Daniel Langhoff

Web-raportointisovelluksen kehitys Fast Report -raportointityökalulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.10.2021

Tiivistelmä

Tekijä:	Daniel Langhoff
Otsikko:	Web-raportointisovelluksen kehitys Fast Report -raportointityökalulla
Sivumäärä:	29 sivua
Aika:	1.10.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Matti Oosi Tuotekehityspäällikkö Thomas Thoden

---

Insinöörityön tavoitteena oli kehittää Hedengren Security Oy:lle web-raportointisovellus, joka toimisi samalla päivityksenä vanhemman Hedsam X -kulunvalvontajärjestelmän raportointiosiolle. Tavoitteena oli myös perehtyä syvemmin Fast Report -raportointityökaluun ja sen ominaisuuksiin sekä itse web-sovelluksen toteutustapaan.

Raporttien luomista varten käytettiin Fast Report -raportointityökalua, joka liitettiin web-sovelluksen kanssa toimimaan yhteen. Toteutuksen parissa käytettiin myös hyödyksi yrityksen REST-ohjelmointirajapintaa, jonka välityksellä saatiin haettua tietoa tietokannasta lopullista raporttia varten.

Insinöörityön lopputuloksena syntyi web-raportointisovellus, jonka avulla käyttäjä pystyy luomaan hänen valitsemilleen henkilöille joko työaikaleimaus-, kulunvalvonta- tai työaikaraportin käyttäjän valitsemilla parametreillaan. Työ toteutettiin käyttämällä ASP.NET Core MVC -ympäristökehystä sekä Fast Report -raportointityökalua.

Avainsanat: Web-sovellus, Fast Report, raportointi, API, MVC

Abstract

Author:	Daniel Langhoff
Title:	Development of Web Reporting Application Using Fast Report
Number of Pages:	29 sivua
Date:	1 October 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Matti Oosi, Senior Lecturer Thomas Thoden, Product Development Manager

---

The aim of the study was to develop a reporting web application for Hedengren Security Oy which would at the same time work as an update to the reporting section of their older access control system Hedsam X. The secondary aim was to become more familiar with the Fast Report reporting tool and its properties and the methods of implementation used to build the web application.

Fast Report was used for generating and creating reports, which were then integrated into the web application for collaboration. The implementation also utilized the company's REST API, which retrieved data from the database for the final report.

The result of the study was a reporting web application that allows the user to create a report of people's working time stamp, access control or working time with the parameters selected by the user. The work was carried out using the ASP.NET Core MVC environment framework and the Fast Report reporting tool.

Keywords: Web application, Fast Report, reporting, API, MVC

Sisällys

Lyhenteet

REST: Representational State Transfer. Eräs arkkitehtuurimalli, jonka avulla voidaan toteuttaa ohjelmointirajapintoja.

API: Application Programming Interface. Ohjelmointirajapinta. Rajapinta, jota vasten ohjelmoidaan. Toimii esimerkiksi käyttäjän ja tietokannan välillä sekä mahdollistaa eri ohjelmien keskustelun keskenään.

MVC: Model-View-Controller. Malli-Näkymä-Käsittelijä. Suunnittelumalli, joka jakaa ohjelman tai sovelluksen komponentit kolmeen osaan. Näin ollen muokkaaminen sallitaan yhdessä komponentissa ilman ongelmien syntymistä muissa komponenteissa.

IIS: Internet Information Services. Microsoftin kehittämä palvelinohjelmistokokonaisuus, jota käytetään Windows-pohjaisissa palvelimissa.

Git: Global Information Tracker. Versionhallintajärjestelmä.

1 Johdanto

Tämän insinöörityön aiheena on kehittää web-sovellus, joka toimisi yhdessä Fast Report -raportointityökalun kanssa. Insinöörityö tehdään Hedengren Security Oy:lle, joka on suomalainen, turvatekniikkaan erikoistunut yritys ja samalla suuri osa yli 100-vuotiasta Hedengren-konsernia. Työn tarkoituksena on kehittää Hedengrenin Hedsam X -kulunvalvontajärjestelmän raportointiosasta käytännöllisempi uusi web-sovellus asiakkaita varten.

Web-sovelluksen avulla käyttäjä pystyy hakemaan erilaista dataa REST-ohjelmointirajapinnan (engl. Application Programming Interface) välityksellä tietokannasta sekä luomaan raportin haetusta datasta. Työssä keskitytään kolmeen erilaiseen raporttimalliin, jotka ovat työaikaleimaus-, kulunvalvonta- ja työaikaraportti.

Käyttäjälle annetaan mahdollisuus vaikuttaa sekä datan hakuun että raportin lopulliseen muotoon käyttäjän valituilla parametreillä. Valittavissa olevia parametrejä ovat muun muassa aikaväli, jolta tietoa haetaan, jokaiselle raportille kuuluvia uniikkeja eri muotoiluja sekä lopullisen raportin näytettävä näkymä html- tai pdf-muodossa. Tämän jälkeen web-sovellus generoi yhdessä Fast Reportin kanssa käyttäjän asettamilla parametreillä koostuvan raportin.

Työssä esitellään muun muassa, mikä Fast Report -raportointityökalu on, miten sitä käytetään sekä miten se saadaan yhdistettyä ja toimimaan yhdessä web-sovelluksen kanssa. Työssä käydään myös läpi web-sovelluksen rakentamiseen käytettäviä tekniikoita sekä miten ja millaiseen lopputulokseen lopulta päädytään.

2 Työn tavoitteet

Insinöörityön tavoitteena on luoda toimiva ja valmis web-sovellus, joka antaa käyttäjälle mahdollisuuden luoda erilaisia raportteja REST API:n välityksellä tietokannasta tulevasta datasta. Työn aikana tullaan keskittymään suunnitteluvaiheessa sovittuihin raporttivaihtoehtoihin. Tarkoitus on saada web-sovellus aluksi Hedengren Securityn sisäiseen käyttöön, jonka jälkeen sitä voisi miettiä eteenpäin asiakkaille.

Tavoitteena on, että käyttäjä pystyisi alkuun valitsemaan, minkälaisen raportin hän haluaa luoda. Riippuen siitä, minkä raportin käyttäjä valitsee, on hänellä mahdollista valita raportin omista parametreistä ne, jotka kiinnostavat häntä eniten. Haulle käyttäjä voi asettaa kalenterin avulla tietyn aikavälin, jolta väliltä tietoja haetaan. Pikatoiminnoiksi on tarkoitus luoda yhden viikon sekä kuukauden aikavälin valitseminen, jotta käyttäjälle olisi olemassa helppo ja nopea tapa asettaa aikaväli raportille.

Web-sovellukseen toteutetaan tietynlainen suodatusominaisuus, jotta käyttäjä voi syöttämillä parametreillaan hajottaa dataa ennen kuin API-kutsuja lähetetään. Suodattimen tehtävänä on mahdollistaa eri henkilöiden suodatus niiden kuuluvien ryhmien, osastojen ja yritysten perusteella. Tavoitteena on myös toteuttaa tehtyjen hakujen tallentaminen sekä nimeäminen, jotta tallennetut haut olisivat aina uudelleen käytettävissä käyttäjälle. Web-sovellukseen kehitetään myös etsi-palkki, jotta käyttäjälle löytyisi kätevämpi sekä nopeampi tapa henkilöiden, ryhmien, osastojen tai yrityksien etsimiseen.

3 Fast Report -raportointityökalu

Projektin oltua vielä suunnitteluvaiheessa, ei Fast Report -raportointityökalun käyttäminen ollut vielä varmaa. Toisena ehdokkaana web-sovelluksen raportointityökaluksi pohdittiin Crystal Reports -nimistä raportointiohjelmaa. Crystal Reports on SAP:n tekemä ohjelma ja on kohdistettu pienille sekä keskisuurille

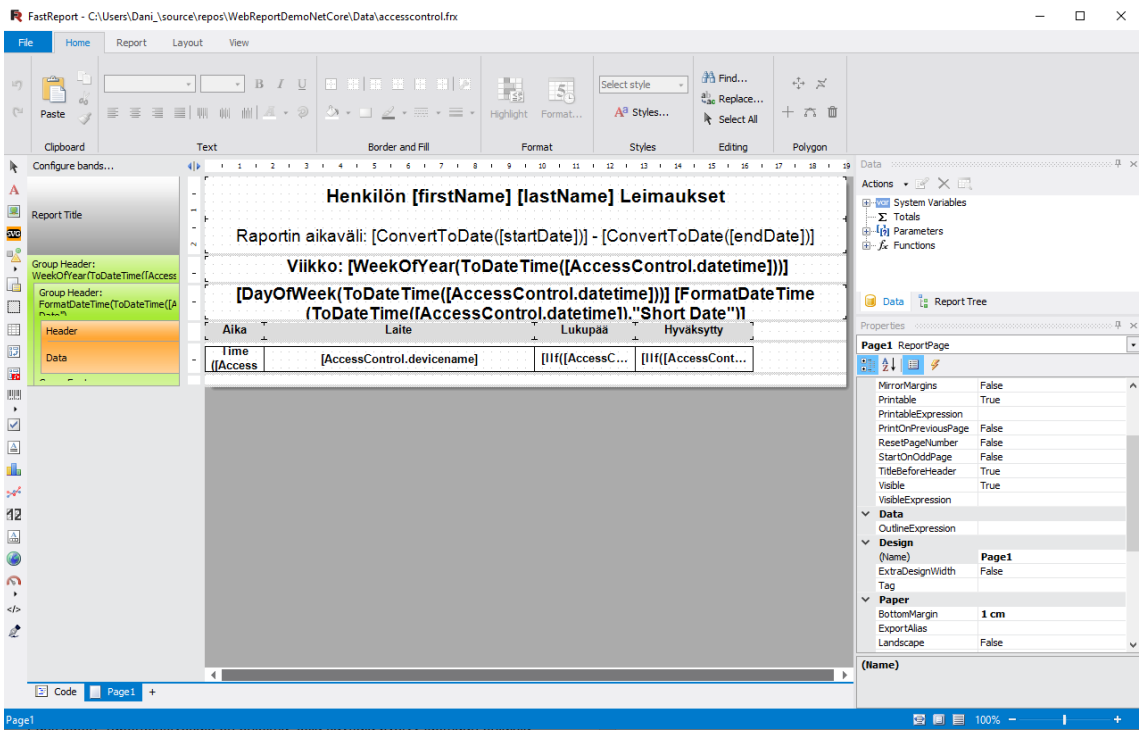
yrityksille. Ohjelmalla pystyy muun muassa saamaan syvempää tietoa liiketoimintatiedoista sekä yhdistää erilaisia raportointitoimintoja suojatulle palvelimelle (SAP Crystal Server). [Crystal Reports.]

Lopulta päädyttiin kuitenkin käyttämään Fast Reportia raportointityökaluna, koska siitä löytyi jo hieman aiempaa kokemusta työpaikalta ja se sopii hyvin yhteen Microsoft Visual Studio 19 -kehitysympäristön kanssa.

Fast Report -raportointityökalu on ohjelma, jolla käyttäjä pystyy itse luomaan erilaisia ja erinäköisiä raportteja. Fast Report -ohjelman käyttäminen tekee raporttien teosta nopeampaa sekä tehokkaampaa itseohjelmoitaviin raportteihin verrattuna.

Ohjelmaan kuuluu muun muassa visuaalinen suunnittelutila raportin tekemistä varten, jossa käyttäjä pystyy itse rakentamaan raportin käyttäen ohjelmaan sisältyviä objekteja avukseen. Objekteilla tarkoitetaan esimerkiksi tekstikenttiä, taulukoita sekä kuvia. Suosituin sekä käytetyin objekti on tavallinen tekstiobjekti. Raportin pystyy myös luomaan manuaalisesti ohjelmointikoodin avulla haluamassaan kehitysympäristössään. Fast Report -ohjelmaan kuuluu myös ennalta määriteltyjä raportteja, jotka voivat toimia niin sanottuina raporttimalleina. Ohjelmasta löytyy myös hyvin käytännöllinen esikatseluikkuna, joka mahdollistaa raportin esikatselun kehittäjälle.

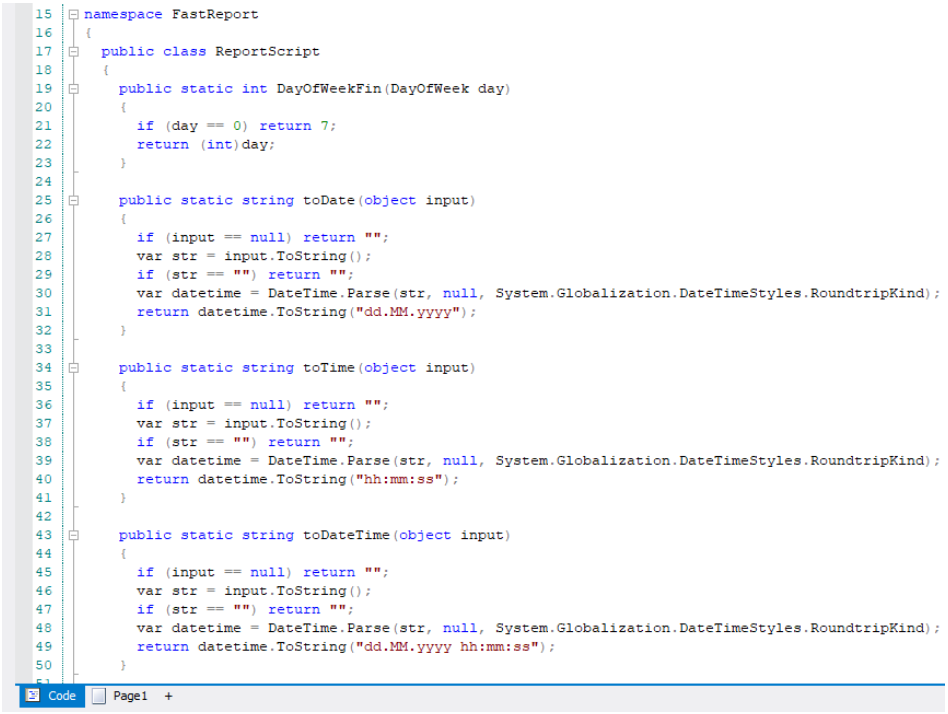
Kuvassa 1 näkyy Fast Report -raportointityökalun visuaalinen suunnittelutila. Suunnittelutilaa hyväksi käyttäen kehittäjä pystyy helposti rakentamaan juuri sellaisia raportteja, joita tarvitaan eri tietojen raportoimista varten.



Kuva 1 Fast Report -raportointityökalun visuaalinen suunnittelutila. Kuvassa näkyy, miltä kulunvalvontaraportti näyttää suunnittelutilassa.

Näiden lisäksi Fast Reportissa löytyy jokaiselle raportille oma ohjelmointiosio (”Code”-välilehti), jossa kehittäjä voi vaikuttaa esimerkiksi raporttien ulkonäköön manuaalisesti ohjelmointikoodin avulla. Ohjelmointiosiossa pystyy myös luomaan omia funktioita, joita voi käyttää suoraan suunnittelutilassa eri objekteissa. Koodia kirjoitetaan tällöin ”ReportScript” -nimiseen luokkaan. Fast Report tarjoaa siis laadukkaat työkalut raporttien suunnitteluun ja toteuttamiseen. [Fast Reports2.]

Kuvassa 2 nähdään pieni osa Fast Reportin ohjelmointiosiosta ja ReportScript -luokassa olevista funktioista. Kuvassa näkyvät muun muassa ”toDate”-, ”toTime”- ja ”toDateTime”-funktiot. Näitä käytetään tarpeen mukaan raportissa, jos halutaan muuttaa dataa erinäköiseksi, eli toisin sanoin formatoida. Esimerkkinä voidaan ottaa API:sta palautuva päivämäärä, joka palautuu ISO 8601 -standardin mukaan. Päivämäärä näyttäisi aluksi tältä: ”1970.01.01”. Kun päivämäärä syötetään toDate-funktiolle, palauttaa se päivämäärän näkyviin näin ”01.01.1970”.



```
namespace FastReport
{
  public class ReportScript
  {
    public static int DayOfWeekFin(DayOfWeek day)
    {
      if (day == 0) return 7;
      return (int)day;
    }

    public static string toDate(object input)
    {
      if (input == null) return "";
      var str = input.ToString();
      if (str == "") return "";
      var datetime = DateTime.Parse(str, null, System.Globalization.DateTimeStyles.RoundtripKind);
      return datetime.ToString("dd.MM.yyyy");
    }

    public static string toTime(object input)
    {
      if (input == null) return "";
      var str = input.ToString();
      if (str == "") return "";
      var datetime = DateTime.Parse(str, null, System.Globalization.DateTimeStyles.RoundtripKind);
      return datetime.ToString("hh:mm:ss");
    }

    public static string toDateTime(object input)
    {
      if (input == null) return "";
      var str = input.ToString();
      if (str == "") return "";
      var datetime = DateTime.Parse(str, null, System.Globalization.DateTimeStyles.RoundtripKind);
      return datetime.ToString("dd.MM.yyyy hh:mm:ss");
    }
```

Code Page1 +

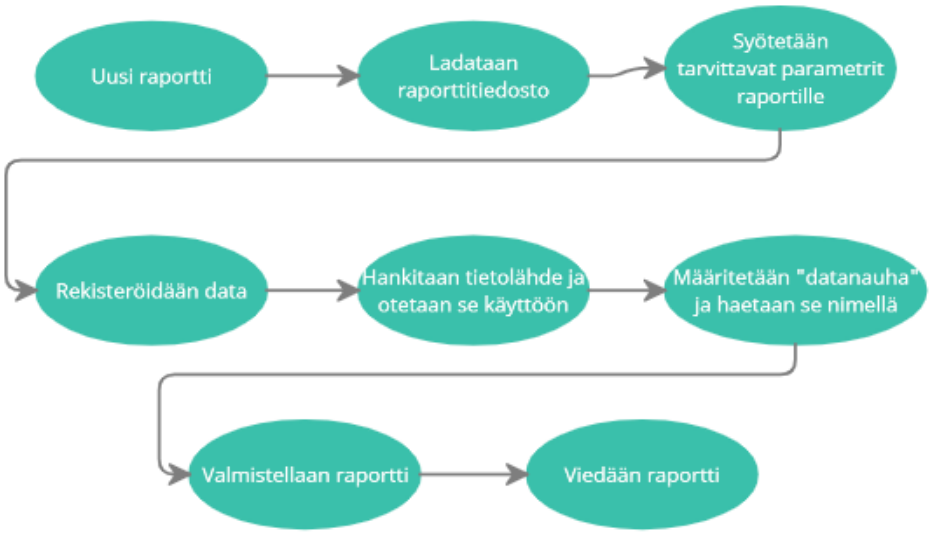
Kuva 2 Fast Reportin "Code"-välilehti yhdelle raportille sekä ReportScript-luokan eri funktioita. Kuvassa ei näy koko luokan sisältöä.

3.1 Raporttipohjat

Raporttipohja on raportointijärjestelmissä yleisesti käytetty termi, jolla tarkoitetaan sitä tiedostoa tai objektia, joka nimeää raportin tarvitsemat elementit, niiden järjestyksen ja muun visuaalisen ilmeen. Kaikki raporttipohjat ja mallit työstetään Fast Report -raportointityökalulla, joista saadaan aikaiseksi Visual Basic (.frx) -tiedostot. Nämä tiedostot ovat binaarilomaketiedostoja, jotka on tarkoitettu muun muassa eri kuvakkeille ja grafiikoille. Tämän jälkeen raporttipohjatiedostot eri raporteille lisätään web-sovellukseen omaan kansioon, jotta niitä voidaan käyttää myöhemmin raportin generointivaiheessa.

3.2 Raportin generointi

Tämä vaihe koostuu monesta eri osasta, ja niiden toteuttamiseen on myös erilaisia tapoja. Usein on myös itse kehittäjästä kiinni, mitä kaikkea tarvitaan ja halutaan käyttää raportin generointi vaiheessa. Tavallisen raportin generointiprosessi aina raportin viemiseen asti voidaan nähdä kuvassa 3.



Kuva 3 Raportin generointiprosessi vaiheineen.

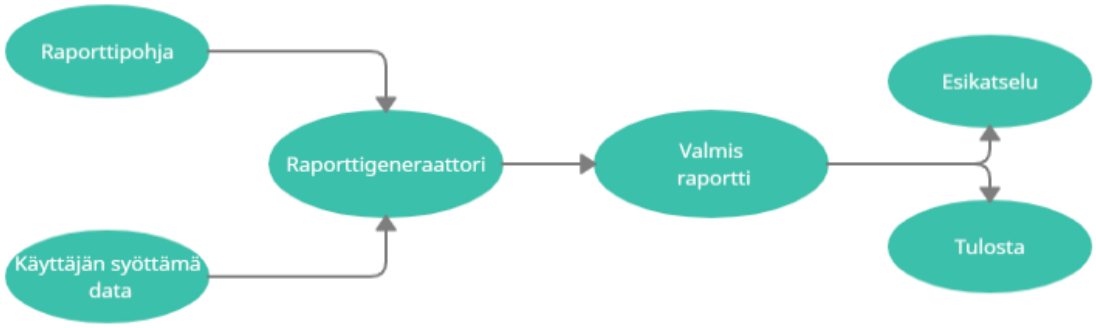
Kuvassa 3 näytetään, miten tavallisen raportin generointiprosessi etenee vaihe vaiheelta. Alkuun luodaan uusi raportti, jota voidaan alkaa täyttämään. Seuraavaksi ladataan raporttipohjatiedosto (.frx) ja syötetään tarvittavat parametrit raportille. Tämän jälkeen data pitää rekisteröidä raportille, ja tietolähteen hankkiminen sekä käyttöönottaminen tapahtuvat sen jälkeen. Lopuksi määritellään ja haetaan niin sanottu ”datanauha” nimellä, valmistellaan raportti ja viedään se eteenpäin joko html- tai pdf-muodossa.

Fast Report mahdollistaa prosessin toteuttamisen Visual Studion puolella, jolloin säästytään esimerkiksi isommasta joukosta yksittäisiä tiedostoja. Fedyashovin artikkelin (Fedyashov 2016) mukaan käytettävyyden kannalta kyseinen prosessi antaa valmiuden myös muuttaa käytettävää raporttimallia sovelluksen ajon aikana. Kehittäjän näkökulmasta raporttien suuri joustavuus sekä

mahdollisuus käyttää ainoastaan yhtä raporttimallia testivaiheessa on suuri etu. [Fedyashov 2016.]

3.3 Valmis raportti

Valmiin raportin luominen koostuu muutamasta eri vaiheesta. Valmiiseen näytettävään raporttiin vaaditaan web-sovelluksen ja Fast Report -raportointityökalun lisäksi myös käyttäjän antamia parametreja.



Kuva 4 Valmiin raportin rakennusprosessi.

Kuvassa 4 voi nähdä, miltä valmiin raportin rakennusprosessi näyttää moduulitasolla. Aluksi valmistellaan kehitysvaiheessa erilaiset raporttipohjat sellaisiksi, kun halutaan, ja asetetaan tarvittavat objektit halutuille paikoille. Objektit täytetään myöhemmin datalla.

Seuraavaksi web-sovelluksen käyttäjä valitsee raporttivaihtoehdoista mieluisensa sekä valitsee mahdollisia parametrejä raportille web-sovelluksen avulla. Tämän jälkeen Fast Report generoi kasaan raportin käyttäjän valitsemilla parametreillä, josta syntyy lopuksi valmis raportti. Tämän jälkeen käyttäjä voi halutessaan tulostaa ulos raportin.

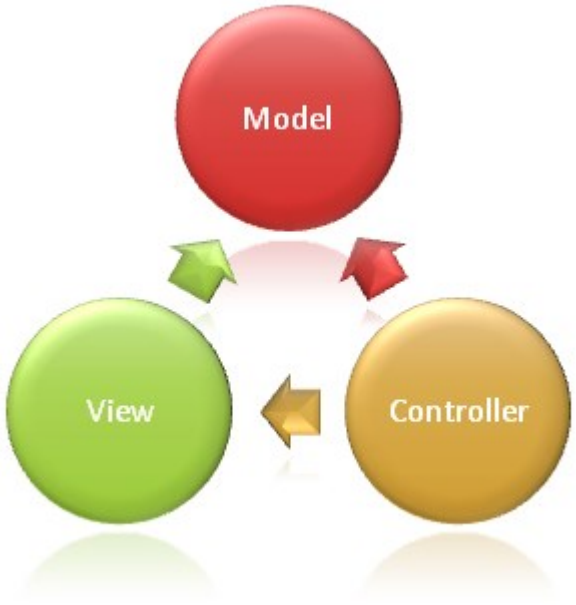
4 Käytettyjä teknologioita web-sovelluksen kehityksessä

4.1 ASP.NET Core MVC

ASP.NET Core MVC on eräänlainen ympäristökehys, joka helpottaa huomattavasti web-sovellusten rakentamista. Se käyttää hyväksi Model-View-Controller suunnittelumallia.

ASP.NET Core on Microsoftin kehittämä ohjelmistokehys, jonka lähdekoodi on avoin kaikille. Tämä ohjelmistokehys on luotu helpottamaan nykyaikaisten, Internetiin yhdistettyjen erilaisten sovellusten rakentamista. ASP.NET Core on monialustainen kehys, joka tarkoittaa sitä, että ohjelmien ja sovellusten kehittäminen onnistuu Windowsin lisäksi myös macOS- ja Linux-käyttöjärjestelmillä. ASP.NET Coren avulla käyttäjä voi muun muassa kehittää web-sovelluksia ja palvelimia sekä paikallisesti että pilvessä. [Roth, Anderson, Luttin 2020.]

Tämä suunnittelumalli tunnetaan myös suomen kielellä nimellä MVC-arkkitehtuuri. MVC-arkkitehtuurin tarkoituksena on jakaa sovelluksen tai ohjelman komponentit kolmeen osaan: malliin (Model), näkymään (View) ja käsittelijään (Controller). [Smith 2020.]



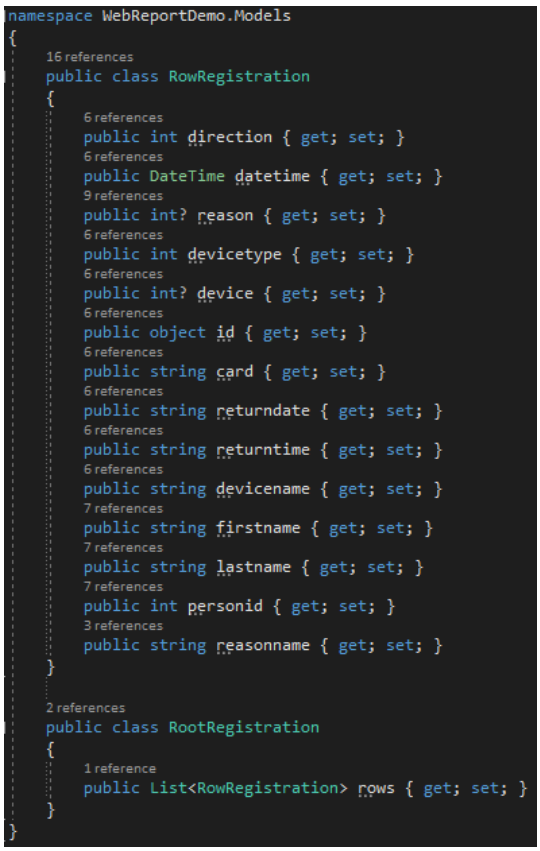
Kuva 5 MVC-arkkitehtuuri ja komponenttien viittaaminen toisiinsa. [Smith 2020.]

Kuvassa 5 pystyy näkemään, miten MVC-arkkitehtuuri käytännössä toimii. Erilaiset käyttäjäpyynnöt ohjautuvat ensiksi käsittelijälle. Käsittelijä toimii yhdessä mallin kanssa ja pyrkii siten suorittamaan käyttäjän haluttuja eri toimenpiteitä. Käsittelijän tehtävänä on myös näyttää tai valita oikea näkymä käyttäjälle ja toimittaa tarvittavat tiedot mallilta.

MVC-arkkitehtuuri tuo mukanansa hyvin paljon hyödyllisiä asioita, joista mahdollisesti tärkeimpänä on se, että komponentteja pystyy helposti muokkailemaan aiheuttamatta ongelmia muissa komponenteissa. Komponenttien ollessa omissa osissaan, helpottaa se huomattavasti sovelluksen tai ohjelman kehitystä, korjaamista sekä testaamista. Uusien kehittäjien on myös helpompi liittyä mukaan aiemmin aloitettuun projektiin, koska MVC-suunnittelumallin ansiosta komponentit pitäisi olla helpommin luettavissa sekä löydettävissä.

4.1.1 Mallin rooli

MVC-arkkitehtuurin osuutta, joka edustaa sovelluksen tai ohjelman tilaa, kutsutaan malliksi. Malliin sisältyvät loogiset toimet suoritetaan aina taustalla ja täten eivät missään vaiheessa ole käyttäjälle näkyvissä. Mallin sanotaan myös sisältävän sovelluksen tai ohjelmiston niin sanotun liiketoimintalogiikan.



```
namespace WebReportDemo.Models
{
    16 references
    public class RowRegistration
    {
        6 references
        public int direction { get; set; }
        6 references
        public DateTime datetime { get; set; }
        9 references
        public int? reason { get; set; }
        6 references
        public int devicetype { get; set; }
        6 references
        public int? device { get; set; }
        6 references
        public object id { get; set; }
        6 references
        public string card { get; set; }
        6 references
        public string returndate { get; set; }
        6 references
        public string returntime { get; set; }
        6 references
        public string devicename { get; set; }
        7 references
        public string firstname { get; set; }
        7 references
        public string lastname { get; set; }
        7 references
        public int personid { get; set; }
        3 references
        public string reasonname { get; set; }
    }

    2 references
    public class RootRegistration
    {
        1 reference
        public List<RowRegistration> rows { get; set; }
    }
}
```

Kuva 6 Tyypillinen malliluokka web-sovelluksessa.

Kuvassa 6 näkyy tyypillinen malliluokka, jonka avulla saa poimittua tietynlaista tietoa API:sta. Tämä luokka (RowRegistration) listaa kaikki tarvittavat muuttujat, jota käytetään silloin, kun haetaan eri työntekijöiden työaikaleimauksia web-sovelluksessa. Kun käyttäjä haluaa nähdä vaikkapa kaikkien työntekijöiden työaikaleimaukset, lähtee komento ensiksi käsittelijälle. Tämän jälkeen se välitetään eteenpäin mallille ja sitten näkymään, jonka käyttäjä lopuksi pystyy näkemään käyttöliittymässä. Lopputuloksena käyttäjä näkee valmiin raportin, jossa on listattu kaikkien valittujen työntekijöiden työaikaleimaukset käyttäjän asettamilla parametreillä.

4.1.2 Näkymän rooli

Näkymän tehtävänä on esittää halutut asiat käyttöliittymän kautta käyttäjälle. Tässä osuudessa ei tulisi löytyä paljon logiikkaan liittyvää asiaa, paitsi tarvittavat logiikat sisällön esittämistä varten. [Smith 2020.]

Projektissani löytyy näkymän (”View”-kansio) alla muutama eri tiedosto omine luokkineen. Tarkoituksena olisi, että jokaisella sivulla on niin sanotusti oma muokattava CSHTML-tiedosto, eli toisin sanoin oma näkymä. CSHTML-tiedosto on Razorin käyttämä C#- ja HTML-verkkosivutiedosto. Näkymän alla sijaitsee myös ”Shared”-kansio, joka on yhteinen näkymä, jossa sijaitsevat sellaiset komponentit, joiden pitäisi näkyä jokaisella sivulla. Tällaisena komponenttiesimerkkinä on esimerkiksi käyttöliittymän yläreunassa sijaitseva navigointipalkki, jonka tulee olla näkyvissä aina, jotta käyttäjällä olisi helppo navigoida näkymästä toiseen.

4.1.3 Käsittelijän rooli

Trygve Reenskaug kirjoittaa, että käsittelijän rooli on olla niin sanotusti linkki käyttäjän ja muun ohjelmiston välillä. Käsittelijälle kuuluu pääsääntöisesti kaksi tehtävää, joista ensimmäinen on eri näkymien päivittäminen käyttäjälle. Toisena tehtävänä käsittelijä kuuntelee ja vastaanottaa käyttäjän antamia syötteitä ja lähettää ne eteenpäin mallille seuraavaa tapahtumaa varten. [Reenskaug 1979.]

4.2 IIS Express

IIS Express (engl. Internet Information Services Express) on Microsoftin kehittämä palvelinohjelmistokokonaisuus. Sitä käytetään projektissa verkkosivujen kehittämiseen ja testaamiseen.

IIS Express mahdollistaa IIS:n viimeisimmän version käytön ja helpottaa samalla käyttäjää verkkosivujen kehittämisessä ja testaamisessa. IIS Express on eräänlainen kevyempi versio IIS:stä, joka on optimoitu kehittäjille. IIS Express sisältää myös muutamia lisäominaisuuksia. Se ei esimerkiksi toimi palveluna eikä vaadi järjestelmänvalvojan käyttöoikeuksia useimpien tehtävien suorittamiseen. IIS Express toimii myös hyvin sujuvasti ASP.NET-sovelluksien kanssa ja tekee monen käyttäjän itsenäisen työskentelyn mahdolliseksi samalla tietokoneella. [Gopalakrishnan 2010.]

IIS Express on siis johdettu IIS 7:stä sekä uudemmista versioista ja tukee IIS:n ydinominaisuuksia. IIS ja IIS Expressillä löytyy kuitenkin keskeisiä eroja. Tärkeimpänä erona on tapa, miten työntekijöiden prosesseja hallinnoidaan. IIS:ssä WAS (engl. Windows Process Activation Service) kontrolloi verkkosovelluksia eikä käyttäjällä ole suoraa hallintaa niistä. IIS Express toimii ilman WAS:ia, joka tarkoittaa sitä, että käyttäjällä itse on täysi hallinta sovellusten aktivoinnista ja deaktivoinnista. [Gopalakrishnan 2010.]

4.3 Git-versionhallinta

Insinöörityön versionhallintana käytetään Git -versionhallintajärjestelmää. Git on kehitetty vuonna 2005 ja kuuluu suosituimpien versionhallintajärjestelmien joukkoon. Git asennetaan lokaalisesti tietokoneelle ja on käytettävissä kokonaan ilman erillisiä pilvipalveluita. Muihin versionhallintajärjestelmiin verrattuna, Git on muun muassa ilmainen sekä helppokäyttöinen. Suurin eroavaisuus muihin järjestelmiin on Git:n käyttämä niin kutsuttu haarautuva malli. Tämä mahdollistaa käyttäjälle eri haarojen luomisen niin, että yksi haara voi koostua itse työstä ja toinen voi olla testihaara. Haarat ovat siten myös helposti yhdistettävissä sekä poistettavissa. [Devmountain.]

4.4 Postman

Postman on todella hyödyllinen apuohjelma API:en tekoon. Ohjelma mahdollistaa tehokkaiden API:en rakentamisen nopeasti. Postman on helppo valinta monelle käyttäjälle, koska sen käyttäminen ei maksa mitään. Ohjelman laaja tukiverkosto vetää myös käyttäjiä puoleensa. Postmanin avulla voi tehdä kaikenlaisia API-kutsuja sekä yhtenä todella hyödyllisenä asiana tarkistaa läpi suuriakin vastauksia eri kutsuille. [Postman.]

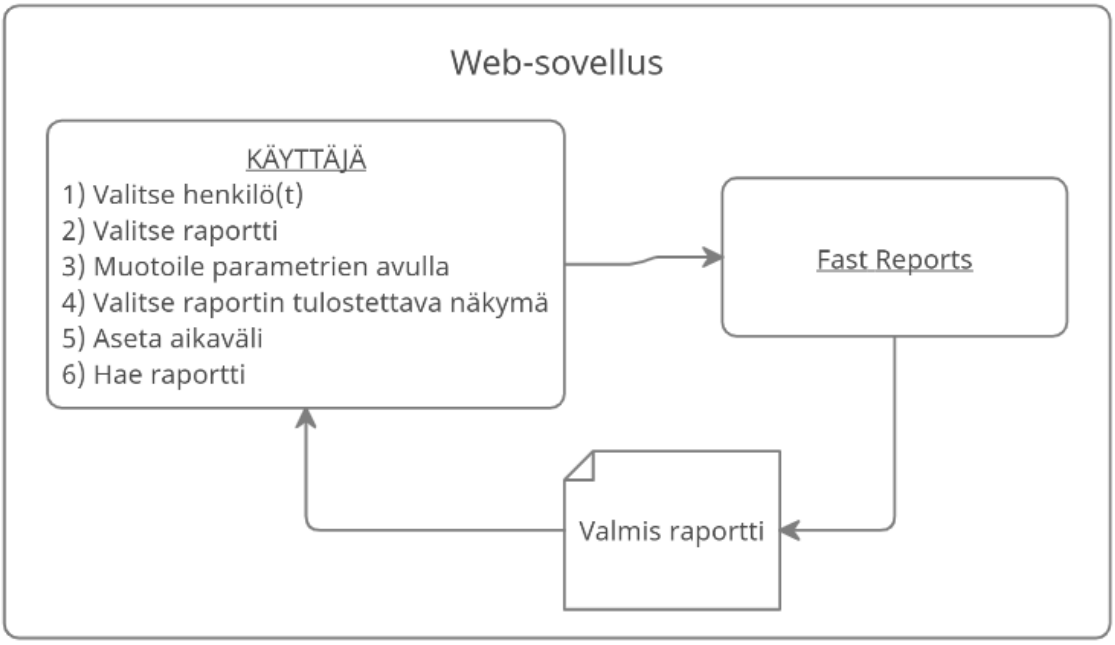
5 Web-raportointisovellus

5.1 Käyttötapaukset

Web-sovelluksella on muutamia eri käyttötapauksia, jotka tulisi olla saavutettavissa sovelluksen avulla käyttäjälle. Tärkeimpiin käyttötapauksiin kuuluu itse raportintekoprosessi, joka koostuu seuraavista vaiheista:

- yhden tai useamman henkilön valitseminen API:sta haetusta henkilölistasta
- halutun raportin valitseminen kolmesta eri raportista
- raportin parametrien valitseminen, jotka halutaan mukaan itse raporttiin ja joiden perusteella raportti muotoutuu
- haettavan raportin muoto tai näkymä (html/pdf)
- aikavälin valitseminen.

Kuvassa 7 voidaan nähdä web-sovelluksen toimivuus käyttäjän näkökulmasta. Kuvassa esitetään kolme erilaista ”moduulia”, jotka muodostavat yhdessä web-sovelluksen toimivuuden. Moduuleina tässä tapauksessa ovat itse käyttäjä, Fast Report -raportointityökalu sekä raportointityökalun avulla valmistuva valmis raportti.



Kuva 7 Sovelluksen toimivuus käyttäjän näkökulmasta. Toiminnallisuus kuvassa edustaa ”Kaikki raportit” -sivua, jossa käyttäjällä on mahdollisuus valita kolmesta raporttimallista haluamansa. Sivulla on myös haettu yrityksen kaikki henkilöt henkilölistaan.

Ajatuksena on, että käyttäjä on kirjautunut onnistuneesti sisään ja suunnannut ”Kaikki raportit” -sivulle. Alkuun käyttäjä pystyy valitsemaan henkilöt, joille hän haluaa tehdä raportin. Sen jälkeen käyttäjän tulee valita haluttu raportti, jota haetaan valituille henkilöille. Seuraavana käyttäjä valitsee valitulle raportille uniikit parametrit, joiden mukaan raportti muotoutuu. Tämän jälkeen käyttäjä valitsee raportin tulostettavan näkymän ja viimeiseksi asettaa aikavälin, jolta väliltä tietoja haetaan raporttiin.

5.2 Toteutus

Web-sovelluksen ohjelmointikielinä toimivat pääasiassa C# ja JavaScript. Muita toteutuksen parissa käytettäviä kieliä ovat HTML (engl. Hypertext Markup Language) ja CSS (engl. Cascading Style Sheets). Projektissa käytetään myös Blazor-nimistä kehystä, jonka avulla interaktiivisen asiakaspuolen verkkokäyttöliittymän rakentaminen onnistuu .NET-kehitysalustaa käyttäen. Projektin versionhallintaan käytetään Git:iä.

5.2.1 AuthController

Sovelluksen käynnistyessä käyttäjältä vaaditaan sisäänkirjautumista. Kirjautuminen tapahtuu käyttäjälle asetetuilla käyttäjätunnuksilla. Kirjautumisesta pitää huolen AuthController-niminen luokka. Web-sovelluksen ollessa yhteydessä REST API:iin oli käyttäjän kirjautuminen luonnollista toteuttaa myös API:n kautta. Tästä johtuen varsinaisen kirjautumisprosessin kehittämiseen ei tarvinnut laittaa sen enempää aikaa tätä työtä tehdessä.

5.2.2 ReportsController

Raportit-sivulla käyttäjä pystyy luomaan kolme erilaista raporttia. Raportit-sivua kontrolloi ReportsController-niminen luokka. Tämä luokka pitää huolen kaikista raportteihin liittyvistä asioista ja funktioista. Muun muassa henkilötietojen hakeminen ja palauttaminen REST API:sta tapahtuu tämän luokan funktion avulla.

Yhtenä tämän tyyppisenä esimerkkinä voidaan ottaa yksityinen asynkroninen GetAllPersons -funktio. Funktiossa annetaan tarvittava URL-osoite, josta tietoa haetaan. Tämän jälkeen luodaan uusi http-pyyntö käyttäen HttpRequestMessage -luokan konstruktoria, jolle annetaan parametreiksi HttpMethod.Get sekä aiemmin luotu URL-osoite. GET-pyynnön mentyä läpi onnistuneesti palautuu palvelimelta ”rowPersons”-niminen lista, joka koostuu yrityksen kaikista henkilöistä. Listan palautumiseen käytetään malliluokkaa RowPersonFlex. Kuvassa 8 voi nähdä GetAllPersons-funktion ohjelmointikoodin osan.

```
private async Task<List<RowPersonFlex>> GetAllPersons()
{
    string url = "v1/persons";
    Request.Cookies.TryGetValue("hedsam-session", out string cookieVal);

    var httpClient = _clientFactory.CreateClient("hedsam");
    var request = new HttpRequestMessage(HttpMethod.Get, url);
    request.Headers.Add("Cookie", "hedsam-session=" + cookieVal);
    var resp = await httpClient.SendAsync(request);

    if (resp.IsSuccessStatusCode)
    {
        var cont = resp.Content;
        var jsonCont = cont.ReadAsStringAsync().Result;
        var jsonArray = "[" + jsonCont + "]";

        List<RootPersonFlex> persons = JsonConvert.DeserializeObject<List<RootPersonFlex>>(jsonArray);
        List<RowPersonFlex> rowPersons = persons[0].rows;

        return rowPersons;
    }

    return null;
}
```

Kuva 8 GetAllPersons-funktio. Hakee yrityksen kaikki henkilöt ja palauttaa ne rowPersons-listana käyttäen hyväksi RowPersonFlex-malliluokkaa.

Edellä mainitun RowPersonFlex-malliluokan avulla saa listattua kaikki tarvittavat muuttujat, joita tarvitaan silloin, kun haetaan yrityksen kaikkia henkilöitä web-sovelluksessa. Tapahtumaa voi aukaista paremmin mvc-arkkitehtuurin avulla seuraavan lailla:

1. Käyttäjän haettaessa yrityksen kaikkia henkilöitä lähtee komento ensiksi käsittelijälle (ReportsController).

2. Seuraavaksi ”komento” välitetään mallille (RowPersonFlex), josta saadaan tarvittavat tiedot kaikkien henkilöiden näyttämiseen. Lista palautetaan takaisin käsittelijälle.

3. Tämän jälkeen käsittelijä on yhteydessä näkymään, joka palauttaa renderöidyn listan takaisin käsittelijälle html-muodossa.

4. Lopuksi käsittelijä lähettää lopullisen tuloksen listana, tässä tapauksessa kaikki yrityksen henkilöt, näkyviin käyttäjälle.

Kuvassa 9 voi nähdä koko RowPersonFlex-malliluokan rakenteen ohjelmointikoodissa.

```
public class RowPersonFlex
{
    27 references
    public string group { get; set; }
    59 references
    public string firstname { get; set; }
    20 references
    public string extra { get; set; }
    23 references
    public string lastname { get; set; }
    27 references
    public string company { get; set; }
    27 references
    public string department { get; set; }
    40 references
    public int id { get; set; }
    11 references
    public int flex { get; set; }
}

10 references
public class RootPersonFlex
{
    5 references
    public List<RowPersonFlex> rows { get; set; }
}
```

Kuva 9 RowPersonFlex-malliluokka. Käytetään GetAllPersons-funktiossa, jotta saadaan lista eri muuttujista näytettäväksi, kun käyttäjä hakee yrityksen kaikkia henkilöitä.

Taulukossa 1 esitetään muutamia ReportsController-luokkaan kuuluvia funktioita ja niiden eri tehtäviä.

Taulukko 1. ReportsController-luokkaan kuuluvia funktioita ja niiden tehtävät.

Funktio	**Tehtävä**
GetReports	Käsittelee käyttäjän antamia parametrejä. Luo valitun raportin niiden perusteella.
GetRegistrationsForMultiplePersons	Hakee API:sta monen ihmisen työaikaleimaukset ja palauttaa niistä listan.

GetRegistrationsForPerson	Hakee API:sta annetun henkilön id:n perusteella työaikaleimaukset ja palauttaa ne listana.
GetAccessControlsForAllPersons	Hakee ja palauttaa kaikkien valittujen henkilöiden kulunvalvontaleimaukset API:sta.
GetAccessControls	Hakee API:sta yhden henkilön kulunvalvontaleimaukset (id:n perusteella).
GetDailyhoursForMultiplePersons	Hakee ja palauttaa valittujen henkilöiden työaikaraportin API:sta.
GetDailyhoursForPerson	Hakee API:sta yhden henkilön työaikaraportin (id:n perusteella).
GetPersonInfo	Hakee henkilön etu- ja sukunimen sekä id:n API:sta ja palauttaa listana.
GetAllTaCodes	Hakee API:sta kaikki olemassa olevat työaika syykoodit ja palauttaa ne listana.

GetAllDevices	Hakee ja palauttaa kaikki olemassa olevat laitteet API:sta ja palauttaa ne listana.

ReportsController-luokan funktiot pitää myös huolen erilaisten tietojen käsittelystä, niin kuin esimerkiksi kahden eri listan yhdistämisestä. Tämä on toteutettu GroupJoin-operaattoria käyttäen. GroupJoin-operaattoria käytetään monessa funktiossa kahden listan yhdistämiseen niiden avaimen perusteella. Tämän jälkeen operaattori ryhmittelee tuloksen vastaavan avaimen avulla ja palauttaa sitten ryhmitetyn tuloksen yhtenä listana. GroupJoinia käytetään muun muassa silloin, kun halutaan hakea henkilöiden kulunvalvontaleimaus raporttia. Raportissa yhtenä kriteerinä on nähdä, millä laitteella eri henkilöt ovat leimanneet sekä mistä ovista on kuljettu.

Kaikkien henkilöiden kulunvalvontaleimaukset palautetaan listana RowAccessControl-nimisen luokan avulla. Kulunvalvontaleimaukset sisältävät viitteen eri laitteisiin, mutta itse laitelista haetaan muualta, koska REST API palauttaa kulunvalvontaleimaukset sekä laitteiden nimet eri osoitteista. Eri laitteiden nimet saadaan siis ulos listana toiselta luokalta nimeltä RowDevice. Listojen yhteisenä avaimena toimii RowAccessControl-luokan kokonaisluku ”device” sekä RowDevice-luokan kokonaisluku ”id”. GroupJoin-operaattorin käyttämistä ohjelmointikoodissa voi hahmottaa kuvassa 10.

```
List<RowAccessControl> accessControls = rowAccessControls.GroupJoin(
    rowDevices,
    x1 => x1.device,
    x2 => x2.id,
    (x1, x2) => new RowAccessControl
    {
        id = x1.id,
        kind = x1.kind,
        datetime = x1.datetime,
        person = x1.person,
        devicetype = x1.devicetype,
        result = x1.result,
        reader = x1.reader,
        device = x1.device,
        card = x1.card,
        firstname = x1.firstname,
        lastname = x1.lastname,
        devicename = x2.Select(x2 => x2.name).FirstOrDefault()
    }).ToList();
```

Kuva 10 GroupJoin-operaattorin käyttäminen GetAccessControlsReport-funktiossa. Kuvassa lisätään ”devicename”-niminen kenttä rowAccessControls-nimiseen listaan, jotta eri laitteiden nimet saadaan mukaan henkilöiden kulunvalvontaleimauksiin.

ReportsController-luokka koostuu myös muistakin funktioista niin kuin esimerkiksi sellaisista, jotka hakevat ja palauttavat API:sta eri ryhmien, osastojen ja yrityksien nimet. Näitä käytetään silloin, kun käyttäjä haluaa suodattaa haettuja henkilöitä tietyn ryhmän, osaston tai yrityksen mukaan.

5.2.3 Kulunvalvonta

Kuvassa 11 nähdään otos valmiista kulunvalvontaraportista. Käyttäjä on hakenut yhden henkilön kulunvalvontaleimaukset parametreineen. Kuvan ”Laite”-sarakkeen alla näkyvien laitteiden nimet edustavat tässä tapauksessa GroupJoinin avulla yhdistettyä ”devicename”-kenttää, eli laitteiden nimiä. ”Aika”-sarake näyttää henkilön leimausajan, ”Henkilö”-sarake taas henkilön nimen, ”Lukupää”-sarake kertoo leimauksen suunnan ja viimeinen sarake ”Hyväksytty” ilmoittaa, onko leimaus onnistunut vai ei.

Viikko: 35

maanantai 30.08.2021

Aika	Henkilö	Laite	Lukupää	Hyväksytty
08.06	Daniel Langhoff	KL: Suunnittelun ulko-ovi	Sisään	Kyllä
13.03	Daniel Langhoff	KL: Varasto / kokoustila	Sisään	Kyllä
14.13	Daniel Langhoff	KL: Varasto / kokoustila	Sisään	Kyllä

Kuva 11 Esimerkki henkilön kulunvalvontaraportista yhdeltä päivältä.

5.2.4 Työaikaleimaus

Kuvassa 12 voidaan nähdä, miltä henkilön työaikaleimausraportti voi näyttää yhdeltä päivältä. Työaikaleimausraportti sisältää sarakkeet ”Aika”, ”Suunta”, ”Syy”, ”Laite”, ”Paluuaika” sekä ”Id”. Tässä tapauksessa henkilö on normaalisti leimannut itsensä sisään ilman syytä töihin tullessaan. Paluuaika on luonnollisesti myös tyhjä sisäänleimauksen yhteydessä. Henkilön lähdettyä töistä on hän leimannut itsensä ulos. Ulosleimauksen yhteydessä voi henkilö asettaa tietyn syyn leimaukselle sekä paluuajan, esimerkiksi paluuaika pidemmältä lomalta. Kuvassa 12 henkilö on leimannut itsensä ulos ilman syytä ja paluuaikaa, eli toisin sanoin tavallinen ulosleimaus työpäivän päätteeksi.

Viikko: 35

maanantai 30.08.2021

Aika	Suunta	Syy	Laite	Paluuaika	Id
08.18	Sisään		TA terminal		202108-827
16.38	Ulos		TA terminal		202108-858

Kuva 12 Yhden henkilön työaikaleimausraportti yhdeltä päivältä.

5.2.5 Työaikaraportti

Viimeisenä raporttina on henkilön työaikaraportti. Kuvassa 13 voidaan nähdä yhden henkilön työaikaraportti kokonaiselta viikolta. Työaikaraportissa nähdään kaikki raportin parametrit, jotka toimivat samalla sarakkeiden otsikkoina. Ne ovat ”Päivä”, ”Suunniteltu vuoro”, ”Tehty vuoro”, ”Normaali”, ”Tauko”, ”Lounas”, ”Liukuma” ja ”Totaali liukuma”.

Työaikaraportin avulla käyttäjä pystyy näkemään muun muassa henkilön sisään- ja ulosleimauksien kellonajat ”Tehty vuoro” -sarakkeen alla. Saman sarakkeen alla löytyy myös yhteenlaskettu työaika tunneissa jokaiselta päivältä. Viereisessä ”Normaali”-sarakkeessa nähdään henkilön normaali työaika (07:30), josta on miinustettu lounastauon aika (00:30). Raportti näyttää myös käyttäjälle henkilön saavutetun liukuman jokaiselta päivältä sekä henkilön nykyisen, eli totaaliliukuman, määrän.

Päivä	Suunniteltu vuoro	Tehty vuoro	Normaali	Tauko	Lounas	Liukuma	Tot.liukuma
Viikko: 34							
ma 23.08.2021	08:00-16:00 (07:30)	07:40-16:06 (07:56)	07:30	00:30	00:30	00:26	05:28
ti 24.08.2021	08:00-16:00 (07:30)	07:38-16:27 (08:19)	07:30	00:30	00:30	00:49	06:17
ke 25.08.2021	08:00-16:00 (07:30)	08:23-16:00 (07:07)	07:30	00:30	00:30	-1:37	05:54
to 26.08.2021	08:00-16:00 (07:30)	07:36-16:00 (07:54)	07:30	00:30	00:30	00:24	06:18
pe 27.08.2021	08:00-16:00 (07:30)	07:56-15:47 (07:21)	07:30	00:30	00:30	-1:51	06:09
la 28.08.2021	08:00-16:00 (07:30)	00:00-00:00 (00:00)	00:00	00:00	00:00	00:00	06:09
su 29.08.2021	08:00-16:00 (07:30)	00:00-00:00 (00:00)	00:00	00:00	00:00	00:00	06:09
Yhteenlasketut	52:30	38:37		02:30	02:30		06:09

Kuva 13 Henkilön yhden viikon työaikaraportti.

5.3 Ulkonäkö ja näkymät

Mitään tiettyjä kriteerejä tai toivomuksia ulkonäön suhteen ei ollut työtä aloitellessa. Alkuun käytettiin ASP.NET Coren automaattisesti luotua projektin oletusmallia. Oletusmallin avulla oli helppo aloittaa web-sovelluksen kehitys, koska valmiit esimerkki näkymät ja kontrollerit olivat jo tehtyinä. Myöhemmin projektin edetessä muokkailtiin esimerkki näkymiä ja muuta ulkonäköä enemmän Hedengrenin näköiseksi.

5.3.1 Sisäänkirjautuminen

Jokaiselle käyttäjälle on tehty omat tunnukset sisäänkirjautumista varten. Käyttäjän kirjautuessa lähetetään HTTP-kutsu (POST) kirjautumisrajapintaan. Kutsu sisältää käyttäjän käyttäjätunnuksen ja salasanan. Tämän jälkeen kutsun vas-

tauksen statuskoodi kertoo kirjautumisen onnistuneen tai epäonnistuneen riippuen siitä, täsmäsikö käyttäjän antama tunnus ja salasana kirjautumisrajapinnan kanssa. Onnistuneen kirjautumisen yhteydessä otetaan myös eväste talteen myöhempiä API-kutsuja varten.

Kuva 14 Web-sovelluksen sisäänkirjautuminen.

Kuvassa 14 nähdään web-sovelluksen sisäänkirjautumissivu. Se koostuu kahdesta eri kentästä, joihin käyttäjän tulee syöttää sekä käyttäjätunnuksensa että salasanansa. Sivulla on myös nappi, jonka avulla käyttäjä voi kirjautua sisään.

5.3.2 Raportit-pääsivu

Raportit-pääsivu koostuu ikkunasta, joka on jaettu kolmeen eri osaan. Ensimmäinen ”Tallennetut haut” -osa tulisi sisältää käyttäjän omia tallennettuja hakuja. Tämän osan toiminnallisuutta ei saatu toimimaan halutulla tavalla insinöörityön aikana, joten sen kehittäminen jatkuu jatkokehityksen parissa.

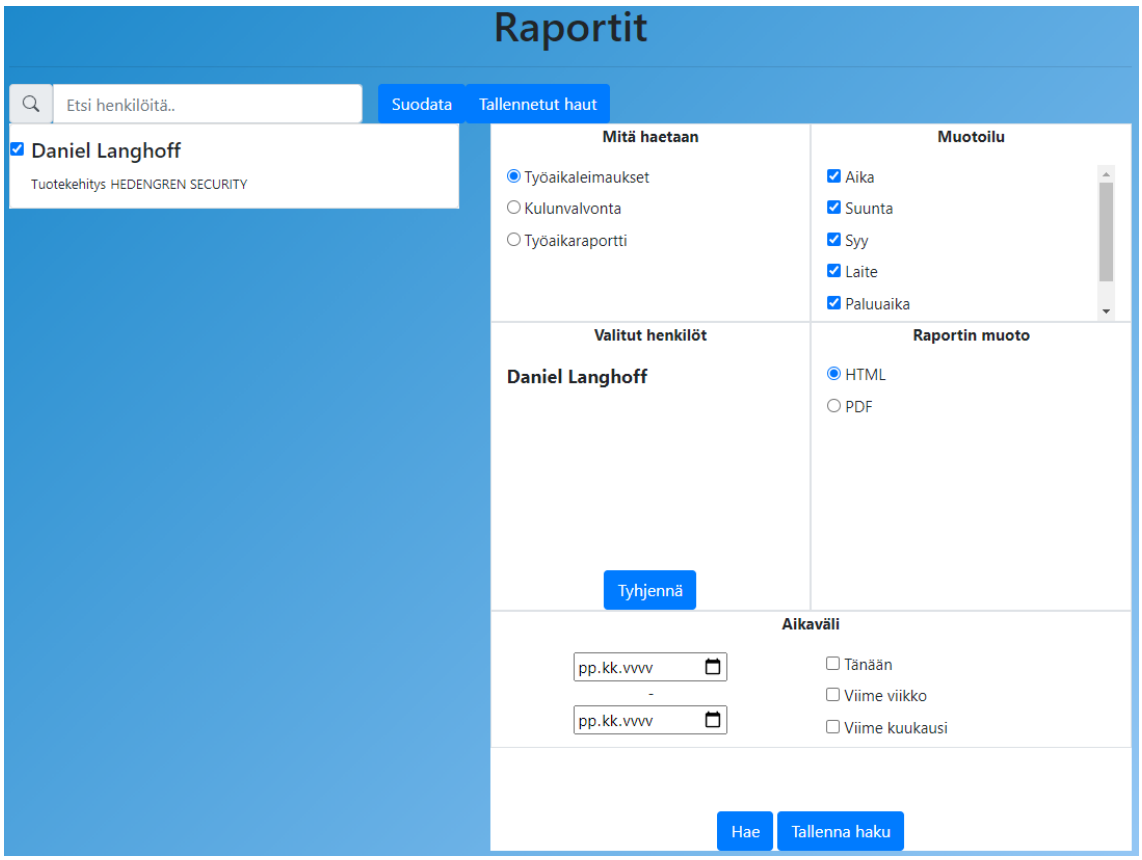
Seuraava ”Raporttimallit”-osa koostuu neljästä eri toimintalinkistä. Käyttäjä valitsee, haluaako hän tehdä raportin työaikaleimauksista, kulunvalvontaleimauksista tai työajasta. Viimeisenä vaihtoehtona ovat kaikki raportit, jotka mahdollistavat minkä tahansa raportin luomisen.

Viimeinen osa koostuu eräänlaisesta hausta, jonka avulla käyttäjä pystyy ensiksi hakemaan henkilöt, jotka kuuluvat pelkästään tiettyyn ryhmään, osastoon tai yritykseen. Haku rajoittaa siis haettujen henkilöitten määrää. Tämän jälkeen käyttäjä voi keskittyä vain tietyn ryhmän, osaston tai yrityksen henkilöihin ja luoda niille haluttuja raportteja. Kuvassa 15 voi nähdä otoksen ”Raportit”-pääsivusta web-sovelluksessa.

Kuva 15 Raportit-pääsivun näkymä web-sovelluksessa.

5.3.3 Kaikki raportit

Näkymä, joka näkyy kuvassa 16, on sivu, jossa käyttäjä pystyy valitsemaan kaikista kolmesta raportista haluamansa. Tällä ”Kaikki raportit” -sivulla listataan kaikki yrityksen henkilöt. Sivulla löytyy hakukenttä, jonka avulla käyttäjä voi etsiä henkilöitä, ryhmiä, osastoja ja yrityksiä. Sivulla löytyy myös suodata-nappi, jonka avulla käyttäjä voi suodattaa haettuja henkilöitä. Tälle sivulle on myös tehty väliaikainen toiminto, joka mahdollistaa käyttäjälle eri hakujen tallentamisen lokaalisti. ”Kaikki raportit” -sivu koostuu myös muista erilaisista parametreista, niin kuin muotoilu, raportin muoto sekä aikaväli, joihin käyttäjä voi vaikuttaa haluamallaan tavallaan. Näitä käytetään sitten valmiissa raportissa.



Kuva 16 ”Kaikki raportit”-sivu, jossa on haettu henkilölistasta tietyn henkilön nimi. Kaikki automaattiset parametrit valittuina.

5.4 Testaus

Web-sovelluksen testausta varten on käytetty manuaalista testaustapaa. Testausta suoritettiin pääsääntöisesti koko ajan, kun jotain uutta kehitettiin tai saatiin toteutettua web-sovellukseen. Myös API-kutsujen toimivuus testattiin päivittäin, kun web-sovellusta ajettiin läpi lukuisia kertoja. Testaukset ajettiin aina Microsoft Visual Studiosta IIS Expressin avulla, joka aukaisi web-sovelluksen selaimeen. Selaimena toimi pääsääntöisesti Google Chrome.

Myös itse Fast Report -raportointityökalua testattiin läpi. Muun muassa päivien ja viikkojen sekä henkilöiden etunimien järjestelyt piti tarkasti testata useamman kerran, jotta saatiin tiedot oikeaan järjestykseen.

Testauksen yhteydessä löydettiin myös pieni ongelma, joka liittyi API-kutsuihin. Ongelmana oli se, kun käyttäjä yritti hakea henkilöitä, vaikka tietyn ryhmän perusteella ja itse ryhmän nimi koostui sekä kirjaimista että erikoisemmista merkeistä. Tämä kuitenkin korjaantui melko helposti käyttämällä WebUtility-luokan UrlEncode-funktiota, joka muuntaa tekstimerkkijonon URL-koodatuksi merkkijonoksi. Tämä toimenpide piti tehdä jokaiselle API-kutsulle, jossa saman kyseisen ongelman toistuminen oli mahdollista.

5.5 Jatkokehitysideat

Yhtenä jatkokehitysideana voisi olla Hedsam X -kulunvalvontajärjestelmän muiden toiminnallisuuksien kehittäminen raportointiosion lisäksi. Esimerkiksi henkilöiden käyttäjäoikeuksien muokkaamisen ja muiden asetuksien asettamisen voisi toteuttaa web-sovellukseen jollain tavalla.

Sovellukseen voisi myös kehittää lisää erilaisia raporttivalintoja. Esimerkiksi henkilöraportti voisi olla tarpeena. Sen avulla käyttäjä saisi luotua raportin valittujen henkilöiden yhteystiedoista. Myös raportti eri henkilöiden leimausmääristä eri laitteilla voisi olla kiinnostava toteuttaa.

Tavoitteena tässä työssä mainittu käyttäjän edellisten hakujen tallentaminen nimellä voisi lisätä myös jatkokehitysideoihin, jotta se saataisiin oikealla ja halutulla tavalla toteutettua.

Web-sovellus on toiminut tähän mennessä IIS Expressin avulla ja käynnistynyt lokaalisti omalla koneella sille määrätyssä portissa. Jatkokehitystä silmällä pitäen olisi web-sovellus tarkoitus saada omalle palvelimelle julkista käyttöä varten.

6 Yhteenveto

Insinöörityössä kehitettiin web-sovellus, joka mahdollistaa raportoinnin henkilöiden työaikaleimauksista, kulunvalvonnasta sekä työajasta. Jotta raporttipohjia ei itse tarvinnut alkaa tekemään alusta asti käsin, käytettiin työssä hyödyksi Fast

Report-nimistä raportointityökalua. Fast Report on oma ohjelmansa, joka liitettiin web-sovelluksen kanssa toimimaan yhteen. Sen avulla aikaansaatiin raporttipohjat, raporttien generointi sekä valmiin raportin näyttäminen käyttäjälle.

Tällä työllä pyrittiin nykyaikaistamaan vanhemman kulunvalvontajärjestelmän raportointiosiota niin, että siitä saataisiin toteutettua itsenäinen web-sovellus, joka pyörisi palvelimella ja olisi asiakkaille käytössä. Kirjoittajan mielestä tässä onnistuttiin melko hyvin, vaikka sovellusta ei vielä saatu asiakkaille asti kokeiluun eikä ihan kaikkia toiminnallisuuksia saatu toteutettua halutulla tavalla.

Raportointisovellusta käytettiin koko insinöörityön ajan testimielessä yrityksen sisällä. Sovelluksen kehitystä tullaan jatkamaan, ja kun se läpäisee yrityksen omat testaukset, voi sitä miettiä eteenpäin asiakkaillekin.

Koko insinöörityö eri vaiheineen on opettanut minua paljon. Ennen työn aloittamista muun muassa Visual Studio -kehitysympäristö, C#-ohjelmointikieli sekä Fast Report -raportointityökalu olivat kaikki uutta. Työ vaati siis paljon oppimista ja tutkimista kaiken muun tekemisen ohella. Olen kuitenkin tyytyväinen saavutettuun lopputulokseen.

Lähteet

Altvater, Alexandra. 2017. What is IIS Express? How It Works, Tutorials, and More. Verkkoaineisto. <https://stackify.com/what-is-iis-express/>. 18.7.2017. Luettu 29.6.2021.

Crystal Reports. Verkkoaineisto. <https://www.crystalreports.com>. Luettu 1.9.2021.

Devmountain. Git vs. GitHub: What's the difference? Verkkoaineisto. <https://blog.devmountain.com/git-vs-github-whats-the-difference/>. Luettu 23.8.2021.

Fast Reports. 2008–2021. Verkkoaineisto. <https://www.fast-report.com/public_download/docs/FRNet/FRNetUserManual-en.pdf>. 2021.2. Luettu 22.6.2021.

Fast Reports2. FastReport .NET. Verkkoaineisto. <https://www.fast-report.com/en/product/fast-report-net/>. Luettu 1.8.2021.

Fedyashov, Dmitriy. 2016. How to create report from user application in FastReport .NET. Verkkoaineisto. <https://www.fast-report.com/en/blog/68/show/>. 28.2.2016. Luettu 16.8.2021.

Gopalakrishnan, Vaidy. 2010. IIS Express Overview. Verkkoaineisto. <https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview>. Luettu 18.7.2021.

Postman. Verkkoaineisto. <https://www.postman.com>. Luettu 1.8.2021.

Reenskaug, Trygve. 1979. Models-Views-Controllers. Verkkoaineisto. <https://folk.universitetetioslo.no/trygver/1979/mvc-2/1979-12-MVC.pdf>. Luettu 1.8.2021.

Roth, Daniel., Anderson, Rick. & Luttin, Shaun. 2020. Introduction to ASP.NET Core. Verkkoaineisto. <https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-2.1>. 17.4.2020. Luettu 20.8.2021.

Salo, Antti. 2015. Model-View-Controller-arkkitehtuuri JavaScriptissä. Verkkoaineisto. <https://www.theseus.fi/bitstream/handle/10024/94063/Salo_Antti.pdf?sequence=1&isAllowed=y>. Luettu 19.7.2021.

Smith, Steve. 2020. Overview of ASP.NET Core MVC. Verkkoaineisto. <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0>. Luettu 18.7.2021.