AI Test Case Generator

The AI Test Case Generator is an intelligent, full-stack web application engineered to significantly enhance the Quality Assurance(QA) process. By leveraging Large Language Models (LLMs),it automates the creation of high-quality, context-aware test cases directly from feature requirements. This document provides a comprehensive overview of its features, architecture, installation, usage, and future development.

Overview: Intelligent Automation for QA

The AI Test Case Generator is more thana simple AI assistant; it's a dynamic, adaptive partner for QA teams. It streamlines the testing workflow by automatically generating detailed testcases, moving beyond manual processes and generic AI outputs. Its core strength lies in its ability to understand context and continuously learn, making it an invaluable asset for accelerating software development cycles.

Accelerated QA Process

Leverages LLMs to generate high-quality, context-aware test cases, significantly reducing manual effort and speeding up testing cycles.

Adaptive Intelligence

Goes beyond generic AI by incorporating a persistent Knowledge Base, a Feedback Loop for continuous learning, and an Iterative Refinement Chat.

Strategic QA Partnership

Transforms the test case generation process into a smart, adaptive collaboration tool, enhancing the efficiency and precision of any QA team.

Key Features: Empowering QA Teams

The AI Test Case Generator is packed with powerful features designed to provide unparalleled flexibility and intelligence in test case creation. From AI-powered generation to dynamic customization and continuous learning, each feature contributes to a more efficient and effective QA workflow.

AI-Powered Generation

Instantly generates test cases in various formats, including human-readable steps and Playwright automation scripts, ensuring comprehensive coverage.

Dynamic Options

Customizes output by selecting test type (Smoke, Regression, etc.), complexity, and the desired number of test cases for tailored results.

Context-Awareness

Utilizes both per-session context (uploaded documentation) and a persistent Knowledge Base for highly relevant and accurate test case generation.

Continuous Learning Loop

A Like/Dislike feedback system enables the AI to learn user preferences, continuously improving the style and quality of its output over time.

Iterative Refinement Chat

Provides a dedicated chat interface to make specific, conversational edits to any generated test case until it perfectly meets requirements.

Seamless Integrations

Connects with Jira for importing user stories and requirements, and Zephyr for sending generated tests back to test management platforms.

OCR Support

Uploads screenshots of UI or documents, extracting text using Tesseract.js to use as requirements for test case generation.

Utility Features

Includes convenient functionalities like copying all generated text, downloading as a .txt file, and clearing the screen for new sessions.

Context-Awareness & Continuous Learning

The true intelligence of the AI Test Case Generator lies in its ability to understand and adapt to specific project contexts and user preferences. This is achieved through a sophisticated two-pronged approach to context awareness and a robust feedback mechanism.

Context-Awareness

- **Per-Session Context:** Users can upload application code, design documents, or specific requirements for a single generation session. This ensures that the AI's output is highly relevant to the immediate task at hand, leveraging up-to-date information that isn't stored long-term.
- **Persistent Knowledge Base:** A dedicated Knowledge Base allows for the permanent storage of project standards, technical specifications, API documentation, and style guides. The AI references this "source of truth" for all future requests, ensuring consistent adherence to established guidelines and a deeper understanding of the project's domain. Need further improvement and upgradations,

Continuous Learning Loop

- **Feedback System:** A simple Like/Dislike mechanism empowers users to provide direct feedback on the generated test cases. This input is crucial for the AI's learning process.
- **Iterative Refinement Chat:**

  For more nuanced adjustments, users can open a dedicated chat for any generated test case. This allows for conversational edits, enabling precise modifications until the test case is perfectly tailored to specific needs, further enhancing the AI's understanding through direct interaction.

This combination of dynamic and persistent context, along with an active feedback loop, enables the AI to continuously improve its output, becoming an increasingly valuable and personalized tool for any QA team.

Architecture Overview

The AI Test Case Generator is built on a robust, scalable, and flexible client-server architecture. This design ensures efficient communication, reliable data management, and high-performance AI inference, providing a seamless user experience while handling complex generation tasks.

Frontend (Client)

A dynamic single-page application crafted with vanilla HTML, CSS, and JavaScript, handling all user interactions and communicating with the backend via RESTful API calls.

AI Service (Groq)

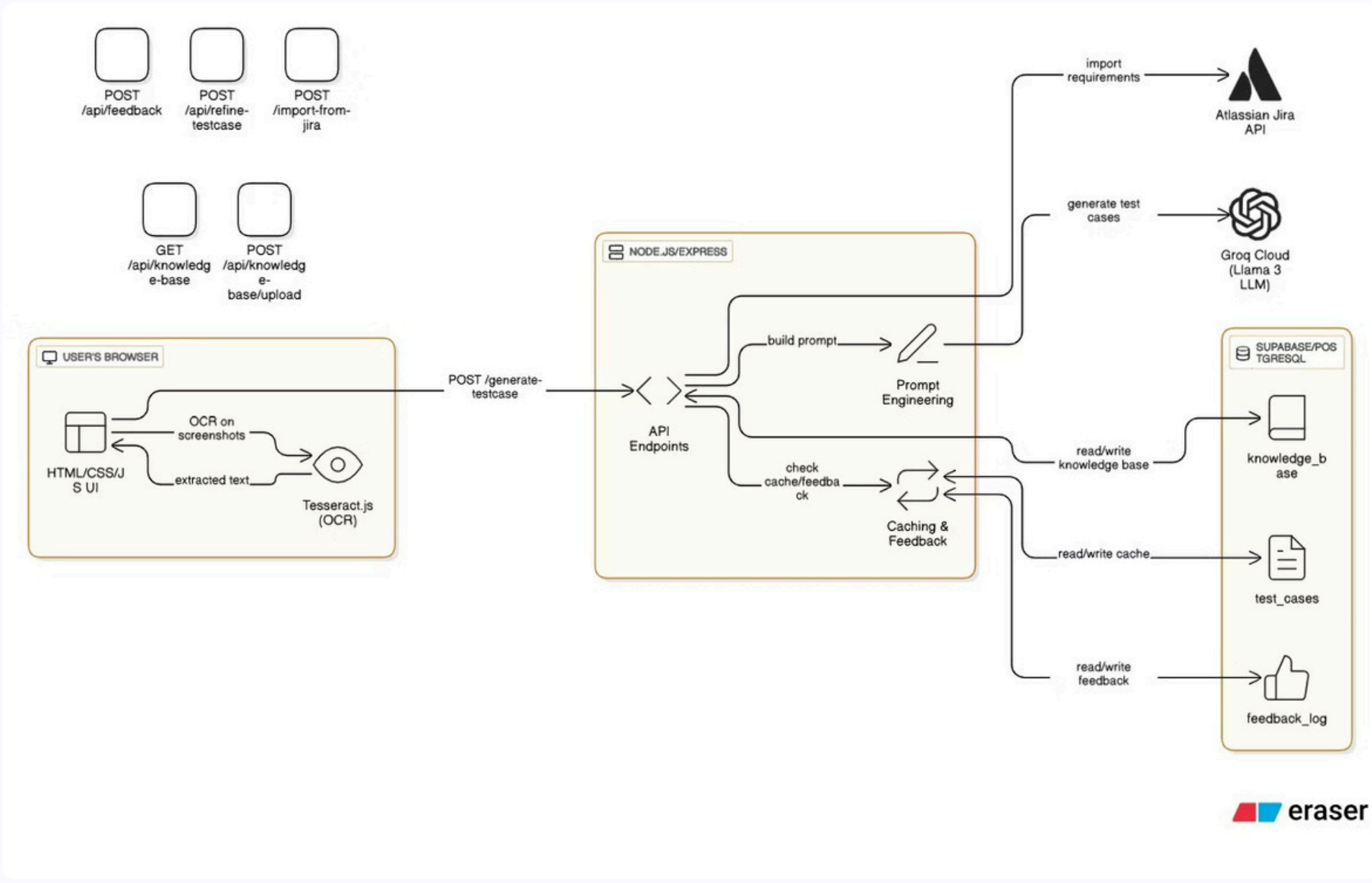
Leverages the Groq API for ultra-high-speed inference of the Llama 3 language model, which is the core engine for all test case generation and refinement capabilities.

Backend (Server)

A Node.js server powered by Express.js, serving as the central orchestrator. It manages API requests, constructs intelligent prompts, and coordinates interactions with the database and external services.

Data Layer (Supabase)

Utilizes a PostgreSQL database managed by Supabase for persistent data storage. This includes a cache for generated tests, a detailed user feedback log, and the comprehensive knowledge base.


POST /api/feedback
POST /api/refine-testcase
POST /import-from-jira
GET /api/knowledge-base
POST /api/knowledge-base/upload
USER'S BROWSER
HTML/CSS/JS UI
OCR on screenshots
extracted text
Tesseract.js (OCR)
POST /generate-testcase
NODE.JS/EXPRESS
API Endpoints
build prompt
Prompt Engineering
check cache/feedback
Caching & Feedback
import requirements
Atlassian Jira API
generate test cases
Groq Cloud (Llama 3 LLM)
read/write knowledge base
read/write cache
read/write feedback
SUPABASE/POSTGRESQL
knowledge_base
test_cases
feedback_log
eraser

Technology Stack: Powering Intelligent QA

The AI Test Case Generator is built upon a modern and efficient technology stack, carefully chosen to ensure high performance, scalability, and ease of development. Each component plays a crucial role in delivering a robust and responsive application.

Frontend Technologies

- **HTML5:** Provides the structural foundation for the web application.
- **CSS3:** Styles the application for a clean and intuitive user interface.
- **Vanilla JavaScript:** Powers the dynamic and interactive elements of the client-side.
- **Tesseract.js:** Enables Optical Character Recognition (OCR) for extracting text from images.
- **Prism.js:** Used for syntax highlighting of code snippets, especially useful for displaying generated Playwright scripts.

Backend, Database & AI

- **Node.js:** The JavaScript runtime environment for the server-side logic.
- **Express.js:** A fast, unopinionated, minimalist web framework for Node.js, used for building the API.
- **Axios:** A promise-based HTTP client for making API requests from the backend.
- **Multer:** Middleware for handling multipart/form-data, primarily used for file uploads (e.g., application context or screenshots).
- **Supabase (PostgreSQL):** The primary database for persistent storage, offering real-time capabilities and robust data management.
- **Groq API (Llama 3 Model):** The core AI service providing high-speed, state-of-the-art language model inference for test case generation.

Setup and Installation

Follow these steps to get the project running on your local machine.

Prerequisites

- **Node.js** (which includes npm)
- Git

Installation Steps

1.Clone the repository:

```
Bash
git clone [your-repository-url]
cd [your-project-folder]
```

2.Install backend dependencies:

```
 cd Backend
npm install
```

3.Configure Environment Variables:

- In the `Backend` folder, create a new file named `.env`.
- Copy the contents of `env.example` (if it exists) or use the template below and fill in your secret keys.

4.Set up Supabase Database:

- Ensure you have created the three required tables in your Supabase project:

```
1. `test_cases` (for caching)

2. `feedback_log` (for the learning loop)

3. `knowledge_base` (for persistent documents)
```

Make sure Row-Level Security (RLS) is **disabled** on these tables only for local development.

5.Run the Backend Server:

Bash

```
node index.js
```

The server should now be running on `http://localhost:5000.`

6.Launch the Frontend:

Navigate to the `Frontend` folder and open the `index.html` file in your web browser.

Environment Configuration (.env file)

Proper configuration of environment variables is crucial for the AI Test Case Generator to function correctly and securely. The `.env` file in your Backend directory holds sensitive API keys and database credentials, which should never be committed directly to version control. Below are the essential keys you must define:

```
# Supabase Credentials
SUPABASE_URL="your_supabase_project_url"
SUPABASE_KEY="your_supabase_api_key"

# Groq API Key
GROQ_API_KEY="your_groq_api_key"
# Jira Credentials
JIRA_DOMAIN="your_company.atlassian.net"
JIRA_EMAIL="your_jira_email"
JIRA_TOKEN="your_jira_api_token"
JIRA_PROJECT_KEY="PROJ"

# Zephyr Configuration
ZEPHYR_TEST_ISSUE_TYPE_ID="12345"
```

- **SUPABASE_URL & SUPABASE_KEY:** These credentials link your application to your specific Supabase project, enabling database interactions for caching, feedback logging, and knowledge base management.
- **GROQ_API_KEY:** Your personal API key for accessing the Groq AI service, which powers the Llama 3 language model for all test case generation tasks.
- **JIRA_DOMAIN, JIRA_EMAIL, JIRA_TOKEN, JIRA_PROJECT_KEY:** Required for integrating with Jira, allowing the application to import user stories and requirements from your Jira instance. The Jira API token ensures secure authentication.
- **ZEPHYR_TEST_ISSUE_TYPE_ID:** Identifies the specific issue type in Zephyr for test cases, enabling seamless export of generated tests to your Zephyr test management system.

Ensure these values are accurate and kept confidential to maintain the integrity and security of your application.

How to Use the Application

The AI Test Case Generator is designed for intuitive use, guiding QA engineers through the process of generating, refining, and managing test cases. Follow these steps to maximize its potential:

1. Enter Requirement

Begin by inputting your feature requirement into the main text area. Alternatively, import requirements directly from Jira issues or upload a screenshot for OCR-based text extraction.

2. Select Options

Utilize the dropdown menus to configure your test case generation. Choose the desired test type (e.g., Smoke, Regression), complexity level, the number of test cases to generate, and the preferred output format.

3. Provide Context (Optional)

- **Application Context:** For session-specific context, use the "Application Context" card to upload relevant code snippets or documentation.
- **Knowledge Base:** To leverage permanently stored project standards, upload documents to the "Knowledge Base" card and ensure the "Use Knowledge Base" checkbox is ticked.

4. Generate Test Cases

Once your requirements and options are set, click the "Generate Test Cases" button. The AI will process the information and produce test cases based on your specifications.

5. Review and Act

- **Feedback:** Use the like / dislike buttons to provide immediate feedback, helping the AI learn and improve.
- **Refine:** Click the "Refine" button to open a conversational chat for making specific edits to a test case.
- **Utilities:** Use the **Copy**, **Download**, and **Clear** buttons for managing the generated output.

Future Roadmap: Enhancing Intelligent QA

The AI Test Case Generator is continuously evolving, with a clear roadmap designed to introduce advanced functionalities and deeper integrations. These planned enhancements will further solidify its position as a leading tool for intelligent QA automation.

- **CI/CD Integration:** Implementing automatic execution of generated Playwright tests via GitHub Actions. This will enable seamless integration into existing Continuous Integration/Continuous Deployment pipelines, automating the testing phase of the development lifecycle.
- **Advanced Analytics Dashboard:** Developing a comprehensive dashboard to visualize user feedback and AI performance metrics. This will provide valuable insights into the AI's learning progress, the quality of generated test cases, and areas for further improvement.
- **Vector Search for Knowledge Base:** Enhancing the Knowledge Base with semantic search capabilities. By leveraging vector embeddings, the AI will be able to retrieve more relevant context from large and complex documents, leading to even more precise and context-aware test case generation.
- **Expanded Integrations:** Broadening support for other popular project management and development tools, including GitHub Issues and Asana. This will ensure wider compatibility and allow more teams to seamlessly incorporate the AI Test Case Generator into their existing workflows.

These future developments underscore our commitment to providing a cutting-edge, adaptable, and indispensable tool for modern QA teams.