Software-Engineering : TEAM-18

Bonus Component

Automated Refactoring:

Detected Design Smells:

```
Outputs from the script are shown below
```

Primitive Obsession

Description: The code uses primitive types (e.g., String, Timestamp) directly instead of creating domain-specific classes

Suggested Refactoring: Create domain-specific classes (e.g., UserDto) for the primitive types used in the code

Shotgun Surgery

Description: Changes to the domain logic require modifications in multiple methods

Suggested Refactoring: Implement a more cohesive design pattern such as Domain-Driven Design to reduce the need for changes in multiple methods when modifying domain logic

Data Clumps

Description: The code passes multiple related data as parameters to methods (e.g., PaginatedList, SortCriteria)

Suggested Refactoring: Encapsulate related data into a single object (e.g., create a class for PaginatedList with user-related attributes) and pass that object as a parameter to methods

Script for Automated Refactoring

```
import os

from dotenv import dotenv_values, load_dotenv
from openai import OpenAI
import json
import random
import re
import subprocess
import requests

load_dotenv()
config = dotenv_values(".env")
```

```
client = OpenAI(api_key=os.environ.get(config['OPENAI_API_KEY']))

github_token =
'github_pat_11AXNX2JY04f8TpuHxtB9w_wR4qyixKNkQ9ELqhVANRZAsHZXlG2H43iupiHeZA
iKgYHNN22FKAxxarNgH'
file_path = 'books-
core/src/main/java/com/sismics/books/core/dao/jpa/UserDao.java'
repo_url = 'https://github.com/serc-courses/se-project-1--_18'
repo_owner = 'serc-courses'
repo_name = 'se-project-1--_18'

def create_git_branch(branch_name):
    try:
        # Run the Git command to create a new branch
        subprocess.run(["git", "branch", branch_name])
        print(f"Branch '{branch_name}' created successfully.")
        subprocess.run(["git", "checkout", branch_name])

    except Exception as e:
        print(f"Error occurred while creating branch '{branch_name}': {e}")

def push_git_branch(branch_name):
    try:
        subprocess.run(["git", "push", "--set-upstream", "origin",
branch_name])
        print(f"Branch '{branch_name}' pushed to remote origin
successfully.")
    except Exception as e:
        print(f"Error occurred while creating branch '{branch_name}': {e}")

def commit_files(file_paths, commit_message):
    try:
        print('wtfffff...................', file_paths)
        # Add the specified files to the staging area
        subprocess.run(["git", "add"] + file_paths)

        subprocess.run(["git", "commit", "-m", commit_message])

        print("Files committed successfully.")
    except Exception as e:
        print(f"Error occurred while committing files: {e}")

def retrieve_code_blocks(input_string):
    pattern = r"```java(.*?)```"
    match = re.search(pattern, input_string, re.DOTALL)
    if match:
        return match.group(1)  # Return the text captured by the first
group
    else:
        return None

def retrieve_file_name(input_string):
    pattern = r"\[(.*?)\]"
    match = re.search(pattern, input_string)
```

```
    if match:
        return match.group(1).strip()  # Return the text captured by the
first group
    else:
        return None  # Return None if no match is found

def detect_design_smells(file_content):
    messages = [
        {
            "role": "user",
            "content": f"Identify design smells (Dont give trivial smells)
in the following code:\n\n{file_content}\n\nGive in the format <id>;<design
smell>;<description>;<suggested refactoring>;\\n",
        }
    ]
    response = client.chat.completions.create(
        messages=messages,
        model="gpt-3.5-turbo",
    )
    print(response.choices[0].message.content)
    detected_smells = [ x.split(';') for x in
response.choices[0].message.content.strip().split('\n')]
    detected_smells = [{'smell name': x[1], 'smell description': x[2],
'suggested refactoring': x[3]} for x in detected_smells]
    return detected_smells

def refactor_code(file_content, design_smells):

    selected_design_smell = random.choice(range(design_smells.__len__()))
    selected_design_smell = design_smells[selected_design_smell]

    prompt = f"Refactor the below code with respect to the specified design
pattern (dont give trivial refactoring)\nOriginal
Code:\n{file_content}\n\nIdentified Design
Smell:\n\n{selected_design_smell}\n\nGive Output in the format for all the
new files or existing file:\n[<file name>]\n```java\n<code that goes into
the file>\n```|\n[<file name>]\n```java\n<code that goes into the
file>\n```|...etc..."

    messages = [
        {
            "role" : "user",
            "content": prompt
        }
    ]

    response = client.chat.completions.create(
        messages=messages,
        model="gpt-3.5-turbo",
    )
    print('-------------------------------------------')
    print('Refactoring')
    print('-------------------------------------------')
    print(response.choices[0].message.content)
```

```
    refactored_code = response.choices[0].message.content
    refactored_code = refactored_code.strip().split('|')
    print(refactored_code)
    refactored_code = [{"file_name": retrieve_file_name(x), 'file_content': 
retrieve_code_blocks(x)} for x in refactored_code]
    for i in refactored_code:
        if(i['file_name'] == None or i['file_content'] == None):
            continue

        base_path = os.path.dirname(file_path)
        new_file_path = os.path.join(base_path, i['file_name'])
        print(new_file_path)
        with open(new_file_path, 'w+') as opened_file:
            opened_file.write(i['file_content'])

    return refactored_code, selected_design_smell

# Step 3: Pull Request Generation
def create_pull_request(refactored_code, repo_owner, repo_name, 
refactored_smell):
    branch_name = refactored_smell["smell name"].replace(' ', '-') + "-" + 
str(random.choice(range(1000)))
    create_git_branch(branch_name)
    files = []
    for i in refactored_code:
        if i["file_name"] != None:
            files.append(os.path.join(os.path.dirname(file_path), 
i["file_name"]))

    commit_files(files, refactored_smell["smell name"] + " Solved")
    push_git_branch(branch_name)

    # Implement GitHub API calls to create a pull request
    url = f'https://api.github.com/repos/{repo_owner}/{repo_name}/pulls'
    headers = {
        'Accept': 'application/vnd.github+json',
        'Authorization': f'Bearer {github_token}',
        'X-GitHub-Api-Version': '2022-11-28'
    }

    # Define the pull request details
    pull_request_data = {
        'title': f'Automated Refactoring: {refactored_smell["smell 
name"]}',
        'body': f'# Smell description\n{refactored_smell["smell 
description"]}\n\n# Refactoring: \n{refactored_smell["suggested 
refactoring"]}',
        'head': branch_name,
        'base': 'master',
    }

    print(url, headers, pull_request_data)
    # Create the pull request
    response = requests.post(url, headers=headers, json=pull_request_data)
    if response.status_code == 201:
        print('Pull request created successfully!')
    else:
        print(f'Error creating pull request. Status code:
{response.status_code}, Message: {response.text}')

# Read the original code from the file

with open(file_path, 'r') as file:
    original_code = file.read()

detected_smells = detect_design_smells(original_code)
json_data = json.dumps(detected_smells, indent=4)

with open('design_smells.json', 'w+') as f:
    f.write(json_data)

print(detected_smells)

refactored_code, refactored_smell = refactor_code(original_code,
detected_smells)


# Step 1: Detect design smells
#print(detected_smells)
# # Step 2: Refactor code
# refactored_code = refactor_code(original_code)

# refactored_code = [
#     {
#         "file_name": "TagDao.java"
#     }
# ]
# refactored_smell = {
#     "smell name": "Magic String Smell",
#     "smell description": "String literals are used directly in queries,
making the code less maintainable",
#     "suggested refactoring": "Use constants or enums to define the query
strings and parameter names."
# }
create_pull_request(refactored_code, repo_owner, repo_name,
refactored_smell)
# # Step 3: Create pull request
# create_pull_request(repo_owner, repo_name, file_path, original_code,
refactored_code, detected_smells)
```

This Python script demonstrates a workflow for automated refactoring of code. Here's a breakdown of how it works:

1. **Importing Libraries**: The script imports necessary libraries like `os`, `dotenv`, `OpenAI`, `json`, `random`, `re`, `subprocess`, and `requests` for various functionalities such as file operations, interacting with

the OpenAI API, making HTTP requests, etc.

2. **Loading Environment Variables**: It loads environment variables from a `.env` file using `dotenv_values()` and sets up an OpenAI client using an API key retrieved from the environment variables.

3. **Defining Constants and Variables**: The script defines constants like `github_token`, `file_path`, `repo_url`, `repo_owner`, and `repo_name`, which are used throughout the script.

4. **Function Definitions**:

   - `create_git_branch()`: Creates a new git branch using subprocess calls.
   - `push_git_branch()`: Pushes a git branch to the remote origin.
   - `commit_files()`: Commits files to the git repository.
   - `retrieve_code_blocks()` and `retrieve_file_name()`: Extract code blocks and file names from input strings using regular expressions.
   - `detect_design_smells()`: Uses the OpenAI API to identify design smells in the given code.
   - `refactor_code()`: Uses the OpenAI API to refactor the code based on identified design smells.
   - `create_pull_request()`: Creates a pull request on GitHub using the GitHub API.

5. **Detecting Design Smells**:

   - The script reads the original code from a file specified by `file_path`.
   - It detects design smells in the original code using the `detect_design_smells()` function and stores the results in `detected_smells`.

6. **Refactoring Code**:

   - The script refactors the original code based on the detected design smells using the `refactor_code()` function.
   - It generates refactored code and selects a random design smell for refactoring.

7. **Creating Pull Request**:

   - The script creates a pull request on GitHub to propose the refactored code.
   - It creates a new branch, commits the refactored code changes, and pushes the branch to the remote repository.
   - Finally, it uses the GitHub API to create a pull request with details such as title, body, head, and base.

8. **Error Handling**: The script includes error handling for various operations like creating branches, committing files, pushing branches, and creating pull requests.

Overall, the script automates the process of identifying design smells, refactoring the code, and creating a pull request for the refactored code on GitHub.

Pull Request Generation:

```

```

```
curl -L   -X POST   -H "Accept: application/vnd.github+json"   -H
"Authorization: Bearer
github_pat_11AXNX2JY00eNth4gTXys6_0Ev95IoUXYBdS2pSC4WnLtkoOUBJ5zkwtmErER05O
FSWR262QDQGQJeMlgY"   -H "X-GitHub-Api-Version: 2022-11-28"
https://api.github.com/repos/serc-courses/se-project-1--_18/pulls   -d "
{'title': 'Automated Refactoring: Inappropriate Intimacy', 'body': '# Smell
description\nThe UserDao class is tightly coupled with the EntityManager
class, making it difficult to change the persistence mechanism in the
future\n\n# Refactoring: \nRefactor the UserDao class to use a separate
Data Access Object \(DAO\) class for handling persistence operations\\n',
'head': 'Inappropriate-Intimacy-223', 'base': 'master'}"
```

We have developed a mechanism to automatically create pull requests with refactored code changes based on detected design smells and applied refactoring techniques. However, despite our efforts, we encountered permission errors when attempting to execute the pull request generation process.

We are currently investigating this issue and working towards resolving it to ensure seamless automation of pull request creation.