Assignment 1 Report

Testing Exception-Handling Policies in Concurrent Systems

Concordia University

COEN 448

Software Testing and Validation

Course Instructor: Yan Liu

By

Mohamad Edelby (40251628)

Section W WA

"I certify that this submission is my original work and meets the Faculty's Expectations of Originality" – Mohamad Edelby (40251628), February 15, 2026

Date Submitted: Friday February 15th, 2026

Github Link: https://github.com/momerz7/448_A1

Task 2 - Implementing Policies

Fail-Fast Policy

The approach taken to implement this policy uses CompletableFuture.allOf in addition to propagates exceptions. The key is that no partial result is returned so that if one service fails, the whole system fails.

Fail-Partial Policy

The approach taken to implement this policy is where only successful results are returned. If there is a failure, it is ignored. The final output only has successful operations, failures aren't determinantal to the system holding.

Fail-Soft Policy

The main part of this policy is that if a failure occurs then a predefined fallback value replaces it. This ensures that the system completes even if a service fails.

Task 3 – Unit Testing

Requirements were provided to incorporate unit testing.

- No Mockito
- All futures must be awaited with timeouts
- Tests must verify policy semantics

And categories were specified:

- Fail-Fast: failure propagates (assertThrows)
- Fail-Partial: partial results returned
- Fail-Soft: fallback values used
- Liveness: no deadlock or infinite wait
- Nondeterminism: completion order observed (not asserted)

My implementation of Unit Testing

Function	Purpose	Strict Rules	Test Categories
Static class FailureHelper extends Microservice	Simulates microservice that always fails, returning CompletableFuture.failedFuture(…)	• No Mockito • Timeout-safe permit futures to still complete	• Fail-fast • Fail-Partial • Fail-Soft
void testFailurePropagates()	Exceptions propagates when any microservice fails	• No Mockito • Timeout: (get(1,TimeUnit.SECONDS)) • Tests policy semantics	• Fail-Fast: failure propagate (assertThrows) • Liveness: no deadlock
void testSuccessFailFast()	Ensures policy succeeds normally when every microservices succeed	• No Mockito • Timeout used • Tests policy semantics	• Fail-fast: full success path • Liveness
void testOnlySuccess_failPartial()	Test that only returns when successful results and ignores failures	• No Mockito • Timeout used • Tests policy semantics (partial results allowed)	• Fail-Partial: partial results returned • Liveness
void testEmptyList_failPartial()	When all microservices fail → Fail-partial returns empty list	• No Mockito • Timeout Used • Tests policy semantics (no exception escapes)	• Fail-Partial: empty output if total failure
void testFallbackFailure_failsoft()	Failed results replaced with a fallback value + successful results kept	• No Mockito • Timeout Used • Tests policy semantics (substitute with fallback value)	• Fail-soft: fallback used for failures
void testAllFail_failsoft()	Ensures Fail-Soft policy is still returning a valid result even if all microservices fail, these valid results would be all fallback values	• No Mockito • Timeout • Never throws	• Fail-soft: always normal completion

In addition to these unit tests that involved the three policies, there was also a nondeterministic requirement. This was achieved with testProcessAsync(): asynchronous pipeline works well when every microservice succeeds (formatting + ordering), testProcessAsync_withDifferentMessages(): execpected output is produced thanks to processor

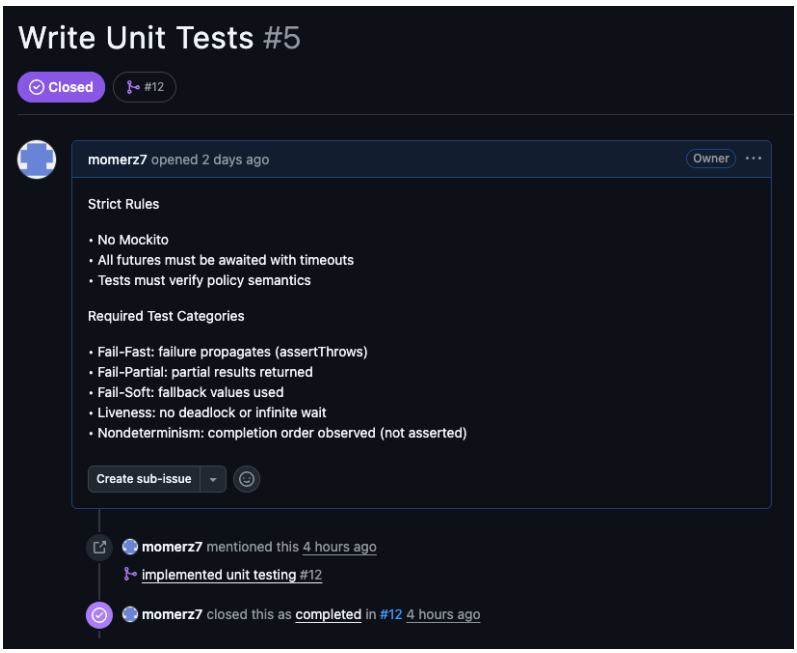
being handled correctly with corresponding inputs, and showNondeterministic_completionOrderVaries(): micoservices complete in nondeterministic order.

Task 4 – Documentation

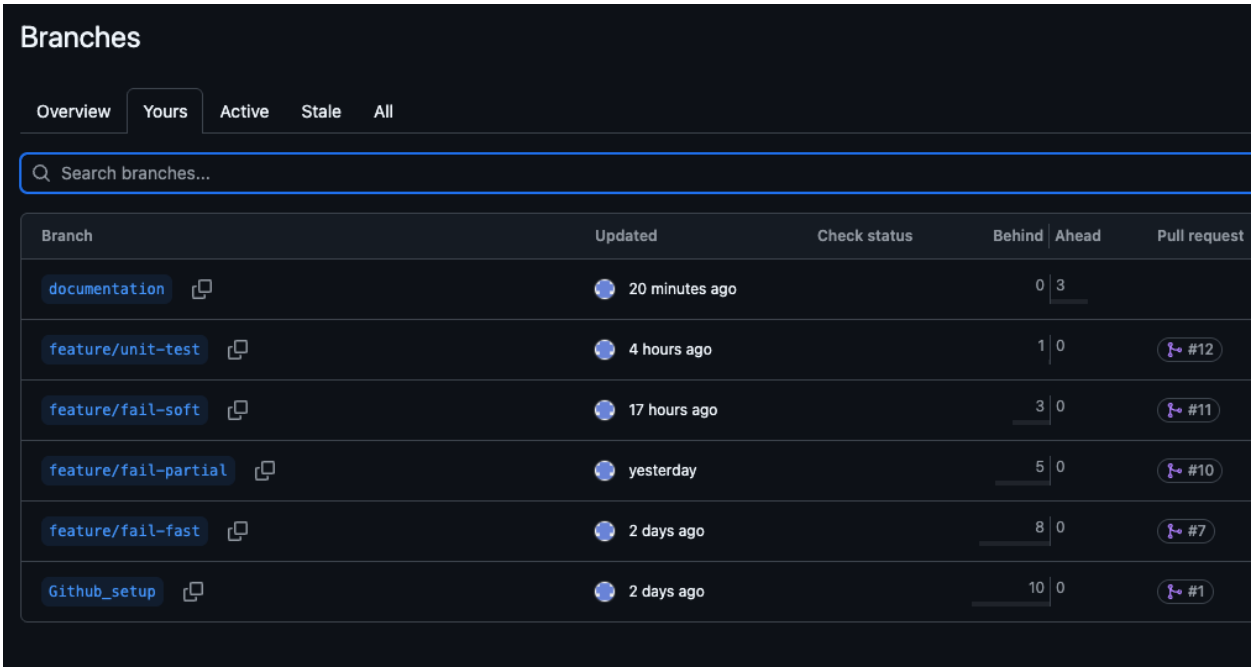
“docs/failure-semantics.md” was created and included in the GitHub repo. This document defines how each of the policies work, their risks, and examples of what kind of systems they can be incorporated in.

GitHub Workflow Screenshots

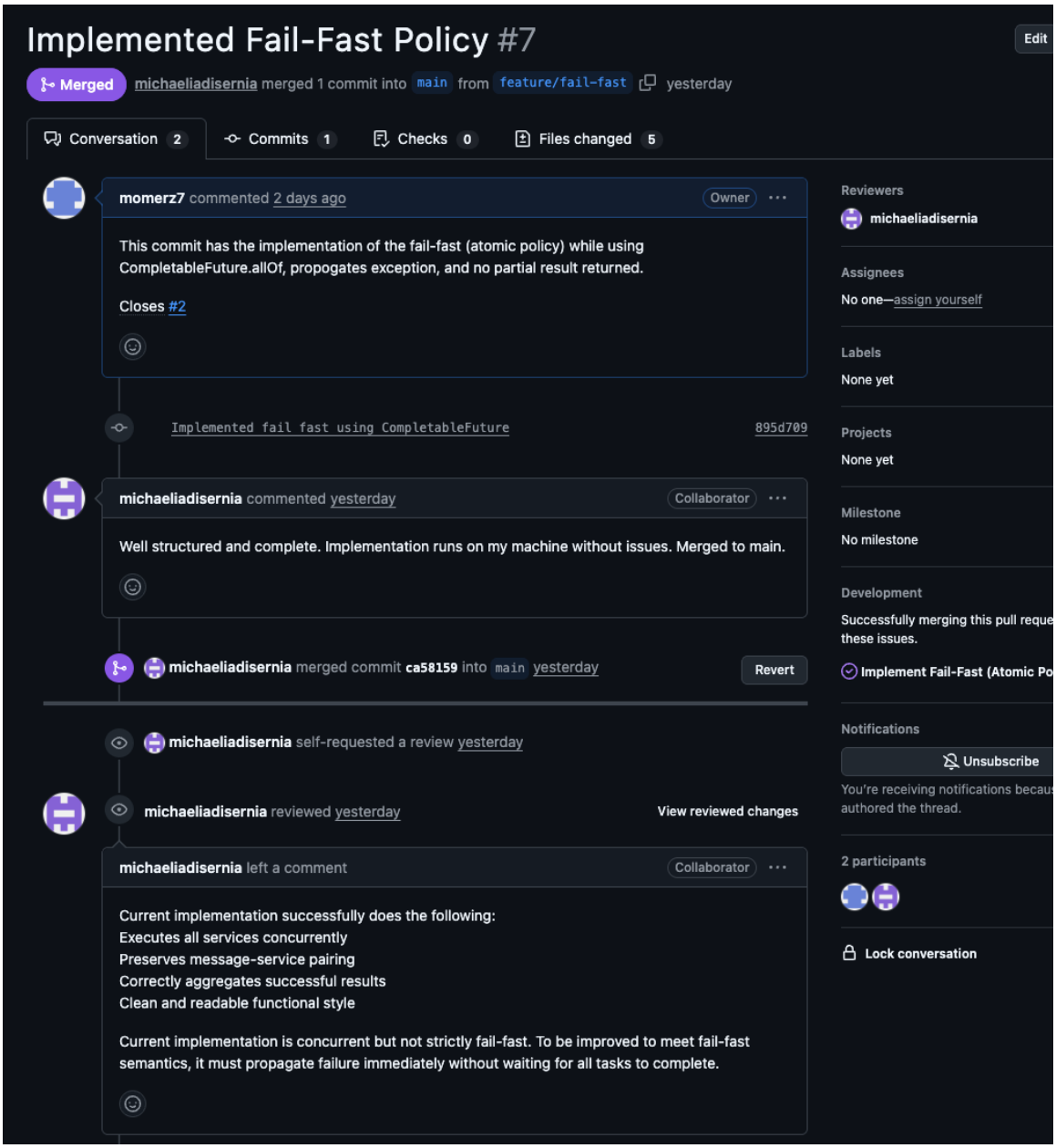
3 Github Issues

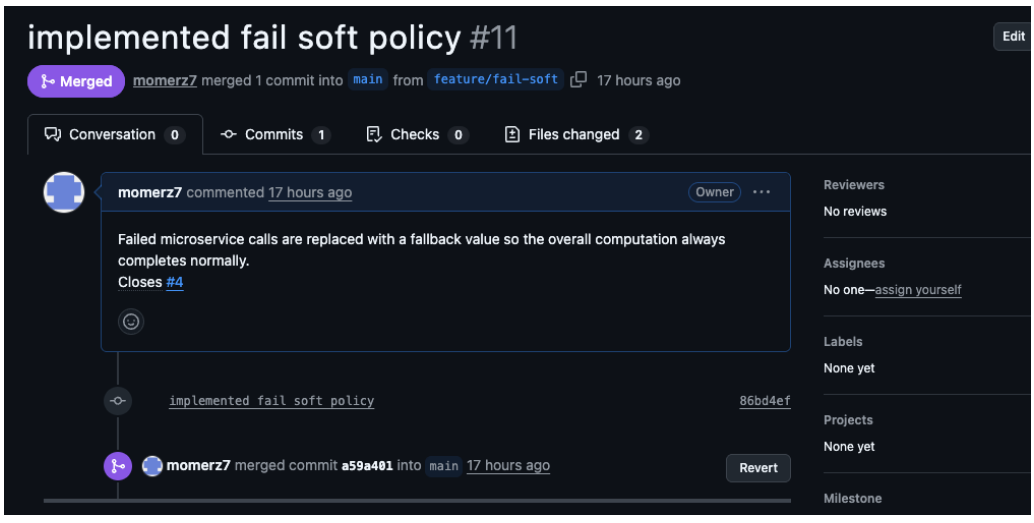


2 Feature Branches



2 Pull Requests





1 Peer Code Review