Playwright MCP & Agents

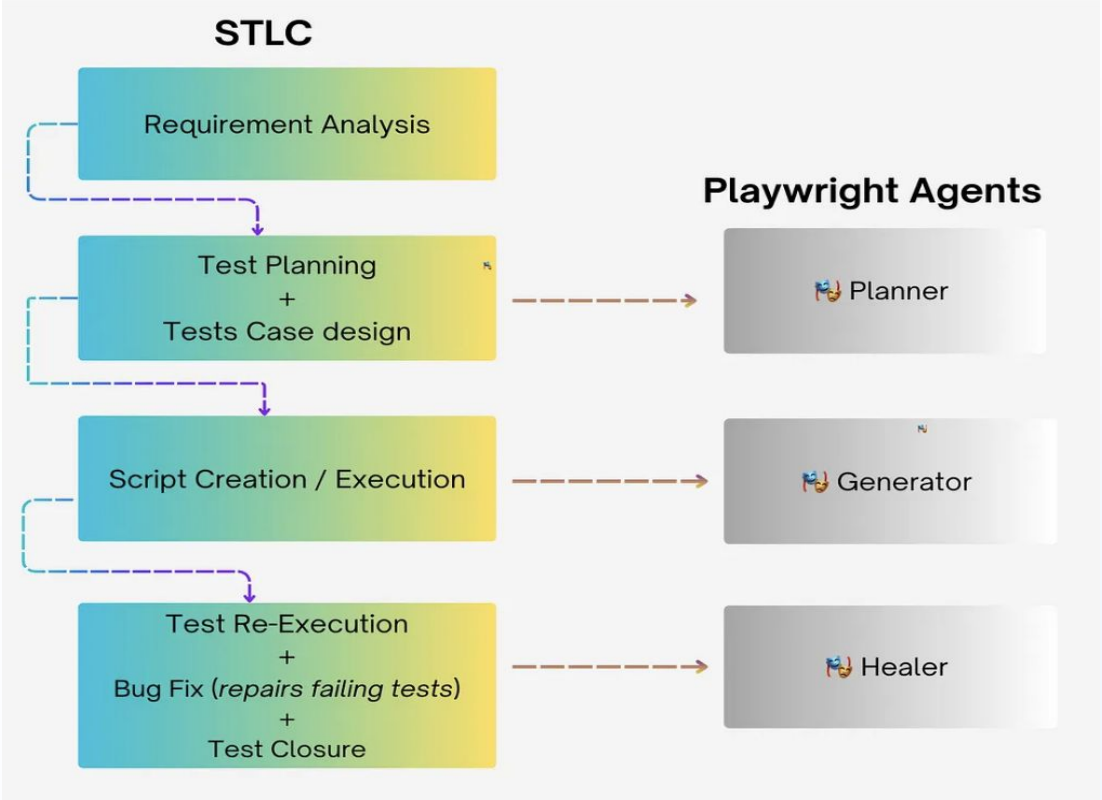
Automating E2E Testing with Planner and Generator Agents in a Local IDE

System Architecture

The Agentic Model

The system uses specialized AI agents that interact via the Model Context Protocol (MCP).

- **Planner Agent:** Explores the application and creates a high-level test plan.
- **Generator Agent:** Converts the plan into executable Playwright test code.
- **Healer Agent:** Executes the test suite and automatically repairs failing tests.
- **Local IDE:** Agents run locally, integrating with GitHub Copilot.

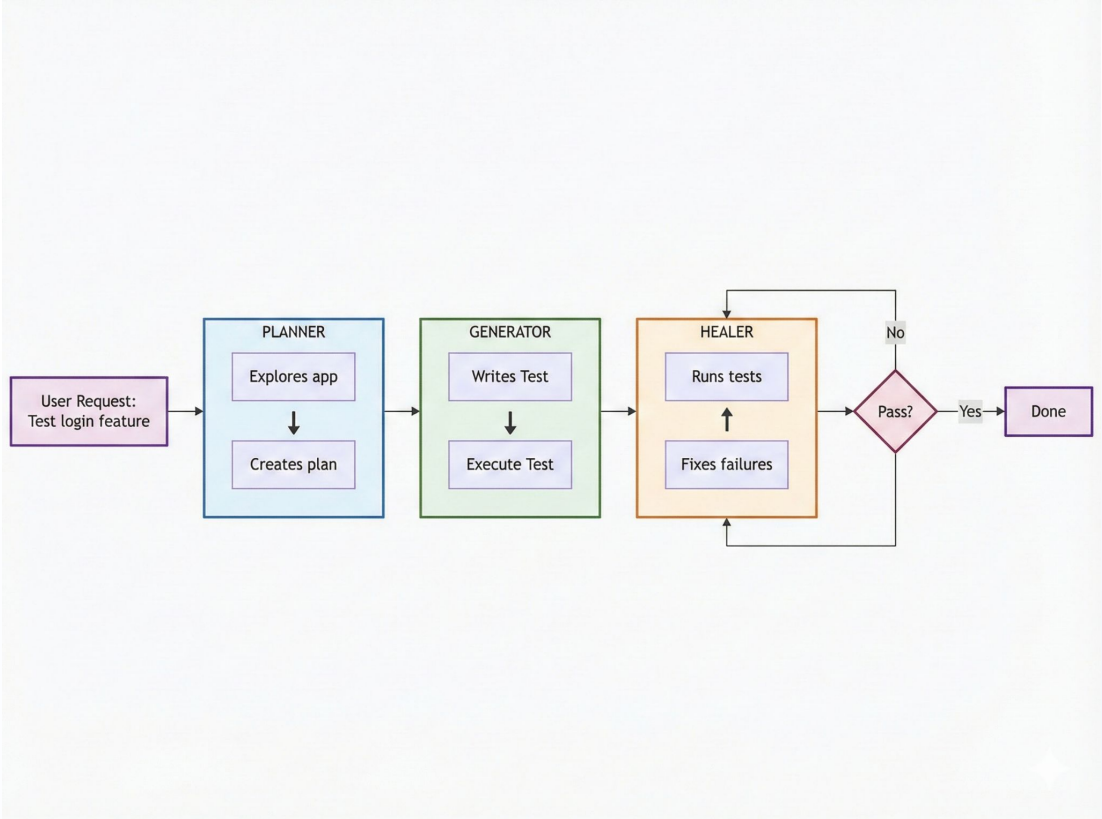


Workflow Diagram

From Prompt to Pass

This flow illustrates how a simple user intent is transformed into a verified test suite.

1. User inputs intent.
2. Planner generates `specs/*.md`.
3. Generator creates `tests/*.spec.ts`.
4. Healer fixes failing tests.
5. Playwright executes tests.

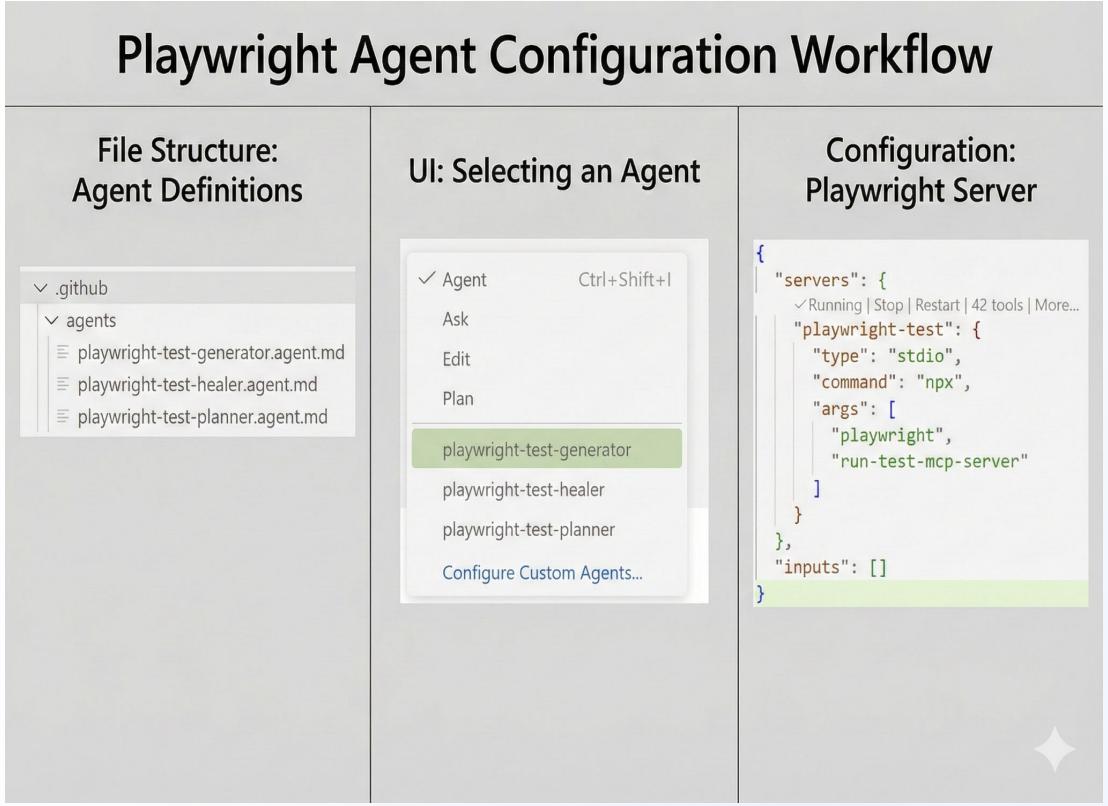


IDE Setup & Integration

MCP & Agents Configuration

We start by initializing the agents in the local environment using `npx playwright init-agents`.

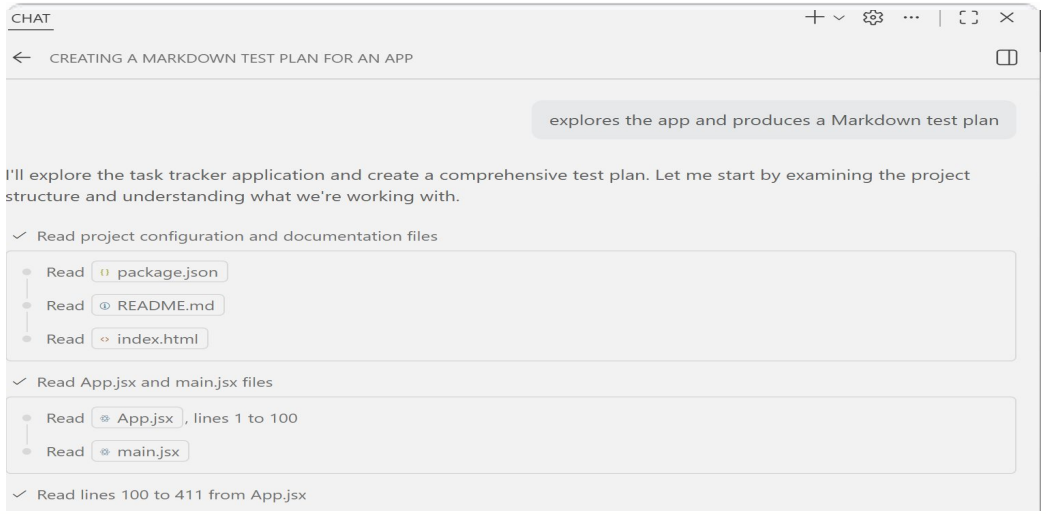
The image on the right shows the agents successfully added to the IDE, ready to communicate with GitHub Copilot Chat.



Using the Planner Agent

Step 1: Initial Prompt

I engaged the Planner agent to explore the app state.



Step 2: Strategy Generation

The Planner outputted a strategy for the test scenarios.

Perfect! I've successfully explored the TaskFlow application and created a comprehensive test plan. Here's a summary of what I discovered and documented:

Application Overview

TaskFlow is a React-based task management application with the following key features:

Core Functionality

- **Task Management**: Create, edit, delete, and mark tasks as complete/incomplete
- **Task Properties**: Title (required), description, priority (low/medium/high), due date
- **Visual Indicators**: Priority badges with color coding, completion state with strikethrough
- **Data Persistence**: Uses localStorage for data persistence across sessions

UI Features

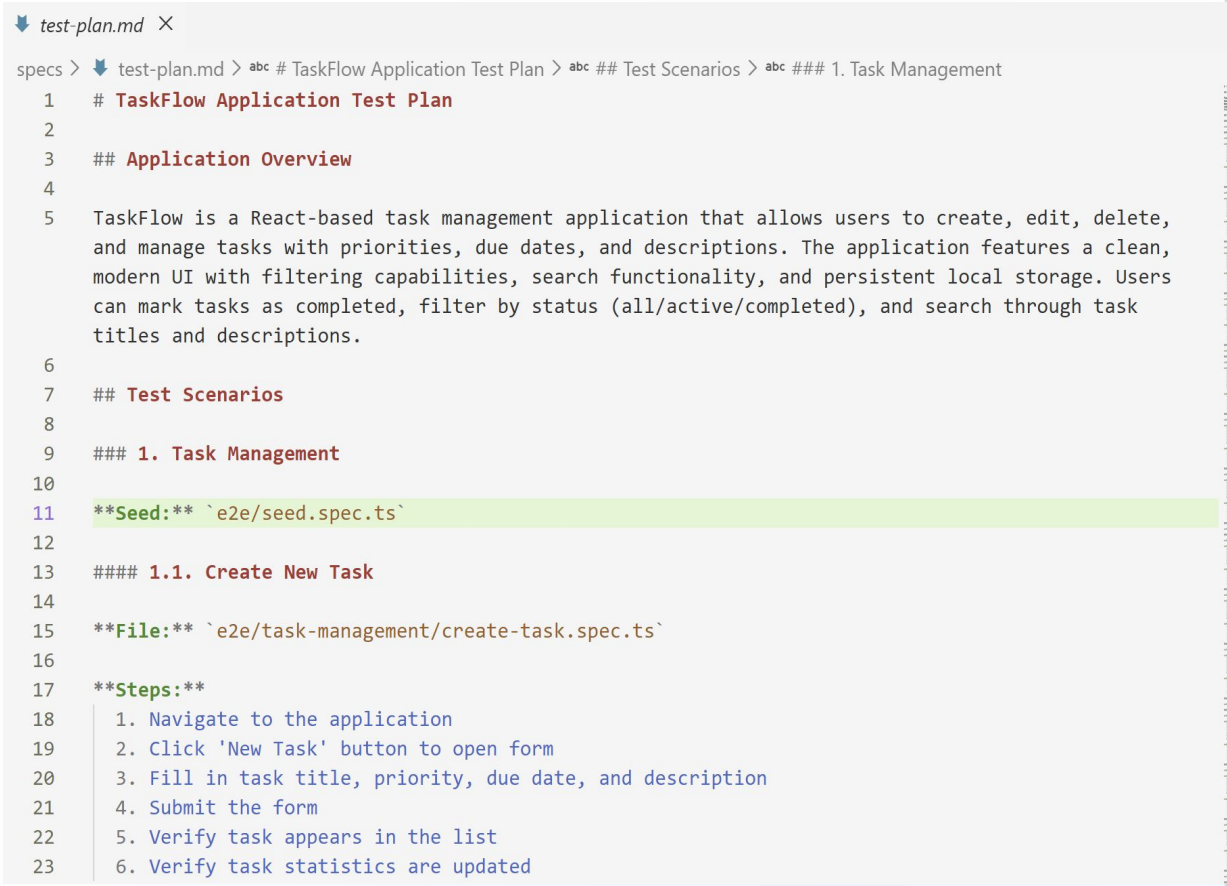
- **Filtering**: View all tasks, active tasks, or completed tasks
- **Search**: Search across task titles and descriptions (case-insensitive)
- **Statistics**: Real-time display of total and completed task counts
- **Responsive Design**: Clean, modern UI with card-based task display

The Test Plan

Markdown Blueprint

The Planner generated this test plan (`specs/test-plan.md`).

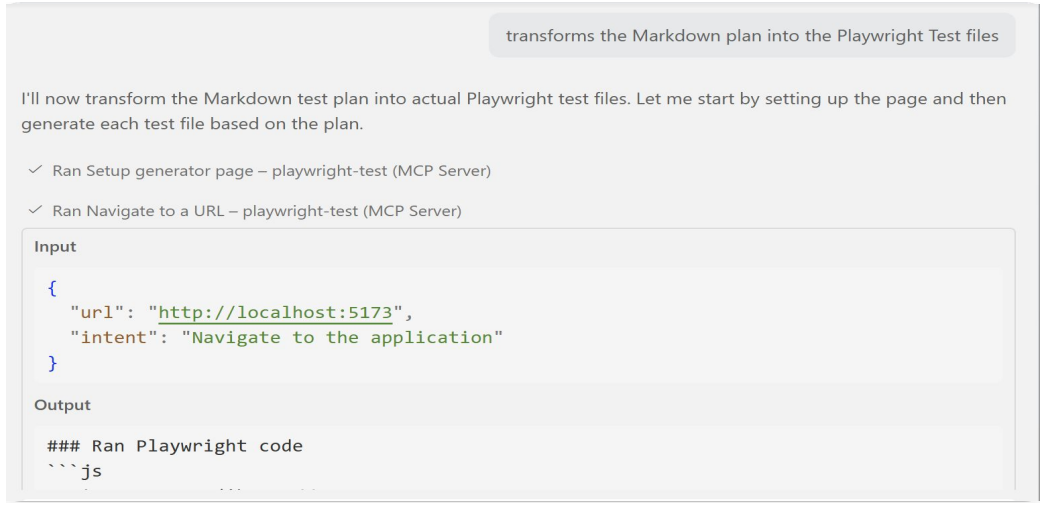
It outlines the testing logic in plain English before any code is written, ensuring the scope is correct.



Using the Generator Agent

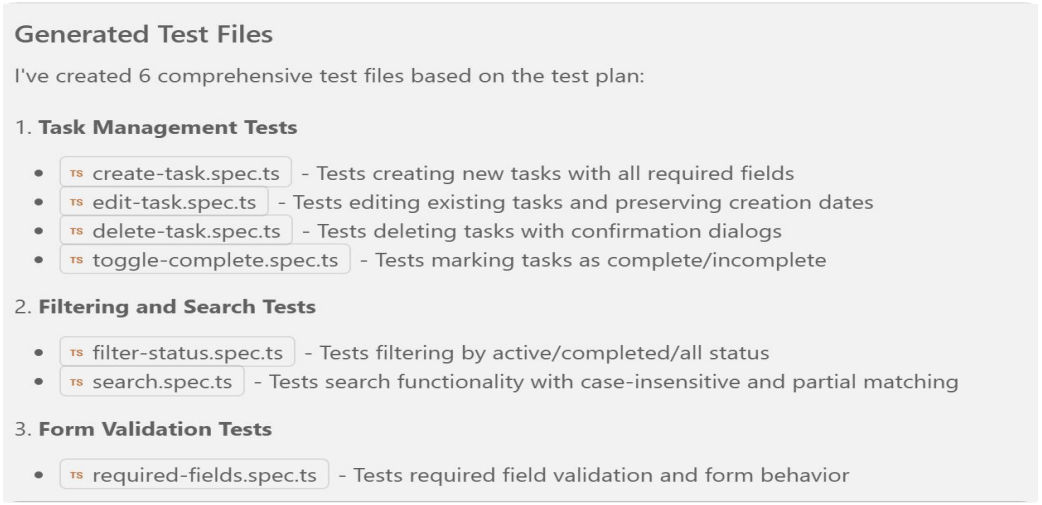
Step 3: Code Generation Prompt

I instructed the Generator to implement the plan.



Step 4: Code Generation

The agent verified selectors against the live DOM and generates test cases.

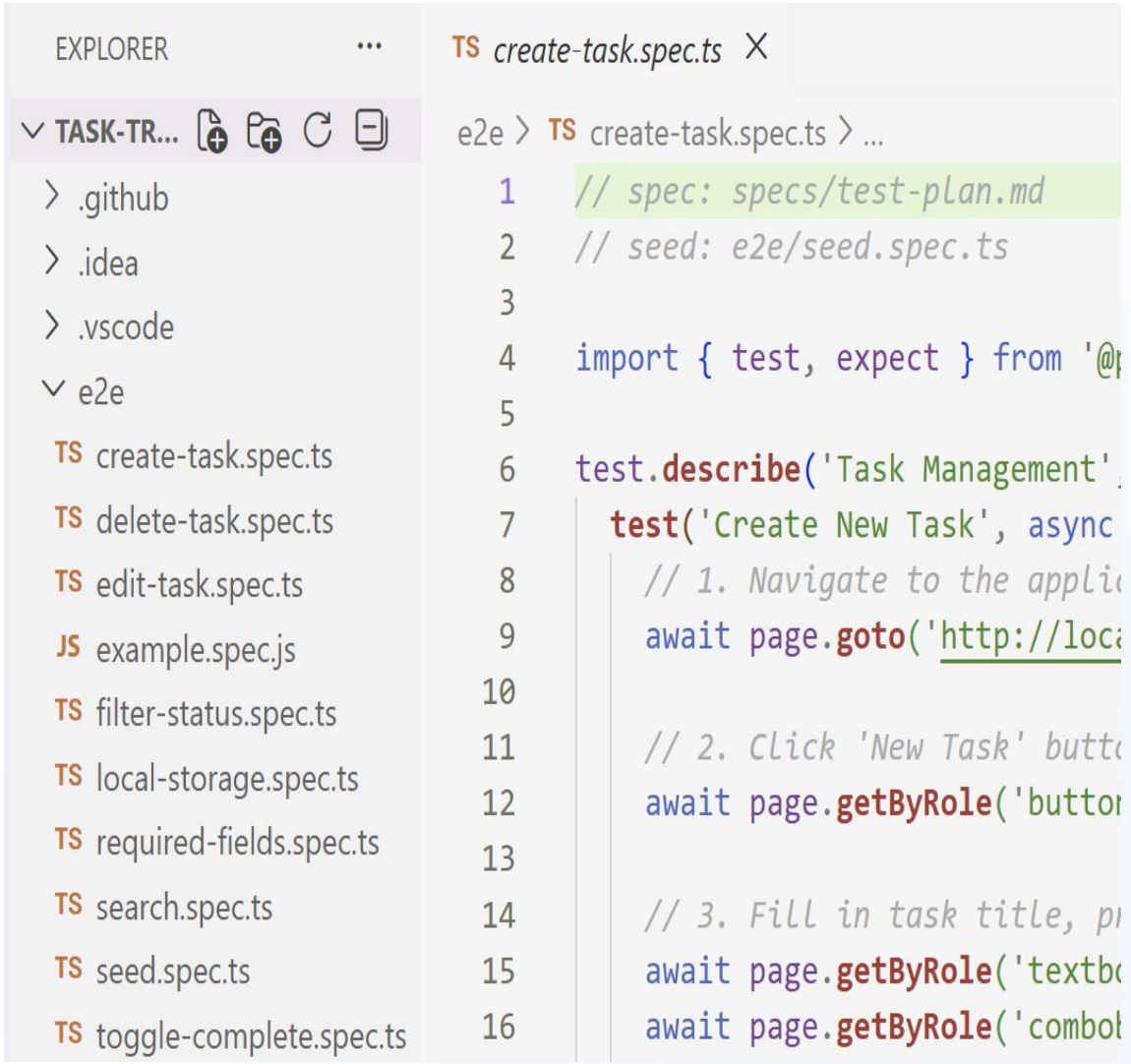


Generated Spec Files

Executable Playwright Code

The Generator created this Typescript spec file (`tests/create-task.spec.ts`).

It includes robust locators, assertions, and follows the steps defined in the Markdown plan.

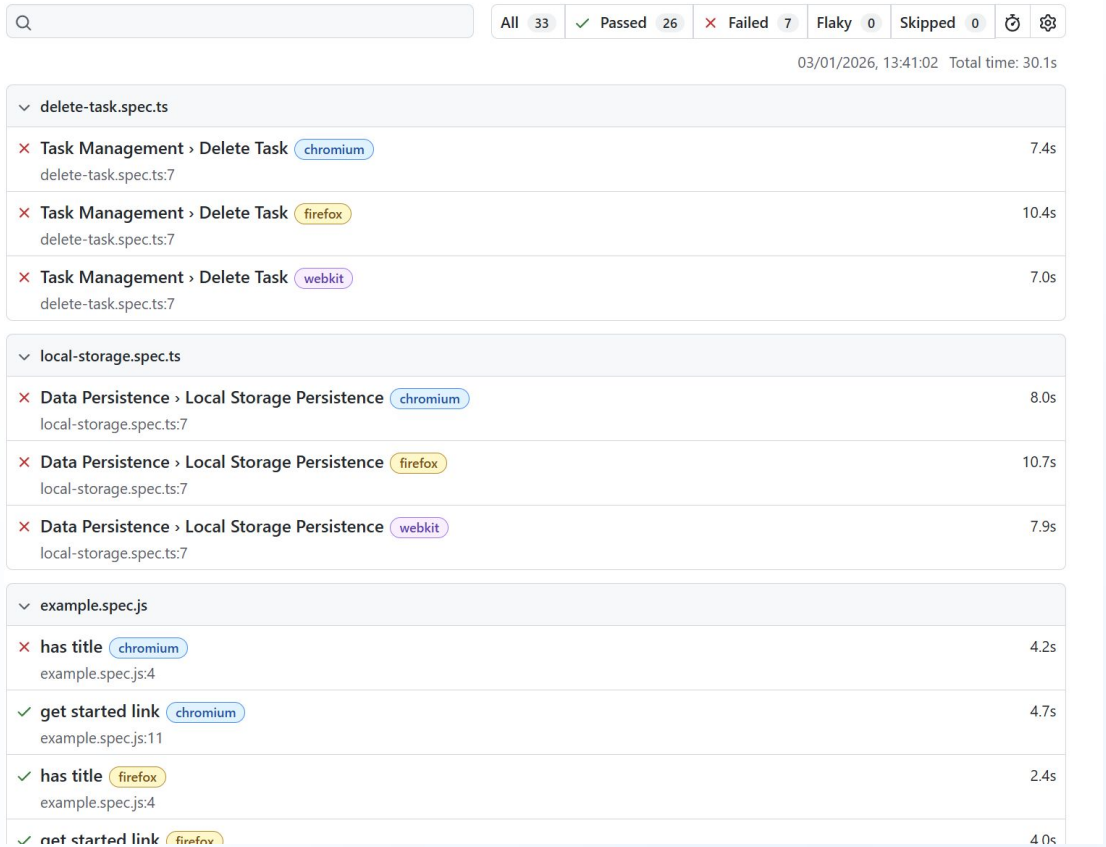


Test Results

Successful Execution

Running the generated tests yielded a passing result.

The output confirms that the agent-generated code correctly validated the application functionality.



Future Step: The Healer Agent

Self-Healing Tests

In the future, we plan to integrate the **Healer Agent** to automatically handle broken tests.

- Analyzes error traces from failed runs.
- Inspects the DOM for selector changes.
- Patches the spec file automatically to fix the break.

GitHub Workflows Integration

Automating in CI

Generated tests are standard Playwright scripts, making them fully compatible with GitHub Actions.

- **Commit:** Push generated `tests/*.spec.ts` to the repo.
- **Trigger:** GitHub Actions workflow detects changes.
- **Execute:** Tests run in parallel using Playwright shards.

Questions?

Thank you for your time.