MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

CNG491 Computer Engineering Design I

Second Iteration of Report and Presentation

Project Name: Git Assessor - Automated Assessment of Git Usage of Students

Project Participants:

Ziya Taner Keçeci – 2152049

Yusuf Mete Kabakçı - 2151983

Dogukan Çatal - 2257996

Mahmut Ali Şahin - 2243673

1. Introduction

Git is a popular source-code version control system that allows software developers to work simultaneously and keep track of their changes over time. In this project, we will develop a web application that can be used by instructors to assess the git repositories of their students.

1. Motivation

For that assessment, instructors will choose some rules which will be added by our team to the app so that instructors will be able to get an evaluation/assessment report of their students' projects. Creating a Web Application to help instructors to assess their students' performances is our motivation. As a result of that improvement of students will be faster and efficient.

2. Aim & Objectives

In Git Assessor, instructors will be able to enter multiple Git Repository links and for every Repository link there will be the list of rules that have been chosen by the instructor. In that list rules instructor will be able to see which rules have been covered and which rules haven't been covered. Also, instructors will be able to see participants of each repository with their performance score. Participant performance score will start from 100 and for every rule violation their point will be deducted, and instructor will be able to see which participant violated which rule. To increase the readability of the Assessment Report there will be bar/pie charts for detailed visualization of students' performances.

3. Methodology

Most critical point of this project is the set of rules that we will add to the application. Utility of the Assessment Report will be dependent on the rules that have been provided and to provide effective rules we will be using an set of rules as cited in [Eraslan et al., 2020]. We will implement the rules that have been covered in this paper and instructors will be able to choose the rules they want. According to that paper, those rules were created by the academics who has experience on the common errors and poor practices among students. For this project we divided into groups of 2. First group is responsible of Web Application side and other group is responsible of Web Service side. For the Web Application we are going to use Django, HTML, CSS and javascript. For the web Service we are going to use FastAPI and PyGithub. Algorithms of the rules will be written in Web Service. Users will enter their repository/repositories and select rule(s) from rule set and that information will be sent to Web Service. In the Web Service, we will use send the queries to Github. Those queries will be determined in the algorithms of rules thus every rule will have different query in theory. For testing we will create a public Github repository and do those common errors and poor practices intentionaly to test our algorithms.

2. Requirements

2.1. Stakeholders

Instructors that will use application to get assessment for their students' git projects.

Students that will be evaluated based on their performance.

2.2. Functional System Requirements

1- An instructor shall be able to provide GitHub repository links to system.

•A text box will be provided as an input for instructors to enter their desired repository links.

2- An instructor shall be able to delete any repository link from the list before getting the report.

•The instructor can use the check boxes to select his/her desired repository links to remove/delete those links by pressing to the remove button.

3- An instructor shall be able to choose which rules will be used for assessment.

•Check boxes will be provided with each rule and the instructor can choose any rule that he/she wants to check whether students obeyed or not.

4- An instructor shall be able to get an assessment report for repositories and participants.

•Assess button will be provided for the instructor to get an assessment report when the repository links and the desired rules are selected.

5- For the rule set, the system shall provide the list of rules and instructors shall be able to configure them.

6- Assessment report will provide Visual representation of the data. Such as Bar charts of student performances.

•A new page will be provided to let the instructor to observe the students' performances, mistakes, and grades. For easier reading some of those data will be provided as charts (eg. Bar chart, …,).

2.3. Non-functional Requirements

1- The System shall be available 7/24 for the users.

2- The System shall provide grade for each participant students about their work in the assessment reports

3- The System shall provide visualized charts in the assessment reports.

4- The system shall use the GitHub repository link that has been provided by the instructor to get data from GitHub.

5- The system shall give error messages when an instructor tries to add an invalid repository link to the repository list.

6- The system shall give an error message when an instructor does not provide any link or/and rule to the system.

2.4. Domain Requirements

1- Git API doesn’t allow too many requests thus the number of requests and repository sizes are constraints.

2- Since GitHub API cannot reach private repositories, only public repositories’ data will be accessed. So, students’ provided repositories should be public.

2.5. Assumptions and Justifications

1- We assume repositories should be public, because GIT doesn't allow receiving data from private repositories.

2- We assume project repositories will not be huge so that GitHub doesnt cut off the connection.

3- We assume students’consents have been taken to be assessed by their performance. Otherwise, it will be violation of ethics.

4- To calculate Jones's First-Order Effort Estimation and Schedule Estimation,

we assume as we are working as shrink-wrap software organization.

5- According to [Prechelt, 2000] Python and Pearl has strong similarities. Thus we accepted LOC/FP value of Python as 20.

3. System Modelling

3.1. Structured Use Case Diagram

In figure 1, we provide Structured Use Case Diagram for our Git Assessor Web Application while User, GitHub (Git API), Git Assessor Web Service are actors. User can analyze either a single or multiple repositories. To analyze repositories some rules must be selected by user and then “Rule Set Check” checks those selected rules to visualize a result which is an assessment report. “Git Assessor Web Service” handles the “Rule Set Check” by communicating with “GitHub” using GitHub API

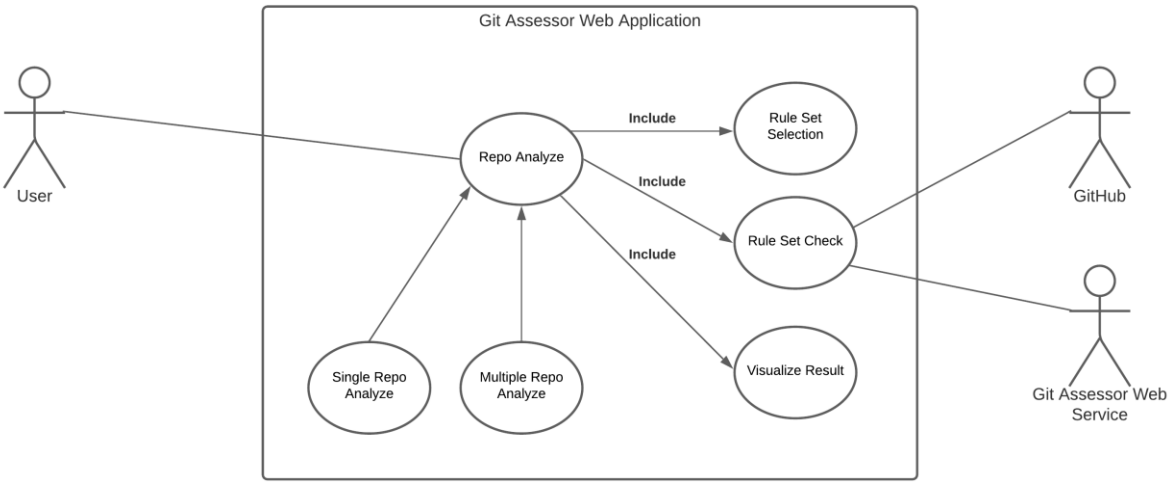

Figure 1 Structured Use Case Diagram

3.2. Sequence Diagram for Major Use Cases

In Figure 2, we provide Sequence Diagram for our major use case. Git Assessor Web Application is getting all the repos and rules from user in one time but asking Git Assessor Web Service about repositores once at a time.

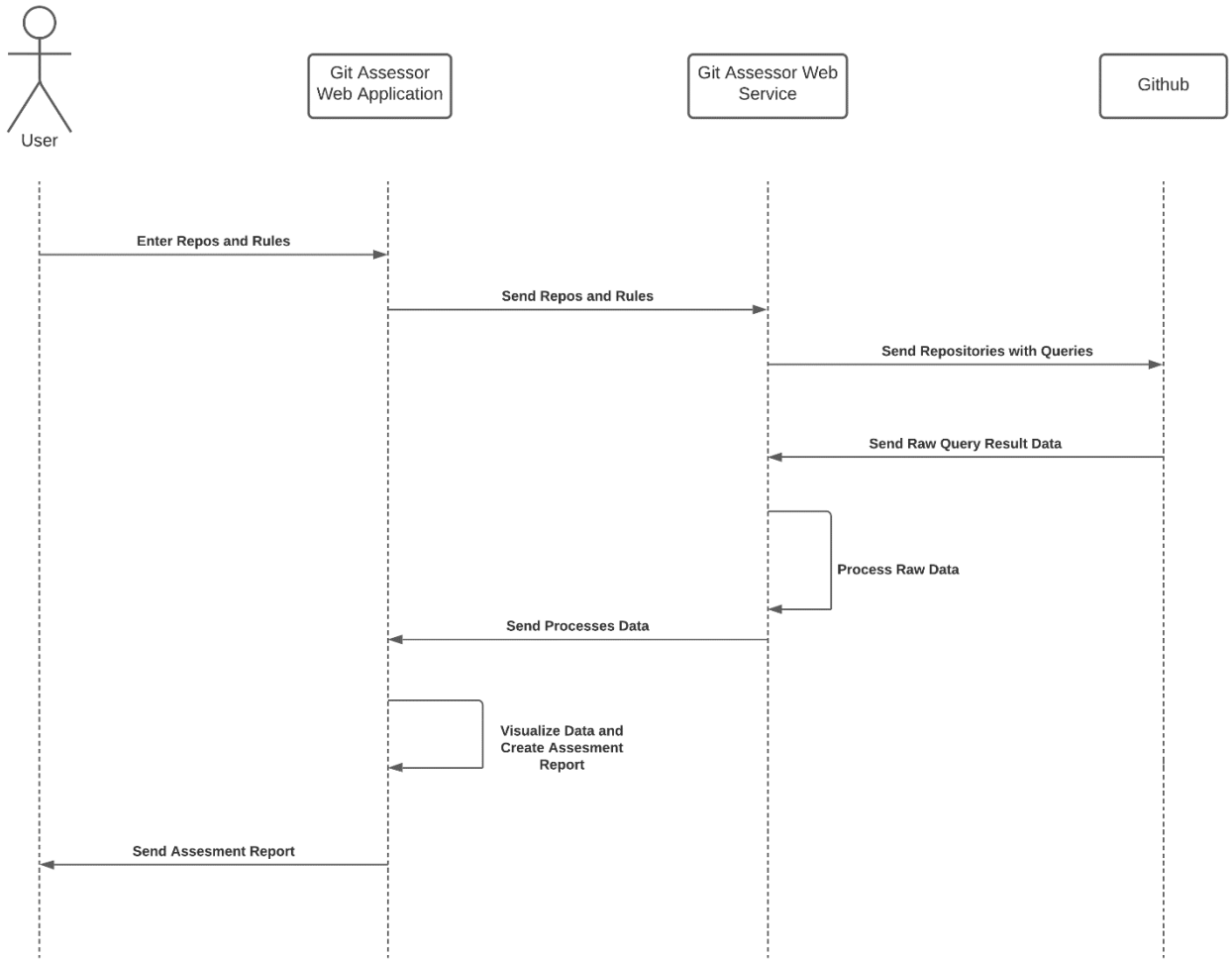

Figure 2: Sequence Diagram

3.3. Context Model

In Figure 3, we provide Context Model. Git Assessor Application is connecting to GitHub through GitAssessor Web Service.

Figure 3 Context Model

3.4. Architectural Model

In Figure 4, we provide Architectural Model. Git Assessor Web Service contains GitHub API and Rule Checker. Git Assessor Application contains Rule Selector (lets user to select desired rules) and Report Generator. Git Assessor Application sends rules and links to the Git Assessor Web Service and receives result (data) from Git Assessor Web Service to create a report. Git Assessor Web Service sends/receives data to/from GitHub and Git Assessor Application (Web Application). GitHub Lets Git Assessor Web Service to retrieve desired data via GitHub API.

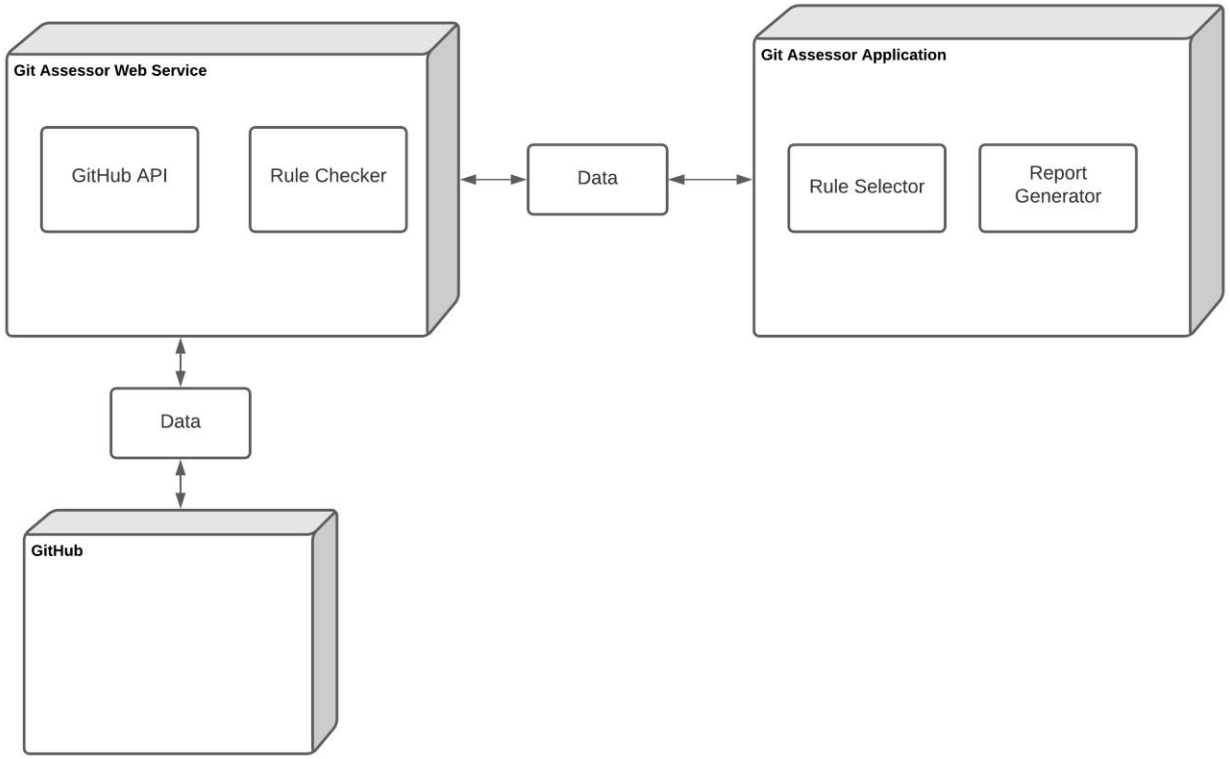

Figure 4 Architectural Model

3.5. Process Model

Application Perspective

Application process will start with asking for a Repo Link to be entered. If the user wants to add more links, he/she will be able to do that. After adding all desired links, the user will need to select rule(s). When all links and rules are selected, links and rules will be sent to the web service. When the result is retrieved from the web service, process will end by creating a report with those retrieved results.

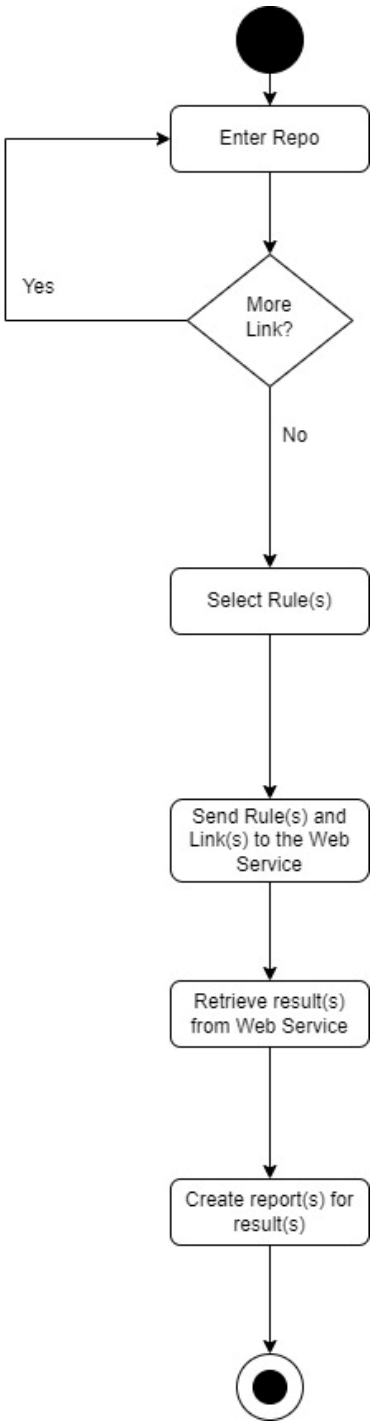

Figure 5: Web Application Perspective of Process Model

Service Perspective

Service process will start with getting repository links and selected rules from the application. Process will continue by retrieving data of repository links from GitHub via GitHub API. Then, selected rules will be checked to create a result and afterwards, When the results are created for each link, process will end by sending those results to the application.

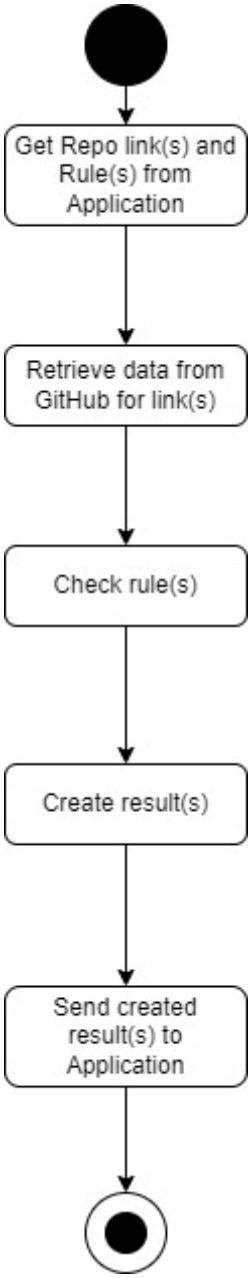

Figure 6: Web Application Perspective of Process Model

4. (Graphical) User Interface

4.1. Home Page

Home Page

Figure 6 shows the “Home page” when the user enters the site. In this page user can enter the repository links to the URL box (where it says “Repo Link”). When the user pastes the desired repository links and clicks to “Add” button those links will be added to the “list of added links”. If user has a change of mind and wants to remove some links, it can be done by clicking to the checkbox next to the links and then pressing to the “Remove” button. There is a list of rules under the “Rules” title, desired rules to be checked by the web service for each link in the list can be selected by clicking to the checkbox next to each rule. After everything is done, user can press to the “Assess” button to open “Assessment Report Page” (Figure 7).

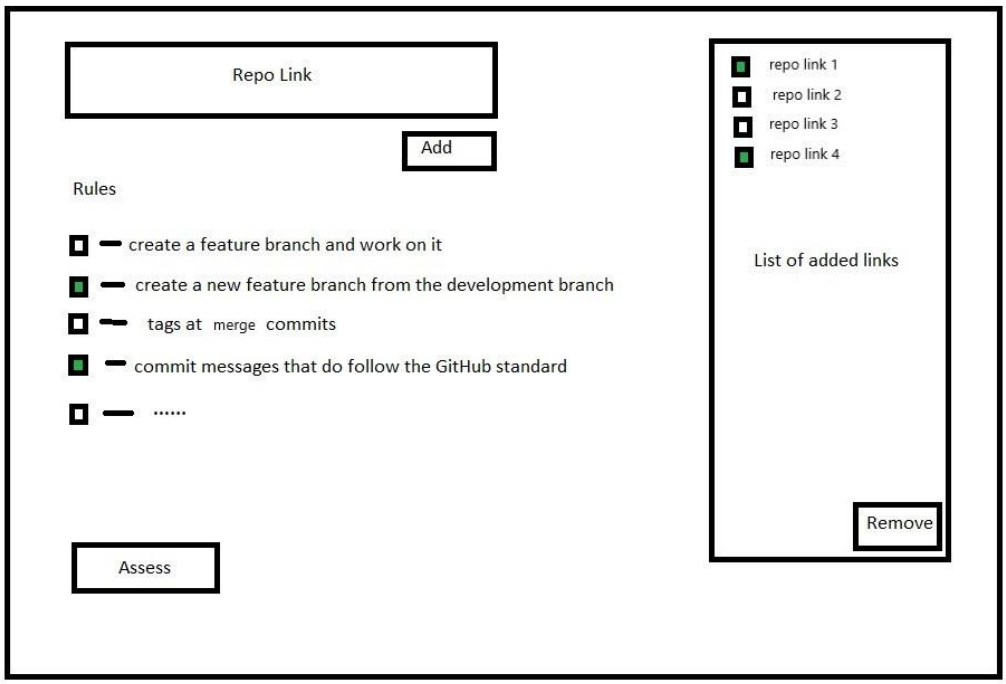

Figure 7 GUI Screen 1

4.2. Assessment Report Page

Figure 7 shows the “Assessment Report Page” which is created when the “Assess” button is clicked in “Home Page” (Figure 6). This page shows the results of checking rules for each repo links by the web service for user to analyze. In this page there is a “REPORT TABS” section on the top of the page. User can select desired repo link tab to analyze report, each tab shows its own report for that repo link. In the report selected rules by the user can be seen and each participant student for that repository is graded by whether they have obeyed to that specific rule or not. As the report goes on, mistakes that have been made can be seen in detailed. Bar/pie charts are also provided for better visualization to analyze. Charts will show students’ overall situations.

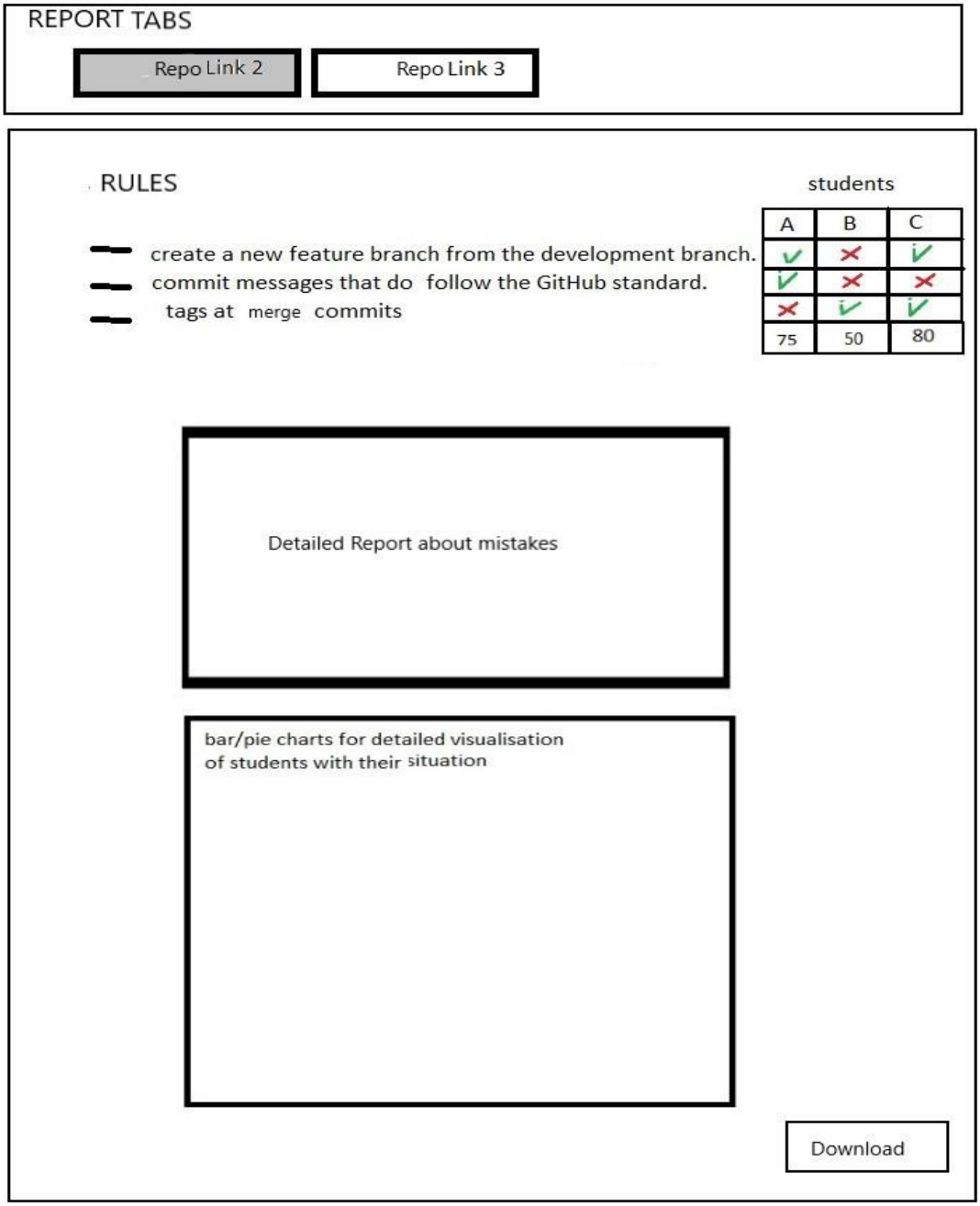

Figure 8 GUI Report Page

5. Agile Development with Scrum

5.1.1 Sprint Backlog

1- Watching an introductory lecture video for Git and GitHub.

2- Reading some introductory Git resources.

3- Reading about some Git workflows.

4- Reading about errors and poor practices of students in using Git.

5- Defining functional and non-functional requirements.

6- Writing an example use-case text

7- Writing domain requirements.

8- Creating structured use case diagram

9- Creating sequence diagram with major use cases

10- Creating a context model

11- Creating an Architectural model

12- Creating a process model

13- Describing first version of GUI with home page and report page

5.1.2. Sprint Burndown Chart

ID	User Story	Hours	Oct 18 – Oct 24	Oct 25 – Oct 31	Nov 1 - Nov 7	Nov 8 – Nov 14	Nov15-Nov21
T1	1- Watching an introductory lecture video for Git and GitHub.	1	1	1	0	0	0
T2	Reading some introductory Git resources.	1	1	1	0	0	0
T3	Reading about some Git workflows.	1	1	1	0	0	0
T4	Reading about errors and poor practices of students in using Git.	1	1	1	1	0	0
T5	Defining functional and non-functional requirements.	1	1	1	1	1	0
T6	Writing an example use-case text	1	1	1	1	1	0
T7	Writing domain requirements.	1	1	1	1	1	0
T8	Creating structured use case diagram	1	1	1	1	1	0
T9	Creating sequence diagram with major use cases	1	1	1	1	1	0
T10	Creating a context model	1	1	1	1	1	0
T11	Creating an Architectural model	1	1	1	1	1	0
T12	Creating a process model	1	1	1	1	1	0
T13	Describing first version of GUI with home page and report page	1	1	1	1	1	0
	TOTAL	13	13	13	10	9	0

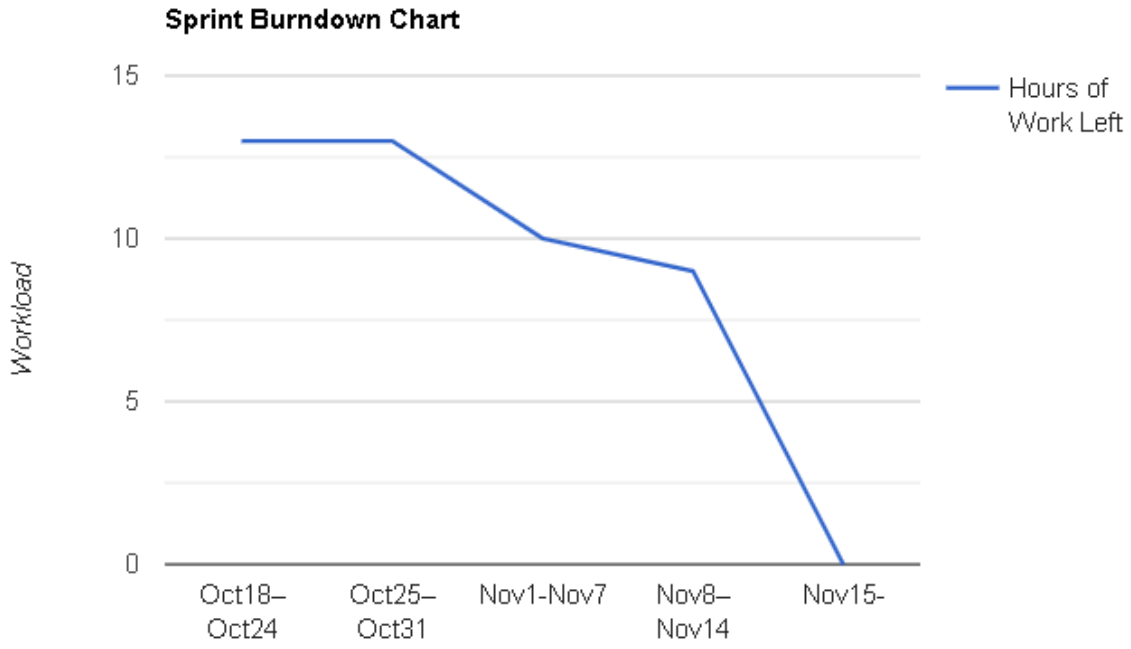

Figure 9: Sprint 1 Burndown Chart

5.1.3. Sprint Review

Everything went as planned in this sprint. We haven't faced with any major problem in this sprint. We have done Introductory reading about Git and Git workflows as a team.

Also, in this sprint, we have decided our functional and non-functional requirements as a team. Then we divided the work between us to prepare Structured Use Case diagram, Sequence diagram, Context model, Architectural model and Process model for the project. We haven't face with any major problem while preparing those diagrams.

Finally, we came together as a team and exchanged ideas on how the GUI should be, and we came to a decision. For the next sprint we are planning to start implementation of our project

5.1.4. Sprint Retrospective

What went well in the sprint?
In this sprint we organized well and manage to finish all the tasks as planned. We have great communication and helping between team members. All team members completed their assigned tasks on time.

What went wrong in the sprint?
Although we managed to catch the due date, it would be better if we had more meetings because we only held one meeting. In this sprint one meeting was enough but this may not be the same for upcoming sprints.

What did we learn?
Although all team members complete their assigned tasks on time, we realized that we need to meet more often as a team and exchange ideas more frequently.

How should the next sprint play out?
In the next sprint, we need to learn from our mistakes that we done in this sprint and come together more often as a team, and we need to exchange ideas more frequently.

5.2.1 Sprint Backlog

1. Research GitHub API for Java
2. Research GitHub API for python
3. Learn and test FastAPI for web service
4. Test GitHup API with FastAPI
5. Creating postgresql and MySQL databases and analyzing the necessity and performance
6. Create Rule algorithms
7. Develop User Interface
8. Adding repo links to link list
9. Request rules from the web service
10. List the rules from web service
11. Address comments of the first iteration report and prepare second report

5.2.2. Sprint Burndown Chart

ID	User Story	Hours	Nov 22 – Nov 28	Nov 29 – Dec 5	Dec 6 - Dec 12	Dec 13 - Dec 19	Dec 20 - Dec 26
T1	Research GitHub API for Java	2	2	0	0	0	0
T2	Research GitHub API for python	2	2	2	0	0	0
T3	Learn and test FastAPI for web service	8	8	8	4	0	0
T4	Test GitHup API with FastAPI	2	2	2	2	2	0
T5	Creating postgresql and MySQL databases and analyzing the necessity and performance	3	3	3	3	3	0
T6	Create Rule algorithms	2	2	2	2	2	2
T7	Develop User Interface	8	8	8	4	4	4
T8	Adding repo links to link list	3	3	3	3	3	3
T9	Request rules from the web service	3	3	3	3	3	3
T10	List the rules from web service	3	3	3	3	3	3
T11	Address comments of the first iteration report and prepare second report	4	4	4	4	4	0
	TOTAL	40	40	38	28	24	15

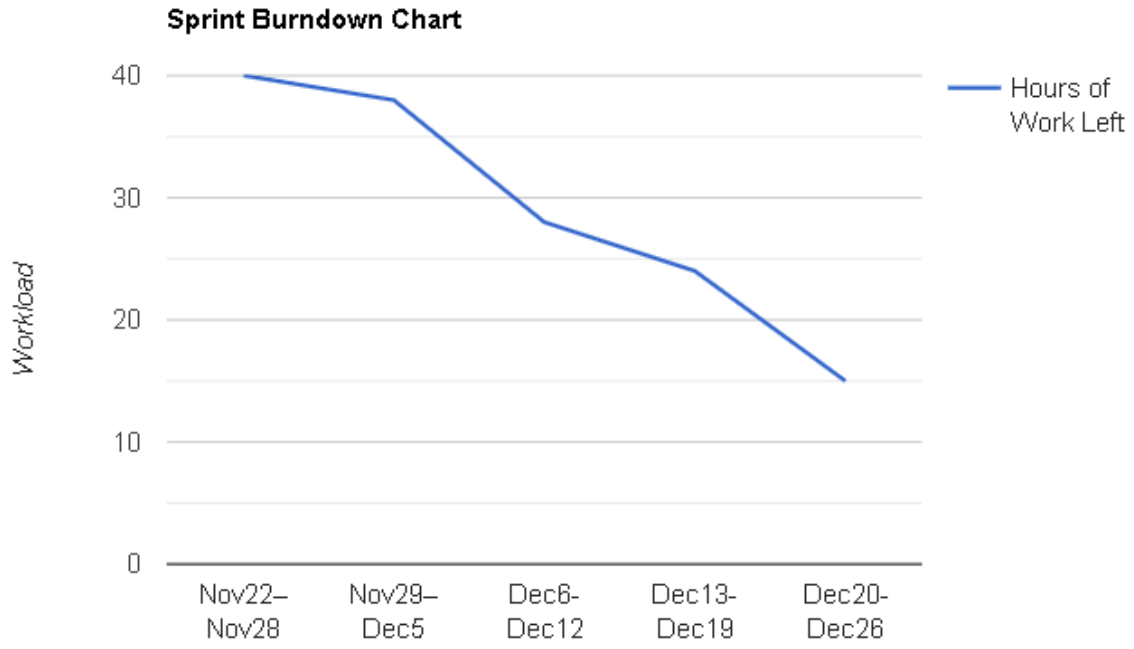

Figure 10: Sprint 2 Burndown Chart

5.2.3. Sprint Review

In this sprint we faced with several problems. While testing GitHub API for java we couldn't find the necessary documentations so that we couldn't retrieve the data as we want. So as a team we decided to develop web service with python. We started to perform our research and tests in this direction.

Also, at the beginning of our planning, we thought that it was better if we use a database, but as a result of our analysis, we realized that there was no need for it. So, we decide not to use a database.

We were also planning to implement some of the basic rule algorithm in this sprint. However, we couldn't manage to implement rule algorithm in this sprint. We left it for the next sprint.

We have started the implementation of our web application using Django in this sprint. We have started to code our template. However, we didn't get to the point we wanted to get in this sprint.

To sum up our workload has doubled in this sprint. Although we did our best, we could not complete all the tasks we wanted due to our busy schedules. Thus we fell behind our schedule.

5.2.4. Sprint Retrospective

What went well in the sprint?
We have made some decisions that are of great importance for the future of our project. Also, helping and communication within the team was generally at a high level.

What went wrong in the sprint?
As a team we couldn’t coordinate well in the tasks. falling behind in web service prevented progress in web application at some points. Also, some changes we made to our project caused us to fall behind in the schedule.
Due to our busy schedule as a team, we were unable to complete some of the tasks we needed to complete in this sprint.

What did we learn?
In this sprint, we realized that as a team, we need to move forward in a more planned manner. We learned that the disruption of one task caused the disruption of other works. For this reason, we must take the communication within the team to the highest level.

How should the next sprint play out?
Since we haven’t managed to finish all the tasks as planned this will further increase the workload on team members in the next sprint. However, we are aware that if we handle the tasks in a more organized way, this will not be a problem for us. In the next sprint, we plan to be more organized and learn from our mistakes that we done in this sprint.

6. Project Estimation

6.1. Project Estimation with COCOMO Model

Program Characteristics	Low Complexity	Medium Complexity	High Complexity
Number of inputs	14 x 3 = 42	1 x 4 = 4	0 x 6 = 0
Number of outputs	2 x 4 = 8	0 x 5 = 0	5 x 7 = 35
Inquiries	5 x 3 = 15	0 x 4 = 0	0 x 6 = 0
Logical Internal files	1 x 7 = 7	0 x 10 = 0	5 x 15 = 75
Unadjusted function-point total			236
Influence Multiplier/VAF			1.13
Adjusted Function-point Total			289.28

Figure 11: Calculation Of AFPT

General System Characteristics

No	GSC	DI
1	Data Communications	5
2	Distributed data processing	5
3	Performance	3
4	Heavily used configuration	1
5	Transaction Rate	5
6	On-Line data entry	2
7	End-user efficiency	5
8	On-Line update	0
9	Complex Processing	4
10	Re-usability	5
11	Installation ease	0
12	Operational ease	5
13	Multiple sites	5
14	Facilitate change	3

Figure 12: General System Characteristics

Brandon Simplified Term	Total Number	Multiplier	Total Functional Points
Number of Places we input values in	15	Times 3	45
Number of places read only	5	Times 4	12
Database queries used to produce simple information	0	Times 3	0
Tables in AX for main query	0	Times 7	0
Outside data sources for information	1	Times 5	5
Business Rules	XXXXXXXX	XXXXX	XXXXXXXXXXXXX
Total functional points			70

Total Degree of Influence = 48
Value Adjust Factor (VAF) = 1.13
(TDI x 0.01) + 0.65 = 48 x 0.01 + 0.65
Function Point Count = 70 x 1.13 = 79.1

$$KDSI = \frac{289.28 * 20}{1000} = 5.7856$$

Development Mode is Organic:
a = 2.4, b = 1.05, c=0.38

$$Man - Month = 2.4 * 5.7856^{1.05} = 15.16$$

$$TDEV = 2.5 * 15.16^{0.38} = 7.02$$

Figure 13: Calculation of TDEV and other parameters

6.1. Project Estimation with Jones's Model

Kind of Software	Best in Class	Average	Worst in Glass
Systems	0.43	0.45	0.48
Business	0.41	0.43	0.46
Shrink-wrap	0.39	0.42	0.45

Our ATFP is 289.28

$$Estimated\ Effort\ (man-month) = ((289.28)^{(3*0.42)})/27 = 46.76$$

$$Rough\ Schedule\ Estimate = (289.28)^{0.42} = 10\ months$$

Figure 14: Project Estimation with Jones's First-Order Effort Estimation

7.7. Project Management

7.1. Milestones and Tasks

List of milestones for this semester :

- Understand how GitHub API works.
  - We should understand the principles of GitHub, and how it works to work on it better.
  - Then, we have to examine different types of APIs for GitHub, and select the best of them for us.
  - There are different programming languages for us to use while creating a service, so we need to select the right one which has a better GitHub API.

- Create a set of rules to be checked.
  - Rules need to be created by us.
  - There will be functions for each rule to be applied.
  - Design pattern of the rule system needs to be selected.
  - Rules should depend on GitHub API's specialities.

- Construct a sample Git repository to be used for the incremental development of the web service.
  - Repository should be created first.
  - Name of the repository should be the same as the name of the project.
  - All team members should be added to the repository.
  - All team members should create a branch to work on it.

- Create front-end prototype
  - Drawings or any other simple type of prototype should be created.
  - Prototype should be discussed.
  - Professional prototype should be demonstrated.

- Create a demo that can communicate with GitHub by using GitHub API.
  - Basic back-end service should be constructed.
  - Decide the programming language for back-end service.
  - Decide the back-end API for service.

- Create a demo that has a simple GUI.
  - Basic front-end service should be constructed.
  - Decide the front-end API.

List of tasks for this semester :

- Watching an introductory lecture video for Git and GitHub.
  - 18/10/2021 - 31/10/2021
- Reading some introductory Git resources.
  - 18/10/2021 - 31/10/2021
- Reading about some Git workflows.
  - 18/10/2021 - 31/10/2021
- Reading about errors and poor practices of students in using Git.
  - 18/10/2021 - 7/11/2021
- Defining functional and non-functional requirements.
  - 18/10/2021 - 14/11/2021
- Writing an example use-case text
  - 18/10/2021 - 14/11/2021
- Writing domain requirements.
  - 18/10/2021 - 14/11/2021
- Creating structured use case diagram
  - 18/10/2021 - 14/11/2021
- Creating sequence diagram with major use cases
  - 18/10/2021 - 14/11/2021
- Creating a context model
  - 18/10/2021 - 14/11/2021
- Creating an Architectural model
  - 18/10/2021 - 14/11/2021
- Creating a process model
  - 18/10/2021 - 14/11/2021
- Describing first version of GUI with home page and report page
  - 18/10/2021 - 14/11/2021

Estimated duration in terms of weeks :

- Understand how GitHub API works
  - 18/10/2021 - 24/10/2021
  - Task Dependency :
    - This task has to be done first.

- Create a set of rules to be checked
  - 25/10/2021 - 31/10/2021
  - Task Dependency :
    - This task should be done before “Creating Git repository”.

- Create front-end prototype
  - 25/10/2021 - 31/10/2021
  - Task Dependency :
    - This task has to be done first for front-end development.

- Construct a sample Git repository to be used for the incremental development of the web service.
  - 01/11/2021 - 07/11/2021
  - Task Dependency :
    - This task has to be done before starting to create a back-end service and front-end.

- Create a demo that can communicate with GitHub by using GitHub API.
  - 08/11/2021 - 21/11/2021
  - Task Dependency :
    - After completing previous tasks, we can start building our back-end service.

- Create a demo that has a simple GUI.
  - 08/11/2021 - 21/11/2021
  - Task Dependency :
    - Front-end should be done after the completion of the prototype of GUI.

Who are responsible for each task :

- Understand how GitHub API works
  - Ziya Taner Keçeci
  - Mahmut Ali Şahin
  - Doğukan Çatal
  - Yusuf Mete Kabakçı

- Create a set of rules to be checked
  - Ziya Taner Keçeci
  - Mahmut Ali Şahin

- Create front-end prototype
  - Doğukan Çatal
  - Yusuf Mete Kabakçı

- Construct a sample Git repository to be used for the incremental development of the web service.
  - Mahmut Ali Şahin
  - Doğukan Çatal
  - Yusuf Mete Kabakçı
  - Ziya Taner Keçeci

- Create a demo that can communicate with GitHub by using GitHub API.
  - Ziya Taner Keçeci
  - Mahmut Ali Şahin

- Create a demo that has a simple GUI.
  - Doğukan Çatal
  - Yusuf Mete Kabakçı

7.2. Gantt Chart

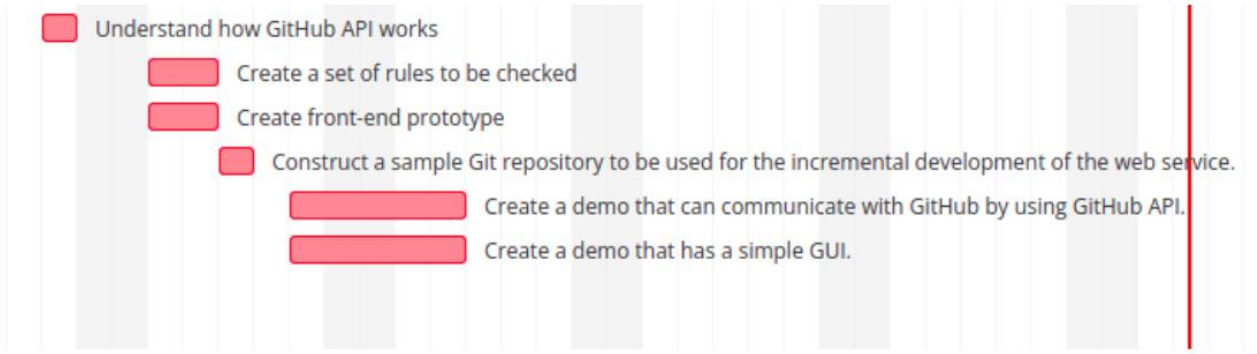

Figure 15: Gantt Chart