ToiletPaper #111

jambit WHERE INNOVATION WORKS

Testing a saga made easy

Autor: Robert Gruner / Software Engineer / Office Leipzig

✕ Problem

Ihr kümmert euch gerade um eine Single Page Application mit React? Für das State Management kommt bei euch Redux zum Einsatz? Ihr habt es sogar schon geschafft, eure asynchronen Arbeitsschritte & Seiteneffekte (API Calls, Zugriff auf Browser Caches etc.) mittels redux-saga zu kapseln? Das könnte dann stellenweise ungefähr so aussehen:

```
export default function* userConfirmationsSaga() {
    yield all([takeEvery(FETCH_USER_CONFIRMATION_START, fetchUserConfirmations), /*here may be more*/]);
}
function* fetchUserConfirmations() {
    try {
        const accessToken = yield select(getAccessTokenFromStore);
        const requestUrl = yield select(getUserConfirmationUrlFromStore);
        const response = yield call(fetch, requestUrl, {});
        yield put(fetchUserConfirmationSuccess(response));
    } catch (e) {
        yield put(fetchUserConfirmationMissing());
    }
}
```

Jetzt fehlt euch eigentlich nur noch ein Schritt zur Meisterschaft: Wie testet man die Pfade/Fälle ab, um sicherzugehen, dass das Ding auch das tut, was es soll?

✓ Lösung

Onkel Jeremy Fairbank hat da was Feines für uns, und es nennt sich redux-saga-test-plan. Mit diesem Spielzeug kann man seine Saga integration testen. D.h. innerhalb eines Testlaufs ist es möglich, die Saga mittels gemockten Redux Actions und speziellen Providern so zu steuern, dass sie einen gewissen Verlauf nimmt (und hoffentlich eine erwartete Redux Action triggert). Wenn man die obige Saga zugrunde legt, könnte ein spezieller Test für den Erfolgsfall also so aussehen:

➔ Beispiel

```
describe('userConfirmationsSaga - when fetching user confirmation', () => {
    it('eventually dispatches success action', () => {
        const payload = { term_of_use_confirmation: new Date().toISOString() };
        return expectSaga(userConfirmationsSaga)
            .dispatch({ type: FETCH_USER_CONFIRMATION_START })
            .provide([
                [select(getAccessTokenFromStore), ''],
                [select(getUserConfirmationUrlFromStore), ''],
                [matchers.call.fn(fetch), payload]
            ])
            .put({ type: FETCH_USER_CONFIRMATION_SUCCESS, payload })
            .silentRun();
    });
});
```

- **.dispatch** steuert, auf welche der "beobachteten" Actions (FETCH_USER_CONFIRMATION_START) reagiert wird.
- Innerhalb von **.provide** wird alles gemockt, was den Verlauf der Saga beeinflusst - d.h. jegliche Selektionen aus dem Redux Application State oder Aufrufe von anderen Funktionen.
- **.put** legt fest, welche Redux Action erwartet wird. Der Test wird nur grün, wenn diese tatsächlich getriggert wird.

✚ Weiterführende Aspekte

- https://twitter.com/elpapapollo
- Es ist auch easy möglich, den Redux State zum Start eines Testablaufs festzulegen. Dazu benötigt man die .withState-Methode.