Google Developer Group
Aivancity, Villejuif

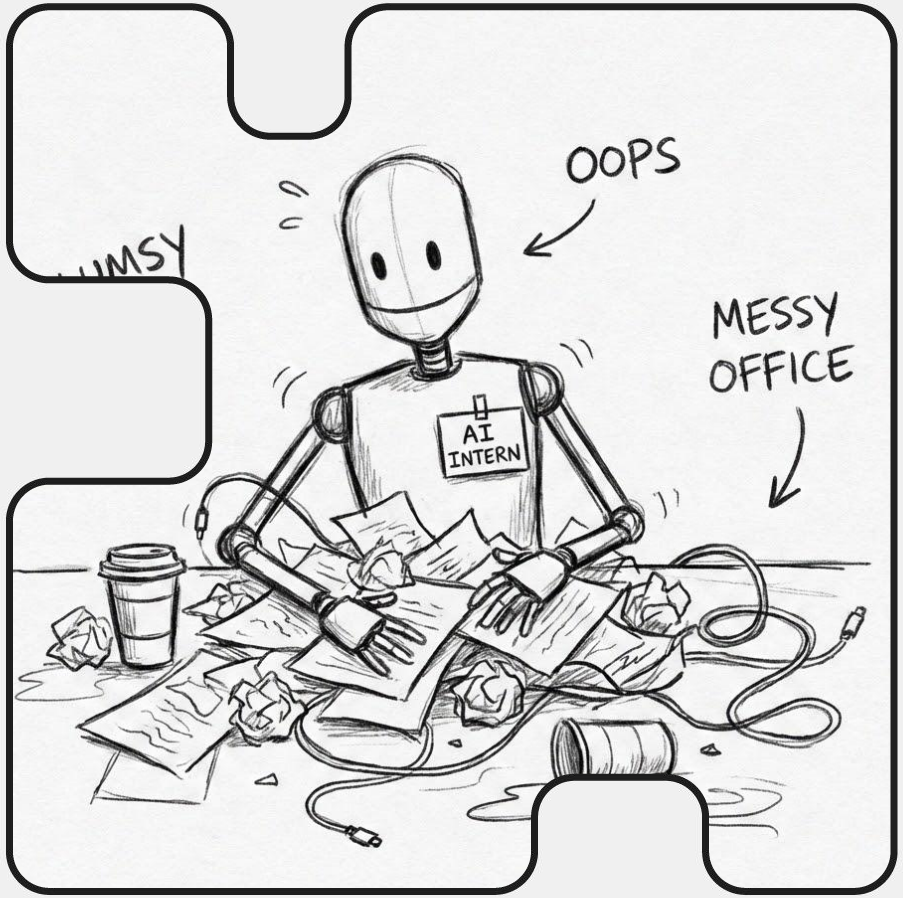
Readying the Web for Agents

Moving from screen-scraping to structured API contracts in the browser.

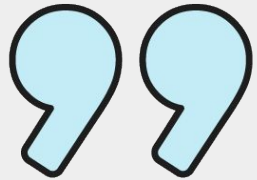
Build with AI

Chapter One

AI: The clumsy Intern

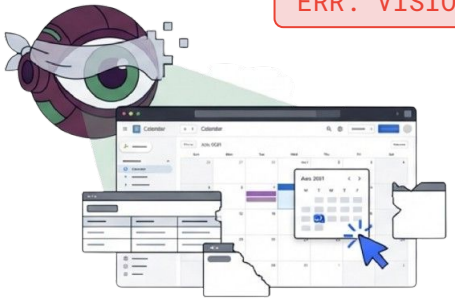


Is your AI just looking at the screen and guessing where to click?



Why current web agents fail to scale

ERR: VISION_MODEL



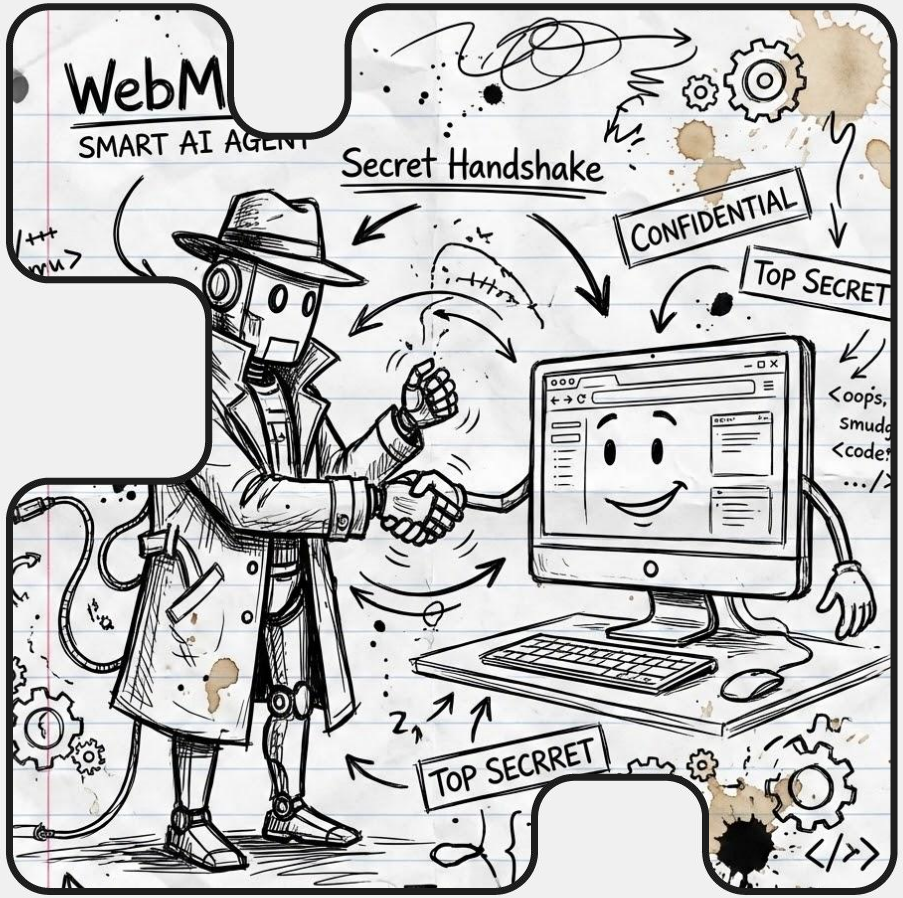
- Vision Models treat the web like a picture
- Slow, expensive, and brittle.
- Fail at complex UIs, hover states, and dynamic DOMs.

ERR: CUSTOM_SERVER

- Custom MCP server treats the web like a backend.
- High friction manual installation.
- Offers absolutely zero on-the-fly discoverability.

Chapter Two

WebMCP: Secret Handshake



Introducing WebMCP

A proposed web standard exposing structured tools for AI agents natively inside existing websites.

Introducing WebMCP

A proposed web standard exposing structured tools for AI agents natively inside existing websites.

Discovery

Agents query available tolls on the fly as they browse, without pre-installation.

Introducing WebMCP

A proposed web standard exposing structured tools for AI agents natively inside existing websites.

Discovery

Agents query available tolls on the fly as they browse, without pre-installation.

Schema

Explicit JSON definitions for input and expected outputs eliminate model hallucinations.

Introducing WebMCP

A proposed web standard exposing structured tools for AI agents natively inside existing websites.

Discovery

Agents query available tolls on the fly as they browse, without pre-installation.

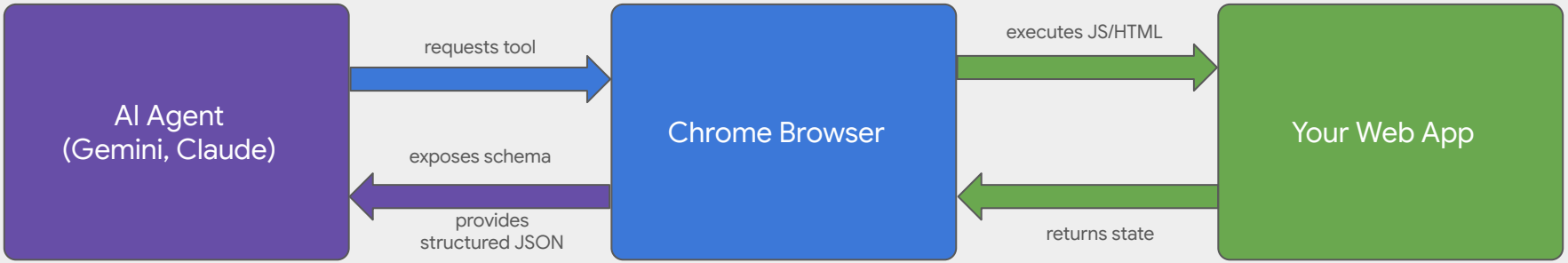
Schema

Explicit JSON definitions for input and expected outputs eliminate model hallucinations.

Discovery

Agents maintain a shared, real-time understanding of page context and available resources

How WebMCP routes Agent Intent



The agent processes natural language and outputs structured JSON.

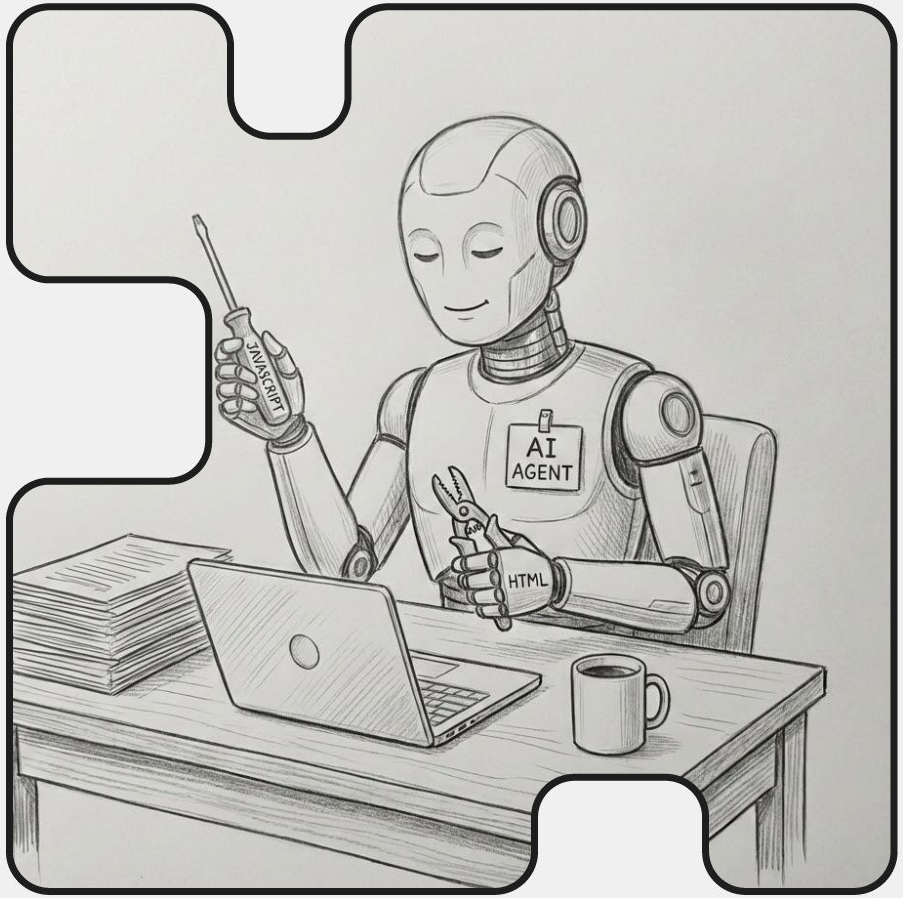
The browser acts as the mediator. It enforces the contract without exposing the backend

The web app executes native logic and updates the UI State.

Chapter Three

How it works: 2 flavours

JavaScript or HTML



Selecting your implementation path

Imperative API	Declarative API
(JavaScript)	(HTML)
• Use registerTool(). • Best for complex logic, dynamic SPAs, and fetching custom backend APIs. • Registered and unregistered dynamically based on page state.	• Use <form> annotation. • Best for standard web forms, simple search inputs and basic CRUD. • The browser automatically translates DOM elements.

Shared Benefit: Both Generate the exact same JSON schemas for the AI agents under the hood.

Registering complex logic with JS

```
const addTodoTool = {
  name: 'addTodo',
  inputSchema: {
    type: 'object',
    properties: {
      text: { type: 'string' }
    }
  },
  execute: ({ text }) => {
    return `Added todo: ${text}`;
  }
};
const controller = new AbortController();
navigator.modelContext.registerTool(addTodoTool,
{ signal: controller.signal });
```

Transforming HTML forms into agent tools

HTML Source

```
<form toolname="my_tool"
     tooldescription="A simple
declarative tool">
     <select name="select"
          required
          toolparamdescription="A
nice description">
</form>
```

Computed Schema

```
{
  'name': 'my_tool',
  'description': 'A simple
declarative tool',
  'parameters': {
    'type': 'object',
    'select': {
      description: 'A nice
description',
    },
    'required': ['select'],
  }
}
```

Transforming HTML forms into agent tools

HTML Source

```
<form toolname="my_tool"
     tooldescription="A simple
declarative tool">
     <select name="select"
          required
          toolparamdescription="A
nice description">
</form>
```

Computed Schema

```
{
  'name': 'my_tool',
  'description': 'A simple
declarative tool',
  'parameters': {
    'type': 'object',
    'select': {
      description: 'A nice
description',
    },
    'required': ['select'],
  }
}
```