Funktionaalinen ohjelmointi frontend-web-kehityksessä

TkK-tutkielma
Turun yliopisto
Tulevaisuuden teknologioiden laitos
Ohjelmistotekniikka
2020
Konsta Purtsi

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

Konsta Purtsi: Funktionaalinen ohjelmointi frontend-web-kehityksessä

TkK-tutkielma, 21 s.
Ohjelmistotekniikka
Kesäkuu 2020

Tarkempia ohjeita tiivistelmäsivun laadintaan läytyy opiskelijan yleisoppaasta, josta alla lyhyt katkelma.
Bibliografisten tietojen jälkeen kirjoitetaan varsinainen tiivistelmä. Sen on oletettava, että lukijalla on yleiset tiedot aiheesta. Tiivistelmän tulee olla ymmärrettävissä ilman tarvetta perehtyä koko tutkielmaan. Se on kirjoitettava täydellisinä virkkeinä, väliotsakeluettelona. On käytettävä vakiintuneita termejä. Viittauksia ja lainauksia tiivistelmään ei saa sisällyttää, eikä myäskään tietoja tai väitteitä, jotka eivät sisälly itse tutkimukseen. Tiivistelmän on oltava mahdollisimman ytimekäs n. 120–250 sanan pituinen itsenäinen kokonaisuus, joka mahtuu ykkäsvälillä kirjoitettuna vaivatta tiivistelmäsivulle. Tiivistelmässä tulisi ilmetä mm. tutkielman aihe tutkimuksen kohde, populaatio, alue ja tarkoitus käytetyt tutkimusmenetelmät (mikäli tutkimus on luonteeltaan teoreettinen ja tiettyyn kirjalliseen materiaaliin, on mainittava tärkeimmät lähdeteokset; mikäli on luonteeltaan empiirinen, on mainittava käytetyt metodit) keskeiset tutkimustulokset tulosten perusteella tehdyt päätelmät ja toimenpidesuositukset asiasanat

Asiasanat: tähän, lista, avainsanoista

Sisältö

1 Johdanto

Tietotekniikan käyttö arkielämässä on yleistynyt viime vuosina räjähdysmäisesti. Etenkin verkkosivujen ja -sovellusten käyttö on suositumpaa kuin koskaan. Kasvanut käyttö on lisännyt merkittäväst web-tekniikoille asetettuja vaatimuksia muun muassa luotettavuuden ja produktiivisuuden kannalta.

Funktionaalinen ohjelmointi on matemaattisten funktioiden evaluointiin perustuva ohjelmointiparadigma. Funktionaalinen ohjelmointi ei ole paradigmana uusi keksintö, mutta sen soveltaminen web-kehitykseen on nostanut päätään vasta viime vuosina. Tämän tutkielman tavoitteena on selvittää, miten funktionaalinen ohjelmointi soveltuu frontend-web-kehitykseen.

Tutkielma käsittelee funktionaalista ohjelmointia yleisellä tasolla ja lisäksi tutkii kahden modernin funktionaalisreaktiivisen frontend-tekniikan toteutusta funktionaalisuudelle. Tutkittaviksi teknologioiksi on valittu React-kirjasto ja Elm-ohjelmointikieli, jotka molemmat kääntyvät selaimen tulkittavaksi JavaScript-kieleksi.

Funktionaalista ohjelmoinnin konsepteja ja syntyä käsittelee tutkielman luku 2. Luvussa selvitetään myös hyötyjä funktionaalisen ohjelmoinnin käytöstä.

Funktionaalista ohjelmointia web-kehityksessä ja frontend-sovelluksen toteutuksessa etenkin funktionaalisen reaktiivisen ohjelmoinnin avulla käsittelevät luvut 3 ja 4.

2 Funktionaalinen ohjelmointi

2.1 Määritelmä

Funktionaalinen ohjelmointi on deklaratiivinen ohjelmointiparadigma, jossa ohjelman suoritus tapahtuu sisäkkäisten matemaattisten funktioiden evaluoimisena. Deklaratiivinen ohjelmointi on ohjelmointiparadigma, jossa kuvaillaan ohjelman lopputulos tai tila. Deklaratiivisessa ohjelmoinnissa ei ole implisiittistä globaalia tilaa, ja siten ohjelma koostuu lausekkeista, joiden välillä tilaa siirretään eksplisiittisesti. Se on vastakohta perinteisemmälle imperatiiviselle ohjelmoinnille. Funktionaalinen ohjelmakoodi pyrkii käsittelemään tilaa tarkemmin ja välttämään muuttuvia arvoja. [1]

Puhtaasti funktionaalinen ohjelmointikieli ei salli lainkaan muuttuvaa tilaa tai datan muokkaamista, eli niin kutsuttuja sivuvaikutuksia (engl. side effect). Tämä vaatii sen, että funktioiden paluuarvo riippuu ainoastaan funktion saamista parametreista, eikä mikään muu kuin parametrit voi vaikuttaa funktion paluuarvoon. Funktionaaliseksi ohjelmointikieleksi lasketaan kuitenkin myös moniparadigmaiset ohjelmointikielet, jotka sisältävät funktionaalisia piirteitä. Yksinkertaisten loogisten toimintojen lisäksi realistinen ohjelmointikieli vaatii epäpuhtauksia muun muassa siirrännän (engl. Input/Output) toteuttamiseen. [2] Moderneissa ohjelmointikielissä tähän on myös funktionaalisia ratkaisuja, esimerkiksi Haskell-kielen IO-monadi [3].

2.2 Lambdakalkyyli

Funktionaalinen ohjelmointi perustuu lambdakalkyyliin – formaalin laskennan malliin jonka kehitti Alonzo Church vuonna 1928. Lambdakalkyyliä pidetään ensimmäisenä funktionaalisena ohjelmointikielenä, vaikka se keksittiinkin ennen ensimmäisiä varsinaisia tietokoneita, eikä sitä pidettykään aikanaan ohjelmointikielenä. Useita moderneja ohjelmointikieliä, kuten Lispiä, pidetään abstraktioina lambdakalkyylin päälle. Lambdakalkyyli voidaan kirjoittaa myös λ-kalkyyli kreikkalaisella lambdakirjaimella. [1]

Churchin lambdakalkyyli koostuu vain kolmesta termistä: muuttujista, abstraktioista ja sovelluksista. Muuttujat ovat yksinkertaisesti merkkejä tai merkkijonoja jotka kuvaavat jotain parametriä tai arvoa. Churchin alkuperäinen lambdakalkyyli ei tuntenut muuttujien asettamista, ainoastaan arvojen syöttämisen parametrina. Abstraktio $\lambda x[M]$ määrittelee funktion parametrille x toteutuksella M. Sovellus $\{F\}(X)$ suorittaa funktion F arvolla X. [4]

2.3 Konseptit

2.3.1 Tilattomuus

Deklaratiivisissa ohjelmointikielissä ei ole implisiittistä tilaa, mikä eroaa imperatiivisista kielistä, joissa tilaa voi muokata lauseilla (engl. commands) ohjelmakoodissa. Tilattomuus tekee laskennasta mahdollista ainoastaa lausekkeilla (engl. expression). Funktionaaliset ohjelmointikielet käyttävät lambda-abstraktioita kaiken laskennan perustana. Esimerkiksi logiikkaohjelmoinnissa vastaava rakenne on relaatio. Tilattomuus sallii vain muuttumattomat vakiot, mikä estää muuttujien arvon muuttamisen ja perinteiset silmukat. Puhtaasti funktionaalisessa ohjelmoinnissa rekursio on ainoa tapa toteuttaa toisto. Esimerkiksi funktio, joka laskee luvun n kertoman voitaisiin

toteuttaa imperatiivisesti Python-kielessä esimerkiksi näin:

```
def factorial(n):
    result = 1
    while n >= 1:
        result = result * n
        n = n - 1
    return result
```

Funktionaalisessa ohjelmointikielessä toistorakenne on toteutettava rekursion avulla, mutta tästä on usein tehty vaivatonta muun muassa yksinkertaisen syntaksin ja tehooptimoinnin avulla. [1] Funktio, joka laskee luvun n kertoman voidaan toteuttaa esimerkiksi näin funktionaalisesti Haskell-kielessä:

```
factorial :: Int -> Int
factorial n = if n < 2
              then 1
              else n * factorial (n-1)
```

2.3.2 Korkeamman asteen funktiot

Funktionaaliset ohjelmointikielet helpottavat funktionaalista ohjelmointityyliä muun muassa sallimalla korkeamman asteen funktiot. Ensimmäisen luokan kansalainen on ohjelmointikielen rakenne, jonka voi syöttää parametrina funktioille, asettaa paluuarvoksi funktiolle ja tallentaa tietorakenteisiin. Korkeamman asteen funktio eroaa tavallisesta funktiosta siten, että se voi ottaa parametrinaan tai palauttaa funktion. Ohjelmointikielissä, jotka tukevat korkeamman asteen funktioita, funktiot ovat ensimmäisen luokan kansalaisia. Korkeamman asteen funktioiden avulla ohjelmakoodi ja data ovat jossain määrin vaihdettavissa, joten niiden avulla voidaan abstrahoida kompleksisia rakenteita. [1] Map-funktio on korkeamman asteen funktio, joka suorittaa parametrina annetun funktion jollekkin tietorakenteelle, esimerkiksi listalle.

Map-funktiota käytetään usein modernissa frontend web-ohjelmoinnissa [5]. Listan renderöinti React-kirjastolla voidaan toteuttaa näin map-funktion avulla:

```
items.map(item => <Item {...item} />)
```

2.3.3 Laiska laskenta

Laiska laskenta (engl. lazy evaluation) on funktion laskentastrategia, jossa lausekkeen arvo lasketaan vasta kun sitä ensimmäisen kerran tarvitaan, mutta ei aikaisemmin. Laiska laskenta on päinvastainen laskentastrategia innokkaaseen laskentaan (engl. eager evaluation), jossa lausekkeen arvo lasketaan heti ensimmäisellä kerralla, kun lauseke esitellään. Yleisimmät ohjelmointikielet käyttävät innokasta laskentastrategiaa. Laiska laskenta vähentää suoritukseen kuluvaa aikaa ja siten voi parantaa ohjelman suorituskykyä eksponentiaalisesti esimerkiksi call by name innokkaaseen laskentastrategiaan verrattuna, jossa lausekkeen arvo voidaan joutua laskemaan useita kertoja. Laiskan laskennan toteuttavat useat puhtaasti funktionaaliset ohjelmointikielet, kuten Haskell. Myös jotkin moniparadigmaiset kielet toteuttavat laiskan laskennan, esimerkiksi Scala-kieli mahdollistaa sen *lazy val*-lausekkeen avulla. [6]

Laiska laskenta mahdollistaa päättymättömät tietorakenteet. Niin sanottujen laiskojen listojen loppupää evaluoidaan vasta kun sitä kutsutaan. Tämä näkyy useissa funktionaalisissa ohjelmointikielissä listan toteutuksena, jossa listaa evaluoidaan alkupäästä loppua kohti. Esimerkiksi Haskellissa voidaan määrittää lista kaikista kokonaisluvuista tietotyypin rajoissa alkaen luvusta n:

```
from :: Int -> [Int]
from n = n : from (n+1)
```

Funktion rekursiivinen kutsu aiheuttaisi innokkaasti laskevissa ohjelmointikielissä päättymättömän rekursion, mutta Haskellin tapauksessa vain määrätty osa listasta evaluoidaan. [6]

Funktionaalisissa ohjelmointikielissä häntärekursio ei aiheuta pinon ylivuotoa, joten päättymätön rekursio on toimiva vaihtoehto päättymättömälle silmukalle. Esimerkiksi Haskell-kielen main-funktioita voidaan toistaa äärettömästi näin:

```
main :: IO ()
main =
  do
    putStrLn "do something"
    main
```

2.3.4 Hahmonsovitus

Hahmonsovitus (engl. pattern matching) on funktionaaliselle ohjelmoinnille tyypillinen piirre, jossa sama funktio voidaan määritellä useita kertoja. Funktiomäärittelyistä vain yhtä sovelletaan tapauskohtaisesti. Modernit funktionaaliset ohjelmointikielet, kuten Haskell, mahdollistavat monimutkaisten tyyppien ja lausekkeiden dekonstruoimisen hahmonsovituksen avulla [3]. Hahmonsovituksen toteutus on käytännössä case-lauseke, jossa lausekkeen ehto kuvaa syötetyn parametrin tietotyyppiä tai sen rakenneta. Tämän avulla ohjelman suoritus jakautuu syötetyn muuttujan tyypin tai rakenteen mukaan. [1]

Esimerkiksi kertomafunktio voidaan toteuttaa rekursiivisesti hahmonsovituksen avulla. Tässä esimerkissä rekursion alkuarvo määritetään syötteelle *0*, ja muut syötteet käsitellään parametrina *n*.

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

2.4 Funktionaalisen laskentamallin seuraukset

Puhtaasti funktionaalinen laskentamalli karsii ohjelmasta pois tietyt bugityypit. Puhdas sivuvaikutukseton laskenta on automaattisesti säieturvallista (engl. thread safe) tarkoittaen, ettei sitä tarvitse manuaalisesti synkronoida. Koska puhtaan funktion lopputulos riippuu ainoastaan syötteistä, ei syötteiden laskujärjestyksellä ole väliä. Laskennan lopputulosta voidaan siten pitää deterministisenä, mikä helpottaa monisäikeistystä.

Toistettavuus (engl. reproducibility) tarkoittaa, että ohjelman tietynlainen toiminta on toistettavissa. Deterministisyys ja toistettavuus parantavat ohjelman testattavuutta. Jos puhdas funktio toimii tietyllä tavalla kerran, se toimii niin myös joka kerta ajon aikana. Toistettavuus helpottaa ajon aikana löydetyn virheellisen tilan toistamista syöttämällä sama tarkkaan määritetty virheellinen tila testiympäristöön. [7]

Funktionaaliset ohjelmointikielet ovat olleet hitaampia kuin niiden perinteisemmät imperatiiviset sukulaiset, kuten C-kieli, mutta ero on ajan kuluessa kaventunut. Kääntäjäoptimoinneista huolimatta imperatiiviset ohjelmointikielt tulevat aina olemaan optimoidumpia imperatiivisella laitteistolla. Imperatiivisille tietorakenteille on nykyään olemassa monia funktionaalisia vaihtoehtoja, jotka voivat parantaa ohjelman suoritusaikaa ja säästää muistia. [8]

3 Web-kehitys

3.1 World Wide Web

World Wide Web tai WWW on järjestelmä tiedon jakamiseen verkkosivuina ja -sovelluksina. Se hyödyntää maailmanlaajuista Internet-tietoverkkoa. Web-sisältöä luetaan selaimella, joka hakee sisällön web-palvelimelta HTTP-siirtoprotokollan avulla. Verkkosivujen sisällön kuvaamiseen on perinteiseti ollut kolme tekniikkaa: HTML-kieli sivun sisällön kuvaamiseen, CSS-kieli sivun ulkoasun kuvaamiseen ja JavaScript-kieli sivun toiminnallisuuden toteuttamiseen. On myös mahdollista käyttää esimerkiksi Flash- ja Java-liitännäisiä, mutta nykyään on tyypillisempää käyttää pelkästään JavaScript-kieltä. [9]

3.1.1 Staattiset verkkosivut

Alkuperäinen tapa jakaa web-sisältöä on staattisena verkkosivuna (engl. static website). Staattinen verkkosivu koostuu HTML-sivuista, dokumenteista ja mediasta, jotka luetaan suoraan web-palvelimen massamuistista. Palvelin lähettää tiedostot selaimelle tekemättä niihin muutoksia ajon aikana. Staattiset verkkosivut sisältävät usein HTML-, CSS- ja multimediasisältöä, sekä selaimessa ajettavia komentosarjoja (engl. script) tai ohjelmia, jotka on kirjoitettu JavaScript-kielellä tai muulla selaimen tulkitsemalla kielellä. [10]

3.1.2 Dynaamiset verkkosivut

Dynaaminen verkkosivu tarkoittaa sivua, joka ei ole suoraan selaimen luettavassa muodossa. Tällaisen sivun backend muodostaa selaimella renderöityvän sivun ajon aikana. Dynaamisen verkkosivun operaatioiden toteutus ei olekaan rajattu selaimen tulkittaviin ohjelmointikieliin, vaan käytännössä minkä tahansa kielen käyttäminen on mahdollista. Yleisimpiä kieliä dynaamisen verkkosivun toteuttamiseen ovat muun muassa Python, Java, C# ja Node.js. Dynaaminen verkkosivu tallettaa useimmiten sisältönsä tietokantaan. [10]

3.2 Backend

Backend tarkoittaa verkkosivun tai -sovelluksen palvelinpäässä ajettavaa osaa, joka ei näy käyttäjälle. Koska backend-ohjelmaa ajetaan palvelinlaitteistolla, sen toteuttamiseen on paljon enemmän vapautta kuin frontendin, jonka täytyy olla selaimen tulkittavissa. Backend vastaa usein kaikesta varsinaisesta toiminnallisuudesta verrattuna frontendiin, joka toteutaa pelkän käyttöliittymän ohjelmalle. Backendiin kuuluu usein myös tietokanta, johon sovelluksen data tallennetaan. [11]

Backend kommunikoi frontendin kanssa jonkin yhteisen rajapinnan (engl. Application programming interface, API) kautta. Rajapinta on yleensä pääasiallinen tiedonsiirtokanava backendin ja frontendin välillä. Frontend lähettää rajapintaan kutsun useimmiten HTTP-protokollan avulla, jonka jälkeen palvelin vastaanottaa kutsun ja suorittaa kutsutun toimenpiteen. Rajapintaa suunniteltaessa tulee muotoilla rajapinnan rakenne vastaamaan sekä ohjelman datarakennetta, että käyttäjän tai frontendin vaatimuksia. Rajapinta mahdollistaa staattiselle frontendille dynaamisen sisällön hakemisen ja muokkaamisen ajonaikana. Rajapinta voi myös mahdollistaa frontendille käyttäjän tunnistautumisen (engl. authentication) ja valtuuttamisen (engl. authorization). [11]

3.3 Frontend

Frontend tarkoittaa verkkosivun tai -sovelluksen käyttäjälle näkyvää osaa. Koska frontendiä suoritetaan selaimessa, se on pakko toteuttaa web-yhteensopivilla tekniikoilla. Yleisimpiä tekniikoita ovat HTML, CSS ja JavaScript. JavaScript kehitettiin alunperin hyvin yksinkertaisiin käyttöliittymä toimintoihin, kuten animaatoiden toteuttamiseen. Kuitenkin nykypäivänä se on täysi ohjelmointikieli, jolla voidaan toteuttaa interaktiivisuutta sekä manipuloida verkkosovelluksen rakennetta ja ulkoasua. Dokumenttioliomalli tai DOM (engl. Document Object Model) kuvaa verkkosivun puurakenteena, jonka pohjalta sivu piirretään (engl. rendering). [11]

Frontendin keskeisin tehtävä on käyttöliittymän ja tiedon näyttäminen käyttäjälle. Tästä huolimatta modernissa front–endissä on paljon muitakin asioita joita tulee ottaa huomioon. Frontend-ohjelmakoodi muodostaa ja muokkaa verkkosivun HTML DOM-puuuta (engl. DOM tree), joka kuvaa sivun rakenteen. Frontend mahdollistaa myös käyttäjält tulevien DOM-tapahtumien (engl. DOM event) vastaanottamisen ja käsittelyn. Modernilta frontendiltä odotetaan myös responsiivista designia, joka tarkoittaa että sama sivu toimii usealla erikokoisella päätelaitteella. [12]

Modernissa frontend–kehityksessä on tyypillistä toteuttaa verkkosovelluksen ohjelmakoodi jollain korkeamman tason ohjelmointikiellä, joka muunnetaan web-natiiviksi käännösvaiheessa. Usein käytetään esimerkiksi CSS-tyylikieleksi kääntyvää SCSS-kieltä. Kääntäjiä, jotka kääntävät frontend–kieliä selainyhteensopiviksi kutsutaan käännöstyökaluiksi (engl. build tool). Esimerkiksi tämän tutkielman käsittelykohteena oleva Elm-kieli ei ole sellaisenaan selainyhteensopiva, vaan vaatii kääntämisen HTML- ja JavaScript-kielille [13]. [11]

3.4 Funktionaalinen web-ohjelmointi

Funktionaalinen web-ohjelmointi on kerännyt suosiota viime vuosina. Funktionaalinen ohjelmointi soveltuu erinomaisesti web-ohjelmointiin sen toistettavuuden ja testattavuuden ansiosta. Eri abstraktiotasojen selkeämpi erottelu on myös perustelu funktionaaliselle web-ohjelmoinnille. Backend-sovelluksen ohjelmointikieleksi voikin valita esimerkiksi Clojure- tai Elixir-kielen. [14] Funktionaalinen ohjelmointi on yleistä etenkin frontend kehityksessä funktionaalisia piirteitä suosivien tekniikoiden, kuten Reactin, ansiosta. Funktionaalinen frontend–ohjelmointi mahdollistaa tyypillisiä funktionaalisen ohjelmoinnin hyötyjä, kuten parantaa ohjelman suorituskykyä ja testattavuutta. [5] Koska tutkielman aihe on funktionaalinen frontend-web-kehitys, käsitellään seuraavaksi frontendia.

Funktionaalista paradigmaa toteuttava frontend perustuu datan ja näkymän yhdistämiseen puhtaiden funktioiden avulla. Tämänkaltainen frontend vaatii jonkin eksplisiittisen mekanismin datan muuttamiseen ja näkymän piirtämiseen uudelleen. Funktionaaliset frontend-tekniikat suosivatkin usein muuttumatonta (engl. immutable) tilanhallintaa, joka vaatii tilan uudelleenasettamisen sen muuttamiseksi. Tämänkaltaiselle tilanhallinnalle on myös tyypillistä säilöä koko ohjelman tila yhdessä paikassa, ja muuttaa sitä lähettämällä sille käskyjä (engl. dispatching actions). Käskyn vastaanottaa puhdas funktio, joka ottaa parametreina käskyn ja aiemman tilan, palauttaen ohjelman uuden tilan. Näin ohjelma on näennäisesti puhtaan funktionaalinen ja sivuvaikutukseton. [5]

Kuten tosielämän ohjelmille on tyypillistä, frontendissa tarvitaan myös sivuvaikutuksia joidenkin toiminnallisuuksien toteuttamiseen. Sivuvaikutukset ovat hyödyllisiä esimerkiksi tiedon tallentamiseen selaimen muistiin ja satunnaisten numeroiden tuottamiseen. Funktionaaliset frontend-ohjelmistokehykset mahdollistavatkin tyypillisesti toiminnallisuuden sivuvaikutusten toteuttamiseen funktionaaliselle ohjelmoinnille tyypillisen rajatusti. [13][15]

3.5 Funktionaalinen reaktiivinen ohjelmointi

Reaktiivinen ohjelmointi tarkoittaa asynkronisen datan ja tapahtumien kytkemistä muuttumattomiin datatyyppeihin ohjelmointikielen tai ohjelmistokehyksen sisäänrakennetuilla tekniikoilla. Muuttumattomien datavirtojen käsittely voidaan toteuttaa puhtaasti funktionaalisilla operaatioilla. Tämä on funktionaalisen reaktiivisen ohjelmoinnin (engl. Functional Reactive Programming, FRP) keskeinen konsepti, jota esimerkiksi React ja Elm hyödyntävät [13][15]. [16]

Funktionaalinen reaktiivinen ohjelmointii mahdollistaa perinteisten puhtaiden funktioiden (esim. map, filter, reduce) käyttämisen sovelluksen kuvaamiseen. Keskeinen idea on, että ohjelman ympäristöstä tuleva data sidotaan näennäisesti muuttumattomiin signaaleihin, joita voidaan käsitellä puhtaasti funktionaalisesti. Kun data muuttuu, suoritetaan laskenta uudestaan uudella arvolla. Toteutus on tavallaan muuttumaton, vaikkakin data on muuttuvaa. Reaktiiviseen ohjelmointimalliin soveltuvia tapahtumia ja datalähteitä frontend-web-sovelluksessa ovat muun muassa DOM-tapahtumat ja rajapintakutsut. [13]

4 Funktionaaliset frontend-ohjelmistokehykset

Tämän tutkielman tutkimuskohteiksi on valittu funktionaaliset reaktiiviset ohjelmistokehykset React-kirjasto ja Elm-ohjelmointikieli. Tutkielma käsittelee React-kirjastoa ja Elm-kieltä yleisesti funtionaalisen ohjelmoinnin kannalta, sekä tarkemmin sitä, miten teknologiat toteuttavat tilankäsittelyn ja mahdollistavatko ne sivuvaikutukset.

4.1 Ohjelmistokehykset

React on Facebookin ylläpitämä avoimen lähdekoodin käyttöliittymäkirjasto, jota käytetään tyypillisesti JavaScript-kielen kanssa. React on alunperin julkaistu vuonna 2013, ja se on kirjoitushetkellä vuonna 2020 maailman suosituin frontend-ohjelmistokehys. Sillä on kirjoitushetkellä noin yhdeksän miljoonaa viikoittaista latausta NPM-palvelusta [17]. React-kirjaston syntaksi on deklaratiivista, ja se mahdollistaa käyttöliittymän pilkkomisen komponentteihin. React ei ota kantaa muihin tekniikkaratkaisuihin, vaan keskittyy pelkästään yksittäisen DOM-puun hallitsemiseen. Ohjelmistokehittäjälle jää näin täysi valta valita muut teknologiat vapaasti. [15]

Elm on funktionaalinen ohjelmointikieli, joka käännetään JavaScript-kieleksi. Elmin on alunperin kehittänyt Evan Czaplicki osana maisterin tutkielmaansa vuonna

2012. Elm ei ole tutkielman kirjoitushetkellä saavuttanut suurta suosiota, vaan sen NPM-paketinhallinnan latausmäärä on vain noin kolmekymmentä tuhatta latausta viikossa [17]. Elm keskittyy web-pohjaisten graafisten käyttöliittymien deklaratiiviseen toteuttamiseen. Elm kytkeytyy tavallisen HTML DOM-puun solmuun. Elm ei myöskään rajoita muita tekniikkaratkaisuja, kuten tyylikielen valintaa, mutta Elm-koodissa on haastavampaa käyttää muilla kuin Elm-kielellä kirjoitettuja komponentteja ja kirjastoja. [13]

4.2 Funktionaalinen ohjelmointi

React korostaa funktionaalista ohjelmointityyliä imperatiivisen ohjelmoinnin sijaan, mutta sallii kuitenkin kaiken tavallisen imperatiivisen JavaScript-syntaksin. React ei pystykään välttämään muuttuvia arvoja tai sivuvaikutuksia mutta rohkaisee silti ohjelmoimaan funktionaalisesti muun muassa tarjoamalla tilanhallinan, joka soveltuu funktionaaliseen ohjelmointiin. [15]

Elm taas pyrkii olemaan puhtaasti funktionaalinen ohjelmointikieli eikä salli lainkaan esimerkiksi muuttuvia arvoja tai epäpuhtaita funktioita. Elm ei myöskään salli tilanteita, jotka voisivat aiheuttaa ajonaikaisia virheitä, vaan pakottaa virheiden käsittelyn funktionaalisesti datana. Elm-kielen ajonaikanen ohjelmistokehys abstrahoi pintapuolisesti kaikki epäpuhtaudet sisällensä, joten Elm-kielellä toteutettu ohjelma on näennäisesti puhtaan funktionaalinen. [13]

4.3 Funktionaalinen tilankäsittely

React tukee lokaalia tilaa missä tahansa komponentissa, joka on ristiriidassa funktionaalisen paradigman kanssa. Tila React-kirjastossa on kuitenkin muuttumaton datarakenne, jonka muuttaminen vaatii sen korvaamisen uudella datalla. Tilan muokkaamiseen on perinteisesti käytetty React-kirjaston Component-luokan setState-

metodia, joka ylikirjoittaa tilan. Seuraavassa on esimerkki React-kirjaston setState-funktion käytöstä:

```
// väärin
this.state.greeting = 'Moikku';
// oikein
this.setState({ greeting: 'Moikku' });
```

React-kirjasto tukee nykyään vakiona myös epäpuhtauksia funktioiksi abstraktoivia koukkuja (engl. hooks), suomenkielinen termi ei ole vakiintunut, mutta mielestäni ymmärrettävä. Koukut mahdollistavat esimerkiksi tilankäsittelyn puhtaassa funktionaalisessa komponentissa, ilman että tarvitaan luokkaa. Esimerkki React-kirjaston useState-koukun käytöstä:

```
const Example = () => {
  // greeting on tila, joka muuttuu reaktiivisesti
  // setGreeting on funktio, jonka kutsuminen muuttaa tilan
  const [greeting, setGreeting] = useState('Moikku');
  return (
    <h1>{greeting}</h1>
  );
}
```

Funktionaalisen React-sovelluksen toteutukseen on tyypillistä käyttää funktionaaliselle ohjelmalle tyypillistä tilamallia, jossa koko sovelluksen tilaa kuvataan yhdessä paikassa. Tämänkaltainen tilanhallinta yksinkertaistaa ymmärtämään tilan muutosten vaikutusta sovellukseen. Tyypillinen tapa toteuttaa keskitetty tilanhallinta funktionaalisesti on käyttää reducer-funktiota, joka palauttaa uuden tilan aiemman tilan ja tapahtuman perusteella. Modernissa React-kirjastossa tämänkaltaista tilanhallintaa voidaan käyttää useReducer-koukun avulla. Vastaavan

toteutuksen tarjoaa myös kolmannen osapuolien toteutuksista suosituimpiin kuuluva Redux-kirjasto [14]. Yksinkertaisiin sovelluksiin on silti riittävää käyttää lyhyemmän syntaksin natiiveja vaihtoehtoja, kuten perinteinen setState-toteutus ja useState-koukku. [15]

Elm-kielen arkkitehtuurissa tila on Model-nimisessä abstraktoidussa datamallissa. Model-tilan kuvaaminen ja päivittäminen on hyvin samankaltaista, kuin React-kirjaston Redux-tilanhallinta. Reduxin kehitys on saanutkin inspiraation Elm-kielen Model-tilanhallinnasta. Model-tilan muokkaaminen tapahtuu kutsumalla update-funktiota message-parametrilla samankaltaisesti kuin Redux-kirjaston reducer-funktiossa. [13] Esimerkki Elmin tilan toteutuksesta:

```
-- MODEL
type alias Model = Int


init : Model
init =
  0


-- UPDATE
type MSG = Increment | Decrement


update : Msg -> Model -> Model
update msg model =
  case msg of
    Increment ->
      model + 1
    Decrement ->
      model - 1
```

Sekä React-kirjaston että Elm-kielen tilanhallinta on hyvin tyypillistä funktionaalista ohjelmointia. Kumpikaan ohjelmistokehys ei suosittele muuttuvaa tilaa, vaan tilaa säilötään muuttumattomassa tietorakenteessa. React-kirjastossa yksittäisellä komponentilla voi olla lokaali tila. Elm-kielessä kaikki tila on oltava yhdessä Model-rakenteessa, kuten Reactissa on myös usein tyypillistä toteuttaa reducer-funktion avulla.

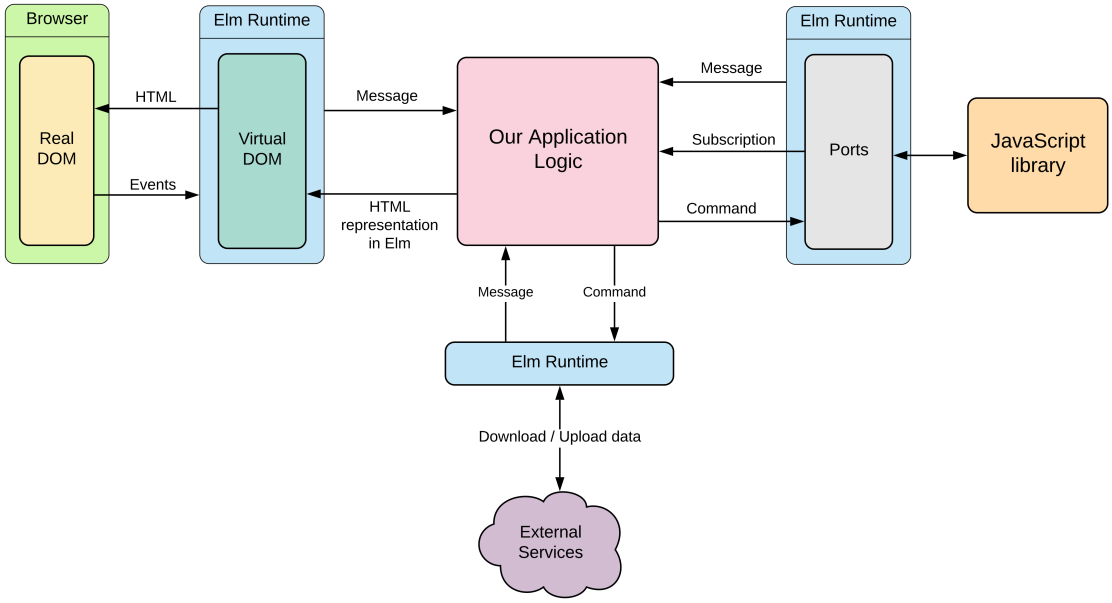
4.4 Sivuvaikutukset

Sivuvaikutukset ovat käytännössä pakollisia tosimaailman frontend-sovellukselle. Tyypillisiä sivuvaikutuksia frontend-ohjelmassa ovat muun muassa ulkoisen datan hakeminen, tapahtumankuuntelijan asettaminen ja manuaalinen DOM-puun muokkaaminen.

React sallii kaiken perinteisen JavaScript-syntaksin, kuten muuttujat ja sivuvaikutukselliset funktiot. Funktionaalinen ohjelmointimalli vaatiikin React-ohjelmoijalta omaa tahtoa kirjoittaa ohjelma funktionaalisesti. React kuitenkin suosii monella tavalla funktionaalista ohjelmointimallia. Moderni React suosii komponenttien toteuttamista niin kutsuttuina funktiokomponentteina, joiden oma tila ja sivuvaikutukset on toteutettava funktionaalisilla ohjelmointitavoilla. Funktiokomponentissa sivuvaikutuksen toteuttaminen vaatii useEffect-koukun käyttämistä, joka tekee sivuvaikutuksen käsittelemisestä näennäisesti funktionaalista. [15]

```
const Example = ({ cardId }) => {
  const [card, setCard] = useState(null);
  // aiheuttaa sivuvaikutuksen, joka hakee uuden datan
  // rajapinnasta kun cardId-parametri muuttuu.
  useEffect(() => {
    const newCard = api.getCardById(cardId);
```

```
    setCard(newCard);
  }, [cardId]);
  return (
    <Card card={card} />
  );
}
```



Elm-kieli ei salli epäpuhtauksia itse ohjelmakoodissa, vaan ne on kaikki toteutettava Elmin ajonaikaohjelmassa (engl. Elm runtime). Näin itse ohjelma on puhtaasti funktionaalinen, mutta sivuvaikutuksellisia ohjelmia on silti mahdollista kirjoittaa. Todellisuudessa Elm-kielen innokas laskentamalli aiheuttaa myös sivuvaikutuksia. [13]

4.5 Vertailu

Elm on suunniteltu mahdollisimman puhtaan funktionaaliseksi kieleksi. Elmin arvot ovatkin kaikki immutatiivisa, eikä Elm salli esimerkiksi poikkeuksia, vaan pakottaa virhetilan käsittelyn arvona. React-sovelluksen ohjelmakoodi on deklaratiivista ja reaktiivista, mutta sisältää usein epäpuhtauksia, kuten lokaalia tilaa. Reactia voi

kuitenkin ohjelmoida funktionaalisesti sisäänrakennettujen koukkujen avulla, mutta se ei vaadi käyttäjältä funktionaalista lähestymistapaa.

5 Loppupäätelmät

Funktionaalinen ohjelmointi on kasvattanut suosiotaan etenkin frontend-ohjelmoinnissa viime vuosina. Funktionaalisesta ohjelmoinnista onkin paljon hyötyjä verrattuna perinteiseen imperatiiviseen ohjelmointiin. Puhtaan funktionaalisen ohjelman tilattomuus tekee ohjelmasta helpommin järkeiltävää ja testattavampaa. Puhdas funktio on myös automaattisesti säieturvallinen. Funktionaaliselle ohjelmointityylille on myös tyypillistä koodin kompaktiutta ja luettavuutta parantavien abstraktioiden, kuten hahmonsovitus, käyttö.

Funktionaalinen paradigma soveltuu erinomaisesti web-kehitykseen. Backend-ohjelmoinnissa suosittuja kieliä ovat muun muassa Clojure- ja Elixir-kielet. Frontend-ohrjelmoinnissa funktionaalinen reaktiivinen ohjelmointi on hyvin tyypillinen toteutustapa verkkosovellukselle muun muassa React-kirjaston myötä.

Tässä tutkielmassa esitellyt teknologiat, React-kirjasto ja Elm-ohjelmointikieli ovat molemmat selaimen tulkittavaksi JavaScript-koodiksi käännettäviä ohjelmistokehyksiä. Molemmat tekniikat toteuttavat funktionaalista ohjelmointiparadigmaa, joskin Elm ei salli lainkaan muita paradigmoja. Näistä kahdesta etenkin React on saavuttanut valtavan suosion, ja onkin kirjoitushetkellä maailman suosituin frontend ohjelmistokehys. Sekä React että Elm mahdollistavat funktionaalisen ohjelmoinnin piirteiden, kuten sivuvaikutusten ja funktionaalisen tilanhallinnan toteuttamisen.

Etenkin funktionaalinen reaktiivinen ohjelmointi on hyvin luonteva tapa toteuttaa frontend-verkkosovellus. Tapahtumavirta on luonteva tapa kuvata frontendin

tapahtumienkäsittelyä, ja se kytkeytyy mukavasti puhtaasti funktionaalisiin tapahtumankäsittelijöihin. Valitettavasti tässä tutkielmassa käsiteltiin pääasiassa frontendin toteutusta yleisesti funktionaalisella ohjelmoinnilla, ja reaktiivisen funktionaalisen ohjelmoinnin tutkimus jäi hyvin pintapuoliseksi. FRP-ohjelmoinnin käyttäminen frontend-sovelluksen toteuttamiseen olisi luonteva jatko tälle tutkielmalle.

Tämän tutkielman tutkimuskohteet on myös rajattu hyvin niukasti Reactin ja Elmin tiettyihin yksityiskohtiin. Molemmat tekniikoista ovat funktionaalisen ohjelmoinnin kannalta erittäin käyttökelpoisia ja mielenkiintoisia, vaikka ne tarjoavatkin erilaisia lähestymistapoja toteutuksiin. Molemmissa teknologioissa olisi kuitenkin vielä paljon tutkittavaa. Frontend-verkkosovelluksen toteutukseen on myös useita muita mielenkiintoisia tekniikkavaihtoehtoja, muun muassa puhtaasti reaktiivisfunktionaaliset RxJS- ja Bacon.js-kirjastot.

Lähdeluettelo

[1] P. Hudak, "Conception, Evolution, and Application of Functional Programming Languages," 1989.

[2] A. Sabry, "What is a purely functional language?" *Journal of Functional Programming*, 1998.

[3] M. Lipovaca, "Learn You a Haskell for Great Good!: A Beginner's Guide," 2011.

[4] F. Cardone ja J. R. Hindley, "History of Lambda-calculus and Combinatory Logic," *Handbook of the History of Logic, vol. 5*, 2006.

[5] A. Banks ja E. Porcello, "Learning React: Functional Web Development with React and Redux," 2017.

[6] D. A. Watt, "Programming Language Design Concepts," 2006.

[7] M. Finifter, A. Mettler, N. Sastry ja D. Wagner, "Verifiable Functional Purity in Java," 2008.

[8] C. Okasaki, "Purely Functional Data Structures," 1996.

[9] D. Flanagan, "JavaScript – The definitive guide (6 ed.)," 2016.

[10] H. Pettersen, "From Static and Dynamic Websites to Static Site Generators," 2016.

[11] C. Northwood, "The Full Stack Developer," 2018.

[12] C. Aquino ja T. Gandee, ”Front-End Web Development: The Big Nerd Ranch Guide,” 2016.

[13] E. Czaplicki, ”Elm website,” 2020.

[14] L. Halvorsen, ”Functional Web Development with Elixir, OTP, and Phoenix,” 2018.

[15] Facebook Inc., ”React website,” 2020.

[16] Z. Hu, J. Hughes ja M. Wang, ”How functional programming mattered,” 2015.

[17] J. Potter, ”NPM Trends,” 2020.