Github Repository Analyzer

Build a modular Python application to analyze GitHub repositories

1

Collect data via GitHub REST API v3

Perform limited web scraping with BeautifulSoup

2

Store data in pickle format for persistence

3

Analyze PR metrics: open/closed counts, contributors, temporal trends

4

Generate visualizations (matplotlib)

5

Implement unit tests with mocking

Demonstrate two different use cases of the main class

Constraints & Edge Cases

1

API rate limits (5,000 requests/hour with token)

2

Handle pagination (100 items per page max)

3

Missing/incomplete data from API responses

4

Timezone-aware datetime handling

Analysis Strategy:

Step 1: Architecture Decision
- **Modular design** → Separation of concerns (API, Analysis, Visualization)
- **Class-based approach** → Reusability and testability
- **Type hints throughout** → Better IDE support and documentation

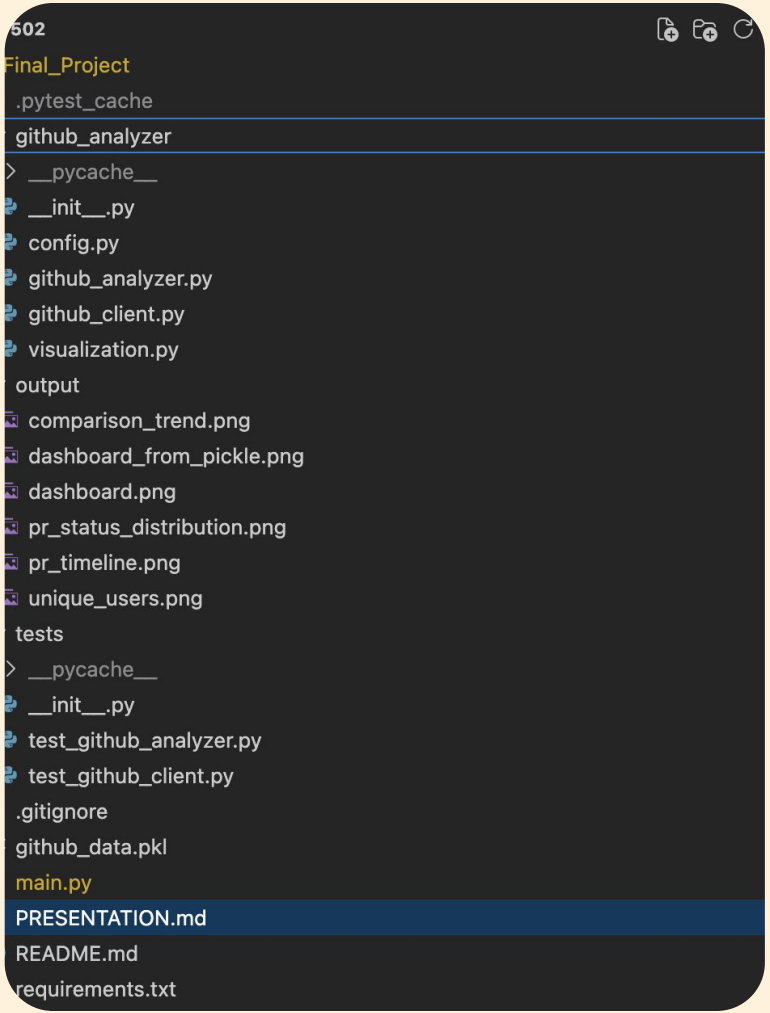
Step 2: Key Decisions

Decision → Rationale → Alternative Considered
- **Pickle storage** → Fast serialization, preserves Python objects → JSON (but loses datetime precision)
- **Requests library** → Lightweight, no external GitHub SDK needed →PyGithub (adds dependency)
- **Matplotlib** → Standard, well-documented → Seaborn (overkill for this project)
- **pytest + mock** → Industry standard, powerful mocking → unit test only (less concise)

Step 3: Trade-offs
- **Scraping limit:** Max 10 users per repo (avoid rate limits)
- **Error handling:** Fail fast with clear messages vs. silent failures
- **Data structure:** Raw data stored separately from processed DataFrames

Project Structure

```
Final_Project/
├── github_analyzer/  # Core package (modular architecture)
│   ├── __init__.py             # Package exports & version info
│   ├── config.py               # Authentication & settings
│   ├── github_client.py        # API client + web scraping
│   ├── github_analyzer.py      # Main analysis logic
│   └── visualization.py        # Plotting functions
│
├── output/ # Generated Plots
│
├── tests/                      # Unit tests (isolated, mockable)
│   ├── test_github_client.py   # API & scraping tests
│   └── test_github_analyzer.py # Analysis logic tests
│
├── main.py                     # Demo script (2 use cases)
├── requirements.txt            # Dependencies
└── README.md                   # Documentation
```



Key Classes & Core Logic

- **Github Client** → Data Collection → Simplifies complex API interactions
- **GitHub Repo Analyzer** → Analysis Engine → Multiple analysis methods, same interface
- **Data Flow** →

API Response → Raw Dict → Pickle → DataFrame → Analysis → Visualization
↓ ↓ ↓ ↓ ↓
JSON data Persistence Pandas Statistics PNG plots

Challenges & Fixes

- **Challenge 1: Rate Limiting**
  - Problem: Hit API limit during testing (403 errors)
  - Solution:
    - Added rate limit detection in error handler
    - Implemented `time.sleep(0.1)` between pagination requests
    - Clear error messages with reset timestamp
- **Challenge 2: Timezone-Aware Datetimes**
  - Problem: Tests failed (`datetime64[ns]` vs `datetime64[ns, UTC]`)
  - Solution: Updated assertions to handle both formats
  ```python
  # Before: assertEqual(dtype, "datetime64[ns]")
  # After:  assertIn("datetime64", str(dtype))
  ```
- **Challenge 3: Mock Testing Edge Cases**
  Problem: Exception handling in scraping not covered
  Solution: Changed from `requests.RequestException` to `Exception`
  Catches all error types in tests

1

Meet the product - Demo

Questions?

GitHub Repository:
https://github.com/sarojk562/INF502/tree/main/Final_Project

Resources

Thank you!