# *GIT Repository Quality Assessment Tool*

**Group members:**

Sindooja Bhagwan Gajam (2888228)

Radhikaben Rajubhai Talpada Kolipatel (2885283)

Manthanbhai Thakkar Sachdev (2873285)

Vishal Yadav Chatharasi (2870632)

Neha Golekar (2871864)

Neeharika Malampati (2887187)

Disha Surendrakumar Patel (2872217)

**Github Link:** https://github.com/vishalch4466/gitccc232.git

## Table Of Content

1. Introduction ........................................ 3
2. Project Purpose and Vision ........................................ 4
3. Objective ........................................ 4
4. Problem Statement ........................................ 5
5. Solution Architecture ........................................ 5
6. Core Features ........................................ 7
7. Detailed Metrics for Git Repository Quality Assessment Tool ........................................ 8
8. Testing ........................................ 11
9. Development Phases ........................................ 11
10. Screenshots of the Project UI ........................................ 14
11. Future Enhancements ........................................ 15
12. Conclusion ........................................ 16

## Table of Figures

- Figure 1: Architecture Diagram ........................................ 6
- Figure 2: Test Results ........................................ 11
- Figure 3: UI_1 ........................................ 14
- Figure 4: UI_2 ........................................ 14
- Figure 5: UI_3 ........................................ 15
- Figure 6: UI_4 ........................................ 15
- Figure 7: UI_5 ........................................ 15

# 1. Introduction

The effectiveness and efficiency of modern software development teams heavily depend on the quality of their codebases and the collaborative efforts of team members. Git repositories serve as the backbone of version control in software projects, but the absence of comprehensive tools to assess and monitor repository health often results in overlooked inefficiencies, technical debt, and suboptimal team performance.

This tool aims to empower development teams by offering real-time insights into repository health, team productivity, and code quality. By integrating advanced analytics and performance metrics into an intuitive dashboard interface, the tool will enable stakeholders to monitor repository activity, detect potential issues early, and make informed decisions to enhance development workflows. Features such as code quality indicators, commit trends, and team performance tracking will provide actionable feedback, fostering a culture of continuous improvement and collaboration.

This report outlines the motivation, technical design, implementation details, and anticipated impact of the proposed tool, showcasing how it can transform repository management and optimize software development practices.

**Need for a Git Repository Quality Assessment Tool**

Modern software development teams require:

- Comprehensive visibility into codebase quality.
- Objective metrics for team performance.
- Real-time insights into potential code vulnerabilities.
- Data-driven approaches to improve development processes.

## 2. Project Purpose and Vision

### 2.1 Purpose

The Git Repository Quality Assessment Tool was meticulously developed with the primary objective of providing a multifaceted approach to repository evaluation, which includes the automated assessment of repository health through systematic analysis. Additionally, the tool offers objective metrics that serve as benchmarks for assessing the quality of repositories, thereby enabling developers and project managers to identify areas in need of enhancement, and ultimately provides actionable recommendations that can be leveraged for the purpose of improvement.

### 2.2 The primary vision of our Git Repository Quality Assessment Tool is to:

- Empower development teams with actionable insights.
- Provide a holistic view of repository and team performance.
- Facilitate continuous improvement in software development processes.

## 3. Objective

The overarching objective of this initiative is to evaluate repositories based on a carefully selected set of key quality indicators that encompass a range of critical factors, including the cleanliness of the code, the completeness of documentation, and the level of activity in development processes. Moreover, the tool aims to furnish standardized scoring mechanisms that can be utilized by developers and project managers alike, facilitating an objective assessment of the current state of their repositories while simultaneously enhancing collaboration by ensuring that repositories are not only organized but also readily accessible to all potential contributors.

## 4. Problem Statement

In the realm of contemporary software development, Git repositories hold a position of paramount importance, functioning as collaborative platforms that facilitate version control while simultaneously enabling the efficient and systematic management of intricate codebases. The management of these repositories, however, is not without its challenges, as they may encounter a variety of issues that can adversely affect their overall functionality, including but not limited to poor documentation practices, insufficiently detailed commit messages, the presence of inactive branches, and the existence of unstructured codebases that hinder effective navigation and understanding. Given the complexities involved in maintaining high standards within a repository, there emerges a critical need for a comprehensive quality assessment tool that can ensure consistency, enhance maintainability, and ascertain the overall readiness of a repository for contributions from various stakeholders.

## 5. Solution Architecture

### 5.1 Technical Architecture:

**Frontend:** Next.js

- Responsive and dynamic user interface.
- Component-based architecture.
- Recharts for data visualization.

**Backend:** Express

- RESTful API development
- Efficient data processing
- Scalable microservices architecture

**Database:** Supa base

- Real-time database capabilities
- Robust data storage and retrieval
- Seamless integration with Next.js
- Advanced querying and filtering

**Testing:** jest.js

## 5.2 Architectural Diagram

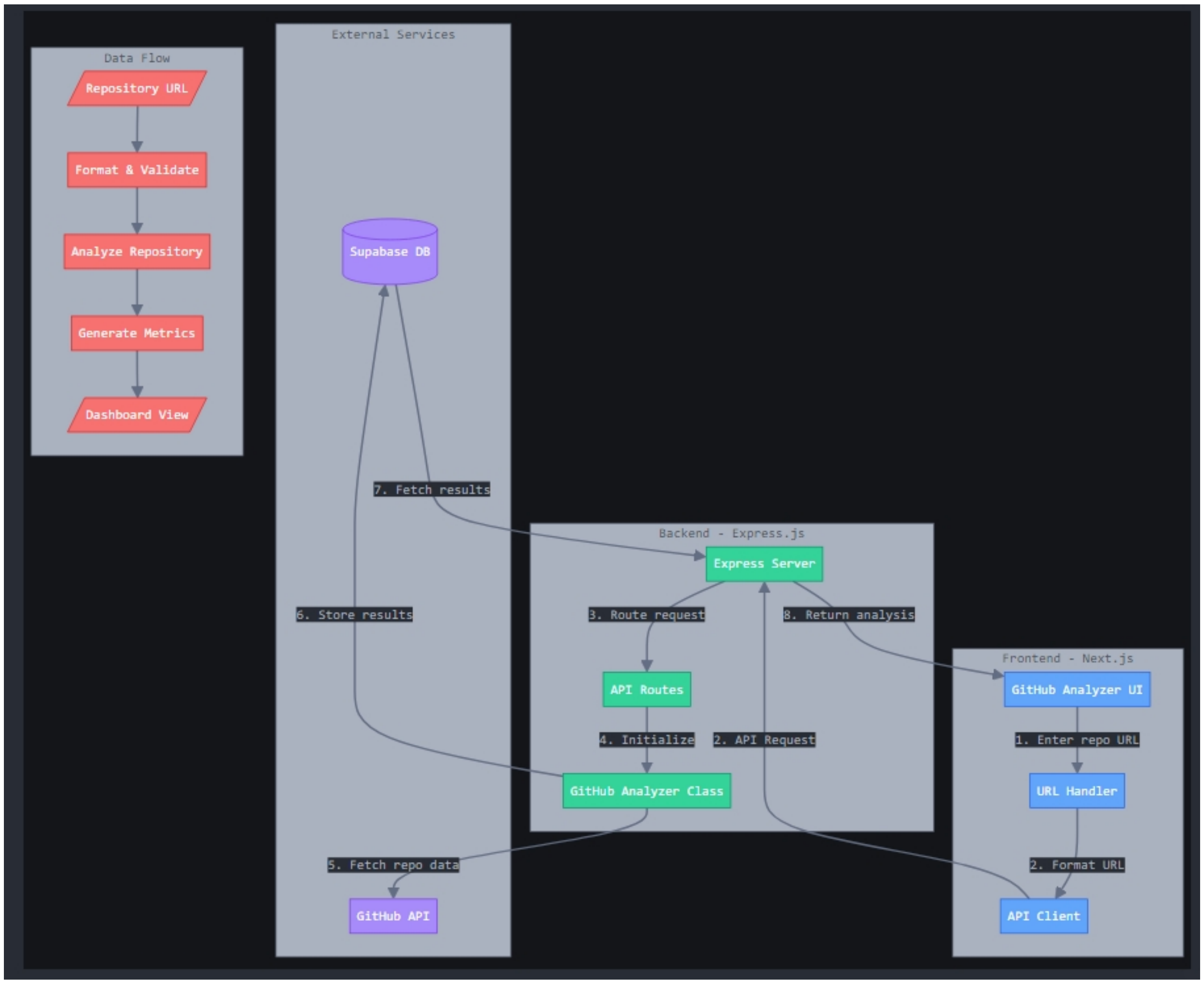


Fig. 1 Architecture Diagram

## 6. Core features:

### 1. Code Quality Indicators:

- Code Complexity Analysis: Highlights sections of the codebase with high complexity, aiding in refactoring decisions.
- Code Duplication Detection: Identifies redundant code to promote modularity and maintainability.
- Linting and Best Practices: Tracks adherence to coding standards and identifies violations that can affect readability and performance.
- Automated Testing Metrics: Provides test coverage statistics and highlights untested sections of the codebase.

### 2. Team Performance Metrics:

- Commit and Contribution Trends: Visualizes individual and team-level contributions over time to identify active participants and potential gaps.
- Pull Request (PR) Insights: Tracks the number, resolution time, and review quality of PRs to improve collaboration and code review efficiency.
- Issue Management Monitoring: Analyzes the creation and resolution of issues to ensure effective prioritization and workload distribution.
- Code Ownership: Maps sections of the codebase to contributors to promote accountability and facilitate team collaboration.

### 3. Real-Time Repository Analytics:

- Provides up-to-the-minute insights into ongoing activities, such as commits, mergers, and branch creations.

    - Detects anomalies, such as unusually high commit activity or unresolved conflicts, to address potential issues before they escalate.
    - Tracks trends in technical debt and repository growth to ensure project scalability.

4. **Visualization Dashboard:**
    - A user-friendly interface designed to present complex data in easily digestible formats like graphs, charts, and heatmaps.
    - Customizable widgets that allow users to tailor the dashboard based on their roles and interests (e.g., developers, managers).
    - Integration with popular third-party tools (e.g., Slack, Jira, or email) to provide notifications and actionable reports.

## 7. Detailed Metrics for Git Repository Quality Assessment Tool

The effectiveness of the Git Repository Quality Assessment Tool depends on its ability to provide actionable metrics that address code quality, team productivity, and overall repository health. Below is a comprehensive list of metrics categorized into Code Quality, Team Performance, Repository Activity, and Predictive Analytics, along with their descriptions and importance.

### 7.1 Code Quality Metrics

These metrics help in identifying the technical health of the codebase and ensuring adherence to best practices.

- Code Complexity:
    - Definition: Measures the cyclomatic complexity of the code to identify overly complicated functions or modules.
    - Importance: High complexity indicates potential maintenance challenges and bugs.

- Code Duplication:
    - Definition: Percentage of duplicated code in the repository.
    - Importance: Highlights areas where code can be refactored to improve maintainability.
- Code Coverage:
    - Definition: The percentage of the codebase covered by automated tests.
    - Importance: Ensures the reliability of the application and reduces the likelihood of introducing defects.

**7.2 Team Performance Metrics**

These metrics track team productivity, collaboration, and workload distribution.

- Commit Frequency:
    - Definition: Average number of commits per day/week by team members.
    - Importance: Indicates team activity and potential productivity bottlenecks.
- Pull Request (PR) Velocity:
    - Definition: Average time taken to review and merge pull requests.
    - Importance: Measures the efficiency of the code review process and collaboration.
- Review Feedback Quality:
    - Definition: Tracks the number of actionable comments left during pull request reviews.
    - Importance: Ensures meaningful code reviews that improve code quality.
- Code Ownership:
    - Definition: Identifies which team members contribute to specific areas of the codebase.

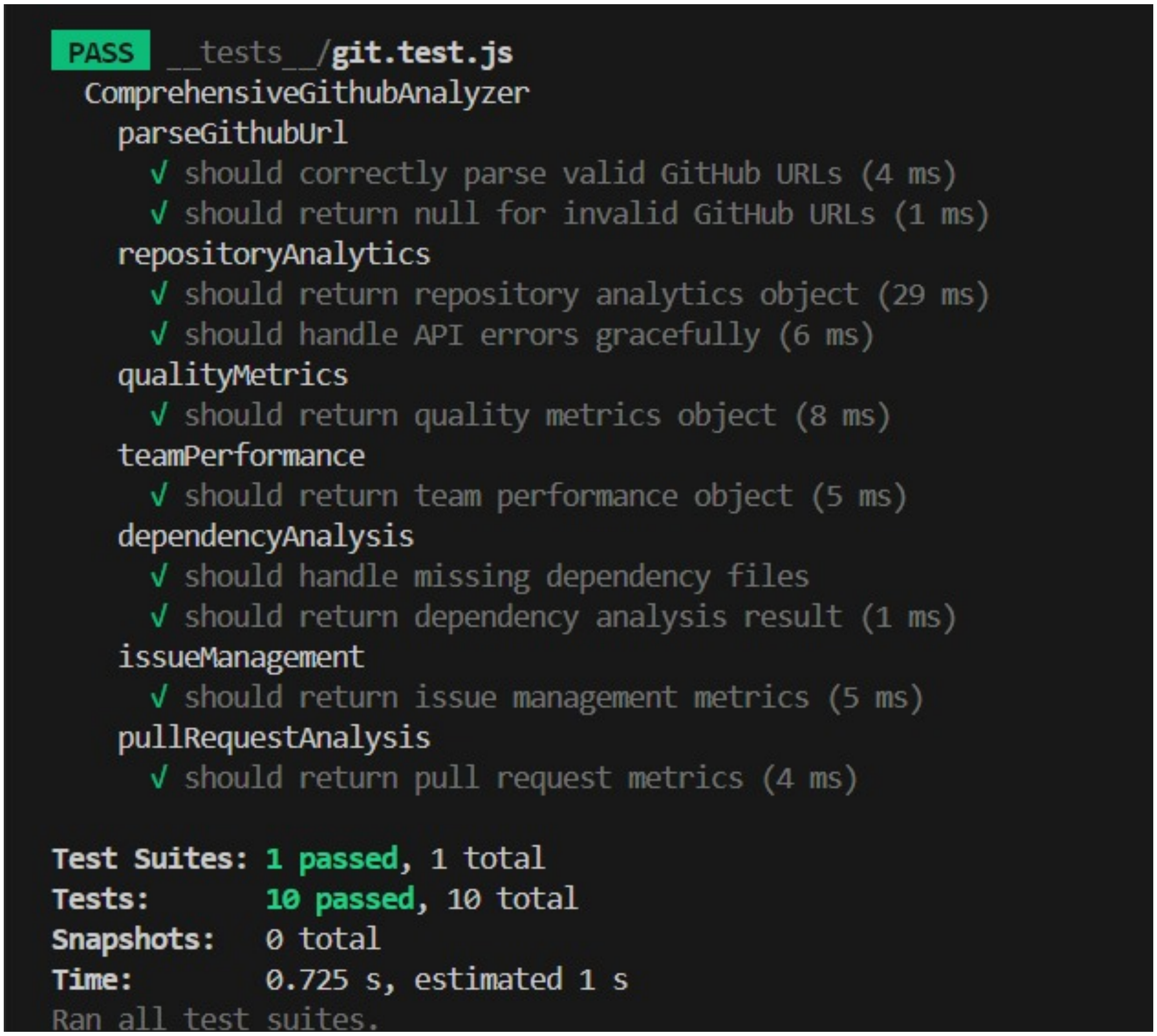
    - Importance: Promotes accountability and helps in managing knowledge silos.
- Issue Resolution Time:
    - Definition: Average time taken to close issues in the repository.
    - Importance: Tracks the team's ability to resolve bugs and implement features efficiently.

**7.3 Repository Activity Metrics**

These metrics monitor the overall activity and trends within the repository.

- Active Contributors:
    - Definition: Number of unique contributors in each timeframe.
    - Importance: Measures engagement and team involvement in the project.
- Commit Trend:
    - Definition: Tracks the pattern of commits over time.
    - Importance: Identifies peaks and lulls in development activity.
- Branch Activity:
    - Definition: Number of active branches and their merge status.
    - Importance: Tracks parallel development efforts and ensures timely merging.
- Merge Conflict Frequency:
    - Definition: Number of conflicts encountered during merges.
    - Importance: High conflict frequency suggests poor branch management practices.
- Churn Rate:
    - Definition: Tracks how often code is edited or rewritten after being added to the repository.
    - Importance: High churn indicates instability or lack of clarity in requirements.

## 8. Testing:



```
PASS  __tests__/git.test.js
  ComprehensiveGithubAnalyzer
    parseGithubUrl
      √ should correctly parse valid GitHub URLs (4 ms)
      √ should return null for invalid GitHub URLs (1 ms)
    repositoryAnalytics
      √ should return repository analytics object (29 ms)
      √ should handle API errors gracefully (6 ms)
    qualityMetrics
      √ should return quality metrics object (8 ms)
    teamPerformance
      √ should return team performance object (5 ms)
    dependencyAnalysis
      √ should handle missing dependency files
      √ should return dependency analysis result (1 ms)
    issueManagement
      √ should return issue management metrics (5 ms)
    pullRequestAnalysis
      √ should return pull request metrics (4 ms)

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        0.725 s, estimated 1 s
Ran all test suites.
```

Fig. 2 Test Results

## 9. Development Phases

The development of the Git Repository Quality Assessment Tool is organized into clearly defined phases to ensure systematic progress, efficient resource utilization, and high-quality delivery. Each phase builds upon the outcomes of the previous one, fostering iterative development and continuous improvement.

**Phase 1: Requirements Gathering and Analysis**

- Objectives:
  - Understand the specific needs of stakeholders, including developers, team leads, and project managers.
  - Identify the key metrics and analytics to be tracked
  - Determine integration requirements with platform like GitHub.
  - Define performance benchmarks and expected outcomes for the tool.
- Deliverables:
  - Comprehensive requirements document.
  - Initial use-case scenarios and workflow diagrams.
  - Feasibility analysis report.

**Phase 2: System Design**

- Objectives:
  - Create a high-level system architecture for the tool using Next.js.
  - Design the database schema to store repository data, performance metrics, and analytics.
  - Develop a modular design for scalability and ease of maintenance.
  - Plan API integration for real-time data fetching from Git repositories.
  - Design mockups and wireframes for the dashboard interface.
- Deliverables:
  - System architecture diagram.
  - Database schema design.
  - API endpoints design documentation.

    - Wireframes and user interface mockups.

**Phase 3: Backend Development**

- Objectives:
    - Set up the Express.js environment for backend development.
    - Develop APIs to interact with Git repositories.
    - Integrate the Supa base for storing repository metrics and user data.
- Deliverables:
    - Fully functional backend with API endpoints.
    - Repository data collection and storage mechanisms.
    - Documentation for backend APIs.

**Phase 4: Frontend Development**

- Objectives:
    - Develop a responsive and user-friendly interface using Next.js
    - Implement customizable dashboard widgets for role-specific metrics.
    - Ensure seamless communication with the backend API for live data updates.
    - Optimize the interface for performance and accessibility.
- Deliverables:
    - Fully functional dashboard interface.
    - Role-based customization features.
    - Frontend-backend integration.

**Phase 5: Testing and Quality Assurance**

- Objectives:
    - Perform unit, integration, and system testing to identify and fix bugs.

- Validate the accuracy of analytics and metrics generated by the tool.
- Optimize the tool for performance, ensuring low latency and smooth operation.

- Deliverables:
  - Test cases and reports.
  - Resolved issues and bug fixes.
  - Finalized and validated tool ready for deployment.

## 10. Screenshots of the project UI:

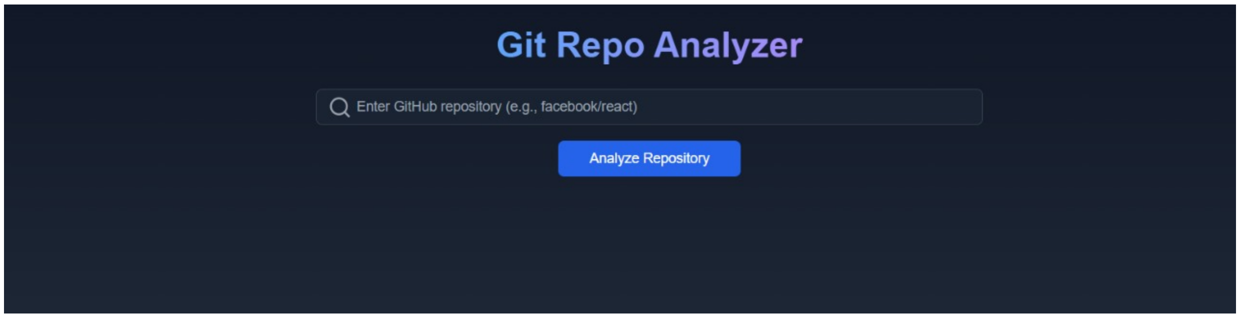
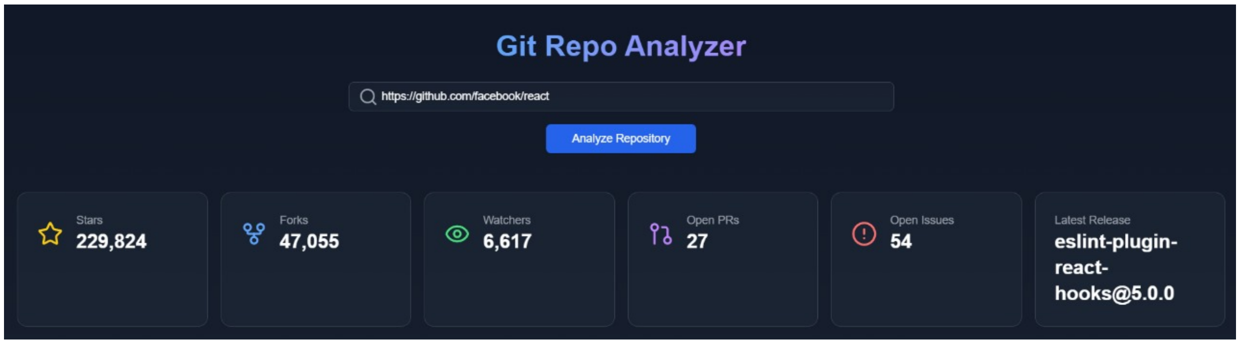


Fig. 3 UI_1



Stars 229,824

Forks 47,055

Watchers 6,617

Open PRs 27

Open Issues 54

Latest Release eslint-plugin-react-hooks@5.0.0

Fig. 4 UI_2

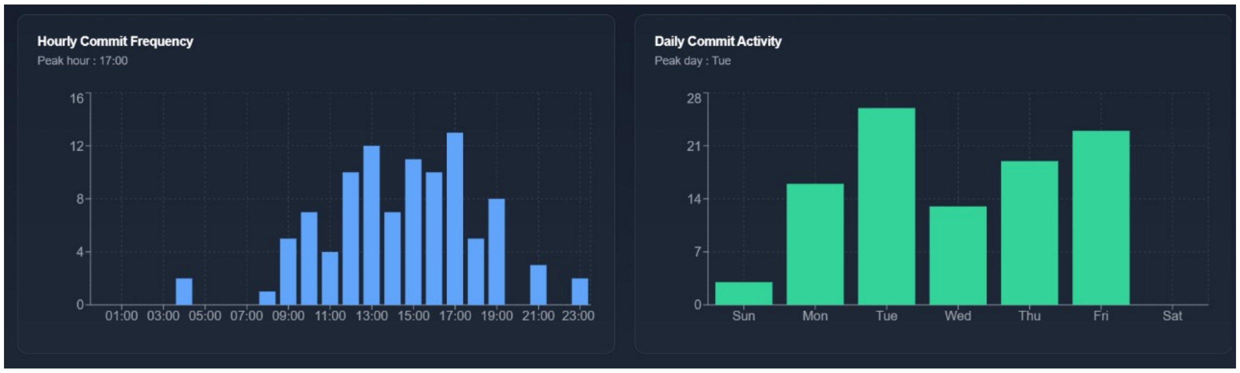


Fig. 5 UI_3

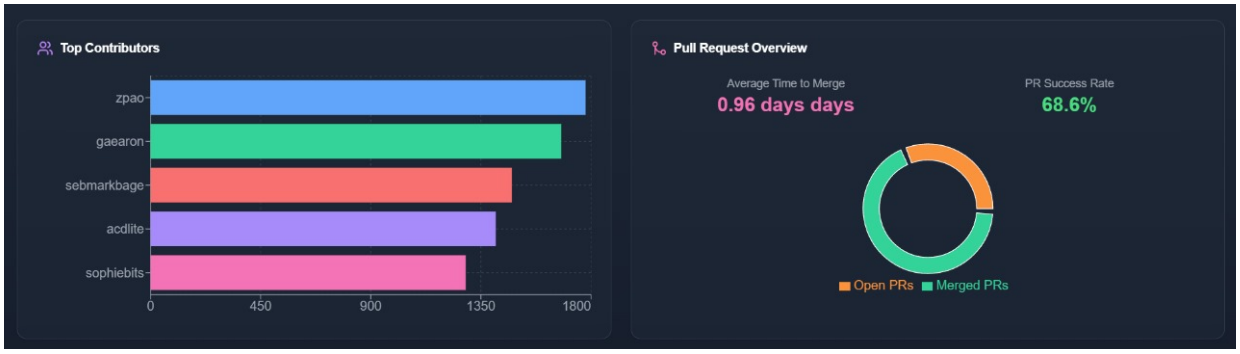


Fig. 6 UI_4

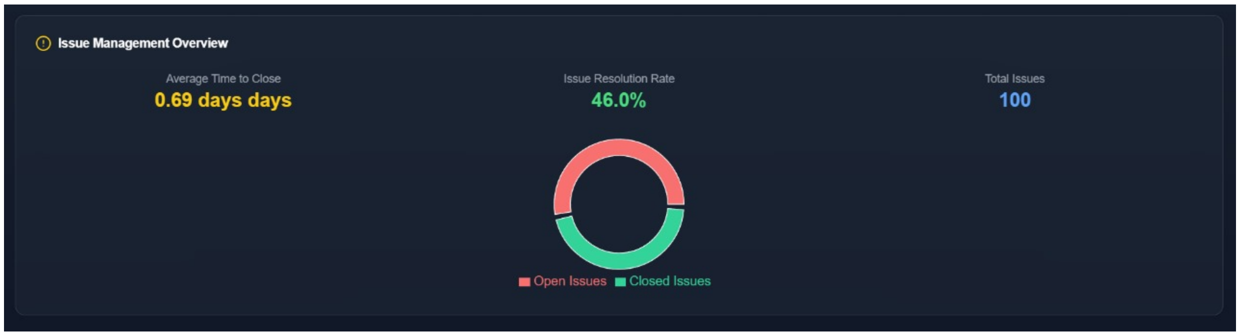


Fig. 7 UI_5

## 11. Future Enhancement

1. **Integration with Additional Platforms:** Expand compatibility to include GitLab, Bitbucket, and other popular version control systems to increase the tool's accessibility for diverse development teams.

2. **Machine Learning for Predictive Analytics:** Implement advanced ML algorithms to predict potential technical debt, identify high-risk modules, and suggest proactive measures for maintaining repository health.
3. **Enhanced Visualization Features:** Introduce 3D visualizations and more interactive widgets to make data interpretation intuitive, particularly for larger codebases with complex branching histories.
4. **Customization and Role-Specific Dashboards:** Develop role-specific dashboards tailored for developers, project managers, and QA engineers, enabling users to focus on metrics relevant to their responsibilities.
5. **Real-Time Collaboration Insights:** Provide deeper insights into collaborative dynamics, such as identifying bottlenecks in pull requests and recommending strategies for improving team productivity.

## 12. Conclusion

The Git Repository Quality Assessment Tool is an innovative and practical solution designed to address the critical challenges faced by software development teams in maintaining code quality and optimizing productivity. This tool integrates real-time analytics, advanced metrics, and predictive insights into a single, intuitive dashboard, enabling teams to monitor repository health effectively and streamline their development processes.

The tool's comprehensive approach to analyzing code quality, team performance, and repository activity provides actionable insights that empower teams to identify risks, enhance collaboration, and prioritize improvements. By incorporating predictive analytics, it offers a forward-looking perspective, helping teams anticipate potential issues and make data-driven decisions to mitigate them proactively.

The tool’s scalability and customization capabilities further enhance its utility, making it adaptable to teams of various sizes and project complexities.

In conclusion, the Git Repository Quality Assessment Tool serves as a critical asset for modern software development teams, fostering a culture of continuous improvement, collaboration, and excellence. By providing deep insights into repository health and team performance, it not only reduces technical debt but also contributes to the delivery of high-quality software, ensuring long-term success and sustainability in development projects.

**Github Link: https://github.com/vishalch4466/gitccc232.git**