Bilkent University

Department of Computer Engineering

Alkahest

An auto code reviewing extension for VSCode

CS 453 Application Lifecycle Management

Term Project Progress Report

Alper Göçmen 22002948

Sarper Arda Bakır 21902781

Deniz Tuna Onguner 22001788

Section 1

Instructor: Eray Tüzün

Teaching Assistant(s): Yahya Elnouby

Progress Report
April 7, 2024, Sunday

1- Problem Statement

Code review is crucial in software development, facilitating collaboration, identifying bugs, and ensuring code quality. For developers, active participation in code reviews fosters a culture of learning and continuous improvement, leading to the delivery of higher-quality software products. According to a study by Bacchelli and Bird (2013), code reviews catch defects early and contribute to knowledge sharing and team cohesion, ultimately improving software reliability and maintainability [1]. With a global developer community expected to reach 28.7 million individuals by 2024 [2], peer code review tools have become increasingly popular.

In software development, practitioners encounter the challenge of navigating through multiple websites, tools, and applications to perform essential tasks such as code duplication, bug identification, and resolution. The fragmented nature of existing solutions often leads to inefficiencies, as developers must switch between disparate platforms, sacrificing productivity and workflow continuity. Spending an average of 20 minutes per day switching between various tools and websites for tasks leads to a 15% loss in productivity [3]. Additionally, developers typically juggle between 5 different platforms, further hindering workflow continuity and collaboration.

To address these challenges effectively, software practitioners require a cohesive and integrated toolset consolidating diverse functionalities within a single environment. Such a comprehensive solution is necessary for workflow optimization, collaboration, and code quality enhancement efforts within the software development community. The proposed VSCode plugin aims to mitigate the challenges associated with fragmented toolsets and elevate the software development experience for practitioners worldwide. This approach will improve developer productivity by 20% and enhance code quality by 10%, benefiting millions worldwide [4]. By providing a unified platform within the familiar environment of VSCode, developers can streamline their workflows, reduce context switching, and focus more effectively on writing high-quality code.

2- Application Use Case

Alkahest is a comprehensive Visual Studio Code (VSCode) plugin tailored for software practitioners. This plugin aims to scan users' projects to streamline bug detection, code duplication identification, and project evaluation tasks by seamlessly integrating with popular development tools such as SonarCloud, SonarQube, Gemini API, and StackOverflow. By providing a unified environment within VSCode, Alkahest empowers users to perform these essential tasks without switching between disparate platforms, saving valuable time and improving productivity.

In contemporary software development practices, developers often use multiple platforms to conduct code reviews and address software issues. For instance, conventional procedures necessitate uploading project repositories onto platforms such as SonarCloud for comprehensive code reviews. Subsequently, developers typically resort to external resources like StackOverflow or Gemini to troubleshoot encountered bugs. However, Alkahest signifies a pivotal departure from this conventional approach. By seamlessly integrating functionalities within the Visual Studio Code (VSCode) environment, Alkahest allows developers to undertake these essential steps without relying on external platforms.

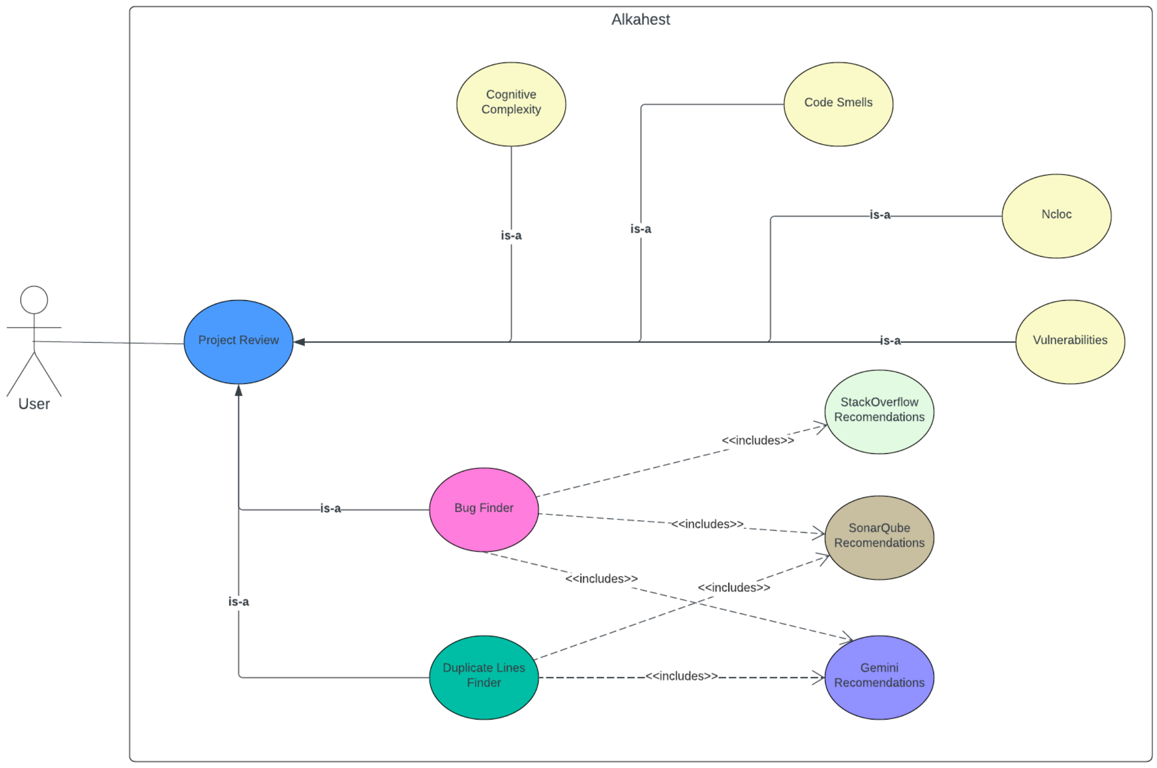


Figure 1: Alkahest Use Case Diagram

3- Business Questions

1. As a software developer/engineer, I want to detect possible bugs and duplicated lines in my code within my VSCode environment on my local before initiating a pull request, so that I can streamline my debugging process and improve code quality without the need to switch between different tools or platforms.
2. As a software developer/engineer, I want to have a seamless and time-efficient solution for code analysis and bug detection directly within my development environment, eliminating the need to manually navigate through external platforms for issue resolution.
3. As a programmer, I want to utilize Gemini AI of Google within my VSCode environment to receive suggestions and references to online sources on how to solve the bugs detected by Alkahest via SonarQube, so that I can expedite the bug-fixing process and optimize my workflow.
4. As a programmer, I want to effortlessly access relevant StackOverflow links and official documentation directly within my VSCode environment via AI-powered assistance provided by Alkahest, eliminating the need for extensive googling and facilitating quick access to valuable resources for issue resolution and learning.
5. As a project manager/team leader, I want to evaluate projects using SonarCloud directly from VSCode, so that I can quickly assess the overall code health and make informed decisions to ensure project quality and maintainability.

4. Application Requirements

Functional Requirements

- Overall Code Review Feature
  - Description: The extension will analyze to code within the developer's VSCode environment and auto-review it with SonarQube to provide overall code review with following metrics: number of bugs, number of code smells, duplicated lines density (percentage), number of uncommented lines, cognitive complexity, and vulnerabilities.
  - Acceptance Criteria:
    - The extension shall scan the codebase and identify all metrics given above.
    - The extension shall showcase these metrics identified with their values and highlight them within the environment.
- Bug Detection Feature
  - Description: The extension will analyze the code within the developer's VSCode environment to detect possible bugs.
  - Acceptance Criteria:
    - The extension shall identify common coding errors and possible bugs such as null pointer exception, division by zero exception, array out of bounds error, infinite loops etc.
    - The extension shall highlight, to emphasize, these bugs found within the VSCode environment.
- Duplicated Lines Detection Feature
  - Description: The extension will analyze the code within the developer's VSCode environment to detect duplicated lines of code.
  - Acceptance Criteria:
    - The extension shall identify the lines of code that are unnecessarily repeated and possibly be reduced without causing any loss in functionality.
    - The extension shall highlight these duplicated lines detected within the VSCode environment.
- Integration with Gemini AI for Fixing and Suggestions
  - Description: The extension will integrate with Gemini AI provided by Google to offer solutions, suggestions and relevant references, such as Stackoverflow links, for resolving bugs and reducing duplicated lines detected by Alkahest via SonarQube.
  - Acceptance Criteria:
    - When a problem is detected, the extension shall utilize Gemini AI to provide relevant suggestions and references from online sources, which shall include potential solutions, code snippets, and relevant links.

        - Developers shall have the option to accept or reject suggestions provided.
- Integration with VSCode
    - Description: The system will integrate with Visual Studio Code (VSCode) to provide code analysis and bug detection functionalities.
    - Acceptance Criteria:
        - The extension shall be installed from the VSCode marketplace.
        - Developers shall be able to initiate code analysis directly within their VSCode environment without switching between different tools or platforms.
- Integration with SonarQube
    - The system will integrate with SonarQube to enable developers to review the projects directly within VSCode.
    - Acceptance Criteria:
        - Developers and project managers/team leaders shall be able to view code review metrics within the VSCode environment.
        - The integration with SonarQube shall speed up the evaluation of the overall health of the code.

Non-Functional Requirements

- Performance
    - Description: The extension will provide real-time auto code review with a minimal impact on the developers' workflows.
    - Acceptance Criteria:
        - The extension shall analyze code within seconds for small to medium-sized codebases; and, for large codebases, it shall return the results within a reasonable timeframe; e.g., seconds to minutes.
- Scalability
    - Description: The extension has to be scalable in order to support multiple codebases and simultaneous users.
    - Acceptance Criteria:
        - Performance should not noticeably decrease as the number of codebases and simultaneous users rises.
- Reliability
    - Description: The extension should be reliable and accurate in detecting bugs and finding duplicated lines within the codebase.
    - Acceptance Criteria:
        - The bug detection shall accurately identify coding errors and potential bugs.
        - The system shall detect duplicated lines or blocks of code within the codebase, minimizing false positives and false negatives.

5- Success Criteria

5.1. User Adoption

One of the pivotal metrics for assessing a plugin's success is the count of downloads and active users. A good plugin typically demonstrates a consistent growth trajectory in its user base. Therefore, monitoring both the number of downloads and active users provides valuable insights into the plugin's performance and success over time. To be specific, we expect approximately 100 users in the first month. The first hundred people will be our friends in the CS department. At the end of first year, we expect at least 10K since when we analyzed the statistics in the VSCode Marketplace, a new plugin not developed by a large company, should reach 10K people. With correct advertisement, these numbers can be increased.

5.2. Productivity Improvement

As we stated in our problem definition, switching between different tools reduces productivity. Actually, it is hard to analyze productivity since we need to monitor the user's activities in daily life. However, we think that we can measure productivity improvement by conducting user surveys or interviews to understand if the plugin has indeed reduced the time they spend on switching between platforms. With the consent of some users, we could do some tests. For instance, for a simple task, some users could be chosen and we could measure the time that they spend with and without the plugin. Therefore, we could see the time reduction thanks to the plugin. We consider success in terms of productivity, if total time is reduced by at least 20 minutes just caused by switching platforms in one day.

5.3. Code Quality Enhancement

As we stated in the problem statement, the plugin aims to enhance code quality. We will use the quality metrics and methods in the Eray Tüzün’s paper [5]. There are various quality metrics such as Code Validity, Code Correctness, Code Security, Code Reliability, and Code Maintainability. We will use code validity and correctness as quality metrics in our project. For the validity part, we expect at least 80% validity from the scanned results. In other words, if there are 10 bugs or errors in the scanned code, the results should give at least 8 bugs or errors. In the paper, the LLM’s can provide at least 90% validity while generating code, so we thought that 80% threshold is quite reachable and reasonable. The correctness part is about the solution that is provided to the user. In the paper, the LLM’s generate correct code 50% of the cases. If we give direct code solutions to the user, we expect minimum 50% correct generated code from the LLM’s. Otherwise, we will guide the user to some solutions or similar questions on the internet. In the second case, we want at least 3 sources to guide the user. Also, we want these sources to include necessary keywords in the problem. We will consider success as if the output includes at least 3 sources including important keywords in the bugs or errors.

5.4. Code Duplication Reduction

Another feature of the plugin is finding the duplicated code lines. Measuring the reduction in the code duplication after using the plugin is important. This can be quantified by comparing the amount of duplicated code before and after the plugin’s usage. The main problem of finding duplicated code lines is that most code clones aren’t identical text, they are copy and paste edited text. To find them accurately, tools need to match similar but not identical code fragments. This means that a tool's accuracy can decrease if it does not understand the language syntax. Thus, we set the threshold at 80% since even if the exact matches are found, similar matches could not be found in some cases. For instance, in a 1000 line codebase, if there exist 500 duplicated lines, which are exact matches of some other lines. we expect from the plugin to find at least 400 of these duplicated lines. Thus, we will be satisfied if 80% of the exact duplicated lines are found.

6- Sample Data

We designed the plugin to analyze the local files. There will be no need to upload files anywhere. In a VSCode environment, you must open the folder or file to do simple tasks such as looking at the files, writing the code etc. The same approach is implemented for the plugin. To analyze the code, the user needs to open a specific file or folder. Any file or folder opened at that time will be analyzed when the plugin is started. The limitations will be on the SonarQube side. In the SonarQube website, they have a variety of languages supported and you can access it by their official documentation provided in the references [6]. Any file including the supported languages and their extensions will give successful results such as .js .py etc. Otherwise, if the files are extensionless or they have conflicting extensions such as XML and Mule language conflicts in some files, the SonarQube will not analyze them. In these scenarios, users can remove these kinds of files from the analysis folder, or try to change file extension to handle conflicts. After changing the settings, they need to re-run the analysis for the changes to take effect. In conclusion, any file including supported languages and their extensions will be welcomed as an input [6], and any file without any extension or conflicting extensions will raise an error or result in wrong analysis.

7-Architecture

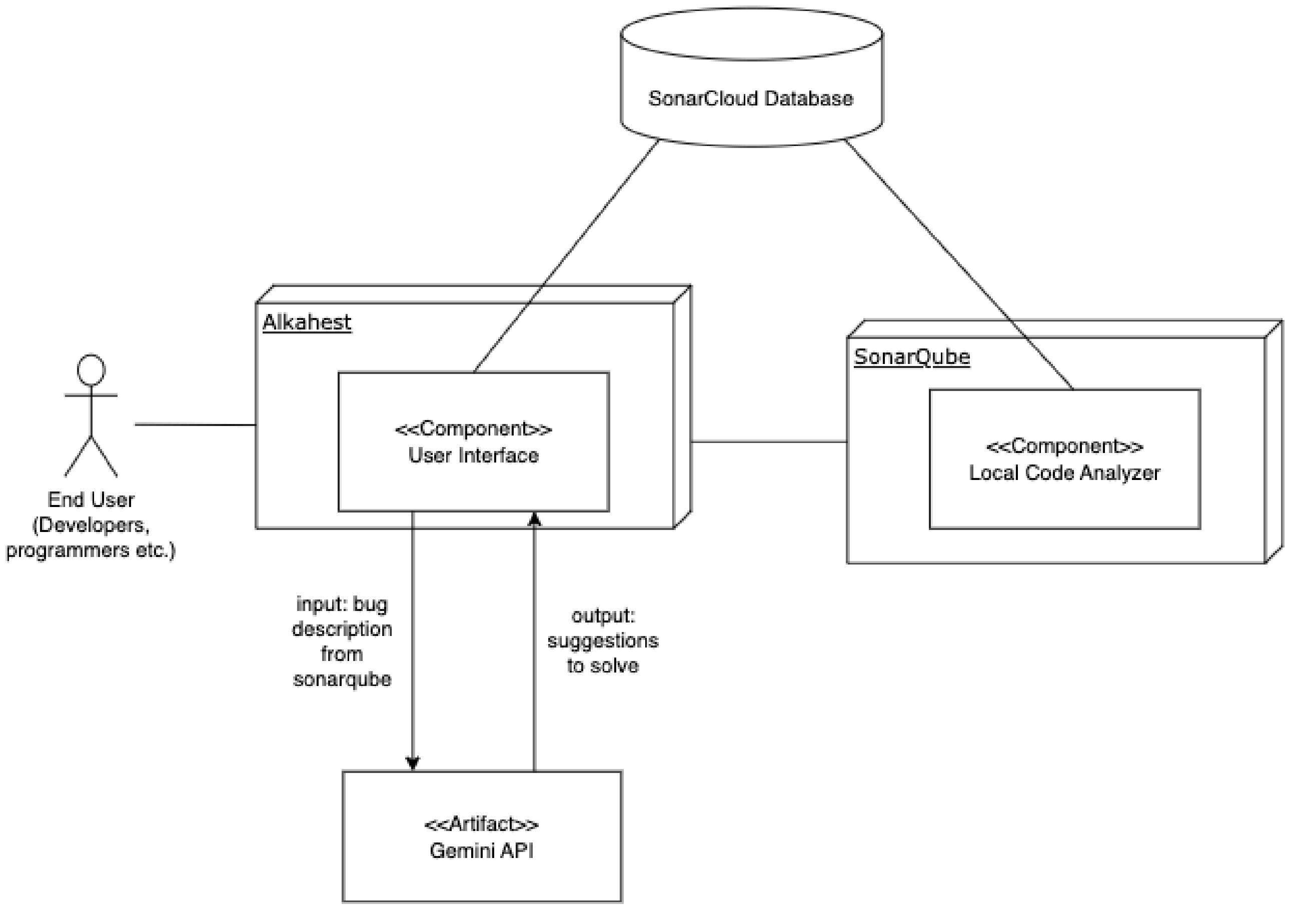


Figure 2: Architecture Diagram

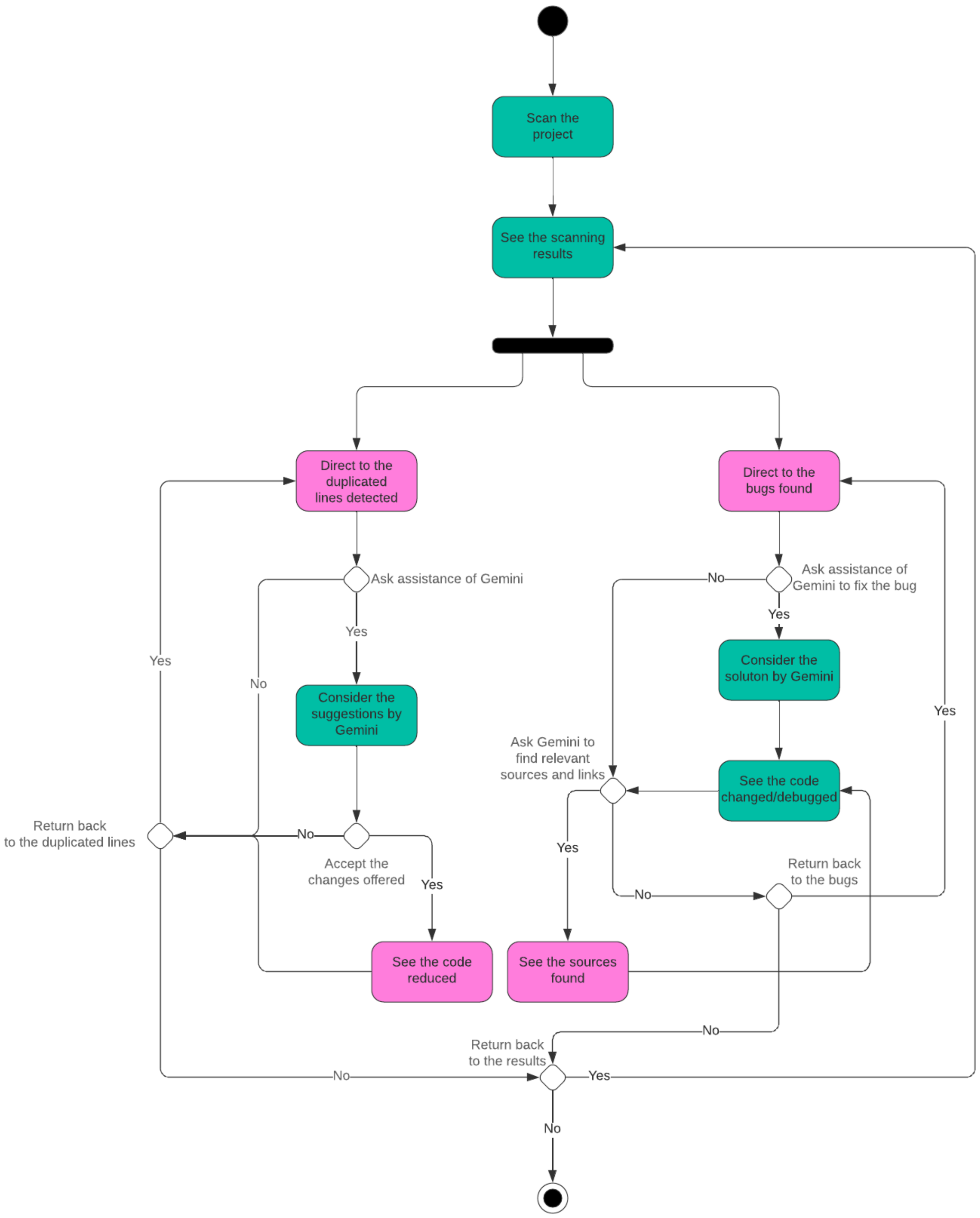

Scan the project
See the scanning results
Direct to the duplicated lines detected
Direct to the bugs found
Ask assistance of Gemini
Ask assistance of Gemini to fix the bug
Yes
No
Consider the suggestions by Gemini
Consider the soluton by Gemini
Ask Gemini to find relevant sources and links
See the code changed/debugged
Return back to the duplicated lines
Accept the changes offered
Return back to the bugs
See the code reduced
See the sources found
Return back to the results
Yes
No

Figure 3: Activity Diagram

8- Technical Benchmark

1. OpenAI / ChatGPT

ChatGPT excels in understanding natural language and can be customized for specific programming languages and coding styles. With access to a vast dataset, it can generate a wide range of potential solutions. However, it may struggle with grasping the broader context of code, resulting in suggestions that may not always be entirely relevant or accurate. While it's a powerful general-purpose language model, it's not explicitly optimized for code analysis and bug detection [7].

A significant limitation of ChatGPT is its restricted access to the internet, meaning it cannot retrieve data before January 2022. Additionally, it doesn't provide links to the sources it uses, unlike Gemini, which includes all citations and sources with each response. Since our project aims to guide users to solutions, with or without providing generated code, ChatGPT falls short in terms of providing sources or links. Moreover, its lack of up-to-date information post-cutoff date results in lower accuracy compared to Gemini when accessing new information or solutions.

Another drawback of ChatGPT is its tendency to prioritize creative text formats over precise solutions, whereas Gemini places less emphasis on creative output, with more emphasis on concise and precise solutions [7].

Given our project's need for guidance and exact solutions to users' problems, Gemini's approach aligns better with our requirements. Despite ChatGPT offering faster responses than Gemini, these two main issues lead us to prefer Gemini over ChatGPT.

2. Github Copilot

Copilot is integrated into popular code editors, seamlessly becoming part of a developer's workflow. With training on a massive volume of code, it can recognize coding patterns, suggest potential improvements, and identify duplication. Additionally, it continuously learns and enhances its suggestions based on user feedback and real-world code examples.

However, the primary limitation of Copilot is its dependency on OpenAI, with no explicit API available. This means we cannot utilize Copilot in our project implementation. Questions may arise about the necessity of our project when a built-in chatbot like Copilot exists. To address this, we emphasize the shortcomings of code generation tools. Our project focuses on guiding users to solutions, with or without generated code.

Notably, according to Eray Tüzün's paper, while Copilot and ChatGPT achieve over 90% code validity, their code correctness rates are 46.3% and 65.2% respectively [5]. Hence, we prefer not to rely solely on generated code. Instead, we aim to provide additional sources or solutions from the internet, which Copilot lacks.

In conclusion, the main reason for choosing Gemini over Copilot and ChatGPT is the aim of our project. Even if we give generated code to the users, they are not always accurate or correct. Thus, we want to provide guidance to the users about how they can solve the problem or where they can find the solution. Gemini is able to provide up-to-date links via searching the internet. Therefore, we conclude that Gemini is the most suitable LLM for our project.

3. Review Board

Review Board is a free and open-source web-based tool that provides a platform for code review, document review, and image review, all in one place. It allows you to review code changes, share feedback, and approve or reject changes. Review Board provides unified views into your code activity for commits, reviews, and comments. Review Board can be customized for an organization with its API, integrations, automated code review, and custom extensions. Your code and data stays private, secure, and in your control. However, to analyze local files, the files should be uploaded and they will be stored on the Review Board Server, similar to SonarQube.

The main reason for choosing SonarQube is that SonarQube is more geared towards analyzing code quality, with functionalities like code coverage assessment, detecting code duplication, and identifying rule violations. In contrast, the Review Board is primarily centered around code review and collaboration, particularly on pull requests, facilitating the review of code modifications, exchanging feedback, and approving or rejecting changes. Since we required a tool that offers thorough code quality analysis and supports various programming languages, we opted for SonarQube.

4. Crucible

Crucible by Atlassian is a powerful tool that provides structured code review processes, ensuring systematic bug and quality checks. It helps catch issues before they're integrated and allows for the review of the quality of commits. Crucible connects with popular issue trackers and development tools, enhancing its utility in a development workflow. However, it primarily relies on human input for code quality assessment and is less advanced in utilizing AI-driven detection of bugs and potential code duplication.

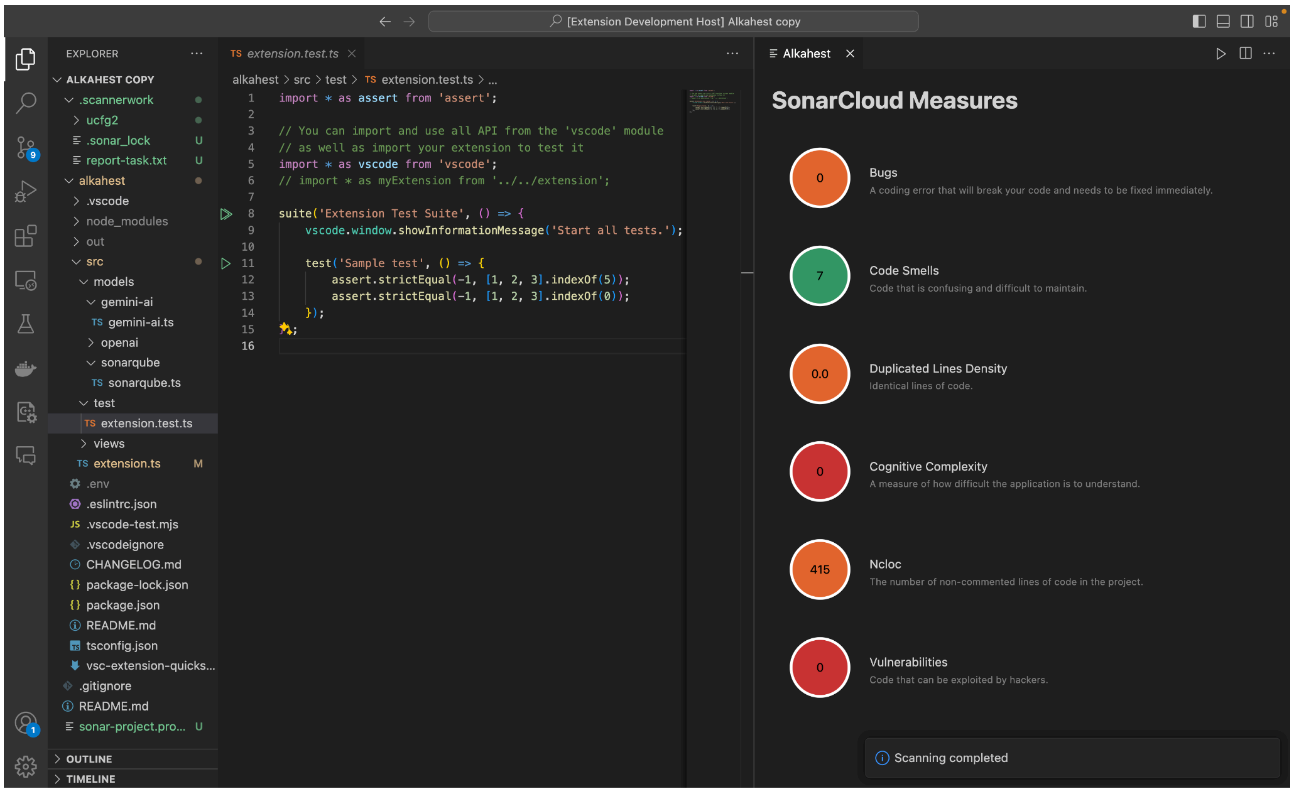
While Crucible is a robust tool, it does have some drawbacks. Reviews can become overwhelming if too many files are included, and there can be disruptions if a file in the repository changes during a review. It may also have issues with extensionless files or files with conflicting extensions, similar to SonarQube. Additionally, generating reports for individual developers can be tedious, and like many Atlassian products, Crucible has a bit of a learning curve and may require some time to master. Lastly, it can entail some overhead regarding configuration and integration with existing workflows. Despite these challenges, Crucible remains a valuable tool for many development teams. It's also worth noting that Crucible, like any software, can have security vulnerabilities, and users should stay updated with Atlassian's security advisories.

While SonarQube is more oriented towards code quality analysis, offering features such as code coverage, code duplication detection, and rule violation detection, Crucible primarily focuses on code review and collaboration, providing a platform for reviewing code changes, sharing feedback, and approving or rejecting changes. We needed a tool that provides comprehensive code quality and supports a wide range of programming languages, so we chose SonarQube.

In conclusion, as mentioned previously, the primary reason for selecting SonarQube is its emphasis on code quality analysis and its code duplication detection capabilities, unlike other tools that prioritize code review and collaboration.

Also, as we stated above, The Crucible has some drawbacks. However, there were no major drawbacks of the Review Board. The main reason for choosing SonarQube over the Review Board was the implementation process. We tried both but we could not manage the implementation of the Review Board to analyze the local files. With SonarQube, it was hard to implement but it was still manageable.

9- Set of Mock-Up Screens



```
import * as assert from 'assert';

// You can import and use all API from the 'vscode' module
// as well as import your extension to test it
import * as vscode from 'vscode';
// import * as myExtension from '../../extension';

suite('Extension Test Suite', () => {
    vscode.window.showInformationMessage('Start all tests.');

    test('Sample test', () => {
        assert.strictEqual(-1, [1, 2, 3].indexOf(5));
        assert.strictEqual(-1, [1, 2, 3].indexOf(0));
    });
});
```

SonarCloud Measures

0 Bugs
A coding error that will break your code and needs to be fixed immediately.

7 Code Smells
Code that is confusing and difficult to maintain.

0.0 Duplicated Lines Density
Identical lines of code.

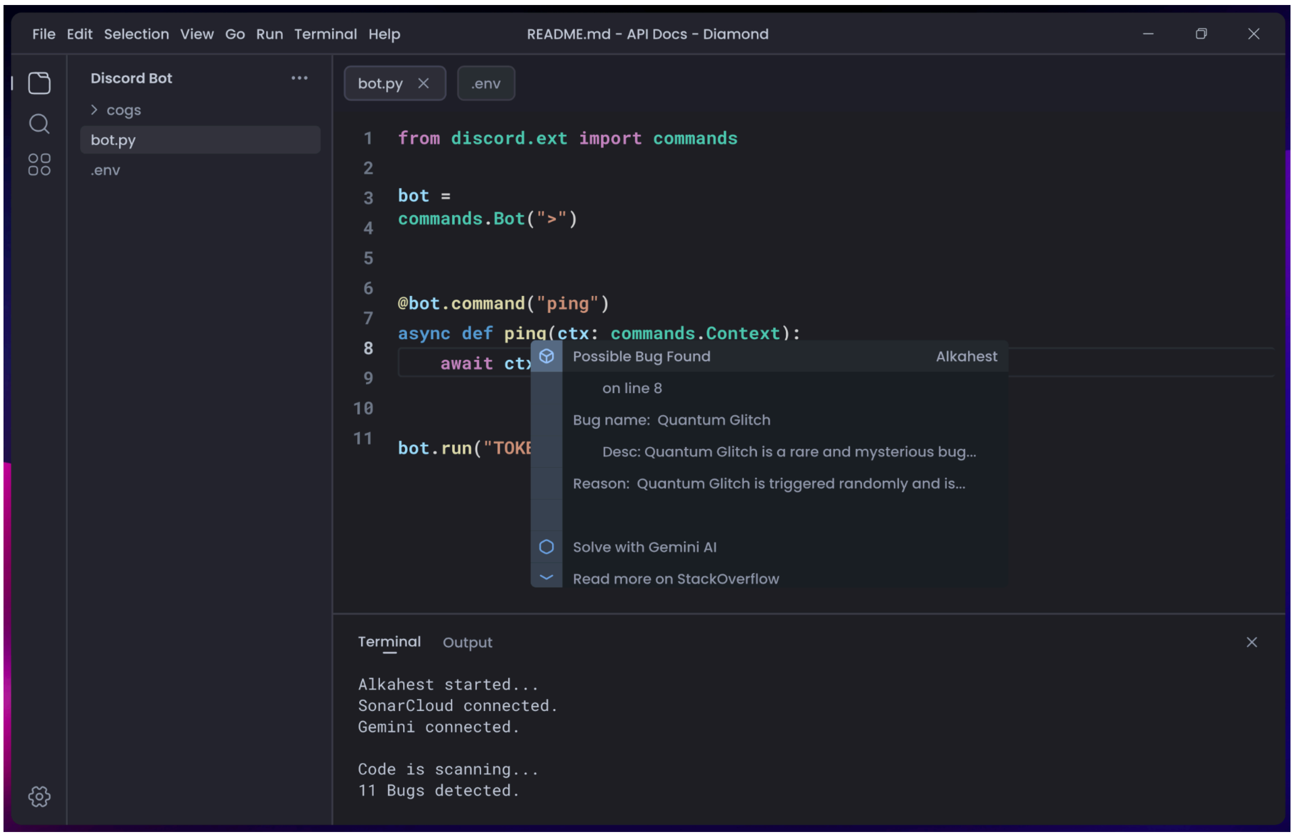
0 Cognitive Complexity
A measure of how difficult the application is to understand.

415 Ncloc
The number of non-commented lines of code in the project.

0 Vulnerabilities
Code that can be exploited by hackers.

Scanning completed

Figure 4: Alkahest Implemented Code Review Screen



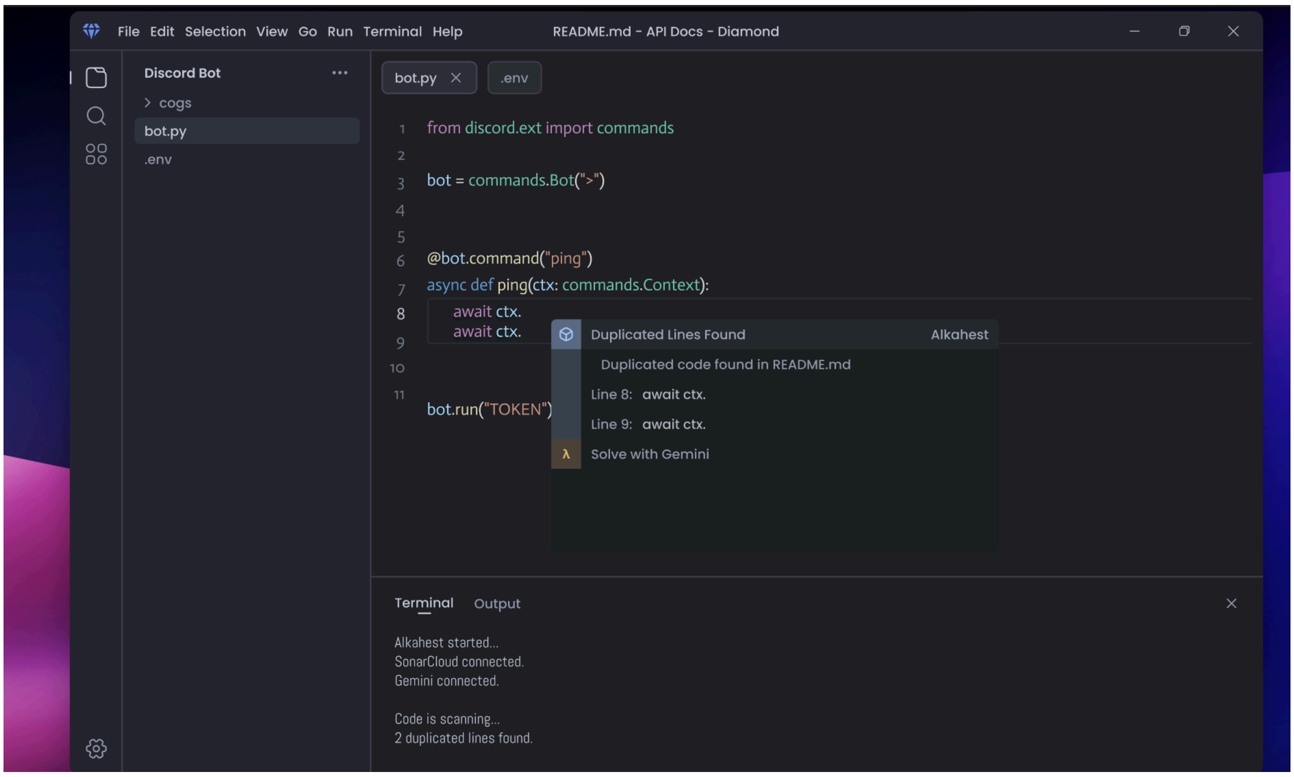
```
from discord.ext import commands

bot =
commands.Bot(">")


@bot.command("ping")
async def ping(ctx: commands.Context):
    await ctx

bot.run("TOKE
```

Possible Bug Found Alkahest
on line 8
Bug name: Quantum Glitch
Desc: Quantum Glitch is a rare and mysterious bug...
Reason: Quantum Glitch is triggered randomly and is...
Solve with Gemini AI
Read more on StackOverflow

Terminal Output

```
Alkahest started...
SonarCloud connected.
Gemini connected.

Code is scanning...
11 Bugs detected.
```

Figure 5: Alkahest Possible Bug Found Pop Up Message

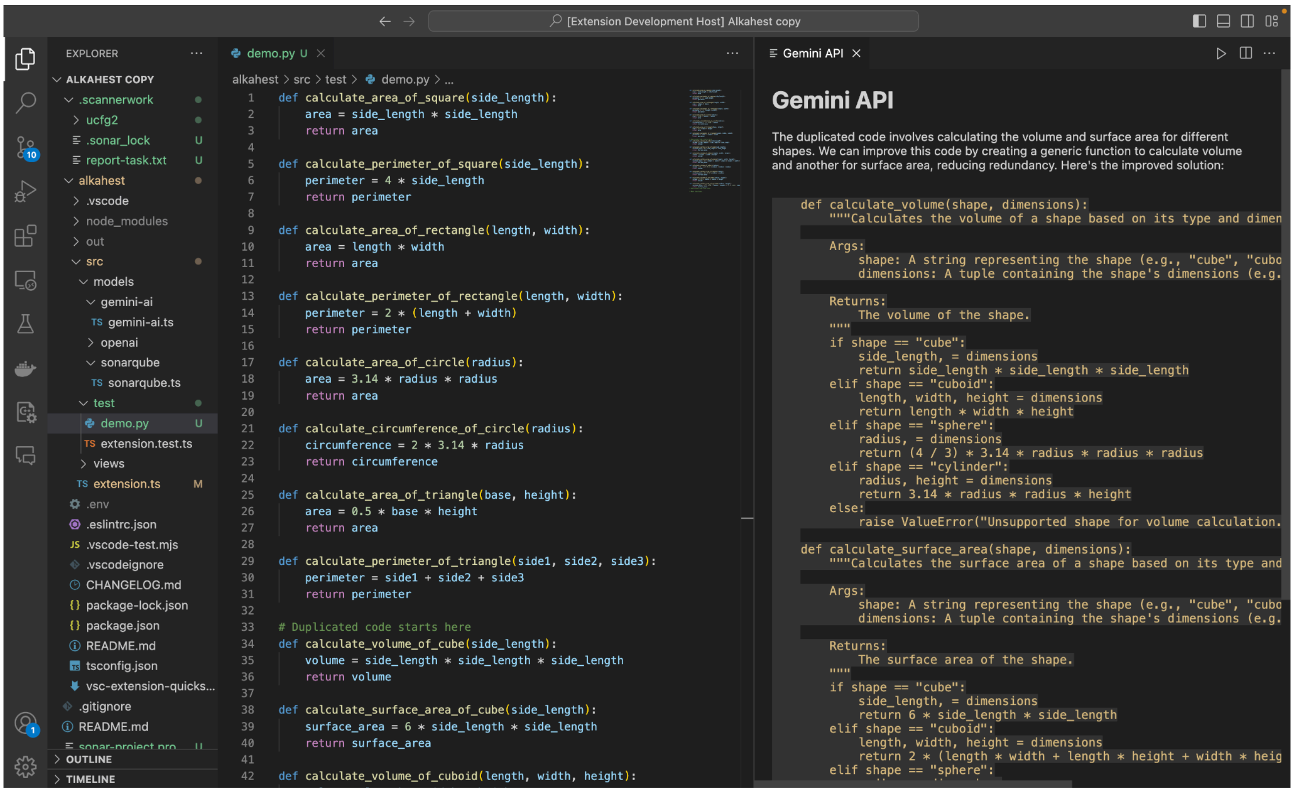

File Edit Selection View Go Run Terminal Help
README.md - API Docs - Diamond
Discord Bot
cogs
bot.py
.env
Terminal
Output

```python
from discord.ext import commands

bot = commands.Bot(">")


@bot.command("ping")
async def ping(ctx: commands.Context):
    await ctx.
    await ctx.


bot.run("TOKEN")
```

```
Duplicated Lines Found                    Alkahest
  Duplicated code found in README.md
Line 8:  await ctx.
Line 9:  await ctx.
Solve with Gemini
```

```
Alkahest started...
SonarCloud connected.
Gemini connected.

Code is scanning...
2 duplicated lines found.
```

Figure 6: Alkahest Duplicated Lines Found Pop Up Message



```python
def calculate_area_of_square(side_length):
    area = side_length * side_length
    return area

def calculate_perimeter_of_square(side_length):
    perimeter = 4 * side_length
    return perimeter

def calculate_area_of_rectangle(length, width):
    area = length * width
    return area

def calculate_perimeter_of_rectangle(length, width):
    perimeter = 2 * (length + width)
    return perimeter

def calculate_area_of_circle(radius):
    area = 3.14 * radius * radius
    return area

def calculate_circumference_of_circle(radius):
    circumference = 2 * 3.14 * radius
    return circumference

def calculate_area_of_triangle(base, height):
    area = 0.5 * base * height
    return area

def calculate_perimeter_of_triangle(side1, side2, side3):
    perimeter = side1 + side2 + side3
    return perimeter

# Duplicated code starts here
def calculate_volume_of_cube(side_length):
    volume = side_length * side_length * side_length
    return volume

def calculate_surface_area_of_cube(side_length):
    surface_area = 6 * side_length * side_length
    return surface_area

def calculate_volume_of_cuboid(length, width, height):
```

Gemini API

The duplicated code involves calculating the volume and surface area for different shapes. We can improve this code by creating a generic function to calculate volume and another for surface area, reducing redundancy. Here's the improved solution:

```python
    def calculate_volume(shape, dimensions):
        """Calculates the volume of a shape based on its type and dimen

        Args:
            shape: A string representing the shape (e.g., "cube", "cubo
            dimensions: A tuple containing the shape's dimensions (e.g.

        Returns:
            The volume of the shape.
        """
        if shape == "cube":
            side_length, = dimensions
            return side_length * side_length * side_length
        elif shape == "cuboid":
            length, width, height = dimensions
            return length * width * height
        elif shape == "sphere":
            radius, = dimensions
            return (4 / 3) * 3.14 * radius * radius * radius
        elif shape == "cylinder":
            radius, height = dimensions
            return 3.14 * radius * radius * height
        else:
            raise ValueError("Unsupported shape for volume calculation.

    def calculate_surface_area(shape, dimensions):
        """Calculates the surface area of a shape based on its type and

        Args:
            shape: A string representing the shape (e.g., "cube", "cubo
            dimensions: A tuple containing the shape's dimensions (e.g.

        Returns:
            The surface area of the shape.
        """
        if shape == "cube":
            side_length, = dimensions
            return 6 * side_length * side_length
        elif shape == "cuboid":
            length, width, height = dimensions
            return 2 * (length * width + length * height + width * heig
        elif shape == "sphere":
```

Figure 7: Alkahest Duplicated Lines Gemini API Recommend Screen

References

[1] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," IEEE Transactions on Software Engineering, vol. 40, no. 1, pp. 1-13, 2013.

[2] L. S. Vailshery, "Developers population worldwide 2018-2024," August 29, 2023.

[3] https://www.synapsestudios.com/learn/software-development-partners-should-avoid-context-switching

[4] https://www.xcubelabs.com/blog/a-comprehensive-guide-to-integrated-development-environments-ides/

[5] E. Tüzün, B. Yetiştiren, I. Özsoy, and M. Ayerdem, *Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT*. doi:10.48550/arXiv.2304.10778

[6] https://docs.sonarsource.com/sonarqube/8.9/analyzing-source-code/languages/overview/

[7] N. Rane, S. Choudhary, and J. Rane, “Gemini versus CHATGPT: Applications, performance, architecture, capabilities, and implementation,” *Journal of Applied Artificial Intelligence*, vol. 5, no. 1, pp. 69–93, Mar. 2024. doi:10.48185/jaai.v5i1.1052