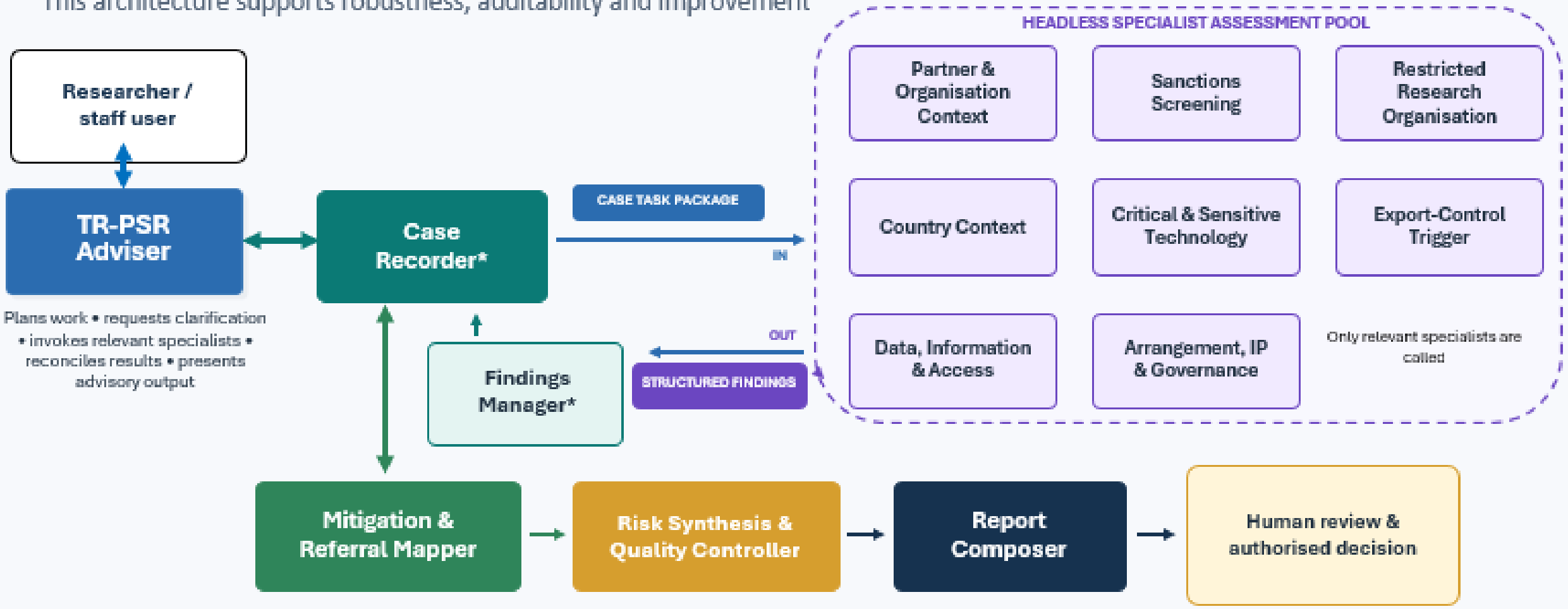
Agents – striking the balance between black box and structured data

Adam Crisp,
Research Systems,
Research Office

9 Nov 2026

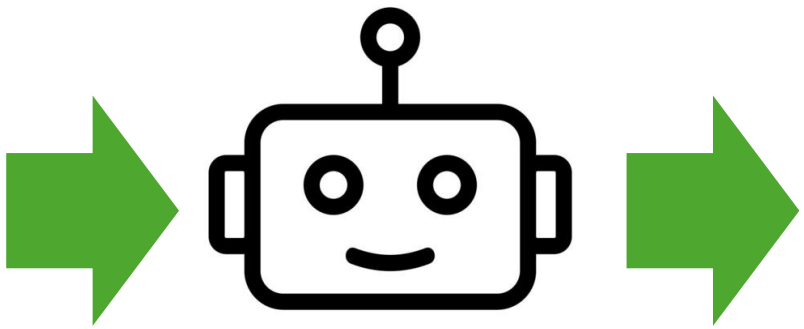
Draft multi-agent architecture

Only the orchestrator (i.e. TR-PSR-Adviser frontend agent) interacts with the user.
Specialist agents return bounded, structured findings
This architecture supports robustness, auditability and improvement



From open to structured data

Agent Inputs (from case manager agent)	
ID	Name
CASE00X	Case number
MSG00X	User message
IEC00X	Entity raw text
IEC00X	Entity context



Partner &
Organisation Agent

Partner & Organisation Agent Outputs	
Name	ID (e.g.)
Entity unique idenitfier	OEC_EUI001
Full name	OEC_FN001
email	OEC_EM001
Proposal / project role	OEC_PR001
Institutional role	OEC_INR001
Institution	OEC_INT001
City	OEC_CIT001
Country	OEC_COU001
AI Confidence	OEC_CON001
AI Assessment	OEC_ASS001
AI Comment	OEC_COM001

AI assisted App build to remove repetitive tasks

Adam Crisp,
Research Systems,
Research Office

9 Nov 2026

Problem

Current situation

- Heavily repetitive task, process stays the same but values vary
- Web-based tool, currently must manually click and fill in
  - Task required not supported by APIs
  - Also wanted to be able to see what is being added/changed if desired
- There must be more to life….

Target Solution

Current situation

- Heavily repetitive task, process stays the same but values vary
- Web-based tool, currently must manually click and fill in
  - Task required not supported by APIs
  - Also wanted to be able to see what is being added/changed if desired
- There must be more to life….

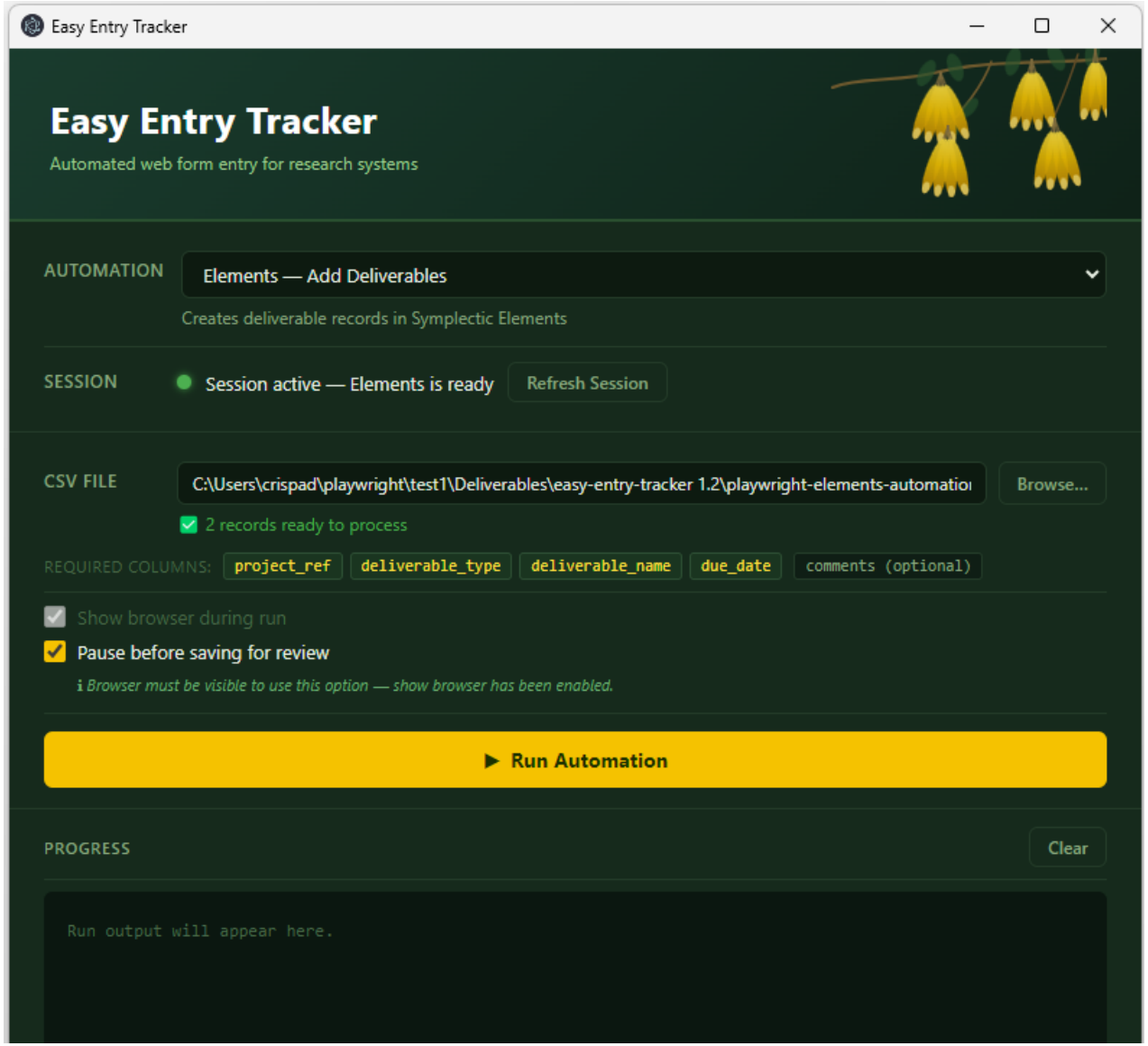


Desired outcome

- Automated as much as practicable
- Minimal / no coding
- High transparency – can see what is happening
- Low risk
  - User access inherited
  - No login info stored
  - Microsoft native not third-party tools
  - Only communicates with web application
- Low/no cost – does not need AI on an ongoing basis, no license costs
- Meets IT expectations - Microsoft native tools
- Friendly for non-technicals

Solution

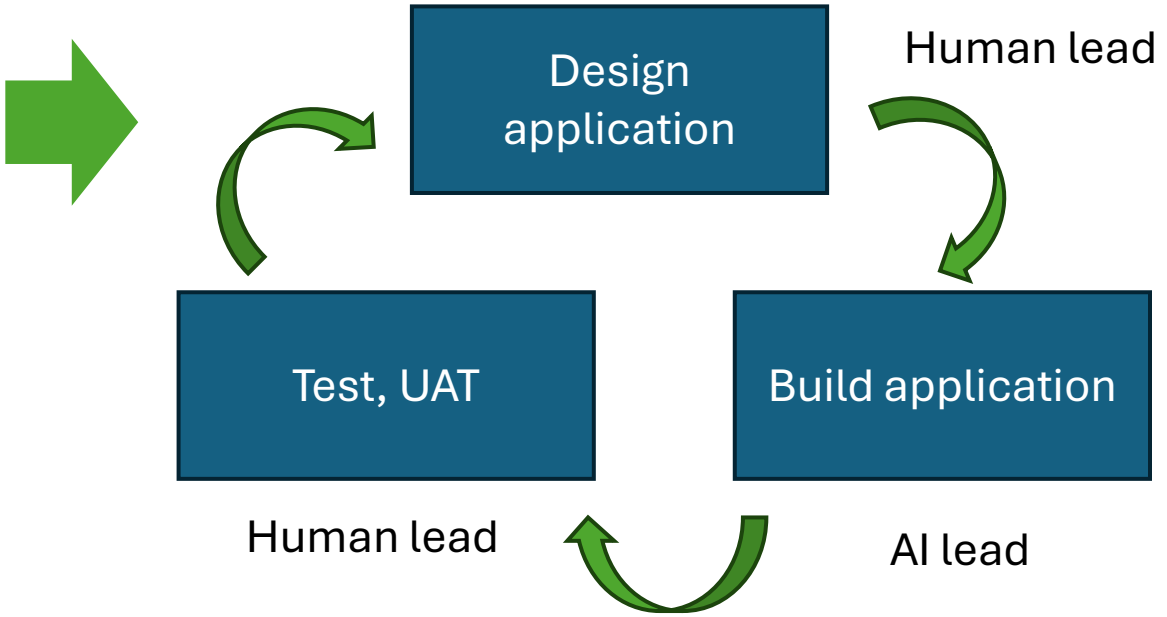
- AI built Electron App using playwright scripts



Steps

Install Playwright, generate script

Parameterise script & test



Steps involved in installing Playwright.docx

```
C:\Users\crispad>cd playwright

C:\Users\crispad\playwright>cd test1

C:\Users\crispad\playwright\test1>npx playwright codegen
```

https://playwright.dev/docs/test-cli

What values will change on each iteration?

Asked AI what to do

Improve this prompt: what microsoft tools that are free without any stealth license costs could a non-techie use to record a website data-entry process? Local install would be good as data is sensitive



Identify **all relevant Microsoft-origin or Microsoft-owned tools** that a non-technical user could use to **record a website data-entry process and turn the recorded actions into a repeatable automation**.

My requirements are:

- **Genuinely free to use**, with no hidden, downstream, premium, subscription, Power Platform, Azure, connector, runtime, or other licensing costs required to create **or run** the automation.
- Prefer tools that can be **installed and run locally on a Windows 11 PC**, because the website and data may contain sensitive information and I want to minimise or avoid sending data to cloud services.
- Suitable for a **non-technical user to initially record the process**, even if some technical configuration or scripting is needed later to parameterise and package the automation.
- The recorded process should be capable of being **parameterised**, so values such as names, IDs, dates and other form entries can be supplied from a file or simple user interface rather than hard-coded.
- Include **older, less prominent, developer-oriented and open-source Microsoft tools**, not just current mainstream Microsoft 365 or Power Platform products. In particular, do not exclude a tool simply because it was originally intended for developers if its recording capability could realistically be used by a non-technical person.

For each viable tool, explain:

1. What it records and how the recording works.
2. How easy the recording stage would be for a non-technical user.
3. Whether it runs **entirely locally** or uses cloud services.
4. Whether sensitive website data, credentials or recorded interactions could leave the local PC.
5. Whether the recorded actions can subsequently be parameterised.
6. **Every licence, subscription or paid dependency** required for development and ongoing use, explicitly identifying anything that could introduce a future cost.
7. Whether the resulting automation could be packaged inside a **free, locally installed desktop application** for a non-technical end user.

The answer for you could be very different...

Conclusion

As of **5 November 2026**, the genuinely zero-additional-cost Microsoft set is:

1. **Microsoft Playwright Codegen/Inspector** — best overall.
2. **Microsoft Playwright Test for VS Code** — easiest Playwright recording interface.
3. **Microsoft Playwright CLI recorder** — less prominent, developer-oriented recorder that can hand the browser to a non-technical demonstrator.
4. **Microsoft Edge DevTools Recorder** — excellent no-install prototype recorder, but requires later conversion for parameterisation and packaging.
5. **Power Automate for desktop Free** — easiest low-code choice, but not genuinely local-only and not freely embeddable in another application.
6. **WinAppDriver UI Recorder** — free legacy fallback, but only conditionally suitable for modern websites.

The strongest fit for all requirements is:

> **Record with Playwright, replace recorded literals with CSV/JSON or form inputs, and package the result as a self-contained .NET desktop application that drives the locally installed Microsoft Edge.**

That route needs no Azure, Power Platform, premium connector, cloud runner, Microsoft 365 subscription, paid IDE or metered service.

1. Microsoft Playwright: strongest licence-free option

This is the option I would put at the top for what you've described previously.

Playwright is a Microsoft-developed browser automation framework capable of controlling Chromium, Firefox and WebKit, and Microsoft's Edge documentation explicitly documents using it to automate Edge. microsoft

It can:

- open a website;
- locate fields and buttons;
- fill text fields;
- select options;
- click buttons;
- navigate between records/pages;
- wait intelligently for page state;
- run loops over input data;
- use variables/parameters;
- automate Edge visibly or headlessly. microsoft

Most importantly for your particular requirement, Playwright's ecosystem supports the **record → generate code → parameterise → repeat** workflow you've been investigating.

Original prompt used

Which approach would you recommend if my goal is to create a completely free, locally installed desktop application around the website automation? The finished application should allow a non-technical user to run the automation through a simple interface without requiring a paid licence, subscription, cloud service, or separate automation platform.

For that exact requirement, I would choose **Playwright + Electron**, using **TypeScript/JavaScript** for both.

Recommended architecture

Electron desktop application

...

Playwright automation engine

...

Why I'd choose this combination

1. No automation-platform licence

2. The user doesn't need to know Playwright exists

3. Playwright is much better suited to the actual browser interaction than Electron

4. TypeScript/JavaScript makes sense across the whole application

Now what?

Improve this prompt: What needs to be on my laptop to install playwright successfully?



“What needs to be installed, enabled or configured on my Windows laptop to successfully install, develop and test a Playwright-based website automation? Identify which components are mandatory and which are optional, including Node.js, npm, browsers, Playwright browser dependencies, Visual Studio Code, permissions and any Windows-specific requirements. Also distinguish between prerequisites needed by me as the developer and those that would eventually be required by a non-technical user running a packaged Electron + Playwright desktop application.”

AI rabbit hole

Shorten the mandatory list to what is not installed by default in Windows 11

Mandatory additions

1. **Node.js**
   - npm is bundled with Node.js, so you don't need to install npm separately. electronjs
2. **Playwright**
   - Installed through npm:

PowerShell

```
1  npm i playwright
```

npmjs

3. **Playwright browser binary**
   - For example, if using Chromium:

PowerShell

```
1  npx playwright install chromium
```

Playwright requires browser binaries appropriate to its version. playwright

Website search “installing playwright on windows” performed much better than my string of AI queries, resulted in https://playwright.dev/docs/intro

“installing playwright on windows javascript” resulted in https://autify.com/blog/playwright-javascript

Building the App: Prompt 1

PROMPT 1

- Your role is to train someone who has installed playwright and successfully run a test recording that captured the website interactions into a javascript how to progress beyond this.
- The goal is the test script is updated into a robust script and surrounding process which can run through the interaction multiple times i.e. run the same process over different records using a delimited file supplied to it.
- The end process will be done by non-technical users so needs to be user friendly.
- Before proceeding with this query point ask any clarifying questions about the process, the goal or any other relevant aspect.

Run using Claude Opus (15 Sep 2026) under Copilot premium license

Building the App: Prompt 2

(Numbered sections in Prompt 2 are a direct response to clarifying questions asked by Claude Opus)

1. The immediate website or application being automated is Symplectic Elements. However I want the architecture and recommended approach to reflect that the approach should be readily adaptable for other website applications.

2. One record represents one iteration of the recorded interaction, which involves two parts: searching for a project record within Elements and then clicking on the project record and within it creating one or more records to record deliverables due throughout the project. As the number of deliverable records per project record is low volume (maximum 10) and the number of project records at any one time that will have deliverables added is low (maximum 10), so the delimited file will have maximum 100 records per run, simplicity and robustness rather than efficiency is the primary objective. In this case simplicity means the approach should only require one delimited file and all records require similar parameter values populated. Robustness means that I wish you to indicate if it is more robust to treat each set of parameters as one complete interaction or whether having a loop to check if the deliverable is being created for the same project as the previous deliverable is a more robust approach.

Prompt 2 continued:

Here is the test script:

```
import { test, expect } from '@playwright/test';


test('test', async ({ page }) => {
  await page.goto('https://elements.wgtn.ac.nz/login.html?returnurl=%2Fhomepage.html%3Fem%3Dfalse');
  // await page.getByRole('textbox', { name: 'Username' }).fill('crispad');
  // await page.getByRole('textbox', { name: 'Username' }).press('Tab');
  //  await page.getByRole('textbox', { name: 'Password' }).fill('xxxxx');
  // await page.getByRole('textbox', { name: 'Password' }).press('Enter');
  await page.getByRole('menuitem', { name: 'System admin' }).click();
  const page1Promise = page.waitForEvent('popup');
  await page.getByRole('menuitem', { name: 'Funding management console' }).click();
  const page1 = await page1Promise;
  await page1.getByRole('button', { name: 'Menu' }).click();
  await page1.getByRole('link', { name: '› Funding Records (Search)' }).click();
  await page1.locator('#ctl_C_Reference').click();
  await page1.locator('#ctl_C_Reference').fill('E_12345');...
```

Key considerations: questions to ask

1. Security

Who can access the data and credentials, and where are they stored?

2. Ease of use

Can a non-technical user run it without help?

3. Robustness

What happens when a step fails, or the site changes?

4. Key behaviours

What must the app always (or never) do?

5. Transparency

Can users see what was done, and what went wrong?

6. Users

Who will run it, and what is their skill level?

7. One record / iteration

What does a single record or one run represent?

8. Type of data

What data is being managed, and how sensitive is it?

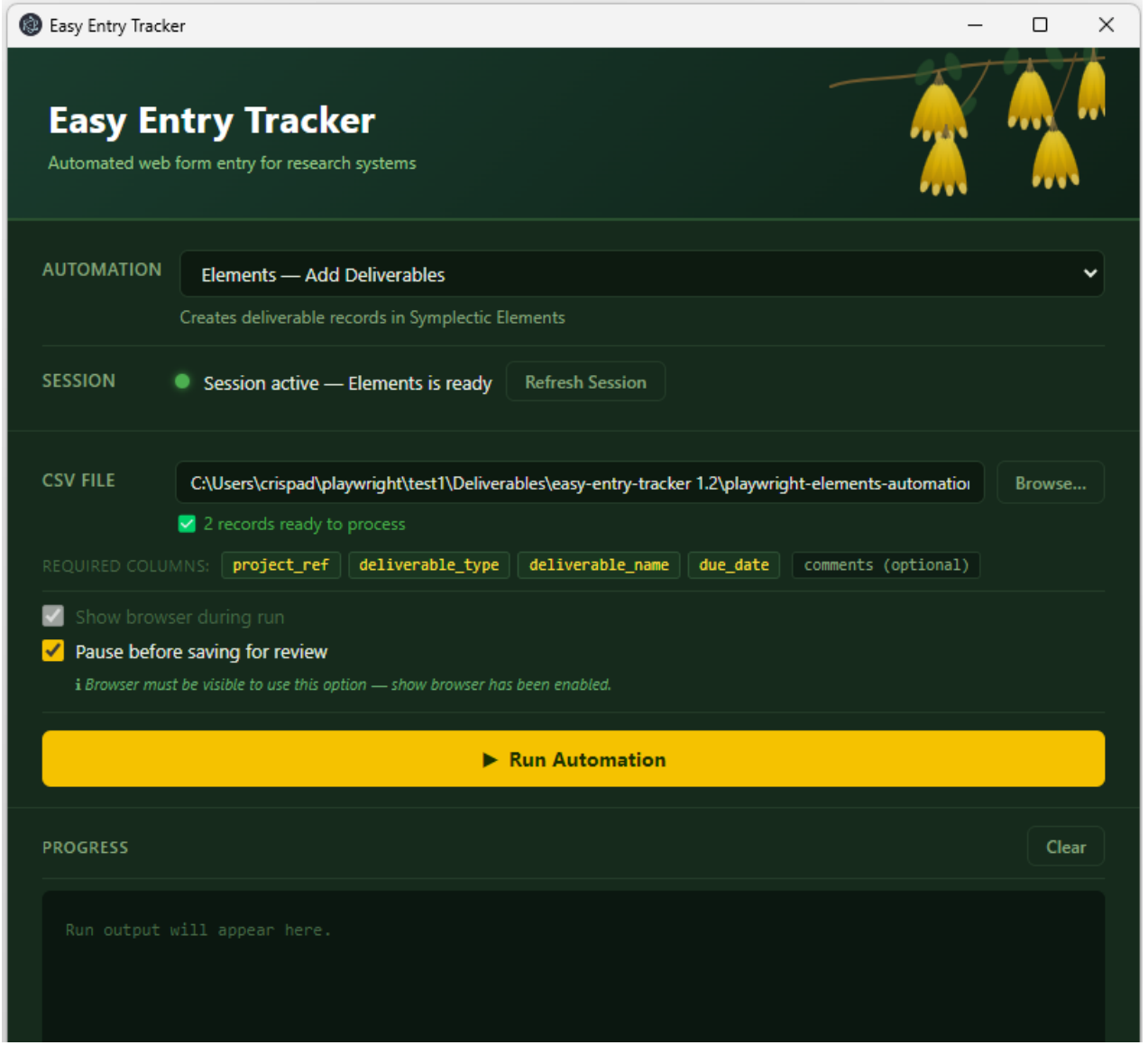
Design reflects considerations

Security, transparency

Ease of use, transparency

Users, transparency

Transparency, robustness



Questions?