

Checkly SREBot

Daniel Paulus - VP Engineering



Empower developers to own and ensure application performance and reliability
- from pull request to post-mortem

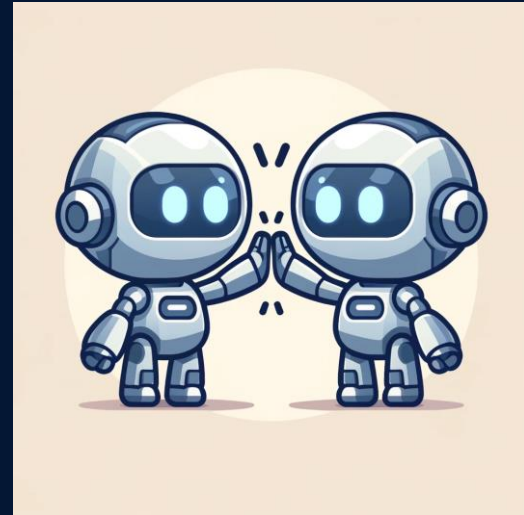
Some thoughts on AI



"AI" is not a new thing. The basics have been around as early as in the 1940s.

Since then it has been developed and improved every decade especially powered by modern hardware.

Every iteration massively improved what these algorithms can do, and this will keep happening.
AI is not going away, but it will become a part of our lives more and more.



Quick word on what Checkly does

Checkly is a Playwright & API based monitoring SaaS.

Users write Playwright scripts for testing, and can use them as production monitors to run on a schedule and receive alerts if something fails around the globe.

They can also define API monitors, that will monitor all of their REST APIs, Websocket endpoints and much more.



The screenshot displays the Checkly dashboard interface. At the top, there are three status indicators: **PASSING 38** (green), **DEGRADED 0** (yellow), and **FAILING 8** (red). Below this is a 'Firewatch' section with a warning icon and the text 'You have 4 checks that started failing in the last 7 days'. A search bar and filters for 'Status', 'Check type', and 'Tags' are present. The main area shows a table of checks for the '00_a Website Group'. The table has columns for NAME, TYPE, LAST RESULTS, 24H, 7D, AVG, P95, and ΔT. Four checks are listed: 'Clickhouse Free Diskpace', 'Clickhouse Version Check', 'Home page', and 'Login Check'. Each check has a status icon, a bar chart, and performance metrics. Below the table, the 'Home page' check is selected, showing its configuration: 'Activated', 'Muted', 'Updated Apr 04', and 'Runs every 10 mins from'. A code editor shows a Playwright script for testing the homepage, including steps for setting viewport size, navigating to the page, and asserting status and title. The bottom of the screen shows 'PRETTIER', 'KEYMAP: DEFAULT', 'Runtime 2023.02', 'Ireland', and a 'Run Script' button.

NAME	TYPE	LAST RESULTS	24H	7D	AVG	P95	ΔT
Clickhouse Free Diskpace	API	100%	100%	2.49s	2.65s	12s	
Clickhouse Version Check	API	100%	100%	2.6s	3.29s	10 min	
Home page	Web	100%	100%	4.03s	4.83s	10 min	
Login Check	API	100%	100%	4.94s	6.31s	10 min	

```
1 import { test, expect } from '@playwright/test'
2 import { defaults } from './defaults'
3
4 // You can override the default Playwright test timeout of 30s
5 // test.setTimeout(60_000)
6
7 test('Checkly Homepage', async ({ page }) => {
8   await page.setViewportSize(defaults.playwright.viewportSize)
9   const response = await page.goto(defaults.pageUrl)
10
11   expect(response?.status()).toBeLessThan(400)
12   await expect(page).toHaveTitle(/Danube WebSigg/)
13   await page.screenshot({ path: 'homepage.jpg' })
14 })
15
```

How to approach an AI project in a start up?



TL;DR

Like every other product development project :-)

We have endless ideas how to use AI, but which one should we go for?

=> We talked to 10 customers, running surveys to understand what problems they have

⇒ We picked one that seemed feasible to solve but also valuable enough for people to pay

The screenshot displays the Playwright test runner interface. At the top, there are three status bars: 'PASSING 38' (green), 'DEGRADED 0' (yellow), and 'FAILING 8' (red). Below this is a 'Firewatch' section with a warning icon and the text 'You have 4 checks that started failing in the last 7 days'. A search bar and filters for 'Status', 'Check type', and 'Tags' are present. The main area shows a table of test results for the '00_a Website Group'.

NAME	TYPE	LAST RESULTS	24H	7D	AVG	P95	ΔT
Clickhouse Free Diskpace	00		100%	100%	2.49s	2.65s	12s
Clickhouse Version Check	00		100%	100%	2.6s	3.29s	10 min
Home page			100%	100%	4.03s	4.83s	10 min
Login Check			100%	100%	4.94s	6.31s	10 min

Below the table, the 'Home page' test is selected, showing its configuration: 'Activated', 'Muted', 'Updated Apr 04 via', 'Settings', and 'Save check'. A code editor shows the test script:

```
1 import { test, expect } from '@playwright/test'
2 import { defaults } from '../defaults'
3
4 // You can override the default Playwright test timeout of 30s
5 // test.setTimeout(60_000)
6
7 test('Checkly Homepage', async ({ page }) => {
8   await page.setViewportSize(defaults.playwright.viewportSize)
9   const response = await page.goto(defaults.pageUrl)
10
11   expect(response?.status()).toBeLessThan(400)
12   await expect(page).toHaveTitle(/Danube WebSigg/)
13   await page.screenshot({ path: 'homepage.jpg' })
14 })
15
```

SREBot was born



It's your virtual Site Reliability Engineer to bring down MTTR and improve dev productivity while cutting engineering on call costs and stress by half.

We talked to our biggest customers and for them this is a value prop they are willing to pay for.

We were closely working with Sales to secure design partners.

<https://github.com/checkly/srebot>

SREBot APP 10:00 AM

Deployment
production - Diff
Repo
checkly-cli

When
a day ago
★ Authors
ferrandiaz, Antoine-C

Summary

feat: allow parsing multiple input types

- Introduced a new method in the `Parser` class to handle multiple input types, allowing users to get all files and dependencies by providing a list of directories, files, or globs.
- Added tests to ensure the new parsing functionality works as expected.

feat: support status pages and services

- Added support for `StatusPage` and `StatusPageService` constructs, allowing the creation and management of status pages and their associated services.
- Updated the `Project` class to include `status-page` and `status-page-service` in its data structure.
- Added end-to-end test resources for status pages and services.

refactor: project parser utility functions


- Moved the `findFilesWithPattern` utility function to a shared utility file for reuse across different modules.

fix: performance improvements

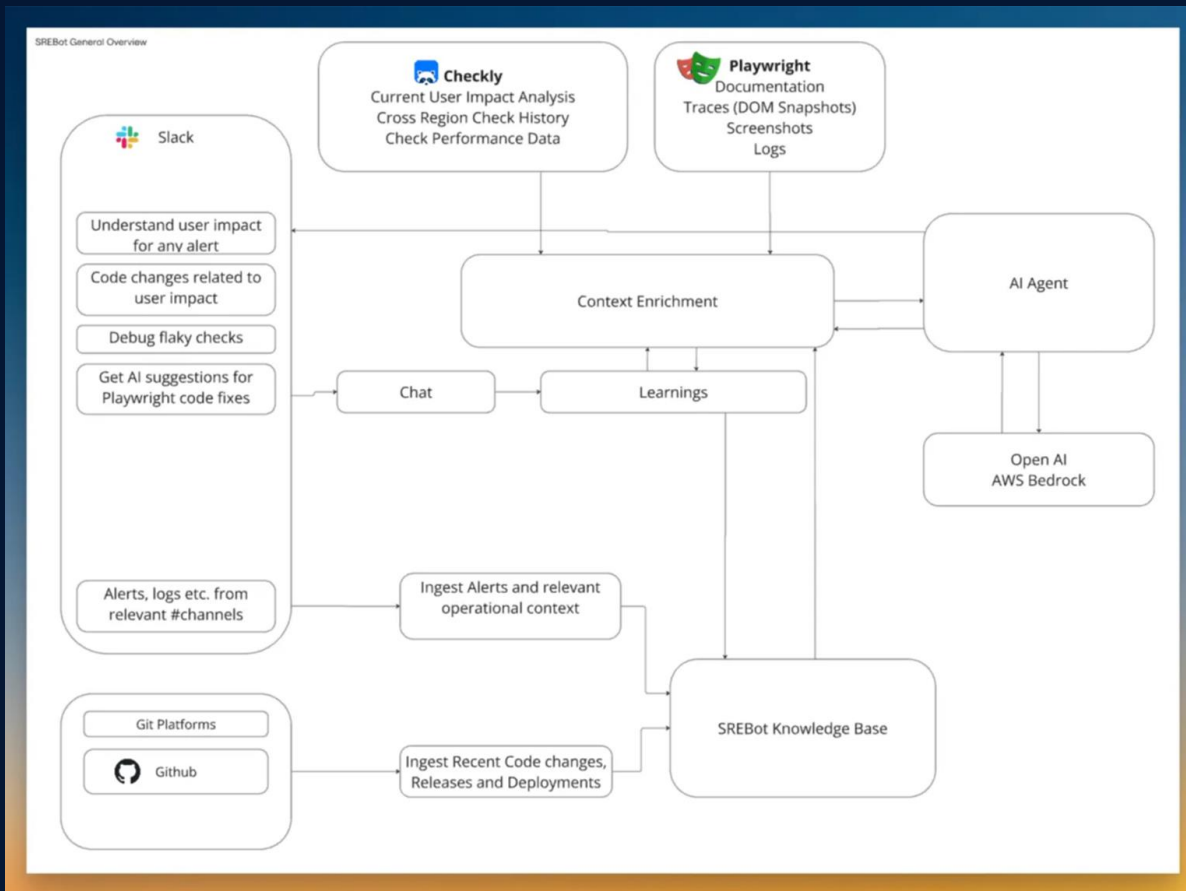
- Improved performance in file validation and dependency resolution processes.

chore: code cleanup and formatting

- Removed unused code and updated comments for clarity.
- Ensured consistent formatting across modified files.

 1 reply 9 days ago

SREBot plan





Tech considerations

- LLMs are really good at code summaries& text generation
- They are quite good at planning small steps
- They cannot magically figure out things, that a human wouldn't be able to figure out either, you have to give context
- You still need to combine algorithmic and structured data with LLMs and Agents
- TypeScript works to build a system like this, but when you want to work with data, we often go back to Python



Building prompts to give good results is the hardest part

- You tune a prompt using intuition, fix it for half the cases but break it for 10% you don't even know about ⇒ This is a data problem
- We tried langfuse for prompt observability
- LLM as a judge approaches
- Writing our own test harness

More Learnings& Conclusion



- It turned out that building a reliable SREBot like in the original idea was very hard
- Giving access to different observability systems was more than large customers were willing to do, the problem is real, but they did not trust us enough to give access
- We decided to leave source code changes out of the picture for now and focus on User Impact analysis. This is a real problem everyone has and we can solve it, because we own the data for it

⇒ Answer one question using AI, fast and with accuracy:

There is an incident or system event, what is the user impact?

Next Step: End User Impact Analysis



The screenshot shows a web browser window with several tabs. The active tab is 'Check result' showing a 'Coolblue wishlist' page. A modal window titled 'Checkly AI analysis' is overlaid on the page. The modal contains the following information:

- Who is impacted?**
Users are unable to add products to their wish list due to a timeout error when attempting to click the 'Geen' button.
[🗨️ Declare incident](#)
- Steps executed**
 1. Navigate to the Coolblue homepage.
 2. Accept cookies.
 3. Search for products using the search box.
 4. Select the first product from the search results.
 5. Attempt to add the product to the wish list.
- How can I resolve this?**
- App:** webshop
- Features:** wish-list, product-search
- Environment:** b2c
- Tags:** enduser:b2c, feature:wish-list, app:webshop, feature:product-search

Below the modal, the browser's console is visible, showing the following entries:

- Duration: 3789ms Prompt tokens: 1818 Completion tokens: 90 Total Token Usage: 1908 [show payload](#)
- > <https://www.coolblue.nl/zoeken?query=> 1 network error(s) status code 200
- > <https://www.coolblue.nl/product/945529/harman-kardon-enchant-900.html> 1 console error(s) 5 network error(s) status code 200

Any questions?



Thanks for listening :-)