

Testbericht
Nachrichtenkommunikation für das THW

na17b

Inhaltsverzeichnis

| | | |
|----------|-------------------------------|----------|
| 1 | Allgemeines | 2 |
| 2 | Tests | 2 |
| | Komponententests | 2 |
| | GUI-Tests | 2 |
| 3 | Continuous Integration | 2 |
| | Glossar | 4 |

1 Allgemeines

Die Tests für dieses Softwareprojekt sind eng an das Qualitätssicherungskonzept geknüpft und sollen das Funktionieren des Codes gewährleisten. Das Vorprojekt führt ausschließlich Komponententests durch, da die Kommunikation zwischen Komponenten nur indirekt über den Store erfolgt. Der Zugriff auf den Store wurde gemockt. Als Testframework wird Jest zusammen mit vue-test-utils verwendet.

2 Tests

Komponententests

Komponententests befinden sich jeweils im gleichen Ordner wie die zu testende Komponente.

GUI-Tests

Die Testspezifikationen befinden sich vom frontend-Verzeichnis aus gesehen in `test/unit/specs`. Dort liegt für jede Komponente eine eigene Datei, welche die geforderten Eigenschaften und Funktionen einer Komponente beschreibt. Es genügt, im Ordner frontend den Befehl `npm run unit` auszuführen; daraufhin werden alle Test-Suites automatisch abgearbeitet. Eine beispielhafte Ausgabe ist in folgender Abbildung zu sehen. Die Tests beschränken sich hier zunächst auf das Prüfen

```
PASS test/unit/specs/THWForm.spec.js
  THWForm
    ✓ binds to hdzIncomingA (61ms)
    ✓ binds to creationTime (24ms)
    ✓ binds to creationDate (24ms)
    ✓ binds to message (26ms)
    ✓ pushes a new ticket to the store (30ms)

PASS test/unit/specs/THWDashboard.spec.js
  THWDashboard
    ✓ displays the author (11ms)
    ✓ displays the creation date (2ms)
    ✓ displays the creation time (2ms)
    ✓ displays the content of a ticket (1ms)

PASS test/unit/specs/THWMenu.spec.js
  THWMenu
    ✓ let's you choose the dashboard (6ms)
    ✓ let's you choose the ticket creation mask (3ms)
    ✓ let's you choose the role selection menu (2ms)

Test Suites: 3 passed, 3 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        1.346s, estimated 2s
Ran all test suites.
```

Abbildung 1: Ausgabe des Befehls `npm run unit`

von Anwesenheit bestimmter Variablen und html-Elementen. Die Ergebnisse sind in folgender Tabelle aufgelistet.

3 Continuous Integration

Zum zweiten Release wurde Continuous Integration eingeführt, um das Einhalten der Vorgaben aus dem Dokumentationskonzept und Coding Standards automatisiert zu testen. Dazu wird

| Komponente | Anzahl Tests | Bestanden |
|--------------|--------------|-----------|
| THWForm | 5 | ja |
| THWDashboard | 4 | ja |
| THWMenu | 3 | ja |

Tabelle 1: Testergebnisse der momentanen Frontend-Komponenten

GitLab CI GitLab CI verwendet. Die `gitlab-ci.yml` beinhaltet XML-Linting sowie JavaScript-Linting und -Testing. Um den Prozess zu beschleunigen wird auf `npm install` verzichtet, stattdessen werden nur für die Tests benötigte Pakete und deren Abhängigkeiten installiert. Anschließend erfolgen linting (`npm run lint`) und testing (`npm run unit`).

Glossar

GitLab CI GitLab CI ist die in GitLab eingebaute Continuous Integration, die sich mit Hilfe der `gitlab-ci.yml` Datei konfigurieren lässt. . 3

Jest Von Facebook entwickeltes Testframework für Javascript. Zeichnet sich durch seine Einfachheit in der Benutzung aus. . 2

vue-test-utils Sammlung an Funktionen um Vue-Komponenten in Unit-Tests verwenden zu können. . 2