TU Wien
Fakultät für Informatik
Institut für Information Systems Engineering
E194

Report Assignment 3
Web Application Engineering & Content Management
183.223 - SS 2018

Group 13

Name	**Student ID**
GLAS Josef	08606876
MAITZ Alexander	01426069
MUSIC Ismar	01328053

Bericht

Bericht (max. 3 Seiten)
Schreiben Sie einen kurzen Erfahrungsbericht, in dem Sie Ihre CI/CD Pipeline vorstellen. Gehen Sie dabei auf folgende Punkte ein:
○ Welche Technologien / Frameworks haben Sie verwendet?
○ Auf welche Probleme sind Sie gestoßen, wie wurden diese gelöst?
○ Hat Ihnen die Dockerisierung Ihres Projekts bei der Umsetzung der CI/CD geholfen oder mehr Arbeit verursacht? Warum?

Es wurde Travis CI zu Erstellung der Pipeline genutzt (https://travis-ci.org/tu1426/wacm_ss18), Google Cloud und Heroku wurde genutzt um die eine Instanz zu deployen, bzw zumindest um dies zu versuchen. Für die Unit/Integration Tests wurde das Jasmine Framework verwendet, für die automatisierten Browsertests wurde Nightwatch genutzt. Dazu wurde zuvor der gesamte Sourcecode auf Github verschoben (Repo: https://github.com/tu1426/wacm_ss18/). Auch ein Refactoring der Folderstruktur wurde durchgeführt, um das Verwenden eines travis.yml Files zu erleichtern. Nach dem fixen einiger Probleme mit dem automatisierten Testen auf Travis CI konnten schließlich die Tests bei jedem Push getriggert werden und fertig implementiert werden. Getriggert wird bei einem Push auf beliebige Branches, der Deployment-Part wird nur bei einem Push auf den Master durchgeführt.
Der Deployment Teil der Pipeline führt anschließend an die erfolgreichen Tests die Task für das Erzeugen und Hochladen eines neuen Docker Images auf Docker Hub (Registry: https://hub.docker.com/r/tu1426/waecm-2018-group-13-bsp-3-web) durch.
Das Pushen der erstellten Images verlief überraschenderweise gut, auch das Encrypten der Docker Hub Credentials war einfach mit dem Travis CI CLI zu erledigen. Um das Server Image deployen zu können, musste zuerst eine MongoDB Instanz deployed werden, dies wurde in der Google Cloud erledigt. Das Deployen mittels Travis selbst wurde auf verschiedenste Art und Weise versucht, mittels Google Cloud und mittels Heroku, leider ohne Erfolg selbst nach mehrmaligem Versuch, sodass dieser Schritt abgebrochen wurde.

Probleme gab es einige, ein Beispiel wäre die Integration von Docker und docker-compose in Travis, was viele kleine Tücken mit sich bringt, wie zum Beispiel das Timen der Docker-Container Starts nach den Builds. Ein weiteres Beispiel stellte das Setup der Browser Tests mittels Nightwatch in Travis dar. Gelöst wurden alle diese Probleme mittels google und zahlreichen Tutorials. Weitere Probleme mit den Ent-to-End Tests sind im Kapitel “Automatic Browser Tests” angeführt.

Die Dockerisierung selbst hat beim Erstellen der Pipeline eigentlich wenig geholfen bzw. mindestens genauso viel Arbeit verursacht durch irgendwelche Besonderheiten als anderswo eingespart wurde.
Beim Deployment hat die Dockerisierung nicht wirklich geholfen, da wir es nicht schafften, diesen Docker Container richtig zu laufen zu bekommen.
Das Pushen des Images war dank der bereits erfolgten Dockerisierung einfach zu erledigen, man musste sich nur mehr bei Docker Hub einloggen und konnte bereits Pushen.

Automatic browser tests

Automatic browser tests have been implemented with nightwatch 0.9.21

Nightwatch.js is an automated testing framework for web applications and websites, written in Node.js and using the W3C WebDriver API (formerly Selenium WebDriver).
[see http://nightwatchjs.org]

Folders:

e2e	folder with test scripts
docs/reports	test reports of test cases and selenium-debug.log

Implemented test cases:

basic.spec.js	checks if frontend is reachable, endpoints /index and /register
valid.login.js	verifies valid login of user
check.i18n.js	verfies language support (english)

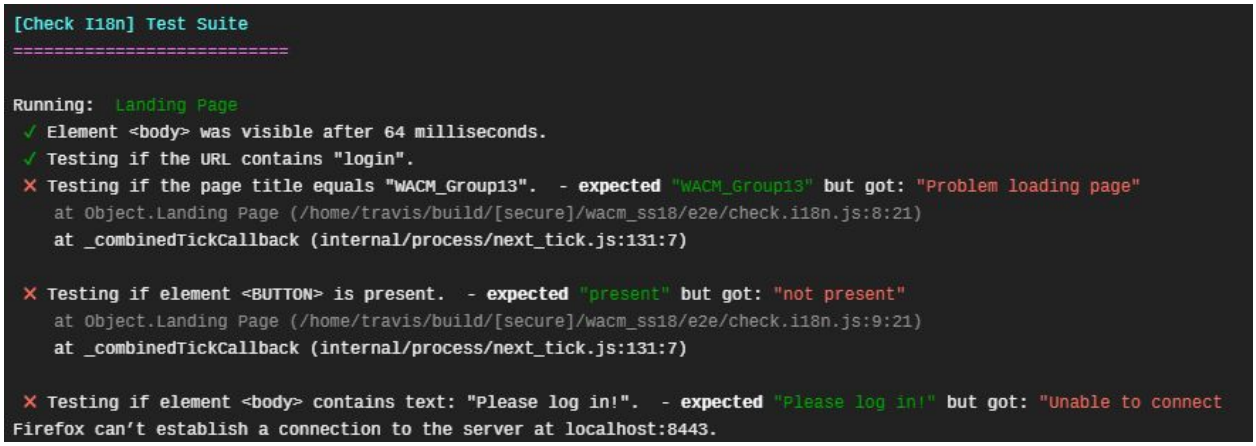
Calls:

./node_modules/.bin/nightwatch --env default --test ./e2e/basic.spec.js
./node_modules/.bin/nightwatch --env chrome-en --test ./e2e/valid.login.js
./node_modules/.bin/nightwatch --env chrome-de --test ./e2e/check.i18n.js

As already mentioned, the configurations of the web drivers which are required to interact with the selenium server are somehow tricky. Various browser options are required for smooth interaction (javascriptEnabled, acceptSslCerts, language settings, acceptInsecureCerts, ...).
The implementation of the test cases was straightforward and the test runs finally went smoothly when tested locally. Firefox as well as Chromium worked fine.

However the integration with Travis basically failed. We could only implement one simple e2e test (basic.specs.js) which checks only the availability of the site.

The more sophisticated test cases failed with general errors like "Problem loading page" or "Unable to connect" (see screenshot)



We have tried various driver options like "--headless" or "--no-sandbox" which all worked locally but had no effect when using Travis.
The problems remained also when we finally tried to run the e2e tests in a docker container.