|  |
| --- |
| **Everi Holdings Inc. - for internal use only** |
|  |
| AUTOMATED CODE REVIEW |
|  |
| Enterprise Architecture |
| 10/10/2024 |
|  |
| © 2023 Everi Payments Inc. and/or its subsidiaries or affiliates (collectively, “Everi”). This is Everi’s proprietary and confidential material. No part of this material may be disclosed, in any manner, to another party without the prior written consent of Everi. |

|  |  |
| --- | --- |
|  |  |

# Control Page

|  |  |  |
| --- | --- | --- |
|  | |  |
| Title: | AUTOMATED CODE REVIEW | |
| Overview: | Automated Code Review Tool, Use Cases | |
| Author: |  | |
| Reviewer: |  | |
| Department: | Enterprise Architecture | |
| Issue Date: | 10/10/2024 | |

Table of Contents

[Control Page 2](#_Toc180415582)

[Automated Code Review Overview 5](#_Toc180415583)

[Programming Languages Supported by Automated Code Review Tool: 5](#_Toc180415584)

[ For Organization Standards files: 5](#_Toc180415585)

[ For Code files: 5](#_Toc180415586)

[Key Features 6](#_Toc180415587)

[1. Code Review: 6](#_Toc180415588)

[2. Linting and Syntax Checks: 6](#_Toc180415589)

[3. Code Comparisons: 6](#_Toc180415590)

[4. Error Classification: 6](#_Toc180415591)

[5. Severity Analysis: 6](#_Toc180415592)

[6. URL Risk Classification: 6](#_Toc180415593)

[Functionality of Automated Code Review 7](#_Toc180415594)

[1. Import Libraries: 7](#_Toc180415595)

[2. Loading Environment Variables: 7](#_Toc180415596)

[3. Load Custom Modules: 8](#_Toc180415597)

[4. Setup Groq API Client: 8](#_Toc180415598)

[5. Streamlit UI Styling and Layout: 8](#_Toc180415599)

[6. File Uploads and User Input: 9](#_Toc180415600)

[7. Processing files: 10](#_Toc180415601)

[8. Generate Code Review: 11](#_Toc180415602)

[9. Optimization Recommendations: 12](#_Toc180415603)

[10. Error Analysis: 13](#_Toc180415604)

[11. Code Comparison and Highlighting: 14](#_Toc180415605)

[12. Display of Code Changes: 15](#_Toc180415606)

[13. Vulnerability Detection in Code: 16](#_Toc180415607)

[14. Metrics to Calculate Score: 16](#_Toc180415608)

[15. Classify Urls: 18](#_Toc180415609)

[16. Determine Severity: 19](#_Toc180415610)

[Use Cases for Automated Code Review 21](#_Toc180415611)

[FOR ONLY NEW FILE 21](#_Toc180415612)

[1. Upload File for Code Review: 21](#_Toc180415613)

[2. Type of Review: 21](#_Toc180415614)

[3. Code Review: 22](#_Toc180415615)

[4. Error Reporting and Vulnerabilities Analysis: 23](#_Toc180415616)

[5. Optimization Recommendations: 24](#_Toc180415617)

[6. Security Review: 26](#_Toc180415618)

[FOR BOTH FILES (CHANGES) 27](#_Toc180415619)

[1. Upload Files for Code Review: 27](#_Toc180415620)

[2. Type of Review: 28](#_Toc180415621)

[3. Feature Comparison: 28](#_Toc180415622)

[4. Code Review: 29](#_Toc180415623)

[5. Error Reporting and Vulnerabilities Analysis: 30](#_Toc180415624)

[6. Code Style Enforcement: 31](#_Toc180415625)

[7. Optimization Recommendations: 32](#_Toc180415626)

# Automated Code Review Overview

Automated code review streamlines the process of assessing code quality by leveraging advanced technologies and algorithms to analyze code against established standards. This approach enhances efficiency by quickly identifying issues such as syntax errors, security vulnerabilities, and adherence to best practices. By integrating features like code comparison, vulnerability detection, and detailed feedback mechanisms, automated code review tools facilitate a comprehensive examination of both new and existing code. This not only helps developers improve their code quality but also fosters collaboration by providing clear insights and recommendations, ultimately leading to more robust and maintainable software.

A diagram of a diagram

Description automatically generated

### Programming Languages Supported by Automated Code Review Tool:

### For Organization Standards files:

* Word Document
* PDF Document
* PPT Document
* Text Document

### For Code files:

* JavaScript
* Python
* SQL
* HTML
* Java
* CSS
* .Net
* C, C++

# Key Features

### Code Review:

The code review process generates a detailed review based on the code provided and organizational standards. If no changes are specified, it requests a concise summary of the new code's functionality. This explanation is obtained by sending a prompt to an AI model, which analyzes the code and returns a clear description of its purpose and operations. This approach helps developers understand the intent behind the code while maintaining clarity and focusing on key aspects, ultimately enhancing the review process by providing immediate insights into the code’s functionality.

### Linting and Syntax Checks:

Provides robust linting and syntax validation for multiple programming languages, including Python, Java, JavaScript, HTML, CSS, and SQL. This feature helps catch potential syntax errors, enforces coding style guides, and ensures compliance with language-specific standards before code is committed.

### Code Comparisons:

Enables developers to compare different versions of code by visually highlighting changes such as additions, deletions, and modifications. This feature makes it easier to track the evolution of code over time, identify accidental regressions, and understand the nature of updates between versions.

### Error Classification:

Automatically categorizes detected errors based on their type—whether they are syntax errors, logical flaws, runtime issues, or potential security vulnerabilities. This allows developers to prioritize fixes and approach debugging in a structured and efficient way.

### Severity Analysis:

Assesses the severity of identified issues, assigning a level of criticality to each. This feature helps developers focus on addressing high-priority issues that may lead to significant performance bottlenecks, security risks, or functionality breaks.

### URL Risk Classification:

Scans URLs embedded in the code for potential risks, identifying those that might expose the system to vulnerabilities such as phishing, malware, or insecure connections. This feature provides security recommendations to mitigate the risks associated with using untrusted or unsafe URLs.

# Functionality of Automated Code Review

The **Automated Code Review** process begins with file uploads, where users submit new, old, or organizational standard code files for review. The system then processes these files, providing either a detailed or summary review based on user selection. It compares the code, highlights changes, and identifies errors, offering visualized reports for better understanding. Metrics and insights are generated to evaluate code quality, including vulnerability detection and issue severity classification. Finally, it delivers a comprehensive review with suggested improvements and a quality score, ensuring the code aligns with best practices.

### Import Libraries:

This section imports all the required libraries and modules necessary for the automated code review system.

**Details**:

* + *dotenv:* Loads environment variables from a .env file into your application’s environment, such as API keys, credentials, and configuration values.
  + *streamlit:* Used to build an interactive web application. Streamlit simplifies creating a user interface for uploading files, displaying results, and interacting with the NLP model.
  + *langchain & openai:* Integrate NLP capabilities into the code review system, enabling it to generate insights or reviews by analyzing the uploaded code.
  + *pdfplumber, docx, pptx:* Libraries to read text from various file formats (PDF, Word, PowerPoint). These are important to extract the content of documents that may hold organization standards or code files.
  + *SequenceMatcher:* Compares two sets of data (e.g., old and new code) to highlight differences.
  + *subprocess:* Executes shell commands, like running code linters or other external tools, to assess the quality of the code.
  + *re (Regular Expressions):* Used for pattern matching, such as searching for specific coding patterns that might introduce errors or vulnerabilities.

### Loading Environment Variables:

To safely load sensitive information like API keys and configuration values that should not be hard-coded into the application.

**Explanation**:

* + **load\_dotenv()**: This function reads from a .env file, loading any defined variables into the environment, making them accessible to your Python code. For instance, the OpenAI or Groq API keys needed to make authenticated requests to their services.

### Load Custom Modules:

This section imports project-specific modules that provide essential functionality for error handling, prompts, and metric calculations.

**Explanation**:

* + **config**: Centralized settings such as API keys or file paths that are reused across the code.
  + **Errors**: Custom functions for handling various error conditions like file upload failures, unsupported file types, etc. These functions ensure that the system handles exceptions gracefully.
  + **prompt**: Contains the template or structure for prompts sent to the LLM. This could include questions about the code file, security vulnerabilities, or general quality assessments.
  + **metrics**: Functions for calculating statistics such as the number of errors, severity levels, or other code health indicators. These metrics help quantify the output of the code review system.

### Setup Groq API Client:

Initializes the API client for Groq, allowing the application to interact with an NLP model for performing code review tasks.

**Explanation**:

* The Groq API requires an authentication key (GROQ\_API\_KEY) to access its language models. By setting up the client, you prepare the system to send code data for analysis, asking the model to review code, detect issues, and offer suggestions.
* **Interaction with Groq**: After setting up the client, you'll later use this client object to send the uploaded files to the Groq model and retrieve responses (like code insights or reviews).

### Streamlit UI Styling and Layout:

Customize the look and feel of the Streamlit UI, ensuring the application is visually appealing and user-friendly.

**Explanation**:

* + **Custom CSS**: Styling the UI elements of the Streamlit app (e.g., background color, font size, button appearance) to match the aesthetic of your organization or provide a professional feel.
  + **HTML injection**: By using st.markdown(), you inject HTML and CSS code directly into Streamlit to apply more advanced styles. This can be helpful for ensuring consistency and branding across your application.

### File Uploads and User Input:

Provide users with an interface for uploading the necessary files (new code, old code, and organizational code standards). Users also choose the depth of review.

**Explanation**:

* + **st.file\_uploader()**: A widget that allows users to upload files. In this case, the system requires at least a code file and optionally an old code file for comparison. The user can also upload the organization's coding standards in various formats (e.g., .docx, .pdf).
  + **st.selectbox()**: Allows users to select the type of review—either a complete code review or a summary. This input helps tailor the results presented based on the user’s preference.
  + **Input handling**: The uploaded files are processed and passed to relevant functions for further analysis. If files are missing or incompatible, appropriate error messages are displayed.



### Processing files:

The uploaded file content depending on the type of file (plain text, code, .docx, or .pdf) to extract the relevant text or code for analysis.

**Explanation**:

* + **File identification**: The function first checks the type of file uploaded. For .docx files, it reads the text content from paragraphs; for plain text or code files, it decodes the file content to a string.
  + **PDF processing**: If the file is a PDF, pdfplumber is used to extract the text, ensuring compatibility with a variety of input formats.
  + **File type handling**: The processed content is returned and stored in a variable for use in further steps, like code analysis or passing it to the NLP model.

Processing Files


### Generate Code Review:

Generates a prompt for the Groq API based on the provided new code and organizational standards.

* **Explanation:**
  + **Input**:
    - new\_code: The new code provided by the user.
    - org\_std\_text: The organizational code standards.
  + **How it works**:
    - This function constructs a structured prompt using the new code and organizational standards. The prompt instructs the model to evaluate the code according to the standards.
    - This structured prompt is passed to the Groq model via the API call.
  + **Return**:
    - It returns the generated prompt that will be used in the subsequent API call.
  + **Key Components**:
    - **Prompt Construction**: The prompt is constructed dynamically using the input parameters.
    - **Clear Instructions**: The prompt is structured to be clear and direct to guide the model in generating the right response.

A screenshot of a white and green page

Description automatically generated

### Optimization Recommendations:

A screenshot of a computer program

Description automatically generated

### Error Analysis:

A screenshot of a computer error

Description automatically generated

### Code Comparison and Highlighting:

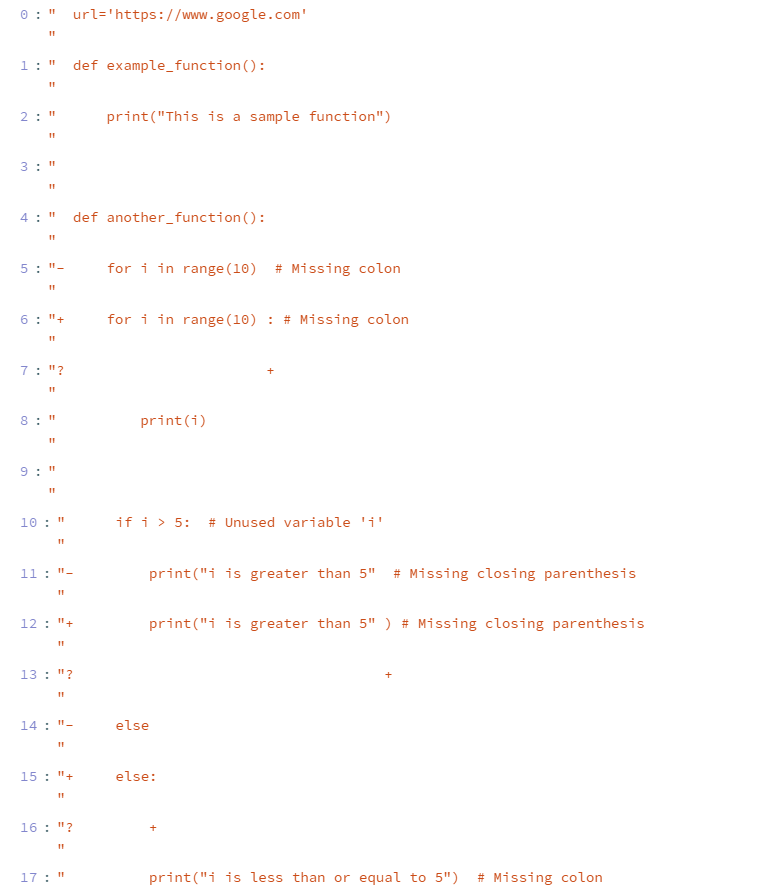
This function compares old and new code files to identify and highlight the differences. Useful for showing the developer what has been added, modified, or removed.

**Explanation**:

* + **highlight\_diff()**: Highlights the differences between the old and new versions of code, line by line, using SequenceMatcher. Differences can be categorized into "Added", "Removed", or "Modified".
  + **Comparison logic**: When both old and new code files are provided, this function analyzes the two files and returns a structured view of the differences.
  + **Change tracking**: By pinpointing the exact code changes, this comparison helps developers see where they made modifications, making it easier to review the code.

**Line-by-Line Comparison**:

* + The old and new code files are split into lines.
  + The difflib.Differ() is used to create a diff object which compares the old and new lines of code.
  + Lines that are either added (+) or removed (-) are extracted and returned.

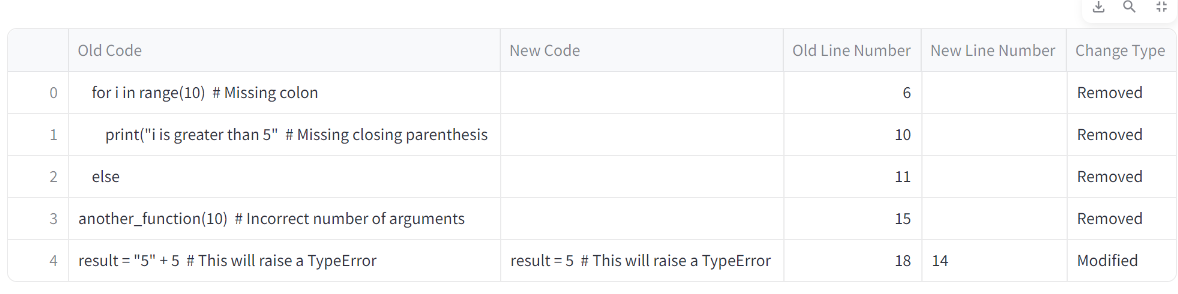


### Display of Code Changes:

Present the differences between the old and new code in a clear, tabular format. This is particularly useful for developers to quickly spot what has changed.

**Explanation**:

* + **Tabular format**: Differences are shown in a structured table, highlighting which lines of code have been added, removed, or modified.
  + **DataFrame presentation**: Streamlit supports displaying pandas DataFrames, allowing for an organized and easily readable presentation of the changes.
  + **Clear visualization**: This structure provides developers with an intuitive way to understand how their code has changed over time.

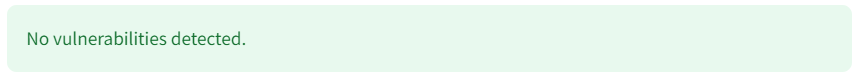


### Vulnerability Detection in Code:

Scan the uploaded code for common security vulnerabilities or bad practices that could compromise security or code quality.

**Explanation**:

* + **Code scanning**: Looks for dangerous patterns like the use of eval(), exec(), or other functions that could introduce security risks.
  + **Static analysis**: By running static analysis, the system identifies problematic code that could lead to vulnerabilities such as SQL injections or cross-site scripting (XSS).
  + **Output**: The results of this scan are returned to the user, giving them actionable feedback on how to secure their code.



### Metrics to Calculate Score:

This function calculates a score for the newly provided code based on how closely it adheres to the organizational code standards.

**Explanation:**

* **Input**: It takes two inputs:
  + org\_std\_text: The organizational standards against which the code should be evaluated.
  + new\_code: The newly provided code that will be scored.
* **How it works**:
  + First, the function constructs a structured prompt for the Groq API using both the new code and the organizational standards.
  + The prompt instructs the Groq model to evaluate the code based on the provided standards and return a score out of 10.
  + The prompt is sent to the Groq API via client.chat.completions.create(), which returns a response containing the score.
* **Return**:
  + The function extracts the score from the response and returns it to the caller.
* **Key Components**:
  + **Groq API Call**: This function uses Groq’s chat.completions.create() method to communicate with the model and get a response.
  + **Message Structuring**: The prompt is carefully structured to instruct the model on how to evaluate the code and generate a score.
  + **Response Parsing:** The function assumes that the response from the API is a score, and it parses the score using regular expressions (re.search()).
  + **Score Display:** Based on the score, the function assigns a color and a message:
    - **Score ≥ 9**: Green color, "Excellent."
    - **Score ≥ 7**: Yellow color, "Good."
    - **Score ≥ 4**: Orange color, "Average."
    - **Score < 4**: Red color, "Poor."
  + These values are displayed in Streamlit using colored HTML, as Streamlit's markdown function allows embedding HTML with unsafe\_allow\_html=True.
  + **Return Value:** The function returns the explanation from the API (which could contain more detailed feedback about the score).

A screenshot of a computer code

Description automatically generated

### Classify Urls:

This function identifies and classifies URLs present in the provided code, categorizing them by security risk levels.

**Explanation:**

* **Input**:
  + code\_text: The complete text of the code which may contain URLs.
* **How it works**:
  + The function uses the re.findall() method to search the code for URLs using a regex pattern.
  + The regex pattern used is r'(https?://[^\s]+|www\.[^\s]+)', which finds URLs that start with http or https, or those starting with www.
  + After finding the URLs, the function categorizes them into four types of risk:
    1. **Low Risk - Secure (HTTPS)**: URLs starting with https.
    2. **Medium Risk - Insecure (HTTP)**: URLs starting with http.
    3. **Low Risk - Localhost**: URLs that reference local domains (localhost or 127.0.0.1).
    4. **High Risk - Unknown/Suspicious Domain**: URLs that don't fall into the previous categories.
* **Return**:
  + It returns a dictionary where the keys are URLs and the values are their classifications.
* **Key Components**:
  + **Regex**: The use of re.findall() and regular expressions helps efficiently extract and classify URLs.
  + **Security Classification**: The URLs are categorized based on security considerations like HTTPS and local domains.

A screenshot of a computer

Description automatically generated

### Determine Severity:

This function calculates the overall severity of the code based on the number of errors of different types (syntax, logical, etc.).

**Explanation:**

* **Input**:
  + syntax\_errors: A list of syntax errors detected in the code.
  + logical\_errors: A list of logical errors.
  + runtime\_errors: A list of runtime errors.
  + compilation\_errors: A list of compilation errors.
  + validation\_errors: A list of validation errors.

**Error Severity Calculation:**

* **Error Weights:**

error\_weights = {

"Syntax Errors": 3,

"Run-time Errors": 4,

"Logical Errors": 2,

"Validation Errors": 1

}

Each error type is assigned a weight, indicating how severe that type of error is. For example, run-time errors are given a higher weight than syntax errors.

* **determine\_severity\_from\_score() Function:**

This function takes the total severity score and assigns a color and severity message based on the score:

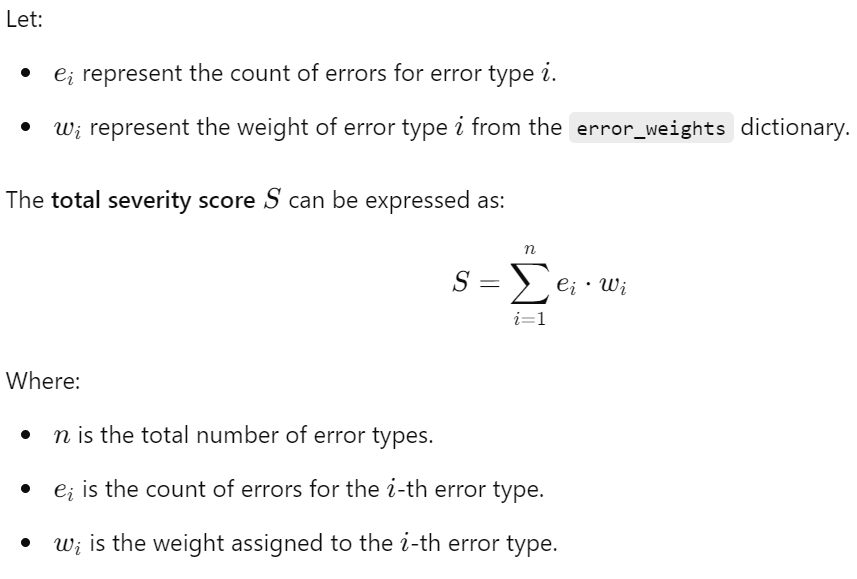
* + **Score = 0**: Dark green, "No errors, perfect!"
  + **Score ≤ 10**: Light green, "Low severity."
  + **Score 11-20**: Yellow, "Medium severity."
  + **Score 21-30**: Orange, "High severity."
  + **Score > 30**: Red, "Critical severity."

This helps display the error severity in a user-friendly and color-coded manner.

* **calculate\_severity() Function:**

This function calculates the total severity score based on a dictionary of error counts. The error counts represent how many errors of each type were found, and the function multiplies the count by the corresponding weight to compute the total score.

**Total Severity Score Calculation**



A screenshot of a computer

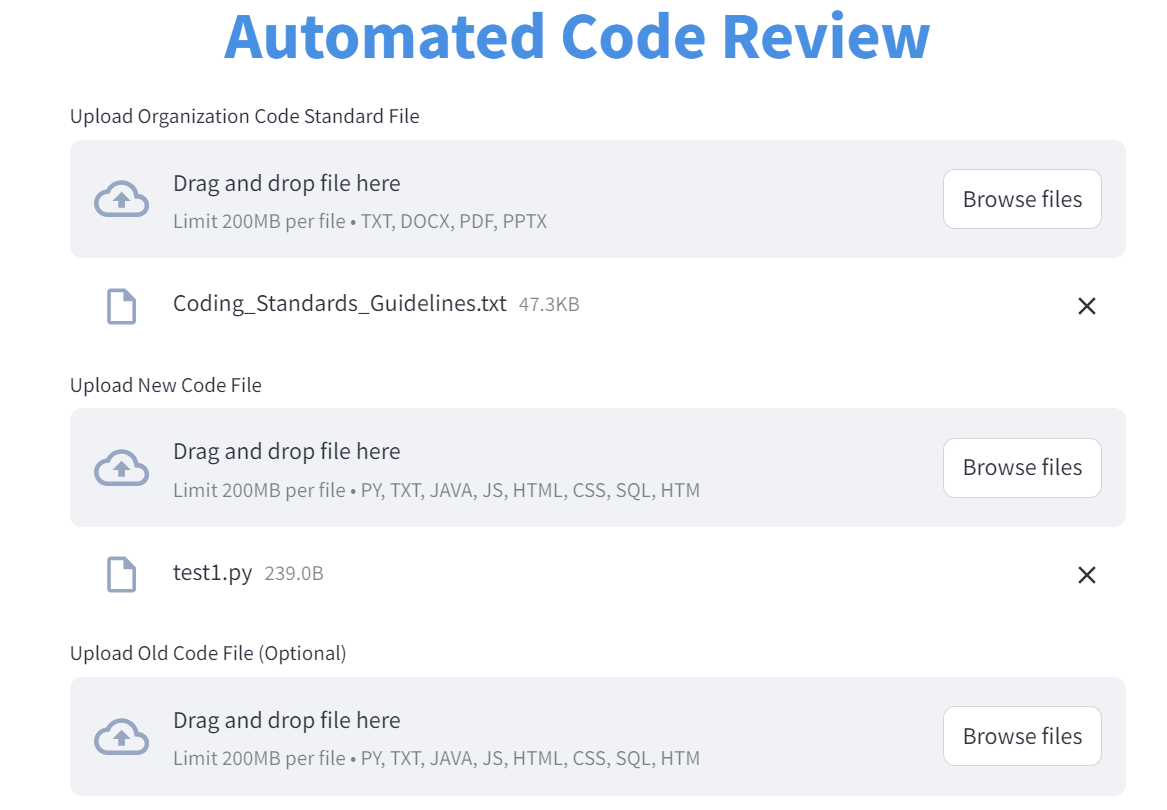
Description automatically generated

# Use Cases for Automated Code Review

### FOR ONLY NEW FILE

### Upload File for Code Review:

The Upload Files for Code Review feature allows users to upload their organization's code standard file and a new code file for review. Additionally, users can upload an optional old code file for side-by-side comparison of changes. The system supports various file types, including `.txt`, `.docx`, `.pdf`, `.pptx` and multiple programming languages, ensuring flexibility in reviewing different code formats. This feature initiates the automated review process, leading to analysis, error detection, and suggestions for improvements.



### Type of Review:

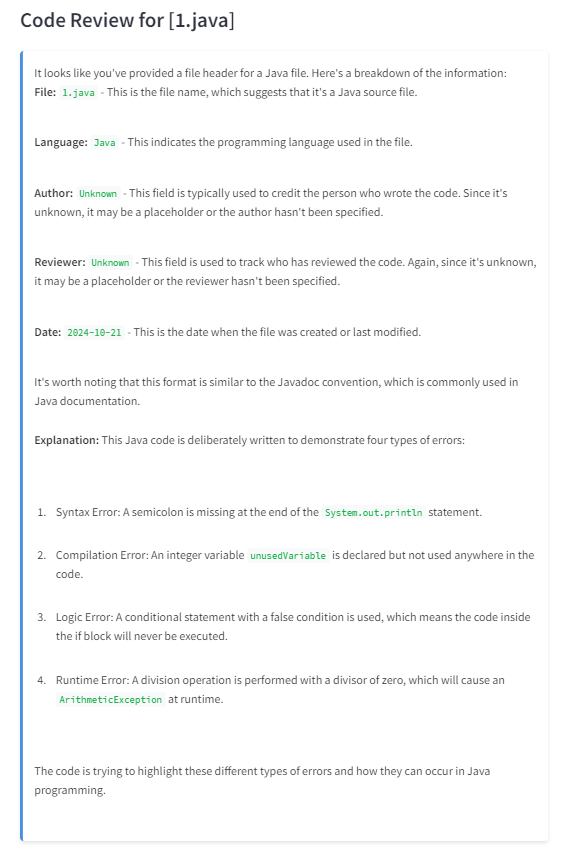
The automated code review process offers two review types: Complete Code Review and Summary Review. The Complete Code Review provides a thorough examination of the entire codebase, offering detailed feedback on coding standards, performance optimization, and security vulnerabilities, making it ideal for in-depth analysis. In contrast, the Summary Review delivers a concise assessment, highlighting key insights and major issues without examining every line of code, suitable for quick evaluations. This dual approach allows users to choose the level of scrutiny that best fits their project needs.

A screenshot of a computer

Description automatically generated

### Code Review:

The code review process generates a detailed review based on the provided code and organizational standards. If no changes are specified, it requests a concise summary of the new code's functionality. This explanation is obtained by sending a prompt to an AI model, which analyzes the code and returns a clear description of its purpose and operations. This approach helps developers understand the intent behind the code while maintaining clarity and focusing on key aspects, ultimately enhancing the review process by providing immediate insights into the code’s functionality.



### Error Reporting and Vulnerabilities Analysis:

The error reporting and vulnerability analysis process systematically evaluates code for various types of errors, including syntax, run-time, logical, and validation errors. It utilizes specific prompts to request detailed feedback from an AI model, highlighting problematic lines in the code. Additionally, the process includes a vulnerability detection mechanism that scans for potential security risks such as code injection, SQL injection, Cross-Site Scripting (XSS), and hardcoded credentials. The findings are presented in a structured format, allowing developers to easily identify and address errors and vulnerabilities, ensuring the code is both functional and secure.

A screenshot of a computer error

Description automatically generated

### Optimization Recommendations:

The generated review provides a comprehensive analysis of the submitted code, highlighting areas for improvement based on predefined organizational standards. This structured approach ensures that developers receive targeted guidance to elevate their coding standards and foster a culture of continuous improvement within their projects.

A close-up of a computer screen

Description automatically generated  
A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Security Review:

In a security review, severity levels are determined based on the classification of errors and vulnerabilities, such as syntax, logical, runtime, compilation, and validation errors. These levels, ranging from **"No errors, perfect!"** for error-free code to **"Critical severity"** for code with significant issues, help developers prioritize remediation efforts effectively. Additionally, the classification of external URLs is crucial for assessing security risks. URLs starting with "https" are considered "Low Risk - Secure," while "http" URLs are labeled "Medium Risk - Insecure." Local URLs like "localhost" or "127.0.0.1" are "Low Risk - Localhost," and unknown or suspicious domains are categorized as "High Risk," requiring scrutiny to ensure code safety.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### FOR BOTH FILES (CHANGES)

### Upload Files for Code Review:

The Upload Files for Code Review feature allows users to upload their organization's code standard file and a new code file for review. Additionally, users can upload an optional old code file for side-by-side comparison of changes. The system supports various file types, including `.txt`, `.docx`, `.pdf`, `.pptx` and multiple programming languages, ensuring flexibility in reviewing different code formats. This feature initiates the automated review process, leading to analysis, error detection, and suggestions for improvements.

A screenshot of a computer

Description automatically generated

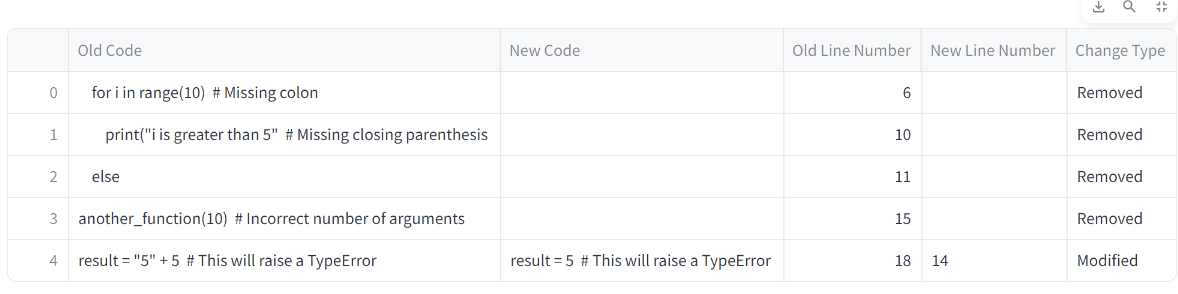
### Type of Review:

The automated code review process offers two review types: Complete Code Review and Summary Review. The Complete Code Review provides a thorough examination of the entire codebase, offering detailed feedback on coding standards, performance optimization, and security vulnerabilities, making it ideal for in-depth analysis. In contrast, the Summary Review delivers a concise assessment, highlighting key insights and major issues without examining every line of code, suitable for quick evaluations. This dual approach allows users to choose the level of scrutiny that best fits their project needs.

A screenshot of a computer

Description automatically generated

### Feature Comparison:

The code comparison process identifies differences between two versions of code by analyzing them line by line. It utilizes a library to detect changes, categorizing them as modified, removed, added, or unchanged. Each identified change is stored in a structured format, which includes the old and new code lines along with their change types. This information is then organized into a DataFrame, allowing for a clear, tabular representation of the differences. This process aids developers in understanding how the code has evolved, highlighting significant modifications and ensuring that changes align with coding standards.

### A screenshot of a computer program Description automatically generatedCode Review:

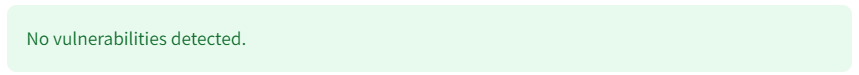
The code review process generates a detailed review based on the provided code and organizational standards. If no changes are specified, it requests a concise summary of the new code's functionality. This explanation is obtained by sending a prompt to an AI model, which analyzes the code and returns a clear description of its purpose and operations. This approach helps developers understand the intent behind the code while maintaining clarity and focusing on key aspects, ultimately enhancing the review process by providing immediate insights into the code’s functionality.

### Error Reporting and Vulnerabilities Analysis:

The error reporting and vulnerability analysis process systematically evaluates code for various types of errors, including syntax, run-time, logical, and validation errors. It utilizes specific prompts to request detailed feedback from an AI model, highlighting problematic lines in the code. Additionally, the process includes a vulnerability detection mechanism that scans for potential security risks such as code injection, SQL injection, Cross-Site Scripting (XSS), and hardcoded credentials. The findings are presented in a structured format, allowing developers to easily identify and address errors and vulnerabilities, ensuring the code is both functional and secure.

A screenshot of a computer error

Description automatically generated



### Code Style Enforcement:

In a security review, severity levels are determined based on the classification of errors and vulnerabilities, such as syntax, logical, runtime, compilation, and validation errors. These levels, ranging from "Excellent" for error-free code to "Poor" for code with significant issues, help developers prioritize remediation efforts. Additionally, the classification of external URLs is crucial for assessing security risks. URLs starting with "https" are considered "Low Risk - Secure," while "http" URLs are labeled "Medium Risk - Insecure." Local URLs like "localhost" or "127.0.0.1" are "Low Risk - Localhost," and unknown or suspicious domains are categorized as "High Risk," requiring scrutiny to ensure code safety.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Optimization Recommendations:

The generated review provides a comprehensive analysis of the submitted code, highlighting areas for improvement based on predefined organizational standards. This review not only identifies errors—such as syntax, runtime, logical, and validation issues—but also offers actionable insights for enhancing code quality and adherence to best practices. By synthesizing feedback from various sources, the review outlines specific recommendations, such as refactoring code, optimizing performance, and improving security measures. This structured approach ensures that developers receive targeted guidance to elevate their coding standards and foster a culture of continuous improvement within their projects.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated