A Map of Studio: Every Screen and What It Is For

Onboarding · https://docs.functionize.com/hc/en-us/articles/43802499690135-A-Map-of-Studio-Every-Screen-and-What-It-Is-For

Studio has nine screens in the sidebar and about a dozen places where real work happens. Nothing about it is hard once you know what each screen is for, but for the first week most people bounce between two of them and never discover the rest.

This is the map. One page, every screen, what it answers, and when you would actually open it.

functionize **Studio**
A map of Studio

www.functionize.com

THE PRODUCT IN ONE SENTENCE

A **conversation** produces a **test**; the test lives in a **project**; several tests run together as an **orchestration**; everything you ran shows up in **reports**. Workflows, extensions and MCP hang off the edges of that sentence and can wait.

EVERY SCREEN, BY HOW OFTEN YOU NEED IT

EVERY DAY	
Home	Prompt box, four quick-start buttons, project selector, recently viewed tests.
Tests	Every test, with filters for project, author, tag and status — and inline tag editing.
Test editor	Not a sidebar item. Three panes: agent, steps, browser and evidence.

EVERY WEEK	
Orchestrations	Groups of tests that run as one job, on demand or on a schedule.
Reports	The only screen that looks across projects. Analytics, not a live console.
Projects	Containers — and the home of the settings every test inside inherits.

OCCASIONALLY	
Sessions	Every conversation ever held. Come here to find out *why* a test looks as it does.
Workflows	Scheduled prose jobs that are not pass/fail tests.
Extensions	Under Assets. Your own Node, Python, Go or Java code.
MCP	Under Integrations. Connect your own AI tool to Studio.

A route through your first week. Home → build one test. Editor → read what it generated, not just whether it passed. Tests → tag it. Orchestrations → run two or three tagged tests as one job. Reports → look once you have something to look at.

Every screen, and how often you actually need it

www.functionize.com

The shape of the product in one sentence

You have a **conversation** that produces a **test**, the test lives in a **project**, several tests run together as an **orchestration**, and everything you ran shows up in **reports**.

Everything else — workflows, extensions, MCP — hangs off the edges of that sentence and can be ignored until you need it.

The screens you will use every day

Home	A prompt box, four quick-start buttons, a project selector and your recently viewed tests. Everything starts here. See Starting a Session: The Studio Home Screen.
Tests	Every test in the account, with filters for project, author, tag and status, and inline tag editing. This becomes your main screen faster than you expect. See *The Tests Screen*.
The test editor	Not a sidebar item — you get here by opening a test. Three panes: the agent conversation on the left, the steps in the middle, the browser and the evidence tabs on the right. See *Understanding a Test Run and Its Results*.

The screens you will use every week

Orchestrations	Groups of tests that run as one job, on demand or on a schedule. This is where a pile of tests becomes something that gates a release. See Orchestrations: Running a Suite as One Job.
Reports	The only screen that looks across projects. Pass rates, trends and a per-project table. Analytics, not a live console. See Reports: Pass Rates, Trends and What They Actually Tell You.
Projects	Containers, and more importantly the home of settings that every test inside inherits. See Projects: Boundaries, Defaults and What Inherits What.

The screens you will use occasionally

Sessions	Every conversation ever held, searchable. You come here when you need to know *why* a test looks the way it does. See Sessions: The Conversations Behind Your Tests.
Workflows	Scheduled jobs written as prose that are not pass/fail tests — checking a price, pulling a figure, confirming a service is up. See Workflows: Scheduled Jobs Written in Plain Language.
Extensions	Under *Assets*. Your own code in Node, Python, Go or Java, callable from a step when the browser cannot do the job. Most teams never need one. See Extensions: Running Your Own Code Inside a Test.
MCP	Under *Integrations*. Connection recipes for driving Studio from your own AI tool. See Getting Started with the Functionize MCP Server.

Where settings actually live

This trips up more people than any other part of the product, because there are two levels and they look identical.

Project settings	The defaults everything inside inherits. Four groups: **General**, **Execution** (timeouts, execution behavior and execution presets), **Access & Auth** (cookies, HTTP headers, basic authentication, certificate handling, mutual TLS) and **Advanced** (browser behavior).
Test settings	The same groups, minus execution presets, with an **Override project default** switch on the execution settings.

Change it on the project unless you have a reason not to. A timeout set on one test helps one test; the same timeout on the project helps every test somebody adds next month. Both screens have a **Search settings** box, which is faster than hunting through the groups. See Settings: Where Everything Lives and What to Change First.

Four things that are not screens, and are worth knowing early

Instructions and actions	You write one sentence; Studio generates several numbered actions from it (`1` becomes `1.1` , `1.2` , `1.3`). The gap between your sentence and those actions is where almost every surprise comes from. This is the single most useful thing to understand in the product. See *Operating the Steps Pane*.
Components	A saved sequence of actions you can drop into any test. They are shared across the whole account, not per-project, which makes naming them well a real favor to everybody.
Tags	Also account-wide. They drive the filters on the Tests screen and are how a suite stays current instead of being a list somebody hand-picked once.
Execution presets	Named sets of variables defined on a project. You can apply one to a single run, to a whole orchestration, or to a session — which is how one set of tests proves a flow against two different data sets. See *Execution Presets*.

Four statuses, not two

Studio does not only say pass or fail, and the two extra states are the ones that carry information.

Passed / **Failed**	The test reached a verdict.
Warning	The run completed with something worth looking at. Not a pass and not a failure. Warnings do not break anything, so they accumulate silently — which is exactly why they are worth a weekly glance.
Incomplete	Described by Studio as "*interrupted before finishing (timeout, crash).*" The test never reached a verdict at all, so it usually says nothing about your application.

Telling *failed* from *incomplete* is the difference between "the application is broken" and "something timed out". They appear as separate options in the status filters, and separately in an orchestration's re-run settings.

> **A route through the product for your first week**
> Home → build one test. Editor → read what it generated, not just whether it passed. Tests → tag it. Orchestrations → put two or three tagged tests in a job and run it on demand. Reports → look once you have something to look at. Everything else can wait.

Where to go next

- Your First 30 Minutes with Functionize Studio — the route above, with the buttons named.
- Your First Two Weeks: An Onboarding Plan — what to do after the first test works.
- Start Here: The Functionize Studio Exercise Series — eight short exercises against a demo store, no setup required.

- *Naming, Tagging and Not Drowning* — the conventions to agree before a second team joins.