jamk

WebAssemblyn käyttökohteet selainsovelluksissa

Toni Pekurinen

Opinnäytetyö, AMK
Lokakuu 2021
Liiketalouden ala
Tradenomi, tietojenkäsittelyn tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Pekurinen, Toni

WebAssemblyn käyttökohteet selainsovelluksissa

Jyväskylä: Jyväskylän ammattikorkeakoulu. Syyskuu 2021, 34 sivua.

Liiketalouden ala. Tradenomi, tietojenkäsittelyn tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Opinnäytetyössä tutkittiin WebAssemblyn käyttökohteita selainsovelluksissa. WebAssembly on melko uusi teknologia ja sen käyttökohteiden selvittäminen osoittautui ajankohtaiseksi. WebAssembly on tehty selaimissa ajettavaksi ja sen on kerrottu parantavan sovelluksien suorituskykyä. Tehtävänä oli ottaa selvää mikä WebAssembly on ja mikä on sen nykytilanne, sekä selvittää sen käytön etuja ja haittoja. Tavoitteena oli löytää mahdollisia sovelluskohteita ja saada kuva sen käytön nykytilanteesta. Opinnäytetyössä keskityttiin verkkoselaimessa toteutuvaan käyttöön.

Opinnäytetyö toteutettiin kvalitatiivisena tutkimuksena ja siinä käytettiin vertailua. Vertailua tehtiin WebAssemblyn ja JavaScriptin välillä. WebAssemblyn ja JavaScriptin vertailussa keskityttiin suorituskykyyn. Suorituskykyä näiden välillä mitattiin Fibonacci ja Matriisin kerronta -testeillä. Sovelluskehyksien vertailua tehtiin Blazor WebAssemblyn ja Angularin välillä. Sovelluskehyksien vertailussa kehitettiin vertailtavat asiakaspuolen sovellukset. Asiakaspuolen sovelluksien vertailussa keskityttiin toteutukseen, arkkitehtuurin eroihin ja suorituskykyyn. Kaikki testit suoritettiin verkkoselaimessa.

WebAssembly oli suurimmassa osassa testejä JavaScriptia nopeampi. Blazor WebAssembly sovelluskehystä ja Angular-sovelluskehystä vertailtaessa sovelluksien toteutus onnistui Angularilla hieman helpommin. Angular oli suorituskykyä mitattaessa Blazor WebAssemblya nopeampi.

Johtopäätöksinä todettiin WebAssemblyn tarjoavan uudenlaisen tavan julkaista sovelluksia verkossa. Lisäksi todettiin sen tarjoavan pääosin hyvää suorituskykyä sovelluksia ajatellen. WebAssemblyn käytön todettiin vaativan syventymistä aiheeseen. Blazor WebAssemblyn todettiin puolestaan tarjoavan mahdollisuuden toteuttaa asiakaspuolen sovelluksia C#-kieltä ja .NET-kirjastoja hyödyntäen, vaikka suorituskyvyllään se ei vielä kilpaillut Angular-sovelluksien kanssa.

Avainsanat (asiasanat)

WebAssembly, Web-sovelluskehitys, Web-teknologiat

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Pekurinen, Toni

Applications of WebAssembly in browser applications

Jyväskylä: JAMK University of Applied Sciences, September 2021, 34 pages.

A degree of business information technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

In thesis research was about applications of WebAssembly in browser applications. WebAssembly is quite new technology and solving out its applications turned out to be topical. WebAssembly has been created to be run in browsers and it has been told that it improves application performance. Task was to find out what WebAssemly is and what is its current situation, and also to find out benefits and disadvantages of the usage. Goal was to find out possible applications for the usage and get picture of the usage in current situation. Thesis focused on the usage in browsers.

Thesis was executed as a qualitative research and comparing was used in it. Comparing was done between WebAssembly and JavaScript. In comparison of WebAssembly and Javascript the focus was on performance. Performance between these was measured with Fibonacci and Matrix multiplication -tests. Framework comparison was done between Blazor WebAssembly and Angular. Comparable client-side applications were developed in comparison of frameworks. Client-side application comparison focused on implementation, architecture differences and performance. All the test were run in browser.

WebAssembly was in most of microtests faster than JavaScript. In comparison of Blazor WebAssembly framework and Angular framework the implementation of application development was succeeded in Angular slightly easier. In performance measurement Angular was faster than Blazor WebAsseembly.

As conclusions was found that WebAssembly offers new way of publishing applications on the web. Also was found out that it offers mainly good performance when thinking about applications. Use of WebAssembly was found to demand deepening in the topic. Blazor WebAssembly was found to offer opportunity for taking advantage of C# language and .NET libraries in implementations of client-side applications, even though in performance it didn't compete with Angular applications.

Keywords/tags (subjects)

WebAssembly, Web-development, Web-technologies

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Käsitteet

WebAssembly	Matalan tason esitysmuoto korkean tason ohjelmointikielille
HTML	Merkintäkieli verkkosivujen toteuttamiseen
CSS	Tyylikieli HTML-merkintäkiellä toteutetuille verkkosivuille
JavaScript	Scriptikieli verkkosivujen interaktiivisuuden parantamiseksi
SPA	Single Page Application
LLVM	Kääntäjän ja työkalujen teknologiapaketti
Emscripten	Kääntötyökalu asm.js ja WebAssembly moduuleille
Blazor WebAssembly	C# ohjelmointikieltä ja .NET-kirjastoja käyttävä sovelluskehys
Angular	TypeScript-kieltä käyttävä sovelluskehys

1 Johdanto

Maaliskuussa 2017 WebAssemblysta julkaistiin ensimmäinen Minimum Viable Product (pienin toimiva tuote), jossa oli riittävästi ominaisuuksia sen käyttöä varten. WebAssembly mahdollistaa koodin ajamisen verkossa kielillä, joita ei voida oltu aikaisemmin käyttää, lähes natiivilla nopeudella. Esimerkiksi työpöytäsovelluksien, kuva-/video-editointisovelluksien, Virtuaalitodellisuussovelluksien (Virtual Reality), lisätyn todellisuuden sovelluksien (Augmented Reality), sekä raskaiden pelien mahdollistaminen selaimella suoritettavaksi ovat vain muutamia esimerkkejä, joita WebAssemblyn avulla voidaan toteuttaa. Web-selaimien ulkopuolella käyttökohteita voivat olla esimerkiksi pelien toimituspalvelut, sekä serveripuolen sovellukset. WebAssemblya tukevat selaimet ovat tällä hetkellä Mozilla Firefox, Google Chrome, Apple Safari ja Microsoft Edge.

WebAssembly on siis varsin uusi tulokas, mikä tekee aiheesta ajankohtaisen. Neljän suuren selainvalmistajan lisättyä tuki WebAssemlylle vaikuttaa sen tulevaisuus lupaavalta. WebAssembly on herättänyt keskustelua foorumeilla, siitä löytyy artikkeleita kehittäjäsivustoilta ja sen ominaisuuksia on esitelty erilaisissa tilaisuuksissa. On siis hyvä ajankohta tutkia WebAssemblya ja sen käyttökohteita web-sovelluskehityksessä. WebAssemblya on myös hyödynnetty selainpuolen sovelluskehyksien toteutuksissa.

Microsoftin kehittämä Blazor WebAssembly on yksi WebAssemblyä hyödyntävistä sovelluskehyksistä, jolla voidaan kehittää selaimessa käytettäviä web-sovelluksia. Tämä on rinnastettavissa JavaScriptillä toteutettuihin sovelluskehyksiin ja kirjastoihin, joilla selaimessa käytettäviä web-sovelluksia toteutetaan. Blazor WebAssemblyllä voidaan hyödyntää .NET-kirjastojen tarjontaa sovelluksia luodessa. Tämän kaltaiset sovelluskehykset voivat mahdollistaa työelämässä uudenlaisia työskentelytapoja web-sovelluskehityksen parissa.

Työelämän kannalta katsottaessa WebAssemblylla voi olla jonkin verran vaikutuksia. Mahdollisuus ohjelmoida sovelluksia webiin muillakin kuin perinteisillä web-sovelluskehityskielillä voi luoda sovelluskehittäjille uudenlaisia mahdollisuuksia työllistyä. Perinteisillä web-sovelluskehitysteknologioilla työtä tekevät sovelluskehittäjät saattavat toisaalta joutua opettelemaan web-sovelluksien kehitystä uusilla tavoilla, varsinkin selainpuolen toteutuksia tehdessä

Tutkimuksessa keskitytään WebAssemblyn ja perinteisten web-sovelluskehitysteknologioiden vertailuun. Tämä toteutaan kehittämällä testit vertailtavien kielien ja sovelluskehyksien välillä. Tutkimuksessa ei oteta kantaa selaimettomiin sovellustoteutuksiin, eikä myöskään muuhun käyttöön, joka ei liity web-sovelluksiin. WebAssemblysta pyritään löytämään piirteitä, joista on hyötyä verrattuna perinteisiin web-sovellusteknologioihin, sekä piirteitä, joista ei ole hyötyä.

2 Opinnäytetyön lähtökohdat / tietoperusta / teoreettinen viitekehys

2.1 Perinteiset web-sovelluskehitysteknologiat ja JavaScript

HTML

Verkkosivun sisällön rakenne määritellään HTML-merkintäkielen avulla. Sisältö saadaan näkymään ja käyttäytymään eri tavoin ympäröimällä tai käärimällä se elementtien sisälle, joista HTML koostuu. Esimerkiksi Hyperlinkin lisääminen sanaan tai kuvaan, sekä fonttikoon muokkaaminen ja kursivointi toteutetaan käyttämällä ympäröiviä ”tageja”. (HTML basics n.d.)

Berners-Lee loi verkkosivujen luomiseen tarkoitetun ja eniten käytetyimmän kielen, HTML-merkintäkielen vuonna 1991. HTML määritelmä HTML 2.0 julkaistiin vuonna 1995 ja se oli ensimmäinen standardi HTML-määritelmästä. HTML 4.01 julkaistiin vuonna 1999, tämä oli pääversio HTML-standardista. (HTML Tutorial n.d.) Seuraaja, sekä uusia elementtejä ja valmiuksia, sekä toiminnallisuuksia parantava tai poistava versio aiemmille HTML-versioille kuten HTML 4.01 oli HTML5. (HTML5 2021.) 28.5.2019 W3C ilmoitti ainoaksi HTML-versioksi WHATWG Living Standard version (HTML5 2021). Living Standard on jatkuvasti kehitettävä HTML-versio.

CSS

HTML kaltaisille merkintäkielille on olemassa ulkoasua määrittävä kieli CSS, CSS-tyylisivut. CSS-kielen avulla määritellään sivustolla ulkoasua kuten tekstin värejä, fonttien tyylejä, tekstin välistyksiä ja kokoja, taustavärit, kuvat ja erilaisia visuaalisia tehosteita. CSS avulla tyylit voidaan määrittää useammalle sivulle, ilman että jokaista sivua tarvitsee muuttaa erikseen, tämä on yksi tärkeimmistä ominaisuuksista CSS käytettäessä. (Pouncey, I. & York, R 2011, luku 1.)

CSS1 julkaistiin joulukuussa 1996 W3C (World Wide Web Consortium) toimesta. CSS1 toimi HTML ”tagien” visuaalisena muotoilumallina ja sisälsi CSS-kielen määritelmän. Toukokuussa 1998 julkaistiin CSS2. (Lakshmi Srinivas 2018.) Kesäkuussa 1999 julkaisiin CSS3 W3C suosituksena (Lakshmi Srinivas 2018).

JavaScript

JavaScript on tulkittu skriptauskieli ja koska se on tulkittu niin se ei tarvitse kääntäjää samalla tavalla avukseen kuten C tai C++ kielet. JavaScript on verkkoon tarkoitettu kieli ja sitä ajetaan suoraan selaimessa. (Anthory Grant 2019.) Anthony Grantin (2019) mukaan verkkosivuja ja verkkosovelluksia voidaan toteuttaa, kun mukaan otetaan HTML ja CSS. Grant myös kertoo, että useimmat modernit verkko- ja mobiiliselaimet tukevat JavaScriptia. (Anthory Grant 2019.) Amit Prajapatin (2020) mukaan useimmissa tapauksissa JavaScriptia käytetään parantamaan käyttäjäkokemusta lisäämällä verkkosivustoille responsiivisia ja interaktiivisia elementtejä. Prajapati kertoo myös, että JavaScriptin avulla voidaan helposti ja nopeasti luoda valikoita, animaatioita, videosoittimia, interaktiivisia karttoja ja jopa yksinkertaisia pelejä. (Amit Prajapati 2020.) Verkkosivuja toteutetaan myös erilaisilla sovelluskehyksillä ja kirjastoilla, joissa ohjelmointi toteutetaan JavaScriptillä.

JavaScript-sovelluskehyksillä on mahdollista tehdä skaalautuvia ja interaktiivisia verkkosovelluksia, ne ovat tärkeässä roolissa modernissa asiakaspuolen kehitystyössä tuoden kehittäjille kokeillut ja testatut työkalut kyseiseen tarkoitukseen. (Understanding client-side JavaScript frameworks 2021.)

2.2 WebAssembly

WebAssembly on matalan tason Assemblyn kaltainen kieli. WebAssembly mahdollistaa monilla eri kielillä ohjelmoidun koodin ajamisen verkossa lähes natiivilla nopeudella (WebAssembly 2021.) Marraskuussa 2017 WebAssemblyn parissa työskentelevä ryhmä tuli yhteisymmärrykseen siitä, että pienin toimiva tuote (Minimum Viable Product) on saavuttanut pisteen, jossa suunnittelutyön jatkaminen ei ole mahdollista ilman toteutuskokemusta ja merkittävää käyttöä. (Roadmap n.d.) WebAssemblyn tavoitteina on olla nopea ja tehokas erilaisilla alustoilla. Lisäksi tavoitteena on

myös olla luettava ja virhekorjattava, vaikka se onkin matalan tason assembly kieli. Muihin tavoitteisiin kuuluu myös turvallisuus, toiminta muiden web-teknologioiden kanssa, sekä taakse päin yhteensopivuus (WebAssembly Concepts n.d.)

WebAssemblyn käyttökohteita selainkäytössä voisi olla esimerkiksi kuvien ja videoiden käsittely, pelit, musiikki sovellukset, virtuaalitodellisuuden ja lisätyn todellisuuden sovellukset, CAD-ohjelmat, tieteellinen visualisointi ja simulointi, kehittäjätyökalut, etätyöpöydät ja paikalliset verkkopalvelimet. Selaimen ulkopuolella käyttökohteita voisivat olla esimerkiksi pelien jakelupalvelut, palvelinpuolen epäluotettavan koodin laskenta, palvelinpuolen sovellukset ja hybiridinatiivisovellukset mobiililaitteille. (Use cases n.d.)

WebAssemblyn kerrotaan olevan nopea. Lin Clarkin (2017) mukaan WebAssembly koodia ei tarvitse jäsennellä abstraktiksi syntaksipuuksi kuten JavaScriptin kohdalla, ainoastaan validointi ja purku on suoritettava sen tarkistamiseksi, että virheitä ei löydy. WebAssembly on myös lähempänä koneen ymmärtämää muotoa, mikä tekee optimoinnin nopeammaksi. Kääntäjän ei tarvitse käyttää aikaa koodin suorittamiseen tarkkailua varten, kun se alkaa kääntää optimoitua koodia, eikä myöskään tarvitse kääntää eri versioita tarkkailuun perustuvien tuloksien pohjalta, optimointeja on tehty jo Ahead Of Time vaiheessa LLVM:ssä. WebAssemlyn ei tarvitse myöskään käydä läpi uudelleenoptimointia. WebAssemblyssä myös koodin suorittaminen on nopeaa, sillä sen ei tarvitse käydä läpi samaa JIT-optimisaatio vaihetta kuten JavaScriptin kohdalla on. WebAssembly tuo kääntäjälle joukon ohjeita, jotka ovat laitteille ihanteellisempia, mikä nopeuttaa prosessia. (Lin Clark 2017.)

C/C++ koodi voidaan kääntää .wasm moduuliksi Emscriptenin avulla. Emscriptenin avulla luodaan myös luoda JavaScript ”liima” koodi, jonka avulla moduulia ajetaan selaimessa, sekä koodin tulokset esittävä HTML-sivu. (WebAssembly Concepts n.d.) Emscripten pitää olla asennettuna tietokoneelle, että tämä on mahdollista. JavaScript API on toinen tapa ajaa wasm moduulia.

JavaScript API sisältää WebAssembly olion, jossa on kaikki toiminnallisuus WebAssemblyyn liittyen. Olio toimii nimiavaruutena kyseisille toiminnoille, sekä on saatavilla sovelluksen laajuisesti selaimessa. (Urvashi 2020.)

2.3 Blazor WebAssembly

Blazor sovelluskehys mahdollistaa käyttöliittymien rakentamisen .NET hyödyntäen. JavaScript-kieltä ei tarvita vaan ohjelmointi tapahtuu C#-kielellä. Samaa .NET kirjoitettua sovelluslogiikkaa voidaan jakaa palvelinpuolen ja käyttöliittymänpuolen välillä. (Introduction to ASP.NET Core Blazor 2020.) Steve Sandersonin (2018) mukaan Blazor siirtyi ASP.NET organisaatiolle kehitykseen alkuvuodesta 2018, kokeelliseen vaiheeseen, jonka tarkoituksena oli selvittää voiko tuotteesta kehittää julkaisuversion. (Steve Sanderson 2018.)

Blazor sovellukset perustuvat komponentteihin, jotka ovat .NET C# luokkia ja määrittävät käyttöliittymän renderöintilogiikkaa, sekä käsittelevät käyttäjän tekemiä toimintoja. Komponentit ohjelmoidaan Razorin merkintäsivuina, joiden tiedostopääte on .razor. Razor syntaksi yhdistää HTML-merkintäkielen ja C#-kielen. (Introduction to ASP.NET Core Blazor 2020.) Razor-sivuilla käytetään HTML-merkintäkieltä ja C#-kielellä toteutettua logiikkaa itse renderöintilogiikan kirjoittamiseen. (Build reusable UI components with Blazor 2019.) Blazorissa ei osoitteellisilla sivuilla ole erillistä tiedostopäätettä, vaan antamalla reitit komponenttiin määritellään mistä sivusta on kyse. Razor @page direktiivi on tyypillinen tapa määrittää reitti. (Project structure for Blazor apps 2020.) Blazorista on kaksi versiota, Blazor Server ja Blazor WebAssembly.

Blazor WebAssembly on sovelluskehys, joka mahdollistaa SPA, eli Single Page Application sovelluksien tekemisen .NET käyttäen. .NET-koodin ajamisen selaimissa mahdollistaa WebAssembly. Blazorin vaatimien .NET-kirjastojen ajamisen selaimessa mahdollistaa se, että .NET-suoritusmoottorin on Blazorin kehitystiimi muuntanut WebAssembly-moduuliksi. (Introduction to ASP.NET Core Blazor 2020; Alex Jones 2021; BlazorGuy n.d.) Kun sovellus käännetään selaimella toimivaksi, käännetään C#-koodi ja Razor-tiedostot .NET-kokoonpanoiksi, jotka ladataan selaimeen .NET-suoritusmoottorin avulla. Blazor WebAssembly käyttää .NET-suoritusmoottoria, sekä kokoonpanoja, hyödyntäen JavaScriptia, joka mahdollistaa toiminnan selaimessa. (Introduction to ASP.NET Core Blazor 2020.)

Mono .NET-suoritusmoottorin avulla Blazor WebAssemblya voidaan ajaa kahdessa tilassa. Tulkittu tila (interpreted mode) ja AOT tila (Ahead of Time mode). Tulkitussa tilassa Mono käännetään WASM-moduuliksi ja sovelluksen käyttämät .NET-kirjastot säilyttävät alkuperäisen muotonsa, joita

sitten Monon avulla ajetaan selaimessa. AOT tilassa .NET-kirjastot käännetään puolestaan WebAssembly-binaareiksi ja tällöin ajon aikana ei tapahdu tulkintaa, vaan koodi ajetaan kuin muukin WebAssembly-koodi. (Steve Sanderson 2018.) Blazor WebAssemblyn AOT tuki on kuitenkin vielä toistaiseksi jätetty toteuttamatta ja se julkaistaan myöhemmin.

2.4 Angular

Dave Gaviganin mukaan (2018) Miško Hevery aloitta Angularin kehittämisen vuonna 2010. Aluksi Hevery kehitti Angularia omien sivuprojektiensa läpi viemiseksi tarkoituksenaan helpottaa kehittämistä, tällöin syntyi AngularJS (Dave Gavigan 2018.) Angular 2 julkaistiin vuonna 2016. Tämä versio oli täysin uudelleen kirjoitettu versio Angularista (Sunny Khanuja, 2019). Tämän jälkeen uusia versioita on tullut pienemmin aikavälein ja tällä hetkellä versio on Angular 12.

Angular voidaan määritellä alustana ja sovelluskehyksenä, jolla voidaan kehittää asiakaspuolen sovelluksia. Angular on kirjoitettu TypeScriptillä ja sillä kehitetään sovelluksia TypeScriptiä, sekä HTML-merkintäkieltä hyödyntäen. (Introduction to Angular Concepts n.d.) Angular käyttää komponentteja, jotka ovat sovelluksen näkymiä. Näiden komponenttien avulla käyttäjälle esitettävää näkymää voidaan muokata sovelluksen logiikan, sekä tiedon mukaan. Komponentit käyttävät palveluita, jotka voidaan injektoida komponenttiin riippuvuutena. Nämä palvelut lisäävät komponenttiin toiminnallisuutta, joka ei ole suoraan sidoksissa itse komponenttiin. (Introduction to Angular Concepts n.d.) Angularissa on lisäksi reititys-palvelu, jolla navigaatio eri näkymien välillä määritellään. (Introduction to Angular Concepts n.d).

2.5 Frontend-suorituskyky

Suorituskyvyn mittaaminen

Frontendin, eli asiakaspuolen suorituskyvyn mittaaminen on viime vuosina noussut pinnalle. Google esimerkiksi on määrittänyt Core Web Vitals nimikkeen alle kolme mittaria, joilla frontendin suorituskykyä voidaan mitata hyvän käyttäjäkokemuksen takaamiseksi. Friedmanin (2021) mukaan suorituskyky asiakaspuolella ei ole enää pelkästään teknisestä näkökulmasta katsottava asia, vaan huomion arvioisia asioita ovat myös saavutettavuus, käytettävyys, sekä hakukoneoptimointi. Suorituskykyä tulisi myös mitata, monitoroida ja parantaa. (Vitaly Friedman 2021.) Tutkimuksessa käytetään laboratorio-olosuhteissa käytettäviä mittareita, joilla voidaan mitata sivuston nopeutta.

Waltonin (2019) mukaan suorituskyvyn mittauksessa tärkeitä mittareita ovat sivuston latautumisnopeus, kuinka nopeasti näkyvät elementit latautuvat sivustolla, kuinka nopeasti koodi ajetaan, eli komponentit vastaavat käyttäjän tekemiin toimenpiteisiin, kuinka nopeasti sivusto vastaa käyttäjän toimenpiteisiin latautumisen jälkeen, toimivatko eri elementit oikein, kun käyttäjä käyttää sivustoa ja toimivatko sivuston visuaaliset elementit sulavasti. (Philip Walton 2019.)

Mittarit

First Contentful Paint

First Contentful Paint on sitä, kun selain renderöi ensimmäisen kerran käyttäjälle jotain näkyvää ja käyttäjä tällöin tiedostaa, että sivusto on latautumassa (First Contentful Paint 2020). First Contentful Paint kertoo ajan siitä, milloin ensimmäinen näkyvä elementti sivustolla on käyttäjän nähtävissä. (Philip Walton 2019.) Näitä sisällön elementtejä voivat olla teksti, kuvat, svg tai canvas joka ei ole valkoinen (First Contentful Paint 2020). Myös Walton (2019) kertoo saman asian, Waltonin mukaan First Contentful Paint on aika siitä, kun sivusto on aloittanut latautumisen ja ensimmäinen näkyvä sisältö on latautunut. (Walton 2019.) Toisin kuin Largest Contentful Paint, tämä ei pyri kertomaan siitä, milloin sivuston pääsisältö olisi käyttäjän nähtävissä. (Philip Walton 2019).

Largest Contentful Paint

Largest Contentful Paint kertoo siitä, milloin sivusto on mahdollisesti latautunut kokonaan ja sivusto on käyttäjän nähtävissä. (Philip Walton 2019.) Friedmanin (2021) mukaan tällaisia elementtejä voivat olla kuvat tai tekstiosiot sivustolla. (Vitaly Friedman 2021.) Friedman (2021) myös kertoo, että hyvä Largest Contentful Paint-tulos olisi alle 2,5 sekunttia, myös Walton (2019) kertoo samaa, että tuloksen pitäisi olla alle 2,5 sekunttia. Tämä on yksi Googlen kolmesta Core Web Vitals mittareista yhdessä First Input Delayn ja Cumulative Layout Shiftin kanssa.

Time To Interactive

Time To Interactive kertoo, milloin sivusto on käytettävissä. Käytännössä sivusto voi näyttää latautuneen, mutta se ei välttämättä ole vielä käytettävissä. (Philip Walton 2019.) Tätä mukailee Friedmanin (2021) kirjoitus, jonka mukaan Time To Interactive on käytännössä sitä, kun käyttäjä voi

tehdä toimintoja sivustolla käyttöliittymässä. Waltonin (2019) mukaan hyvä tulos TTI:lle on alle 5 sekunttia kun suorituskyvyn mittauksiin käytetään keskitason matkapuhelimia.

Total Blocking Time

Total Blocking Time on aika joka kuluu First Contentful Paintista Time To Interactiveen. Tänä aikana pääsäie on varattu ja se ei vastaa käyttäjän toimenpiteisiin (Philip Walton 2019). Mikäli toimenpide vie aikaa yli 50 millisekunttia, luokitellaan se pitkäksi toimenpiteeksi ja näin ollen pääsäie luokitellaan estyneeksi (Philip Walton 2019). Hyvä tulos TBT:lle on alle 300 millisekunttia keskitason matkapuhelimilla (Philip Walton 2019). Total Blocking Time korreloi hyvin First Input Delay arvojen kanssa ja ovat laboratoorio-olosuhteissa mitattavia. Total Blocking Time arvojen mukaan tehdyt parannukset vaikuttavat myös First Input Delay tuloksiin. (Philip Walton 2019.)

Cumulative Layout Shift

Cumulative Layout Shift on yksi Google Core Web Vitals-mittareista. Tämä mittari kertoo siitä, kuinka usein käyttäjälle ilmaantuu odottamattomia elementtien siirtymisiä sivustolla (Vitaly Friedman 2021). Waltonin ja Mihajlijan (2019) mukaan kehittäjien pitäisi pyrkiä 0.1 sekunttiin tai tätä pienempään tulokseen Cumulative Layout Shift pisteissä, että sivusto takaisi hyvän käyttäjäkokemuksen. Myös Friedman (2021) sanoo, että suositeltava tulos olisi alle 0,1 sekunttia.

3 Tutkimuksen tarkoitus, tavoitteet ja tutkimustehtävät

3.1 Tutkimuksen tavoite, rajaus ja tutkimuskysymykset

Tavoitteena opinnäytetyössä on tutkia mikä WebAssembly on ja mihin sitä voitaisiin käyttää web-sovelluskehityksessä. WebAssembly on varsin uusi tulokas ja siitä on vielä vähän tietoa saatavilla, tämä tekee tutkimuksesta ajankohtaisen. Aihe liittyy web-sovelluskehitykseen ja siihen mihin WebAssemblya teknologiana voisi mahdollisesti soveltaa. Keskeisiä mielenkiinnonkohteita on WebAssembly teknologiana: mikä se on, mitä ominaisuuksia sille on luvattu, nykytilanne, sekä mihin sitä voidaan soveltaa. Näin saadaan kuva WebAssemblyn käyttökohteista selainsovelluksissa.

WebAssemblyn websivun (ks. Use Cases n.d) mukaan WebAssemblylla on monia muitakin käyttökohteita kuin pelkästään selaimella esitettävät sovellukset, kuten serveripuolen sovellukset. Rajauksessa suljetaan pois kaikki muut kuin selaimella näytettävät web-sovellukset, tarkoituksena on pitää aihealue järkevän kokoisena. Aiheessa ei myöskään perehdytä sovelluksien ohjelmointiin eri ohjelmointikielillä (C, C++, jne.), koska tarkoituksena on tutkia WebAssemblyn käyttöä. Tutkimuksessa kuitenkin otetaan huomioon Microsoft Blazor WebAssembly, joka mahdollistaa .NET-kirjastojen käytön ja on tarkoitettu web-sovelluskehityksen asiakaspuolen toteutuksiin. Blazor WebAssembly käyttää WebAssemblya, tämä sivuaa noiden muiden kielien käyttöä.

- Mikä webassembly on ja mikä on sen nykytilanne?
- Mitkä ovat sen käytön edut ja haitat?

Deskriptiivisen tutkimuksen avulla pyritään kuvaamaan ilmiötä. Mitä-kysymykset eli mistä on kyse ja mitkä tekijät (= muuttujat) vaikuttavat ilmiöön, toimivat vastauksena. (Kananen, 2014, 38.) Tutkimuksen tarkoituksena on selvittää mikä WebAssembly on ja WebAssemblysta pyritään löytämään piirteitä, joista on hyötyä, sekä piirteitä, joista ei ole hyötyä verrattuna perinteisiin web-sovelluskehitysteknologioihin. Näistä tuloksista voidaan päätellä mahdollisia sovelluskohteita. Tutkimuksessa pyritään löytämään näkökulmia WebAssemblyn käyttöön web-sovelluskehityksen näkökulmasta.

3.2 Menetelmät, aineistonkeruu ja analyysi

Tutkimusmentelmänä käytetään kvalitatiivista tutkimusta, eli laadullista tutkimusta. Laadullinen tutkimus sopii tässä tapauksessa, kun tarkoituksena on selvittää WebAssemblyn nykytilaa, sekä mikä on WebAssemblyn paikka web-sovelluskehityksessä.

Kun etsitään vastausta kysymykseen ”mistä tässä on kyse”, voidaan käyttää laadullista tutkimusta (Kananen, 2014, 16). Kun ilmiöstä tiedetään vähän, kvalitatiivinen tutkimus todennäköisemmin on ainoa kysymykseen tuleva vaihtoehto (Kananen, 2014, 17).

Tutkimus toteutetaan hyödyntämällä verkosta löytyvää materiaalia, sekä kehittämällä vertailtavat sovellukset. Tietoa on vähän saatavilla, joten kehittämällä testaamiseen sopivat sovellukset voidaan kerätä tietoa, josta saadaan tuloksia tutkimusta varten.

Tietoperustan lähdeaineiston tiedonhaussa hyödynnetään hakukoneita, sekä kirjaston palveluita. Tiedonhaussa pyritään hyödyntämään teknologiakehittäjien virallisia aineistoja, että saadaan teknologioista tarkkaa tietoa. Tietoperustan lähdeaineiston tiedonhaussa käytetään myös artikkeleita ja blogeja tarpeen mukaan.

Aineistoa kerätään kehittämällä testisovellukset. WebAssemblyä ja JavaScriptia vertaillaan sovelluksilla, joissa suoritetaan laskutoimenpiteitä. Näillä sovelluksilla mitataan sovelluksen suoritukseen kulunutta aikaa, joka kertoo suorituskyvystä. Frontend-kehitystä mitataan kehittämällä vertailtavat sovellukset Blazor WebAssemlylla ja Angularilla. Blazor WebAssembly käyttää WebAssemblya ja on yksi kehittyneimmistä sovelluskehyksistä, joka hyödyntää WebAssemblya. Angular on suosituimpia sovelluskehyksiä Reactin ja Vuen rinnalla. Koska Angular on täysi sovelluskehys, on se vertailtavissa Blazor WebAssemblyyn. Angular on kehitetty TypeScriptillä, jolla myös varsinainen ohjelmointi tehdään.

Samaan lajiin kuuluvia, mutta jollakin tavalla toisistaan eroavia tapauksia tai yksilöitä tutkittaessa tutkija käyttää vertailua aineistoa hyödyntäen (Routio n.d.). Aineistoa saadaan kehittämällä sovellukset tutkimuksen aikana. Aineisto analysoidaan vertailemalla eroavaisuuksia kehittämisen pohjalta saatavista tuloksista, sekä hyödynnetään tietoa mitä saadaan verkosta. Tarkoituksena on kehittää sovellukset, joita mittaamalla tietoa saadaan tutkimusta varten. Lisäksi kehittämisen tuloksena saadaan itse kehitysprosessista tietoa, jota voidaan käyttää vertailussa. Tämä on varsinkinkin frontend-kehittämisen huomioon ottaen tärkeää tietoa. WebAssemblya ja JavaScriptia verrattaessa tutkimus kohdistuu lähinnä suorituskykyyn, mikä on WebAssemblyn kannalta merkittävin kriteeri.

Katsaus WebAssembly sovelluskehyksiin tehdään hakukoneita hyödyntämällä. Katsauksessa pyritään saamaan kuva siitä, mikä on sovelluskehyksien saatavuuden nykytilanne. Tarkoitus on kuvailla hieman nykytilannetta tällä tasolla. Kanasen (2014, 75.) mukaan fyysisen maailman ilmiöitä, sekä verkon ilmiöitä voidaan havainnoida käyttämällä verkkohavainnointia.

4 Toteutus

4.1 Vertailukriteerit

Taulukko 1. Vertailukriteerit

Suoritusnopeus	WebAssembly C-kielellä vastaan JavaScript
Sovelluskehyksien suorituskyky	Blazor WebAssembly vastaan Angular
Kehittäminen	Blazor WebAssemblyn ja Angularin erot. Kehittäminen, julkaisu ja arkkitehtuuri.
Katsaus WebAssembly sovelluskehyksiin	Nykytilanne sovelluskehyksien osalta

4.2 Vertailu WebAssembly ja JavaScript

Tutkimuksessa kehitettiin vertailtavat testit C-kielellä ja JavaScriptilla. C-kielinen koodi käännettiin wasm-moduuliksi, että sitä voidaan ajaa selaimessa HTML-sivulla. Suoritettaviksi testeiksi valikoituivat rekursiivinen Fibonacci-laskukaava ja matriisin kertolasku.

Fibonacci-testi tehtiin C-kielellä omaan tiedostoonsa, joka käännettiin Wasm-moduuliksi. Laskukaavana sovellettiin Tutorialspoint sivulla esiteltyä koodia. (ks. Samual, S 2018). Wasm-moduulin lisäksi syntyi JavaScript-tiedosto, jonka avulla moduulia ajettiin HTML-sivulla. Wasm-moduuliksi kääntäminen tapahtui Emscriptenin avulla, jolla toimenpide voitiin suorittaa. Kääntäminen tapahtui komentokehotteessa komennolla: **emcc fibonacci.c -O3 -s WASM=1 -s NO_EXIT_RUNTIME=1 -s "EXPORTED_RUNTIME_METHODS=['ccall']" -o fibonacci.js**

Testimetodin ajamisessa hyödynnettiin Emscriptenin HTML-tiedostoa, joka otti käyttöön Wasm-moduulin hyödyntämällä kääntämisessä syntynyyttä JavaScript-tiedostoa. Sivulle lisättiin painike

testimetodin ajamista varten, sekä tekstikenttä tuloksen tulostamista varten. Testin ajamisen mahdollistamiseksi lisättiin painiketta varten sivulle vielä tapahtuman käsittelijä, jossa kutsutaan testimetodia, kun painiketta käytetään. JavaScript-testi tehtiin omaan tiedostoonsa, joka otettiin käyttöön HTML-sivulla. JavaScript testiä varten HTML-sivulle lisättiin painike, jolla testi voidaan ajaa, sekä tekstikenttä, johon saadaan tulos.

Matriisin kertolasku testi tehtiin samalla periaatteella, erotuksena C-kielisessä testissä ajan laskeminen, joka toteutettiin C-kielisen koodin sisällä ja tuloksen tulostamiseen hyödynnettiin Emscripten HTML-tiedoston omaa tekstikenttää.

Matriisin kertolaskun Wasm-moduulit luotiin komennolla: **emcc arraymethod.c -O3 -s WASM=1 -s INITIAL_MEMORY=512MB -s TOTAL_STACK=256MB --shell-file shell_minimal.html -s NO_EXIT_RUNTIME=1 -s "EXPORTED_RUNTIME_METHODS=['ccall']" -o matrixmethod.html**

4.3 Vertailtava asiakaspuolen sovellus

Testisovelluksien teemana on musiikinlistaus-sovellus, jossa on yksinkertaisina toimintoina musiikin lisääminen listaan, sekä sen poistaminen listasta. Musiikin lisääminen tapahtuu lomakkeelta ja poistaminen tehdään klikkaamalla painiketta kappaleen vieressä. Sivulla on navigaatiovalikko ylälaidassa, jolta voidaan navigoida musiikin listaukseen ja kappaleen lisäämiseen. Musiikin listauksessa näytetään kappaleesta sen id, artistin nimi, albumin nimi ja kappaleen nimi.

4.4 Angular-sovellus

Angular-sovelluksen toteuttaminen aloitettiin luomalla uusi projekti antamalla seuraava komento terminaaliin: **ng new musiclist**

Tämän jälkeen Angular CLI kysyi halutaanko käyttää Angular routing, sekä mitä tyyliformaattia halutaan käyttää. Routing otettiin käyttöön ja tyyliformaatiksi valittiin perinteinen CSS.

```
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

Kuvio 1. Angular projektin luomisen valinnat

Seuraavaksi luotiin sovelluksen tarvitsemat komponentit (component). Kompontit esittävät sovelluksen näkymän. Ensimmäisenä luotiin list-kompnentti. Komponentin luominen tapahtuu komennolla: **ng generate component components/list**

```
CREATE src/app/components/list/list.component.html (19 bytes)
CREATE src/app/components/list/list.component.spec.ts (612 bytes)
CREATE src/app/components/list/list.component.ts (267 bytes)
CREATE src/app/components/list/list.component.css (0 bytes)
```

Kuvio 2. Komponentille luodut tiedostot

Komponenttien luomisen jälkeen luotiin niille palvelu (service). Palvelun avulla saadaan komponenteille välitettyä tietokannasta tietoa http-kutsuilla. Komponentin luominen tapahtuu komennolla: **ng generate service services/list**

```
CREATE src/app/services/list.service.spec.ts (347 bytes)
CREATE src/app/services/list.service.ts (133 bytes)
```

Kuvio 3. Palvelulle luodut tiedostot

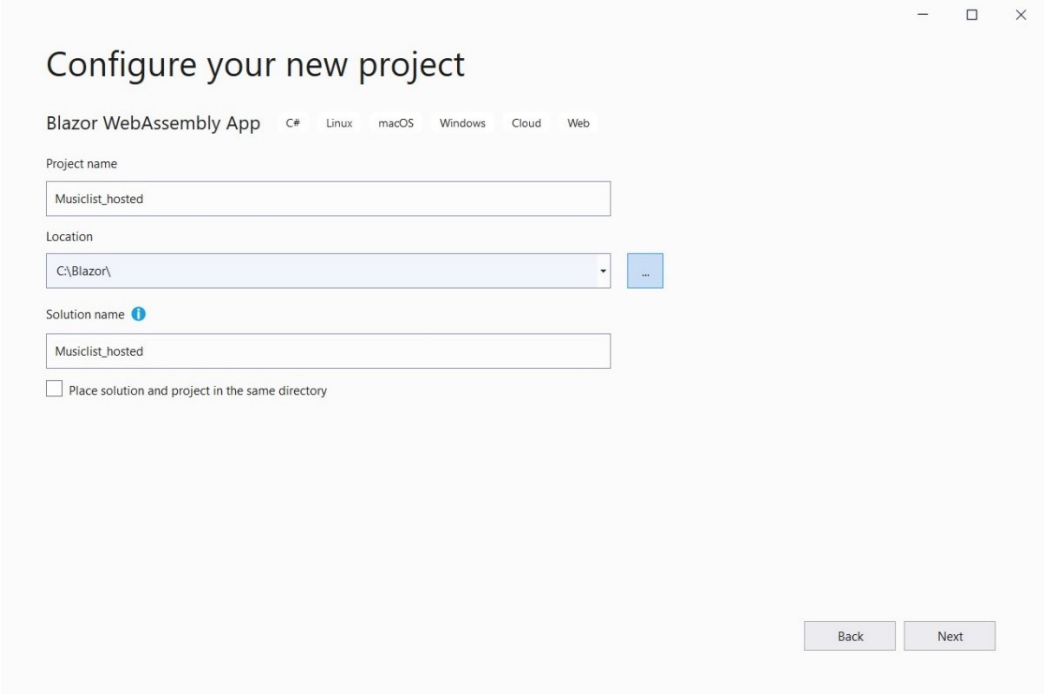
Palvelun luomisen jälkeen luotiin rajapinta (interface), joka vastaa rakenteeltaan tietokannasta välitettyä musiikkikappaletta.

Seuraavaksi luotiin komponentteihin sivuston ulkoasu ja toiminnallisuus. Ulkoasun luomisessa hyödynnettiin bootstrap-lisäosan avulla, joka Angularille on saatavissa. Tämä otettiin käyttöön, koska Blazor WebAssembly hyödyntää sitä, näin sovelluksien ulkoasut saatiin vastaamaan toisiaan. Komponentin toiminnallisuudesta vastaavaan osaan lisättiin metodit, joilla voidaan välittää palvelussa tapahtuvat http-kutsut komponentille.

Palveluun luotiin http-kutsuja käyttävät metodit, joilla tieto liikkuu palvelimen välityksellä Angular-sovelluksesta tietokantaan ja sieltä Angular-sovellukseen.

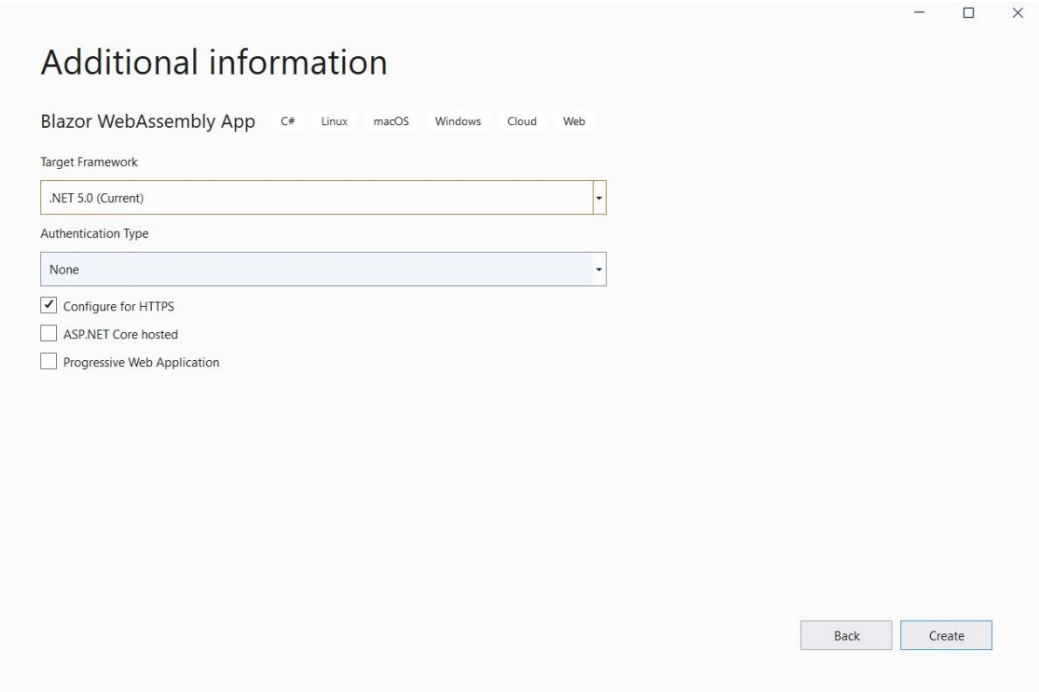
4.5 Blazor WebAssembly-sovellus

Blazor WebAssembly sovelluksen luominen suoritettiin Visual Studiossa. Visual Studio valittiin työkaluksi, koska sen asennusvaiheessa saadaan helposti asennettua Blazor WebAssembly projektin tarvitsemat kirjastot ja työkalut. Uuden projektin luominen aloitettiin valitsemalla uuden projektin luomisnäkymässä Blazor WebAssembly App. Tämän jälkeen projektille annettiin nimi ja valittiin tallennushakemisto.



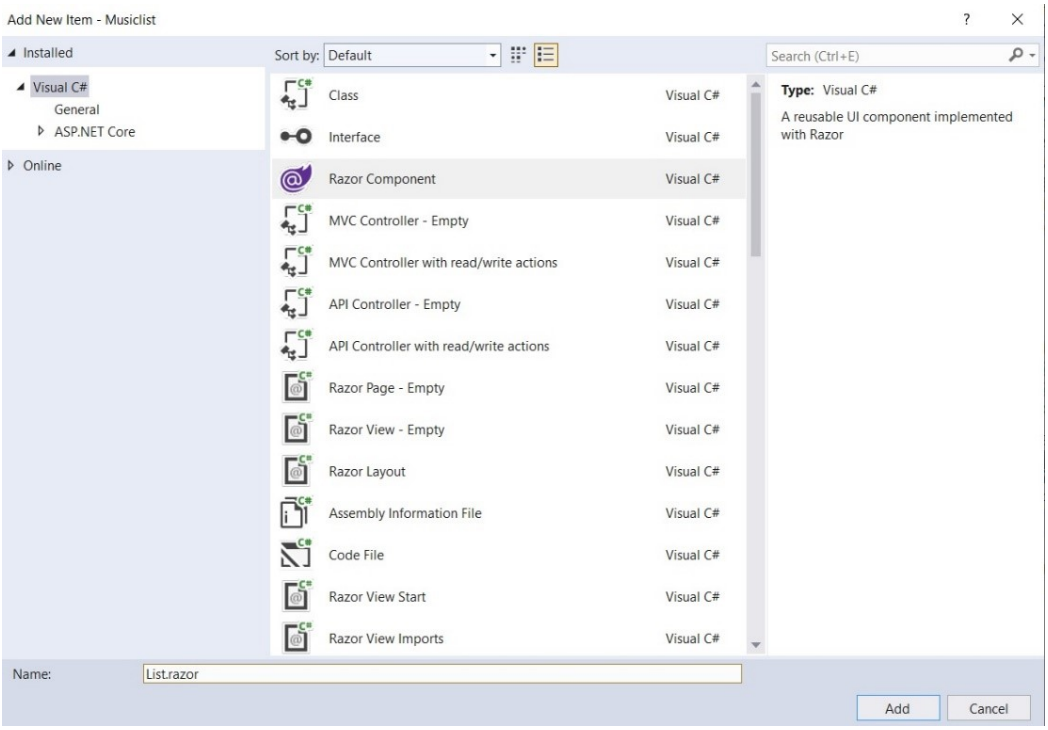
Kuvio 4. Blazor WebAssembly projektin luominen, nimi ja hakemisto

Seuraavaksi valittiin kohde sovelluskehys .NET Core 5.0. Lopuksi projektin luominen suoritettiin loppuun painamalla luomispainiketta.



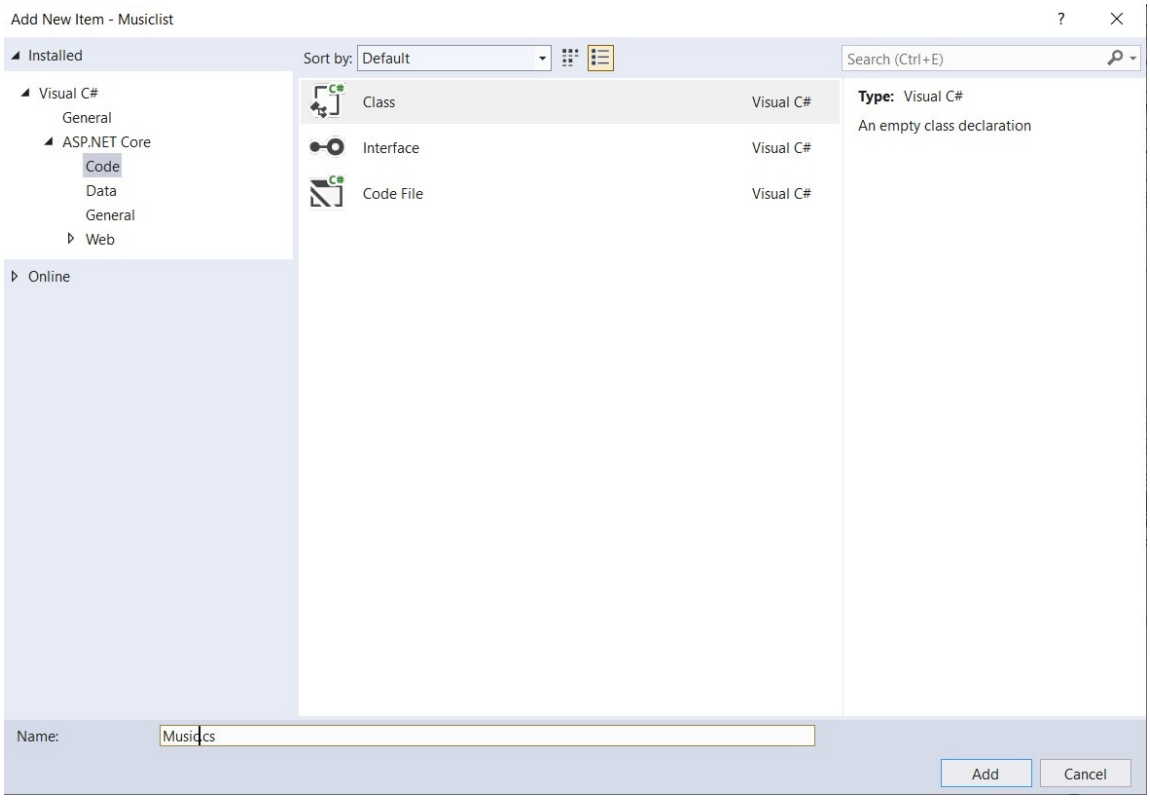
Kuvio 5. Blazor WebAssembly sovelluksen luominen .NET 5.0 valinta

Komponentit luotiin klikkaamalla hiiren oikealla painikkeella pages-hakemiston päällä ja valitsemalla Add, sekä Razor Component. Komponentille annettiin nimi ja klikattiin Add-painiketta.



Kuvio 6. Komponentin lisääminen Blazor WebAssembly sovellukselle

Sovellukseen tehtiin myös uusi C#-luokka, joka vastaa rakenteeltaan tietokannasta saatavaa musiikkikappaletta. Uusi luokka luotiin klikkaamalla hiiren oikealla painikkeella Models-hakemiston päällä ja valitsemalla Add, sekä Class. Tiedostolle annettiin nimi ja klikattiin Add-painiketta.



Kuvio 7. Luokan lisääminen Blazor WebAssembly sovellukseen

Seuraavaksi toteutettiin sovelluslogiikka REST API:n kutsumista varten. List.razor-tiedostoon tuli sovelluslogiikka listauksen näyttämistä varten, sekä kappaleen poistoa varten. List.razor-tiedostoon tuli myös näyttämisestä vastaavat elementit ja poistopainike. Add.razor-tiedostoon tuli kappaleen lisäämisestä vastaava sovelluslogiikka ja lomake-elementit. Lopuksi sovelluksen navigaatio ja ulkoasu tehtiin valmiiksi hyödyntämällä Blazor WebAssemblyn käyttämää bootstrapia.

5 Tulokset

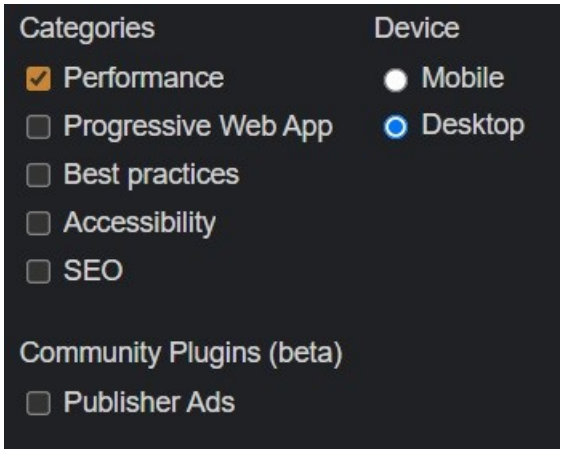
5.1 Testiympäristö ja suorituskykymittaukset

Testeissä käytettiin Internet Information Services (IIS) palvelinta, jonka avulla testisovelluksia voitiin ajaa paikallisesti Windows 10 käyttöjärjestelmällä. Testilaitteistossa oli Intelin kahdeksannen

sukupolven core i5-8350U suoritin ja 16 gigaa RAM-muistia. Blazor WebAssembly- ja Angular-testit ajettiin Google Chromen selaimessa, jonka versio oli 90.0.4430.72. Wasm C-kielen ja JavaScriptin testit ajettiin myös Mozilla Firefox-selaimella, jonka versio oli 89.0.2 ja Google Chrome-selaimella, jonka versio oli 91.0.4472.124.

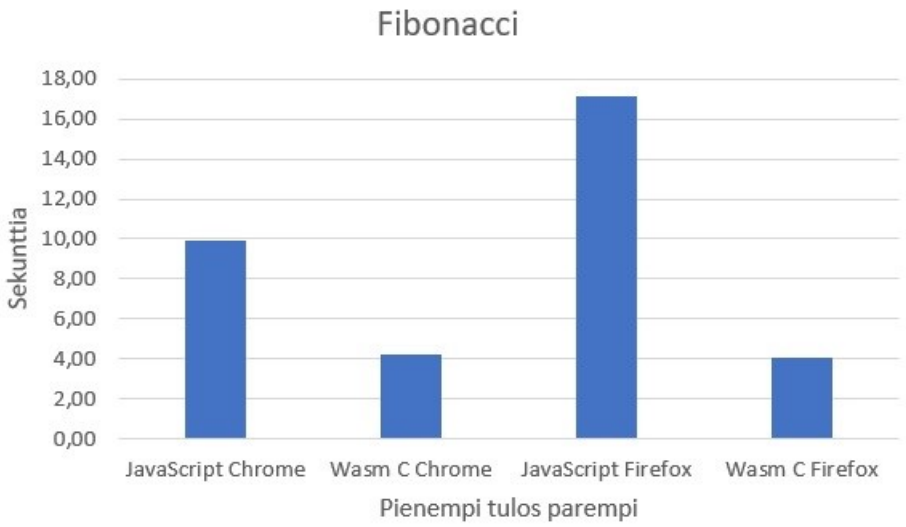
C-kielisen testisovelluksen, sekä JavaScriptilla toteutetun testisovelluksen testit ladattiin omaan verkko-osoitteeseensa, johon selaimella päästiin käsiksi. Verkko-osoitteen alta testit löytyivät kukin omasta hakemistopolustaan. Testit ajettiin kirjoittamalla haluttu osoite selaimen osoiteriville ja täydentämällä sitä hakemistopolulla, jossa testi sijaitsi. Kun testi oli suoritettu, kirjattiin tulos ylös. Testit ajettiin 3 kertaa ja näistä laskettiin keskiarvo.

Blazor WebAssembly- ja Angular-sovelluksien testeillä molemmilla oli myös omat verkko-osoitteensa. Sovelluksien backend toteutettiin Expressillä Node.js sovelluksessa ja tietokantaa pidettiin yllä paikallisesti MongoDB-tietokantaa hyödyntäen. Testit suoritettiin Google Chromen Lighthousella. Testit ajettiin kirjoittamalla sovelluksen osoite selaimeen, painamalla F12 painiketta, josta avautuu kehityskonsoli ja sieltä valitsemalla Lighthouse-välilehti. Asetuksiksi testiin valittiin performance-asetukset ja suoritusympäristöksi desktop. Testit suoritettiin painamalla Generate report-painiketta.



Kuvio 8. Lighthouse asetukset

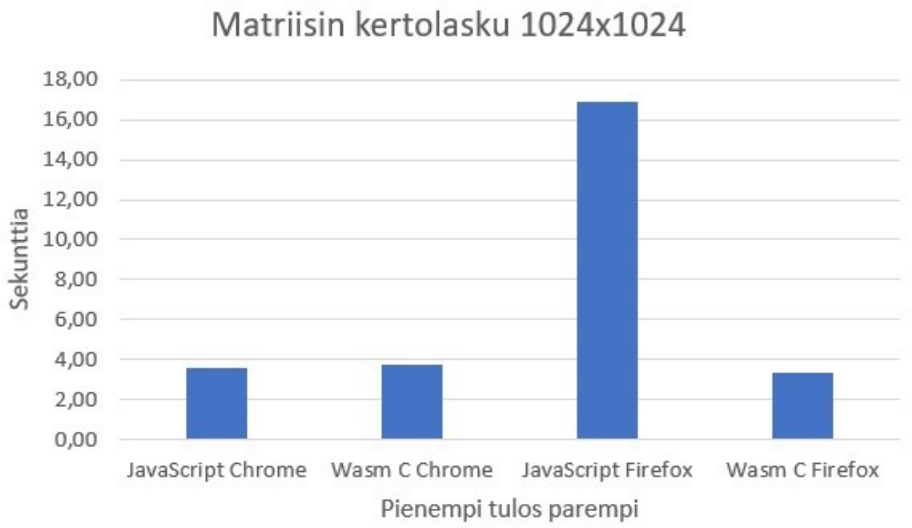
5.2 WebAssembly C ja JavaScript



Kuvio 9. Fibonacci Wasm C vastaan Javascript

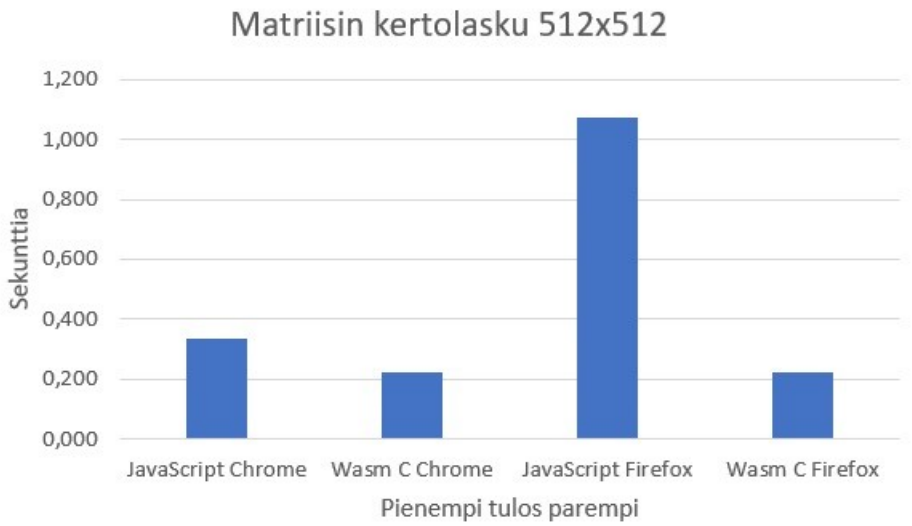
Fibonacci testit ajettiin kolme kertaa, joista laskettiin keskiarvo. Suorituksien välillä ei ollut tuloksissa suuria eroja. C-kielellä kirjoitettu testi suoriutui nopeammin. Chrome-selaimella C-kielellä kirjoitettu testi pystyttiin suorittamaan ajassa 4,19 sekunttia ja JavaScriptilla kirjoitettu testi pystyttiin suorittamaan ajassa 9,89 sekunttia. Firefox-selaimella C-kielellä kirjoitettu testi pystyttiin suorittamaan ajassa 4,07 sekunttia ja JavaScriptilla kirjoitettu testi pystyttiin suorittamaan ajassa 17,13 sekunttia. Ero on huomattava Chrome-selaimen eduksi JavaScriptin tapauksessa. C-kielellä toteutettu testi suoriutui molemmilla selaimilla lähes samassa ajassa.

Matriisin kertolaskutesteissä taulukoiden koot olivat 256x256, 512x512 ja 1024x1024. Testit ajettiin kolme kertaa ja tuloksista laskettiin keskiarvot.



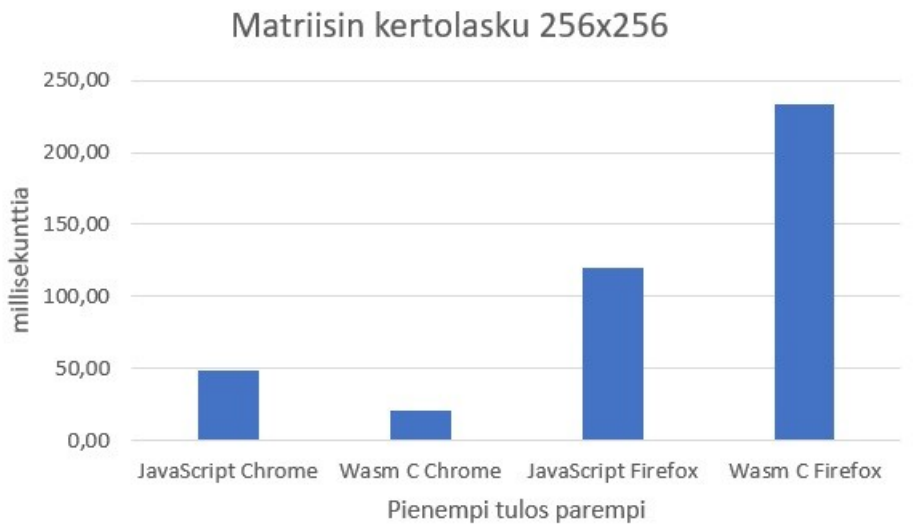
Kuvio 10. Matriisin kertolasku 1024x1024 Wasm C vastaan JavaScript

Ensimmäisenä matriisin kertolaskut tehtiin 1024x1024 kokoisilla taulukoilla. Tuloksista voidaan päätellä C-kielellä toteutetun sovelluksen olevan JavaScript sovellusta nopeampi Firefox-selaimella. C-kielellä testin suorittaminen kesti 3,31 sekunttia ja JavaScriptilla testin suorittaminen kesti 16,87 sekunttia. Tilanne oli kuitenkin erilainen, kun testi ajettiin Chrome-selaimella. C-kielellä testin suorittaminen kesti 3,75 sekunttia ja JavaScriptillä testin suorittaminen kesti 3,53 sekunttia.



Kuvio 11. Matriisin kertolasku 512x512 Wasm C vastaan JavaScript

Toisessa testissä taulukot olivat 512x512 kokoisia. Tässä tapauksessa C-kielellä toteutettu testi oli molemmilla selaimilla nopeampi. Chrome-selaimella C-kielellä toteutettu testi suoriutui ajassa 0,224 sekunttia ja JavaScriptilla toteutettu testi suoriutui ajassa 0,336 sekunttia. Firefox-selaimella C-kielellä toteutettu testi suoriutui ajassa 0,224 sekunttia ja JavaScriptilla toteutettu testi suoriutui ajassa 1,072 sekunttia.

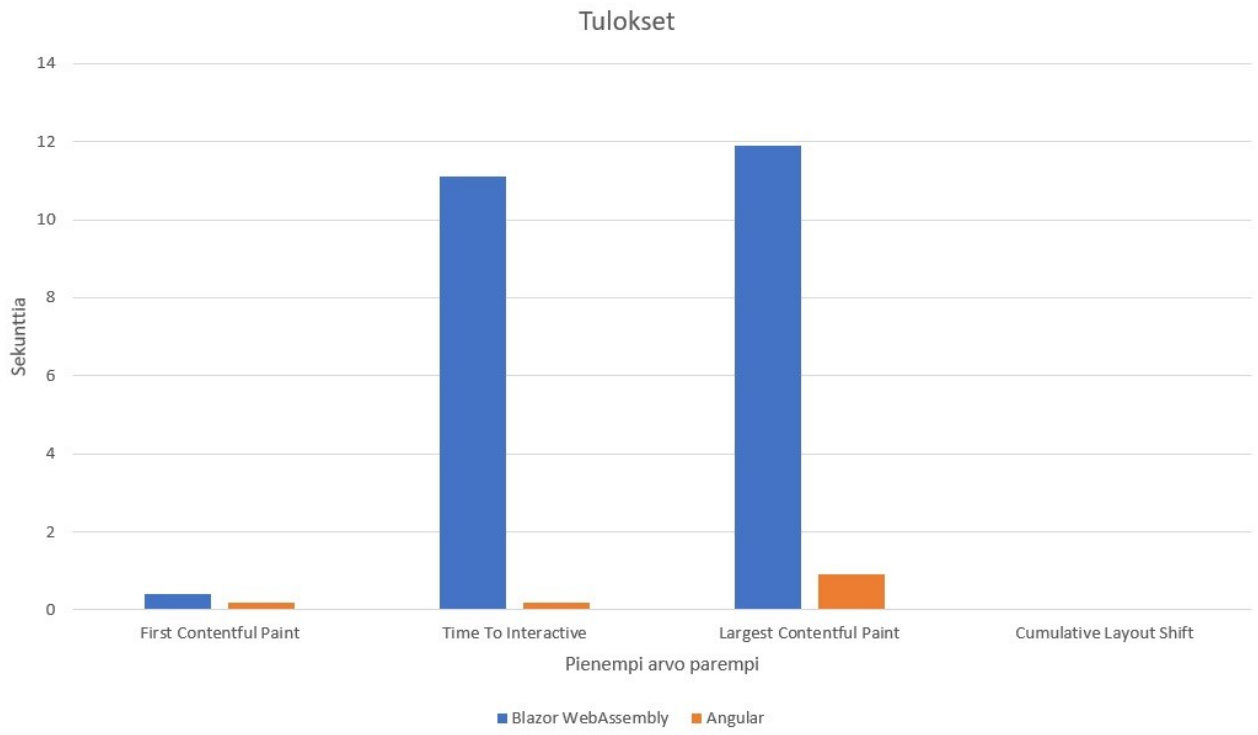


Kuvio 12. Matriisin kertolasku 256x256 Wasm C vastaan JavaScript

Kolmannessa testissä taulukot olivat 256x256 kokoisia. Chrome-selaimella C-kielellä toteutettu testi oli edelleen nopeampi, mutta Firefox-selaimella testi suoriutui JavaScriptia hitaammin. Chrome-selaimella C-kielellä toteutettu testi suoriutui ajassa 20,67 millisekunttia ja JavaScript testi suoriutui ajassa 48 millisekunttia. Firefox-selaimella C-kielellä toteutettu testi suoriutui ajassa 233,33 millisekunttia ja JavaScriptilla toteutettu testi suoriutui ajassa 120 millisekunttia.

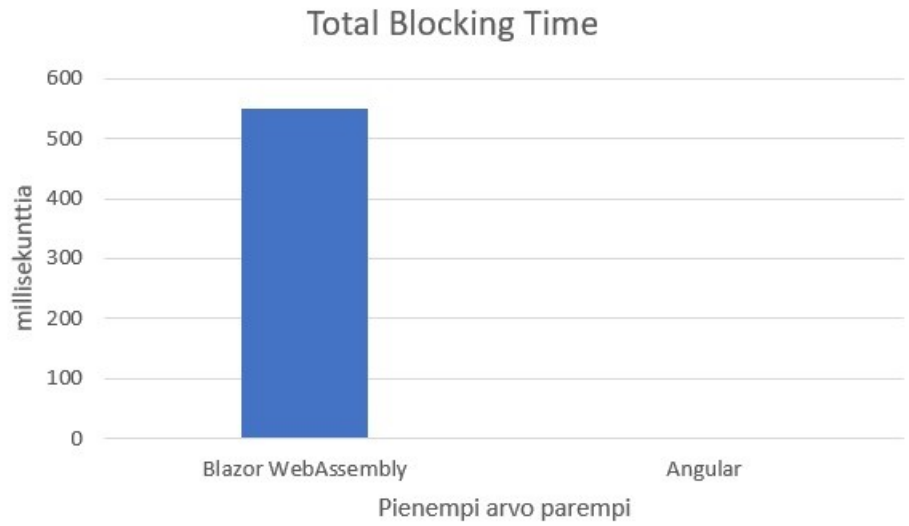
5.3 Blazor WebAssembly ja Angular

Seuraavat tulokset saatiin ajamalla testit Google Chromen Lighthousella.



Kuvio 13. FCP, TTI, LCP, CLS tulokset Blazor WebAssembly vastaan JavaScript

Tuloksissa näkyy, että Blazor WebAssembly on tällä hetkellä selvästi Angularia hitaampi, kun sovellusta ladataan. Ensimmäinen elementti, joka sivustolle ladataan, tulee molemmilla sovelluskehyksillä nopeasti. Pääsäie sovelluksessa vapautuu kuitenkin käyttäjälle yli 11 sekunttia myöhemmin Blazor WebAssembly-sovelluksessa verrattuna Angular-sovellukseen. Myös suurin sivustolla piirrettävä elementti näkyy käyttäjälle n. 11 sekunttia myöhemmin Blazor WebAssembly-sovelluksessa verrattuna Angular-sovellukseen. Eroja sovelluksien elementtien sijaintimuutoksilla sivustoa käytettäessä ei käytännössä ole.



Kuvio 14. TBT tulokset Blazor WebAssembly vastaan JavaScript

Blazor WebAssemblylla toteutetun sovelluksen pääsäie oli 550 millisekunttia varattuna. Tulos on suuri verrattuna Angulariin, joka sai testissä tulokseksi 0 millisekunttia.

5.4 Frontend-kehittämisen erot

Tutkimuksessa sovellukset toteutettiin eri kehitystyökaluilla. Angular-sovelluksen tapauksessa käytettiin Visual Studio Codea, Blazor WebAssemblyn tapauksessa käytettiin Visual Studio 2019. Molemmissa tapauksissa pitää opiskella kuinka mitäkin työkalua käytetään. Tutkimuksessa todettiin, että molemmissa tapauksissa löytynyt ohjeistus on kattavaa työkalujen käytön suhteen. Angular-sovelluksiin löytyy hyvä dokumentaatio ja sovelluksien kehittämiseen Angular-sivuston virallinen dokumentaatio antoi enemmän esimerkkejä. Blazor WebAssemblyn kohdalla dokumentaatio on hyvä, mutta dokumentaation lisäksi jouduttiin etsimään myös esimerkkejä tutoriaaleista. Sovelluksen kehityksessä Blazor WebAssemblyn kohdalla tuli myös vastaan ongelmia, joihin oli hankalampi löytää varsinaista ratkaisua. Sovelluksen ulkoasun muotoilu Bootstrapin kanssa esimerkiksi ei ollut täysin ongelmatonta, mikä taas Angular-sovelluksen toteutuksessa ei tuottanut ongelmia. Varsinainen komponenttien luominen onnistui molemmilla sovelluskehyksillä ilman ongelmia, ja sovelluksista saatiin vastaavat niin ulkoisesti, että toiminnallisesti.

Angularilla web-sovelluksen kehitys on sujuvaa, sillä Visual Studio Code tarjosi siihen riittävän hyvät mahdollisuudet ja Angular Cli:n käyttö oli helppoa. Blazoria kehitettiin Visual Studio 2019-sovelluksella ja myös sen käyttö oli yksinkertaista, poikkeuksena Angulariin eri komponenttien luominen voitiin toteuttaa valikoita hyödyntämällä. Angularin tapauksessa komponentit luotiin käyttämällä komentoriviä.

Arkkitehtuurit ja rakenteet Blazor WebAssemblyn ja Angularin välillä poikkesivat toisistaan hieman. Komponentit Blazor WebAssembly-sovelluksessa pitivät sisällään HTML-merkintäkieltä, sekä C#-ohjelmointikielellä toteutettua logiikkaa. Angular-sovelluksessa HTML-merkintäkieltä sisältävä tiedosto oli erillään itse logiikan toteuttavasta tiedostosta. Molemmissa tapauksissa hyödynnettiin erillistä kappaleen tietorakenteen mallitiedostoa. Blazor WebAssemblyn tapauksessa mallitiedosto otettiin komponentissa HTML-merkintäkielen yläpuolella, kun taas Angularissa mallitiedosto otettiin käyttöön komponentin logiikkaa toteuttavassa osiossa. Blazor WebAssembly-sovelluksessa backendin kanssa keskusteluun käytettävistä http-kutsuista vastaava luokka otettiin käyttöön myös komponentissa HTML-merkintäkielen yläpuolella. Angular-sovelluksessa toteutettiin erillinen palvelu, jossa http-kutsuista vastaava luokka otettiin käyttöön. Blazor WebAssembly-sovelluksessa komponentin reitti määritettiin HTML-merkintäkielellä kirjoitetun osan yläpuolella hyödyntäen @page-direktiiviä. Angular-sovelluksessa reitit toteutettiin erillisessä reiteistä vastaavassa tiedostossa.

Julkaisuversion tuottaminen Blazor WebAssembly-sovelluksesta Visual Studiossa oli yksinkertaisempaa kuin Angular version käyttämässä komentorivitoteutuksessa. Käytännössä Visual Studiossa julkaistun version toteuttaminen on ennestään tuttua Windows käyttäjälle, joka on omaksunut Windowsin käyttämisen, kun taas Visual Studio Codessa on tukeuduttava enemmän dokumentaatioon ja tämän pohjalta tehtävä ratkaisuja komentoriviä käyttäen. Kummassakin tapauksessa seurattiin virallista dokumentaatiota julkaisusta.

5.5 Sovelluskehykset

Tutkimuksessa keskityttiin asiakaspuolen sovelluskehyksien etsimiseen hakukoneita hyödyntämällä. WebAssembly asiakaspuolen sovelluskehyksiä on kehitteillä laajalti ja eniten niitä kehitetään RUST-kielellä, näitä on lukuisia. Eri verkkolähteistä, kuten keskustelupalstat saatiin yleinen kuva siitä, että vielä lähiaikoina WebAssembly sovelluskehykset eivät tulisi korvaamaan JavaScript

sovelluskehyksiä, vaan ne vaativat vielä aikaa saavuttaakseen samankaltaisen tason. Kiinnostusta WebAssemblyä hyödyntäviä sovelluskehyksiä kohtaan on olemassa.

Tällä hetkellä WebAssembly sovelluskehyksistä ei löydy tutkimuksessa haetun tiedon perusteella mitään virallista listausta niiden suosiosta, kuten JavaScript sovelluskehyksien kohdalla. Tutkimuksessa löydettiin vain listauksia, joita kehittäjät ovat tehneet omille sivustoilleen. Näitä listauksia löytyi jonkin verran, mutta ne olivat vaihtelevia sisällöltään.

6 Pohdinta ja johtopäätökset

6.1 luotettavuus

Teoriatiedon pohjana on käytetty teknologiakehittäjien sivustoilta saatua virallista materiaalia, sekä erilaisia kehittäjien kirjoittamia artikkeleita ja blogikirjoituksia. Teknologiakehittäjien sivustoilta saatu virallinen materiaali on laadultaan luotettavaa teknologiaa aukaisevaa dokumentaatiota. Kehittäjien kirjoittamat artikkelit ja blogikirjoitukset sisältävät asiaa teknologioista, joka usein toistui myös muissa lähteissä, mikä vahvistaa luotettavuutta. Puolueettomuutta näiden kohdalla ei voida taata, mikä jonkin verran heikentää tutkimuksen teoriapohjan luotettavuutta.

Tutkimuksessa käytettiin kahta eri tapaa mitata suorituskykyä WebAssemblylle, nämä olivat testejä, jotka ovat suuntaa antavia. Teknologian soveltaminen oikeassa elämässä antaisi todellisempia tuloksia tiettyä käyttöä tarkasteltaessa. Frontend-sovelluskehyksien kohdalla sovellukset olivat yksinkertaisia ja on mahdollista, että tilanne suorituskyvyn ja kehittämistyön suhteen on erilainen suuremman luokan sovelluksissa. Tulokset kuitenkin ovat luotettavia, sillä testiympäristö ei muuttunut, kun vertailevia testejä ajettiin.

6.2 Keskeisten tulosten tarkastelu suhteessa alkuosan teoreettiseen viitekehykseen

WebAssemblyn sijoittuminen asiakaspuolen sovelluksen rakenteessa oli kuten teoreettisessa viitekehyksessä on esitetty. Testisovellukset ajettiin JavaScript tagien sisällä HTML-tiedostossa hyödyntämällä Emscriptenillä luotuja tiedostoja. WebAssemblyn nopeudesta ja siihen liittyvistä seikoista kerrottiin teoreettisessa viitekehyksessä ja tulokset osoittivat, että WebAssembly on nopea.

Blazor WebAssembly-sovelluksen arkkitehtuuri vastasi teoreettisessa viitekehyksessä esitettyä mallia. Komponenteissa oli HTML-merkintäkieltä ja C#-logiikka. Reititys tapahtui @page-direktiiviä hyödyntäen. Sovellus myös käytti .NET kirjastoja. .NET kirjastojen käyttö kasvatti sovelluksen kokoa, mikä hidasti sovelluksen lataamista selaimeen käytettäväksi ja näin ollen Angular oli monella mittarilla nopeampi. Myös Angular-sovelluksen arkkitehtuuri vastasi teoreettisessa viitekehyksessä esitettyä arkkitehtuurimallia kuten tuloksissa kävi ilmi.

Teoreettisessa viitekehyksessä kuvattiin mikä WebAssembly on, tämä vastasi osittain ensimmäiseen tutkimuskysymykseen. Empiirisestä osasta saatiin vastauksena myös, että WebAssembly tuo uusia mahdollisuuksia kehittämiseen JavaScriptin valtaamalla alalla. Suorituskykyä mittaavilla tuloksilla, sekä Frontend-sovelluskehyksien kehityksestä saaduilla tuloksilla saatiin vastauksia myös ensimmäiseen tutkimuskysymykseen siitä, mikä on WebAssemblyn nykytilanne. Nykytilanteen voidaan sanoa olevan hyvä, kun WebAssembly moduulia käytetään asiakaspuolella, sillä on suorituskykynsä puolesta paljon annettavaa raskaammille sovelluksille, jotka upotetaan verkkosivulle. Myös Frontend-sovelluskehykset tarjoavat uusia tapoja ohjelmoida web-sovelluksia, mikä puolestaan tuo uusia mahdollisuuksia kehittäjille, jotka haluavat JavaScriptin sijaan käyttää jotain muuta ohjelmointikieltä.

WebAssemblyn käytön eduiksi voidaan laskea hyvä suorituskyky ja mahdollisuus käyttää sovelluksien toteuttamiseen muitakin kieliä kuin JavaScript. Frontend-sovelluskehyksien käytön etuna on edellä mainittu mahdollisuus käyttää verkkosovelluksen toteuttamiseen muita ohjelmointikieliä JavaScriptin sijaan. Käytön haitoiksi voidaan lukea, että wasm-moduuli pitää aina upottaa verkkosivulle JavaScriptin avulla, mikä aloittelijalle voi olla haasteellinen, jos käytössä on JavaScript API. Myös Emscriptenin tarjoamien valmiiden HTML-merkintäsivujen muokkaus haluttuun muotoon voi olla haasteellista. Sovelluskehyksien käytön haittoina tutkimuksen tuloksien pohjalta on suoritusnopeus. Sovelluskehyksien suoritusnopeutta ei voida yleistää, sillä Blazor WebAssembly on vain yksi monista WebAssemblya hyödyntävistä sovelluskehyksistä.

6.3 Johtopäätökset ja kehittämisehdotukset

Tutkimuksen tarkoituksena oli selvittää WebAssemblyn käyttökohteita Web-sovelluskehityksessä. Mielenkiinnonkohteita olivat mikä on WebAssembly, mitä ominaisuuksia sille on luvattu, nykytilanne ja mihin sitä voitaisiin soveltaa.

Tutkimuksessa saadun tiedon perusteella WebAssembly on teknologia, joka mahdollistaa uudenlaisen tavan julkaista sovelluksia verkossa, lisäksi se tarjoaa tällä hetkellä pääosin hyvää suorituskykyä selaimessa esitettäville sovelluksille. On kuitenkin otettava huomioon, että selaimet kehittyvät koko ajan ja myös JavaScript-moottoria kehitetään jatkuvasti eteenpäin. Chrome päivitti selaimensa JavaScript-moottorin toukokuun lopussa ja se näyttäisin vaikuttavan tuloksien perusteella suorituskykyyn positiivisesti JavaScriptin eduksi, erot kaventuivatkin WebAssemblyn ja JavaScriptiin välillä. Mozilla Firefox-selaimen kohdalla asiat olivat eri tavalla ja WebAssembly oli pääosin huomattavasti nopeampi JavaScriptiin verrattuna.

WebAssemblyä käytetään myös sovelluskehyksissä, mikä mahdollistaa asiakaspuolen sovelluksien ohjelmoinnin muillakin kielillä kuin JavaScriptillä. Tutkimuksessa vertailussa oli mukana Microsoftin Blazor WebAssembly, tämä kuitenkaan ei suorituskyvyn puolesta vielä pärjää tutkimuksessa vertailun kohteena olleelle Angular-sovelluskehykselle, mutta tarjoaa uuden tavan toteuttaa asiakaspuolen sovelluksia, mikä on JavaScriptiin tottumattomille ja C# kieltä, sekä .NET-kirjastoja käyttäneille positiivinen asia. Käytettävyys Blazor-sovelluksessa oli kuitenkin hyvä.

Tutkimuksen perusteella WebAssemblyn nykytilanne on se, että sitä voidaan käyttää sovelluksien julkaisuun ja se tarjoaa suorituskykyä, kun verkossa halutaan julkaista raskaampi sovellus, esimerkiksi sovellus, joka suorittaa raskasta laskentaa. WebAssembly soveltuisi myös hyvin tilanteissa, joissa sovellus on olemassa jollakin muulla kielellä kirjoitettuna ja tämä sovellus haluttaisiin julkaista verkossa. Mikäli sovellus ei ole raskas ja sitä ei ole olemassa, on JavaScript käytön helppoutensa kannalta parempi vaihtoehto. WebAssemblya on myös onnistuttu käyttämään sovelluskehyksien toteutuksissa, kuten tutkimuksessa mukana olleessa Blazor WebAssembly-sovelluskehyksessä. Tutkimuksen perusteella WebAssemblyn käyttö vaatii paljon syventymistä aiheeseen ja sen käytön opiskelua, mikä voi osoittautua ongelmalliseksi, kun tietoa on tällä hetkellä vähän saatavilla, sekä yhteisöissä käytyä keskustelua kehittämisestä ja eteen tulleista käytön ongelmista on niukasti.

Jatkotutkimuskohteina mielenkiintoisia aiheitakin nousi esille tutkimuksen aikana. Selaimet kehittyvät jatkuvasti ja tästä hyvä esimerkki oli, kun havaittiin Google Chrome-selaimen tulokset verrattuna Mozilla Firefox-selaimen tuloksiin, kun katsottiin JavaScript-suorituskykyä. Tämän perusteella

tutkimus selaimien nykytilanteesta voisi olla aiheellista ottaa tarkasteluun. Toinen mielenkiintoinen tutkimusaihe olisi eri WebAssemblya hyödyntävien asiakaspuolen sovelluskehyksien vertailu keskenään. Tutkimuksen aikana eri kieliä hyödyntäviä sovelluskehyksiä löytyi lukuisia ja niiden nykytilatta voisi hyvinkin tutkia, mikä antaisi laajempaa kuvaa WebAssemblyn hyödyntämisestä ja sen nykytilanteesta.

Ongelmalliseksi tutkimuksessa muodostui aiheen laajuus ja rajausta oli tehtävä paljon. Tämä tarkoittaa sitä, että tutkimuksen tuloksia ei voi yleistää, koska mukaan vertailuun olisi voinut ottaa enemmänkin WebAssembly-sovelluskehyksiä. Ongelmana oli myös sopivien testien valinta vertailuun, kun mitattiin eroja C-kielisen Wasm-moduulin ja JavaScriptin välillä. Muitakin mahdollisia testejä suorituskyvyn mittaamiseen olisi olemassa paljon. Näiden testien perusteella saatiin kuitenkin kuva WebAssemblyn käytettävyydestä.

Lähteet

Alex Jones. 2021. An Introduction to Blazor WebAssembly. Artikkeli Rock Solid Knowledge sivustolla 2.2.2021. Viitattu 31.3.2021 https://www.rocksolidknowledge.com/articles/an-introduction-to-blazor-webassembly.

Amit Prajapati. 2020. JavaScript: What is javascript used for and why should you learn it. Artikkeli Medium sivustolla 8.5.2020. Viitattu 2.8.2021. https://medium.com/@amitprajapati1993.ap/what-is-javascript-used-for-and-why-you-should-learn-it-e9f2ebc11866.

Anthory Grant. 2019. What is JavaScript and How Does It Work? Artikkeli Makeuseof sivustolla 6.12.2019. Viitattu 2.8.2021. https://www.makeuseof.com/tag/what-is-javascript/.

BlazorGuy. N.d. How Blazor WebAssembly Works. Artikkeli BlazorGuyn sivustolla. Viitattu 12.4.2021. https://blazorguy.net/how-blazor-webassembly-works/.

Build reusable UI components with Blazor. 2019. Dokumentaatio Microsoft.com sivustolla18.9.2019. Viitattu 23.8.2021. https://docs.microsoft.com/en-us/dotnet/architecture/blazor-for-web-forms-developers/components.

First Contentful Paint. 2020. First Contentful Paint. Artikkeli Mozillan kehittäjäsivustolla. Viitattu 22.4.2021. https://developer.mozilla.org/en-US/docs/Glossary/First_contentful_paint.

Lakshmi Srinivas. 2018. Comparison of CSS versions. Tutorialspoint sivustolla 13.4.2018. Viitattu 9.8.2021. https://www.tutorialspoint.com/Comparison-of-CSS-versions.

Dave Gavigan. 2018. The History of Angular. Artikkeli Medium sivustolla 4.3.2018. Viitattu 21.7.2021. https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7.

Getting Started. N.d. WebAssembly verkkosivun dokumentaatio käytön aloittamisesta. Viitattu 9.2.2021. https://webassembly.org/getting-started/developers-guide.

HTML basics. N.d. Mozillan verkkosivun dokumentaatio HTML:stä. Viitattu 8.3.2021. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.

HTML Tutorial. N.d. Tutorialspointin HTML tutoriaali. Viitattu 8.3.2021. https://www.tutorialspoint.com/html/index.htm.

HTML5. 2021. Mozillan verkkosivujen dokumentaatio HTML5:sta 26.6.2021. Viitattu 19.8.2021. https://developer.mozilla.org/en-US/docs/Glossary/HTML5.

Introduction to ASP.NET Core Blazor. 2020. Dokumentaatio ASP.NET sivustolla 25.9.2020. Viitattu 30.3.2021. https://docs.microsoft.com/fi-fi/aspnet/core/blazor/?view=aspnetcore-5.0.

Introduction to Angular Concepts. N.d. Dokumentaatio Angular.io sivustolla. Viitattu 27.7.2021. https://angular.io/guide/architecture.

Kananen, J. 2014. Verkkotutkimus opinnäytetyönä. Laadullisen ja määrällisen verkkotutkimuksen opas. Suomen yliopistopaino Oy. Viitattu 2.2.2021.

Kananen, J. 2014. Laadullinen tutkimus opinnäytetyönä: Miten kirjoitan kvalitatiivisen opinnäytetyön vaihe vaiheelta. Jyväskylä: Jyväskylän ammattikorkeakoulu. Viitattu 9.2.2021.

Lin Clark. 2017. What Makes WebAssembly Fast. Artikkeli Mozilla hacks sivustolla 28.2.2017. Viitattu 8.4.2021. https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/.

Philip Walton. 2019. Metrics. Measuring performance and user experience. Artikkeli web.dev sivustolla. Viitattu 19.4.2021. https://web.dev/metrics/.

Pouncey, I. & York, R. 2011. Beginning CSS: Cascading style sheets for web design. Kolmas painos. Wiley Pub., Inc. Viitattu 8.3.2021. https://janet.finna.fi, Skillsoft Books ITPro.

Project structure for Blazor apps. 2020. Dokumentaatio Microsoft.com sivustolla 20.11.2020. Viitattu 23.8.2021. https://docs.microsoft.com/en-us/dotnet/architecture/blazor-for-web-forms-developers/project-structure.

Roadmap. N.d. Dokumentaatio WebAssemblyn virallisilla sivuilla. Viitattu 7.4.2021. https://webassembly.org/roadmap/.

Routio, P. N.d. Vertailu. Oppimateriaali Virtuaaliyliopisto sivuilla. Viitattu 15.2.2021. http://www.uiah.fi/virtu/materiaalit/tuotetiede/html_files/14112_totea.html.

Samual Sam. 2018. C++ Program to Find Fibonacci Numbers using Recursion. Tutorialspoint sivustolla 6.11.2018. Viitattu 1.9.2021. https://www.tutorialspoint.com/cplusplus-program-to-find-fibonacci-numbers-using-recursion.

Steve Sanderson. 2018. Blazor: a technical introduction. Steve Sandersonin blogissa 6.2.2018. Viitattu 31.3.2021 https://blog.stevensanderson.com/2018/02/06/blazor-intro/.

Sunny Khanuja. 2019. Difference Among Angular 8, 7, 6, 5, 4, 3, 2, Breakdown, New Features, and Changes. Artikkeli Medium sivustolla 10.2.2019. Viitattu 21.7.2021. https://medium.com/@lifenshades/difference-among-angular-8-7-6-5-4-3-2-breakdown-new-features-and-changes-811fb5f8e6f0.

Understanding client-side JavaScript frameworks. 2021. Dokumentaatio Mozillan kehittäjäsivustolla 14.4.2021. Viitattu 2.8.2021. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks.

Urvashi. 2020. Loading WebAssembly Modules in JavaScript. Artikkeli Medium sivustolla 30.8.2020. Viitattu 30.7.2021. https://ihsavru.medium.com/loading-webassembly-modules-in-javascript-dc09fbd5eac2.

Use Cases. N.d. WebAssembly websivun dokumentaatio käyttötapauksista. Viitattu 9.2.2021. https://webassembly.org/docs/use-cases/.

Vitaly Friedman. 2021. Front-end performance checklist 2021. Artikkeli Smashing Magazine sivustolla. Viitattu 20.3.2021. https://www.smashingmagazine.com/2021/01/front-end-performance-2021-free-pdf-checklist/#getting-ready-planning-and-metrics.

Walton, P & Mihajlija, M. 2019. Cumulative Layout Shift. Artikkeli web.dev-sivustolla. Viitattu 20.3.2021. https://web.dev/cls/.

WebAssembly Concepts. N.d. Mozillan kehittäjänsivuston dokumentaatio. Viitattu 23.2.2021 https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts.

Liitteet

Liite 1. Testisovellukset

https://github.com/PekToni/wasmtests