МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

**Отчёт по лабораторной работе №4**

Специальность ПО11

Выполнил

С. С. Жватель

студент группы ПО11

Проверил

А. А. Крощенко

ст. преп. кафедры ИИТ,

12.04.2025 г.

Брест 2025

Цель работы: научиться работать с Github API, приобрести практические навыки написания программ для работы с REST API или GraphQL API

**Общее задание:** используя Github API, реализовать предложенное задание на языке Python. Выполнить визуализацию результатов, с использованием графика или отчета. Можно использовать как REST API (рекомендуется), так и GraphQL

**Задание 1. Автоматизированный мониторинг активности в GitHub-репозитории**

Условие:

Напишите Python-скрипт, который анализирует активность в указанном

GitHub-репозитории и отслеживает:

1. Новые коммиты

2. Созданные и закрытые issues

3. Открытые и закрытые pull requests

4. Новых контрибьюторов

5. Упоминания в комментариях (@username)

6. Количество звёзд и форков

Скрипт должен уметь:

* Подключаться к GitHub API и получать последние изменения в репозитории.
* Фильтровать события по времени (например, за последние 24 часа).

Выполнение:

**Код программы:**

"""Module to monitor and visualize GitHub repository activity."""  
  
import os  
import re  
from datetime import UTC, datetime, timedelta  
from typing import Dict, List, Set, Tuple  
  
import matplotlib.pyplot as plt  
import requests  
  
# Configuration  
GITHUB\_TOKEN = os.getenv("GITHUB\_TOKEN", "ghp\_5pL4GC027TedyLfjK1jKQYjoPs3aBi04GICK")  
BASE\_URL = "https://api.github.com"  
HEADERS = {"Authorization": f"token {GITHUB\_TOKEN}"}  
REQUEST\_TIMEOUT = 10 # seconds  
  
  
def get\_time\_range(hours: int) -> str:  
 """Get ISO formatted time string for the given hours in the past."""  
 past\_time = datetime.now(UTC) - timedelta(hours=hours)  
 return past\_time.isoformat() + "Z"  
  
  
def fetch\_github\_data(url: str, params: Dict = None) -> List[Dict]:  
 """Generic function to fetch data from GitHub API."""  
 try:  
 response = requests.get(url, headers=HEADERS, params=params, timeout=REQUEST\_TIMEOUT)  
 response.raise\_for\_status()  
 return response.json()  
 except requests.RequestException as e:  
 print(f"Error fetching data from {url}: {e}")  
 return []  
  
  
def is\_valid\_issue(issue: Dict, since: str, check\_closed: bool = False) -> bool:  
 """Check if an issue meets the criteria for being new or closed."""  
 is\_not\_pr = not issue.get("pull\_request")  
 if check\_closed:  
 return issue["state"] == "closed" and issue["closed\_at"] >= since and is\_not\_pr  
 return issue["created\_at"] >= since and is\_not\_pr  
  
  
def get\_commits(repo: str, since: str) -> List[Dict]:  
 """Fetch commits since the specified time."""  
 url = f"{BASE\_URL}/repos/{repo}/commits"  
 return fetch\_github\_data(url, {"since": since})  
  
  
def get\_issues(repo: str, since: str) -> Tuple[List[Dict], List[Dict]]:  
 """Fetch issues (created and closed) since the specified time."""  
 url = f"{BASE\_URL}/repos/{repo}/issues"  
 issues = fetch\_github\_data(url, {"since": since, "state": "all"})  
 new\_issues = [issue for issue in issues if is\_valid\_issue(issue, since)]  
 closed\_issues = [issue for issue in issues if is\_valid\_issue(issue, since, check\_closed=True)]  
 return new\_issues, closed\_issues  
  
  
def get\_pull\_requests(repo: str, since: str) -> Tuple[List[Dict], List[Dict]]:  
 """Fetch pull requests (created and closed) since the specified time."""  
 url = f"{BASE\_URL}/repos/{repo}/pulls"  
 pulls = fetch\_github\_data(url, {"state": "all"})  
 new\_pulls = [p for p in pulls if p["created\_at"] >= since]  
 closed\_pulls = [p for p in pulls if p["state"] == "closed" and p["closed\_at"] >= since]  
 return new\_pulls, closed\_pulls  
  
  
def get\_contributors(commits: List[Dict]) -> Set[str]:  
 """Identify new contributors from commits."""  
 return {commit["author"]["login"] for commit in commits if commit.get("author")}  
  
  
def get\_mentions(repo: str, since: str) -> Set[str]:  
 """Fetch mentions in issue and PR comments."""  
 mentions = set()  
 url = f"{BASE\_URL}/repos/{repo}/issues/comments"  
 comments = fetch\_github\_data(url, {"since": since})  
 for comment in comments:  
 mentions.update(re.findall(r"@([a-zA-Z0-9-]+)", comment["body"]))  
 return mentions  
  
  
def get\_repo\_stats(repo: str) -> Tuple[int, int]:  
 """Get current stars and forks count."""  
 url = f"{BASE\_URL}/repos/{repo}"  
 data = fetch\_github\_data(url)  
 return data.get("stargazers\_count", 0), data.get("forks\_count", 0)  
  
  
def plot\_activity(data: Dict, repo: str, hours: int):  
 """Create a bar plot of activity metrics."""  
 metrics = {  
 "Commits": data["new\_commits"],  
 "New Issues": data["new\_issues"],  
 "Closed Issues": data["closed\_issues"],  
 "New PRs": data["new\_pulls"],  
 "Closed PRs": data["closed\_pulls"],  
 "Mentions": data["mentions"],  
 "New Contributors": data["new\_contributors"],  
 "Stars (k)": data["stars"] / 1000, # Convert stars to thousands  
 "Forks (h)": data["forks"] / 100, # Convert forks to hundreds  
 }  
  
 plt.figure(figsize=(12, 6))  
 plt.bar(metrics.keys(), metrics.values(), color="skyblue")  
 plt.title(f"Activity in {repo} (Last {hours} hours, Stars in Thousands, Forks in Hundreds)")  
 plt.xlabel("Metrics")  
 plt.ylabel("Count (Stars in k, Forks in h)")  
 plt.xticks(rotation=45, ha="right")  
 plt.tight\_layout()  
 plt.savefig("activity\_plot.png")  
 plt.close()  
  
  
def validate\_repo(repo: str) -> bool:  
 """Validate repository format (owner/repo)."""  
 return bool(re.match(r"^[a-zA-Z0-9-]+/[a-zA-Z0-9-]+$", repo))  
  
  
def collect\_repo\_data(repo: str, hours: int) -> Dict:  
 """Collect all repository activity data."""  
 since = get\_time\_range(hours)  
 commits = get\_commits(repo, since)  
 new\_issues, closed\_issues = get\_issues(repo, since)  
 new\_pulls, closed\_pulls = get\_pull\_requests(repo, since)  
 contributors = get\_contributors(commits)  
 mentions = get\_mentions(repo, since)  
 stars, forks = get\_repo\_stats(repo)  
  
 return {  
 "new\_commits": len(commits),  
 "new\_issues": len(new\_issues),  
 "closed\_issues": len(closed\_issues),  
 "new\_pulls": len(new\_pulls),  
 "closed\_pulls": len(closed\_pulls),  
 "new\_contributors": len(contributors),  
 "mentions": len(mentions),  
 "stars": stars,  
 "forks": forks,  
 "contributors\_list": contributors,  
 "mentions\_list": mentions,  
 }  
  
  
def display\_results(data: Dict, repo: str, hours: int):  
 """Display repository activity results."""  
 print(f"\nMonitoring activity in {repo} for the last {hours} hours:")  
 print(f"{data['new\_commits']} new commits")  
 print(f"{data['new\_pulls']} new pull requests ({data['closed\_pulls']} closed)")  
 print(f"{data['new\_issues']} new issues ({data['closed\_issues']} closed)")  
 print(f"{data['new\_contributors']} new contributors: {', '.join(data['contributors\_list'])}")  
 print(f"{data['mentions']} mentions: {', '.join(data['mentions\_list'])}")  
 print(f"{data['stars']} stars, {data['forks']} forks")  
  
  
def main():  
 """Main function to collect and display GitHub repository activity."""  
 repo = input("Enter repository (owner/repo): ")  
 if not validate\_repo(repo):  
 print("Invalid repository format. Use 'owner/repo'.")  
 return  
  
 try:  
 hours = int(input("Enter time range (hours): "))  
 if hours <= 0:  
 raise ValueError("Hours must be positive.")  
 except ValueError as e:  
 print(f"Invalid input: {e}")  
 return  
  
 if not GITHUB\_TOKEN:  
 print("GitHub token not found. Set GITHUB\_TOKEN environment variable.")  
 return  
  
 # Collect and process data  
 data = collect\_repo\_data(repo, hours)  
  
 # Display and visualize results  
 display\_results(data, repo, hours)  
 plot\_activity(data, repo, hours)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Спецификация ввода:**

<владелец/название репозитория>

<временной промежуток>

**Пример:**

Enter repository (owner/repo): kortix-ai/suna

Enter time range (hours): 24

**Спецификация вывода:**

Мониторинг активности в {репозиторий} за последние {часы} часов:

{количество} новых коммитов

{количество} новых запросов на включение ({количество} закрытых)

{количество} новых задач ({количество} закрытых)

{количество} новых участников: {список участников, разделенный запятыми}

{количество} упоминаний: {список упоминаний, разделенный запятыми}

{количество} звезд, {количество} форков

**Пример:**

Мониторинг активности в octocat/hello-world за последние 24 часов:

10 новых коммитов

3 новых запросов на включение (1 закрытых)

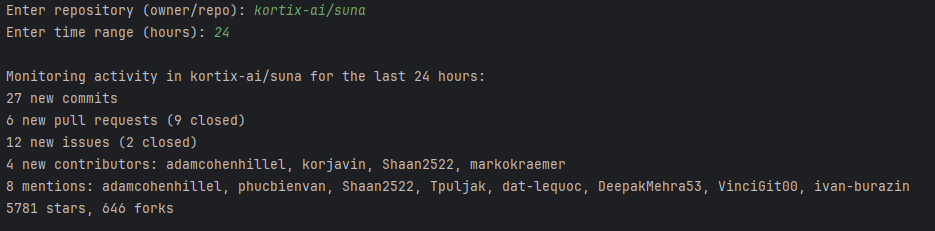
5 новых задач (2 закрытых)

2 новых участников: user1, user2

4 упоминаний: user3, user4, user5, user6

5700 звезд, 570 форков

**Рисунки с результатами работы программы:**

****

**Вывод:** закрепил базовые знания Github API, Python и научился работать с Rest API при решении практических задач.