Self-Healing and Timing: Controlling How Studio Recovers

Features · https://docs.functionize.com/hc/en-us/articles/43767855694615-Self-Healing-and-Timing-Controlling-How-Studio-Recovers

Studio adapts tests when your application changes — it finds a moved or renamed element instead of failing on a broken selector. How hard it tries, and how long it waits before giving up, are things you control. This article explains those settings, where they live, and when to change them.

Watch: Functionize Studio 101: Self-Healing Tests in Action
Self-healing happening to a real test. Worth watching before the section below on what self-healing will not do, which is the part that matters.



Watch: youtube.com/watch?v=by7tfSpSB3o

Where the settings live

There are two levels, and knowing which you are editing matters.

Project Settings	The defaults every test in the project inherits. Open a project and go to **Settings**.

Test Settings	Overrides for one test. Open the test and click the **gear** in the top toolbar, between Quick Select and Run Test.

In both, the controls are under **Execution → Execution behavior**.

At test level each control carries an **Override project default** toggle. Leave it off and the test follows the project. Turn it on and this test does its own thing. That is the whole model: set a sensible project default once, override only the tests that genuinely need it.

The two controls that matter

Timing model

"How long the AI waits for an element based on confidence. Aggressive = less wait, Conservative = more wait."



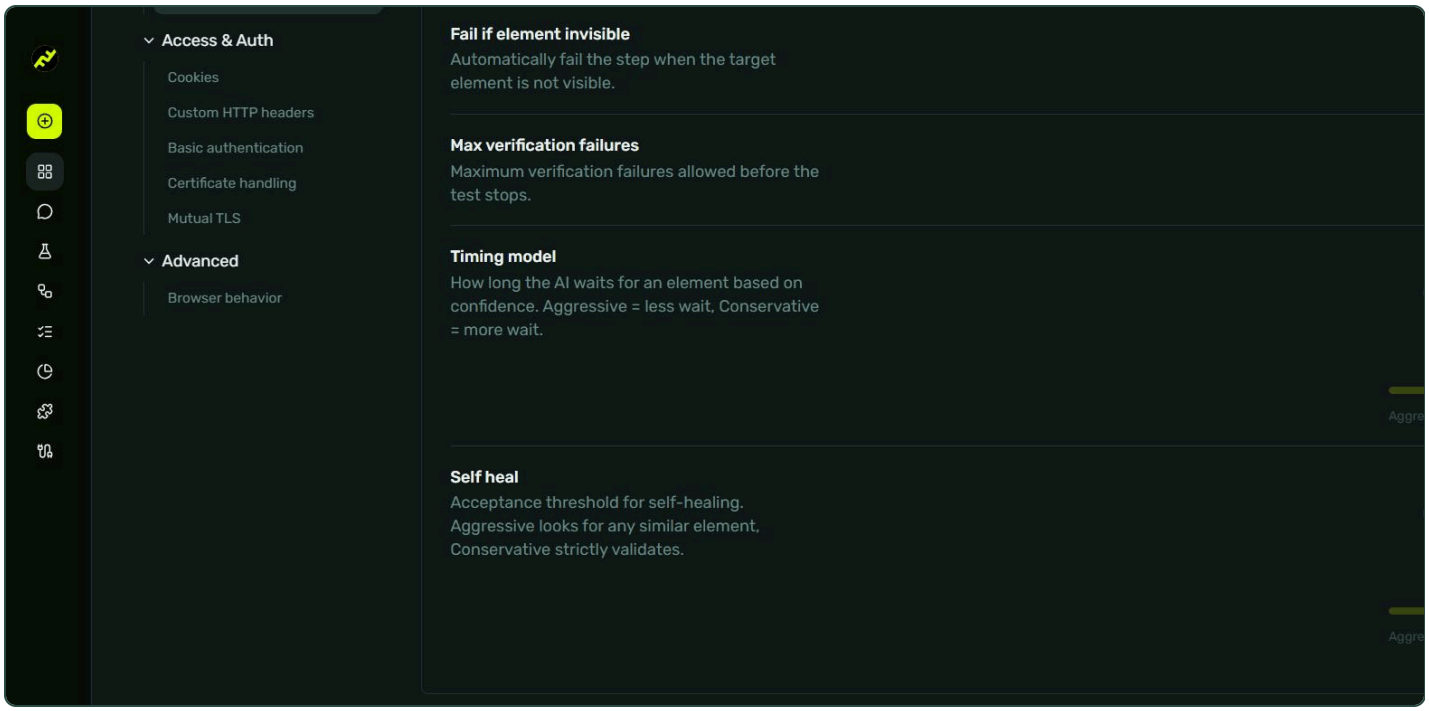
A slider from **Aggressive** to **Conservative**, defaulting to the middle. This is about *patience*, not correctness.

- **Move toward Conservative** when your application is slow, heavy with client-side rendering, or served from a distant environment. Tests take longer but stop failing on things that would have appeared a moment later.
- **Move toward Aggressive** when your application is fast and you want quicker feedback — and you would rather a genuinely missing element failed immediately than held the suite up.

If a test fails intermittently and the screenshot shows a page that looks half-built, this is the setting to reach for before anything else.

Self heal

"Acceptance threshold for self-healing. Aggressive looks for any similar element, Conservative strictly validates."



This is about *how sure* Studio must be before accepting a different element as the one the step meant.

- **Aggressive** accepts a looser match. Tests survive bigger redesigns, at the risk of quietly acting on the wrong element — clicking a similar-looking button that does something else.
- **Conservative** validates strictly. Fewer false matches, but more failures you have to look at when the page changes.

The trade-off is real and there is no universally right answer. A rule of thumb: the more consequential the flow, the more conservative you want it. A smoke test that checks a page renders can afford to be forgiving. A test that confirms the right amount was charged should not be guessing which element it is reading.

The rest of Execution behavior

The same screen carries four related settings worth knowing:

Continue on error	Keep going if a non-verify action fails. A second failure stops the test.
Continue on verification failure	Keep going even when a verify action fails — useful when you want every assertion's result in one run rather than stopping at the first.
Fail if element invisible	Fail the step when the target element is present but not visible.
Max verification failures	How many verification failures to tolerate before the test stops.

Alongside Execution behavior you will also find **Timeouts**, and under Advanced, **Browser behavior**. Project Settings adds **Execution presets**, for saving a combination you use repeatedly.

You can also just ask

You do not have to open Settings. The agent understands these in plain language, and the test editor's chat suggests it:

```
Be more conservative about fixing this test on its own.
```

```
This test is failing on timing - extend the page-load timeout.
```

Asking is often better than sliding, because the agent explains what it changed and you get the reasoning in the test's conversation, where the next person will find it.

What self-healing will not do

This is the important limit. Self-healing fixes **drift** — the test is trying to prove the right thing but looking in the wrong place, or too early. It will not fix a **disagreement**, where the test asserts something the application genuinely does not do.

No amount of aggressive healing will make a price check pass when the page shows a different price. In that situation the agent stops and tells you, rather than rewriting your assertion to match whatever it found — because doing that would silently delete the failure you wanted to catch.

That behavior is worked through, with a real failure, in Diagnosing a Failed Test in Functionize Studio, and you can reproduce it yourself in Exercise 5.

A sensible starting point

1. Leave everything at the project default to begin with. Defaults are fine for most applications.
2. If tests fail intermittently on elements that do exist, move **Timing model** toward Conservative at project level.
3. If tests keep breaking on cosmetic redesigns, move **Self heal** toward Aggressive — but not on tests that check money, permissions or anything destructive.
4. Override at test level only for genuine exceptions, and say why in the test's description so the next person knows.

See also Understanding a Test Run and Its Results for reading what a run actually did.