Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

JUSSI LAATU

Blazor tuotantovalmius

SÄHKÖ- JA AUTOMAATIOTEKNIIKAN
KOULUTUSOHJELMA
2021

Tekijä(t)
Laatu, Jussi

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
Toukokuu, 2021

Sivumäärä
27

Julkaisun kieli
Suomi

Julkaisun nimi
Blazor tuotantovalmius

Tutkinto-ohjelma
Sähkö- ja automaatiotekniikka

Tiivistelmä

Opinnäytetyön tavoitteena oli tutkia Blazor ohjelmistokehyksen tuotantovalmiutta sekä sitä olisiko se korvaajaksi ReactJS:lle.

Opinnäytetyön ensimmäisessä osassa tutustuttiin Blazor ohjelmistokehykseen ja sen toimintaan. Perehdyttiin Blazorin komponenttipohjaiseen toimintamalliin, komponentin elinkaareen sekä käytiin läpi erilaisia tapoja suorittaa Blazor ohjelmaa. Lisäksi tutustuttiin kolmeen eri komponenttikirjastoon sekä Fluxor tilanhallintaan Blazor sovelluksessa.

Toisessa osassa opinnäytetyötä käytiin läpi tutkimustulokset. Niistä paljastui, että Blazorissa on vielä jonkin verran puutteita, joiden takia en suosittelisi Blazorin käyttöä keskikokoisessa tai sitä isommassa sovelluksessa tuotannossa. Pienemmän kokoisessa sovelluksessa, jossa ei ole monimutkaisia graafeja tai muita haastavia komponentteja on mahdollista toteuttaa ilman isompia ongelmia. Yleisesti Blazor vaikutti erittäin lupaavalta ja uskoisin sen olevan muutamassa vuodessa soveltuva myös haastavampaan käyttöön.

Asiasanat
Blazor, Fluxor

Author(s) Jussi, Laatu	Type of Publication Bachelor's thesis	Date May, 2021
	Number of pages 27	Language of publication: Finnish

Title of publication
Blazor production readiness

Degree programme
Electrical and Automation Engineering

Abstract

The goal of this thesis was to research the production readiness of the Blazor framework and whether it would be good replacement for ReactJS.

In the first part of the thesis, we looked what is Blazor framework and how it works. Blazor's component-based operating model, component life cycle, and couple of ways to run Blazor program were reviewed and in addition, we got introduced in three different component libraries and Fluxor state management in Blazor application

In the second part of this thesis, we reviewed the research results. They revealed that there are still some shortcomings in Blazor, which is why I would not recommend the use of Blazor in a medium- to larger size application in production. In a smaller application without complicated graphs or other challenging components, it is possible to use Blazor without major problems. Overall, Blazor seemed very promising, and I would think it would be suitable for more challenging use in a few years.

Key words
Blazor, Fluxor

Sisällysluettelo

LYHENNELUETTELO

Avoin lähdekoodi (open source)
Julkisesti saatavilla oleva kokoelma koodia, jota voidaan vapaasti käyttää ja muokata tarpeiden mukaan. Avointa lähdekoodia voi ylläpitää yritys, yhteisö tai yksityinen henkilö.

Azure CDN (Content Delivery Network)
Hajautettu verkosto palvelimia, joilla voidaan välittää palvelua loppukäyttäjälle. CDN:llä pyritään saavuttamaan erittäin hyvä saavutettavuus sekä suorituskyky.

Debuggaus (virheenjäljitys)
Virheen paikantamista ohjelmasta.

DOM (Document Object Model)
Standardi, jolla voidaan manipuloida HTML sekä XML-dokumenttien dokumenttipuuta.

GitHub
Nykyään Microsoftin omistama Git-versiohallintaa käyttävien sovellusten tallennus ja jakelupaikka. Sivustolla on myös mahdollista tehdä erilaisia automatisointeja.

Hot reaload
Sovellusta kehittäessä, kun tallentaa muutokset ne tulevat automaattisesti voimaan omassa kehitysympäristössä, ilman että tarvitsee käynnistää sovellusta uudelleen.

Luonti (render)
Tapahtuma, jossa luodaan DOM ohjelmaa suorittaessa.

Ohjelmistokehys (Framework)
Ohjelmistotuote, joka määrittää rungon kehitettävälle sovellukselle. Sen tarkoituksena on toimia kehityksen apuvälineenä. Yleensä ohjelmistokehys sisältää joitakin valmiiksi tehtyjä ohjelmisto-osia kehitystyön helpottamiseksi.

Ominaisuudet (Props / Properties)

Argumenttien tapaisia tiedon välitys osia, joita välitetään komponentilta toiselle HTML:stä tuttujen attribuuttien avulla.

Razor

HTML ja C# kirjoitettava Markup-kieli, jolla voidaan toteuttaa dynaamista verkkosisältöä.

SASS (syntactically awesome style sheets)

Skriptikieli joka, käännetään CSS tyylitiedostoksi.

SPA (Single-page application)

Yhdensivun sovellus eli selaimessa oleva sovellus, jossa kaikki toimita tapahtuu yhdellä sivulla, sivun sisältöä vaihtamalla.

Väliohjelmisto (middleware)

Ohjelmistokomponentti, jota suoritetaan sovelluksen tai sen osien välisenä rajapintana tai palveluna.

Ylikirjoitettava (override)

Avainsana, joka on mahdollista antaa luokan jäsenelle. Tämän jälkeen tätä luokkaa käytettäessä on mahdollista muuttaa kyseisen ”Override” avainsanan omaavien jäsenten toimintaa käyttötarpeiden mukaan.

1 JOHDANTO

JavaScript on hallinnut selainohjelmointikielenä jo yli parikymmentä vuotta. Sen lukuisat ohjelmistokehykset sekä kirjastot ovat tulleet monelle kehittäjälle erittäin tutuksi. Puhtaan JavaScriptin haastajaksi on viime vuosina noussut TypeScript, joka eroaa JavaScriptistä vahvemmalla tyypityksellä, mutta on muutoin pääosin JavaScriptiä vastaava. Näiden kahden lisäksi Microsoft on kehittänyt myös Blazor ohjelmistokehyksen, jonka logiikka kirjoitetaan pääasiassa C# kieltä käyttäen.

Tässä tutkimuksessa tullaan perehtymään Blazor teknologiaan ja sen toimintaan. Käydään läpi muutama Blazor komponenttikirjasto sekä tutustutaan Fluxor tilanhallintaan Blazor sovelluksessa. Tutkimuksen rinnalla tullaan toteuttamaan Blazor sovellus, joka tehdään aikaisemman ReactJS sovelluksen pohjalta. Kolmannessa osassa tullaan tehdyn sovelluksen perusteella käymään läpi Blazor teknologiaan liittyvät havainnot.

Tutkimuksen tavoitteena on selvittää Blazorin tuotantovalmius sekä onko se varteenotettava vaihtoehto ReactJS:n korvaajaksi. Lisäksi pohditaan Blazor sovelluksen kehittämisen mielekkyyttä kehittäjän näkökulmasta. Opinnäytetyön toimeksiantajana toimi Profit Software Oy.

2 BLAZOR

2.1 Mikä on Blazor

Blazor on Microsoftin vuonna 2018 (Microsoft 2018). julkaisema interaktiivisten selain käyttöliittymien tekemiseen tarkoitettu työkalu, joka mahdollistaa selainsovellusten tekemisen C#, HTML sekä CSS ohjelmointikieliä käyttäen. C# kielen käyttö mahdollistaa koodikirjastojen jakamisen palvelin- sekä käyttöliittymäohjelmiston kesken tilanteissa, jossa palvelin ohjelma on kirjoitettu C# ohjelmointikieltä käyttäen. Myös JavaScriptin suorittaminen C# koodin rinnalla on mahdollista ja joskus jopa pakollista Blazorissa olevien puutteiden vuoksi. Näistä puutteista kerrotaan myöhemmässä vaiheessa lisää. (Microsoft 2021 a.)

Blazor on osa avoimeen lähdekoodiin perustuvaa .NET sovelluskehystä. Se on järjestelmäriippumaton, mikä mahdollistaa kehitystyön sekä sovelluksen suorittamisen eri käyttöjärjestelmillä. .NET sovelluskehystä pitää yllä Microsoft sekä .NET yhteisö GitHubissa. (Microsoft 2021 a.)

2.2 Komponentti pohjainen

Blazor ohjelmassa käyttöliittymien tekeminen perustuu komponentteihin. Komponentit mahdollistavat selkeämmän sekä pienemmän koodi määrän, kun usein käytettyjä käyttöliittymä elementtejä on mahdollista käyttää uudelleen komponenttien muodossa. Näiden komponenttien tekeminen sekä niiden käyttäminen muistuttaa ReactJS:tä tuttua tyyliä tehdä ja uudelleen käyttää komponentteja sovelluksen eri osissa.

Komponentti voi sisältää perinteisten HTML elementtien lisäksi muita Blazor komponentteja. Tässä tilanteessa ylemmän tason komponentti on isäntäkomponentti ja se sisältää yhden tai useampia lapsikomponentteja. Isäntäkomponentin on mahdollista lähettää dataa lapsikomponentille propsien avulla. Lapsikomponentin on tämän jälkeen mahdollista muokata tai käyttää suoraan saatua dataa omien tarpeidensa mukaan.

Kuvassa 1. on havainnollistettu, että silloin kun halutaan suorittaa ehdollistaminen tai käyttää muuttujien arvoja sivun luonnissa, tulee muuttujan tai ehtolauseen eteen Razor-tiedostoista tuttu *"@"* merkki. Komponentin koodi, muuttujat sekä parametri tulevat *"@code {}"* ohjelma palikan sisälle tai ne on mahdollista siirtää omaan tiedostoonsa, joka on nimetty vastaamaan komponentin tiedostoa ja lisätty perään pääte *".cs"* esimerkiksi: *"Komponentti.razor.cs"*.

```
4
5   @using Models
6
7   <tr class="header-row" role="row">
8       <td colspan="2">@Item.TypeName</td>
9       <td colspan="1">@Item.ExPostOne.Year</td>
10      <td colspan="2">@Item.ExPostTwo.Year</td>
11      <td colspan="2">@Item.BudgetOne.Year</td>
12      <td class="bg-light-g borderLeft" colspan="2">@Item.ForecastOne.Year</td>
13      <td class="bg-light-g" colspan="2">@Item.ForecastTwo.Year</td>
14      <td class="bg-light-g" colspan="2">@Item.ForecastThree.Year</td>
15      <td class="bg-light-g" colspan="2">@Item.ForecastFour.Year</td>
16      <td class="bg-light-g" colspan="2">@Item.ForecastFive.Year</td>
17  </tr>
18
19  @code {
20      [Parameter]
21      public FinancialTableRow Item { get; set; }
22  }
```

Kuva 1. Blazor komponentti, jossa yksi parametri

2.3 Komponentin elinkaari

Komponenteille on kolme erilaista ylikirjoitettavaa elinkaarifunktiota, joista jokaisesta löytyy sekä synkroninen että epäsynkroninen versio. Nämä ovat nimeltään OnInitialized, OnParametersSet ja OnAfterRender. Näitä funktioita ylikirjoittamalla on mahdollista muuttaa komponentin toimintaa sen eri elinkaaren vaiheissa. (Microsoft 2022 h.)

2.3.1 OnInitialized ja OnInitializedAsync

Tämä funktio kutsutaan vain kerran silloin, kun komponentti luodaan ensimmäisen kerran. Jos isäntäkomponentti muuttaa annettuja parametrejä ensimmäisen luonnin jälkeen tämä vaihe ohitetaan. Tässä vaiheessa komponenttia ei ole vielä lisätty DOM:iin.

Joten JavaScriptin suorittaminen, jossa on riippuvuuksia DOM:ista löytyvään elementtiin ei tässä vaiheessa komponentin elinkaarta voi suorittaa. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

2.3.2 OnParametersSet ja OnParametersSetAsync

Tätä funktiota kutsutaan aina OnInitialized ja OnInitializedAsync funktioiden jälkeen sekä tilanteissa, joissa isäntäkomponentti muuttaa annettuja parametrejä. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

2.3.3 OnAfterRender ja OnAfterRenderAsync

OnAfterRender ja OnAfterRenderAsync funktioita kutsutaan joka kerta kun komponentti palautetaan uudelleen. Tässä vaiheessa komponentin elinkaarta on mahdollista suorittaa JavaScriptiä, joka on riippuvainen komponentin löytymisestä DOM:ista. Tässä kohtaa on myös hyvä suorittaa JavaScript kirjastojen alustus, jos komponentti sellaisia käyttää. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

Näiltä kahdelta elinkaarifunktiolta löytyy myös parametri *”firstRender”*. Tämä on totuusarvo, joka on totta vain ensimmäisen luonnin aikana. Jos komponentti vaatii jotakin tapahtuvan vain ensimmäisen luontikerran aikana, on se järkevää suorittaa tässä vaiheessa sovelluksen elinkaarta. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

2.3.4 Elinkaaren havainnollistaminen

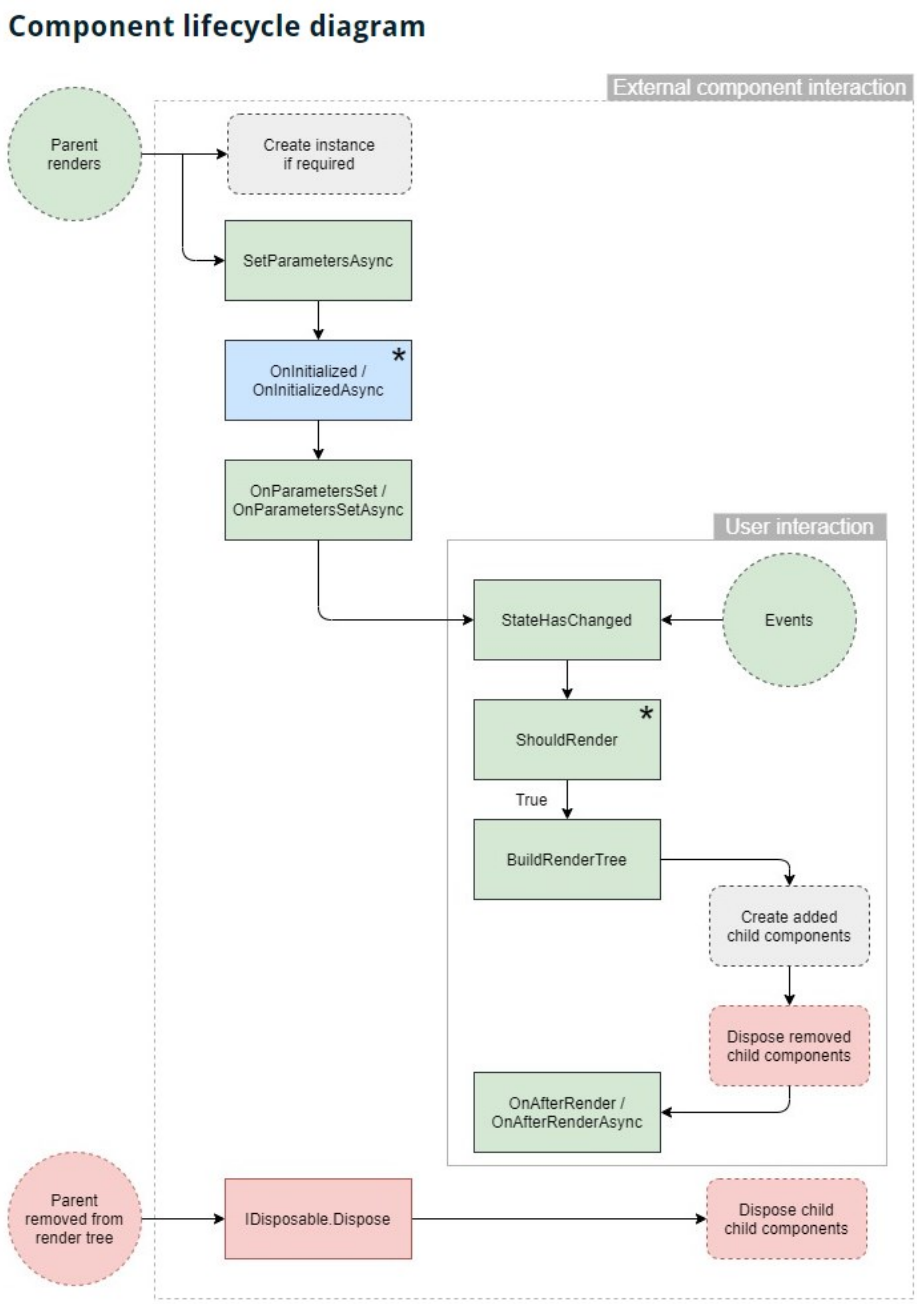
Kuvasta 2 on nähtävissä komponentin eri elinkaaren vaiheiden järjestys, sekä se että elinkaaren aikana tapahtuu muitakin toimenpiteitä, joita ei ole mahdollista koodissa muuttaa. SetParametersAsync suoritetaan aina ensimmäisenä, kun komponenttia palautetaan. Tämän jälkeen suoritetaan aikaisemmin mainitut OnInitialized / OnInitializedAsync ja OnParametersSet ja OnParametersSetAsync. Nämä ovat niin sanotusti komponenttiin ulkoisesti vaikuttavia ominaisuuksia. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

Komponentti kutsuu automaattisesti StateHasChanged funktiota aina, kun se havaitsee, että komponentin tila on riittävästi muuttunut ja uudelleen palauttaminen on tarpeellista suorittaa. StateHasChanged on myös koodissa mahdollista kutsua. Esimerkiksi tilanne, jossa tehdään useampi epäsynkroninen kutsu, on mahdollista suorittaa palautus näiden välissä, joka mahdollistaa käyttäjäystävällisemmän kokemuksen sovelluksessa. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

ShouldRender funktiolla on mahdollista estää komponentin luonnin määrittelemällä se palauttamaan false. Tämä kyseinen funktio on myös yksi ylikirjoitettavista funktioista, joten sen logiikka on helposti muunneltavissa komponentti kohtaisia tarpeita varten. ShouldRender funktiolla onnistuu suorituskyvyn parantamisen toteuttaminen tilanteissa, jossa tiedetään, että seuraavan luonnin tuloksena syntyisi nykytilannetta vastaava lopputulos. On myös hyvä huomioida, että tätä funktiota ei suoriteta ensimmäisellä luonti kerralla. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

BuildRenderTree funktiossa luodaan komponentin sisältö virtuaaliseen DOM rakenteeseen (RenderTree). Tämän jälkeen virtuaalista DOM:ia verrataan selaimessa olevaan DOM rakenteeseen ja tehdään sinne tarvittavat muutokset. (Blazor-University 2019 a, Blazor-Tutorial 2022.)

Tilanteissa, joissa luodaan HTML elementtejä ohjelmallisesti toistaen, on suositeltavaa lisätä elementille attribuutti *”@key”* ja tälle yksilöivä arvo. Tämä parantaa virtuaalisen DOM:in suorituskykyä merkittävästi tilanteissa, joissa sovelluksen suorituksen aikana luodaan uusia tai poistetaan ja muokataan vanhoja elementtejä. Virtuaalinen DOM voi tämän avaimen avulla suorittaa tarvittavat toimenpiteet oikeille elementeille ja turhaa suorittaminen vähenee. (Blazor-University 2019 b.)



Kuva 2. Komponentin elinkaari (Blazor-University 2019.)

2.4 Blazor ohjelman suorittaminen

Blazor sovellusta on mahdollista suorittaa kolmella eri tavalla. Käyttäen uusimpia selain versioita, josta löytyy WebAssembly tuki, suorittamalla koodia palvelimella ja lähettämällä saatu HTML sivu käyttäjän selaimeen tai mobiili laitteessa natiivina sovelluksena.

2.4.1 Hybridi

Blazor hybridi mahdollistaa natiivin mobiili tai työpöytäsovelluksen toteuttamisen. Tämä teknologia on kuitenkin vasta kehitysvaiheessa ja sitä ei suositella käyttämään tuotannossa. Tässä toteutuksessa ohjelmaa ei suoriteta selaimessa eikä WebAssembly ole käytössä. Lisäksi sovelluksen on mahdollista käyttää laitteen eri toimintoja .Net sovelluskehyksen avulla. (Microsoft 2022 c.)

2.4.2 WebAssembly

WebAssembly versiossa Blazorin C# ohjelma suoritetaan käyttäjän selaimessa. Tämä mahdollistaa nopeamman käyttöliittymä kokemuksen verrattuna palvelinversioon. Blazor WebAssemblyllä on myös mahdollista toteuttaa offline sovelluksia. Tällaisen offline sovelluksen jakaminen on mahdollista mm. Azure CDN välityksellä. Tämän opinnäytetyön rinnalla toteutettu Blazor sovellus on toteutettu käyttäen Web-Assembly versiota. (Microsoft 2022 d.)

2.4.3 Palvelinversio

Palvelinversiossa Blazorin ohjelmaa suoritetaan selaimen sijaan palvelimella. Palvelin luo käyttäjälle sivun ja lähettää sen käyttäjän päätelaitteeseen. Käyttäjän ja palvelimen välinen kommunikointi suoritetaan SignalR teknologiaa käyttäen. Koska sovelluksen logiikka suoritetaan palvelimella, voi käyttöliittymän nopeus vaihdella palvelimen ja selaimen yhteyden laadusta riippuen. (Microsoft 2022 d.)

Hyviä puolia palvelinversiosta:

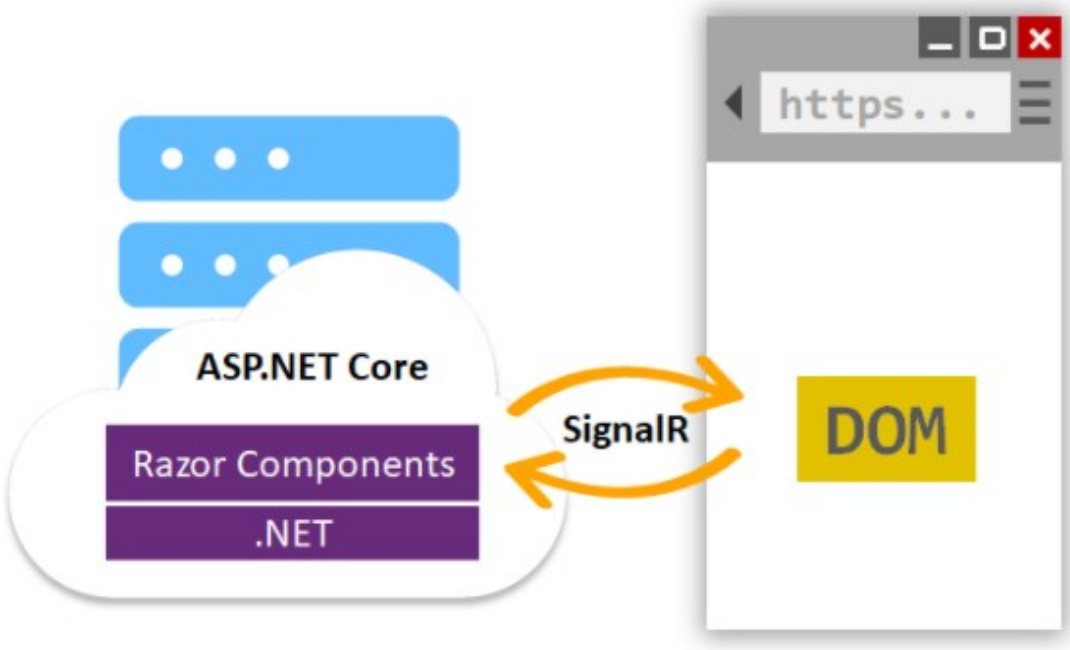
- Pienemmät pakettien koot verrattuna WebAssemby versioon, joten sovellus latautuu nopeammin.
- Sovellukset toimivat myös selaimilla, joissa ei ole WebAssembly tukea.
- Debuggaukselle parempi tuki Visual Studiossa.

Huonoja puolia palvelinversiossa:

- Logiikka suoritetaan palvelimella, joten valtaosasta käyttäjän syötteistä seuraa verkkoliikennettä, joka yhteyden laadusta riippuen hidastaa sovelluksen yleistä toimintaa
- Offline sovellukset eivät ole mahdollisia
- Palvelimetonta mahdollisuutta ei ole saatavilla

2.4.4 SignalR

SignalR on osa ASP.NET sovelluskehystä. Se mahdollistaa reaaliaikaisten sovellusten tekemisen siten, että käyttäjän ei tarvitse päivitellä selaimensivua. Kun palvelimen päässä saadaan uutta tietoa, se lähetetään kaikille aktiivisille käyttäjille, jotka käyttävät silla hetkellä sovellusta. Yleisiä käyttökohteita SignalR:lle ovat sosiaalisenmedian uusien päivitysten automaattinen näyttäminen, pelit, live tilastot sekä ilmoituksia lähettävät sovellukset. Kuvassa 3. havainnollistetaan SignalR teknologiaa käyttävän sovelluksen toimintaa Blazor sovelluksessa. (Microsoft 2022 d.)



Kuva 3. Microsoftin dokumentaatiosta Blazor Server SignalR toimintaa havainnollistava kuva

Blazorin palvelin versiossa käytetään SignalR käyttöliittymän päivittämiseen. SignalR toimii palvelimen ja käyttäjän selaimen väissä. SignalR hoitaa kaikkien käyttäjien liikenteen palvelimelle, jolloin palvelimen ei tarvitse niitä erikseen ylläpitää. (Microsoft 2022 d.)

Kaikki käyttäjän tekemiset käyttöliittymällä lähetetään SignalR välityksellä palvelimelle, jossa ne prosessoidaan ja lopuksi palautetaan muutokset ja liitetään ne käyttäjän DOM:iin. Tämän lisäksi palvelimen on mahdollista lähettää käyttäjälle tai käyttäjille dataa tilanteen vaatiessa. (Microsoft 2022 d.)

2.5 JavaScript käyttö Blazor sovelluksessa

Osa selaimessa tutuksi tulleissa ominaisuuksista on Blazorin nykyisessä versiossa hankala tai mahdoton toteuttaa. Tästä syystä Blazor sovelluksen sisällä on mahdollista suorittaa JavaScript koodia, tuomalla IJSRuntime JavaScriptiä tarvitseville komponenteille. Tämä mahdollistaa JavaScript funktioiden suorittamisen Blazor koodin sisällä. Jos halutaan suorittaa kustomoituja JavaScript funktioita, tulee ne tehdä suoraan index.html sisälle tai linkittämällä JS tiedosto index.html tiedostoon. Moni komponentti kirjasto vaatii toimiakseen JavaScriptiä. Näissä tilanteissa pitää lisätä index.html tiedostoon vaadittava linkki oikeaan tiedostoon, josta kirjaston vaativat JavaScript tiedostot löytyvät.

2.5.1 JavaScriptin kutsuminen Blazor sovelluksessa

Yksi tapa suorittaa JavaScript koodia Blazor sovelluksessa on tehdä wwwroot kansioon JS tiedosto ja linkittää se index.html tiedostossa normaaliin tapaan. Tähän JavaScript tiedostoon on mahdollista tehdä tarvittavat funktiot, joita voidaan kutsua Blazor sovelluksen komponenteilta.

Kuvassa 4 on erilaista esimerkkiä JavaScript funktioista. Funktiossa *”ShowAlertFromJavaScript”* näytetään varoitus ja ei palauteta mitään. *”ShowConfirmFromJavaScript”* näyttää selaimessa vahvista ikkunan ja palauttaa käyttäjän syötteen mukaan totuus arvon tosi tai epätosi. Kolmas funktio *”GetRandomInt”* on esimerkki

tapauksesta, jossa kutsutaan Blazor sivulla olevaa C# koodia, joka palauttaa kokonaisluvun ja sen jälkeen tämä saatu kokonaisluku asetetaan *”document.getElementById”* avulla käyttöliittymälle näkyviin. (Microsoft 2022 f.)

Kuvan 4 ”GetRandomInt” funktiossa oleva *”DotNet.invokeMethodAsync”* mahdollistaa staattisten C# funktioiden kutsumisen JavaScript koodissa. Tässä kaksi ensimmäistä parametriä järjestyksessä ovat: *'{assembly name}', '{.Net method id}'*. Kaikki näiden jälkeen annetut parametrit Blazor käsittelee siten, että ne välitetään kutsutulle funktiolle annetussa järjestyksessä. Esimerkissä maxValue on ainut parametri, joka välitetään *”GetRandomInt”* funktiolle, joka suoritetaan Blazor komponentin C# koodissa. (Microsoft 2022 f.)

```
204
205  ### wwwroot/js/CustomJacaScript.js ###
206  function ShowAlertFromJavaScript(msg) {
207      alert(msg);
208  };
209
210  function ShowConfirmFromJavaScript(msg) {
211      return `Result from JS is: ${confirm(msg).toString()}`;
212  };
213
214  function GetRandomInt(maxValue) {
215      DotNet.invokeMethodAsync("Client", "GetRandomInt", maxValue)
216          .then(result => {
217              document.getElementById("random-int").innerText = result
218          });
219  };
220
```

Kuva 4. Esimerkki JavaScript koodia Blazor sovelluksessa

Kuvassa 5 on neljä JavaScript kutsu esimerkkiä C# koodista. Jotta JavaScript toimii komponentilla, on ensiksi tuotava rajapinta IJSRuntime. Tämä onnistuu lisäämällä *”@inject IJSRuntime JS”* tiedoston alussa. Tämä rajapinta mahdollistaa InvokeAsync ja InvoceVoidAsync funktioiden käytön.

ShowAlert ja ShowAlertJS funktioissa käytetään InvoceVoidAsync funktiota. Näissä kutsuissa ei odoteta takaisin mitään, jotenka ”void” funktion käyttö on mahdollista. Tämä funktio ottaa ensimmäiseksi parametrikseen JavaScript funktion nimen, joko sisäänrakennetun tai kehittäjän oma tekemän funktion nimen. Tämän jälkeiset parametrit välitetään JavaScript funktiolle annetussa järjestyksessä.

ShowConfirm ja ShowConfirmJs funktioissa käytetään InvokeAsync funktiota koska näissä odotetaan totuusarvoa takaisin funktion suorittamisen jälkeen. On myös huomioitava, että InvokeAsync vaatii funktion perään paluuarvo tyyppi määritelmän. InvokeAsync parametrien toiminta on sama kuin InvoceVoidAsync funktiolla.

GetRandomInt on C# funktio, jolle on annettu *[JSInvokable]* attribuutti. Tämän avulla on mahdollista kutsua kyseistä funktiota JavaScript koodista. C# funktioiden kutsuminen mahdollistaa Blazor komponentin tilassa olevien datojen käytön helpommin, kuin JavaScript koodi tietyissä tilanteissa. Funktioita ei myöskään tarvitse kirjoittaa kahteen kertaan molemmille kielille, kun on mahdollista kutsua molemmilla kielillä kirjoitettuja funktioita sekä C# että JavaScript koodista. (Microsoft 2022 e.)

```
155
156 ### Pages/JavaScriptSample.razor ###
157 @page "/JavaScriptSample"
158 @inject IJSRuntime JS
159
160 <div class="js-sample9">
161     <h3>JavaScriptSample</h3>
162
163     <button @onclick="(() => ShowAlert())">Show alert</button>
164     <button @onclick="ShowConfirm">Show confirm</button>
165     <button @onclick="ShowAlertJS">Show alert from custom JS</button>
166     <button @onclick="ShowConfirmJS">Show confirm from custom JS</button>
167
168     <h3>Random int demo</h3>
169     <span id="random-int"></span>
170     <button onclick="GetRandomInt(1000)">Get random value.</button>
171 </div>
172
173 @code {
174     private async void ShowAlert(string msg = "This is alert message!")
175     {
176         await JS.InvokeVoidAsync("alert", msg);
177     }
178     private async void ShowConfirm()
179     {
180         var confirm = await JS.InvokeAsync<bool>("confirm", "Are you sure about this?");
181
182         if (confirm)
183             ShowAlert("Confirm == true");
184         else
185             ShowAlert("Confirm == false");
186     }
187     private async void ShowAlertJS()
188     {
189         await JS.InvokeVoidAsync("ShowAlertFromJavaScript", "This is alert message!");
190     }
191     private async void ShowConfirmJS()
192     {
193         var result = await JS.InvokeAsync<string>("ShowConfirmFromJavaScript", "This is alert message!");
194         ShowAlert(result);
195     }
196     [JSInvokable]
197     public static Task<int> GetRandomInt(int maxValue)
198     {
199         return Task.FromResult(new Random().Next(maxValue));
200     }
201 }
202
```

Kuva 5 Esimerkki JavaScript koodin kutsumisesta Blazor sovelluksessa

2.6 Komponentti kirjastot

Koska Blazor on vielä suhteellisen uusi teknologia, siihen tehtyjä kolmannen osapuolen kirjastoja on rajallinen määrä ja näiden sisältö kirjastosta riippuen on suhteellisen suppea. ReactJS:ään verrattuna varsinkin avoimeen lähdekoodiin perustuvia kolmannen osapuolen kirjastoja on vähäinen määrä. Tässä osiossa käydään läpi kolme kolmannen osapuolen komponenttikirjastoa.

2.6.1 Radzen

Radzen komponentti kirjasto on avointa lähdekoodia ja sen käyttö on ilmaista kaupallisessa toiminnassa. Asentaminen onnistuu NuGet palvelua käyttämällä. Tämä kirjasto sisältää yli 60 komponenttia ja ne eivät ole riippuvaisia mistään JavaScript kirjastosta. Radzen komponentit toimivat sekä palvelin että WebAssembly versiossa. Radzen kirjasto sisältää komponentteja ainakin datan visualisointiin, lomakkeiden tekemiseen liittyviä elementtejä, navigointiin, rakenteeseen ja kuviin liittyviä elementtejä sekä datan esittämiseen liittyviä tauluja ja listoja. (Blazor.Razen 2022.)

2.6.2 Blazorise

Blazorise on avoimeen lähdekoodiin perustuva komponentti kirjasto, joka sisältää yli 70 komponenttia. Blazorise tukee CSS ohjelmistokehyksiä, jotka ovat Material, Bulma, Bootstrap ja AntDesing. Erillisten JavaScript kirjastojen käyttäminen on pyritty pitämään minimissä. Blazorise tukee molempia Blazor kehitys tapoja eli palvelin ja WebAssembly versioita ja se on asennettavissa NuGet palvelun kautta. Blazorise sisältää lomake, navigointi, erilaisia datan visualisointi sekä erilaisia rakenteeseen liittyviä komponentteja. (Blazorise 2022.)

2.6.3 Fluent UI

Fluent UI on Microsoftin ylläpitämä avoimmen lähdekoodin komponentti kirjasto, joka toimii usealla eri kehitys alustalla mm. .NET, ReactJS, Vue ja Blazor. Se on asennettavissa NuGet palvelun kautta se toimii sekä palvelin että WebAssembly versioissa.

Fluent UI sisältää yleisimmät syöte elementit, navigointiin ja rakenteeseen liittyvät elementit sekä erilaisia taulu ja lista näkymiä. Microsoft käyttää Fluent UI kirjastoa monissa sen Office 365 sovelluksissaan (Microsoft 2021 b.)

2.7 Fluxor

SPA, jollainen Blazor sovellus on, yhdeksi haasteeksi muodostuu sovelluksen tilanhallinta. Blazorissa ei ole sisäänrakennettuna ominaisuutta, jolla tämän voisi ratkaista helposti, joten useimmiten tarvitaan ulkopuolinen kirjasto täydentämään tämä puute. (Dev to Eric King 2021).

Fluxor on tähän yksivaihtoehto ja se vastaa toiminnaltaan ReactJS:stä tuttua Reduxia. Fluxorissa on mahdollista väliohjelmiston avulla käyttää Redux:sta tuttua Chrome lisäosaa *"Redux DevTools"*, jolloin sovelluksen tilan havainnollistaminen selaimen avulla on helppoa. (Dev to Eric King 2021).

Fluxor Store koostuu viidestä osasta, jotka ovat State, Actions, Feature, Reducer ja Effect. Seuraavissa kuvissa havainnollistetaan miltä nämä osat näyttävät koodillisesti ja kerrotaan tarkemmin mitä ne tekevät. (Dev to Eric King 2021).

2.7.1 State

State määrittää minkä nimisiä ja tyyppisiä kenttiä tilassa on. State voi olla record kuten kuvassa 6 tai luokka. Tilan kentillä ei saa olla set; toimintoa, koska ainoa tapa, jolla halutaan muuttaa Fluxorin tilaa on Effect- tai Reducer funktioita käyttämällä.

```
35      // State
36      public record GraphState
37      {
38          public bool Loading { get; init; }
39          public bool Initialized { get; init; }
40          public GraphQueryResponse Graph { get; init; }
41      }
42
```

Kuva 6. Fluxor State toteutus

2.7.2 Feature

Kuvassa 7 selviää, että GraphFeature luokka periytyy Feature luokasta, joka on tässä tapauksessa tyyppiä GraphState, joka on määritelty kuvassa 6. Näiden lisäksi luokasta löytyy funktiot GetName ja GetInitialState, jotka Feature luokka vaatii. GetInitialState funktiolla alustetaan tässä tapauksessa GraphStatelle oletus tila.

```
44
45      // Feature
46      public class GraphFeature : Feature<GraphState>
47      {
48          public override string GetName() => "Graph";
49
50          protected override GraphState GetInitialState()
51          {
52              return new GraphState
53              {
54                  Loading = false,
55                  Initialized = false,
56                  Graph = new (),
57              };
58          }
59      }
60
```

Kuva 7. Fluxor Feature toteutus

2.7.3 Effect

Effect luokkaa tarvitaan silloin, kun halutaan hakea staten ulkopuolelta dataa esimerkiksi tehdä http kutsu datanhakua varten. Kuvan 8 esimerkissä GraphState:n tilassa päivitetään ensimmäisenä *”loading = true”*. Sen jälkeen tehdään API kutsu ja päivitetään GraphState:n graph kenttään kutsusta saatu data ja lopuksi päivitetään loading tila false:ksi.

```
92
93      // Effect
94      public class GraphEffects
95      {
96          private readonly HttpClient Http;
97
98          public GraphEffects(HttpClient http)
99          {
100             Http = http;
101         }
102
103         [EffectMethod(typeof(GraphLoadAction))]
104         public async Task FetchGraphData(IDispatcher dispatcher)
105         {
106             dispatcher.Dispatch(new GraphSetLoadingAction(true));
107
108             var graph = await Http.GetFromJsonAsync<GraphQueryResponse>(YritysConsts.graphUrl);
109
110             dispatcher.Dispatch(new GraphSetAction(graph));
111             dispatcher.Dispatch(new GraphSetLoadingAction(false));
112         }
113     }
114
```

Kuva 8. Fluxor Effectin toteutus

2.7.4 Actions

Kuvassa 9 luodaan neljä eri toimintoa (Action). Fluxor käyttää näiden toimintojen nimiä silloin kun halutaan muuttaa (Dispatch) Fluxorin tilaa. Näitä toimintoja suoritetaan käyttäjän napinpainalluksilla, sivulta toiselle siirryttäessä tai kirjoittamalla johonkin sivulla olevaan kenttään.

```
62
63      // Actions
64      public class GraphSetInitializedAction { }
65
66      public class GraphLoadAction { }
67
68      public class GraphSetAction
69      {
70          public GraphQueryResponse Graph { get; }
71
72          public GraphSetAction(GraphQueryResponse graph)
73          {
74              Graph = graph;
75          }
76      }
77
78
79
80
81      public class GraphSetLoadingAction
82      {
83          public bool Loading { get; }
84
85          public GraphSetLoadingAction(bool loading)
86          {
87              Loading = loading;
88          }
89      }
90
```

Kuva 9. Fluxor Action toteutus.

2.7.5 Reducer

Reducer luokka on staattinen ja sisältää funktiot, jotka muuttavat Fluxorin tilaa. Jokaisella funktiolla on parametrinä vähintään state, joka vastaa sen hetkistä Fluxorin tilaa, sekä mahdollisesti action, joka vastaa jotakin kuvassa 6 määriteltyä luokkaa. Tästä esimerkki kuvassa 10.

```
116
117     //Reducers
118     public static class GraphReducer
119     {
120         [ReducerMethod]
121         public static GraphState OnSetGraphData(GraphState state, GraphSetAction action)
122         {
123             return state with
124             {
125                 Graph = action.Graph
126             };
127         }
128
129
130         [ReducerMethod]
131         public static GraphState OnSetGraphLoading(GraphState state, GraphSetLoadingAction action)
132         {
133             return state with
134             {
135                 Loading = action.Loading
136             };
137         }
138
139         [ReducerMethod(typeof(GraphSetInitializedAction))]
140         public static GraphState OnSetGraphInitialized(GraphState state)
141         {
142             return state with
143             {
144                 Initialized = true
145             };
146         }
147     }
148
```

Kuva 10. Fluxor Reducer toteutus

3 MITKÄ OLIVAT TUKIMUKSEN TULOKSET?

3.1 Tutkimuksen kulku

Tässä opinnäytetyön osiossa käydään läpi sovelluksen teon yhteydessä havaitut puutteet sekä asiat, jotka toimivat hyvin Blazor kehitysympäristössä. Ensimmäisenä kerrotaan toiminta Visual Studio 2019 kehitys ympäristön näkökulmasta, jossa tehty sovellus on pääasiassa kehitetty. Tämän jälkeen toisessa osiossa käydään läpi Visual Studio 2022 ympäristöön siirtyminen sekä kerrotaan mitkä Visual Studio 2019 ympäristössä havaitut puutteet on korjattu uudemmassa versiossa. Sovellusta kehitettiin kahdella eri versiolla, koska aloitusvaiheessa Visual Studio 2022 ei ollut vielä julkaistu. Opinnäytetyön tarkoituksena on selvittää Blazor kehitysympäristön tuotantokelpoisuus ja onko se pätevä korvaaja ReactJS kehitysympäristölle.

3.2 Blazor .Net 5

Taulukko 1 .Net5 kehitysympäristön työkalujen versio tiedot

Tuote	**Versio**
Visual Studio 2019	16.11.11
.Net	5.0
Windows 10 Pro	Uusin versio

Taulukosta 1. selviää tässä osiossa käytetyn ympäristön versiotiedot. Blazor ei tarjoa tällä hetkellä mahdollisuutta luoda tyhjää projektia, joten uutta projektia aloittaessa on lähdettävä liikkeelle Blazorin *”Hello world”* aloitus sovelluksesta. Aloitus sovelluksessa on hyvä tapa tutustua Blazorin kansio rakenteeseen sekä komponenttien toimintaan. Uutta tyhjää sovellusta varten tarvitsee poistaa seuraavat tiedostot:

- Shared/SurveyPrompt.razor
- Pages/FetchData.razor
- Pages/Counter.razor
- wwwroot/sample-data

Näiden tiedostojen poistamisen lisäksi on suositeltavaa myös siistiä *”wwwroot/css/app.css”* tiedostosta pois kaikki turhat tyylit.

3.2.1 Havaitut ongelmat

Tässä versiossa suurimmiksi ongelmiksi sovelluksen kehityksen kannalta ilmenivät CSS tyylien komponenttikohtaiset toimimattomuudet ja hot reaload. Komponenttikohtaiset CSS tiedostot ei latautuneet muun sovelluksen mukana ja asian selvittämisenkin jälkeen ei näitä saatu toimimaan halutulla tavalla. Tässä tapauksessa ongelman ratkaistiin ottamalla SASS käyttöön sovelluksessa jonka jälkeen tyylit toimivat kuten niiden oli tarkoitus. SASS mahdollisti myös tyylitiedostojen muokkaamisen sovelluksen ollessa päällä siten, että tyylit tulivat näkyviin, kun niitä muutettiin. Tämä kuitenkin vaati, että sovellusta suoritettiin ei Debug tilassa, jolloin Visual Studion hot reload ei toiminut ja HTML ja C# koodimuutokset vaativat sovelluksen uudelleen käynnistämisen. Hot reload vaati toimiakseen sovelluksen suorittamista Debug tilassa ja silloinkin sovellusta tarvitsi käynnistellä uudelleen, jos tehdyt muutokset olivat isompia. ReactJS:ään verrattuna Blazorin hot reload toimi tässä versiossa erittäin puutteellisesti.

3.3 Blazor .Net 6

Taulukko 2 .Net6 kehitysympäristön työkalujen versio tiedot

Tuote	**Versio**
Visual Studio 2022	17.1.13
.Net	6.0
Windows 10 Pro	Uusin versio

Taulukosta 2. selviää päivitetyn ympäristön työkaluversiot. Sovelluksen päivittäminen .Net5:sta .Net6:seen oli pääosin ongelmatonta. Visual Studio 2022 asentamisen jälkeen, tarvitsee projektin TargetFramework muuttaa .Net 6 versioon sekä päivittää asennetut NuGet paketit. Vain ”Web Compiler” pakettiin, jota käytettiin SASS kirjaston tyylien luomiseen ei ollut saatavilla tälle .Net versiolle toimivaa versiota. Tästä johtuen tarvitsi tämä kirjasto korvata *”CssBuilder”* kirjastolla. Tällä kirjastolla ei ollut mahdollista luoda SASS tiedostosta CSS tiedostoa sovelluksen ollessa päällä vaan se vaati projektin uudelleen käynnistyksen. Tämä ongelma ei kuitenkaan ole Blazor:sta johtuva ja uskon tähän tulevan jokin ratkaisu lähitulevaisuudessa, joka mahdollistaa SASS ja muiden vastaavien tyyli kirjastojen käytön.

Vanhassa versiossa havaitut hot reload sekä CSS ongelmat olivat tässä versiossa merkittävästi paremmin toimivia. CSS tyylejä on uusimmassa versiossa mahdollista muuttaa sovelluksen ollessa käynnissä, jos kirjoitettu CSS on oikein formatoitua, tulee se näkyviin sovellukseen jopa enne tallentamista. Hot reload toimi uudemmassa versiossa huomattavasti paremmin. Se mahdollisti isompienkin koodi muutosten tekemisen ilman että sovellusta tarvitsi käynnistää uudelleen ja tilanteessa, jossa sovellus tarvitsi käynnistää muutoksista johtuen aikaisemmin itse uudelleen osaa Blazor nyt ehdottaa uudelleen käynnistystä. .Net6 versiossa hot reload oli jo paljon käytännöllisempi, mutta ReactJS:ään verrattuna se vaatii vielä jatkokehitystä.

3.4 Blazor kehittämisestä yleisesti

Blazor kehitysympäristön huonoina puolina näen kirjastojen vähäisyyden sekä tämänhetkisen vähäisen käytön. Monimutkaisempia komponentteja kuten graafeja sisältäviä kirjastoja on erittäin rajallinen määrä. Tästä johtuen monimutkaisten sovellusten tekeminen vaatii mahdollisesti huomattavasti enemmän työtä, kun komponentteja tarvitsee kirjoittaa alusta alkaen isompimäärä. Tämänhetkisen vähäisen käytön näen osittain ongelmana, jos tulevaisuudessa käyttäjämäärä ei kasva nousee riski Blazor teknologian kehityksen lopettamiselle tai ainakin sen hidastumiselle.

Yleisesti ottaen Blazor vaikuttaa kuitenkin erittäin lupaavalta. Sen komponentti pohjainen rakenne muistuttaa ReactJS:stä tuttua kehitys tyyliä ja C# mahdollistaa kirjastojen ja luokkien jakamisen käyttöliittymä- ja palvelin koodien kesken. Blazor sovelluksen kehittämisen aloittaminen on helppoa, jos kehittäjältä löytyy aikaisempaa kokemusta C# ohjelmointikielestä sekä käyttöliittymien tekemisestä ReactJS:llä. Ongelmatilanteiden selvittäminen oli pääasiassa vaivatonta ja ratkaisut löytyivät suhteellisen helposti. Pääasiallisesti kehittäminen oli erittäin mieluisaa ja Blazorin opettelu helppoa.

4 POHDINTA

Opinnäytetyön tavoitteena oli tutustua ja selvittää, onko Blazor valmis käytettäväksi tuotantoympäristössä ReactJS korvaajana. Työn alussa käytiin läpi mikä Blazor on, tutustuttiin sen toimintaan sekä tutustuttiin muutamaan kirjastoon, jotka ovat saatavilla Blazorille. Opinnäytetyön lisäksi tästä aiheesta tehtiin olemassa olevan ReactJS sovelluksen pohjalta vastaava uusi Blazor toteutus, jossa perehdyin kehitysympäristön toimintaan sekä itse sovelluksen kehitystyöhön.

Opinnäytetyötä kirjoittaessa suurimmiksi haasteiksi koin hyvien kolmannen osapuolen lähteiden löytämisen. Suurin osa hyvistä artikkeleista oli Microsoftin omasta dokumentaatiosta. Kolmansilta osapuolilta oli hankala löytää ajankohtaista tietoa. Osasyynä tälle varmasti on Blazorin nuori ikä ja sen nopea kehitystahti. Blazorin kannalta haastavinta oli ympäristökohtaiset tyyli ja hot reload ominaisuuksien puutteellinen toimivuus .Net5 ympäristössä. Nämä haitat suurimmilta osin kuitenkin olivat saaneet korjauksen uudempaan versioon siirryttäessä.

Suuripana puutteena Blazor kehityksessä koen olevan kirjastojen vähäisyyden, joilla on mahdollista nopeuttaa sovellusten kehitystyötä. Varsinkin monimutkaisempia graafeja sisältäviä komponenttikirjastoja oli hankala löytää. ReactJS:ään verrattuna moni Blazor kirjasto ei myöskään ollut avointa lähdekoodia, mikä tuo mukanaan omat lisä kustannuksensa.

Blazor sovellusta tehdessäni erityisesti .Net6 versioon paranneltu hot reload sekä siistimpi tiedosto kohtainen sisältö jäivät mieleen. ReactJS:ää ja C# käyttäneenä Blazor kehitysympäristöön kiinnipääseminen tuntui erittäin helpolta ja ohjelman kirjoittaminen tutulta. Lisäsi koen positiivisena asiana mahdollisuuden jakaa kirjastoja ja luokkia palvelin sekä käyttöliittymä ohjelmistojen kesken.

Aikaisemmin mainituista ongelmista huolimatta koen, että Blazorilla on mahdollista korvata ReactJS käyttöliittymä ainakin pienemmän mittakaavan sovelluksissa, joissa ei tarvita haastavia graafeja. Keskikokoisissa ja sitä isommissa sovelluksissa en koe vielä järkeväksi aloittaa Blazorin tuotanto käyttö, koska Blazor kehittyy tällä hetkellä

nopeaa tahtia ja siitä johtuen migraatio ongelmat olisivat todennäköisiä. Blazorin tulevaisuuden kannalta koen, että sen käyttö tulee yleistymään muutaman vuoden sisällä, kunhan Blazor teknologiaa on saatu vielä jonkin verran kehitettyä.

LÄHTEET

Blazor.Razen 2022. Radzen Blazor Components. Haettu 28.3.2022. https://blazor.radzen.com/

Blazorise 2022. Blazorise Component Library. Haettu 28.3.2022. https://blazorise.com/

Blazor-Tutorial 2022. Lifecycle Methods. Haettu 25.3.2022. https://blazor-tutorial.net/lifecycle-methods

Blazor-University 2019 a. Component lifecycles. Haettu 26.3.2022. https://blazor-university.com/components/component-lifecycles/

Blazor-University 2019 b. Optimising using @key. Haettu 26.3.2022. https://blazor-university.com/components/render-trees/optimising-using-key/

Dev to Eric King 2021. Advanced Blazor State Management Using Fluxor, part 2 - Starting with Fluxor. Haettu 15.3.2022. https://dev.to/mr_eking/advanced-blazor-state-management-using-fluxor-part-2-io7

Microsoft 2018. Get started building .NET web apps that run in the browser with Blazor Haettu 20.11.2021. https://devblogs.microsoft.com/dotnet/get-started-building-net-web-apps-in-the-browser-with-blazor/

Microsoft 2021 a. Blazor. Haettu 27.11.2021. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

Microsoft 2021 b. Fluent UI Web Components Overview. Haettu 28.3.2022. https://docs.microsoft.com/en-us/fluent-ui/web-components/

Microsoft 2022 c. Blazor Hybrid. Haettu 22.3.2022. https://docs.microsoft.com/en-us/aspnet/core/blazor/hybrid/?view=aspnetcore-6.0

Microsoft 2022 d. Blazor. Haettu 20.2.2022. https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0

Microsoft 2022 e. Call .NET methods from JavaScript functions in ASP.NET Core Blazor. Haettu 1.4.2022. https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-dotnet-from-javascript?view=aspnetcore-6.0

Microsoft 2022 f. Call JavaScript functions from .NET methods in ASP.NET Core Blazor. Haettu 22.3.2022. https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-javascript-from-dotnet?view=aspnetcore-6.0

Microsoft 2022 g. Razor component lifecycle. Haettu 18.3.2022. https://docs.microsoft.com/en-us/aspnet/core/blazor/components/lifecycle?view=aspnetcore-6.0

Microsoft 2022 h. Razor components. Haettu 11.3.2022. https://docs.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-6.0