Q-RAVENS

Autonomous QA Agent Swarm

Product Requirements Document

Version	1.0
Date	January 2026
License	Open Source (MIT)
Status	Draft

Table of Contents

1. Executive Summary

1.1 Vision

Q-Ravens is an open-source, autonomous multi-agent system designed to democratize web application testing. By leveraging a swarm of specialized AI agents, Q-Ravens enables any user, regardless of technical expertise, to conduct comprehensive quality assurance testing through simple natural language instructions.

The name "Ravens" symbolizes intelligence, adaptability, and collaborative problem-solving, qualities that define our agent architecture.

1.2 Problem Statement

Current challenges in web application testing include:

- High barrier to entry: Testing requires specialized technical knowledge
- Time-intensive: Manual test creation and execution consume significant resources
- Fragmented tooling: Teams must juggle multiple disconnected testing tools
- Limited accessibility: Non-technical stakeholders cannot participate in QA processes
- Scaling difficulties: Expanding test coverage requires proportional human effort

1.3 Solution

Q-Ravens provides a team of autonomous AI agents that work together like a professional QA department. Users simply describe what they want tested in plain English, and the agent swarm handles analysis, test design, execution, and reporting automatically.

1.4 Key Differentiators

- Natural language interface: No coding or technical knowledge required
- Autonomous decision-making: Agents think, plan, and adapt independently
- Multi-LLM support: Users choose their preferred AI providers
- Open source: Fully transparent, customizable, and community-driven
- Professional-grade output: Enterprise-quality testing accessible to everyone

2. Product Overview

2.1 Core Concept

Q-Ravens operates as a virtual QA team where each agent has a specialized role. The system uses LangGraph for orchestration, enabling complex decision loops, state persistence, and human-in-the-loop capabilities.

2.2 User Experience Philosophy

The fundamental principle is delegation without friction. Users should feel like they are assigning tasks to a capable team of employees, not operating software. Key UX principles include:

- Zero-configuration start: Provide a URL, get comprehensive testing
- Conversational interaction: Chat naturally, ask questions, give feedback
- Transparent progress: Always know what agents are doing and why
- Actionable results: Reports designed for decision-making, not data dumps

2.3 Target Users

User Type	Use Case	Value Proposition
Product Managers	Validate features before release	Test without waiting for QA team
Startup Founders	Ensure quality with limited resources	Enterprise QA at startup cost
QA Engineers	Accelerate test creation	Focus on strategy, not scripting
Developers	Quick regression checks	Catch bugs before PR review
Business Analysts	Verify requirements implementation	Direct validation without intermediaries

3. System Architecture

3.1 High-Level Architecture

Q-Ravens follows a hierarchical multi-agent architecture with a central orchestrator coordinating specialized agents. The system is built on LangGraph for stateful workflow management.

Architecture Diagram

The system consists of three layers:

- User Interface Layer: Chat interface, file upload, report delivery
- Orchestration Layer: Central coordinator managing agent delegation and state
- Agent Layer: Specialized agents (Analyzer, Designer, Executor, Reporter, DevOps)

3.2 Technology Stack

Component	Technology	Rationale
Agent Framework	LangGraph	Stateful workflows, cycles, checkpointing, human-in-the-loop
Browser Automation	Playwright	Modern, reliable, cross-browser, excellent API
Performance Testing	Lighthouse	Industry standard, comprehensive metrics
Accessibility Testing	Axe-core	WCAG compliance, detailed reporting
Security Scanning	OWASP ZAP	Open source, comprehensive vulnerability detection
Vector Memory	ChromaDB	Simple setup, good performance, free
State Persistence	PostgreSQL	Robust, scalable, excellent tooling
Observability	LangSmith / Langfuse	Agent tracing, debugging, evaluation
User Interface	Streamlit / Gradio	Rapid development, Python native

3.3 LLM Provider Support

Q-Ravens supports multiple LLM providers, allowing users to choose based on preference, cost, or privacy requirements.

Provider	Models	Best For	Cost
Anthropic	Claude Sonnet 4, Opus 4	Complex reasoning, code	Pay per token
OpenAI	GPT-4o, GPT-4o-mini	General purpose	Pay per token
Google	Gemini 2.0	Multimodal tasks	Pay per token
Groq	Llama 3.1 70B	Fast inference	Free tier available
Ollama (Local)	Llama 3, Mistral	Privacy, offline	Free (self-hosted)

Azure OpenAI	GPT-4 variants	Enterprise compliance	Enterprise pricing

4. Agent Specifications

4.1 Orchestrator Agent (Project Manager)

The central coordinator responsible for understanding user intent, delegating tasks, managing workflow, and communicating results.

Responsibilities

- Parse and interpret user requests (natural language, URLs, documents)
- Create and maintain test project plans
- Delegate tasks to specialist agents
- Track progress and handle blockers
- Request user clarification when needed
- Compile and deliver final reports

Capabilities

- Context understanding across conversation history
- Priority management and resource allocation
- Conflict resolution between agent recommendations
- User preference learning and adaptation

4.2 Analyzer Agent (QA Analyst)

Responsible for understanding the application under test through automated exploration and analysis.

Responsibilities

- Crawl and map website structure
- Identify interactive elements (forms, buttons, links)
- Detect technology stack and frameworks
- Discover authentication and user flows
- Assess accessibility structure
- Generate application model documentation

Tools

- Playwright crawler for DOM exploration
- Network traffic analyzer
- Technology detection (Wappalyzer-style)
- Sitemap parser

4.3 Designer Agent (Test Architect)

Creates comprehensive test strategies and detailed test cases based on analysis and requirements.

Responsibilities

- Generate test scenarios from requirements
- Design test data sets
- Prioritize tests by risk and business impact
- Create BDD-style specifications (Gherkin)
- Design edge cases and negative tests
- Identify cross-browser and device coverage

Outputs

- Test Plan Document
- Test Case Specifications
- Test Data Requirements
- Coverage Matrix

4.4 Executor Agent (Automation Engineer)

Implements and runs automated tests, capturing evidence and handling dynamic scenarios.

Responsibilities

- Generate Playwright test code from specifications
- Execute tests in headless browsers
- Handle dynamic elements and waits
- Capture screenshots, videos, and logs
- Implement retry logic for flaky tests
- Report pass/fail with detailed evidence

Capabilities

- Multi-browser execution (Chrome, Firefox, Safari, Edge)
- Mobile viewport emulation
- Network interception and mocking
- Visual regression detection

4.5 Reporter Agent (QA Lead)

Compiles results into actionable reports for different audiences.

Responsibilities

- Generate executive summaries (non-technical)
- Create detailed technical reports
- Produce visual dashboards
- Track trends across test runs
- Highlight critical issues and blockers

- Recommend next steps and priorities

Output Formats

- PDF executive reports
- HTML interactive dashboards
- Markdown technical documentation
- Slack/Teams notifications

4.6 DevOps Agent (Infrastructure)

Manages test environments and infrastructure concerns.

Responsibilities

- Provision test environments
- Manage browser containers
- Handle CI/CD integration
- Schedule recurring test runs
- Monitor resource usage

Integrations

- Docker for containerized execution
- GitHub Actions for CI/CD
- Kubernetes for scaling (optional)

5. Functional Requirements

5.1 User Input Handling

ID	Requirement	Description	Priority
FR-001	URL Input	Accept single URL and begin comprehensive analysis	Must Have
FR-002	Natural Language	Parse plain English test instructions	Must Have
FR-003	Document Upload	Accept requirements docs (PDF, DOCX, MD)	Must Have
FR-004	Sitemap Import	Parse XML sitemaps for test scope	Should Have
FR-005	API Spec Import	Parse OpenAPI/Swagger for API testing	Should Have

5.2 Test Execution

ID	Requirement	Description	Priority
FR-010	Functional Tests	Execute UI interaction tests	Must Have
FR-011	Accessibility Tests	WCAG 2.1 AA compliance checking	Must Have
FR-012	Performance Tests	Lighthouse audits and Core Web Vitals	Must Have
FR-013	Security Scans	Basic vulnerability detection	Should Have
FR-014	Visual Regression	Screenshot comparison across runs	Should Have
FR-015	Cross-Browser	Chrome, Firefox, Safari, Edge support	Should Have

5.3 Agent Collaboration

ID	Requirement	Description	Priority
FR-020	Inter-Agent Comms	Agents can request info from each other	Must Have
FR-021	Human Escalation	Agents can pause and ask user for input	Must Have
FR-022	Approval Gates	Require human approval for sensitive actions	Must Have

FR-023	Shared Memory	Agents share context and learnings	Must Have
FR-024	Retry Logic	Automatic retry with backoff for failures	Must Have

6. Non-Functional Requirements

6.1 Performance

- Analysis of a typical website (50 pages) should complete within 5 minutes
- Test execution should run in parallel where possible
- System should handle 100+ concurrent test cases
- Report generation should complete within 30 seconds

6.2 Scalability

- Horizontal scaling via containerization
- Support for distributed test execution
- Queue-based task management for large test suites

6.3 Reliability

- Checkpoint system for long-running tests
- Automatic recovery from transient failures
- State persistence across system restarts
- 99.5% uptime target for hosted deployments

6.4 Security

- API keys stored encrypted at rest
- No test data transmitted to third parties (except chosen LLM)
- Support for air-gapped deployments with local LLMs
- Audit logging for all agent actions

6.5 Usability

- Zero-configuration quick start
- Intuitive chat interface
- Clear progress indicators
- Helpful error messages with suggested actions

7. Implementation Roadmap

7.1 Phase Overview

Phase	Duration	Focus	Deliverable
Phase 1	Weeks 1-2	Foundation	Basic 3-agent system
Phase 2	Weeks 3-4	Intelligence	Full agent suite + tools
Phase 3	Weeks 5-6	Polish	UI + memory + reports
Phase 4	Weeks 7-8	Production	API + CI/CD + scaling

7.2 Phase 1: Foundation (Weeks 1-2)

Goals

- Establish LangGraph workflow structure
- Implement core agents: Orchestrator, Analyzer, Executor
- Integrate Playwright for basic browser automation
- Create CLI interface for testing

Success Criteria

- User can provide URL and receive basic test results
- Agents communicate via shared state
- Basic pass/fail reporting functional

7.3 Phase 2: Intelligence (Weeks 3-4)

Goals

- Add Designer and Reporter agents
- Implement conditional routing and retry logic
- Integrate Lighthouse, Axe-core tools
- Add checkpointing and state persistence

Success Criteria

- Full test lifecycle automated
- Human-in-the-loop approval gates working
- Tests can resume from checkpoints

7.4 Phase 3: Polish (Weeks 5-6)

Goals

- Build chat UI (Streamlit/Gradio)
- Implement vector memory for learning

- Create PDF/HTML report generation
- Add LangSmith integration for debugging

Success Criteria

- Non-technical users can operate system
- Reports suitable for stakeholder review
- System learns from past test runs

7.5 Phase 4: Production (Weeks 7-8)

Goals

- Deploy API for programmatic access
- Add CI/CD integration (GitHub Actions)
- Implement scheduled/triggered runs
- Performance optimization and scaling

Success Criteria

- API fully documented and functional
- Can integrate with existing CI pipelines
- Handles enterprise-scale test suites

8. Open Source Strategy

8.1 License

Q-Ravens will be released under the MIT License, chosen for maximum adoption and contribution potential while maintaining attribution requirements.

8.2 Repository Structure

The project will follow a monorepo structure:

- q-ravens/core: Agent framework and orchestration
- q-ravens/agents: Individual agent implementations
- q-ravens/tools: Test automation tool integrations
- q-ravens/ui: User interface components
- q-ravens/docs: Documentation and examples

8.3 Contribution Guidelines

- Clear CONTRIBUTING.md with code standards
- Issue templates for bugs, features, and agents
- Pull request template with checklist
- Code of conduct for community interactions
- Regular community calls and roadmap discussions

8.4 Documentation

- Getting Started guide (5-minute quickstart)
- Architecture deep-dive
- Agent development guide (for contributors)
- Tool integration guide
- API reference
- Video tutorials

9. Success Metrics

9.1 Product Metrics

Metric	Target (6 months)	Target (12 months)
GitHub Stars	1,000	5,000
Monthly Active Users	500	2,500
Community Contributors	25	100
Test Runs Executed	10,000	100,000

9.2 Quality Metrics

- Bug detection rate: 85%+ of manually-found bugs also caught by Q-Ravens
- False positive rate: Below 10%
- User satisfaction: 4.5/5 average rating
- Time to first test: Under 2 minutes from URL input

10. Risks and Mitigations

Risk	Impact	Likelihood	Mitigation
LLM API costs escalate	High	Medium	Support local LLMs, implement caching, optimize prompts
Agent hallucinations	High	Medium	Verification steps, human approval gates, structured outputs
Complex sites break automation	Medium	High	Robust retry logic, fallback strategies, user escalation
Security vulnerabilities	High	Low	Security audits, sandboxed execution, minimal permissions
Low community adoption	Medium	Medium	Strong documentation, example projects, active engagement

11. Appendix

11.1 Glossary

Term	Definition
LangGraph	A framework for building stateful, multi-agent applications using graph-based orchestration
Agent	An autonomous AI system that can perceive, decide, and act to achieve goals
Orchestrator	The central agent that coordinates work among specialist agents
Checkpoint	A saved state of workflow execution that enables pause/resume
Human-in-the-loop	Pattern where AI systems pause for human input or approval
Vector Memory	Storage system using embeddings to enable semantic search over past data
Playwright	Modern browser automation framework by Microsoft
WCAG	Web Content Accessibility Guidelines - international accessibility standards

11.2 References

- LangGraph Documentation: https://langchain-ai.github.io/langgraph/
- Playwright Documentation: https://playwright.dev/
- Axe-core: https://github.com/dequelabs/axe-core
- Lighthouse: https://developer.chrome.com/docs/lighthouse/
- OWASP ZAP: https://www.zaproxy.org/

11.3 Document History

Version	Date	Author	Changes
1.0	Jan 2026	Initial Draft	Initial PRD creation

— End of Document —