AUTOMATIZE · AI-FIRST TESTING PLATFORM

# Test Guard.

## Describe the test. Ship the test. Skip the boilerplate.

Author browser, API and behaviour-driven tests from a natural-language prompt, a live recording or a specification. Run them across three browser engines and three viewport classes. When one breaks, stop on the broken step with the page still live, take an AI-proposed repair, and keep it as a version — never a silent mutation.

UI · REST · SOAP · BDD | Live debugger | Self-healing, versioned | Self-hosted | 43 agent tools

| 3 | 9 | 5 | 14 |
|---|---|---|---|
| TEST TYPES, ONE SUITE | ENGINE × VIEWPORT RUNS | CODE EXPORT LANGUAGES | INTERFACE LANGUAGES |

QA ENGINEERING · PLATFORM TEAMS · AGENCIES · REGULATED IT

PRODUCT BROCHURE 2026.1

THE PROBLEM

# Automated testing fails on maintenance, not on authoring.

Writing the first hundred tests is a project anyone can finish. Keeping them green through a year of front-end change is what actually defeats teams. A selector moves, forty tests go red, someone spends a morning finding out that thirty-nine of them were fine — and the suite quietly stops being trusted, which is the same as not having one.

### Brittle selectors

Recorded tests anchored to markup that was never meant to be an interface contract.

### Blind failures

A red run, a stack trace and a screenshot — but the page that broke is long gone.

### Split tooling

One tool for the browser, another for the API, a third for the scenarios, three sets of history.

### Silent repairs

Self-healing that rewrites the test without telling you is not maintenance, it is drift.

THE SOLUTION

## Five ways to author. One version chain. A debugger that holds the failure open.

1. **Author**
   Prompt, record, write a scenario, edit code, or generate a whole programme of suites from a URL and a description.
2. **Run**
   Chromium, Firefox and WebKit across desktop, tablet and mobile viewports — each combination its own recorded run.
3. **Debug**
   Stop on the broken step with the page still in the state that broke it. Watch expressions, the page's own console and network, an element picker.
4. **Repair**
   Fix in place, pick the element from the page, or ask the model. The repair lands in a draft and only becomes a version when you save it.
5. **Promote**
   Deploy the version through your environment chain, with drift visible on the badge before it is visible to users.

THE SHARPEST CAPABILITY

### The page is still live when you get there.

Most tools hand you a screenshot of a corpse. TestGuard pauses *on* the failing step with the browser still open and the application still in the failing state — so you can evaluate an expression against the real page, read the console as it actually ran, point at the element you meant, and retry the step without re-running the whole test.

**Repairs are versioned, never silent.** Every change — created, edited, re-authored, self-healed, re-recorded, imported, saved, restored — is one of eight immutable version kinds, with an author and a changelog. Restoring an old version creates a new version rather than rewriting history.

### Grounded generation

Before writing a plan, the platform opens a real browser and reads what the application actually contains — so the suite is grounded in your app, not in a guess.

### Selector discipline

Recording prefers a test id, then a stable id, then a role and name, then a label — and verifies each candidate matches exactly one element before keeping it.

### Human-approved plans

Whole-programme generation produces a named plan of suites and tests for you to approve first, then builds it as a tracked job.


© 2026 Automatize · developed together with A4IT & A4ME

AUTHOR & EXECUTE

# Three kinds of test, one suite, one history.

## 1 Authoring surfaces

- **Prompt** — describe the test and the target; get structured steps and assertions as version one
- **Record** — a live server-side browser captures your interactions, with typing accumulated per field and duplicate events negotiated away
- **Feature** — write or generate Gherkin; recorded steps convert into scenarios plus the glue those sentences need
- **Code** — a syntax-highlighting editor with Playwright-aware completion, inline AI completion, and explain, refactor, fix and extend actions
- **Import** — from an OpenAPI specification or a WSDL
- Steps and code round-trip exactly: what you edit in one view is what you see in the other
- Export to **TypeScript, JavaScript, Python, Java or C#**

## 2 Execution

- Chromium, Firefox and WebKit from warm pre-launched browsers — never a cold start per request
- Desktop, tablet and mobile viewport classes; every engine and viewport pair is its own run record
- REST tests with bearer, basic and API-key authentication, headers, bodies and JSON path assertions
- **SOAP tests** with envelope construction, namespaces and fault capture — rare in a modern tool, and decisive for a legacy estate
- Structured assertions on status, header, body, latency, schema, URL, outcome and text — never free-form script
- **Visual regression** with per-engine, per-viewport baselines, pixel and ratio tolerances, and a diff painted over the dimmed baseline
- Run settings on the version: console and error listeners, request blocking and mocking, timeouts, slow motion, tracing, stored sessions
- Dry run — execute an API draft through the real executors and write nothing at all
- Behaviour-driven execution with exact scenario semantics and standard report output

## 3 The live debugger

- Breakpoints on any step; run from here, run only this, step over, pause, slow motion
- “Debug this failure” from a failed run lands directly on the broken step
- Watch expressions evaluated in the live page
- The page's own console and network as it ran; element-under-pointer picker
- Fix in place and retry, or request an AI repair against the failing step, the error and the live markup
- Every ordinary run is also debuggable — pause, next and continue, with pause-on-failure

## 4 Versions, environments, promotion

- Eight immutable version kinds with author, changelog and environment badges
- Structured step and assertion diff between any two versions
- Environment chains per project — development through production, each with its own targets and credentials
- Promotion with automatic update-versus-new detection and a full deployment audit
- **Drift badges** computed from the version chain, never stored as a stale flag
- A pipeline view across the chain showing deployed versions, health and drift

**Saved browser sessions as first-class credentials.** Authenticated state is captured, encrypted in the database, and scoped to a project, an environment and a name. A run that asks for a missing or expired session fails immediately *saying so* — instead of failing three steps later on an unexpected login page, which is where a working day usually goes.


© 2026 Automatize · developed together with A4IT & A4ME

OPERATE & REPORT

# Scheduling, analytics and agent access.

## 5 Scheduling & smoke testing

- Schedules are data, not deployment configuration — created, edited and scoped to one test or one suite in one environment
- Cadence read in the schedule's own timezone
- **Exactly-once firing**: a week of downtime wakes to one run, not a backlog of two hundred
- Daily smoke test with today's status, a pass streak and a three-week pass/fail calendar
- Failure notification when a smoke run breaks

## 6 Email reporting

- Per-project configuration with an optional per-suite override — a weekly stakeholder digest and an on-call list for one critical suite
- **16 report sections** — summary, indicators, outcome breakdown, type split, failed tests, failure analysis, results, slowest, most failing, flakiest, environment health, API health, trend, version info, artefact links, footer
- Each section declares whether it fits a single run or a time window, so the interface cannot offer a block the report could not fill
- Triggers on a finished suite run, a scheduled run, a manual action or a timezone-aware digest
- Send always, on failure only, or never — with manual sends overriding the gate
- Every attempt audited with the recipients and sections as they were at send time
- Five delivery transports behind one interface, configured per organisation

## 7 Analytics

- Pass rate, run count, average and 95th-percentile duration
- Outcome breakdown and the split across browser, API and scenario tests
- Pass-rate, volume and duration trends; per-environment health cards
- Leaderboards for most-failing, slowest and flakiest tests
- API health: status-code distribution, average and 95th-percentile latency, uptime, protocol split, slowest endpoints
- Filterable by all, scheduled or manual triggers

## 8 Agents & integration

- A native **Model Context Protocol server** exposing **43 tools** across catalogue, authoring, administration, runs, execution, analytics and deployment
- Six capability tiers, sixteen distinct permissions, checked both when listing tools and again when calling one
- An agent key can only narrow the rights of the person who issued it
- Rows in another organisation are reported as *absent*, not forbidden — so the platform cannot be mapped by watching refusals
- Every call logged, allowed and denied alike, with secret-named arguments elided
- **343 REST endpoints** with OpenAPI, behind authentication

### Seven model providers

Including three that run entirely on your own network. Configured per project with a purpose, a default and a timeout — and keys encrypted at rest.

### Deep white-labelling

19 brandable colour tokens with derived tints, logo, font, tagline and links — resolved field by field down organisation, tenant, platform and default.

### Installs itself

A seven-step wizard that boots without a database, creates it, runs the migrations, makes the first account and writes a service unit.


© 2026 Automatize · developed together with A4IT & A4ME

WHO BUYS IT

# Teams whose release confidence is a business risk.

### In-house QA & platform teams
Mid-market and enterprise engineering organisations running regression, smoke and contract testing across several applications and environments.

### Digital agencies
Many client projects under one deployment, each isolated, each brandable, each with its own environment chain — the tenancy model is built for reselling.

### Banking & insurance
Large SOAP estates that modern testing tools simply do not address, alongside a modern web front end — both in one suite.

### Telco & logistics
Integration-heavy landscapes where the contract between systems matters more than the screen, and where the specification is a WSDL.

### Public sector & defence
Fully self-hosted, PostgreSQL only, no content delivery network, no external asset fetch, and local models — for organisations that cannot send page snapshots to a foreign service.

### Healthcare software
Regulated release processes needing an immutable version chain, deployment audit and a demonstrable link between a tested version and a promoted one.

### Enterprise resource planning
Long-lived applications where the front end changes underneath a suite that must keep working — exactly the case self-healing was built for.

### Behaviour-driven teams
Gherkin scenarios with standard report output that drops straight into existing continuous-integration reporting.

### Agent-first teams
Organisations driving their tooling from coding agents. The platform is a first-class agent target with permissions, tiers and a call log.

WHO USES IT

Platform operator · Organisation owner · QA lead · Test engineer · Stakeholder / viewer · On-call engineer · AI agent

Six roles, narrowed further by project-scoped grants. A QA lead can promote through every stage *except* the last one in the chain — production is defined structurally, as the final environment, never by its name.

WHAT IT CONSOLIDATES

| TYPICALLY SEPARATE | IN TESTGUARD |
|---|---|
| Browser test framework | One suite |
| API test client | Same suite |
| SOAP test tool | Same suite |
| BDD runner | Same suite |
| Visual regression service | Built in |
| Test-management system | Versions & runs |
| Reporting dashboard | Shared history |


© 2026 Automatize · developed together with A4IT & A4ME

TECHNICAL SPECIFICATION

# Everything in one process. Nothing at the edge.

| | |
|---|---|
| **Runtime** | Java 25 on Spring Boot 4; native-image build configured |
| **Database** | PostgreSQL, 43 tables, 18 versioned migrations |
| **Browsers** | Playwright for Java in-process — no scanner CLI, no sidecar, no subprocess runner |
| **Front end** | Server-rendered with vanilla ES6 — charts, editor and controls all hand-written, zero content-delivery-network references |
| **Storage** | Artefacts on disk under a tenant-prefixed tree; the database holds relative paths, so the root can be remounted without migration |
| **Artefacts** | Run metadata, steps, screenshots, traces, request and response bodies, SOAP envelopes and faults, analysis |
| **API** | 343 REST endpoints, a Model Context Protocol server, SCIM 2.0, and a live browser channel |
| **Topology** | All-in-one, or split into web, API and scheduler tiers |
| **Quality** | 235 test classes; more test code than production code |

SECURITY & GOVERNANCE

- Six roles, narrowed by project-scoped grants; tenant, organisation and project isolation enforced at query and directory level
- **Single sign-on** with per-organisation client registration, plus LDAP and Active Directory
- **SCIM 2.0 provisioning** — user lifecycle and group-to-role mapping from your identity provider
- **TOTP multi-factor** with selectable algorithms for stricter environments
- Password policy with a length and complexity rule, a common-password blocklist and a breach check
- Rate limiting per address and per account on authentication
- **AES-256-GCM at rest** for identity secrets, directory credentials, mail secrets, model keys and saved browser sessions
- **A shipped default key is rejected by identity** — a production deployment that forgets to set one refuses to boot rather than encrypting with a public value
- Sessions stored as digests only, capped per user, revocable from the administration console
- Strict transport security, frame denial, per-request content-security nonces with no inline scripting, cross-site request protection on every surface
- Activity audit across creation, editing, runs, deployments and reports

COMMERCIAL

## Licensed per installation, not per test run.

TestGuard is sold per deployment, with no execution meter and no per-parallel-session charge. Testing volume grows with the product instead of waiting for a renegotiation.

| REVENUE | PER YEAR |
|---|---|
| **Up to €500k** | €900 |
| **€1M** | €1,500 |
| **€5M** | €2,500 |
| **€25M** | €5,500 |

| REVENUE | PER YEAR |
|---|---|
| **€100M** | €13,000 |
| **€500M** | €37,000 |
| **€1B** | €52,000 |

Annual licence on your own turnover, ex VAT. The schedule is marginal like income tax — each slice of turnover is charged at its own rate, so there are no cliffs — with a €900 minimum and any yearly increase capped at 25%. Every tranche carries unlimited users, projects and organisations inside one tenant, self-hosting and support that rises with the tranche. Hosting and hardware are yours, AI runs on your own provider keys and is billed by them, and consultancy, setup, migration and training are quoted separately. Illustrative: a cloud test grid at €150 per parallel session a month bills ten lanes €18,000 a year; at €5M turnover this is €2,500.

NEXT STEP

### Point it at your staging URL.

We will let it read the application, propose a suite, build it, and run it across three engines — in an afternoon, not a quarter.


| | |
|---|---|
| **Sales** | tim@automatize.eu |
| **Partners** | luc@automatize.eu |
| **Web** | automatize.eu |



© 2026 Automatize · developed together with A4IT & A4ME