Gesamtabgabe "Code und Build"

Technisches Management von Cloud Solutions PT

Thomas Stadler[1], Klaus Thüringer[2], and Stephan Pilwax[3]

[1]2110781014@fh-burgenland.at
[2]2110781007@fh-burgenland.at
[3]2110781003@fh-burgenland.at

June 13, 2022

Contents

1 Einleitung **2**
1.1 Aufgabenstellung . . . 2
1.2 Praktische Umsetzung . . . 2

2 Browser IDE Vergleich **3**
2.1 github.dev . . . 3
2.2 stackblitz.com . . . 3
2.3 gitpod.io . . . 3

3 Dokumentation der Projektumsetzung **4**
3.1 Schritt-für-Schritt-Anleitung . . . 4
3.1.1 CI Pipeline & Unit Tests . . . 4
3.1.2 Linting . . . 6
3.1.3 Dependabot version updates . . . 7
3.1.4 Statische Codeanalyse mit Sonarqube . . . 7

1 Einleitung

1.1 Aufgabenstellung

- Herausarbeiten und dokumentieren der Unterschiede zwischen der Editierung und den Features mit VSCode Frontend der Seiten https://github.dev/ , https://stackblitz.com/ und https://gitpod.io
- Umsetzung und Dokumentation (High Level) der Projektumsetzung (Inklusive Zugriff zum Source Code; entweder ein öffentliches Repo oder Einladung ins private Repo)
  - Import des Projekts https://github.com/t-stefan/FHB-Assignment-Backend

Aus diesen Punkten ist frei zu wählen (Die Anzahl der zu wählenden Punkte ergibt sich auch der Gruppengröße; Bei 2 Personen in der Gruppe sind mindestens 3 Punkte auszuwählen, Bei 3 Personen in der Gruppe sind mindestens 5 Punkte auszuwählen; Bei 4+ Personen sind alle Punkte zu erledigen.):

- Anlage eines automatischen Builds der bei jedem Pull-Request in den Main läuft und auch bei jedem Push in den Main Branch selbst
- Anlage von mindestens 3 Unit Tests
- Aufnahme der Unit Tests in den Build für jeden Pull-Request in den Main Branch sowie bei jedem Push in den Main Branch selbst
- Installation und Konfiguration (beliebige Konfiguration von jenen die bei der Installation vorgeschlagen werden) von ESLint für das Projekt
- Aufnahme von ESLint in den Build bei jedem Pull-Request in den Main Branch
- Konfiguration eines Automatismus zum Update von Fremdkomponenten wenn es eine neue Version gibt (z.B.: snyk, Dependabot, ...)
- Konfiguration von Statischer Code Analyse inklusive Quality Gate(s) Diese sollen auch bei jedem Pull-Request in den Main Branch ausgeführt werden.

1.2 Praktische Umsetzung

Die beschriebenen Schritte wurden praktisch umgesetzt im GitHub Repository `https://github.com/thomasstxyz/fhb-mcce-tmcsp-assignment-backend`

2 Browser IDE Vergleich

In diesem Abschnitt werden die drei Browser IDEs github.dev [1], stackblitz.com [2], und gitpod.io [3] grob miteinander vergleichen. Dabei wird auf den Editor und die wichtigsten Features eingegangen.

2.1 github.dev

Der Dienst github.dev von Github bietet eine leichtgewichtige IDE, welche vollständig im Browser ausgeführt wird. Mit dem webbasierten Editor können Benutzer durch Dateien und Quellcode Repositorys von GitHub navigieren und Codeänderungen vornehmen. Diese Verwendung des webbasierten Editors ist kostenlos verfügbar. Ebenfalls bietet github.dev viele der Vorteile von Visual Studio Code, z. B. Suche, Syntaxhervorhebung und eine grafische Versionskontrollansicht. Es bietet ebenfalls Settings Sync, um eigene VSCode-Einstellungen mit dem Editor zu synchronisieren. Der webbasierte Editor wird vollständig in der Sandbox des Browsers ausgeführt. Der Editor klont dabei das Repository nicht, sondern verwendet stattdessen die Github Repositories Erweiterung, um die meisten der verwendeten Funktionen auszuführen. Die laufende Anwendung wird dabei im lokalen Speicher des Browsers gespeichert, bis der Commit erfolgt. Zwischenstände können so ohne regelmäßige commits leicht verloren gehen. Zusätzlich zu Github.dev wird von Github noch die online IDE Github Codespaces angeboten, welche nicht kostenfrei genutzt werden kann. Github.dev ist dabei eher als web basierter Editor zu verstehen, als eine vollwertige Browser IDE, und besitzt daher deutlich weniger Features als Codespaces. Beispielsweise können so für Codespaces verschiedene Performance Tiers für die Entwicklungsumgebung gebucht werden, da Codespaces in einer eigenen dedizietrten VM betrieben wird. Ein wichtiges Feature, welches Codespaces von Github.dev unterscheidet ist das Terminal, welches bei Github.dev nicht vorhanden ist. Um am schnellsten Weg zum github.dev Editor zu gelangen, muss nur am URL des GitHub Repository die Top-Level-Domain ".com" durch ".dev" ausgetauscht werden.

2.2 stackblitz.com

Die Stackblitz Web IDE ist prinzipiell eher auf Web Development ausgelegt. Die Online-IDE ist dabei an Visual Studio Code angelehnt und ermöglicht es Projekte mit anderen zu teilen, anzusehen oder zu forken. Projekte können ebenfalls in komprimierter Form heruntergeladen werden und auf dem lokalen Rechner gestartet werden. Für die Nutzung gibt es keinerlei Beschränkungen hinsichtlich der genutzten Bandbreite und dem Teilen mit andern Usern. Die Bezahlten Subscriptions ermöglichen zusätzlich noch das Bearbeiten von privaten Projekten, welche auch in privaten Github Repos liegen können und ermöglichen zusätzlich noch unlimitierte Uploads von Files. Im höchsten Tier des Abos können zusätzlich neue Features früher genutzt werden und auch zum weiteren Entwicklungsprozess von Stackblitz beigetragen werden.

2.3 gitpod.io

Gitpod bietet ebenfalls eine web-basierte IDE, die ebenfalls Visual Studio Code als Editor nutzt und eine vollständige Betriebssystemumgebung bietet, um dort den entwickelten Code auszuführen. Im Gegensatz zu Github.dev bietet Gitpod ein Terminal Interface, durch welches auch der Container, in welchem Gitpod läuft bearbeitet werden kann. Während der Entwicklung können Ports beispielsweise direkt im Browser angesehen werden, oder auf den lokalen Rechner weitergeleitet werden. Für die Nutzung benötigt Gitpod dafür Github und ist mit diesem integriert. Dabei kann prinzipiell jedes Github Repo zur Erstellung einer Gitpod Workspace verwendet werden, indem eine einfache Gitpod URL daraus gebildet wird, in folgender Form `https://gitpod.io/#<url_of_github_repo>`. Gitpod bietet ebenfalls eine Google Chrome Erweiterung an, welche in GitHub einen Gitpod Button in Repos erscheinen lässt und den User an die Gitpod URL des Projekts weiterleitet.

3 Dokumentation der Projektumsetzung

3.1 Schritt-für-Schritt-Anleitung

3.1.1 CI Pipeline & Unit Tests

Zuerst wird das Basis-Repository `https://github.com/t-stefan/FHB-Assignment-Backend` im GitHub Webinterface `https://github.com/new/import` importiert.
Als nächstes müssen die GitHub Actions für das Repository unter `https://github.com/<account>/<repository>/settings/actions` aktiviert werden. Wir wählen hierfür den Punkt `Allow all actions and reusable workflows` aus und drücken anschließend auf `Save`.

Nun kann zu `https://github.com/<account>/<repository>/actions/new` navigiert werden. Hier wählen wir den Workflow `Node.js by GitHub Actions` (`https://github.com/<account>/<repository>/new/main?filename=.github%2Fworkflows%2Fnode.js.yml&workflow_template=node.js`). Nun öffnet sich ein beispielhafter Workflow als neue Datei im Editor. Mit `Start commit` speichern wir diesen. Anschließend kann zu `https://github.com/<account>/<repository>/actions/workflows/node.js.yml` navigiert werden und der erste Durchlauf unseres Workflows live angesehen werden.

Als nächsten Schritt laden wir unser Repository in einer gitpod.io Workspace (`https://gitpod.io/#https://github.com/<account>/<repository>`). Nachdem die Workspace fertig geladen ist, initialisieren wir mit

```
$ npm install
```

und starten die Applikation mit

```
$ npm start
```

Nun öffnen wir ein zweites Terminal und installieren Mocha [4] mit dem Befehl

```
$ npm install mocha --save-dev
```

Nach der Installation von Mocha, öffnen wir die Datei `package.json` und ändern den Wert von `scripts.test` auf `"mocha --exit"`. Wir navigieren zum ursprünglichen Terminal, und brechen den Prozess mit der Tastenkombination `Ctrl+c` ab.

Anschließend hängen wir folgende Zeile an das Ende der Datei `index.js`:

```
module.exports = app;
```

Weiters installieren wir das Modul `superset`.

```
$ npm install --save-dev supertest
```

Nun legen wir ein Verzeichnis namens `test` an. In diesem Verzeichnis legen wir eine Datei `index.test.js` mit folgendem Inhalt an:

```
const request = require('supertest');
const app = require('../index');

describe('GET /', () => {
it('should return "Hello World!"', () => {
return request(app)
.get('/')
.expect('<h1>Hello World!</h1>')
});

it('should return error 404, if /nonexistentpath is requested', () => {
```

```
return request(app)
.get('/non/existent/path')
.expect(404)
});
});

describe('POST /api/notes', () => {
it('should return json {"error":"content missing"}, if request body is missing', () => {
return request(app)
.post('/api/notes')
.expect('{"error":"content missing"}')
});
let query = {
content: "my super cool note",
important: true
}
it('should return 200, if json was requested with content and important fields', () => {
return request(app)
.post('/api/notes')
.set('Content-type', 'application/json')
.send( query )
.expect(200);
});
});

describe('GET /api/notes', () => {
it('should return 200 and Content-Type: application/json', () => {
return request(app)
.get('/api/notes')
.expect(200)
.expect('Content-Type',/json/)
});
});

describe('GET /api/notes/:id', () => {
it('should return 200 and known data, if existing item with id 1 is requested', () => {
return request(app)
.get('/api/notes/1')
.expect(200)
.expect('Content-Type',/json/)
.expect('{"id":1,"content":"HTML is easy","date":"2022-01-10T17:30:31.098Z","important":true}')
});
});
```

Dies sind unsere Tests, welche mit dem Befehl

```
$ npm run test
```

ausgeführt werden können.

Schließlich haben wir alle nötigen Änderungen vorgenommen. Diese müssen nur noch von gitpod zurück ins Repository gepusht werden.

```
$ git add .
$ git commit -m "add unit tests with mocha and superset"
$ git push
```

3.1.2 Linting

Zuerst installieren wir die ESLint:

```
$ npm install eslint --save-dev
```

Dann erstellen wir eine Konfigurationsdatei

```
$ npm init @eslint/config
```

und folgen dem interaktiven Installations-Setup.

Dies erzeugt die Datei `.eslintrc.js` mit folgendem Inhalt:

```
module.exports = {
env: {
browser: true,
es2021: true,
},
extends: [
'airbnb-base',
],
parserOptions: {
ecmaVersion: 'latest',
sourceType: 'module',
},
rules: {
},
};
```

Jetzt kann der Lint-Prozess ausgeführt werden:

```
$ npx eslint index.js
```

Mit `--fix` können wir dem Programm sagen, dass es versuchen soll, die Änderungen selbst automatisch vorzunehmen.

```
$ npx eslint --fix-dry-run index.js
$ npx eslint --fix index.js
```

Schließlich lassen wir den Lint-Prozess in unserer als Teil unserer CI Pipeline laufen. Dazu fügen wir den Befehl in `.github/workflows/node.js.yml` direkt nach der Zeile mit `npm ci` ein:

```
...
- run: npm ci
- run: npx eslint --fix-dry-run index.js
- run: npm run build --if-present
...
```

Als letztes müssen wir die Änderungen noch zurück in unser Repository pushen.

```
$ git add .
$ git commit -m "add ESLint syntax & style linting to workflow"
$ git push
```

3.1.3 Dependabot version updates

Dependabot kann für uns automatisch Versionen updaten. Wir erstellen eine Datei `.github/dependabot.yml` mit folgendem Inhalt:

```
version: 2
updates:
# Enable version updates for npm
- package-ecosystem: "npm"
# Look for ‘package.json‘ and ‘lock‘ files in the ‘root‘ directory
directory: "/"
# Check the npm registry for updates every day (weekdays)
schedule:
interval: "daily"
```

und pushen unsere Änderungen ins Repository:

```
$ git add .
$ git commit -m "add .github/dependabot.yml"
$ git push
```

Dependabot öffnet nun selbstständig Pull Requests, welche die Versionen der npm-Pakete updaten.

3.1.4 Statische Codeanalyse mit Sonarqube

Hierfür öffnen wir `https://github.com/marketplace/sonarcloud` im Browser und erstellen uns einen Account. Anschließend wählen wir aus, auf welche Repositories SonarCloud zugreifen darf. Um eine Analyse zu starten wählen wir das gewünschte Projekt aus und klicken auf `Set up`. Danach wird das Projekt analysiert, dies kann abhängig von der Größe des Projekts einige Zeit in Anspruch nehmen.

Jetzt wollen wir die statische Code-Analyse in einem GitHub Workflow als Teil unserer CI Pipeline definieren.

Zuerst erstellen wir ein Authentication Token in SonarCloud `https://sonarcloud.io/account/security/`. Dazu vergeben wir einen beliebigen Namen und bestätigen mit dem Button `Generate`. Danach wird unser Token ausgegeben. Dieses kopieren wir in unsere Zwischenablage. Nun erstellen wir auf GitHub in unserem Repository ein Secret unter `https://github.com/<account>/<repository>/settings/secrets/actions`. Der Name muss dabei `SONAR_TOKEN` entsprechen, und als Value fügen wir unser Token aus der Zwischenablage ein.

Nun legen wir eine neue Datei `sonar-project.properties` im Rootverzeichnis unserer Repository an, mit folgendem Inhalt:

```
sonar.organization=<replace with your SonarCloud organization key>
sonar.projectKey=<replace with the key generated when setting up the project on SonarCloud>

# relative paths to source directories. More details and properties are described
# in https://sonarcloud.io/documentation/project-administration/narrowing-the-focus/
sonar.sources=.
```

Die Variablen `sonar.organization` und `sonar.projectKey` ändern wir. Unseren SonarCloud Organization Key können wir unter `https://sonarcloud.io/account/organizations` einsehen. Den Project Key sehen wir uns ebenfalls im SonarCloud Web Interface an.

Als nächstes legen wir die Datei `.github/workflows/static-code-analysis.yml` für den Workflow an, mit folgendem Inhalt:

```
on:
# Trigger analysis when pushing in main or pull requests, and when creating
# a pull request.
push:
branches:
- main
pull_request:
types: [opened, synchronize, reopened]
name: Main Workflow
jobs:
sonarcloud:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
with:
# Disabling shallow clone is recommended for improving relevancy of reporting
fetch-depth: 0
- name: SonarCloud Scan
uses: sonarsource/sonarcloud-github-action@master
env:
GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

Schließlich pushen wir unsere Änderungen ins Repository:

```
$ git add .
$ git commit -m "add workflow for static code analysis with SonarCloud"
$ git push
```

Jetzt kann die automatische Analyse in der SonarCloud deaktiviert werden, weil sonst im GitHub Workflow deswegen ein Fehler geworfen wird. Dies im SonarCloud Project unter Administration - Analysis Method vorgenommen werden.

References

[1] github.dev. https://github.dev/, 2022. (Accessed on 05/29/2022).

[2] stackblitz.com. https://stackblitz.com/, 2022. (Accessed on 05/29/2022).

[3] gitpod.io. https://gitpod.io/, 2022. (Accessed on 05/29/2022).

[4] mochajs.org. https://mochajs.org/, 2022. (Accessed on 05/29/2022).