SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

TEKNOLOGIAPÄIVITYS – ASP.NET CORE 3.0 & ANGULAR 8

TEKIJÄ/T: Ville Puurunen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Ville Puurunen	
Työn nimi Teknologia Päivitys – ASP.NET Core 3.0 & Angular 8	
Päiväys .12.2020	Sivumäärä/Liitteet 23
Ohjaaja(t) Keijo Kuosmanen ja Veijo Pitkänen	
Toimeksiantaja/Yhteistyökumppani(t) Innofactor Software oy	

Tiivistelmä

Opinnäytetyön aiheena oli web-sovelluksen migraatio teknologioista ASP.NET Core 1.1 sekä Angular 2 -versioihin ASP.NET Core 3.0 sekä Angular 8. Tutkimuksessa käsiteltiin migraatiossa käytettyjä menetelmiä, etenemistä ja lopputulosta. Näissä osa-alueissa pyrittiin vastaamaan kysymyksiin: Miksi migraatio on tarpeellinen? Miten migraatiota lähdetään työstämään? Mitä mahdollisia haittoja ja hyötyjä migraation tekemistä voi seurata?

Migraatio suoritettiin ajanvarausjärjestelmä-sovellukseen, jonka etenemistä tarkasteltiin opinnäytetyössä, mutta pääasiassa se keskittyy teknologioihin tehtyihin muutoksiin sekä migraatiomenetelmiin.
Ensimmäisissä osioissa selostettiin Angular 2, Angular 8, ASP.NET Core 1.1 sekä ASP.NET Core 3.0 teknologioiden piirteitä sekä käytäntöjä.

Tutkimusasetelmaosiossa käytiin läpi migraation tavoite, josta edettiin menetelmiin, joita käytetään migraatiota tehdessä ja joilla migraatiota lähestyttiin.

Tutkimuksen toteuttaminen -osiossa syvennyttiin tarkemmin tärkeisiin kohtiin, joita todellisuudessa tehtiin web-sovellukseen ja miten nämä kyseisin muutokset tehtiin. Luvussa tarkasteltiin joitakin kriittisimpiä muutoksia sekä toiminnallisuuksia, joita muutokset koskivat.
Lopuksi vielä arvioitiin migraation onnistuneisuutta, hyötyjä sekä haittoja. Tässä osiossa myös tehtiin yhteenveto tilanteista, ongelmista sekä onnistumisista, joita migraatiossa kohdattiin.

Avainsanat

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Ville Puurunen			
Title of Thesis Technology Upgrade – ASP.NET Core 3.0 & Angular 0			
Date	10 December 2020	Pages/Appendices	23
Supervisor(s)			
Client Organisation /Partners Innofactor Software oy			

Abstract

The aim of the thesis was to make a migration from older to newer technologies in a web application used for appointments. The original web application was built using ASP.NET Core 1.1 as the site server in the backend and Angular 2 in the frontend. In the migration, the technologies were upgraded to ASP.NET Core 3.0 and Angular 8. The purpose of the migration is to bring more security and performance as the latest versions of the technologies have fixed the shortcomings of the previous versions as well as the risk factors.

The migration was done for Innofactor Software oy as a customer work. It was documented and tested according to the customer's wishes, so that it achieved the desired result which was a more secure and efficient web application. The development environment for the web application was Microsoft Visual Studio in the backend and Visual Studio Code in the frontend. The web application was then released to the Azure server, from where it was taken into production using the Azure swap feature.

As a result of this thesis, the web application became more secure and efficient. The user interface did not change as wanted and all the functions went successfully through testing. Most of the work required a back-end share with the ASP.NET core software framework because its practices had changed significantly.

Keywords

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön tavoitteena oli tehdä migraatio ajanvarauksessa käytettävään web-sovellukseen vanhemmista teknologia versioista uudempiin. Migraation tarkoitus on tuoda lisää tietoturvaa sekä suorituskykyä, koska teknologioiden uusimmissa versioissa on korjattu aikaisempien versioiden puutteita sekä riskitekijöitä.

Web-sovellus on rakennettu käyttäen ASP.NET Core 1.1-teknologiaa sivuston palvelimena eli backend:ssä sekä Angular 2 teknologiaa verkkoselaimessa eli frontend:ssä. Migraatiossa teknologiat nostettiin versioihin ASP.NET Core 3.0 sekä Angular 8.

Työ on tehty Innofactor Software oy yritykselle asiakastyönä. Työ suunniteltiin, dokumentoitiin ja testattiin asiakkaan toiveiden mukaisesti, jotta pääsimme haluttuun lopputulokseen eli tietoturvallisempaan sekä tehokkaampaan ajanvaraus web-sovellukseen.

Kehitysympäristönä web-sovellukselle on käytetty Microsoft Visual Studiota backend:ssä sekä Visual Studio Code:a frontend:ssä. Web-sovellus julkaistiin sittemmin Azure palvelimelle, josta se vietiin tuotantoon Azuren swap ominaisuudella.

1.1 Lyhenteet ja termistö

Frontend: Tällä tarkoitetaan verkkoselaimessa ajettua selainpuolen koodia, jonka rakenne perustuu yleisimmin kolmesta eri osasta. Html vastaa sivun rakenteesta, css vastaa sivun ulkoasusta sekä javascript vastaa sivun tapahtumista ja toiminnallisuudesta. Javascript on korvattu tässä kyseisessä työssä teknologialla Typescript, koska sitä käytetään useasti Angularin kanssa.

Backend: Tällä tarkoitetaan sivuston palvelimella ajettua koodia, joka vastaa useasti kirjautumisesta, integraatioista, tietokannan käsittelystä sekä logiikasta, josta frontend ottaa selvää rapapintakyselyillä.

Migraatio: Laitteiden ja teknologioiden kehitys on todella nopeaa, joten migraatiota käytetään tässä hyväksi. Migraatiolla tarkoitetaan järjestelmien sekä laitteiden siirtämistä uudempiin tai toisenlaisiin teknologioihin. Yleensä migraation syynä on heikko tietoturva, huono suorituskyky tai vanhentuminen.

2 ANGULAR

Angular on avoimen lähdekoodin omaava ohjelmistokehys, joka pohjautuu Typescript-ohjelmointikieleen. Angular julkaistiin vuonna 2016 versiolla Angular 2, joka oli uudelleen kirjoitus AngularJS ohjelmistokehyksestä, uusin versio Angular 9 julkaistiin 2020. Angularin ylläpidosta sekä kehityksestä vastaa Googlen Angular Team. (Angular Documentation)

2.1 Angular 2

Angular 2 eli uudelleenkirjoitus AngularJS ohjelmistokehyksestä on Typescript-pohjainen ohjelmistokehys, joka rakennettiin komponentti käsitteen ympärille. Se on skaalautuvampi, nopeampi sekä modernimpi ohjelmistokehys, kuin edeltäjänsä ja se sopeutuu hyvin web, mobiili sekä työpöytä -sovellusten kehittämiseen. Hierarkiset riippuvuusinjektiot ovat myös tuotettuja ominaisuuksia.

Kuvassa (Kuva 1) näemme esimerkin komponenttirakenteesta, jossa on määritelty selector 'admin', jonka avulla komponenttia voidaan kutsua toisesta tiedostosta. templateUrl määrittää html tiedoston sijainnin sekä styleUrls määrittää tyylitiedoston sijainnin.

```
@Component({
    selector: 'admin',
    templateUrl: 'admin.component.html',
    styleUrls: ['admin.component.scss']
})
```

Kuva 1. Admin-component.ts tiedostossa olevat selector, templateUrl sekä styleUrls. (Puurunen Ville, 2020)

Riippuvuusinjektiot ovat todella olennainen osa sovelluksen toimivuutta ja näiden avulla voidaan jakaa komponentteja tuomalla sekä injektoimalla haluttuun tiedostoon. Alla olevissa kuvissa on esimerkki importtauksesta (Kuva 2) sekä injektoimisesta, jossa kyseisessä tiedostossa on haluttu käyttää Service:ä BookingService.

```
import { BookingService } from '../../services/booking.service';
```

Kuva 2. Import lauseke (Puurunen Ville, 2020)

```
constructor(
    public translate: TranslateService,
    public bookingService: BookingService) {
}
```

Kuva 3. Constuctor (Puurunen Ville, 2020)

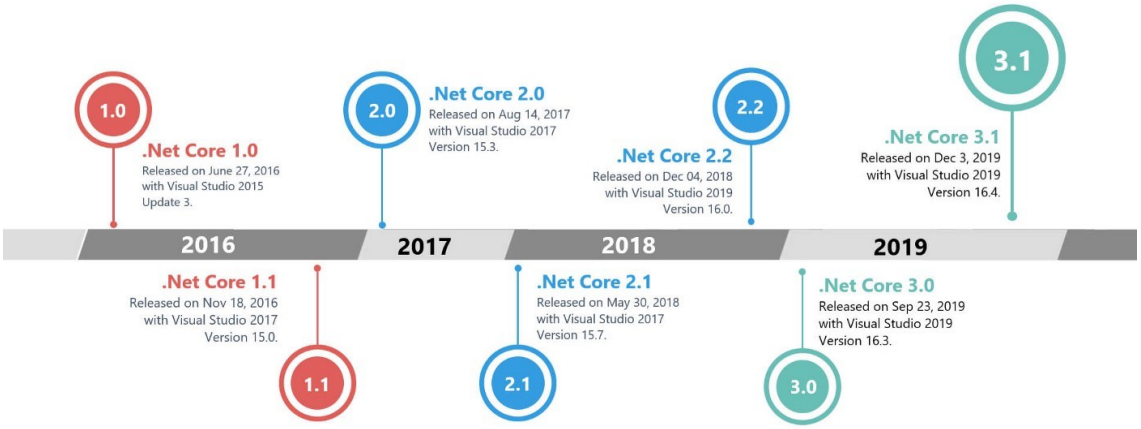
2.2 Angular 8

Angular 8 julkaistiin 2019 ja sen toimintaperiaate on samanlainen, kuin aikaisemmissa Angular-versioissa, mutta siihen on tehty parannuksia sekä uusia ominaisuuksia. Versioiden Angular 2 sekä Angular 8 välissä on julkaistu versiot 4, 5, 6 ja 7.

Angular 8-versiossa parannuksia olivat generoidun koodin helppolukuisuus, debuggauksen helppous, nopeampi re-build aika, parannettu hyötykuorman koko, parannettu tyyppitarkastus sekä monia muita muutoksia, joita käsittelemme tarkemmin Angular migraatio sekä menetelmät osioissa.

3 ASP.NET CORE

ASP.NET Core on avoimen lähdekoodin omaava ohjelmistokehys, jonka kehityksestä vastaa Microsoft. Ensimmäinen versio ASP.NET Core 1.0 julkaistiin 2016 kesäkuussa ja tämän jälkeen tulivat versiot 1.1(11.2016), 2.0(8.2017), 2.1(5.2018), 2.2(12.2018), 3.0(9.2019) sekä 3.1(12.2019). Julkaisuaikajana näkyy alla olevassa kuvassa (Kuva 4). ASP.NET Core:a voidaan kehittää kielillä C#, Visual Basic sekä F#, joista yleisimmin käytetään kieltä C#. (.NET Core overview).



Kuva 4. Asp.Net Core History

ASP.NET Core on uudelleenkirjoitus aikaisemmista erillisistä kehyksistä ASP.NET MVC sekä ASP.NET-Web API, jotka yhdistettiin yhdeksi ohjelmointimalliksi. Vaikka ASP.NET Core on uusi ohjelmistokehys, niin se on yhteensopiva aikaisempien versioiden kanssa. Rinnakkaisversioita voidaan ajaa samassa koneessa ASP.NET Core:n kanssa. Tämä on uusi ominaisuus, joka ei ollut mahdollinen aikaisemmissa ASP.NET-versioissa. (.NET Core overview).

ASP.NET Core:n ominaisuuksia.

- Käännöskomentoa ei tarvitse kutsua, vaan sovelluksen kokoaminen on jatkuvaa
- Sisäinen tuki riippuvuusinjektiolle
- Internettiä varten optimoitu ajoaika
- Avoin lähdekoodi
- Hostaus tapahtuu IIS avulla tai sisäisesti
- Sisäinen tuki riippuvuusinjektiolle
- Sovelluksia voidaan ajaa Windows-, Mac- sekä Linux-ympäristöissä
- Tukee rinnakkaissovelluksen versiointia
- Hyvä suorityskyky, turvallisuus sekä vähä kustanteinen
- Käyttää NuGet-paketteja. Voidaan valita vain tarvittavat sovelluksen optimointia varten

3.1 ASP.NET Core 1.1

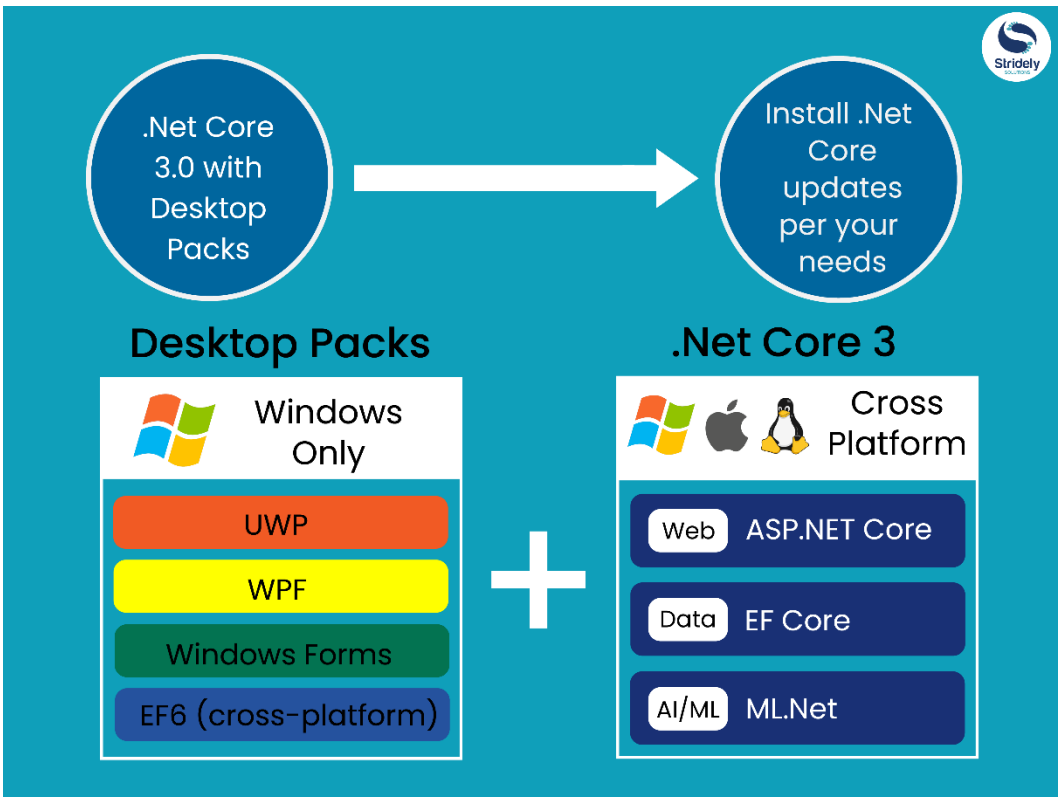
ASP.NET Core 1.1 on 1.0 versiota seuraava versio, joka julkaistiin 2016 marraskuussa. Siihen tuotiin useita uusia ominaisuuksia, jotka helpottavat ASP.NET Core:n käyttöä ja luovat uusia mahdollisuuksia ohjelmistokehityksessä. (What's new in ASP.NET Core 1.1.)

Tärkein uusi ominaisuus oli URL-osoitteiden uudelleenkirjoittaminen. Tämä mahdollisti URL-osoitteiden muokkaamisen ennalta määritettyjen sääntöjen perusteella. Tällä voidaan korvata, siirtää ja optimoida URL-osoitteille tarkoitettuja resursseja pyyntöjä varten.

ASP.NET Core 1.1 uusina ominaisuuksina myös URL osoitteiden uudelleenkirjoittaminen sekä pyyntöjen muokkaaminen, View-komponentit, kustomoidut filtterit, evästepohjainen TempData-palveluntarjoaja, mahdollisuus kirjautua Azure App-palveluun ja WebSocket-tuki. Nämä ominaisuudet loivat enemmän mahdollisuuksia kehittäjälle sekä antoivat uusia mahdollisuuksia erilaisten tarpeitten täyttämiseen ohjelmistokehityksessä.

3.2 ASP.NET Core 3.0

ASP.NET Core 3.0 julkaistiin 2019 syyskuussa ja sitä edeltävä versio oli ASP.NET Core 2.2. ASP.NET Core 3.0 tuli paljon uusia ominaisuuksia, sekä parannuksia edelliseen versioon nähden. (What's new in ASP.NET Core 3.0.)



Kuva 5. Primary Features of .NET Core 3.0

Suurimpina muutoksina sekä uusina ominaisuuksina ASP.NET Core 3.0 versiossa toimi (Blazor) uusi kehys web-liittymän rakentamiseen, (gRPC) tehokas etäproseduuripuhelu-kehys, (C# 8.0) päivitetty ohjelmistokieli uusilla ominaisuuksilla, suoritettavien tiedostojen rakentaminen oletusarvoisena, .NET Core SDK Windows lataaja sekä TLS 1.3-tuki. Nämä ominaisuudet lisäsivät mahdollisuuksia luoda tietoturvallisempi, helpommin ymmärrettävä sekä suorituskykyisempi ohjelmistokoodi.

4 TYÖN TAVOITTEET

4.1 Migraation tavoite

Migraation tavoitteena oli lähteä päivittämään ajanvaraus web-sovelluksessa käytettyjen teknologioiden Angular 2 sekä ASP.NET Core 1.1 versioihin Angular 8 sekä ASP.NET Core 3. Ennen työn aloittamista yrityksessä keskusteltiin sisäisesti migraation, mitä hyötyjä sekä mahdollisia haittoja migraatiosta seuraa, kuinka työläs migraatiosta tulisi, miten mahdolliset haitat saadaan minimoitua sekä kuinka migraatio tehdään.

Migraation suurin peruste oli tietoturvan vahvistaminen sekä käyttöiän pidentäminen. Tietoturva vanhemmissa ohjelmistokoodiversioissa oli uudempiin verrattuna heikompi, koska uudemmissa versioissa kaikki löydetyt riskit ovat poistettu sekä mahdolliset vanhentuneet kirjastot ovat poistettu käytöstä, koska näistä on voinut löytyä haavoittuvaisuuksia. Käyttöiän pidentäminen näkyy ohjelmistokehysten tuen päättymisessä, koska vanhempia versioita ei enää päivitetä tai niistä löytyneitä puutteita ei enää korjata. Uusimpia ohjelmistokehyksiä taas päivitetään jatkuvasti, joten niiden elinkaari tulee kestämään pidemmälle, kuin vanhempien, joiden elinkaaren loppu on jo tavoitettu.

Tietoturvan vahmistamisen sekä käyttöiän pidentämisen lisäksi migraatiolla toimi perusteina suorituskyvynparantaminen, jotta mahdollisilta vanhojen teknologioiden aiheuttamilta hitausongelmilta pystyttäisiin välttymään jatkossa sekä sovelluksesta saataisiin käyttäjäystävällisempi sekä sulavampi.

Migraatiosta ajateltiin olevan hyötyä myös muille ohjelmoijille tai jatkokehitystä varten, koska yrityksen sisällä olevien ohjelmien on hyvä käyttää samoja versioita ohjelmistokoodeista, jotta tulevaisuudessa tulevia mahdollisia kehitysideoita on helpompi toteuttaa muiden työntekijöiden puolesta. Kun ohjelmistokoodin versiot ovat samat, on ohjelmistokoodia helpompi ymmärtää, koska versiot ja käytännöt ovat tuttuja muista sovelluksista.

4.2 Menetelmät

Migraatioon käytettävä menetelmä oli hyvin keskeinen asia toteuttamisen kannalta. Tätä lähestyttiin miettien, kuinka migraatio saataisiin tehtyä mahdollisimman vähällä työmäärällä sekä ilman, että ohjelmistokoodia joudutaan rakentamaan kokonaan uudelleen. Tärkeänä asiana oli myös löytää uudet versiot kirjastoille, joita web-sovellus käytti, koska useat niistä oli päivitettävä myös ohjelmistokoodin lisäksi.

Ensiksi menetelmänä mietittiin ohjelmistokoodin täyttä uudelleenrakentamista, koska migraatiossa tapahtuva versiohyppy oli suhteellisen iso huomioiden, että ohjelmistokehysten käytännöt ja toimintamallit olivat muuttuneet huomattavan paljon. Tätä menetelmää ei kuitenkaan otettu käyttöön, koska kansiorakenteen ja projektin kokonaisrakenne ei ollut muuttunut niin paljoa, että uudelleenrakentaminen olisi ollut työmäärän syystä kannattava ratkaisu. Monet ratkaisut olivat toimivia myös uudempien versioiden kanssa ja toimintaperiaate ei ollut muuttunut niin paljoa, jotta tämä menetelmä olisi ollut tarpeelliset, koska kyseessä oli kuitenkin migraatio, jossa versiot päivitettiin. Jos kyseessä olisi ollut migraatio toiseen ohjelmistokehykseen, silloin uudelleenrakentaminen olisi ollut paras ratkaisu.

Menetelmistä kaikkein sopivimmaksi löytyi versioiden päivittäminen migraatio ohjekirjojen avulla. Näistä ohjekirjoista saa suuren avun, koska niiden avulla näemme versioiden nostojen jälkeiset muutokset ja kuinka ne muuttavat ohjelmistokoodissa käytettyjä periaatteita. Kansiorakenne päätettiin pitää samanlaisena sekä komponenttien asettelu, joten koodia ei lähdetty uudelleenrakentamaan. Migraatio tekeminen suunniteltiin vaiheisiin, joissa ensimmäiseksi lähdettiin korjaamaan muuttuneet ohjelmistokoodin käytännöt. Sitten teknologiat nostettiin tuleviin versioihin ja ohjelmistokoodiin todennäköisesti tulevia virheitä lähdettiin korjaamaan.

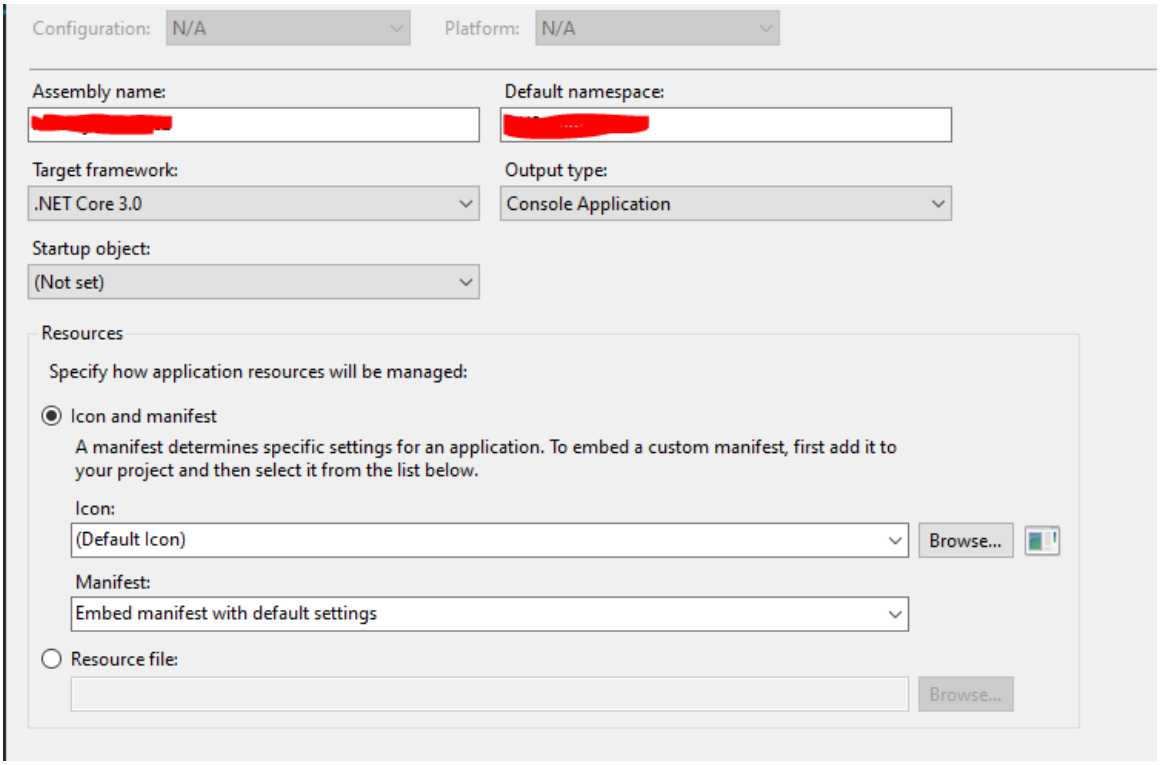
5 MIGRAATION TOTEUTTAMINEN

5.1 ASP.NET Core migraatio

Migraatio lähdettiin tekemään ensiksi back-end puolella, koska sen avulla saamme projektin kääntymään, jotta pääsemme näkemään tulevia korjattavia ongelma tilanteita toimivuuden kannalta. ASP.NET Core 3.0 versioon nostaessa muutoksia on tapahtunut paljon startup.cs tiedostoon ja sen toimintaan sekä käytäntöihin, niin tämän tiedoston virheiden korjaaminen oli hyvin olennainen osa, jotta pääsemme tekemään kehitystä myös projektin muilla osa-alueilla.

5.1.1 ASP.NET Core-Versionnosto

Migraatio aloitettiin nostamalla projektin ASP.NET Core ohjelmistokehys versiosta 1.1 versioon 3.0. Tämä tapahtuu menemällä projektin ominaisuuksiin ja muuttamalla Target Framework arvo haluttuun versioon. Alla kuva (kuva 6) version 3.0 asettamisesta.



Kuva 6. Target Framework. (Puurunen, 2020)

Kun asetus laitetaan voimaan, muutokset näkyvät projektin .csproj tiedostossa TargetFramework arvon kohdalla. Tämän jälkeen täytyy myös päivittää ASP.NET Core:n käyttämät NuGet paketit Visual Studiosta löytyvän NuGet Package Managerin avulla, jolla voidaan lisätä, poistaa sekä päivittää paketteja.

Tämän osuuden muutokset pääsemme sitten näkemään .csproj tiedostosta sekä sieltä voimme myös tarkistaa, että versionnosto sekä NuGet pakettien päivitys tapahtui onnistuneesti. Alapuolella näemme kaksi kuvaa, joista kuva 7 näyttää tilanteen ennen versionnostoa sekä kuva 8 näyttää tilanteen versionnoston jälkeen. Näistä kuvista myös huomaamme, kuinka paljon NuGet paketit ovat muuttuneet versioiden 1.1 sekä 3.0 välillä.

```
- <Project ToolsVersion="15.0" Sdk="Microsoft.NET.Sdk.Web">
-   <PropertyGroup>
-     <TargetFramework>netcoreapp1.1</TargetFramework>
-     <TypeScriptCompileBlocked>true</TypeScriptCompileBlocked>
-     <IsPackable>false</IsPackable>
-     <ApplicationInsightsResourceId>/subscriptions/[redacted]/resourcegroups/A
-     <ApplicationInsightsAnnotationResourceId>/subscriptions/[redacted]cbce8/resourc
-   </PropertyGroup>
-   <ItemGroup>
-     <PackageReference Include="IdentityServer4" Version="1.5.2" />
-     <PackageReference Include="jose-jwt" Version="2.3.0" />
-     <PackageReference Include="MailKit" Version="1.16.1" />
-     <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.2.0" />
-     <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
-     <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="1.1.2" />
-     <PackageReference Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="1.1.2" />
-     <PackageReference Include="Microsoft.AspNetCore.Identity" Version="1.1.2" />
-     <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />
-     <PackageReference Include="Microsoft.AspNetCore.Owin" Version="1.1.2" />
-     <PackageReference Include="Microsoft.AspNetCore.ResponseCaching" Version="1.1.2" />
-     <PackageReference Include="Microsoft.AspNetCore.SpaServices" Version="1.1.0" />
-     <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />
-     <PackageReference Include="Microsoft.EntityFrameworkCore" Version="1.1.1" />
-     <PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="1.1.1" />
-     <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="1.1.1" />
-     <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="1.1.1" />
-     <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer.Design" Version="1.1.1" />
-     <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="1.1.0" />
-     <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />
-     <PackageReference Include="Newtonsoft.Json" Version="10.0.2" />
-     <PackageReference Include="Serilog.Extensions.Logging" Version="1.4.0" />
-     <PackageReference Include="System.Data.SqlClient" Version="4.3.0" />
-   </ItemGroup>
-   <ItemGroup>
```

Kuva 7. .csproj ennen nostoa. (Puurunen, 2020)

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.0</TargetFramework>
  <MvcRazorCompileOnPublish>false</MvcRazorCompileOnPublish>
  <!-- <RazorCompileOnPublish>false</RazorCompileOnPublish> -->
  <!-- <TypeScriptCompileBlocked>true</TypeScriptCompileBlocked> -->
  <TypeScriptToolsVersion>Latest</TypeScriptToolsVersion>
  <IsPackable>false</IsPackable>
  <SpaRoot>ClientApp\</SpaRoot>
  <DefaultItemExcludes>$(DefaultItemExcludes);$(SpaRoot)node_modules\**</DefaultItemExcludes>
  <RuntimeIdentifier>win10-x64</RuntimeIdentifier>
  <!-- Set this to true if you enable server-side prerendering -->
  <BuildServerSideRenderer>false</BuildServerSideRenderer>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="IdentityModel" Version="4.0.0" />
  <PackageReference Include="jose-jwt" Version="2.4.0" />
  <PackageReference Include="MailKit" Version="2.3.2" />
  <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.8.2" />
  <PackageReference Include="Microsoft.AspNetCore.Authentication.Core" Version="2.2.0" />
  <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="3.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="3.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Http.Abstractions" Version="2.2.0" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc.Core" Version="2.2.5" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation" Version="3.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.SpaServices.Extensions" Version="3.0.0" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation" Version="3.0.0"/>
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="3.0.0">
```

Kuva8. .csproj noston jälkeen. (Puurunen, 2020)

5.1.2 Startup tiedoston muuttaminen.

Startup tiedostostossa kiinnitetään ensiksi huomio Main metodiin, jota on yksinkertaistettu huomattavasti versioiden välillä. ASP.NET Core 1.1 versiossa Kestrel-, IISIntegration- sekä ContentRoot-kutsut tehtiin main metodissa hostin luonnin yhteydessä ennen host.Run() kutsua. Kuva 9 näyttää tilanteen versiossa 1.1.

```
public class Startup
{

    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel(k => k.AddServerHeader = false)
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseIISIntegration()
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
```

Kuva 9. ASP.NET Core 1.1 main metodi (Puurunen, 2020)

ASP.NET Core 3 versioon siirryttäessä main metodi ei vaadi näin montaa kutsua, vaan siinä riittää ainoastaan kutsut UseStartup sekä Build (Kuva 10).

```
public class Startup
{

    public static void Main(string[] args)
    {
        var host = WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
```

Kuva 10. ASP.NET Core 3.0 main metodi (Puurunen, 2020)

Tässä huomaamme myös, kuinka versioiden välillä ohjelmistokehystä on päivitetty yksinkertaisemmaksi ja helpommin ymmärrettävämmäksi. Main metodi on nyt hyvin helppo ymmärtää sekä erilaisten kutsujen käyttö on poistettu.

Main metodin muokkaamisen jälkeen siirryimme katsomaan auhentikointiin liittyneet muutokset. ASP.NET Core 1.1 versiossa autentikointi tapahtui yhdellä kutsulla, jossa konfiguroitiin kaksi ominaisuutta AutomaticAutenticate sekä AutomaticChallenge luokasta AuthenticationOptions. Tästä näemme esimerkin alla olevassa kuvassa Kuva 11., joka on otettu projektin versiosta 1.1.

```
.AddAuthenticationSchemes(JwtBearerDefaults.AuthenticationScheme).RequireClaim("hetu")
.RequireAuthenticatedUser()
.Build());
```

Kuva 11. Authentication scheme 1.1(Puurunen, 2020).

Tähän tapahtui muutos jo ennen ASP.NET Core 3.0 versiota versiossa 2.0, jossa AddAuthentication metodi muutettiin ”overload” versioksi, jossa asetetaan enemmän kuin yksi ominaisuus. Kuvassa (Kuva 11) on esimerkki, kuinka autentikointi muutettiin migraation yhteydessä.

```
services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
```

Kuva 12. Authentication scheme 3.0 (Puurunen, 2020)

Siirryttäessä versioon myös 2.2 versiossa vielä olleesta UseMvc metodista luovuttiin. Tätä metodia käytettiin määrittämään sovelluksen reitit erilaisten ohjainten kutsumista varten. (Kuva 13).

```
        app.UseStaticFiles();
        app.UseResponseCaching();
        app.UseMvc(routes =>
        {
        routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");

        routes.MapSpaFallbackRoute(
                    name: "spa-fallback",
                    defaults: new { controller = "Home", action = "Index" });
        });
    }
}
```

Kuva 13. 1.1 endpoints (Puurunen, 2020)

Versiossa 3.0 kutsu muutettiin vastaavanlaiseksi (Kuva 14).

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapRazorPages();
    endpoints.MapControllerRoute(
        name: "api",
        pattern: "api/{controller}/{action=Index}/{id?}");
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller}/{action=Index}/{id?}");

});
```

Kuva 14. 3.0 endpoints (Puurunen, 2020)

Tässä UseMvc tai UseSingalR korvattiin UseEndpoints kutsulla. Huomioitavaa oli, että autentikointi kutsu sovellukselle on tehtävä ennen UseEndpoints kutsua, koska muuten autentikointi ei olisi toiminut halutulla tavalla.

5.1.3 Muut ohjelmistokoodimuutokset

Startup.cs tiedoston lisäksi jouduttiin päivittämään joitakin osia back-end ohjelmistokoodin osalta. Projektissa käytetyt NuGet kansiot päivittyivät ja nämä myös vaikuttivat ohjelmistokoodin toimivuuteen, koska monien pakettien kutsut sekä käytännöt olivat muuttuneet. Pieniä muutoksia tuli hyvinkin paljon ja tässä osiossa esitellään näistä esimerkkejä.

JWT-tokenin purkamiseen käytetty kutsulauseke muuttui paketin päivittämisen mukana hieman sekä tokenin eteen muodostunut väli puuttui päivitetyssä versiossa, joten tämä korjattiin pienellä ohjelmistokoodimuutoksella. Alhaalla näemme kaksi kuvaa (Kuva 15, Kuva 16) tilanteesta ennen sekä jälkeen.

```
try
{
    Microsoft.Extensions.Primitives.StringValues re = "";
    Request.Headers.TryGetValue("Authorization", out re);
    var v = re.ToString().Replace("Bearer ", "").Trim();
    var d = Jose.JWT.Decode(v, TokenAuthOption.CryptoKey);
    return d;
}
catch (Exception ex)
{
    throw ex;
}
```

Kuva 15. Token purku 1.1 (Puurunen, 2020)

```
try
{
    Microsoft.Extensions.Primitives.StringValues re = "";
    Request.Headers.TryGetValue("Authorization", out re);
    var v = re.ToString().Replace("Bearer", "").Trim();
    var d = JWT.Decode(v, TokenAuthOption.CryptoKey);
    return d;
}
catch (Exception ex)
{
    throw ex;
}
```

Kuva 16. Token purku 3.0 (Puurunen, 2020)

5.2 Angular migraatio

Back-end migraation jälkeen lähdettiin suorittamaan front-end migraatiota Angular ohjelmistokehyksen kanssa. Angular 2 versio oli tarkoitus päivittää versioon 8 ja tärkeimpinä kohtina tässä olivat erilaisten käytäntöjen muuttaminen, käyttämien kirjastojen muuttaminen sekä Angularin nostaminen versioon 8.

5.2.1 Ohjelmistokoodi muutokset ennen version nostoa

Ennen Angular:n päivittämistä versioon 8 piti huomioida erilaiset ohjelmistokoodin käyttämien service:n sekä templaattien kutsujen muutokset. Template tagit muuttuivat versioiden noston yhteydessä, joten ensiksi kaikki template tagit oli muutettava ng-template tageiksi. Tämä tapahtui helposti nimeämällä muutos ohjelmistokoodiin.

Moduulit sekä service:t olivat muuttuneet Angular-versioiden välillä huomattavasti ja näistä isoimmat muutokset tulivat HttpModule sekä http servicen käyttöön. HttpModule täytyi vaihtaa moduuliin nimeltä HttpClientModule sekä http service täytyi vaihtaa HttpClient service:en. Tämä oli hyvin helppo muutos, joka tehtiin tiedostoon app.module.ts. Tiedostosta täytyi vaihtaa import lausekkeen kutsut

uusiin nimiin, josta näemme alhaalla olevassa kuvassa (Kuva 17) esimerkin tilanteesta, jossa punainen (-) rivi on Angular 2 version käyttämä import lause sekä vihreä (+) rivi Angular 8 version käyttämä lause.

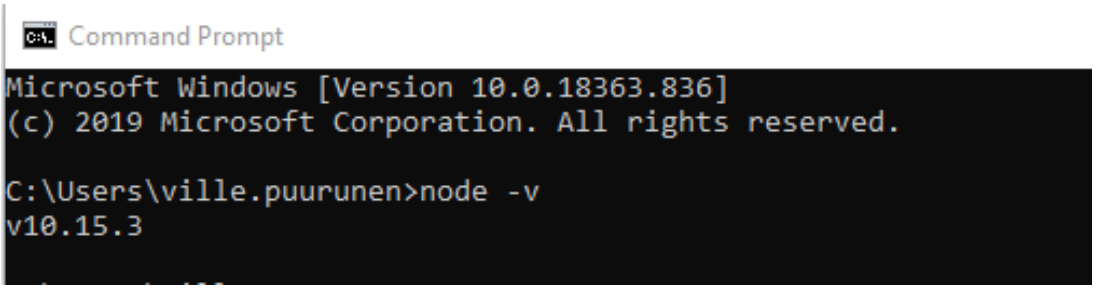
```
1   1     import { NgModule, Inject } from '@angular/core';
2   2     import { RouterModule } from '@angular/router';
3   3     import { CommonModule, APP_BASE_HREF } from '@angular/common';
4       - import { HttpModule, Http, RequestOptions } from '@angular/http';
    4   + import { HttpClientModule, HttpClient, RequestOptions } from '@angular/http';
5   5     import { FormsModule } from '@angular/forms'
6   6     import { Ng2BootstrapModule, DatepickerModule, ModalModule } from 'ngx-bootstrap';
7   7     import { BsDropdownModule } from 'ngx-bootstrap/dropdown';
```

Kuva 17. Angular module sekä service muutos (Puurunen, 2020)

App.module.ts tiedostossa tapahtuva muutos piti myös viedä kaikkiin tiedostoihin, jotka käyttävät kyseistä moduulia tai serviceä, mutta tätä lähestytään ”5.2.3 Ohjelmistokoodi muutokset version noston jälkeen” otsikon alla.

5.2.2 Angular versionnosto

Ennen Angular-version nostamista ensiksi piti päivittää Node.js vähintään versioon 10 ja tämä tapahtui Windows-työasemalla seuraavasti. Menimme Node.js kotisivuille lataamaan Windows Installer:in uusimmalle node versiolle ja tämä ajettiin paikallisesti, kunnes lataus oli suoritettu. Latauksen suorittamisen jälkeen pystyimme tarkistamaan Node.js version komentokehotteen avulla alla olevan kuvan (Kuva 18) mukaisesti komennolla ”node -v”. Kuvasta näemme, että meillä on nyt käytössä versio 10.15.3, joka on riittävä Angular 8 versiota varten



Kuva 18. Node.js versionumero (Puurunen, 2020)

Kun Node.js versio oli päivitetty, terminaaliin ajettiin komento ”ng update @angular/cli@8 @angular/core@8”. Tämä nostaa käytössämme olevan Angular-version haluttuun versioon 8 ja muutoksen voimme tarkistaa sitten package.json tiedostosta (Kuva 19), jossa näemme Angular-version, sekä Angularin käyttämien riippuvaisuuksien versiot.

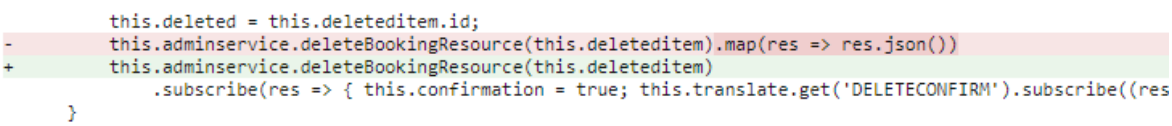
```
{
  "name": "[redacted]",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "build:ssr": "ng run [redacted]server:dev",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "8.0.0",
    "@angular/common": "8.0.0",
    "@angular/compiler": "8.0.0",
    "@angular/core": "8.0.0",
    "@angular/forms": "8.0.0",
    "@angular/platform-browser": "8.0.0",
    "@angular/platform-browser-dynamic": "8.0.0",
    "@angular/platform-server": "8.0.0",
    "@angular/router": "8.0.0",
    "@nguniversal/module-map-ngfactory-loader": "8.0.0-rc.1",
    "aspnet-prerendering": "^3.0.1",
    "bootstrap": "^4.3.1",
    "core-js": "^2.6.5",
    "jquery": "3.4.1",
    "oidc-client": "^1.9.0",
    "popper.js": "^1.14.3",
    "rxjs": "^6.4.0",
    "zone.js": "~0.9.1"
  },
```

Kuva 19. package.json tiedosto (Puurunen, 2020)

5.2.3 Ohjelmistokoodimuutokset version noston jälkeen

Version noston jälkeen kohtasimme useita virheitä konsolissa, kun koitimme käynnistää sovellusta ja näiden korjaaminen oli ehdotonta toimivuuden kannalta. Virheitä tuli huomattavan paljon, joten tässä kappaleessa käsitellään tärkeimmät ja yleisimmät muutokset joita ohjelmistokoodiin tehtiin.

Http service:n muuttuminen HttpClient service:ksi muutti oletusergonomiaa tukemaan automaattisesti palautusarvojen kartoittamisen JSON-muodoksi. Angular-versiossa 2 jouduimme kartoittamaan palautusarvot JSON-muotoon komennolla ”.map(res => res.json())” ja tätä ei enää tarvinnut tehdä erikseen versiossa 8, vaan kartoitus tapahtui automaattisesti. Kuvassa (Kuva 20) näemme entisen kutsumisen palautusarvoille punaisella (-) rivillä ja uuden kutsumisen vihreällä (+) rivillä.



```
        this.deleted = this.deleteditem.id;
-       this.adminservice.deleteBookingResource(this.deleteditem).map(res => res.json())
+       this.adminservice.deleteBookingResource(this.deleteditem)
            .subscribe(res => { this.confirmation = true; this.translate.get('DELETECONFIRM').subscribe((res
    }
```

Kuva 20. .map(…) poisto (Puurunen, 2020)

Koska Angular:in käytössä oleva rxjs kirjasto muuttui versioiden välillä, niin rxjs-compat jouduttiin poistamaan erikseen, jotta pystyimme migratoimaan rxjs kirjaston versioon 6. Rxjs on hyvin keskeinen kirjasto Angular 8-versiossa ohjelmoinnissa, koska se tekee asynkronisen ohjelmistokoodin sekä palautusarvojen tekemisestä huomattavasti helpompaa. Rxjs kirjastossa ei tapahtunut suuria muutoksia, mutta joitain import lausekkeita jouduttiin muokkaamaan hieman uudessa versiossa, koska uudessa versiossa rxjs kirjastosta ei tarvinnut erikseen kutsua import lausekkeissa tiettyjä osia. Tähän liittyvät muutokset näemme alla olevassa kuvissa (Kuva 21). Kuvassa esiintyvä muutos jouduttiin tekemään useampaan import lausekkeeseen, koska Subscription oli hyvin käytetty ominaisuus kyseisessä projektissa.

```
3       import { Meta, Title, DOCUMENT, MetaDefinition } from '@ang
    -   import { Subscription } from 'rxjs/Subscription';
4   +   import { Subscription } from 'rxjs';
5       import { isPlatformServer } from '@angular/common';
```

Kuva 21. Rxjs subscription muutos (Puurunen, 2020)

6 TULOS

Projektin migraatio onnistui hyvinkin vaivattomasti ja varsinkin Angular osuus ei vaatinut paljoa työtä kehittäjänä. Eniten työtä vaati back-end osuus ASP.NET core ohjelmistokehyksen kanssa, koska sen käytännöt olivat muuttuneet huomattavasti.

ASP.NET Core migraatio vei eniten työtunteja sekä vaati myös paljon paneutumista dokumentaation. Migraation aikana jouduttiin useasti rakentamaan eri osia uudelleen ja varsinkin startup.cs tiedoston uudelleen rakentaminen vaati huomattavan paljon aikaa ja siihen jouduttiin tekemään muutoksia koko projektin ajan, koska huomattiin useasti jonkin palan toiminnallisuuden virheen johtuvan startup.cs tiedostosta.

Angular migraatio olin hyvin helppo toteuttaa ja siihen löytyi kattavat Angular:n itsetoimittavat dokumentaatiot. Erilaiset Angular:n käyttämät metodit sekä käytännöt eivät juurikaan muuttuneet, vaan niiden käyttämistä oli yksinkertaistettu ja tämä teki migraatiosta Angular:in osalta hyvinkin mielekästä.

Migraation jälkeen sovellus testattiin toimivaksi ja migraatio todettiin onnistuneeksi. Sovelluksen erilaiset ominaisuudet sekä toiminnallisuudet eivät käyttäjän puolesta muuttuneet lainkaan, mutta tietoturva sekä tehokkuus parani jonkin verran migraation ansiosta. Huomattiin myös eroa azuresta löytyvän palvelun kuormituksessa sovelluksen julkaisun jälkeen. Tärkeänä saavutuksena toimi myös sovelluksen helppolukuisuus sekä mahdollisuus jatkokehitykselle, koska migraation jälkeen ohjelmistokoodin toimintaa oli helpompi tulkita ja siihen voitiin tehdä jatkokehitystä hyödyntäen nyt mahdollisia käytteen otettavia uusia ominaisuuksia, jotka tulivat uusien versioiden mukana.

7 LÄHTEET

Angular Documentation [verkkoainesto]. [Viitattu 2020-04-28.] Saatavissa: https://angular.io/

Angular Update Guide [verkkoainesto]. [Viitattu 2020-05-04.] Saatavissa: https://update.angular.io/#2.0:8.0l3

Migrate from ASP.NET Core 1.x to 2.0 [verkkoainesto]. [Viitattu 2020-05-04.] Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/migration/1x-to-2x/?view=aspnetcore-3.1

Migrate from ASP.NET Core 2.0 to 2.1 [verkkoainesto]. [Viitattu 2020-05-04.] Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/migration/20_21?view=aspnetcore-3.1

Migrate from ASP.NET Core 2.1 to 2.2 [verkkoainesto]. [Viitattu 2020-05-04.] Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/migration/21-to-22?view=aspnetcore-3.1&tabs=visual-studio

Migrate from ASP.NET Core 2.2 to 3.0 [verkkoainesto]. [Viitattu 2020-05-04.] Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/migration/22-to-30?view=aspnetcore-3.1&tabs=visual-studio

Introduction with Updated ASP.NET Core 3.1 [kuva]. [Viitattu 2020-04-28.] Saatavissa: https://medium.com/@valianttechnosoft/introduction-with-updated-asp-net-core-3-1-434b80f0facf

.NET Core overview [verkkoaineisto]. [Viitattu 2020-04-29.] Saatavissa: https://docs.microsoft.com/en-us/dotnet/core/about

What's new in ASP.NET Core 1.1 [verkkoaineisto]. [Viitattu 2020-04-29.] Saatavissa. https://docs.microsoft.com/fi-fi/aspnet/core/release-notes/aspnetcore-1.1?view=aspnetcore-1.1

What's new in ASP.NET Core 3.0 [verkkoaineisto]. [Viitattu 2020-04-29.] Saatavissa. https://docs.microsoft.com/fi-fi/aspnet/core/release-notes/aspnetcore-3.0?view=aspnetcore-3.1