Test Case ID	Test Case	Test Steps	Expected Results
TC 001	Check Malik and Mike are not friends, not blocked and have valid accounts	1. Malik searches for Mike 2. Malik sends friend request to Mike	- Verify Mike receives pending invite notification. - Malik sees Pending friend request for Mike. - Verify invite sent successfully and PENDING status is shown for both
TC 002	Check Pending status between Mike and Malik	1. Mike opens Pending request 2. Mike 'Accepts' request	- Verify status is changed from Pending to Friend and both players see each other as a Friends. - Successful popup notification should be shown
TC 003	Check Pending status exists between Malik and Mike	1. 'Rejects' button is shown 2. Mike opens pending request 3. Mike clicks on 'Rejects' button	- Verify Malik should receive rejected notification - Relation should be deleted as no-friends
TC 004	Check Pending status exists between Malik and Mike	1. 'Cancel' button is shown 2. Malik opens pending request 3. Malik clicks on 'Cancel' button	- Verify Mike should no longer sees pending request - relation should no longer exists
TC 005	Check sending friend request to existing friend	1. Malik and Mike are friends 2. No friend request button is shown	- Verify system should prevent sending friend requests to existing friend OR should show error notification
TC 006	Check Block status for existing friend	1. 'Block' button is shown 2. Malik should open Mike 's profile and can block Mike 3. Mike should open Malik's profile and block Malik	- Verify blocked status should be shown when Malik blocks Mike - Blocked status should be shown when Mike blocks Malik - Message shouldn't be sent or

			should show disabled button or unavailable
TC 007	Check Un-block status for existing friend	1. 'Un-block' button is shown 2. Malik should open Mike 's profile and can un-block Mike 3. Mike should open Malik's profile and un-block Malik	- Verify players should be able to interact - Status should be changed for both players - Should show successful notification
TC 008	Check Block status for non-friend	1. Malik and Mike exist as players 2. 'Block' button shown on Mike's profile 3. 'Block' button shown on Malik's profile 4. One can block other ONLY 5. Malik can click 'Block' button on Maik's profile	- Verify status of relation should be changed - Mike should be blocked OR Malik should be blocked AND System should not allow mutual blocks or NOTIFY "Blocked" - Should not send friend request - Should show successful notification
TC 009	Check Un-block status for non-friend	1. Malik or Mike are blocked 2. 'Un-Block' button shown on Mike's profile 3. 'Un-Block' button shown on Malik's profile 4. Malik or Mike can click 'Un-Block' button on Mike's profile	- Verify players should be able to interact - Status should be changed for both players - Should show successful notification
TC 010	Check relationship status	1. Mike & Malik can view each other's relation status	- Should show correct relationship status PENDING/REJECTED/FREIND
TC 011	Check player can send Self friend request	1. Malik/Mike can send self-request	- Should show validation error notification OR self-request button should not be displayed -

TC 012	Check resend invite after unblocking	1. Malik sends friend request to Mike after un-block 2. Mike sends friend request to Malik after un-block	- Block status should be removed and successfully request should be sent. - Should show success notification
Edge Case	Check if player removes account having friends		-Should show inactive or removed to all friends
	Check player register again with same username		-Should show new profile OR restored old relation as intended

Part 2 – Regression Test for a Known Bug

In **WW3**, armies can have configurable tokens (buffs for attack/defense/speed). Tokens:

- Have visibility rules (`own armies`, `allies`, `neutrals`, `enemies`).
- Can be consumed by time-based expiration or by game events (battles).

A past bug occurred when certain visibility/consumption combinations failed to behave correctly.

Your task:

1. Design and implement regression tests that cover:
    - All visibility permutations.
    - Both consumption strategies.
2. Include at least **one failing test case** that would have caught this bug.
3. Deliver a short **test plan document** explaining your test design and approach.

Test Scope:

1. **Objective:**
   There is incorrect interaction between
    i. Token visibility
    ii. Consumption strategies

The goal is to achieve combinatorial coverage for two primary variables

i. Visibility permutations (x4)
ii. Consumption strategies (x2)

- **Test Categories:**

i. Visibility Permutations

- Own armies can see own tokens
- Allies can/cannot see ally tokens (configurable)
- Neutrals cannot see any tokens
- Enemies cannot see enemy tokens
- Mixed visibility scenarios

ii. Consumption Strategies

- Time-based expiration (TTL)
- Event-based consumption
- Combined time + event consumption

iii. Edge Cases & Bug Scenarios

- Token visibility changes during consumption
- Simultaneous consumption events
- Token state consistency across different viewers

Known Bug Pattern

The original bug occurred when tokens with restricted visibility were consumed by events, causing inconsistent state between different army perspectives.

2. Test Design Approach (Combinatorial Matrix)

This test plan uses 4x2 matrix to cover every combination of the defining variables. By this strategy finding bugs caused by variable interactions is effective.

Variables:

Variable	Permutations	Description
Visibility	OWN, ALLIES, NEUTRAL, ENEMIES	effected by tokens/who can see
Consumption	TIME-BASED, GAME-EVENT, COMBINED TIME + EVENT CONSUMPTION	how token is removed from enemy

Combinatorial Matrix (4x2):

Test Id	Visibility	Consumption	Scenario	Expected Results
TC 001	**Own**	**Time-Based**	Buff visible for own OR Time-based token persists past expiration	Verify buff should be visible to own; invisible to others and automatically expires after duration is passed
TC 002	**Own**	**Game-Event**	Buff visible for own	Verify buff is removed after any game event
TC 003	**Allies**	**Time-Based**	Buff visible to allies	Verify buff visible to allies, own. Should be removed after any battle event
TC 004	**Allies**	**Game-Event**	Buff visible to allies	Verify buff visible to own, allies. should be removed after allies involved in battle
TC 005	**Neutral**	**Time-Based**	Buff visible to neutral	Verify buff visible to own, neutral. Invisible for enemies. Expires after duration
TC 006	**Neutral**	**Game-Even**	Buff visible to neutral	Verify buff visible to own, neutral. Expired after neutral battled
	Enemies	**Time-Based**	Buff visible to enemy	Verify Buff visible to own, enemy. Disappears after overtime

TC 007				
TC 008	**Enemies**	**Game-Based**	Buff visible to enemy	Verify buff visible to own, enemy. Expires correctly overtime

Test Implementation Strategy

- Unit tests for core token logic
- Integration tests for visibility rules
- Scenario tests for consumption edge cases
- Property-based tests for state consistency

Key Test Cases:

1. Visibility Testing

- Own visibility
- Allies visibility
- Neutrals visibility
- Enemies visibility

2. Consumption Testing

- Time-based expiration with TTL values
- Event-based consumption (battle, siege events)
- Mixed consumption scenarios
- Race condition handling

Critical Bug Scenarios:

- Visibility consistency after consumption:

Ensures consumed tokens are invisible to ALL viewers

- Simultaneous consumption:

Tests race conditions between time and event consumption

- State consistency:

Verifies no corruption during complex multi-token operations

Failing Test Cases (Would Have Caught Original Bug)

The test suite includes specific scenarios that would have failed in the buggy version:

- Token visibility inconsistency after event consumption

- Race conditions in simultaneous consumption events

- State corruption with multiple visibility rules

This document fulfills the scope, variables and 2x4 matrix used to ensure full coverage of the token systems visibility and consumption permutations.

This plan outlines the design and provides the rationale for bug catching test case.

Part 3 – QA Tooling & Process Improvements

As a senior QA engineer, you will help us evolve our QA strategy.

Your task:

- Propose **2–3 QA tools or process improvements** that would strengthen our testing workflows.

- Example areas:

    - Automated test data generation for armies/tokens.
    - Reporting dashboard for API tests.
    - CI/CD integration for Unity builds.
    - Flaky test detection and alerting.

Deliver your proposal as a **short document**.

The ideas that I have as a senior QA are aimed at three major areas that can enhance the process of testing

i. Managing complex game test data
ii. Ensuring environment stability
iii. Improving test reliability in CI

Automated test data generation:

Features like token system (visbility x consumption) and army composition require testing a huge number of combinatorial states. Manual setting up specific token/army configuration for API and E2E tests is time consuming and error prone.

Proposal:

Implementation of dedicated data generation library (e.g. faker.js or Python faker) or custom JSON/YML template.

1. Modelling game constraints: Generate armies with realistic data (attack, defense, speed) and token configuration based on: generate x enemy armies, x with TIME-BASED tokens, x with BATTLE=BASED tokens.
2. Initialization state: Integrating this factory setup into test setup method (fixtures) to quickly access to database or game server to test complex each test run.

Impact:

i. It reduces the time of engineer to spend time in scripting data creation.
ii. Easier to cover boundary values and edge-cases.

Process Improvements: CI/CD “Isolated Lane” for flaky tests

Problem:

When build is failed, developers time wasted for debugging/investigating and finding issue. Without any code changes flaky tests passed or failed which slow down the CI/CD pipeline.

For encountering this issue:

Isolated lane: To identify a “flaky” test after three consecutive failures, introduce a mechanism (CI tool e.g. Jenkins/Circle CI/Gitlab CI) The failed tests are moved to isolated lane or non blocking stage.

Dedicated daily report: By setting daily dashboard that reports only the status of isolated tests. Assigning it to team members/Senior QA/Lead on daily basis to prioritize and fix in same lane which ensure that main workflow is not blocked.

Impact:
i. The improved pipeline stability would impact faster for developers and receives faster feedback on their commits.

ii.This would also impact on QA effort by improving quality of test suite itself and fixing unreliable tests.

API Reporting Dashboard:

To get the real time monitoring into health and performance of CORE API's and environments (Dev, staging, Prods) to catch issues before they effect players or development teams.

API Status Dashboards:

I will leverage Visualization tool like Grafana integrated into API testing framework and monitoring stack (e.g. Datadog) to create critical views.

i. API Test health: A realtime view showcasing the pass/fail rate and average latency of the critical API regression suite (e.g. token configuration endpoints, Friend lists API)
ii. Environment Stability: Dashboard measuring key metrics across environments. Latency spikes, error rates (5xx, 4xx errors) and CPU/Memory consumption.

Impact:

i. Catching the performance degradation and environment issues (e.g. memory leakage in server)
ii. Providing clear metrics