Author: Madhurima Rawat

Code Review Automation with CodeFactor and Codacy

Improve the project's code quality by integrating CodeFactor or Codacy with GitHub. These tools automatically analyze the code changes and provide feedback on style, complexity, and potential issues.

Steps to Set Up Code Review Automation

1. Prerequisites

- A GitHub repository.
- Admin access to the repository.
- A CodeFactor or Codacy account (create one for free).

2. Set Up CodeFactor Integration

1. **Log In to CodeFactor**
   - Visit CodeFactor and log in using your GitHub credentials.
2. **Link Your Repository**
   - Click on **Add Project** and authorize CodeFactor to access your GitHub repositories.
   - Select the repository you want to integrate.
3. **Configure Settings**
   - Adjust analysis settings for your repository, such as ignoring specific files or directories.
4. **Enable GitHub Integration**
   - Navigate to your repository settings on CodeFactor and enable **GitHub Status Updates**.
   - This ensures that analysis results are displayed on your pull requests.
5. **Commit and Push Code**
   - CodeFactor will now analyze every new commit and provide feedback directly in the pull request.

3. Set Up Codacy Integration

1. **Log In to Codacy**
   - Visit Codacy and sign in using GitHub.
2. **Add Your Repository**
   - Click **Add a Project** and choose the repository from GitHub.
   - Codacy will scan your codebase to establish a baseline report.
3. **Set Up GitHub Status Checks**
   - Navigate to **Settings** > **Integrations** in Codacy and enable **GitHub** as a status provider.
4. **Customize Analysis**
   - Go to **Settings** > **Repository** > **Patterns** to tailor Codacy's checks to your project's needs.
5. **Review Pull Requests**
   - Codacy will analyze every new pull request, providing inline comments for issues and a detailed status summary.

4. Sample Code for Testing

Here are two Python scripts for testing in both **CodeFactor** and **Codacy**:

Code with Issues for Codacy

```
# sample_code_with_errors.py

def calculateSum(Num1, Num2):
    return Num1 + Num2


def divideNumbers(num1, num2):
    result = num1 / num2
    return result


def main():
    print("Sum: ", calculateSum(5, 10))
    print("Division Result:", divideNumbers(10, 0))


main()
```

Issues:

1. **Naming Convention**: Function names like `calculateSum` and `divideNumbers` violate Python's `snake_case` convention (should be `calculate_sum` and `divide_numbers`).
2. **PEP-8 Violations**: Missing spaces around arguments and operators, e.g., `calculateSum(5,10)` (should be `calculate_sum(5, 10)`).
3. **ZeroDivisionError**: The `divideNumbers` function does not handle division by zero, causing a runtime error.
4. **Missing Type Annotations**: Functions lack type hints, such as `def calculate_sum(num1: int, num2: int) -> int:`.

Code with Issues for CodeFactor

```
# sample_code_with_errors_codefactor.py

# Function with naming violations and redundancy
def CalculateSum(Num1, Num2):
    # Redundant logic
    result = Num1 + Num2
    return result


# Duplicate function logic
def AddTwoNumbers(Num1, Num2):
    # Redundant logic
    result = Num1 + Num2
    return result


# Function without type hints and missing error handling
def DivideNumbers(Num1, Num2):
    if Num2 == 0:  # This is flagged as overly simplistic error handling.
        return "Cannot divide by zero"
    return Num1 / Num2


# Function with inconsistent spacing and missing type hints
def MultiplyNumbers(Num1, Num2):
    Result = Num1 * Num2  # Inconsistent naming
    return Result


# Main function with improper formatting and repetition
def Main():
    print("Sum (Method 1):", CalculateSum(5, 10))
    print("Sum (Method 2):", AddTwoNumbers(5, 10))  # Duplication of logic
    print("Division Result:", DivideNumbers(10, 0))
```

```
    print("Multiplication Result:", MultiplyNumbers(5, 2))


Main()
```

Issues:

1. **Naming Convention**: Function names like `CalculateSum` and `Main` do not follow Python's `snake_case` standard.
2. **Redundant Code**: `CalculateSum` and `AddTwoNumbers` have identical functionality, leading to duplication.
3. **Error Handling**: `DivideNumbers` has overly simplistic error handling for division by zero; proper exception handling should be added.
4. **Inconsistent Naming and Formatting**: Variable names like `Result` and inconsistent spacing around operators violate PEP-8 guidelines.

Corrected Code

```
# sample_code_corrected.py

def calculate_sum(num1: int, num2: int) -> int:
    """Calculate the sum of two integers."""
    return num1 + num2


def divide_numbers(dividend: int, divisor: int) -> float:
    """
    Safely divide two numbers. Returns a float result.
    Handles division by zero with a custom message.
    """
    try:
        return dividend / divisor
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")
        return float("inf")  # Return infinity to indicate error.


def main() -> None:
    """Main function to demonstrate addition and division operations."""
    num1, num2 = 5, 10
    print("Sum:", calculate_sum(num1, num2))

    dividend, divisor = 10, 0
    print("Division Result:", divide_numbers(dividend, divisor))
```

```
if __name__ == "__main__":
    main()
```

Fixes:

- Function name updated to follow `snake_case`.
- Extra spaces removed for PEP-8 compliance.
- Type annotations added for better code clarity.

5. Analysis and Evaluation

Benefits of Integration

- **Automatic Code Reviews**: Save time by automating style and complexity checks.
- **Feedback in Pull Requests**: View issues directly in GitHub pull requests.
- **Customizable Rules**: Focus on the metrics that matter most for your project.
- **Improved Code Quality**: Maintain consistent standards across your team.

Example Workflow

1. Developer submits a pull request.
2. CodeFactor/Codacy analyzes the code changes.
3. Issues and suggestions are displayed directly in the pull request.
4. Developer resolves the issues before merging.

Troubleshooting

- **Analysis Not Triggering**: Ensure the repository is public or that the service has access to your private repository.

We can commit these files to our GitHub repository, and both **CodeFactor** and **Codacy** will analyze and report on the issues in the first file while approving the second file.