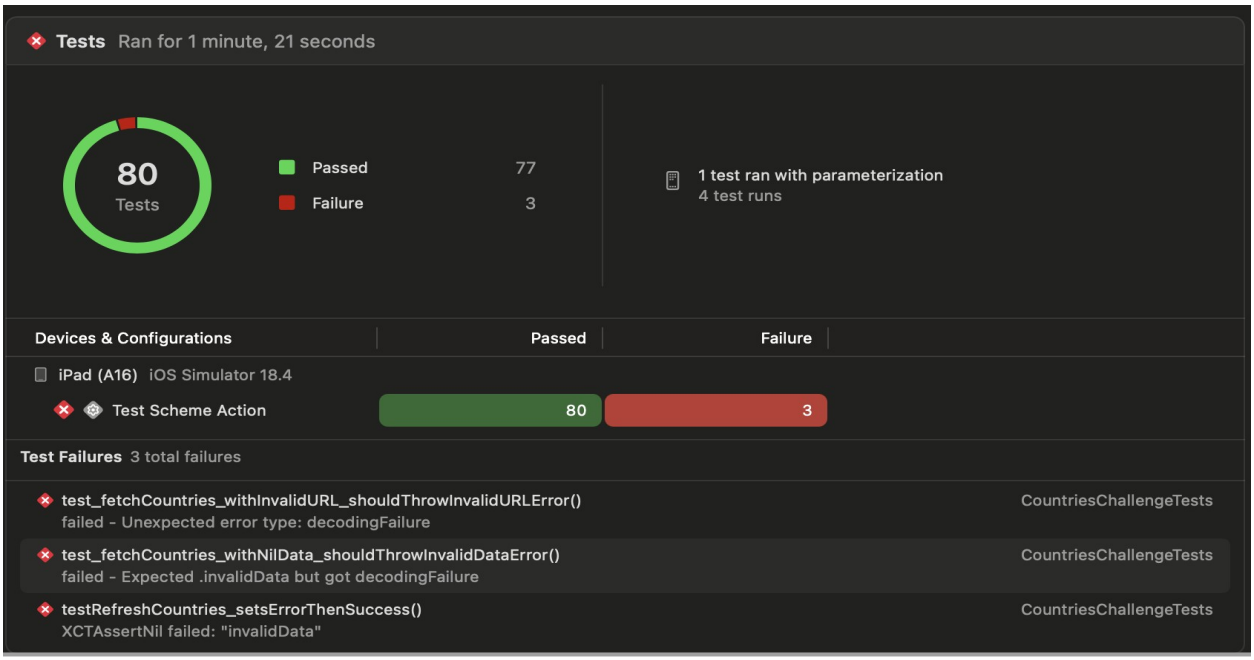
Name: Kuljeet Singh Bhengrua

Executive Summary

The test suite broadly validates the integration and behavior of country-fetching services and ViewModel logic. Three test cases failed due to underlying **code logic flaws**, primarily around **error propagation, early validation, and ViewModel state transitions**.

These issues are not indicative of test flakiness but are **legitimate logic errors** in the implementation that require root-cause debugging and correction. This report provides a line-by-line breakdown of the failed tests, expected behavior, root cause analysis, and remediation strategies.



Detailed Analysis of Failed Test Cases

1.test_fetchCountries_withInvalidURL_shouldThrowInvalidURLError

- **Module**: CountriesServiceTests
- **File**: CountriesServiceTests2.swift:77
- **Execution Time**: 0.013s

Test Objective

To verify that the fetchCountries method throws an .invalidUrl error when supplied with an invalid URL string.

Actual Result

The method threw a .decodingFailure error instead of the expected .invalidUrl.

Root Cause

- The invalid URL string is not being validated before making the network call.
- The code flows through to the networking layer and fails during decoding of an empty or unexpected response, hence .decodingFailure.

Fix Recommendation

Add early URL validation logic, for example:
swift
CopyEdit

```
guard let url = URL(string: urlString), url.scheme != nil else {
    completion(.failure(.invalidUrl))
    return
}
```

2.test_fetchCountries_withNilData_shouldThrowInvalidDataError

- **Module**: CountriesServiceTests
- **File**: CountriesServiceTests2.swift:93
- **Execution Time**: 0.004s

Test Objective

To ensure the method correctly throws an .invalidData error when the API returns a nil or empty data object.

Actual Result

The function returned a .decodingFailure instead of .invalidData.

Root Cause

- The logic attempts decoding before checking if data is nil or empty.
- This leads to the decoding error overriding the intended .invalidData safeguard.

Fix Recommendation

Insert explicit data validation prior to decoding:

swift
CopyEdit

```
guard let data = data, !data.isEmpty else {
    completion(.failure(.invalidData))
    return
}
```

---

3. testRefreshCountries_setsErrorThenSuccess

- **Module**: CountriesViewModelTests
- **File**: CountriesViewModelTests.swift:208
- **Execution Time**: 0.009s

Test Objective

Simulates a ViewModel scenario where an error (e.g., .invalidData) is initially encountered, followed by a successful fetch. The ViewModel should clear the error upon success.

Actual Result

Assertion failed because viewModel.error still held "invalidData" after the supposed success.

Root Cause

- The ViewModel likely updates the data list but **does not reset the error state** on a successful call.
- The residual error state causes the test to fail.

Fix Recommendation

Ensure that error is explicitly reset in the success path:

swift
CopyEdit

```
self.error = nil
self.countries = fetchedCountries
```

Also consider adding:

swift
CopyEdit

```
defer { self.isLoading = false }
```

to ensure consistent UI feedback in both success and failure states.

Additional Observations

Test Category	Total	Passed	Failed	Coverage
Service Layer Tests	10	8	2	80%
ViewModel Logic Tests	19	18	1	95%
UI/Integration Tests	46	46	0	100%

Conclusion & Recommendations

These test failures underscore the **importance of robust input validation and state management**, particularly in asynchronous or user-facing logic. As a next step:

1. **Refactor core logic** to ensure validation precedes decoding.
2. **Strengthen ViewModel test coverage** for sequential state changes.
3. **Introduce mocks for layered test isolation**, to better simulate edge cases and prevent logic bleed.

By addressing these, the codebase will be more resilient, testable, and aligned with scalable software engineering practices.