Multi-agent AI Testing for Retail Enterprises

Introduction:

Mindfire developed a Multi-Agent AI Testing System for a leading US-based retail company to enable continuous, intelligent testing of its enterprise application. The platform supports critical business functions including order management, patient services, point-of-sale (POS), reporting, and ERP integration.

Prior to implementation, every requirement created in Azure DevOps demanded several hours of manual effort to author test cases, build test plans, and create BDD feature files, These tasks required in-depth knowledge of over 2,000 existing automation step definitions. To eliminate this bottleneck, Mindfire introduced a Multi-Agent AI Testing System comprising of four specialized AI agents. These are operated by the Manual QA team and powered by GitHub Copilot / Claude Code, and deterministic Node.js scripts.

The agents now automatically generate manual test cases, create Azure DevOps test plans, produce BDD Gherkin feature files, and generate live test data. The resulting feature files are then routed to the automation repository's PendingFeatures folder, where Mindfire's SDET team implements, validates and executes them.

This AI-driven workflow reduced the end-to-end effort per feature from 4–6 hours to approximately 35–45 minutes, significantly accelerating test design, automation readiness, and release cycles.

Client details:

Name: Confidential | **Industry**: Retail | **Location**: USA

Technologies:

QA Automation & AI Enablement

Node.js, GitHub Copilot / Claude Code, steering documents; Azure DevOps, Confluence, Figma; C# / .NET 8, SpecFlow (BDD), NUnit, Selenium WebDriver, Page Object Model; Microsoft SQL Server.

Multi-agent AI Testing for Retail Enterprises

Problem Statement:

In its existing state, the application was sluggish and sub-optimal.

- Creating manual test cases from Azure DevOps work items was time-consuming, often taking hours per feature, with output quality varying across engineers.
- Building test plans and linking test cases in Azure DevOps was a manual, error-prone process, frequently resulting in incorrect iterations, QA assignments, or suite mappings.
- Authoring BDD feature files required familiarity with over 2,108 C# step definitions, limiting manual step reuse to just 30–50%.
- Generating test data involved manually walking the full UI flow, taking 30–60 minutes per dataset, repeated across eight environments.
- Requirement-to-test-data traceability was inconsistent and largely dependent on individual engineer diligence.

Project Description:

This Multi-Agent AI Testing System is built around four specialized AI agents. Each is powered by either GitHub Copilot or Claude Code, guided by steering documents that encode the team's standards, and backed by Node.js scripts that handle deterministic work: API calls, step matching, linting and excel generation. This ensures that the AI is never left to guess. The agents form a traceable pipeline:

- **Agent 1:** Test Case Generator: pulls the work item (and optional BRD/Figma), generates positive, negative, edge, UX and data-validation cases, then runs a deterministic lint pass and a 3-persona AI review (Senior BA, PO and QA Lead) before exporting an Excel workbook.
- **Agent 2:** Test Plan Creator: reads the approved Excel and creates the Azure DevOps test plan, suites and test-case work items, with pre-flight checks for the Automation tag and duplicate plans.
- **Agent 3:** Feature File Generator: generates BDD Gherkin .feature files, matching steps against 2,108 C# bindings for 90%+ reuse, and pushes the result to the repository's PendingFeatures folder for the SDET team.
- **Agent 4:** Test Data Generator: a standalone helper that executes SpecFlow/Selenium tests to create real data — orders, patients, appointments, across 8 environments.

Multi-agent AI Testing for Retail Enterprises

The solution spans both Manual QA and SDET teams. Manual QA manages the AI agents and reviews the generated output. The SDET team owns the SpecFlow framework, picks up the generated feature files from the PendingFeatures folder, implements any genuinely new steps, and runs the suite. Throughout the process, Azure DevOps remains the single source of truth, ensuring end-to-end traceability of every artifact back to the original work item.

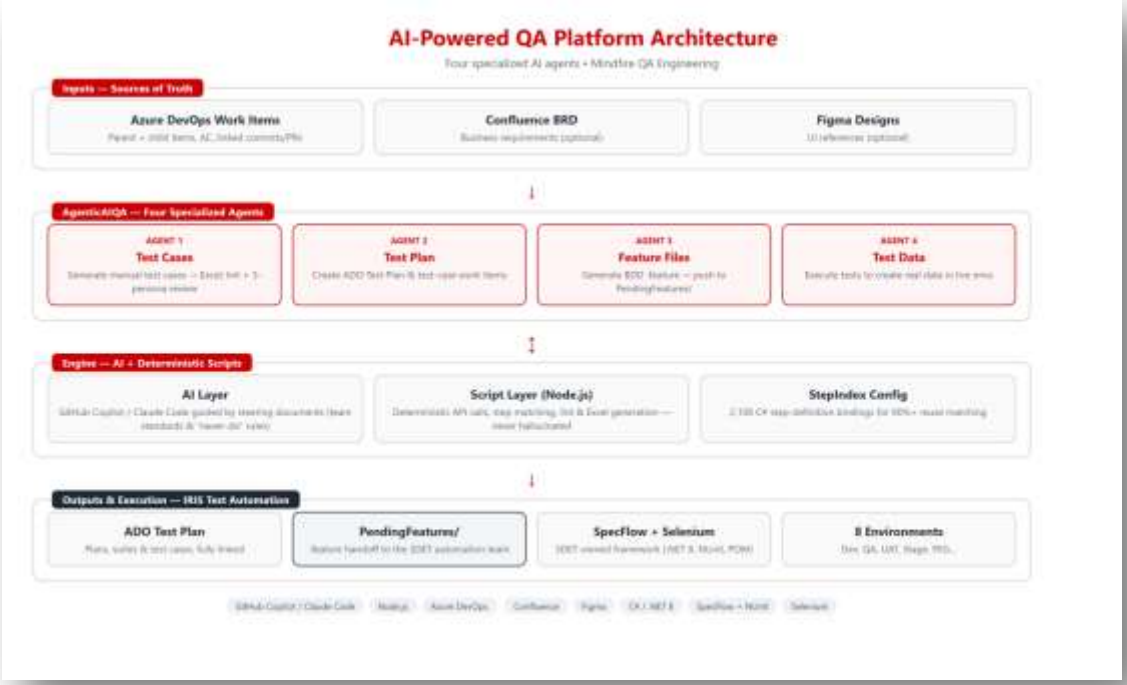
Methodology:

Mindfire's approach keeps AI output consistent and trustworthy by separating deterministic work from creative work:

- Steering documents: markdown rule-sets encode naming conventions, tag rules and explicit "never-do" rules, constraining non-deterministic AI to the team's standards.
- Deterministic scripts: Node.js handles API calls, step matching, lint validation and Excel generation; the AI only fills in genuinely new steps and never invents test data.
- Two-pass review: a deterministic validate-and-lint pass must be clean before a 3-persona AI review (Senior BA, PO and QA Lead) checks coverage and judgment-call gaps.
- Quality gates: the Automation tag plus pre-flight checks (tag readiness and a duplicate-plan guardrail) stop bad input before it reaches Azure DevOps or the automation repo.
- Human review & traceability: engineers approve Agent 1's Excel and Agent 3's feature file before any push, and every artifact links back: work item → test cases → test plan → feature file → test data.

Multi-agent AI Testing for Retail Enterprises

Architecture :



Workflow :

Figure 2. End-to-end workflow — the manual QA team runs Agents 1–3, then hands the generated feature files to the SDET automation team via the PendingFeatures folder.

Multi-agent AI Testing for Retail Enterprises

Screenshots:

The following illustrative views show the chain in action — from a test case in Azure DevOps, to the feature file an agent generates and delivers to PendingFeatures, to the results of an automated regression run.

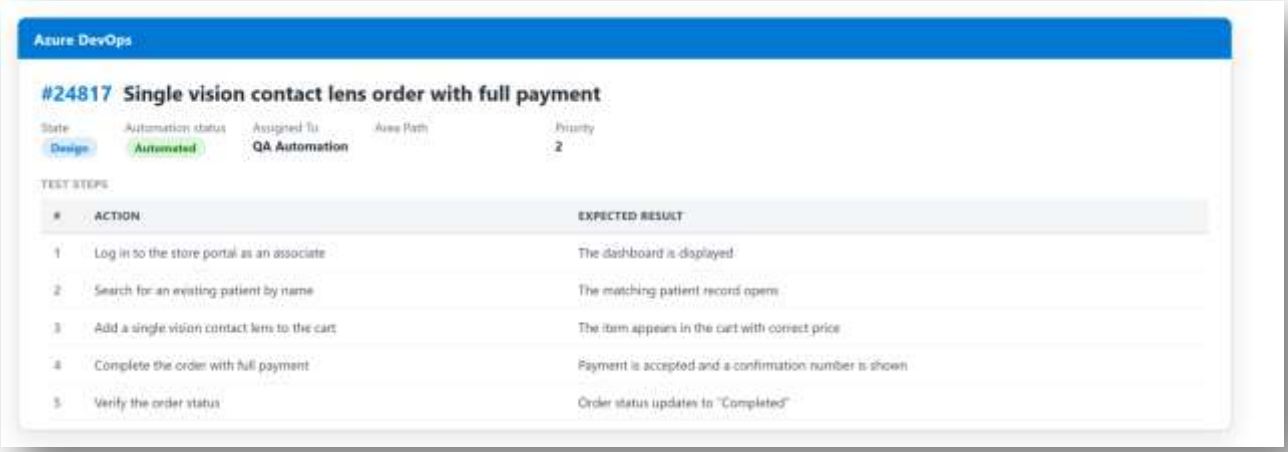


Figure 3. The source test case in Azure DevOps, created by Agent 2 with automation status synced back.

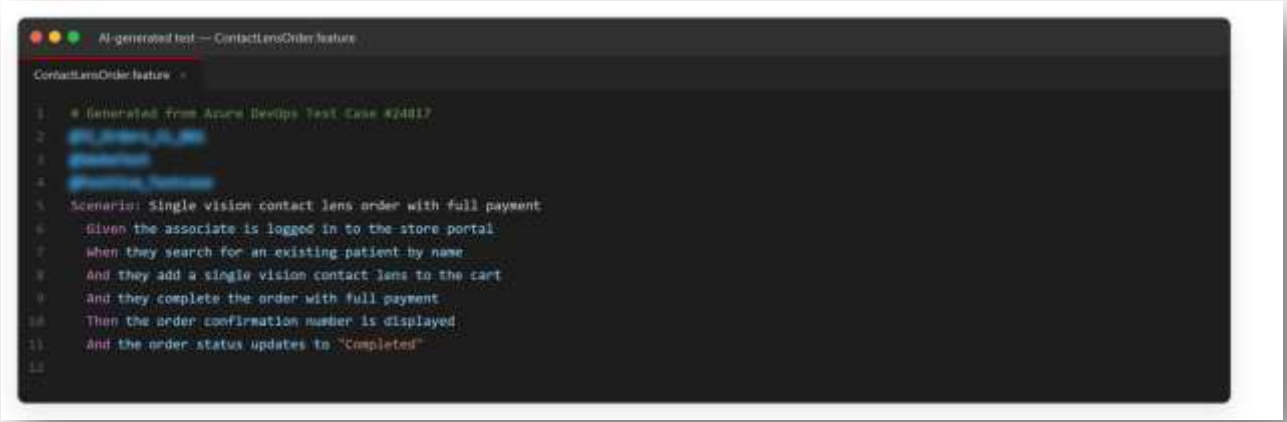


Figure 4. The BDD feature file Agent 3 generates and delivers to the PendingFeatures folder, with 90%+ reused steps.

Multi-agent AI Testing for Retail Enterprises

Figure 5. A nightly Selenium regression run executed by the SDET team — results captured and reported.

Key Outcomes & Business Impact:

- **Dramatically faster:** end-to-end effort per feature fell from 4–6 hours to 35–45 minutes, an 80–90% reduction.
- **High reuse, Low maintenance:** automated multi-tier matching against 2,108 C# bindings lifts step reuse from 30–50% to 90%+, keeping the suite DRY.
- **Consistent quality:** steering documents, deterministic linting and a 3-persona AI review enforce standards on every run, regardless of who triggers it.
- **Zero ADO configuration errors:** common manual mistakes (wrong iteration, QA field, suite visibility, duplicate plans) are eliminated by the master script and pre-flight guardrails.
- **Test data in minutes:** Agent 4 creates real data across 8 environments in 3–5 minutes per set instead of 30–60 minutes of manual UI clicking.
- **Full traceability & lower barrier:** 100% acceptance-criteria coverage with a full work-item-to-test-data chain, and any QA engineer can trigger the agents. There is no senior step-definition expertise required.