Tampereen yliopisto

Tuomas Kaukoranta

ELIXIR-OHJELMOINTIKIELI JA PHOENIX-OHJELMISTOKEHYS WWW-PALVELINTEN TOTEUTUKSESSA

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Mikko Nurminen
Huhtikuu 2020

TIIVISTELMÄ

Tuomas Kaukoranta: Elixir-ohjelmointikieli ja Phoenix-ohjelmistokehys WWW-palvelinten toteutuksessa
Kandidaatintyö
Tampereen yliopisto
Tietotekniikan koulutusohjelma
Huhtikuu 2020

Tämä kandidaatintyö käsittelee Elixir-ohjelmointikieltä, Phoenix-ohjelmistokehystä, sekä näiden soveltuvuutta WWW-palvelinten toteutukseen. Aluksi käsitellään WWW-palvelinten merkitys, toiminta ja niiden kehittämisen haasteet. Tämä antaa tarvittavan kontekstin ymmärtämään miksi työn aihetta käsitellään.

Seuraavaksi esitellään Elixir-ohjelmointikieli. Osiossa käsitellään myös Elixirin edeltäjää Erlangia ja näiden molempien käyttämää virtuaalikonetta BEAMia. Lukijalle kerrotaan kielen ominaisuuksista, taustoista ja vahvuuksista. Esittelyistä ohjaudutaan tutkimuskysymykseen seuraavassa osuudessa, jossa kerrotaan Elixirin WWW-palvelinten kehitykseen erikoistuneesta Phoenix-ohjelmistokehyksestä ja sen ominaisuuksista. Lopuksi esitellään kahta suurta olemassa olevaa Elixirillä kehitettyä ohjelmistoa: Pleromaa ja Discordia. Tämä osuus havainnollistaa, ettei kieli ole pelkästään kokeellisella tasolla vaan soveltuu reaalimaailman haasteisiin.

Tutkimuskysymykseen vastaaminen pohjautuu Phoenix-kehyksen käsittelyyn ja olemassa oleviin Elixir-projekteihin. Phoenix on suunniteltu nimenomaan helpottamaan WWW-palvelinten kehitystä Elixir-kielellä. Kehys kattaa tärkeimpiä tarpeita mitä WWW-palvelinten toteutuksessa yleensä tarvitaan, kuten WWW-protokollien tuki, tietokantaintegraatio, sekä HTML-näkymien luonti mallipohjien avulla. Esitellyistä ohjelmistoista Pleroma antaa esimerkin Elixirin ja Phoenixin onnistuneesta käytöstä haasteellisessa projektissa. Toinen esitelty ohjelmisto Discord taas kertoo Elixirin skaalautumisesta jopa miljoonien samanaikaisten käyttäjien palvelemiseen. Elixirin ja Phoenixin esittelyjen pohjalta sekä onnistuneiden ohjelmistojen perusteella voidaan siis todeta että Elixir-ohjelmointikieli ja Phoenix-ohjelmointikehys ovat varteenotettavia vaihtoehtoja WWW-palvelinten toteutuksessa.

Avainsanat: Elixir, Phoenix, WWW-palvelimet

ALKUSANAT

Tämän työn kirjoittaminen on ollut pitkäaikainen haaste henkilökohtaisista syistä, mutta aiheen Elixir-ohjelmointikieli ei ole missään vaiheessa poistunut mielestä. En ole Elixir-kehittäjä, mutta työssä käsitelty projekti Pleroma on itselleni tärkeä monessa mielessä. Toivon että työ herättää mielenkiintoa lukijassa perehtymään tarkemmin Elixir-kieleen. Suurkiitos Mikko Nurmiselle ohjauksesta, neuvoista ja kärsivällisyydestä.

Tampereella, 30.4.2020

Tuomas Kaukoranta

SISÄLLYSLUETTELO

1. JOHDANTO

WWW eli World Wide Web tai web on yksi internetin kulmakivistä. WWW on hajautettu järjestelmä, joka koostuu protokollista, joilla palvelimet ja asiakkaat vaihtavat informaatiota. Tämä informaatio voi olla esimerkiksi dokumentteja tai mediaa. Webin levinneisyyden vuoksi WWW-palvelimet ovat ohjelmistokehityksen alalla tärkeä kehityskohde. Jokainen WWW-sivusto tarvitsee taustalle palvelinohjelmiston, joka toteuttaa WWW-palvelimen tehtävät. WWW-palvelinten kehittämiseen on olemassa lukuisia eri teknologioita ja yleisessä käytössä on muun muassa ohjelmointikieliä kuten Java, C# ja PHP. Kaikki nämä kielet perustuvat jollain tasolla imperatiiviseen ohjelmointiin tai olio-ohjelmointiin. WWW-palvelinten vaatimuksiin voisi sopia myös ohjelmointikieliä, jotka ovat suunniteltu toisenlaisilla lähestymistavoilla.

Tämä kandidaatintyö käsittelee Elixir-ohjelmointikieltä WWW-palvelinten yhteydessä. Elixir on funktionaalinen ohjelmointikieli, joka pohjautuu vanhempaan Erlang-kieleen ja käyttää sen virtuaalikonetta BEAMia.

Elixirin tarkoituksena on pitää kaikki Erlangin tarjoamat hyödylliset abstraktiot, mutta samalla modernisoida syntaksia ja tuoda uusia ominaisuuksia muista ohjelmointikielistä. Tutkimuskysymyksenä on, kuinka hyvin Elixir-ohjelmointikieli soveltuu käyttöön WWW-palvelimissa. Tutkimuskysymykseen vastataan käymällä läpi, miten Elixirin vahvuudet sopivat web-palvelinten vaatimuksiin.

Aiheen käsittely on jaettu neljään osaan. Ensin luvussa 2 esitellään WWW-palvelimet, mitä ne ovat ja miten ne toimivat osana koko World Wide Webiä. Samassa luvussa käsitellään myös mitä ominaisuuksia palvelimilta käytännössä vaaditaan. Luvussa 3 esitellään Elixir-ohjelmointikieltä, sekä Erlang-järjestelmää, johon Elixir pohjautuu. Käsiteltävinä kohteina ovat ominaisuudet, jotka ovat työn ja tutkimuskysymyksen kannalta merkityksellisiä. Luvussa 4 tutustutaan ensin Phoenix-ohjelmistokehykseen, joka mahdollistaa Elixir-kielen käytön palvelinkäytössä. Luvussa myös käsitellään itse tutkimuskysymystä käymällä läpi miten Elixirin eri ominaisuudet sopivat WWW-palvelinten vaatimuksiin ja miksi. Luvussa 5 esitellään näyttäviä Elixir-kieltä ja Phoenix-kehystä käyttäviä ohjelmistoja ja mihin ne pystyvät. Lopuksi yhteenvedossa kootaan lukujen 4 ja 5 johtopäätökset ja vastataan tutkimuskysymykseen.

2. WWW-PALVELINTEN KEHITYS

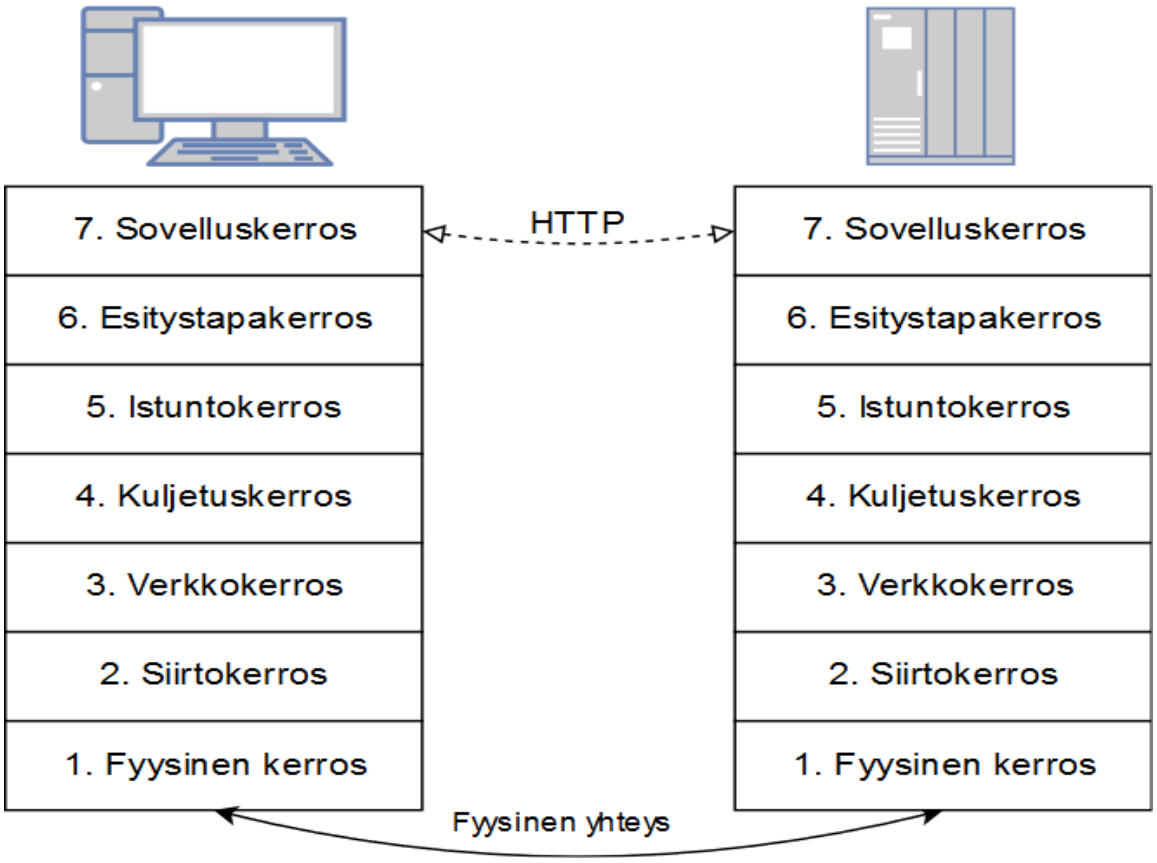
Nykyajan tietotekniikassa World Wide Web eli lyhyemmin WWW tai web on tärkeä osa lähes kaikkea tietokoneiden välistä kommunikaatiota. Verkon selaaminen tietokoneilla tai puhelimilla on monille arkipäivää ja käyttäjien määrä kasvaa jatkuvasti. Kun web kasvaa, asiakkaita on paljon, palvelimiakin on paljon ja palvelimien on myös pystyttävä palvelemaan yhä useampia asiakkaita. WWW-palvelimet vastaavat asiakkaan pyyntöihin ja palauttavat tietoa, jota voidaan esimerkiksi WWW-selainten tapauksessa piirtää valmiiksi verkkosivuiksi käyttäjän laitteen näytölle. Muutkin kuin selaimet voivat toimia asiakkaina, muutkin ohjelmat ja toiset palvelimetkin voivat kommunikoida palvelinten kanssa erikseen määriteltyjen rajapintojen (API, application programming interface) kautta.

Palvelimella voidaan tarkoittaa sekä palvelintietokonetta että palvelinohjelmistoa. Tässä työssä WWW-palvelimella tarkoitetaan nimenomaan ohjelmistoa, joka vastaanottaa, käsittelee ja vastaa asiakkaiden pyyntöihin.

2.1 Toiminta

Verkossa tietokoneiden välinen toiminta ja kommunikaatio perustuu fyysisen infrastruktuurin lisäksi tarkkaan määriteltyihin protokolliin. Web käyttää pääasiassa HTTP-protokollaa (HyperText Transfer Protocol). HTTP perustuu asiakas-palvelin arkkitehtuuriin, jossa asiakkaat (kuten esimerkiksi WWW-selaimet) ottavat yhteyttä palvelimiin ja luovat pyyntöjä, joita palvelimet käsittelevät ja vastaavat niihin. Protokollana HTTP sijaitsee tiedonsiirron OSI-mallissa sovelluskerroksella, eli kaikkein korkeimmalla kerroksella [7].

Tämän kerrosmallin ideana on kuvata, kuinka tiedonsiirto tapahtuu tietokoneiden välillä. Sovelluksesta lähetettävä tieto valuu alaspäin sovelluskerrokselta fyysiseen kerrokseen kaikkien muiden kerrosten läpi. Jokainen kerros pakkaa tai muuntaa tietoa lähemmäksi sellaista muotoa, joka voidaan lähettää fyysisesti bitteinä toisille laitteille. Vastaanottavassa päädyssä kerrokset käydään läpi päinvastaisessa järjestyksessä, kunnes sovelluskerros saa tiedon siinä muodossa, missä se on lähetetty. Kerrokset eivät ota kantaa siihen, miten muut kerrokset käsittelevät tietoa, esimerkiksi esitystapakerros ei välitä käyttääkö sovelluksen tieto HTTP-protokolla vai jotain muuta. OSI-malli ja HTTP:n paikka mallissa on havainnollistettu kuvassa 1. [11]



Kuva 1. *HTTP ja kerrokset OSI-mallissa [7][10].*

HTTP-protokollassa kaikki asiakkaan ja palvelimen välinen liikenne lähtee asiakkaan pyynnöistä, pyyntöjen tärkeimmät osat ovat metodi ja polku. Metodi määrittelee mitä pyynnöllä halutaan tehdä, esimerkiksi GET tai POST. Polku määrittelee mitä resurssia haetaan tai minne pyyntö tarkalleen on lähetetty, esimerkiksi */hakemisto/dokumentti.html*. Muuta pyyntöihin kuuluvaa ovat versionumero, otsikot ja POST tai PUT metodien yhteydessä mahdollista lisäinformaatiota kuten tekstiä tai tiedostoja. Nämä tiedot kertovat palvelimelle lisää tietoa asiakkaasta, jotta palvelin pystyy vastaamaan asiakkaan toivomalla tavalla. [7]

Palvelin lähettää asiakkaalle vastauksen jokaista pyyntöä kohden. Vastaus koostuu ensin tilakoodista, joka kertoo jos pyynnössä meni jotain vikaan tai jos pyyntö onnistui normaalisti. Tilakoodi on kolminumeroinen luku, joka kertoo miten pyyntö on käsitelty tai mitä on mennyt vikaan, kuten *200 – OK*, *404 – ei löydy* tai *500 – sisäinen palvelinvirhe*.

Onnistuneisiin pyyntöihin vastataan usein myös halutulla resurssilla, kuten esimerkiksi pyydetyllä HTML-dokumentilla.

Protokollan määritelmän mukaan palvelimen on aina vastattava asiakkaan HTTP-pyyntöihin, jos ei mitenkään muuten niin asianmukaisella virhekoodilla. Tämä vaatimus tarkoittaa kuitenkin sitä, että on käytettävä luotettavaa siirtoprotokollaa, ja että palvelin ei saisi jättää palvelematta asiakkaan pyyntöä, vaikka se olisikin kuormitettu suurella määrällä muita pyyntöjä. [7]

2.2 Vaatimukset

Hyvän WWW-palvelimen toteutuksessa on tiettyjä laadullisia vaatimuksia pelkästään protokollien noudattamisen lisäksi. Palvelimen tarjoaman sivuston käyttämisen tulisi olla käyttäjille turvallista, virheetöntä ja ripeää. Palvelimen täytyy myös kyetä tallentamaan ja käsittelemään tallennettua tietoa tietokannassa, eli palvelimen täytyy toimia tietokannanhallintajärjestelmän kanssa sujuvasti.

Luottamuksellisen informaation kanssa on tärkeää, että palvelintoteutus on mahdollisimman turvallinen. Monet haavoittuvuudet ovat palvelinohjelmiston ulkopuolella, kuten kuljetustasolla, käyttäjissä, tai palvelimen ympäristössä, mutta eivät kaikki [8]. Jotkin haavoittuvuudet, kuten mielivaltaisen ohjelmakoodin suorittaminen palvelintietokoneella, ovat riippuvaisia itse palvelinohjelmiston toteutuksesta. Jos hyökkääjä pääsee suorittamaan haluamaansa ohjelmakoodia palvelimella, hyökkääjä pystyy tekemään sivustolle mitä haluaa. [19]

Palvelimen olisi suotavaa olla luotettava ja vakaa. Tämä tarkoittaa käytännössä sitä, että palvelin osaa käsitellä virhetilanteet ja pystyy jatkamaan toimintaansa niistä riippumatta. Mahdollisia virhetilanteita voi tulla käyttäjän syötteistä, liiallisesta kuormituksesta, virheistä ohjelmakoodissa ja monesta muusta syystä. Tässä taas nousee esille HTTP-protokollan asettama vaatimus, jossa palvelimen kuuluu vastata jokaiseen pyyntöön. Toisin sanoen edes virhetilanteissa palvelin ei siis saisi joutua tilaan, jossa se ei kykene vastaamaan pyyntöihin. [7]

2.3 Haasteet

Eräs tärkeistä haasteista web-kehityksessä on suorituskyky. Web-palvelimien yhteydessä suorituskykyä voidaan käsitellä monessa osassa. Yksi tärkeä tekijä on viive, eli kuinka pitkään asiakkaan pyynnöstä vastaukseen. Verkkosivuston suorituskyky voi vaikuttaa huomattavasti sivuston suosioon, käyttäjät pysyvät pidempään sivustoilla, jotka

latautuvat nopeammin [17][18]. Palvelimen tehokkuus on vain yksi osa kokonaisviivettä, mutta silti huomionarvoinen kohde kokonaisviiveen parantamiseksi [18].

Ohjelmiston ja sen monimutkaisuuden kasvaessa sen ylläpidettävyys myös kärsii. Tämä on ongelmallista, koska palvelimiin voidaan haluta lisätä uusia ominaisuuksia tai muokata vanhaa olemassa olevaa koodia. Joskus palvelimen alkuperäinen visio ei olekaan sitä, mitä siltä tulevaisuudessa halutaan ja sitä tarvitsee uusia. Tavanomaisessa ohjelmistoprojektissa suurin osa itse kehitystyöstä kuluu ylläpitoon. Ylläpidettävyyteen vaikuttaa kuitenkin monta tekijää, kuten ohjelmakoodin luettavuus, muokattavuus ja testattavuus. [16]

3. ELIXIR-OHJELMOINTIKIELI

Elixir on dynaamisesti tyypitetty funktionaalinen ohjelmointikieli. Se on suunniteltu skaalautuvien ja helposti ylläpidettävien ohjelmien kirjoittamiseen. [3] Elixir on suhteellisen uusi ohjelmointikieli ja siitä on julkaistu 1.0.0-versio vasta vuonna 2014. [6]

Elixir pohjautuu Erlang-ohjelmointikieleen ja Elixiriä suoritetaan myös Erlangin virtuaalikoneessa, BEAMissa. Juuret Erlangissa ja BEAMissa tekee näin uudesta kielestä vakuuttavamman, kun se on rakennettu tosimaailmassa käytetyn teknologian päälle. [3] Tässä yhteydessä virtuaalikoneella tarkoitetaan ohjelmistoa, joka lukee ja tulkitsee korkeamman tason ohjelmakoodia ja muuttaa sitä laitteistotason käskyiksi. BEAMin tapauksessa Erlang- ja Elixir-kieliä molempia käännetään koodiksi BEAMille.

3.1 Funktionaalisuus

Elixir on Erlangin tapaan funktionaalinen ohjelmointikieli. Funktionaalinen ohjelmointi on toisenlainen lähestymistapa ohjelmointiin verrattuna perinteisempään laitteistoa lähempänä olevaan imperatiiviseen ohjelmointiin. Se perustuu lambdakalkyylin matemaattisiin teorioihin, ja funktioita käsitellään niiden matemaattisen määritelmän mukaan eikä alirutiineina kuten imperatiivisessa ohjelmoinnissa. Lyhyesti funktionaalisuus tarkoittaa, että funktioiden ulostulo riippuu ainoastaan syötteistä eikä muusta ohjelman tilasta. Funktioilla ei myöskään saa olla sivuvaikutuksia, eli funktiot eivät voi suoranaisesti tehdä mitään, mikä muuttaa ohjelman tilaa. [11]

Funktionaalisuuden ohjelmoinnin asettamat rajoitteet ohjelmien rakenteessa mahdollistavat monia muita hyödyllisiä asioita. Ensinnäkin ohjelmoijan on vaikeampi saada vahingossa aikaan ei-toivottua toimintaa, kun funktioilla ei ole sivuvaikutuksia. Toiseksi, kun tiedetään funktioiden aina palauttavan samat arvot samoilla syötteillä, suoritusjärjestyksellä ei ole niin paljon väliä. Funktiokutsuja voidaan siis teoriassa alkaa suorittamaan vasta, kun niiden paluuarvoja tarvitaan. [11]

Funktionaalisuus auttaa myös rinnakkaisuudessa eli monen suoritussäikeen samanaikaisessa suorituksessa. Tyypillisesti ongelmaksi rinnakkaisuudessa tulee jaettujen resurssien käsittely monella säikeellä, eli monta suoritussäiettä saattavat haluta käyttää esimerkiksi samaa muistialuetta tai ulkoista laitetta samaan aikaan. Puhtaassa funktionaalisessa ohjelmoinnissa ilman sivuvaikutuksia tiedetään, että puhtaat funktiot eivät koske ulkoiseen tilaan eivätkä siis voi lukea tai kirjoittaa jaettuihin resursseihin kesken

suorituksen. Tämä vähentää tapauksien määrää, joissa jaettuja resursseja käytetään samaan aikaan ja tekee niiden hallitsemisesta helpompaa. [11]

3.2 Erlang ja BEAM-virtuaalikone

Erlang on Ericssonin kehittämä alun perin vuonna 1986 julkaistu funktionaalinen ohjelmointikieli, joka on suunniteltu vakaiden vikasietoisten sovellusten kirjoittamiseen. Muita Erlangin tavoittelemia ominaisuuksia ovat keveys, massiivinen rinnakkaistuvuus ja pehmeä reaaliaikaisuus. [9]

Erlang on ollut käytössä alusta lähtien pääosin televiestintäjärjestelmissä, joita varten se on alun perinkin luotu [14]. Nykyään Erlangia käytetään myös ohjelmistoihin, joiden täytyy kyetä lukuisten samanaikaisten yhteyksien palvelemiseen, kuten miljoonien käyttäjien käyttämään Whatsapp-palveluun. Vuonna 2012 Whatsapp on palvellut kahta miljoonaa yhtäaikaista yhteyttä yhdeltä Erlangia suorittavalta palvelimelta. [15].

3.2.1 Prosessit

Erlang käyttää prosesseja jakamaan osia ohjelmasta itsenäisiin suoritusyksiköihin. Näitä Erlangin sisäisiä prosesseja ei tule sekoittaa kuitenkaan käyttöjärjestelmäntason prosesseihin. Erlangin sisäiset prosessit on rakennettu käyttämään vain vähän muistia ja suorituskykyä, sekä nopeiksi luoda ja tuhota. Prosessit pystyvät kommunikoimaan keskenään viesteillä. [12]

Virheen sattuessa prosessin sisällä prosessi tuhoutuu ja virheilmoitus lähetetään kaikille muille linkitetyille prosesseille, joissa virheiden seuraukset voidaan käsitellä. Erlang on suunniteltu niin, että prosessien on hyvä antaa tuhoutua tällä tavalla, sen sijaan että virhetilanteita tarkistetaan etukäteen. Tämä tekee virheenhallinnasta ja vikasietoisten ohjelmien kirjoittamisesta yksinkertaisempaa. [12]

Prosessit helpottavat myös muistinhallintaa, koska jokainen prosessi saa luomisen yhteydessä vakiokokoisen muistikeon, jota dynaamisesti kasvatetaan sen mukaan, miten prosessi tarvitsee muistia. Tällä tavalla muisti saadaan varattua yksinkertaisesti ilman päällekkäisyyttä prosessien välillä. Jokainen prosessi suorittaa itse roskienkeruun muistin vapauttamiseksi. Koska yksittäisten prosessien käyttämä muistin määrä on suhteellisen pieni, roskienkeruuseen kuluva aika ei näy suorituskyvyssä. [13]

3.2.2 Vuorontaja

Erlangin BEAM-virtuaalikoneen sisällä prosessien suorituksesta vastaavat vuorontajat. Jokaista käytettävissä olevaa käyttöjärjestelmän suoritussäiettä kohtaan voidaan luoda vuorontaja. Työn alla olevat prosessit ovat vuorontajilla jonossa, johon ei kuitenkaan sisälly niitä prosesseja, jotka ovat vain odottavassa tilassa, esimerkiksi valmiina vastaanottamaan viestejä. [9]

Vuorontaja ottavat aktiivisesti muiden vuorontajien jonoista prosesseja itselleen, jos niiden omat jonot ovat tyhjinä, jotta laskentaresurssit saadaan hyödynnettyä mahdollisimman tehokkaasti. Erlangin funktionaalinen luonne mahdollistaa prosessien odottamisen ja eriaikaisen suorituksen, koska puhtaasti funktionaaliset suoritusketjut ovat riippuvaisia ainoastaan niille annetuista lähtöarvoista eivätkä muusta ohjelman tilasta. [9]

Vuorontajilla on myös keino välttää tilanteita, joissa yksittäiset pitkäkestoiset prosessit voisivat jumittaa säikeitä liian pitkäksi aikaa. Aina kun prosessiin on käytetty tietty määrä laskentaa, se siirretään jonon kärjestä muualle, jotta sen takana odottavat prosessit saadaan suoritettua. Tämä käytännössä varmistaa, että lyhyet prosessit saadaan suoritettua, vaikka pidemmät prosessit kuormittaisivatkin vuorontajia. [9]

3.3 Elixirin omia ominaisuuksia

Teknisellä tasolla suoritettuina kielinä Erlang ja Elixir ovat luonnollisesti lähellä toisiaan, koska molempia suoritetaan samalla virtuaalikoneella. Mahdolliset eroavaisuudet saman tehtävän tekemisessä johtuvat ohjelmointikielen kääntäjän tehokkuudesta muuttaa lähdekoodia virtuaalikoneen käskyiksi.

Käytännön suuremmat erot johtuvat siitä, miten sujuvaa ohjelmointi on kullakin kielellä. Uudempana ohjelmointikielenä Elixir on saanut vaikutteita muista suosituista Erlangia uudemmista ohjelmointikielistä. Elixir pystyy käyttämään kaikkia Erlangille valmiiksi saatavia kirjastoja sekä lisäksi nimenomaan Elixirille luotuja omia työkaluja, jotka avustavat ohjelmien kirjoittamisessa.

3.3.1 Syntaksi

Eräs Elixir-ohjelmointikielen suurimmista eroista Erlang-ohjelmointikieleen on syntaksissa. Elixir on lähes 30 vuotta uudempi ohjelmointikieli, jonka takia käytäntöjä ja tekniikoita syntaksissa on muutettu modernimmiksi. Alla olevissa koodiesimerkeissä oletetaan lukijan tuntevan joitain yleisten ohjelmointikielien perusteita.

Alla on esimerkki nimetyn funktion määrittelystä. Nimetyt funktiot voivat sisältyä ainoastaan moduuleihin. Ensin määritellään moduulin nimi, sitten funktion nimi ja sen käyttämät argumentit. Lopulta avainsanojen *do* ja *end* välissä on itse funktion sisältö. Ohjelman 1 esimerkin funktio yksinkertaisesti antaa paluuarvona annetun argumentin kaksinkertaisena, eli suorittaa lähtöarvolle tuplauksen.

```
1     # Matikkamoduuli sisältää tuplausfunktion
2     defmodule Matikka do
3     def tuplaus(numero) do
4     numero * 2
5     end
6     end
8     # Tulostaa arvon 8
9     IO.puts(Matikka.tuplaus(4))
```

Ohjelma 1. *Nimetty funktio.*

Sama funktio voidaan toteuttaa ilman moduulia anonyyminä funktiona käyttäen *fn* –avainsanaa, ja vielä lyhyemmin käyttäen kaappausoperaattoria &. Nimensä mukaisesti anonyymeillä funktioilla ei ole nimeä, joten niiden kutsumiseksi ne täytyy asettaa johonkin muuttujaan tai käyttää niitä suoraan funktioiden syötteinä tai paluuarvoina.

Kaappausoperaattorilla ei tarvita erillistä argumenttilistaa, vaan &1 on automaattisesti ensimmäinen argumentti, &2 toinen, ja niin edelleen. Kaappausoperaattorin tarkoituksena on mahdollistaa anonyymien funktioiden kirjoitus nopeammin ja tiiviimmin. Tilanteesta riippuen kaappausoperaattorin tiiviimpi muoto tai normaalin anonyymin funktion nimetyt argumentit voivat olla parempi vaihtoehto. Molemmat tavat on havainnollistettu ohjelmassa 2.

```
1     # Normaali anonyymi funktio asetetaan muuttujaan 'tuplaus'
2     tuplaus = fn (numero) -> numero * 2 end
3     # Sama toteutettu kaappausoperaattorilla.
4     tuplaus = &(&1 * 2)
5     # Kutsuttaessa tarvitaan . nimen jälkeen.
6     IO.puts(tuplaus.(4))
```

Ohjelma 2. *Anonyymit funktiot.*

Yksi Elixirin hienoista ominaisuuksista on putkioperaattori |>, jolla saadaan ketjutettua funktiokutsuja yksinkertaisella syntaksilla. Putkioperaattori ottaa edellisen lausekkeen tuloksen, ja antaa sen argumentiksi seuraavalle lausekkeelle. Ohjelman 3 esimerkissä halutaan antaa yhteenlaskun tulos argumentiksi tuplaukselle, jonka tulos halutaan tulostaa.

```
1     # Ilman putkioperaattoria.
2     IO.puts(Matikka.tuplaus(2+2))
3     # Käyttäen putkioperaattoria.
4     (2+2) |> Matikka.tuplaus |> IO.puts
```

Ohjelma 3. *Esimerkki putkioperaattorin käytöstä.*

Funktiokutsujen ketjuttaminen putkioperaattorilla helpottaa lähdekoodin luettavuutta, erityisesti funktionaalisissa ohjelmointikielissä. Sisäkkäiset kutsut luovat rakenteen, jossa ikään kuin mennään syvemmälle suorituksessa, mutta putkituksella sama asia näyttää enemmän samalla tasolla tapahtuvista peräkkäisiltä operaatioilta.

3.3.2 Mix-koontityökalu

Mix on koontityökalu, joka tulee Elixir-asennusten mukana. Sen avulla helpotetaan Elixir-projektien luontia, kääntämistä, testausta, sekä riippuvuuksien hallintaa. Toimivat koontityökalut auttavat siirtymään yksinkertaisten funktioiden testaamisesta kokonaisten sovellusten kehittämiseen. [20]

Riippuvuuksien hallinta suuremmissa projekteissa on yksi ylläpidollisista haasteista. Riippuvuuksilla tarkoitetaan olemassa olevia kirjastoja tai muuta koodia, joita ohjelmisto käyttää hyödykseen. Haasteet johtuvat siitä, kun riippuvuudet saattavat toimia keskenään ristiriidassa, ja niiden eri versioissa voi olla eroavaisuuksia toiminnassa. Mix tarjoaa käyttöön Hex-paketinhallintatyökalun riippuvuuksien hallintaan. [20]

4. ELIXIR-OHJELMOINTIKIELEN SOPIVUUS WWW-PALVELIMIIN

4.1 Phoenix-ohjelmistokehys

Kokonaisen web-palvelimen toteutus tyhjältä pöydältä ei ole käytännöllistä ohjelmointikielestä riippumatta. Todellista käyttöä varten olisi suotavaa olla jonkinlainen ohjelmistokehys palvelinten kirjoittamiseen. Ohjelmistokehys on kokoelma valmista koodia tai työkaluja, jonka tarkoituksena on luoda abstraktiokerros helpottamaan tietyn tyypin ohjelmien kirjoittamista.

Phoenix on nimenomaan Elixir-ohjelmointikielelle tarkoitettu ohjelmistokehys web-palvelinten kehitykseen. Phoenix tarjoaa monia ominaisuuksia ja abstraktioita helpottamaan web-palvelimien kehitystä lähes mihin tahansa yksilöllisiin tarkoituksiin. Ohjelmistokehyksenä Phoenix koostuu monesta erillisestä osasta, joista osa on olemassa olevia Erlang- ja Elixir-kirjastoja. Kokonaisuutena Phoenix on liian laaja tämän työn käsiteltäväksi, mutta tärkeimpien komponenttien pääpiirteet käydään läpi.

Kehyksenä Phoenix luo tapoja rakentaa web-sovelluksia ja se sisältää tarvittavat perusosat ja abstraktiot helpottamaan kehitystä. Ytimessään se sisältää HTTP-palvelimen, joka käsittelee matalan tason käsittelyn HTTP-kyselyille ja muille yhteyksille. Seuraavalla tasolla Phoenix sovelluksissa on reitittäjä, jonka kautta palvelimen kehittäjä määrittelee mitä ohjelmakoodia suoritetaan ja mitä asiakkaalle vastataan milläkin HTTP-pyynnön polulla. Tämän lisäksi Phoenix tarjoaa abstraktioita malli-näkymä-käsittelijä–arkkitehtuuria varten. Mallia varten on myös tuki tietokannoille, joihin WWW-palvelimet tallentavat tietoa. Vastauksien luomista varten Phoenixissa on HTML-mallipohjajärjestelmä, jonka avulla voidaan yksinkertaisesti suunnitella HTTP-vastauksiin sisältyvät HTML-dokumentit mallipohjina, joihin Phoenix täyttää muuttuvat tiedot ohjelmoijan määrittämällä tavalla. Muita hyödyllisiä ominaisuuksia ovat automaattitestaustyökalut ja kanavat muiden ohjelmien kanssa kommunikoimiseen, jotka eivät välttämättä käytä HTTP-protokollaa.

4.2 Rakenne

Phoenix on suunniteltu ja rakennettu modulaariseksi. Se koostuu monesta erillisestä osasta, jotka kommunikoivat keskenään muodostaakseen kokonaisuuden. Myös Phoenixin päälle rakennetut sovellukset ovat modulaarisia. Tämä johtuu osaksi jo Elixirin funktionaalisesta luonteesta ja kuinka se käyttää omia prosessejaan. On yksinkertaisempaa pitää asiat modulaarisena, jos moduuleilla on selvästi määritetyt syötteet ja paluuarvot, eivätkä ne riipu ympäröivästä tilasta.

Rajapinnoilla voidaan varmistaa, että järjestelmät voivat kommunikoida keskenään valmiiksi sovituilla tavoilla. Phoenix käyttää Plugia, joka määrittelee rajapinnan moduuleille, jotta niitä voidaan käyttää vaivattomasti keskenään. Plug on suunniteltu juuri web-sovelluksia varten yksinkertaistamaan tiedon käsittelyä ja välitystä pyynnöissä. Plugin rajapinnan voi täyttää jo pelkällä funktiolla tai moduulilla, jonka jälkeen niitä voidaan ketjuttaa muodostamaan suurempia kokonaisuuksia. Yksittäiset operaatiot mitä Plug-moduulit voivat tehdä voivat olla esimerkiksi HTTP-otsikoiden asettaminen, tai lokitus. Ohjelmassa 4 on luotu yksinkertainen reititin Phoenix-kehyksessä käyttäen Plugia.

```
1  # Määritetään reititysmoduuli.
2  defmodule PalvelinSofta.Reititin do
3    # Kerrotaan Elixirille, että käytämme Phoenixin :router-toimintoja.
4    use PalvelinSofta, :router
5
7    # Muodostetaan Plug-liukuhihna kaikille selainkyselyille.
8    pipeline :selain do
9      plug :accepts, [”html”]
10     plug :fetch_session
11     plug :put_secure_browser_headers
12   end
13
14   # Määritetään polku jonka kyselyihin vastataan
15   scope ”/”, PalvelinSofta do
16     # Kaikki pyynnöt tähän polkuun kulkevat :selain-putken läpi
17     pipe_through :selain
18     # Moduulin OletusController funktio sivu vastaa kyselyyn
19     get ”/”, OletusController, :sivu
20   end
21 end
```

Ohjelma 4. *Esimerkki Plug-liukuhihnasta Phoenix-reitittimessä.*

Esimerkin *pipeline*-lohkossa määritetään mitä kaikkia Plug-toimintoja on ketjutettu tehtäväksi kyselylle. Tässä esimerkissä on käytetty vain Phoenixin mukana tulleita toimintoja, joilla muutetaan HTTP-otsikoita ja varmistetaan että muut toiminnot pääsevät kyseiseen

istuntoon käsiksi. Reitittimen tarkoitus on yhdistää selaimen tai muun sovelluksen tekemä kysely johonkin toimintoon, eli tässä esimerkissä *OletusController*-moduulin sivufunktioon, joka voi tavanomaisessa tilanteessa vastata kyselyyn HTML-dokumentilla.

4.3 Tietokantaintegraatio

Varteenotettavilla WWW-palvelimilla täytyy olla mahdollisuus käyttää tietokantoja tietojen luotettavaan ja tehokkaaseen tallentamiseen. Käytännössä tämä tarkoittaa sitä, että palvelinohjelmisto pystyy käyttämään jotain tietokannanhallintajärjestelmää. Phoenix pystyy käyttämään useita eri tietokannanhallintajärjestelmiä. Näihin kuuluvat esimerkiksi PostgreSQL ja MySQL.

Phoenix käyttää Elixir-kirjastoa nimeltä Ecto joka kääriytyy halutun tietokannanhallintajärjestelmän päälle, jotta palvelinohjelmiston kehittäjän ei tarvitse huolehtia kyseisen järjestelmän yksityiskohdista. Ecto auttaa tietokantojen suunnittelussa ja kyselyiden muodostamisessa, sekä suojelee tietokantaa tietyntyyppisiltä hyökkäyksiltä, joita huolimaton ohjelmoija ei välttämättä osaa ottaa huomioon kirjoittaessaan WWW-palvelinta.

4.4 Cowboy

Phoenixin alin kerros on sen käyttämä Erlangilla kirjoitettu HTTP-palvelinohjelmisto. Cowboy vastaanottaa HTTP-pyynnöt ja Phoenix kytkee Cowboyn palvelinohjelmiston Elixirillä kirjoitettuun ohjelmakoodiin. Cowboy on suunniteltu kevyeksi, nopeaksi ja luotettavaksi, mikä tekee sen sopivaksi osaksi Phoenix-kehystä.

Cowboy tukee HTTP-protokollan versioita HTTP/1.1 ja HTTP/2, sekä Websocket-protokollaa ja REST-rajapintoja. Kaikki nämä ovat modernissa web-kehityksessä olennaisia teknologioita, mutta ne kuuluvat tämän työn ulkopuolelle.

4.5 Näkymät

Näkymät ovat Phoenixissa viimeinen abstraktiokerros ennen kuin vastaus lähetetään asiakkaalle. Näkymät ovat vastuussa esimerkiksi HTML-dokumentin luomisesta käyttäen mallipohjia, joihin syötetään tietoa aiemmista vaiheista, joka on esimerkiksi haettu tietokannasta ja käsitelty sopivaan muotoon. Näkymän ei ole pakko muodostaa HTML-dokumenttia, vaan se voi myös muodostaa esimerkiksi JSON-muotoista dataa.

Mallipohjiin Phoenix käyttää ”Embedded Elixir”-järjestelmää (EEx), jonka avulla voidaan upottaa Elixir-koodia vapaamuotoisen tekstin sisään ja näin täyttää ohjelmallisesti arvoja mihin tahansa tekstimuotoiseen sisältöön. Phoenixin käyttötarkoituksessa EEx:n avulla voidaan luoda HTML-muotoisia mallipohjia, joissa ohjelmallisesti muuttuvat osat ovat toteutettu upotetulla Elixir-koodilla. Tämä mahdollistaa yksittäisten arvojen täyttämisen lisäksi esimerkiksi listamuotoisen tiedon käsittelyä ja näyttämistä.

```
1  <div>
2    <h1>Hei, <%= kayttajanimi %></h1>
3    <p>
4      Katsaus ostoksistasi:
5    </p>
7    <%= for tuote <- ostokset do %>
8      <h4><%= tuote.nimi %></h4>
9      <p><%= tuote.tilausnumero %></p>
10   <% end %>
11 </div>
```

Ohjelma 5. *Esimerkki EEx-muotoisesta HTML-mallipohjasta.*

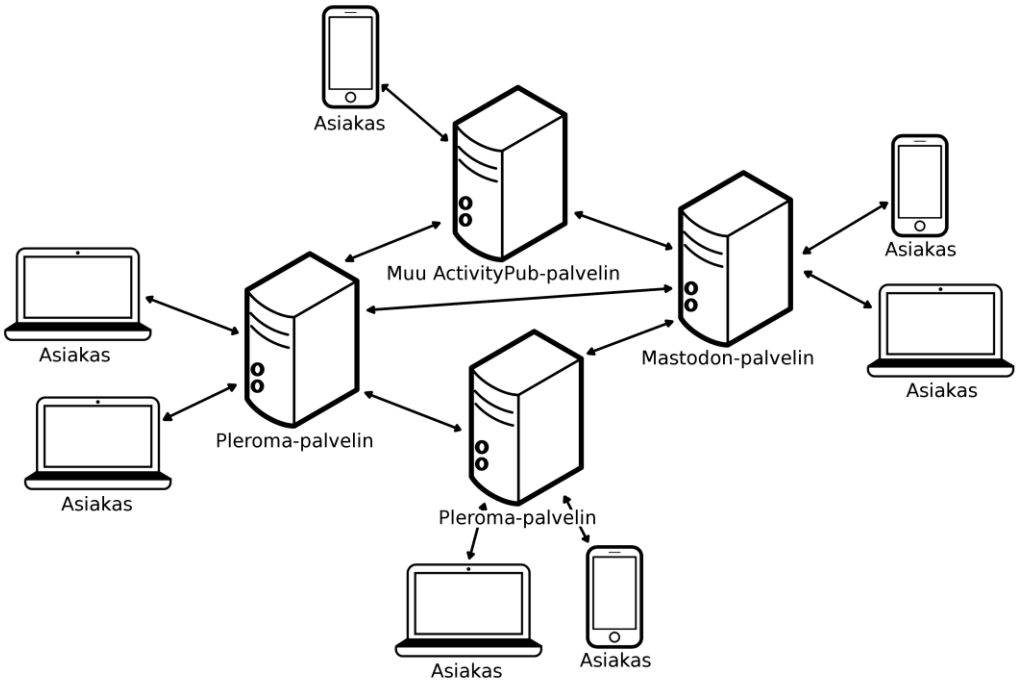
Ohjelmassa 5 on esimerkki HTML-mallipohjasta, jossa Elixir-osuuksien alut ovat merkitty <% tai <%= –merkinnällä ja loput %> –merkinnällä. Kaikki muut osuudet ovat normaalia HTML-muotoista dokumenttia. Esimerkissä huomataan yksittäinen arvo otsikossa, jossa *kayttajanimi* on muuttujan nimi, ja sen tilalle asetetaan sen arvo lopullisessa dokumentissa. Tämän jälkeen on myös lista joka on luotu käyttäen *for*-avainsanaa Elixir-lohkon sisällä. Kaikki sisältö *for* ja *end*-avainsanojen välissä toistetaan niin monta kertaa, kuin *ostokset*-listarakenteessa on kohteita.

5. ELIXIR MAAILMALLA

Toistaiseksi on vielä melko vaikeaa löytää useita hyviä esimerkkejä Elixirin käytöstä maailmalla. Puhumattakaan Elixirin käytöstä Phoenix-kehyksen kautta. Mahdollisia syitä tähän ovat Elixirin uutuus kielenä ja sen takia myös Phoenix voidaan nähdä kokeellisena. Kuitenkin muutaman vuoden aikana on ilmaantunut joitain näyttäviä ohjelmistoprojekteja, jotka käyttävät Elixir-ohjelmointikieltä tärkeässä roolissa.

5.1 Pleroma

Avoin hajautettu mikrobloggaukseen tarkoitettu palvelinohjelmisto Pleroma on hyvä esimerkki Elixirin ja Phoenix-kehyksen käytöstä suuremmassa sovelluskokonaisuudessa [21].



Kuva 2. *ActivityPub-verkoston rakenne.*

Palvelimena se ei pelkästään palvele asiakkaita, kuten selaimia ja muita loppukäyttäjien sovelluksia, mutta se myös kommunikoi muiden vastaavien palvelinten kanssa luoden suuremman hajautetun verkoston, jossa käyttäjät voivat vuorovaikuttaa palvelimelta toiselle. Pleroma käyttää pääasiassa ActivityPub-protokollaa kommunikoidessaan muiden

palvelimien kanssa ja HTTP-protokollaa asiakkaiden kanssa [21][22]. Tämänkaltainen taakka on raskaampi mitä useampaan muuhun palvelimeen se on yhdistynyt ja mitä useampi käyttäjä sillä on, koska jokainen yksittäinen palvelin käsittelee jokaiselta muulta palvelimelta saadut tapahtumat ja lähettää kaikki sen omat tapahtumat muille palvelimille.

Haasteellisista vaatimuksista riippumatta Pleroma on kuitenkin tarkoitettu kevyeksi ja nopeaksi. Sen kehittäjän mukaan se on tarpeeksi kevyt jopa yhden piirilevyn Raspberry Pi -tietokoneilla suoritettavaksi [21]. Pleroman 1.0.0 versio on julkaistu 28.6.2019 ja kehitys on edelleen aktiivista [23]. Kirjoitushetkellä ActivityPub-verkostossa on yli 650 aktiivista Pleroma-palvelinta, joille on rekisteröitynyt yhteensä yli 37000 käyttäjää [24].

5.2 Discord

Discord on vuonna 2015 julkaistu Elixir-kielellä kehitetty suosittu teksti- ja puheviestintäpalvelu, joka palvelee miljoonia samanaikaisia käyttäjiä. [25] Miljoonat samanaikaiset käyttäjät ovat haaste mille tahansa palvelulle, mutta erityisesti puheviestinnässä palvelimen pitää pystyä pitämään asiakkaan ja palvelimen välinen viive kurissa.

Elixir ja Erlang on auttanut luomaan hyvän skaalautuvan pohjan Discordille. Toisin kuin Pleroma, Discord käyttää Elixiriä ilman Phoenix-kehystä. Discordin käyttää toiminnassaan pääosin WebSocket ja WebRTC-protokollia HTTP-protokollan sijaan. [26] Elixirin valinta Discordin tapauksessa perustuu BEAM-virtuaalikoneen vahvuuksiin ja kielen moderniuteen ja käyttäjäystävällisyyteen. [25]

Kielen ja virtuaalikoneen vahvuuksista huolimatta Discord on kohdannut useita teknisiä haasteita suorituskyvyssä. Eräässä suorituskykyongelman ratkaisussa Discordin kehittäjät ovat hyödyntäneet BEAM-virtuaalikoneen ominaisuutta suorittaa konekielisiä funktioita, jotka ovat kirjoitettu matalamman tason kielellä kuten C tai Rust-ohjelmointikielillä. [27] Tämä ominaisuus mahdollistaa kriittisten funktioiden uudelleenkirjoittamisen nopeammaksi toisella kielellä ilman että kokonaiskuvassa luovuttaisiin Elixirin vahvuuksista. Tämä ominaisuus korostaa Elixirin joustavuudesta miljoonienkin käyttäjien skaalassa.

6. YHTEENVETO

Elixir on rakennettu Erlangin vankalle pohjalle ja käyttää sen virtuaalikonetta BEAMia. Erlang on suunniteltu varta vasten tehokkaisiin ja vikasietoisiin televiestintäjärjestelmiin ja Elixir perii kaikki Erlangin vahvuudet tässä mielessä. Erlang ja Elixir ovat molemmat funktionaalisia ohjelmointikieliä, funktionaalinen ohjelmointi on suosittu ohjelmointiparadigma nykyään. Kuitenkin Elixirin syntaksi on modernia ja lähestyttävää, mikä tekee siitä helpomman vaihtoehdon Erlangiin verraten.

WWW-palvelimen kehittäjille Elixir on myös varteenotettava vaihtoehto, koska Phoenix tarjoaa modernin ohjelmistokehyksen kyseiseen tarkoitukseen. Phoenix on rakennettu varta vasten Elixirille ja se tarjoaa paljon ominaisuuksia, jotka ovat web-kehityksessä olennaisia, kuten pyyntöjen käsittely eri protokollien kautta, abstraktiot reititykselle ja näkymille ja tietokantaintegraation. Lisäksi Phoenix on rakennettu modulaariseksi ja auttaa pitämään palvelinohjelmiston arkkitehtuurin yksinkertaisempana ja ylläpidettävänä.

Käsiteltyjen asioiden valossa Elixirin ja Phoenixin yhdistelmä on hyvinkin sopiva valinta WWW-palvelinten kehitykseen, erityisesti jos kehittäjä on omaksunut funktionaalisen ohjelmoinnin paradigman. Elixir ei kärsi samaan tapaan tuen ja kirjastojen puutteesta kuten jotkut muut uudet ohjelmointikielet, koska se toimii Erlangin pohjalla ja Phoenix on hyvin kattava ohjelmistokehys ilman suurempia puutteita.

Maailmalla Elixir ja sen käyttämä virtuaalikone BEAM on saanut jo mainetta Discord – chat-palvelun kautta, joka on käyttänyt Elixiriä palvelemaan miljoonia samanaikaisia käyttäjiä. Pleroma taas on hyvä esimerkki Elixirin ja Phoenixin yhteiskäytöstä ja sen matalat laitteistovaatimukset kertovat Elixirin tehokkuudesta. Voisi olettaa, että Elixir jatkaa kasvua lähivuosina ja sen käyttöä tullaan näkemään laajemminkin, joko Phoenixin kanssa tai ilman.

LÄHTEET

[1] W. Loder, Erlang and Elixir for Imperative Programmers, 2016.

[2] G. Fedrecheski, L. C. P. Costa, M. K. Zuffo, Elixir programming language evaluation for IoT, 2016 IEEE International Symposium on Consumer Electronics (ICSE), Sao Paulo, 2016, s. 105-106.

[3] Elixir, verkkosivu. Saatavissa (viitattu 10.9.2017): https://elixir-lang.org/

[4] F. Cesarini, S. Thompson, Erlang Behaviours: Programming with Process Design Patterns, Springer Berlin Heidelberg, Berlin, 2010. s. 19-41.

[5] T. Bonald, J. Roberts, Internet and the Erlang Formula, ACM SIGCOMM Computer Communication Review, vol. 42/no. 1, 2012, s. 23-30.

[6] Release v1.0.0 – elixir-lang/elixir, verkkosivu. Saatavissa (viitattu 1.10.2017): https://github.com/elixir-lang/elixir/releases/tag/v1.0.0

[7] An overview of HTTP, verkkosivu. Saatavissa (viitattu 15.10.2017): https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[8] H. Samsul, Web Server Vulnerability Analysis in the context of Transport Layer Security (TLS), International Journal of Computer Science Issues (IJCSI), Mahebourg, 2016, s. 11-19.

[9] R. Virding, Hitchhiker's Tour of the BEAM, Erlang User Conference, 2014, puhe. Saatavissa (viitattu 8.10.2017): https://erlangcentral.org/videos/euc-2014-robert-virding-hitchhikers-tour-of-the-beam/#.WeP9XHJzqHt

[10] The OSI Model's Seven Layers Defined and Functions Explained, verkkosivu. Saatavissa (viitattu 27.10.2017): https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained

[11] K. Heisen, The Promises of Functional Programming, Computing in Science & Engineering, vol. 11/n. 4, 2009.

[12] Processes, Erlang Reference Manual, verkkosivu. Saatavissa (viitattu 29.10.2017): http://erlang.org/doc/reference_manual/processes.html

[13] Academic and Historical Questions, Frequently Asked Questions about Erlang, verkkosivu. Saatavissa (viitattu 29.10.2017): http://erlang.org/faq/academic.html#idp33105616

[14] Inside Erlang – creator Joe Armstrong tells his story, verkkosivu. Saatavissa (viitattu 21.5.2018): https://www.ericsson.com/en/news/2014/12/inside-erlang--creator-joe-armstrong-tells-his-story

[15] 1 million is so 2011 , verkkosivu. Saatavissa (viitattu 21.5.2018): https://blog.whatsapp.com/196/1-million-is-so-2011

[16] Software maintenance, Clarity in Code, verkkosivu. Saatavissa (viitattu 28.5.2018): http://www.clarityincode.com/software-maintenance/

[17] The need for mobile speed: How mobile latency impacts publisher revenue, DoubleClick by Google, verkkosivu. Saatavissa (viitattu 28.5.2018): https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/

[18] Driving user growth with performance improvements, Pinterest Engineering, Medium, verkkosivu. Saatavissa (viitattu 28.5.2018): https://medium.com/@Pinterest_Engineering/driving-user-growth-with-performance-improvements-cfc50dafadd7

[19] Common Attacks on WordPress Sites 101: File Inclusion, and Arbitrary Code Execution, BlogVault, verkkosivu. Saatavissa (viitattu 28.5.2018): https://blogvault.net/common-attacks-on-wordpress-sites-101-file-inclusion-arbitrary-code-execution/

[20] Mix, Elixir-dokumentaatio, verkkosivu. Saatavissa (viitattu 28.5.2018): https://hexdocs.pm/mix/Mix.html

[21] What is Pleroma?, Lainblog, verkkosivu. Saatavissa (viitattu 17.12.2019): https://blog.soykaf.com/post/what-is-pleroma/

[22] Pleroma Encyclical: ActivityPub, Lainblog, verkkosivu. Saatavissa (viitattu 17.12.2019): https://blog.soykaf.com/post/pleroma-encyclical-activity-pub/

[23] Pleroma 1.0.0, Lainblog, verkkosivu. Saatavissa (viitattu 17.12.2019): https://blog.soykaf.com/post/pleroma-1.0/

[24] Pleroma Instances, Fediverse Network, verkkosivu. Saatavissa (viitattu 26.4.2020): https://fediverse.network/pleroma

[25] How Discord Scaled Elixir to 5,000,000 Concurrent Users, Discord, verkkosivu. Saatavissa (viitattu 17.12.2019): https://blog.discordapp.com/scaling-elixir-f9b8e1e7c29b

[26] How Discord Handles Two and Half Million Concurrent Voice Users using WebRTC, Discord, verkkosivu. Saatavissa (viitattu 26.4.2020): https://blog.discord.com/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc-ce01c3187429