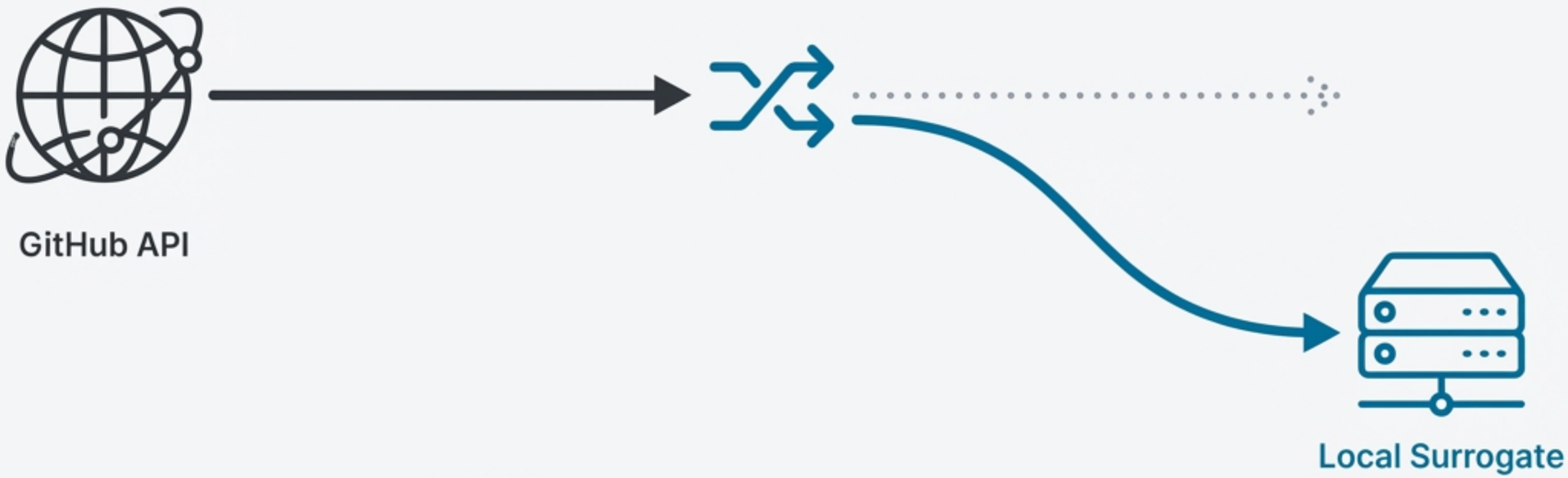
Slide Presentation

The GitHub API Surrogate

A Case Study in Transforming Our Test Suite



Internal Project Debrief

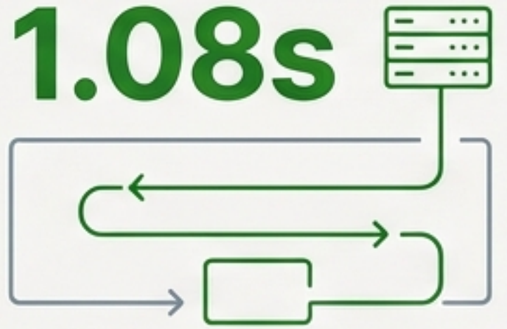
We made our integration tests 20x faster.

BEFORE

20 bS

AFTER

1.08s



20x Speedup

By replacing live GitHub API calls with a local, in-memory surrogate, we reduced the runtime for 553 tests from over 20 seconds to just 1 second, with zero changes to the core test logic.

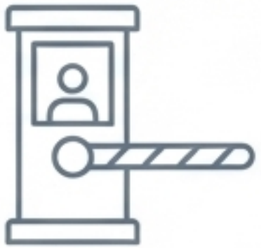
Testing against the live GitHub API was slow, flaky, and complex.

Speed

Each HTTP request takes 200-800ms, making test suites painfully slow.

Authentication

Required valid Personal Access Tokens (PATs), complicating CI/CD pipelines.

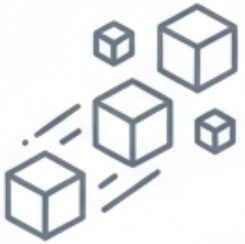


Rate Limits

Hitting the 5,000 requests/hour limit could block development and CI runs.

Network Dependency

Tests failed when offline or during GitHub service degradation.



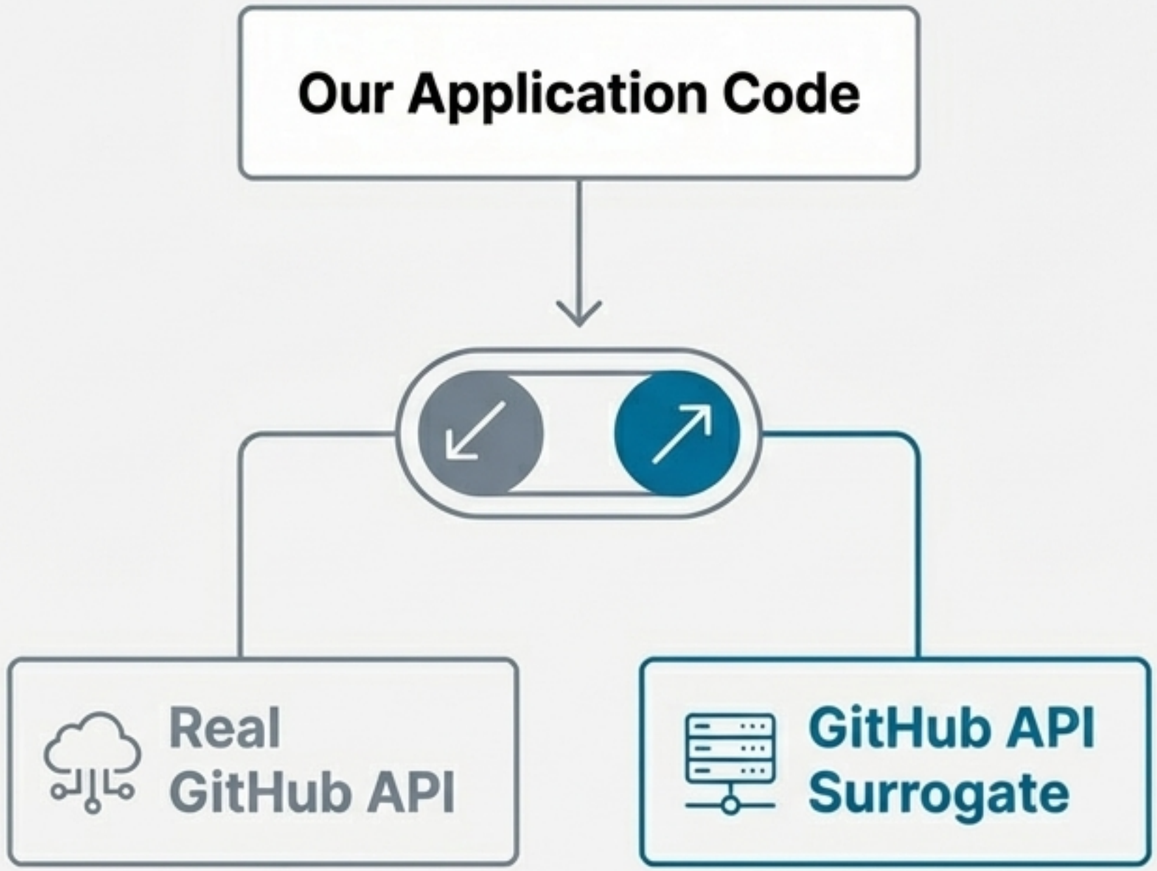
Side Effects

Tests created real resources that required complex cleanup routines.

Non-Determinism

Variable network latency and API responses caused flaky, unreliable tests.

Our solution: A local, in-memory surrogate of the GitHub API.



- A **FastAPI application** that mirrors the GitHub REST API endpoints our code uses.
- Runs **entirely in-memory** with no network calls or disk I/O.
- Maintains its own state for repositories, secrets, and organisations for the duration of the test run.
- Returns responses that precisely match GitHub's real API format, including error conditions.

The transformation is absolute: from network-dependent to self-contained.

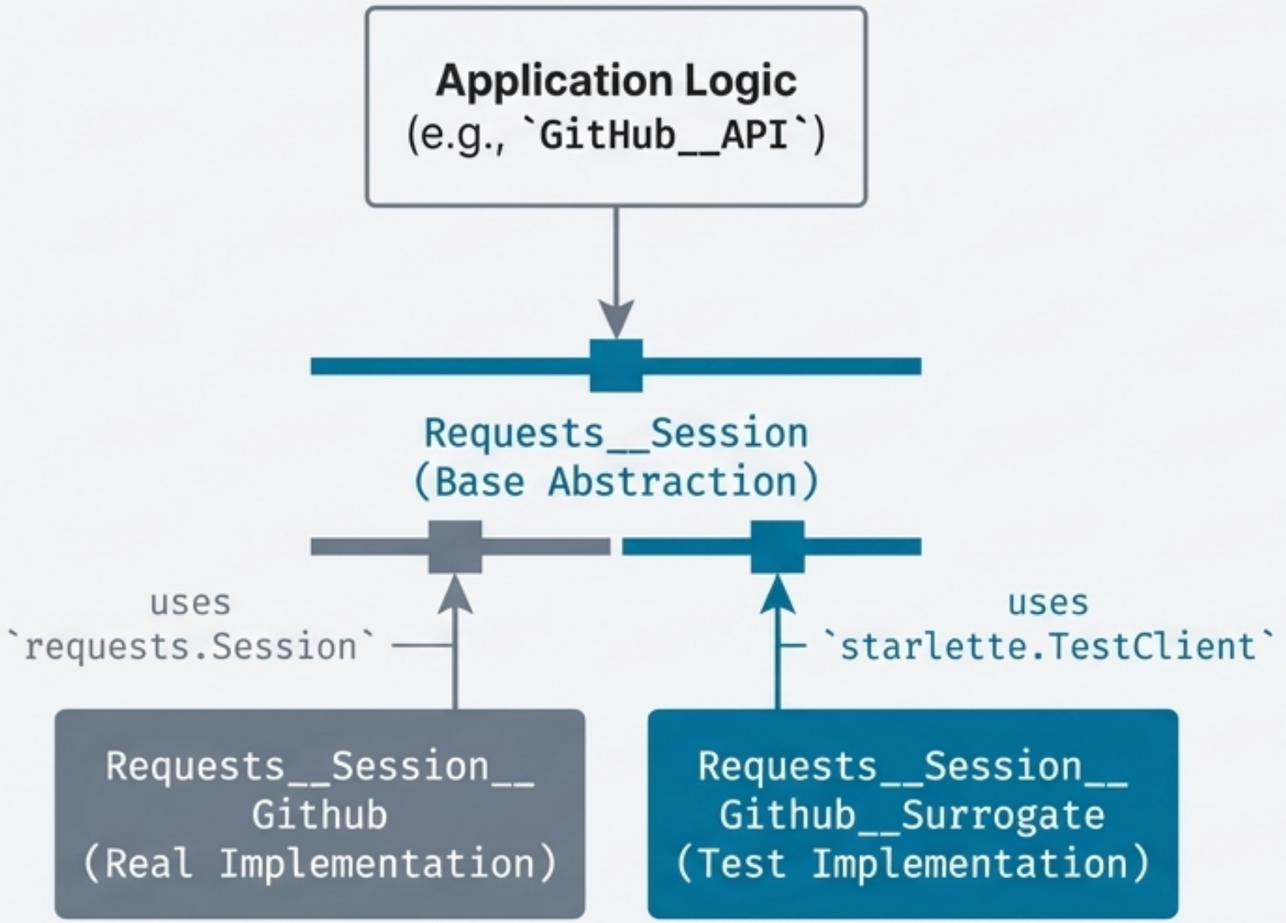
Metric	Before (Real API)	After (Surrogate)
Total test time	~20 seconds	**1.08 seconds**
Network required	Yes (❌)	**No (✔)**
GitHub PAT required	Yes (❌)	**No (✔)**
Rate limit concerns	Yes (❌)	**No (✔)**
Test reliability	Non-deterministic	**100% Deterministic**

The most I/O-heavy tests saw speedups of over 200x.

Test Class	Before (Real API)	After (Surrogate)	Speedup
`test_GitHub__Secrets`	18.5 sec	89 ms	**208x**
`test_Routes__GitHub__Secrets__Env`	6.3 sec	52 ms	**121x**
`test_Routes__GitHub__Secrets__Org`	6.5 sec	58 ms	**112x**
`test_Routes__GitHub__Secrets__Repo`	5.3 sec	64 ms	**83x**

**553 tests passed using the surrogate. 5 tests were skipped for endpoints not yet implemented.*

The key was a clean abstraction for the HTTP session.



We introduced a minimal `Requests__Session` interface.

Production code uses the real implementation that makes network calls.

In tests, we transparently swap in the surrogate implementation that routes calls to the local FastAPI app.

The application code is completely unaware of the difference.

The surrogate implements the exact API surface our application needs.

Key Components

`GitHub__API__Surrogate`: The FastAPI application that mimics the API.

`GitHub__API__Surrogate__Test_Context`: A simple context manager for easy test setup and teardown.

In-Memory State: Manages test data (repos, secrets, etc.) for isolated runs.

Implemented Endpoints

Endpoint	Methods
`'/user'`	GET
`'/rate_limit'`	GET
`'/repos/{...}/actions/secrets/public-key'`	GET
`'/repos/{...}/actions/secrets/{name}'`	GET PUT DELETE
`'/repos/{...}/environments/{...}/secrets'`	GET PUT DELETE
`'/orgs/{...}/actions/secrets'`	GET PUT DELETE

Migrating a test requires only a few lines of setup code.

Before

```
# Before
class TestSomething(TestCase):
    def setUp(self):
        # Complex setup with env vars
        load_dotenv()
        self.pat = get_env("GITHUB_PAT")
        self.api = GitHubAPI(self.pat)
        # Cleanup code in tearDown
```

After

```
# After
class TestSomething(TestCase):
    @classmethod
    def setUpClass(cls):
        cls.surrogate_context = GitHub__API__S
        cls.api = cls.surrogate_context.github
        # Add test data programmatically
        cls.surrogate_context.add_repo(...)

    @classmethod
    def tearDownClass(cls):
        cls.surrogate_context.teardown()
```

The core test logic and assertions remain completely unchanged.

The impact on the developer workflow is immediate.

Run the full suite in 1 second

Get instant feedback before every commit. No more skipping tests locally.

Work completely offline

Code and test on a plane, in a cafe, or during a network outage.

No more secret management

Eliminate `.env` files and the need to provision PATs for local development.

Fully deterministic tests

The same test code produces the same result, every single time.

Easier debugging

Step directly from your test code into the surrogate's Python code to inspect state.

Our CI/CD pipeline is now faster, more secure, and more reliable.

Faster Builds

Test execution time in CI is reduced by **20x**.

Improved Security

No need to store and manage GitHub PATs as secrets in the CI environment.

Eliminated Flakiness

Network-related test failures are a thing of the past.

No Rate Limiting

Run as many parallel builds as needed without fear of being blocked by the GitHub API.

Perfectly Isolated

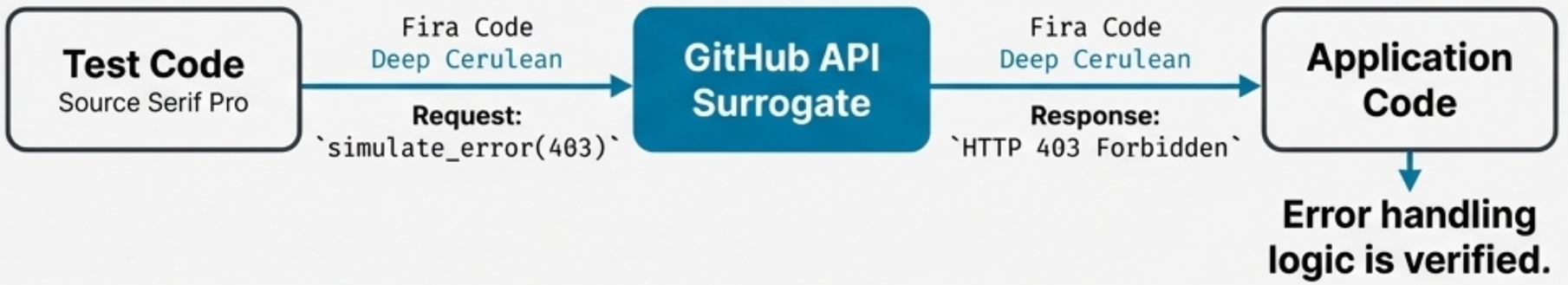
Each parallel test run gets its own isolated, in-memory state.

We can now easily test edge cases and error handling.

The surrogate allows us to simulate failure conditions that are difficult or impossible to create with the live API.

Authentication Failures: What happens when a token is invalid (`401 Unauthorised`) or lacks permissions (`403 Forbidden`)?

Rate Limiting: Verify that our application handles `429 Too Many Requests` gracefully.



Missing Resources: How does the code handle a `404 Not Found` response for a repository or secret?

Invalid Inputs: Test server-side validation errors (`422 Unprocessable Entity`).

A good abstraction is the ultimate enabler for testing.

By introducing the `Requests__Session` interface, we created a “seam” where production and test implementations can be swapped transparently. The production code has no idea it isn’t talking to the real GitHub API.

Total Implementation Effort: **~4 hours**

Ongoing Maintenance: **Minimal** (add new routes as needed)

Benefit: **Immediate and compounding** with every single test run by every developer.

This pattern transformed our testing experience from slow and flaky to fast, reliable, and self-contained.