Ohjelmistojen testaus harjoitustyö osa 2

COMP.SE.200-2025-2026-1 Software Testing

Jalmari Mieho H299715
Antti Veikkolainen 153938689

https://github.com/jarmlari/COMP.SE.200-2024-2025-1

https://coveralls.io/github/jarmlari/COMP.SE.200-2024-2025-1

Lyhennykset

Npm: Node package manager

Npx: Node package execution (tool)

Johdanto

Tässä dokumentissa käydään läpi aiemmin määritetyn testi suunnitelman toteutus vaihetta, siitä koituneita löydöksiä, sekä tuloksia.

Dokumentin tarkoituksena tuottaa ymmärrystä siitä, kuinka testit toteutetiin, sekä muodostaa analyysiä siitä, millaisella tasolla testattavat yksiköt olivat.

CI-pipeline toteutus ja testit

Toteutus ja muutokset suhteessa suunnitelmaan

Issues toiminto ei ollut saatavilla repositoriossa, joten bugeja tai muita ongelmia ei voitu raportoida repositorio tasolla. Sen sijaan kuitenkin löydetyt ongelmat raportoidaan myöhemmin tässä raportissa.

Yksikkötestit kirjoitettiin samoille funktioille kuin työn osassa yksi suunniteltiin. Add.js testeistä jätettiin pois vaatimus siitä, että suuret parametri arvot palauttaisivat false. Add.js testeihin lisättiin testitapauksia katsomaan, että jos funktiota kutsuu väärän tyyppisillä parametreillä, palauttaa funktio esimerkiksi false.

GitHub Actions kuvaus

```yaml
# GitHub Actions Workflow for Running Tests and Uploading Coverage to Coveralls

name: "main push"

on: push

jobs:
  first_job:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      # Set up Node.js environment
      - name: npm install
        run: npm install --save-dev c8 coveralls

      # Run tests and generate coverage report
      - name: Run tests with coverage
        run: npx c8 --reporter=lcov npm test

      # Upload coverage report to Coveralls
      - name: Coveralls
        if: always()
        uses: coverallsapp/github-action@v2
        with:
          github-token: ${{ secrets.GITHUB_TOKEN}}
          coveralls-token: ${{ secrets.COVERALLS_REPO_TOKEN }}
          path-to-lcov: ./coverage/lcov.info

```

Kuva 1. GitHub Actions 8.12.2025

GitHub Action (ks. Kuva 1) on määritetty suoritettavaksi, aina kun projektin Git repositorioon pusketaan uutta koodia. Actions putki ajaa komennon *”npm install –save-dev c8 overalls”*. Komento asentaa c8 työkalun ja coveralls paketin. Komento *“npx c8 --reporter=lcov npm test”* ajaa testit ja käynnistää c8 työkalun, joka tuottaa kattavuusraportin. Lopussa määritellään coveralls integraatio tai yhteys.

Paikallisesti, yksikkötestit voidaan ajaa *”npm test”* tai *”npm test add.js”* kommennolla, jos halutaan ajaa testit yksittäiselle tiedostolle. Paikallisesti ajettuna, kattavuus raportit menevät kansioon coverage/lcov-report.

Ohjeet testien ajamiseen ja kattavuusraportin luomiseen paikallisesti

Testejä ajettiin paikallisesti avaamalla jokin terminaali projektin kansion sisällä ja ajamalla komento npm test. Ensimmäisellä kerralla npm install on varmaan tarpeen. npm test FunktionNimi.js komennolla pystyi ajamaan vain tietyn funktion testit. Kattavuusraportin luominen onnistuu komennolla *“npx c8 --reporter=lcov npm test”*. Raportti menee kansioon coverage/lcov-report ja löytyy tiedostosta index.html.

Ohjeet askel askeleelta:

1. Kopioi meidän GitHub repositorio itsellesi paikallisesti.
2. Avaa jokin komentokehote juuri kloonatun paikallisen repon sisällä ja aja komento: *npx c8 --reporter=lcov npm test*

3. Kattavuusraportti tulee kansioon coverage/lcov-report tiedostoon index.html.

Erot testeissä AI:lla ja ilman AI:ta

Add.js

Ilman kielimallin apua lähdin muodostamaan funktiolle testejä yrittämällä ajatella erilaisia tapauksia, miten funktiota voisi kutsua. Mielessä oli “normaalit” kutsut suhteellisen pienillä luvuilla ja sitten ääritapaukset todella suurilla tai pienillä luvuilla. Väärillä parametri tyypeillä testaaminen muistui mieleen vasta työn osan 1 palautteesta, kun työn tarkastaja mainitsi tästä palautteessa.

Annoin add.js funktion Clauden kielimallille ja pyysin sitä muodostamaan funktiolle jest yksikkötestit. Sain vastaukseksi suhteellisen samanlaiset testit kuin olin itsekin muodostanut. Molemmat koostuivat karkeasti jaoteltuna “normaalien tapausten” testeistä, ääriarvotesteistä ja väärien parametri tyyppien testeistä. Clauden muodostama testisetti oli kattavampi kuin omani. Kaikkien edellä mainittujen kategorioiden testejä oli Clauden setissä enemmän.

Tulkitsen tästä, että itse tekemäni testisetti oli suhteellisen kattava eikä täysin päin puuta tehty, sillä yksikkö testit ovat ymmärtääkseni kohtuullisen triviaali asia ja siten kielimallille helppo tai sopiva työ ja omat testini vastasivat Clauden tuottamia testejä suurin piirtein. Siten voin myös sanoa, että testien kirjoitus itse yhdelle funktioille ei ollut suunnattoman työlästä. Mietin, että kielimallia voi pyytää avuksi yksikkötestien kirjoittamiseen silloin, kun niiden tuottamiseen on kova kiire tai niiden laatua halutaan hienosäätää.

Filter.js

Voidaan kyseenalaistaa, ovatko funktion palautus arvossa bugi, vai funktion dokumentaatiossa virhe. Funktio sanoo että palautettava arvo olisi array [] tyyppinen, mutta todellisuudessa se on 2D array [[]].

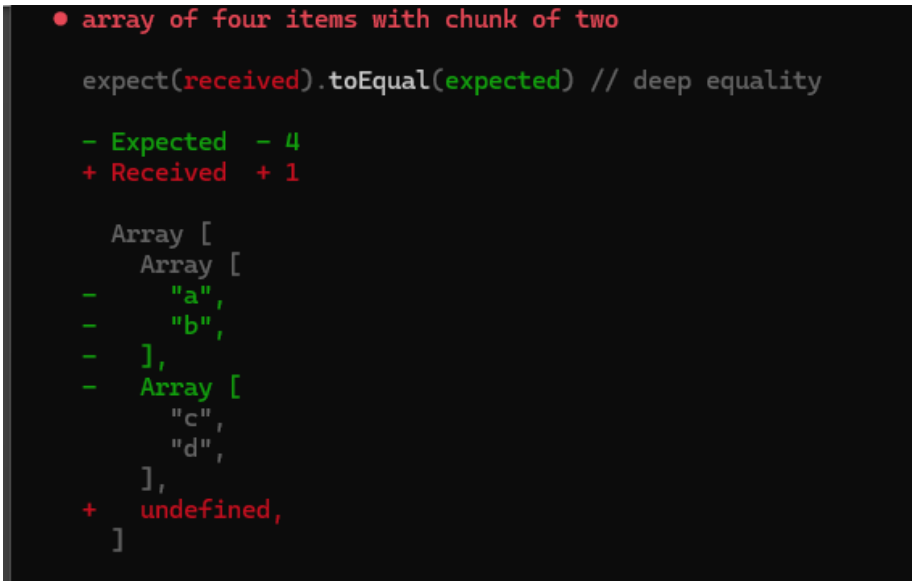
Löydökset ja päätelmät

Bugiraportit

Chunk.js

Chunk.js vaikuttaa toimivan virheellisesti. Funktion palauttamien Array olioiden loppupäähän ilmestyy ylimääräisiä undefined arvoja. Funktiolle kirjoitetut testit kutsuvat sitä esimerkiksi sen omassa

tiedostossa kirjoitettujen esimerkkien mukaisesti. Esimerkiksi kutsu: chunk(['a', 'b', 'c', 'd'], 2) pitäisi palauttaa [['a', 'b'], ['c', 'd']], mutta se antaa kuvan mukaisen palautuksen:



Kuva 2.Chunk.js funktion esimerkki virhe.

Toinen chunkin omassa tiedostossa oleva esimerkki kutsu chunk(['a', 'b', 'c', 'd'], 3) pitäisi palauttaa [['a', 'b', 'c'], ['d']], mutta se palauttaa:

Kuva 3. Chunk.js funktion esimerkki virhe 2.

Capitalize.js

Tyhjällä parametrillä tämä funktio palauttaa merkkijonon ”Undefined”. Tämä johtuu siitä, että toString metodi muuttaa avainsanan *”undefined”* merkkijonoksi ja tekee operaation tälle. Teknisesti, funktio toimii, mutta funktion todennäköisesti pitäisi sanitoida tyhjät merkkijonot tämän välttämiseksi.

Funktiot sallivat kutsuja väärillä parametri tyypeillä

Monet funktioista sallivat kutsumisen väärän tyyppisillä parametreillä ja yrittävät tehdä palautuksen, vaikka annettu parametri olisi väärä. Esimerkiksi add.js tiedostossa on kirjoitettu, että funktio ottaa

kaksi parametriä tyyppiä *number*. Kuitenkin funktiota voi kutsua esimerkiksi add(false, "123"), jolloin se palauttaa ”false123”.

Testatun kirjaston laatuarvio ja testien kattavuus

Suurin osa testatuista funktioista yrittää palauttaa jonkinlaisen arvon oli sille annetut parametrit minkä tyyppisiä tahansa. Tämä johtaa mahdollisesti odottamattomiin ja outoihin palautusarvoihin, mihin kutsuva koodi ei välttämättä ole varautunut. Tämä voi johtaa myös siihen, että sovellus kaatuu eli sovelluksen käyttö estyy mahdollisesti täysin, jos funktioita onnistutaan kutsumaan väärillä parametreillä. Koska kyseinen haaste koskee suurinta osaa funktioista ja esimerkiksi Chunk.js tiedostossa on suurempi vika, kirjastoa ei voi käyttää vielä tuotannossa. Toisaalta koko tilanteen arvioimiseksi olisi hyvä nähdä funktioita kutsuva koodi. Jos ne varautuvat vikatilanteisiin sekä vääriin arvoihin aina, niin tilanne on toinen. Herää myös ajatus, että ennen funktioiden kutsumista voidaan periaatteessa varmistaa, että kutsu tehdään oikeilla parametreillä.

Sovellus, joka testattua kirjastoa käyttäisi, voi mahdollisesti olla tuotanto valmis, riippuen osaavatko kutsuvat metodit sanitoida tulokset tai käsittelevätkö ne virheet huolellisesti, sekä riippuen miten kirjastoa käytetään. Kuitenkin riippuen siitä kuinka kriittisiksi bugit halutaan luokitella, voidaan tarvittaessa todeta, että sovellus ei ole tuotanto valmis.

Kirjaston testit kattavat noin 93 prosenttia valituista funktioista. Testit kattavat myös 112 / 158 eri oksaa, tai reittiä, joita funktiot voivat kulkea. (Ks Kuva)

AI ja testaus

AI:n toteutunut käyttö suhteessa suunnitelmaan

AI:n muu käyttö kuin yksikkötestauksen apuna

Claude:n ja Copilot:n tarjoamilta kielimalleilta kysyttiin apua GitHub Actions tiedoston kirjoittamisessa, silloin kun vastausta kysymyksiin tai ongelmiin ei löytynyt dokumentaatiosta tai muualta netistä. Kielimalleilta kysyttiin apua myös Coveralls GitHub yhteyden luomiseen. Työkalujen dokumentaatioista oli vaikea saada selville, mitä komentoja pitää kirjoittaa mihinkin, jotta halutut asiat tapahtuvat. Kielimallit osasivat kertoa selvemmin mitä mikäkin komento tai rivi tekee.

Tekoälyn luomat testit *ai_add.js* tiedostossa *hallusinoivat* siten, että tekoälyn luomat testit toimivat väärin. Esimerkiksi ” add › edge case -testit › toimii yhdellä argumentilla” testin tulisi palauttaa 5, mutta tekoälyn testi olettaa sen olevan NaN (not a number). Tämä johtuu siitä, että javascriptissä number tyypin oletus arvo on 0 ja 5 + 0 = 5.

Palaute kurssista & oppimisen reflektointi

Jalmari

Yksikkötestien tekeminen funktioille ja GitHub Actions rutiinin luominen tuntuivat ymmärrettäviltä ja suoraviivaisilta. Testaus suunnitelman ja raportin yhdistäminen tehtyyn (esimerkiksi omassa mielessä) testaukseen voisi olla ehkäpä helpompaa kokonaisella tai suuremmalla projektilla. Nyt esimerkiksi yksikkötestien teko oli kohtuullisen nopeaa ja helppoa, jolloin niihin nähtynä suurehkon suunnitelman ja dokumentaation tekeminen eivät kannustaneet tai innostaneet lukemaan ja noudattamaan sitä.

Työn määrä tuntui sopivalta ja yksikkötestien teko ja harjoittelu oli mieluisaa. GitHub Actions putken luomisesta ei tuntunut löytyvän hyvin ohjemateriaaleja netistä, joten oli vaikeaa tietää tai saada selville mitä kyseiseen tiedostoon pitäisi kirjoittaa. Coveralls yhteys herätti samanlaisia ajatuksia.

Toisaalta GitHub Actions tiedoston kirjoittaminen oli ainakin Jalmarille uutta ja mielekästä tutustuttavaa. Yksikkötestien kirjoittaminen oli ennestään tuttua, esimerkiksi ohjelmointi 3 kurssilta. Ennen kurssia odotin hieman, että oppimisin tekemään integraatio testausta ja käyttämään Robot Frameworksiä. Nämä eivät sisältyneet harjoitustyöhön mikä aiheutti pienen pettymyksen.

Antti

Testauksen perusteet ovat hyvin hallinnassa ja tuttua alan töistä. Kurssin aikana tutustuin muutamaan uuteen käsitteeseen, mutta kurssin ”toteutusvaiheessa” ei juurikaan mitään uutta ollut.

Coveralls ja npm yksikkötestien ”alustaminen” tai tarvittavien konfiguraatioiden lisääminen oli mielenkiintoista, sillä en sitä ollut aiemmin tehnyt.