**GitHub Analytics**

DISSERTATION

Submitted in partial fulfillment of the requirements of the

M. Tech. Software Engineering Degree programme

By

**NIKESH T T**

**(2018HS70029)**

Under the supervision of

**LALLU ANTHOOR**

**(Senior Developer)**

Dissertation work carried out at

SAP Labs, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

June, 2022

**GitHub Analytics**

DISSERTATION

Submitted in partial fulfillment of the requirements of the

M. Tech. Software Engineering Degree programme

By

**NIKESH T T**

**(2018HS70029)**

Under the supervision of

**LALLU ANTHOOR**

**(Senior Developer)**

Dissertation work carried out at

SAP Labs, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

June, 2022

**Acknowledgement**

I feel proud to present my dissertation project on the topic “GitHub Analytics”.

I would like to show my gratitude for Lallu Anthoor, Senior Developer, SAP for investing time from his busy schedule in mentoring me.

I extend my sincere and heartfelt thanks to my examiners, Sudeep Sukumar & Prof. Bhaskar K for providing me with the right guidance and advice at the crucial junctures and for showing me the right way.

In preparation of this project, I had to take the help and guidance from many online communities which helped me to get familiar with the technology and its implementations.

I would also like to expand my gratitude to my teammates and colleagues who have directly and indirectly helped me whenever I faced any blocker.

This Mid Term report is for my undergoing course of Birla Institute of Technology & Science, Pilani of WILP course in M.Tech Software Engineering**.**

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

SECOND SEMESTER 2021-22

SESAP ZG629T DISSERTATION

Dissertation Title : GitHub Analytics

Name of Supervisor : LALLU ANTHOOR

Name of Student : NIKESH T T

ID No. of Student : 2018HS70029

# **Abstract**

A large number of organizations are using GitHub as their primary source for collaborative development. GitHub offers a great support for enabling development towards a common goal when it comes organizations having a huge employee strength. One of the examples is the trunk-based development where the changes are made in small increments in separate child branches and merged into the master branch once the development activity is complete. Due to the huge number of robust features which GitHub offers many companies uses GitHub as part of their tech stack.

When it comes to cloud-based development it is important to ensure the cloud qualities of the product. There is a good amount of metadata generated as part of any operations which are performed in the GitHub. If we enable proper ways for extracting this metadata many useful information can be derived using the metadata which will help in ensuring the cloud quality of the product.

GitHub Apps are a great way to retrieve the metadata from GitHub.

## **Overview of GitHub Apps**

* Modern way for third party application or users to integrate with the GitHub via the API (i.e., building integrations on top of GitHub)
* Can be considered as a bot that you give access to different objects within your GitHub repos and allow those bots to automate/check things for you

## **Benefits of GitHub Apps**

* Autonomy - The best way for third parties to act autonomously on protected GitHub resources
* Improved security model - Tokens expire for additional security
* Dedicated rate limits
* Used by GitHub to develop its own products
* Offers significant benefits over OAuth app for third party integrators
* Supported by rich ecosystem of libraries and tools
* Granular permissions
* Singular WebHook for all events (across different organizations and repositories)
* Grantable on individual resources (e.g., giving access to only one repo in the organization)
* Better insights to user identity

Once the metadata is retrieved it could be transformed to extract out the useful information and to perform various data analysis on the derived information. This useful information can be visualized in a central UI which will help in ensuring many of the cloud quality aspects of the product which is being developed.

# **List of Symbols & Abbreviations Used**

| **Term** | **Definition** |
| --- | --- |
| GitHub | GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. |
| GitHub Repository | Repositories in GIT contain a collection of files of various different versions of a Project |
| PR | A pull request is an event in Git where a contributor asks a maintainer of a Git repository to review code they want to merge. |
| GitHub Webhooks | GitHub Webhooks allow you to create or set up integrations on GitHub server that subscribe to specific events. |
| GitHub App | An individual who contacts the SaaS application to delete his or her personal data. |
| JPA | Java Persistence API |

**List of Tables**

* Project Plan
* Plan for Remainder

**List of Figures**

|  |  |  |
| --- | --- | --- |
| Sl. No. | Name | Page No. |
| 1 | Figure 1: Architecture Diagram | 12 |
| 2 | Figure 2: PR Quality Metrics | 14 |
| 3 | Figure 3: Branch Quality Metrics | 15 |
| 4 | Figure 4: Developer Efficiency Quality Metrics | 15 |
| 5 | Figure 5: Sample ER Diagram of the Database | 16 |
| 6 | Figure 6: Class Diagram | 17 |
| 7 | Figure 7: Home Page | 18 |
| 8 | Figure 8: Radar Chart with Developer Efficiency Quality Metrics | 18 |
| 9 | Figure 9: Bar Chart PR Quality Metrics | 19 |

# **Table of contents**

[Abstract 4](#_Toc106230644)

[Overview of GitHub Apps 4](#_Toc106230645)

[Benefits of GitHub Apps 4](#_Toc106230646)

[List of Symbols & Abbreviations Used 6](#_Toc106230647)

[Table of contents 7](#_Toc106230648)

[Introduction & Background 8](#_Toc106230649)

[Business Process Flow 9](#_Toc106230650)

[Problem Statement 10](#_Toc106230651)

[Objective of the Project 11](#_Toc106230652)

[Uniqueness of the Project 11](#_Toc106230653)

[Scope of Work 11](#_Toc106230654)

[Solution Architecture 12](#_Toc106230655)

[GitHub App Installation 13](#_Toc106230656)

[GitHub Analytics Service 13](#_Toc106230657)

[Quality Metrics 14](#_Toc106230658)

[Database Layer 16](#_Toc106230659)

[Service Layer 17](#_Toc106230660)

[UI Layer 18](#_Toc106230661)

[Conclusion 20](#_Toc106230662)

[Future Scope 20](#_Toc106230663)

[Resources Needed 21](#_Toc106230664)

[Project Plan & Deliverables 22](#_Toc106230665)

[Key challenges faced during the project 22](#_Toc106230666)

[Plan for Remainder of the Project 23](#_Toc106230667)

# **Introduction & Background**

The GitHub analytics tool will help the user to collect various meta information regarding multiple aspect of the GitHub repositories owned by the user. The tool will help to transform the collected information and derive many useful insights out of it. Meta information includes the events raised by the GitHub while performing create, update, or delete operations on the GitHub repositories. Following are some of the examples of events raised with the meta information: a new branch is created in the repository – CREATE event, a new pull request is created - PULL\_REQUEST event, user adds a comment to one of the existing issues – ISSUE\_COMMENT event. Using the extracted data we can derive many meaningful insights like state of open PRs, inactive branches in a repository, quality of the PR reviews, ownership of different modules, stability of the pipeline.

Existing process of collecting the information involves a lot of manual work. Activities such as monitoring the list of open PRs, monitoring the state of available branches in a GitHub repository, ensuring the quality of PR reviews, ensuring the stability of the pipeline etc. requires the user to manually collect the data related to each of these categories from the GitHub.

Some of the available solutions involve active monitoring of the GitHub data which put a lot of pressure on the GitHub server.

This application will help the user to derive meaningful insights and analytics from the GitHub using a passive way of collecting the data. This will also reduce the load on the GitHub server side.

# **Business Process Flow**

The Business Process Flow for the GitHub analytics tool is as follows:

* Data Extraction - Passive data extraction using GitHub Webhooks
  + Creation of GitHub App with the required configurations required for the data extraction
  + Installation of the GitHub App in the target organizations
  + Querying the GitHub APIs to fetch the initial set of data
  + Passive data extraction via GitHub Webhooks
* Data analysis – Data analysis and generation of meaningful insights
  + Persistence of the useful data
  + Defining SQL Views to fetch aggregated results from the bulk data at run time
  + Cleanup of unused data
* Visualization - This feature allows the users to view the derived information using graphs and tables in a centralized UI
  + Plotting various graphs and table using the available SAP UI5 libraries to make best sense out of the available information

# **Problem Statement**

Existing process of collecting meta data from GitHub involves lot of manual work and it compels the user to manually collect the data related to each of these categories from the GitHub.

Some of the available alternative solutions involved active monitoring of the data:

* Active monitoring require user to query the GitHub APIs to fetch the data, It involves:
  + A lot of data transfer
  + Good amount of processing on the server side
* Data is fetched in a scheduled manner during fixed time periods. Hence the real-time data is not used for the analytics. The fetched data is already old.
* Users need to fetch the complete set of data every time

# **Objective of the Project**

The objective of the project are as follows:

* To gather the GitHub repository data using passive methods
* Deriving meaningful insights from the gathered data
* Building a centralized UI for the visualization of the information

# **Uniqueness of the Project**

There are already existing tools to gather the available GitHub data through active monitoring.

The proposed solution is unique in many ways. Here data is gathered using passive data collection methods to reduce the load on the GitHub server. Instead of displaying the gathered data as it is, different aspects of the gathered data is analyzed to derive many useful insights such as state of open PRs, inactive branches in a repository, quality of the PR reviews, ownership of different modules, stability of the pipeline. Aggregated results are calculated using the gathered data. Derived meaningful information and the aggregated results are displayed on a centralized UI so that user will be able view all of the information at a single place.

Since the data extraction and the analysis process is dynamic in nature user does not have to put any additional manual effort here. Using Webhooks, the data will be transferred as delta to the baseline, we need to query the system only once to set the initial state. Rest of the changes will come as delta via the Webhook deliveries.

Using the extracted data we can derive many meaningful insights like state of open PRs, inactive branches in a repository, quality of the PR reviews, ownership of different modules, stability of the pipeline

# **Scope of Work**

The scope of this dissertation is to make use of the available GitHub Enterprise Server APIs to extract the useful GitHub repository data and perform various analysis on the collected data to generate meaningful insights and analytics. The data extraction is passive using Webhooks. This way we are not adding any additional load on the GitHub Enterprise Server. The derived information will be visualized in a meaningful and useful way using static and dynamic time series charts so that the consumer will be able to take necessary actions based on the information.

# **Solution Architecture**

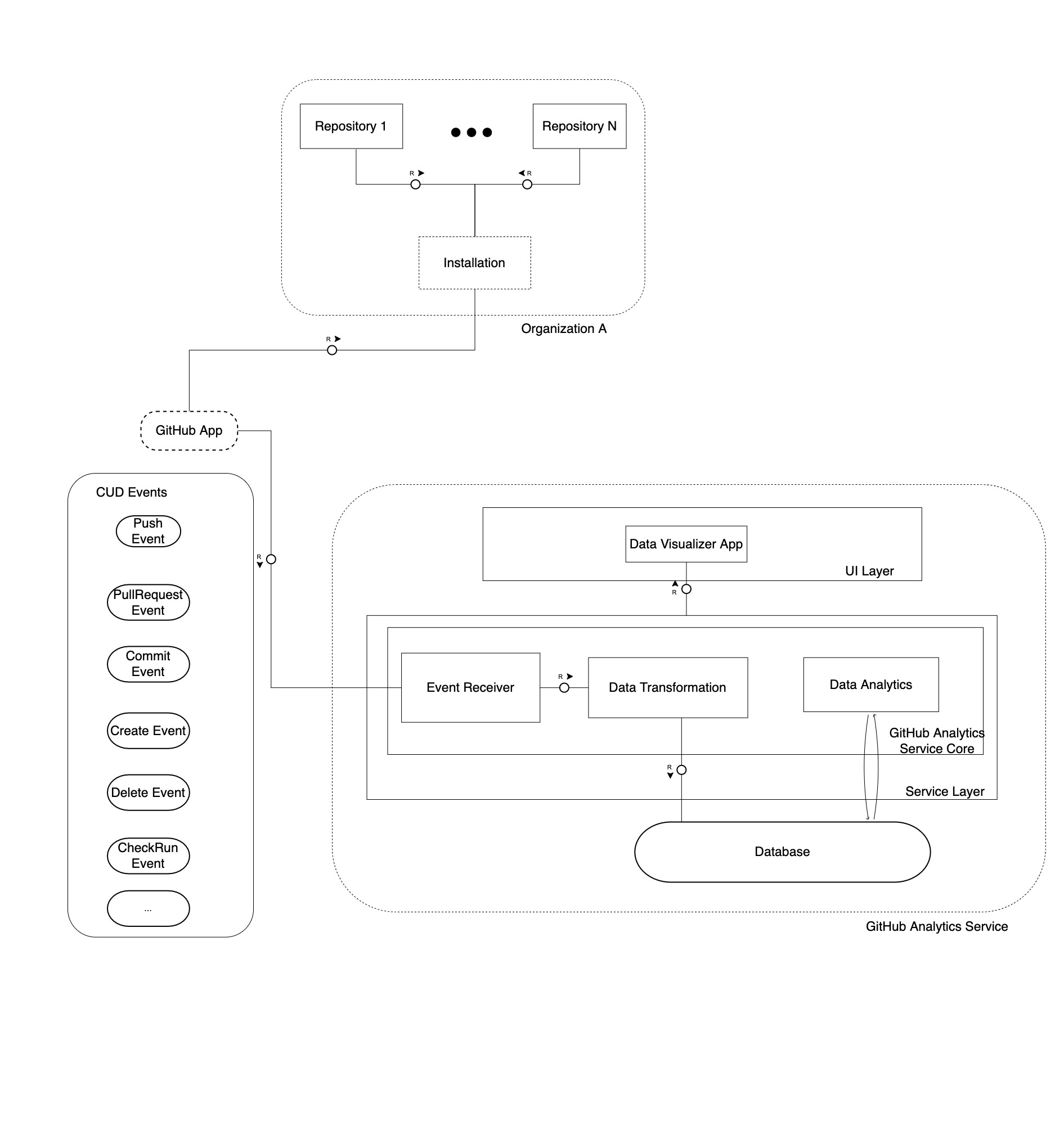


Figure : Architecture Diagram

## **GitHub App Installation**

* Connects a GitHub app to one or more repositories owned by organization or user
* Here the two installations on the different organizations are independent, so they are isolated from each other
* But they point back to the same GitHub app
* Permissions and WebHook handlers are shared

## **GitHub Analytics Service**

* GitHub Analytics Service can be majorly categorized into three components
  + UI Layer – Visualization of the information
  + Service Layer – Receiving and processing of the data
  + Database Layer – Persistence of the transformed data

When create, update or delete operations happens in any of the repositories where the GitHub App is installed, it will raise events to the endpoint which is configured as the WebHook URL. As an example, we could consider the following scenarios:

* A new branch is created in the repository – CREATE Event
* A new pull request is created - PULL\_REQUEST Event
* User adds a comment to one of the existing issues – ISSUE\_COMMENT Event

These events will be received by the controller layer present inside the GitHub Analytics Service Core. Once the payload is received Payload Conversion Service takes care of mapping and converting the payload with the help of predefined model classed. Data transformer service takes care of converting the data to extract out the useful information from the data, it also interacts with the Data Analytics module to generate the aggregated results.

Once the data is transformed to the desired format it is persisted inside the data base layer.

UI layer takes care of representing the useful information to the user in a centralized UI by plotting various graphs and table. UI layer also interacts with the service layer in order to fetch the persisted data.

## **Quality Metrics**

Quality metrics are objective ways to measure, evaluate, and monitor the cloud development and delivery quality. The below given figure is a mind map model representation of the useful meta information associated with a pull-request event from GitHub. Using this meta information we can determine the **Pull-Request Quality** metrics. Mind map representation of the data will greatly help in identifying the parameters that needs to be extracted from the metadata and for mapping this metadata to various quality metrics.

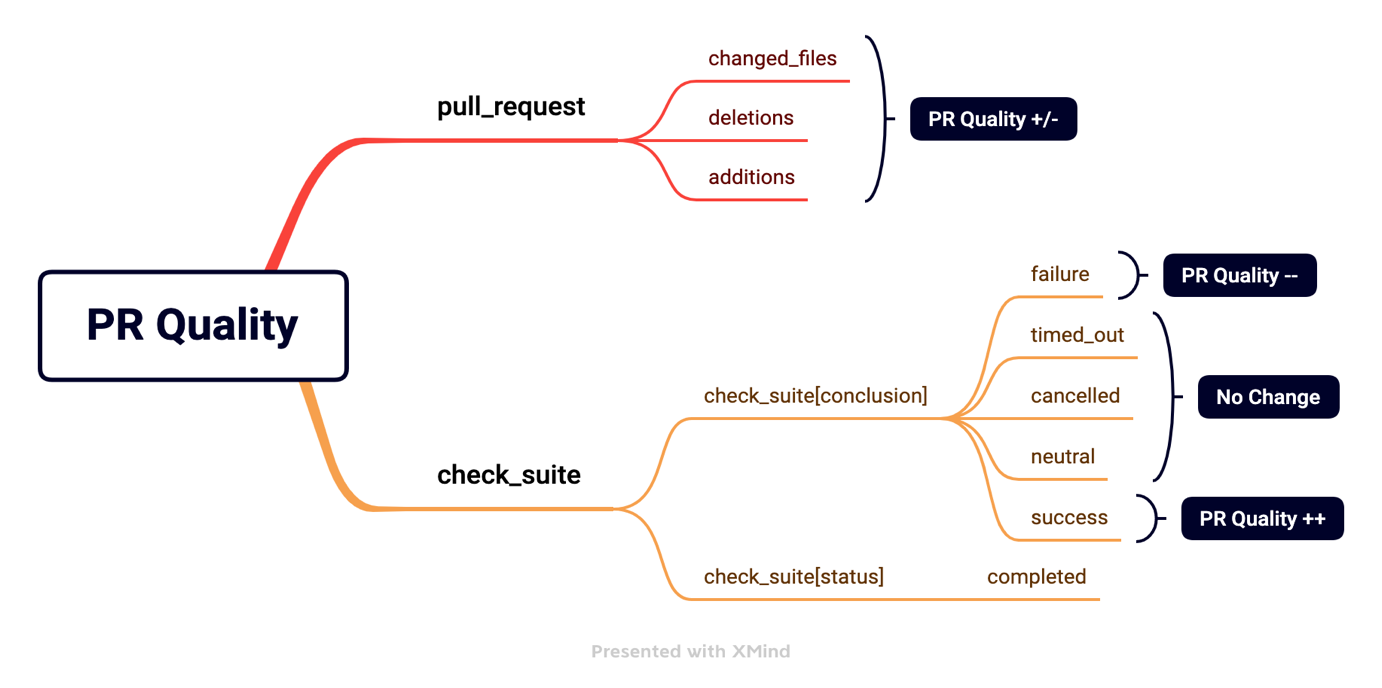


Figure : PR Quality Metrics

The above given mind map represents the pull-request quality metrics. Whenever a pull-request is created or updated in a repository the **pull-request** event is raised. The payload for this event contains multiple meta information which will help to calculate the quality of the pull-request. In trunk-based development model new future are developed in feature branches and a pull-request for these changes are created which can be reviewed by other developers before the pull-request changes are merged to the master branch. The basic rule to ensure the quality in trunk-based development is to bring new feature in small increments. The meta information like number of files changed, number of lines added, number of lines deleted in the pull-request can be mapped to the pull-request quality metrics scores. This will ensure that the new features are brought in small increment. Big bang number of changes will reduce the pull-request quality, code quality, test case coverage quality eventually resulting in poor cloud delivery quality.

Whenever there are additions, deletions, or change of files the corresponding quality metrics scores are calculated based on a predefined boundary value. The following table shows the criticality configuration for pull-request quality metrics.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quality Metric | Positive | Neutral | Negative | Critical |
| Number of files changed  (Big bang PR) | x <= 9 | 10 <= x <= 19 | 20 <= x <= 29 | x >= 30 |
| Number of additions  (Big bang deletions) | x <= 249 | 250 <= x <= 499 | 500 <= x <= 749 | x >= 750 |
| Number of deletions  (Big bang deletions) | x <= 249 | 250 <= x <= 499 | 500 <= x <= 749 | x >= 750 |



Figure : Branch Quality Metrics

The above given mind map represents the branch quality metrics. Whenever any changes are pushed to a branch **push** event is raised. The event payload contains the timestamp information indicating when the changes are pushed to the given branch. In a trunk-based development environment it is important to merge the new features to the master branch in a frequent manner. If a repository contains lots of feature branches, it will become difficult to keep track of the changes. Due to this reason if a branch is inactive and does not contain any relevant changes then it should be removed. The meta information like the last updated time stamp will help to determine whether a given branch is active or inactive.

Different quality metrics can be mapped under a large umbrella to take corrective measures and action items. These quality metrics can be used by a developer or by the people in the product management level. A developer can use the quality metrics to understand his efficiency and take necessary action items. A product manager can analyze the quality of different repositories under the product and ensure that a good repository quality is maintained.

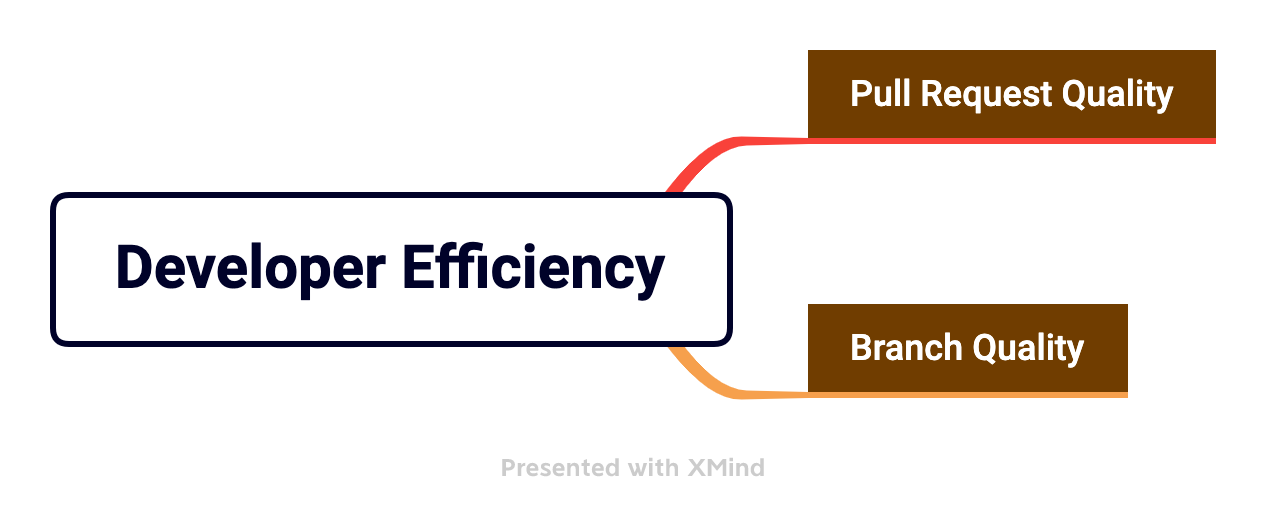


Figure 4: Developer Efficiency Quality Metrics

As you can see in the above figure the two quality metrics **Pull Request Quality** and **Branch Quality** can be mapped under **Developer Efficiency** quality metrics. This will help the developer to understand the action items required from his side to ensure a better efficiency in his daily development activities.

## **Database Layer**

Java Persistence API (JPA) is used to create tables for persistence of the required metadata associated with the webhook events. One of the publicly available libraries (kohsuke github api) which defines the object-oriented representation of the GitHub API is included as a dependency. The library defines classes that is corresponding to the domain model of GitHub (e.g. **GHRepository** and **GHPullRequest**). JPA annotations are added on the model classes to define the entities and association between the entities. The generated tables are normalized.

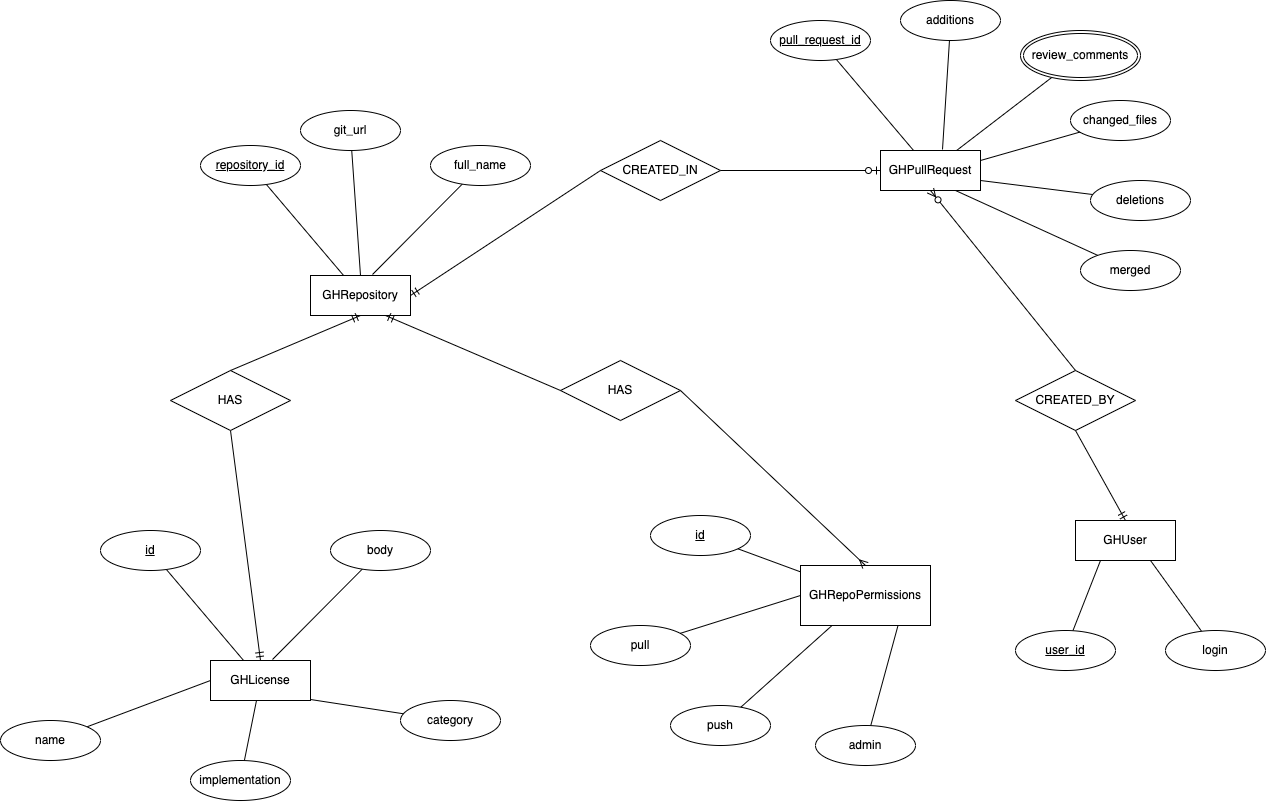


Figure 5: Sample ER Diagram of the Database

## **Service Layer**

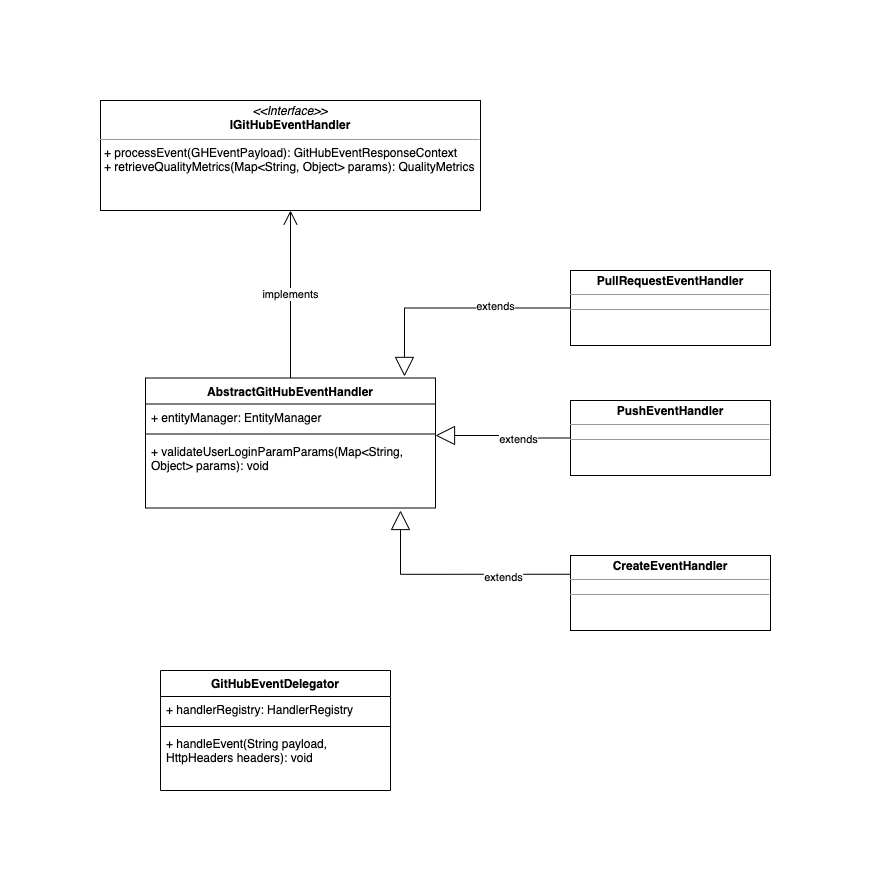


Figure 6: Class Diagram

Different handlers are defined to handle each type of webhook evets from GitHub. Each handler is selected based on the event type by the **GitHubEventDelegator**, here this component acts as an orchestrator for the event flow. Each handler overrides the implementation for persisting the required metadata and for retrieving the quality metrics.

## **UI Layer**

Various charts and diagrams from the SAP UI5 library are used to plot the derived quality metrics data.

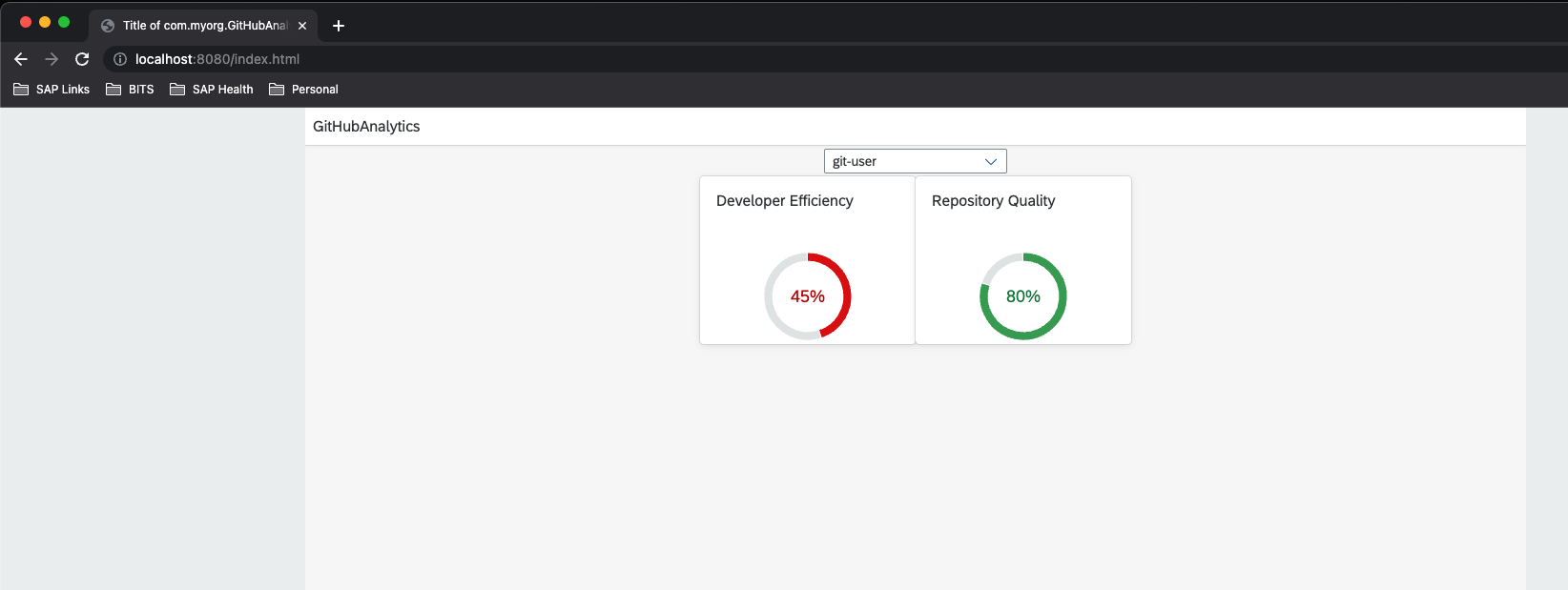


Figure 7: Home Page

Radial charts are included in the home page which helps in identifying the overall Developer Efficiency and Repository Quality scores in a single percentage figure

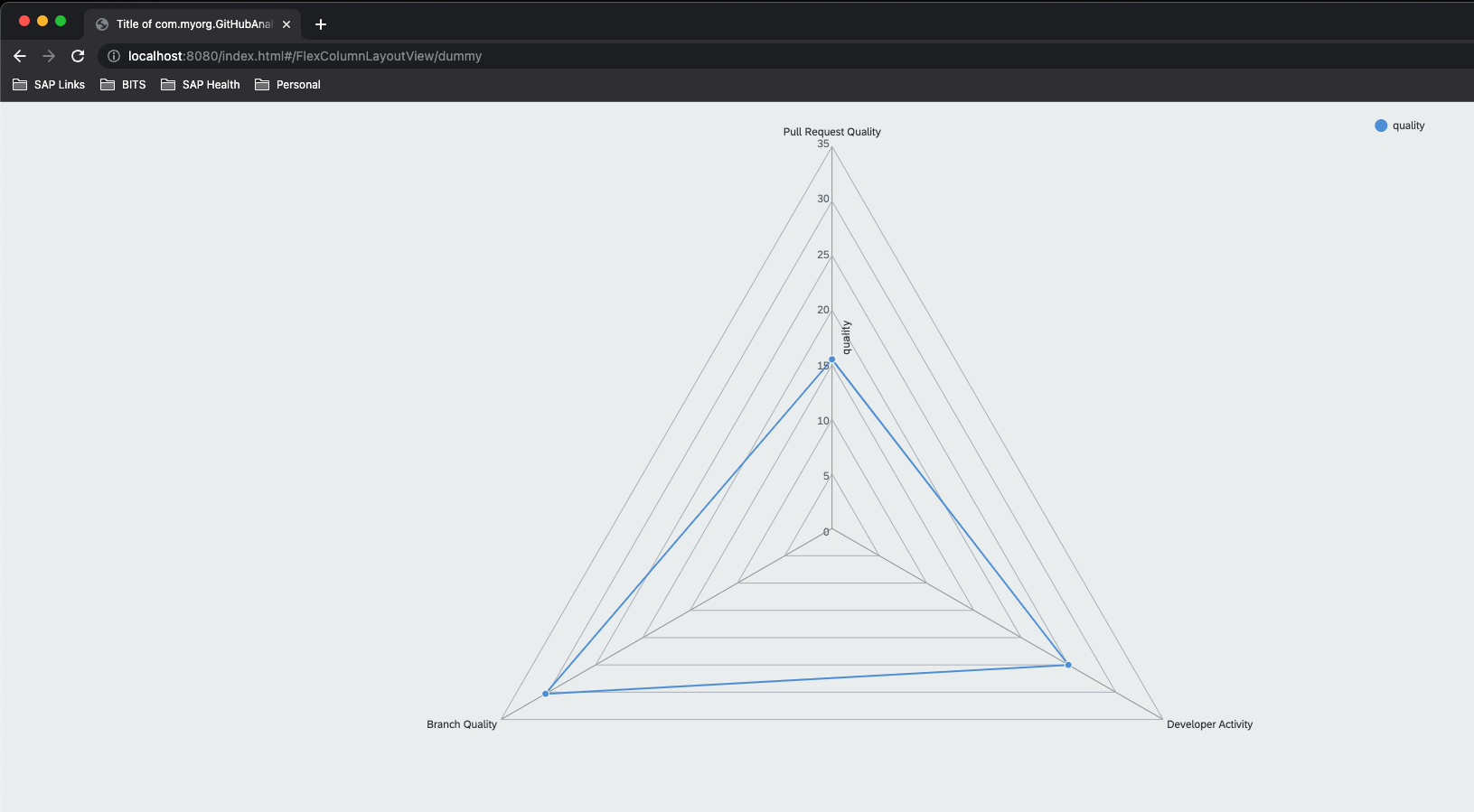


Figure 8 : Radar Chart with Developer Efficiency Quality Metrics

The radar chart helps to display multivariant data in two dimensional form. In the above given figure each axis on the radar chart represents different quality scores which contributes to the Developer Efficiency Quality metrics. The data point on the axis represents the respective quality metrics score.

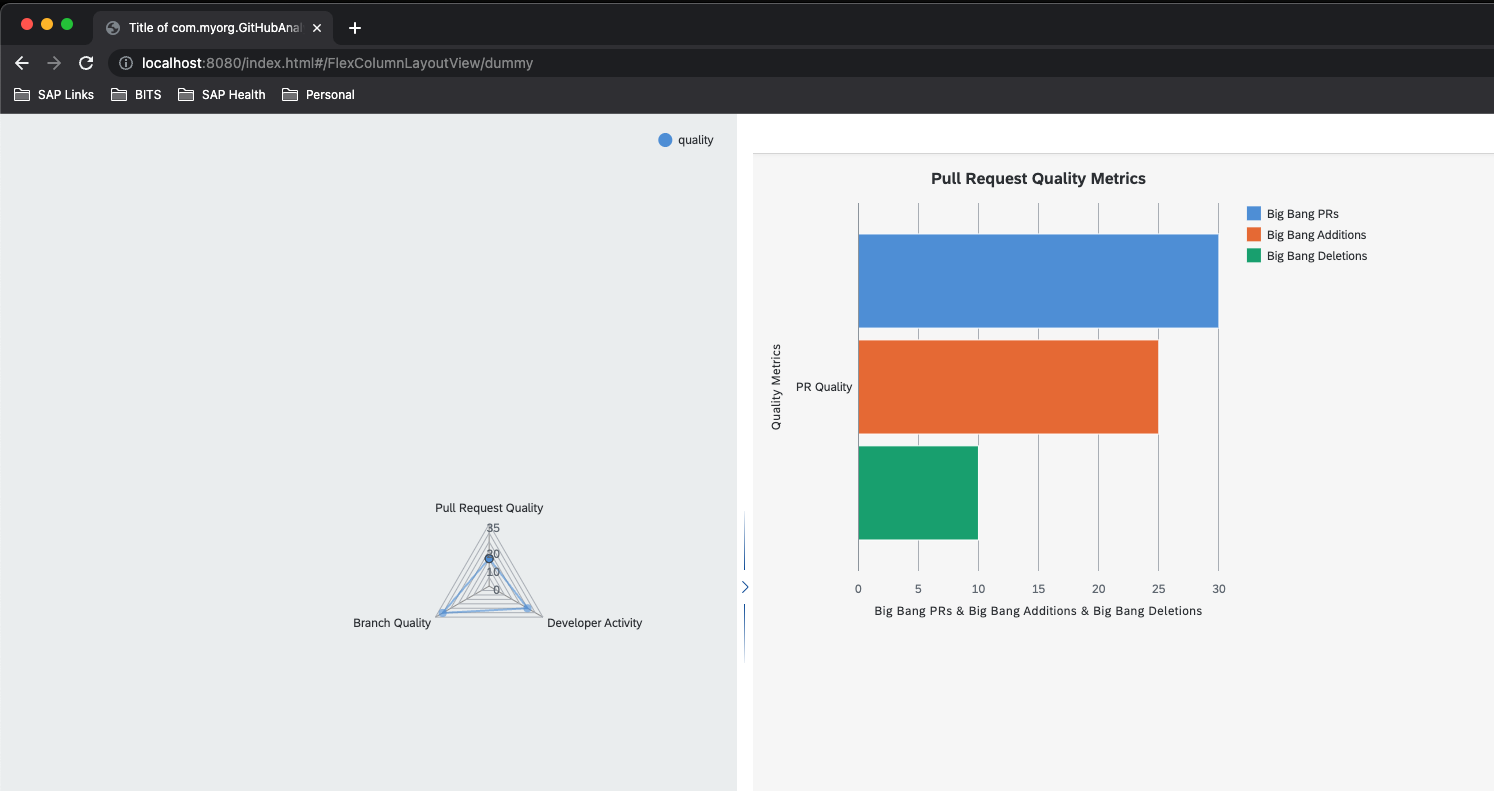


Figure 9: Bar Chart PR Quality Metrics

Once a quality metric score is selected in the radial chart it expands the quality metrics related bar chart view. The bar chart give more detailed insights to the specific quality metrics, including the different individual quality scores contributed to the overall quality metric score.

# **Conclusion**

Many organizations are following the cloud-based product development and delivery model. It is important to ensure the quality of the cloud-based product delivery process. The meta information generated from the GitHub can be used to ensure better quality in the delivery process. Meta information includes the events raised by the GitHub while performing create, update, or delete operations on the GitHub repositories.

These metadata information can be mapped to various quality metrics. Different related quality metrics can be brought under a generic quality metric. The quality metrics helps developers and people in the product management area to get better insights required for ensuring the quality of the cloud delivery process.

The data extraction is passive using Webhooks. This way we are not adding any additional load on the GitHub Enterprise Server. The data will be transferred as delta to the baseline.

# **Future Scope**

GitHub raises events containing various metadata information corresponding to almost all the create, update, and delete operations performed. For the completion of this project only limited number of metadata information is processed to derive the quality metrics. The scope of the project can be expanded in the future to further analyze other metadata information to derive even more useful insights. The webhook events can also be used to trigger any pipeline builds, update a backup mirror, deployment of the application to the production server. You’re only limited by your imagination

# **Resources Needed**

The various resources required for the project are:

Hardware Requirements:

* Java runtime entertainment 11
* UI5 runtime
* Postgres database
* 2GB Memory

# **Project Plan & Deliverables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Serial Number of Task/Phases | Tasks or subtasks to be done | Start Date-End Date | Planned duration in weeks | Specific Deliverable in terms of the project |
| 1 | Identifying available GitHub Enterprise Server APIs | 27/02/’22 – 05/03/’22 | 1 | Detailed list of information about usable GitHub Enterprise Server APIs |
| 2 | Application module to handle GitHub APIs | 06/03/’22 – 12/03/’22 | 1 | GitHub APIs management application module |
| 3 | Explore extractable GitHub repository data | 13/03/’22 – 19/03/’22 | 1 | Information about extractable GitHub repository data |
| 4 | Application module to extract the useful GitHub repository data | 20/03/’22 – 02/04/’22 | 2 | GitHub repository data management application module |
| 5 | Explore various data analysis methods/tools | 03/04/’22 – 16/04/’22 | 2 | Details regarding various data analysis methods/tools in tabular format with pros and cons for each |
| 6 | Data analysis and generation of meaningful insights | 17/04/’22 – 07/05/’22 | 3 | Extracted meaningful insights from the gathered data |
| 7 | Testing and Bug Fixes | 08/05/’22 – 14/05/’22 | 1 | Make the application stable |
| 8 | Explore SAP UI5 based visualization libraries | 15/05/’22 – 28/05/’22 | 2 | Details of SAP UI5 based visualization tools |
| 9 | Visualization of the information using static and dynamic time series charts | 29/05/’22 – 18/05/’22 | 3 | Consumer UI for visualization of the information |

# **Key challenges faced during the project**

* Identifying & categorizing the useful information available as part of GitHub WebHook payload
* Understanding the architecture of the GitHub App
* Developing Application module to handle GitHub APIs

# **Plan for Remainder of the Project**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Serial Number of Task/Phases | Tasks or subtasks to be done | Start Date-End Date | Planned duration in weeks | Specific Deliverable in terms of the project | Status |
| 1 | Identifying available GitHub Enterprise Server APIs | 27/02/’22 – 05/03/’22 | 1 | Detailed list of information about usable GitHub Enterprise Server APIs | Done |
| 2 | Application module to handle GitHub APIs | 06/03/’22 – 12/03/’22 | 1 | GitHub APIs management application module | Done |
| 3 | Explore extractable GitHub repository data | 13/03/’22 – 19/03/’22 | 1 | Information about extractable GitHub repository data | Done |
| 4 | Application module to extract the useful GitHub repository data | 20/03/’22 – 02/04/’22 | 2 | GitHub repository data management application module | Done |
| 5 | Explore various data analysis methods/tools | 03/04/’22 – 16/04/’22 | 2 | Details regarding various data analysis methods/tools in tabular format with pros and cons for each | Done |
| 6 | Data analysis and generation of meaningful insights | 17/04/’22 – 07/05/’22 | 3 | Extracted meaningful insights from the gathered data | Done |
| 7 | Testing and Bug Fixes | 08/05/’22 – 14/05/’22 | 1 | Make the application stable | Done |
| 8 | Explore SAP UI5 based visualization libraries | 15/05/’22 – 28/05/’22 | 2 | Details of SAP UI5 based visualization tools | Done |
| 9 | Visualization of the information using static and dynamic time series charts | 29/05/’22 – 18/05/’22 | 3 | Consumer UI for visualization of the information | Done |

**References**

[1] GitHub API for Java – - Kohsuke Kawaguchi

[Online]

https://github.com/hub4j/github-api/blob/main/src/main/java/org/kohsuke/github/GitHub.java