Palvelinpuolen renderöinti Rust-ohjelmointikielellä

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
Kevät, 2023

Mikko Vasankari

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus | Tiivistelmä
Tekijä Mikko Vasankari | Vuosi 2023
Työn nimi Palvelinpuolen renderöinti Rust-ohjelmointikielellä
Ohjaaja Elina Vartiainen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli selvittää, miten Rust-ohjelmointikielellä voidaan toteuttaa verkkosivu, joka hyödyntää palvelinpuolen renderöintiä. Opinnäytetyössä tutkittiin kuinka palvelinpuolen renderöinnin voi toteuttaa Rust-ohjelmointikielen Yew-sovelluskehyksellä.

Opinnäytetyön tietopohja perustuu verkkosivujen renderöintitapoihin sekä Rust-ohjelmointikielen perusteisiin. Opinnäytetyössä käytiin läpi myös, kuinka verkkosivujen renderöintitavat (SSR ja CSR) eroavat toisistaan. Projektissa kehitettiin yksinkertainen verkkosivu käyttäen Rust-ohjelmointikielen Yew-sovelluskehyksen palvelinpuolen renderöintiä. Projektin verkkosivua testattiin Googlen kehittämällä Lighthouse-työkalulla, jossa keskityttiin testaustyökalun antamiin suorituskykyarvoihin.

Opinnäytetyön tutkimuksessa verrattiin Rust-ohjelmointikielen ja Javascript-ohjelmointikielen palvelinpuolen renderöinnin suorituskykyä. Tutkimuksessa todettiin, että ohjelmointikielen valinnalla ei ole suurta merkitystä verkkosivun suorituskykyyn. Rust-ohjelmointikielellä toteutettu verkkosivu oli kuitenkin suorituskyvyttömämpi johtuen verkkosivun käyttämästä WebAssembly-tekniikasta.

Avainsanat Rust-ohjelmointikieli, SSR, CSR
Sivut 26 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology Abstract

Author Mikko Vasankari Year 2023

Subject Server-side rendering with Rust programming language

Supervisor Elina Vartiainen

ABSTRACT

The Purpose of this thesis was to get better understanding of different website rendering methods and understanding of creating websites using Rust programming language. The thesis focuses on creating server-side rendering with Rust programming language using a web framework called Yew.

The knowledge base of the thesis consists of different website rendering methods and the basics of Rust programming language. The thesis goes through the differences of website rendering methods (SSR and CSR). The project part of the thesis creates a simple website using Yew frameworks server-side rendering functionality. For testing of the website, a tool developed by Google called Lighthouse was used to test the performance of the website.

The research part of thesis compares the performance differences between Rust and Javascript programming languages. Based on the analysis, it can be said that there was no great difference between the performance of the programming languages. However, the website created with Rust programming language was more inefficient because of its way to use WebAssembly to create websites with functionalities.

Keywords Rust programming language, SSR, CSR

Pages 26 pages and appendices 1 page

Sanasto

CSR	Client-side rendering / asiakaspuolen renderöinti, palvelimen lähettämä verkkosivu renderöidään/rakennetaan asiakkaan selaimessa.
SSR	Server-side rendering / palvelinpuolen renderöinti, verkkosivujen renderöinnin tapahtuminen palvelimen puolella.
SEO	Search Engine Optimization, Hakukoneoptimointi, tarkoitetaan hakukoneiden hakutuloksia parantavia toimenpiteitä.
HTML	HyperText markup Language, Internet sivujen määrittelykieli.
HTTP	Hypertext Transfer Protocol, Protokolla, jonka avulla verkkopalvelimet ja selaimet voivat keskustella toistensa kanssa.
Rust	Mozilla Foundation kehittämä ohjelmointikieli
Cargo	Rust-ohjelmointikielen pakettienhallintatyökalu
Yew	Rust-ohjelmointikielelle kehitetty sovelluskehys, jonka tarkoituksena on luoda verkkosovelluksia WebAssemblyn avulla
Actix web	Rust-ohjelmointikielelle kehitetty sovelluskehys, jonka tarkoituksena on luoda verkkopalveluita Rust-ohjelmointikielellä.
Trunk	WebAssembly verkkosovellusten rakentajatyökalu Rust-ohjelmointikielelle
Lighthouse	Google Lighthouse, Googlen avoimenlähdekoodin verkkosivujen testaustyökalu

Sisällys

1 Johdanto ... 1
2 Verkkosivujen renderöintitavat ... 2
2.1 Asiakaspuolen renderöinti ... 2
2.2 Palvelinpuolen renderöinti ... 4
3 Rust-ohjelmointikieli ... 6
3.1 Rust-ohjelmointikielen ominaisuuksia ... 6
3.1.1 Tyypitys ... 6
3.1.2 Omistajuus ... 7
3.1.3 Muuttumaton tieto ... 8
3.1.4 Pakettien hallinta ... 8
3.2 Rust-ohjelmointikielen hyödyt ... 9
3.2.1 Muistinhallinta ... 9
3.2.2 Luotettavuus ... 9
3.3 Rust-ohjelmointikieli ja WebAssembly ... 10
4 Projektin tarkoitus ja suunnittelu ... 11
4.1 Actix web ... 11
4.2 Yew ... 12
4.3 Google Lighthouse ... 13
5 Verkkosivupalvelin Rust-ohjelmointikielellä ... 16
5.1 Asiakaspuolen toteutus ... 17
5.2 Palvelinpuolen toteutus ... 19
5.3 Palvelimen lopputulos ... 20
6 Projektin tulokset ja johtopäätökset ... 22
6.1 Tulokset ... 22
6.2 Tuloksien vertailua ... 23
6.3 Pohdintaa ... 24
7 Yhteenveto ... 26
Lähteet ... 27

Kuvat, ohjelmakoodit ja taulukot

Komento 1 Paketinhallinta työkalu Cargo:n projektin luomiskomento ... 8

Kuva 1 Sekvenssikaavio asiakaspuolen renderöinnistä (Danilec, 2020). 3
Kuva 2 Sekvenssikaavio palvelinpuolen renderöinnistä (Danilec, 2020). 5
Kuva 3 Rust-ohjelmointikielen kääntäjän virheilmoitus.....10
Kuva 4 Rust-ohjelmointikielen kääntäjän ilmoitus käyttämättömästä muuttujasta 10
Kuva 5 Google Lighthouse -työkalun tulokset.....13
Kuva 6 Lighthouse-työkalun suorituskykyosion esimerkki tulokset.....15
Kuva 7 Projektin kansio rakenne eri osioiden luomisen jälkeen.....16
Kuva 8 Projektin kansio rakenne asiakaspuolen ohjelman rakentamisen jälkeen 18
Kuva 9 Projektin tuloksena syntynyt yksinkertainen blogiverkkosivusto 21

Ohjelmakoodi 1 Omistajuuden siirtäminen Rust-ohjelmointikielessä.....7
Ohjelmakoodi 2 Muuttuva ja muuttumaton muuttuja Rust-ohjelmointikielessä 8
Ohjelmakoodi 3 HTTP-palvelimen luonti Actix web -sovelluskehyksellä.....11
Ohjelmakoodi 4 Yew-sovelluskehyksen HTML-makro komponentti 12
Ohjelmakoodi 5 Palvelinpuolen renderöinti Yew-sovelluskehyksessä (*Yew Documentation*, 2022).....13
Ohjelmakoodi 6 Asiakaspuolen ohjelman Cargo.toml tiedoston riippuvuudet.....17
Ohjelmakoodi 7 Lähde HTML-tiedosto trunk-työkalulle.....17
Ohjelmakoodi 8 Asiakaspuolen ohjelman main.rs tiedosto.....18
Ohjelmakoodi 9 Palvelinpuolen Cargo.toml tiedosto 19
Ohjelmakoodi 10 Palvelimen GET-pyyntöön vastaava funktio 19
Ohjelmakoodi 11 Projektin Cargo.toml tiedosto.....20

Taulukko 1 Numeraalisien arvojen tyypitys 7
Taulukko 2 Google Lighthouse -työkalun tulokset projektin verkkosivustosta 23
Taulukko 3 Next.js verkkosivuston Lighthouse-työkalun tulokset (Paukkunen, 2022)...24

Liitteet

Liite 1 Aineistonhallintasuunnitelma

1 Johdanto

Nykypäivänä verkkosovelluspalvelut ovat kehittyneet suuresti ja niille on valtava määrä kysyntää ja tarjontaa. Tämän kysynnän pohjalta on syntynyt monia erilaisia verkkosovelluskehyksiä erilaisilla ohjelmointikielellä. Yleisimpänä näistä on Javascript-pohjaiset verkkosovelluskehykset kuten esimerkiksi Express.js ja Next.js. Javascriptin yleisyys näissä ei kuitenkaan tarkoita sitä, että se olisi välttämättä optimaalisin ohjelmointikieli verkkosovelluspalveluiden kehittämiseen.

Tässä opinnäytetyössä tutkitaan Rust-ohjelmointikielen mahdollisuuksia kehittää verkkosovelluspalveluita ja sen tehokkuutta suorittamaan näitä palveluita. Opinnäytetyössä myös tutkitaan verkkosivujen erilaisia renderöintistrategioita ja niiden eroavaisuuksia. Opinnäytetyön projektin tarkoituksena on luoda verkkosivupalvelu Rust-ohjelmointikielellä, jossa luodaan palvelinpuolen renderöintitoiminnallisuus.

Opinnäytetyön projektissa luodaan yksinkertainen blogiverkkosivu, jonka suorituskykyä testataan Google Lighthouse -ohjelmalla. Projektin tutkimuksessa tullaan vertaamaan saatuja tuloksia samanlaiseen yksinkertaiseen blogiverkkosivuston tuloksiin, jonka ohjelmointikielenä on käytetty Javascript-ohjelmointikieltä.

Opinnäytetyön tutkimuskysymykset ovat:

- Mikä on palvelinpuolen renderöinti?
- Mitä hyötyä palvelinpuolen renderöinnistä?
- Onko kannattavaa käyttää Rust-ohjelmointikieltä verkkosivupalvelimen kehittämisessä?

2 Verkkosivujen renderöintitavat

Verkkosivujen ja -sovellusten luomiseen on monenlaisia työkaluja. Nykypäivän verkkosovelluskehyksien yleistyessä verkkosivujen renderöinti eli tietojen muuttaminen selaimen näytettäväksi useimmiten tapahtuu sivuston käyttäjän selaimessa. Renderöinnin toteuttaminen selaimessa voi tehdä suurista sovelluksista erittäin raskaita ja hitaasti latautuvia. Tähän ratkaisuna nykyiset sovelluskehykset ovat alkaneet suorittamaan verkkosivujen renderöintiä palvelimen puolella. (Danilec, 2020)

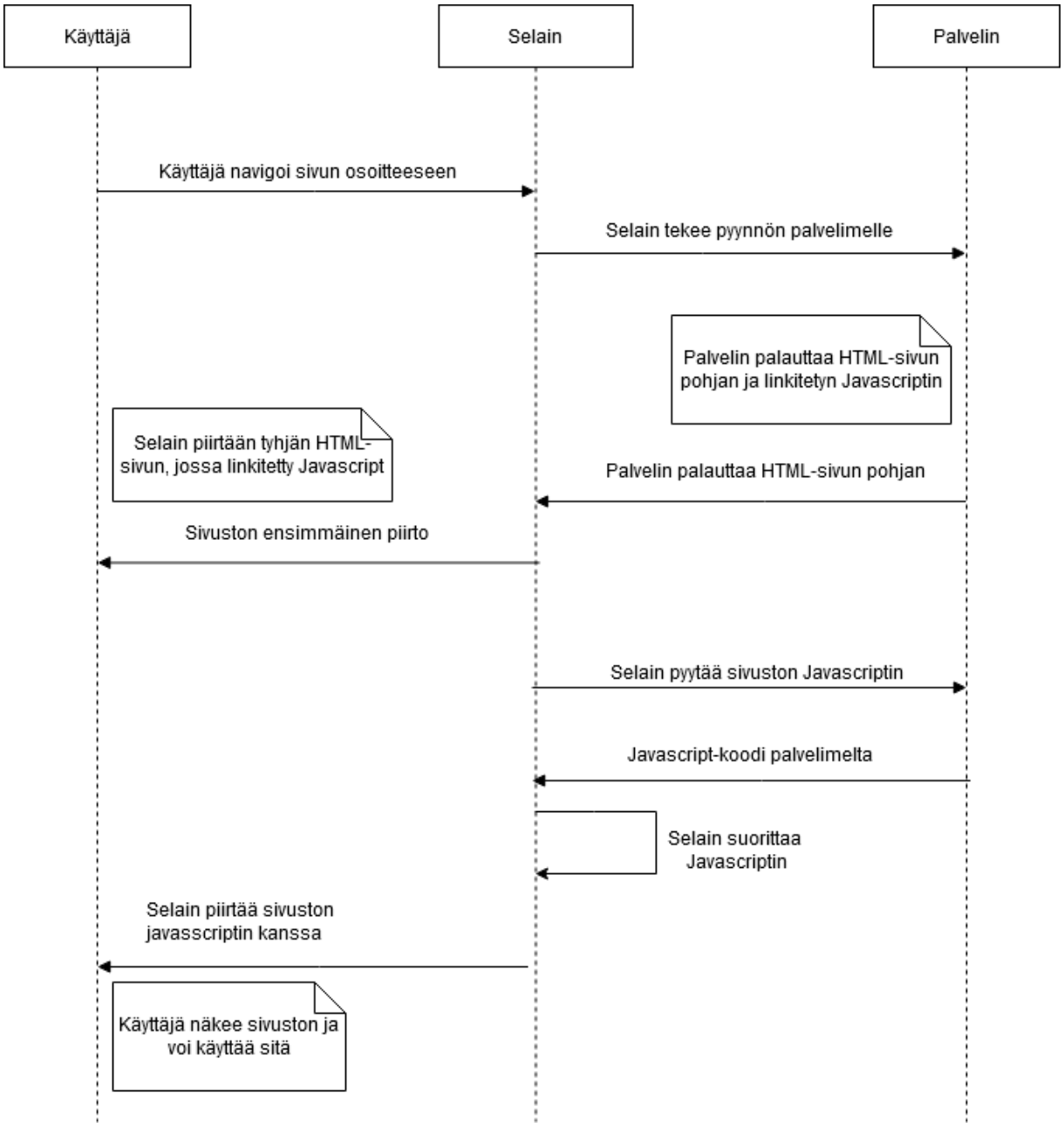
Tämän osion tarkoituksena luoda ja antaa käsitys siitä miten asiakaspuolen sekä palvelinpuolen renderöinti toimii. Tarkoituksena on myös antaa ymmärrystä siitä milloin ja mihin tarkoitukseen nämä erilaiset teknologiat soveltuvat.

2.1 Asiakaspuolen renderöinti

Asiakaspuolen renderöinti (engl. client-side rendering) on verkkosivujen renderöintitapa, jossa sivuston renderöinti tapahtuu sivuston käyttäjän eli asiakkaan selaimessa. Tämä renderöintitapa on yleistynyt nykypäiväisten kehityskehyksien ansiosta. (Danilec, 2020)

Kuvasta 1 voi nähdä kuinka asiakaspuolen renderöinti toimii käytännössä. Ensimmäisenä verkkosivunkäyttäjä lähettää selaimensa avulla pyynnön palvelimelle. Palvelin tämän jälkeen palauttaa selaimelle tyhjän HTML-tiedoston, johon on linkitetty tarpeellinen Javascript-koodi sivuston piirtämistä varten. HTML-tiedoston saatuaan selain pyytää tämän linkitetyn Javascript-koodin avulla palvelimelta kaiken tarpeellisen sivuston piirtämistä varten, jonka jälkeen selain suorittaa tämän Javascript-koodin. Javascript-koodin suorittamisen jälkeen selain piirtää käyttäjälle sivuston ja käyttäjä voi tämän jälkeen nähdä sivuston ja käyttää sivuston toiminnallisuuksia. (Danilec, 2020)

Kuva 1 Sekvenssikaavio asiakaspuolen renderöinnistä (Danilec, 2020).



Asiakapuolen renderöinti kannattaa toteuttaa verkkosivustolla silloin kuin halutaan luoda verkkosivu, jossa on paljon erilaisia toiminnallisuuksia ja vaativa käyttöliittymä. Asiakaspuolen renderöinnin hyödyiksi voidaan katsoa sen nopea renderöinti ensimmäisen sivun renderöinnin jälkeen, vaativan käyttöliittymän hyödyntäminen ja sen toimivuus erilaisten toiminnallisuuksien kanssa kuten esimerkiksi käyttäjän auktorisointi sivustolle.

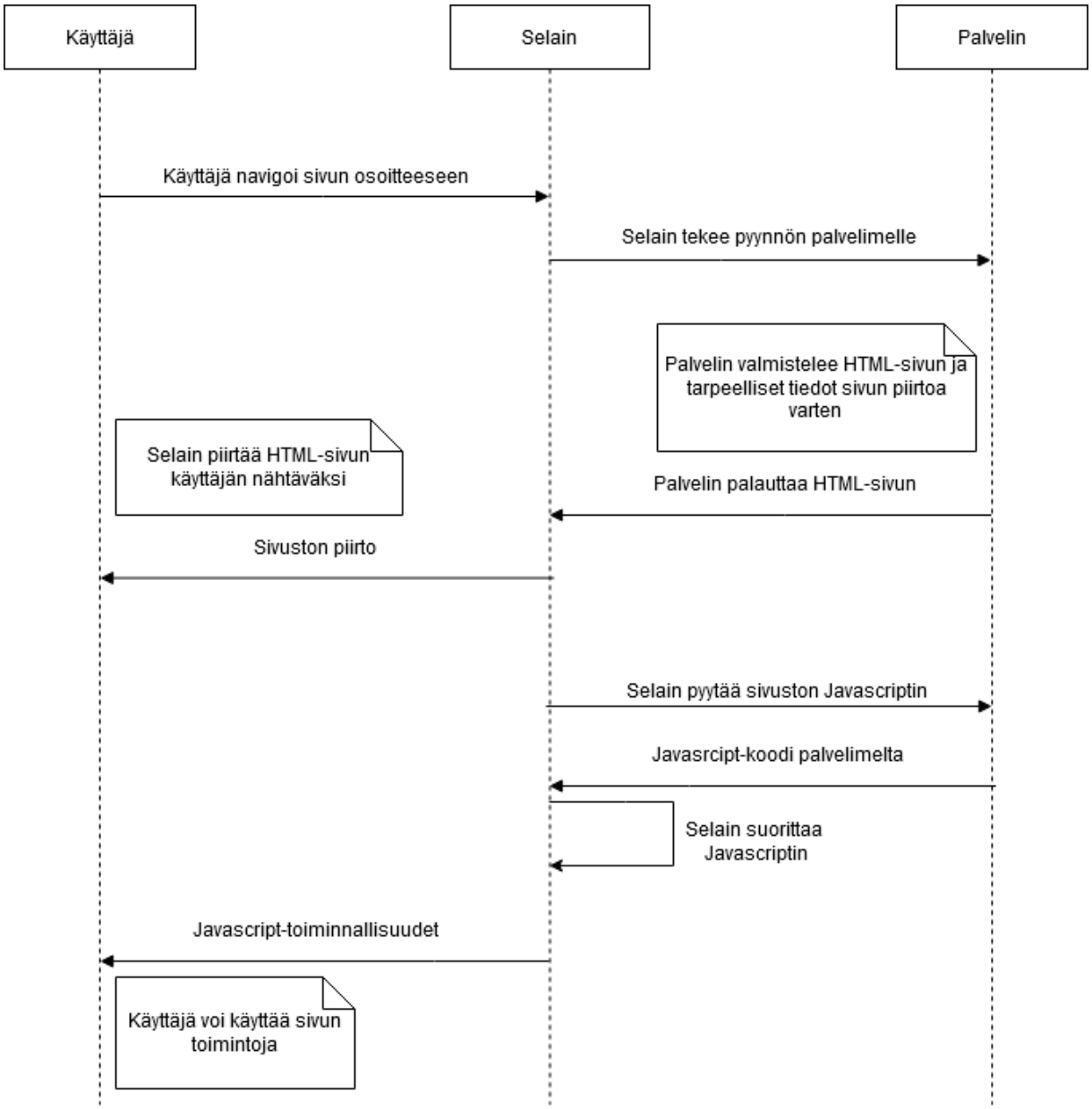
Asiakaspuolen renderöinnin haitoiksi voi osoittautua sivuston hidas ensimmäinen lataus ja sivuston huono hakukoneoptimointi (engl. Search Engine Optimization). (Danilec, 2020)

2.2 Palvelinpuolen renderöinti

Palvelinpuolen renderöinti (engl. server-side rendering) on verkkosivujen renderöintitapa, jossa sivun renderöinti nimensä mukaisesti tapahtuu palvelimen puolella. Palvelinpuolen renderöinnissä, kun käyttäjä pyytää verkkosivua palvelimelta palvelin valmistelee HTML-tiedoston ja lähettää tämän käyttäjän selaimen tulkattavaksi. (Baker, 2021)

Kuvasta 2 voi nähdä kuinka palvelinpuolen renderöinti toimii käytännössä. Prosessi alkaa, kun käyttäjä selaimellaan lähettää pyynnön palvelimelle verkkosivustolle pääsystä. Tämän jälkeen palvelin valmistelee HTML-tiedoston, joka lähetään käyttäjän selaimelle tulkattavaksi. HTML-tiedoston mukana tulee myös sivuston mahdolliset tarvittavat Javascript-koodit. Selaimen tulkkauksen jälkeen sivusto on jo käyttäjän nähtävissä, mutta kuitenkaan sen toiminnallisuudet eivät ole. Samalla hetkellä, kun selain jo näyttää sivustoa käyttäjällä selain suorittaa HTML-tiedoston mukana saatuja Javascript-koodeja. Javascript-koodien suorittamisen jälkeen sivuston toiminnallisuudet ovat käyttäjän käytettävissä. Palvelinpuolen renderöinnissä tämä prosessi toteutuu aina kun käyttäjä siirtyy tämän palvelun sisällä sivustolta toiselle. (Baker, 2021)

Kuva 2 Sekvenssikaavio palvelinpuolen renderöinnistä (Danilec, 2020).



Palvelinpuolen renderöinnin eduiksi katsotaan sen oleminen mahdollisimman hakukoneoptimisoitu ja sen nopeus sivuston ensimmäisessä latauksessa. Palvelinpuolen renderöintiä kannattaa yleisesti käyttää sivustoissa, jotka ovat staattisia eivätkä vaadi vaativia toiminnallisuuksia. Palvelinpuolen renderöinnin heikkoutuna on sen tapa aina uuden sivuston pyynnön kohdalla suorittaa renderöinti prosessi kokonaan uudelleen. (Baker, 2021)

3 Rust-ohjelmointikieli

Rust-ohjelmointikielen kehitys on aloitettu vuonna 2006 kun Mozilla Foundation kehittäjät törmäsivät ongelmiin C++-kielen kanssa. Heidän päätuotteensa Firefox-verkkoselain oli kehitetty C++-kielellä ja tuotteen jatkokehittäminen C++-kielellä oli koettu liian haasteelliseksi. Rust-ohjelmointikielen tarkoituksena on korjata C ja C++ -kielten mahdolliset ongelmat. Nykypäivänä Rust-ohjelmointikieli on levinnyt Mozilla Foundation ulkopuolelle ja suurin osa kielen kehittäjistä onkin Mozillan ulkopuolisia henkilöitä. Rust on myös levinnyt muiden yritysten käyttöön kuten esimerkiksi Dropbox-pilvitiedostopalvelu ja Cloudflare-pilvipalvelu käyttävät Rust-ohjelmointikieltä palveluissaan. (Vainio, 2019)

3.1 Rust-ohjelmointikielen ominaisuuksia

Rust-ohjelmointikielessä on muutamia sille tyypillisiä ominaisuuksia, joita ei yleisesti kaikista ohjelmointikielistä ole. Muutama esimerkki niistä on sen omistajuus ja muuttujien automaattinen muuttumattomuus. Näistä ja muista Rust-ohjelmointikielelle tyypillisistä ominaisuuksista enemmän tässä osiossa.

3.1.1 Tyypitys

Jokaisella arvolla Rust-ohjelmointikielessä on jonkinlainen tyypitys. Tyypityksen tarkoituksena on kertoa määritetyn arvon tyyppi ja miten sen kanssa tulisi työskennellä. Rust on staattisesti kirjoitettu ohjelmointikieli, joten sen täytyy tietää kaikkien arvojen tyypit ennen kuin sen ohjelmakoodia ollaan kääntämässä. Rust-ohjelmointikielessä numeraalisien arvojen tyypit voivat olla etumerkillisiä (engl. signed) tai etumerkittömiä (engl. unsigned). Näillä arvoilla on myös määritelty suuruus. Esimerkiksi jos numeraalisen arvon tyypiksi on määritelty etumerkillinen ”i8” voi sen minimi arvo olla -128 ja sen maksimi arvo voi olla 127. Kun taas jos numeraalisen arvon tyypiksi on määritelty etumerkitön ”u8” voi sen minimi arvo olla 0 ja maksimiarvo olla 255. Taulukosta 1 voi nähdä miten tätä on toteutettu Rust-ohjelmointikielessä. (Klabnik & Nichols, 2022b)

Taulukko 1 Numeraalisien arvojen tyypitys

Etumerkillinen (signed)		
tyyppi	minimi	maksimi
i8	-2^{7}	2^{7} -1
i16	-2^{15}	2^{15} -1
i32	-2^{31}	2^{31} -1
i64	-2^{63}	2^{63} -1
i128	-2^{127}	2^{127}-1

Etumerkitön (unsigned)		
tyyppi	minimi	maksimi
u8	0	2^{8} -1
u16	0	2^{16} -1
u32	0	2^{32} -1
u64	0	2^{64} -1
u128	0	2^{128}-1

3.1.2 Omistajuus

Omistajuus on yksi Rust-ohjelmointikielen uniikeista ominaisuuksista. Rust-ohjelmat käyttävät omistajuutta hallitsemaan ohjelman muistia. Yleisesti ohjelmointikielet käyttävät automaattista roskien keräilyä (engl. carbage collector) muistin vapauttamiseksi ohjelmissa tai sitten muistia joudutaan määrittämään ja vapauttamaan itse. Rust-ohjelmointikielessä tämä muistinhallinta on toteutettu omistajuusjärjestelmällä. Jos ohjelman kääntämisvaiheessa jokin ohjelman osa rikkoo omistajuusjärjestelmää, ohjelma ei suostu kääntämään sitä. Rust-ohjelmointikielen omistajuusjärjestelmään kuuluu säännöt: jokaisella arvolla on omistaja, jokaisella arvolla voi olla vain yksi omistaja kerrallaan ja kun arvon omistaja menee ohjelman rajojen (engl. scope) ulkopuolelle, arvo tiputetaan pois muistista. Rust-ohjelmointikielessä arvojen omistajuuden siirtäminen on toteutettu lainaamisella (engl. borrowing). Lainaamisessa arvon omistajuus pysyy nykyisellä omistajalla, mutta lainaajalla arvo näkyy viitteenä (engl. reference). Ohjelmakoodissa 1 käytetään merkkiä ”&” lainauksen toteuttamiseksi, joka on Rust-ohjelmointikielen tapa merkata lainaus. (Klabnik & Nichols, 2022e)

Ohjelmakoodi 1 Omistajuuden siirtäminen Rust-ohjelmointikielessä

```
fn say_hello_world(string: &String) {
    println!("Hello {}!", string);
}

fn main() {
    let s = String::from("World");
    say_hello_world(&s);
}
```

3.1.3 Muuttumaton tieto

Rust-ohjelmointikielessä kaikki muuttujat ovat automaattisesti muuttumattomia (engl. immutable). Tämä on tarkoituksella tehty sen takia, jotta voitaisiin hyödyntää Rust-ohjelmointikielen turvallisuutta. Rust-ohjelmointikielessä muuttujan voi muuttaa muuttuvaksi lisäämällä muuttujan eteen sanan ”mut” (Ohjelmakoodi 2). Rust-ohjelmointikielen kääntäjä kääntämisvaiheessa kuitenkin ehdottaa mahdollisista halutuista muuttujien muutoksista virheen muodossa, jos niitä ei ole tehty. Tämä auttaa mahdollisten ongelmien löytämisessä ohjelmakoodista ja estää ei toivottuja ongelmia syntymästä ohjelmaan.(Klabnik & Nichols, 2022d)

Ohjelmakoodi 2 Muuttuva ja muuttumaton muuttuja Rust-ohjelmointikielessä

```
fn main() {
    let mut x:i8 = 5;
    let y:i8 = 6;
}
```

3.1.4 Pakettien hallinta

Rust-ohjelmointikielessä pakettien hallitsemiseen käytetään työkalua nimeltä Cargo. Cargo työkalua voi käyttää niin projektin luomisesta aina projektin rakentamiseen asti. Cargo työkalun käyttämiä paketteja kutsutaan nimellä laatikko (engl. crate) ja näitä laatikoita Cargo käyttää projektin rakentamisessa. Näitä paketteja Rust-ohjelmointikielen ohjelmoitsijat voivat hakea Rust yhteisön pakettirekisteristä crates.io.(*The Cargo Book*, n.d.) Projektia rakentaessa Cargo tarkistaa laatikon riippuvuudet ja tiedot ”Cargo.toml ”nimisestä tiedostosta projektin sisältä. Tämän Cargo.toml-tiedoston voi luoda itse projektin sisälle tai sitten luoda sen automaattisesti käyttäen Cargo:n projektin luomiskomentoa (Komento 1).

Komento 1 Paketinhallintatyökalu Cargo:n projektin luomiskomento

```
cargo new hello_world
```

Cargo työkalun hyötyihin kuuluu sen mahdollisuus luoda ja rakentaa projektin mille tahansa käyttöjärjestelmällä samoilla komennoilla. Tämä helpottaa ohjelmoitsijoiden mahdollisuuksia rakentaa Rust-sovelluksia millä tahansa käyttöjärjestelmällä. (Klabnik & Nichols, 2022c)

3.2 Rust-ohjelmointikielen hyödyt

Tämän osion tarkoituksena on kertoa Rust-ohjelmointikielen ominaisuuksien tuomista hyödyistä, ja siitä minkälaisen ohjelmointiympäristön ne luovat Rust-ohjelmointikieleen.

3.2.1 Muistinhallinta

Rust-ohjelmointikielen omistajuuden hallinta luo omaperäisen, mutta turvallisen ympäristön Rust-ohjelmointikieleen. Rust-ohjelmointikielen omistajuusjärjestelmä mahdollistaa ohjelmointikielen huolehtivan tiedosta silloin kun se on ohjelmalle tarpeellinen ja vapauttaa tiedon muistista silloin sitä ei enää tarvita. Tämä järjestelmä estää ohjelmaan syntymästä mahdollisia muistin korruptoitumisia (engl. memory corruption). Se myös estää ohjelmaa säilyttämästä tarpeetonta tietoa, joka voisi aiheuttaa ohjelmaan muistivuotoja (engl. memory leak). (Vainio, 2019)

3.2.2 Luotettavuus

Yhtenä Rust-ohjelmointikielen hyötynä pidetään sen luotettavuutta. Rust-ohjelmointikielen luotettavuus tulee sen kääntäjästä (engl. compiler), jonka tekeminen on vaatinut paljon vaivaa. Kääntämisvaiheessa kääntäjä tarkistaa ohjelmakoodin kokonaan ja ei käännä sitä, jos siitä löytyy kielen sääntöjä rikkovia kohtia. Kääntäjä ei kuitenkaan kerro vain virheestä vaan myös kertoo mahdollisesta tavasta, jolla voi korjata tämän virheen (Kuva 3). Kääntäjä ei myöskään ilmoita vain virheistä vaan se kertoo myös ohjelmakoodin käyttämättömistä osista. (Kuva 4). Luotettavuuden pohjana Rust-ohjelmointikielessä toimii sen omistajuusjärjestelmä ja arvojen vahva tyypitys.(Klabnik & Nichols, 2022a)

Kuva 3 Rust-ohjelmointikielen kääntäjän virheilmoitus.

```
error: literal out of range for `i8`
 --> src\main.rs:2:17
  |
2 |     let x: i8 = 500;
  |                 ^^^
  |
  = note: the literal `500` does not fit into the type `i8` whose range is `-128..=127`
  = help: consider using the type `i16` instead
  = note: `#[deny(overflowing_literals)]` on by default

error: could not compile `testi` due to previous error
```

Kuva 4 Rust-ohjelmointikielen kääntäjän ilmoitus käyttämättömästä muuttujasta

```
warning: unused variable: `x`
 --> src\main.rs:3:9
  |
3 |     let x: i16 = 500;
  |         ^ help: if this is intentional, prefix it with an underscore: `_x`
  |
  = note: `#[warn(unused_variables)]` on by default

warning: `testi` (bin "testi") generated 1 warning
```

3.3 Rust-ohjelmointikieli ja WebAssembly

Rust-ohjelmointikielestä on mahdollista tuottaa WebAssembly-koodia, joka mahdollistaa Rust-ohjelmointikielen sovellusten suorittamisen moderneissa verkkoselaimissa. Javascript-kielen ollessa yksi suosituimmista verkkosovellusten kehityskielistä WebAssemblyn tarkoitus ei ole korjata Javascript-ohjelmointikieltä, vaan sen on tarkoitus toimia yhteistyössä tämän kanssa luoden uusia mahdollisuuksia web-kehittäjille ja paikata Javascript-kielen heikkouksia. WebAssembly on assembly-kielen kaltainen binäärimuodossa oleva ohjelmointikieli, joka antaa mahdollisuuden toteuttaa ohjelmointikieliä kuten C/C++, C# ja Rust verkkoselaimessa. (MDN contributors, 2022)

4 Projektin tarkoitus ja suunnittelu

Opinnäytetyöni projektin tarkoituksena on kehittää yksinkertainen verkkosivu käyttäen Rust-ohjelmointikielelle kehitettyjä sovelluskehyksiä Yew ja Actix web. Projekti kehitetään hyödyntäen asiakas-palvelin-arkkitehtuuria projektin rakenteessa. Yew-sovelluskehyksen tarkoituksena projektissa on luoda asiakaspuolen ohjelma ja tarkoituksena on hyödyntää Yew-sovelluskehyksen tarjoamaa palvelinpuolen renderöinti toiminnallisuutta. Actix web -sovelluskehyksen tarkoituksena on luoda projektiin http-palvelin, joka mahdollistaa asiakaspuolen sovelluksen jakamisen verkkoselaimeen. Projektissa luotua yksinkertaista verkkosivua testataan hyödyntäen Googlen kehittämää Lighthouse-työkalua, jonka tarkoituksena on testata verkkosivun suorituskyvykkyyttä. Projektin verkkosivun testaamisen tarkoituksena on saada tietoa Rust-ohjelmointikielellä kehitetyn verkkosivun suorituskyvykkyydestä, jotta sen suorituskykyä voitaisiin verrata Javascript-ohjelmointikielellä kehitetyn verkkosivupalvelun suorituskykyyn. Projektin tarkoituksesta ja osien suunnittelusta enemmän tässä osiossa.

4.1 Actix web

Projektin Palvelinpuolen suorittamiseen valikoin Actix web nimisen sovelluskehyksen. Actix web on yksi kehitetyimmistä sovelluskehyksistä Rust-ohjelmointikielelle, joka mahdollistaa http-palvelimen luonnin Rust-ohjelmointikielellä (Ohjelmakoodi 3). Actix web -sovelluskehystä voisi verrata Javascript-pohjaiseen express.js nimiseen sovelluskehykseen.

Ohjelmakoodi 3 HTTP-palvelimen luonti Actix web -sovelluskehyksellä

```
use actix_web::{get, App, HttpResponse, HttpServer, Responder};

#[get("/")]
async fn hello() -> impl Responder {
    HttpResponse::Ok().body("Hello world!")
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .service(hello)
    })
    .bind(("127.0.0.1", 8080))?
    .run()
    .await
}
```

Actix web on sovelluskehys, joka on kehittynyt Actix nimisestä sovelluskehyksestä Rust-ohjelmointikielelle. Suurimmaksi osaksi Actix web -sovelluskehyksestä ei ole mitään tekemistä enää Actix sovelluskehyksen kanssa, mutta jotkin sen osat vielä vaativat Actix-sovelluskehyksellä ominaisuuksia. Actix web -sovelluskehyksellä kehitetyt sovellukset luovat aina http-palvelimen, jonka on mahdollista käyttää HTTP/2-protokolla tai TLS-protokollan käytön. (Actix Team, 2023)

4.2 Yew

Yew on Rust-ohjelmointikielellä kehitetty sovelluskehys, jonka tarkoituksena on tuottaa asiakaspuolen verkkosovelluksia. Yew-sovelluskehys usein verrataan Javascript-maailman React-kirjastoon sen samanlaisten toiminnallisuuksien takia. Yew-sovelluskehys on myös komponenttipohjainen niin kuin React ja Yew-sovelluskehyksessä on myös HTML-makro niminen komponentti, joka mahdollistaa HTML-koodin kirjoittamisen Rust-ohjelmointikielen sisällä (Ohjelmakoodi 4). Tätä HTML-makroa usein verrataan React-kirjaston JSX-elementtiin, joka mahdollistaa HTML-koodin kirjoittaminen Javascript-kielen sisällä. (*Yew Documentation*, 2022)

Ohjelmakoodi 4 Yew-sovelluskehyksen HTML-makro komponentti

```
#[function_component]
fn App() -> Html {
    return html! {
        <>
         <div>
            <p> {"Hello world"} </p>
         </div>
        </>
    };
}
```

Yew-sovelluskehyksellä kehitetyt sovellukset oletukseltaan käyttävät asiakaspuolen renderöinti toiminallisuusta, mutta Yew-sovelluskehyksestä löytyy myös palvelinpuolen renderöinnin toiminnallisuus. Palvelinpuolen renderöinnin toiminnallisuuden takia valitsin Yew-sovelluskehyksen toimimaan projektin asiakaspuolen viitekehyksenä. Ohjelmakoodista

5 voi nähdä kuinka Yew-sovelluskehyksen palvelinpuolen renderöinti ominaisuutta voidaan käyttää.

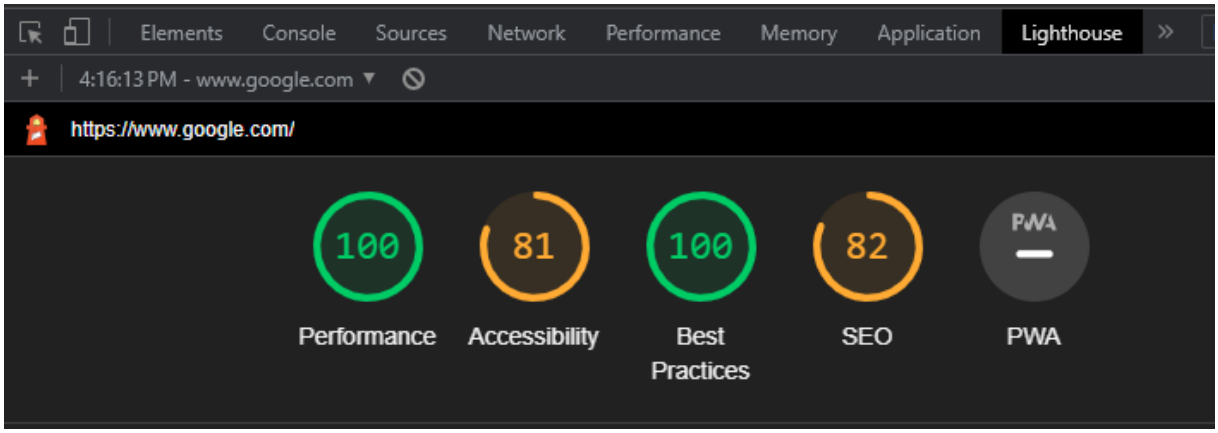
Ohjelmakoodi 5 Palvelinpuolen renderöinti Yew-sovelluskehyksessä (*Yew Documentation*, 2022).

```
let renderer = ServerRenderer::<App>::new();
let rendered = renderer.render().await;
```

4.3 Google Lighthouse

Projektin tutkimuksen tarkoituksena on saada tietoa siitä kuinka suorituskyvykkään verkkosivupalvelimen Rust-ohjelmointikielellä voi tuottaa. Verkkosivupalvelimen suorituskykyä mittaamaan valitsin Google Lighthouse nimisen työkalun. Lighthouse on googlen kehittämä avoimen lähdekoodin työkalun, jonka tarkoituksena on parantaa verkkosivujen laatua. Lighthouse mittaa verkkosivujen laatua suorituskyvyn, saavutettavuuden, parhaiden käytäntöjen ja hakukoneoptimoinnin mukaan (Kuva 5). Lighthouse-työkalua voi käyttää Google Chrome -selaimen kehittäjätyökaluista tai sitten työkalun asentamisen jälkeen komentorivistä (*Google Lighthouse*, 2022a).

Kuva 5 Google Lighthouse -työkalun tulokset



Projektini tarkoituksena on tutkia verkkosivupalvelimen suorituskykyä, joten Lighthouse-työkalun suorituskykyosio on näistä tärkein. Lighthouse-työkalu käyttää kuutta eri metriikkaa verkkosivujen suorituskyvyn mittaamiseen (Kuva 6). Nämä kuusi ovat:

- First Contentful Paint (FCP) eli ensimmäinen sisällön piirto. Tämä arvo mittaa kuinka kauan verkkosivuston ensimmäisen tekstin tai kuvan piirtämiseen kestää. Tämän piirron arvoa mitataan sekunneissa. (*Google Lighthouse*, 2022b)

- Speed Index (SI) eli nopeus indeksi. Arvo mittaa kuinka nopeasti sivuston sisältö latautuu selaimeen. Lighthouse-työkalu nappaa videon sivuston latautumisesta ja laskee kehityksen tämän videon kuvien välillä. Arvo mitataan sekunneissa.(*Google Lighthouse*, 2022b)

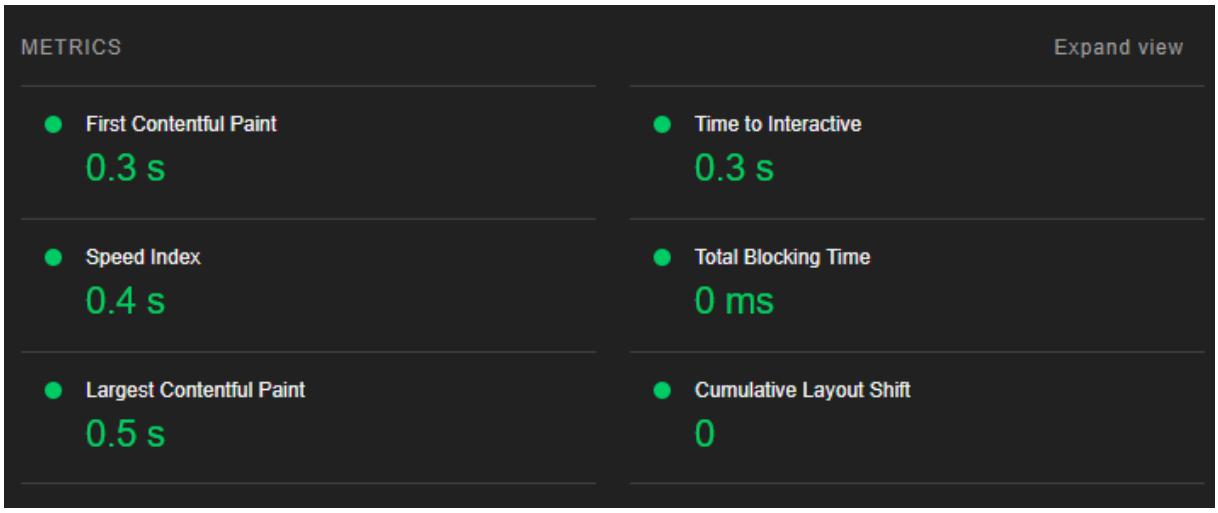
- Time to Interactive (TTI) eli aika toiminnallisuuteen. Arvo mittaa kuinka kauan sivuston latautumisessa kestää, että sen toiminnallisia elementtejä voidaan käyttää. Arvo mitataan sekunneissa. (*Google Lighthouse*, 2022b)

- Total Blocking Time (TBT) eli estetty aika. Tämä arvo mittaa kuinka kauan sivuston latautumisessa kestää, että sivuston käyttäjä voi painaa hiirensä näppäintä sivustolla tai sitten kirjoittaa näppäimistöllä. Arvo mitataan usein millisekunneissa. (*Google Lighthouse*, 2022b)

- Largest Contentful Paint (LCP) eli suurin sisällön piirto. Tämä arvo mittaa aikaa sekunneissa, kuinka kauan sivuston suurimman kuvan tai tekstin piirtämiseen kestää aikaa. (*Google Lighthouse*, 2022b)

- Cumulative Layout Shift (CLS) eli kumulatiivinen ulkoasun muutos. Tämä arvo mittaa suurinta ulkoasun muutosta yhteen laskettuna verkkosivuston elinaikana. Tähän arvoon voi vaikuttaa esimerkiksi kuvat tai mainokset sivustolla, joille ei ole merkattu suuruus arvoja oikein. (Walton, 2022)

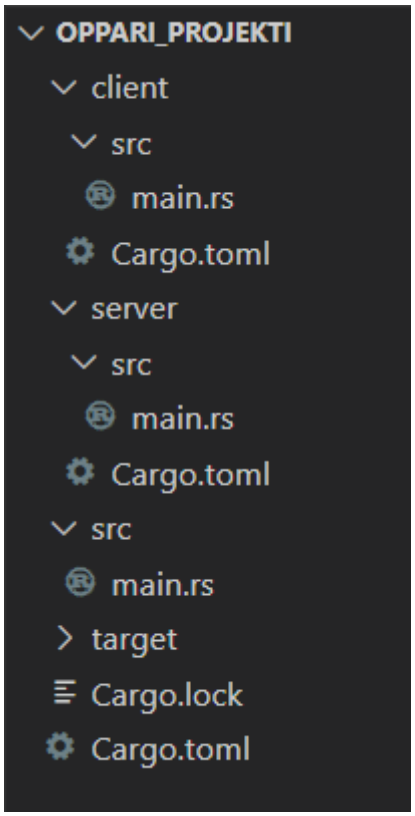
Kuva 6 Lighthouse-työkalun suorituskykyosion esimerkki tulokset



5 Verkkosivupalvelin Rust-ohjelmointikielellä

Tämän osion tarkoituksena on kertoa, miten luoda verkkosivupalvelin Rust-ohjelmointikielellä ja miten siihen luodaan palvelinpuolen renderöinnin toiminallisuus. Projektin tarkoituksena ei ole kertoa, miten Rust asennetaan vaan se olettaa, että Rust-ohjelmointikielen kääntäjä on jo asennettu järjestelmääsi. Projektin aikana on käytetty Rust-ohjelmointikielen versiota 1.66.0 ja kehitysympäristönä on käytetty Visual Studio Code -tekstieditoria. Käyttäen Rust-ohjelmointikielen pakettienhallintatyökalua Cargo:a projekti luotiin käyttämällä komentoa ”cargo new projektinnimi”. Jotta projektiin saatiin asiakas-palvelin-arkkitehtuuria käyttävä ohjelmistomalli siinä, käytetään Cargo:n komentoa vielä luomaan kaksi kansiota ”client” ja ”server” projektin sisälle, jotka tulevat erottamaan ohjelman eri osiot toisistaan. ”Client” kansion sisälle luodaan asiakaspuolen sovellus, jonka tarkoituksena on luoda kaikki asiakaspuolen elementit kuten HTML-elementit. ”Server” kansioon tehdään ohjelma, jonka tarkoituksena on luoda http-palvelin ja renderöidä asiakaspuolen ohjelma. Kuvasta 7 voidaan nähdä miltä projektin kansio rakenne näyttää tiedostojen luonnin jälkeen.

Kuva 7 Projektin kansio rakenne eri osioiden luomisen jälkeen



5.1 Asiakaspuolen toteutus

Asiakaspuolen sovelluskehyksenä projektissa toimi Rust-ohjelmointikielelle tehty Yew-sovelluskehys. Projektissa käytetyn Yew-sovelluskehyksen version oli 0.20 ja tämä versio on lisätty asiakaspuolen ohjelman Cargo.toml tiedoston riippuvuuksiin (Ohjelmakoodi 6). Yew-viitekehyksen dokumentaatio suositteli käyttämään Trunk nimistä työkalua Yew-ohjelman rakentamiseen, joten sitä on käytetty tässä projektissa. Trunk-työkalu asennettiin käyttämällä komentoa ”cargo install trunk”.

Ohjelmakoodi 6 Asiakaspuolen ohjelman Cargo.toml tiedoston riippuvuudet

```
[package]
name = "client"
version = "0.1.0"
edition = "2021"

[dependencies]
yew = { version = "0.20.0" , features = ["csr", "hydration"] }
```

Trunk on Rust-ohjelmointikielelle tehty WebAssembly verkkosovellusten rakentajatyökalu. Sen tarkoituksena on rakennuttaa kaikki tarvittavat resurssit kuten WebAssembly-, Javascript-, kuva- ja CSS-tiedostot ohjelman lähde HTML-tiedostoon. Tämä lähde HTML-tiedosto pitää itse luoda ohjelman sisälle ja sen nimi pitää olla projektissa index.html. (*Trunk*, 2021). Ohjelmakoodista 7 voidaan nähdä miltä tämä lähde HTML-tiedosto voi esimerkiksi näyttää.

Ohjelmakoodi 7 Lähde HTML-tiedosto trunk-työkalulle

```
<html>
<head>
    <title>Server-side render App</title>
    <link data-trunk rel="css" href="style.css" />
    <link data-trunk rel="rust" data-bin="client" />
    <link data-trunk rel="icon" href="icon.png"/>
</head>
</html>
```

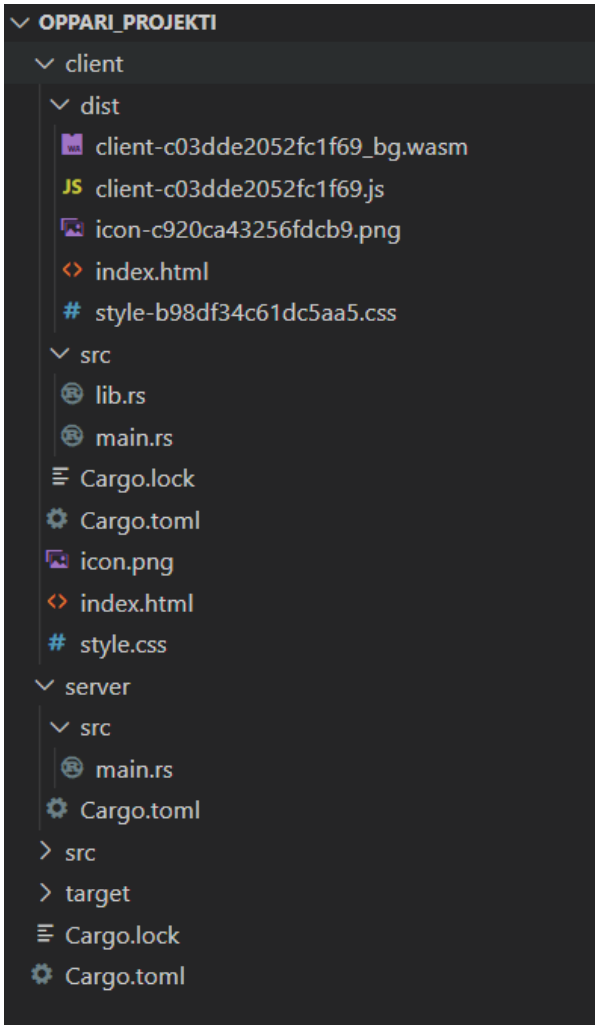
Asiakaspuolen kansioon ”src” luotiin myös uusi tiedosto nimeltä lib.rs. Tämän lib.rs tiedoston tarkoituksena on sisältää asiakaspuolen ohjelman käyttämä Yew-sovelluskehyksellä tehty HTML-makro komponentti (Ohjelmakoodi 4). Asiakaspuolen ohjelman main.rs tiedostoon on luotu metodi, jonka tarkoituksena on renderöidä tämä haluttu HTML-makro komponentti lib.rs tiedostosta (Ohjelmakoodi 8).

Ohjelmakoodi 8 Asiakaspuolen ohjelman main.rs tiedosto

```
use client::App;

fn main() {

    yew::Renderer::<App>::new().hydrate();

}
```

Tiedostojen valmistelun jälkeen oli asiakaspuolen ohjelma valmis rakennettavaksi Trunk-työkalulla. Ohjelman rakentaminen suoritettiin käyttämällä Trunk-työkalun komentoa ”trunk build”. Tämä komento luo lähde HTML-tiedostosta uuden HTML-tiedoston, johon on linkitetty kaikki ohjelmalle tarpeelliset tiedostot kansion ”dist” alle. Kuvasta 8 voidaan katsoa miltä projektin kansio rakenne näyttää asiakaspuolen ohjelman rakennuttamisen jälkeen.

Kuva 8 Projektin kansio rakenne asiakaspuolen ohjelman rakentamisen jälkeen



5.2 Palvelinpuolen toteutus

Palvelinpuolen sovelluskehyksenä toimi Actix web ja sen versio 4.0.0. Palvelinpuolelle on myös lisätä Yew-sovelluskehyksestä sama version kuin asiakaspuolelle, jotta sen palvelinpuolen renderöintiä voidaan hyödyntää. Ohjelmakoodista 9 voidaan nähdä kaikki tarpeelliset riippuvuudet palvelinpuolen suorittamiseksi.

Ohjelmakoodi 9 Palvelinpuolen Cargo.toml tiedosto

```
[package]
name = "server"
version = "0.1.0"
edition = "2021"

[dependencies]
yew = { version = "0.20.0" , features = ["ssr"]}
client = { path = "../client" }
actix-web = "4.0.0"
actix-files = "0.6"
```

Riippuvuuksien lisäämisen jälkeen palvelinpuolelle on lisätty funktio, jonka tarkoituksena on vastata selaimen lähettämään GET-pyyntöön. Tämän GET-pyynnön vastauksen tarkoituksena on lähettää selaimelle takaisen palvelimen puolella renderöity HTML-tiedosto ja siihen linkitetyt tiedostot. Ohjelmakoodista 10 voidaan nähdä projektin mukaisen GET-pyynnön, jossa käytetään palvelinpuolen renderöintiä luomaan GET-pyyntöön vastaukseksi lähetettävä HTML-tiedosto.

Ohjelmakoodi 10 Palvelimen GET-pyyntöön vastaava funktio

```
#[get("/")]
async fn render_client_app(_req: HttpRequest) -> Result<HttpResponse,
Error> {
    let index_html:String =
fs::read_to_string("./client/dist/index.html").unwrap();

    let renderer:ServerRenderer::<ClientApp> =
ServerRenderer::<ClientApp>::new();

    let content:String = renderer.render().await;

    Ok(
        HttpResponse::Ok()
            .content_type("text/html; charset=utf-8")
            .body(index_html.replace("<body>", &format!("<body>{}",
content)))
    )
}
```

Jotta palvelinpuolelle saadaan HTTP-palvelin, on siihen lisätty ohjelmakoodi 3:n mukainen HTTP-palvelin ohjelma. Tähän HTTP-palvelimen ohjelmakoodiin on lisätty palveluksi ohjelmakoodi 10:ssä tehty palvelu. Tämän jälkeen palvelinpuolen ohjelma oli valmis ajettavaksi.

5.3 Palvelimen lopputulos

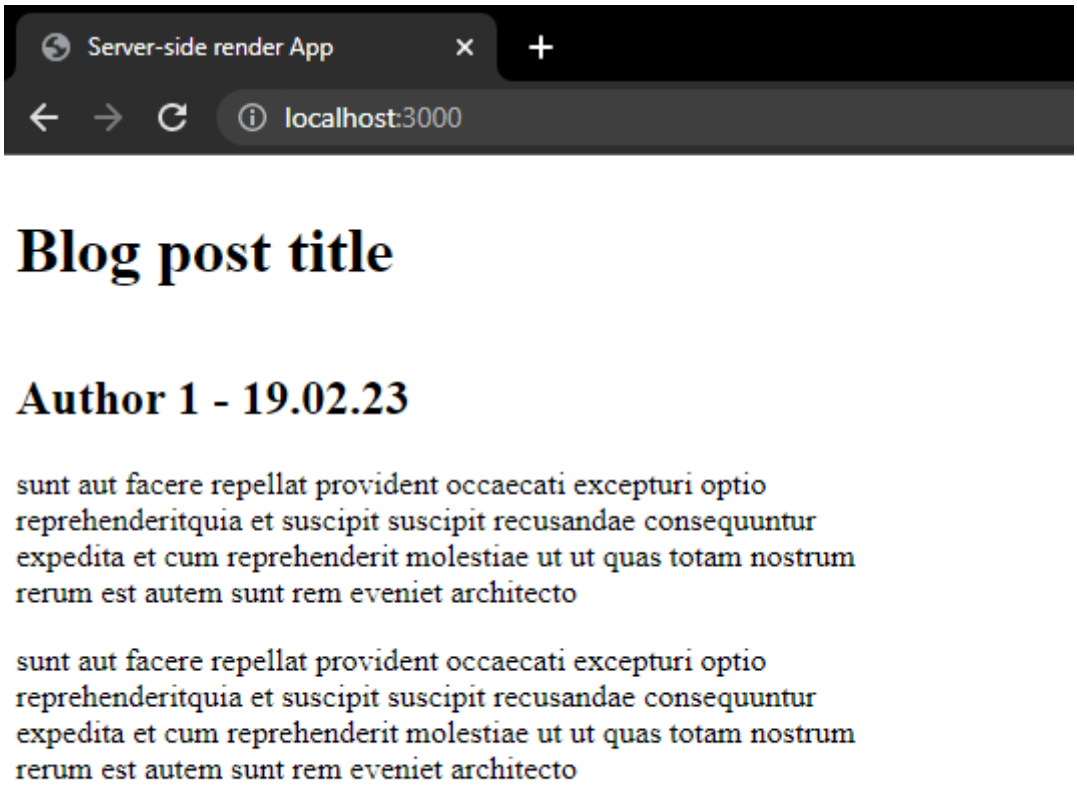
Ennen kuin projekti voidaan suorittaa, muokattiin vielä projektin Cargo.toml tiedostoa. Cargo.toml tiedostoon lisättiin projektiin kaikki itse luodut laatikot kuten client ja server. Server-laatikosta tehtiin projektin ensisijainen laatikko kuten ohjelmakoodissa 11. Tätä laatikkoa ohjelman suorittaminen käyttää niin sanotusti pääohjelmana.

Ohjelmakoodi 11 Projektin Cargo.toml tiedosto

```
[package]
name = "oppari_projekti"
version = "0.1.0"
edition = "2021"

[workspace]
members = ["client", "server"]
default-members = ["server"]
```

Laatikoiden lisäämisen jälkeen oli projekti valmis ajettavaksi. Projektin ajamisen suorittamiseksi käytin cargo työkalun komentoa ”cargo run”. Tällä komennolla saadaan ensin ladattua kaikki projektin riippuvuuksien laatikot kuten esimerkiksi Yew ja Actix web. Laatikoiden lataamisen jälkeen komento suorittaa ohjelmien koodit ja käynnistää HTTP-palvelimen. Projektin lopputuloksena syntyi yksinkertainen verkkosivu, joka palauttaa yksinkertaisen blogiverkkosivun palvelimen puolella renderöitynä (kuva 9). Sivustolle renderöidyt tiedot on haettu JSONplaceholder (*JSONPlaceholder*, 2022) nimestä API-palvelusta, joka on tarkoitettu testaamisen ja prototyyppien tekemiseen.

Kuva 9 Projektin tuloksena syntynyt yksinkertainen blogiverkkosivusto



6 Projektin tulokset ja johtopäätökset

Projektini tutkimuksen tarkoituksena on testata kuinka suorituskykyisen verkkosivupalvelimen Rust-ohjelmointikielellä voi tehdä. Palvelimen testaamisen on käytetty projektin suunnittelu -osiossa esiteltyä Google Lighthouse -työkalua. Tuloksien saamisen jälkeen tarkoituksena on verrata saatuja tuloksia samanlaisen projektin tuloksiin, jossa verkkosivupalvelin on toteutettu Javascript-ohjelmointikielellä. Verraten tuloksia toisen ohjelmointikielen tuloksiin saadaan parempi kuva siitä, kuinka suorituskyvykäs verkkosivupalvelin projektista syntyi.

6.1 Tulokset

Tuloksien saamiseksi on asennettu Google Chrome -selain, jotta sen Google Lighthouse-työkalua voitaisiin käyttää testaamisessa. Testaamiseen käytetty versio Google Chrome -selaimesta oli 109.0.5 ja Lighthouse-työkalun version oli 9.6.8. Verkkosivujen testaaminen Lighthouse-työkalulla on tehty hyvin helpoksi. Siihen vaaditaan vain testaamista varten tehty verkkosivu ja Google Chrome -selaimen kehittäjätyökalujen Lighthouse-työkalun testin ajamisen verkkosivua vastaan. Lighthouse-työkalun testit projektin tuloksissa on ajettu simuloiden pöytäkonetta. Lighthouse-työkalussa on myös mahdollisuus simuloida mobiililaitetta. Tuloksien oikeellisuuden takaamiseksi ajoin Lighthouse-työkalun testin muutamaan eri otteeseen. Taulukosta 2 nähdään Lighthouse-työkalun antamat tulokset projektissa kehitetylle verkkosivustolle. Lighthouse-työkalun metriikat ovat selitetty opinnäytetyön osiossa 4.3.

Taulukko 2 Google Lighthouse -työkalun tulokset projektin verkkosivustosta

Rust verkkosivu	
Metriikka	Tulos
First Contentful Paint	0,30 s
Speed Index	0,30 s
Time to Interactive	0,30 s
Total Blocking Time	0 ms
Largest Contentful Paint	2,00 s
Cumulative Layout Shift	0

6.2 Tuloksien vertailua

Tuloksien vertaamiseksi käytän Petri Paukkusen opinnäytetyössä nimeltä ”Next.js-verkkosivujen kehittämiseen” saatuja tuloksia. Tässä opinnäytetyössä käsiteltiin Next.js nimistä sovelluskehystä, jossa käytetään nimensä mukaisesti Javascript-ohjelmointikieltä. Tässä opinnäytetyössä kehitettiin yksinkertainen blogiverkkosivusto käyttäen palvelinpuolen renderöintiä, joten koin tämän hyväksi kohteeksi vertailulle ohjelmointikielten välillä. Paukkusen opinnäytetyössä käytettiin myös Google Lighthouse -työkalua verkkosivujen suorituskyvyn testaamiseen. Taulukosta 3 voi katsoa Paukkusen opinnäytetyössä saadut tulokset hänen kehittämälleen verkkosivustolle. Vertaillen taulukoiden 2 ja 3 tuloksia huomataan, että suurimmassa osassa metriikoista verkkosivustot ovat saaneet samanlaisia arvoja. Ainoa erovaisuus verkkosivustojen tuloksissa on metriikassa Largest Contentful Paint, jolla viitataan sivuston suurimman elementin piirtämiseen käytettyä aikaa sekunneissa.

Taulukko 3 Next.js verkkosivuston Lighthouse-työkalun tulokset (Paukkunen, 2022)

Javascript verkkosivu	
Metriikka	Tulos
First Contentful Paint	0,30 s
Speed Index	0,30 s
Time to Interactive	0,30 s
Total Blocking Time	0 ms
Largest Contentful Paint	0,57 s
Cumulative Layout Shift	0

6.3 Pohdintaa

Tuloksien vertailu osiossa huomattiin, että tuloksien ainoa eroavaisuus oli metriikassa Largest Contentful Paint. Rust-ohjelmointikielellä tuotetussa verkkosivustossa oli tässä metriikassa suurempi arvo, joka tarkoittaa, että Rust-ohjelmointikielellä tuotettu verkkosivusto latautui hitaammin kuin Javascript-ohjelmointikielellä tehty verkkosivusto. Largest contentful paint -metriikkaan vaikuttaa hyvin paljon se, kuinka kauan tiedoston, josta sivusto saa tietonsa kestää ladata. Rust-ohjelmointikielellä tuotetussa verkkosivustossa tämä tiedosto oli projektini tuottama WebAssembly-tiedosto. Päätin koittaa ajaa Lighthouse-työkalun testin uudelleen projektini sivustolle ilman että siihen tuli WebAssembly-tiedostoa. Poistamalla WebAssembly-tiedoston linkityksen projektini HTML-tiedostosta sain laskea projektini largets contentful paint metriikan alas 0,50 sekuntiin. Tällä todistaen, että WebAssembly-tiedosto oli suurin vaikuttava tekijä tähän metriikkaan projektissa. Poistaen WebAssembly-tiedoston projektista kuitenkin vaikutetaan sivuston toiminnallisuuksiin, jolloin sivusto onkin enää vain tekstiä sisältä HTML-tiedosto ilman mitään toiminnallisuuksia.

Projektin testaukset ovat myös suoritettu suotuisissa olosuhteissa. Palvelin mitä testattiin, sijaitsi samalla laiteella millä oli myös testausalusta. Näillä testaustuloksilla ei välttämättä saada todellista kuvaa siitä minkälainen o verkkosivuston suorituskyky olisi oikeassa tilanteessa. Google Lighthouse -työkalu käyttää simulaatiossaan aina tietynlaista internet-

yhteyden nopeutta ja prosessorin tehokkuutta, jolloin saadaan tuloksia verkkosivun suorituskyvystä näillä asetuksilla. Oikeassa tilanteessa kuitenkaan kaikilla verkkosivun käyttäjillä ei ole samanlainen internet-yhteys ja prosessori laitteessa, jolla he käyttävät sivustoa.

Näissä tuloksissa pitää myös ottaa huomioon se, että Rust-ohjelmointikielellä tuotetussa verkkosivustossa käytettiin Yew-sovelluskehystä, jonka palvelinpuolen renderöinti on vielä kokeiluvaiheessa heidän dokumentaationsa mukaisesti. Tuloksissa pitää myös ottaa huomioon Yew-sovelluskehyksen käyttämä WebAssembly on vielä uusi tekniikka tuottaa verkkosivustoja, kun taas suurin osa nykypäiväisistä verkkosivustoista käyttää Javascript-ohjelmointikieltä verkkosivujen toteuttamiseen.

7 Yhteenveto

Palvelinpuolen renderöinnissä sivuston renderöinti nimensä mukaisesti tapahtuu palvelimen puolella. Tässä tilanteessa sivuston käyttäjän selaimen ei tarvitse suorittaa verkkosivuston renderöintiä vaan se tapahtuu palvelimen puolella, kun käyttäjä on navigoinut verkkosivustolle. Palvelimen puolella renderöityjen verkkosivujen etuina on hakuoptimisointi ja nopeampi sivuston latautuminen.

Opinnäytetyön projektissa luotiin verkkosivusto käyttäen Rust-ohjelmointikieltä ja sen sovelluskehyksiä Yew ja Actix web. Projektissa tutkittiin kuinka suorituskyvykkään verkkosivun Rust-ohjelmointikielellä voi kehittää verraten sitä Javascript-ohjelmointikielellä toteutettuun verkkosivuun. Tuloksista huomattiin kuinka ohjelmointikielten välillä ei ollut suurta merkitystä verkkosivun suorituskyvykkyyteen.

Vaikka Rust-ohjelmointikielellä toteutettu verkkosivu jäi suorituskyvyllään Javascript verkkosivuston alle voi se silti olla hyvä vaihtoehto seuraavaan verkkosivu projektiin. Rust-ohjelmointikieli mahdollistaa suorituskyvykkään ja muistiturvallisen projektin rakentamisen. Rust on myös erittäin hyvä vaihtoehto, jos haluaa projektissaan hyödyntää WebAssembly-tekniikkaa.

Lähteet

Actix Team. (2023). *Actix web documentation*. The Actix Team. https://actix.rs/docs

Baker, O. (2021, July 8). Server-Side Rendering vs. Client-Side Rendering: Your Complete Guide. *Server-Side Rendering vs. Client-Side Rendering: Your Complete Guide*. https://devm.io/javascript/server-side-rendering-174672

Danilec, A. (2020, September 21). Client-Side Rendering or Server-Side Rendering—What Is the Best Solution for Your Next Application? *Client-Side Rendering or Server-Side Rendering - What Is the Best Solution for Your Next Application?* https://www.blog.duomly.com/client-side-rendering-vs-server-side-rendering-vs-prerendering/#intro-to-client-side-rendering-vs-server-side-rendering

Google Lighthouse Overview. (2022a). Chrome Developers. https://developer.chrome.com/docs/lighthouse/overview/

Google Lighthouse Performance Audits. (2022b). Chrome Developers. https://developer.chrome.com/docs/lighthouse/performance/

JSONPlaceholder. (2022). typicode. https://jsonplaceholder.typicode.com/

Klabnik, S., & Nichols, C. (2022a). *The Rust Programming Language Book*. https://doc.rust-lang.org/book/

Klabnik, S., & Nichols, C. (2022b). *The Rust Programming Language Book Data Types*. https://doc.rust-lang.org/book/ch03-02-data-types.html

Klabnik, S., & Nichols, C. (2022c). *The Rust Programming Language Book Hello Cargo*. https://doc.rust-lang.org/book/ch01-03-hello-cargo.html

Klabnik, S., & Nichols, C. (2022d). *The Rust Programming Language Book Variables and Mutability*. https://doc.rust-lang.org/book/ch03-01-variables-and-mutability.html

Klabnik, S., & Nichols, C. (2022e). *The Rust Programming Language Book What Is Ownership?* https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html

MDN contributors. (2022). *WebAssembly*. Mozilla, MDN Web Docs. https://developer.mozilla.org/en-US/docs/WebAssembly

Paukkunen, P. (2022). *Next.js-verkkosivujen kehittämiseen*. https://www.theseus.fi/handle/10024/780667

The Cargo Book, A book on Rust's package manager and build system. (n.d.). https://doc.rust-lang.org/cargo/index.html

Trunk. (2021). Trunk Maintainers. https://trunkrs.dev/

Vainio, J. (2019, March 11). Rust-ohjelmointikieli – C:n ja C++:n korvaaja? *Rust-Ohjelmointikieli – C:N Ja C++:N Korvaaja?* https://aiso.fi/2019/03/11/rust-ohjelmointikieli/

Walton, P. (2022, October 19). Cumulative Layout Shift (CLS). *Cumulative Layout Shift (CLS)*. https://web.dev/cls/

Yew documentation. (2022). https://yew.rs/

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyön projektin aikana tulevaa tietoa pidetään tekijän tietokoneella C-asemalla. Siitä säilytetään ajanmukainen varmuuskopio omalla henkilökohtaisella muistitikulla. Projektista saatua tietoa tullaan säilyttämään ainakin vuoden verran opinnäytetyön valmistumisen jälkeen. Opinnäytetyössä ei ole käsitelty luottamuksellista tietoa tai tietoa, jota tarvitsi tuhota. Teoriaosuuden lähteet on merkitty lähdelistaan.