ChatGPT

Project Amogh SSOT Test Documentation

Overview

This document consolidates the Single Source of Truth (SSOT) test plan, code files, scripts, and logs for the Amogh Financial News and Investment Dashboard. The aim is to provide a benchmark-level test plan and supporting artifacts that meet or exceed the standards of Apple and Google projects.

Roger's Executive Review

Roger, our code architect, evaluated the project plan and identified strengths and enhancements required. Key strengths include a clear purpose, a futuristic visionOS-inspired UI, rigorous fact-checking, layered architecture, modularity, and security considerations. Enhancements include:

- Simulated failure tests to verify system resilience, such as intentionally injecting invalid API keys or civic content to ensure rejection.
- Real-device testing across multiple mobile platforms.
- Performance benchmarks for smooth scrolling and load resilience.
- Stress testing the SSOT to handle high update volumes.
- Resilient CI/CD with deliberate failure injection for automatic recovery.

SSOT Test Document Structure

The SSOT test document is structured as follows:

Section 1: Summary

This includes the execution date, commit hash, trigger method, environment details, and high-level results. It ensures each test run is uniquely identifiable.

Section 2: Pipeline Logs

This section records each phase (UI build, API tests, SSOT injection) with the trigger, result, and log reference. Each log contains the executed command, console output, expected versus actual result, and the AI agent signature.

Section 3: Real-World Failure Simulation Tests

Intentional failures are injected to verify the system's robustness. Examples include:

- **Civic Content Rejection:** Inserting civic topics and verifying they are rejected by the content validation layer.
- **API Authentication Error:** Removing API keys to trigger fallbacks and ensure user alerts appear.
- **SSOT Schema Mismatch:** Introducing a law-related topic field to ensure the schema rejects non-financial topics.

Section 4: Coverage Summary

Test coverage targets are set to 90% for API tests and 85% for UI components, with real percentages recorded. End-to-end tests also have a defined threshold.

Section 5: Device Compatibility

The UI is tested on multiple devices (e.g. iPhone 14, Redmi Note 10, iPad Air) to ensure cross-platform compatibility.

Section 6: Security and Validation

Ensures Zod schema validation, JWT expiry, role-based access control, and environment variable handling are tested and logged.

Section 7: AI Execution Evidence

Logs must include timestamps, commands, and AI agent signatures to prove that tasks were executed on a real environment, not hallucinated.

Test Files and Scripts

`tests/api/content.test.ts`

This Vitest suite verifies the paginated mock feed and intentionally fails when civic content appears:

```
import { describe, expect, it } from 'vitest'
import { getMockFeed } from '@/api/content'

describe('Feed API Test Suite', () => {
  it('should return 10 items in mock paginated feed', async () => {
    const data = await getMockFeed(1)
    expect(data.length).toBe(10)
  })

  it('should simulate civic content rejection', async () => {
    const civicItem = { title: 'कानूनी फैसला', topic: 'law' }
    const isValid = civicItem.topic !== 'law'
    expect(isValid).toBe(false) // must fail
  })
})
```

`scripts/ssotInject.ts`

This script attempts to write an invalid topic into the SSOT store. It logs a failure when the topic violates the schema:

```
import fs from 'fs'
import path from 'path'

const data = {
  title: 'FDI rises 28%',
  content: 'Foreign investment increased this quarter',
  topic: 'law' // intentionally invalid
}

try {
  const schema = ['title', 'content', 'topic']
  if (data.topic === 'law') {
    throw new Error('SSOT schema violation: Topic must be finance/
investment')
  }
  fs.writeFileSync(path.join(__dirname, '../ssot-store/feed.json'),
JSON.stringify(data, null, 2))
  console.log('  SSOT Injected')
} catch (e) {
  console.error('  SSOT Injection Failed:', e.message)
  process.exit(1)
}
```

`.github/workflows/ci.yml`

The CI workflow runs tests and enforces coverage thresholds:

```
name: CI Pipeline

on:
  push:
    branches: [main]
  pull_request:

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install deps
        run: npm ci
      - name: Run Vitest
        run: npm run test:unit
      - name: Check coverage threshold
        run: |
          npm run coverage
          COVER=$(node checkCoverage.js)
          if [ $COVER -lt 90 ]; then exit 1; fi
```

`checkCoverage.js`

This script reads the coverage summary and exits with a non-zero status if the line coverage drops below 90%:

```
const fs = require('fs')
const json = JSON.parse(fs.readFileSync('coverage/coverage-summary.json',
'utf8'))
const pct = json.total.lines.pct
console.log('Coverage:', pct)
if (pct < 90) process.exit(1)
```

Example Log: `logs/ssot-fail.log`

An excerpt from the log produced by a failing SSOT injection:

```
[2025-08-08T14:58:31.102Z] Running script: ssotInject.ts
[2025-08-08T14:58:31.150Z]   SSOT Injection Failed: Topic must be finance/
investment
[2025-08-08T14:58:31.151Z] Triggered by: Gemini-CLI v2.5-pro
```

Outstanding Tasks

Based on the current status, the following items remain open:

Frontend

- Consolidate routing to use either the Next.js App Router or the Pages Router.
- Implement strict language toggles for Hindi and English.
- Integrate the infinite feed into the main dashboard layout.

API and Tests

- Re-enable API tests by adding a test harness or mocking `next/server` for the `src/app/api` endpoints.
- Remove duplicate mocks from `__mocks__` and `src/lib/test-utils/__mocks__/`.

Data and Services

- Replace the mock feed with real data sources, configure environment secrets, and decide between Prisma models or raw SQL with migrations.
- Implement a data ingestion and refresh engine (polling, webhooks, or WebSockets).

Security and Auth

- Implement NextAuth with role-based access control.
- Add Zod or Joi validation to all endpoints.

Infrastructure and Deployment

- Add missing Docker Compose files (e.g. `Dockerfile.websocket`, `Dockerfile.ingestion`, nginx/redis configurations).
- Move CI workflow to the repository root.
- Verify Kubernetes monitoring and configure Prometheus/Grafana with health endpoints.

Automation (MCP)

- Fix `start-mcp.js` to align with the export shape of `mcp-server.js` before using MCP tools.

Documentation

- Update the README with commands to run the dashboard locally (e.g. `npm ci`, `npm run dev`), test and coverage commands, and required environment variables.

Test Evidence Document

Each test run must be documented in a versioned SSOT markdown file (e.g. `SSOT_Project_Amogh_V1_2025-08-08_14-30-00_IST.md`). Include: - The full list of tests and their intentions. - All real test logs and console outputs. - Timestamps and agent names. - Results of intentional failure tests.

Conclusion

This SSOT document and supporting files provide a comprehensive, verifiable testing framework for the Amogh dashboard. By adhering to this standard, the project can maintain transparency, traceability, and resilience, ensuring that it remains a benchmark for future industry projects.
