CORIOLIS
TECHNOLOGIES

Industry Project Report

Automated Code Review and Improvement Assistant

Author: Deepanshu Mittal

Supervisors: Mr. Sudhir Kumar & Mr. Rohan Nandode

Email: deepanshuharsh2209@gmail.com

CONTENT

ACKNOWLEDGEMENT

I would like to extend my profound gratitude to Coriolis Management and the CEO, Mr. Basant Rajan, for providing me the invaluable opportunity to pursue an internship at Coriolis. My heartfelt thanks go to my mentors, Mr. Sudhir Kumar and Mr. Rohan Nandode, for their unwavering enthusiasm, tireless efforts, and expert guidance throughout my internship. Their insights and constructive feedback were instrumental in helping me achieve my goals and completing my work. This internship has enriched me with an abundance of knowledge and hands-on experience, which will always remain close to my heart. I am deeply appreciative of the exceptional support from all the other interns, which made this journey even more memorable.

CODE REVIEW AND IMPROVEMENT ASSISTANT

1. Abstract

This project report presents an innovative application, "Automated Code Review and Improvement Assistant," integrated with GitLab to streamline code review and enhancement processes. Leveraging the power of GPT-4 models and the LangChain framework, this application automates the generation of comprehensive code reviews and targeted improvements for merge requests. Detailed prompt engineering ensures refined, structured outputs, directly displayed in GitLab comments for seamless workflow integration. This solution addresses the challenges of manual code reviews, including inconsistency and time consumption, by providing instant feedback on errors, performance issues, security concerns, and edge case handling. By automating these tasks, the application enhances productivity, reduces errors, and allows developers to focus on innovative solutions, transforming the software development experience. This report highlights the application's development, integration, and the transformative impact it has on the software development lifecycle, illustrating a significant leap towards the future of automated code review and improvement.

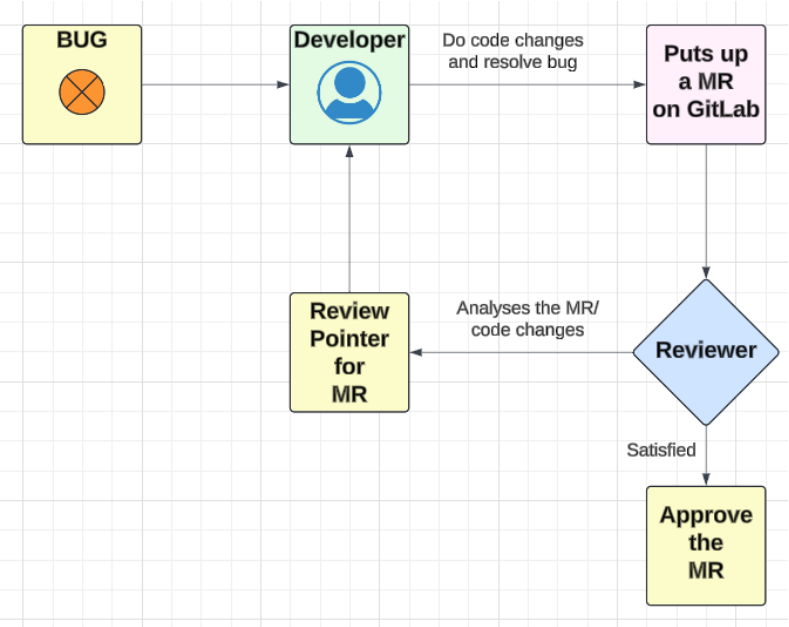
2. Problem Statement

In the fast-paced software development environment, ensuring code quality while meeting tight deadlines is a major challenge. Manual code reviews and code improvements are time-consuming, inconsistent, and prone to errors, and as projects scale, these issues intensify, placing additional pressure on development teams. Addressing these challenges, this project aims to develop a solution that gives automated code reviews & code improvements suggestions ensuring consistency, reducing time burdens, and enhancing overall code quality.

Key Challenges of the Project:

- Creating detailed and efficient prompts to ensure consistent LLM output.
- Structuring output to be aesthetically appealing and user-friendly.
- Developing an intuitive process for generating code reviews and improvements.
- Integrating GitLab with the AI application for a smooth user experience.
- Generating comprehensive and detailed code reviews efficiently.
- Ensuring the application is fast, responsive, secure, and robust.

3. Conventional way of solving MR



Two things are manual here:

1) Review pointers by the reviewer.
2) Code improvement done by the developer

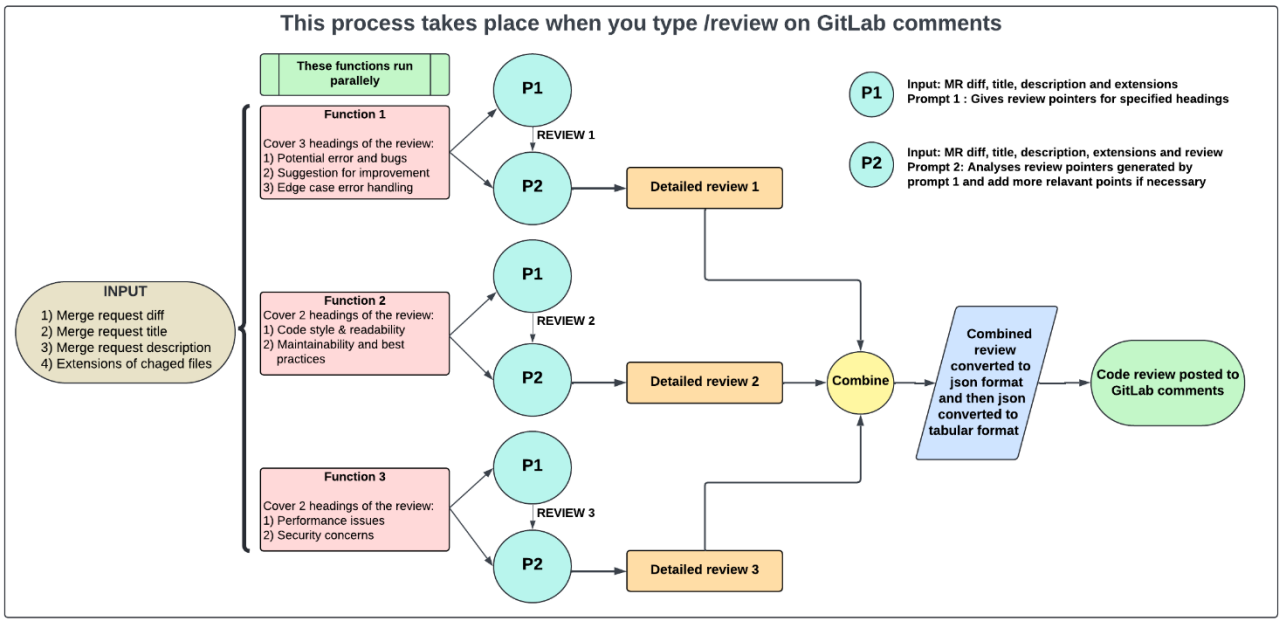
Our application assists in automating these two manual tasks.

4. Large Language Models

Large Language Models (LLMs) are advanced AI systems trained on vast amounts of text data to understand and generate human-like language. They can produce coherent and contextually relevant content, making them invaluable for a wide range of applications, from conversational agents to content creation. LLMs come in both open-source and proprietary forms, each with its own advantages. Open-source LLMs offer flexibility and transparency, allowing developers to modify and adapt the models to specific needs. Proprietary LLMs, such as OpenAI's GPT-4, provide cutting-edge performance and are backed by extensive research and development. In our project, we leverage the power of LLMs to automate code reviews and suggest code improvements for GitLab Merge Requests. By integrating LLMs, we ensure that our application delivers detailed, accurate, and human-like feedback, enhancing the efficiency and effectiveness of the code review process. The use of LLMs allows our system to understand the context of code changes, identify potential issues, and provide constructive suggestions, thus streamlining the development workflow and improving code quality.

5. Workflow of our Application

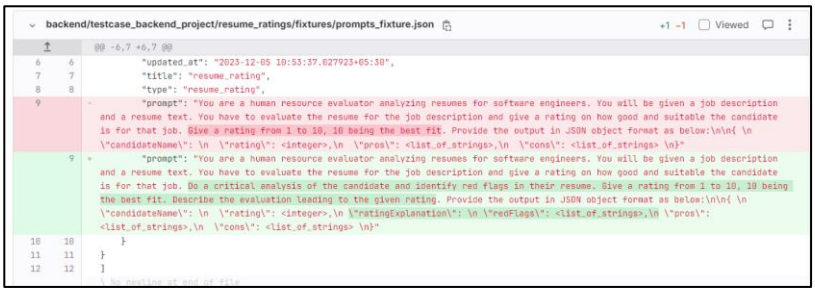
The "Automated Code Review and Improvement Assistant" application is intricately designed to enhance the code review process by integrating advanced AI capabilities with GitLab's merge request (MR) functionalities. The workflow of this application can be comprehensively understood through the following steps:



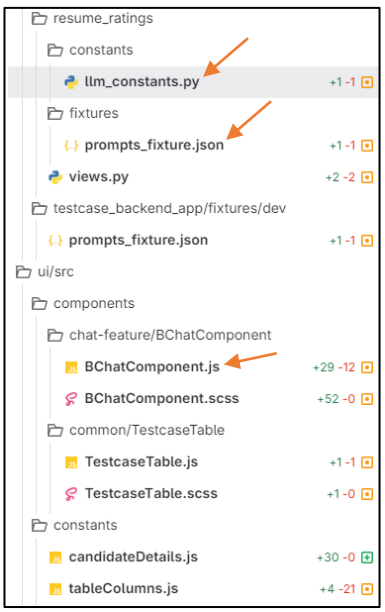
(Workflow diagram of review generation)

Input Extraction

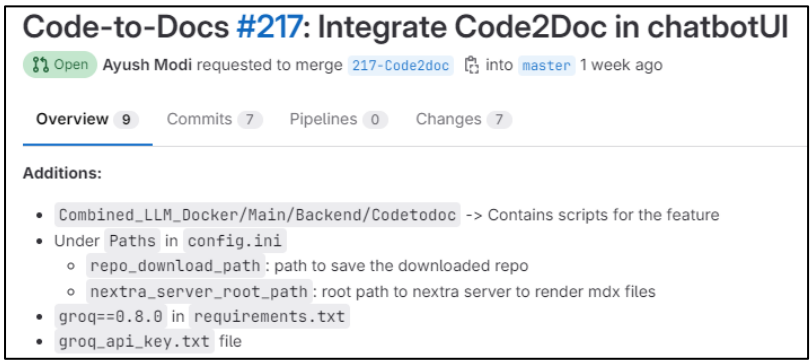
Upon a user commenting with "/review" or "/improve" on a GitLab MR, the application is triggered via webhooks. The application then extracts critical details directly from the MR, including the code diff, title, description, and file extensions of the changed files.



(example of code diff)



(different file extensions)



(example of MR title and description)

Parallel Function Execution

The extracted inputs are processed by three parallel functions, each focusing on different aspects of the code review:

- **Function 1**: Analyses potential errors and bugs, suggests improvements, and handles edge case error handling.
- **Function 2**: Evaluates code style and readability, and assesses maintainability and adherence to best practices.
- **Function 3**: Identifies performance issues and addresses security concerns.

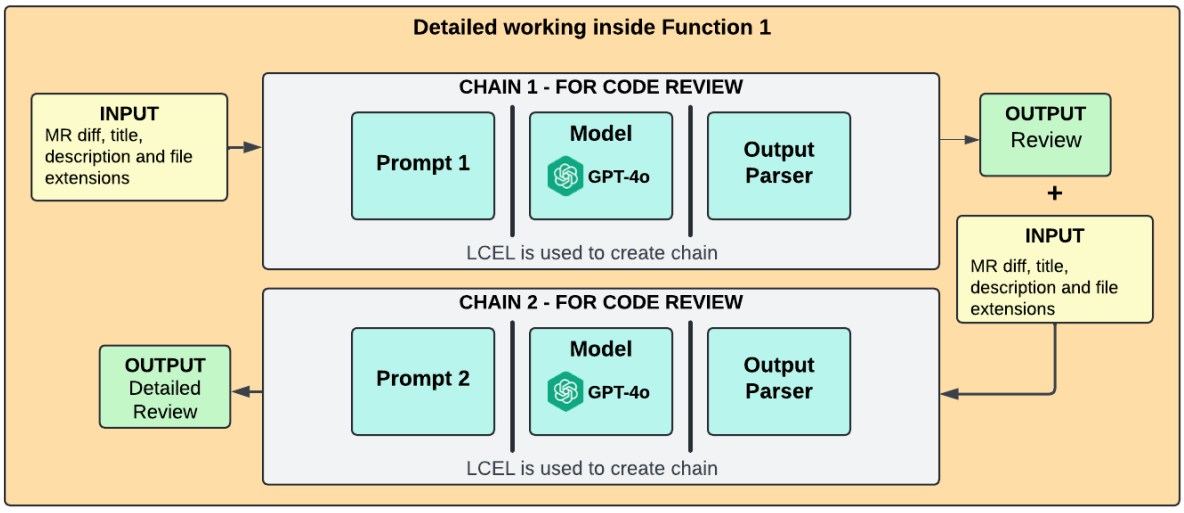
Review Generation

For each function, two distinct processing phases (P1 and P2) are executed using the (GPT-4o) model, orchestrated by the LangChain framework:

- **P1**: Takes the MR diff, title, description, and extensions as input and generates initial review pointers for the specified headings. This phase is handled by the first chain created using LangChain.
- **P2**: Further analyses the initial review pointers generated by P1, adding more relevant points if necessary. This phase is handled by the second chain created using LangChain.

Each of these phases produces a detailed review that addresses the specific review headings assigned to the function. P1 and P2 are chains that run sequentially inside the function.

Detailed working of Function 1



LangChain Expression Language (LCEL) is used to create chains inside the function. The 3 elements of the chain are Prompt, Model and StrOutputParser. After streaming the chain 1, we get review then we pass this review along with the previous input to chain 2 to receive more detailed review. Function 2 and Function 3 works in similar fashion for different review headings.

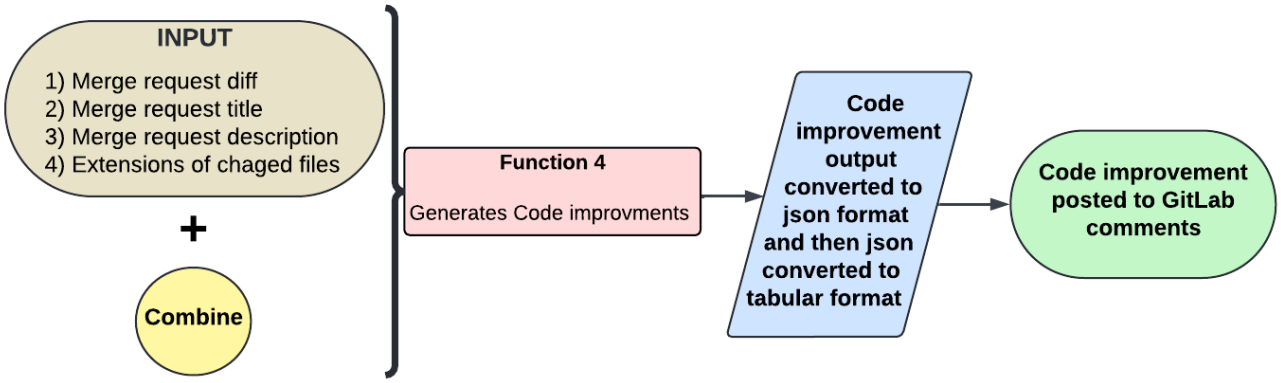
Combining Reviews

The detailed reviews from each function are then combined into a comprehensive review. This combined review is first converted into JSON format and then transformed into a tabular format for better readability and structured presentation.

Posting Review on GitLab

Finally, the structured review is automatically posted back into the GitLab MR comments, providing immediate and actionable feedback to developers. This whole process is done within 30 seconds. The seamless integration ensures that code reviews are both thorough and timely, significantly enhancing the efficiency of the development process.

This process takes place when you type /improve on GitLab comments after /review.



(Workflow diagram of code improvement generation)

Code Improvement Generation

Following the generation of a review with "/review," the "/improve" command can be used to generate code improvements based on the same review pointers. This ensures that the review points and code improvements are in sync.

Function 4: This function is dedicated to generating code improvements. It utilizes the same inputs and review pointers (combine) to provide specific code modifications and enhancements.

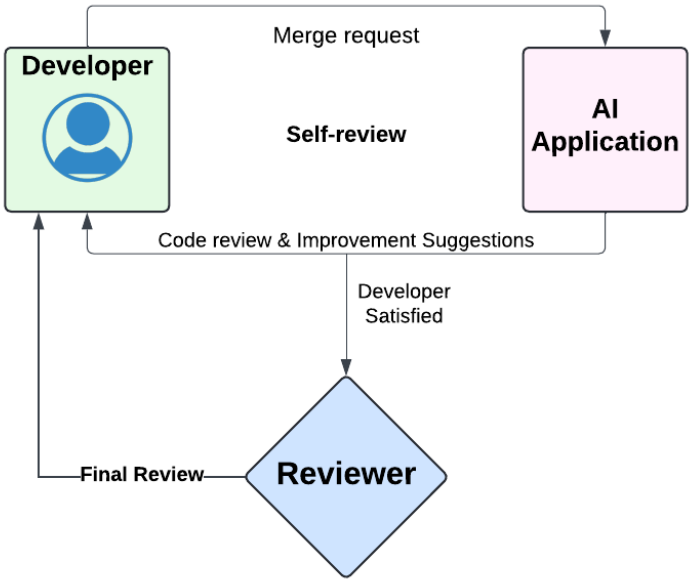
The code improvement output is then converted into JSON format and subsequently transformed into a tabular format for clear presentation. Finally, the formatted code improvements are posted back into the GitLab MR comments, providing developers with actionable code enhancements directly within their workflow.

This workflow demonstrates the powerful synergy between AI and version control systems, leveraging advanced language models and the LangChain framework to automate and improve the code review and improvement processes within GitLab.

6. Application Usage

People should use our application in the following way:

- Developer can do self-review. They will keep on reviewing until they are satisfied.
- Once satisfied they will ask reviewer to do final review.
- Reviewer will review the final one.



7. Prompt Engineering

In the development of our project, extensive experimentation was conducted with various prompt engineering techniques to enhance the quality of the generated output. The primary goal was to ensure that the responses from our functions are:

Precise & Crisp: Each output is targeted to address the specific query or requirement without unnecessary information. The responses are concise, avoiding verbosity while delivering the essential content.

Structured: The format and organization of the responses are systematically arranged, making them easy to follow and understand.

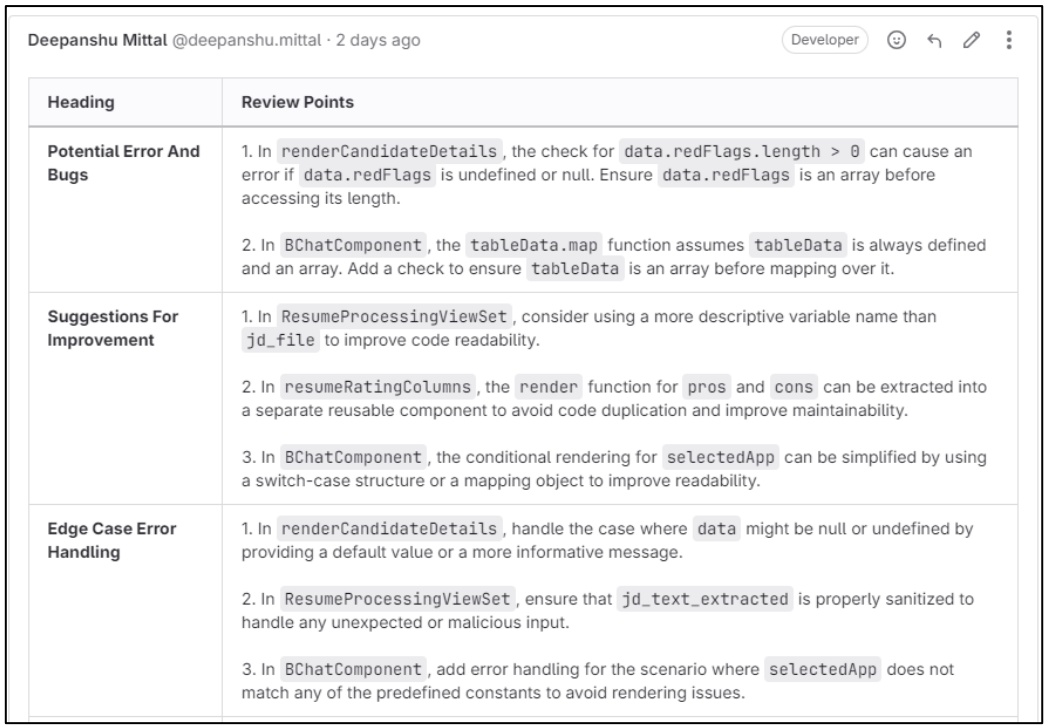
Human-like: The language used in the response mimics natural human communication, enhancing readability and relatability.

By refining prompts and iterating through various configurations, we achieved a balance between detail and brevity, ensuring that the outputs are not only functionally effective but also user-friendly.

8. Output Aesthetics

While language models (LLMs) generate output in a human-readable format, converting this output into an aesthetically appealing format is essential for the long-term usage and efficiency of our application. Ensuring that the output is structured, tabular, and easy to understand is particularly important for integration with GitLab's comments, where clarity and organization are paramount for effective code reviews.

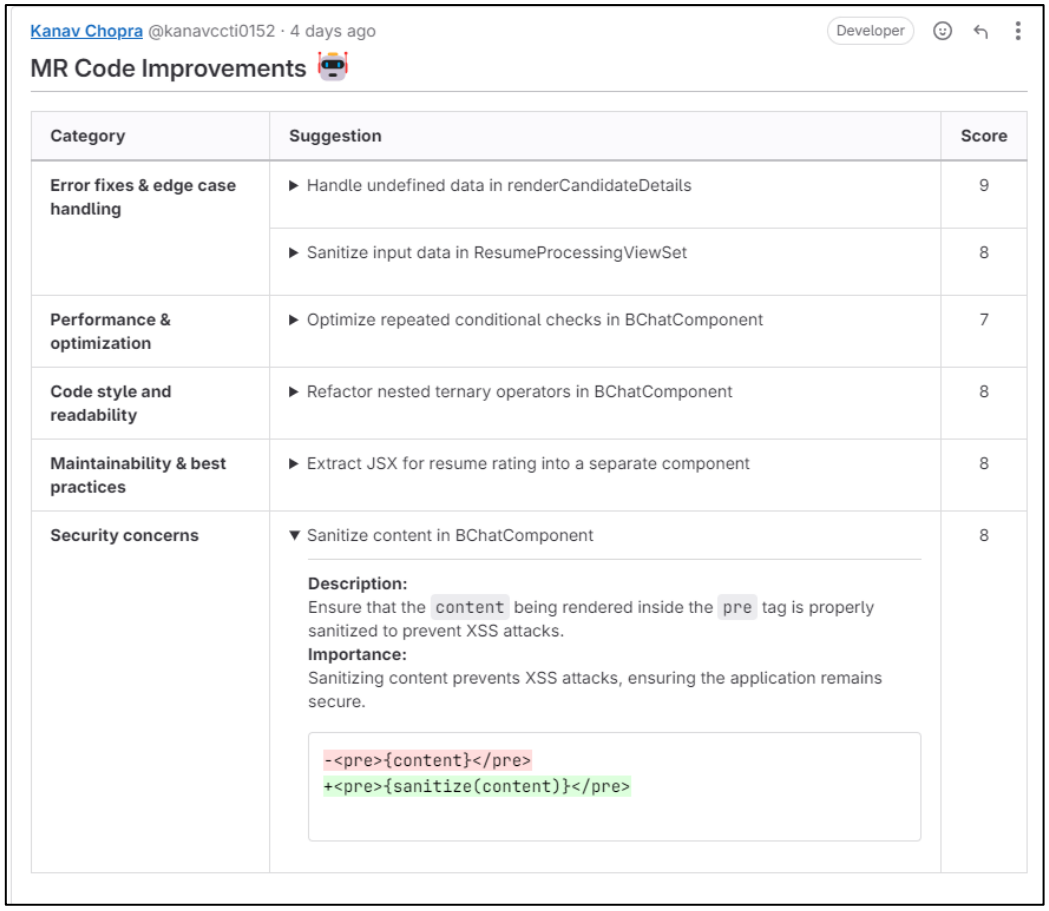
To achieve this, the raw output from the LLM is first converted into a JSON format. This intermediate step allows for a flexible and standardized representation of the data, facilitating further processing and customization. The JSON data is then transformed into a tabular format, providing reviewers with a modular and easily parsable structure. This approach not only enhances readability but also streamlines the review process, enabling reviewers to quickly and accurately interpret the results. By prioritizing output aesthetics, we ensure that the information presented is not only accurate and relevant but also visually organized and accessible, thereby improving the overall user experience and effectiveness of our application within the GitLab environment.



Deepanshu Mittal @deepanshu.mittal · 2 days ago Developer

Heading	Review Points
Potential Error And Bugs	1. In `renderCandidateDetails`, the check for `data.redFlags.length > 0` can cause an error if `data.redFlags` is undefined or null. Ensure `data.redFlags` is an array before accessing its length. 2. In `BChatComponent`, the `tableData.map` function assumes `tableData` is always defined and an array. Add a check to ensure `tableData` is an array before mapping over it.
Suggestions For Improvement	1. In `ResumeProcessingViewSet`, consider using a more descriptive variable name than `jd_file` to improve code readability. 2. In `resumeRatingColumns`, the `render` function for `pros` and `cons` can be extracted into a separate reusable component to avoid code duplication and improve maintainability. 3. In `BChatComponent`, the conditional rendering for `selectedApp` can be simplified by using a switch-case structure or a mapping object to improve readability.
Edge Case Error Handling	1. In `renderCandidateDetails`, handle the case where `data` might be null or undefined by providing a default value or a more informative message. 2. In `ResumeProcessingViewSet`, ensure that `jd_text_extracted` is properly sanitized to handle any unexpected or malicious input. 3. In `BChatComponent`, add error handling for the scenario where `selectedApp` does not match any of the predefined constants to avoid rendering issues.

(Review for 3 headings in a structured tabular format)



Kanav Chopra @kanavccti0152 · 4 days ago

Developer

MR Code Improvements

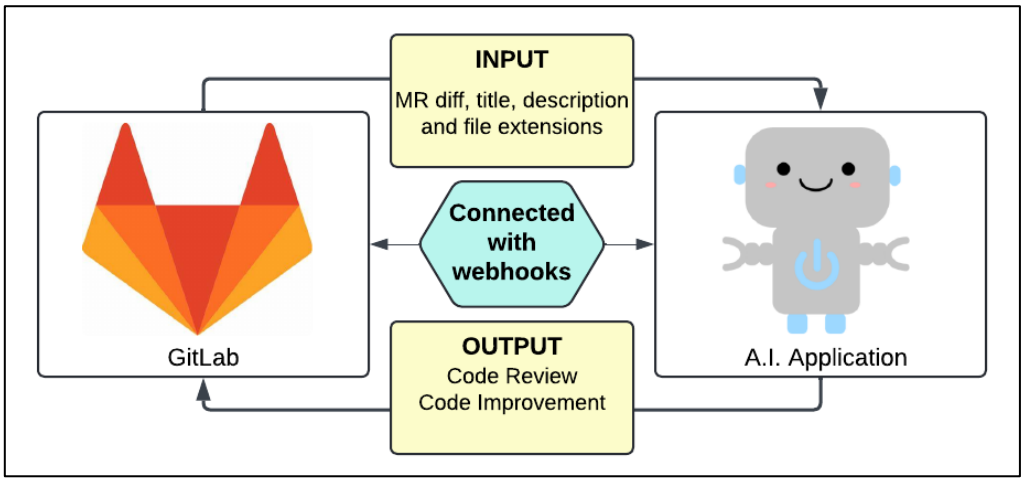
Category	Suggestion	Score
Error fixes & edge case handling	▶ Handle undefined data in renderCandidateDetails	9
	▶ Sanitize input data in ResumeProcessingViewSet	8
Performance & optimization	▶ Optimize repeated conditional checks in BChatComponent	7
Code style and readability	▶ Refactor nested ternary operators in BChatComponent	8
Maintainability & best practices	▶ Extract JSX for resume rating into a separate component	8

Security concerns	▼ Sanitize content in BChatComponent
	Description:
	Ensure that the `content` being rendered inside the `pre` tag is properly sanitized to prevent XSS attacks.
	Importance:
	Sanitizing content prevents XSS attacks, ensuring the application remains secure.
	`-<pre>{content}</pre>`
	`+<pre>{sanitize(content)}</pre>`
		8

(Code Improvement for the review pointers in a structured tabular format)

9. Integration of GitLab and Application

The integration between GitLab and the "Automated Code Review and Improvement Assistant" application is designed to seamlessly enhance the code review process. This integration is facilitated through the use of GitLab webhooks, which allow real-time interaction between the GitLab repository and the AI-powered application. Upon the occurrence of specific events within a Merge Request (MR) in GitLab—specifically, when a user comments with "/review" or "/improve"—the webhook triggers the application to initiate its review or improvement functionalities. The application then extracts essential details directly from GitLab, including the code diff, title, description, and file extensions associated with the MR. Within 30 seconds, the application processes these details, leveraging advanced language models to generate a comprehensive code review or suggest improvements. These results are then automatically posted back into the MR comments on GitLab, providing immediate feedback and enhancing the efficiency and effectiveness of the code review process. This automated interaction ensures that developers receive timely and actionable insights directly within their workflow, promoting better code quality and streamlining the development process.



(AI Application integrated with Gitlab using webhooks)

10. Results

The "Automated Code Review and Improvement Assistant" has demonstrated significant efficacy in streamlining the code review process and enhancing code quality. Through rigorous testing, the application consistently provided insightful reviews and actionable improvements for a diverse range of codebases. The integration with GitLab enabled seamless extraction of merge request details, resulting in accurate and contextually relevant feedback.

User comment "/review" on a MR

After commenting "/review" within 30 seconds review table is generated covering all 7 headings.

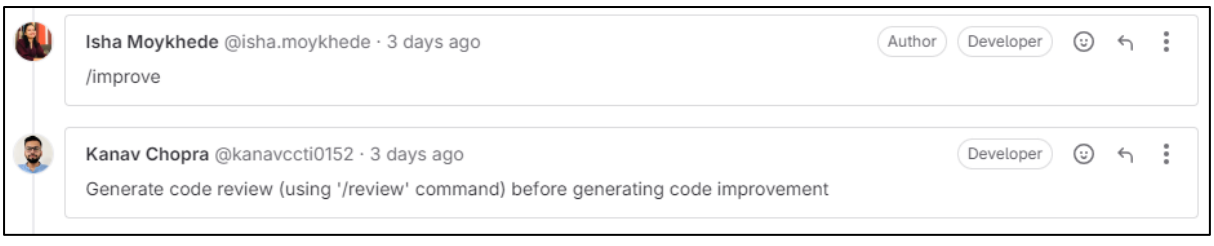
Review Table

Heading	Review Points
Potential Error And Bugs	1. In `ResumeProcessingViewSet`, the change from `jd_file.name` to `\n{jd_file.name}` and `jd_text_extracted` to `\n{jd_text_extracted}` might introduce formatting issues in the response content. 2. In `BChatComponent`, the `renderCandidateDetails` function is imported but not used consistently across different conditions, which can lead to inconsistent UI rendering. 3. In `TestcaseTable`, changing `bordered` from `true` to `false` might affect the visual separation of table rows, potentially causing readability issues. 4. In `resumeRatingColumns`, the removal of the `Candidate Name` and `Rating` columns might lead to confusion as these are critical pieces of information for the resume rating.
Suggestions For Improvement	1. In `ResumeProcessingViewSet`, consider using a more robust method for formatting multiline strings to ensure consistent output. 2. In `BChatComponent`, refactor the conditional rendering logic to ensure `renderCandidateDetails` is used uniformly for all relevant cases. 3. In `resumeRatingColumns`, consider adding a tooltip or additional context for the `pros` and `cons` columns to enhance user understanding. 4. In `renderCandidateDetails`, add error handling for cases where `data` might be undefined or null to prevent runtime errors. 5. In `BChatComponent`, ensure that the `scrollable-box` class is applied consistently to all relevant content sections to maintain a uniform user experience.
Edge Case Error Handling	1. In `renderCandidateDetails`, add a check to handle cases where `data.redFlags` might be undefined or not an array to avoid rendering issues. 2. In `BChatComponent`, ensure that `tableData` is always an array before mapping over it to prevent potential runtime errors. 3. In `ResumeProcessingViewSet`, add validation to ensure `jd_file` and `jd_text_extracted` are not null or undefined before attempting to format them. 4. In `renderCandidateDetails`, handle cases where `data.candidateName`, `data.rating`, or `data.ratingExplanation` might be missing to prevent incomplete rendering of candidate details.
Code Style And Readability	1. In `ResumeProcessingViewSet`, consider adding comments to explain the purpose of the `response_data` modifications for better clarity. 2. In the `json` files, ensure there is a newline at the end of the file to adhere to POSIX standards. 3. In `BChatComponent`, the nested ternary operators reduce readability. Consider refactoring them into a more readable format. 4. In `renderCandidateDetails`, the JSX structure could benefit from comments explaining each section for better maintainability. 5. In `scss` files, group related styles together and add comments to separate different sections for better readability. 6. In `resumeRatingColumns`, ensure consistent font sizes and styles by defining a common style class instead of inline styles.

Maintainability And Best Practices	1. In `BChatComponent`, consider extracting the JSX for `selectedApp === constants.RESUME_RATING` into a separate component to reduce complexity and improve maintainability. 2. In `renderCandidateDetails`, handle the case where `data.redFlags` might be undefined to avoid potential runtime errors. 3. In `TestcaseTable`, the `rowSelection` logic is duplicated. Consider creating a helper function to determine the `rowSelection` configuration. 4. In `scss` files, avoid hardcoding values like `#ccc` and `#f9f9f9`. Use variables to make the code more maintainable and consistent. 5. In `resumeRatingColumns`, the `render` functions for `pros` and `cons` could be extracted into separate components to improve readability and reusability. 6. In `renderCandidateDetails`, consider adding PropTypes to validate the `data` prop to ensure it has the expected structure and types.
Performance Issues	1. In `BChatComponent`, the repeated conditional checks for `selectedApp` can be optimized by storing the result in a variable to avoid redundant evaluations. 2. In `BChatComponent`, the `tableData.map` function inside the render method can cause unnecessary re-renders. Consider memoizing the mapped data or using a more efficient rendering strategy. 3. In `renderCandidateDetails`, the JSX structure can be simplified to reduce the number of nested elements, which can improve rendering performance. 4. In `TestcaseTable`, the `bordered` prop change from `true` to `false` might affect the table's rendering performance. Ensure this change is necessary and does not degrade user experience.
Security Concerns	1. In `ResumeProcessingViewSet`, ensure that `jd_file.name` and `jd_text_extracted` are properly sanitized before being included in the response to prevent potential injection attacks. (Threat score: 7) 2. In `renderCandidateDetails`, validate and sanitize `data.candidateName`, `data.rating`, `data.ratingExplanation`, and `data.redFlags` to prevent XSS attacks. (Threat score: 8) 3. In `BChatComponent`, ensure that the `content` being rendered inside the `pre` tag is properly sanitized to prevent XSS attacks, especially since it is displayed in a scrollable box.

User comment "/improve" before "/review" on a MR

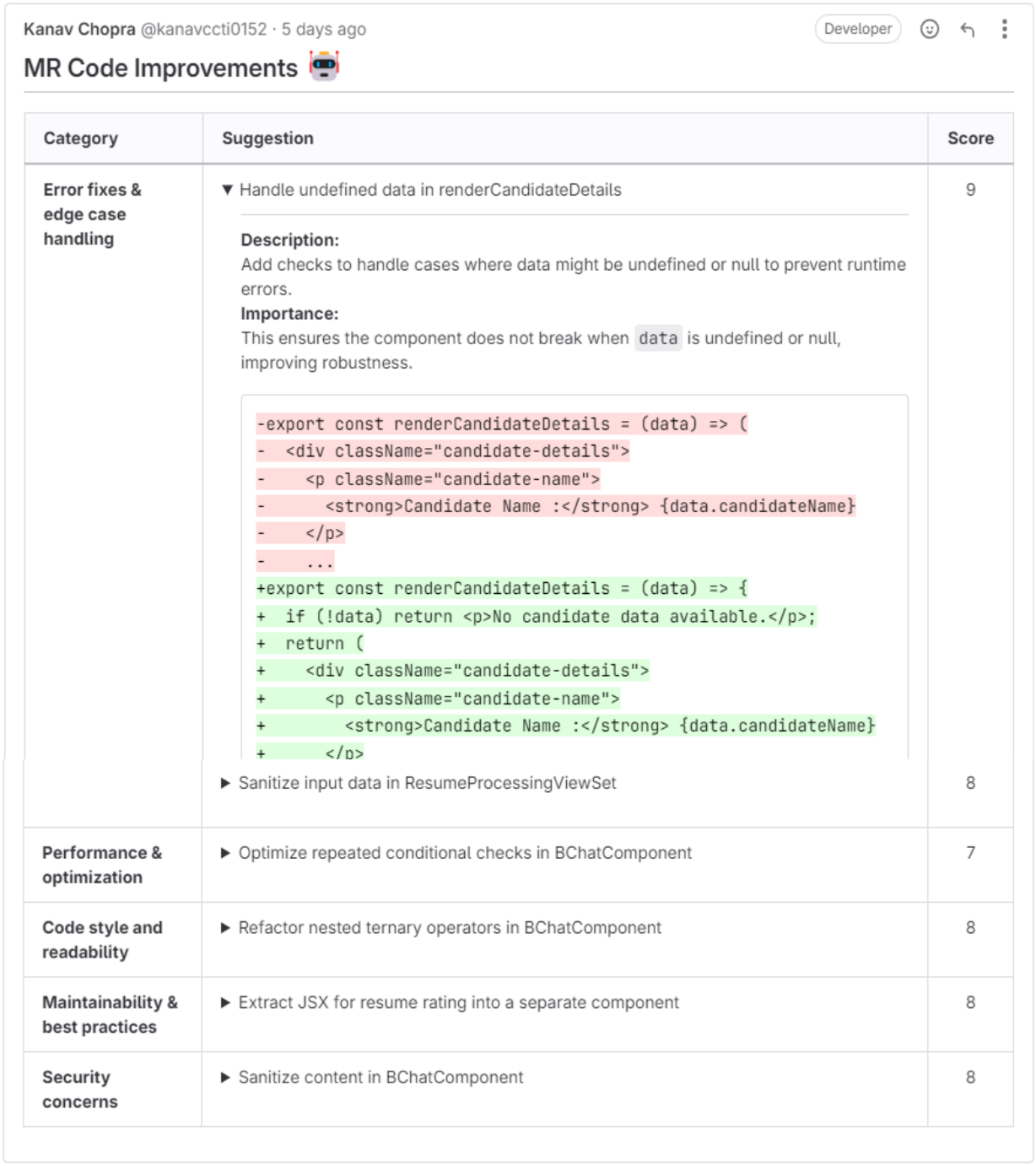
If reviewer comments "/improve" before generating review table using "/review" command. A warning comment prints.



User comment "/improve" after "/review" on a MR

After commenting "/improve" after "/review" within 30 seconds code improvement table is generated covering code suggestions for some of the review pointers from the above review table.

Code Improvement Table



Kanav Chopra @kanavccti0152 · 5 days ago — Developer

MR Code Improvements

Category	Suggestion	Score

Error fixes & edge case handling	▼ Handle undefined data in renderCandidateDetails
	Description:
	Add checks to handle cases where data might be undefined or null to prevent runtime errors.
	Importance:
	This ensures the component does not break when `data` is undefined or null, improving robustness.
	`-export const renderCandidateDetails = (data) => (`
	`- <div className="candidate-details">`
	`- <p className="candidate-name">`
	`- <strong>Candidate Name :</strong> {data.candidateName}`
	`- </p>`
	`- ...`
	`+export const renderCandidateDetails = (data) => {`
	`+ if (!data) return <p>No candidate data available.</p>;`
	`+ return (`
	`+ <div className="candidate-details">`
	`+ <p className="candidate-name">`
	`+ <strong>Candidate Name :</strong> {data.candidateName}`
	`+ </p>`
		9

	▶ Sanitize input data in ResumeProcessingViewSet	8
Performance & optimization	▶ Optimize repeated conditional checks in BChatComponent	7
Code style and readability	▶ Refactor nested ternary operators in BChatComponent	8
Maintainability & best practices	▶ Extract JSX for resume rating into a separate component	8
Security concerns	▶ Sanitize content in BChatComponent	8

11. Conclusion

The development and implementation of the "Automated Code Review and Improvement Assistant" have proven to be a valuable asset in the realm of software development. By leveraging advanced language models and seamless integration with GitLab, the application effectively automates the code review process, providing developers with detailed, actionable feedback and suggestions for improvement. This has led to enhanced code quality, reduced review times, and increased productivity for development teams. The successful deployment and positive user feedback underscore the application's potential to transform traditional code review practices. Moving forward, further enhancements and broader adoption could solidify its role as an indispensable tool for developers seeking to maintain high standards of code integrity and efficiency.

12. Future Work

- Facility to generate code improvements for human approved review pointers from the review table.
- Heat Map scheme depicting importance of each review pointer.
- Experimentation with different prompts and models to get more precise & accurate outputs.
- Making the application more generalizable for other git hosting services like GitHub, Bitbucket etc.
- Applying content compression strategies to code diffs to optimize cost.
- Introducing more functionalities other than “review” and “improve” to make a one stop solution for development team.

13. Appendix

Tools & Frameworks

- Python
- LangChain Framework
- OpenAI GPT models
- GitLab & Gitlab Webhooks

References

LangChain Documentation

CodiumAI GitHub Repository

GitLab Webhook Documentation