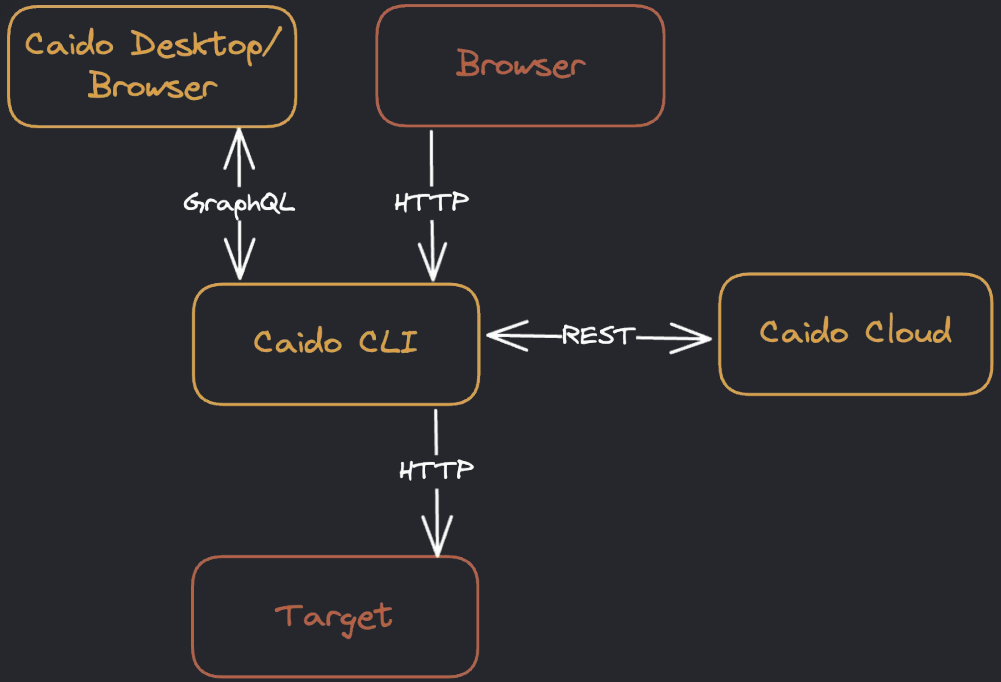
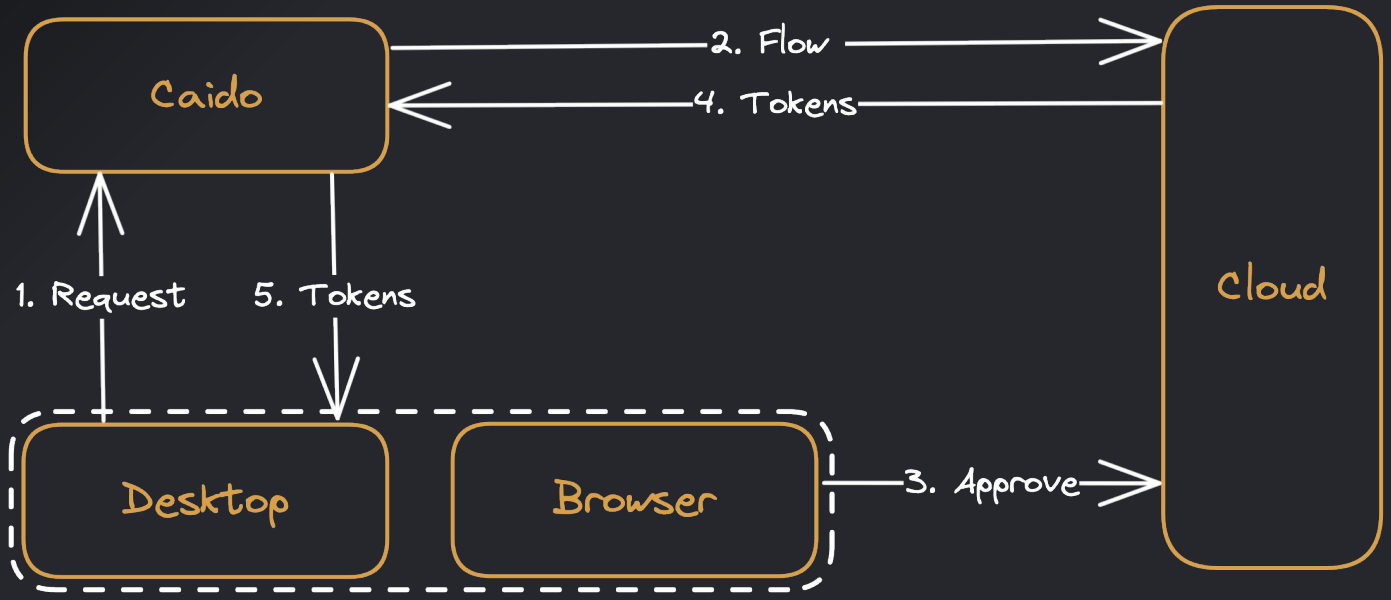
Masterclass

Part 1: Architecture & GraphQL

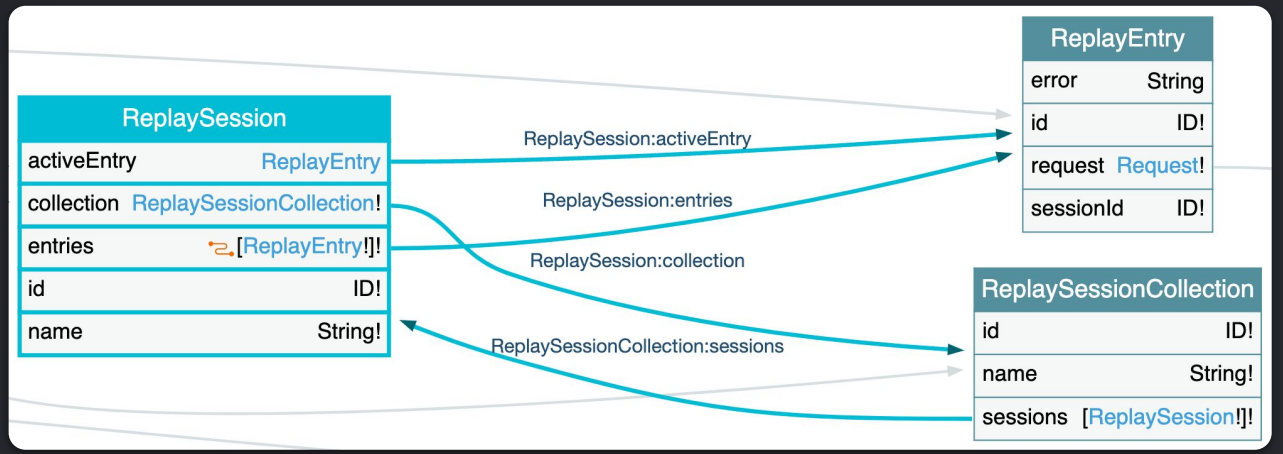

Caido Desktop/
Browser
Browser
GraphQL
HTTP
Caido CLI
REST
Caido Cloud
HTTP
Target



https://docs.caido.io/internals/authentication.html

- Schema: https://github.com/caido/caido/blob/main/plugin/schema.graphql
- Explorer: https://graphql-kit.com/graphql-voyager/
- Playground: http://localhost:8080/graphql
- Design
  - Mutations
    - Format: [present tense verb][Model] (deleteRequest)
    - Return: Payload with optional value and error
  - Query
    - Format: [model(s)] (requests)
    - Return: Object or Collection
  - Subscription
    - Format: [past tense verb][Model] (createdProject)
    - Return: Payload with snapshot

- Nesting
  - Entry => Session => Collection



- Connection: Used for lazy loading
  - Input
    - Pagination: Cursor (faster) or Offset
    - Filtering: Migrating to HTTPQL
    - Ordering & Scope: Custom for Caido
  - Output
    - Count
    - PageInfo
  - Example:
    - `requests(after: String, before: String, first: Int, last: Int, filter: FilterClauseRequestResponseInput, order: RequestResponseOrderInput, scopeId: ID): RequestConnection!`
    - `requestsByOffset(limit: Int, offset: Int, filter: FilterClauseRequestResponseInput, order: RequestResponseOrderInput, scopeId: ID): RequestConnection!`

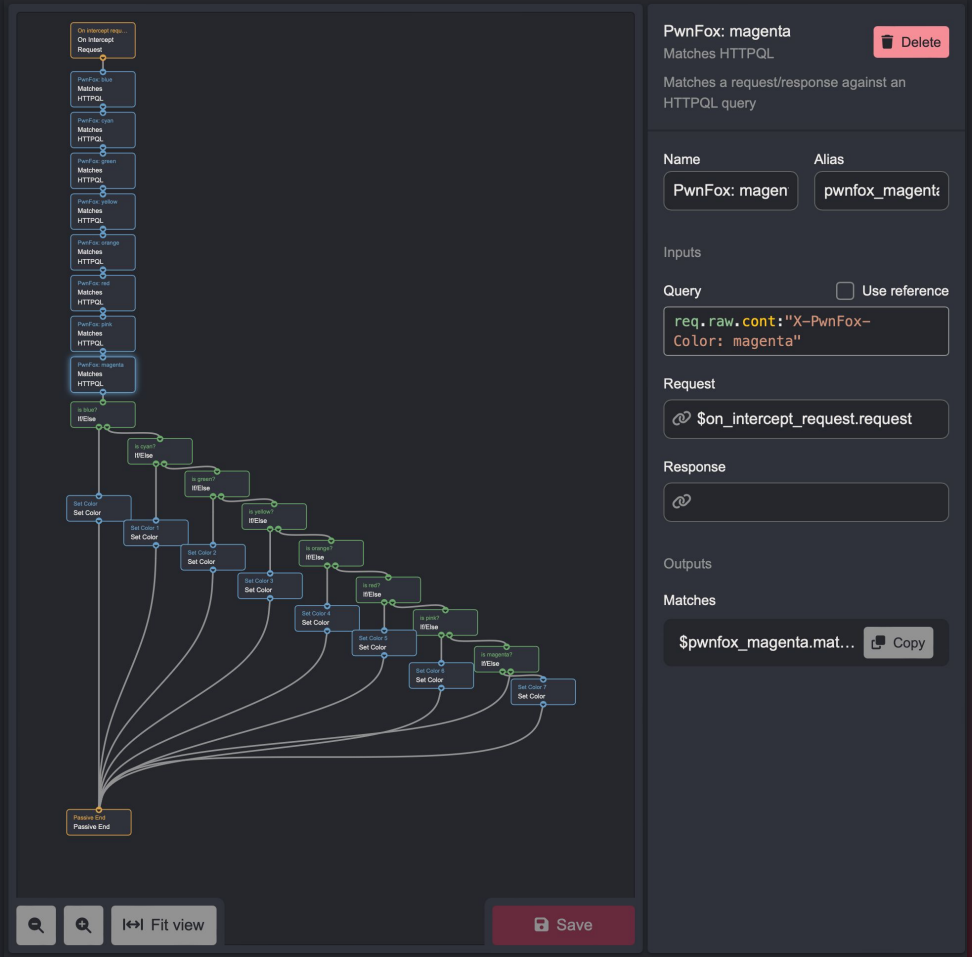
- Snapshot: Allows you to know if an operation was included in a result set
  - Query requests with snapshot 10
  - Subscription createdRequest with snapshot 9 (already in requests, ignore)
  - Subscription createdRequest with snapshot 11 (not in requests, process)

- Push/Pull data in/out of Caido
  - Integrations with external tools
- Frontend plugin with advanced features
  - Adding metadata to requests, displaying the results of a new tool, etc.
- Alternate interfaces for specific needs
  - Browser extension

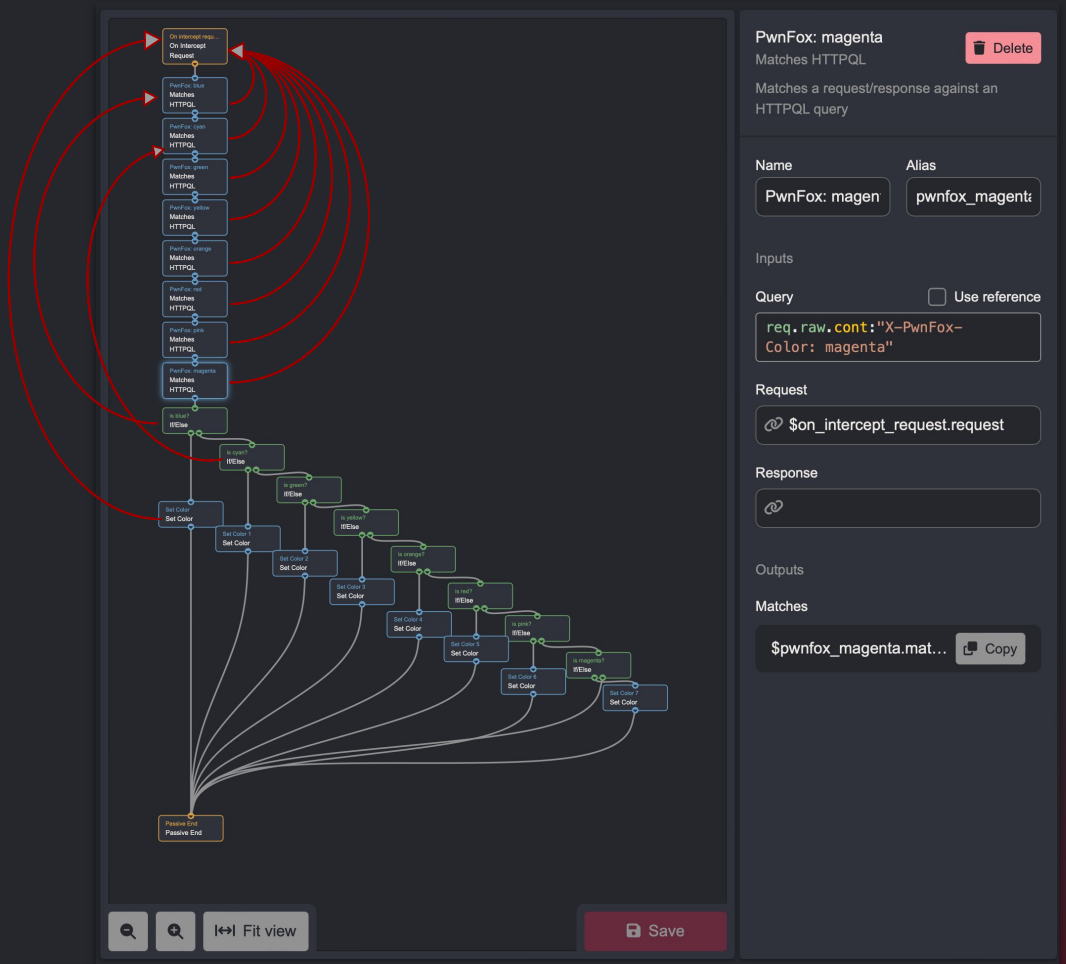
Part 2: Backend processing

- Current plan: https://github.com/caido/caido/issues/501
- Why workflows
  - Allowing everybody to create plugins
  - Modify existing plugins to fit your needs
  - Minimal usage of code, no setup required
- Beyond workflows
  - We are thinking about offering a “traditional” plugin interface
  - Let us know what you need

- Execution flow
  - Lines on the graph
  - Require a Control node to diverge



- Execution flow
  - Lines on the graph
  - Require a Control node to diverge
- Data flow
  - Each node has inputs and outputs
  - References link them
  - Doesn't require a direct relation



- JSON file representing a graph
  - https://github.com/caido/workflows/blob/main/convert/URL%20Decode/URL%20Decode.json

Node

```
{
  "id": 1,
  "alias": "end",
  "name": "End",
  "definition_id": "caido/convert-end",
  "version": "^0.1.0",
  "inputs": [
    {
      "alias": "data",
      "value": {
        "kind": "ref",
        "data": "$url_decode.data"
      }
    }
  ],
  "display": {
    "x": 0,
    "y": 230
  }
},
```

Edge

```
"edges": [
  {
    "source": {
      "node_id": 0,
      "exec_alias": "exec"
    },
    "target": {
      "node_id": 2,
      "exec_alias": "exec"
    }
  },
```

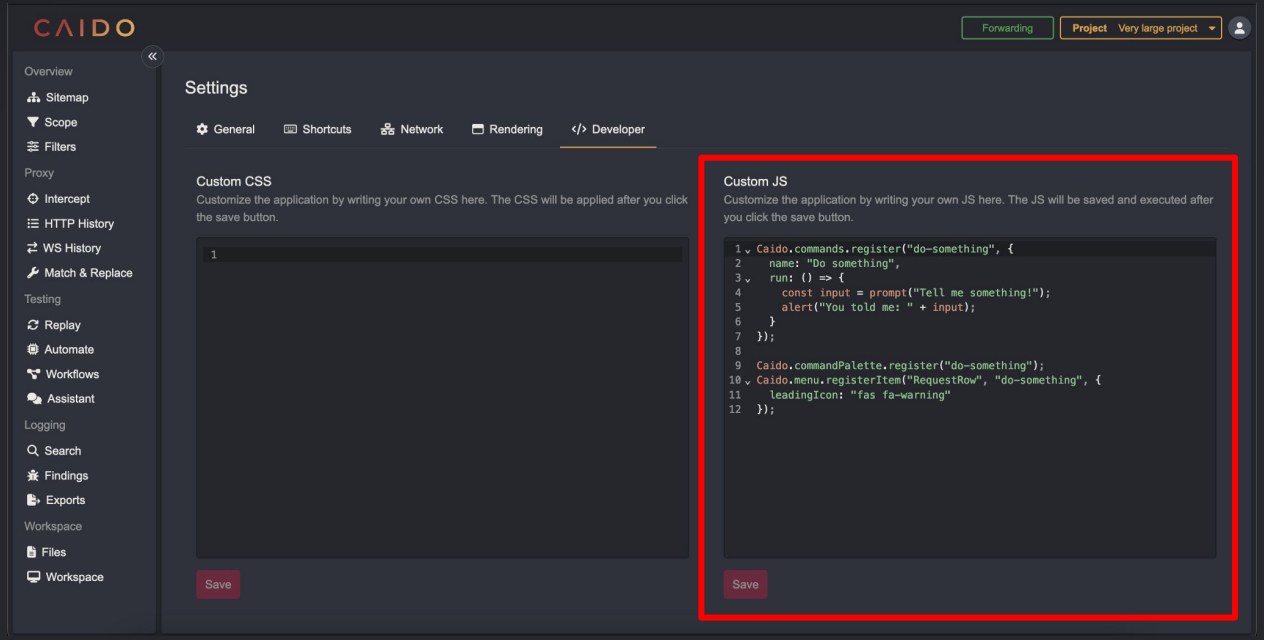
- Format: Bytes in => Bytes Out
- Special Nodes
  - Shell: Sends the bytes on the STDIN, reads STDOUT
  - Javascript: Run some javascript with QuickJS
    - https://bellard.org/quickjs/
  - Javascript If: Control node, returns a bool instead (will be replaced eventually)

```
export function run(input, sdk) {
  let parsed = sdk.asString(input);
  sdk.console.log(parsed);
  return parsed;
}
```

- Format: Request/Response In => No Output
- Only asynchronous for now
- Special Nodes
  - Shell: Sends the data as environment variables, reads STDOUT
    - CAIDO_REQUEST: base64 of request
    - CAIDO_URL: Url of the request
    - CAIDO_RESPONSE: base64 of response
    - *CAIDO_PROJECT: Project name
    - *CAIDO_HEADER__HEADER_NAME: For each header
  - Matches HTTPQL
    - Allows you to write a rule to match request and/or response
    - Pipe the result into an If/Else control node.
  - JS Node (coming soon)

Part 3: Frontend Plugins

Custom JS


CAIDO
Forwarding
Project Very large project
Overview
Sitemap
Scope
Filters
Proxy
Intercept
HTTP History
WS History
Match & Replace
Testing
Replay
Automate
Workflows
Assistant
Logging
Search
Findings
Exports
Workspace
Files
Workspace
Settings
General
Shortcuts
Network
Rendering
Developer
Custom CSS
Customize the application by writing your own CSS here. The CSS will be applied after you click the save button.
1
Save
Custom JS
Customize the application by writing your own JS here. The JS will be saved and executed after you click the save button.
1 Caido.commands.register("do-something", {
2 name: "Do something",
3 run: () => {
4 const input = prompt("Tell me something!");
5 alert("You told me: " + input);
6 }
7 });
8
9 Caido.commandPalette.register("do-something");
10 Caido.menu.registerItem("RequestRow", "do-something", {
11 leadingIcon: "fas fa-warning"
12 });
Save

Custom JS

```
onload = () => {
        const js = getCustomJS();
        eval(js);
}
```

Custom JS

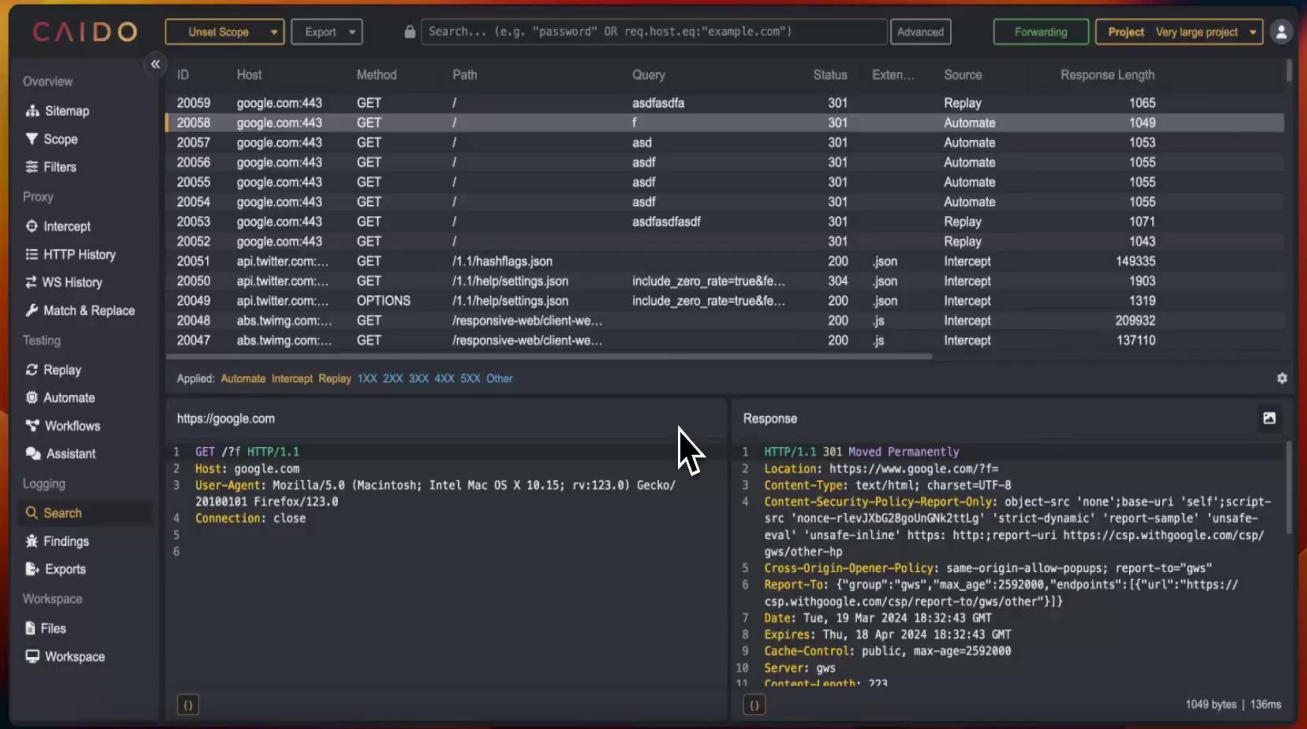
- Same functionality as the developer console
- Use it to manipulate the DOM to your liking
  - Add buttons/links
  - Perform GraphQL calls
  - Show/Hide features
  - Fix our bugs ???

window.Caido

- Heavily inspired by VSCode's API
  - Uses "Commands" as building blocks for customization

- Customize the command palette
- Customize context menus
- Customize the sidebar
- Create/Read/Update/Delete scopes
- ... more to come

Frontend API



Tips & Tricks

- Use the window.Caido variable as much as possible.
- When manipulating the DOM...
  - Use elements that have the `c-[...]` class prefix
  - Don't query `data-v-[...]` attributes, these are specific to VueJS and are unstable
- Add the `?safe` query parameter to load Caido without running the JS plugins
- Checkout the API docs here: https://github.com/caido/caido/tree/main/plugin/ui-api