Software Engineer Case Studies

Nguyễn Hoàng Phúc

Table of Contents

A. Synchronous to Asynchronous Conversion

A.1. Problem

There are many scenarios where synchronous code needs to be converted to asynchronous code. This can be due to performance requirements, responsiveness needs, or architectural decisions. This document outlines the key considerations and steps involved in this conversion process.

An example scenario is an application that receives submissions of codes and needs to run tests on them in order to return the result (such as verdict, status of each tests, etc.). The latency here can be high for example at least 5 seconds per submission due to the time taken to run the tests. If the application is synchronous, it will block the user until the tests are completed, leading to a poor user experience. By converting this process to asynchronous, the application can immediately acknowledge the submission and allow users to continue interacting with the system while the tests are processed in the background.

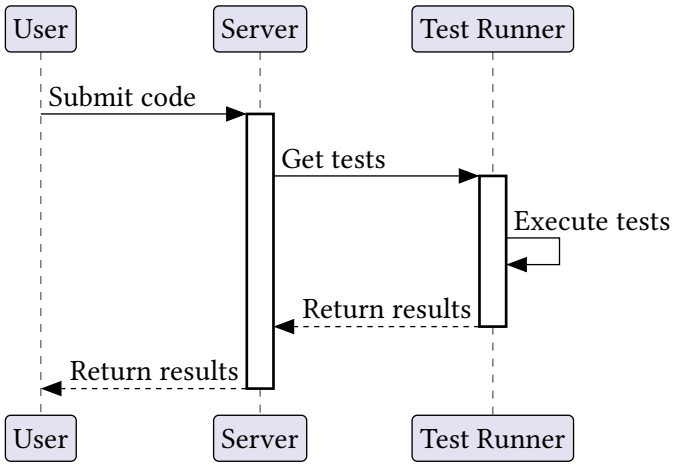

Figure 1: Synchronous Code Execution Scenario

A.2. Solution

We can use a ticket-based queue system to manage asynchronous tasks. This approach decouples the submission from the processing, allowing better scalability and reliability.

A.2.a. Processing Flow (Common to Both Approaches)

The processing pipeline is identical regardless of how results are returned:

1. **User submits code** → Server generates a unique ticket and returns it immediately
2. **Server pushes to queue** → Ticket is added to a queue for processing
3. **Test Runner pulls from queue** → Test Runner fetches tickets from the queue and processes them
4. **Tests execute** → Test Runner executes the tests in the background
5. **Results stored** → Once tests complete, results are stored and associated with the ticket

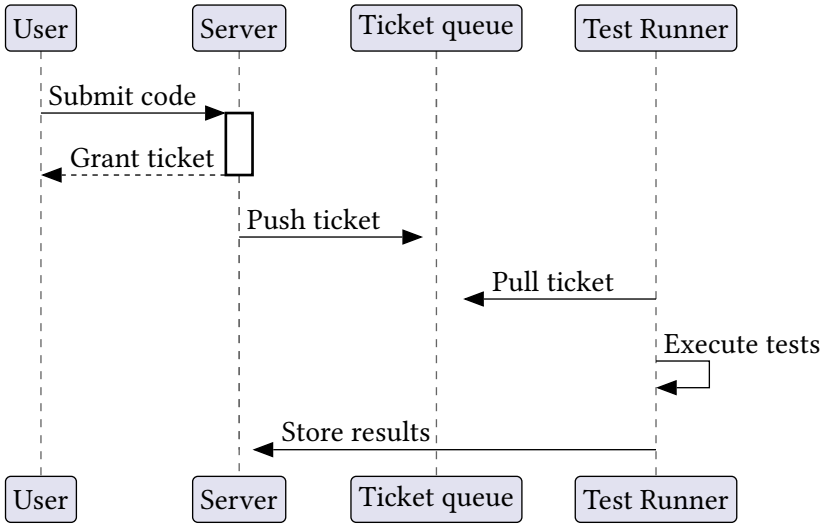

Figure 2: Ticket Queue Processing Pipeline

A.2.b. Result Retrieval: Polling vs Webhook

The key difference between approaches is how results are delivered back to the user.

A.2.b.a. Approach 1: Polling Model

User must periodically check for results using the ticket. This is simpler to implement but requires the user to actively poll.

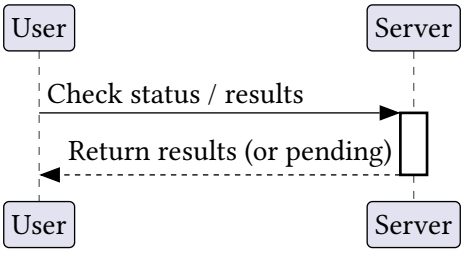

Figure 3: Polling: User-Initiated Result Retrieval

A.2.b.b. Approach 2: Webhook/Callback Model

Server proactively notifies the user when processing is complete. The user provides a callback URL when submitting code, and the server invokes this endpoint with the results.

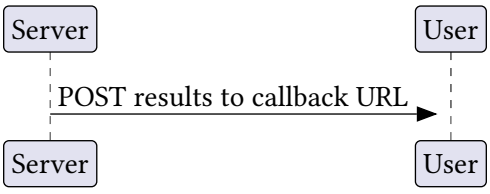

Figure 4: Webhook: Server-Initiated Result Delivery

A.3. Comparison of Approaches

The following table compares the two main async processing approaches:

Criteria	**Queue-Based Polling**	**Webhook/Callback**
User Notification	User must poll	Push (immediate)
Network Overhead	High (frequent polls)	Low (notify once)
Implementation	Moderate	Moderate-Complex
Scalability	Excellent	Excellent
Latency	Depends on poll interval	Minimal
Decoupling	Server, Queue, Test Runner	Server, Queue, Test Runner
Best Use Case	Distributed systems	Real-time notifications
Failure Handling	Auto-retry in queue	Retry webhook calls

Table 1: Comparison of Queue-Based vs Webhook Approaches

A.4. Conclusion

Both approaches effectively convert synchronous processing to asynchronous. The choice depends on application requirements, user experience considerations, and infrastructure capabilities. Polling is simpler but may introduce latency, while webhooks provide real-time updates at the cost of increased complexity.