Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2022

Mikael Thornberg

Yksikkö- ja integraatiotestien yhdistäminen jatkuvan integraation putkeen GitHubissa

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2022 | 19

Mikael Thornberg

Yksikkö- ja integraatiotestien yhdistäminen jatkuvan integraation putkeen GitHubissa

Yksikkö- ja integraatiotestaus ovat menetelmiä, joilla varmistetaan, että ohjelmisto toimii odotetulla tavalla. Jatkuva integraatio (CI) on ohjelmistokehityksessä käytetty käytäntö, minkä tarkoituksena on automatisoida koodimuutosten yhdistäminen yhteiseen päähaaraan tietovarastossa.

Tämän opinnäytetyön tavoitteena oli toteuttaa Proof of Concept -ohjelma, jonka yksikkö- ja integraatiotestit suoritetaan luodussa jatkuvan integraation putkessa GitHubin tarjoamalla Actions alustalla. Teoreettisessa osassa opinnäytetyötä tarkastellaan GitHubin Actions alustan ominaisuuksia ja muita käytettyjä teknologioita PoC-ohjelmassa. Tämän lisäksi työssä perehdytään syvällisemmin yksikkö- ja integraatiotestaamiseen parhaimpiin käytäntöihin ja niiden tuomiin etuihin sovelluskehityksessä.

Opinnäytetyön tuloksena saatiin suunniteltua Proof of Concept -ohjelma valmiiksi. Ohjelmalle tehtiin yksikkötestejä, joita päästiin ajamaan luodussa CI-putkessa käyttäen GitHubin Actions alustaa. Actions todettiin toimivaksi alustaksi jatkuvan integraation putken toteuttamiseksi ohjelmistoprojekteille riippumatta ohjelmointikielestä ja ajoympäristöstä.

Asiasanat:

GitHub, CI-putki, Proof of Concept, Actions

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2022 | 19

Mikael Thornberg

Unit and Integration testing in a GitHub CI pipeline

Unit and integration testing are methods which are used to ensure that the software works as expected. Continuous integration (CI) is a practice used in software development to automate the merging of code changes to a common main branch in the remote repository.

The purpose of this thesis was to implement a Proof of Concept application on which unit and integration tests are run on created continuous integration pipeline in GitHub’s Actions platform. The theoretical part of the thesis examines the features of GitHub Actions and other used technologies in the Proof of Concept application. In addition, the thesis examines more in-depth unit and integration testing best practices and the benefits that they bring in the process of developing software.

As a result of the thesis, the planned Proof of Concept application was completed. Unit tests were created for the program, which were successfully run in the created pipeline using the GitHub Actions platform. Actions platform was verified as a suitable platform for implementing a continuous integration pipeline for software projects regardless of the programming language and the runtime.

Keywords:

GitHub, pipeline, Actions, Proof of Concept

Sisältö

Käytetyt lyhenteet

CD	Jatkuva julkaisu (engl. Continuous deployment)
CI	Jatkuva integraatio (engl. Continuous integration)
PoC	Soveltuvuusselvitys (engl. Proof of Concept)
SUT	Testattava järjestelmä (engl. System under test)

1 Johdanto

Ohjelmistotestaus on prosessi, jossa arvioidaan ja mitataan, että ohjelmisto tai sovellus toimii odotetulla tavalla. Ohjelmistotestauksen hyötyjä ovat muuan muassa virheiden ennenaikainen havainnointi, kehitystyön kustannusten laskeminen ja suorituskyvyn lisääminen. Tämän vuoksi on tärkeää, että ohjelmistotestaaminen on osa projektia kehitystyön aikaisessa vaiheessa. [1]

Yksikkötestaaminen on tietokoneohjelman testaamisen ja laadunvarmistuksen menetelmä, jossa lähdekoodin osat testataan yhdessä tai erikseen. Yksikkötestaamisen tarkoituksena on eristää kirjoitettu koodi muusta toiminnallisuudesta ja varmistaa koodin toiminta. [1]

Tässä opinnäytetyössä tarkastellaan automaattisesti suoritettavien yksikkö- ja integraatiotestien lisäämistä osaksi jatkuvan integraation (engl. Continuous Integration, CI) putkea. Yksikkö- ja integraatiotestit toteutetaan Proof of Concept (PoC) -ohjelmaan. Jatkuvan integraation putki toteutetaan GitHubissa käyttäen hyödyksi palvelun tarjoamia toimintoja (engl. *Actions*). Opinnäytetyön tarkoituksena on tarkastella, mitä on yksikkö- ja integraatiotestaaminen ja miten näitä voidaan hyödyntää osana jatkuvan integraation putkea.

2 Teknologiat

Ohjelmiston käyttämiä ja kehitystyössä käytettävien komponenttien ja teknologioiden kokonaisuutta kutsutaan teknologiapinoksi (engl. Tech stack), joka koostuu muuan muussa eri ohjelmointikielistä, ohjelmistokehyksistä (engl. Frameworks) ja kirjastoista. [2]

2.1 JavaScript ja TypeScript -ohjelmointikielet

JavaScript on kevyt, dynaaminen ja yksisäikeinen ohjelmointikieli, jota käytetään verkkoympäristöissä ohjelmointikielenä. JavaScript tukee olio-ohjelmointia ja funktionaalista ohjelmointityyliä. [3]

TypeScript on Microsoftin kehittämä ja ylläpitämä tyypitetty ohjelmointikieli, joka on rakennettu JavaScriptin ympärille. TypeScript laajentaa JavaScriptin toiminnallisuutta ja havaitsee kehittäjän kehitysvaiheessa tekemät virheet, joka mahdollistaa turvallisemman kehittäjäkokemuksen. TypeScript-koodi kääntyy JavaScript-koodiksi, mikä mahdollistaa sen käytön samoissa ympäristöissä kuin JavaScriptinkin. [4]

2.2 Node.js-ajoympäristö

Node.js on avoimen lähdekoodin monialustainen asynkroninen tapahtumapohjainen JavaScript-ajoympäristö. Node.js-ohjelmat kirjoitetaan JavaScriptillä tai TypeScriptillä, jotka voidaan suorittaa Windows-, Linux- ja OS X -ympäristöissä. [5]

2.3 Express.js-palvelinkehys

Express.js on minimaalinen Node.js verkko-ohjelmien verkkokehys. Express.js on suunniteltu verkkosovelluksien ja API-ohjelmointirajapintojen nopeaan

suunnitteluun ja toteutukseen. Se on suosituin Node.js verkkokehys, jota on kutsuttu myös de facto standardi verkkokehykseksi Node.js-ohjelmille. [6]

2.4 Yarn-pakettienhallinta työkalu

Yarn on nopea, luotettava ja turvallinen pakettienhallinta työkalu Node.js-ympäristöille. Yarnin tärkeimpiin ominaisuuksiin kuuluvat muun muassa pakettien asentaminen välimuistista ja pakettien asennus rinnakkaisajona. [7]

2.5 Jest-ohjelmointikehys

Jest on Facebookin kehittämä JavaScript-testaamiseen tarkoitettu ohjelmointikehys, jossa on kaikki tärkeimmät testaukseen liittyvät työkalut testitapausten kirjoittamiseen kattavalla dokumentaatiolla. Jest tähtää yksinkertaiseen käyttökokemukseen ja siihen, että se toimisi ilman konfigurointia mahdollisimman monessa JavaScript-ohjelmointikieltä tukevassa suoritusympäristössä. [8]

Kuva 1 näyttää Jestillä ajetuiden testien tulosteen, jossa 12 testiä on ajettu onnistuneesti.

```
D:\Projects\oppari-backend> yarn test
yarn run v1.22.5
$ jest
 PASS  src/tests/unit/services/tokenService.test.ts
 PASS  src/tests/unit/services/hashService.test.ts
 PASS  src/tests/unit/controllers/helloController.test.ts
 PASS  src/tests/unit/controllers/authController.test.ts

Test Suites: 4 passed, 4 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        3.316 s, estimated 22 s
Ran all test suites.
Done in 4.40s.
```

Kuva 1. Komentorivin tuloste onnistuneista yksikkötesteistä.

3 CI/CD-menetelmät

CI/CD on menetelmä, jonka tarkoituksena on automatisoida sovelluksen julkaisuprosessi. CI/CD-menetelmää käyttämällä sovelluksen uudistuksien ja päivityksien toimittaminen loppukäyttäjille on joutuisaa. [9]

3.1 Jatkuva integraatio

Jatkuva integraatio (CI) on DevOps-malli, mikä on ohjelmistokehityksessä käytetty käytäntö, jossa kehittäjät julkaisevat säännöllisesti muutoksia yhteiseen koodikantaan. Tyypillisiä jatkuvan integraation putkessa suoritettavia prosesseja ovat esimerkiksi yksikkö- ja automaatiotestien suorittaminen. Jatkuvan integraation tarkoituksena on paikantaa virheitä tehokkaammin, parantaa ohjelmiston laatua ja luotettavuutta sekä nopeuttaa validointia ja muutosten julkaisua. [10]

3.2 Jatkuva toimitus ja julkaisu

Lyhenteellä CD viitataan jatkuvaan toimitukseen ja julkaisuun. Jatkuvalla toimituksella tarkoitetaan lähdekoodin automaattista validointia ennen sen yhdistämistä yhteiseen päähaaraan tietovarastossa. Jatkuva julkaisu on viimeinen CI/CD-menetelmän osa, joka tarkoittaa sovelluksen automaattista julkaisua tuotantoon loppukäyttäjille. [9]

3.3 Versiohallinta

Versiohallinta, joka tunnetaan myös nimellä lähdehallinta, on ohjelmistokoodin muutoksien seurantaan tarkoitettu apukeino. Versiohallintajärjestelmät ovat ohjelmistotyökaluja, jotka auttavat ohjelmistotiimejä hallitsemaan lähdekoodin muutoksia. Versiohallinta tarkkailee jokaista ohjelmistokoodin muutosta ja tallentaa nämä siihen tarkoitettuun erikoistietokantaan. [11]

3.4 Actions

GitHubin CI/CD-alustaa kutsutaan nimellä Actions. Alusta mahdollistaa esimerkiksi automaattisen testaamisen ja ympäristöjen automaattisen konfiguraation. Kuten Kuvasta 2 voidaan havaita alustalla voi määritellä suoritusprosesseja (engl. Workflows), jotka automaattisesti konfiguroivat ympäristön suoritettavalle lähdekoodille ja suorittavat esimerkiksi yksikkö- ja integraatiotestit luodussa ympäristössä. Kehittäjien tekemän muutosehdotuspyynnön jälkeen tietovarastossa, esimerkiksi GitHubissa, ajetaan nämä Workflow’t automaattisesti. [12]

Kuva 2. Esimerkki suoritetusta Workflow’sta GitHubin käyttöliittymässä.

3.5 Workflow

Workflow on konfiguroitava automaattinen prosessi, joka ajaa yhden tai useamman tehtävän. Workflow’t on määritelty YAML-tiedostoon GitHubin tietovarastossa. Ne voidaan suorittaa tietovarastoissa tapahtuneen tapahtuman seurauksena tai manuaalisesti käyttäjän toimesta. [12]

GitHubissa Workflow’t on määritelty kansiopolkuun ”.github*/workflows”* tietovarastossa, joka sijoitetaan ohjelmistoprojektin juurihakemistoon. Tähän kansion luodaan YAML-tiedosto, joka sisältää sarjan vaiheita (engl. jobs).

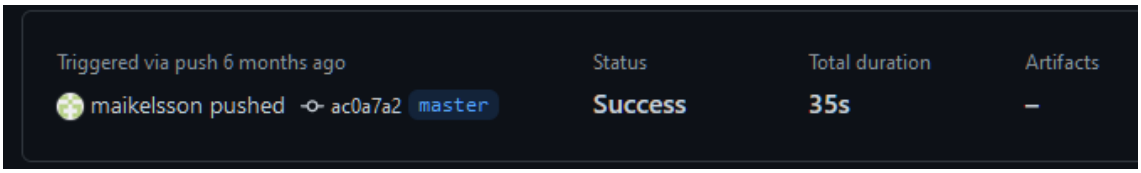
Vaiheet suoritetaan järjestyksessä ylhäältä alaspäin. [12] Kuten Kuvasta 3 käy ilmi, ensimmäisenä Workflow'lle annetaan tunnistettava nimi ja laukaiseva tapahtuma, jolloin ajon halutaan tapahtuvan. Tämän jälkeen voidaan antaa sarja suoritettavia tehtäviä.

```yaml
name: Unit tests CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [14.x, 16.x]
    steps:
      - uses: actions/checkout@v2
      - name: Use Node.js ${{ matrix.node-version }}
        uses: actions/setup-node@v1
        with:
          node-version: ${{ matrix.node-version }}
      - run: yarn --frozen-lockfile
      - run: yarn test
```

Kuva 3. Esimerkki YAML-tiedoston sisällöstä.

3.6 Tapahtuma

Tapahtuma on toiminto tietovarastossa, mikä käynnistää Workflow'n. Toiminto voi olla esimerkiksi kehittäjän tekemä vetoehdotus (engl. pull request) lähdekoodiin tai lähdekoodin varastointi (engl. push) tietovarastoon, kuten Kuvasta 4 voidaan havaita. [12]



Kuva 4. Esimerkki Worfklow'n laukaisevasta tapahtumasta.

4 Yksikkötestaaminen

Yksikkötestaaminen on testausmenetelmä, joka on tärkeässä roolissa jo varhain ohjelmiston kehittämisvaiheessa. Jos yksikkötestit on kirjoitettu hyvien käytäntöjen mukaisesti, voivat ne paljastaa aikaisessa vaiheessa tehdyt virheet ohjelmistossa. Myöhemmässä kehitysvaiheessa virheiden paikantaminen voi olla vaikeampaa. [1]

4.1 Mitä on yksikkötestaaminen

Yksikkötesti on metodi, joka alustaa pienen osan koodia ohjelmasta ja tarkistaa sen toiminnallisuuden eristyksessä muista ohjelman riippuvuuksista. Kuten Kuvasta 5 käy ilmi, tyypillisesti yksikkötesti koostuu kolmesta vaiheesta, joka tunnetaan nimellä AAA-pattern. Alustusvaiheessa (engl. Arrange) murto-osa ohjelmasta alustetaan, eli se osa ohjelmakoodia, jota ollaan testaamassa. Tämä on tyypillisesti luokan instanssi ja tunnetaan nimellä testattava järjestelmä (engl. sut). Toimintavaiheessa (engl. Act) suoritetaan tyypillisesti funktiokutsu tämän alustetun luokan metodilla, jota ollaan testaamassa. Validointi (engl. Assert) on toimintavaiheen tuloksen vertailua odotettuun lopputulokseen. [13]

```
test("should throw error when username already exists", async () => {
  // Arrange
  mockUserRepository.insert = jest.fn().mockImplementation(() => {
    throw new Error("Register failure");
  });
  const sut = new AuthController(mockUserRepository);
  // Act
  const result = await sut.register({ username: "mock", password: "mock" });
  // Assert
  expect(result).toEqual({ error: "Register failure" });
});
```

Kuva 5. Esimerkki AAA-patternista.

4.2 Hyvän yksikkötestin ominaisuudet

Hyvältä yksikkötestiltä vaaditaan useita ominaisuuksia. Eräs ominaisuus on se, että yksikkötestin kirjoittamisen pitää olla helppoa, joka tarkoittaa sitä, että lähdekoodin pitää olla helposti testattavaa. Helposti testattava koodi vähentää kehittäjän taakkaa kirjoittaa yksikkötestejä. Kehittäjät kirjoittavat yksikkötestejä, jotka mittaavat ohjelman toiminnallisuutta monelta kannalta. On myös tärkeää, että yksikkötestin tarkoitus on selkeä. Hyvä yksikkötesti kertoo tarinan kehittäjälle, mitä ja miten ohjelman osan toiminnallisuutta ollaan testaamassa. Yksikkötestin tulee olla luotettava, eli yksikkötesti epäonnistuu vain, jos ohjelman lähdekoodissa on virhe ilman ulkoisia vaikuttavia tekijöitä. Hyvän yksikkötestin tulisi olla riippumaton alustasta, ulkoisista riippuvuuksista tai suoritusjärjestyksestä. Yksikkötesti testaa toiminnallisuutta. Testattavan järjestelmän ei tulisi kutsua ulkoisia palveluita esimerkiksi tehdä verkkopyyntöjä tai lukea dataa tietokannasta. Tämä eliminoi ulkoisten riippuvuuksien vaikutuksen yksikkötestien toiminnallisuuteen. [13]

5 Integraatiotestaaminen

Integraatiotestaamisessa testataan ohjelmistomoduulien ja -komponenttien integrointia sekä niiden toiminnallisuutta yhtenä kokonaisuutena. Tyypillisesti ohjelmisto koostuu useista pienistä eri ohjelmistomoduuleista, jotka yhdistetään toisiinsa tyypillisesti rajapintojen avulla. Integraatiotestaamisen tarkoituksena on paljastaa mahdolliset virheet ohjelmassa, jotka ilmenevät näiden ohjelmistomoduulien integrointivaiheessa. [14]

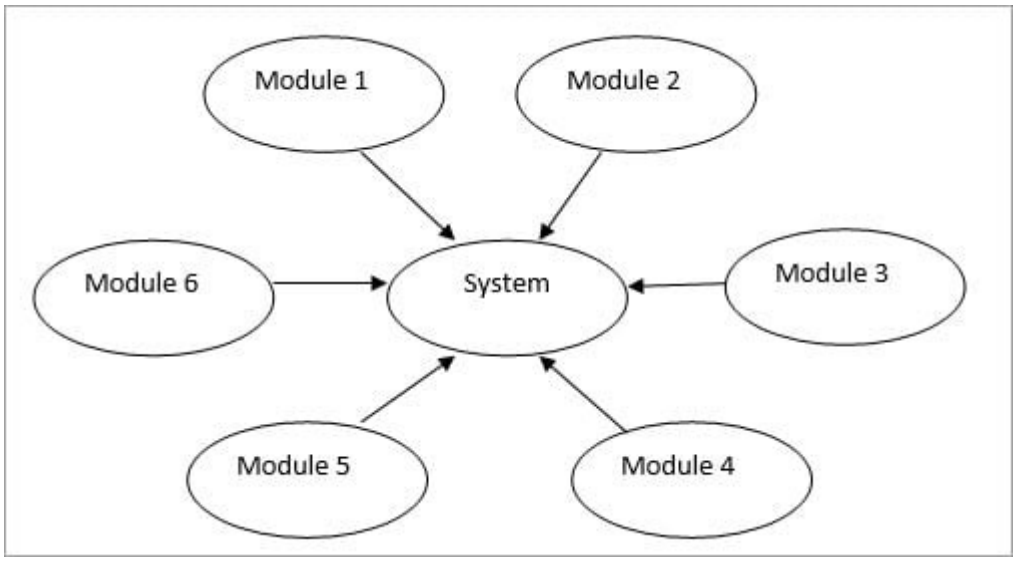
5.1 Integraatiotestausstrategiat

On olemassa erilaisia strategioita integraatiotestien suorittamiseksi. Tunnetuimmat strategiat ovat Big-Bang-lähestymistapa ja lisääntyvä lähestymistapa, joista jälkimmäinen on jaettu seuraaviin omiin kategorioihinsa: [15]

- ylhäältä alas -lähestymistapa
- alhaalta ylöspäin -lähestymistapa
- inkrementaalinen lähestymistapa - ylhäältä alas ja alhaalta ylöspäin yhdistelmä.

5.2 Big-Bang

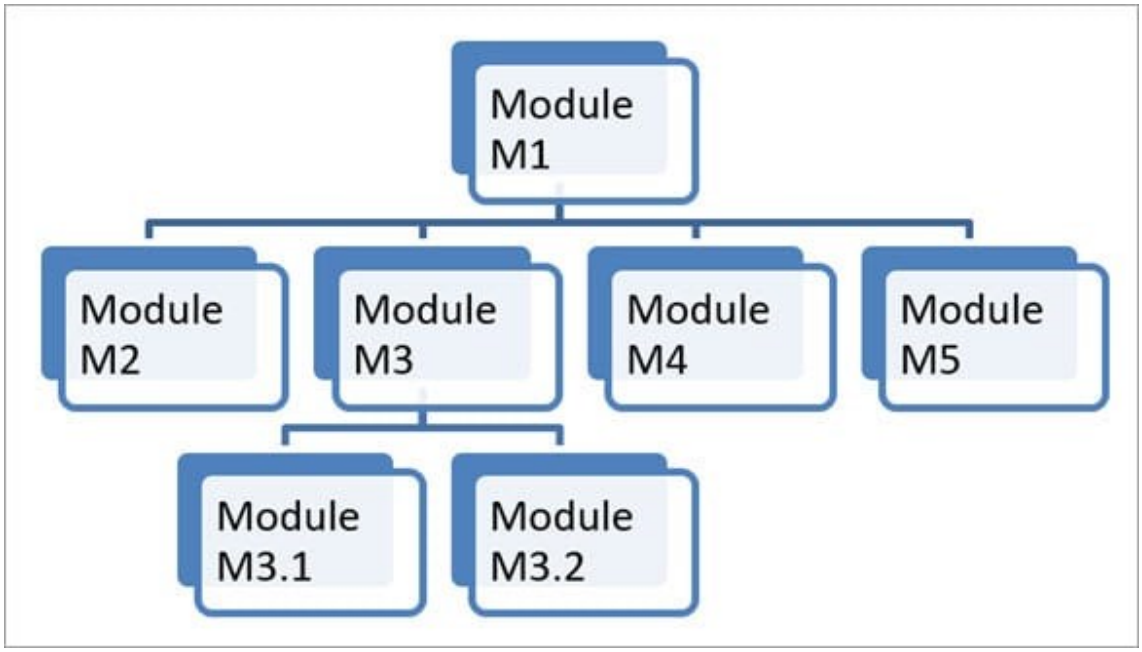
Big-Bang integraatiotestaus strategia yhdistää kaikki ohjelmistomoduulit kokonaisuudeksi kuten Kuviossa 1 on esitetty. Strategia varmistaa ohjelmiston toiminnan kokonaisuudessaan. Jos kokonaisuudessa havaitaan ongelma, on haastavaa havaita ongelman aiheuttanut ohjelmistomoduuli. Big-Bang on todettu hyväksi lähestymistavaksi ohjelmistoille, joiden ohjelmistomoduulien määrä on pieni. [15]



Kuvio 1. Big-Bang strategian mallinnus. [15]

5.3 Lisääntyvä lähestymistapa

Lisääntyvän lähestymistavan integraatiotestauksen strategiassa ohjelmistomoduulit testataan ensimmäiseksi yksikkötestien avulla. Yksikkötestauksen jälkeen ohjelmistomoduuleita lisätään yksitellen mukaan integraatiotesteihin, joiden toiminnallisuutta voidaan jäljitellä (engl. Stubbing), eli toiminnallisuus voi olla esimerkiksi ohjelmistomoduulin rajapinnan vastaus. Ohjelmistomoduuleita lisätään mukaan integraatiotesteihin, kunnes järjestelmän kaikki ohjelmistomoduulit ovat yhdistetty toisiinsa muodostaen kokonaisen ohjelmiston kuten Kuviossa 2 on esitetty. [16]



Kuvio 2. Ohjelmistomoduulien testaaminen lisääntyvän lähestymistavan strategialla. [16]

6 Lopuksi

Tämän opinnäytetyön tavoitteena oli toteuttaa Proof of Concept -ohjelma, jonka lähdekoodia on mahdollista yksikkö- ja integraatiotestata jatkuvan integraation putkessa käyttäen hyödyksi GitHubin Actions-ominaisuutta. Opinnäytetyön tuloksena luotiin toimiva ja yksinkertainen Proof of Concept -ohjelma. Tämän lisäksi luotiin CI-putki GitHubiin käyttäen Actions alustaa.

Teknologioiden valinnat onnistuivat ja osoittautuivat helppokäyttöisiksi. CI-putken toteutus GitHubiin sujui helposti ja vaivattomasti. Opinnäytetyön työstämisen aikana opin paljon GitHubin Actions-ominaisuudesta ja yksikkötestaamisen parhaimpia käytäntöjä. Tämän työn tuloksena voin todeta, että GitHubin Actions-ominaisuus soveltuu hyvin ohjelmistoprojekteille, jossa on tarvetta jatkuvan integraation putkelle riippumatta ohjelmiston ohjelmointikielestä tai suoritusympäristöstä. Olen tyytyväinen työn tulokseen.

Työtä voisi jatkaa tutkimalla kattavammin jatkuvan toimituksen ja julkaisun putkea jatkuvan integraation yhteydessä ja näiden tuomia etuja sovelluskehityksessä. Tämän toteuttaminen vaatisi pilvipalvelimen, johon sovelluksen koodi taltioitaisiin, sekä ylimääräistä konfigurointia niin ohjelman lähdekoodissa kuin mahdollisella palvelimella.

Lähteet

[1] TechTarget Contributor, ”What Is Unit Testing?,” [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing. [Haettu 01 06 2022].

[2] stackshare, ”What is a Tech Stack? What tools do popular tech companies use in 2022?,” [Online]. Available: https://stackshare.io/stacks. [Haettu 8 Joulukuu 2022].

[3] MDN Contributors, ”JavaScript,” Mozilla Foundation, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript#reference. [Haettu 29 Toukokuu 2022].

[4] Microsoft, ”What is TypeScript?,” Microsoft, [Online]. Available: https://www.typescriptlang.org/. [Haettu 30 Marraskuu 2022].

[5] tutorialspoint.com, ”Node.js - Introduction,” [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm#. [Haettu 01 Kesäkuu 2022].

[6] Simplilearn, ”What Is Express JS In Node JS?,” [Online]. Available: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js. [Haettu 01 Kesäkuu 2022].

[7] Waverley Team, ”Yarn vs NPM: Why and How to Migrate from NPM to Yarn,” 17 Maaliskuu 2017. [Online]. Available: https://waverleysoftware.com/blog/yarn-vs-npm/. [Haettu 8 Joulukuu 2022].

[8] SoftwareTestingHelp, ”Jest Tutorial – JavaScript Unit Testing Using Jest Framework,” [Online]. Available: https://www.softwaretestinghelp.com/jest-testing-tutorial/. [Haettu 01 Kesäkuu 2022].

[9] Red Hat, ”What is CI/CD?,” [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd. [Haettu 8 Joulukuu 2022].

[10] M. REHKOPF, ”What is continuous integration?,” [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration. [Haettu 02 Kesäkuu 2022].

[11] Atlassian, ”What is version control?,” [Online]. Available: https://www.atlassian.com/git/tutorials/what-is-version-control. [Haettu 02 Kesäkuu 2022].

[12] Github Inc., ”Understanding GitHub Actions,” [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. [Haettu 7 Syyskuu 2022].

[13] Toptal, ”How to write testable code and why it matters,” [Online]. Available: https://www.toptal.com/qa/how-to-write-testable-code-and-why-it-matters. [Haettu 31 Toukokuu 2022].

[14] R. Awati, ”integration testing or integration and testing (I&T),” [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/integration-testing. [Haettu 30 Marraskuu 2022].

[15] SoftwareTestingHelp, ”What is Integration Testing: Learn with Integration Testing Examples,” [Online]. Available: https://www.softwaretestinghelp.com/what-is-integration-testing/. [Haettu 31 Toukokuu 2022].

[16] S. Qadir, ”What is incremental testing?,” [Online]. Available: https://www.educative.io/answers/what-is-incremental-testing. [Haettu 9 Joulukuu 2022].