Tampereen ammattikorkeakoulu

Web-käyttöliittymän kehitys ASP.NET Core Frameworkilla

Tommi Aalto

OPINNÄYTETYÖ
Huhtikuu 2019

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

AALTO, TOMMI:
Web-käyttöliittymän kehitys ASP.NET Core Frameworkilla

Opinnäytetyö 25 sivua
Huhtikuu 2019

Työn tarkoituksena oli kehittää web-sovellus ja käyttöliittymä, joka hyödyntää C#-datakirjastoa. Työssä tutkittiin vaihtoehtoja sovelluksen toteuttamiseen ja perehdyttiin teknologioiden ominaisuuksiin ja niiden tarjoamiin mahdollisuuksiin. Työ päätettiin toteuttaa Microsoftin ASP.NET Core Frameworkilla, joka on kirjasto modernien ja tehokkaiden web-ohjelmistojen kehittämiseen.

Työssä toteutettava sovellus tuli korvaamaan vanhat järjestelmät ja sen tärkeimpänä vaatimuksena oli vanhan järjestelmän toiminnan parantaminen. Työ toteutettiin täysin uutena projektina, sillä vanhasta järjestelmästä haluttiin eroon, eikä sen laajentaminen tai parantaminen ollut mielekästä. Järjestelmässä haluttiin siirtyä uudempiin ja tehokkaampiin teknologioihin.

Työ koostuu kahdesta vaiheesta: laskutusjärjestelmästä ja raportointityökalusta. Molemmat toteutettiin samaan käyttöliittymään. Suurimmat ongelmat liittyivät sivun reaaliaikaiseen päivittämiseen, prosessin seurantaan ja taustaprosessien hallintaan. Myös CSV-tiedoston luonti oli haastavaa ja vaati uusien lisäosien käyttöönottoa ja sen ominaisuuksien opiskelua. Työssä kehitettiin myös yhteyksiä tietokantaan, johon kehitettiin ratkaisut ASP.NET Coren lisäosista.

Työn tuloksena kehitettiin web-palvelu, jossa on molemmat tarvittavat käyttöliittymän osat. Siinä on valmius suorittaa laskutusajo ja sillä voidaan generoida CSV-raportteja. Työssä myös selvitettiin ASP.NET Core Frameworkin peruskäsitteitä sekä yleisimpiä ominaisuuksia ja lisäosia ja testattiin niiden toimintaa käytännössä. Lisäksi tutustuttiin MVC-arkkitehtuuriin ja opittiin siihen liittyvät vaatimukset ja erityisominaisuudet.

Projektiin jätettiin lukuisia jatkokehitysmahdollisuuksia ja sen kehitystä voidaan jatkaa myöhemmin. Ohjelmiston käyttöliittymän voidaan kehittää, kun raportointiin halutaan lisätä hakukohteita. Muut jatkokehitysmahdollisuudet löytyvät taustalla ajettavasta ohjelmistosta sekä datakirjaston hyödyntämisessä.

Asiasanat: ASP.NET Core, C#, web-ohjelmointi, tietokannat

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT-Engineering
Software Engineering

AALTO, TOMMI:
Web user interface development with ASP.NET Core Framework

Bachelor's thesis 25 pages
April 2019

The goal of this thesis was to develop a web application and user interface, which is using C# data library. During development different options for the application was investigated and research was made to find out necessary features to fulfil the needs of the application. The application was developed using ASP.NET Core Framework which is modern and powerful tool for developing web applications.

The application should replace the old systems used for the same purposes and the main goal was to be able to make things better than the old system. The application was developed as totally new project as instead of trying to make the old one better because new technologies were found better and more powerful.

In the application there are two functions: invoicing system and reporting tool. Both were developed to work in the same user interface. Biggest problems and challenges in development were real time rendering, make the process followable and controlling the background processes. Also creating a CSV file was challenging and it required multiple add-ons to be installed. Application also makes connection with database, which were handled also with ASP.NET Core add-ons.

As a result, functional web application was developed that has both required functionalities. It can run invoicing and generate CSV-reports. Also, many functionalities and features of ASP.NET Core Framework were learnt, and those features were tested in action. MVC architecture was explained and its pros and cons were found.

Project was made possible to continue development in the future. User interface can be expanded and developed if reporting tool needs more search criteria. Most of those future development challenges are in the data library and in the background processes.

Key words: ASP.NET Core, C#, web development, databases

SISÄLLYS

ERITYISSANASTO

.NET Framework	Microsoftin ohjelmistokomponenttikirjasto
ASP.NET Core	Microsoftin web-kehityskirjasto
C#	Microsoftin kehittämä C-ohjelmointikieliperheeseen kuuluva ohjelmointikieli, joka kehitettiin .NET-kirjastoa varten.
HTML	Hypertext Markup Language, verkkosivujen kehittämiseen käytetty merkintäkieli
JavaScirpt	Alustariippumaton, web-kehitykseen käytetty, oliopohjainen komentosarjakieli
MVC	Arkkitehtuurimalli, jossa sovellus on jaettu malleihin, näkymiin ja ohjaimiin
Razor Pages	ASP.NET-web-kehityksessä käytetty lisäosa web-sivujen kehitykseen
SQL	Structured Query Language. Kyselykieli, jolla hallitaan tietokantoja.

1 JOHDANTO

Työn tarkoituksena on kehittää web-sovellus ja siihen soveltuva käyttöliittymä, jolla suoritetaan käyttöraporttien luonti ja laskutusajo sekä generoidaan CSV-raportteja. Web-sovelluksen taustalla käytetään C#-kirjastoa, jolloin työn toteutustavaksi ja työkaluksi oli luonnollista valita ASP.NET Core Framework, joka on Microsoftin web-kehitysympäristö. Myös järjestelmään liittyvät muut rajapinnat oli toteutettu Microsoftin ympäristöön, joka tuki osaltaan valintaa.

Työssä tutustutaan ASP.NET Core Frameworkin taustaan, ominaisuuksiin sekä sen moniin lisäosiin. Niitä hyödynnetään projektin eri vaiheissa. Projektin alussa otettiin selvää ASP.NET Coren ja C#-ohjelmointikielen peruskäsitteistä ja ominaisuuksista. Lisäksi kartoitettiin projektin vaatimuksia yhdessä pääkäyttäjien kanssa. Projektin keskeiseksi tavoitteeksi asetettiin laskutusprosessin nopeuttaminen ja sen varmuuden parantaminen. Lisäksi raportointityökalusta tuli tehdä laajempi ja raportteihin piti saada huomattavasti kattavampaa dataa kuin aikaisemmassa versiossa.

Työn pääpaino on käyttöliittymän ominaisuuksien toteutuksessa ja siinä käytettävien erilaisten lisäosien käyttöönotossa. Työssä perehdytään eri ominaisuuksien takana olevaan teknologiaan ja kirjastoihin. Erityistä huomiota kiinnitetään Razor Pages -lisäosaan, joka tuo erilaisen näkökulman web-sovelluksen kehitykseen sekä SignalR-kirjastoon, joka tuo reaaliaikaisen sivun päivittymisen mahdollisuuden sovellukseen. Lisäksi perehdytään MVC-arkkitehtuurimalliin, selvitetään sen taustaa ja hyviä puolia sekä käytetään sen ominaisuuksia osana ASP.NET Core -projektia.

Projektin molemmat keskeiset ominaisuudet luotiin samaan projektiin ja molemmat toimivat samassa web-palvelussa. Molemmille luotiin omanlaisensa käyttöliittymä, mutta molemmat palvelut käyttävät osittain samoja ohjelmiston osia.

Projekti yhdistettiin datakirjastoon, joka tarjoaa lähes kaiken datan, jota web-sovellus käyttää. Tämän kirjaston sisältöön ei tässä työssä tehty muutoksia tai kehitystyötä, vaan kirjasto tarjosi välineen web-sovelluksen toteuttamiseksi. Sen avulla tarvittava data haettiin eri tietokannoista. Web-sovellukseenkin tuli kuitenkin kehittää myös tietokantayhteys.

2 PROJEKTIN TAUSTA

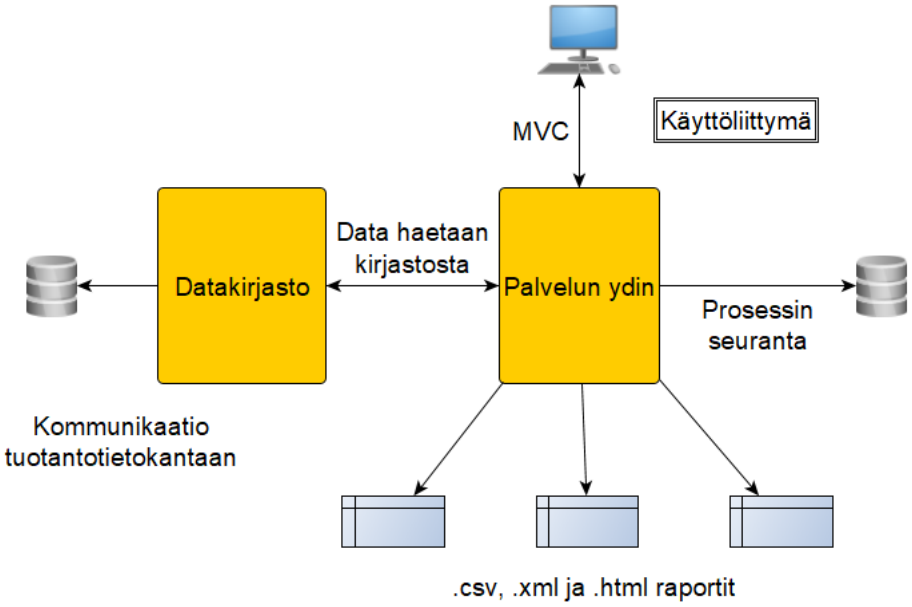
Projektin vaatimusmäärittely toteutettiin yhteistyössä uuden järjestelmän pääkäyttäjän kanssa. Vaatimuksena oli kehittää toimiva käyttöraportteja luova laskutusjärjestelmä sekä erillinen raportointityökalu.

Vanha laskutusjärjestelmä on rakennettu kokonaan Classic ASP -web-ohjelmiston sisään ja siinä on useita ongelmia ja epäkäytännöllisyyksiä. Keskeisin vaatimus oli järjestelmän käyttövarmuuden parantaminen. Vanha järjestelmä on altis virheille ja kaatuu usein ajon aikana. Vanhan järjestelmän kaatuessa on mahdotonta jäljittää, missä kohdassa prosessia jokin oli mennyt vikaan ja se joudutaan aloittamaan alusta. Uuteen järjestelmään tuli rakentaa prosessin seuranta ja mahdollisuus prosessin jatkamiseen ja pysäyttämiseen.

Raportointityökalua varten tuli luoda web-lomake, jolla voidaan määrittää useita eri hakukriteerejä. Kriteerien pohjalta tuli generoida CSV-tiedosto, jota voidaan käyttää datan analysointiin taulukkolaskentaohjelmistolla. Mahdollisia hakukriteerejä tuli olla useita ja haun tuloksena generoidussa CSV-tiedostossa tuli olla huomattavasti enemmän dataa kuin vanhassa järjestelmässä.

Molempien toiminnallisuuksien tuli hyödyntää laskutusdatan hakevaa datakirjastoa. Kirjasto sisältää kaiken sovelluksien tarvitseman datan sekä sitä käsittelevän logiikan ja muun muassa yhteydet tietokantaa. Tuotantokriittinen data on suojattu kirjaston sisään, eikä sovelluksessa ole mahdollisuutta datan muokkaukseen. Sen sisään on rakennettu myös poikkeuskäsittely ja muita hyödyllisiä ominaisuuksia. Järjestelmän arkkitehtuuria kuvataan kuvassa 1.

Aikaisemmat järjestelmät on pääosin rakennettu Windows-ympäristöön, jolloin myös tämän järjestelmän kehitys oli suositeltavaa tehdä Microsoftin työkaluilla. Projekti päätettiin toteuttaa ASP.NET Core Frameworkilla, joka on moderni työkalu web-kehitykseen sisältäen tehokkaita työkaluja C# ohjelmointikielellä ja .NET kirjastoista. Kehityksessä hyödynnettiin useita ASP.NET Coren kirjastoja ja lisäosia.



KUVA 1. Uuden operaattorilaskutusjärjestelmän arkkitehtuuri

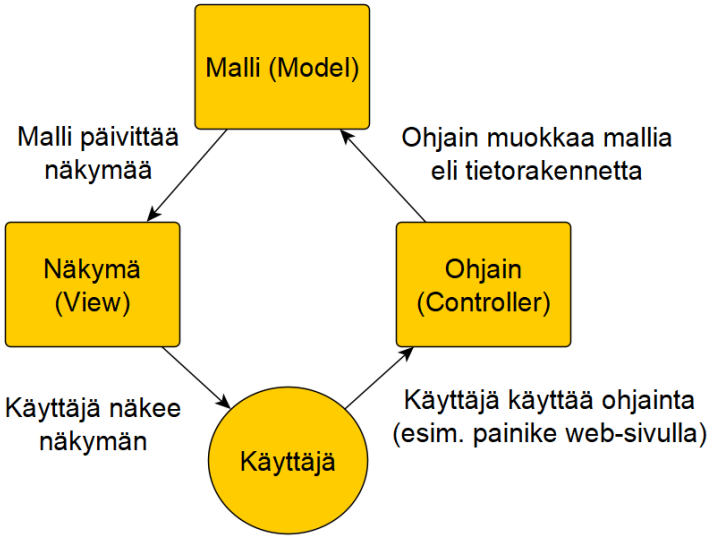
ASP.NET Core tarjoaa helpon mahdollisuuden hyödyntää myös MVC-arkkitehtuuria projektissa. Siinä on sisäänrakennettuna arkkitehtuurimallin vaatimat komponentit ja käyttöä helpottavat ominaisuudet.

2.1 MVC

MVC eli *Model-View-Controller* on ohjelmistoarkkitehtuurin malli, jossa ohjelmiston pääkomponentit jaetaan kolmeen osaan, malliin (*Model*), ohjaimeen (*Controller*) ja näkymään (*View*). Malli sisältää sovelluksen datarakenteen ja siihen liittyvän logiikan sekä mahdollisesti yhteyden tietokantaan. Malli on käytännössä tavallinen luokka.

Näkymässä käyttäjälle esitetään mallin sisältämä data erilaisissa käyttöliittymäkomponenteissa kuten tekstikentissä tai listoissa. Näkymä voidaan toteuttaa monella tavalla kuten web-sivulle tai työpöytäsovellukseen. Tässä työssä näkymät ovat HTML-web-sivuja, jotka on toteutettu monia web-ohjelmoinnin teknologioita, kuten JavaScriptia ja Bootstrapia käyttäen. ASP.NET Core Frameworkilla tehtävään sovellukseen käytetään Razor-kirjastoa, jolla HTML-sivuihin voidaan lisätä C#-koodia.

Ohjainten avulla käyttäjän tekemät valinnat sovelluksessa välitetään mallille, jossa käyttäjän valintojen perusteella suoritetaan loogisia operaatioita ja päivitetään jälleen näkymää. ASP.NET-projektissa ohjaimet periytyvät *Controller*-luokasta, jolloin ohjaimiin saadaan tiettyjä arkkitehtuuria tukevia ominaisuuksia, kuten asynkroniset operaatiot. Ohjaimia ovat esimerkiksi painikkeet ja tekstinsyöttökentät. MVC-mallin toimintaa on esitelty kuvassa 2.



KUVA 2. MVC-arkkitehtuuri

MVC-arkkitehtuuri mahdollistaa ohjelmointityön jakamisen pienempiin osiin. Eri osia voidaan muokata lähes toisistaan riippumatta, jolloin työ voidaan jakaa eri ohjelmoijien kesken. MVC-arkkitehtuurissa selkeytetään myös koodin rakennetta, kun jokaisella koodin osalla on selkeä oma tehtävänsä.

2.2 ASP.NET Core

ASP.NET Core on Microsoftin .NET-kirjaston mukana julkaisema web-ohjelmistokirjasto, jolla rakennetaan uudenaikaisia web-palveluita. Se on suunniteltu tehokkaaksi, helposti testattavaksi ja laajennettavaksi (Microsoft n.d.). Sen ensimmäinen versio julkaistiin osana .NET Frameworkia vuonna 2002, ja sen edeltäjä oli niin sanottu classic *ASP*, joka tulee sanoista *Active Server Pages.* Core-version mukana tuli tuki myös Linux- ja macOS-järjestelmille. .NET Core julkaistiin vuonna 2016.

.NET Framework on Microsoftin avoimen lähdekoodin ohjelmistokomponenttikirjasto monenlaisten sovellusten kehittämiseen. Sen ensimmäinen versio julkaistiin vuonna 2002. Sen tukemia ohjelmointikieliä ovat muun muassa C# ja Visual Basic (Microsoft 2019).

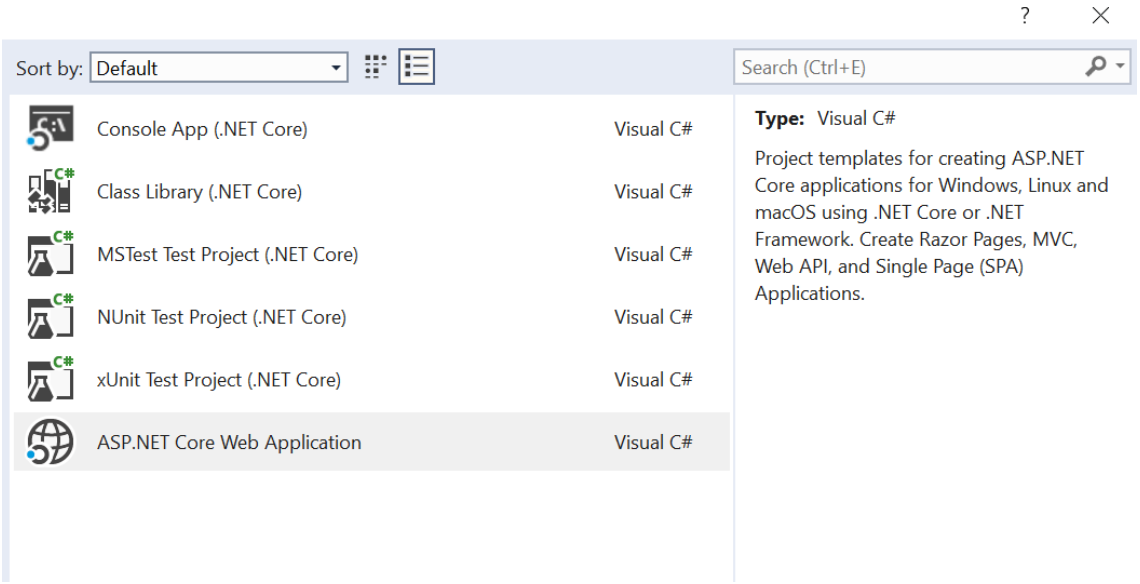
C# (*C sharp*) on helppokäyttöinen, moderni sekä oliopohjaiseen, että funktionaaliseen ohjelmointiin soveltuva .NET-kirjastoa varten kehitetty ohjelmointikieli. Se on helppoa oppia ja sillä pystytään aloittamaan työskentely, mikäli ohjelmoijalla on kokemusta muun muassa C-, C++- tai Java -ohjelmointikielistä. Siihen on rakennettu monia helpottavia ominaisuuksia, jotka yksinkertaistavat muun muassa C++:n ominaisuuksia. Se tarjoaa useita tehokkaita ominaisuuksia kuten lambda-määritykset ja *nullable*-parametrit (Microsoft 2018).

2.3 Työkalut

Edellä mainituille kielille ja kirjastoille luonnollinen kehitystyökalu on Visual Studio 2017. Lisäksi tietokantojen hallintaan käytettiin SQL Server Management Studiota, jolla pystyttiin tarkastelemaan ja etsimään tietokannoissa olevia tauluja sekä tallennettuja proseduureja (engl. *stored prosedure*). Lisäksi voitiin testata tietokantakutsujen toimintaa ja varmistua niiden oikeanlaisesta toiminnasta.

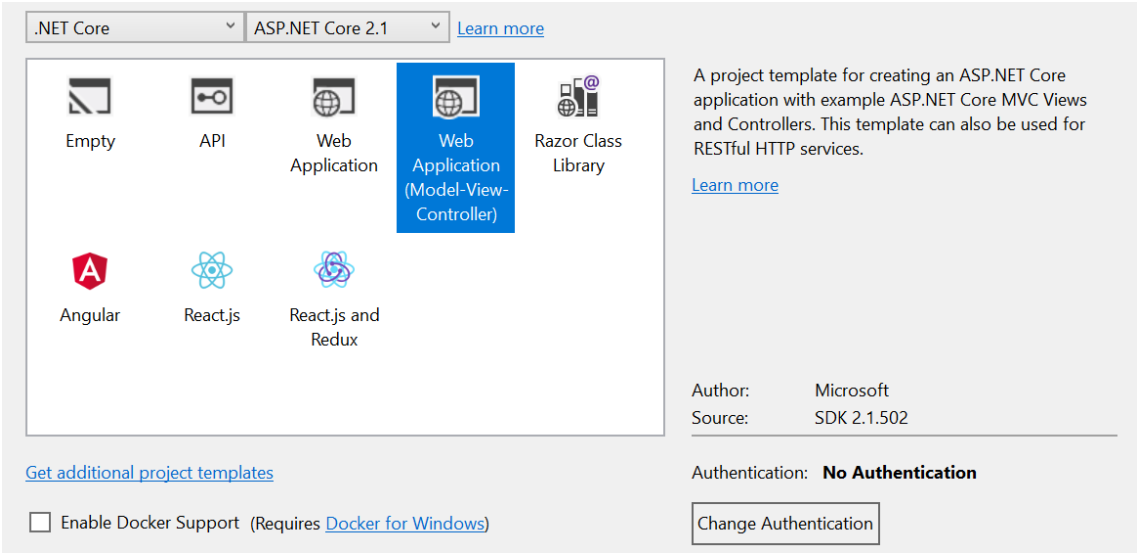
SQL Server Management Studio on myös Microsoftin kehittämä työkalu palvelimien ja tietokantojen hallintaan. Tässä työssä sitä käytettiin tietokantojen testaamiseen ja selvittämään sitä, minkälaisia SQL-komentoja datakirjastossa tuli kutsua oikeanlaisen lopputuloksen aikaansaamiseksi. Projektissa käytetyt tietokannat ovat laajoja ja sisältävät paljon dataa, jolloin niiden tutkimiseen oli käytettävä runsaasti aikaa. Sillä myös testattiin prosessin seurantaan kehitettyä tietokantaa ja sen avulla valittiin oikeanlaiset parametrit ja tietotyypit.

Visual Studio on Microsoftin kehittämä ohjelmointiympäristö eli IDE (*integrated development environment)* monenlaisten ohjelmistojen kehitykseen, virheenetsintään, suorittamiseen ja julkaisuun (Microsoft 2019). Microsoftin ympäristöissä ja tekniikoilla työskennellessä Visual Studio on hyvä valinta. Siihen voidaan asennuksen yhteydessä asentaa useita valmiita pohjia ASP.NET-projekteille (kuva 3).



KUVA 3. ASP.NET Core -projektin luonti

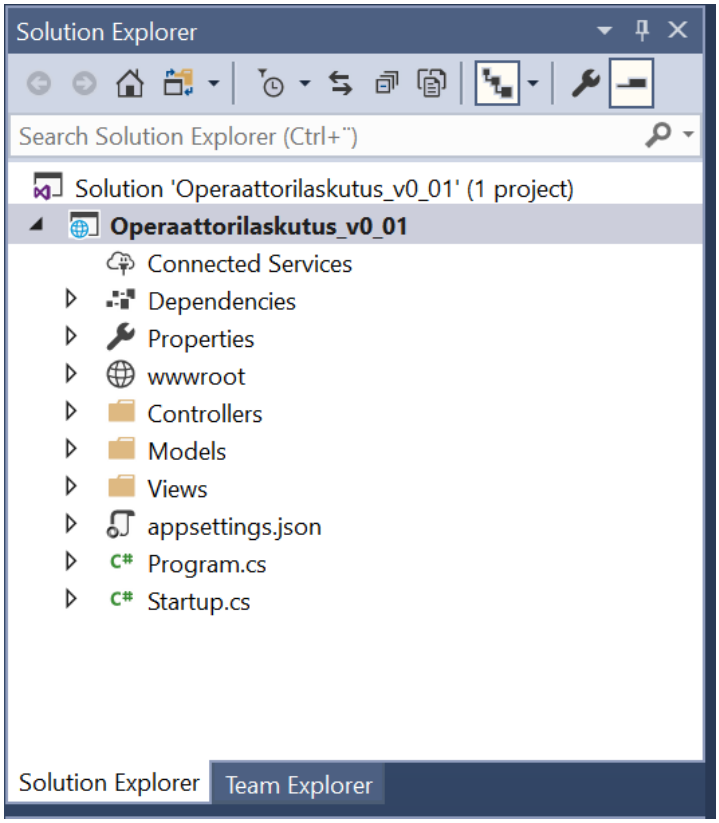
Projekti voidaan luoda täysin tyhjänä tai siihen voidaan jo asennusvaiheessa liittää useita lisäosia (kuva 4), mukaan lukien modernissa web-ohjelmistokehityksessä käytetyt Angular ja React. Tämän työn pohjana käytettiin ASP.Net Core Web Application (Model-View-Controller) -pohjaa, sillä siinä on valmiina useimmat projektissa käytetyt lisäosat, kuten MVC ja Razor Pages.



KUVA 4. ASP.NET Core -projektin luonti, mahdollisia lisäosia

Visual Studio generoi valmiiksi MVC-arkkitehtuurin eli kansiot malleille, näkymille ja ohjaimille. Projektiin generoidaan myös *Program.cs*- ja *Startup.cs* -luokat, joissa sovel-

luksen käyttämät paketit konfiguroidaan. Kaikki käyttöön otetut lisäosat ja -palvelut täytyy ottaa käyttöön *Startup.cs*-luokassa. Web-käyttöliittymän CSS- ja JavaScript-tiedostot ovat *wwwroot*-kansiossa (kuva 5).



KUVA 5. ASP.NET Core -projektin automaattisesti generoitu tiedostorakenne

ASP.NET-projektissa on sisäänrakennettuna useita valmiita lisäosia. Erityisen hyödyllisiä ovat Bootstrap ja jQuery, joiden liitännät projektiin on automaattisesti luotu projektin *Views*-juurikansioon.

3 KÄYTTÖLIITTYMÄN TOTEUTUS

3.1 Razor Pages

ASP.NET Core -sovelluksen näkymä eli käyttöliittymä toteutetaan *Views*-kansioon luoduilla tiedostoilla. Eri mallien näkymät sijoitetaan eri kansioihin ja nimetään mallien mukaisesti. Kutakin mallia kohden luodaan kansio mallin näkymille ja näkymät nimetään halutulla tavalla. Mallin oletusnäkymä on yleensä nimetty *index*-tunnisteella. Näkymät rakennetaan HTML- ja C# -kielillä käyttäen Razor-syntaksia, jonka avulla HTML-sivun joukkoon voidaan lisätä C#-koodia, jolloin sivustolle on helppo tuottaa myös logiikkaa ja tuoda dataa sovelluksen malleista.

Jokaiseen näkymään, johon halutaan liittää jokin malli, määritellään HTML-tiedoston alkuun haluttu malli kuvassa 6 esitetyllä syntaksilla. Tästä eteenpäin näkymässä voidaan viitata malliin *Model*-tunnisteella.

```
1   @model WebApplication8.Models.ReportingTool
```

KUVA 6. Mallin tuonti HTML-sivulle Razor-syntaksilla

Tässä työssä raportointityökaluun luotiin lomake, joka sisältää useita alasvetovalikoita, joilla käyttäjä voi valita hakukriteerit, joiden pohjalta generoidaan CSV-tiedosto. Razor-syntaksissa lomakkeet liitetään ohjaimeen *Tag Helpereilla*, joilla määritettiin, mihin ohjaimeen lomakkeen tiedot lähetetään ja mitä metodeja kutsutaan ohjaimen sisällä. Ne tarjoavat myös muita hyödyllisiä ominaisuuksia HTML-elementeissä olevan datan hallintaan.

Kuvassa 7 on lomakkeen aloitustagi, jonka sisään on luotu *asp-controller* ja *asp-action Tag helperit*. *Asp-controller* määrittää lomakkeen ohjaimeksi *ReportingTool*-ohjaimen ja metodiksi *GenerateReport,* jolloin lomakkeen vahvistamisesta seuraa kyseisessä ohjaimessa kyseisen funktion suorittaminen.

```
<h2>Invoicing Reporting Tool</h2>
<p>Choose what to search:</p>

<form asp-controller="ReportingTool" asp-action="GenerateReport">
```

KUVA 7. *ReportingTool* -näkymään tehtiin lomake, joka kutsuu *ReportingTool*-ohjaimen metodia *GenerateReport*.

ReportinToolController-luokassa on määritelty *GenerateReport*-funktio, joka suoritetaan, kun näkymän lomake lähetetään (kuva 8).

```
public IActionResult GenerateReport(ReportingTool model)
{
```

KUVA 8. *GenerateReport*-funktio, jota kutsutaan lomakkeesta *Reporting Tool* -näkymässä.

Lomakkeen alasvetovalikossa valittavina ovat arvot *ReportingTool*-luokassa olevan *CompanyIds* listan objektit. Razor-syntaksin avulla *foreach*-silmukka oli helppo toteuttaa näkymään ja saada kaikki halutut tiedot näkymään pienellä määrällä koodia. Silmukassa käydään läpi kaikki listassa olevat yritysten tunnisteet ja lisätään ne alasvetovalikon sisään. Lisäksi *Tag Helperillä asp-for* vietiin valittu arvo malliin ja asetettua *CompanyId*-parametriin (kuva 9).

```
Company ID:
<select id="CompanyIdSelection" class="dropdown form-control" asp-for="CompanyId">
    <option value="0">-</option>
    @foreach (int item in Model.CompanyIds)
    {
        <option value="@item">@item</option>
    }
</select>
```

KUVA 9. Razor-syntaksilla luotu for-silmukka, jossa käytetään *ReportingTool*-luokan listaa *CompanyIds*.

Razor ei tarjoa mahdollisuutta näkymän reaaliaikaiseen päivittymiseen ilman sivun uudelleenlatausta. Sovelluksen luonteen takia reaaliaikainen päivittyminen on tärkeä ominaisuus. Ominaisuus voitaisiin toteuttaa esimerkiksi Reactilla tai Angularilla. Tässä projektissa käytettiin ASP.NET Coren lisäosaa SignalR-kirjastoa, jolla sivusto saadaan päivittymään JavaScript-funktioilla.

3.2 SignalR

ASP.NET Core SignalR-kirjasto tarjoaa yksinkertaistetun mahdollisuuden lisätä reaaliaikaista toiminnallisuutta web-sovellukseen. Sitä suositellaan käytettäväksi sovelluksiin, jotka vaativat jatkuvaa päivitystä tietokannasta (Microsoft 2018). Se käyttää *Hub*-luokkaa sovelluksen ja palvelimen väliseen kommunikointiin (kuva 10). Sovellukseen kehitettiin *JobProgressHub* -luokka, joka periytyy *Hub*-luokasta, joka tarjoaa useita metodeja keskusten hallintaan, kuten asiakasohjelman määrityksen.

```
8   namespace WebApplication8.Hubs
9   {
        3 references
10      public class JobProgressHub : Hub
11      {
            0 references
12          public async Task CheckStatus(string tablename)
13          {
```

KUVA 10. *JobProgressHub*-luokka

Hub-luokka määritellään projektin *startup*-luokkaan, jossa keskukselle annetaan myös nimi (kuva 11). Samaa keskuksen nimeä tulee käyttää myös JavaScript-koodissa keskukseen viitattaessa.

```
80              app.UseSignalR(routes =>
81              {
82                  routes.MapHub<JobProgressHub>("/jobprogress");
83              });
```

KUVA 11. Keskuksen määritys *Startup*-luokassa

Kun yhteys (engl. *connection*) on määritelty *startup*-luokkaan, tehdään näkymään JavaScript-funktio, jolla yhteys rakennetaan, kun näkymä latautuu. Kuvassa 12 on esitetty yhteyden määrittely- sekä aloitusfunktiot. Määrityksessä käytetään samaa *Hub*-luokan nimeä kuin *Startup*-luokassa.

Kun näkymä halutaan päivittää, kutsutaan yhteydessä JavaScript-koodissa (kuva 13) *CheckStatus*-funktiota, jolloin *JobProgressHub*-luokassa suoritetaan *CheckStatus*-funktio (kuva 10).

```
77  <script>
78      //Initialize connection
79      var connection = new signalR.HubConnectionBuilder().withUrl("/jobprogress").build();
80
81      //Start connection
82      connection.start().catch(function (err) {
83          return console.error(err.toString());
84      });
```

KUVA 12. Keskuksen määritys *ReportingTool* -näkymän *script*-osiossa JavaScript-funktiolla

```
90          connection.invoke("CheckStatus", tablename).catch(function (err) {
91              return console.error(err.toString());
92          });
```

KUVA 13. Yhteyden ollessa käynnissä voidaan kutsua keskuksen funktioita

Keskuksen *CheckStatus*-funktio suoritetaan ja kaikkien vaiheiden jälkeen keskus kutsuu JavaScript funktiota *ReceiveMessage,* joka puolestaan päivittää näkymää (kuva 14). *Clients.All.SendAsync()*-metodilla haluttu funktio suoritetaan kaikissa asiakasohjelmissa.

```
28              await Clients.All.SendAsync("ReceiveMessage", all, done);
29          }
```

KUVA 14. *JobProgressHub*-luokka päivittää näkymää kutsumalla *ReceiveMessage*-funkitiota

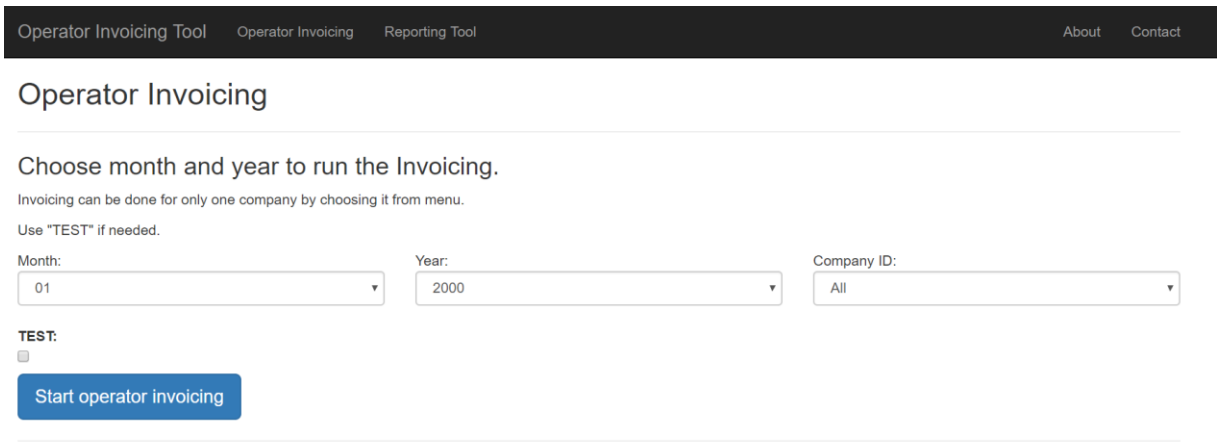
JavaScriptissa määritellyssä yhteydessä suoritetaan haluttu operaatio, esimerkiksi näkymän päivittämisen (kuva 15).

```
95      //JobProcessHub calls this function which updates progress bar
96      connection.on("ReceiveMessage", function (all, done) {
```

KUVA 15. JavaScript-funktio, joka kutsutaan keskuksen *ReceiveMessage*-funktiolla.

3.3 Operaattorilaskutustyökalu

Laskutustyökalussa voidaan suorittaa halutun kuukauden laskutusajo. Se hakee tiedon aktiivisista yrityksistä ja hakee datakirjastoa hyödyntäen yrityksen data. Ajo suoritetaan taustaprosessina ja sitä seurataan web-käyttöliittymästä. Ajo aloitetaan valitsemalla käyttöliittymästä haluttu vuosi, kuukausi ja yritys (kuva 16).

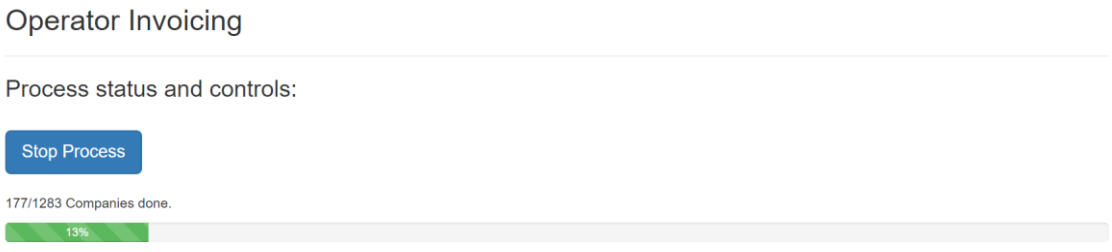


KUVA 16. Laskutusajon aloitusnäkymä

Laskutusajo on pitkä prosessi, minkä takia käyttöliittymään on tehty prosessin seurannan mahdollisuus. Käyttöliittymässä on edistymispalkki (engl. *progress bar*), joka on toteutettu Bootstrap-kirjastolla. Kuvassa 17 on esimerkki edistymispalkin implementoinnista HTML-koodissa. Kuvassa 18 on näkymä laskutusajon aikana, jossa näkyy edistymispalkin visuaalinen näkymä.

```
<div class="progress">
    <div class="progress-bar progress-bar-success progress-bar-striped active"
            role="progressbar" id="bar" aria-valuemin="0" aria-valuemax="100"
            style="width:40%">
           Progress
    </div>
</div>
```

KUVA 17. Bootstrap *Progress Bar* -elementti



KUVA 18. Prosessin eteneminen kuvattuna edistymispalkissa

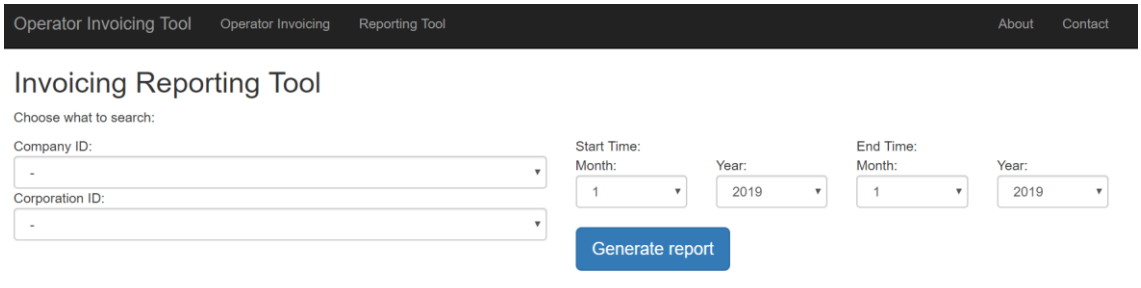
Edistymispalkkia ohjataan kuvan 19 JavaScript-funktiolla. Kun keskus kutsuu *Receive-Message*-funktiota, JavaScript *getElementById*-funktiolla etsitään haluttu komponentti, johon *setAttribute*- ja *innerHtml*-funktioilla asetetaan elementille eri tyyliparametreille arvoja. Muuttamalla tyyliä *width*, voidaan asettaa edistymispalkin vihreä osuus tietyn mittaiseksi.

```
connection.on("ReceiveMessage", function (all, done) {
    document.getElementById("processStatus").innerHTML = " " + done + "/" + all;
    document.getElementById("bar").setAttribute("style", "width:" + done / all *
    100   + "%");
    document.getElementById("bar").innerHTML = "" + done / all * 100 + "%";
});
```

KUVA 19. Bootstrap-edistymispalkin toiminnallisuuden toteutus JavaScriptilla

3.4 Raportointityökalu

Raportointityökalun käyttöliittymä on yksinkertainen web-lomake, joka koostuu useasta alasvetovalikosta sekä Bootstrap-painikkeesta. Osassa alasvetovalikoista on mahdollistettu usean objektin valinta. Se toteutettiin lisäämällä *select*-elementin sisään *multiple*-atribuutti. Lomakkeessa määritellään myös yhteydet haluttuun ohjaimeen sekä ohjaimessa määriteltyyn funktioon Razor-syntaksilla. Raportointityökalun käyttöliittymän ulkoasu vastaa laskutusajon näkymää. (kuva 20).



KUVA 20. Raportointityökalun käyttöliittymä

4 PROJEKTIN MUUT VAIHEET

4.1 ADO.NET

ADO.NET on .NET-kirjasto, jota käytettään sovelluksien ja tietokantojen väliseen kommunikaatioon. Se erottelee tietokantaan yhdistämisen, tietokantakomentojen suorittamisen ja tuloksien esittämisen. Kaksi pääkomponenttia datan käsittelyyn ovat *.NET Framework data providers* ja *ADO.NET DataSet* -objekti (Microsoft 2017).

.NET Framework data providers ovat komponentteja, jotka on suunniteltu datan käsittelyyn. *Connection*-objektilla mahdollistetaan yhteys tietokantaan. *Command*-objekti mahdollistaa komentojen ajamisen tietokantaan tai esimerkiksi tallennettujen proseduurien (engl. *stored procedures*) ajamisen. *DataReader*-objekti tarjoaa suorituskykyisen väylän *DataAdapter*-objektiin, jolla data ohjataan *DataSet*-objektiin. *DataSet* voi sisältää *DataTable*-objekteja, joissa data on jaettu riveihin ja sarakkeisiin ja jotka sisältävät avaimia sekä rajoitteita kuten tietokantojen taulut (Microsoft 2017). Kuvassa 21 on esitetty esimerkki ADO.NET-kirjaston käytöstä.

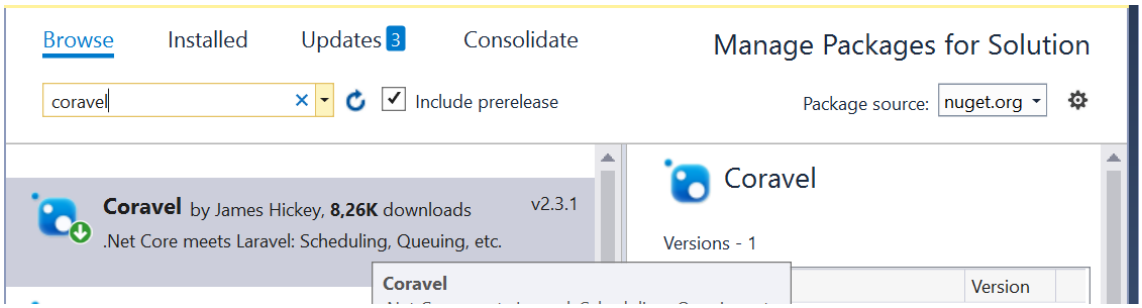
```
string connectionString =
                ”Secret Connection String";


using (SqlConnection connection =
   new SqlConnection(connectionString))
     {
       // Create the Command and Parameter objects.
       // Set sommand to be stored prosedure
       SqlCommand cmd = new SqlCommand("SQL_Command", connection);
       cmd.Parameters.AddWithValue("@Parameter", parameterObject);
       cmd.CommandType = CommandType.StoredProcedure;
       // Create and execute the DataReader, writing the result
       connection.Open();
       SqlDataReader reader = cmd.ExecuteReader();
            if (reader.HasRows)
                {
                while (reader.Read())
                    {
                        list.Add((int)reader.GetValue(0));
                        Company cmpn = new Company();
                        cmpn.CompanyId = (int)reader.GetValue(0);
                        cmpn.CompanyName = reader.GetValue(3).ToString();
                        pairList.Add(cmpn);
                    }
                }
           else
                {
                    Console.WriteLine("No rows found.");
                }
                reader.Close();
```

KUVA 21 - ADO.NET:n käyttö datan hakuun

4.2 Taustaprosessin hallinta

ASP.NET Core tarjoaa useita tapoja taustaprosessien jonouttamiseen ja käynnistämiseen. Yleisin tapa on kirjasto nimeltä *Hangfire* (Pelser 2019). Pelserin ohjeen mukaan työssä on kuitenkin käytetty *Coravel*-kirjastoa. Se on kolmannen osapuolen julkaisema kirjasto, johon löytyy myös hyvä dokumentaatio. Kirjasto asennettiin Visual Studion *NuGet Package Managerilla*, jolla asennetaan myös muita riippuvuuksia (kuva 22).



KUVA 22. Visual Studio 2017 NuGet Package Manager ja *Coravel*-paketin asennus

Taustalla ajettavaa prosessia on käytettävä, sillä muuten pitkän prosessin aikana sivu ei päivittyisi lainkaan, vaan vasta kun prosessi on valmis. Tämänkaltainen toiminnallisuus ei ole toivottavaa käyttäjäkokemuksen kannalta. Prosessin tilaa tulee pystyä seuraamaan sen aikana.

Taustaprosessin jonouttamiseksi haluttuun kontrolleriin injektoidaan *IQueue*-instanssi (kuva 23), jonka jälkeen kutsutaan *QueueAsyncAction*-metodia, joka suorittaa metodissa määritetyt vaiheet taustaprosessina. Prosessi jää taustalle ja ohjaimessa voidaan palauttaa näkymä (kuva 24).

```
private readonly IQueue _queue;
```

KUVA 23. *IQueue*-instanssi

```
52          _queue.QueueAsyncTask(() => PerformOperation(jobId,
53              invoicing.Month, invoicing.Year, invoicing.CompanyId, invoicing.Test));
54
55          return RedirectToAction("Progress", new { jobId });
```

KUVA 24. *QueueAsyncTask*-metodi, joka suorittaa *PerformOperation*-funktion tausta-prosessina.

4.3 Tietokanta prosessin seurantaan

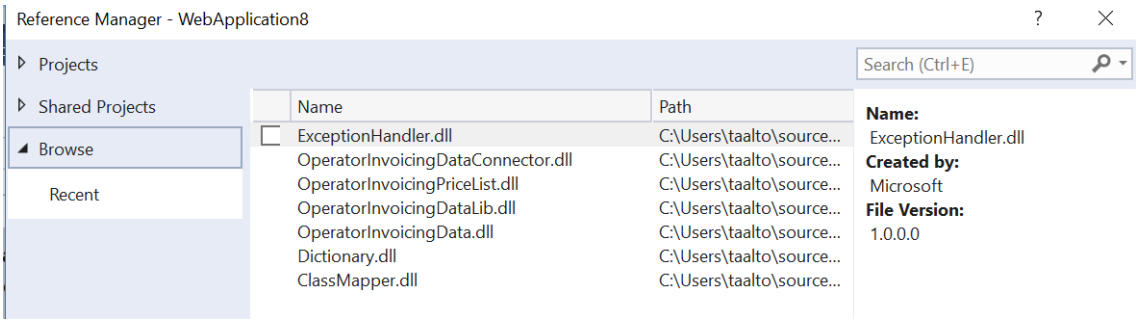
Järjestelmässä on myös sisäinen tietokanta, jonka avulla jokaisesta laskutusajosta järjestelmään jää tieto ajon tuloksesta ja mahdollisista virhetilanteista. Myös ajon reaaliaikainen seuranta on helppo toteuttaa tällä erillisellä tietokannalla käyttäen SignalR-kirjastoa sekä JavaScriptia. Tietokantaan luotiin prosessin vaiheiden tallennukseen soveltuva taulu kuvassa 25 esitetyllä SQL-komennolla.

```
CREATE TABLE [dbo].[progress] (
    [ID] INT NOT NULL IDENTITY(1,1),
    [Company] INT NOT NULL,
    [DataReturned] BIT,
    [CsvOk] BIT,
    [XmlOk] BIT,
    [HtmlOk] BIT,
    [ReportDate] DATETIME,
    [RowWritten]   BIT,
    [RowWrittenDateTime] DATETIME,
    PRIMARY KEY CLUSTERED([ID] ASC));
```

KUVA 25. SQL-komento, jolla luotiin prosessin seurantataulu tietokantaan.

4.4 Ulkoiset riippuvuudet

Projektin käyttämä datakirjasto lisättiin projektin kehitysvaiheessa projektiin käyttäen DLL-tiedostoja. Tiedostot lisätään projektiin käyttäen Visual Studion Reference Manageria (kuva 26). Ne on ensin tallennettava projektin juurikansioon.



KUVA 26. Visual Studio Reference Manager ja DLL-tiedostojen lisäys projektiin

4.5 CSV-tiedoston luonti ja tallennus

Raportointityökalussa käyttäjä valitsee haluamansa hakuparametrit käyttöliittymässä olevista alasvetovalikoista ja klikkaa ”Generate Report” -painiketta, jolloin selain lataa parametreista generoidun CSV-tiedoston. Painikkeen klikkaaminen aiheuttaa ohjaimessa *GenerateReport*-funktion kutsumisen. Se on *IActionResult*-tyyppinen funktio, joka toteutettiin kuvassa 27 esitetyllä tavalla.

```
[HttpPost]
[Route("report.csv")]
[Produces("text/csv")]
public IActionResult GenerateReport(ReportingTool model)
{
return Ok(output);
}
```

KUVA 27. HTTP Post -metodi, joka palauttaa *output*-oliosta generoidun CSV-tiedoston.

Data haetaan datakirjastosta käyttäen hakukriteerinä haettavan yrityksen tunnistetta sekä kuukautta ja vuotta. Datakirjasto palauttaa kerrallaan yhden kuukauden datan yhtä yritystä kohden.

Data parsittiin luokka ja parametri kerrallaan *CsvOutput*-olioon, joka kuvaa yhtä riviä CSV-tiedostossa. Lukuisten *foreach*-silmukoiden avulla *UsageReport*-luokka käytiin läpi ja jokainen *CsvOutput*-rivi lisättiin listaan, joka lopulta palautettiin *HTTPPost*-metodilla, jolloin selain lataa tiedoston automaattisesti. CSV-tiedoston datan muokkaaminen käyttäjien haluamaan muotoon ja selkeästi esitettäväksi oli työläs työvaihe.

4.6 Versionhallinta

Projektille luotiin Git-repositorio, johon sekä datakirjasto että tässä työssä tehty projekti tallennettiin. Projektin jatkokehityksen aloittaminen on helppoa, kun projekti on helposti saatavilla etärepositoriossa. Git:n ammattimaisen käytön opettelu oli myös yksi projektin tavoitteista, joskin ei lopputuloksen kannalta tärkeä. Ammattimainen modernin ohjelmistokehityksen työkalujen käyttö on tärkeää jokaiselle ohjelmistokehittäjälle.

4.7 Järjestelmän testaus

Sovellusta testattiin jatkuvasti kehityksen aikana. Testituloksia raportoitiin järjestelmän pääkäyttäjälle ja saatua palautetta hyödynnettiin projektin kehityksessä. Käyttäjien kannalta tärkeimpänä testattavana ominaisuutena on CSV-tiedostolle generoitavan datan oikeanlainen esitystapa ja CSV-tiedoston muotoilu helppokäyttöiseksi. Laskutusajoa ei projektin aikana testattu, sillä projektin painopiste siirtyi raportointityökalun kehitykseen.

Testeillä havaittiin useita puutteita datakirjaston tarjoamassa datassa ja siksi datakirjastoon tehtiin myös useita muutoksia kehityksen aikana. Lisäksi CSV-tiedoston asettelua muokattiin käyttäjien toiveiden mukaiseksi.

Järjestelmän kehitys ja testaus tehtiin käyttäen testitietokantaa, joka vastaa ominaisuuksiltaan tuotannon tietokantaa, mutta uusinta tuotantodataa ei testitietokannassa ole. Kun järjestelmä siirretään hakemaan dataa tuotantotietokannasta, voidaan järjestelmän datan oikeellisuus varmistaa vertailemalla vanhassa järjestelmässä olevaa dataa. Lopullinen testaus on tämän projektin ulkopuolinen työvaihe.

5 POHDINTA

Projektin tuloksena kehitettiin web-sovellus, jossa on valmius suorittaa laskutusajo ja generoida CSV-raportteja sekä niille soveltuva käyttöliittymä. Nämä ominaisuudet kehitettiin käyttäen ASP.NET Core -kirjastoa ja sen lisäosia. Lisäksi hyödynnettiin MVC-arkkitehtuuria sovelluksen runkona. MVC-arkkitehtuurin ominaisuuksia ja sen soveltuvuutta web-kehityksen pohjana tarkasteltiin. Valmiin sovelluksen monet ominaisuudet liittyivät MVC-arkkitehtuurin keskeisten ominaisuuksien oppimiseen ja käyttöön.

Käytettyjen Microsoftin työkalujen todettiin soveltuvan erinomaisesti sovellukseen, jossa taustalla käytetään C#-kielellä kehitettyä kirjastoa. Se tarjoaa monipuolisia työkaluja sovelluksen eri vaiheiden toteuttamiseen. Erityisen paljon kehitystä helpotti Microsoftin palveluista löytyvä kattava dokumentaatio. Haluttuja ominaisuuksia varten löydettiin tarvittavat lisäosat tai kirjastot. Erityisesti SignalR- ja Razor-kirjastoihin perehdyttiin kattavasti. Lisäksi tutustuttiin tietokannan hallintaan.

Projektin keskeisiksi tavoitteiksi asetettiin laskutusajon toiminnan varmuuden parantaminen ja raportointityökalun laajentaminen. Projektin alussa painopiste oli laskutusajon kehityksessä, mutta siirtyi vaatimuksien tarkentuessa raportointityökalun kehitykseen. laskutusajon suorittava osuus saatiin osittain valmiiksi mutta testausta ei suoritettu tai järjestelmää kokeiltu käytännössä. Raportointityökalu saatiin kehitettyä pitkälle, mutta etenkin testausta, hienosäätöä sekä datakirjaston toimintaa voidaan jatkossa kehittää. Lisäksi järjestelmässä kehitettyjä omaisuuksia voidaan mahdollisesti ottaa käyttöön muissakin järjestelmissä.

LÄHTEET

Microsoft. 2013. ASP.NET MVC Overview. Julkaistu 28.2.2013. Luettu 26.2.2019. https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

Microsoft. 2017. ADO.NET Architecture. Julkaistu 30.3.2017. Luettu 26.2.2019. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-architecture

Microsoft. 2017. ADO.NET Overview. Julkaistu 30.3.2017. Luettu 26.2.2019. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview

Microsoft. 2018. C# Guide. Julkaistu 30.1.2018. Luettu 2.4.2019. https://docs.microsoft.com/en-us/dotnet/csharp/

Microsoft. 2018. Introduction to ASP.NET Core SignalR. Julkaistu 25.4.2019. Luettu 26.2.2019. https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-2.2

Microsoft. 2019. Welcome to the Visual Studio IDE. Julkaistu 19.3.2019. Luettu 5.4.2019. https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

Microsoft. 2019. What is .NET? Luettu 19.2.2019. https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

Microsoft. n.d. Introduction to ASP.NET Core. Päivitetty 4.7.2019. Luettu 4.7.2019. https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

Pelser. J. 2019. Communicate the status of a background job with SignalR. Julkaistu 28.6.2019. Luettu 26.3.2019. https://www.jerriepelser.com/blog/communicate-status-background-job-signalr