Project Guide · HTML + CSS + JavaScript

Build an AI Flaky Test Root Cause Analyzer, step by step

A beginner-friendly tutorial. You will build a real working tool that finds unreliable tests in your CI history and explains why they fail, with diagrams and complete copy-paste code.

Start building → Get full code

8	3	0	1
Easy steps	Diagrams	Libraries needed	File to run

LIMITED SEATS

LEVEL UP YOUR TECH CAREER

Become a Playwright Testing Expert

Master modern automation testing with practical training, real-world projects and expert mentorship.

✓ Live Classes
✓ Real Projects
✓ Certificate
✓ Placement Support

Explore Course ↗

Learn. Practice. Get Job-Ready.

UNCODEMY

PROJECT OVERVIEW

What are we building?

A **flaky test** is a test that sometimes passes and sometimes fails even though the code did not change. Flaky tests waste time and hide real bugs. Our tool, **FlakeScope**, reads a list of test runs, gives every test a flakiness score from 0 to 100, guesses the root cause from the error message, and suggests a fix.

Input

Paste JSON or CSV test history, or load sample data.

Score

Counts how often a test flips between pass and fail.

Root cause

Matches error text with known patterns such as timeouts.

Fix tips

Shows a practical suggestion for each cause.

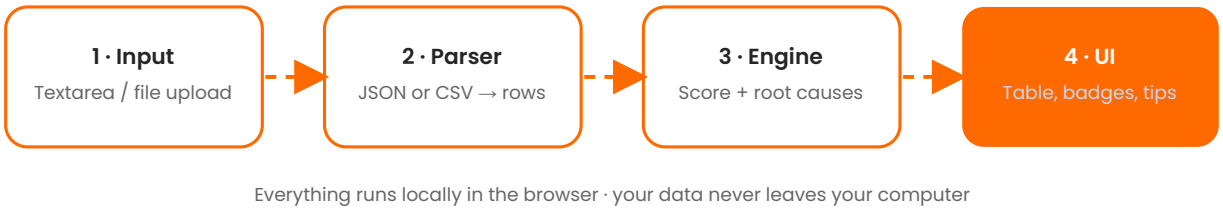
You will learn: DOM handling, parsing JSON and CSV, grouping data, regular expressions, a simple scoring formula and responsive CSS.

UNDERSTAND BEFORE YOU CODE

Project diagrams

Diagram 1: System architecture

Four layers work one after another, all inside the browser. No server and no database are needed.



Everything runs locally in the browser · your data never leaves your computer

Diagram 2: Flowchart of the analysis

This is the exact path your code follows when the user clicks `Analyze`.

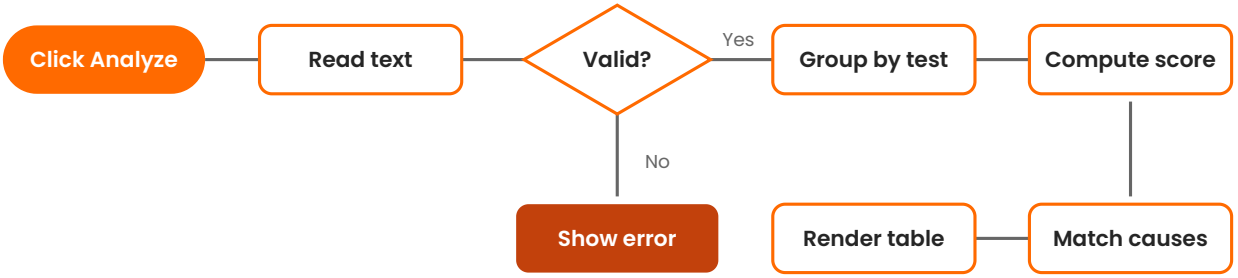
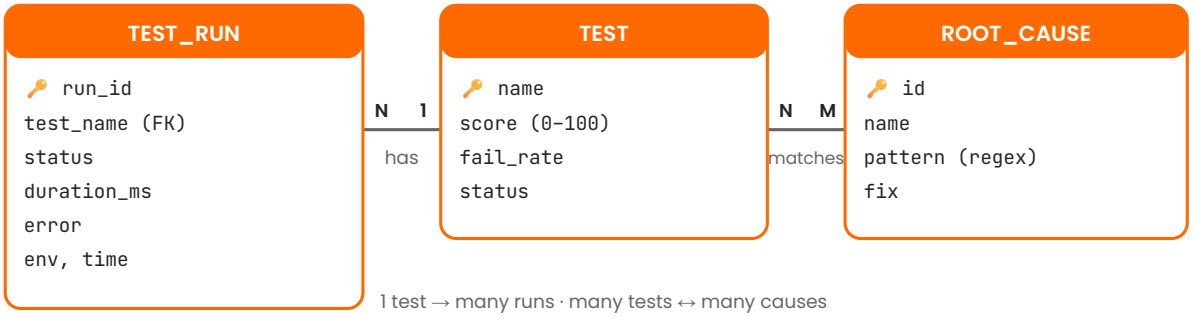


Diagram 3: Data model (ER diagram)

Even without a database, the data has a clear structure. One **test** has many **runs**, and a test can match many **root causes**.



1 test → many runs · many tests ↔ many causes

> Read it as: one TEST (1) has many TEST_RUN rows (N). TEST and ROOT_CAUSE are many-to-many (N : M) because one test can match several causes.

TRY IT LIVE

FlakeScope Demo

Click the button to see the analyzer in action with sample data.

▶ Show Demo

1

Set up the project

You only need a text editor (VS Code is great) and a browser. No installation, no npm.

1. Create a folder named `flakescope`.
2. Inside it, create one file named `index.html`.
3. Open the folder in VS Code. Optional: install the **Live Server** extension so the page refreshes when you save.

terminal

```
mkdir flakescope\ncd flakescope\ncode index.html
```

> We use one single file. HTML, CSS and JavaScript live together, which makes the project easy to share with your teacher.

2

Write the HTML structure

The page needs a title, a text box for data, two buttons and an empty table. JavaScript will fill the table later. Give each element an `id` so JavaScript can find it.

index.html · inside <body> Copy

```
<main class="app">
  <h1>FlakeScope</h1>
  <p>Paste test runs (JSON or CSV), then click Analyze.</p>
  <textarea id="inp" rows="8" placeholder="test,status,error"></textarea>
  <div><button id="go">Analyze</button><button id="demo">Load sample</button>
</div>
  <div style="overflow-x:auto"><table id="out"></table></div>
</main>
```

3

Style it with CSS

CSS makes the tool clean and readable. Use a max width so it looks good on laptops and a scrolling wrapper so the table fits on phones. Put this inside a `<style>` tag in the `<head>`.

CSS · inside <style> Copy

```
:root{--brand:#5b4bff;--bad:#dc2626;--ok:#16a34a}
*{box-sizing:border-box}
body{margin:0;font:15px/1.5 system-ui,sans-
serif;background:#f6f7fb;color:#14172b}
.app{max-width:900px;margin:auto;padding:24px 16px}
textarea{width:100%;padding:10px;font:13px monospace;border:1px solid
#d5d8e8;border-radius:10px}
button{margin:10px 8px 14px 0;padding:9px 16px;border:0;border-
radius:10px;background:var(--brand);color:#fff;font-weight:600;cursor:pointer}
table{width:100%;min-width:620px;border-collapse:collapse;background:#fff}
th,td{padding:10px;text-align:left;border-bottom:1px solid #eceef6}
th{background:#eef0fb;font-size:12px;text-transform:uppercase}
.tag{padding:2px 9px;border-radius:99px;font-size:12px;font-
weight:700;color:#fff}
.dot{display:inline-block;width:7px;height:16px;margin-right:2px;border-
radius:2px;background:var(--ok)}
.dot.f{background:var(--bad)}
```

4

Read and clean the data

Users can paste JSON or CSV. First we try JSON. If that fails, we treat the text as CSV: the first line holds column names, every other line is one run. We keep only rows that have a `test` name and a `status`.

The `RULES` list holds our “knowledge”: a cause name, a pattern to look for in error messages, and a fix tip. The `esc()` helper keeps the page safe from harmful text.

JavaScript · part 1 Copy

```
const $ = id ⇒ document.getElementById(id);
const esc = s ⇒ String(s ?? '').replace(/[&<>"']/g, c ⇒ '&#' +
c.charCodeAt(0) + ';');
const RULES = [
  ['Timeout / slow dependency', /time(d)? ?out|deadline|exceeded/i, 'Find the
slow step and stub slow services.'],
  ['Network issue', /ECONN|ENOTFOUND|socket|50[234]|network/i, 'Mock the
network or wait for health checks.'],
  ['Async / race condition', /stale|not interactable|not
found|detached|promise/i, 'Use explicit waits instead of sleep().'],
  ['Shared state / order', /duplicate|already exists|unique|leftover/i, 'Give
every test its own data.'],
  ['Time / timezone', /timezone|midnight|expired|clock/i, 'Use a fake clock
and set TZ=UTC.']
];
function parse(text) {
  text = text.trim();
  let rows;
  try { rows = JSON.parse(text); }
  catch {
    rows = text.split(/\r?\n/).filter(Boolean).map(l ⇒ l.split(','));
    const head = rows.shift().map(h ⇒ h.trim().toLowerCase());
```

5

Build the analysis engine

This is the brain of the project. For each test we calculate:

- **Failure rate** = failed runs ÷ total runs.

- **Flip rate** = how often the result changes from one run to the next.
- **Score** = 100 × (0.75 × flip rate + 0.4 × balance), capped at 100, where balance = 4 × p × (1 - p).

Then we test every failure message against `RULES` and pick the cause that matches most failures.

JavaScript · part 2 Copy

```
function analyze(runs) {
  const groups = {};
  runs.forEach(r => (groups[r.test] = groups[r.test] || []).push(r));
  return Object.entries(groups).map(([name, rs]) => {
    const n = rs.length, fails = rs.filter(r => !r.ok);
    const p = fails.length / n;
    let flips = 0;
    for (let i = 1; i < n; i++) if (rs[i].ok !== rs[i - 1].ok) flips++;
    const flip = n > 1 ? flips / (n - 1) : 0;
    const score = Math.round(100 * Math.min(1, 0.75 * flip + 0.4 * 4 * p * (1
- p)));
    const hits = RULES.map(rule => [rule, fails.filter(r =>
rule[1].test(r.err)).length]).filter(h => h[1]).sort((a, b) => b[1] - a[1]);
    return { name, p, score, rs, cause: hits[0] ? hits[0][0] : null };
  }).sort((a, b) => b.score - a.score);
}
```

> **Example:** a test that goes Pass, Fail, Pass, Fail, Pass has 5 runs, 2 failures, 4 flips. Failure rate = 0.4, flip rate = 1, balance = 0.96 → score 100.

6

Show results and connect the buttons

`render()` turns the results into table rows with a colored status badge, the likely cause, a fix tip and a strip of green and red boxes for recent runs. Finally we connect the buttons and add a sample data generator so you can test without your own data.

JavaScript · part 3 Copy

```
const label = r => r.p === 1 ? ['Broken', '#dc2626'] : r.p === 0 ? ['Stable',
'#16a34a'] : r.score >= 50 ? ['Highly flaky', '#dc2626'] : r.score >= 25 ?
['Flaky', '#d97706'] : ['Mild', '#5b4bff'];
function render(list) {
  $('out').innerHTML = '<tr><th>Test</th><th>Score</th><th>Status</th>
<th>Likely cause & fix</th><th>Runs</th></tr>' +
    list.map(r => {
      const l = label(r);
      return `<tr><td>${esc(r.name)}</td><td><b>${r.score}</b></td>
        <td><span class="tag" style="background:${l[1]}">${l[0]}</span></td>
        <td>${r.cause ? esc(r.cause[0]) + '<br><small>Fix: ' + esc(r.cause[2])
+ '</small>' : '-'}</td>
        <td>${r.rs.slice(-20).map(x => `<i class="dot${x.ok ? '' : ' f'}">
</i>`).join('')}</td></tr>`;
    }).join('');
}
function sample() {
  let s = 7; const rnd = () => (s = s * 16807 % 2147483647) / 2147483647, out
= [];
  [['login', 0, ''], ['checkout', .3, 'Timeout 5000ms exceeded'], ['orders',
.25, 'ECONNRESET socket hang up'], ['db users', .3, 'duplicate key unique
constraint'], ['reports', .2, 'timezone mismatch']]
```

7

Test your project

- ✓ Click **Load sample**: a ranked table should appear.
- ✓ Check that **login** is Stable, and the others are flaky.
- ✓ Paste a tiny CSV with only a header and see that nothing crashes.
- ✓ Paste broken text such as `hello` and check that an error message appears.
- ✓ Resize the browser to phone width and check that the table scrolls sideways.

> Press **F12** in your browser and open the Console tab. Red messages there show mistakes in your JavaScript.

8

Run, share and improve

1. Double-click `index.html` to run it locally.
2. Upload the file to GitHub Pages, Netlify or any static host to get a public link.
3. Improve it: add more `RULES`, a search box, a dark mode, or a file upload button.

EVERYTHING TOGETHER

Complete project code

Click the button to copy the full ready-to-run file. Paste it into `index.html` and open it. It is built from the same parts shown in steps 2 to 6.

Copy full code

Show / hide code