React
Best Practices

5 – Code Quality & Testing

Timo Mämecke
TH Köln // MI Master // Weaving the Web
25. Juni 2019

Inhalt

Enforcing Code Style.

Integration Testing.

Enforcing Code Style

Warum um Code Style kümmern?

Jeder hat seinen eigenen Stil beim Schreiben von Code.

Code Style kann zu unnötigen Diskussionen führen.

Mancher Code Style ist anfälliger für Fehler.

Dateien werden mit verschiedenen Stilen unübersichtlich.

Code Changes werden unübersichtlich.

...

Wie um Code Style kümmern?

"Linting"

Code Style Regeln für das Projekt festlegen.

Regeln mit Tools prüfen.

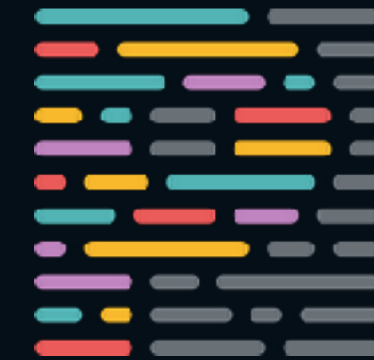
Unsauberer Code kann automatisch gefixt werden.

Empfehlenswerte Tools

eslint

Linter
Code Quality Tool

komplett konfigurierbar



prettier

Code Formatter

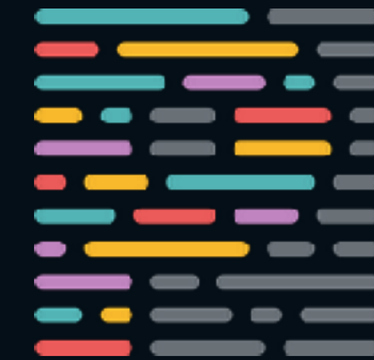
opinionated, nur
teilweise konfigurierbar

Empfehlenswerte Tools

eslint

Linter
Code Quality Tool

komplett konfigurierbar



prettier

Code Formatter

opinionated, nur
teilweise konfigurierbar

Beide miteinander verbinden: 👌💖

Quick Guide

1

```
$ npm install -D prettier eslint babel-eslint eslint-config-prettier eslint-plugin-prettier
```

- babel-eslint für Babel Syntax
- eslint-config-prettier, eslint-plugin-prettier verbinden eslint und prettier

2 .prettierrc erstellen

```
{ "semi": false, "singleQuote": true }
```

3 .eslintrc erstellen

```
{
  "parser": "babel-eslint",
  "env": { "browser": true },
  "extends": ["eslint:recommended", "plugin:prettier/recommended"]
}
```

4

```
$ npx eslint .
```

Außerdem nützlich:

- lint script zur package.json hinzufügen
  "npm run lint" statt "npx eslint ..."

```
{
  "scripts": {
    "lint": "eslint ."
  }
}
```

- Linter zum Editor hinzufügen!

- Linting mit Continuous Integration verbinden.

Weitere Linting Regeln hinzufügen

Von Anfang an empfehlenswert:

- eslint-plugin-react
- eslint-plugin-react-hooks

Mein Vorgehen: Linting Regeln dann hinzufügen, wenn sie gebraucht werden.

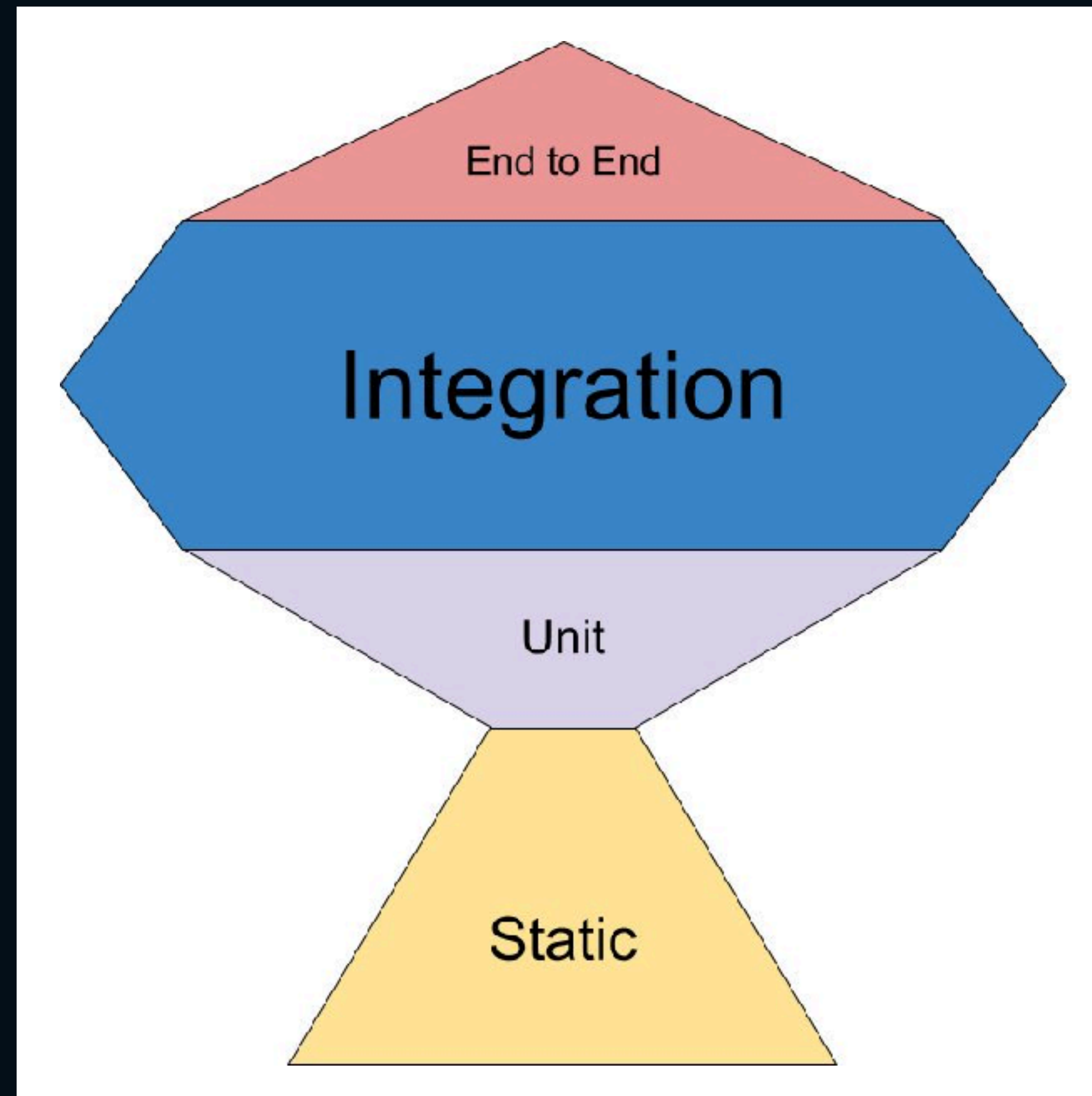
Möglich, mache ich aber nicht: weitere vordefinierte Regeln installieren

- z.B. eslint-config-airbnb

Integration Testing

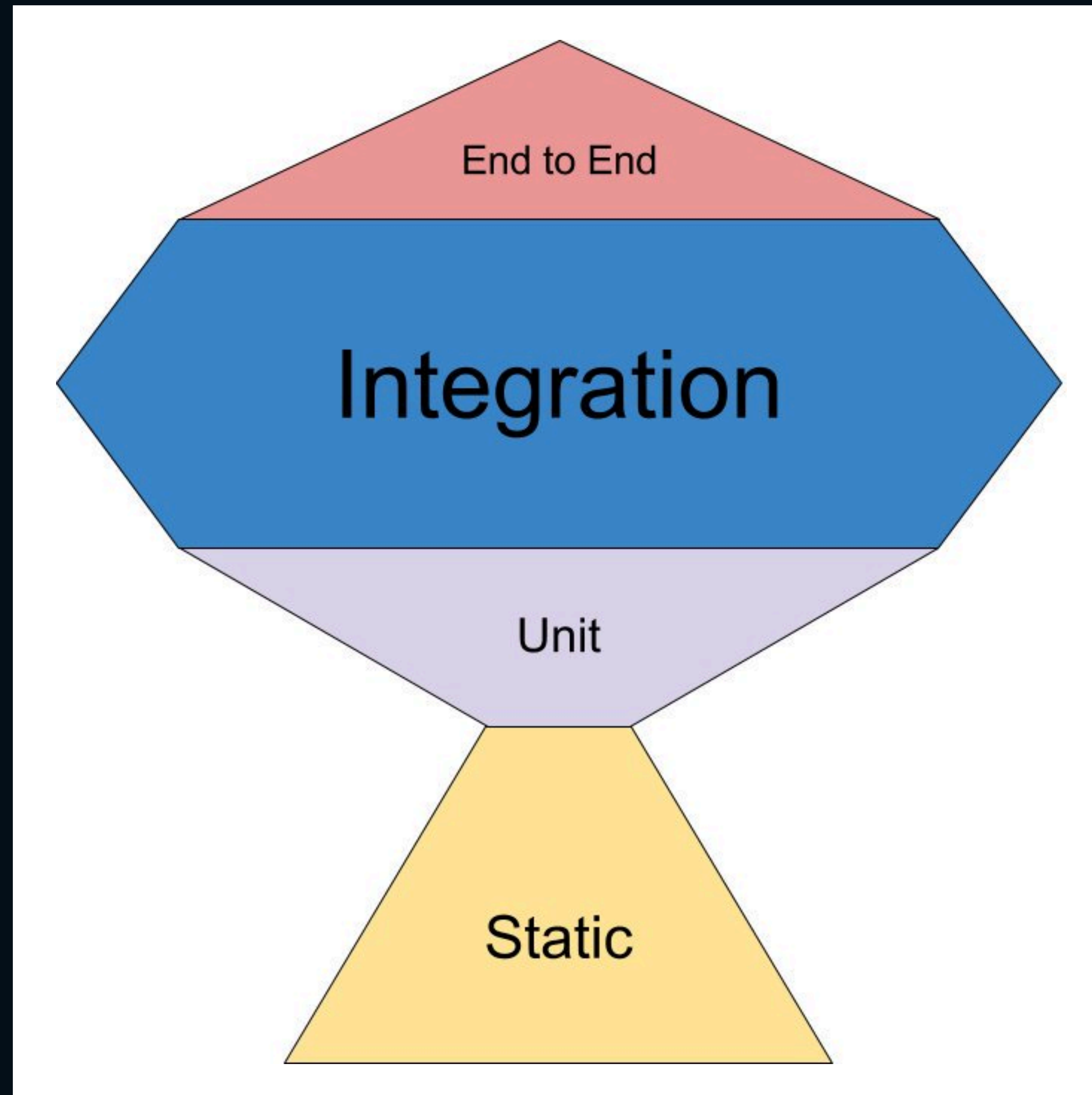
The Testing Trophy

Wie viele Tests man schreiben sollte:



* Siehe: https://twitter.com/kentcdodds/status/960723172591992832

The Testing Trophy



Testing

End to End Tests: langsam, aufwändig, sehr verlässlich

Integration Tests: perfekte Mischung aus Aufwand und Verlässlichkeit

Unit Tests: schnell, einfach, weniger verlässlich

Linting

schnell aufzusetzen
vermeidet Syntaxfehler/-probleme

* Siehe: https://twitter.com/kentcdodds/status/960723172591992832

Unit test vs. Integration test

Sinnvolle Integration Tests[1]

Tests sollten nicht failen: durch internes Refactoring.
Interne Logik testen sorgt für viele failing Tests bei Änderungen.

Tests sollten failen: wenn sich für den User etwas geändert hat.

Testen, was der Nutzer machen kann.

Nur assertions auf Dinge, die sich für den Nutzer geändert haben.

...am besten direkt auf dem DOM!

[1] ... für React UIs

react-testing-library

React Components direkt im DOM[1] rendern.

Echte User-Aktionen simulieren:
- Klick auf einen Button
- Eingabe von Text

Assertion, was sich für den Nutzer geändert hat:
- Loading-Spinner erscheint
- Text ändert sich
- Elemente hinzugekommen oder verschwunden

* Siehe: https://github.com/testing-library/react-testing-library
[1] respektive JSDOM Implementation

Beispiel:

Button mit einem Counter. Der Counter startet bei 0 und erhöht sich bei Klick um 1.

```
import React from 'react'
import { render, fireEvent } from '@testing-library/react'
import Counter from './Counter'

test('count starts with 0', () => {
  const { getByTestId } = render(<Counter />)
  expect(getByTestId('button')).toHaveTextContent(/0 times/)
})

test('count increases by 1 after click', () => {
  const { getByTestId } = render(<Counter />)
  fireEvent.click(getByTestId('button'))
  expect(getByTestId('button')).toHaveTextContent(/1 times/)
})
```

Was Nutzer initial sieht.

Nutzer macht eine Aktion und erwartet Änderung.

tl;dr

Code Style automatisch mit prettier[1] & eslint[2] prüfen.

Prettier & eslint miteinander verbinden.[3]

Integration Tests bieten Sicherheit bei Änderungen.

Tests so schreiben, wie sie den Nutzer betreffen.
Dafür eignet sich react-testing-library[4]

[1] https://prettier.io
[2] https://eslint.org
[3] https://prettier.io/docs/en/integrating-with-linters.html#eslint
[4] https://github.com/testing-library/react-testing-library