Agentic Test Case Generator Tool — Documentation

1. Overview

- Product Name: Agentic Test Automation for Healthcare
- Version: 1.0.0
- Release Date: 2024-07-25
- High-level description: An AI-powered, agentic testing assistant that generates manual test cases, automation scripts, and performance plans. It integrates with project management tools like Jira and Notion and understands project context through uploaded documents to produce traceable, compliant, and executable test assets for healthcare applications.
- Key Value Proposition:
    - Autonomous Agentic Workflow: Automatically triggers a complete testing asset generation pipeline when a feature is marked as ready for testing.
    - Tri-Modal Asset Generation: Creates three critical testing assets in a single run: manual test cases, executable automation scripts, and strategic performance test plans.
    - Context-Aware Generation: Leverages Retrieval-Augmented Generation (RAG) by ingesting project manuals and technical documents to produce highly relevant and accurate test cases.
    - Built-in Compliance and Traceability: Automatically maps generated tests to healthcare standards (HIPAA, FHIR) and provides a clear traceability matrix from requirements to test cases.
    - Role-Based Access Control: Provides distinct views and permissions for Testers, Compliance Auditors, and Admins to ensure a secure and organized workflow.
    - Multi-Format Export: Allows for easy integration into existing workflows by exporting test cases to JSON, Gherkin, Excel, and Postman formats.
- Target Users: Testers, QA Leads, Developers, Product Managers, Compliance Auditors.

---

2. System Architecture

- Diagram: The application includes a built-in system architecture viewer that provides a visual block diagram of its components and data flows, showcasing the relationships between the UI, API, AI services, and data stores.
- Components:
    - Frontend: A responsive reviewer dashboard built with React for reviewing,

managing, and exporting test assets.

- Backend: A set of FastAPI APIs and webhooks that handle business logic, data processing, and communication with external services.
- AI Orchestration: A core engine that uses a Planner, Generator, and Validator agent pattern, powered by Gemini (Vertex AI), to manage the logic of asset generation. It leverages RAG for contextual understanding.
- Vector Store: Pinecone or Vertex AI Matching is used to store document embeddings for the RAG workflow, enabling semantic search over project documentation.
- Database: A combination of a relational database (e.g., Cloud SQL) for transactional data (users, audit logs) and object storage (e.g., GCS) for documents.
- Integrations: Connectors for project management tools including Jira, Asana, Notion, and Confluence to fetch requirements and post updates.
- Notifications: An integrated service for sending notifications via Email (SMTP/SES) and posting comments directly to Jira tickets.
- Compliance: Built-in modules for PHI redaction before LLM processing and for mapping test artifacts to HIPAA/FHIR compliance controls.

---

3. Features

- Agentic Monitoring Engine: A dashboard that monitors ticket statuses from integrated PM tools and automatically triggers the test generation workflow when a ticket moves to the "Test" state.
- Tri-Modal Asset Generation: The core capability to generate three types of test assets simultaneously:
  - Manual: Detailed test cases with steps, expected results, and types (Positive, Negative, Security, etc.).
  - Automation: Executable test scripts (e.g., Selenium with Python) ready for integration into a CI/CD pipeline.
  - Performance: A high-level test plan including scenarios (load, stress), key metrics, and recommended tools.
- Contextual Document Ingest: Users can upload project manuals (.txt, .md, .pdf), which the agent ingests to understand project-specific terminology, user flows, and technical constraints.
- Compliance & Traceability:
  - Compliance Report: An auto-generated report that maps generated test cases against a checklist of healthcare compliance controls (e.g., HIPAA Access Control).
  - Traceability Matrix: A clear view linking requirements directly to the test cases that cover them.

- Role-Based Dashboards: The UI adapts based on the user's role (Admin, Tester, Compliance Auditor), showing only relevant views and actions.
- Multi-Format Export: Test cases can be exported in multiple formats (JSON, Gherkin, Excel, Postman) to fit different testing toolchains.
- In-App Architecture Viewer: A visual diagram of the system's architecture for educational and onboarding purposes.
- User Management & Audit: Admins can manage users and roles, while all significant user actions are captured in an immutable audit log.

---

4. Installation & Deployment

Local Dev Setup

- Prerequisites: Python 3.9+, Node.js 20+, Docker, GCP SDK.
- Clone repo & install dependencies:

  git clone [repo-url]

  cd [repo-folder]/frontend && npm install

  cd ../backend && pip install -r requirements.txt
-
- Configure API Key: In the frontend/ directory, create a config.js file and add your Gemini API key:

  ```
  // config.js
  window.APP_CONFIG = {
    API_KEY: "YOUR_GEMINI_API_KEY_HERE"
  };
  ```
- Run backend: uvicorn app.main:app --reload
- Run frontend: npm run dev

Cloud Deployment

- GCP requirements: Cloud Run (for frontend and backend services), Secret Manager (for API keys), Cloud SQL, Vertex AI (for Gemini models and Matching Engine), Pub/Sub (for asynchronous tasks), Cloud Storage.
- Terraform/Deployment Manager scripts: Infrastructure as Code scripts are used to provision and manage cloud resources.
- CI/CD pipeline setup: A pipeline (e.g., using GitHub Actions or Cloud Build) is configured to automatically build, test, and deploy the application upon code changes.

---

5. Configuration

- Environment variables (Backend): DATABASE_URL, JIRA_CLIENT_ID, VERTEX_PROJECT_ID, SMTP_SETTINGS.
- Environment variables (Frontend): The API_KEY for Gemini is configured in config.js.
- Secret storage: All sensitive credentials and API keys are stored in GCP Secret Manager and accessed by services at runtime.
- Integration setup: API tokens for Jira and Notion are managed via the in-app "Integration Settings" modal, where they are saved to the user's local browser storage.

---

6. User Manual

The application includes a comprehensive user manual. Please refer to USER_MANUAL.md for a step-by-step guide on logging in, using the agentic workflow, reviewing assets, and administration.

---

7. API Reference

- Authentication model: The backend uses a JWT-based authentication model. Users log in to receive a token that must be included in the header of subsequent requests.
- Core endpoints:
    - POST /connect: Establishes and tests a connection to an external PM tool using a provided API token.
    - POST /ingest: Handles the upload and processing of a project manual for RAG context.
    - POST /webhook/jira: A webhook endpoint that listens for status changes on Jira tickets to trigger the agentic workflow.
    - GET /artifacts/{project_id}: Fetches all generated artifacts (test cases, scripts, plans) for a given project or requirement.
-
- Response formats with examples: The API primarily uses JSON. For test case generation, the response adheres to a specific schema:

  Code: JSON

```
{
  "testCases": [
    {
      "id": "TC-001",
```

```
    "requirementId": "REQ-01",
    "title": "Verify Patient Record Access",
    "description": "Ensure only authorized users can access patient records.",
    "type": "Security",
    "steps": [
     { "action": "Log in as a nurse", "expectedResult": "Patient records are accessible." }
    ]
   }
  ]
}
```


---

8. Workflows

- Sequence diagrams for:
    - Jira ticket → script generation:
        1. A developer moves a Jira ticket to "In Test".
        2. Jira sends a webhook to the POST /webhook/jira endpoint.
        3. The backend API places a generation job in a queue (e.g., Pub/Sub).
        4. The AI Orchestration service picks up the job, fetches ticket details, and triggers the Generator agent.
        5. The agent generates manual tests, an automation script, and a performance plan using Gemini.
        6. The assets are stored in the database.
        7. The user receives a notification, and a comment is posted back to the Jira ticket.
    - Document ingest → RAG → test case generation:
        1. A user uploads a project manual via the UI.
        2. The frontend sends the document to the POST /ingest endpoint.
        3. The backend chunks the document, generates embeddings, and stores them in the Vector DB.
        4. During generation, the AI Orchestration's Planner agent identifies that context is needed.
        5. It queries the Vector DB for relevant document chunks based on the

requirement text.

6. The retrieved context is passed to the Generator agent along with the primary prompt, resulting in context-aware test cases.

- Export → download:
    1. The user clicks an "Export" button in the UI.
    2. The frontend fetches the relevant test case data.
    3. A client-side service (exportService.ts) formats the JSON data into the desired format (e.g., Gherkin text, Excel binary).
    4. The formatted content is converted into a Blob, and a download link is programmatically created and clicked to save the file to the user's machine.

---

9. Compliance & Security

- Data handling policies: All data containing potential PHI is processed in a secure environment. PHI is redacted before being sent to any third-party LLM APIs.
- PHI redaction logic: A backend service uses a combination of regular expressions and Named Entity Recognition (NER) models to identify and mask PHI (names, addresses, IDs) in user-provided requirement text and documents.
- HIPAA/FHIR mapping tables: The application maintains an internal list of key HIPAA and FHIR controls. The Compliance Report functionality maps these controls to generated test cases based on their type (e.g., 'Security' tests provide evidence for 'Access Control').
- Audit logs & traceability features: The application provides an immutable audit log tracking all user actions and a traceability matrix that visually links requirements to the test cases that validate them.

---

10. Troubleshooting & FAQ

- Common issues:
    - Integration Errors: Ensure API keys are correct and have the necessary permissions in the source tool (e.g., Notion).
    - Generation Failed: This can be due to an invalid or expired Gemini API key, or if the prompt violates safety policies. Check your key and simplify the requirement text.
    - Missing Notifications: For email, this is a simulated mailto: link. Ensure you have a default email client configured.
- Error codes and resolutions:

    - 401 Unauthorized: Your Gemini API key is invalid or missing.
    - 400 Bad Request: The requirement text may be empty or malformed.
    - 503 Service Unavailable: The AI service is temporarily down or overloaded. Please try again later.
- Performance tuning tips:
    - Provide high-quality, detailed project manuals for the best contextual results.
    - Write clear, unambiguous requirements in your PM tool tickets.
    - For very large, complex features, break them down into smaller user stories for more focused test generation.

---

11. Release Notes & Roadmap

Current release highlights (v1.0.0):

- Initial release featuring the core Agentic Workflow.
- Support for Tri-Modal Asset Generation (Manual, Automation, Performance).
- Contextual awareness via .txt and .md document uploads.
- Full suite of reports: Compliance, Audit Log, and Traceability.
- Role-based access control for Admin, Tester, and Auditor roles.
- Mocked integration with Notion for demonstration purposes.

Known limitations:

- Jira integration is not yet fully implemented.
- Notifications are simulated (e.g., mailto: links).
- API calls to PM tools are currently mocked on the client-side.

Planned features:

- Full, backend-powered Jira and Notion integration.
- AI-powered explainability to describe why certain test cases were generated.
- Support for additional automation frameworks (e.g., Playwright, Cypress).
- Multi-agent orchestration for more complex testing scenarios.
- Real-time notification system via Slack or email.

---

12. Appendices

Glossary of terms:

- Agentic: A system that is autonomous, proactive, and capable of taking actions to achieve a goal.
- RAG (Retrieval-Augmented Generation): An AI technique that provides an LLM with external knowledge (retrieved from a document) to improve the accuracy and relevance of its responses.
- PHI: Protected Health Information.
- HIPAA: Health Insurance Portability and Accountability Act.
- FHIR: Fast Healthcare Interoperability Resources.

References to standards:

- HIPAA (Health Insurance Portability and Accountability Act)
- FHIR (Fast Healthcare Interoperability Resources)
- ISTQB (International Software Testing Qualifications Board)

External API documentation links:

- Google Gemini API: https://ai.google.dev/docs
- Jira Cloud API: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/
- Notion API: https://developers.notion.com/