELEASED		07/05/16	28/09/16	Version 1.8	•••
Version 1.5	RELEASED		06/20/16	10/01/16	Version 1.5	•••
Version 1.3	RELEASED		06/12/16	28/09/16	Version 1.3	•••
Version 1.2	RELEASED		06/12/16	09/12/16	Version 1.2	•••
Version 1.0	RELEASED		05/20/16	09/12/16	Version 1.0	•••

Figure 4.21: JIRA Board and Plan Tracker extracted from [17]

real-time information). Using JIRA as an example, a webhook can be customized to alert an application whenever an issue is updated or when a sprint is started [1].

Webhooks use HTTP POST requests and the format is usually JSON or XML, making them easily integrated into web services without the need of adding new software components [20].

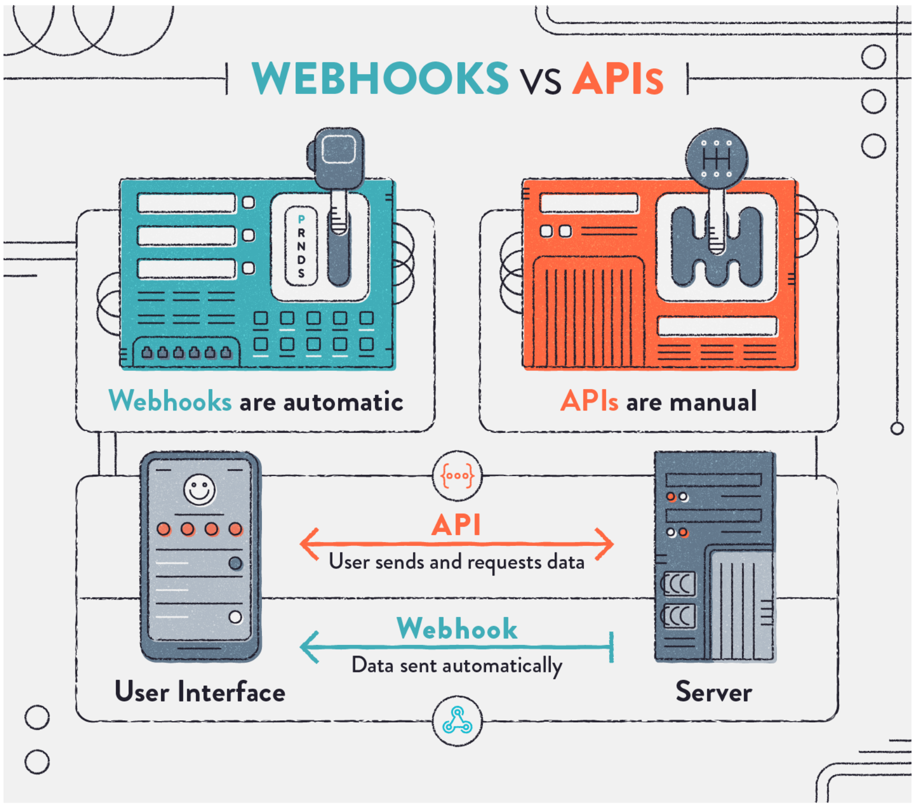


Figure 4.22: Webhooks vs APIs extracted from [18]

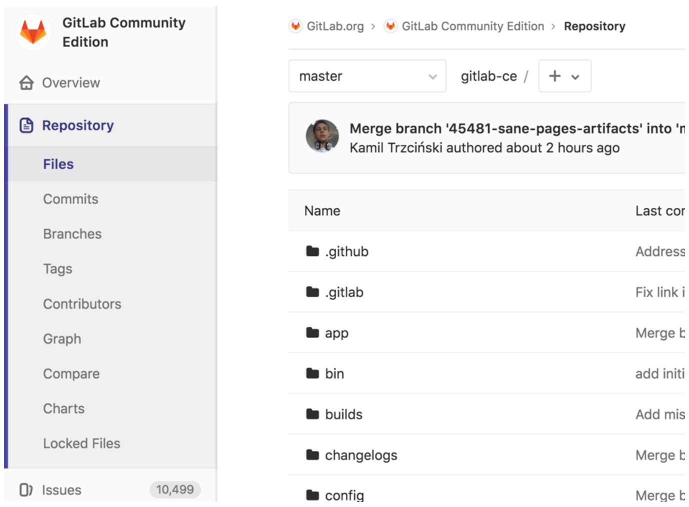
Within the scope of the project in mind, webhooks are an incredibly reliable option as the need to constantly monitor if events are happening in order to get real-time data from the API is removed (a very important subject to have in mind with process mining) and a much more customized approach can be developed. Multiple webhooks can be configured to listen only to certain events, opening up multiple development options and a more versatile way to fragment information and split it into small portions.

4.3 GitLab

GitLab

Figure 4.23: The Gitlab Logo extracted from [32]

GitLab is an open-source web-based DevOps platform created by Dmitriy Zaporozhets and Valery Sizov [32]. GitLab streamlines software workflow into a single application, helping teams reducing product lifecycles and boosting productivity by providing a way for teams to perform all tasks in the DevOps lifecycle within one single platform [14].

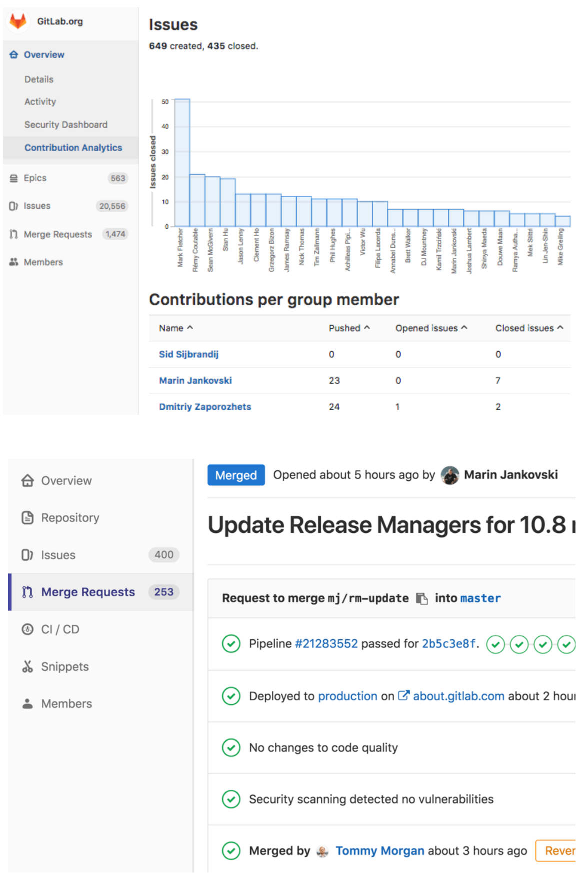


Merge branch '45481-sane-pages-artifacts' into 'n
Kamil Trzciński authored about 2 hours ago

Name	Last cor
.github	Address
.gitlab	Fix link i
app	Merge b
bin	add initi
builds	Add mis
changelogs	Merge b
config	Merge b

Figure 4.24: GitLab Repository Interface extracted from [14]

GitLab features Git-repositories with a variety of capabilities such as Source Code Management (issue-tracking, version control), Code Reviews and built-in wiki for projects. It also helps manage projects by providing built-in metric tools to boost deliveries, verifying quality with Continuous Integration methodologies (performing a variety of automated tests in order to verify *commits* before these are pushed to a live version of a project), security and monitoring tools in order to ensure the quality of the produced software [14].



Issues

649 created, 435 closed.

Contributions per group member

Name ^	Pushed ^	Opened issues ^	Closed issues ^
Sid Sijbrandij	0	0	0
Marin Jankovski	23	0	7
Dmitriy Zaporozhets	24	1	2

Merged Opened about 5 hours ago by Marin Jankovski

Update Release Managers for 10.8

Request to merge `mj/rm-update` into `master`

Pipeline #21283552 passed for `2b5c3e8f`.

Deployed to production on about.gitlab.com about 2 hou

No changes to code quality

Security scanning detected no vulnerabilities

Merged by Tommy Morgan about 3 hours ago

Figure 4.25: Different GitLab Interfaces extracted from [14]

Our next step into the research would fall into the study and sample demonstrations of the GitLab Webhook, API and what information could be drawn from these.

4.3.1 GitLab Webhook Demonstration

For the demonstration of a Webhook directly applied to GitLab, the GitLab Webhook Documentation was consulted (available here: `https://docs.gitlab.com/ee/user/project/integrations/webhooks.html`).

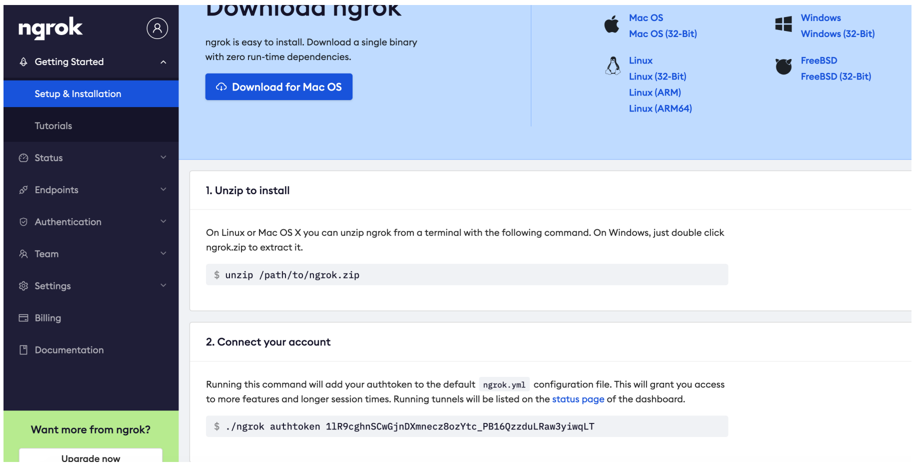
To provide a better visual presentation and enable the testing of the Webhook, the tool *Ngrok* was used. Ngrok provides an easy way to test Webhook implementations by exposing local servers to the public internet via secure tunneling [24], meaning one can implement and test Webhooks in a web service even when working *localhost*.

Setting Up Ngrok

The first step is creating an account in the Ngrok website via `https://dashboard.ngrok.com/login`, this a necessary step in order to connect the Ngrok account to the web service using the *authtoken* of the account.

After the creation of the account, login into the dashboard of *Ngrok* to retrieve the *authtoken*.

Next, we downloaded Ngrok from `https://ngrok.com/download` and unzip the file. The following commands are then used to authenticate the account and start Ngrok.



Download ngrok

ngrok is easy to install. Download a single binary with zero run-time dependencies.

1. Unzip to install

On Linux or Mac OS X you can unzip ngrok from a terminal with the following command. On Windows, just double click ngrok.zip to extract it.

```
$ unzip /path/to/ngrok.zip
```

2. Connect your account

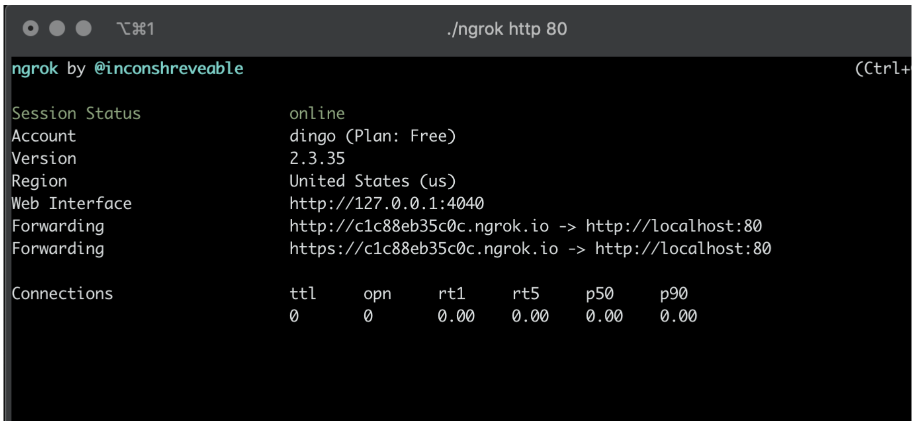
Running this command will add your authtoken to the default `ngrok.yml` configuration file. This will grant you access to more features and longer session times. Running tunnels will be listed on the status page of the dashboard.

```
$ ./ngrok authtoken 1lR9cghnSCwGjnDXmnecz8ozYtc_PB16QzzduLRaw3yiwqLT
```

Figure 4.26: Location of the *authtoken*

```
./ngrok authtoken [your_auth_token_here]
./ngrok http [port]
```

After running these commands, the output should be like the one in the following figure.



```
ngrok by @inconshreveable                                                    (Ctrl+

Session Status                online
Account                       dingo (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://c1c88eb35c0c.ngrok.io -> http://localhost:80
Forwarding                    https://c1c88eb35c0c.ngrok.io -> http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Figure 4.27: Ngrok Output

After this step, the setting up of Ngrok is complete. Looking at the output there's two noticeable lines that are important to understand.

- **Web Interface**: Acessing the URL displays the Ngrok Web Interface allowing for a better presentation of the contents of the Webhook;
- **Forwarding**: Ngrok randomly generates a tunneling URL for the web service. This URL will be used to configure the GitLab Webhook.

Configuring the Webhook

Before configuring the webhook, a webhook receiver is needed in order to fetch the webhook request. For demonstration purposes, a simple webhook receiver written in *Ruby* was used directly from the GitLab Webhook documentation shown below.

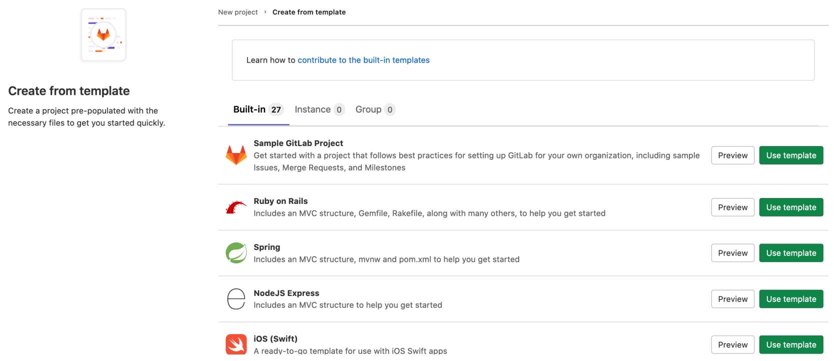
```
require 'webrick'
```

```
server = WEBrick::HTTPServer.new(:Port => ARGV.first)
server.mount_proc '/' do |req, res|
  puts req.body
end

trap 'INT' do
  server.shutdown
end
server.start
```

This code can be run with the following line, noting that the port for the service must be the same as the one used to run Ngrok in the previous setup.

```
ruby print_http_body.rb [port]
```

GitLab comes into play after running the ruby code. An account is needed to use GitLab and can be created here: `https://gitlab.com/users/sign_up`. For this demonstration, a project template for GitLab was selected, this project template comes already populated with sample data and is extremely useful for testing Webhooks.


New project › Create from template

Learn how to contribute to the built-in templates

Create from template

Create a project pre-populated with the necessary files to get you started quickly.

Built-in 27 | Instance 0 | Group 0

Sample GitLab Project
Get started with a project that follows best practices for setting up GitLab for your own organization, including sample Issues, Merge Requests, and Milestones
Preview | Use template

Ruby on Rails
Includes an MVC structure, Gemfile, Rakefile, along with many others, to help you get started
Preview | Use template

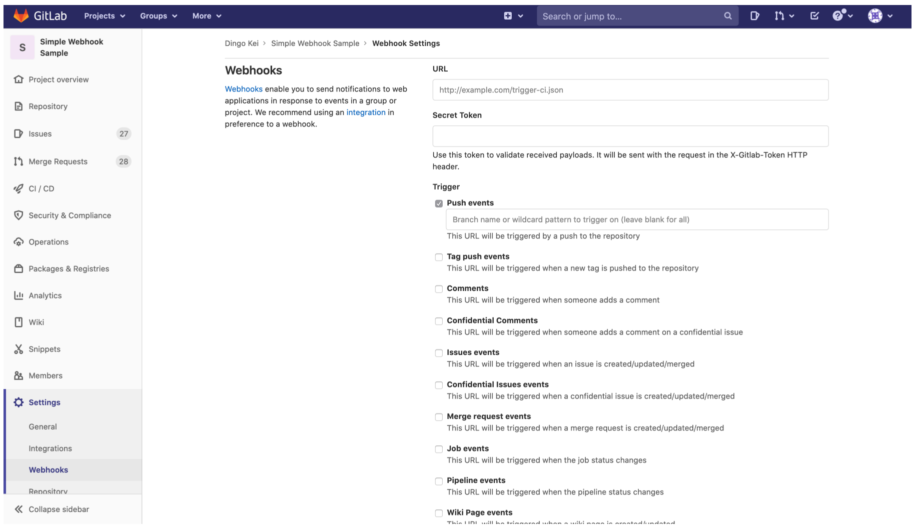
Spring
Includes an MVC structure, mvnw and pom.xml to help you get started
Preview | Use template

NodeJS Express
Includes an MVC structure to help you get started
Preview | Use template

iOS (Swift)
A ready-to-go template for use with iOS Swift apps
Preview | Use template

Figure 4.28: Available GitLab template repositories

After selecting the project, the Webhook setting should be found in **Settings -> Webhooks**.



Dingo Kei › Simple Webhook Sample › Webhook Settings

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

http://example.com/trigger-ci.json

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

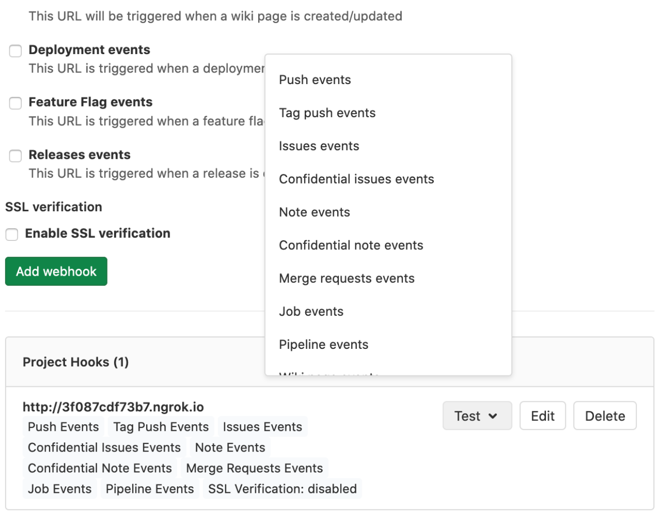
- [✓] **Push events**
  Branch name or wildcard pattern to trigger on (leave blank for all)
  This URL will be triggered by a push to the repository
- [ ] **Tag push events**
  This URL will be triggered when a new tag is pushed to the repository
- [ ] **Comments**
  This URL will be triggered when someone adds a comment
- [ ] **Confidential Comments**
  This URL will be triggered when someone adds a comment on a confidential issue
- [ ] **Issues events**
  This URL will be triggered when an issue is created/updated/merged
- [ ] **Confidential Issues events**
  This URL will be triggered when a confidential issue is created/updated/merged
- [ ] **Merge request events**
  This URL will be triggered when a merge request is created/updated/merged
- [ ] **Job events**
  This URL will be triggered when the job status changes
- [ ] **Pipeline events**
  This URL will be triggered when the pipeline status changes
- [ ] **Wiki Page events**

Figure 4.29: Webhook configuration

The page should be similar to the one in the above figure.

- **URL:** The URL to which the webhook should send the information to. The *Forwarding* URL Ngrok generated is going to be placed here. However, since Ngrok randomly generates an URL each time it is executed, old URLs from previous executions won't be valid;
- **Secret Token:** This field is not necessary for the demonstration;
- **Triggers:** The triggers are the events to which the Webhook will respond and send data everytime they occur. Multiple of these can be selected or multiple webhooks can be configured to only respond to specific triggers.

After configuring the webhook, clicking the **Add webhook** button which is not displayed in the figure will create a new webhook with the specified configuration.



This URL will be triggered when a wiki page is created/updated

- [ ] **Deployment events**
  This URL is triggered when a deploymer
- [ ] **Feature Flag events**
  This URL is triggered when a feature fla
- [ ] **Releases events**
  This URL is triggered when a release is

SSL verification

- [ ] **Enable SSL verification**

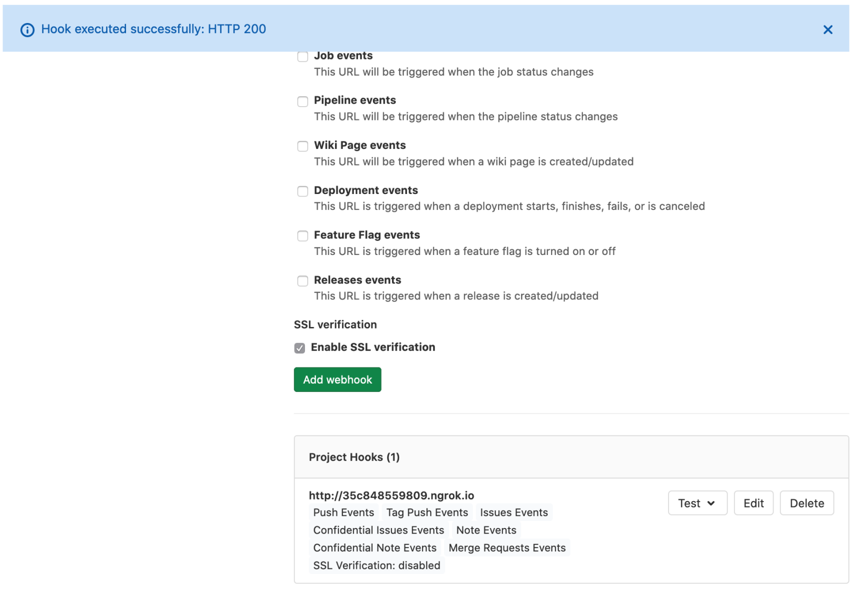
Project Hooks (1)

http://3f087cdf73b7.ngrok.io
Push Events Tag Push Events Issues Events
Confidential Issues Events Note Events
Confidential Note Events Merge Requests Events
Job Events Pipeline Events SSL Verification: disabled

Figure 4.30: Created webhook and Test events

The webhook is now created, GitLab provides an extremely useful *Test* feature to the webhook allowing the firing of a sample trigger to verify if the configuration is working.

Manually executing an event within the template repository is also possible if that event is part of a trigger the webhook is listening to.



Hook executed successfully: HTTP 200

Job events
This URL will be triggered when the job status changes

Pipeline events
This URL will be triggered when the pipeline status changes

Wiki Page events
This URL will be triggered when a wiki page is created/updated

Deployment events
This URL is triggered when a deployment starts, finishes, fails, or is canceled

Feature Flag events
This URL is triggered when a feature flag is turned on or off

Releases events
This URL is triggered when a release is created/updated

SSL verification
Enable SSL verification

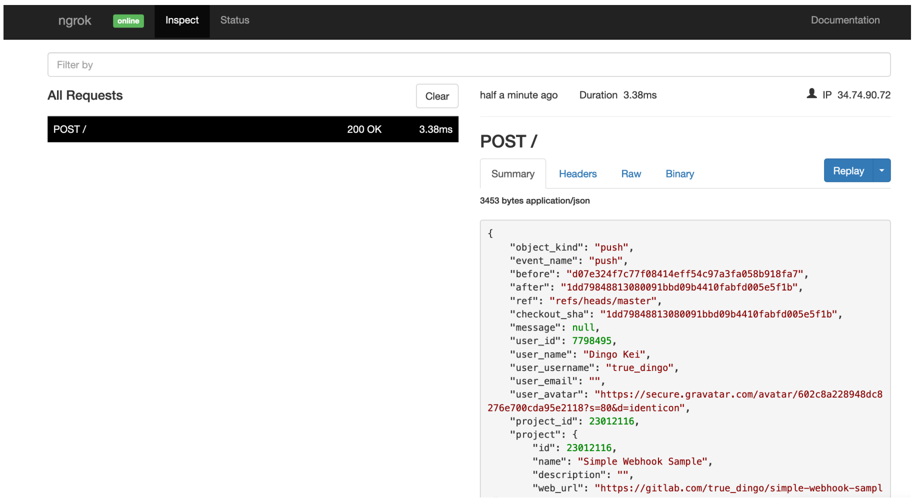
Add webhook

Project Hooks (1)

http://35c848559809.ngrok.io
Push Events Tag Push Events Issues Events
Confidential Issues Events Note Events
Confidential Note Events Merge Requests Events
SSL Verification: disabled

Figure 4.31: Successful webhook output

GitLab should return an HTTP 200 message on a successfully completed webhook. The next step is checking the Ngrok Web Interface for the request.



POST / 200 OK 3.38ms

half a minute ago Duration 3.38ms IP 34.74.90.72

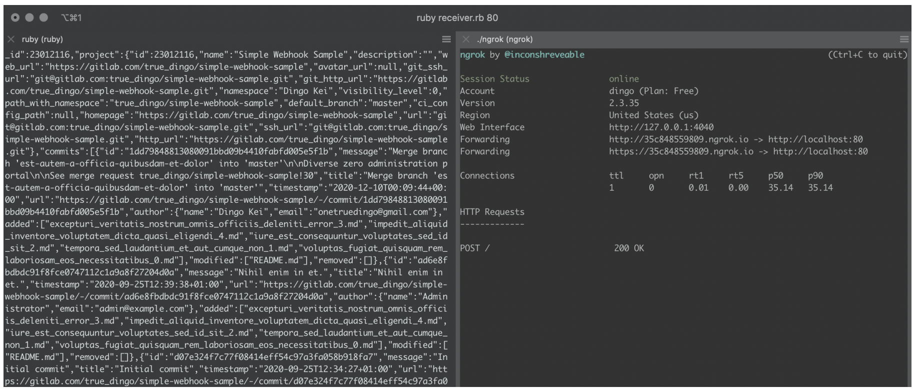
POST /

3453 bytes application/json

```
{
    "object_kind": "push",
    "event_name": "push",
    "before": "d07e324f7c77f08414eff54c97a3fa058b918fa7",
    "after": "1dd79848813080091bbd09b4410fabfd005e5f1b",
    "ref": "refs/heads/master",
    "checkout_sha": "1dd79848813080091bbd09b4410fabfd005e5f1b",
    "message": null,
    "user_id": 7798495,
    "user_name": "Dingo Kei",
    "user_username": "true_dingo",
    "user_email": "",
    "user_avatar": "https://secure.gravatar.com/avatar/602c8a228948dc8
276e700cda95e2118?s=80&d=identicon",
    "project_id": 23012116,
    "project": {
        "id": 23012116,
        "name": "Simple Webhook Sample",
        "description": "",
        "web_url": "https://gitlab.com/true_dingo/simple-webhook-sampl
```

Figure 4.32: Ngrok Web Interface with the available request

The request was executed successfully and a preview of the contents of the request should be available within the Ngrok Web Interface. In the example above, a push event trigger was fired and the result is the JSON data referring to the information of the dummy push that was triggered.



```
_id":23012116,"project":{"id":23012116,"name":"Simple Webhook Sample","description":"","w
eb_url":"https://gitlab.com/true_dingo/simple-webhook-sample","avatar_url":null,"git_ssh_
url":"git@gitlab.com:true_dingo/simple-webhook-sample.git","git_http_url":"https://gitlab
.com/true_dingo/simple-webhook-sample.git","namespace":"Dingo Kei","visibility_level":0,"
path_with_namespace":"true_dingo/simple-webhook-sample","default_branch":"master","ci_con
fig_path":null,"homepage":"https://gitlab.com/true_dingo/simple-webhook-sample","url":"gi
t@gitlab.com:true_dingo/simple-webhook-sample.git","ssh_url":"git@gitlab.com:true_dingo/s
imple-webhook-sample.git","http_url":"https://gitlab.com/true_dingo/simple-webhook-sample
.git"},"commits":[{"id":"1dd79848813080091bbd09b4410fabfd005e5f1b","message":"Merge branc
h 'est-autem-a-officia-quibusdam-et-dolor' into 'master'\n\nDiverse zero administration p
ortal\n\nSee merge request true_dingo/simple-webhook-sample!30","title":"Merge branch 'es
t-autem-a-officia-quibusdam-et-dolor' into 'master'","timestamp":"2020-12-10T00:09:44+00:
00","url":"https://gitlab.com/true_dingo/simple-webhook-sample/-/commit/1dd79848813080091
bbd09b4410fabfd005e5f1b","author":{"name":"Dingo Kei","email":"onetruedingo@gmail.com"},"
added":["excepturi_veritatis_nostrum_omnis_officiis_deleniti_error_3.md","impedit_aliquid
_inventore_voluptatem_dicta_quasi_eligendi_4.md","iure_est_consequuntur_voluptates_sed_id
_sit_2.md","tempora_sed_laudantium_et_aut_cumque_non_1.md","voluptas_fugiat_quisquam_rem_
laboriosam_eos_necessitatibus_0.md"],"modified":["README.md"],"removed":[]},{"id":"ad6e8f
bdbdc91f8fce0747112c1a9a8f27204d0a","message":"Nihil enim in et.","title":"Nihil enim in
et.","timestamp":"2020-09-25T12:39:38+01:00","url":"https://gitlab.com/true_dingo/simple-
webhook-sample/-/commit/ad6e8fbdbdc91f8fce0747112c1a9a8f27204d0a","author":{"name":"Admin
istrator","email":"admin@example.com"},"added":["excepturi_veritatis_nostrum_omnis_offici
is_deleniti_error_3.md","impedit_aliquid_inventore_voluptatem_dicta_quasi_eligendi_4.md",
"iure_est_consequuntur_voluptates_sed_id_sit_2.md","tempora_sed_laudantium_et_aut_cumque_
non_1.md","voluptas_fugiat_quisquam_rem_laboriosam_eos_necessitatibus_0.md"],"modified":[
"README.md"],"removed":[]},{"id":"d07e324f7c77f08414eff54c97a3fa058b918fa7","message":"In
itial commit","title":"Initial commit","timestamp":"2020-09-25T12:34:27+01:00","url":"htt
ps://gitlab.com/true_dingo/simple-webhook-sample/-/commit/d07e324f7c77f08414eff54c97a3fa0
```

```
ngrok by @inconshreveable                                          (Ctrl+C to quit)

Session Status                online
Account                       dingo (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://35c848559809.ngrok.io -> http://localhost:80
Forwarding                    https://35c848559809.ngrok.io -> http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              1       0       0.01    0.00    35.14   35.14

HTTP Requests
-------------

POST /                         200 OK
```

Figure 4.33: Output of the Web Receiver

The webhook receiver also contains the request data however it is far less readable.

GitLab Webhook usage in the GitLab Application

With this demonstration the ease of use and versatility of webhooks is directly demonstrated using GitLab. Webhooks are very useful as gateway to getting real time information without using more heavy processes to achieve the same.

However, the GitLab Webhook documentation specifies some constraints in the usage of Webhooks such as a limit to how many commits can be sent with a push event. These constraints apply to some of the triggers available and should be taken into account. This is not expected to be aproblem, as we are using the webhook as an event trigger, followed by an API call to access more detailed data. The different kinds of information that can be retrieved is also vastly different upon simple glance into the GitLab API Documentation, making a study into the API necessary.

The demonstration and research into the GitLab Webhook solidifies its place in the implementation when expanding the ProjectScanner.

While useful, webhooks are not enough to fetch all possible relevant data but its importance is still significant, acting as a simple approach to accurately signal whenever new information is available and retrieve it easily.

4.3.2 GitLab API Demonstration

After establishing that an implementation based on webhooks alone would not have the detail required to perform in-depth mining of software processes due to constraints pointed out in the documentation and the fact that GitLab API offered a plethora of extra depth with its requests, the objective with the study was to understand how the API worked, what resources were available from it and which of these would be considered relevant for future work into the *Data Preparation* phase of the GitLab application.

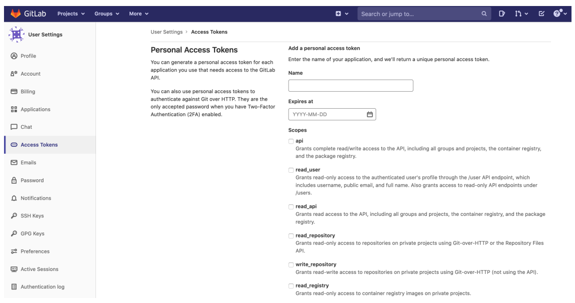
All information present in this subsection is based on the GitLab API Docs (`https://docs.gitlab.com/ee/api/`).

Making an API Request

Every request made to the API begins with the base URL endpoint for the request, this URL will serve as the base of communications featuring the API.

`https://gitlab.com/api/v4/projects`

To fetch specific data, all that needs to be done is add the specific resource to the endpoint. According to the request that is specified and the constructed endpoint, authentication may be required to perform that certain request. There are many authentication methods made available by the API but the simplest one that will be used to provide request examples will be the *personal access token*, this token can be retrieved from the User Settings of the user's GitLab account. A project id is also required for the majority of requests, this id can be found in the Settings tab of the specific project in GitLab.



Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

Name

Expires at

YYYY-MM-DD

Scopes

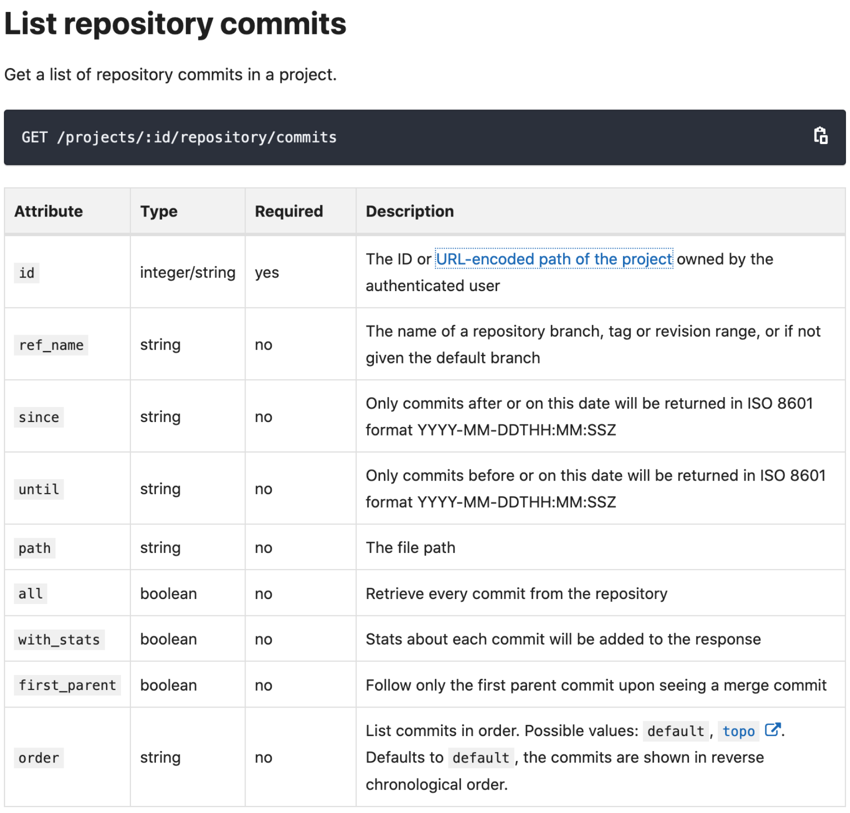
- **api**
  Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- **read_user**
  Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- **read_api**
  Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- **read_repository**
  Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- **write_repository**
  Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
- **read_registry**
  Grants read-only access to container registry images on private projects.

Figure 4.34: GitLab's User Setting for generating access tokens

For the examples of requests, the **Commit** section of the documentation will be followed (`https://docs.gitlab.com/ee/api/commits.html`).

Looking at the above figure, a request to the API that fetches all commits in a project, the endpoint can be constructed with:

`https://gitlab.com/api/v4/projects/[project_id]/repository/commits`



List repository commits

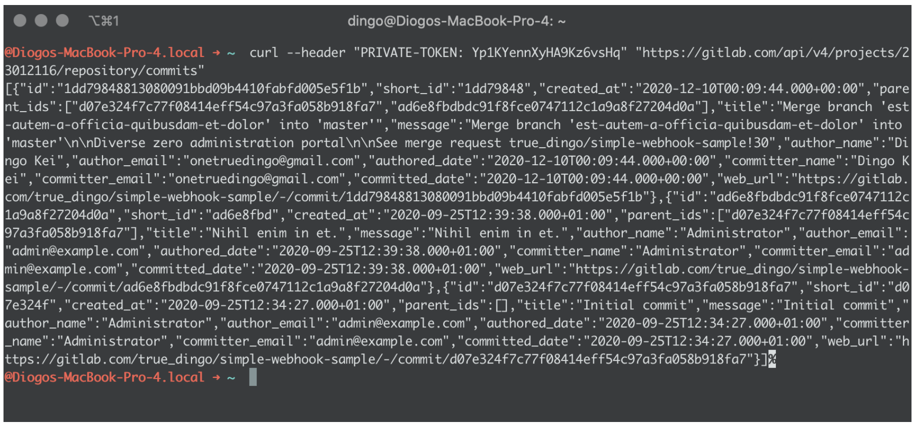
Get a list of repository commits in a project.

```
GET /projects/:id/repository/commits
```

Attribute	Type	Required	Description
`id`	integer/string	yes	The ID or URL-encoded path of the project owned by the authenticated user
`ref_name`	string	no	The name of a repository branch, tag or revision range, or if not given the default branch
`since`	string	no	Only commits after or on this date will be returned in ISO 8601 format YYYY-MM-DDTHH:MM:SSZ
`until`	string	no	Only commits before or on this date will be returned in ISO 8601 format YYYY-MM-DDTHH:MM:SSZ
`path`	string	no	The file path
`all`	boolean	no	Retrieve every commit from the repository
`with_stats`	boolean	no	Stats about each commit will be added to the response
`first_parent`	boolean	no	Follow only the first parent commit upon seeing a merge commit
`order`	string	no	List commits in order. Possible values: `default`, `topo`. Defaults to `default`, the commits are shown in reverse chronological order.

Figure 4.35: Commit Endpoints

The following figure represents the output of this request, the project used was the Template Project that was created for the webhook demonstration.

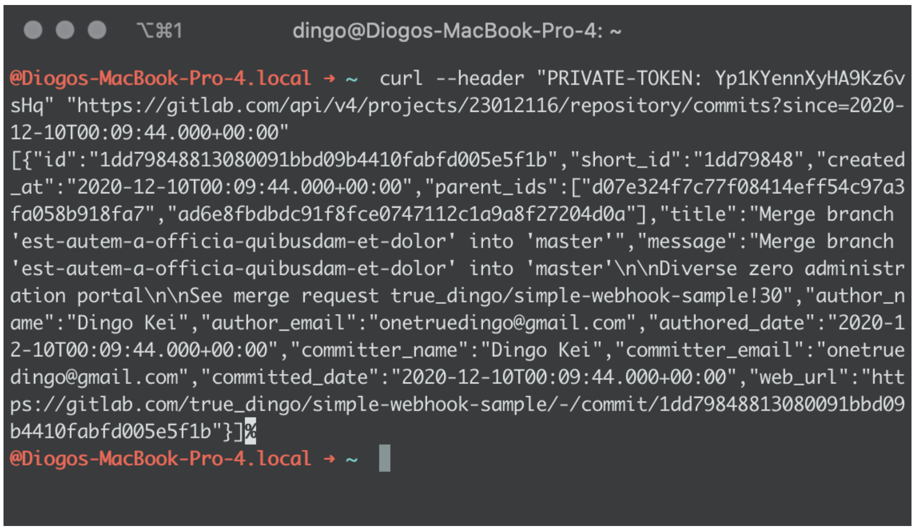


```
dingo@Diogos-MacBook-Pro-4: ~
@Diogos-MacBook-Pro-4.local ➜ ~  curl --header "PRIVATE-TOKEN: Yp1KYennXyHA9Kz6vsHq" "https://gitlab.com/api/v4/projects/2
3012116/repository/commits"
[{"id":"1dd79848813080091bbd09b4410fabfd005e5f1b","short_id":"1dd79848","created_at":"2020-12-10T00:09:44.000+00:00","pare
nt_ids":["d07e324f7c77f08414eff54c97a3fa058b918fa7","ad6e8fbdbdc91f8fce0747112c1a9a8f27204d0a"],"title":"Merge branch 'est
-autem-a-officia-quibusdam-et-dolor' into 'master'","message":"Merge branch 'est-autem-a-officia-quibusdam-et-dolor' into
'master'\n\nDiverse zero administration portal\n\nSee merge request true_dingo/simple-webhook-sample!30","author_name":"Di
ngo Kei","author_email":"onetruedingo@gmail.com","authored_date":"2020-12-10T00:09:44.000+00:00","committer_name":"Dingo K
ei","committer_email":"onetruedingo@gmail.com","committed_date":"2020-12-10T00:09:44.000+00:00","web_url":"https://gitlab.
com/true_dingo/simple-webhook-sample/-/commit/1dd79848813080091bbd09b4410fabfd005e5f1b"},{"id":"ad6e8fbdbdc91f8fce0747112c
1a9a8f27204d0a","short_id":"ad6e8fbd","created_at":"2020-09-25T12:39:38.000+01:00","parent_ids":["d07e324f7c77f08414eff54c
97a3fa058b918fa7"],"title":"Nihil enim in et.","message":"Nihil enim in et.","author_name":"Administrator","author_email":
"admin@example.com","authored_date":"2020-09-25T12:39:38.000+01:00","committer_name":"Administrator","committer_email":"ad
min@example.com","committed_date":"2020-09-25T12:39:38.000+01:00","web_url":"https://gitlab.com/true_dingo/simple-webhook-
sample/-/commit/ad6e8fbdbdc91f8fce0747112c1a9a8f27204d0a"},{"id":"d07e324f7c77f08414eff54c97a3fa058b918fa7","short_id":"d0
7e324f","created_at":"2020-09-25T12:34:27.000+01:00","parent_ids":[],"title":"Initial commit","message":"Initial commit","
author_name":"Administrator","author_email":"admin@example.com","authored_date":"2020-09-25T12:34:27.000+01:00","committer
_name":"Administrator","committer_email":"admin@example.com","committed_date":"2020-09-25T12:34:27.000+01:00","web_url":"h
ttps://gitlab.com/true_dingo/simple-webhook-sample/-/commit/d07e324f7c77f08414eff54c97a3fa058b918fa7"}]%
@Diogos-MacBook-Pro-4.local ➜ ~
```

Figure 4.36: Output for commits in the sample project

Using attributes, it is possible to narrow the request to more specific data. The following example will feature a request to the API that fetches the commits created *since* a specific date, the endpoint can be constructed with:

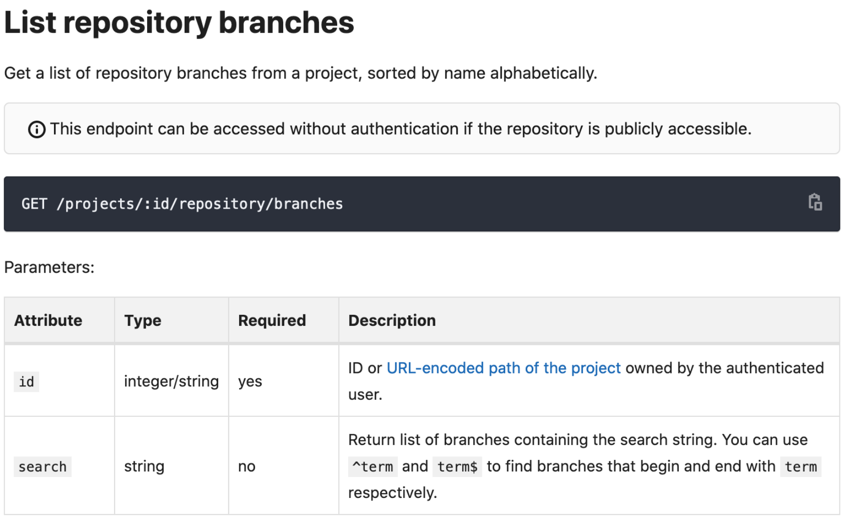
```
https://gitlab.com/api/v4/projects/[project_id]/
/repository/commits?since=YYYY-MM-DDTHH:MM:SSZ
```



```
@Diogos-MacBook-Pro-4.local ➜ ~  curl --header "PRIVATE-TOKEN: Yp1KYennXyHA9Kz6v
sHq" "https://gitlab.com/api/v4/projects/23012116/repository/commits?since=2020-
12-10T00:09:44.000+00:00"
[{"id":"1dd79848813080091bbd09b4410fabfd005e5f1b","short_id":"1dd79848","created
_at":"2020-12-10T00:09:44.000+00:00","parent_ids":["d07e324f7c77f08414eff54c97a3
fa058b918fa7","ad6e8fbdbdc91f8fce0747112c1a9a8f27204d0a"],"title":"Merge branch
'est-autem-a-officia-quibusdam-et-dolor' into 'master'","message":"Merge branch
'est-autem-a-officia-quibusdam-et-dolor' into 'master'\n\nDiverse zero administr
ation portal\n\nSee merge request true_dingo/simple-webhook-sample!30","author_n
ame":"Dingo Kei","author_email":"onetruedingo@gmail.com","authored_date":"2020-1
2-10T00:09:44.000+00:00","committer_name":"Dingo Kei","committer_email":"onetrue
dingo@gmail.com","committed_date":"2020-12-10T00:09:44.000+00:00","web_url":"htt
ps://gitlab.com/true_dingo/simple-webhook-sample/-/commit/1dd79848813080091bbd09
b4410fabfd005e5f1b"}]%
@Diogos-MacBook-Pro-4.local ➜ ~
```

Figure 4.37: Output for API request using the *since* argument

The previous examples featured the usage of the *commit* resource that is also available as a push trigger using the webhook, in order to further establish the need of the GitLab API the last example will feature a resource that is not available using the webhook.



List repository branches

Get a list of repository branches from a project, sorted by name alphabetically.

ⓘ This endpoint can be accessed without authentication if the repository is publicly accessible.

```
GET /projects/:id/repository/branches
```

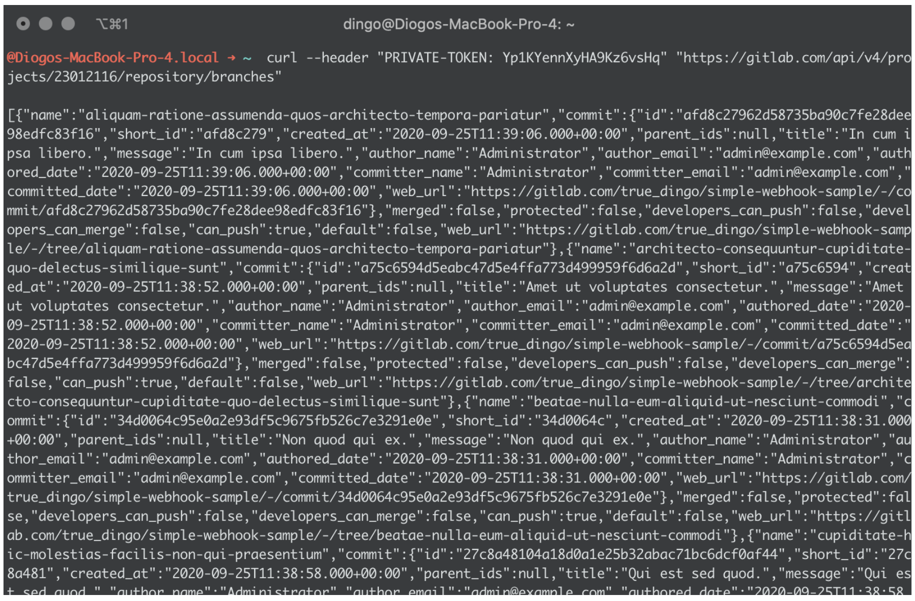
Parameters:

Attribute	Type	Required	Description
`id`	integer/string	yes	ID or URL-encoded path of the project owned by the authenticated user.
`search`	string	no	Return list of branches containing the search string. You can use `^term` and `term$` to find branches that begin and end with `term` respectively.

Figure 4.38: Branch Endpoints

The endpoint can be constructed as:

`https://gitlab.com/api/v4/projects/[project_id]/repository/branches`



```
dingo@Diogos-MacBook-Pro-4: ~
@Diogos-MacBook-Pro-4.local ➜ ~  curl --header "PRIVATE-TOKEN: Yp1KYennXyHA9Kz6vsHq" "https://gitlab.com/api/v4/pro
jects/23012116/repository/branches"

[{"name":"aliquam-ratione-assumenda-quos-architecto-tempora-pariatur","commit":{"id":"afd8c27962d58735ba90c7fe28dee
98edfc83f16","short_id":"afd8c279","created_at":"2020-09-25T11:39:06.000+00:00","parent_ids":null,"title":"In cum i
psa libero.","message":"In cum ipsa libero.","author_name":"Administrator","author_email":"admin@example.com","auth
ored_date":"2020-09-25T11:39:06.000+00:00","committer_name":"Administrator","committer_email":"admin@example.com","
committed_date":"2020-09-25T11:39:06.000+00:00","web_url":"https://gitlab.com/true_dingo/simple-webhook-sample/-/co
mmit/afd8c27962d58735ba90c7fe28dee98edfc83f16"},"merged":false,"protected":false,"developers_can_push":false,"devel
opers_can_merge":false,"can_push":true,"default":false,"web_url":"https://gitlab.com/true_dingo/simple-webhook-samp
le/-/tree/aliquam-ratione-assumenda-quos-architecto-tempora-pariatur"},{"name":"architecto-consequuntur-cupiditate-
quo-delectus-similique-sunt","commit":{"id":"a75c6594d5eabc47d5e4ffa773d499959f6d6a2d","short_id":"a75c6594","creat
ed_at":"2020-09-25T11:38:52.000+00:00","parent_ids":null,"title":"Amet ut voluptates consectetur.","message":"Amet
ut voluptates consectetur.","author_name":"Administrator","author_email":"admin@example.com","authored_date":"2020-
09-25T11:38:52.000+00:00","committer_name":"Administrator","committer_email":"admin@example.com","committed_date":"
2020-09-25T11:38:52.000+00:00","web_url":"https://gitlab.com/true_dingo/simple-webhook-sample/-/commit/a75c6594d5ea
bc47d5e4ffa773d499959f6d6a2d"},"merged":false,"protected":false,"developers_can_push":false,"developers_can_merge":
false,"can_push":true,"default":false,"web_url":"https://gitlab.com/true_dingo/simple-webhook-sample/-/tree/archite
cto-consequuntur-cupiditate-quo-delectus-similique-sunt"},{"name":"beatae-nulla-eum-aliquid-ut-nesciunt-commodi","c
ommit":{"id":"34d0064c95e0a2e93df5c9675fb526c7e3291e0e","short_id":"34d0064c","created_at":"2020-09-25T11:38:31.000
+00:00","parent_ids":null,"title":"Non quod qui ex.","message":"Non quod qui ex.","author_name":"Administrator","au
thor_email":"admin@example.com","authored_date":"2020-09-25T11:38:31.000+00:00","committer_name":"Administrator","c
ommitter_email":"admin@example.com","committed_date":"2020-09-25T11:38:31.000+00:00","web_url":"https://gitlab.com/
true_dingo/simple-webhook-sample/-/commit/34d0064c95e0a2e93df5c9675fb526c7e3291e0e"},"merged":false,"protected":fal
se,"developers_can_push":false,"developers_can_merge":false,"can_push":true,"default":false,"web_url":"https://gitl
ab.com/true_dingo/simple-webhook-sample/-/tree/beatae-nulla-eum-aliquid-ut-nesciunt-commodi"},{"name":"cupiditate-h
ic-molestias-facilis-non-qui-praesentium","commit":{"id":"27c8a48104a18d0a1e25b32abac71bc6dcf0af44","short_id":"27c
8a481","created_at":"2020-09-25T11:38:58.000+00:00","parent_ids":null,"title":"Qui est sed quod.","message":"Qui es
t sed quod.","author_name":"Administrator","author_email":"admin@example.com","authored_date":"2020-09-25T11:38:58
```

Figure 4.39: Branch request Output

GitLab API usage in the GitLab Application

The GitLap API is a necessary tool for the GitLab application to bypass some of the constraints identified with the usage of webhooks as well as further extend the possible information that can be fetched from repository activity allowing us to further develop our future dataset and discover better process models.

These technologies will be used in a balance, the webhook will be able to control exactly whenever requests to the API are necessary. According to the type of trigger, information available from the trigger and if any webhook contraint is active or not, the usage of the GitLab API can be managed and limited to the necessary requests, acting as a complement to the webhook and vastly increasing the efficiency of fetching the necessary data for the application.

4.3.3 Identification of Relevant Information for *Data Preparation*

After the study into the webhook and the API, one can now attempt to select the data that will be used to construct the *Data Preparation* dataset. For this, the API and Webhook documentation were analyzed to identify which webhook Triggers and API Endpoints would serve the purpose of contributing to the depiction of what a workflow of software development activites resembles within a GitLab repository.

The following tables 4.3 and 4.4 represent the initial assessment of triggers, endpoints and the specific relevant data that can be obtained. As explained previously in 4.2.2, *Data Preparation* can change as a better understanding is reached and process mining algorithms are applied to the dataset, meaning that these tables are not the final and will be most likely changed according to the development of the project.

Trigger	Event Type	Relevant Event Data
Push events	push	user, timestamp, added, modified, removed, branch, commits
Issue events	issue	user, timestamp(created, updated), state (updated, closed, reopened), title, description
Merge request events	merge_request	user, target_branch, source_branch, timestamp(created, updated), last_commit, title, description, state(opened, closed, locked, merged), merge_status(can_be_merged, cannot_be_merged)
Pipeline events	pipeline	branch, source, status, stages, timestamp(created, finished), duration
Job events	build	build_name, build_stage, build_status, timestamp(started, finished, duration), pipeline
Deployment events	deployment	status(created, running, success, failed, canceled)
Release events	release	timestamp(created, released), name, description

Table 4.3: Webhook identified triggers and the relevant data

Endpoint	Relevant Event Data
Issues API	user, timestamp(updated, closed, created), state(updated, closed, reopened), title, descripton
Issue Statistics	counts, all, closed, opened
Jobs API	commit, user, status, timestamp(created , started , finished), duration,
Merge requests API	title, description, timestamp(merged_at, created_at, updated_at, closed_at), target_branch, source_branch, state(merged, cannot_be_merged),
Pipelines API	status, branch, timestamp(created, updated, finished, commited), duration
Releases API	name, description, timestamp(created, released)
Branches API	name, merged, commit
Commits API	user, title, message, timestamp(commit_date, created)
Deployments API	timestamp(created, updated, finished), status(created, running, success, failed, canceled)

Table 4.4: GitLab API identified endpoints and the relevant data

4.4 PM4Py - Process Mining for Python

PM4Py is an open-source python library developed by the process mining team of Fraunhofer FIT (Fraunhofer Institute for Applied Information Technology). It allows the extraction of process models from datasets using a variety of available process mining algorithms.

This library is going to be essential for future work into the GitLab process mining application as its results will represent the main graphical presentation of the data fetched from GitLab, allowing for an analysis of the processes extracted, how they are working and how well they reflect the actual processes of development.

PM4Py offers a large range of implemented approaches for process mining algorithms. These won't be explained as it is out of scope of this thesis work, however, as mentioned in 4.2.3, process mining algorithms focus on extracting process models using different kinds of graphs such as Petri Nets and annotated rooted-trees.

For the initial assessment of algorithms, the main goal was focusing on algorithms that deemed relevant for the GitLab process mining app. More algorithms can be selected as a better understanding of the overall work is reached but as of now, the following list represents the algorithms that will be explored in the future:

Process Discovery Algorithms

- **Inductive Miner** - Discovering block-structured process models from event logs-a constructive approach;
- **Inductive Miner Infrequent** - Discovering block-structured process models from event logs containing infrequent behaviour;
- **Inductive Miner Directly-Follows** - Scalable process discovery and conformance checking;
- **Heuristics Miner** - Process mining with the heuristics miner-algorithm; [25]

Conformance Checking Algorithms

- **Token-based Replay** - A Novel Token-Based Replay Technique to Speed Up Conformance Checking and Process Enhancement;
- **Alignments** - Conformance checking using cost-based fitness analysis;
- **Decomposed/Recomposed Alignments** - Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining;
- **Log Skeleton** - Log skeletons: A classification approach to process discovery; [25]

4.5 Conclusion

The current chapter detailed all the base work necessary in order to build the ProjectScanner. It details every major decision related to the exploratory work that went into shaping the development process of the tool. This chapter also introduces our previous point of focus and establishes decisions and the thought process behind an eventual expansion of our tool with the inclusion of Process Mining. The next chapter will introduce the decisions that were made for the implementation of the system itself as a result of the preliminary work in this chapter.

Chapter 5

System Specification

For this chapter, the System Description is introduced, containing the User Stories, Use Cases, Functional and Non-Functional Requirements, Mockups, Navigation Diagram, the two abstraction models of the C4 model and the Entity-Relationship Diagram.

5.1 User Stories

User stories are informal and a general overview of a software feature written from the perspective of the end user of said software [1]. Since our software doesn't have a specific client, being mainly an internal tool for project repository analysis, this meant that we had no pre-requisite requirements to work with or client specifications. These requirements had to be built while working on the Exploration Phase.

User Stories provided a quick and preliminary way to build a starting set of requirements. User Stories usually sprang from brainstorming ideas from meetings across the second semester and would be explored within the following weeks to evaluate relevancy.

From the meetings and exploration, we would eventually focus on the following User Stories as a decent starting point into building the requirements for the software:

User Story Template: "As a **[user]**, I **[want to]**, **[so that]**."

A set of examples of the User Stories will be presented in this subsection, with the total set of Use Cases being fully present in the Appendix.

User Stories related with the ProjectScanner:

- "As a user, I want to be able to upload my git repository so that I can visualize information about it."
- "As a user, I want to be able to configure the used regex of my Project so that the information displayed can more accurately represent the reality of my project."
- "As a user, I want to be able to upload multiple projects unto the platform so that I can monitor multiple projects at once."

User Stories related to the Nature of the Effort

[1] https://www.atlassian.com/agile/project-management/user-stories

- "As a user, I want to be able to visualize the overall nature of the effort of the project, divided by the categories identified by the regex so that I can have a better understanding of where that effort is being applied."
- "As a user, I want to be able to visualize the nature of the effort of the team members, sorted by the categories identified by the regex so that I can have a better understanding of where that effort is being applied per team member."
- "As a user, I want to be able to visualize the nature of the effort of a specific team member across time within the categories identified by the regex so that I can have a better understanding of how that effort was distributed across time."

User Stories related to Project Files

- "As a user, I want to be able to visualize a specific project file so that I can better understand its role within the project."
- "As a user, I want to be able to visualize the history of modifications so that I can understand which team members were working on a specific file across time."
- "As a user, I want to be able to visualize which project files a specific team member worked on so that I can have establish the focus of that specific team member's work."

User Stories related to Directory Trees

- "As a user, I want to be able to visualize the complete directory tree of my project sorted by the categories identified by the regex so that I can better understand how which project file is connected to the overall project as well as their internal roles."
- "As a user, I want to be able to visualize the directory tree per commit, sorted by the categories identified by the regex so that I can have a better understanding of the evolution of the project as well as the internal roles of the files."
- "As a user, I want to be able to visualize the directory tree of a specific team member, sorted by the categories identified by the regex so that better understand the project state when a specific team member was working on the project."

5.2 Use Cases

The following step into establishing the requirements for the Project Scanner was taking the User Stories and transforming them into Use Cases. Use Cases provide a more detailed outlook of how tasks will actually be performed within the software itself and are a necessary step to establish basic interactions of the user with the platform. Resulting ideas of meetings that happened during/after the writing of the Use Cases were added solely as one/multiple Use Cases but don't have a User Story counterpart.

A set of examples of Use Cases will be presented in this subsection, with the total set of Use Cases being fully present in the Appendix.

We used the following template to write the Use Cases:

Name – Identifier of the use case, should provide the clear purpose of the use case.

ID – Unique identifier of the use case.

Actor – Who or what is using the use case.

Description – A short description of the use case.

Preconditions – The prerequisite state of the system before the usage of the use case.

Postconditions – The postcondition state of the system after the usage of the use case.

Triggers – Event which causes the use case to trigger.

Basic Flow – Steps taken by the actor to fulfill the purpose of the use case.

Alternative Flow – Variations of the Basic Flow of the use case. These can contain alternative paths of the use case as well as exceptions caused by malfunctions of the system.

Name	Create Project
ID	FR01
Actor	User
Description	The user creates a new project.
Trigger	Clicking "Novo Projeto" located in the Homepage.
Preconditions	1. There is not another project in the Database with the same name.
Basic Flow	1. User is in the Homepage 2. User clicks "Novo Projeto" 3. User inputs the Project Name 4. User inputs the git repository HTTPS URL 5. User clicks "Criar Projeto" 6. User clicks the "Confirmar" button.
Postconditions	Project is created and populated in the Database.
Alternative Flow	6. If there is already a project with the name the user input, the user will be redirected to the new project page to try again. 7. Project may fail to correctly populate the Database due to no internet connection to fetch the needed logs, incorrect URLs or an unrelated failure on Git's part.

Table 5.1: Create Project Use Case

Name	Create Regex
ID	FR02
Actor	User
Description	The user creates a regex configuration to be applied to the project.
Trigger	1. a) Redirected after creating a project b) Clicking the "Modificar Regex" button located in that project's page.
Preconditions	1. There is an active project selected.
Basic Flow	1. User is in the Configure Regex Page 2. User clicks "Modificar Regex" 3. User clicks "Criar nova configuração". 4. User inputs the regex name. 5. User inputs at least one role name and ruleset. 6. User clicks the "Utilizar Regex".
Postconditions	Regex is populated in the Database and applied to the project.
Alternative Flow	7. If there is already a Regex Configuration with the name the user input, the user will be redirected to the regex creation page to try again. 8. Due to a system failure, Regex failed to create/correctly populate.

Table 5.2: Create Regex Use Case

Name	Display Artifact View - Tree per Commit
ID	FR17
Actor	User
Description	The user consults the file directory tree per commit within the Artifact View page, visualizing the data displayed.
Trigger	1. Clicking the "Árvore por Commit" button on the Artifact View - Tree Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar. 2. User clicks on the "Árvores" button. 4. User clicks the "Árvore por Commit" button.
Postconditions	The Artifact View - Tree per Commit is displayed with the associated data.
Alternative Flow	None

Table 5.3: Display Artifact View - Tree per Commit Use Case

Name	View File
ID	FR20
Actor	User
Description	The user consults information about a file, visualizing the data displayed.
Trigger	1. a) Clicking the file name on any Artifact Tree page. b) Clicking the "Ver Ficheiro" button on any Artifact View - File page.
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar 2. User clicks the "Ficheiros" button 3. User clicks the "Categoria de Ficheiros" button 4. User clicks the "Ver Ficheiro" button
Postconditions	The Social View - Handover of Work is displayed with the associated data.
Alternative Flow	1. User clicks the "Vista Social" button on the navigation bar 1.1 User clicks the "Indíviduos" button 1.2 User clicks the "Árvore de Indivíduo" button 1.3 User clicks on one of the files 2. User clicks the "Árvores" button 3. a) User clicks the "Árvore Completa" button b) User clicks the "Árvore por Commit" button c) User clicks the "Árvore CHURN" button 4. User clicks on one of the files

Table 5.4: View File Use Case

5.3 Functional Requirements

The follow-up to the completion of the Use Cases was specifying them into the Functional Requirements of the system. This entailed gathering all the Use Cases and organizing them into priorities of development. The priorities were categorized as such:

Must Have: Requirement necessary for the project.

Should Have: Requirement is important but not a necessity for the project.

Could Have: Requirement is preferable for the project but not crucial.

Project General Functional Requirements

Name	**ID**	**Priority**
Create Project	FR01	Must Have
Create Regex	FR02	Should Have
Select Regex	FR03	Must Have
Delete Regex	FR04	Could Have
Select Project	FR05	Must Have
Update Project	FR06	Must Have
Delete Project	FR22	Could Have

Table 5.5: Project General Functional Requirements

Effort Views Functional Requirements

Name	ID	Priority
Display Project View	FR07	Must Have
Display Social View - Team Effort	FR08	Must Have
Display Social View - Individual Nature of the Effort	FR09	Must Have
Display Social View - Individual Effort Across Time	FR10	Must Have

Table 5.6: Effort Views Functional Requirements

Files Functional Requirements

Name	ID	Priority
Display Social View - Manipulated Files	FR11	Must Have
Display Artifact View - File Category	FR13	Must Have
Display Artifact View - Most Modified Files	FR14	Should Have
Display Artifact View - Modified Files by Most People	FR15	Should Have
View Modification History	FR19	Must Have
View File	FR20	Must Have

Table 5.7: Files Functional Requirements

Trees Functional Requirements

Name	ID	Priority
Display Social View - Individual Tree	FR12	Must Have
Display Artifact View - Complete Tree	FR16	Must Have
Display Artifact View - Tree per Commit	FR17	Should Have
Display Artifact View - CHURN Tree	FR18	Should Have
Display Social View - Handover of Work	FR21	Could Have

Table 5.8: Trees Functional Requirements

Due to the nature of the project and what was explained during the Exploration Phase, the focus of the project was shifted from Process Mining specifically to Mining Software Repositories and Data Visualization. However, we ended up still including one Functional Requirement associated with Process Mining (FR21) in case there was enough time to expand and include this specific requirement.

5.4 Non-Functional Requirements

Non-functional requirements are used to specify how a software system should work by defining quality attributes and constraints if they exist. For the ProjectScanner, we considered the following categories.

5.4.1 Usability

One of the key points we considered for the ProjectScanner was the user-friendly interface that would allow an user to navigate through the many different functionalities seamlessly.

ID	NFR1
Source	User
Stimulus	Navigate through the application and use the different functionalities
Artifact	ProjectScanner
Environment	Normal
Response	Navigation should be efficient and provide no confusion to the user
Metric	Time spent using the navigation bar, number of clicks, number of backtracks

Table 5.9: Non-Functional Requirement - Usability

5.4.2 Availability

Other of the aspects we considered was Availability, if the ProjectScanner finds eventual use outside internal applications, it's important to guarantee that the system is consistently available.

ID	NFR2
Source	User
Stimulus	Response from server
Artifact	Server
Environment	Server failure
Response	The server should be available and have as little downtime as possible
Metric	Total amount of downtime

Table 5.10: Non-Functional Requirement - Availability

5.4.3 Security

Security is one of the most important qualities of software and since ProjectScanner deals with sensitive project information that can come from private sources, it's important to assure that the data it contains can't be accessed by outside sources. This has been one topic we have mentioned before and one of the reasons we decided to develop the tool with Django.

ID	NFR3
Source	Unauthorized sources
Stimulus	Unauthorized sources attempt to access database information
Artifact	ProjectScanner
Environment	Normal
Response	All the database information is encrypted and there are protocols in place that help against attacks such as cross-site scripting, Cross-site request forgery, and SQL Injection.

Table 5.11: Non-Functional Requirement - Security

5.5 Mockups and Navigation Diagram

The Navigation Diagram and Mockups are presented in Figure 5.2, Figure 5.3, Figure 5.4 and Figure 5.7. These aid the visualization of key components as well as how each page interacts with each other prior to the start of development and represent the preliminary work done with the User Stories and Use Cases.

The Mockups were used to help establish a baseline for the how each page would be presented and while weren't strictly followed, this baseline was extremely important to the development of the ProjectScanner as a starting point into joining the User Stories, Use Cases, Functional Requirements and the Navigation Diagram together.

In this section, only a few mockup images will be shown as examples, the full mockups are available in the Appendix.

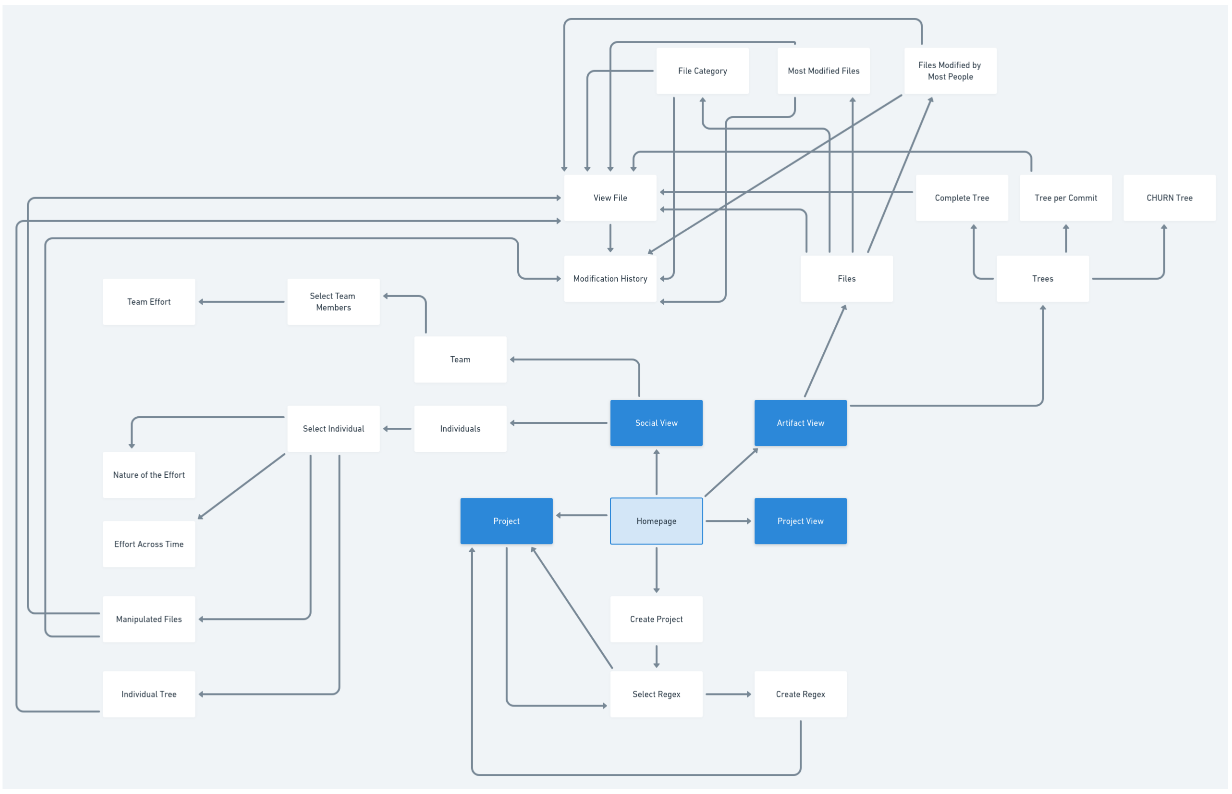


Figure 5.1: Navigation Diagram

For simplicity purposes, the pages presented in the Navigation Diagram are split into three color coded categories:

- **Blue:** Represents a page that can be accessed by all pages using the navigation bar.
- **Light Blue:** Represents the page to where the user will be directed when entering ProjectScanner for the first time and a page that can be accessed by all pages using the navigation bar.
- **White:** A standard page, navigation shown by the diagram is applied.

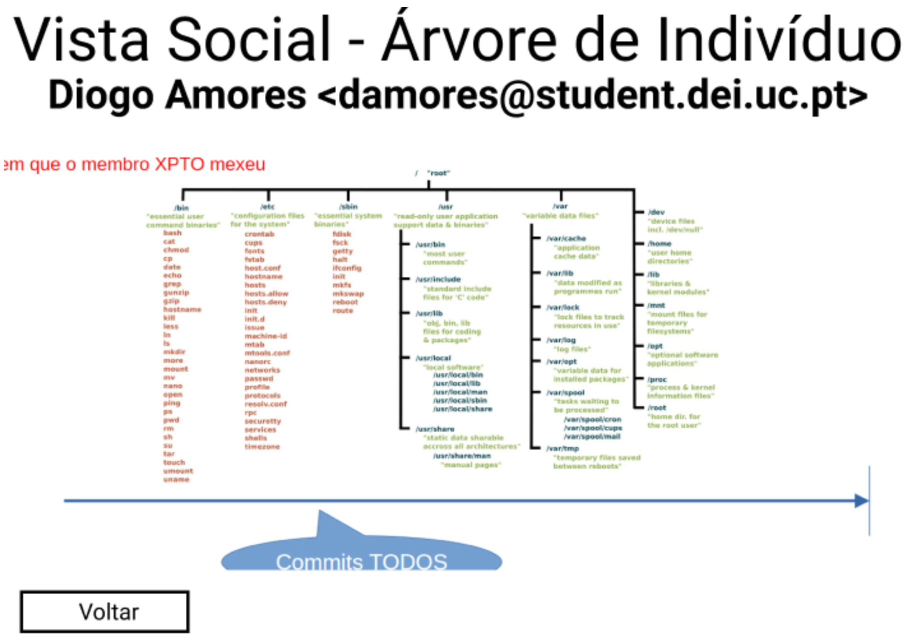

Vista Social - Árvore de Indivíduo
Diogo Amores <damores@student.dei.uc.pt>
em que o membro XPTO mexeu
Commits TODOS
Voltar

Figure 5.2: Initial mockup for the Individual Tree page

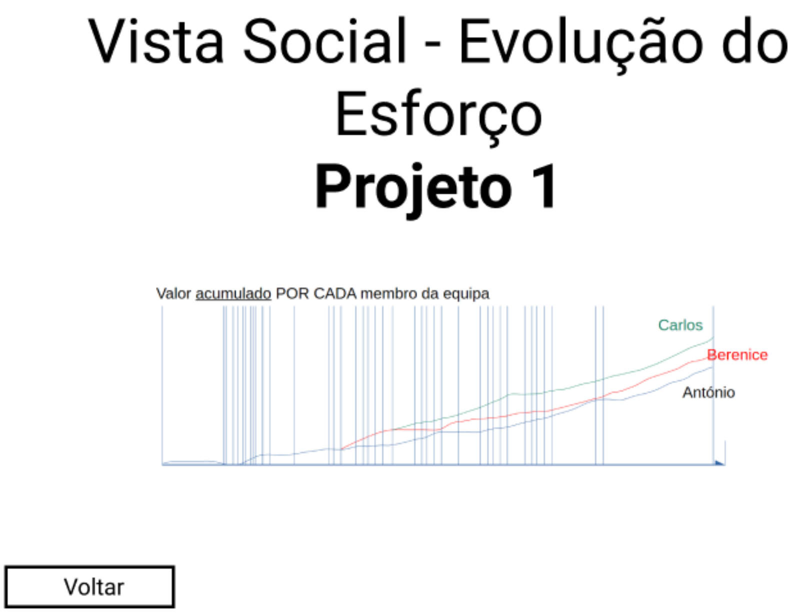

Vista Social - Evolução do Esforço
Projeto 1
Valor acumulado POR CADA membro da equipa
Carlos
Berenice
António
Voltar

Figure 5.3: Initial mockup for the Social View - Effort Across Time page

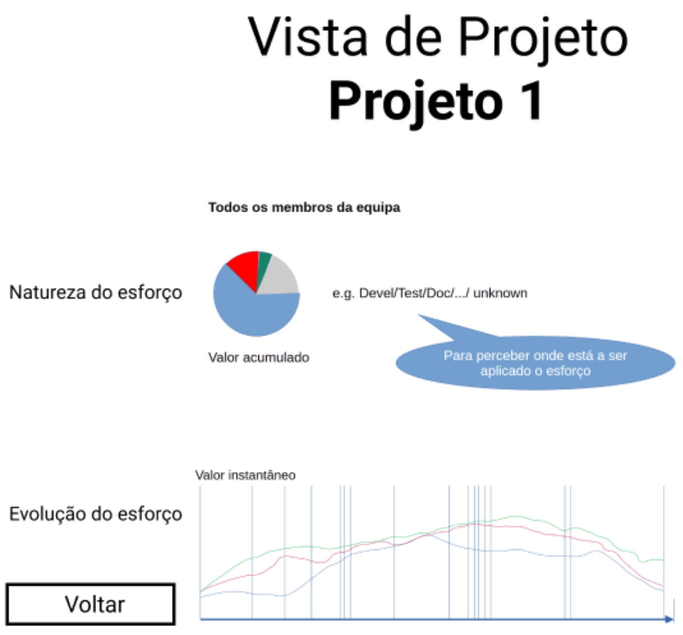

Vista de Projeto
Projeto 1
Todos os membros da equipa
Natureza do esforço
e.g. Devel/Test/Doc/.../ unknown
Valor acumulado
Para perceber onde está a ser aplicado o esforço
Valor instantâneo
Evolução do esforço
Voltar

Figure 5.4: Initial mockup for the Project View page

5.6 C4 Models

For the System Architecture, the C4 Model was applied to create the first two levels of the model, the Context and Container diagrams for the system. The C4 model allows a phased abstraction into each model level, with each level having an increased focus into the whole system.

5.6.1 Context Diagram

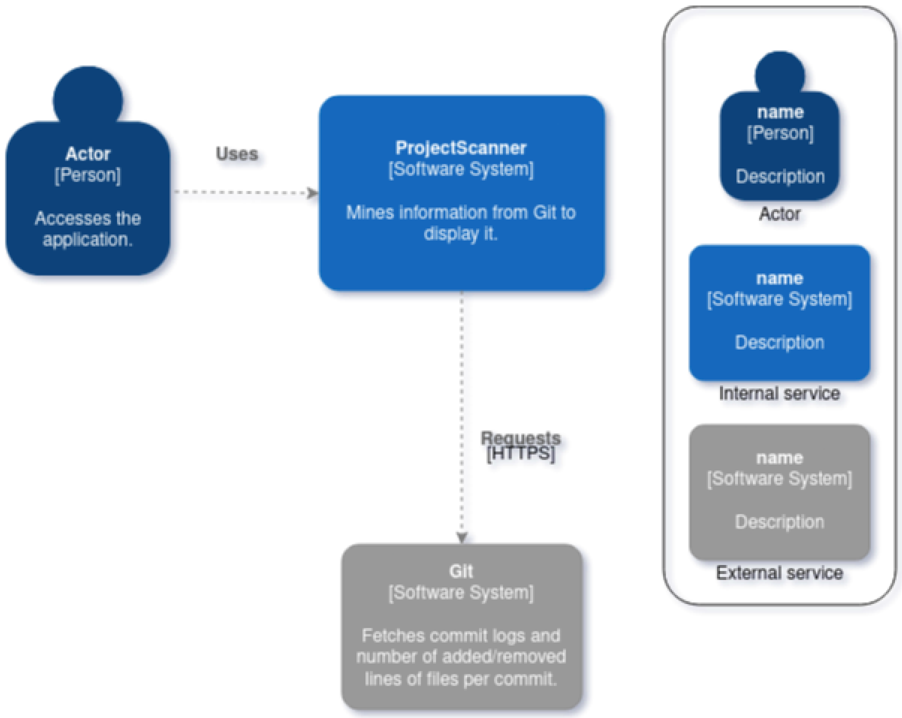


Figure 5.5: Context Diagram

Figure 5.5 represents the context diagram of the system and displays our main external entity interacts with the system.

Firstly, a clone of the git repository is temporarily stored in the project's folders and a complete log of all commit information will be requested, after this log is parsed and populated in the database, the ProjectScanner will request a second set of logs (one log for each commit in the repository) to populate the database with the number of lines added and removed per file and per commit. After this last step, the project is then deleted from the project folder.

5.6.2 Container Diagram

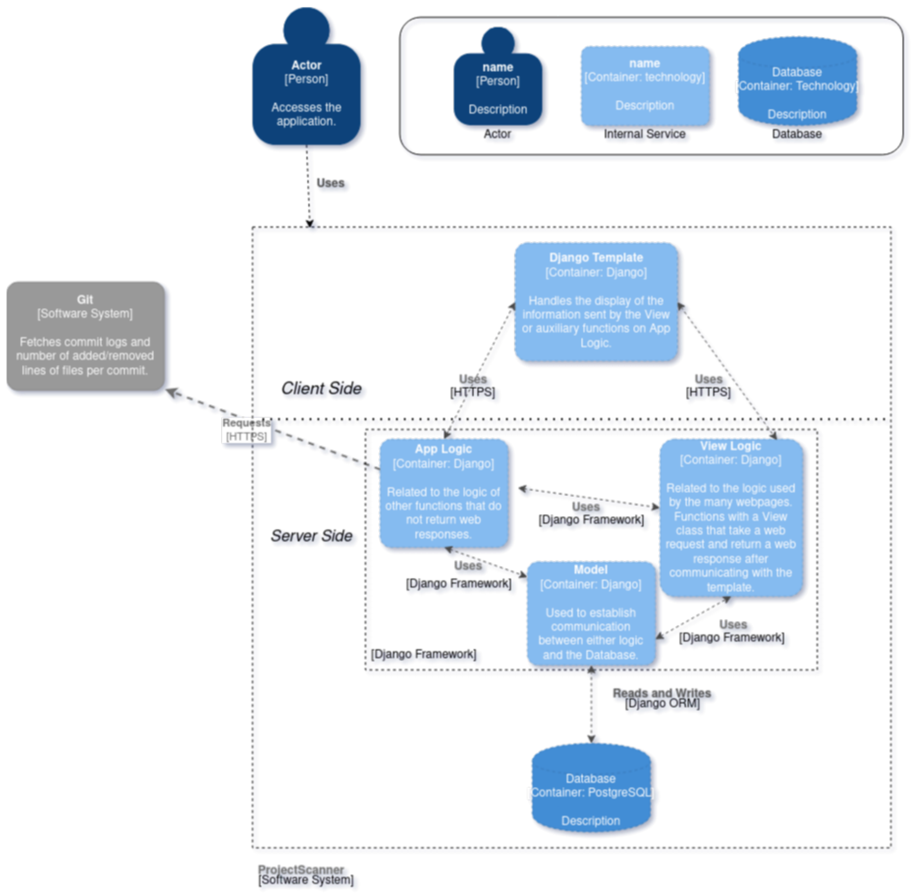


Figure 5.6: Container Diagram

Figure 5.6 depicts the second level of the C4 model and displays in a more detailed way the components that shape the internal service. These components will be explained in greater detail in the 6 but for a succinct explanation, a Django project can be divided into:

- **View Logic:** This component handles all logic related to the pages, this includes web responses, redirects and renders of pages.
- **App Logic:** This component handles all logic unrelated to the View Logic, this includes implementation of all auxiliary functions that can handle different aspects of the ProjectScanner (building of the File Directory Trees, parsing log files and logic for using Git for instance).
- **Model:** Establishes the internal communication between the Database and the above components.
- **Template:** Handles the display of the information dealt by the Server Side components.

5.6.3 Entity-Relationship Diagram

Figure 5.7 represents the ER diagram of the Project Scanner. All tables have a primary key presented as the id (these are auto-generated by Django).

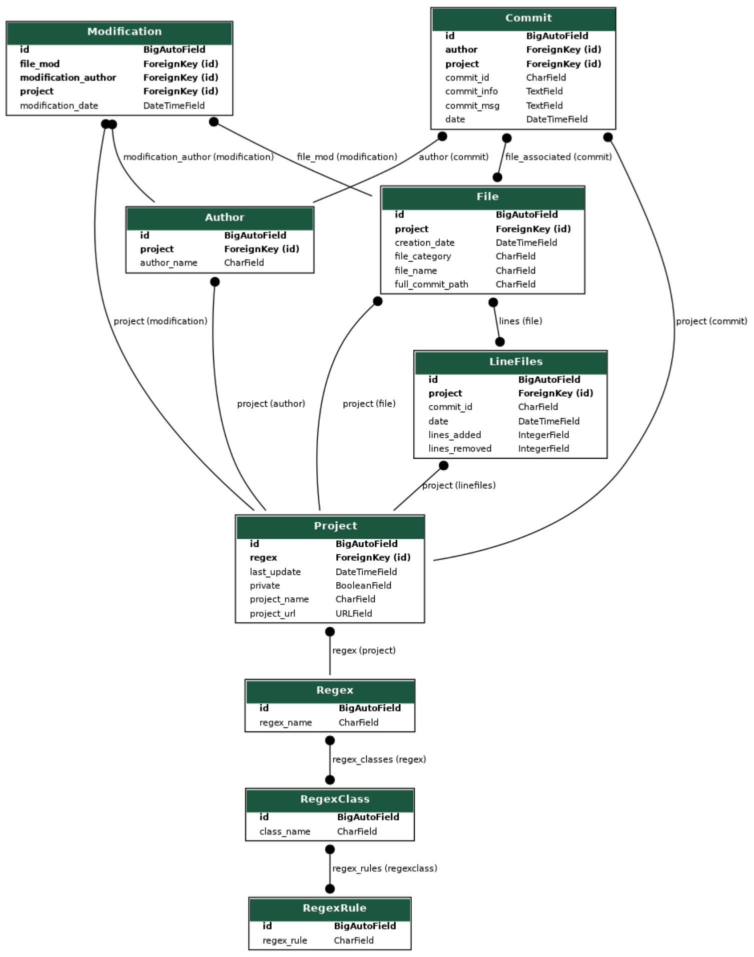


Figure 5.7: Entity-Relationship Diagram

The Project Table has all information related to projects in ProjectScanner. A Project has an id, a project name (has to be unique among all projects), a project URL, when it was last updated, a boolean characterizing whether that project was created from a private repository, and finally, a foreign key relationship with the Regex table, used to identify which regex is selected for each project. The Project Table is also used as a foreign key to most other tables that relate to project-specific information. This relationship immensely helps specific Django queries as it guarantees faster lookups (for example, we can easily search for all Files contained in one project by simply using the id of the project associated with the Files instead of iterating through the Commits of the Project to find the Files associated with that Commit to once again iterate through them. We also would have to be aware of repeated files among the Commit information, as a file can have multiple commits applied to it).

The Regex Table is applied as a foreign key to the Project Table as stated above and is mainly used to specify a regex configuration for the project. The Regex table has an id, a class name, and a Many to Many relationship with the RegexClass table (named regex classes), this means a Regex Configuration can have multiple Regex Classes associated with it. The same logic is applied to the RegexRule Table, which contains an id and a regex rule. A Many to Many relationship is also established between the RegexClass and the RegexRule. The regex rule field in the RegexRule Table is a CharField that is populated with a regular expression meaning that multiple regular expressions entail one specific Regex Class and at least one or more Regex Classes entail a Regex Configuration.

Moving on to the Project-specific information, a Project contains one or more Commits. A Commit has an id, a foreign key relationship with the Author (a commit has one author but an author can have multiple commits), the associated project, a commit id (which is unique to every commit), commit info (a TextField containing the files that were changed by that specific commit), a commit message (containing the message that the author wrote when they pushed the specific commit) and the date of the commit. A commit also has a Many to Many relationship with the File Table (named file associated).

The File Table contains an id, the associated project (presented by a foreign key relationship), creation date, file category (which is a CharField containing the regex class name associated with that file using the regular expressions under the RegexRule Table), file name and the full commit path of the File. This path is associated with the directories leading to that file. The File Table has a Many to Many relationship with the LineFiles Table (named lines) and a foreign key relationship with the Modification Table.

The Modification Table serves to easily identify how many times a File was changed after a Commit. These changes entail the author of the modification, the date, the associated project, and the File.

LineFiles is used to specify the number of lines added and removed from a specific file after a commit. The LineFiles Table contains an id, associated project, commit id, date, and then the number of lines added and removed on that specific commit. A File can have multiple LineFiles Tables associated with it according to the number of commits that actively change that specific file.

The Author Table has an id, associated project, and author name. This Table (similarly to the Project Table) is also used as a foreign relationship with multiple other tables related to Commit information.

5.7 Conclusion

This chapter featured an overview of all relevant elements that shape and enable the development of the ProjectScanner to start. User Stories, Use Cases, Fundamental Requirements, Non-Fundamental Requirements are outlined and culminate in the Mockups and Navigation Diagram that form the early stages of development. The Context and Container Diagrams describe different abstraction levels of how each component in the overall system interacts with each other and finally, the Entity-Relationship demonstrates the different table relationships.

For the next chapter, an overview of the Development phase will be introduced. This includes our internal organization, the general structure of a Django project and the structure of our own tool.

Chapter 6

Development

This chapter begins by explaining the basic course of action behind the development. After this, it details the Project Structure showing the different components that follow a Django project and how these relate to each other.

6.1 Project Organization

Organization is crucial to a smooth development cycle and to ensure the quality of the final product. When discussing organization, we briefly explored ideas such as a Trello board or similar frameworks but we ultimately decided using these frameworks wasn't necessary and concluded on a more internal system.

This system was ultimately based on the already established Functional Requirements and the Mockups. Firstly, we decided on an order of Development based on the Functional Requirements presented in Tables 5.5, 5.6, 5.7 and 5.8.

The first step before the actual development was establishing the Navigation Diagram as designed in Figure 5.7. After the base navigation was implemented, the development followed the order of Table 6.1. Table 6.1 illustrates the different development cycles and sequential order of development of the different requirements.

Requirement	**ID**	**Priority**	**Development Cycle**
1. Project General	-	-	-
Create Project	FR01	Must Have	First
Select Regex	FR03	Must Have	First
Select Project	FR05	Must Have	First
Update Project	FR06	Must Have	First
2. Effort Views	-	-	-
Display Project View	FR07	Must Have	First
Display Social View - Team Effort	FR08	Must Have	First
Display Social View - Individual Nature of the Effort	FR09	Must Have	First
Display Social View - Individual Effort Across Time	FR10	Must Have	First
3. Files	-	-	-
Display Social View - Manipulated Files	FR11	Must Have	First
Display Artifact View - File Category	FR13	Must Have	First
View Modification History	FR19	Must Have	First
View File	FR20	Must Have	First
4. Trees	-	-	-
Display Social View - Individual Tree	FR12	Must Have	First
Display Artifact View - Complete Tree	FR16	Must Have	First
5. Trees	-	-	Second
Display Artifact View - Tree per Commit	FR17	Should Have	Second
Display Artifact View - CHURN Tree	FR18	Should Have	Second
6. Files	-	-	-
Display Artifact View - Most Modified Files	FR14	Should Have	Second
Display Artifact View - Modified Files by Most People	FR15	Should Have	Second
7. Project General	-	-	-
Create Regex	FR02	Should Have	Second
8. Project General	-	-	-
Delete Regex	FR04	Could Have	Third
Delete Project	FR22	Could Have	Third
9. Trees	-	-	-
Display Social View - Handover of Work	FR21	Could Have	Third

Table 6.1: Development Order and Priorities

Requirements were then split into the following categories:

- **To Do:** Development of this requirement hasn't begun.
- **Ongoing:** Requirement is currently being implemented.
- **Done:** Requirement is completed.
- **Done with Issues:** Baseline of the requirement is completed but it has issues/problems and should be revisited as soon as possible.

These four categories were essentially applied to all requirements according to Table 6.1. Whenever a requirement was considered **Done** or **Done with Issues** (sometimes there were problems with requirements that couldn't be solved at the time or that required certain inputs from a meeting) it was inserted into a simple 'To do' List and the process continued.

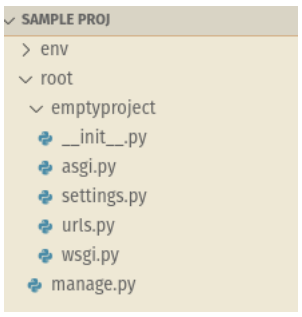
6.2 Project Structure

It is of extreme relevance to a Project that the mainline conventions of a project structure remain in accordance to the framework being used in order to guarantee that the code can be decently maintained and scalable in the future.

As we've already discussed, there are plans to introduce further functionalities into the ProjectScanner, mainly in regards to Process Mining as it would be a logical step into advancing the ProjectScanner. For this reason, we have even stronger motives to guarantee that the ProjectScanner can be maintained and further expanded in the future.

This subsection's main purpose is exploring the Project Structure of a Django Application. We'll begin with introducing what a Django project directory looks like in its default state and then advance into the specifics of the ProjectScanner.

6.2.1 Base Structure



```
SAMPLE PROJ
> env
v root
  v emptyproject
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    manage.py
```

Figure 6.1: Base Structure of a Django Project

Figure 6.1 represents the initial state of a Django Project after creation. It is important to note that this base template only contains the base Project directory. This directory, named "emptyproject" contains the base configuration files for Django applications and are described as follows:

- **root:** The root directory of the project.
- **emptyproject:** This corresponds to the python package to the project, the name of this package is the name used to import anything project specific contents (such as urls).
- **__init__.py:** An empty file whose purpose is simply telling the python interpreter that this folder is considered as a python package This is very standard python behaviour and is used in numerous python based projects and frameworks.
- **asgi.py and wsgi.py:** Enable compatibility for ASGI and WSGI for deploying purposes.
- **settings.py:** As the name implies, this file is used to specify settings to the Django Project. These include for instance, installing applications that are going to be used to actually develop the Project or configuring Database connections.
- **urls.py:** Used to declare every URL and the specific path across the project.

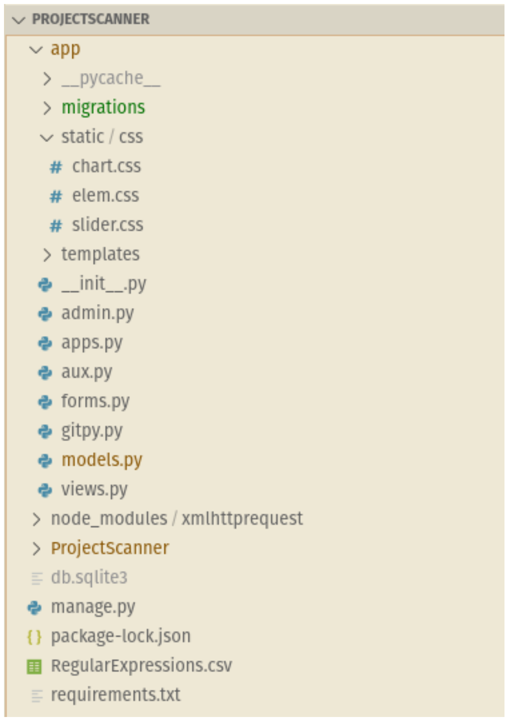
- **manage.py:** Command line utility used to interact with the Django Project, this includes running the webserver, maintaining and running database migrations or configuring some aspects of the Admin Panel (such as creating a super user).

It is noticeable that this base structure by itself does not allow us to begin writing code for the Project as it is mostly filled with configuration files (asides from the urls.py which we will actively use to update and create the URLs for the project). This is where the notion of Application becomes relevant.

Django Applications are the actual place where code will be developed. These are extremely useful because they allow the project to be split into different components and it makes the code incredibly easy to be maintained and scaled. We mentioned that the logical step into scaling the ProjectScanner would be to implement Process Mining methodologies in the future. Using Django applications allows different requirements to be easily added to the project without the risk of hindering progress already made by simply using another application to build the Process Mining requirements. If one wishes to switch projects, Django Applications allow the developer to move applications into other projects without much effort[1].

6.2.2 Application Structure

Figure 6.2 presents the ProjectScanner's application named "app", where all code was implemented for this Project. The ProjectScanner's project directory is the same as the base structure featured in 6.1 (here, the project folder is named "ProjectScanner" instead of "root").



```
PROJECTSCANNER
  app
    __pycache__
    migrations
    static / css
      chart.css
      elem.css
      slider.css
    templates
    __init__.py
    admin.py
    apps.py
    aux.py
    forms.py
    gitpy.py
    models.py
    views.py
  node_modules / xmlhttprequest
  ProjectScanner
  db.sqlite3
  manage.py
  package-lock.json
  RegularExpressions.csv
  requirements.txt
```

Figure 6.2: Structure of the ProjectScanner Application

[1]https://docs.djangoproject.com/en/3.2/intro/tutorial01/ and https://djangobook.com/mdj2-django-structure/

There are a two important files outside the application folder that are relevant to the project, these are:

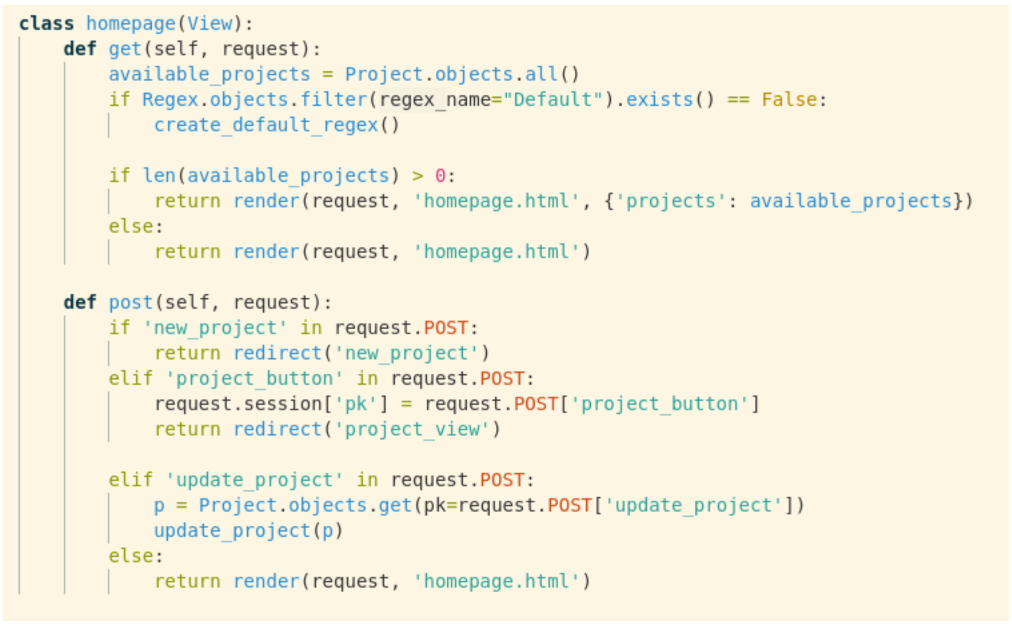
- **RegularExpressions.csv**: This file contains the Regex for the Default category that ProjectScanner features upon creation. When first entering the ProjectScanner, the Default Regex will be immediately created and available to use.
- **requirements.txt**: A text file featuring all the different libraries used in the Project. Extremely useful for setting the project up in multiple environments. These can be installed into a python Virtual Environment using a simple script line[2].

Our application contains the following files:

- **views.py:** This file features all functions or classes that deal with webpage logic. Webpage logic can be considered any function or class that receives a web request and returns some kind of web response or redirect. Figure 6.3 contains an example of a View function of the ProjectScanner.
- **models.py:** File that contains the models of the Project. A Django Model is used to create tables and the different relationships between entities. A Django Model features python code that the Django Framework converts into SQL when communicating with the Database. Figure 6.4 contains an example of a Model used in ProjectScanner.
- **gitpy.py:** Auxiliary file containing all functions that deal with the use of Git and relate to the gitpy library. Figure 6.5 contains an example of one of these functions.
- **forms.py:** File that contains the Django Forms used in the Project. Django by default provides a Form Class that can be used to create HTML forms as it would normally be done in the HTML. Django Forms allow the developer to create these forms using python and even permit the usage of a Django model as the basis for the Form. Figure 6.6 contains an example of a Form used in the ProjectScanner.
- **aux.py:** File that contains another set of auxiliary functions. Functions in this file deal with the parsing of the logs and their correct addition to the database, building the D3 dendogram for displaying the File Directory Tree, Regex functions and other small functions associated with sorting and checks. Figure 6.7 contains one of the functions used in the ProjectScanner.
- **apps.py:** A standard configuration file of the application.
- **admin.py:** Used to indicate which Models are going to be displayed in Django's Admin Panel. The Admin Panel can be used to create/update/delete Models or Users with their specific set of permissions. For the ProjectScanner, usage of the Admin Panel is not required but can be useful. Figure 6.8 contains the Admin Panel of the ProjectScanner.
- **static/css:** Folder containing the css code used within the project.
- **templates:** This folder contains all templates used in the ProjectScanner. As mentioned in 5.6.2, Django Templates deal with the display of information dealt by the

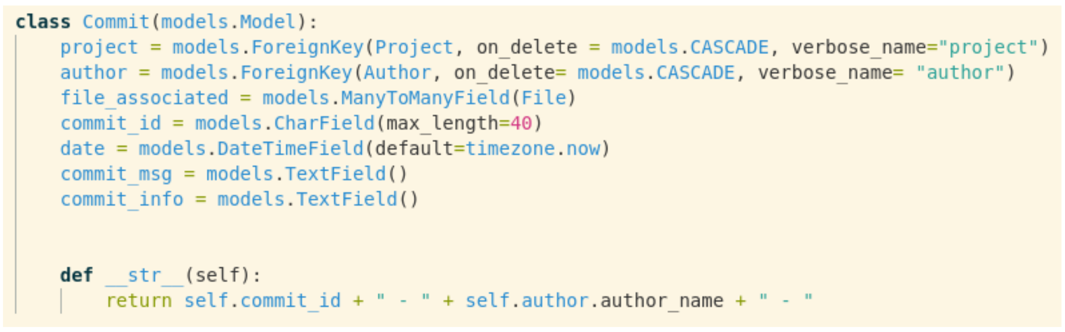
[2]https://docs.djangoproject.com/en/3.2/topics/forms/ and https://djangobook.com/mdj2-django-structure/

other Server Components. These are HTML files that use the Django Template Language. Figure 6.9 shows the different files of the ProjectScanner template and Figure 6.10 showcase one sample Template of the ProjectScanner.



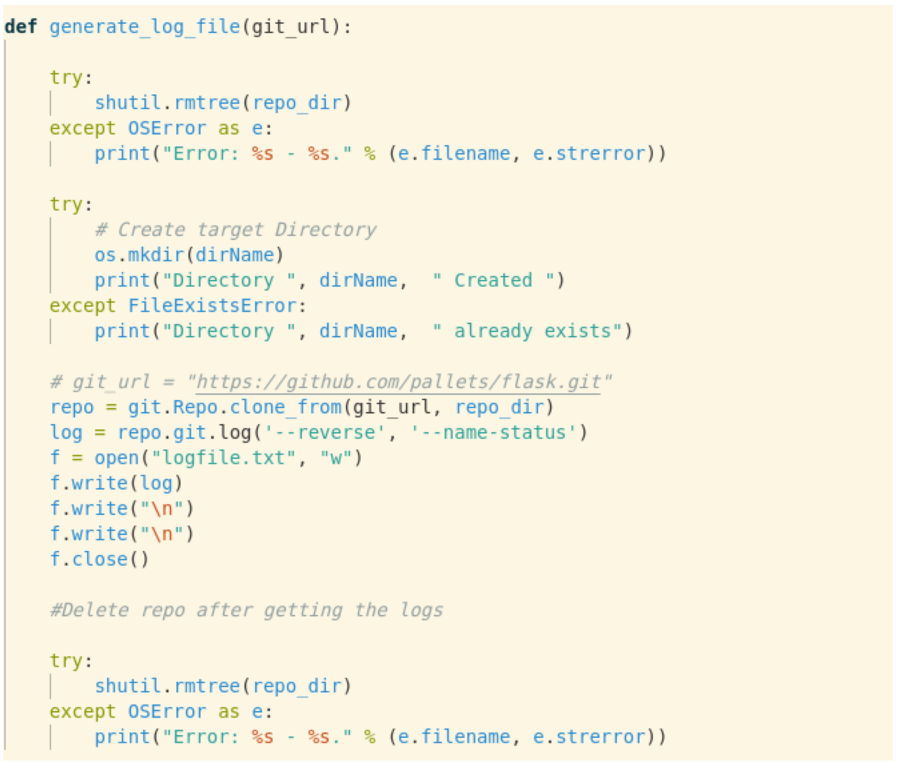
```python
class homepage(View):
    def get(self, request):
        available_projects = Project.objects.all()
        if Regex.objects.filter(regex_name="Default").exists() == False:
            create_default_regex()

        if len(available_projects) > 0:
            return render(request, 'homepage.html', {'projects': available_projects})
        else:
            return render(request, 'homepage.html')

    def post(self, request):
        if 'new_project' in request.POST:
            return redirect('new_project')
        elif 'project_button' in request.POST:
            request.session['pk'] = request.POST['project_button']
            return redirect('project_view')

        elif 'update_project' in request.POST:
            p = Project.objects.get(pk=request.POST['update_project'])
            update_project(p)
        else:
            return render(request, 'homepage.html')
```

Figure 6.3: Homepage View of the ProjectScanner



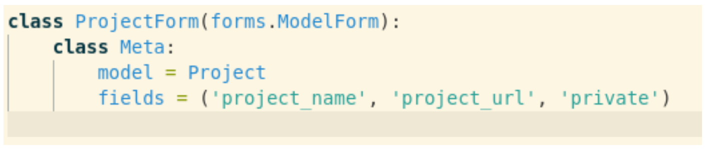
```python
class Commit(models.Model):
    project = models.ForeignKey(Project, on_delete = models.CASCADE, verbose_name="project")
    author = models.ForeignKey(Author, on_delete= models.CASCADE, verbose_name= "author")
    file_associated = models.ManyToManyField(File)
    commit_id = models.CharField(max_length=40)
    date = models.DateTimeField(default=timezone.now)
    commit_msg = models.TextField()
    commit_info = models.TextField()


    def __str__(self):
        return self.commit_id + " - " + self.author.author_name + " - "
```

Figure 6.4: Commit Model of the ProjectScanner



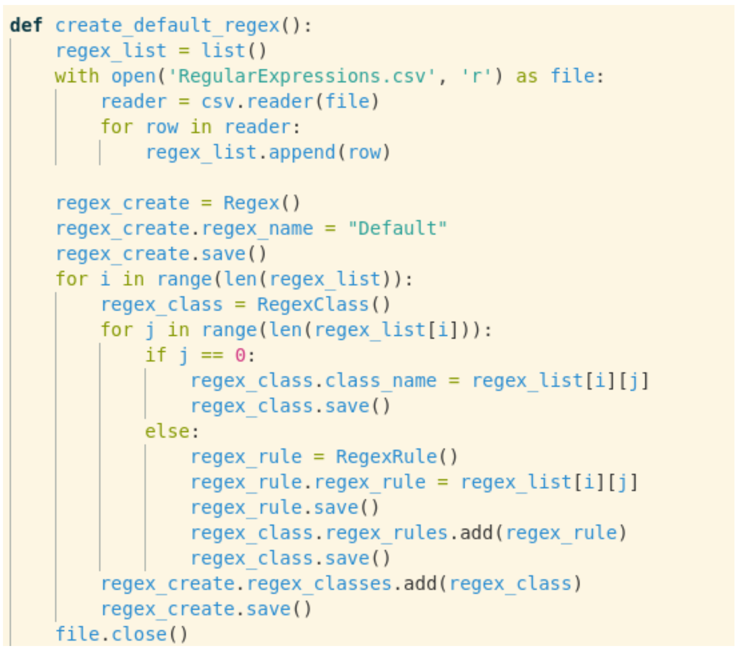
```python
def generate_log_file(git_url):

    try:
        shutil.rmtree(repo_dir)
    except OSError as e:
        print("Error: %s - %s." % (e.filename, e.strerror))

    try:
        # Create target Directory
        os.mkdir(dirName)
        print("Directory ", dirName,  " Created ")
    except FileExistsError:
        print("Directory ", dirName,  " already exists")

    # git_url = "https://github.com/pallets/flask.git"
    repo = git.Repo.clone_from(git_url, repo_dir)
    log = repo.git.log('--reverse', '--name-status')
    f = open("logfile.txt", "w")
    f.write(log)
    f.write("\n")
    f.write("\n")
    f.close()

    #Delete repo after getting the logs

    try:
        shutil.rmtree(repo_dir)
    except OSError as e:
        print("Error: %s - %s." % (e.filename, e.strerror))
```

Figure 6.5: Gitpy function that fetches the Commit Log



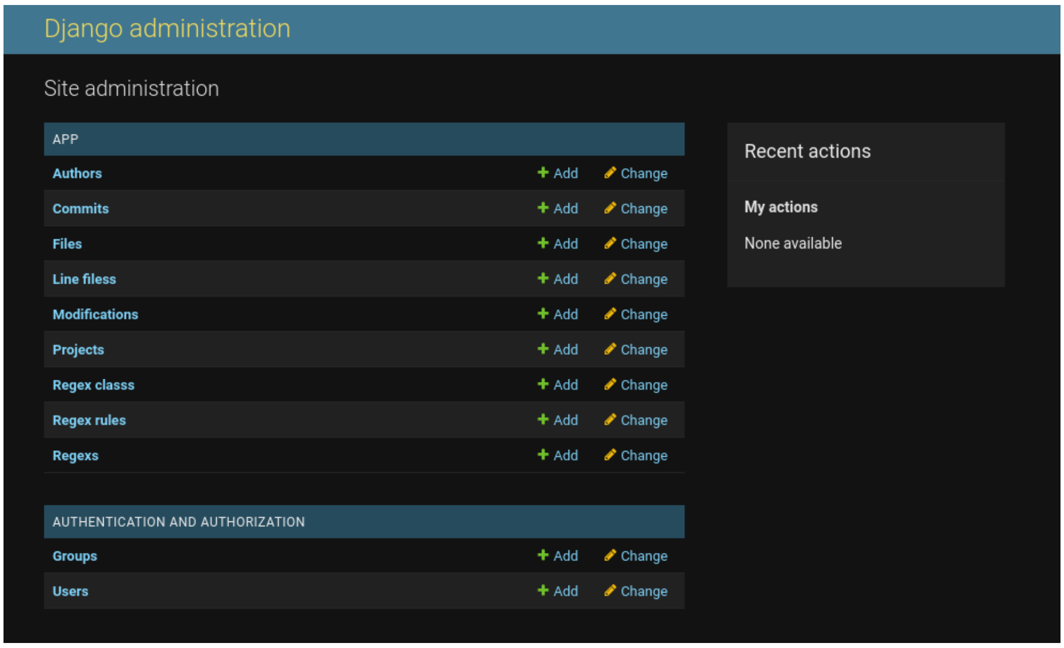
```python
class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ('project_name', 'project_url', 'private')
```

Figure 6.6: Project Form in the ProjectScanner



```
def create_default_regex():
    regex_list = list()
    with open('RegularExpressions.csv', 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            regex_list.append(row)

    regex_create = Regex()
    regex_create.regex_name = "Default"
    regex_create.save()
    for i in range(len(regex_list)):
        regex_class = RegexClass()
        for j in range(len(regex_list[i])):
            if j == 0:
                regex_class.class_name = regex_list[i][j]
                regex_class.save()
            else:
                regex_rule = RegexRule()
                regex_rule.regex_rule = regex_list[i][j]
                regex_rule.save()
                regex_class.regex_rules.add(regex_rule)
                regex_class.save()
        regex_create.regex_classes.add(regex_class)
        regex_create.save()
    file.close()
```

Figure 6.7: Function that creates the Default Regex



Django administration

Site administration

APP		
Authors	+ Add	Change
Commits	+ Add	Change
Files	+ Add	Change
Line filess	+ Add	Change
Modifications	+ Add	Change
Projects	+ Add	Change
Regex classs	+ Add	Change
Regex rules	+ Add	Change
Regexs	+ Add	Change

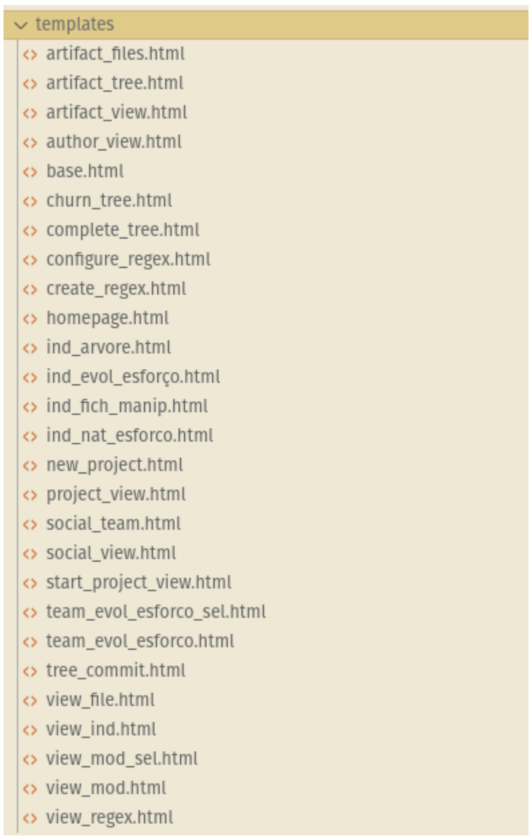
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

Recent actions

My actions

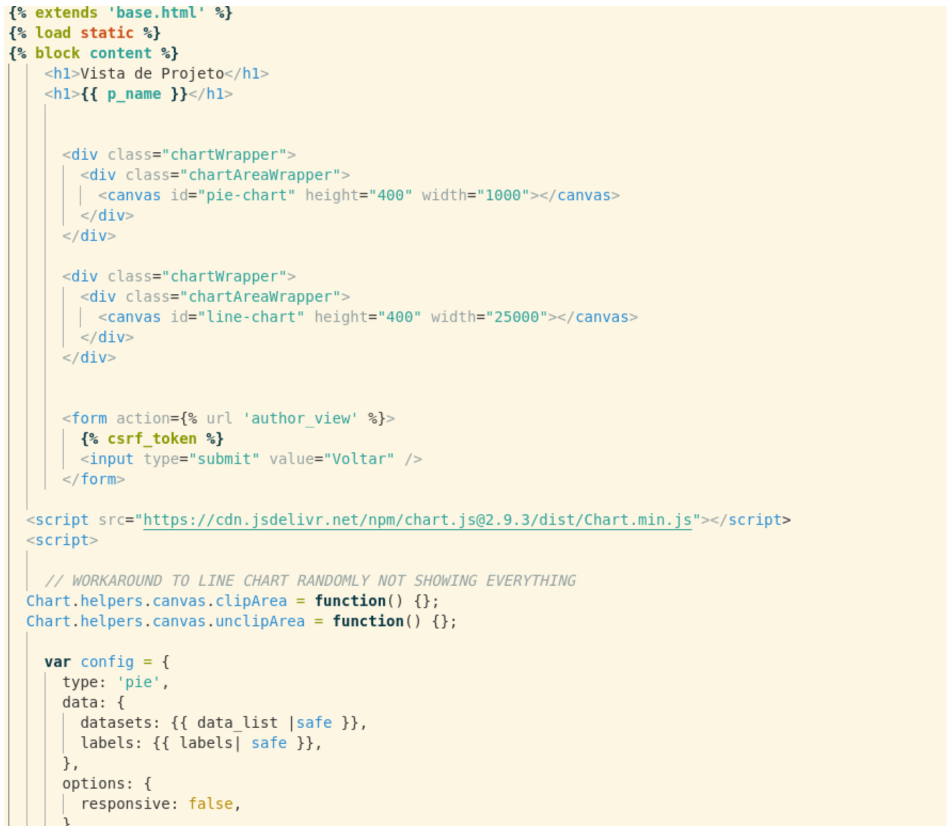
None available

Figure 6.8: ProjectScanner's Admin Panel



templates
artifact_files.html
artifact_tree.html
artifact_view.html
author_view.html
base.html
churn_tree.html
complete_tree.html
configure_regex.html
create_regex.html
homepage.html
ind_arvore.html
ind_evol_esforço.html
ind_fich_manip.html
ind_nat_esforco.html
new_project.html
project_view.html
social_team.html
social_view.html
start_project_view.html
team_evol_esforco_sel.html
team_evol_esforco.html
tree_commit.html
view_file.html
view_ind.html
view_mod_sel.html
view_mod.html
view_regex.html

Figure 6.9: Template folder of the ProjectScanner



```
{% extends 'base.html' %}
{% load static %}
{% block content %}
    <h1>Vista de Projeto</h1>
    <h1>{{ p_name }}</h1>


      <div class="chartWrapper">
        <div class="chartAreaWrapper">
          <canvas id="pie-chart" height="400" width="1000"></canvas>
        </div>
      </div>

      <div class="chartWrapper">
        <div class="chartAreaWrapper">
          <canvas id="line-chart" height="400" width="25000"></canvas>
        </div>
      </div>


      <form action={% url 'author_view' %}>
        {% csrf_token %}
        <input type="submit" value="Voltar" />
      </form>

  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
  <script>

    // WORKAROUND TO LINE CHART RANDOMLY NOT SHOWING EVERYTHING
  Chart.helpers.canvas.clipArea = function() {};
  Chart.helpers.canvas.unclipArea = function() {};

    var config = {
      type: 'pie',
      data: {
        datasets: {{ data_list |safe }},
        labels: {{ labels| safe }},
      },
      options: {
        responsive: false,
```

Figure 6.10: Template of the Project View

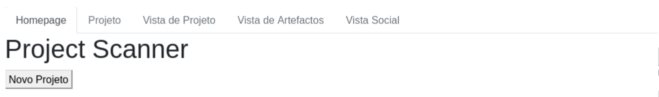
The next Chapter will focus on an overview of the ProjectScanner and aims to showcase what was accomplished as a culmination point of the main objective of this dissertation.

Chapter 7

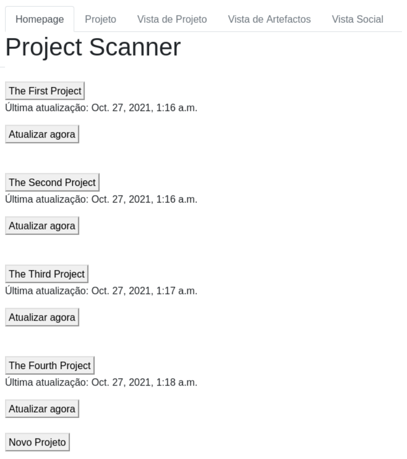
ProjectScanner Overview

This chapter will provide an overview of all the implemented requirements of the ProjectScanner in order to showcase the tool. As a side note, there was an increased focus on the visual aspects when applied to certain requirements such as the File Directory Tree and the required charting of the information present in the Commit Log and Lines Log. However, other aspects were kept minimal with the objective of being improved in the future.

7.1 Homepage



(a) Homepage



(b) Homepage with projects

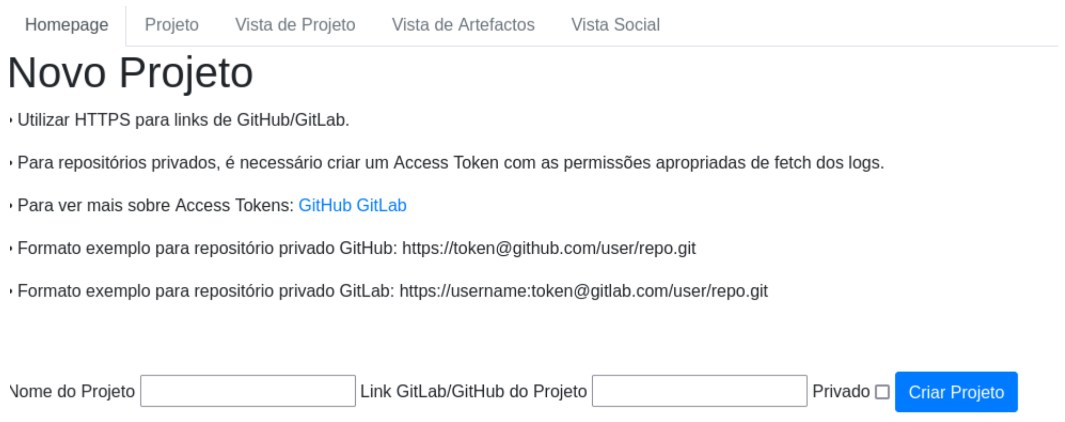
Figure 7.1: The Homepage of the ProjectScanner

Figure 7.1a and Figure 7.1b represent the Homepage, the first page the user is directed to after accessing the ProjectScanner. In this page, the user is able to:

- Create a new project;
- If projects already exist, the user can select or update it.

On this page, along with all the others, the user has access to the **Navigation Bar**. The Navigation Bar allows the user to easily switch between pages and access the many functionalities of the ProjectScanner. If no Project has been selected, the user is unable to access the other pages (this is indicated by the gray color in the Navigation Bar if any project is selected the color will change to blue) and any attempts will redirect the user to the Homepage. ProjectScanner will remember the last project selected for one week.

7.1.1 Create Project



Novo Projeto

- Utilizar HTTPS para links de GitHub/GitLab.
- Para repositórios privados, é necessário criar um Access Token com as permissões apropriadas de fetch dos logs.
- Para ver mais sobre Access Tokens: GitHub GitLab
- Formato exemplo para repositório privado GitHub: https://token@github.com/user/repo.git
- Formato exemplo para repositório privado GitLab: https://username:token@gitlab.com/user/repo.git

Figure 7.2: Create Project Page

Figure 7.2 presents the Create Project Page. Within this page, the user can insert the project name, the URL to the git-based repository and select whether the repository is a private one or not. On this page, the steps to successfully create a project are detailed to the user so that this process can be simpler. These entail:

- Use HTTPS for GitHub/GitLab links;
- For private repositories, an Access Token is required, this Access Token requires the adequate permissions for log extraction;
- The page then details two links for the GitHub and GitLab documentation that explain how to configure these Access Tokens;
- Finally, the example format for private repositories for GitHub and GitLab is shown.

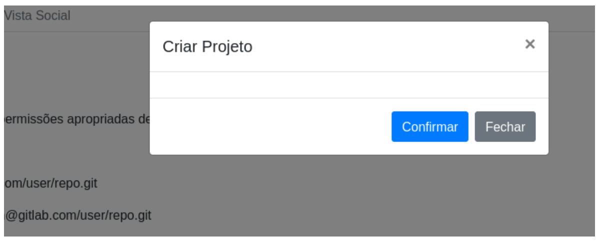
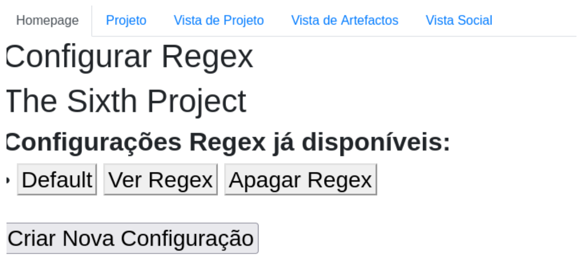


Figure 7.3: Confirmation of project creation

After clicking the button displayed in blue, the user will be asked to confirm the creation of the project as shown in Figure 7.3.

7.1.2 Regex



Configurar Regex

The Sixth Project

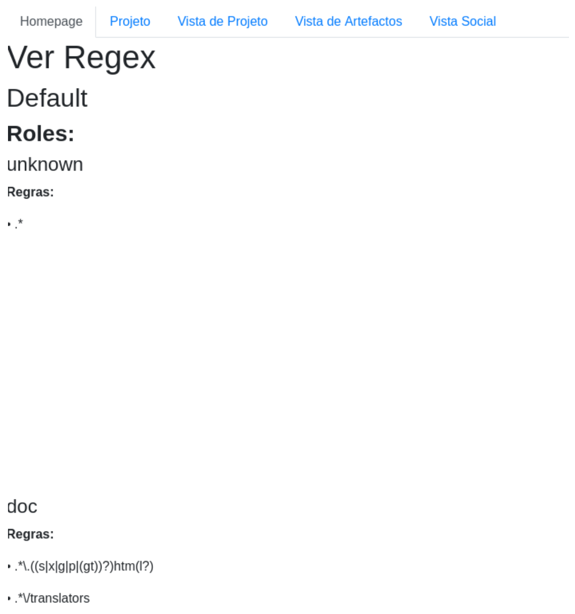
Configurações Regex já disponíveis:

- Default | Ver Regex | Apagar Regex

Criar Nova Configuração

Figure 7.4: Configure Regex

After successfully creating a project, the user will be redirected to the Configure Regex page as shown in 7.4. The Default Regex Configuration is already created and is immediately available to use. A user can view a Regex Configuration (7.5), which will display the development activities considered along with the rules that were applied, delete a configuration and create a new one (7.6).



Ver Regex

Default

Roles:

unknown

Regras:

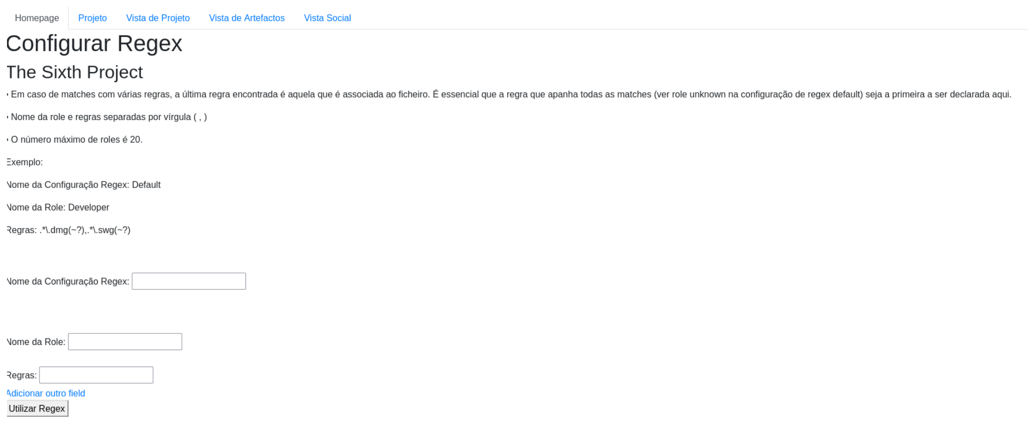
- .*

doc

Regras:

- .*\.((s|x|g|p|(gt))?)htm(l?)
- .*\/translators

Figure 7.5: View Regex



Configurar Regex

The Sixth Project

- Em caso de matches com várias regras, a última regra encontrada é aquela que é associada ao ficheiro. É essencial que a regra que apanha todas as matches (ver role unknown na configuração de regex default) seja a primeira a ser declarada aqui.
- Nome da role e regras separadas por vírgula (,)
- O número máximo de roles é 20.

Exemplo:

Nome da Configuração Regex: Default

Nome da Role: Developer

Regras: .*\.dmg(~?),.*\.swg(~?)

Nome da Configuração Regex:

Nome da Role:

Regras:

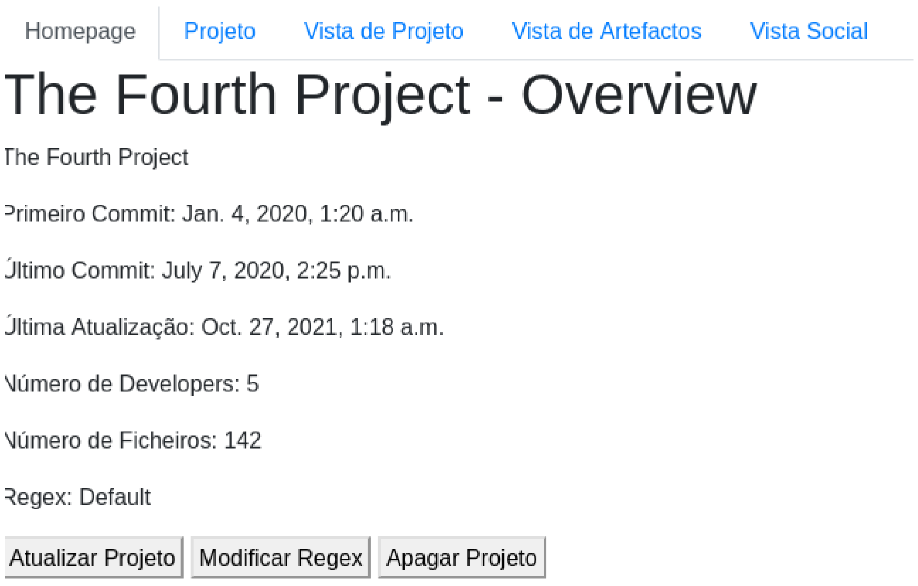
Figure 7.6: Create New Regex

Figure 7.6 presents the page to create a new configuration and explains to the user some concepts to keep in mind when creating custom configurations such as:

- In case of a tie in activities, the ProjectScanner considers the last found activity for the categorization of that file;
- It is extremely important that the standard activity that catches any remaining files that could not be categorized by the all the other activities is created last (in the default configuration, this activity is named unknown);
- The maximum amount of activities of a Regex Configuration is twenty.

The user will be required to input a name to the Regex Configuration and assign any number of development activities (named Roles on the creation page) along with the desired regular expressions to apply.

7.2 Project



The Fourth Project - Overview

The Fourth Project

Primeiro Commit: Jan. 4, 2020, 1:20 a.m.

Último Commit: July 7, 2020, 2:25 p.m.

Última Atualização: Oct. 27, 2021, 1:18 a.m.

Número de Developers: 5

Número de Ficheiros: 142

Regex: Default

Figure 7.7: Project Information

After creating a project and selecting an available Regex Configuration the user will be redirected to the Project page, the user can also simply click on the "Projeto" button in the Navigation Bar to be redirected here. On this page, the following information about the project is displayed:

- Date of the first commit;
- Date of the last commit;
- Date of the last update to the ProjectScanner's logs of this project;
- Number of developers;
- Number of files;
- The Regex Configuration currently applied to the project.

The user can also update the project, modify the current Regex Configuration being applied (which will redirect the user to 7.4) and delete the project.

7.3 Project View

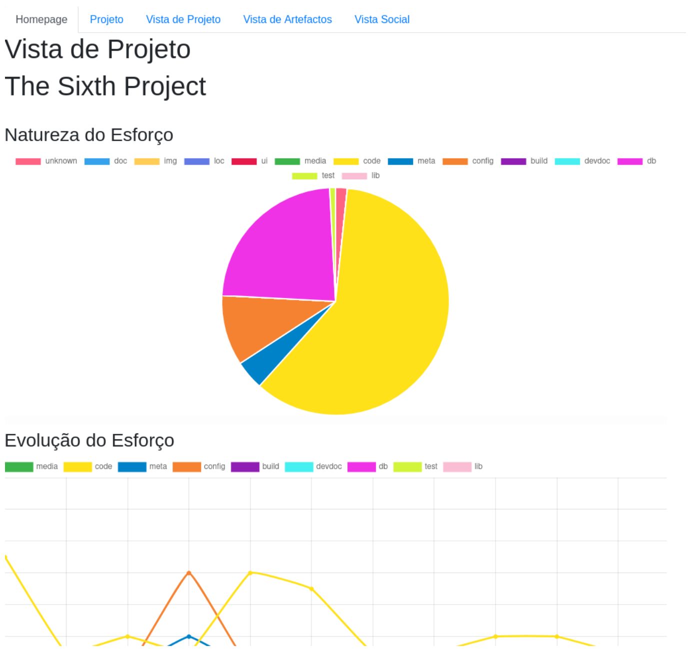


Figure 7.8: Project View

Figure 7.8 showcases the Project View page, this page is available at any time by clicking the "Vista de Projeto" button on the Navigation Bar. This page displays the Nature of the Effort and Evolution of the Effort on a project scale.

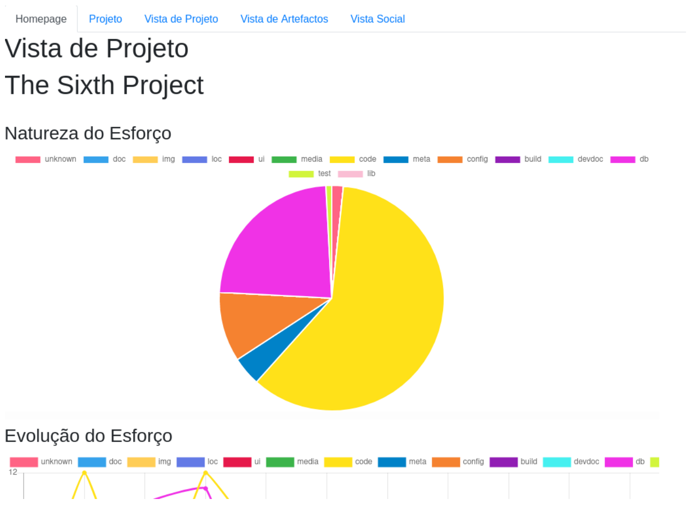


Figure 7.9: Project View - Nature of the Effort

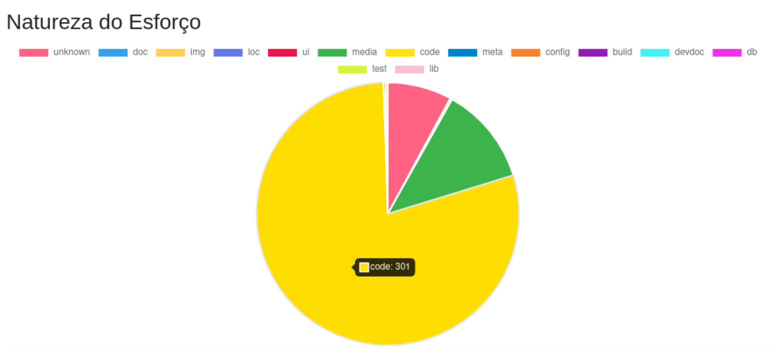


Figure 7.10: Project View - Nature of the Effort (Tooltip)

Figure 7.9 and 7.10 present the Nature of the Effort applied to each development activity by every member of the project.

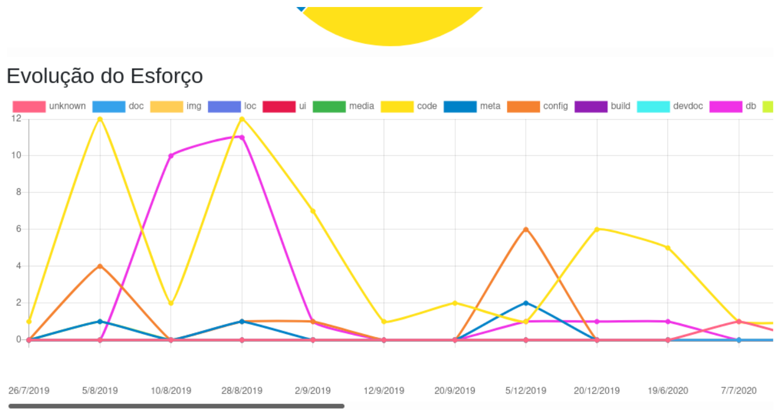


Figure 7.11: Project View - Evolution of the Effort

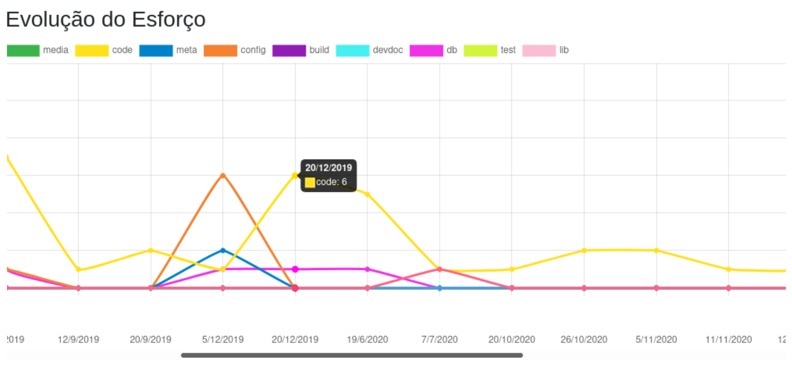
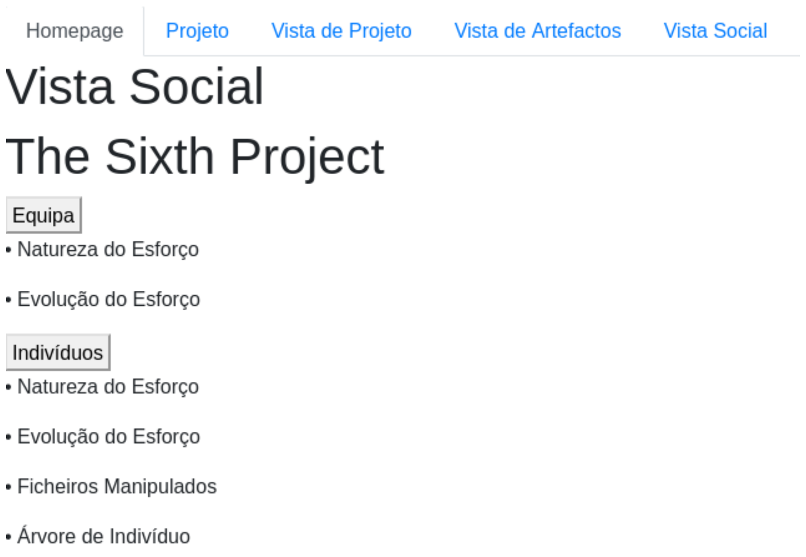


Figure 7.12: Project View - Evolution of the Effort (Tooltip)

Figure 7.11 and 7.12 feature the Evolution of the Effort, showcasing how each activity progressed throughout the development of the project. The user can scroll horizontally through the chart to visualize the remaining graph if needed. It is also possible to get increased information by hovering over any element in the chart, as shown in 7.12.

7.4 Social View



Vista Social

The Sixth Project

- Natureza do Esforço
- Evolução do Esforço

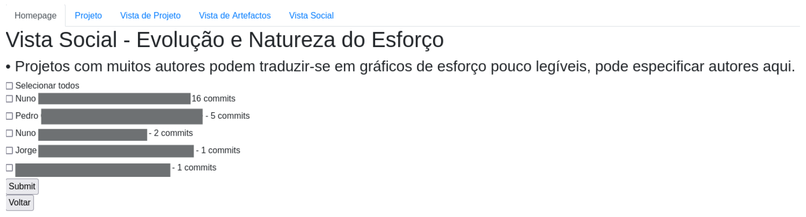
- Natureza do Esforço
- Evolução do Esforço
- Ficheiros Manipulados
- Árvore de Indivíduo

Figure 7.13: Social View

Figure 7.13 displays the Social View page of the ProjectScanner. The user can access this page at any time by clicking the "Vista Social" button on the Navigation Bar. Here, the user can select one of two options:

- **Team:** Displayed in 7.13 as "Equipa" will allow the user to consult the Nature of the Effort and Evolution of the Effort of any number of team members.
- **Individuals:** Displayed in 7.13 as "Indivíduos" will allow the user to consult the Nature of the Effort, Evolution of the Effort, Manipulated Files and the Individual File Directory Tree of a specific team member.

7.4.1 Team



Vista Social - Evolução e Natureza do Esforço

- Projetos com muitos autores podem traduzir-se em gráficos de esforço pouco legíveis, pode especificar autores aqui.

☐ Selecionar todos
☐ Nuno [redacted] 16 commits
☐ Pedro [redacted] - 5 commits
☐ Nuno [redacted] - 2 commits
☐ Jorge [redacted] - 1 commits
☐ [redacted] - 1 commits

Figure 7.14: Social View Team - Select Team Members

Figure 7.14 displays the page following the click of the "Equipa" button shown in 7.13. Here, the user will be provided with a list of all the authors who contributed to the project along with the number of commits they pushed into the repository. The user can select any number of members available from this page.

We considered the inclusion of this filter extremely important, as during testing we would find that projects with many members would generate extremely convoluted charts that could be hard to effectively visualise. When using open source repositories this was even more evident, as the charts would display a vast amount of authors that had minimal contributions.

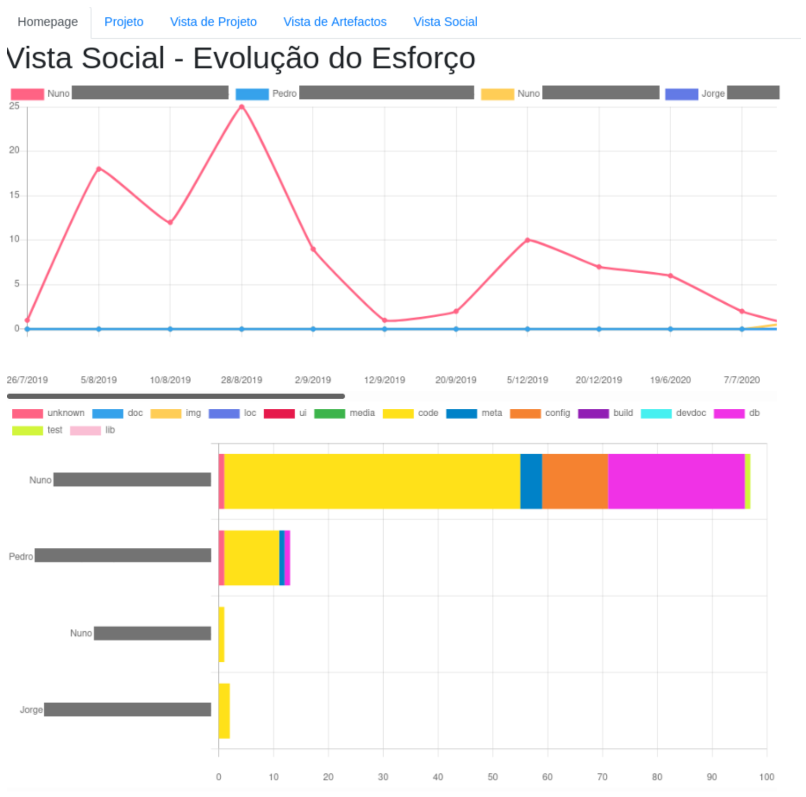


Figure 7.15: Social View Team - Nature and Evolution of the Effort

Figure 7.15 showcases the Evolution of the Effort and Nature of the Effort respectively for the selected team members. The Line Chart features the different team members and the evolution of their effort across the development. The Nature of the Effort presents the selected team members and relates each commit change done by that member to the different development activities established. The user can hover over any of the charts to look at concrete values via the available tool tip.

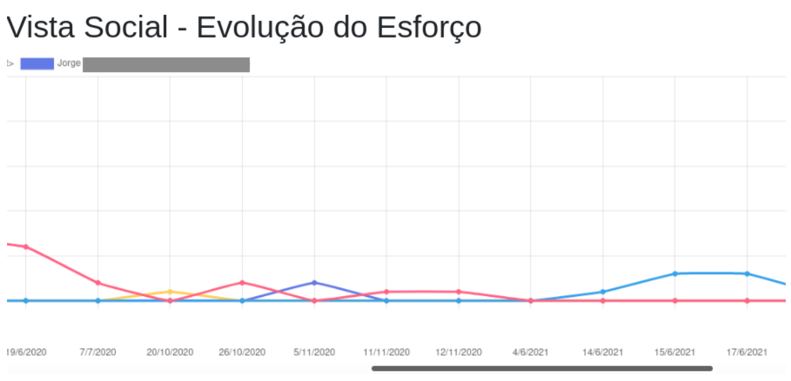
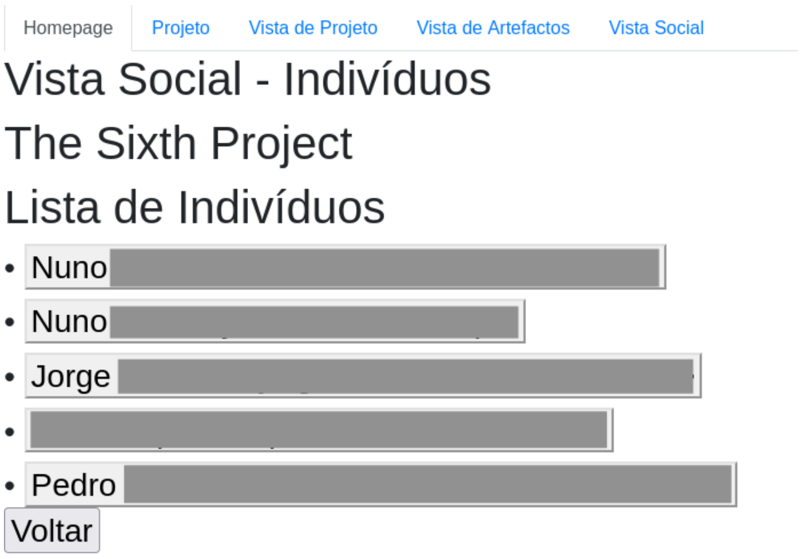


Figure 7.16: Social View Team - Evolution of the Effort (Scroll)

7.4.2 Individuals



Vista Social - Indivíduos

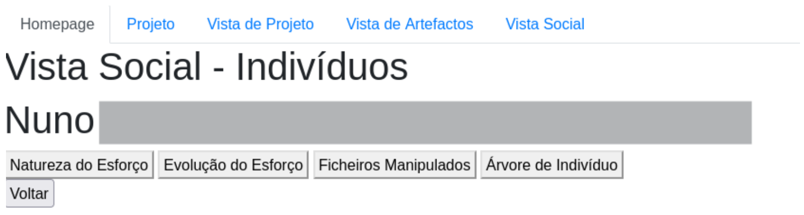
The Sixth Project

Lista de Indivíduos

- Nuno [redacted]
- Nuno [redacted]
- Jorge [redacted]
- [redacted]
- Pedro [redacted]

Figure 7.17: Social View Individual - Select Team Member

Figure 7.17 displays the page following the click of the "Indivíduo" button shown in 7.13. Here, the user will be provided with a list of all the authors who contributed to the project. The user can then select one of the members from this page.



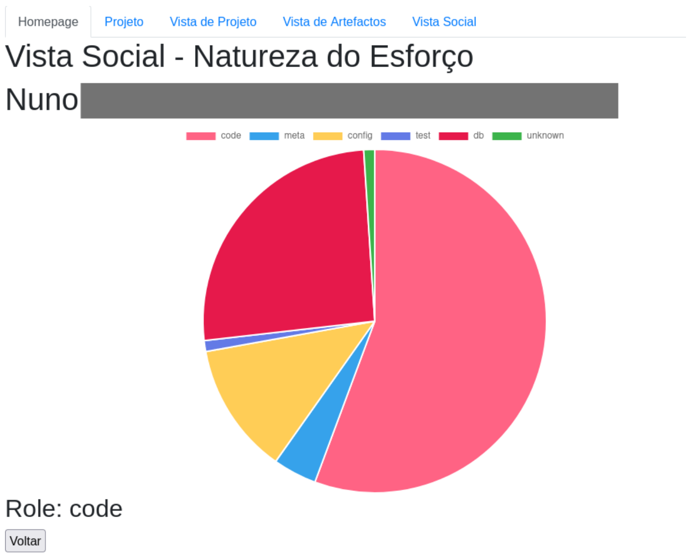
Vista Social - Indivíduos

Nuno [redacted]

Figure 7.18: Social View Individual

From 7.18, the user can consult the Nature of the Effort, Evolution of the Effort, Manipulated Files and the File Directory Tree on an individual scope.

Nature of the Effort



Vista Social - Natureza do Esforço

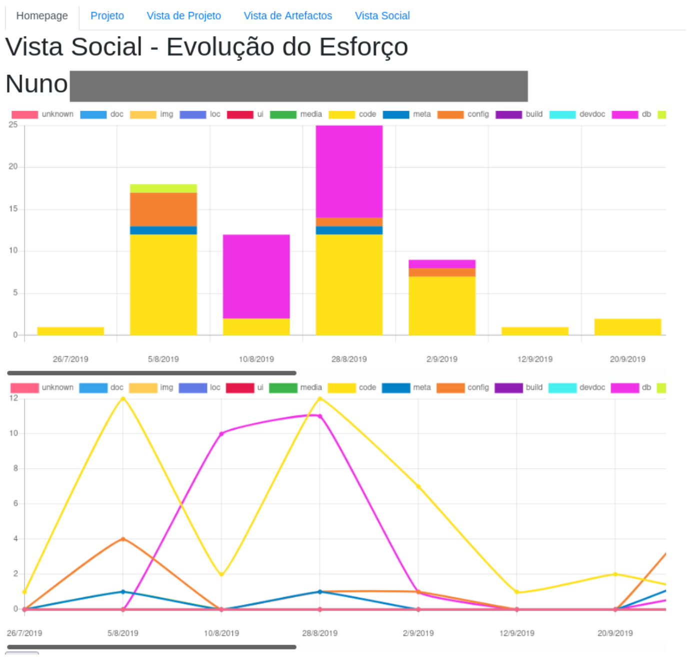
Nuno [redacted]

Role: code

Figure 7.19: Social View Individual - Nature of the Effort

Figure 7.20 displays the Nature of the Effort related to the specific individual. This effort is categorized by the different development activities the author was involved with. Below the Pie chart, a role is attributed to the author based on the activity the user had the most effort in.

Evolution of the Effort



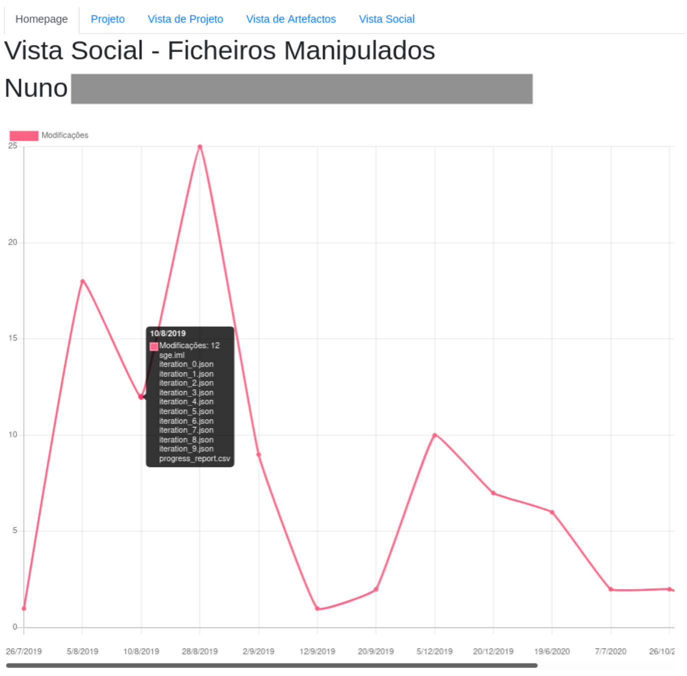
Vista Social - Evolução do Esforço

Nuno [redacted]

Figure 7.20: Social View Individual - Evolution of Effort

On the page shown in 7.20 the Evolution of the Effort is displayed in two different ways. The first one is the stacked bar chart, relating the changes the specific user pushed to the repository and the categorized development activities. The Line chart below it displays the same information with each line representing the different development activities.

Manipulated Files

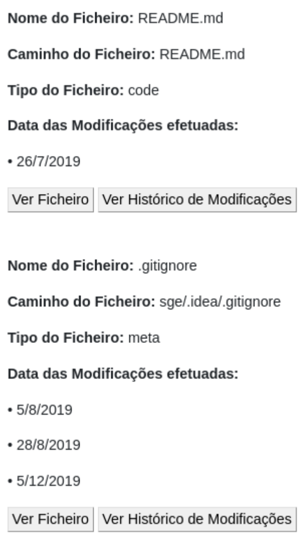


Vista Social - Ficheiros Manipulados

Nuno [redacted]

10/8/2019
🟥 Modificações: 12
sge.iml
iteration_0.json
iteration_1.json
iteration_2.json
iteration_3.json
iteration_4.json
iteration_5.json
iteration_6.json
iteration_7.json
iteration_8.json
iteration_9.json
progress_report.csv

Figure 7.21: Social View Individual - Modified Files



Nome do Ficheiro: README.md

Caminho do Ficheiro: README.md

Tipo do Ficheiro: code

Data das Modificações efetuadas:

• 26/7/2019

Nome do Ficheiro: .gitignore

Caminho do Ficheiro: sge/.idea/.gitignore

Tipo do Ficheiro: meta

Data das Modificações efetuadas:

• 5/8/2019

• 28/8/2019

• 5/12/2019

Figure 7.22: Social View Individual - Modified Files

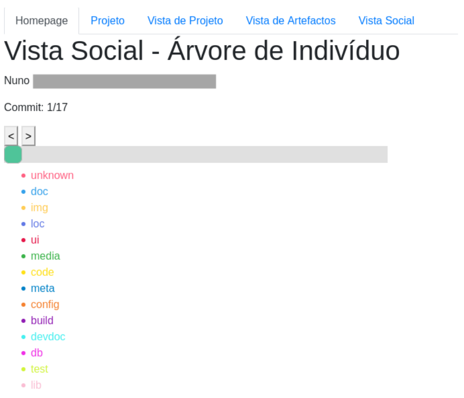
Figure 7.22 and 7.22 present the Manipulated Files page of a specific author. Here, the Line Chart displays the different modifications the author did on a set of files through the project's development. The user can hover any specific point in the chart to get the list of files that were changed on that specific date.

Below the Line Chart, extra information about the files that member worked on is presented:

- Name of the file;
- Full path of the file;
- Development activity applied to the file;
- The date of every modification that was done to the file by the specified member;

This page also allows the user to consult the View File and the Modification History pages. These pages will be shown in 7.6

File Directory Tree


Homepage | Projeto | Vista de Projeto | Vista de Artefactos | Vista Social

Vista Social - Árvore de Indivíduo

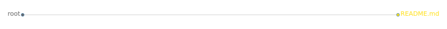
Nuno [redacted]

Commit: 1/17

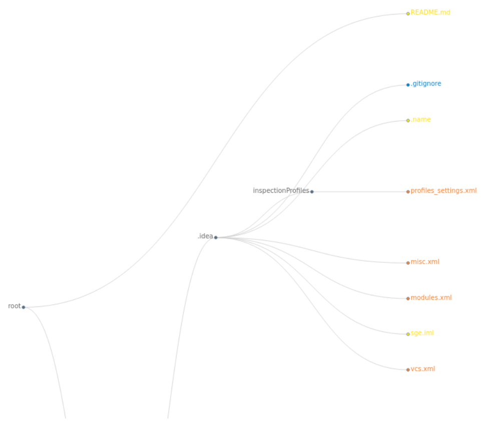
< >

- ● unknown
- ● doc
- ● img
- ● loc
- ● ui
- ● media
- ● code
- ● meta
- ● config
- ● build
- ● devdoc
- ● db
- ● test
- ● lib

Figure 7.23: Social View Individual - File Directory Tree



(a) File Directory Tree after the first commit



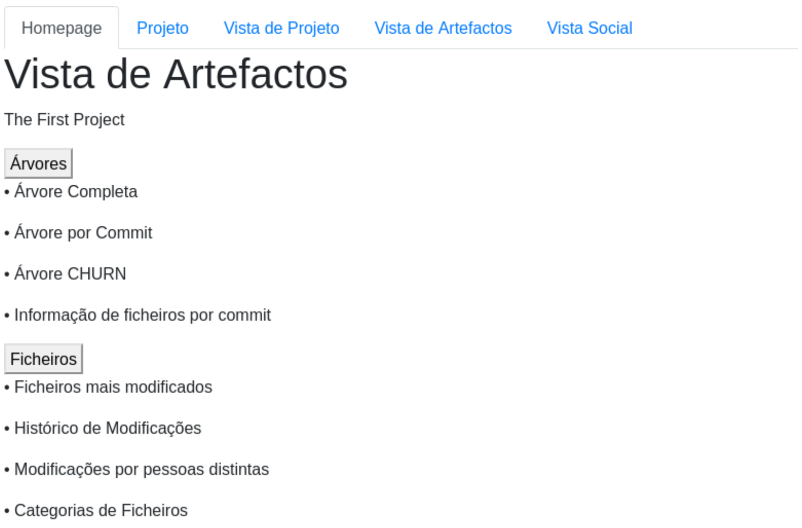
(b) File Directory Tree after the third commit

Figure 7.24: File Directory Tree of the selected team member

Figure 7.23 and 7.24 displays the different components of the Social View - File Directory Tree page. Figure 7.23 contains a legend that attributes a color to different software development activities. The user is also able to view the File Directory Tree at different commits pushed by the specified team member. The buttons or the slider can be used to advance commits and visualize the changes in the tree.

Figure 7.24 contains two examples of the presentation of the File Directory Tree after one and two commits respectively. The user can also click on any of colored files to be redirected to that file's View File page (7.6).

7.5 Artifact View



Homepage | Projeto | Vista de Projeto | Vista de Artefactos | Vista Social

Vista de Artefactos

The First Project

Árvores
- Árvore Completa
- Árvore por Commit
- Árvore CHURN
- Informação de ficheiros por commit

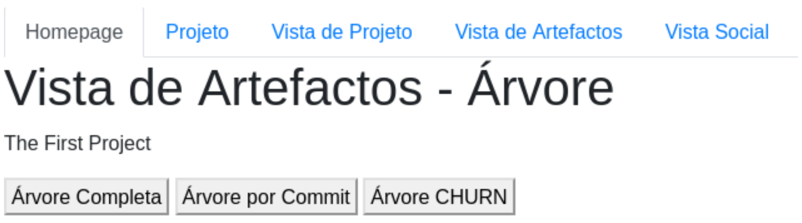
Ficheiros
- Ficheiros mais modificados
- Histórico de Modificações
- Modificações por pessoas distintas
- Categorias de Ficheiros

Figure 7.25: Artifact View Individual

Figure 7.25 displays the Social View page of the ProjectScanner. The user can access this page at by clicking the "Vista de Artefacto" button on the Navigation Bar. Here, the user can select one of two options:

- **Trees:** Displayed in 7.25 as "Árvores" will allow the user to consult the File Directory Tree, File Directory Tree per Commit and the CHURN File Directory Tree.
- **Files:** Displayed in 7.25 as "Ficheiros" will allow the user to consult Most Modified Files, Modified Files by the Most People and Activities of the Files.

7.5.1 File Directory Tree



Homepage | Projeto | Vista de Projeto | Vista de Artefactos | Vista Social

Vista de Artefactos - Árvore

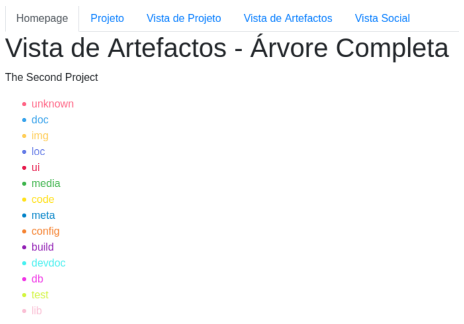
The First Project

Árvore Completa | Árvore por Commit | Árvore CHURN

Figure 7.26: Artifact View - Trees

From 7.26, the user can consult the File Directory Tree, File Directory Tree per Commit and the CHURN File Directory Tree on an project scope.

File Directory Tree


Homepage Projeto Vista de Projeto Vista de Artefactos Vista Social

Vista de Artefactos - Árvore Completa

The Second Project

- unknown
- doc
- img
- loc
- ui
- media
- code
- meta
- config
- build
- devdoc
- db
- test
- lib

Figure 7.27: Artifact View - Trees

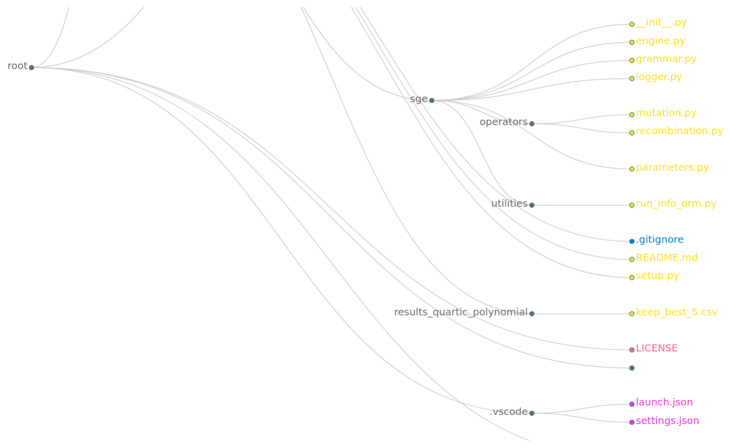
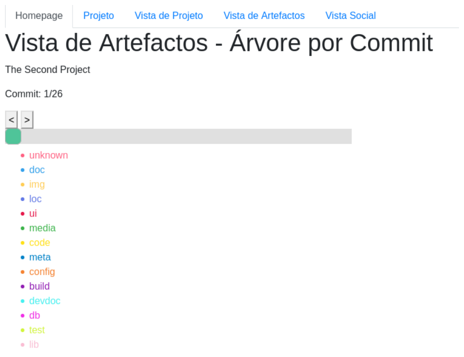


Figure 7.28: Artifact View - Snippet of the File Directory Tree

Figure 7.27 and 7.28 displays the different components of the Artifact View - File Directory Tree page. Figure 7.27 contains a legend that attributes a color to different software development activities.

Figure 7.28 contains a snippet of the File Directory Commit after the last commit pushed to the repository. The user can also click on any of colored files to be redirected to that file's View File page (7.6).

File Directory Tree per Commit


Homepage Projeto Vista de Projeto Vista de Artefactos Vista Social

Vista de Artefactos - Árvore por Commit

The Second Project

Commit: 1/26

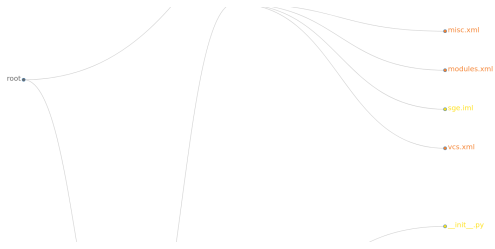
< >

- unknown
- doc
- img
- loc
- ui
- media
- code
- meta
- config
- build
- devdoc
- db
- test
- lib

Figure 7.29: Artifact View - File Directory Tree per Commit

root
README.md

(a) File Directory Tree after the first commit



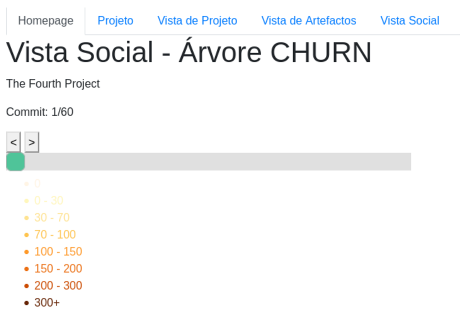
(b) File Directory Tree after the third commit

Figure 7.30: File Directory Tree of the Project

Figure 7.29 and 7.30 display the different components of the Artifact View - File Directory Tree page. Figure 7.29 contains a legend that attributes a color to different software development activities. The user is also able to view the File Directory Tree at different commits pushed to the repository. The buttons or the slider can be used to advance commits and visualize the changes in the tree.

Figure 7.30 contains two examples of the presentation of the File Directory Tree after one and three commits respectively. The user can also click on any of colored files to be redirected to that file's View File page (7.6).

CHURN File Directory Tree



Vista Social - Árvore CHURN

The Fourth Project

Commit: 1/60

- 0
- 0 - 30
- 30 - 70
- 70 - 100
- 100 - 150
- 150 - 200
- 200 - 300
- 300+

Figure 7.31: Artifact View - CHURN File Directory Tree

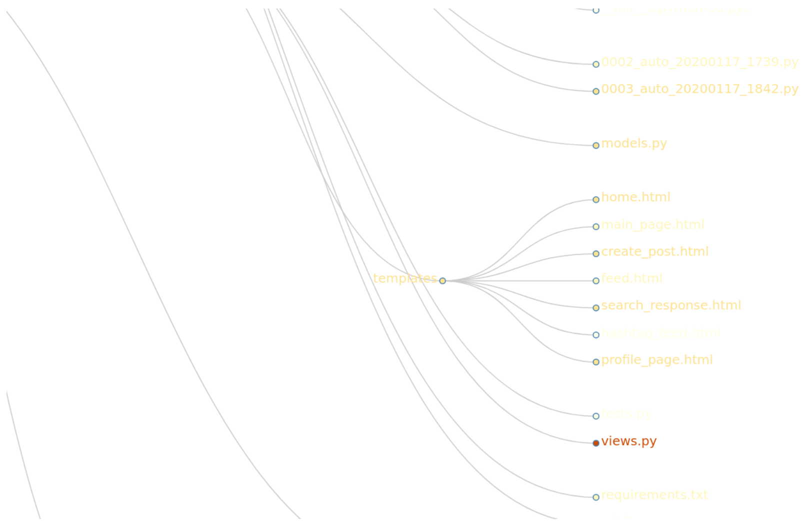


Figure 7.32: Artifact View - CHURN File Directory Tree

Figure 7.31 and 7.31 display the components of the Artifact View - CHURN File Directory Tree page. Figure 7.31 contains a legend that attributes a color to total number of lines added and removed. The user is also able to view the CHURN File Directory Tree at different commits pushed by every contributor to the project. The buttons or the slider can be used to advance commits and visualize the changes in the tree.

Figure 7.32 contains a snippet of a CHURN File Directory Tree. Here, the files present in the tree are colored according to the number of lines added and removed at that specific instance and the legend in 7.31.

7.5.2 Files

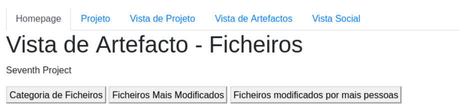
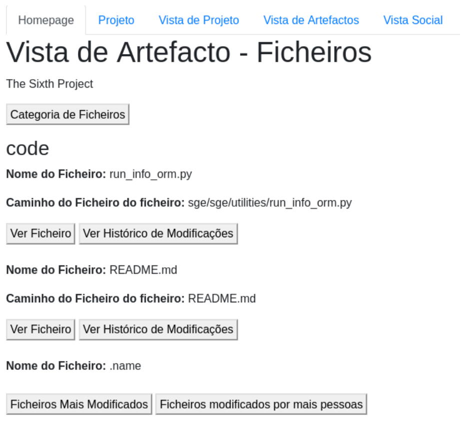


Figure 7.33: Artifact View - Files

Figure 7.33 presents the Artifact View - Files pages following the click on the "Ficheiros" button in 7.25.

Here, the user can consult the Activities of the Files, Most Modified Files and Modified Files by the Most people by clicking the shown buttons respectively. The user can click any of the button to show/hide the corresponding information about the files. From any of these buttons, the user will also be able to consult the View Page and Modification History of any of the shown files (7.6).


Homepage
Projeto
Vista de Projeto
Vista de Artefactos
Vista Social

Vista de Artefacto - Ficheiros

The Sixth Project

code

Nome do Ficheiro: run_info_orm.py

Caminho do Ficheiro do ficheiro: sge/sge/utilities/run_info_orm.py

Ver Ficheiro
Ver Histórico de Modificações

Nome do Ficheiro: README.md

Caminho do Ficheiro do ficheiro: README.md

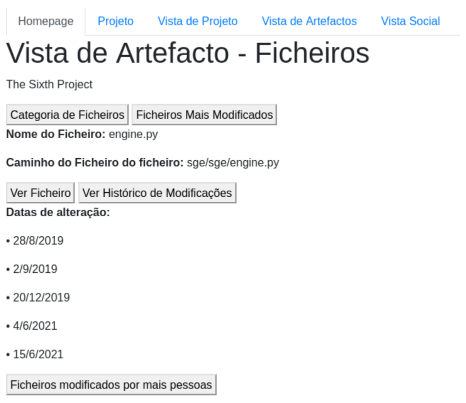
Ver Ficheiro
Ver Histórico de Modificações

Nome do Ficheiro: .name

Ficheiros Mais Modificados
Ficheiros modificados por mais pessoas

Figure 7.34: Artifact View - Files - Activities of the Files

Figure 7.34 presents the contents of the page after clicking the "Categoria de Ficheiros" button, corresponding to the Activities of the Files. The many files available in the repository will be shown sorted by the development activities established by the current Regex.


Homepage | Projeto | Vista de Projeto | Vista de Artefactos | Vista Social

Vista de Artefacto - Ficheiros

The Sixth Project

Categoria de Ficheiros | Ficheiros Mais Modificados

Nome do Ficheiro: engine.py

Caminho do Ficheiro do ficheiro: sge/sge/engine.py

Ver Ficheiro | Ver Histórico de Modificações

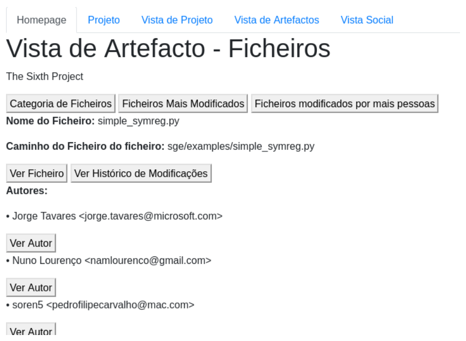
Datas de alteração:

- 28/8/2019
- 2/9/2019
- 20/12/2019
- 4/6/2021
- 15/6/2021

Ficheiros modificados por mais pessoas

Figure 7.35: Artifact View - Files - Most Modified Files

Figure 7.35 showcases the page after clicking the "Ficheiros Mais Modificados" button, corresponding to the Most Modified Files. Here, the files are sorted in descending order according to the number of modifications.


Homepage | Projeto | Vista de Projeto | Vista de Artefactos | Vista Social

Vista de Artefacto - Ficheiros

The Sixth Project

Categoria de Ficheiros | Ficheiros Mais Modificados | Ficheiros modificados por mais pessoas

Nome do Ficheiro: simple_symreg.py

Caminho do Ficheiro do ficheiro: sge/examples/simple_symreg.py

Ver Ficheiro | Ver Histórico de Modificações

Autores:

- Jorge Tavares <jorge.tavares@microsoft.com>

Ver Autor

- Nuno Lourenço <namlourenco@gmail.com>

Ver Autor

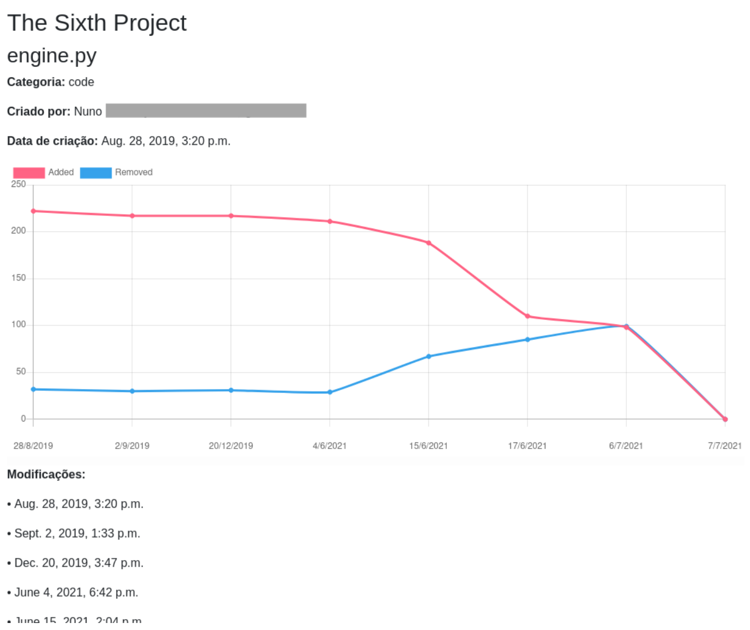
- soren5 <pedrofilipecarvalho@mac.com>

Ver Autor

Figure 7.36: Artifact View - Files - Modified Files by the Most People

Figure 7.36 showcases the page after clicking the "Ficheiros modificados por mais pessoas" button, corresponding to the Modified Files by Most People. Here, the files are sorted in descending order according to the number of people that changed the file.

7.6 View File and View Modification History



The Sixth Project

engine.py

Categoria: code

Criado por: Nuno [redacted]

Data de criação: Aug. 28, 2019, 3:20 p.m.

Modificações:

• Aug. 28, 2019, 3:20 p.m.

• Sept. 2, 2019, 1:33 p.m.

• Dec. 20, 2019, 3:47 p.m.

• June 4, 2021, 6:42 p.m.

• June 15, 2021, 2:04 p.m.

Figure 7.37: View File

Figure 7.38 features the View File page after clicking the "Ver Ficheiro" button from either 7.4.2 or any of the File Directory Tree pages. Here, the user can visualize information about the selected file. This page displays:

- Name of the file;
- The activity associated with the file;
- Date of creation;
- A line chart displaying the evolution of the file according to the number of added and removed lines;
- The date of every modification done to the file;

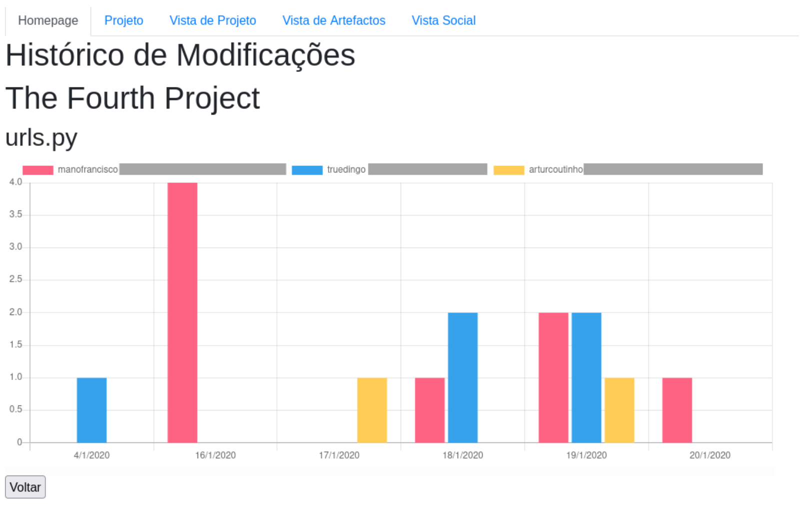


Figure 7.38: Modification History

Figure 7.38 features the Modification History of the selected file. Here, the user can visualise the different contributions of every member involved in the project related to the selected file across the whole development.

Chapter 8

Conclusion and Further Work

With the present thesis, we had the objective of encapsulating the process that went into the development of the ProjectScanner and other topics that we deemed relevant and that resulted directly from the approach we took into the development.

As a starting point for the thesis, the ProjectScanner began as a tool that had an increased focus on GitLab and explored Process Mining. However, as our understanding of Process Mining evolved as a direct result of research, we realized that Mining Software Repositories worked in very close proximity to Process Mining and could be considered groundwork for the implementation of process mining techniques. The correct parsing of the vast event data available from software repositories is one of the mandatory requisites for Process Mining techniques to be applied. With this in mind, the decision to focus on the field of Mining Software Repositories seemed logical, the data would need to be parsed to fit the Process Mining criteria and the exploration of the directly available data from the logs made sense from a perspective of scaling the tool.

ProjectScanner evolved into a tool that represents the perfect environment to explore Process Mining techniques in the future while still being useful on its own by presenting tangible results that can be applied to development teams and help in understanding, monitoring, and improving development processes.

The heavily exploratory nature of this thesis did mean that some backtracking was mandatory after the work into Process Mining was completed for the first semester. However, we ultimately found that most decisions that were previously established only needed some minor adjustments, which solidifies the relationship between the different focus points of our thesis.

The first semester was largely focused on understanding Process Mining and its applications. We deemed this a crucial step to grasp its usefulness in aiding the improvement of software development processes. After the groundwork, framework and language decisions were established and a study into the necessary development tools was produced. An initial version of the system architecture was built to define our planned structure and establish how our system interacted with the different components.

For the second semester, we focused on revising some of the previously established concepts as our understanding of the nature of the tool we wanted to develop deepened. As mentioned earlier, not many changes were made and many of the groundwork established in the first semester remained unchanged. The new additions had an increased focus on being beneficial to the current route of the ProjectScanner and also aiding in the exploration of our previous topic of focus in the future. The first step into the actual development focused on selecting the required tools for data extraction from software repositories. After this, we focused on understanding this data and what could be drawn from it, this allowed us to dynamically construct our set of requirements as the exploration occurred and relationships were drawn between the work that we were doing and the research that had been done in advance.

After the exploration phase, we were able to identify and build our set of functional and non-functional requirements for the project. User Stories began as a simple way to put into writing the work done in the Exploration Phase(2) and the results from the different brainstorming meetings. These User Stories were then refined into proper Use Cases after establishing their relevancy for the Project and sorted into Functional Requirements where a priority was assigned for each requirement according to their importance to the project. The Mockups and Navigation Diagram were designed as means to understand the different interactions of the ProjectScanner and how these relate with the established requirements. The entity-relationship diagram was created, showcasing the different relationships of our data structure. The System Architecture was revised from the previous first semester version due to the exploratory nature of our work and the newly established nature of our tool. After each development cycle was completed, we had implemented every *Must Have*, *Should Have* requirement and two out of the three *Could Have* requirements, with the one missing being related to Process Mining. We considered this development more than satisfactory for the completion of the MVP(Minimum Value Product) of the ProjectScanner.

In regards to future work, we discussed some possibilities that would further improve the ProjectScanner. The possibility of switching between an English and Portuguese interface, which would widen the range of use of our tool and provide more accessibility for non-Portuguese speakers in case the tool finds use outside internal applications. A more diverse range of timestamps such as days, weeks, and months and the ability to filter through a user input date or a commit message can also be interesting additions. Establishing project KPIs like ActiVCS (2.2.4) implements can provide increased insight by comparing the different KPIs across projects. A Compare View could be developed that showcases the different KPI values for the uploaded projects in the tool. A simple and efficient design can also improve the usability of the ProjectScanner as not much work went into the actual design of the tool outside some mandatory points related to the many charts of the project and the File Directory Tree. Finally, and one of the points we consider the most important is applying Process Mining techniques to the data, this has been a topic of discussion multiple times throughout development and many of our decisions were based on this inclusion. ProjectScanner provides a perfect environment by providing data that can easily be transformed into a compatible dataset for process mining algorithms, these algorithms can be used to further improve, discover development processes and enrich our tool.

References

[1] Atlassian. Webhooks. https://developer.atlassian.com/server/jira/platform/webhooks/. Access: January 2021.

[2] Atlassian. What is jira used for? https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for#Jira-for-requirements-&-test-case-management. Access: January 2021.

[3] Vangie Beal. Webopedia - repository. https://www.webopedia.com/definitions/repository/. Access: September 2021.

[4] The Django Book. "workflow mining: Discovering process models from event logs." ieee transactions on knowledge and data engineering 16, no. 9. https://doi.org/10.1109/TKDE.2004.47, 2004.

[5] Celonis. What is process mining? https://www.celonis.com/process-mining/what-is-process-mining. Access: January 2021.

[6] Chart.js. Chart.js. https://www.chartjs.org/.

[7] Thomas S.W. & Hassan Chen, TH. A survey on the use of topic models when mining software repositories. https://doi.org/10.1007/s10664-015-9402-8, 2016.

[8] D3.js. D3 data-driven documents. https://d3js.org/.

[9] Wil M.P. van der Aalst Dirk Fahland. Simplifying discovered process models in a controlled manner. http://www.padsweb.rwth-aachen.de/wvdaalst/publications/p716.pdf, 2012. Access: January 2021.

[10] MDM Web Docs. Django introduction. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. Access: January 2021.

[11] H. Gall, K. Hajek, and M. Jazayeri. Proceedings. international conference on software maintenance (cat. no. 98cb36272). https://doi.org/10.1109/ICSM.1998.738508, 1998.

[12] D3.js Graph Gallery. D3.js graph gallery - dendogram. https://www.d3-graph-gallery.com/dendrogram.

[13] Gartner. Business process management (bpm) definition. https://www.gartner.com/en/information-technology/glossary/business-process-management-bpm. Access: January 2021.

[14] GitLab. Gitlab devops lifecycle. https://about.gitlab.com/stages-devops-lifecycle/. Access: January 2021.

[15] Gitpy. Gitpy. https://pypi.org/project/gitpy/.

[16] Ahmed E. Hassan. The road ahead for mining software repositories. https://www.researchgate.net/publication/264799710_The_Road_Ahead_for_Mining_Software_Repositories, 2008.

[17] JIRA. Jira. https://www.atlassian.com/software/jira. Access: January 2021.

[18] KC Karnes. What are webhooks? and why should you get hooked? https://clevertap.com/blog/what-are-webhooks/, 2020. Access: January 2021.

[19] Klipfolio. What is a kpi? https://www.klipfolio.com/resources/articles/what-is-a-key-performance-indicator.

[20] Phil Leggetter. What are webhooks and how do they enable a real-time web? https://www.programmableweb.com/news/what-are-webhooks-and-how-do-they-enable-real-time-web/2012/01/30, 2012. Access: January 2021.

[21] Eryk Lewinson. Introduction to process mining. https://towardsdatascience.com/introduction-to-process-mining-5f4ce985b7e5, 2020. Access: January 2021.

[22] Horia Ciocarlie Maria Laura Sebu. Applied process mining in software development. https://www.researchgate.net/publication/269301615_Applied_process_mining_in_software_development, 2014. Access: January 2021.

[23] MSRConf. Mining software repositories. https://www.msrconf.org/. Access: September 2021.

[24] Ngrok. Ngrok. https://ngrok.com/product. Access: January 2021.

[25] PM4Py. Implemented approaches - scientific work implemented in pm4py. https://pm4py.fit.fraunhofer.de/implemented-approaches. Access: January 2021.

[26] Paul Kneringer Saimir Bala and Jan Mendling. Discovering activities in software development processes. http://ceur-ws.org/Vol-2793/paper6.pdf, 2020.

[27] Tamanna Siddiqui and Ausaf Ahmad. Data mining tools and techniques for mining software repositories: A systematic review. https://www.researchgate.net/publication/320213363_Data_Mining_Tools_and_Techniques_for_Mining_Software_Repositories_A_Systematic_Review, 2018.

[28] Walter Tichy. An interview with prof. andreas zeller: Mining your way to software reliability. https://doi.org/10.1145/1880066.1883621, 2010.

[29] Olivier Vandecruys, David Martens, Bart Baesens, Christophe Mues, Manu De Backer, and Raf Haesen. Mining software repositories for comprehensible software fault prediction models. https://www.semanticscholar.org/paper/Mining-software-repositories-for-comprehensible-Vandecruys-Martens/b264d893c37494d61af0b7636fe29f64df4183b5, 2008.

[30] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. On the variation and specialisation of workload-a case study of the gnome ecosystem community. https://doi.org/10.1007/s10664-013-9244-1, 2014.

[31] Wikipedia. Bug tracking system. https://en.wikipedia.org/wiki/Bug_tracking_system. Access: September 2021.

[32] Wikipedia. Gitlab - wikipedia. https://en.wikipedia.org/wiki/GitLab. Access: January 2021.

[33] Wikipedia. Microsoft teams. https://en.wikipedia.org/wiki/Microsoft_Teams. Access: September 2021.

[34] Wikipedia. Repository (version control). https://en.wikipedia.org/wiki/Repository_(version_control). Access: September 2021.

[35] Wikipedia. Software repository. https://en.wikipedia.org/wiki/Software_repository. Access: September 2021.

Appendices

This page is intentionally left blank.

Appendix A

User Stories related with the ProjectScanner:

- "As a user, I want to be able to upload my git repository so that I can visualize information about it."
- "As a user, I want to be able to be able to immediately use the ProjectScanner even if I have no regex configuration so that I can effortlessly visualise information."
- "As a user, I want to be able to configure the used regex of my Project so that the information displayed can more accurately represent the reality of my project."
- "As a user, I want to be able to upload multiple projects unto the platform so that I can monitor multiple projects at once."
- "As a user, I want to be able to update my project with the most recent commits at any time so that I don't have to upload the project again."
- "As a user, I want to be able to easily see information such as number of files, number of developers involved and activities assessed by the regex so that I can better understand the initial state of the repository within the platform."
- "As a user, I want the information the tool can provide to be easily accessed so that I know without much effort the available functionalities."

User Stories related to the Nature of the Effort Applied

- "As a user, I want to be able to visualize the overall nature of the effort of the project, divided by the categories identified by the regex so that I can have a better understanding of where that effort is being applied."
- "As a user, I want to be able to visualize the overall nature of the effort of the project across time, sorted by the categories identified by the regex so that I can have a better understanding of when that effort was being applied."
- "As a user, I want to be able to visualize the nature of the effort of the team members, sorted by the categories identified by the regex so that I can have a better understanding of where that effort is being applied per team member."
- "As a user, I want to be able to visualize the nature of the effort of the team members across time, divided by the categories identified by the regex so that I can have a better understanding of when that effort was being applied per team member."
- "As a user, I want to be able to visualize the nature of the effort of a specific team member within the categories identified by the regex so that I can have a better understanding of the work that team member did."
- "As a user, I want to be able to visualize the nature of the effort of a specific team member across time within the categories identified by the regex so that I can have a better understanding of how that effort was distributed across time."

User Stories related to Project Files

- "As a user, I want to be able to visualize a specific project file so that I can better understand its role within the project."
- "As a user, I want to be able to visualize the number of lines added and removed across time so that I can better understand the evolution of the file across time."
- "As a user, I want to be able to visualize who created a specific file, when that files was created and the number of modifications a file has so that I can better understand the role of a specific file within the project."
- "As a user, I want to be able to visualize the history of modifications so that I can understand which team members were working on a specific file across time."
- "As a user, I want to be able to visualize which project files a specific team member worked on so that I can have establish the focus of that specific team member's work."
- "As a user, I want to be able to visualise the project files sorted by the categories identified by the regex so that I can have a better understanding of the role of each file within the project."
- "As a user, I want to be able to visualise the most modified project files so that I can have better understanding of what files were the most worked on."
- "As a user, I want to be able to visualise the project files most modified by different team members so that I can have a better understanding of the project flow."

User Stories related to Directory Trees

- "As a user, I want to be able to visualize the complete directory tree of my project sorted by the categories identified by the regex so that I can better understand how which project file is connected to the overall project as well as their internal roles."
- "As a user, I want to be able to visualize the directory tree per commit, sorted by the categories identified by the regex so that I can have a better understanding of the evolution of the project as well as the internal roles of the files."
- "As a user, I want to be able to visualize the directory tree per commit with the corresponding heatmap of the files, also sorted by the categories identified by the regex so that I can have a better understanding of the evolution of the project, the internal roles of the files and the effort being applied to each file across the project cycle."
- "As a user, I want to be able to visualize the directory tree of a specific team member, sorted by the categories identified by the regex so that better understand the project state when a specific team member was working on the project."

Name	Select Regex
ID	FR03
Actor	User
Description	The user selects a regex configuration from the available ones and applies it to the project.
Trigger	1. a) Redirected after creating a project b) Clicking the "Modificar Regex" button located in that project's page.
Preconditions	1. There is an active project selected.
Basic Flow	1. User is in the Configure Regex Page 2. User selects and clicks one of the available Regexes.
Postconditions	Project is updated with the new Regex.
Alternative Flow	Regex can't be applied due to a system failure.

Table 1: Select Regex Use Case

Name	Delete Regex
ID	FR04
Actor	User
Description	The user selects a regex configuration from the available ones and deletes it.
Trigger	1. a) Redirected after creating a project b) Clicking the "Modificar Regex" button located in that project's page.
Preconditions	1. There is an active project selected.
Basic Flow	1. User is in the Configure Regex Page 2. User selects one of the available Regexes and clicks the "Apagar Regex" button.
Postconditions	Regex is deleted from the Database and from the active project.
Alternative Flow	3. Regex can't be deleted due to a system failure.

Table 2: Delete Regex Use Case

Name	Select a Project
ID	FR05
Actor	User
Description	The user selects a project making it the active project.
Trigger	1. Clicking the project name on the Homepage.
Preconditions	None
Basic Flow	1. User is in the Homepage 2. User selects one of the available projects by clicking its name.
Postconditions	Project now becomes the active project.
Alternative Flow	3. Project can't be selected due to a system failure.

Table 3: Select Project Use Case

Name	Update Project
ID	FR06
Actor	User
Description	The user updates a project, fetching all git logs up to the current point.
Trigger	1. Clicking the "Atualizar Projeto" button on the Project page.
Preconditions	There is an active project selected.
Basic Flow	1. User is in the Project page. 2. User clicks "Atualizar Projeto"
Postconditions	Project is updated with the latest information.
Alternative Flow	3. Project may fail to correctly populate the Database due to no internet connection to fetch the needed logs, incorrect URLs or an unrelated failure on Git's part.

Table 4: Update Project Use Case

Name	Display Project View
ID	FR07
Actor	User
Description	The user consults the Project View, visualizing the data displayed about the project.
Trigger	1. Clicking the "Vista de Projeto" button on the navigation bar.
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Projeto" button on the navigation bar.
Postconditions	The project view is displayed with the associated data.
Alternative Flow	None

Table 5: Display Project View Use Case

Name	Display Social View - Team Effort
ID	FR08
Actor	User
Description	The user consults the Team Effort within the Social View page, visualizing the data displayed about the team members.
Trigger	1. Clicking the "Equipa" button on the Social View page.
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista Social" button on the navigation bar. 2. User clicks on the "Equipa" button. 3. User selects team members from the available selection.
Postconditions	The Social View - Team Effort is displayed with the associated data.
Alternative Flow	None

Table 6: Display Social View - Team Effort Use Case

Name	Display Social View - Individual Nature of the Effort
ID	FR09
Actor	User
Description	The user consults the Individual Nature of the Effort within the Social View page, visualizing the data displayed about the individual.
Trigger	1. Clicking the "Natureza do Esforço" button on the Team member Social View Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista Social" button on the navigation bar. 2. User clicks on the "Indivíduos" button. 3. User selects a team member from the available selection. 4. User clicks "Nature of the Effort" button.
Postconditions	The Social View - Individual Nature of the Effort is displayed with the associated data.
Alternative Flow	None

Table 7: Display Social View - Individual Nature of the Effort Use Case

Name	Display Social View - Individual Effort Across Time
ID	FR10
Actor	User
Description	The user consults the Individual Effort Across Time within the Social View page, visualizing the data displayed about the individual.
Trigger	1. Clicking the "Evolução do Esforço" button on the Team member Social View Page.
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista Social" button on the navigation bar. 2. User clicks on the "Indivíduos" button. 3. User selects a team member from the available selection. 4. User clicks "Evolução do Esforço" button.
Postconditions	The Social View - Individual Effort is displayed with the associated data.
Alternative Flow	None

Table 8: Display Social View - Individual Effort Use Case

Name	Display Social View - Manipulated Files
ID	FR11
Actor	User
Description	The user consults the Manipulated Files of an individual within the Social View page, visualizing the data displayed.
Trigger	1. Clicking the "Ficheiros Manipulados" button on the Individual Social View Page.
Preconditions	There is an active project selected.
Basic Flow	1. User clicks the "Vista Social" button on the navigation bar. 2. User clicks on the "Indivíduos" button. 3. User selects a team member from the available selection. 4. User clicks "Ficheiros Manipulados" button.
Postconditions	The Social View - Manipulated Files is displayed with the associated data.
Alternative Flow	None

Table 9: Display Social View - Manipulated Files Use Case

Name	Display Social View - Individual Tree
ID	FR12
Actor	User
Description	The user consults the Individual Tree of an individual within the Social View page, visualizing the data displayed.
Trigger	1. Clicking the "Árvore de Indivíduo" button on the Individual Social View Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista Social" button on the navigation bar. 2. User clicks on the "Indivíduos" button. 3. User selects a team member from the available selection. 4. User clicks "Árvore de Indivíduo" button.
Postconditions	The Social View - Individual Tree is displayed with the associated data.
Alternative Flow	None

Table 10: Display Social View - Individual Tree Use Case

Name	Display Artifact View - File Category
ID	FR13
Actor	User
Description	The user consults the File Category of the Project within the Artifact View page, visualizing the data displayed.
Trigger	1. Clicking the "Categoria de Ficheiros" button on the Artifact View - Files Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar. 2. User clicks on the "Ficheiros" button. 4. User clicks "Categoria de Ficheiros" button.
Postconditions	The Artifact View - File Category is displayed with the associated data.
Alternative Flow	None

Table 11: Display Artifact View - File Category Use Case

Name	Display Artifact View - Most Modified Files
ID	FR14
Actor	User
Description	The user consults the most modified files of the Project within the Artifact View page, visualizing the data displayed.
Trigger	1. Clicking the "Ficheiros Mais Modificados" button on the Artifact View - Files Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar. 2. User clicks on the "Ficheiros" button. 4. User clicks "Ficheiros Mais Modificados" button.
Postconditions	The Artifact View - Most Modified Files is displayed with the associated data.
Alternative Flow	None

Table 12: Display Social View - Most Modified Files Use Case

Name	Display Artifact View - Modified Files by Most People
ID	FR15
Actor	User
Description	The user consults the modified files by most people within the Artifact View page, visualizing the data displayed.
Trigger	1. Clicking the "Ficheiros modificados por mais pessoas" button on the Artifact View - Files Page
Preconditions	There is an active project selected.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar. 2. User clicks on the "Ficheiros" button. 4. User clicks "Ficheiros modificados por mais pessoas" button.
Postconditions	The Artifact View - Modified Files by Most People is displayed with the associated data.
Alternative Flow	None

Table 13: Display Artifact View - Modified Files by Most People Use Case

Name	Display Artifact View - Complete Tree
ID	FR16
Actor	User
Description	The user consults the complete file directory tree within the Artifact View page, visualizing the data displayed.
Trigger	1. Clicking the "Árvore Completa" button on the Artifact View - Tree Page
Preconditions	There is an active project selected.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar. 2. User clicks on the "Árvores" button. 4. User clicks the "Árvore Completa" button.
Postconditions	The Artifact View - Complete Tree is displayed with the associated data.
Alternative Flow	None

Table 14: Display Artifact View - Complete Tree Use Case

Name	Display Artifact View - CHURN Tree
ID	FR18
Actor	User
Description	The user consults the CHURN file directory tree within the Artifact View page, visualizing the data displayed.
Trigger	1. Clicking the "Árvore CHURN" button on the Artifact View - Tree Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar. 2. User clicks on the "Árvores" button. 4. User clicks the "Árvore CHURN" button.
Postconditions	The Artifact View - CHURN Tree is displayed with the associated data.
Alternative Flow	None

Table 15: Display Artifact View - CHRUN Tree Use Case

Name	View Modification History
ID	FR19
Actor	User
Description	The user consults the modification history of a file, visualizing the data displayed.
Trigger	1. Clicking the "Ver Histórico de Modificações" button on any Artifact View - File Page.
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Artefactos" button on the navigation bar 2. User clicks the "Ficheiros" button 3. User clicks the "Categoria de Ficheiros" button 4. User clicks the "Ver Ficheiro" button
Postconditions	The View Modification History is displayed with the associated data.
Alternative Flow	3. User clicks the "Ficheiros mais modificados" button 3.1 User clicks the "Ficheiros modificados por mais pessoas" button

Table 16: View Modification History Use Case

Name	Display Social View - Handover of Work
ID	FR21
Actor	User
Description	The user consults the handover of work, visualizing the data displayed.
Trigger	1. Clicking the "Handover of Work" button on the Social View - Team page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Vista de Social" button on the navigation bar 2. User clicks the "Equipa" button 3. User clicks the "Handover of Work" button
Postconditions	The Social View - Handover of Work is displayed with the associated data.
Alternative Flow	None

Table 17: Display Social View - Handover of Work Use Case

Name	Delete Project
ID	FR22
Actor	User
Description	The user selects a project from the available ones and deletes it. displayed.
Trigger	1. Clicking the "Apagar Projeto" button on the Project Page
Preconditions	There is an active project selected with a regex configuration.
Basic Flow	1. User clicks the "Projeto" button on the navigation bar 2. User clicks the "Apagar Projeto" button
Postconditions	The selected project is deleted from the database
Alternative Flow	3. The project can't be deleted due to a system failure

Table 18: Display Social View - Delete Projet Use Case

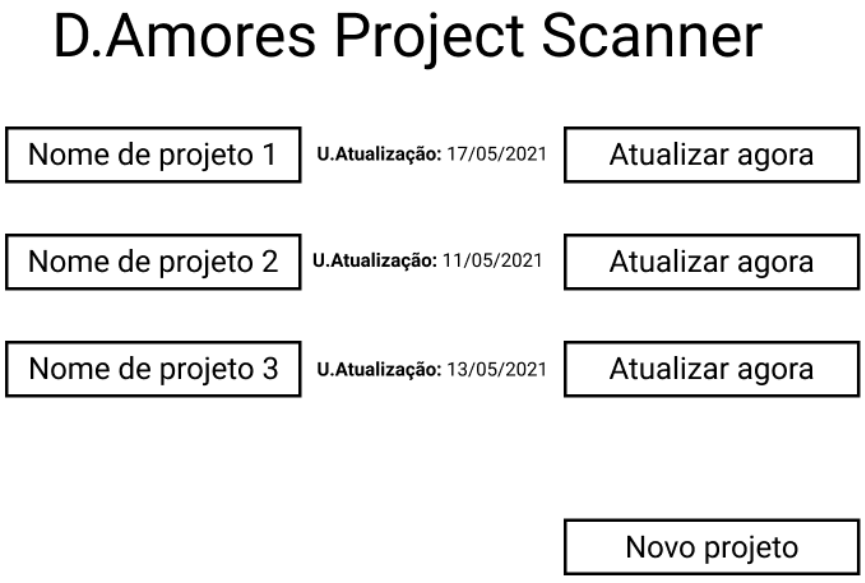


Figure 1: Mockup for the Homepage

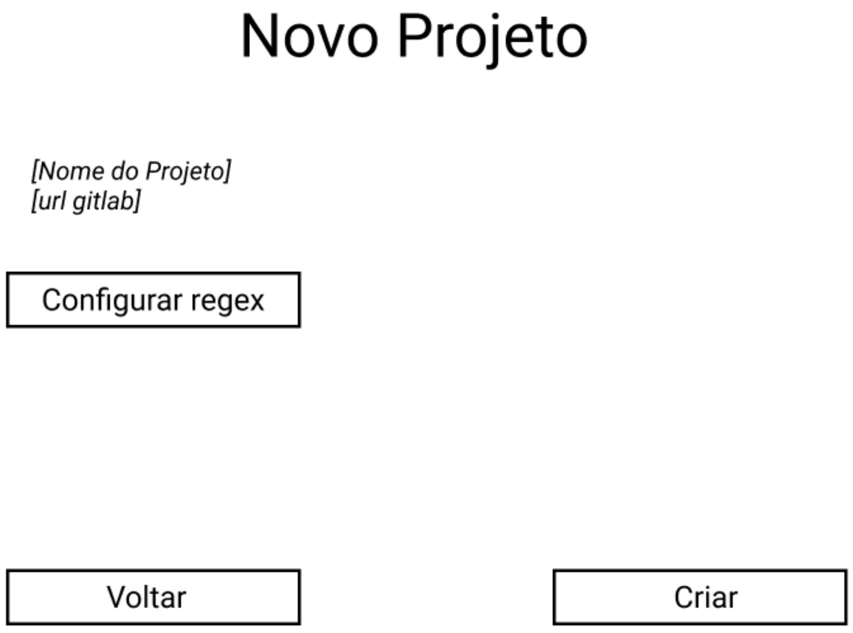


Figure 2: Mockup for the New Project page

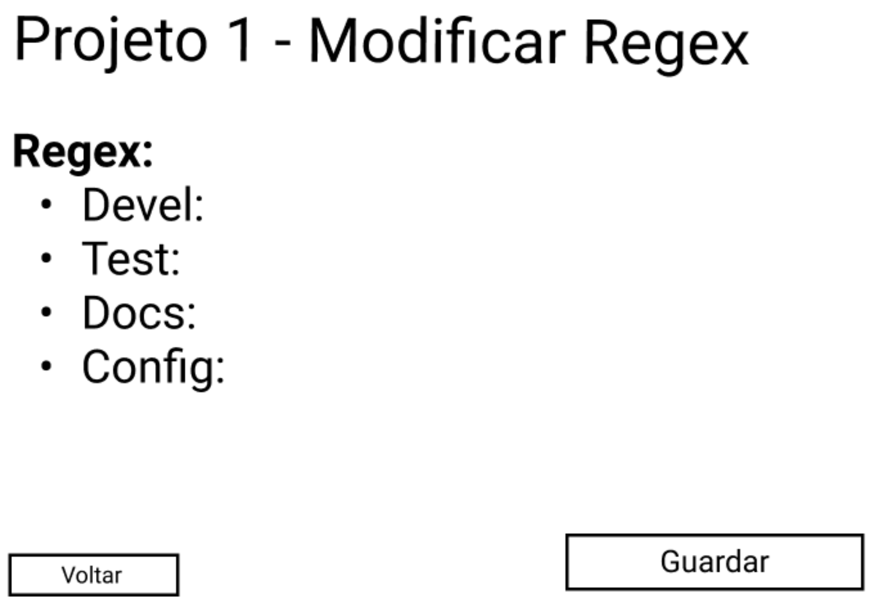


Figure 3: Mockup for the Modify Regex Page

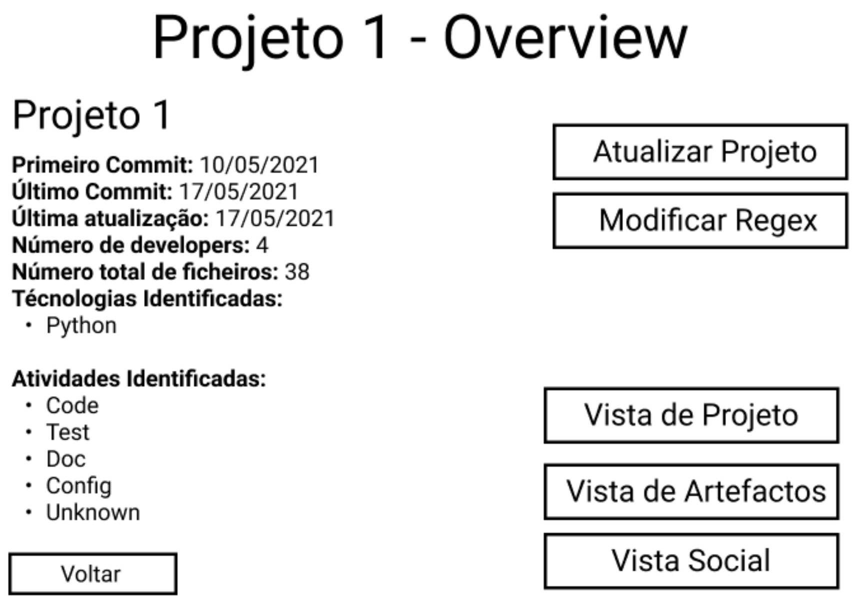


Figure 4: Mockup for the Project Page

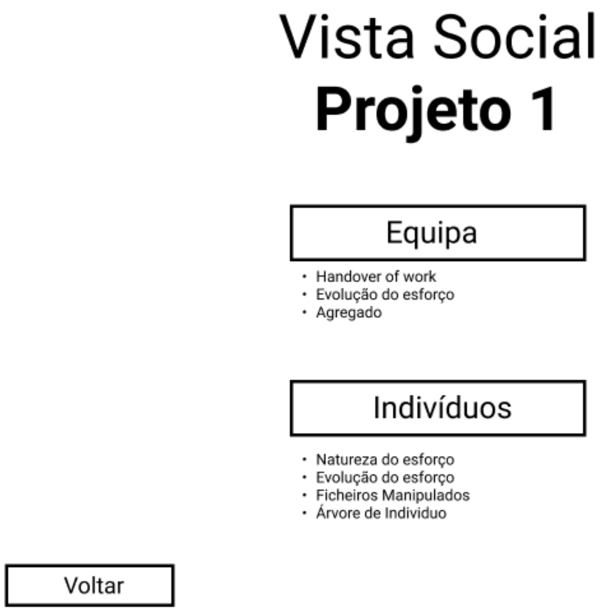


Figure 5: Mockup for the Social View Page

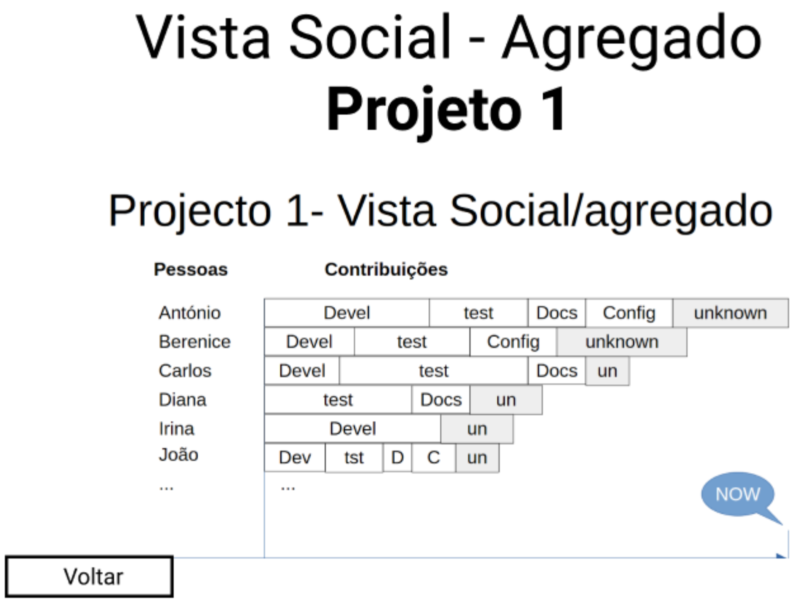


Figure 6: Mockup for the Contribution per Activity Chart in the Social View Page

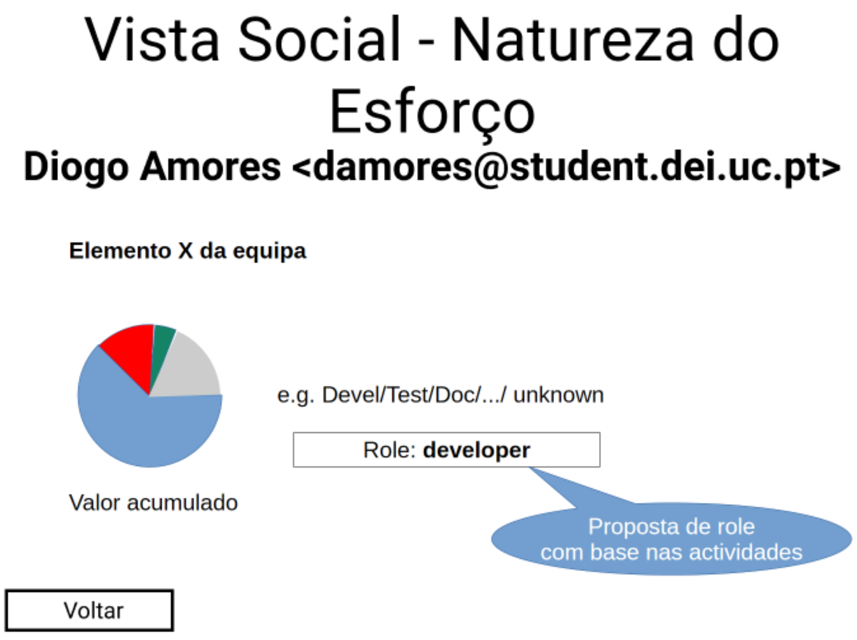

Vista Social - Natureza do Esforço
Diogo Amores <damores@student.dei.uc.pt>
Elemento X da equipa
e.g. Devel/Test/Doc/.../ unknown
Role: developer
Valor acumulado
Proposta de role com base nas actividades
Voltar

Figure 7: Mockup for the Nature of the Effort Chart in the Social View of an Individual

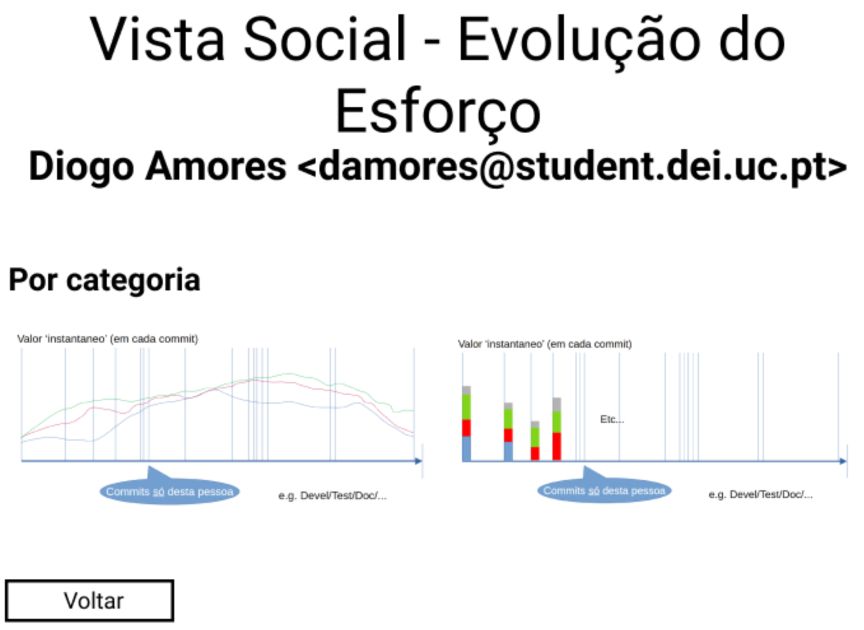
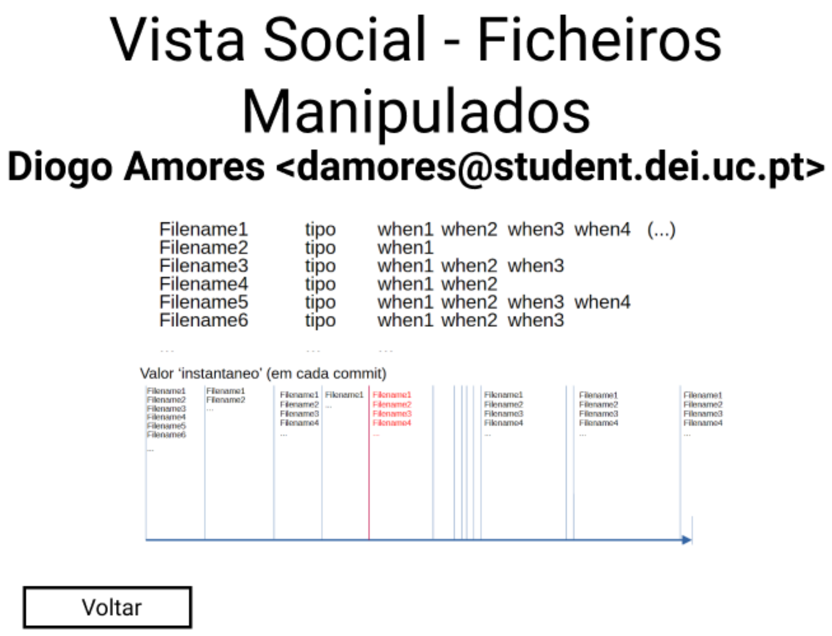

Vista Social - Evolução do Esforço
Diogo Amores <damores@student.dei.uc.pt>
Por categoria
Valor 'instantaneo' (em cada commit)
Commits só desta pessoa
e.g. Devel/Test/Doc/...
Valor 'instantaneo' (em cada commit)
Etc...
Commits só desta pessoa
e.g. Devel/Test/Doc/...
Voltar

Figure 8: Mockup for the Evolution of the Effort per Category Charts in the Social View of an Individual

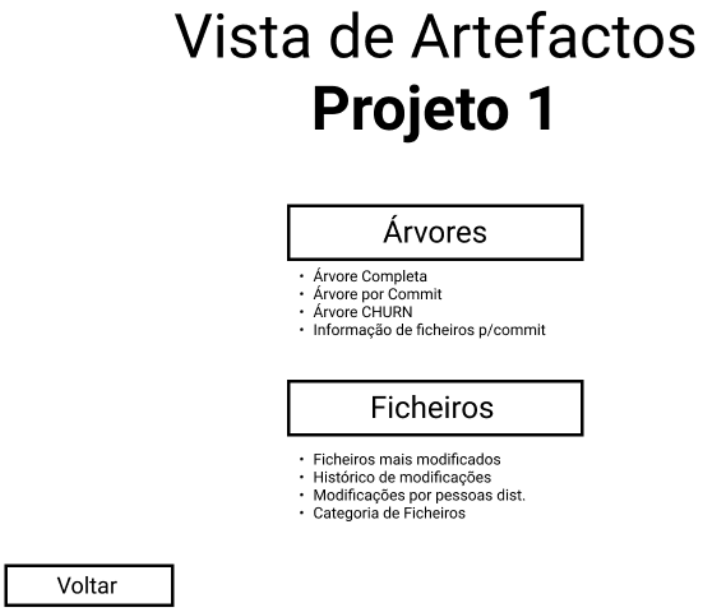

Vista Social - Ficheiros Manipulados

Diogo Amores <damores@student.dei.uc.pt>

Filename1	tipo	when1 when2 when3 when4 (...)
Filename2	tipo	when1
Filename3	tipo	when1 when2 when3
Filename4	tipo	when1 when2
Filename5	tipo	when1 when2 when3 when4
Filename6	tipo	when1 when2 when3
...	...	...

Valor 'instantaneo' (em cada commit)
Filename1 Filename2 Filename3 Filename4 Filename5 Filename6 ...
Filename1 Filename2 ...
Filename1 Filename2 Filename3 Filename4 ...
Filename1 ...
Filename1 Filename2 Filename3 Filename4 ...
Filename1 Filename2 Filename3 Filename4 ...
Filename1 Filename2 Filename3 Filename4 ...
Filename1 Filename2 Filename3 Filename4 ...
Voltar

Figure 9: Mockup for the Manipulated Files Page in the Social View on an Individual


Vista de Artefactos

Projeto 1

Árvores
- Árvore Completa
- Árvore por Commit
- Árvore CHURN
- Informação de ficheiros p/commit

Ficheiros
- Ficheiros mais modificados
- Histórico de modificações
- Modificações por pessoas dist.
- Categoria de Ficheiros

Figure 10: Mockup for the Artifact View Page

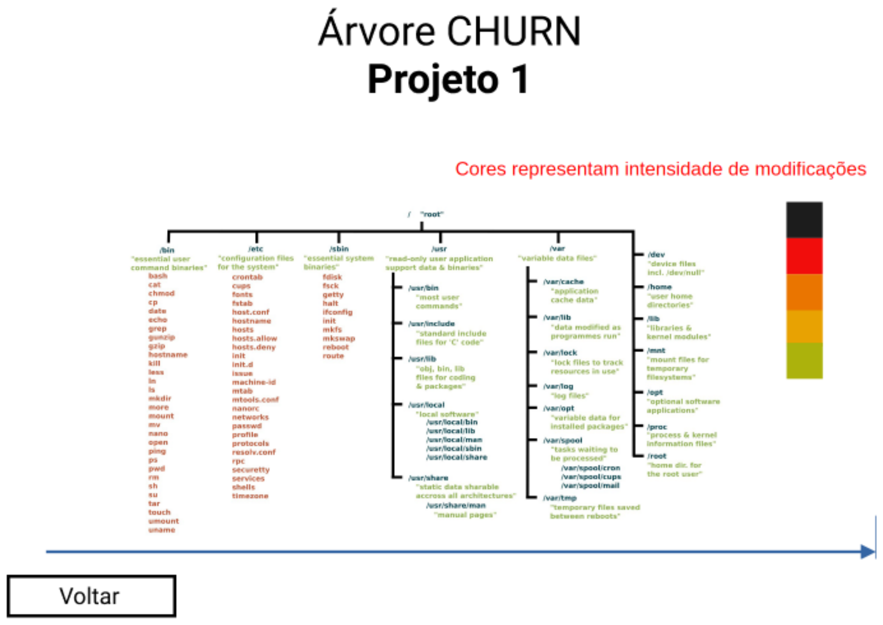
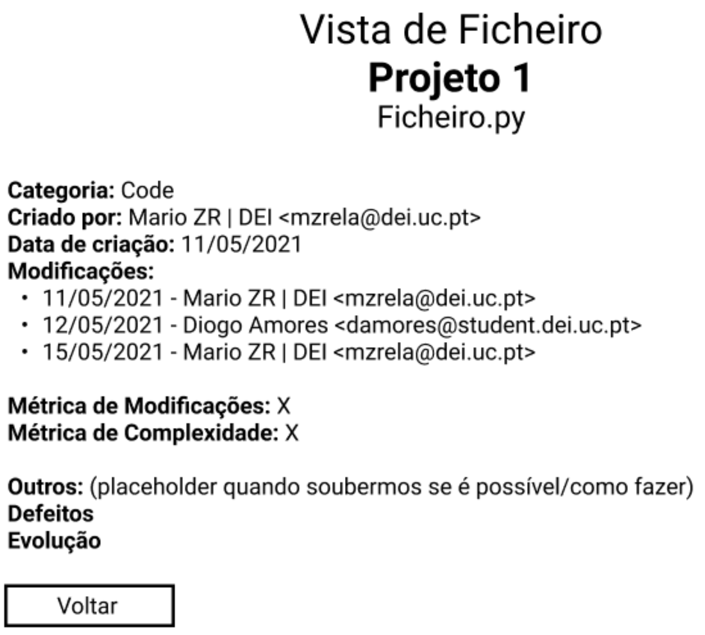


Figure 11: Mockup for the CHURN Tree Page in the Artifact View Page



Vista de Ficheiro
Projeto 1
Ficheiro.py

Categoria: Code
Criado por: Mario ZR | DEI <mzrela@dei.uc.pt>
Data de criação: 11/05/2021
Modificações:
- 11/05/2021 - Mario ZR | DEI <mzrela@dei.uc.pt>
- 12/05/2021 - Diogo Amores <damores@student.dei.uc.pt>
- 15/05/2021 - Mario ZR | DEI <mzrela@dei.uc.pt>

Métrica de Modificações: X
Métrica de Complexidade: X

Outros: (placeholder quando soubermos se é possível/como fazer)
Defeitos
Evolução

Figure 12: Mockup for File View Page of a selected file

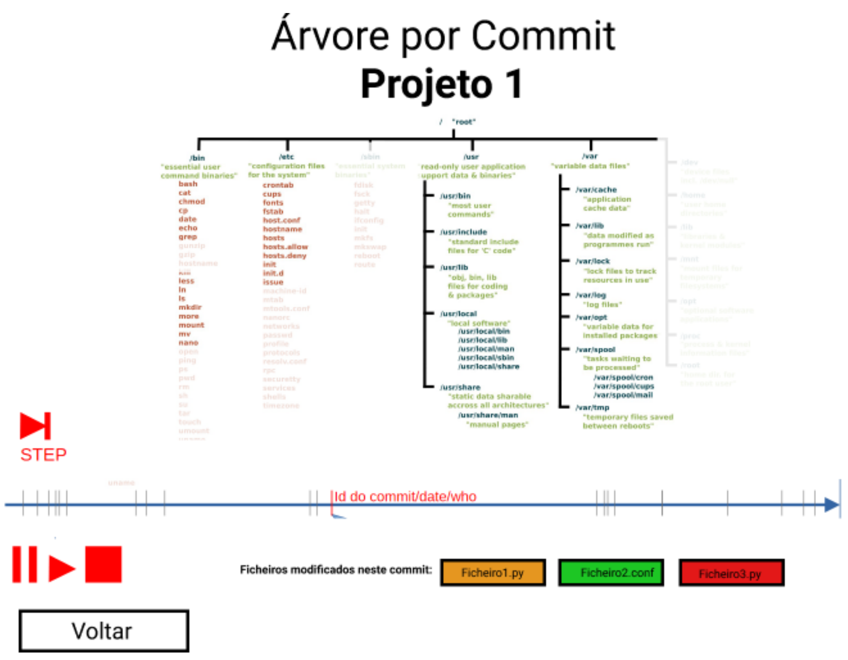

Árvore por Commit
Projeto 1
/ "root"
/bin
"essential user command binaries"
/etc
"configuration files for the system"
/sbin
/usr
"read-only user application support data & binaries"
/var
"variable data files"
STEP
Id do commit/date/who
Ficheiros modificados neste commit:
Ficheiro1.py
Ficheiro2.conf
Ficheiro3.py
Voltar

Figure 13: Mockup for File View Page of a Selected File

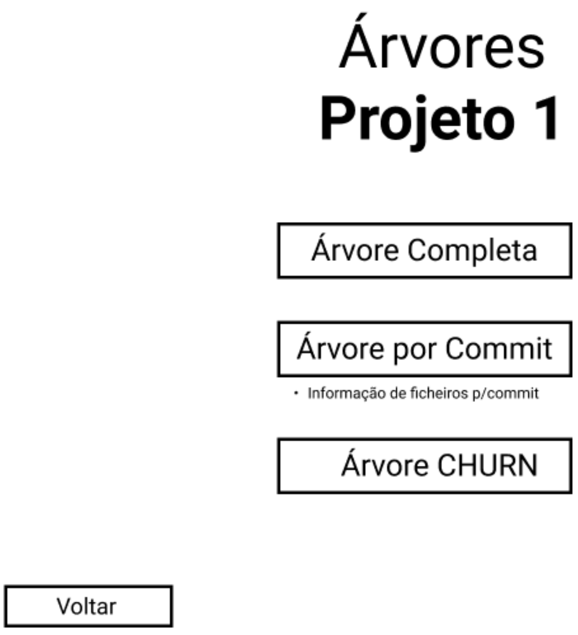
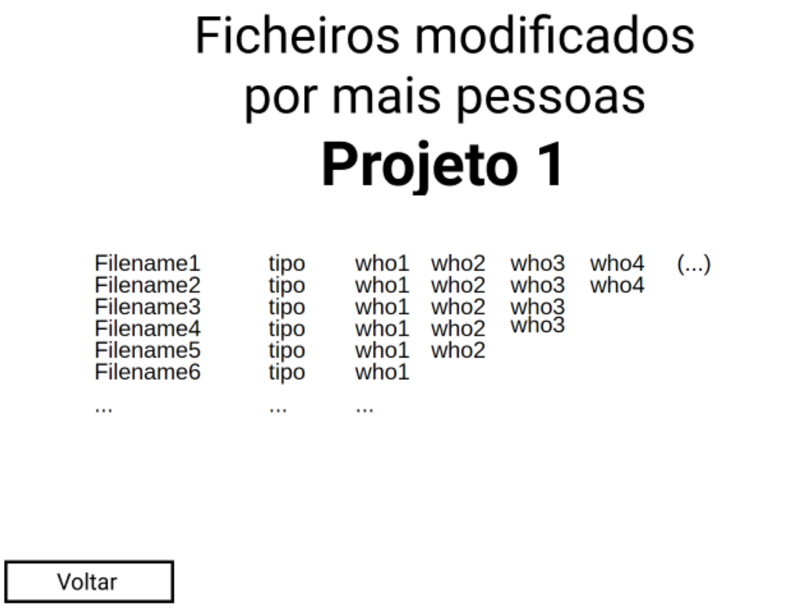


Figure 14: Mockup for Tree Page in the Artifact View



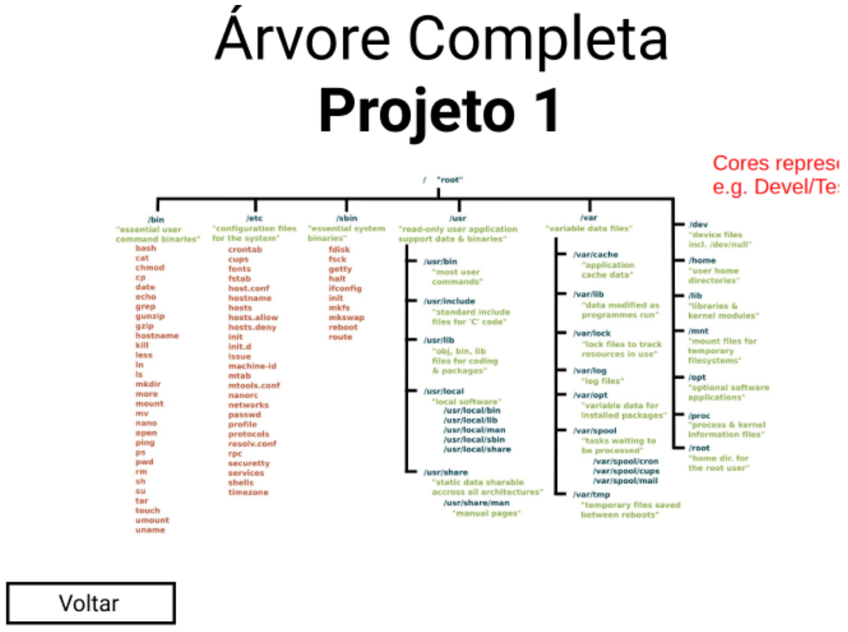
Ficheiros modificados
por mais pessoas

Projeto 1

Filename1	tipo	who1	who2	who3	who4	(...)
Filename2	tipo	who1	who2	who3	who4	
Filename3	tipo	who1	who2	who3		
Filename4	tipo	who1	who2	who3		
Filename5	tipo	who1	who2			
Filename6	tipo	who1				
...	...	...				

Voltar

Figure 15: Mockup for File View Page of a selected file

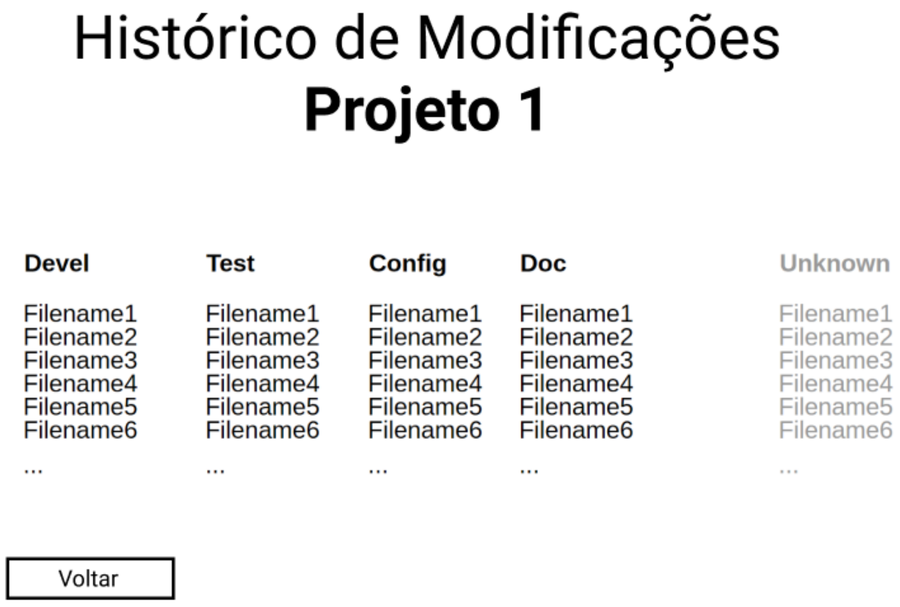


Árvore Completa

Projeto 1

Voltar

Figure 16: Mockup for Directory Tree Page in the Artifact View

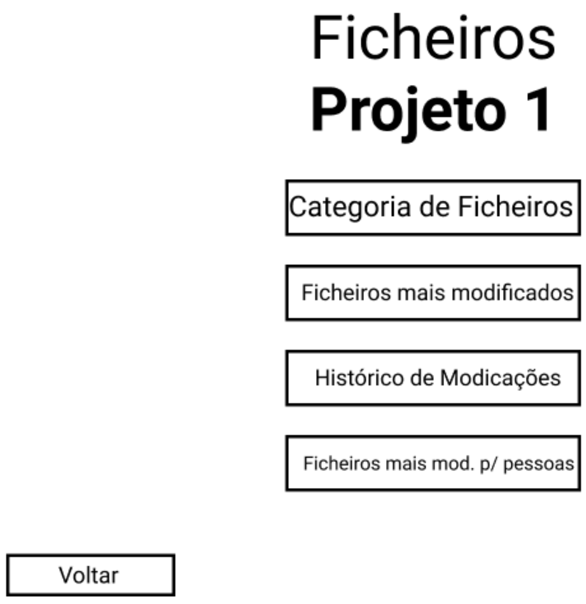


Histórico de Modificações

Projeto 1

Devel	Test	Config	Doc	Unknown
Filename1	Filename1	Filename1	Filename1	Filename1
Filename2	Filename2	Filename2	Filename2	Filename2
Filename3	Filename3	Filename3	Filename3	Filename3
Filename4	Filename4	Filename4	Filename4	Filename4
Filename5	Filename5	Filename5	Filename5	Filename5
Filename6	Filename6	Filename6	Filename6	Filename6
...	...	...	...	...

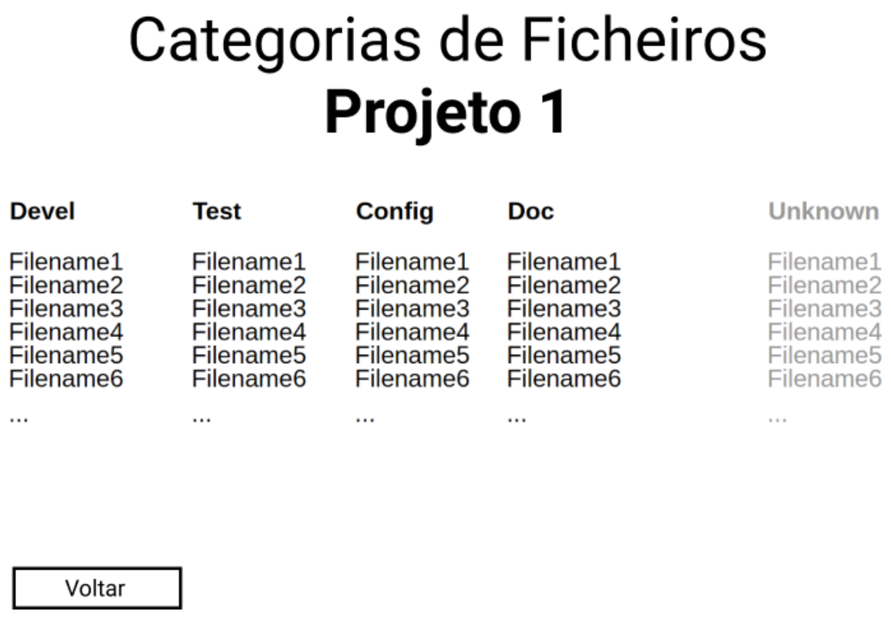
Figure 17: Mockup for the Modification History Page in the Artifact View



Ficheiros

Projeto 1

Figure 18: Mockup for the File Page in the Artifact View

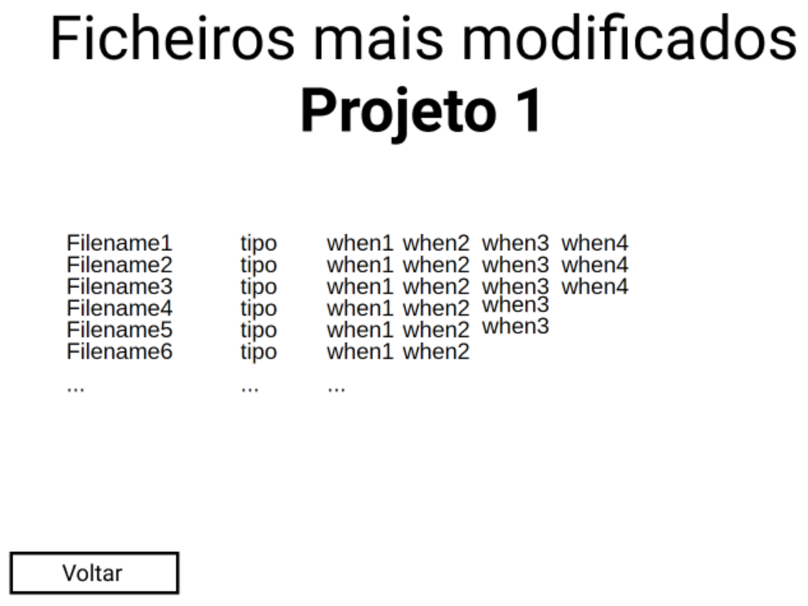


Categorias de Ficheiros
Projeto 1

Devel	Test	Config	Doc	Unknown
Filename1	Filename1	Filename1	Filename1	Filename1
Filename2	Filename2	Filename2	Filename2	Filename2
Filename3	Filename3	Filename3	Filename3	Filename3
Filename4	Filename4	Filename4	Filename4	Filename4
Filename5	Filename5	Filename5	Filename5	Filename5
Filename6	Filename6	Filename6	Filename6	Filename6
...	...	...	...	...

Voltar

Figure 19: Mockup for the File Category Page in the Artifact View



Ficheiros mais modificados
Projeto 1

Filename1	tipo	when1	when2	when3	when4
Filename2	tipo	when1	when2	when3	when4
Filename3	tipo	when1	when2	when3	when4
Filename4	tipo	when1	when2	when3	
Filename5	tipo	when1	when2	when3	
Filename6	tipo	when1	when2		
...	...	...			

Voltar

Figure 20: Mockup for the Most Modified Files Page in the Artifact View

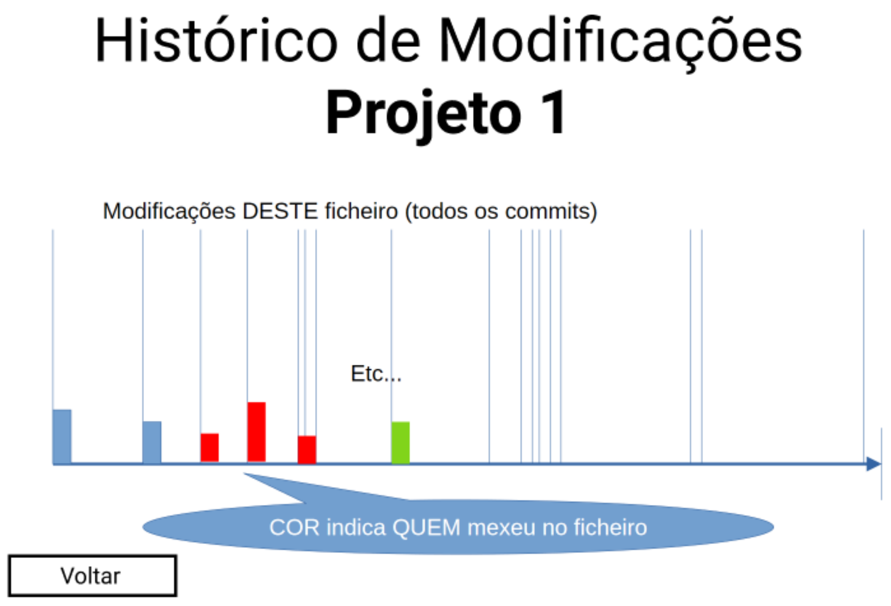

Histórico de Modificações
Projeto 1
Modificações DESTE ficheiro (todos os commits)
Etc...
COR indica QUEM mexeu no ficheiro
Voltar

Figure 21: Mockup for the Modification History Chart in the Artifact View