MINISTRY OF EDUCATION AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Internship report

Review Mind — AI based tool for code reviewing

Team No.9

Team members: Bujor-Cobili Alexandra, FAF-232 ———
Ciumacenco Pavel, FAF-232 ———
Istrati Ștefan, FAF-232 ———
Mihalevschi Alexandra, FAF-232 ———
Pleșca Denis, FAF-232 ———
TUM Intership Coordinator: Gogoi Elena, ———

Submission date: 01.10.2025

Chișinău, 2025

Abstract

The project Review Mind - AI based tool for code reviewing was developed by Plesca Denis, Istrati Ștefan, Mihalevschi Alexandra, Bujor-Cobili Alexandra, Ciumacenco Pavel, students at the Technical University of Moldova, as part of the production internship program. This report is structured into several chapters, covering the introduction and objectives of the internship, theoretical foundations, solution design and implementation, behavioral and structural modeling, followed by solution implementation, conclusions, and future perspectives.

The purpose of the project was to design and implement a workflow that automates code review processes, release note generation, and issue tracking within a GitLab environment. The general objectives included applying software engineering knowledge in practice, integrating security measures into the workflow, and producing a functional Minimum Viable Product (MVP). Methodologies such as component-based system design, behavioral modeling with use case and sequence diagrams, and automation with n8n workflows, supported by a Python backend and the integration of Google Vertex AI models, were applied throughout the project.

The obtained results demonstrate that the system can successfully analyze merge requests, generate structured AI-driven summaries, create diagrams, and automate the creation of GitLab release notes and issues. The modular architecture ensures extensibility and security through credential management and controlled API access. Possible applications of the research extend to software development teams seeking to enhance productivity, maintain project documentation, and streamline secure release cycles.

Keywords: GitLab, Workflow Automation, AI Integration, Webhook, HTTP request.

Content

Introduction

The domain studied in this internship project is the field of software engineering automation, with a particular focus on AI-assisted code review tools. Code review is an important activity in software development, because it ensures code quality, security, and maintainability. Still, traditional manual review processes can be time-consuming and prone to human error. Recent advances in artificial intelligence and natural language processing have opened new opportunities to automate and improve this process. The project targets to explore and implement an tool for automated code review, powered by VertexAI language models and Google's tool like DevAI.

The motivation for choosing this topic occurs from the increasing complexity of software projects and the growing need for efficient quality assurance practices. Many organizations are now integrating automated testing, continuous integration, and static analysis into their workflows, but code reviews still rely heavily on manual effort. Introduction of an AI-based review assistant can reduce repetitive work, speed up the feedback cycle, and catch issues earlier. This will lead to improved productivity and software reliability. From an academic perspective, the topic is also attractive because it combines multiple areas of knowledge: **programming**, **machine learning**, **prompt engineering**, and **software security**.

The structure of the report follows a logical progression. After this introduction, the first chapter presents the state of the researched domain and a short overview of related tools and technologies. The second chapter describes the system architecture and implementation details, including the design choices for the commands and processes, integration of DevAI. The third chapter highlights the security considerations and the presentation of the developed tool.

Beyond the technical aspects, the internship has also provided a valuable learning experience. Working on this project offered the opportunity to deepen knowledge in areas such as prompt engineering, DevOps practices, and AI model integration. It has also developed practical skills in problem-solving, teamwork, and independent research. The process of designing, testing, and improving the tool made stronger both technical competence and project management abilities - highly beneficial in future professional and academic activities.

1 Domain Analysis

1.1 Problem Overview

Code review is a critical process in software engineering, ensuring that code is readable, secure, efficient, and maintainable before integration into larger systems. However, traditional code review methods are largely manual, requiring significant time and expertise. As projects scale, developers face increasing pressure to review large volumes of code under tight deadlines. This can lead to:

- Missed security vulnerabilities or performance issues;
- Inconsistent review quality across teams;
- Longer development cycles due to bottlenecks in peer reviews.

The significance of this issue lies in its impact on software quality assurance and developer productivity. According to industry surveys (e.g., JetBrains Developer Ecosystem Report, GitHub State of the Octoverse), developers spend 10–20% of their time on code reviews, yet these reviews often lack depth due to fatigue, repetitive checks, or uneven expertise among reviewers.

Modern software engineering is becoming increasingly complex. Teams work with very large codebases, fast release cycles, distributed collaboration, and developers with different levels of experience. In this environment, code review is a crucial step. It helps make sure that the code is consistent, maintainable, secure, and of high quality. But the way code review is usually done manually therefore it has several challenges:

a) Time and slow feedback. Manual reviews often take longer than expected. A developer submits code, then waits for a reviewer to carefully go through it, leave comments, get a response, and sometimes repeat the process. These cycles can delay deployment and slow down the whole development flow. In many projects, review delays have even been identified as a major bottleneck;

b) Human error and inconsistency. Code reviews depend on people, and people are not always consistent. Reviewers have different levels of expertise and personal preferences. Some might overlook important issues, while others might focus too much on smaller style problems. Fatigue also plays a role — when a review involves hundreds of lines of code across multiple files, it's easy to miss something important;

c) Scalability problems. As projects grow, the workload for reviews grows too. The amount of code to be checked doesn't scale neatly with the number of reviewers available. This makes manual reviews harder to manage in large or fast-moving projects;

d) Late detection of issues. Catching problems early saves time and money. Unfortunately, traditional reviews don't always spot deeper issues like hidden security vulnerabilities, long-term maintainability problems, or code smells. These are often discovered later in testing or production, where fixing them is much more costly;

e) Cognitive burden on reviewers. To review code effectively, a person needs to understand not only the lines that changed but also how they fit into the bigger system. This requires jumping between files, documentation, and dependencies. The constant context switching is mentally draining and increases the chances of overlooking issues.

Why this matters? These limitations add up to real-world consequences: economic cost (delays and late-found bugs cost more to fix), security risks (missed vulnerabilities can lead to serious breaches), team morale (long, repetitive reviews and inconsistent feedback frustrate developers), compliance pressure (in industries like finance or healthcare, there are strict standards for code quality and security, making reliable review processes even more important). This is why improving the code review process is so important. With the help of AI, there's an opportunity to make reviews faster, more consistent, and less burdensome, while still keeping the human judgment that makes them valuable.

Recent studies provide valuable evidence on the benefits and challenges of automating the code review process. They help put the project into context and show how AI-assisted tools can realistically impact software engineering practices.

One important finding comes from the paper Automated Code Review In Practice (Umut Cihan et al., 2024) [1]. The researchers tested their AI-powered review tool on a real project and observed that the time developers spent actively reviewing code was reduced. However, they also noted that the overall pull request closure time — the period between opening and successfully merging a request — actually increased, from around 5 hours and 50 min to 8 hours and 20 min. At first glance this seems like a drawback, but their conclusion was that the extra time was not wasted. Instead, it reflected more thorough reviews and stronger adherence to good coding practices. This shows that AI tools can shift the focus from speed to quality, which in the long run benefits the reliability and maintainability of software.

Another relevant study, Toward Effective Secure Code Reviews [2]: An Empirical Study, highlights the role of AI in strengthening software security. Their results showed that AI-assisted review systems are highly efficient at detecting weaknesses and vulnerabilities in code, with error rates as low as 6–9%. This suggests that, while not perfect, AI reviewers are already capable of catching a large share of issues that humans might miss, especially in the critical area of security.

Finally, the paper The Effects of Change Decomposition on Code Review[3] provides insights into how the structure of code changes impacts review effectiveness. The researchers compared large, "tangled" pull requests that contained multiple types of changes against smaller, more focused requests, each containing one type of change. They found that the decomposed pull requests led to more effective reviews, with reviewers spending the same amount of time but engaging in more context-seeking behaviors and producing higher quality feedback. This supports the idea that not only the tools but also the way reviews are structured can make a big difference.

Taken together, these studies show that AI-assisted code review tools can reduce reviewer workload, improve security, and raise the quality of feedback. They also point out that workflow design — such as how changes are grouped — plays a key role in review efficiency. The project builds on these insights by aiming to combine AI-driven analysis with practical improvements in review workflows, so that both speed and quality can be balanced in real development environments.

1.2 Target Audience

The primary audience for an AI-assisted code review tool is software development teams. These can range from large professional companies with multiple teams working in parallel, to open source projects run by distributed contributors, to smaller groups of students or junior developers who are still building their skills. All of these groups face similar challenges: they need to maintain code quality, ensure security, and keep up with demanding development schedules. That said, in the context of this internship project, the tool was designed with a specific partner company (Planet Group International) in mind. This gave us the advantage of working directly with real developers and maintainers, gathering their feedback, and shaping the features around their actual day-to-day needs. Instead of building a generic assistant that might or might not fit, there was the possibility to ground the solution in real workflows and pain points:

a) Company-Specific Needs: From the beginning, Planet Group International's developers made it clear that they wanted something practical, not just experimental. They already had experience with tools like CodeRabbit and saw value in automated assistance, but they also noticed gaps where those tools fell short. Based on their suggestions, the solution was focused the efforts on several improvements;
b) Workflow integration: Reviews needed to fit smoothly into their development pipeline. The work was made on integrating with n8n for workflow automation and added Git webhooks so that reviews could be triggered automatically whenever a merge request was opened. This reduced manual setup and made the tool feel like a natural part of their process;
c) Smarter feedback: One of the frustrations developers mentioned was irrelevant or repetitive issues being flagged. To address this, the prompts was used by the AI so that feedback was more context-aware and meaningful. This meant fewer distractions and a higher signal-to-noise ratio in the review output;
d) Clearer communication: For maintainers dealing with multiple contributions, long and detailed feedback can become overwhelming. To solve this, short summaries of reviews was introduced, giving a quick snapshot of what changed and what issues were most important;
e) Better documentation: The company also emphasized the importance of proper release management. Also the generation of release notes and changelogs after merges was implemented, ensuring that updates were documented consistently without extra manual effort;
f) Visual understanding: For more complex or architectural changes, plain text reviews were sometimes

not enough. In response, there was added the ability to generate UML diagrams, making it easier to grasp how new code fits into the bigger picture.

These improvements were not random add-ons, they came directly from the company's developers and maintainers. The result was a system that didn't just automate reviews, but actually helped solve the specific problems they faced, whether that was cutting down wasted time, improving clarity, or supporting better long-term project documentation.

Although the tool was shaped in close collaboration with the internship company, its usefulness is not limited to a single environment. The same challenges that their developers face are shared across much of the software industry, which means the project can have broader impact if adopted by other groups:

- Professional Development Teams in Companies - in large companies, consistency and speed are everything. Teams working on fast-moving projects benefit from automated checks that enforce standards, catch common mistakes, and reduce review delays. An AI assistant like ours ensures quality without forcing teams to choose between speed and thoroughness;
- Open Source Contributors - maintainers of open source projects often juggle many incoming pull requests with limited time. For them, this tool can act as a first line of review, automatically providing feedback to contributors. This reduces the backlog, improves contributor experience, and lets maintainers spend their energy on higher-level decisions;
- Early-Career Developers and Students - for juniors and students, feedback is not only about catching mistakes but also about learning. Automated reviews can act as a consistent mentor — pointing out errors, explaining best practices, and reinforcing standards. This helps newcomers grow faster and reduces the time senior developers need to spend on basic corrections;
- Security and Compliance Teams - in industries where security and compliance are non-negotiable, automated reviews provide another layer of protection. By flagging vulnerabilities and ensuring certain coding rules are followed, the tool can help teams meet internal policies and external regulations. The ability to generate clear logs and structured outputs also makes compliance audits easier.

In short, while the immediate target audience was the developers at internship — whose direct feedback shaped everything from workflow integration to prompt refinement — the project has clear value for a much wider community. Whether it's a fast-paced corporate team, an open source maintainer, a student learning best practices, or a compliance officer guarding against vulnerabilities, the tool addresses real challenges that many groups face. What began as a company-specific solution has the potential to grow into a versatile assistant that improves how code reviews are done across the industry.

1.3 Solution Concept

The developed solution is an AI-assisted code reviewer that connects directly to a company's Git workflow. The idea is not to replace human reviewers but to act as a helpful assistant that takes away

repetitive work, speeds up feedback, and adds clarity to the review process. Whenever developers push changes or open a merge request, the system is triggered. At that point, the helper in face of n8n gathers the modified code (with the option of including more context or even the full project if needed) and passes it on to an AI model.

Guided by carefully designed prompts, the AI examines the changes and returns structured feedback. This feedback is flexible, but it usually includes:

- Explanations of issues, often with suggested code snippets;
- Severity ratings (Critical, High, Medium, Low) so the team knows what needs immediate attention;
- Short summaries of the merge request, useful for quickly understanding what has changed;
- UML diagrams when architectural changes are involved, making it easier to see the bigger picture;
- Automatically drafted release notes that keep documentation and changelogs up to date.

This way, the AI does a lot of the heavy lifting—spotting issues, writing repetitive comments, and even generating supporting documentation. Developers and maintainers still make the final decisions, but their time and energy can go toward higher-value work: design discussions, mentoring, or solving deeper technical problems.

The advantages of this approach are clear. Reviews become faster because developers don't have to wait as long for initial feedback. They become more consistent because the AI applies the same standards every time. And they become easier to follow thanks to summaries and diagrams that reduce the cognitive load of understanding large or complex changes. In short, the tool helps teams keep up coding standards while reducing the stress and overhead of traditional manual reviews.

Technically, the system ties together Git webhooks, prompt-based AI review pipelines, and extensible rules that can be adapted to different teams. It's flexible enough to evolve—whether that means improving prompts, refining severity levels, or adding new automated checks. The result is a code review assistant that feels like part of the team: always available, never tired, and consistently focused on code quality.

1.4 Market Research

Code reviews are a fundamental part of the software development process, but they consume a significant amount of developer time. Research indicates that: Developers spend an average of 6 hours per week on code reviews [4], In some organizations, this can extend to 2–5 hours weekly, with a smaller percentage dedicating 5–10 hours [5]. These statistics underscore the substantial time investment required for code reviews, highlighting the potential benefits of automation in this area.

Studies have shown that: 90% of developers believe code reviews contribute to improved software quality [6]. A study by De Silva et al. found that code reviews significantly improve software quality by identifying defects, enhancing readability, and ensuring compliance with coding standards [7]. However,

the effectiveness of code reviews can vary, with some studies indicating that formal code reviews detect approximately 80% of defects, while others report lower effectiveness [6].

Despite their benefits, code reviews present several challenges: time constraints: developers often face time pressures, leading to rushed reviews and potential oversight of critical issues [8]; workload and manpower issues: a significant number of developers report that workload and lack of manpower are major obstacles to conducting thorough code reviews [9]; communication barriers: providing constructive feedback without causing defensiveness can be challenging, impacting the effectiveness of the review process [10]. These challenges highlight the need for solutions that can streamline the code review process, reduce manual effort, and maintain high-quality standards.

The integration of AI into code reviews is gaining traction as a means to address these challenges: A recent survey by Google revealed that 90% of developers now use AI tools regularly, a significant increase from just 14% in 2024. Among these, 65% report heavy reliance on AI, dedicating a median of two hours per day to such tools. The benefits include increased productivity for 80% of users and improved code quality for 59% [11]. Atlassian's study found that 68% of developers save over 10 hours per week using AI tools, up from 46% the previous year [11]. These statistics demonstrate the growing confidence in AI's ability to enhance the code review process by automating routine tasks and providing consistent feedback.

Insecure coding practices can lead to significant security breaches: A report by SecureFlag found that 74% of companies experienced at least one security breach in the past year due to insecure coding practices [12]. This underscores the importance of incorporating security-focused code reviews to identify and mitigate vulnerabilities early in the development process.

Another important aspect worth highlighting is the solution domain. During the research phase of the internship, it became evident that the market already provides a considerable number of code review and automation tools. These range from traditional static analysis platforms to modern AI-assisted solutions. However, despite this diversity, none of the available tools were fully aligned with the specific needs and objectives of the internship company. For example, some tools focus primarily on security vulnerabilities, while others emphasize metrics and maintainability, but very few combine automated summaries, issue creation, and release note generation into a single integrated workflow.

This gap in the existing landscape was one of the main motivations for developing a tailored solution during the internship. By combining automation workflows (through n8n), AI-powered summarization (VertexAI), and GitLab integration, the proposed project aims to address the missing functionalities in a more complete way. To better understand the current ecosystem, a comparative analysis of several existing solutions has been conducted. Their strengths and weaknesses are summarized in Table 1.1, which serves as the foundation for justifying the novelty and relevance of the developed project.

The current landscape of code reviews reveals a substantial time investment by developers, with

an average of 6 hours per week spent on this task. While code reviews play a crucial role in enhancing software quality, challenges such as time constraints, workload issues, and communication barriers persist. The integration of AI into the code review process has shown promising results, with developers reporting significant time savings and improvements in code quality. Additionally, the emphasis on secure coding practices highlights the need for comprehensive review processes to prevent security breaches.

Table 1.1 - Comparison of Existing Solutions

Solution	**Main Features**	**Advantages**	**Limitations**
CodeRabbit	AI-assisted code reviews, pull request comments, integration with GitHub/GitLab	Provides automated review feedback in natural language, easy integration with PR workflows	Limited customization of feedback, focused mainly on PR comments without extended release automation
Qodo	Collaborative code review platform, task management, version control integration	Combines code review with team collaboration, supports tracking of issues and discussions	Less focus on automation, no AI-based summarization or automated release notes
DeepCode (Snyk Code)	AI-driven code analysis, security vulnerability detection	Uses ML models to detect complex issues, strong security focus	Limited support for generating summaries, not designed for workflow automation
GitHub Copilot for PRs	AI-assisted suggestions, code explanation, inline reviews in pull requests	Natural language summaries of changes, AI explanations for developers	Still limited to GitHub ecosystem, not directly usable for GitLab workflows or automated releases

The current AI-assisted code review tool aims to address these challenges by automating routine tasks, providing consistent feedback, and supporting secure coding practices, ultimately enhancing the efficiency and effectiveness of the code review process.

2 System Design

2.1 Technical Requirements

In order to develop the automated code review generation tool, a number of technical requirements are necessary. These requirements ensure the system is reliable, secure, and well integrated with existing development workflows, such as GitLab merge requests and automated pipelines. The requirements can be grouped into two categories: functional and non-functional.

a) *Functional Requirements*

The functional requirements describe the core features and operations of the system. These include:

- merge request handling;
- summary generation;
- diagram generation;
- issue creation;
- release notes.

The system must detect when a merge request is open or merged into the main branch and trigger the workflow accordingly. The code changes or comments associated with the merge request must be passed to the AI model, which generates a structured summary in markdown or JSON format. When required, the AI output can be extended with automatically generated diagrams that visually represent the changes or dependencies. Based on the review findings, the workflow must be able to create new GitLab issues automatically, ensuring that problems are tracked. Once a merge request is finalized, the system must prepare a release note. This includes cleaning and formatting the AI summary, generating a version tag, and publishing the release note to GitLab.

b) *Non-Functional Requirements*

The non-functional requirements describe the standards for performance, usability, and security of the system. These include:

- infrastructure;
- integration with AI model;
- security and access;
- usability;
- performance.

The system should rely on modern automation and management tools. The workflow engine (n8n) can be deployed either locally using Docker or on a cloud platform. A GitLab environment is required, providing the repositories and merge request events that trigger the system. Secure webhook exposure must be ensured through reverse proxies such as Nginx or tunneling services like ngrok. The system must inte-

grate a generative AI for text analysis and summarization. In this case, models such as Google VertexAI's CodeChatModel and ChatModel are used, though alternative APIs may also be integrated. The backend in Python, powered by Flask, coordinates the requests, while n8n workflows make use of built-in nodes (Webhook, HTTP Request, Switch, GitLab) and custom JavaScript nodes. Secure access to external services must be maintained. A GitLab personal access token with API rights is required for creating releases, tags, and issues. In addition, a valid service account or API key must be available for accessing the AI models. Secrets must be managed using the credential storage mechanisms of n8n or environment variables and Secret Manager API service. To prevent unauthorized calls, webhook signing or token-based authentication must be applied. The output (summaries, diagrams, release notes) should be formatted in a clear and standardized way to be easily integrated into project documentation. The system should remain simple to configure and maintain by development teams. The workflows must execute within reasonable time limits to provide timely outputs after merge requests. AI inference latency and GitLab API response times must be considered to avoid delays.

To implement these requirements, the project combines several methodologies. Experimental prototyping is used to test AI summarization quality and to design n8n workflows. Iterative development allows gradual improvements in formatting and automation reliability. Finally, integration testing is applied to validate the interaction between GitLab webhooks, AI outputs, and the generated release artifacts.

2.2 Behavioral Modeling

In order to better understand how the system behaves in different situations, behavioral modeling was used during the development process. This approach makes it possible to describe how users (in the case of the following solution - developers and admins), in addition with external systems interact with the application, as well as how the internal components respond to those interactions. For this project, two main types of diagrams were employed: use case diagrams and sequence diagrams.

a) *Use Case Diagrams*

When considering the use cases, it's important to remember that the user is divided into 2 instances: the developer and the admin, The developer has access to the most functionality, but opposed to the admin, does not have any possibility to modify any of the setting or prompts for the AI. The admin, instead, can freely change the prompts, temperature, and even make other changes appropriate to the company culture.

The use case diagram presented (Figure 2.1) above illustrates the main interactions between the developer and the functionalities of the automated code review tool. The developer is the primary actor in the system, and all of the use cases are designed to assist them in reviewing and managing code changes. The central functionality is "Review code", which represents the core purpose of the system. This use case can be extended in two directions: Review whole project context, where the developer requests a broader analysis of the entire repository to better understand the global impact of changes; review changed MR

context, which focuses only on the modifications introduced in a specific merge request.

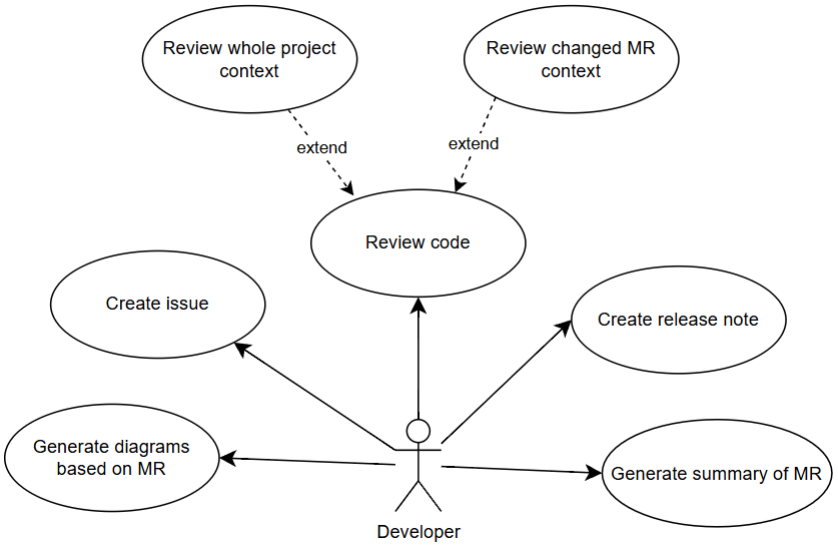

Figure 2.1 - Use cases for a developer

Besides reviewing, the developer has access to additional automated features that extend the usefulness of the system: Generate summary of MR, create release note, create issue, generate diagrams based on MR.

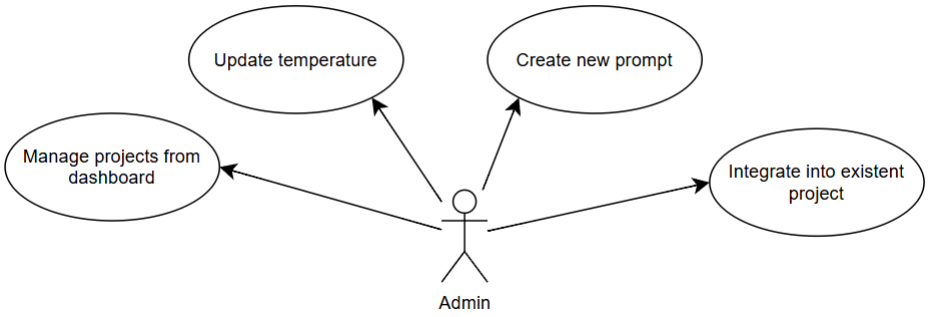

Figure 2.2 - Use cases for an admin

This use case diagram (Figure 2.2) shows the interactions of the Admin with the system, focusing on configuration and management tasks. The Admin can update temperature (adjusting AI model behavior), create new prompts for different review or automation scenarios, manage projects from the dashboard to oversee ongoing work, and integrate the tool into existing projects to extend its use. The diagram highlights the administrative role as the one responsible for customizing and maintaining the system's setup.

b) *Sequence Diagrams*

When someone pushes code changes to GitLab, GitLab automatically tells N8N, then N8N talks to the Flask web app, which handles all the backend work (Figure 2.3).

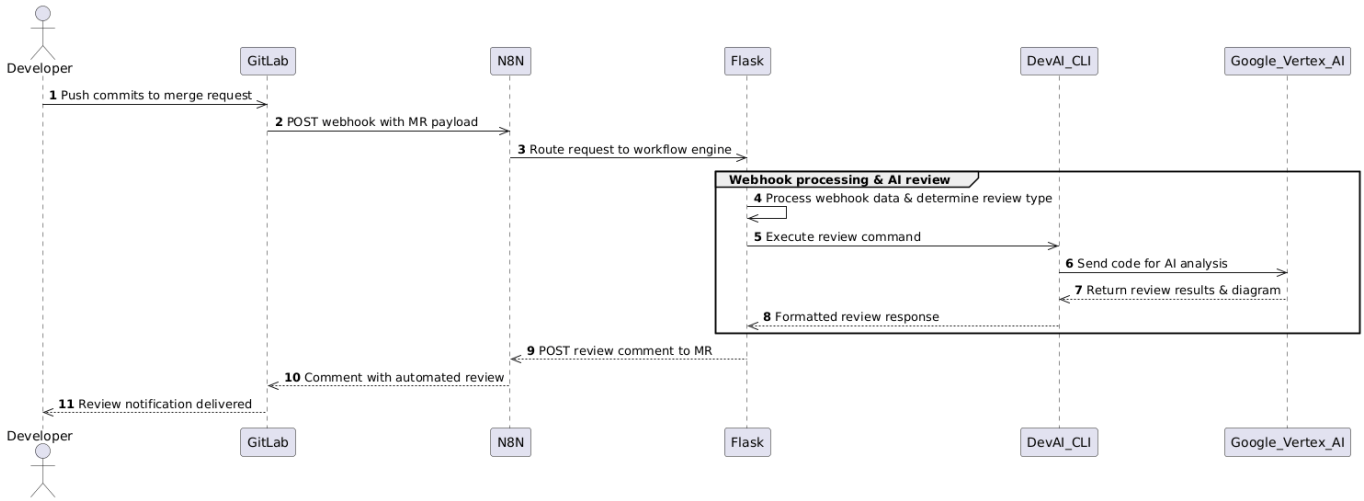

Figure 2.3 - Merge Request Review Process

From there, it uses the DevAI CLI tool to send the code to Google's AI service for analysis. The AI figures out if there are any issues and sends back suggestions. All this gets packaged up and posted back to GitLab as a comment on the merge request.

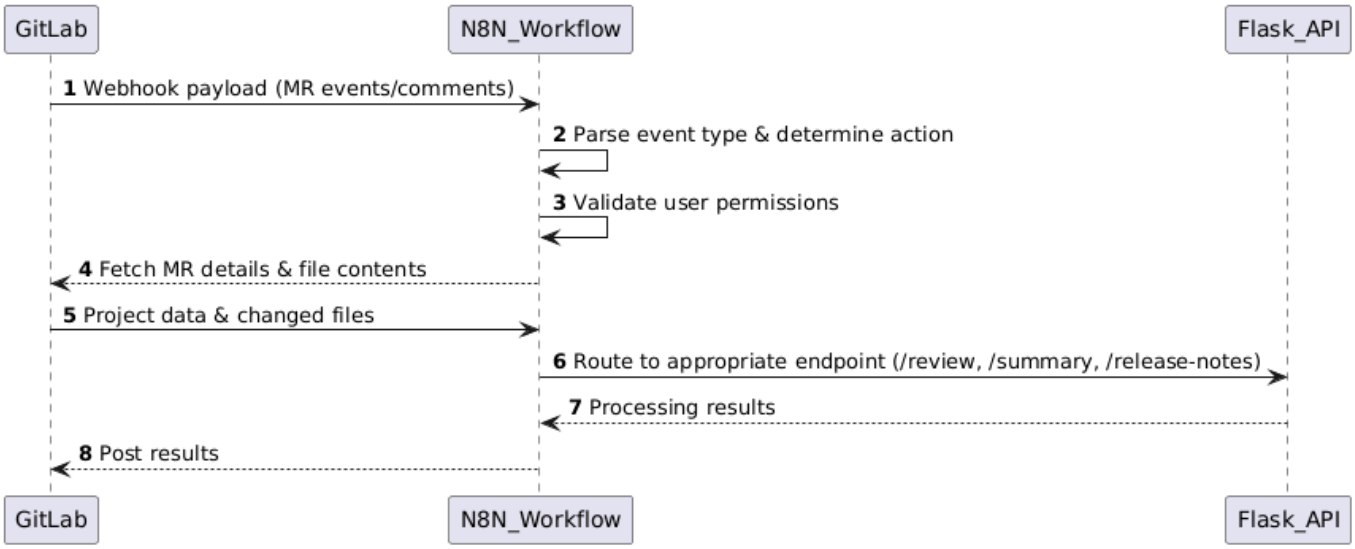

Figure 2.4 - N8N Workflow

The diagram depicted in the figure above (Figure 2.4), shows the N8N workflow. In general, N8N gets events from GitLab and decides what to do with them. For example, if someone comments on a merge request with a command, N8N parses that and collects the right data from GitLab. Then it calls the right API endpoint on the Flask server, whether it is for a code review, summary, or something else. Once it gets the results, it sends them back to GitLab as comments or creates new issues. N8N keeps everything organized and makes sure the right actions happen.

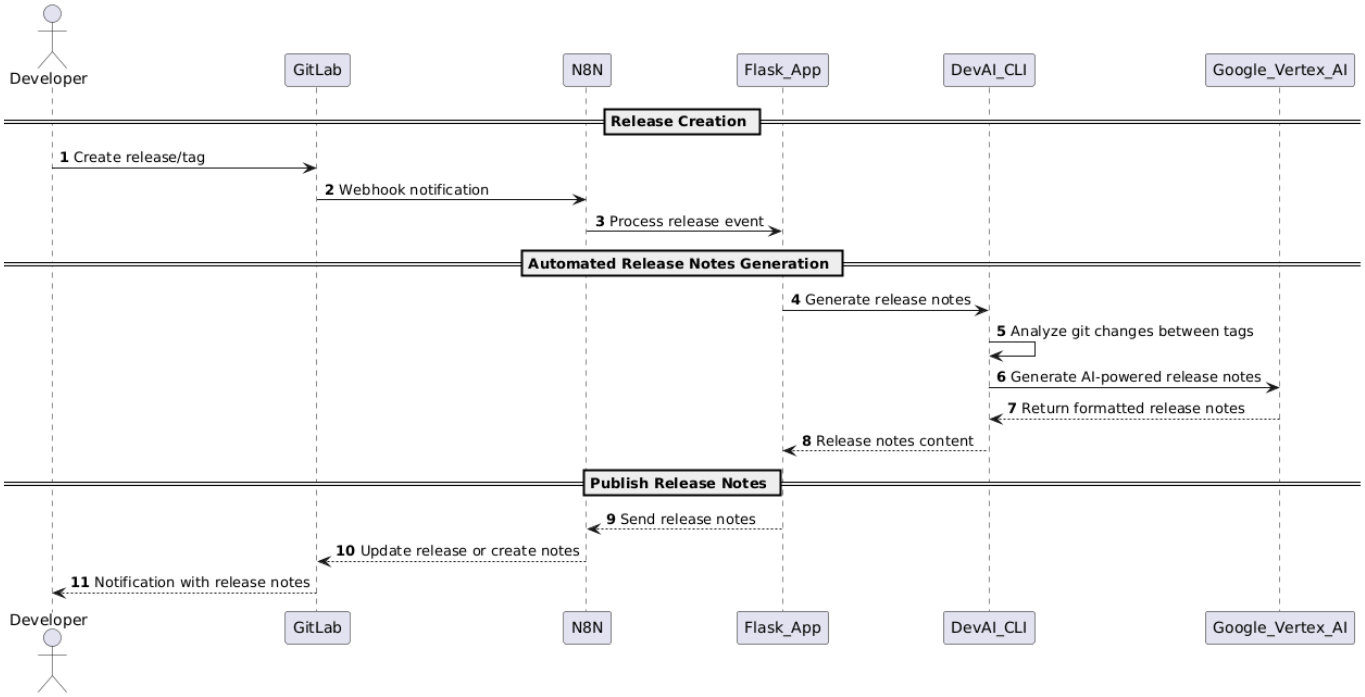

Figure 2.5 - Release Notes Process

Figure 2.5 has the release notes automation process. After a merge request gets merged in GitLab, N8N gets notified. It pulls together the project information and sends it to Flask to create the release notes. Flask processes all that git history stuff and generates nice looking release notes. N8N then takes those and publishes them as an official release in GitLab.

2.3 Structural Modeling

Structural modeling is used to represent the static parts of a system and how they are organized. Unlike behavioral modeling, which shows interactions and workflows, structural modeling focuses on the components that make up the system and the relationships between them. For this project, the main type of diagram used is the component diagram. A component diagram shows the main building blocks of the application, such as the automation tool (n8n), the GitLab environment, the AI service, and the backend services, as well as how these parts connect to each other.

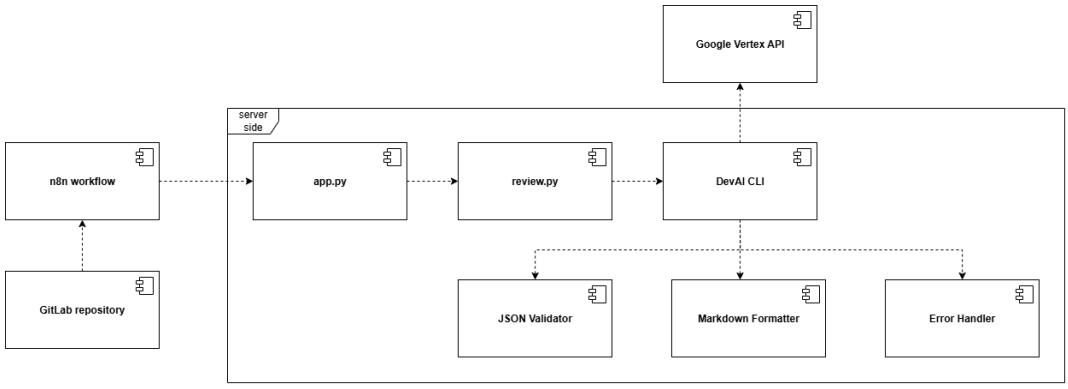

Figure 2.1 - Component diagram for review code flow

This component diagram (Figure 2.1) illustrates the main structural elements of the project and how they interact. The workflow begins in n8n, which retrieves merge request information from the GitLab repository. This data is passed to the server-side logic, where app.py coordinates the process and forwards requests to review.py. The review component relies on the DevAI CLI, which interacts with the Google Vertex API for AI-powered analysis. Supporting modules, such as the JSON Validator, Markdown Formatter, and Error Handler, ensure proper validation, formatting, and error management. Together, these components form a modular pipeline for automated code review and release note generation.

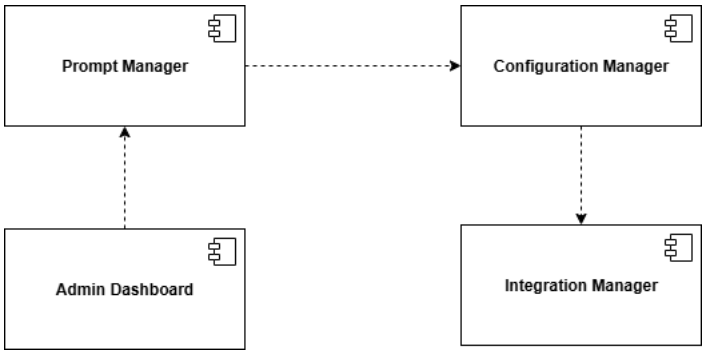

Figure 2.2 - Component diagram for prompt management

This diagram (Figure 2.2) outlines the high-level architecture of the prompt administration system. The core component is the Prompt Manager, which is centrally managed through the Admin Dashboard. This structure is supported by two key services: the Configuration Manager, responsible for handling system settings and parameters, and the Integration Manager, which facilitates connections and data exchange.

3 Solution Implementation

The Solution Implementation chapter presents how the proposed system was developed in practice. It describes the implemented features, the chosen technologies, the applied security measures, and the overall workflow of the project. This section bridges the design phase with the functional application, showing how theoretical planning was transformed into a working solution.

3.1 Technological Stack

The DevAI Assistant project is built upon a integrated technological stack designed to deliver a robust and scalable system for automated code analysis and prompt management. The backend is powered by the Flask framework, which serves as the core web framework. It handles all HTTP routing for the admin dashboard and critical API endpoints, manages user authentication and session security, and serves the primary prompt management interface. Furthermore, Flask provides the restful API endpoints that enable seamless integration with the n8n automation workflows and handle webhook calls from GitLab, forming the central nervous system of the application[13].

For developer interaction, the project features a Command Line Interface (CLI). This CLI provides developers with an intuitive tool for direct code analysis and prompt management. It encompasses a suite of sub-commands that facilitate operations such as running automated code reviews, generating documentation, and managing custom prompt templates, all while integrating smoothly with version control systems.

The core AI processing capabilities are delivered through an integration with Google Cloud Vertex AI. This service provides access to powerful Gemini models, which perform the intelligent code review, natural language processing for documentation, and advanced code analysis. Secure access to these AI services is managed via service account authentication with appropriate IAM roles. The entire application is containerized using Docker, ensuring consistent deployment across all environments. The CI/CD pipeline is automated through Google Cloud Build, which uses a defined cloudbuild.yaml configuration to build and push container images to the Google Artifact Registry.

Workflow automation is orchestrated by the n8n platform, which acts as the connective tissue between various services. It listens for webhook events from GitLab and manages the complex logic for different code review requests, classifying comments and routing them appropriately. This enables seamless communication between repository events and the backend AI processing services.

The frontend for the prompt management system is constructed using the Bootstrap 5 framework, which provides the foundational layout and core UI components for the entire admin dashboard. Custom CSS is then layered on top to fine-tune the appearance, adding features like hover effects on prompt cards and a monospace font for the YAML editor. Dynamic page generation is handled by Jinja2 HTML5 templates, which use template inheritance to maintain this consistent UI structure while rendering project-

specific views and forms. Integration with version control is achieved directly through the GitLab API, which allows the system to handle webhook-based events from merge requests, post automated review comments, and analyze file changes.

Configuration across the system is managed through a flexible YAML-based system. This allows for structured prompt templates complete with metadata, project-specific customizations, and the organization of prompts into logical categories such as security and performance. For quality assurance, the pytest framework is employed for comprehensive automated testing, covering unit tests for CLI commands, integration tests for APIs, and configuration validation. Advanced code analysis is further enhanced by AST (Abstract Syntax Tree) Parsers for multiple languages, which enable deep dependency analysis and code structure examination.

To support visualization, the system incorporates Mermaid.js for automated diagram generation, producing visual representations of code architecture and workflows. These diagrams are rendered into image formats on the server side using Puppeteer, a headless browser tool that ensures consistent output. Core data processing and file handling leverage robust Python standard libraries like pathlib for file system operations and subprocess for executing external commands. Security is a foundational concern, with Werkzeug providing critical web security functions such as password hashing and session management, while all sensitive credentials are securely managed through environment-based configuration.

The application implements a straightforward authentication system. For the web interface, user credentials are verified using check_password_hash() to validate login attempts against pre-configured admin users. Session management is handled natively by Flask, with a secret key sourced from environment variables to secure session data. API access is protected through a dual-layer approach. When users are already authenticated via the web session, they can access API endpoints directly. For external integrations, the system accepts an X-API-Key header that must match an API key stored in environment variables. All sensitive configuration, including the Flask secret key, admin credentials, and API keys, is managed through environment variables to maintain separation from the application code.

This technological stack creates a robust, scalable system that combines modern web technologies with advanced AI capabilities, providing both automated code analysis and user-friendly management interfaces. The architecture supports both automated workflows through n8n and direct user interaction through web interfaces and CLI tools.

3.2 Features Description

During the internship, a lot of features were implemented and integrated into the workflow. Thus, a description of each functionality is needed in order to understand in more depth the complexity of the project.

Automated Code Review: The automated code review is an AI-driven tool that assists developers

in finding code issues before merging into the main project. It uses AI to examine code logic, algorithm efficiency, maintainability, security, and adherence to coding standards.

The goal is to confirm code meets quality standards and to prevent bugs or performance issues in the final product. Developers can initiate this feature via GitLab comments like ”@bot review changed files” or ”@bot review”. It also works through the API using the /review endpoint, or via the command line with devai review code.

The system gives a markdown report with a dashboard that breaks down issues by severity. For each problem, it gives specific advice on how to fix it. It pulls code files from the GitLab repository using secure API calls. Google Vertex AI’s Gemini model is then used to analyze the code for logical errors, performance bottlenecks, security risks, and style mismatches. The analysis offers guidance based on development practices, useful for maintaining code quality across the team.

MR Analysis: The merge request analysis tool is an automated system that looks at how large, complex, and impactful the code changes are in the merge requests. It provides some metrics to help reviewers understand the scale and risk level of proposed modifications.

It runs automatically when merge requests are made or changed in GitLab. It gives complexity scores from 0 to 10 and change stats. This data goes into charts and summaries, which show files added, removed, changed, and the methods affected.

The feature works by examining git diff data through GitLab’s API to identify all files touched. Each change is marked as added, changed, or removed. Complexity scores take into account file count, how often methods are changed, and how big the changes are. The numbers are used to make analysis and reporting better in other system functions.

Summary Generation: The summary generation feature gives developers a short overview of code changes in merge requests. This makes it simpler for team members to understand the overall impact and purpose of the changes being proposed.

The feature analyzes modifications in a merge request and produces short summaries that highlight the main functional changes and what they are supposed to do in the system. It gives a clear picture of what changed and why. Users can activate this feature by adding specific triggers in GitLab or through direct API calls. For example, someone could just comment ”@bot summary” or ”summary” in a merge request and the system will generate a summary.

Once triggered, users get a formatted comment in GitLab with a summary paragraph that focuses on key functionality and the purpose behind the changes. It skips over technical details. The system works by pulling merge request data from GitLab’s API, which includes info about changed files, additions, deletions, and other modifications. Then it uses AI powered by Google Vertex AI to process this data and create a summary that focuses on functional changes and business impact. It filters out things like placeholder text

or documentation updates that don't actually represent real code changes.

Content Analysis: This feature enables flexible code examination across different scopes within a project. It provides developers with a full analysis result that matchs current workflow context. Whether a developer needs a broad overview of the entire codebase or a focused look at recent changes, this system makes sure they get relevant information without being overwhelmed by unnecessary details.

Content Analysis activates when users add specific comment triggers in their GitLab merge requests using natural language phrases that show their analysis scope preferences. Comments like "all context" or "changed context" tell the system's text classification system to automatically recognize and process the intended analysis depth. When triggered, users will see analysis results posted as comments within the merge request. The outputs adjust based on the selected scope.

The feature uses natural language processing to classify incoming user comments and route them through the right processing pathways. When someone requests "all context" analysis, the system retrieves and examines the entire repository content through GitLab's API. For "changed context" requests, the analysis narrows down to just the files affected by the current merge request. The AI then executes the corresponding depth of analysis and delivers results in formats that are easy to understand within GitLab's interface.

Diagram Generation: The diagram generation feature creates visual representations of complex code structures. By generating various diagram types that show how different code components interact, this feature uses GitLab's native Mermaid rendering capabilities to transform abstract code structures into accessible visual formats. This visual approach is especially useful for teams managing large or complicated projects.

Activation happens when users add diagram-specific requests into their code review commands, using syntax like "@bot review –diagram sequence" to specify the desired diagram type. The system then generates and displays the diagram directly within the GitLab merge request comment using native Mermaid syntax.

Users can expect to see properly formatted diagram code that renders as clear visual graphics within GitLab. This includes various diagram types like dependencies, sequences, classes, and component relationships. The feature works by first parsing the entire codebase through Abstract Syntax Tree methodologies to understand structural relationships and code organization. This structural insight then gets translated into Mermaid diagram syntax, making sure it's fully compatible with GitLab's native Mermaid renderer. Validation processes check each diagram for correct syntax and compatibility before presentation, which guarantees that visual representations stay accurate.

Issue Creation: Issue creation feature automatically transforms critical code review findings into GitLab issues. This makes sure high-priority problems actually get the attention they need in the project's

management system. By creating a direct path from code analysis feedback to formal issue tracking.

The feature kicks in when users add specific comment triggers after code review completion, usually through commands like ”create issues for problems”. Once it’s activated, users will see new issues pop up in their GitLab project. Each one details individual critical and high severity findings with the right categorization and priority labels. These issues include all the important stuff like specific file names, method locations, and detailed problem descriptions, so developers can actually understand and fix the issues without much trouble.

The feature works by scanning through previously generated code review results to find issues marked as CRITICAL or HIGH severity once it gets the trigger command. Then the system uses GitLab’s API to automatically create structured issues and assigns relevant titles based on the severity levels and issue types it found. Each issue gets filled with technical details and context, which gives developers a good starting point for fixing problems and following up on them later.

Release Notes: Release notes, generates comprehensive changelog documentation for code releases. It transforms technical commit messages and merge request changes into narratives that communicate new features, bug fixes, and significant updates to users. This feature helps development teams track version specific changes.

When developers include specific comment triggers within merged merge requests, commonly using phrases like ”@tag review” to begin the release notes generation workflow. When activated, users will see structured markdown content returned to the project, organizing changes into distinct categories like new features, bug fixes, improvements, and breaking changes. The output also includes automatic version tagging and direct release publication within GitLab’s native release management system, creating a completely streamlined release documentation process.

The feature works by analyzing merged commits and changes across different version, using artificial intelligence to categorize and explain changes. This process creates both polished release notes content and actual GitLab releases, complete with version tags and comprehensive notes. This makes sure every release gets documented within the project’s version control system.

Interactive Comments: The interactive comments feature is basically the main way developers communicate with the GitLab Code Review Bot through GitLab’s comment system. This makes the bot accessible to team members, so they can use different code analysis features without needing to know specialized commands.

The feature starts working automatically when users write comments in merge requests. Users can trigger specific features by writing natural language requests, anything from asking for a summary to requesting a full review. Once activated, users get responses that match what they asked for, like detailed analysis result, or confirmations. For example, asking for help returns a list of available commands, while

asking for a specific analysis will activate that feature.

The system uses machine learning-based text classification to understand user comments and figure out what they're trying to do. It sends validated requests to the right features. When the system is not confident about what a comment means, it gives helpful guidance about available commands. This creates a learning environment where users gradually pick up on how the system works and how to interact with it.

3.3 Security Measures

In this section is provided an overview of the main security issues relevant to the project. The focus will be on outlining protective measures, identifying potential risks, and ensuring the system operates in a secure and reliable manner.

The integration between n8n and GitLab relies on a personal access token (PAT) that enables authentication and authorization for repository actions. Because the PAT directly inherits the permissions of the account that created it, it poses a security risk if compromised. To manage this, one of the best solutions is to avoid the use of the tokens from the high-privilege accounts and instead create a new developer-level account, generate the access token from its settings, and then connect this token to the n8n workflow. Within n8n, the PAT is stored securely using its credential storage system, which provides encryption and access control. Additionally, if more people have access to the workflow, add a restriction from n8n settings that allows only the admin to access the credentials. This layer reduces the risk of exposure of credentials by mistake.

Another potential security issue might appear in the connection through webhooks. Webhooks act as the communication bridge, sending to the n8n workflow events and updates made in the repository. Because the data is sent across the internet, securing the path is also a priority. Thus, all webhook communications were designed to use HTTPS requests, which provide encryption of the data sent, preventing attackers from intercepting or altering it. For additional protection, a secret token has been added to the webhook so that n8n can validate the authenticity of incoming requests. The webhook itself is also limited to only the repository events needed for automation. This reduces the attacker's possibilities of exploiting the webhook.

The integration with the AI service requires careful management to ensure both functionality and security. In this project, the AI is connected in two distinct stages: first, when we use the AI in n8n to manage the user input and determine the appropriate action to follow; second time we implemented the connection to perform the code review, where the AI analyses and provides structured feedback according to the provided context (changes made or all files depending on user) and the template how the feedback does look. For both stages, authentication relies on a private key, ensuring that only authorized applications can access the AI services. This key is treated as a sensitive secret and is not exposed directly in code or logs. Within the workflow environment, only administrators have permission to manage these credentials.

3.4 Application Workflow

The assistant runs automatically, taking a code change and producing feedback without the developer having to do anything in between. Below it is explained how the parts of the system connect and how the information flows during analysis.

Trigger Mechanism: The workflow begins when a developer performs a specific action in the version control system. The most common trigger is opening a merge request, another being writing comments with specific requests. At this point, Git webhooks that have been configured in the repository detect the event and send an HTTP POST request to the specified endpoint. It contains all the necessary metadata about the change including the branch name, commit identifiers, author information, and references to the modified files.

Data Collection and Preprocessing: After the webhook, the system extracts the necessary information from it. This includes identifying which files were changed, added, or deleted, and retrieving the actual code differences. Depending on the configuration, the system can work with just the changed lines, the full context of modified files, or even the entire project structure.

The code retrieval step involves making API calls to the Git hosting platform to fetch file contents. Authentication tokens are used to guarantee secure access. If the merge request includes multiple commits, the system merges all changes across all of them to provide a complete picture of what has been modified. In cases where extra information is needed, for example when understanding dependencies or architectural relationships, the system can also retrieve all the files from the repository, but only on developer request. This approach helps the AI model make better decisions by giving it more information, but at the same time is more challenging and can take more time to process.

Prompt Construction and AI Invocation: The next step is making the prompt that will be sent to the AI model. The prompt includes:

- the instructions for what kind of review is required;
- the code changes themselves;
- extra context such as file paths or commit messages;
- specific points to check, such as security issues or style problems.

The prompt is sent to the VertexAI language model through an API call. The model processes the input and generates a detailed review based on the patterns it has learned and the instructions provided. The response is returned in a structured format, typically JSON, which makes it easy to parse and process programmatically.

Response Processing and Formatting: The system processes the response to ensure it meets quality standards. This includes validating the structure, extracting key information such as issue severity and

code suggestions, and filtering out any irrelevant feedback. The processing step may also involve additional transformations of information in order to better fit in the Gitlab comments section. For example, if the AI suggests code improvements, the system can format these as inline comments that appear directly on the relevant lines in the merge request interface.

Feedback Delivery: The final step is delivering the review feedback to the developers. This is done by posting comments directly to the merge request through the Git platform's API. The comments are organized by file and line number, making it easy for developers to locate and address each issue. It can also create summary reports that highlight recurring issues across projects, and developers can provide custom prompts if they want the feedback to focus on certain aspects.

Docker Containerization and Deployment: The entire workflow is packaged and deployed using Docker containers. Containerization ensures that the system runs consistently across different environments, whether on a local development machine, a staging server, or a cloud infrastructure. The Docker image includes all necessary dependencies, such as the n8n workflow engine, API client libraries, and configuration files. Environment variables are used to manage sensitive information like API keys and webhook URLs, ensuring that the system remains secure and adaptable to different deployment scenarios. When the Docker container starts, it initializes the n8n workflows, registers webhook endpoints, and begins listening for incoming events.

Error Handling and Logging: Throughout the workflow, the system implements error handling to ensure robustness. If a webhook payload is malformed, if API calls fail, or if the AI model returns an unexpected response, the system logs the error and attempts to recover gracefully. Developers are notified if critical failures occur, such as being unable to post review comments. Logging is comprehensive and includes timestamps, request identifiers, and details about each step of the workflow. This makes it possible to diagnose issues, monitor system performance, and audit the review process for compliance purposes.

The architecture can be changed, though it is not simple. Plans exist to make scaling and editing easier in the future. The workflow can be customized to match the specific needs of different teams. For example, some organizations may want reviews to be triggered only on certain branches, while others may require additional checks before feedback is posted. The n8n platform allows these variations to be implemented through visual workflow editing, without requiring changes to the underlying code. This modularity also means that new features can be integrated into the workflow without disrupting existing functionality. New features, such as support for other AI models or version control systems, can be added without breaking the current setup. The first installation, however, can be difficult and may cause issues.

Conclusions

During this internship, the following project was designed and planned as the foundation for a secure and automated development workflow. The main findings indicate that combining workflow automation through n8n, backend support in Python, and AI-driven text analysis (via Google Vertex AI models) provides a way to streamline code review processes, generate summaries, and manage issue ticket releases. Furthermore, the integration with GitLab ensures that the system fits naturally into existing development pipelines.

Among the key achievements are the design and implementation of functional workflows that handle merge requests, automatic release note creation, and issue generation. A modular system architecture was established, supported by component and behavioral modeling. Security aspects such as credential management, token-based authentication, and secure API access were also addressed. Challenges encountered included ensuring proper handling of webhook events, maintaining consistent formatting of AI outputs, and designing workflows that remain adaptable to future extensions.

The next steps for further development in the PBL course will focus on implementing a fully functional prototype, refining the automation pipelines, and extending the system with additional features such as advanced diagram generation and improved changelog management. Early considerations for implementation highlight the need for scalable deployment options, improved error handling, and continuous testing to ensure both reliability and security.

In summary, the internship successfully achieved its objectives of researching, designing, and planning a secure automation tool for software development processes. The results provide a solid basis for the upcoming implementation phase, ensuring a smooth transition from research to practice.

Bibliography

[1] U. Cihan, V. Haratian, A. İçöz, et al., "Automated Code Review In Practice," *arXiv:2412.18531*, Dec. 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.18531

[2] M. Di Biase, M. Bruntink, A. Van Deursen, and A. Bacchelli, "The Effects of Change Decomposition on Code Review—a Controlled Experiment," *PeerJ Computer Science*, vol. 5, p. e193, May 2019. [Online]. Available: https://doi.org/10.7717/peerj-cs.193

[3] W. Charoenwet, P. Thongtanunam, V.-T. Pham, and C. Treude, "Toward Effective Secure Code Reviews: An Empirical Study of Security-Related Coding Weaknesses," *Empirical Software Engineering*, vol. 29, no. 4, p. 88, 2024. [Online]. Available: https://doi.org/10.1007/s10664-024-10496-y

[4] JetBrains, "How much time developers spend on code reviews," LinkedIn, 2024. [Online]. Available: https://www.linkedin.com/posts/jetbrains_how-much-time-developers-spend-on-code-reviews-activity-7308224032296869890-kEOL

[5] Reddit, "Code review: How much time is right?" r/softwaredevelopment, 2023. [Online]. Available: https://www.reddit.com/r/softwaredevelopment/comments/10yajzu/code_review_how_much_time_is_right

[6] S. K. et al., "A Study on Code Review Practices and Challenges," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 12, no. 2, pp. 1–8, 2023. [Online]. Available: https://www.ijrte.org/wp-content/uploads/papers/v12i2/B76660712223.pdf

[7] A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," in *Proc. International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICSE202013-codereview.pdf

[8] GitLab, "Challenges of Code Reviews," GitLab Blog, 2023. [Online]. Available: https://about.gitlab.com/blog/challenges-of-code-reviews

[9] Google, "No AI Overload Just Yet: Google's New Survey Reveals How Developers Are Really Using AI at Work," TechRadar, 2024. [Online]. Available: https://www.techradar.com/pro/no-ai-overload-just-yet-googles-new-survey-reveals-how-developers-are-really-using-

[10] Hypersense, "Importance of Code Reviews in Software Development," Hypersense Blog, Aug. 2024. [Online]. Available: https://hypersense-software.com/blog/2024/08/14/importance-of-code-reviews-in-software-development

[11] M. B. et al., "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data," in *Proc. International Symposium on Empirical Software Engineering and Measurement*, 2013. [Online]. Available: https://www.researchgate.net/publication/260648247_The_Impact_of_Design_and_Code_Reviews_on_Software_Quality_An_Empirical_Study_Based_on_PSP_Data

[12] A. Author et al., "Title of the arXiv Preprint," *arXiv:2405.18216*, 2024. [Online]. Available: https://arxiv.org/html/2405.18216v1

[13] ISStephy Group, "AI Code Review Assistant," GitLab repository, 2024. [Online]. Available: https://gitlab.com/isstephy-group/assistant.git