INTERNAL TOOLING

Record a browser task once, replay it for every row

A desktop recorder, a drag-and-drop column mapper, and a code generator that puts the loop in the right place.

Sytze P.
Solution Engineer

The problem

A lot of back-office work is one web form, filled a few hundred times from a spreadsheet. The site has no API and no bulk import, so somebody does it by hand.

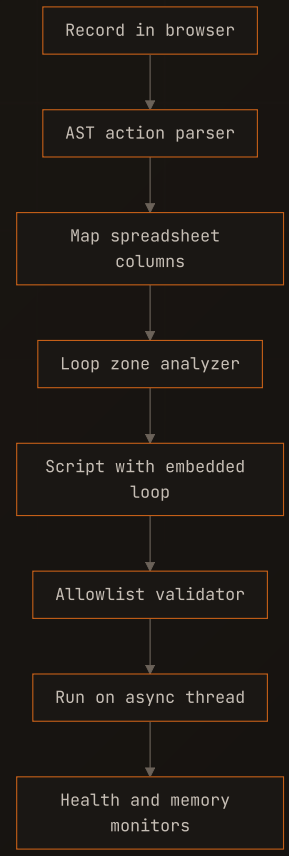
Recording the clicks is the easy half. Replay is where it falls apart. Run the whole recording once per row and the login page gets visited three hundred times, and the final submit fires three hundred times with it.

APPROACH

Recording to spreadsheet-driven run

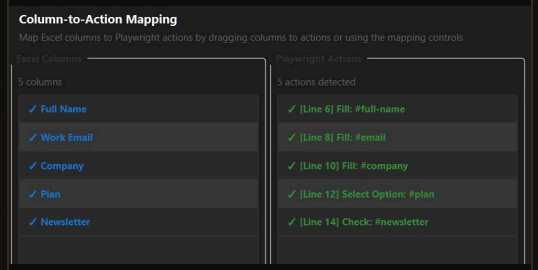
Recording uses Playwright's own codegen against a real browser. An AST parser turns that script into a list of actions, and you drag spreadsheet columns onto the ones that take data.

Those mappings locate the loop. An action with a column mapped to it belongs inside it, everything before the first mapped action is setup, everything after the last is cleanup, and the ambiguous actions between are placed by action type. The output is one script with the loop written in and row values injected, instead of the same script run N times.



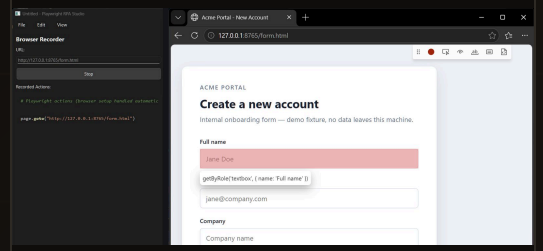
HIGHLIGHTS

How it works



THE MAPPING DECIDES WHERE THE LOOP GOES

Any action with a spreadsheet column mapped to it runs per row. Navigation and the final submit land outside the loop, where they belong, and you can move any of them by hand.



RECORDING IS A REAL BROWSER, NOT A SCRIPT LANGUAGE

You click through the flow in an actual browser session and the generated Python appears in the editor as you go. Nothing to learn before the first run.

Recorded Actions:

```
# Playwright actions (browser setup handled automatically)

page.goto("http://127.0.0.1:8765/form.html")
page.fill("#full-name", "Jane Doe")
page.fill("#email", "jane.doe@northwind.test")
page.fill("#company", "Northwind Trading")
page.select_option("#plan", "business")
page.check("#newsletter")
```

GENERATED CODE IS CHECKED BEFORE IT RUNS

The script is parsed to an AST and matched against an allowlist of page calls, then executed in a namespace with no imports, no file access and no eval.

```
[20:33:24] [INFO] Starting execution with Excel data and 5 mappings
[20:33:24] [INFO] Execution started
[20:33:24] [INFO] Starting Excel-driven execution with data mapping
[20:33:24] [INFO] Loaded 12 rows from Excel
[20:33:25] [INFO] Execution started: 12 rows to process
[20:33:25] [INFO] Execution completed successfully: 12 rows in 0.21s
[20:33:25] [INFO] Execution completed: 12/12 rows succeeded (100.0%)
```

BUILT TO SURVIVE A LONG RUN

Failed steps retry with backoff, a missing selector falls back through id, class, text and role, and a memory circuit breaker restarts the browser rather than letting the run die at row two hundred.

RESULTS

Results & impact

one browser session

setup and cleanup run once, not per row

validated before it runs

AST allowlist, then a restricted namespace

42 test modules

parser, engine, coordinator and UI

STACK

Tech stack

PySide6
desktop UI and threading

Playwright
recording and replay

Python asyncio
dedicated loop thread

openpyxl
spreadsheet read and write

psutil
browser memory watch

pytest
unit and integration tests

Sytze P.

Solution Engineer

A desktop recorder, a drag-and-drop column mapper, and a code generator that puts the loop in the right place.