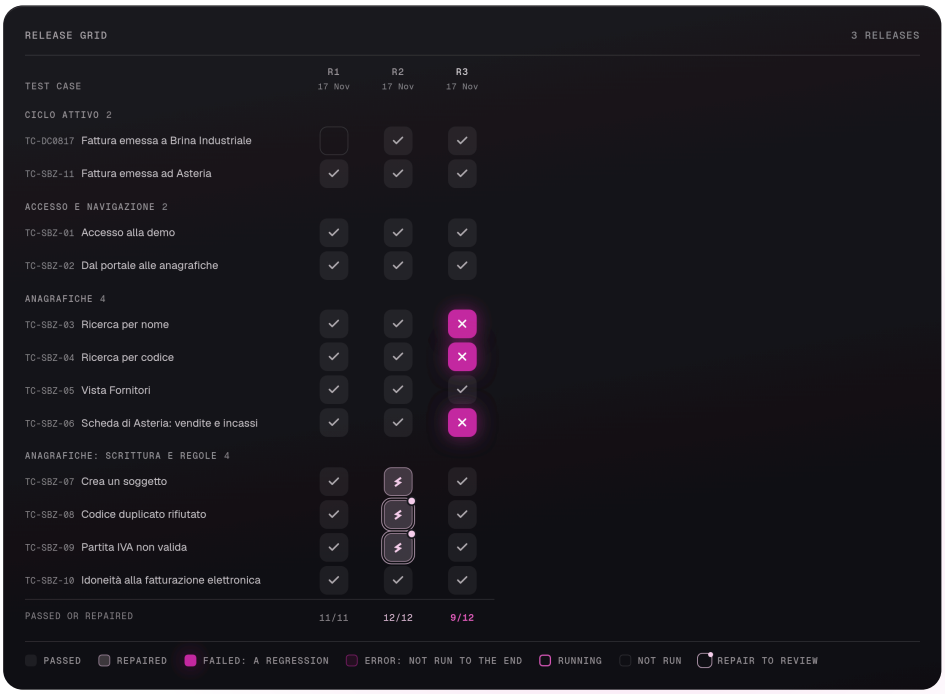
PRODUCT OVERVIEW · NOVEMBER 2026

Automated tests that repair themselves, inside your network.

Describe a test in plain language. SparkTest turns it into a browser script, runs it at every release and repairs it when the interface changes. Everything runs on one machine in your network.



Functions and regressions

SCREENSHOTS AND API TIMINGS FOR EVERY RUN

Inside your network

YOUR DATA STAYS ON THE MACHINE

Licensed per maintained test

RUNS AND REPAIRS ARE NOT COUNTED

Two questions at every release.

Do the new functions work? Do the old ones still work? The second question has to be asked again at every release, across the whole application: by hand nobody can repeat it.

Internal applications	Management systems, project portals and test environments are not reachable from a cloud service.
Data that cannot leave	Screenshots, records and credentials have to stay inside the company network.
Tests that age	Classic scripts break at every change of the interface, and someone has to repair them.

02 HOW IT WORKS FIVE STEPS · THREE OUTCOMES

From a sentence to a script that keeps working.

SparkTest answers with one machine in your network, which reaches your applications the way a user does. **The AI writes the test, code runs it:** the local model works when a test is written or repaired; every run in between is a script, with no model and no tokens. Every run ends with one of three outcomes.

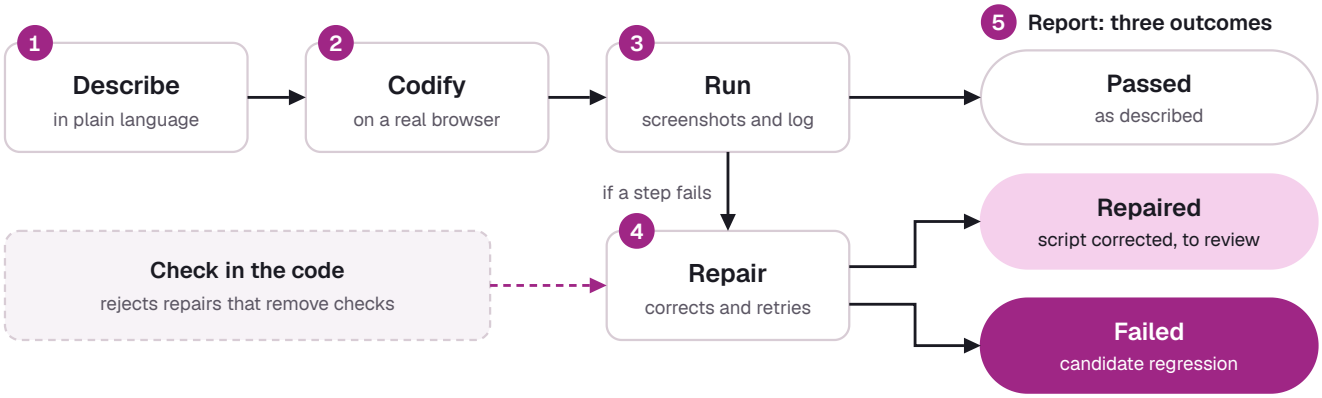


FIGURE 1 The cycle of a test. A repair is checked in the code and reported apart: it never counts as a plain pass.

OUTCOME

Passed

The application behaves as described.

WHAT TO DO

Nothing.

OUTCOME

Repaired

The interface changed, the behaviour did not. The script was corrected.

WHAT TO DO

Someone reviews the difference between before and after.

OUTCOME

Failed

The behaviour changed and a repair is not enough.

WHAT TO DO

It is the candidate regression, with screenshots, API calls and the log.

Whoever knows the application writes the test.

«Sign in as a planner, open Orders, filter the Supplier column by ACME Steel, check that a matching order is listed.» That is a test case.

The documentation, once	The application's documentation is loaded once and used by every test, instead of being repeated in each one.
Suggested test cases	The local model reads the documentation and the test cases you already have, and proposes the ones that are missing, in the language of your cases. A value it cannot find in your sources is flagged. Nothing is added until you choose.
An assistant	It helps sharpen a description that is ambiguous, before it is codified.
Credentials apart	They live in a dedicated document and are injected when the test runs. They never enter the model's prompts or the scripts.

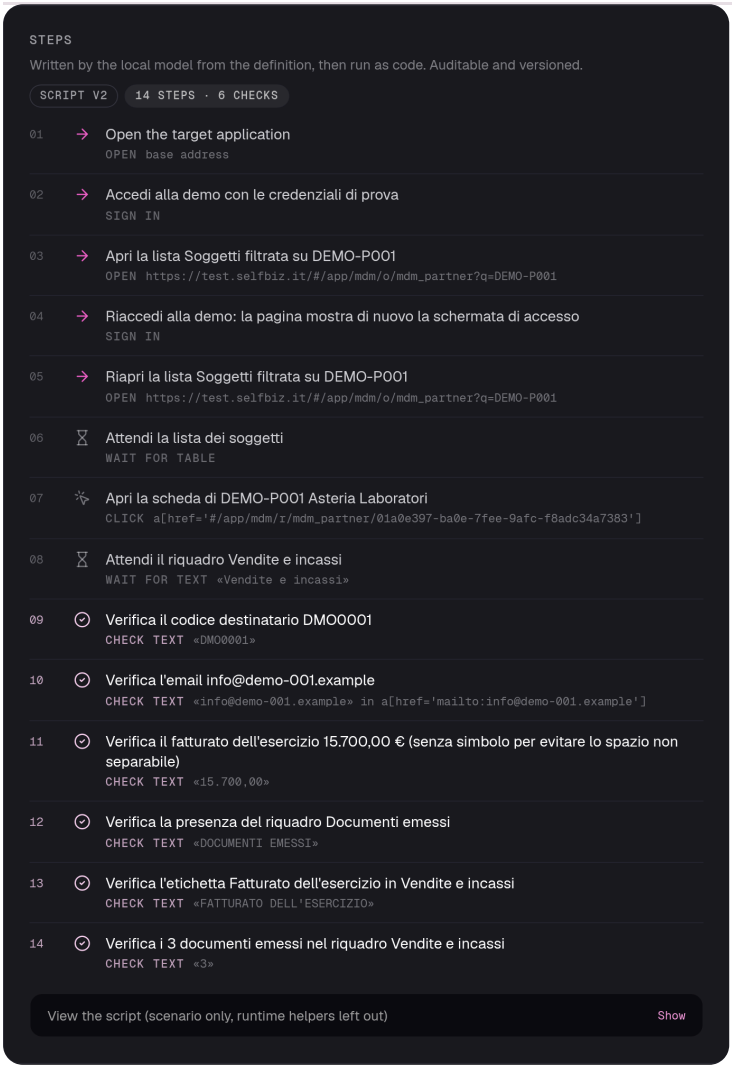


FIGURE 2 The steps of a test case written in Italian, as the console shows them: written by the local model from the definition and run as code. Console screenshot.

The agent opens a real browser on the application and reads the page: fields, labels, buttons, tables. It asks the local model for the next few steps, runs them at once and keeps only the ones that worked. A step that fails goes back with the error and the page as it is now. Fixed budgets bound the exploration.

The result is a readable script: a plan of verified steps and the runtime that executes it, in one JavaScript module, versioned with the test case. From the console it downloads as one file with a small runner, which runs with Node and Puppeteer without SparkTest.

For applications built on metadata, recurring interactions are maintained actions, not guesses: sign in and profile, catalog and grids, filters and records.

Every run keeps the proof.

Every step, on screen	A screenshot for each step of the scenario, in order: what the browser saw, not what the script hoped.
Every API call, timed	Method, address, status, size and duration of the calls the application made, recorded while the test runs. Errors and slow calls stand out.
A diagnosis when it fails	The local model reads the failing step, the error and the API calls around it, and writes a short diagnosis in the language of the test: probable cause, evidence, where to look.
Before and after	Each step of the run next to the same step of the last passed run of the test case, as a slider or side by side. The comparison opens on the step that fails.

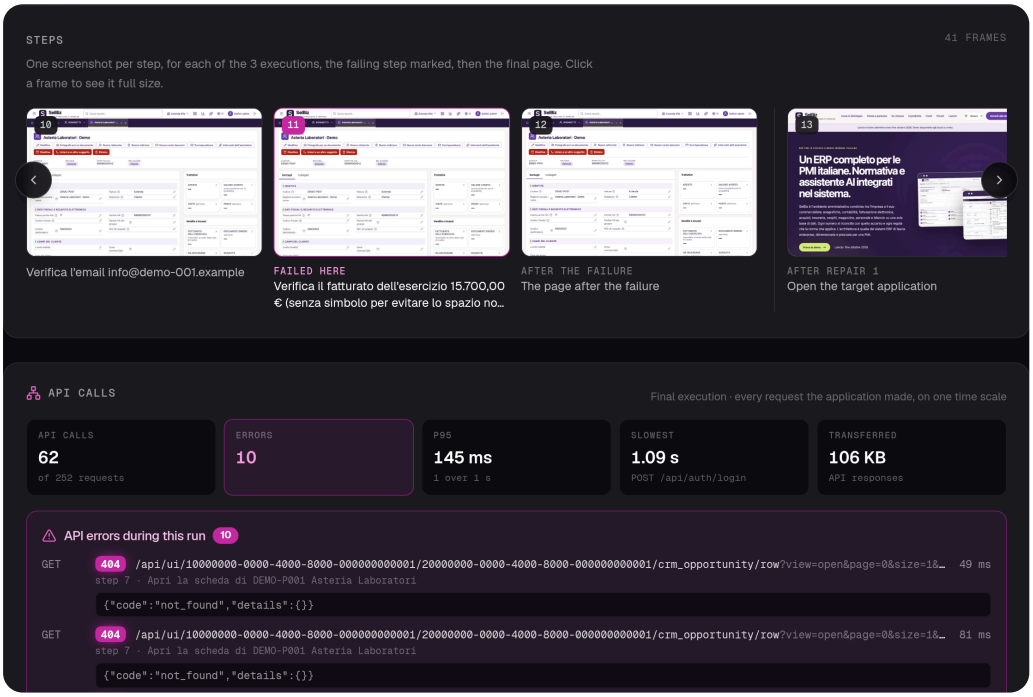


FIGURE 3 The test case of section 03 in release R3, a failed run as the console shows it: the screenshots around step 11, where it failed, then the API calls of the run, ten of them answered 404. Console screenshot.

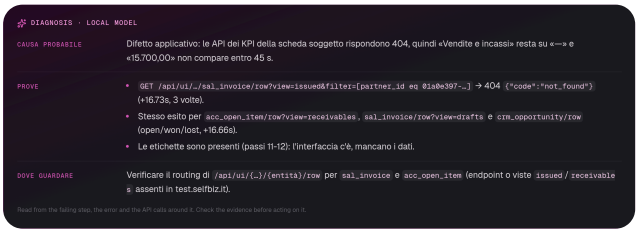


FIGURE 4 The diagnosis the local model wrote for that run, in the language of the test: an application defect, the calls answered 404 as evidence, where to look. Console screenshot.

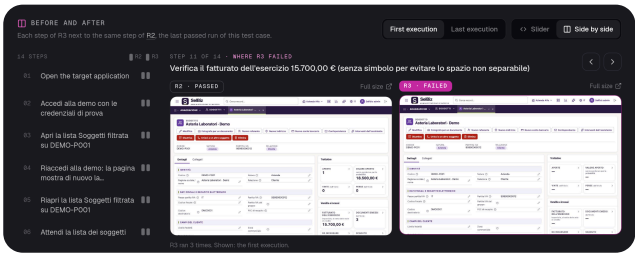


FIGURE 5 Before and after, side by side: step 11 in release R2, passed, next to the same step in R3, failed. In R2 the panel shows the turnover, in R3 only dashes. Console screenshot.

05 REPAIRS UNDER CONTROL CHECKED · SHOWN · REVIEWED

Every repair is checked, and shown.

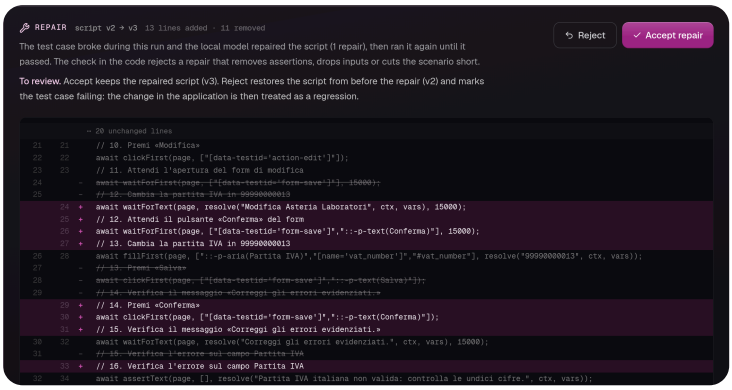


FIGURE 6 A repair waiting for review, as the console shows it: in release R2 the form button «Salva» became «Conferma», so the repaired script waits for the form and presses «Conferma». The value it types and the checks are the same. Console screenshot.

A check in the code compares the corrected script with the previous one and rejects it when assertions were removed, most of the inputs were dropped or the scenario was cut short. **A repair may change selectors and waits, never what a test checks or the data it types.** A rejected repair ends the run as failed, with the reason.

A repair that passes the check is reported as **repaired, never as a plain pass,** with the difference for a person to review. Accepting it keeps the new script; rejecting it brings the previous one back and marks the test as failing.

06 API INSIGHTS MEASURED AT EVERY RELEASE

Your APIs, measured at every release.

Per endpoint	The API calls recorded by the tests become a measure of your APIs: calls, calls per run, errors, p50, p95 and payload size, with the p95 of each of the last releases.
Signals	Endpoints that started failing or slowing down, pages that call the same endpoint many times (N+1), heavy responses, a parameter that a release stopped sending.
Tips from the local model	On request, the SparkNode reads the statistics and writes up to six tips, by priority, pointing to the endpoints concerned. The traffic never leaves the machine.

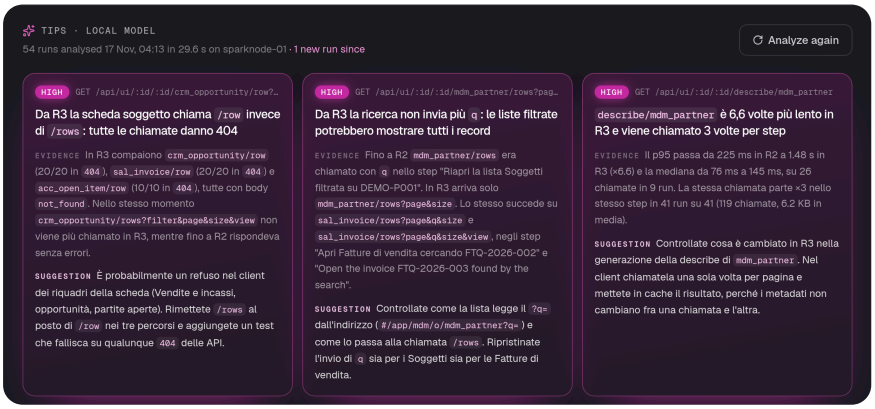


FIGURE 7 Six tips written by the local model on the SparkNode from 54 runs, in the language of the test cases; here the three with high priority, all about release R3: calls answered 404, a search that no longer sends its text, a call 6.6 times slower. Console screenshot.

It runs when you release. It tells the right people.

From your pipeline	One call with the release name starts the whole suite of an application. Each application has its own token, shown once and stored only as a hash.
Every night	The suite runs on its own at the time you choose, labelled with the date.
In parallel	The SparkNode runs several tests at the same time, as many as the size allows: 2 on S, 4 on M and 8 on L.
The release grid	Tests in rows, releases in columns, an outcome in each cell. Repairs waiting for review are marked; open regressions and repairs to review have their own lists.
Repairs to approve	A person accepts or rejects each repair. A rejected repair counts as a regression.
Notifications	Regressions, repairs to review, API errors and slow calls, API insights ready, completed releases, a SparkNode that stops answering, licence limits and expiry. In the console, by email, on Teams or Slack, or through a signed webhook. Each person chooses which events reach them, as they happen or in a daily digest, and can mute an application.

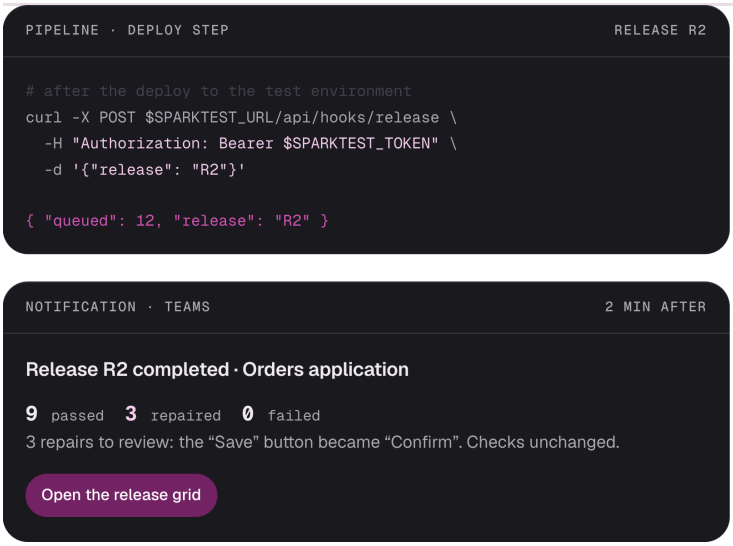


FIGURE 8 The pipeline announces a release; the summary reaches the team when the suite is done. Illustration with example data.

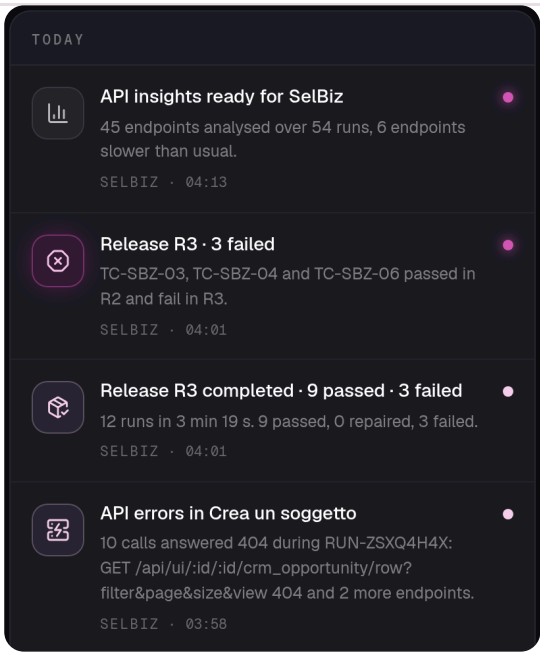


FIGURE 9 The notification centre of the console after release R3, newest first: API insights ready, three test cases that passed in R2 and fail in R3, the summary of the release, the API errors of a run. Console screenshot.

08 WHAT WE INSTALL ONE PACKAGE · ONE MACHINE

One machine, inside your network.

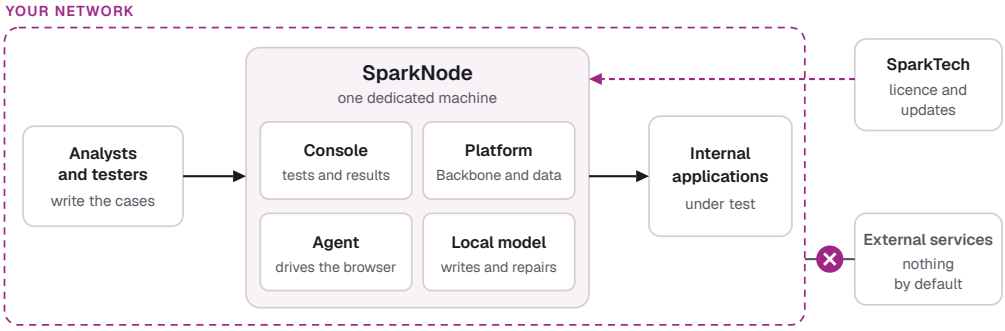


FIGURE 10 Everything that touches your applications runs on the SparkNode, a dedicated machine in your network. From outside, only the licence and the updates come in.

Console	The web application where tests are written, runs started and results read.
Platform	The SparkTech Backbone platform and its database: applications, tests, runs and history.
Agent	Codifies, runs and repairs the tests, driving a browser without a screen.
Local model	Writes and corrects the scripts, diagnoses failures, suggests test cases and API tips, on the machine itself: no external service, nothing billed per run.
Licence	A signed file that sets size, tests, applications, parallel runs and expiry. It is verified on the machine, offline; its limits apply when tests and applications are created.
The machine	A configuration we specify for each size, bought to our specification or supplied by us. Local inference needs memory and an accelerator.

09 SECURITY BY DESIGN

Your data stays where it is.

Data stays home	Screenshots, records, API calls and credentials stay on the SparkNode. What comes in from SparkTech is the licence and the updates.
Credentials	Never in the model's prompts or in the scripts: a script refers to them by name, the values are injected when the test runs and masked in the recorded API calls.
Documentation	When the local model reads it to suggest tests or write API tips, documents with passwords are left out and lines that mention credentials are withheld.
Secrets	Pipeline tokens and node keys are stored only as hashes. Webhooks are signed, so the receiver can check where they come from.
Repairs	Never a plain pass; rejected if it changes what a test verifies or the data it types.

Same quality, three capacities.

The sizes differ in capacity, not in quality: the model is the same on all of them, and so is the quality of codifying and repairing.

SIZE	S	M	L
For	A team, one or two applications	A department, several applications	An application portfolio or a programme
A month, annual contract	€590	€1,690	€3,900
Maintained tests	**100**	**400**	**1,500**
Applications	**2**	**10**	**No limit**
Parallel runs	**2**	**4**	**8**
50 more tests, a month	€350	€250	€150

Subscription prices, in euros, VAT excluded. Hardware, installation and configuration are excluded and quoted separately. The full terms are in **SparkTest on premise: sizes, pricing and terms**.

Included	The licence, the updates to the codifying engine, to the maintained actions and to the model, and support.
Not counted	Runs, repairs and console users.
A maintained test	A test case that SparkTest keeps running and repairing: up to 25 steps and 5 minutes. A longer journey counts as more than one test.
In the cloud	For applications on the internet, the same product runs on SparkTech's nodes, from €149 a month.

THE FIRST STEP

Start with one application.

We install the machine, connect your environments and codify the first tests together with your team.

WRITE TO US
info@sparktech.it
PEC sparktech@legalmail.it

ONLINE
sparktech.it
autotest.sparktech.it

COMPANY
SparkTech S.R.L.
Corso Valdocco 2, 10122 Torino
P. IVA 02835910023