Bonus Task
Automated Design Smell Detection and Refactoring Pipeline Report

Team 11

1 Introduction

This report documents the design and implementation of an automated pipeline for detecting design smells in a GitHub repository, refactoring the identified issues, and generating pull requests for the changes.

2 Pipeline Overview

The pipeline consists of three main components:

1. Automated Design Smell Detection
2. Automated Refactoring
3. Pull Request Generation

3 Code Snippets Explanation

3.1 Refactoring Code with ChatGPT

This function (`refactor_code_with_chatgpt`) takes the path to a Java file in a GitHub repository and refactors the code using the OpenAI ChatGPT model. The refactored code is then written to a new file in the repository.

3.2 Creating Branch

Function (`create_branch`) creates a new branch in the local repository to push the refactored code changes.

3.3 Main Function

The main function (`main`) orchestrates the entire pipeline by cloning the repository, refactoring the code, committing and pushing the changes to a new branch, and creating a pull request.

4 How to Run the Code

To run the code, follow these steps:

1. Set up the OpenAI API key, GitHub personal access token, and repository information in the code.

2. Ensure you have the necessary Python packages installed (`gitpython`, `requests`).

3. Change the `local_repo_path` in the `main` function to your local path.

4. Manually select the Java file path for which the refactoring needs to be done.

5. Execute the Python script.

6. Note: Some paths are hardcoded due to limitations in using the OpenAI ChatGPT API for the entire repository.

5 Expected Way to Run

1. The code first clones the repository into your local directory specified in the `main` function.

2. Then it selects the file that needs to be refactored pushes to the Open AI API, takes the refactored code that is given by ChatGPT.

3. Now this refactored code is placed in the same folder.

4. A pull request with the file in the branch refactor-llms will be created.

6 Hardcoded Values and Configuration

- **API Key:** The OpenAI API key is hardcoded in the `API_KEY` variable.
- **GitHub Token and Username:** The GitHub personal access token and username are hardcoded in the `GITHUB_TOKEN` and `GITHUB_USERNAME` variables.
- **Repository Information:** The repository owner, repository name, and branch name are hardcoded in the `REPO_OWNER`, `REPO_NAME`, and `NEW_BRANCH` variables `local_repo_path` and `file_path` are also hardcoded.

7 Conclusion

The automated pipeline successfully detects design smells, refactors the code, and generates pull requests for the changes. Further enhancements can be made to improve the robustness and efficiency of the pipeline.

8 Code Snippet

```python
import os
import subprocess
import json
import requests
from git import Repo, GitCommandError
import openai

# Set up OpenAI API
API_KEY = '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'
# Your GitHub Personal Access Token
GITHUB_TOKEN = '[redacted]'
# Your GitHub Username
```

```
GITHUB_USERNAME = 'bhaskarahanuma'
# Repository owner and name
REPO_OWNER = 'serc-courses'
REPO_NAME = 'se-project-1--_11'
# Branch where the refactored code will be pushed
NEW_BRANCH = 'refactor-branch-llms'


def refactor_code_with_chatgpt(repo_path, file_path):
    with open(os.path.join(repo_path, file_path), "r") as file:
         code = file.read()
    prompt = f"Review the following Java code and refactor the code in all ways possible and give the refactored code:\n\n{code}\n\n"

    openai.api_key = API_KEY
    response = openai.ChatCompletion.create(model="gpt-3.5-turbo",messages=[{"role": "system", "content": prompt}],max_tokens=800,
    temperature=0.7, top_p=1)

    refactored_code = response.choices[0].message['content'].strip()
    directory, filename = os.path.split(file_path)
    new_filename = f"refactored_{filename}"

    new_file_path = os.path.join(repo_path, directory, new_filename)

    with open(new_file_path, "w") as file:
         file.write(refactored_code)

    print("Code Refactoring Completed.")
    print(f"Refactored code written to: {new_file_path}")
    #print(refactored_code)
    # Return the refactored code
    return refactored_code

```

```python

def create_branch(repo_path, branch_name):
    os.chdir(repo_path)
    repo = Repo(repo_path)
    #print(repo)
    if branch_name in repo.branches:
        print(f"Branch {branch_name} already exists.")
    else:
        try:
            repo.git.checkout('-b', branch_name)
            print(f"Created branch {branch_name}")
        except GitCommandError as e:
            print(f"An error occurred while creating
               branch {branch_name}: {e}")

def main():
    local_repo_path = '/home/hanuma/Desktop/repo'

    if os.path.isdir(local_repo_path):
        print(f"Directory '{local_repo_path}' already
           exists.")
    else:
        repo_url = f"https://{GITHUB_USERNAME}:{
           GITHUB_TOKEN}@github.com/{REPO_OWNER}/{
           REPO_NAME}.git"
        subprocess.run(['git', 'clone', repo_url,
           local_repo_path])

    os.chdir(local_repo_path)
    file_path = 'books-web/src/main/java/com/sismics/
       books/rest/resource/AppResource.java'
    refactored_code = refactor_code_with_chatgpt(
       local_repo_path, file_path)

    subprocess.run(['git', 'add', '.'])

    # commit the changes
    subprocess.run(['git', 'commit', '-m', 'Refactor
       code'])

    # create the branch if it doesn't exist
    create_branch(local_repo_path, NEW_BRANCH)

    try:
        subprocess.run(['git', 'pull', 'origin', 'main'
           ])
        print("Latest changes pulled from the base
           branch.")
    except GitCommandError as e:
        print(f"An error occurred while pulling changes:
            {e}")

    try:
        subprocess.run(['git', 'push', 'origin',
           NEW_BRANCH])
        print("Changes committed and pushed to the
           remote repository.")
    except GitCommandError as e:
        print(f"An error occurred while pushing changes:
            {e}")

    # Headers for the API request
    headers = {
        'Authorization': f'token {GITHUB_TOKEN}',
        'Accept': 'application/vnd.github.v3+json',
    }

    data = {
        'title': 'Refactor code changes',
        'head': NEW_BRANCH,
        'base': 'master',  # Ensure that 'main' matches
           the name of the existing base branch
        'body': 'This PR includes the following changes
           :\n- Detected design smells\n- Applied
           refactoring techniques\n- Metrics\n\nPlease
           review the changes.',
    }


```

```python
    # Before creating the pull request, check if it already exists
    response = requests.get(
        f'https://api.github.com/repos/{REPO_OWNER}/{REPO_NAME}/pulls',
        headers=headers,
        params={'state': 'open', 'head': f'{GITHUB_USERNAME}:{NEW_BRANCH}'}
    )

    if response.status_code == 200:
        pr_list = response.json()
        if not pr_list:
            # No open PR found, create a new one
            response = requests.post(
                f'https://api.github.com/repos/{REPO_OWNER}/{REPO_NAME}/pulls',
                headers=headers,
                data=json.dumps(data)
            )
            if response.status_code == 201:
                print('Pull request created successfully')
            else:
                print(f'Failed to create pull request: {response.text}')
        else:
            print('A pull request for this branch already exists.')
    else:
        print(f'Failed to check for existing pull requests: {response.text}')

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

Listing 1: Python script for the pipeline