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", "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")
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

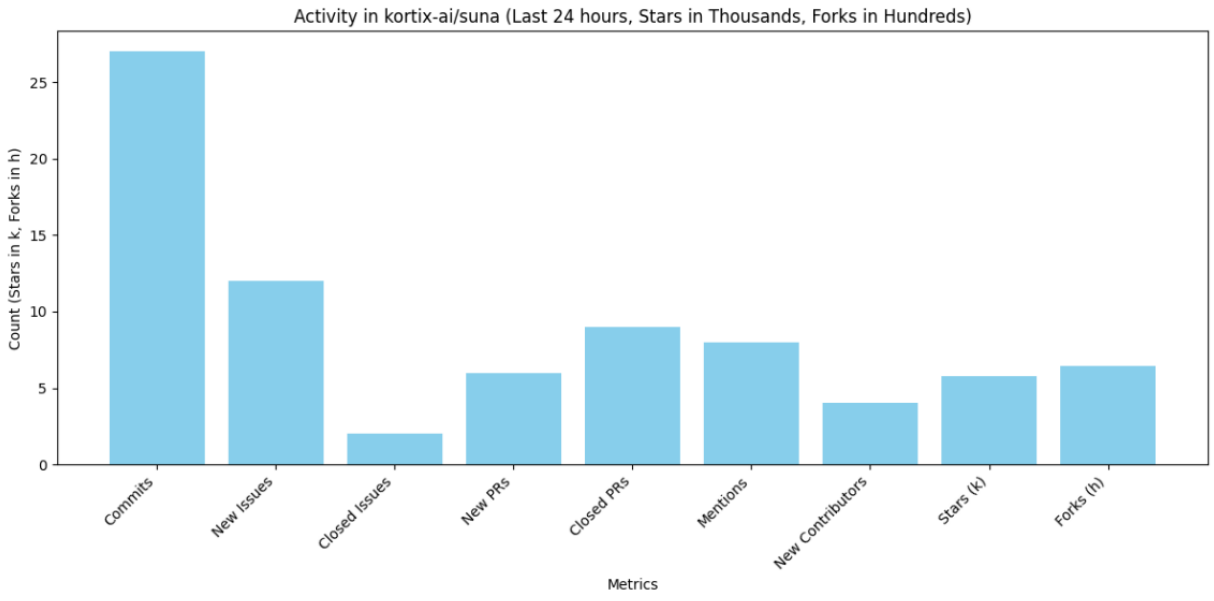
Спецификация вывода:

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:



```
Enter repository (owner/repo): kortix-ai/suna
Enter time range (hours): 24

Monitoring activity in kortix-ai/suna for the last 24 hours:
27 new commits
6 new pull requests (9 closed)
12 new issues (2 closed)
4 new contributors: adamcohenhillel, korjavin, Shaan2522, markokraemer
8 mentions: adamcohenhillel, phucbienvan, Shaan2522, Tpuljak, dat-lequoc, DeepakMehra53, VinciGit00, ivan-burazin
5781 stars, 646 forks
```

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