Practical frontend testing

Atte Keinänen
atte@metabase.com

Slides & extra material:
http://tinyurl.com/frontendtesting

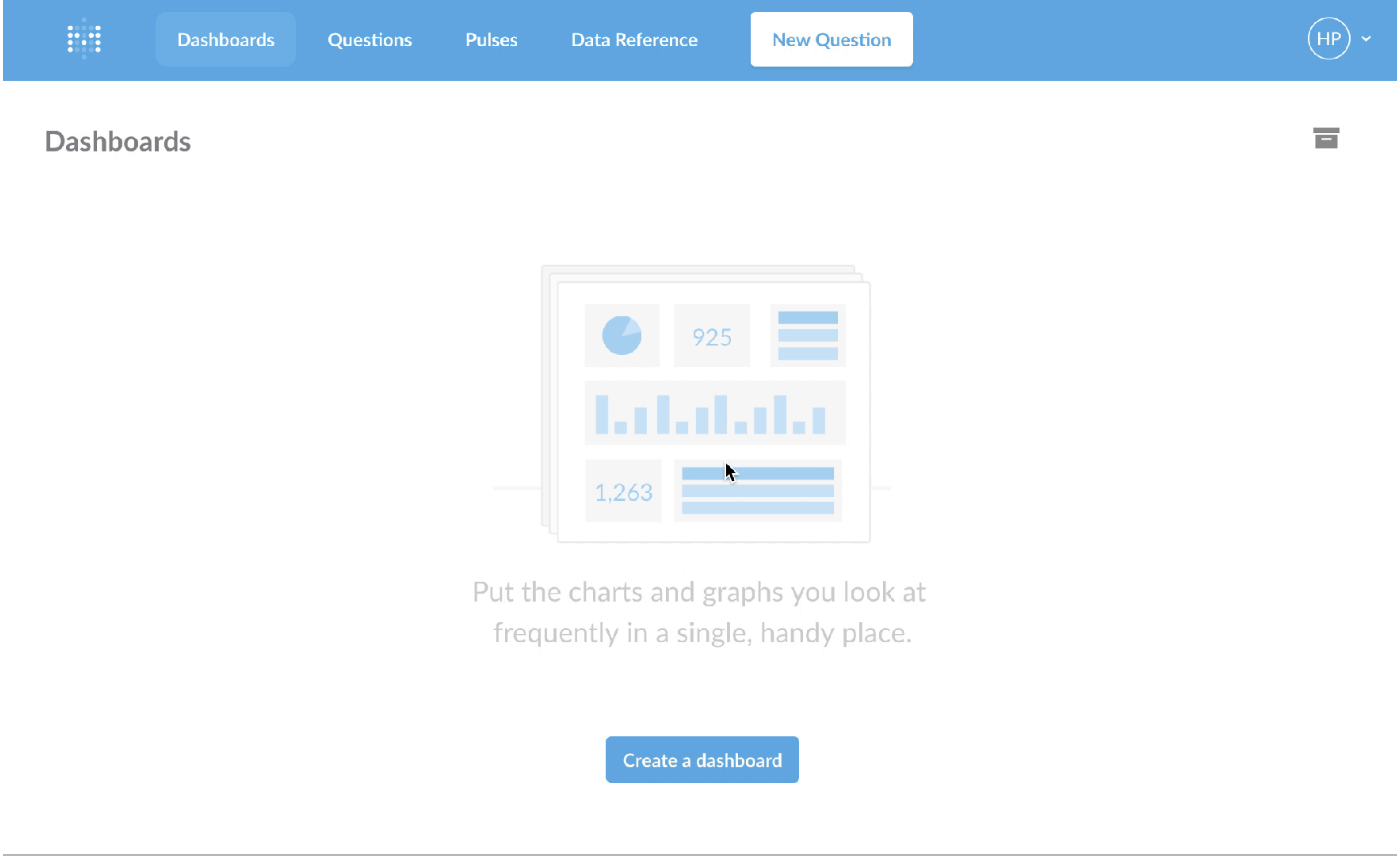
Briefly about myself

- I spent a year in San Francisco through Startuplifers, an internship program of Aalto University
- I'm a part of the core team of Metabase, a SF-based startup developing an open source tool for data exploration and visualization
- I sing in a mixed choir and I absolutely love travelling (especially in Asia)

What do I mean with ”frontend testing”?

Making sure that the user of your app

- sees what you expect him to see
- is able to achieve his goals
- doesn’t feel that the app is too slow



Dashboards

Put the charts and graphs you look at frequently in a single, handy place.

```
describe("dashboards list", () => {
    it("should let you create a dashboard when there are no existing dashboards", async () => {
        // Initialize Redux store and navigate to dashboard list url
        const store = await createTestStore();
        store.pushPath("/dashboards")
        const app = mount(store.getAppContainer());

        // Wait for dashboards to load
        await store.waitForActions([FETCH_DASHBOARDS])

        // Trigger the creation modal by clicking the "Create a dashboard" button
        const newDashboardButton = app.find(".Button.Button--primary")
        click(newDashboardButton)
        const modal = app.find(CreateDashboardModal)
        expect(modal.length).toBe(1)

        // Set the input values
        setInputValue(modal.find('input[name="name"]'), "HelsinkiJS Demo Dashboard")
        setInputValue(modal.find('input[name="description"]'), "Frontend Testing Fun")
        clickButton(modal.find(".Button--primary"))

        // Should navigate to the newly created dashboard
        await store.waitForActions([BROWSER_HISTORY_PUSH, FETCH_DASHBOARD])
        expect(app.find(Dashboard).length).toBe(1)
    })
})
```

Using some techie jargon:

- Doing functional testing for verifying that user-facing parts of your app behave as expected
- Doing performance testing for detecting performance bottlenecks

Why should you write frontend tests?

1. It improves your code quality

- It encourages you to write easy-to-understand modular code

```
export const AlertAboveGoalToggle = (props) =>
    <AlertSettingToggle {...props} setting="alert_above_goal" />
```

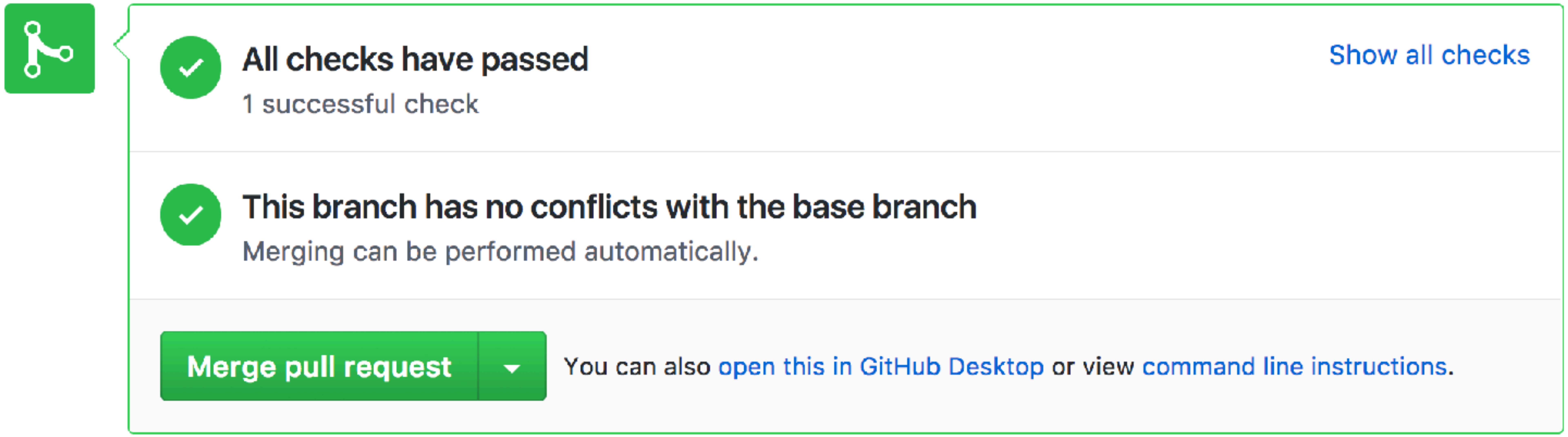
```
export const AlertFirstOnlyToggle = (props) =>
    <AlertSettingToggle {...props} setting="alert_first_only" />
```

```
export const AlertSettingToggle = ({ alert, onAlertChange, title, trueText, falseText, setting }) =>
    <div className="mb4 pb2">
        <h3 className="text-dark mb1">{title}</h3>
        <Radio
            value={alert[setting]}
            onChange={(value) => onAlertChange({ ...alert, [setting]: value })}
            options={[{ name: trueText, value: true }, { name: falseText, value: false }]}
        />
    </div>
```

2. It makes you more confident

- Makes you less anxious whether your app is working as a whole or not

- Makes you push code to production more frequently



All checks have passed

1 successful check

Show all checks

This branch has no conflicts with the base branch

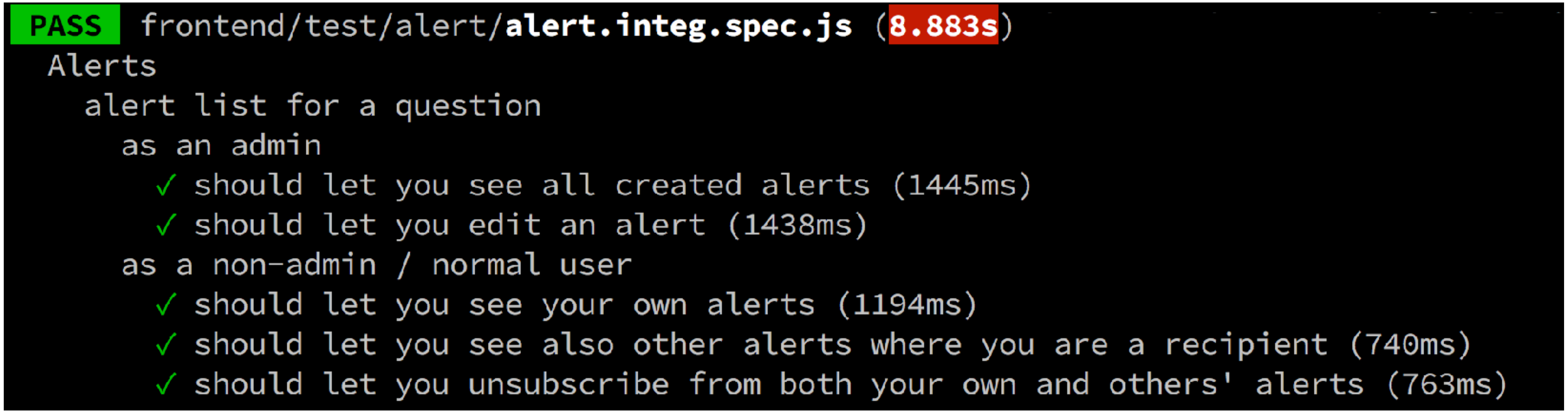
Merging can be performed automatically.

Merge pull request

You can also open this in GitHub Desktop or view command line instructions.

3. You spend less time trying to decipher source code

- Tests serve as an up-to-date documentation which is often faster to digest than plain source code



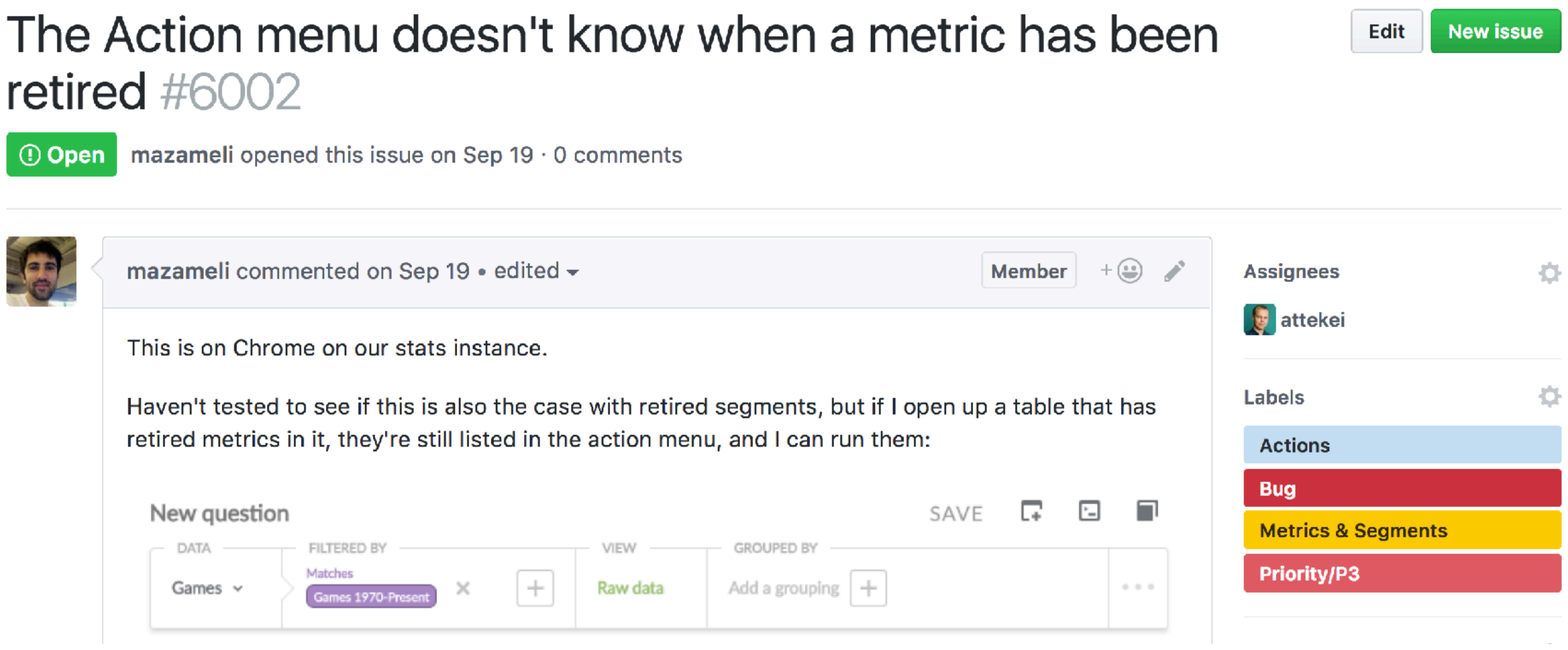
```
PASS  frontend/test/alert/alert.integ.spec.js (8.883s)
  Alerts
    alert list for a question
      as an admin
        ✓ should let you see all created alerts (1445ms)
        ✓ should let you edit an alert (1438ms)
      as a non-admin / normal user
        ✓ should let you see your own alerts (1194ms)
        ✓ should let you see also other alerts where you are a recipient (740ms)
        ✓ should let you unsubscribe from both your own and others' alerts (763ms)
```

How to start writing frontend tests?

- Incrementally; trying to get from zero to 100% test coverage tends to be unrealistic
- Focus on critical interaction paths; don’t aim to cover everything
- A word of encouragement: the first test cases might take a long time to write, but the more tests you have, the easier it gets

One easy-to-adapt strategy that has worked well for us has been writing a test case for each new regression bug we encounter

Original bug report:



The Action menu doesn't know when a metric has been retired #6002

Edit | New issue

Open mazameli opened this issue on Sep 19 · 0 comments

mazameli commented on Sep 19 • edited

Member

This is on Chrome on our stats instance.

Haven't tested to see if this is also the case with retired segments, but if I open up a table that has retired metrics in it, they're still listed in the action menu, and I can run them:

Assignees

attekei

Labels

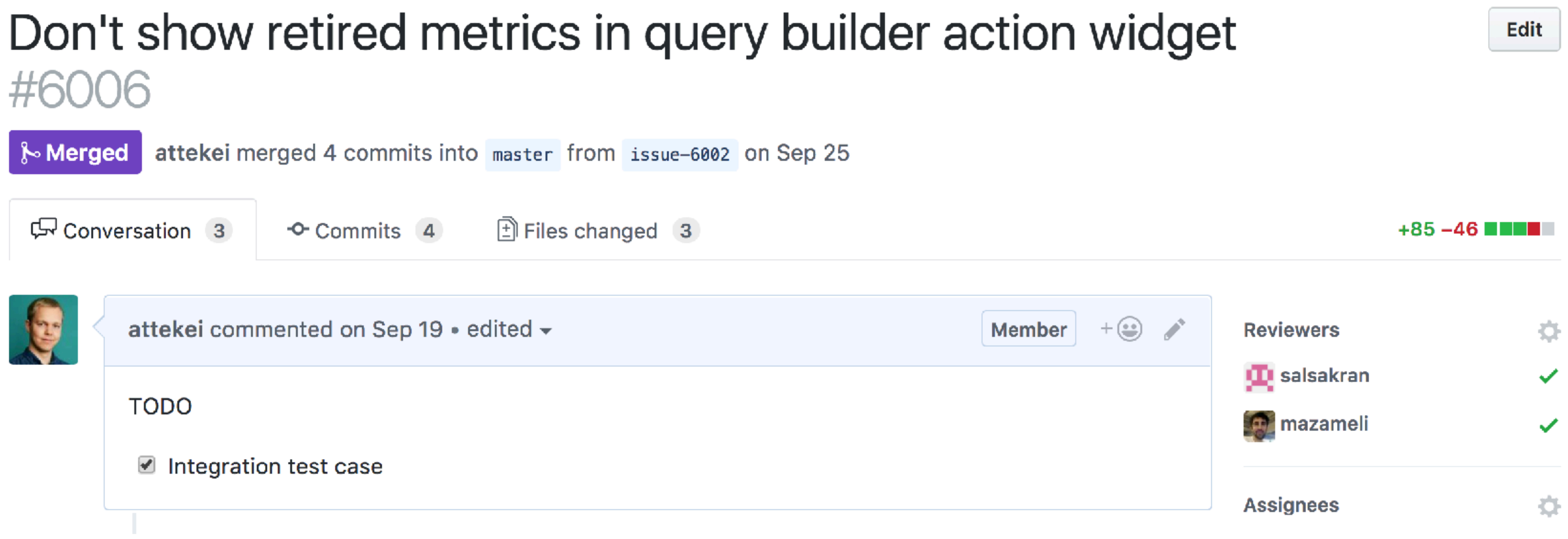
Actions

Bug

Metrics & Segments

Priority/P3

A fix to that bug contains a frontend test case:



Don't show retired metrics in query builder action widget #6006

Edit

Merged attekei merged 4 commits into `master` from `issue-6002` on Sep 25

Conversation 3 Commits 4 Files changed 3 +85 −46

attekei commented on Sep 19 • edited

Member

TODO

- [x] Integration test case

Reviewers

salsakran ✓

mazameli ✓

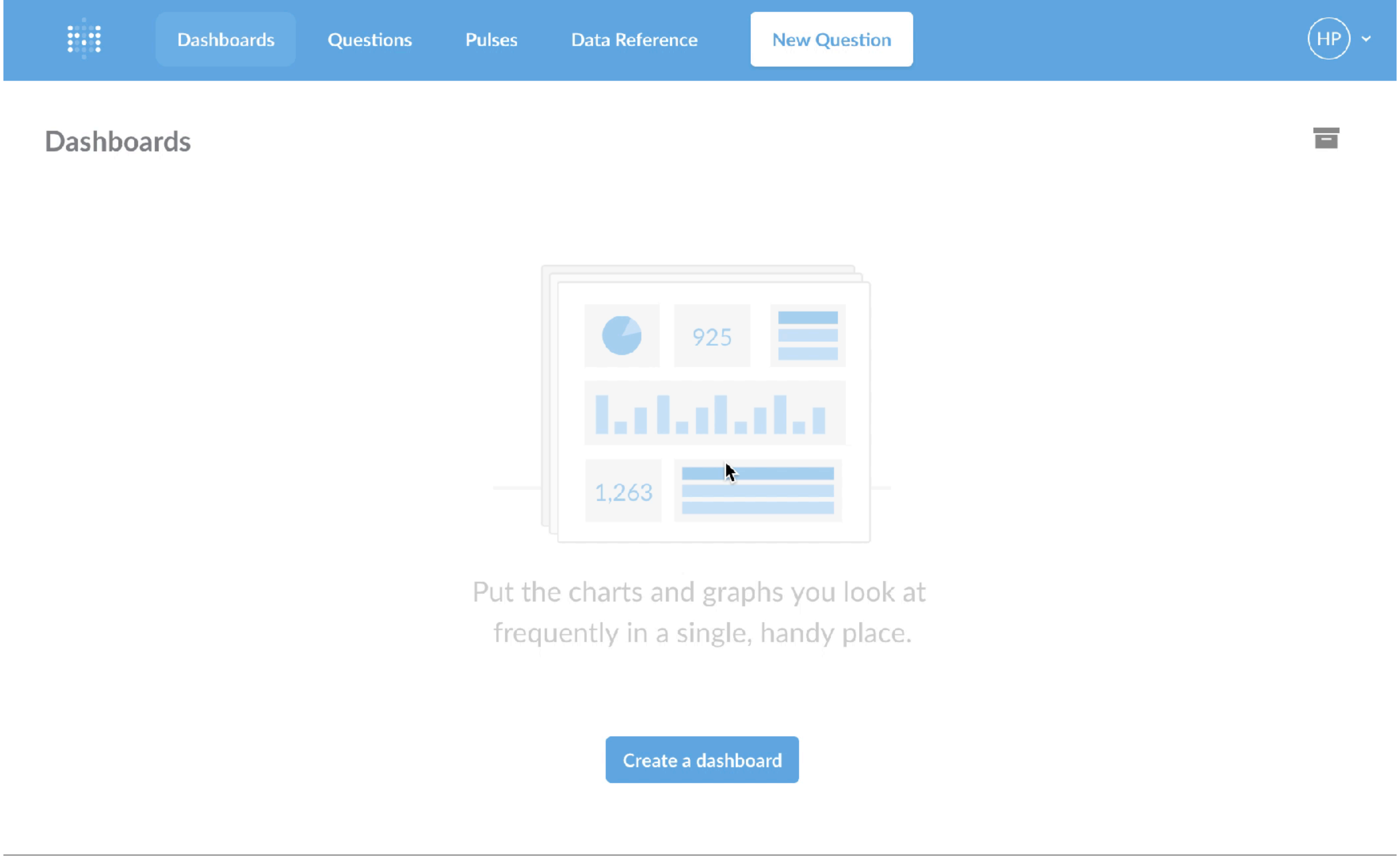
Assignees

Which parts of my codebase should my tests cover?

(this is more opinionated than the previous sections!)

- Consider rendering the whole app in your tests.
  It’s nicer to know that ”this whole dashboard list page shows up correctly” than being only limited to a single component inside that page.

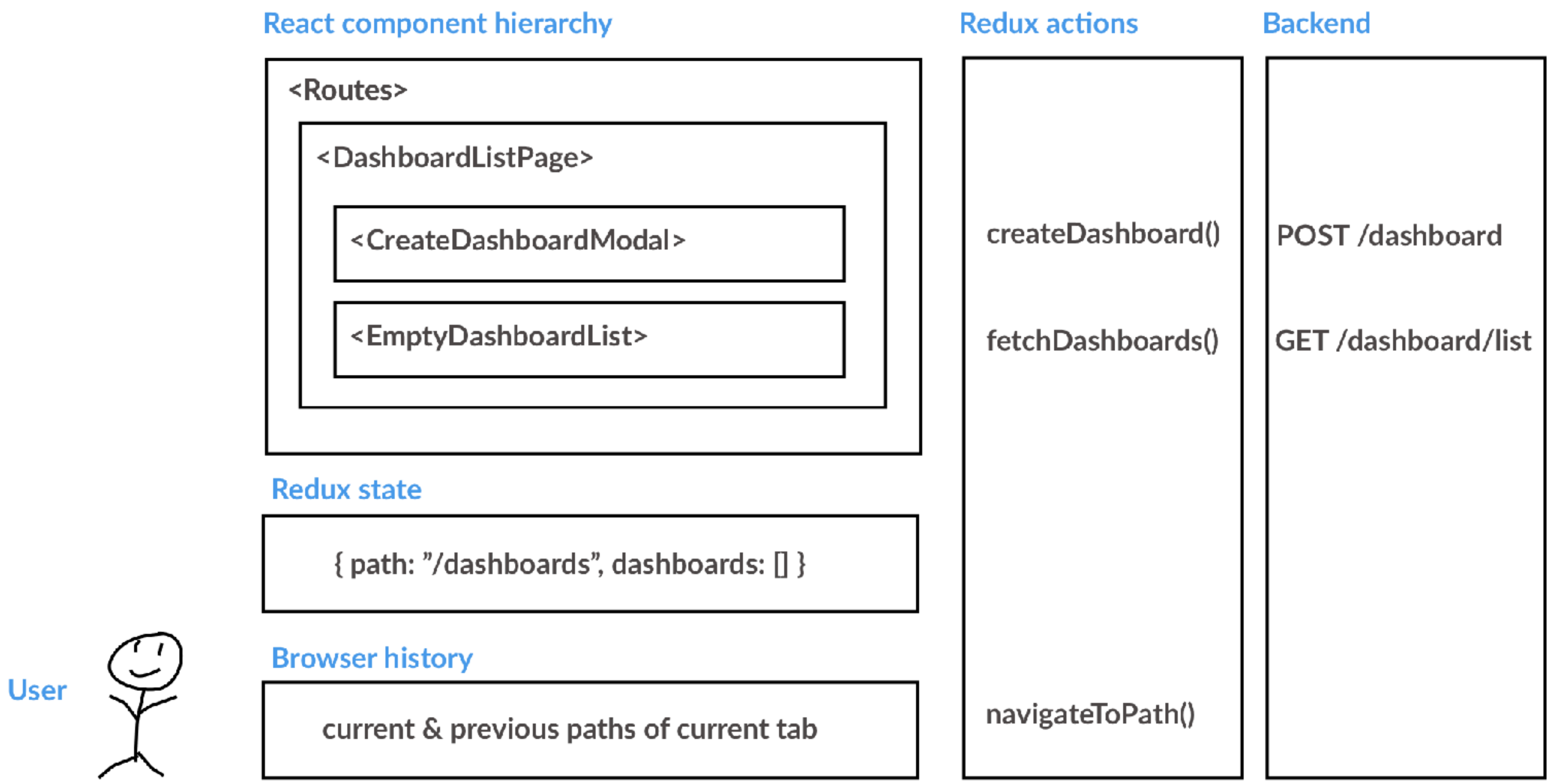
- Consider using a real backend.
  Use fake API responses only when absolutely necessary. If you use a real backend, your tests will catch issues caused by changes in your API endpoints.



Dashboards

Put the charts and graphs you look at frequently in a single, handy place.

You can cover these all with a single test case!



Obstacles you might encounter

- Tests are slow. We’ve had to spend a fair amount of time optimizing the launch time of our test runner and tests itself. For instance, we currently reuse a same user login in every test.
- Tests are unstable. We had to rewrite our frontend test infrastructure after we realized that our old tests (based on Selenium) were failing too often.

Thanks!

Slides & a deep dive to technical implementation: http://tinyurl.com/frontendtesting