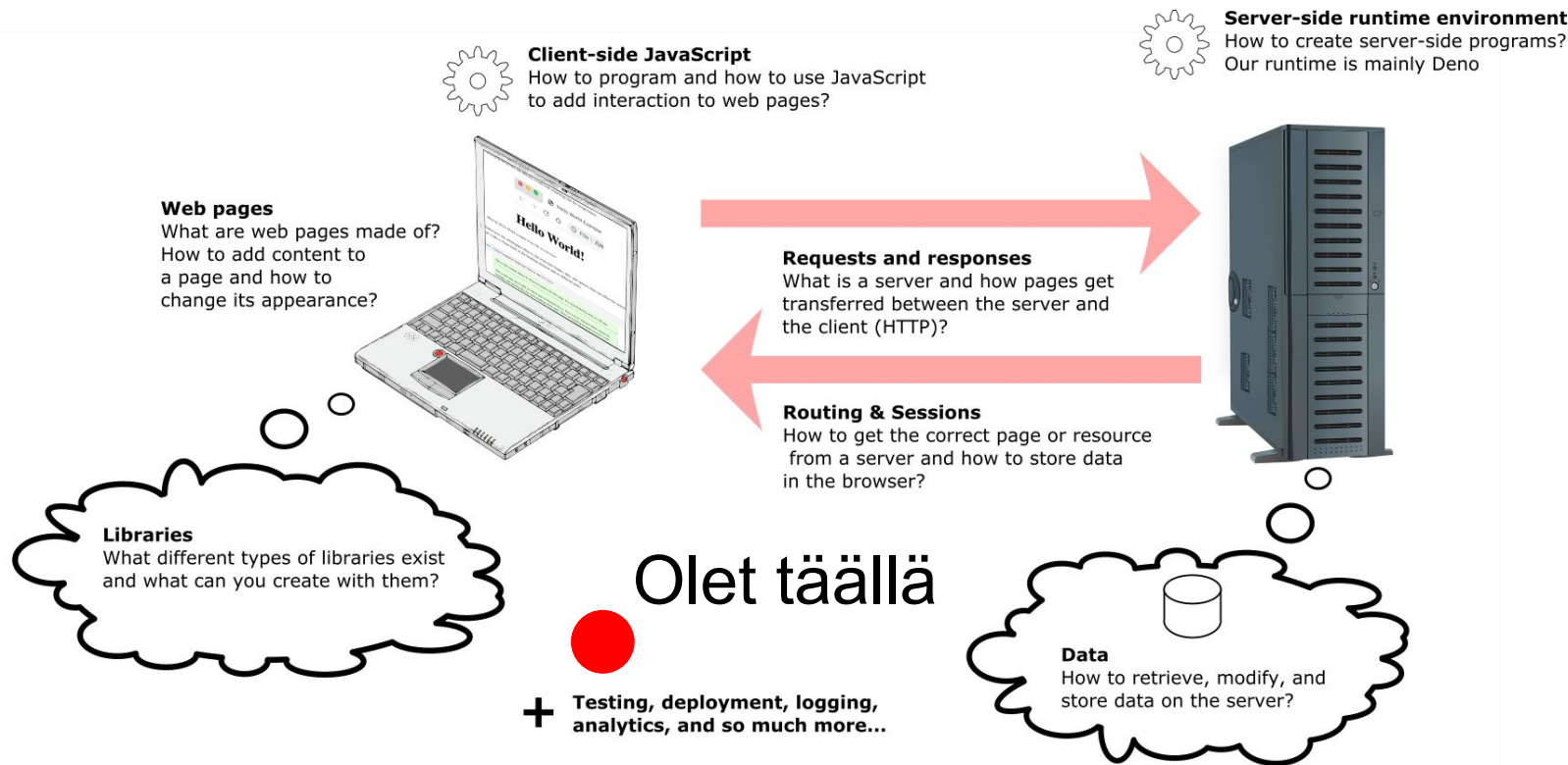


Päivä 7 - Sovellusten testaaminen, CI/CD

2021-12-03

AaltoPRO - Websovelluskehitys

Web-sovellukset korkealla tasolla



Päivä 7

- 9-12 Aamupäivä
 - Kertaus: Yksikkötestaus ja integraatiotestaus, TDD
 - Kahvitauko
 - Web-sovelluksen testaaminen: HTTP-rajapinnat
 - Yhteenveto
- 12:00 - 13:00 Lounas
- 13:00 - 16:00 Iltapäivä
 - Web-sovelluksen testaaminen: Selainautomaatio
 - Kahvitauko
 - Continuous Integration ja Continuous Delivery
 - Yhteenveto

Kertaus - project-06-temperatures

- Yksikkötesti ja toteutus

- Toteuta tiedostoon `arithmeticService` funktio `smaller`, joka saa parametrina kaksi lukua. Funktion tulee palauttaa annetuista luvuista pienempi. Tee funktiolle yksi testi.
- Toteuta tiedostoon `arithmeticService` funktio `greater`, joka saa parametrina kaksi lukua. Funktion tulee palauttaa annetuista luvuista suurempi. Tee funktiolle yksi testi.

- Integraatiotestejä

- Toteuta tiedostoon `temperatureService` parametrin funktio `hasMeasurements`. Funktion tulee palauttaa arvo `true` mikäli tietokannassa on mittauksia. Muulloin funktion tulee palauttaa arvo `false`. Tee funktiolle yksi testi.
- Toteuta tiedostoon `temperatureService` parametrin funktio `isFreezingOnAverage`. Funktion tulee palauttaa arvo `true` mikäli tietokannassa olevien lämpötilojen keskiarvo on pienempi kuin nolla. Muulloin funktion tulee palauttaa arvo `false`. Tee funktiolle yksi testi.

“Nice to know” - testikattavuus

- Testikattavuudella tarkastallaan automaattisissa testeissä läpikäytyjä vaihtoehtoisia suorituspolkuja. Mitä korkeampi testikattavuus, sitä suurempi osa vaihtoehtoisista suorituspoluista käydään läpi.
- Huom! Tarkastellaan läpikäytyjä suorituspolkuja, ei testien hyvyyttä tai sitä, tehdäänkö testeissä edes tarkastuksia.

“Nice to know” - testikattavuus

- **Kaksi komentoa**

- `deno test --coverage=cov` → suorittaa testit ja luo tilastoja kansioon `cov`
- `deno coverage cov` → luo kansiossa `cov` olevista tilastoista luettavan raportin

- **Voidaan ajaa myös docker-composen avulla**

- `docker-compose run --rm temperatures deno test --coverage=cov`
- `docker-compose run --rm temperatures deno coverage cov`


“Nice to know” - testikattavuus

- **Kaksi komentoa**

- `deno test --coverage=cov` → suorittaa testit ja luo tilastoja kansioon `cov`
- `deno coverage cov` → luo kansiossa `cov` olevista tilastoista luettavan raportin

- **Voidaan ajaa myös docker-composen avulla**

- `docker-compose run --rm temperatures deno test --coverage=cov`
- `docker-compose run --rm temperatures deno coverage cov`



Huom! Kansion cov-oikeudet eivät tule käyttäjälle (ainakin Linux&Mac)

“Nice to know” - testikattavuus


Kokeile testikattavuuden
selvittämistä nyt!

- Kaksi komentoa

- `deno test --coverage=cov` → suorittaa testit ja luo tilastoja kansioon `cov`
- `deno coverage cov` → luo kansiossa `cov` olevista tilastoista luettavan raportin

- Voidaan ajaa myös `docker-compose` avulla

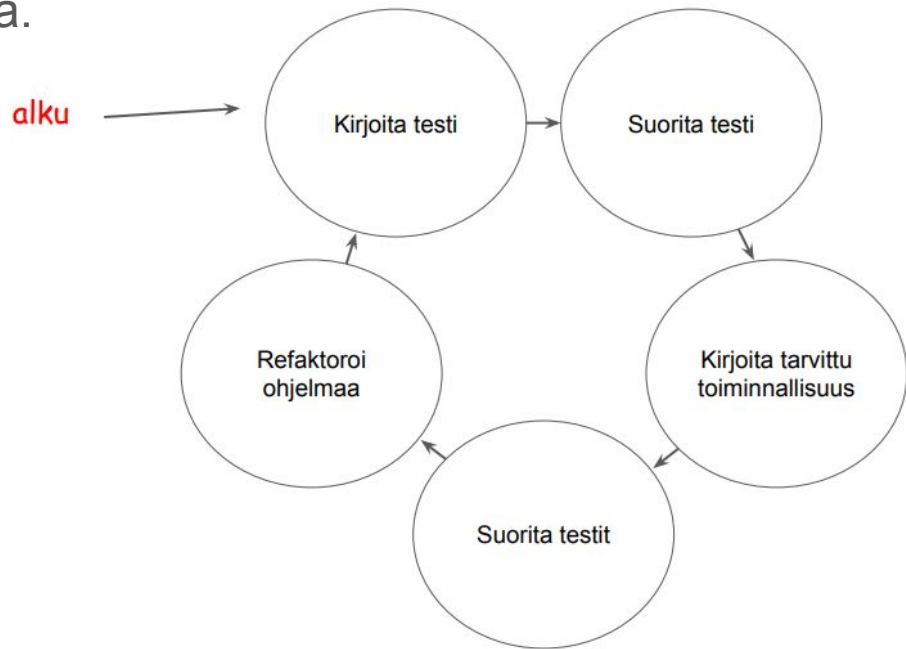
- `docker-compose run --rm temperatures deno test --coverage=cov`
- `docker-compose run --rm temperatures deno coverage cov`



Huom! Kansion `cov`-oikeudet eivät tule
käyttäjälle (ainakin Linux&Mac)

Hands-on: project-06-temperatures

- Test-driven Development menetelmää noudattaen, lisää funktioille `smaller`, `greater`, `hasMeasurements` ja `isFreezingOnAverage` **testejä** ja tarkastele miten testikattavuus kasvaa.



Kahvitauko?

Kertaus: polut ja metodit


- HTTP-pyyinnöt tehdään aina johonkin palvelimen polkuun
- HTTP-pyyinnöt tehdään aina jollain metodilla
 - GET-pyyntöä käytetään resurssin hakemiseen
 - POST-pyyntöä käytetään resurssin muokkaamiseen
- Vastauksessa aina statuskoodi (esim 200), usein myös runko

Web-sovelluksen testaus: HTTP-apit

- Web-sovellusta testattaessa tehdään pyyntöjä sovelluksen tarjoamiin polkuihin - tätä seuraa aina vastauksen tarkastelu
- Oak-sovelluskehyksellä tehtyjä sovelluksia testataan usein SuperOak-kirjastolla
- SuperOak tarjoaa toiminnallisuuden sovelluksen HTTP-apien testaamiseen

Web-sovelluksen testaus: HTTP-apit

- Web-sovellusta testattaessa tehdään pyyntöjä sovelluksen tarjoamiin polkuihin - tätä seuraa aina vastauksen tarkastelu
- Oak-sovelluskehyksellä tehtyjä sovelluksia testataan usein SuperOak-kirjastolla
- SuperOak tarjoaa toiminnallisuuden sovelluksen HTTP-apien testaamiseen




Voimme tehdä pyyntöjä
polkuihin ja tarkastella
pyyntöihin saatuja vastauksia

Reitit sovelluksessa project-06-temperatures (routes.js)

```
router.get("/", ({ response }) => response.redirect("/temperatures"));  
router.get("/temperatures", temperatureController.showTemperatures);  
router.post("/temperatures", temperatureController.addMeasurement);
```

Reitit sovelluksessa project-06-temperatures (routes.js)

*Voidaan testata tuleeko
vastauksena uudelleenohjaus
(statuskoodi 302)*



```
router.get("/", ({ response }) => response.redirect("/temperatures"));  
router.get("/temperatures", temperatureController.showTemperatures);  
router.post("/temperatures", temperatureController.addMeasurement);
```

Reitit sovelluksessa project-06-temperatures (routes.js)

*Voidaan testata tuleeko
vastauksena uudelleenohjaus
(statuskoodi 302)*

*Voidaan mm. testata sisältääkö
vastaus halutunkaltaisen
merkkijonon*

```
router.get("/", ({ response }) => response.redirect("/temperatures"));  
router.get("/temperatures", temperatureController.showTemperatures);  
router.post("/temperatures", temperatureController.addMeasurement);
```


Reitit sovelluksessa project-06-temperatures (routes.js)

*Voidaan testata tuleeko
vastauksena uudelleenohjaus
(statuskoodi 302)*

*Voidaan mm. testata sisältääkö
vastaus halutunkaltaisen
merkkijonon*

```
router.get("/", ({ response }) => response.redirect("/temperatures"));  
router.get("/temperatures", temperatureController.showTemperatures);  
router.post("/temperatures", temperatureController.addMeasurement);
```

*Voidaan mm. testata lisätäänkö
lähetetty data tietokantaan*

Esimerkki SuperOak-testistä

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";  
  
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Noudetaan sovellus
(app) ja superoak

Esimerkki SuperOak-testistä

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";  
  
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Esimerkki SuperOak-testistä

Noudetaan sovellus
(app) ja superoak

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";
```

Määritellään testi

```
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Esimerkki SuperOak-testistä

Noudetaan sovellus
(app) ja superoak

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";
```

Määritellään testi

```
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Luodaan sovellusta
käsittlevä
superoak-instanssi



Esimerkki SuperOak-testistä

Noudetaan sovellus
(app) ja superoak

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";
```

Määritellään testi

```
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Luodaan sovellusta
käsittelevä
superoak-instanssi

Tehdään GET-pyyntö
polkuun "/" ja odotetaan
että vastauksen
statuskoodi on 200

Esimerkki SuperOak-testistä

Noudetaan sovellus
(app) ja superoak

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";
```

Määritellään testi

```
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Luodaan sovellusta
käsittelevä
superoak-instanssi

Testataan ja korjataan!

Tehdään GET-pyyntö
polkuun "/" ja odotetaan
että vastauksen
statuskoodi on 200

Esimerkki SuperOak-testistä

Noudetaan sovellus
(app) ja superoak

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";
```

Määritellään testi

```
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(200);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Luodaan sovellusta
käsittelevä
superoak-instanssi

Testataan ja korjataan!

Tehdään GET-pyyntö
polkuun "/" ja odotetaan
että vastauksen
statuskoodi on 200

Esimerkki SuperOak-testistä

Noudetaan sovellus
(app) ja superoak

```
import { app } from "../app.js";  
import { superoak } from "../deps.js";
```

Määritellään testi

```
Deno.test({  
  name: "GET / palauttaa uudelleenohjauksen",  
  fn: async () => {  
    const testClient = await superoak(app);  
    await testClient.get("/").expect(302);  
  },  
  sanitizeOps: false,  
  sanitizeResources: false,  
});
```

Luodaan sovellusta
käsittlevä
superoak-instanssi

Testataan ja korjataan!

Tehdään GET-pyyntö
polkuun "/" ja odotetaan
että vastauksen
statuskoodi on 302

Testissä voidaan tehdä useampi tarkastelu

- SuperOak testit mahdollistavat `expect`-käskyjen ketjuttamisen
 - “vastauksen statuskoodin tulee olla 302 ja uudelleenohjauksen tulee tapahtua polkuun `/temperatures`”

```
await testClient.get("/")  
    .expect(302)  
    .expect("Location", "/temperatures");
```

- “vastauksen statuskoodin tulee olla 200 ja palautettavan sivun tulee sisältää teksti `Measurements`”

```
await testClient.get("/temperatures")  
    .expect(200)  
    .expect(/Measurements/);
```

Testissä voidaan tehdä useampi tarkastelu

- SuperOak testit mahdollistavat `expect`-käskyjen ketjuttamisen
 - “vastauksen statuskoodin tulee olla 302 ja uudelleenohjauksen tulee tapahtua polkuun `/temperatures`”

```
await testClient.get("/")  
    .expect(302)  
    .expect("Location", "/temperatures");
```

- “vastauksen statuskoodin tulee olla 200 ja palautettavan sivun tulee sisältää teksti `Measurements`”

```
await testClient.get("/temperatures")  
    .expect(200)  
    .expect(/Measurements/);
```

“Säännöllinen
lauseke”

Huom! Yksi pyyntö per superoak-instanssi

```
const testClient = await superoak(app);  
await testClient.get("/").expect(302);  
await testClient.get("/measurements").expect(200);
```



```
let testClient = await superoak(app);  
await testClient.get("/").expect(302);  
  
testClient = await superoak(app);  
await testClient.get("/measurements").expect(200);
```



Hands-on: project-06-temperatures

- Toteuta kolme superoak-testiä, joissa seuraavat tapaukset
 - GET-pyyntö juuripolkuun / palauttaa statuskoodin 302
 - GET-pyyntö polkuun `/temperatures` palauttaa statuskoodin 200
 - GET-pyyntö polkuun `/temperature` palauttaa statuskoodin 404

Tiedon lähettäminen

- Tiedon lähettäminen tapahtuu POST-pyynnöillä: lähetettävä data määritellään `send`-funktion sisälle. Esim.

```
await testClient.post("/temperatures")  
    .send("measurement=4")  
    .expect(302);
```

Tiedon lähettäminen

- Tiedon lähettäminen tapahtuu POST-pyyntöillä: lähetettävä data määritellään `send`-funktion sisälle. Esim.

```
await testClient.post("/temperatures")  
    .send("measurement=4")  
    .expect(302);
```

“Lähetetään palvelimelle
muuttuja `measurement`, jonka
arvona on 4”

Tiedon lähettäminen

- Tiedon lähettäminen tapahtuu POST-pyyntöillä: lähetettävä data määritellään `send`-funktion sisälle. Esim.

```
await testClient.post("/temperatures")  
    .send("measurement=4")  
    .expect(302);
```

"Lähetetään palvelimelle
muuttuja `measurement`, jonka
arvona on 4"

Tämä vastaa tilannetta, missä
lomakkeessa olisi tekstikenttä,
jonka nimi on `measurement`, ja jonka
arvoksi syötetään 4

Tiedon lähettäminen

- Tiedon lähettäminen tapahtuu POST-pyyntöillä: lähetettävä data määritellään `send`-funktion sisälle. Esim.

```
await testClient.post("/temperatures")  
    .send("measurement=4")  
    .expect(302);
```

"Lähetetään palvelimelle
muuttuja `measurement`, jonka
arvona on 4"

Odotetaan myös, että
vastauksen statuskoodi on 302
(eli uudelleenohjaus)

Tämä vastaa tilannetta, missä
lomakkeessa olisi tekstikenttä,
jonka nimi on `measurement`, ja jonka
arvoksi syötetään 4

Tiedon lähettäminen ja lähetetyn tiedon tarkastelu

- Testejä kirjoittaessa voi mukaan tuoda myös muita palveluita. Alla käytössä `temperatureService` (...on toki pitänyt tuoda käyttöön).

```
const temps = await temperatureService.listTemperatures();  
const testClient = await supertest(app);  
await testClient.post("/temperatures")  
    .send("measurement=4")  
    .expect(302);  
const tempsAfter = await temperatureService.listTemperatures();  
  
assertEquals(tempsAfter.length, temps.length + 1);
```

Tiedon lähettäminen ja lähetetyn tiedon tarkastelu

- Myös useamman pyynnön tekeminen on ok!

```
let testClient = await supertest(app);  
await testClient.post("/temperatures")  
    .send("measurement=4")  
    .expect(302);
```

```
testClient = await supertest(app);  
await testClient.get("/temperatures")  
    .expect(200)  
    .expect(/Measurements/);  
    .expect(/4/);
```

Hands-on: project-06-temperatures

- Toteuta viisi superoak-testiä, joissa seuraavat tapaukset
 - POST-pyyntö juuripolkuun / palauttaa statuskoodin 404
 - POST-pyyntö polkuun /temperatures siten, että pyynnössä on mittausarvo 5, palauttaa statuskoodin 302
 - POST-pyyntö polkuun /temperatures siten, että pyynnössä on mittausarvo -3, palauttaa statuskoodin 302, ja tarkastaa että tietokannassa olevien rivien määrä on kasvanut yhdellä
 - POST-pyyntö polkuun /temperatures siten, että pyynnössä on mittausarvo HUIJAAN, palauttaa statuskoodin 302, ja tarkastaa että tietokannassa olevien rivien määrä ei ole kasvanut yhdellä → muokkaa lopulta sovellusta siten, että tämäkin testi menee läpi
 - Ensin tyhjennetään tietokanta, jonka jälkeen tehdään POST-pyyntö polkuun /temperatures. Pyyntössä tulee olla mittausarvo 42. Tämän jälkeen tehdään GET-pyyntö polkuun /temperatures ja tarkastetaan, että palautetussa sivussa on arvo 42.

Lounas

Web-sovelluksen testaus: Selainautomaatio

- SuperOakilla ja HTTP-apeille yleisesti ottaen testejä kirjoittaessa tehdään pyyntöjä polkuihin, jonka jälkeen tarkastellaan palvelimelta saatua vastausta
- Web-sovelluksen sivujen testaaminen tällä tavalla ei kovin suoraviivaista
 - Mikäli sivulla on JavaScriptillä toteutettua dynaamista toiminnallisuutta, kattava sovelluksen testaaminen HTTP-apien kautta ei ole käytännössä mahdollista

Web-sovelluksen testaus: Selainautomaatio

- Selainautomaatiokirjastot kuten Cypress, Puppeteer ja Selenium mahdollistavat selaimen toiminnan automatisoinnin sekä sivuilla näkyvien asioiden tarkkailun
- Tyypilliset toiminnot: mene osoitteeseen, hae tekstiä, hae elementtiä, paina nappia, ...

Web-sovelluksen testaus: Selainautomaatio

- Tarkastellaan kahta esimerkkiä
 - Puppeteer (<https://pptr.dev/>)
 - Cypress (<https://www.cypress.io/>)

Esimerkki: project-07-todo-collections

- Projektissa kaksi docker-compose -tiedostoa:
 - `docker-compose.yml` -- käytetään sovelluksen käynnistämiseen (kuten normaalisti)
 - `docker-compose.e2e.yml` -- käytetään sovelluksen ja e2e-testien käynnistämiseen
- Tiedostoon `docker-compose.e2e.yml` määritelty mahdollisuus sekä Puppeteerin että Cypressin käynnistämiseen.
- Tietyn palvelun käynnistäminen annetusta docker-compose -tiedostosta:
 - `docker-compose -f docker-compose.e2e.yml up e2e-puppeteer`
 - `docker-compose -f docker-compose.e2e.yml up e2e-cypress`

Esimerkki: project-07-todo-collections

- Projektissa kaksi docker-compose -tiedostoa:
 - `docker-compose.yml` -- käytetään sovelluksen käynnistämiseen (kuten normaalisti)
 - `docker-compose.e2e.yml` -- käytetään sovelluksen ja e2e-testien käynnistämiseen
- Tiedostoon `docker-compose.e2e.yml` määritelty mahdollisuus sekä Puppeteerin että Cypressin käynnistämiseen.
- Tietyn palvelun käynnistäminen annetusta docker-compose -tiedostosta:
 - `docker-compose -f docker-compose.e2e.yml up e2e-puppeteer`
 - `docker-compose -f docker-compose.e2e.yml up e2e-cypress`

Toimivat kun sovellus on ensin
käynnistetty normaalisti komennolla
`docker-compose up --build`

Puppeteer (Denoon kanssa)

```
Deno.test("Visit a page, check that page has text", async () => {  
  const browser = await createBrowser();  
  const page = await browser.newPage();  
  await page.goto(`${address}`);  
  
  const content = await page.content();  
  assertStringIncludes(content, "Expected text");  
  await browser.close();  
});
```

Puppeteer (Denoon kanssa)

```
Deno.test("Visit a page, check that page has text", async () => {  
  const browser = await createBrowser();  
  const page = await browser.newPage();  
  await page.goto(`${address}`);  
  
  const content = await page.content();  
  assertStringIncludes(content, "Expected text");  
  await browser.close();  
});
```

Luo selain
ja "täbi"

Puppeteer (Denon kanssa)

```
Deno.test("Visit a page, check that page has text", async () => {  
  const browser = await createBrowser();  
  const page = await browser.newPage();  
  await page.goto(`${address}`);  
  
  const content = await page.content();  
  assertStringIncludes(content, "Expected text");  
  await browser.close();  
});
```

Luo selain
ja "täbi"

Mene
osoitteeseen

Puppeteer (Denon kanssa)

```
Deno.test("Visit a page, check that page has text", async () => {  
  const browser = await createBrowser();  
  const page = await browser.newPage();  
  await page.goto(`${address}`);  
  
  const content = await page.content();  
  assertStringIncludes(content, "Expected text");  
  await browser.close();  
});
```

Luo selain
ja "täbi"

Mene
osoitteeseen

Hae sisältö ja
tarkastele

Puppeteer (Denon kanssa)

```
Deno.test("Visit a page, check that page has text", async () => {  
  const browser = await createBrowser();  
  const page = await browser.newPage();  
  await page.goto(`${address}`);  
  
  const content = await page.content();  
  assertStringIncludes(content, "Expected text");  
  await browser.close();  
});
```

Luo selain
ja "täbi"

Mene
osoitteeseen

Hae sisältö ja
tarkastele

Sulje selain

Puppeteer

- Mahdollisuus siirtää kursori elementtiin ja kirjoittaa

```
await page.focus('input[type="text"]');  
await page.keyboard.type("My another task!");
```

- Mahdollisuus klikata elementtiä

```
await page.click('input[value="Add collection!"]');
```

- Ja niin edelleen.

Puppeteer

Hae input-elementti,
jonka type-attribuutin
arvo on "text"

- Mahdollisuus siirtää kursori elementtiin ja kirjoittaa

```
await page.focus('input[type="text"]');  
await page.keyboard.type("My another task!");
```

- Mahdollisuus klikata elementtiä

```
await page.click('input[value="Add collection!"]');
```

- Ja niin edelleen.

Puppeteer

*Hae input-elementti,
jonka type-attribuutin
arvo on "text"*

- Mahdollisuus siirtää kursori elementtiin ja kirjoittaa

```
await page.focus('input[type="text"]');  
await page.keyboard.type("My another task!");
```

- Mahdollisuus klikata elementtiä

```
await page.click('input[value="Add collection!"]');
```

- Ja niin edelleen.

*Mistä tällaisen loitsun
saa? Demo selaimessa!*

Cypress (ei Deno)

```
it("Can add a new collection", () => {  
  cy.visit("/");  
  
  cy.get("input[type='text']").type("A new collection with cypress.");  
  cy.get("form").submit();  
  
  cy.contains("A new collection with cypress.");  
});
```

Cypress (ei Denoa)

Selain ja "täbi"
luotu valmiiksi

```
it("Can add a new collection", () => {  
  cy.visit("/");  
  
  cy.get("input[type='text']").type("A new collection with cypress.");  
  cy.get("form").submit();  
  
  cy.contains("A new collection with cypress.");  
});
```

Cypress (ei Denoa)

Selain ja "täbi"
luotu valmiiksi

```
it("Can add a new collection", () => {
```

```
  cy.visit("/");
```

Mene sovelluksen
juuripolkuun

```
  cy.get("input[type='text']").type("A new collection with cypress.");
```

```
  cy.get("form").submit();
```

```
  cy.contains("A new collection with cypress.");
```

```
});
```

Cypress (ei Denoa)

Selain ja "täbi"
luotu valmiiksi

```
it("Can add a new collection", () => {
```

```
  cy.visit("/");
```

Mene sovelluksen
juuripolkuun

Etsi tekstikenttä ja
syötä siihen tekstiä

```
  cy.get("input[type='text']").type("A new collection with cypress.");
```

```
  cy.get("form").submit();
```

```
  cy.contains("A new collection with cypress.");
```

```
});
```

Cypress (ei Denoa)

Selain ja "täbi"
luotu valmiiksi

```
it("Can add a new collection", () => {
```

```
  cy.visit("/");
```

Mene sovelluksen
juuripolkuun

Etsi tekstikenttä ja
syötä siihen tekstiä

```
  cy.get("input[type='text']").type("A new collection with cypress.");
```

```
  cy.get("form").submit();
```

Etsi lomake ja lähetä se

```
  cy.contains("A new collection with cypress.");
```

```
});
```

Cypress (ei Denoa)

Selain ja "täbi"
luotu valmiiksi

```
it("Can add a new collection", () => {
```

```
  cy.visit("/");
```

Mene sovelluksen
juuripolkuun

Etsi tekstikenttä ja
syötä siihen tekstiä

```
  cy.get("input[type='text']").type("A new collection with cypress.");
```

```
  cy.get("form").submit();
```

Etsi lomake ja lähetä se

```
  cy.contains("A new collection with cypress.");
```

```
});
```

Tarkasta sivun sisältö

Puppeter vs. Cypress

- Kumpaakin käytetään selainautomatisaatioon
- Kummallakin oma tapa testien kirjoittamiseen
- Kummatkin hyviä

Hands-on: project-07-todo-collections

Toimivat kun sovellus on ensin
käynnistetty normaalisti komennolla
`docker-compose up --build`

- Suorita olemassaolevat cypress-testit komennolla:
`docker-compose -f docker-compose.e2e.yml up e2e-cypress`
- Mene kansioon `e2e-cypress` sisällä olevaan kansioon `cypress`; avaa kansio `videos`. Käynnistä joku kansiossa olevista videoista -- mitä näet?
- Luo kansioon `integration` uusi tiedosto `todos.spec.js`
 - Luo tiedostoon testi, joka (1) menee sovelluksen juuripolkuun, (2) luo uuden kokoelman, (3) klikkaa kokoelman linkkiä, (4) luo uuden todon, ja (5) tarkastaa onko todo sivulla
 - Luo tiedostoon testi, joka (1) menee sovelluksen juuripolkuun, (2) luo uuden kokoelman, (3) klikkaa kokoelman linkkiä, (4) luo uuden todon, (5) tarkastaa onko todo sivulla, (6) poistaa todon, ja (7) tarkastaa että todo ei ole sivulla

Kahvitauko?

“Se toimi omalla koneellani, mutta...”

- Ohjelmistoja kannattaa kehittää siten, että ohjelmistosta on jatkuvasti olemassa julkaisukunnossa oleva versio.

“Se toimi omalla koneellani, mutta...”

Version ei tarvitse
olla viimeisin!

- Ohjelmistoja kannattaa kehittää siten, että ohjelmistosta on jatkuvasti olemassa julkaisukunnossa oleva versio.

“Se toimi omalla koneellani, mutta...”

Version ei tarvitse
olla viimeisin!

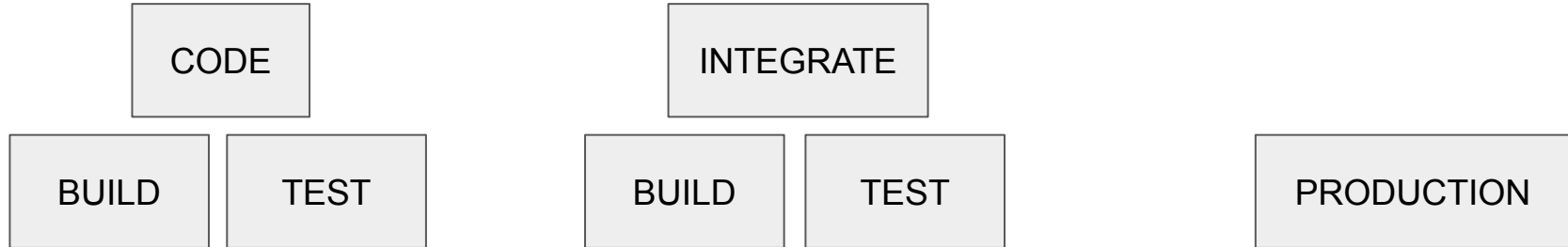
- Ohjelmistoja kannattaa kehittää siten, että ohjelmistosta on jatkuvasti olemassa julkaisukunnossa oleva versio.

Continuous Delivery?

- *Continuous Delivery on lähestymistapa ohjelmistokehitykseen, missä ohjelmistoja kehitetään siten, että ne voidaan julkaista tuotantoon milloin tahansa.*

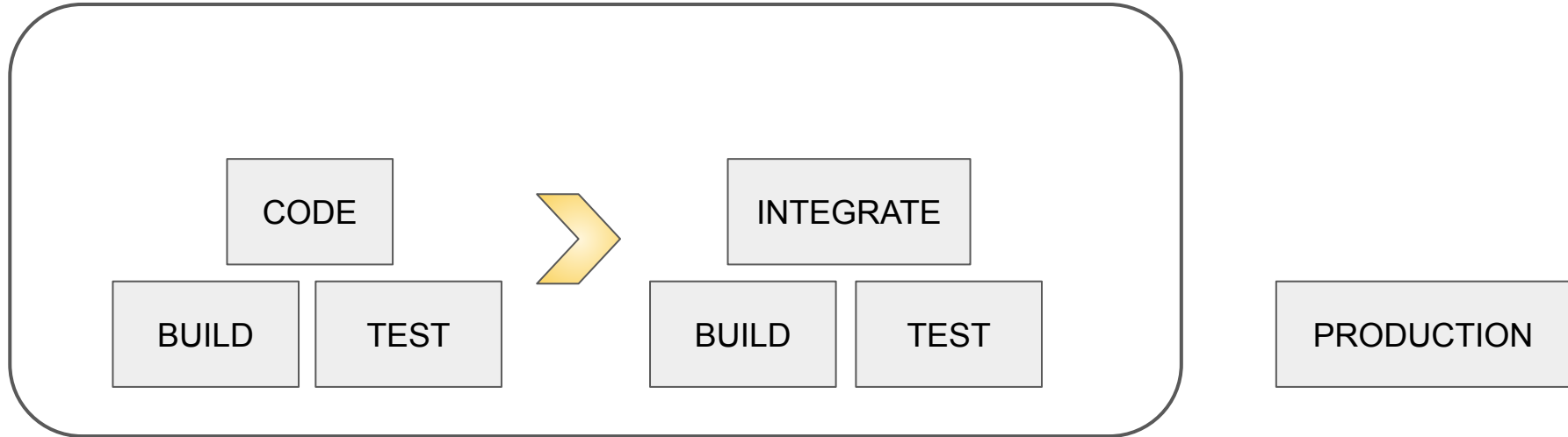
Continuous Integration, Continuous Delivery, ...

Continuous Integration, Continuous Delivery, ...



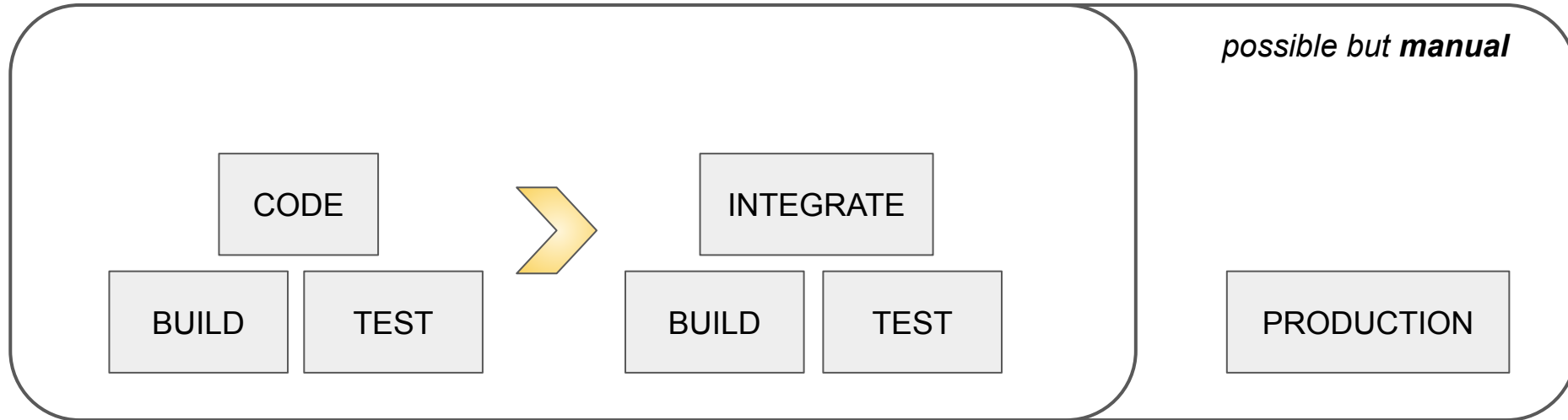
Continuous Integration, Continuous Delivery, ...

- Continuous Integration → *changes committed often, automated tests*



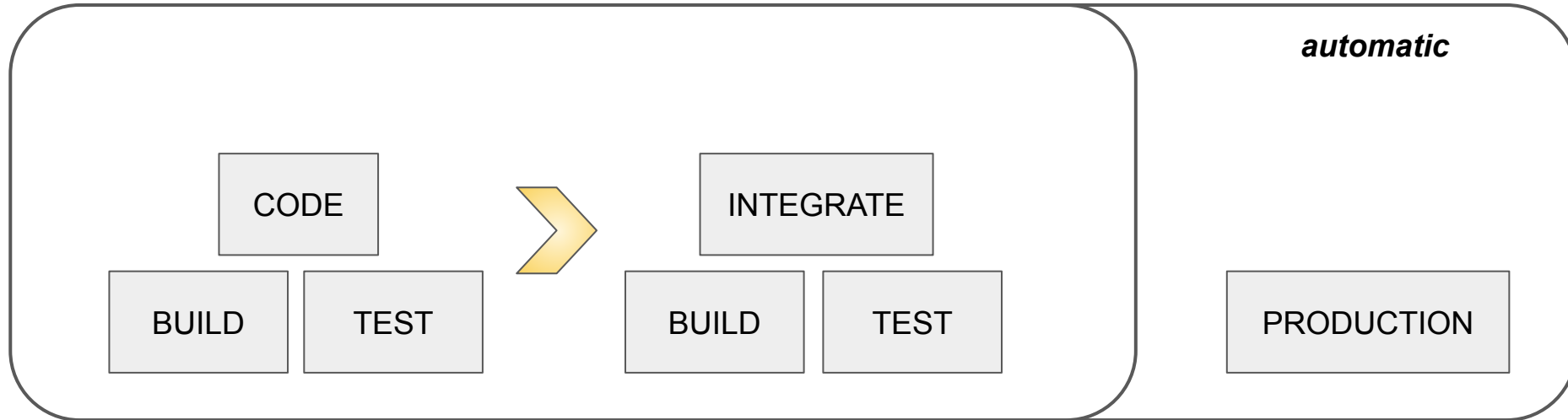
Continuous Integration, Continuous Delivery, ...

- Continuous Integration → *changes committed often, automated tests*
- Continuous Delivery → *software can be released to production at any time*



Continuous Integration, Continuous Delivery, ...

- Continuous Integration → *changes committed often, automated tests*
- Continuous Delivery → *software can be released to production at any time*
- Continuous Deployment → *latest version of the software is the live version*



Sovellus verkkoon: demo

<https://wsd.cs.aalto.fi/24-web-application-architecture-ii/3-serverless-architecture/>

Hands-on: vapaavalintaista harjoittelua

- Vaihtoehtoisesti:
 - Luo sovellus, joka näyttää käyttäjälle viestin “Hei maailma!” ja lisää se verkkoon Deno Deploy -palvelua käyttäen

Yhteenveto iltapäivästä

+ *seuraava lähipäivä lyhyesti*