Async GitHub API Wrapper

Project: Asynchronous API Wrapper Library
Includes: async_github.py (library), example.py (usage), and sample output
Note: This PDF includes simulated example output because this environment
does not have internet access to call the real GitHub API.

async_github.py

```
# async_github.py
import aiohttp
import asyncio
import time
from typing import AsyncGenerator, Dict, Any, Optional
from dataclasses import dataclass

class APIError(Exception): pass
class NotFoundError(APIError): pass
class RateLimitError(APIError): pass
class ServerError(APIError): pass
class BadRequestError(APIError): pass

@dataclass
class RateLimit:
    remaining: int
    limit: int
    reset: float

class GitHubAPI:
    BASE_URL = "https://api.github.com"
    DEFAULT_PER_PAGE = 100

    def __init__(self, token: Optional[str] = None, *, max_retries: int = 3,
    backoff_factor: float = 1.0):
        self._token = token
        self._session: Optional[aiohttp.ClientSession] = None
        self.rate_limit = RateLimit(remaining=5000, limit=5000, reset=0.0)
        self._max_retries = max_retries
        self._backoff = backoff_factor

    async def __aenter__(self):
        headers = {"Accept": "application/vnd.github.v3+json"}
        if self._token:
            headers["Authorization"] = f"Bearer {self._token}"
        self._session = aiohttp.ClientSession(headers=headers)
        return self

    async def __aexit__(self, exc_type, exc, tb):
        if self._session:
            await self._session.close()
            self._session = None

    async def _request(self, method: str, endpoint_or_url: str, *, params: Dict[str, Any]
    = None, json: Any = None, headers: Dict[str, str] = None) -> Dict[str, Any]:
        if self._session is None:
            raise APIError("Client session not started. Use `async with GitHubAPI(...) as
    api:`")
        url = endpoint_or_url if endpoint_or_url.startswith("http") else
    f"{self.BASE_URL.rstrip('/')}/{endpoint_or_url.lstrip('/')}"
        params = params or {}
        attempt = 0
        while True:
            attempt += 1
            try:
                async with self._session.request(method, url, params=params or None,
json=json, headers=headers) as resp:
                    # update rate limit
                    self.rate_limit = RateLimit(
                        remaining=int(resp.headers.get("X-RateLimit-Remaining",
self.rate_limit.remaining)),
                        limit=int(resp.headers.get("X-RateLimit-Limit",
self.rate_limit.limit)),
                        reset=float(resp.headers.get("X-RateLimit-Reset",
self.rate_limit.reset)),
                    )
                    if resp.status == 429:
                        retry_after = resp.headers.get("Retry-After")
                        wait = float(retry_after) if retry_after else 60.0
                        await asyncio.sleep(wait + 0.5)
                        continue
                    if resp.status == 404:
                        raise NotFoundError("404 Not Found")
                    if 400 <= resp.status < 500:
                        raise BadRequestError(f"{resp.status} Client Error")
                    if 500 <= resp.status < 600:
                        if attempt <= self._max_retries:
                            backoff = self._backoff * (2 ** (attempt - 1))
                            await asyncio.sleep(backoff)
                            continue
                        else:
                            raise ServerError(f"{resp.status} Server Error after retries")
                    text = await resp.text()
                    if not text:
                        return {}
                    try:
                        return await resp.json()
                    except Exception:
                        return {"raw": text}
            except aiohttp.ClientError as e:
                if attempt <= self._max_retries:
                    backoff = self._backoff * (2 ** (attempt - 1))
                    await asyncio.sleep(backoff)
                    continue
                raise APIError(f"Network error: {e}") from e

async def get_user(self, username: str) -> Dict[str, Any]:
    return await self._request("GET", f"users/{username}")

async def list_user_repos(self, username: str, per_page: int = DEFAULT_PER_PAGE) ->
AsyncGenerator[Dict[str, Any], None]:
    params = {"per_page": per_page, "page": 1}
    while True:
        result = await self._request("GET", f"users/{username}/repos", params=params)
        if not isinstance(result, list) or not result:
            break
        for repo in result:
            yield repo
        params["page"] += 1
```

example.py

```
# example.py
import asyncio
import os
from async_github import GitHubAPI

async def main():
    token = os.getenv("GITHUB_TOKEN")  # optional
    async with GitHubAPI(token) as api:
        user = await api.get_user("torvalds")
        print("User:", user.get("login"), "| name:", user.get("name"))
        count = 0
        async for repo in api.list_user_repos("torvalds", per_page=10):
            print(repo.get("name"), repo.get("stargazers_count"))
            count += 1
            if count >= 10:
                break

if __name__ == "__main__":
    asyncio.run(main())
```

Simulated Output:

```
# Simulated example.py output (live data not available here)

User: torvalds | name: Linus Torvalds

First 10 repos:
 - linux (stars: 157000)
 - subsurface (stars: 2000)
 - sparse (stars: 400)
 - test-tlb (stars: 12)
 - pet-project (stars: 3)
 - repo-six (stars: 1)
 - repo-seven (stars: 0)
 - repo-eight (stars: 2)
 - repo-nine (stars: 0)
 - repo-ten (stars: 0)

(Actual live output varies when run with internet access / a GitHub token.)
```