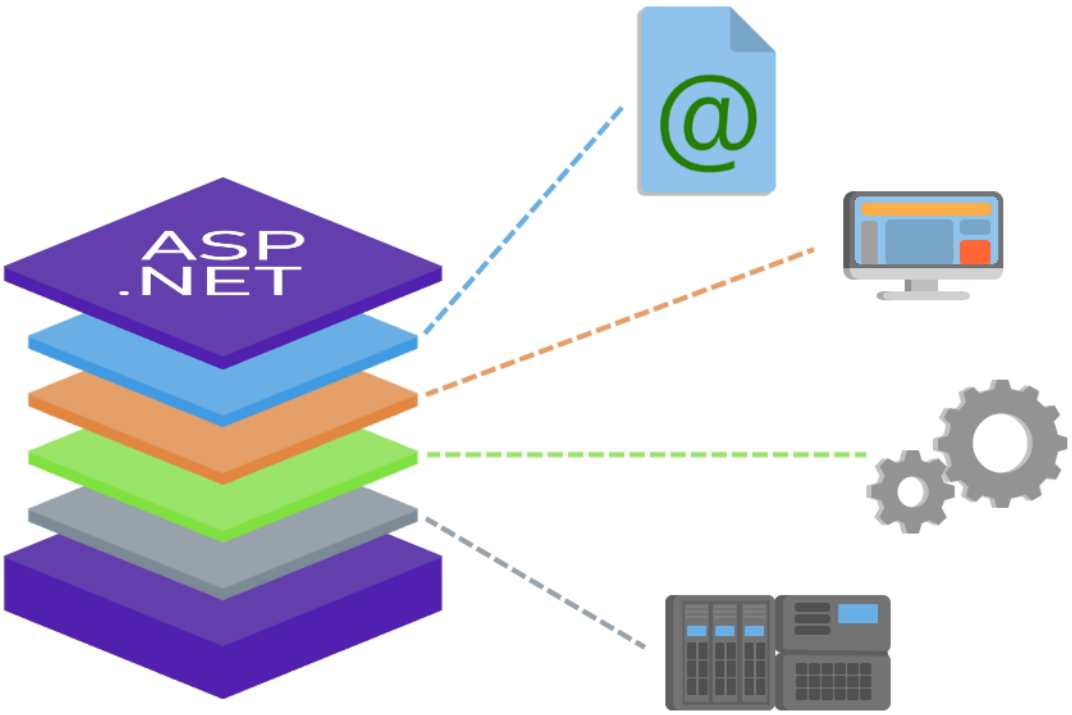
Pekka Ronkainen

ASP.NET Core 2.2 -verkkosovelluksen päivitys versioon 3.0 sekä Razor Pagesin käyttöönotto



Insinööri (AMK)
Tieto- ja viestintätekniikka
Kevät 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Ronkainen Pekka

Työn nimi: ASP.NET Core 2.2 -verkkosovelluksen päivitys versioon 3.0 sekä Razor Pagesin käyttöönotto

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: ASP.NET, ASP.NET Core 3.0, MVC, Razor Pages, Päivitys

Opinnäytetyön toimeksiantaja oli KajaPro Oy. KajaPro Oy tarjoaa ohjelmistotuotannon alihankintapalveluja ja konsultointipalveluja. Yrityksen toimipisteet ovat Kajaanissa ja Oulussa. Tavoitteena työssä oli toimeksiantajan verkkosovelluksen ASP.NET Core -sovelluskehyksen päivittäminen ja Razor Pages -ominaisuuden käyttöönoton testaus. Sovelluskehys haluttiin päivittää, koska uusi versio oli juuri julkaistu ja vanhan version tuki oli päättymässä. Razor Pages -ominaisuuden käyttöönotossa kiinnosti eniten sivujen yksinkertaisuus ja projektin kansiorakenteen yksinkertaistaminen.

Ennen sovelluskehyksen päivittämistä tutkittiin sen hyötyjä ja kannattavuutta ja päivityksen jälkeen tutkittiin muutoksia sovelluksen suorituskyvyssä. Hyötyjä ja kannattavuutta tutkittiin myös Razor Pages -ominaisuuden kanssa. Sovelluksesta muutettiin kaksi sivua Razor Pages -sivuiksi ja toiselle niistä tehtiin myös yksikkötestit.

Sovelluskehyksen päivitys onnistui, ja se paransi sovelluksen suorituskykyä. Razor Pages -ominaisuuden käyttöönotto myös onnistui suurimmalta osalta, mutta sivuja ei voitu täysin muuttaa Razor Pages -sivuiksi. Tämä teki projektin kansiorakenteesta monimutkaisemman, joten päädyttiin siihen lopputulokseen, että Razor Pages -ominaisuuden käyttö ei ole kannattavaa kyseisessä verkkosovelluksessa.

KAMK • University of Applied Sciences

Abstract

Author: Ronkainen Pekka

Title of the Publication: Migrating a Web Application from ASP.NET Core 2.2 to 3.0 And Switching to Razor Pages

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: ASP.NET, ASP.NET Core 3.0, MVC, Razor Pages, Update, Migration

This thesis was commissioned by KajaPro Oy. They provide software production and development contracting services and ICT consulting services. They have offices in Kajaani and Oulu. The purpose of the thesis was to migrate the commissioner's web application from ASP.NET Core 2.2 to 3.0 and to test Razor Pages. One reason for updating the ASP.NET Core framework was because a new version of the framework was just released and the support for the old version was ending. Razor Pages were tested because they are supposed to be simpler than MVC and make the project more organized.

The pros and the worthwhileness of the framework update and Razor Pages implementation were considered before the work began. After updating the framework, the performance changes in unit tests and page loading times were measured. Two pages were changed into Razor Pages and unit tests were also made for one of them.

The framework update was successful, and it improved the performance of the application. The two modified pages were also mostly changed into Razor Pages, but not fully. This just made the project's folder structure more complicated, so it was decided that it's not worthwhile to use Razor Pages in this application.

Sisällys

Lyhenteet ja määritelmät

Controller	Ohjain, back-end, sivujen logiikka MVC:ssä
Model	Malli, sovelluksen data, sovelluksen yhteys tietokantaan
MVC	Model View Controller, ohjelmiston suunnittelumalli
MVVM	Model View ViewModel, ohjelmiston suunnittelumalli
View	Näkymä, käyttöliittymä, front-end, käyttäjän näkemät sivut
ViewModel	PageModel, Sivumalli, sivujen logiikka Razor Pagesissa

1 Johdanto

Opinnäytetyön toimeksiantaja on KajaPro Oy. KajaPro Oy tarjoaa ohjelmistotuotannon alihankintapalveluja ja konsultointipalveluja. Yrityksen toimipisteet ovat Kajaanissa ja Oulussa. Työssä tarkoituksena on päivittää toimeksiantajan verkkosovelluksen ASP.NET Core -sovelluskehys versiosta 2.2 versioon 3.0 sekä tutkia päivityksen kannattavuutta. Lisäksi sovellus päivitetään käyttämään Razor Pages -ominaisuutta, jotta tiedetään, onko sitä kannattavampaa käyttää kuin MVC:tä päivitettävässä sovelluksessa. Alustavasti pari sovelluksen sivua muutetaan Razor Pages -muotoon, mutta enemmänkin sivuja päivitetään, jos se todetaan kannattavaksi ja aikaa riittää.

Työssä tutustutaan Core-versioiden eroihin ja tutkitaan päivityksen kannattavuutta. MVC:n ja Razor Pagesin kanssa tehdään sama. Päivityksen kohteena on toimeksiantajan ASP.NET Core 2.2 -verkkosovellus, joka ei käytä Razor Pages -ominaisuutta. Olin kehittämässä päivitettävää sovellusta koko harjoittelujaksoni ajan, joten se on minulle hyvin tuttu. Päivitettävä sovellus on työntekijöiden työajanseurantaan tarkoitettu ohjelmisto.

Päivitys ASP.NET Core 3.0:aan tehdään, koska se on työn aloituksen hetkellä uusin ASP.NET Coren versio ja päivitys on hyvin ajankohtainen, sillä Core 3.0 julkaistiin 23.9.2019 (1). Lisäksi .NET Core 2.2:n tuki on päättynyt 23.12.2019 (2). ASP.NET Core 3.1 julkaistiin 3.12.2019 eli opinnäytetyön aloituksen jälkeen (3). Siihen päivittäminen 3.0:sta pitäisi olla hyvin yksinkertaista (4).

Razor Pages otetaan käyttöön, koska se on suhteellisen uusi ja suosittu tapa kehittää verkkosovelluksia. Sen on tarkoitus mahdollistaa yksinkertaisempi tapa järjestää koodia (5). Razor Pages -ominaisuuden käyttöönotto ei vaadi Coren version päivittämistä 3.0:aan, sillä Razor Pages lisättiin osaksi Corea jo Core 2.0:ssa (5).

Työn ensimmäinen vaihe on tutkia Core-versioiden eroja sekä miettiä päivityksen kannattavuutta. Tämän jälkeen päivitetään sovelluksen Core-versio, varmistetaan toimivuus sekä tutkitaan lopputulosta. Sen jälkeen tutkitaan MVC:n ja Razor Pagesin eroja ja käyttötarkoituksia sekä kannattavuutta ottaa Razor Pages käyttöön. Sitten päivitetään projekti käyttämään Razor Pages -ominaisuutta ja muutetaan olemassa olevia sivuja Razor Pages -muotoon. Lopuksi tutkitaan tuloksia.

Opinnäytetyö on onnistunut, jos päivitetty sovellus käyttää ASP.NET Core 3.0:aa ja päivitetystä sovelluksesta ei puutu mitään toimintoja, jotka ovat päivittämättömässä versiossa. Päivitettyyn sovellukseen pystytään lisäämään Razor Pages -sivuja ilman mitään muutoksia olemassa oleviin

tiedostoihin. Lisäksi sovelluksen Core-version päivityksen ja Razor Pages -ominaisuuden käyttöönoton kannattavuutta on käsitelty.

2 .NET-kehitysympäristö

.NET on Microsoftin kehittämä ilmainen, monialustainen, avoimen lähdekoodin kehitysympäristö. .NET-sovelluksia voidaan luoda mobiilille, verkkoon, työpöydälle, peleille tai IoT:lle. .NET-sovelluksia voi kirjoittaa C#:lla, F#:lla tai Visual Basic -kielellä. (6.)

2.1 .NET Core

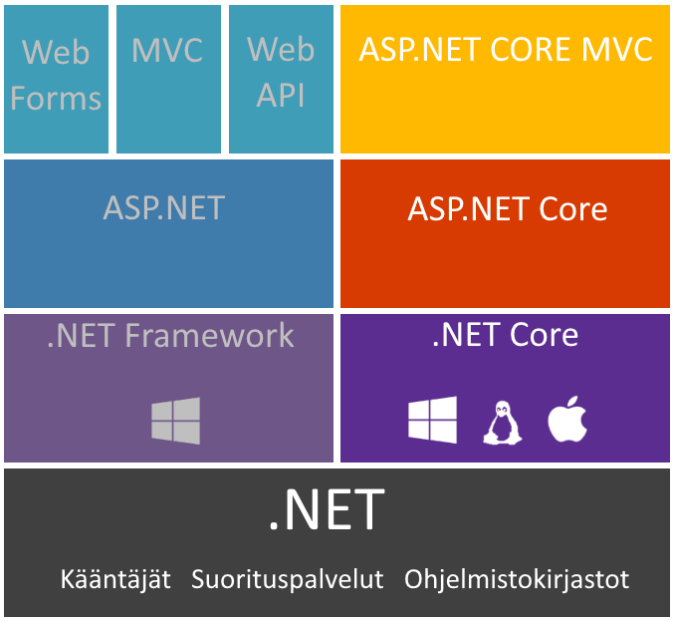
.NET Core on monialustainen .NET-toteutus, ja se on tarkoitettu verkkosivujen, palvelimien ja konsolisovellusten kehitykseen Windowsille, Linuxille ja macOS:lle. Se on myös Microsoftin kehittämä ilmainen, avoimen lähdekoodin kehitysympäristö. Vaihtoehtoinen .NET toteutus on .NET Framework, joka on tarkoitettu sovellusten kehitykseen vain Windowsille. (6,7.)

Ensimmäinen .NET Coren versio eli .NET Core 1.0 julkaistiin samaan aikaan Microsoft Visual Studio 2015 Update 3:n kanssa 27.6.2016 (8). Tässä työssä päivitettävän sovelluksen .NET Core versio on 2.2, joka julkaistiin 4.12.2018 (9). .NET Core 3.0, johon sovellus päivitetään, julkaistiin 23.9.2019, ja kirjoitushetken uusin versio on 3.1, joka julkaistiin 3.12.2019 (3,10). Microsoft aikoo lisäksi marraskuussa 2020 julkaista .NET Coren seuraajan .NET 5 (11).

2.2 ASP.NET Core

ASP.NET Core on Microsoftin kehittämä monialustainen sovelluskehys. Se on tarkoitettu modernien .NET-verkkosovellusten kehitykseen (12). Uusin versio ASP.NET Coresta on 3.1, ja se julkaistiin 3.12.2019 (3).

ASP.NET Corella voi luoda esimerkiksi verkkosovelluksia ja -palveluita sekä IoT-sovelluksia (13). Kuvassa 1 on ASP.NET Coren arkkitehtuuri (14).



Kuva 1. ASP.NET Core -arkkitehtuuri

Kuvasta 1 nähdään, että ASP.NET Core -sovellukset on rakennettu .NET Coren päälle, joka on rakennettu .NETin päälle, ja ASP.NET Core on monialustainen, kun taas ASP.NET on vain Windowsille.

2.2.1 ASP.NET Core 3.0

ASP.NET Core 2.2:sta 3.0:aan siirryttäessä isoin lisäys on Blazor. Blazor on uusi sovelluskehys, joka on tarkoitettu interaktiivisen asiakaspuolen käyttöliittymän kehitykseen. Myös gRPC-niminen RPC-sovelluskehys on uutena ASP.NET Core 3.0:ssa. (15.) RPC-sovelluskehysten tarkoituksena on yksinkertaistaa koodin ajaminen verkon yli ja tehdä siitä yhtä helppoa kuin paikallisen koodin ajaminen (16).

Sisäänrakennettu Microsoftin kehittämä JSON-sarjallistaja System.Text.Json korvaa Newtonsoft.Json-sarjallistajan. JSON-sarjallistajalla tarkoitetaan kirjastoa, jonka avulla JSON-objektit muutetaan .NET-objekteiksi ja toisinpäin. Uudella JSON-sarjallistajalla pitäisi olla parempi suorituskyky vanhaan verrattuna. ASP.NET Core SignalR, Microsoftin RPC-kirjasto, käyttää System.Text.Json-pakettia JSON-viestien sarjallistamiseen. Kaksi uutta Razor-direktiiviä lisättiin, @attribute ja @implements. @attribute lisää annetun attribuutin sivun tai näkymän luokkaan. @implements toteuttaa rajapintaluokan sivun tai näkymän luokkaan. Uutena on myös Worker

Service -sovellusmalli, jota voi käyttää aloituskohtana pitkään käynnissä olevien .NET Core -palveluiden tekoon. (15.)

ASP.NET Core 3.0 on nopeampi ja käyttää vähemmän muistia, mutta ASP.NET Core 3.0 pystytään ajamaan vain .NET Core 3.0:lla eikä enää tue .NET Frameworkia. ASP.NET Core 2.2 pystyttiin ajamaan sekä .NET Corella että .NET Frameworkilla. (15.)

2.2.2 ASP.NET Core -verkkosovellus

ASP.NET Core -verkkosovellus on internetiin yhteydessä oleva ohjelmisto, joka käyttää ASP.NET Core -sovelluskehystä. ASP.NET Core -verkkosovellukset voivat käyttää sekä MVC-mallia että Razor Pages -mallia. (13.) ASP.NET Core -verkkosovelluksissa palvelinpuolen logiikka ajetaan C#:lla ja asiakaspuoli on toteutettu HTML5:llä, CSS:llä ja JavaScriptillä (17). Tässä työssä päivitettävä sovellus on ASP.NET Core 2.2:lla luotu ohjelmisto, joka käyttää MVC-mallia.

2.3 Entity Framework Core

Entity Framework Core on avoimen lähdekoodin monialustainen ORM-työkalu, joka mahdollistaa tietokantojen käsittelyn .NET-objekteilla eli malleilla. Malli koostuu olioluokista ja kontekstiobjektista, jolla on yhteys tietokantaan. (18.)

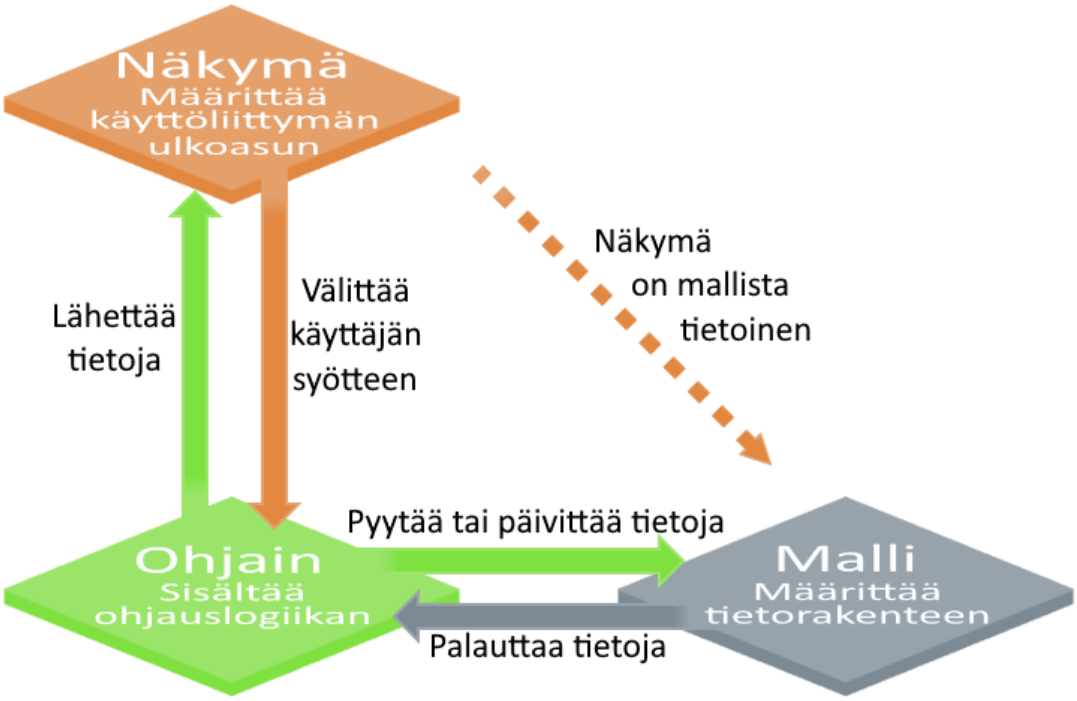
Entity Framework Core 3.0 julkaistiin ASP.NET Core 3.0:n ja .NET Core 3.0:n yhteydessä (10). Isoimmat lisäykset EF Core 3.0:ssa ovat tuki C# 8.0:lle ja LINQ-palvelun uudelleensuunnittelu (19). LINQ-palvelun uudelleensuunnittelusta kerrotaan enemmän versioiden vertailussa.

3 Ohjelmiston suunnittelumallit

Päivitettävä sovellus käyttää MVC-arkkitehtuuria, mutta käyttöönotettava Razor Pages -ominaisuus käyttää MVVM-arkkitehtuuria. Seuraavaksi käydään läpi, mitä MVC ja MVVM tarkoittavat.

3.1 MVC-arkkitehtuuri

MVC on suunnittelumalli, jonka tarkoituksena on erottaa käyttöliittymä, data sekä logiikka toisistaan. MVC on lyhenne sanoista Model, View ja Controller. Model eli malli tarkoittaa sovelluksen dataa. View eli näkymä tarkoittaa sovelluksen käyttöliittymää. Controller eli ohjain tarkoittaa sovelluksen logiikkaa. ASP.NET Coressa käyttäjän pyynnöt ohjataan ohjaimelle, joka hoitaa logiikan mallin tietoja käyttäen ja antaa mallin näkymälle. Näkymä näyttää lopullisen näkymän käyttäen mallin tietoja. (20.) Kuvassa 2 on MVC-mallin toimintaperiaate.

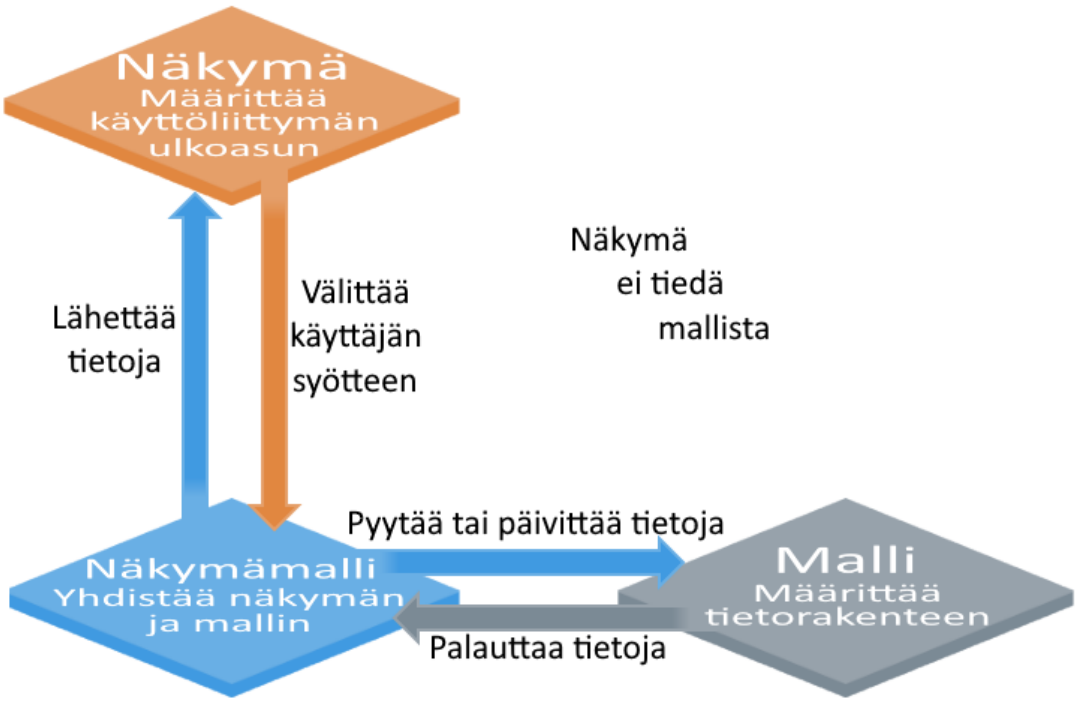


Kuva 2. MVC-arkkitehtuuri

MVC:ssä ohjaimilla voi olla monia toimintoja ja asiat ryhmitellään funktion mukaan. Monimutkainen reititys on helppoa, ja MVC toimii hyvin, jos sovelluksessa on paljon dynaamisia sivuja, REST- ja AJAX-kutsuja. (21,22.)

3.2 Razor Pages – MVVM-arkkitehtuuri

Razor Pages on uudempi vaihtoehto MVC-mallille, ja sen on tarkoitus olla helpompi ja yksinkertaisempi. Razor Pages julkaistiin osana ASP.NET Core 2.0:aa 14.8.2017. Razor Pages -mallissa ohjain on korvattu sivumallilla (engl. PageModel). Sivumalli on osa MVVM-konseptia. MVVM on lyhenne sanoista Model, View ja ViewModel. Sivumalli on tässä kontekstissa ViewModel eli näkymämalli. Razor Pages on tapa luoda sivuja, joka keskittyy näkymien luontiin ja käyttöliittymään. (23.) Kuvassa 3 on MVVM-mallin toimintaperiaate.



Kuva 3. MVVM-arkkitehtuuri

Razor Pages -sivut ovat erinomaisia sovelluksissa, joissa on yksinkertaisia sivuja. Asiat ryhmitellään tarkoituksen mukaan. Jokainen sivu on itsenäinen, ja näkymä ja koodi ovat samassa paikassa. (21,22.)

4 Käytettävät työkalut

Työssä käytettiin Windows 10 -tietokonetta ja koodieditorina käytettiin Visual Studio Codea. ASP.NET Core -päivitystä varten jouduttiin lataamaan .NET Core 3.0 SDK. Core-päivityksen ja Razor Pages -käyttöönoton muutoksia testattiin Google Chromella.

4.1 Visual Studio Code

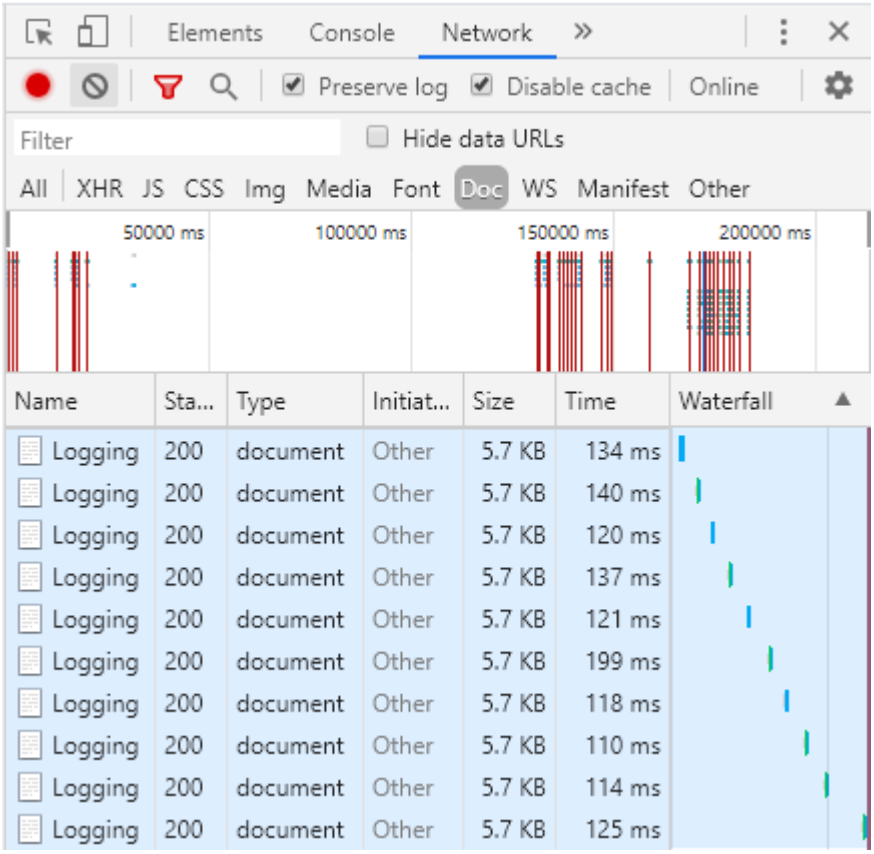
Työssä käytettiin Microsoftin kehittämää Visual Studio Codea, joka on kevyt koodieditori Windowsille, MacOS:lle ja Linuxille. Siinä on sisäänrakennettu tuki JavaScriptille, TypeScriptille ja Node.js:lle, ja laajennuksilla saadaan tuki muille kielille kuten C#, Python ja Go. (24.) VS Code valittiin, koska työssä päivitettävä sovellus kehitettiin enimmäkseen sitä käyttäen. Myös Visual Studio Community 2019 olisi ollut hyvä vaihtoehto, mutta VS Code on kevyempi ja laajennuksilla saatiin kaikki tarvittava toiminnallisuus.

4.2 .NET Core 3.0 SDK

.NET Core SDK:ssa on kaikki tarvittava .NET Core -sovellusten ja kirjastojen luontiin millä tahansa tekstieditorilla ja ajamiseen komentoriviltä. .NET Core SDK on kokoelma kirjastoja ja työkaluja, ja se sisältää esimerkiksi .NET Core komentoliittymän eli CLI:n ja suorituspalvelun (engl. runtime). (25,26.)

4.3 Google Chrome

Sovelluksen verkkosivuihin otettiin yhteys Google Chromella. ASP.NET Core -päivitystä tehdessä Chromella etsittiin rikkoontuneita sivuja ja päivityksen tulosta testattiin Chromen kehittäjän työkaluilla (engl. developer tools). Razor Pages -sivujen toimivuus testattiin myös Chromella. Kuvassa 4 on esimerkki kehittäjän työkalujen käytöstä verkkosivujen latausnopeuden mittaamisessa.



Kuva 4. Google Chromen kehittäjän työkalujen verkkovälilehti

Kuten kuvasta 4 nähdään, Chromen kehittäjän työkaluissa on paljon tietoa verkkosivuista. Työkaluilla voidaan sivujen latausnopeuksien mittaamisen lisäksi tutkia esimerkiksi JavaScript-lähdekoodeja tai muistinkäyttöä.

5 ASP.NET Core -päivitys

Tässä luvussa esitellään ensin päivitettävä sovellus, tutkitaan päivitykseen vaikuttavia eroja ASP.NET Core- ja Entity Framework Core -sovelluskehysten versioiden välillä ja mainitaan päivityksen hyödyt. Sitten tehdään itse päivitys, ja lopuksi tutkitaan päivityksen vaikutuksia yksikkötesteissä ja sivujen latausajoissa.

5.1 Päivitettävä sovellus

Työssä päivitettävä sovellus on toimeksiantajan työajanseurantajärjestelmä, joka ajetaan ASP.NET Core 2.2:n päällä. Sovelluksessa on ohjaimia, jotka sisältävät useita sivuja, mikä tarkoittaa, että yksi ohjain voi jakautua useaksi Razor Pages -sivumalliksi. Monet sovelluksen sivut päivittyvät AJAX-kutsujen avulla, eli sivun tiedot päivittyvät ilman sivun uudelleenlatausta. Tämän takia sivujen ohjaimet ovat monimutkaisia, mikä voi hankaloittaa niiden muuttamista Razor Pages -muotoon.

5.2 Päivitykseen vaikuttavat versioiden erot

ASP.NET Coren ja Entity Framework Coren uusissa versioissa on paljon eri muutoksia, mutta suurin osa niistä ei vaadi mitään muutoksia päivitettävään sovellukseen. Seuraavaksi mainitaan suurimmat asiat, jotka vaativat muutoksia päivitettävän sovelluksen asetuksiin tai koodiin.

5.2.1 ASP.NET Core

Monet ASP.NET Core -ominaisuudet on siirretty NuGet-paketeista osaksi jaettua Microsoft.AspNetCore.App-sovelluskehystä, joten käyttämättömät NuGet-paketit täytyy poistaa projektin asetuksista. Osa ominaisuuksista on myös siirretty jaetusta sovelluskehyksestä NuGet-paketteihin eli projektin asetuksiin joudutaan lisäämään mahdollisesti paketteja. (27.) Päivityksen alussa käydään läpi muutokset projektin NuGet-paketteihin.

5.2.2 Entity Framework Core

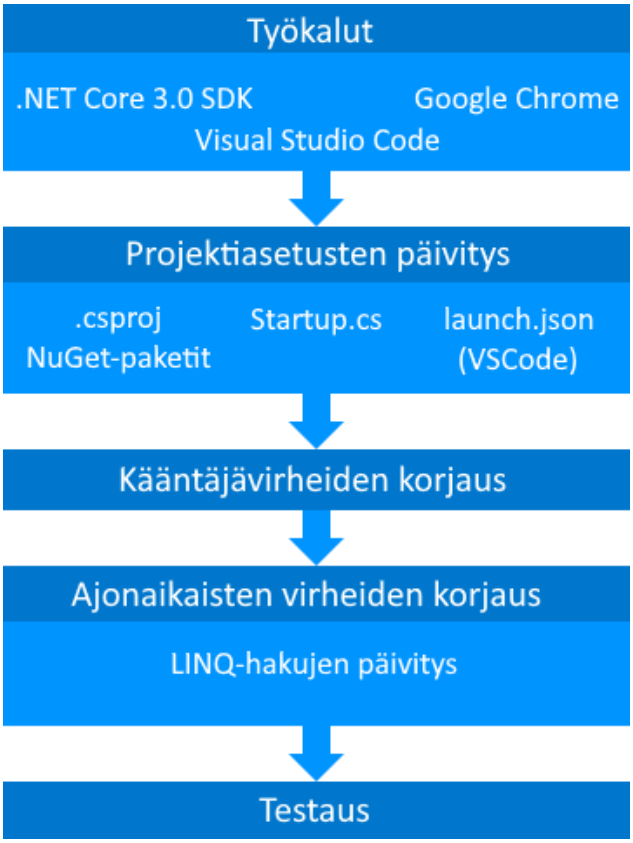
EF Core 3.0:ssa LINQ-palvelu on uudelleensuunniteltu niin, että useammat LINQ-haut voidaan muuttaa SQL-hauiksi ja että hitaat haut huomataan helpommin. LINQ-hakuja, joita ei pystytä kääntämään SQL-hauiksi, ei enää ajeta automaattisesti asiakaspuolella, vaan ne aiheuttavat virheilmoituksen ajon aikana. (19.) Tämän takia joudutaan etsimään ja korjaamaan kaikki LINQ-haut, jotka aiheuttavat virheilmoituksen.

5.3 Päivityksen kannattavuus

Päivitys vaikuttaa kannattavalta, koska ASP.NET Core 3.0:aan päivitys pakottaa kirjoittamaan parempia LINQ-kyselyjä, mikä pystyy huomattavasti parantamaan suorituskykyä. Lisäksi ASP.NET Core 2.2:n tuki on päättynyt, ja päivitys 3.0:sta 3.1:een, jossa tuki jatkuu vielä pitkään, tulee olemaan helppoa (2).

5.4 Päivitys

Visual Studio Code oli asennettuna ja Microsoftin sivuilta oli asennettu .NET Core 3.0 SDK. Kuvassa 5 on projektin päivitysprosessi.



Kuva 5. Projektin päivitysprosessi

Kuten kuvasta 5 nähdään, ensin päivitettiin projektin asetukset. Sitten korjattiin kaikki ilmestyneet kääntäjän löytämät virheet, ja lopuksi korjattiin ajonaikaiset virheet, joista useimmat löydettiin yksikkötestit ajamalla. Päivityksen jälkeen testattiin muutoksia sovelluksen suorituskyvyssä.

5.4.1 Projektin asetusten muuttaminen

Ensiksi sovelluksen projektitiedostosta eli .csproj-tiedostosta kohdesovelluskehys vaihdettiin 3.0-versioon ja NuGet-paketit päivitettiin. Taulukossa 1 on muutokset NuGet-paketteihin.

Ennen		Jälkeen	
NuGet-paketti	Versio	NuGet-paketti	Versio
BuildBundlerMinifier	2.9.406	BuildBundlerMinifier	3.2.435
Microsoft.EntityFrameworkCore	2.2.4	Microsoft.EntityFrameworkCore	3.0.0
Microsoft.EntityFrameworkCore.Tools	2.2.4	Microsoft.EntityFrameworkCore.Tools	3.0.0
Microsoft.VisualStudio.Web.CodeGeneration.Design	2.2.3	Microsoft.VisualStudio.Web.CodeGeneration.Design	3.0.0
MockQueryable.Moq	1.1.0	MockQueryable.Moq	3.0.0
Microsoft.AspNetCore.App	-	Microsoft.AspNetCore.Identity.UI	3.0.0
Microsoft.AspNetCore.Razor.Design	2.2.0	Microsoft.AspNetCore.Mvc.NewtonsoftJson	3.0.0
		Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore	3.0.0
		Microsoft.AspNetCore.Identity.EntityFrameworkCore	3.0.0
		Microsoft.EntityFrameworkCore.InMemory	3.0.0
		Microsoft.EntityFrameworkCore.SqlServer	3.0.0

Taulukko 1. Muutokset NuGet-paketteihin

Taulukosta 1 nähdään, että viisi NuGet-pakettia päivitettiin. Kaksi pakettia poistettiin, koska ne ovat nyt osa jaettua sovelluskehystä, ja kuusi pakettia lisättiin, koska ne eivät ole enää osa jaettua sovelluskehystä. Liitteessä 1 on projektitiedostoon tehdyt muutokset. Yksikkötestausprojektille tehtiin samat muutokset.

Koodissa 1 on Startup.cs-tiedostoon tehtyjä muutoksia. Startup-tiedostossa otetaan käyttöön eri palveluja ja MVC-ominaisuuksia sekä asetetaan eri asetuksia.

```
services.AddMvc().AddJsonOptions(options => {
    options.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
});
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

```
services.AddControllersWithViews()
    .AddNewtonsoftJson(options => {
        options.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
    });
```

Koodi 1. Näkymien ja ohjainten käyttöönotto ja JSON-sarjallistajan asetusten asettaminen

Sovellus käyttää NewtonsoftJson-paketin ominaisuuksia, mutta NewtonsoftJson ei ole enää osa jaettua sovelluskehystä, koska sen korvasi System.Text.Json. Tämän takia se piti erikseen lisätä AddNewtonsoftJson-metodilla. (27.) SetCompatibilityVersion-metodi ei tee ASP.NET Core 3.0:ssa mitään, joten se poistettiin (28). AddMvc-metodin tilalle vaihdettiin AddControllersWithViews,

koska AddMvc ottaa ohjainten ja näkymien lisäksi käyttöön Razor Pages -sivut, mitä ei vielä tässä vaiheessa haluta tehdä (27). Koodissa 2 on loput Startup.cs-tiedostoon tehdyt muutokset.

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

```
app.UseRouting();
app.UseAuthorization();
app.UseEndpoints(endpoints =>
{
    endpoints.MapDefaultControllerRoute();
});
```

Koodi 2. Auktorisoinnin ja ohjainten reitityksen käyttöönotto sekä oletusreitin asettaminen.

UseRouting lisättiin ja UseMvc vaihdettiin UseEndpoints-metodiin, koska UseMvc ei tue päätepistereititystä ASP.NET Core 3.0:ssa. Koska sovellus käyttää auktorisointiominaisuuksia, kuten [Authorize]-attribuuttia, lisättiin UseAuthorization. (27.) MapDefaultControllerRoute tekee saman asian kuin poistetun koodin MapRoute-metodi asetuksineen (29). Lopuksi launch.json-tiedostosta, joka on VS Coden luoma tiedosto, muutettiin sovelluksen polku. Koodissa 3 on launch.json-tiedostossa muutettu rivi.

```
"program": "${workspaceFolder}/example/bin/Debug/netcoreapp2.2/web.dll",
```

```
"program": "${workspaceFolder}/example/bin/Debug/netcoreapp3.0/web.dll",
```

Koodi 3. Sovelluksen polku päivitettiin launch.json-tiedostossa.

Sovelluksen polku pitää päivittää, jotta oikea versio sovelluksesta käynnistyy projektia ajettaessa.

5.4.2 Muutokset ja korjaukset

Kun kaikki projektin asetukset oli muutettu käyttämään ASP.NET Core 3.0:aa, korjattiin kääntäjän ilmoittamat virheet.

IHostingEnvironment korvattiin kaikkialla IWebHostEnvironmentilla. Koodissa 4 on Startup.cs-tiedostossa päivitetty koodirivi.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

Koodi 4. IHostingEnvironment-tyypin korvasi IWebHostEnvironment.

Sekä Microsoft.Extensions.Hosting- että Microsoft.AspNetCore.Hosting-paketissa oli IHostingEnvironment-tyyppi, mikä aiheutti "ambiguous reference" -kääntäjävirheen, kun molemmat nimiavaruudet olivat käytössä, minkä takia se korvattiin (30,31).

IAuthorizationPolicyProvider-rajapintaluokan toteutukset vaativat GetFallbackPolicyAsync-metodin, koska ASP.NET Core 3.0:ssa auktorisointiväliohjelmisto tarvitsee metodin, jota kutsua, kun käytäntöä ei ole määritetty (32). Liitteessä 2 on lisätty metodi.

Muita kääntäjän virheitä ei ollut, joten seuraavaksi korjattiin kääntöajan virheet, jotka kaikki olivat LINQ-hakuja. LINQ-hakuja, joita ei voitu kääntää SQL-hauiksi, ei enää automaattisesti ajettu asiakaspuolella. Tämän takia ne piti muuttaa muotoon, jossa ne voitiin kääntää SQL-hauiksi tai pakottaa ne ajamaan asiakaspuolella. Haut halutaan kääntää SQL:ksi, koska SQL-kutsut ajetaan tietokannan puolella, mikä nopeuttaa niiden ajoa huomattavasti (19). Seuraavat koodinpätkät ovat esimerkkejä siitä, millaisia muutoksia tehtiin LINQ-hakuihin, jotta ne saatiin käännettyä SQL:ksi, tai kun se ei onnistunut, ajettua kutsut asiakaspuolella. Jotta haku saatiin käännettyä SQL:ksi, tarvittiin tietyissä tapauksissa vain poistaa koodia, kun taas toisissa jouduttiin kirjoittamaan huomattava määrä koodia.

GroupBy on yksi komento, joka voidaan vain tietyissä tapauksissa kääntää SQL-käskyksi, joten useimmiten se laitettiin ajamaan asiakaspuolelle (33). Koodissa 5 on esimerkki siitä, kuinka LINQ-haku saadaan ajettua asiakaspuolella.

```
// Return permissions grouped by page
var permissionsPaged = _context.Permissions
    .GroupBy(p => p.Page)
    .ToList();
```

```
var permissionsPaged = _context.Permissions
    .ToList()
    .GroupBy(p => p.Page);
// GroupBy can't be translated into SQL in most cases
```

Koodi 5. ToList-metodin jälkeiset metodit ajetaan asiakaspuolella

Mikä tahansa haun osa saadaan ajettua asiakaspuolella siirtämällä se Tolist- tai AsEnumerable-metodin jälkeen. ToList ja AsEnumerable suorittavat SQL-kutsun ja kaikki niiden jälkeiset metodit ajetaan asiakaspuolella. (34.)

Monia LINQ-metodien sisällä kutsuttuja metodeja, kuten omia funktioita, ei pystytä kääntämään SQL:ksi. Liitteessä 3 on kaksi esimerkkiä LINQ-metodien sisällä olevista metodeista, joita ei voida kääntää SQL:ksi. Liitteen 3 alemmassa esimerkissä GroupBy ja SelectMany poistettiin, koska ne olivat jäänteitä aiemmasta koodista eivätkä ne tehneet mitään tärkeää. Mutta vaikka ne poistettiin, ei kutsua siltikään voitu kääntää SQL:ksi, koska PermissionName.FromString-metodia ei voida kääntää. Koska esimerkkien LINQ-hakuja ei voitu muuttaa niin, että ne pystyttäisiin kokonaan kääntämään SQL:ksi, ne ajettiin asiakaspuolelle lisäämällä AsEnumerable kutsun alkuun.

Jotkin LINQ-haut olivat turhan monimutkaisia tai niihin oli vain jäänyt vanhaa koodia. Monet näistä hauista saatiin kääntymään yksinkertaistamalla koodi. Koodissa 6 on esimerkki LINQ-hausta, joka saatiin kääntymään SQL:ksi.

```
// Get all full pages, and filter all pages by them
var full_pages = _context.Permissions
    .GroupBy(p => p.Page)
    .Where(g => g.Select(p => p.Page).Count() >= 64)
    .Select(s => s.Key)
    .ToList();
```

```
// Get all full pages, and filter all pages by them
var full_pages = _context.Permissions
    .GroupBy(p => p.Page)
    .Where(g => g.Count() >= 64)
    .Select(g => g.Key)
    .ToList();
// g.Select was unnecessary and without it the query can be translated to SQL
```

Koodi 6. Koodin korjaus SQL-kääntyväksi.

Koodissa 6 LINQ-kutsua ei voitu kääntää SQL:ksi Selectin takia. Select pystyttiin poistamaan, koska heti sen jälkeen kutsuttava Count-metodi toimii ilmankin Selectiä. Tämän jälkeen LINQ-kutsu pystyi kääntymään SQL:ksi.

Koodissa 7 on myös esimerkki siitä, kuinka LINQ-haun yksinkertaistaminen mahdollistaa sen kääntymisen SQL:ksi.

```
// Get all permissions on the wanted page
var selected_page = _context.Permissions
    .GroupBy(p => p.Page)
    .Where(g => g.Key == page)
    .FirstOrDefault()
    .Select(pages => pages.Position) // Select only position
    .ToList(); // List of positions that are taken on this page
```

```
var selected_page = _context.Permissions
    .Where(g => g.Page == page)
    .Select(pages => pages.Position) // Select only position
    .ToList(); // List of positions that are taken on this page
// No need to group if we only want a list of positions on a specific page
```

Koodi 7. LINQ-haun yksinkertaistus.

Koodi 7 oli turhan monimutkainen tapa saada lista tietyllä sivulla olevien oikeuksien paikoista. Muuttamalla LINQ-kutsu alempaan muotoon kutsu pystyy kääntymään SQL-kutsuksi ja samalla myös koodin luettavuus parani.

Liitteen 4 koodissa on LINQ-metodin sisällä oleva metodi, joka esti SQL:ksi kääntymisen. Se onnistuttiin korvaamaan koodilla, joka tekee saman asian. Koska koodissa DateTime.ToString(”yyyy-MM”)-metodia ei pystytä kääntämään SQL:ksi, tehtiin kuukausien ja vuosien vertaus numeroina. Samankaltaisia tapauksia oli useita.

Liitteen 5 koodissa jouduttiin kirjoittamaan vähän enemmän, jotta saatiin SQL:ksi kääntyvä haku samalla toiminnallisuudella. GroupBy ja Selectin sisällä oleva FirstOrDefault estivät haun kääntämisen SQL:ksi. Sama toiminnallisuus saatiin korvaamalla ne Selectillä ja Distinctillä, mikä tässä tapauksessa mahdollisti kääntymisen. Samankaltaisia tapauksia oli useita.

Liitteessä 6 on korjattu LINQ-haku, jossa vertaillaan merkkijonoja ja käytetään LINQ-metodien sisällä omia funktioita. Koodissa 8 on esimerkki merkkijonojen vertailusta, jota käytetään liitteessä 6.

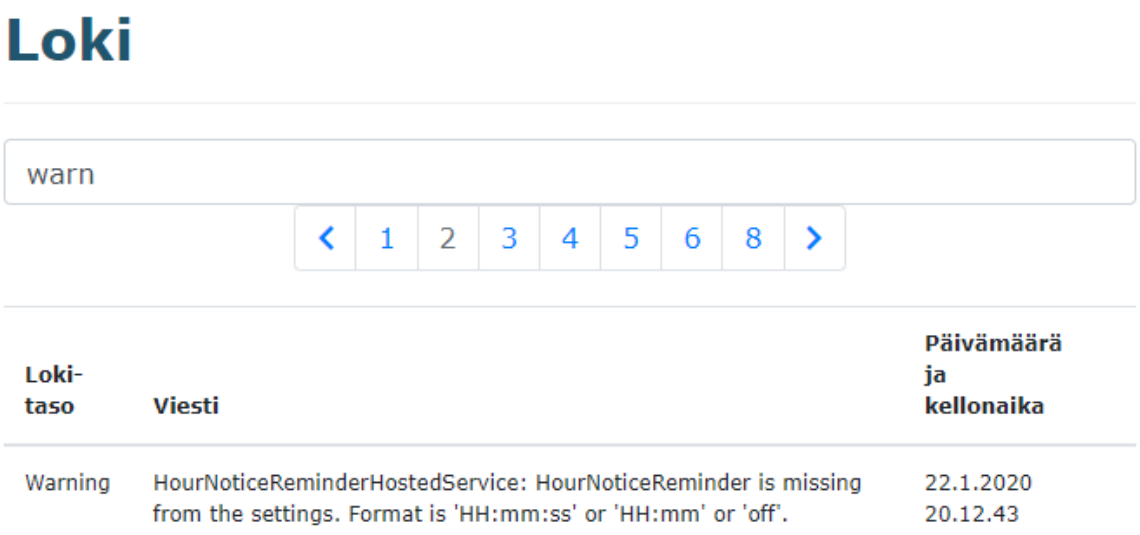
```
customers = customers
    .Where(b => b.Name.Contains(searchString, StringComparison.OrdinalIgnoreCase);
```

```
var searchStringUpper = searchString.ToUpper();
customers = customers
    .Where(b => b.Name.ToUpper().Contains(searchStringUpper));
```

Koodi 8. Merkkijonojen vertailu

Koodissa 8 on käytetty Contains-metodin sisällä asetusta StringComparinson.OrdinalIgnoreCase, joka mahdollistaa merkkijonojen vertailun isoista ja pienistä kirjaimista välittämättä, mutta se ei käänny SQL:ksi. Tämän takia merkkijonot muutetaan isoiksi kirjaimiksi ToUpper-metodilla ja niitä verrataan. Liitteessä 6 on lisäksi käytetty FullName-metodia, joka vain yhdistää etu- ja sukunimen, joten se saatiin helposti kirjoitettua SQL:ksi kääntyvään muotoon. Samankaltaisia tapauksia oli useita.

Liitteessä 7 olevassa koodissa muutetaan monia erityyppisiä arvoja merkkijonoiksi, mikä teki sen muuttamisen SQL-kääntyväksi hyvin hankalaa ilman suuria muutoksia toimintaan. Kuvasta 6 nähdään, miltä lokien hakeminen näytti ennen päivitystä.



Kuva 6. Lokien hakeminen ennen päivitystä.

Vaikein tapaus oli lokimerkintöjen kanssa. Lokimerkintöjä pystyi etsimään päivämäärän, kellonajan, lokitason tai lokiviestin perusteella. Kuvasta 6 nähdään, että tämä tapahtui yhden hakupalkin kautta. Ongelmana oli, että monien eri tyyppien ToString-metodeja ei pystytty kääntämään SQL:ksi. Aluksi yritettiin säilyttää täysin sama toiminnallisuus, mutta sitä varten kirjoitettu koodi oli liian pitkää ja vaikealukuista. Hylätty koodi löytyy liitteestä 8. Lopulta päädyttiin siihen, että on parempi lisätä käyttöliittymään omat hakukentät päivämäärälle, kellonajalle, lokitasolle ja lokiviestille, mikä huomattavasti helpotti SQL:ksi kääntyvän koodin kirjoittamista. Liitteestä 7 löytyy päivitetty koodi ja kuvassa 7 nähdään lisätyt hakukentät.

Kuva 7. Lokien hakeminen päivityksen jälkeen.

5.4.3 Päivityksen testaus ja tulokset

Päivitys oli enimmäkseen aika nopeaa ja helppoa, ja päivittäessä vastaan ei tullut suurempia ongelmia. Suurin ongelma oli edellisessä kappaleessa mainitun lokisivun kanssa, kun yritettiin saada sen LINQ-kutsua SQL-kääntyväksi, mutta senkään ratkaisemisessa ei mennyt pitkään. LINQ-hakuja päivitettiin noin 20, ja monet niistä päivitettiin alle minuutissa.

Päivityksen jälkeen testattiin, kuinka nopeita yksikkötestit ovat, kuinka nopeasti sivu, jota ei ole muokattu, latautuu ja kuinka nopeasti muokattu sivu latautuu. Yksikkötestejä oli 425, ja ne ajettiin molemmilla sovelluksen versioilla 10 kertaa. Sovelluksen sivujen latausnopeus mitattiin molemmilla versioilla 30 kertaa per testi. Yksikkötestit ajettiin VS Coden terminaalin kautta, ja niiden kesto saatiin terminaalista sekunteina neljän desimaalin tarkkuudella. Sivujen latausnopeus mitattiin millisekunnin tarkkuudella Google Chromen kehittäjän työkaluilla.

Yksikkötestien ajo kesti vanhassa versiossa noin 10,9 sekuntia ja päivitetyssä versiossa noin 9,7 sekuntia. Yksikkötestit nopeutuivat 10,8 % eli noin 1,2 sekuntia. Liitteessä 9 on yksikkötestien tarkemmat tulokset.

Ensimmäisen testattavan sivun koodia ja käyttöliittymää ei muutettu päivityksessä ollenkaan ja sen LINQ-kutsut ajettiin jo ennen päivitystä tietokannassa, joten sen latausnopeus on hyvä esimerkki siitä, kuinka paljon pelkästään ASP.NET Coren päivitys voi nopeuttaa sivuja. Sivua testattiin kolmesti, ensiksi paikallisessa tietokannassa 100 ja 10000 tietueella ja lopuksi oikeassa verkossa olevassa tietokannassa. 100 tietueen testissä latausnopeuserot olivat pienet, lataus nopeutui keskimäärin vain 4,7 %, mikä oli 1,4 millisekuntia. 10000 tietueen testissä lataus nopeutui huomattavat 84,7 %, joka oli 152,8 millisekuntia. Tarkemmat tulokset testeille löytyvät liitteestä 10. Lopuksi sivua testattiin verkossa olevassa SQL-tietokannassa, jossa testiin vaikuttavien tietueiden määrää oli vaikea määrittää. Tässä testissä sivun lataus nopeutui keskimäärin 22,9 %, joka oli 58,8 millisekuntia. Liitteessä 11 on tämän testin tarkemmat tulokset.

Toinen testattava sivu oli viime luvussa mainittu lokisivu, jota muokattiin niin, että sen LINQ-kutsut pystytään ajamaan tietokannassa. Sivua testattiin kahdesti verkossa olevassa tietokannassa, jossa testiin vaikuttavia tietueita oli ensin noin 200 ja sitten noin 1000. 200 tietueen testissä sivun lataus nopeutui 5,1 %, joka oli 8,6 millisekuntia. 1000 tietueen testissä lataus nopeutui 16,5 %, joka oli noin 32,0 millisekuntia. Tarkemmat tulokset ovat liitteessä 12.

ASP.NET Coren päivitys versioon 3.0 nopeutti yksikkötestejä ja sivujen latausta jokaisessa testissä. Tulosten perusteella voidaan kuitenkin päätellä, että hyvin pienissä tietokannoissa päivitys ei nopeuta sivujen latausta huomattavasti. Mutta sivujen latausnopeuksien erot ASP.NET Core -versioiden välillä kasvavat huomattavasti, kun tietokannan koko vähänkin kasvaa. Päivityksessä ei myöskään menetetty mitään verkkosovelluksen toimintoja. Tämän takia päädyttiin siihen lopputulokseen, että päivitys oli kannattavaa.

6 Razor Pages

Tämän luvun alussa tutkitaan MVC:n ja Razor Pages -sivujen eroja. Tämän jälkeen kerrotaan, mitä halutaan saavuttaa ottamalla Razor Pages -ominaisuus käyttöön päivitettävässä sovelluksessa. Sitten käydään vaiheittain läpi Razor Pages -ominaisuuden käyttöönotto, MVC-sivun muuttaminen Razor Pages -sivuksi ja yksikkötestin teko Razor Pages -sivulle. Lopuksi pohditaan, onko koko sovellus kannattavaa kääntää käyttämään Razor Pages -sivuja.

6.1 MVC:n ja Razor Pagesin erot

Razor Pages -sivujen näkymä ja koodi löytyvät samasta paikasta. Asiat ryhmitellään tarkoituksen mukaisesti, kun taas MVC:llä ohjaimet ryhmitellään funktioiden mukaan. MVC:ssä mallit, näkymät ja ohjaimet ovat eri paikoissa. Razor Pagesissa jokaisella sivulla on oma sivumalli ja omat toiminnot. Razor Pages toimii erinomaisesti yksinkertaisten sivujen kanssa, jotka eivät tarvitse monia eri REST- ja AJAX-kutsuja. MVC:llä monimutkainen reititys on helpompaa. (21,22.)

6.2 Käyttöönoton kannattavuus

Jos Razor Pages -sivut selventävät projektin kansiorakennetta ilman, että menetetään tärkeitä ominaisuuksia, päivitys kannattaa. Jos kaikki sivut voidaan muuttaa Razor Pages -muotoon, Views- ja Controllers-kansiot voidaan poistaa ja tilalle tulee Pages-kansio. Jos Pages-kansion lisäys ilman Views- ja Controllers-kansioiden poistoa ei tee kansiorakenteesta liian sekavaa, uudet sivut voitaisiin mahdollisesti tehdä Razor Pages -sivuina.

6.3 Razor Pages -ominaisuuden käyttöönotto

Ennen kuin sivuja muutettiin Razor Pages -muotoon, Razor Pages piti ottaa käyttöön projektin asetuksissa ja projektiin piti luoda pari tiedostoa, joita Razor Pages -sivut käyttävät. Kun tämä oli tehty, muutettiin kaksi eri sivua Razor Pages -muotoon. Lopuksi luotiin toiselle päivitetyistä sivuista yksikkötesti.

6.3.1 Projektin asetusten muuttaminen

Sovellus saatiin valmiiksi Razor Pages -sivujen luomiselle lisäämällä muutama rivi Startup.cs-tiedostoon ja lisäämällä pari uutta tiedostoa. Koodissa 9 on Startup.cs-tiedostoon lisätyt rivit.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddRazorPages().AddRazorPagesOptions(options => {
        options.Conventions.AddPageRoute("/Home/Index", "");
    });
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
        endpoints.MapRazorPages();
    });
    ...
}
```

Koodi 9. Razor Pages -ominaisuuden käyttöönotto.

Razor Pages -ominaisuus saatiin päälle lisäämällä services.AddRazorPages() Startup.cs-tiedostoon. AddPageRoute-metodi lisää oletusreitin sivulle /Home/Index, eli se on verkkosivujen etusivu. MapRazorPages-metodi lisää reitityksen Razor Pages -sivuille.

Tämän jälkeen luotiin Pages-kansio, johon kopioitiin _ViewImports.cshtml ja_ViewStart.cshml Views-kansiosta. Razor Pages -sivumallit tulevat käyttämään nimiavaruutta example.Pages, joten @using example.Pages lisättiin _ViewImports.cshtml-tiedostoon, mikä tarkoittaa, että sitä ei tarvitse kirjoittaa erikseen jokaiseen Razor Pages -sivuun. _ViewStart.cshtml-tiedostoon voi kirjoittaa koodia, joka ajetaan ennen jokaisen sivun oman cshtml-tiedoston läpi käyntiä. _ViewStart.cshtml-tiedostossa asetetaan ulkoasu, jota kaikki sivut käyttävät, ja koska Razor Pages -sivut käyttävät samaa ulkoasua kuin MVC-sivut, ei tiedostoa tarvinnut muuttaa.

6.3.2 Sivujen päivitys Razor Pages -muotoon

Sivun päivittämiseksi Razor Pages -muotoon jouduttiin muokkaamaan ja siirtämään sivun cshtml-tiedostoa sekä luomaan sivulle cshtml.cs-tiedosto, johon siirrettiin sivun palvelinpuolen logiikka. Ensiksi muutettava sivu siirrettiin Views-kansion alta Pages-kansioon, ja sitten sivun alkua muokattiin. Koodissa 10 on sivun alkuun tehdyt muokkaukset.

```
@model IEnumerable<example.Models.FrontPageBulletin>
@page "{handler?}"
@model IndexModel
```

Koodi 10. Sivun muuttaminen Razor Pages -sivuksi.

Kuten koodista 10 nähdään, sivun alkuun lisättiin @page ja @model vaihdettiin listasta sivumalliin, joka luotiin seuraavassa vaiheessa. @page tekee sivusta Razor Pages -sivun. IndexModel tulee olemaan tässä tapauksessa sivumalliluokka, jolla on Bulletins-niminen muuttuja, joka on tyyppiä IEnumerable< example.Models.FrontPageBulletin>. Sivulta lisäksi kaikki viittaukset Model-muuttujaan korvattiin Model.Bulletins-muuttujalla.

Kansioon, johon sivu siirrettiin, luotiin samanniminen tiedosto, mutta päätteeksi tuli cshtml.cs. Sivun logiikka siirrettiin ohjaimesta cshtml.cs-tiedostoon. Liitteessä 13 on yksinkertaisen sivun logiikka, joka siirrettiin ohjaimesta sivumalliin. Liitteessä 14 on sama sivun logiikka Razor Page -sivumallissa. Sivumallin koodi muistuttaa hyvin paljon ohjaimen koodia. Sivumallin metodien (engl. handler methods) nimet pitää olla muodossa OnGetMetodinNimiAsync. Getin tilalla voi olla Post, Put, jne. Metodin nimi ja Async ovat vapaaehtoisia. Sen sijaan, että palautusarvona on View ja sen parametrinä malli, Razor Page -sivumalli on itsessään palautettava malli, jonka muuttujiin sivu pääsee käsiksi @modelin kautta. Koodissa 11 on toinen esimerkki siitä, miten sivun logiikka voidaan siirtää ohjaimesta sivumalliin.

```
[Authorize]
public class CommonController : Controller
{
    ...
    [Permission(PermissionName.User)]
    public async Task<IActionResult> Info()
    {
        // Get first employee
        var employee = _context.Employees.FirstOrDefault();

        ViewBag.Culture = CultureInfo.CurrentCulture.TwoLetterISOLanguageName;
        return View(employee);
    }
    ...
```

```
[Authorize]
[Permission(PermissionName.User)]
public class InfoModel : PageModel
{
    ...
    public Employee Employee { get; set; }
    [ViewData]
    public String Culture { get; set; }

    public async Task<IActionResult> OnGetAsync()
    {
        // Get first employee
        var employee = _context.Employees.FirstOrDefault();

        Employee = employee;
        Culture = CultureInfo.CurrentCulture.TwoLetterISOLanguageName;
        return Page();
    }
    ...
```

Koodi 11. Palautusarvo ja ViewBagin käyttö ohjaimessa ja Razor Pages -sivumallissa.

Kuten koodeista 11 nähdään, [Permission], joka on auktorisointiattribuutti, on siirretty metodista luokkaan. Auktorisointiattribuutteja, kuten [Authorize] tai tässä tapauksessa [Permission], ei voida laittaa sivumallin metodeihin eli yksittäisiin Get/Post-metodeihin, joten joko [Permission] pitää siirtää sivumalliluokkaan tai metodit, joilla on [Permission], pitää jättää ohjaimeen. Jos kaikilla metodeilla ei ole sama Permission, on helpointa jättää ne ohjaimeen. Mutta jos osa metodeista on Razor Page -sivumallilla ja osa ohjaimella, se voi tehdä koodista ja kansiorakenteesta sekavamman, kuin jos kaikki sivun metodit ovat yhdessä paikassa. ViewBagin käyttö on korvattu

sivumallin muuttujalla, jolla on [ViewData]-attribuutti. Vaihtoehtoisesti ViewDataan voidaan myös suoraan asettaa arvoja ilman erillistä muuttujaa sivumallissa.

Metodeja siirrettäessä ohjaimesta sivumalliin tiettyjen metodien kanssa tuli ongelmia, sillä page-nimistä parametriä ei voitu käyttää, koska se on Razor Pagesin varaama termi. Muita varattuja termejä ovat esimerkiksi action, area, section, model. (35.)

6.3.3 Yksikkötestin teko Razor Pagesille

Jokaisella sivumallilla on oma testiluokka. Testiluokan konstruktorissa testitietokannan alustamisessa ei ole muutoksia, mutta sivumallin luominen on hieman monimutkaisempaa kuin ohjainten kanssa. Liitteissä 15 ja 16 nähdään ohjaimen ja sivumallin alustuksen erot. Sivumallin alustus vaatii enemmän koodia, koska ViewDataa ei voida suoraan asettaa. Sivumallin ViewDatan asettaminen vaatii PageContextin, joka vaatii ActionContextin, joka vaatii ModelStateDictionaryn. Myös HttpContext ja ModelMetaDataProvider-palvelu joudutaan luomaan, mikä myös on yksinkertaisempaa ohjaimen testiluokassa. Koodista 12 nähdään, että itse testit toimivat täysin samoin kuin ohjainten kanssa.

```
[Theory]
[InlineData(null)]
public async Task EmployeeEdit_NotFound(int? id)
{
    // Act
    var action = await _controller
        .EmployeeEditPartial(id) as NotFoundResult;
    // Assert
    Assert.NotNull(action);
}
```

```
[Theory]
[InlineData(null)]
public async Task OnGetEmployeeEditAsync_NotFound(int? id)
{
    // Act
    var action = await _pageModel
        .OnGetEmployeeEditPartialAsync(id) as NotFoundResult;
    // Assert
    Assert.NotNull(action);
}
```

Koodi 12. Esimerkki yksikkötestistä ohjaimelle ja sivumallille.

Koodissa 12 ohjain on vaihdettu sivumalliin ja metodien nimet ovat muutettu Razor Pages -metodien nimeämiskäytänteen takia.

6.3.4 Päivityksen tulos

Yksinkertaiset sivut, joilla oli vain yksi metodi, oli hyvin helppo kääntää Razor Pages -muotoon. Mutta jos sivun eri metodit vaativat eri auktorisointiattribuutteja, ei sivua voitu kokonaan muuttaa Razor Pages -sivuksi, vaan osa metodeista täytyi jättää ohjaimeen. Jos koko sovellus päivitettäisiin käyttämään Razor Pages -sivuja, jouduttaisiin auktorisointiattribuuttien takia jättämään hyvin moni metodi ohjaimiin tai jättää osa sivuista kääntämättä. Tämä tekisi projektin kansiorakenteesta entistä sekavamman, mutta koko Razor Pages -päivityksen tarkoituksena oli tehdä projektista siistimpi. Tämän takia todettiin, että Razor Pages -ominaisuuden käyttöönotto ei ole tässä sovelluksessa kannattavaa.

7 Yhteenveto

Työn tavoitteena oli päivittää toimeksiantajan verkkosovelluksen ASP.NET Core versio 2.2:sta 3.0:aan sekä ottaa Razor Pages käyttöön. Sovellus saatiin päivitettyä 2.2:sta 3.0:aan, mikä nopeutti yksikkötestejä ja sivujen latautumista. Core-päivitysprosessi oli suhteellisen helppo ja jatkokehitysideana voisi päivittää 3.1:een, jonka tuki tulee jatkumaan pitkään. Päivitys 3.0:sta 3.1:een pitäisi olla paljon yksinkertaisempaa kuin 2.2:sta 3.0:aan.

Razor Pages -ominaisuuden käyttöönottokin onnistui, mutta päädyttiin siihen lopputulokseen, että kaikkien sovelluksen sivujen päivittäminen Razor Pages -sivuiksi ei ole kannattavaa sivumallien metodien rajoitusten takia. Vain joidenkin sivujen päivitys tai uusien sivujen teko Razor Pages -sivuiksi ei myöskään kannata, koska se tekisi projektin kansiorakenteesta sekavamman. Lopputuloksena saatiin kaksi uutta versiota toimeksiantajan verkkosovelluksesta. Ensimmäisessä versiossa ASP.NET Core -sovelluskehys on päivitetty 3.0:aan. Toisessa sovelluskehys on päivitetty ja kaksi sovelluksen sivua on muutettu Razor Pages -muotoon.

Lähteet

(1) Roth D. ASP.NET Blog | ASP.NET Core and Blazor updates in .NET Core 3.0. 2019; Saatavilla: https://devblogs.microsoft.com/aspnet/asp-net-core-and-blazor-updates-in-net-core-3-0/. Viitattu 18.11.2019.

(2) .NET Core official support policy. Saatavilla: https://dotnet.microsoft.com/platform/support/policy/dotnet-core. Viitattu 25.11.2019.

(3) Lander R. Announcing .NET Core 3.1. 2019; Saatavilla: https://devblogs.microsoft.com/dotnet/announcing-net-core-3-1/. Viitattu 7.2.2020.

(4) Addie S. Migrate from ASP.NET Core 3.0 to 3.1. Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/migration/30-to-31. Viitattu 7.2.2020.

(5) Smith S. ASP.NET Core - Simpler ASP.NET MVC Apps with Razor Pages. Saatavilla: https://docs.microsoft.com/en-us/archive/msdn-magazine/2017/september/asp-net-core-simpler-asp-net-mvc-apps-with-razor-pages. Viitattu 18.11.2019.

(6) What is .NET? An open-source developer platform. Saatavilla: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet. Viitattu 2.12.2019.

(7) Download .NET (Linux, macOS, and Windows). Saatavilla: https://dotnet.microsoft.com/download. Viitattu 10.2.2020.

(8) Lander R. Announcing .NET Core 1.0. 2016; Saatavilla: https://devblogs.microsoft.com/dotnet/announcing-net-core-1-0/. Viitattu 10.2.2020.

(9) Lander R. Announcing .NET Core 2.2. 2018; Saatavilla: https://devblogs.microsoft.com/dotnet/announcing-net-core-2-2/. Viitattu 10.2.2020.

(10) Lander R. Announcing .NET Core 3.0. 2019; Saatavilla: https://devblogs.microsoft.com/dotnet/announcing-net-core-3-0/. Viitattu 28.11.2019.

(11) Lander R. Introducing .NET 5. 2019; Saatavilla: https://devblogs.microsoft.com/dotnet/introducing-net-5/. Viitattu 10.2.2020.

(12) Choose between ASP.NET 4.x and ASP.NET Core. Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/choose-aspnet-framework. Viitattu 28.11.2019.

(13) Roth D, Anderson R, Luttin S. Introduction to ASP.NET Core. Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/. Viitattu 28.11.2019.

(14) Recio S. What you need to know about ASP.net core. 2018; Saatavilla: https://codeburst.io/what-you-need-to-know-about-asp-net-core-30fec1d33d78. Viitattu 17.2.2020.

(15) What's new in ASP.NET Core 3.0. Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-3.0. Viitattu 7.2.2020.

(16) Arpaci-Dusseau RH, Arpaci-Dusseau AC. Introduction to Distributed Systems. 2014; Saatavilla: http://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf.

(17) ASP.NET | Open-source web framework for .NET. Saatavilla: https://dotnet.microsoft.com/apps/aspnet. Viitattu 28.11.2019.

(18) Overview of Entity Framework Core - EF Core. Saatavilla: https://docs.microsoft.com/en-us/ef/core/. Viitattu 4.12.2019.

(19) New features in Entity Framework Core 3.0 - EF Core. Saatavilla: https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.0/. Viitattu 3.12.2019.

(20) ASP.NET MVC Pattern | .NET. Saatavilla: https://dotnet.microsoft.com/apps/aspnet/mvc. Viitattu 28.11.2019.

(21) Jones M. How Does Razor Pages Differ From MVC In ASP.NET Core? 2019; Saatavilla: https://exceptionnotfound.net/razor-pages-how-does-it-differ-from-mvc-in-asp-net-core/. Viitattu 16.12.2019.

(22) Watson M. ASP.NET Razor Pages vs MVC: Benefits and Code Comparisons. 2017; Saatavilla: https://stackify.com/asp-net-razor-pages-vs-mvc/. Viitattu 11.2.2020.

(23) Fritz J. ASP.NET Blog | Announcing ASP.NET Core 2.0. 2017; Saatavilla: https://devblogs.microsoft.com/aspnet/announcing-asp-net-core-2-0/. Viitattu 28.11.2019.

(24) Documentation for Visual Studio Code. Saatavilla: https://code.visualstudio.com/docs. Viitattu 2.12.2019.

(25) Download .NET Core 3.0 (Linux, macOS, and Windows). Saatavilla: https://dotnet.microsoft.com/download/dotnet-core/3.0. Viitattu 18.11.2019.

(26) .NET Core SDK overview. Saatavilla: https://docs.microsoft.com/en-us/dotnet/core/sdk. Viitattu 10.2.2020.

(27) Addie S, Anderson R. Migrate from ASP.NET Core 2.2 to 3.0. Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/migration/22-to-30. Viitattu 28.11.2019.

(28) Anderson R. Compatibility version for ASP.NET Core MVC. 2019; Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/mvc/compatibility-version. Viitattu 9.1.2020.

(29) MapDefaultControllerRoute Method. Saatavilla: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.controllerendpointroutebuilderextensions.mapdefaultcontrollerroute. Viitattu 7.2.2020.

(30) B AB. IHostingEnvironment's and IApplicationLifetime's marked obsolete and replaced. Saatavilla: https://github.com/dotnet/docs/issues/14926. Viitattu 7.2.2020.

(31) Ross C. [Announcement] IHostingEnvironment's and IApplicationLifetime's marked obsolete and replaced. 2019; Saatavilla: https://github.com/dotnet/aspnetcore/issues/7749.

(32) Conroy B. Breaking change in IAuthorizationPolicyProvider. Saatavilla: https://github.com/dotnet/docs/issues/14834. Viitattu 7.2.2020.

(33) Patel S. Complex Query Operators - EF Core. Saatavilla: https://docs.microsoft.com/en-us/ef/core/querying/complex-query-operators. Viitattu 7.2.2020.

(34) Client vs. Server Evaluation - EF Core. Saatavilla: https://docs.microsoft.com/en-us/ef/core/querying/client-eval. Viitattu 7.2.2020.

(35) Pranav K. MVC Reserved Keywords. Saatavilla: https://github.com/aspnet/AspNetCore/wiki/MVC-Reserved-Keywords.

Liiteluettelo

Liite 1 Projektin kohdesovelluskehyksen ja NuGet-pakettien päivitys projektitiedostossa

```
<TargetFramework>netcoreapp2.2</TargetFramework>
.
.
.
<PackageReference Include="BuildBundlerMinifier" Version="2.9.406" />
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="2.2.4" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.2.4" />
<PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design"
    Version="2.2.3" />
<PackageReference Include="MockQueryable.Moq" Version="1.1.0" />

<PackageReference Include="Microsoft.AspNetCore.App" />
<PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.2.0" />
```

```
<TargetFramework>netcoreapp3.0</TargetFramework>
.
.
.
<PackageReference Include="BuildBundlerMinifier" Version="3.2.435" />
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="3.0.0" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="3.0.0" />
<PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design"
    Version="3.0.0" />
<PackageReference Include="MockQueryable.Moq" Version="3.0.0" />

<PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="3.0.0" />
<PackageReference Include="Microsoft.AspNetCore.Mvc.NewtonsoftJson" Version="3.0.0" />
<PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore"
    Version="3.0.0" />
<PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore"
    Version="3.0.0" />
<PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="3.0.0" />
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.0.0" />
```

Liite 2 Vaaditun metodin lisäys rajapintaluokan toteutukseen

```
internal class PermissionPolicyProvider : IAuthorizationPolicyProvider
{
    .
    .
    .
    public Task<AuthorizationPolicy> GetFallbackPolicyAsync()
    {
        return FallbackProvider.GetFallbackPolicyAsync();
    }
}
```

Liite 3 Kaksi esimerkkiä LINQ-metodien sisällä olevista metodeista

```
// Get all permission groups from settings
var groups_in_settings = _context.Settings
    .Where(s => s.Key == "PermissionGroup")
    .Select(g => g.Value.Split(' ', StringSplitOptions.RemoveEmptyEntries))
    .OrderBy(g => g[0]);
```

```
// Get all permission groups from settings
var groups_in_settings = _context.Settings
    .Where(s => s.Key == "PermissionGroup")
    .AsEnumerable()
    .Select(g => g.Value.Split(' ', StringSplitOptions.RemoveEmptyEntries))
    .OrderBy(g => g[0]);
// g.Value.Split() => String.Split() can't be translated to SQL
```

```
// Permissions that will be displayed on page
var permissions =
    _context.Permissions
    .GroupBy(p => p.Page) // Group by Page
    .SelectMany(p => p.ToList()) // Combine lists
    // Order by permission group order
    .OrderBy(p => PermissionName.User.FromString(p.Name).GetGroupOrder())
    .ThenBy(p => p.Name); // Then order by name
```

```
// Permissions that will be displayed on page
var permissions = _context.Permissions
    .AsEnumerable()
    // Order by permission group order
    .OrderBy(p => PermissionName.User.FromString(p.Name).GetGroupOrder())
    .ThenBy(p => p.Name) // Then order by name
    .ToList();
// GroupBy and SelectMany were unnecessary
// PermissionName.User.FromString() can't be translated to SQL
```

Liite 4 Koodin muokkaus SQL:ksi kääntyvään muotoon

```
// Get employee projects that are not archived and are created before the compilation
var employeeProjects = await _context.EmployeeProjects
    .AsNoTracking()
    .Include(i => i.Project)
    .Where(e =>
    (e.EmployeeId == employee.EmployeeId)
    &&
    (
        (!e.IsArchived && (e.CreatedDate < selectedMonthDate))
        ||
        (e.CreatedDate.ToString("yyyy-MM") == selectedMonthDate.ToString("yyyy-MM"))
        ||
        (e.ArchivedDate.ToString("yyyy-MM") == selectedMonthDate.ToString("yyyy-MM"))
        ||
        ((e.CreatedDate < selectedMonthDate) && (e.ArchivedDate > selectedMonthDate))
    )
    ).ToListAsync();
```

```
// Get employee projects that were not in the archive on the selected month
var employeeProjects = await _context.EmployeeProjects
    .AsNoTracking()
    .Include(i => i.Project)
    .Where(e =>
    (e.EmployeeId == employee.EmployeeId)
    &&
    (
        (!e.IsArchived && (e.CreatedDate < selectedMonthDate))
        ||
        (e.CreatedDate.Year == selectedMonthDate.Year
            && e.CreatedDate.Month == selectedMonthDate.Month)
        ||
        (e.ArchivedDate.Year == selectedMonthDate.Year
            && e.ArchivedDate.Month == selectedMonthDate.Month)
        ||
        ((e.CreatedDate < selectedMonthDate) && (e.ArchivedDate > selectedMonthDate))
    )
    ).ToListAsync();
// ToString() can't be translated to SQL so we compare year and month directly
```

Liite 5 LINQ-haku kääntyväksi

```
project.EmployeeProjects = _context.EmployeeProjects
    .Where(e => e.ProjectId == project.ProjectId)
    .Include(e => e.Employee)
    .GroupBy(ep => ep.EmployeeId)
    .Select(g => g.FirstOrDefault())
    .OrderBy(e => e.Employee.IsArchived)
    .ToList();
```

```
project.EmployeeProjects = _context.EmployeeProjects
    .AsNoTracking()
    .Select(e => new
    {
        ProjectId = e.ProjectId,
        Employee = new
        {
            FirstName = e.Employee.FirstName,
            LastName = e.Employee.LastName,
            IsArchived = e.Employee.IsArchived
        }
    })
    .Where(e => e.ProjectId == project.ProjectId)
    .Distinct()
    .OrderBy(e => e.Employee.IsArchived)
    .Select(e => new EmployeeProject
    {
        ProjectId= e.ProjectId,
        Employee = new Employee
        {
            FirstName = e.Employee.FirstName,
            LastName = e.Employee.LastName,
            IsArchived= e.Employee.IsArchived
        }
    })
    .ToList();
// GroupBy and FirstOrDefault inside Select couldn't be translated.
// They were replaced with Select and Distinct
// AsNoTracking was added because it should be added whenever
// we're not going to modify anything.
```

Liite 6 Merkkijonojen vertailu isoista ja pienistä kirjaimista välittämättä

```
//Show everything that includes the searchString
customers = customers.Where(b =>
    b.Name.Contains(searchString, StringComparison.OrdinalIgnoreCase)
    || b.StreetAddress.Contains(searchString, StringComparison.OrdinalIgnoreCase)
    || b.BusinessId.Contains(searchString, StringComparison.OrdinalIgnoreCase)
    || b.Country.Contains(searchString, StringComparison.OrdinalIgnoreCase)
    || b.PostalCode.Contains(searchString, StringComparison.OrdinalIgnoreCase)
    || b.ContactPersons.Any(c =>
        c.FullName.Contains(searchString, StringComparison.OrdinalIgnoreCase))
    || b.Projects.Any(c => c.Name.Contains(searchString, StringComparison.OrdinalIgnoreCase))
    );
```

```
var searchStringUpper = searchString.ToUpper();
// Show everything that includes the searchString
customers = customers.Where(b =>
    b.Name.ToUpper().Contains(searchStringUpper)
    || b.StreetAddress.ToUpper().Contains(searchStringUpper)
    || b.BusinessId.ToUpper().Contains(searchStringUpper)
    || b.Country.ToUpper().Contains(searchStringUpper)
    || b.PostalCode.ToUpper().Contains(searchStringUpper)
    || b.ContactPersons.Any(c =>
        (c.FirstName + " " + c.LastName).ToUpper().Contains(searchStringUpper)
        || (c.LastName + ", " + c.FirstName).ToUpper().Contains(searchStringUpper))
    || b.Projects.Any(c => c.Name.ToUpper().Contains(searchStringUpper))
    );
// StringComparison.OrdinalIgnoreCase Can't be translated to SQL,
// so instead we compare things in uppercase
```

Liite 7 Vaikea LINQ-haku saada kääntymään SQL:ksi.

```
public async Task<IActionResult> LogIndexPartial(int? pageNumber, string searchString)
{
    var logEntries = _context.LogEntries.AsNoTracking().OrderByDescending(l => l.CreatedDate);
    if (!String.IsNullOrEmpty(searchString))
    {
        logEntries = logEntries.Where(b =>
            b.CreatedDate.ToString().Contains(searchString, StringComparison.OrdinalIgnoreCase)
            || b.LogEntryId.ToString().Contains(searchString, StringComparison.OrdinalIgnoreCase
            || b.LogLevel.ToString().Contains(searchString, StringComparison.OrdinalIgnoreCase)
            || b.Message.Contains(searchString, StringComparison.OrdinalIgnoreCase))
        .OrderByDescending(l => l.CreatedDate);
    }
    var paginatedList = await PaginatedList<LogEntry>
        .CreateAsync(logEntries, pageNumber ?? 1, 20);
    return PartialView("Logging/IndexPartial", paginatedList);
}
```

```
public async Task<IActionResult> LogIndexPartial(int? pageNumber, string searchString,
    List<LogLevel> logLevels, DateTime startTime, DateTime endTime)
{
    var logEntries = _context.LogEntries.AsNoTracking();
    if (!String.IsNullOrEmpty(searchString))
    {
        var searchStringUpper = searchString.ToUpper();
        logEntries = logEntries.Where(b => b.Message.ToUpper().Contains(searchStringUpper));
    }
    if(logLevels.Count > 0)
    {
        logEntries = logEntries.Where(b => logLevels.Contains(b.LogLevel));
    }
    if(startTime != DateTime.MinValue || endTime != DateTime.MaxValue)
    {
        if(endTime == DateTime.MinValue) endTime = DateTime.MaxValue;
        logEntries = logEntries.Where(b => b.CreatedDate >= startTime
            && b.CreatedDate <= endTime);
    }
    logEntries = logEntries
        .OrderByDescending(l => l.CreatedDate);
    var paginatedList = await PaginatedList<LogEntry>
        .CreateAsync(logEntries, pageNumber ?? 1, 20);
    return PartialView("Logging/IndexPartial", paginatedList);
}
```

Liite 8 Hylätty lokisivun koodi

```
var logEntries = _context.LogEntries.AsNoTracking();

if (!String.IsNullOrEmpty(searchString))
{
    var searchStringUpper = searchString.ToUpper();

    Expression<Func<LogEntry, bool>> myFilter = (b =>
        b.Message.ToUpper().Contains(searchStringUpper));

    object resultEnum;
    if(searchString.All(c => "0123456789.:/-".Contains(c)))
    {
        var searchNumbers = new List<int>();
        var firstNumbers = searchString.Split(new char[]{'.',':','/','-'});
        if(firstNumbers.Length == 1 && firstNumbers[0] != "")
        {
            var firstInt = Int32.Parse(firstNumbers[0]);
            searchNumbers.Add(firstInt);
        }
        else if(firstNumbers.Length == 2)
        {
            if(firstNumbers[0] != "")
            {
                var firstInt = Int32.Parse(firstNumbers[0]);
                searchNumbers.Add(firstInt);
            }
            else
            {
                searchNumbers.Add(-1);
            }
            if(firstNumbers[1] != "")
            {
                var secondInt = Int32.Parse(firstNumbers[1]);
                searchNumbers.Add(secondInt);
            }
            else
            {
                searchNumbers.Add(-1);
            }
        }
        else if(firstNumbers.Length == 3)
        {
            if(firstNumbers[0] != "")
            {
                var firstInt = Int32.Parse(firstNumbers[0]);
                searchNumbers.Add(firstInt);
            }
            else
            {
```

```
            searchNumbers.Add(-1);
        }
        if(firstNumbers[1] != "")
        {
            var secondInt = Int32.Parse(firstNumbers[1]);
            searchNumbers.Add(secondInt);
        }
        else
        {
            searchNumbers.Add(-1);
        }
        if(firstNumbers[2] != "")
        {
            var thirdInt = Int32.Parse(firstNumbers[2]);
            searchNumbers.Add(thirdInt);
        }
        else
        {
            searchNumbers.Add(-1);
        }
    }

    var culture = Request.HttpContext.Features.Get<IRequestCultureFeature>()
        .RequestCulture.Culture.TwoLetterISOLanguageName;
    if(searchNumbers.Count == 1 && searchNumbers[0] != -1)
    {
        for(int i = 0; i < 3; i++)
        {
            switch (i)
            {
                case 0:
                    myFilter = myFilter.Or(l => l.CreatedDate.Day == searchNumbers[0]);
                    myFilter = myFilter.Or(l => l.CreatedDate.Hour == searchNumbers[0]);
                    break;
                case 1:
                    myFilter = myFilter.Or(l => l.CreatedDate.Month == searchNumbers[0]);
                    myFilter = myFilter.Or(l => l.CreatedDate.Minute == searchNumbers[0]);
                    break;
                case 2:
                    myFilter = myFilter.Or(l => l.CreatedDate.Year == searchNumbers[0]);
                    myFilter = myFilter.Or(l => l.CreatedDate.Second == searchNumbers[0]);
                    break;
                default:
                    break;
            }
        }
    }
    else if(searchNumbers.Count == 2)
    {
```

```
for(int i = 0; i < 2; i++)
{
    switch (i)
    {
        case 0:
            if(culture == "fi")
            {
                myFilter = myFilter.Or(l =>
                    (searchNumbers[0] != -1 ?
                        (l.CreatedDate.Day == searchNumbers[0]) : true)
                    && (searchNumbers[1] != -1 ?
                        (l.CreatedDate.Month == searchNumbers[1]) : true));
            }
            else if(culture == "en")
            {
                myFilter = myFilter.Or(l =>
                    (searchNumbers[0] != -1 ?
                        (l.CreatedDate.Month == searchNumbers[0]) : true)
                    && (searchNumbers[1] != -1 ?
                        (l.CreatedDate.Day == searchNumbers[1]) : true));
            }
            else
            {
                myFilter = myFilter.Or(l =>
                    (searchNumbers[0] != -1 ?
                        (l.CreatedDate.Year == searchNumbers[0]) : true)
                    && (searchNumbers[1] != -1 ?
                        (l.CreatedDate.Month == searchNumbers[1]) : true));
            }
            myFilter = myFilter.Or(l =>
                (searchNumbers[0] != -1 ?
                    (l.CreatedDate.Hour == searchNumbers[0]) : true)
                && (searchNumbers[1] != -1 ?
                    (l.CreatedDate.Minute == searchNumbers[1]) : true));
        break;
        case 1:
            if(culture == "fi")
            {
                myFilter = myFilter.Or(l =>
                    (searchNumbers[0] != -1 ?
                        (l.CreatedDate.Month == searchNumbers[0]) : true)
                    && (searchNumbers[1] != -1 ?
                        (l.CreatedDate.Year == searchNumbers[1]) : true));
            }
            else if(culture == "en")
            {
                myFilter = myFilter.Or(l =>
                    (searchNumbers[0] != -1 ?
                        (l.CreatedDate.Day == searchNumbers[0]) : true)
```

```
                        && (searchNumbers[1] != -1 ?
                            (l.CreatedDate.Year == searchNumbers[1]) : true));
                }
                else if(culture == "sv")
                {
                    myFilter = myFilter.Or(l =>
                        (searchNumbers[0] != -1 ?
                            (l.CreatedDate.Month == searchNumbers[0]) : true)
                        && (searchNumbers[1] != -1 ?
                            (l.CreatedDate.Day == searchNumbers[1]) : true));
                }
                myFilter = myFilter.Or(l =>
                    (searchNumbers[0] != -1 ?
                        (l.CreatedDate.Minute == searchNumbers[0]) : true)
                    && (searchNumbers[1] != -1 ?
                        (l.CreatedDate.Second == searchNumbers[1]) : true));
            break;
            default:
            break;
        }
    }
}
else if(searchNumbers.Count == 3)
{
    if(culture == "fi")
    {
        myFilter = myFilter.Or(l =>
            (searchNumbers[0] != -1 ?
                (l.CreatedDate.Day == searchNumbers[0]) : true)
            && (searchNumbers[1] != -1 ?
                (l.CreatedDate.Month == searchNumbers[1]) : true)
            && (searchNumbers[2] != -1 ?
                (l.CreatedDate.Year == searchNumbers[2]) : true));
    }
    else if(culture == "en")
    {
        myFilter = myFilter.Or(l =>
            (searchNumbers[0] != -1 ?
                (l.CreatedDate.Month == searchNumbers[0]) : true)
            && (searchNumbers[1] != -1 ?
                (l.CreatedDate.Day == searchNumbers[1]) : true)
            && (searchNumbers[2] != -1 ?
                (l.CreatedDate.Year == searchNumbers[2]) : true));
    }
    else if(culture == "sv")
    {
        myFilter = myFilter.Or(l =>
            (searchNumbers[0] != -1 ?
                (l.CreatedDate.Year == searchNumbers[0]) : true)
```

```
                    && (searchNumbers[1] != -1 ?
                        (l.CreatedDate.Month == searchNumbers[1]) : true)
                    && (searchNumbers[2] != -1 ?
                        (l.CreatedDate.Day == searchNumbers[2]) : true));
            }
            myFilter = myFilter.Or(l =>
                (searchNumbers[0] != -1 ?
                    (l.CreatedDate.Hour == searchNumbers[0]) : true)
                && (searchNumbers[1] != -1 ?
                    (l.CreatedDate.Minute == searchNumbers[1]) : true)
                && (searchNumbers[2] != -1 ?
                    (l.CreatedDate.Second == searchNumbers[2]) : true));
        }
    }
    else if(Enum.TryParse(typeof(LogLevel), searchString, true, out resultEnum))
    {
        var searchEnum = (LogLevel)resultEnum;
        myFilter = myFilter.Or(b => b.LogLevel == searchEnum);
    }

    logEntries = logEntries
        .Where(myFilter);
}
```

Liite 9 Yksikkötestien ajoaika ASP.NET Core 2.2:ssa ja 3.0:ssa. Ajoaika nopeutui noin 11 % eli 1,2 ms.

#	s (2.2)	s (3.0)
1	10,9408	10,1964
2	10,9467	9,8669
3	10,9243	9,534
4	11,1175	9,4167
5	10,7545	9,9207
6	10,7816	9,532
7	10,7787	9,4152
8	10,8116	9,7836
9	10,799	9,5736
10	10,9441	9,7834
Summa	108,7988	97,0225
Keskiarvo	10,8799	9,7023

Testejä	Δs	Δ%
425	-1,1776	-10,8239

Liite 10 Ensimmäisen testisivun latausnopeus paikallisessa tietokannassa. Latausnopeus mitattiin 30 kertaa 100 ja 10000 tietueen tietokannoissa. 100 tietueen testissä sivun latautuminen nopeutui keskimäärin noin 4,7 % eli 1,4 ms, ja 10000 tietueen testissä sivun latautuminen nopeutui noin 85 % eli 152 ms.

#	ms (2.2)	ms (3.0)
1	34	27
2	30	30
3	27	28
4	32	25
5	26	31
6	27	29
7	31	43
8	25	37
9	24	32
10	24	27
11	32	35
12	34	31
13	29	27
14	25	24
15	31	20
16	29	20
17	26	21
18	27	22
19	34	27
20	25	24
21	31	26
22	23	32
23	26	22
24	33	21
25	33	31
26	33	24
27	31	33
28	29	25
29	25	23
30	32	30
Summa	868	827
Keskiarvo	28,9333	27,5667

Tietueita	Δms	Δ%
100	-1,3667	-4,7235

#	ms (2.2)	ms (3.0)
1	183	29
2	202	29
3	179	29
4	202	32
5	175	30
6	172	31
7	172	26
8	181	29
9	173	30
10	175	54
11	177	25
12	179	28
13	175	26
14	210	30
15	171	28
16	174	25
17	179	26
18	175	24
19	178	30
20	182	33
21	210	25
22	171	26
23	181	24
24	168	22
25	170	23
26	181	21
27	172	25
28	171	23
29	191	21
30	179	21
Summa	5408	825
Keskiarvo	180,2667	27,5

Tietueita	Δms	Δ%
10000	-152,7667	-84,7448

Liite 11 Ensimmäisen testisivun latausnopeus oikeassa tietokannassa. Latausnopeus mitattiin 30 kertaa. Tietueiden määrää tietokannassa ei osattu arvioida, mutta sivun lataus nopeutui päivityksen jälkeen noin 23 % eli 58 ms.

#	ms (2.2)	ms (3.0)
1	281	223
2	236	209
3	261	202
4	250	193
5	244	211
6	263	193
7	252	194
8	277	191
9	280	197
10	243	192
11	247	189
12	252	190
13	240	193
14	249	183
15	264	202
16	243	193
17	238	188
18	251	189
19	249	189
20	255	186
21	252	182
22	262	209
23	260	186
24	254	195
25	256	181
26	241	194
27	256	207
28	239	191
29	243	200
30	242	195
Summa	7580	5847
Keskiarvo	252,6667	194,9000

Tietueita	Δms	Δ%
?	-57,7667	-22,8628

Liite 12 Toisen testisivun latausnopeus oikeassa tietokannassa. Latausnopeus mitattiin 30 kertaa noin 200 ja 1000 tietueen tietokannoissa. 200 tietueen testissä sivun latautuminen nopeutui keskimäärin noin 5,1 % eli 8,6 ms, ja 1000 tietueen testissä sivun latautuminen nopeutui noin 16 % eli 32 ms.

#	ms (2.2)	ms (3.0)
1	191	160
2	191	150
3	159	160
4	160	160
5	169	161
6	176	160
7	157	159
8	169	157
9	176	155
10	165	160
11	157	152
12	160	157
13	163	154
14	181	151
15	166	160
16	160	156
17	162	165
18	183	160
19	168	162
20	156	158
21	170	149
22	168	159
23	171	158
24	160	164
25	156	161
26	171	187
27	165	158
28	164	164
29	168	165
30	166	149
Summa	5028	4771
Keskiarvo	167,6000	159,0333

Tietueita	Δms	Δ%
~200	-8,5667	-5,1114

#	ms (2.2)	ms (3.0)
1	190	157
2	226	156
3	182	158
4	198	161
5	184	159
6	193	157
7	208	166
8	181	170
9	195	161
10	180	164
11	181	165
12	189	157
13	203	167
14	173	157
15	222	164
16	193	163
17	224	162
18	180	158
19	228	167
20	192	157
21	191	162
22	210	169
23	199	156
24	162	162
25	202	178
26	190	153
27	168	163
28	187	167
29	199	166
30	189	158
Summa	5819	4860
Keskiarvo	193,9667	162,0000

Tietueita	Δms	Δ%
~1000	-31,9667	-16,4805

Liite 13 Ohjaimesta poistettu sivun logiikka

```
namespace example.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        private readonly DBContextClass _context;

        public HomeController(DBContextClass context)
        {
            _context = context;
        }

        // GET: FrontPageBulletins
        public async Task<IActionResult> Index()
        {
            // Check if we are on production, staging or testing
            var baseURL = HttpContext.Request.Host;
            ViewData["status"] = HttpServiceHelper.urlcheck(baseURL.ToString());

            var bulletins = await _context.FrontPageBulletins
                .Where(b => b.IsArchived == false)
                .OrderByDescending(p =>p.DateTime)
                .Include(f => f.Poster)
                .Include(f => f.Editor)
                .ToListAsync();
            return View(bulletins);
        }
        .
        .
        .
    }
}
```

Liite 14 Päivitetyn sivun logiikka Razor Pages -sivumallissa

```
namespace example.Pages
{
    [Authorize]
    public class IndexModel : PageModel
    {
        private readonly DBContextClass _context;

        public IndexModel(DBContextClass context)
        {
            _context = context;
        }

        public IEnumerable<FrontPageBulletin> Bulletins { get;set; }

        public async Task OnGetAsync()
        {
             // Check if we are on production, staging or testing
            var baseURL = HttpContext.Request.Host;
            ViewData["status"] = HttpServiceHelper.urlcheck(baseURL.ToString());

            Bulletins = await _context.FrontPageBulletins
                .Where(b => !b.IsArchived)
                .OrderByDescending(p => p.DateTime)
                .Include(b => b.Poster)
                .Include(b => b.Editor)
                .ToListAsync();
            return Page();
        }
    }
}
```

Liite 15 Ohjaimen luonti yksikkötestiluokassa

```
namespace example.UnitTests
{
    public class CommonControllerTests
    {
        readonly CommonController _controller;
        readonly DBContextClass _context;

        public CommonControllerTests()
        {
            //Initialize database
            ...
            _controller = new CommonController(_context);
            _controller.TempData = new TempDataDictionary(
                new DefaultHttpContext(), Mock.Of<ITempDataProvider>());
            _controller.ViewData = new ViewDataDictionary(
                new EmptyModelMetadataProvider(), new ModelStateDictionary());

            _controller.TempData["Username"] = "XUnit";
            _controller.TempData["GUID"] = "GUID";
        }
    }
}
```

Liite 16 Sivumallin luonti yksikkötestiluokassa

```
namespace example.UnitTests
{
    public class InfoPageTests
    {
        readonly InfoModel _pageModel;
        readonly DBContextClass _context;
        public InfoPageTests()
        {
            //Initialize database
            ...
            var services = new ServiceCollection();
            services.AddTransient<IModelMetadataProvider,
                EmptyModelMetadataProvider>();

            var httpContext = new DefaultHttpContext()
            {
                RequestServices = services.BuildServiceProvider()
            };
            var modelState = new ModelStateDictionary();
            var actionContext = new ActionContext(httpContext,
                new RouteData(), new PageActionDescriptor(), modelState);
            var viewData = new ViewDataDictionary(
                new EmptyModelMetadataProvider(), modelState);
            var pageContext = new PageContext(actionContext)
            {
                ViewData = viewData
            };

            _pageModel = new InfoModel(_context)
            {
                PageContext = pageContext,
                Url = new UrlHelper(actionContext),
                TempData = new TempDataDictionary(httpContext,
                    Mock.Of<ITempDataProvider>())
            };

            _pageModel.TempData["Username"] = "Xunit";
            _pageModel.TempData["GUID"] = "GUID";
        }
    }
}
```