SQL SERVER -TIETOKANNAN KÄSITTELY ASP.NET MVC - SOVELLUKSESSA WEB API -RAJAPINNAN KAUTTA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan korkeakoulukeskus
Tietojenkäsittelyn koulutusohjelma

Syksy, 2019

Kati Mattila

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Hämeenlinnan korkeakoulukeskus

Tekijä Kati Mattila **Vuosi** 2019

Työn nimi SQL Server -tietokannan käsittely ASP.NET MVC -sovelluksessa Web API -rajapinnan kautta

Työn ohjaaja Tommi Lahti

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli ohjelmoida toimeksiantajayritykselle ASP.NET-pohjainen käyttöliittymä ja ohjelmointirajapinta. Toimeksiantaja halusi pysyä nimettömänä. Tarkoitus oli luoda ASP.NET-sovellus, jolla Microsoftin SQL Serverissä sijaitsevaa tietokantaa voitaisiin käyttää tietoturvallisesti rajapinnan kautta. Tutkimuskysymykset liittyivät tietoturvallisuuteen ja siihen, mikä olisi sopiva tapa toteuttaa ohjelmointirajapinta .NET-ympäristössä. Toimeksiantajalla ei vielä ollut tämänkaltaista sovellusta ja työn tulos on tekniikkademo, jota tullaan mahdollisesti käyttämään tulevaisuuden projekteissa.

Työssä tutkitaan Microsoftin .NET-ympäristöä, ohjelmointirajapintoja, tietokantoja ja ASP.NET-sovelluksen tietoturvaa. .NET-ympäristöön sopiva sovellus oli toimeksiantajan rajaus, sillä heidän nykyiset sovelluksensa toimivat .NET-ympäristössä. Käyttöliittymä päädyttiin tekemään ASP.NET MVC -mallia hyödyntäen ja ohjelmointirajapinnaksi valittiin Web API, joka noudattaa REST-arkkitehtuuria. Web API koettiin sopivaksi ohjelmointirajapinnaksi .NET-ympäristöön. Tietokantoja käydään läpi yleisellä tasolla ja selvitetään myös, kuinka luoda tietokanta käyttäen hyväksi ASP.NET-sovelluksessa kirjoitettua luokkakoodia. Tietoturvallisuuteen liittyen tutkittiin sovelluksiin kohdistuvia yleisimpiä uhkia ja ratkaisuja niihin.

Opinnäytetyössä onnistuttiin ohjelmoimaan toimeksiantajalle tekniikkademo, jossa käytetään hyödyksi ASP.NET MVC -mallia sekä Web API:a tietoturva-asiat mahdollisimman hyvin huomioon ottaen.

Avainsanat ASP.NET MVC, Web API, ohjelmointirajapinnat

Sivut 36 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Author Kati Mattila **Year** 2019

Subject Using SQL Server database from ASP.NET MVC application via Web API

Supervisor Tommi Lahti

ABSTRACT

The aim of this thesis was to program ASP.NET-based web application which uses Microsoft SQL Server database via Web API. All functions should be secured so anonymous user should not be able to make any changes to database. The commissioner was interested how this kind of application would work because they did not have anything like this before. They wanted to stay anonymous.

A theory part of this thesis consists of Microsoft .NET environment, Application Programming Interfaces (API's), databases and security of ASP.NET-application. .NET environment was researched because the commissioner works with .NET-applications so they defined that application should work in .NET environment.

Web application was programmed with ASP.NET MVC -model and API was created with Web API. Web API works well in .NET environment and MVC design pattern was easy and efficient to work with. Databases were explored generally. Entity Framework Code First approach was also clarified. In a security chapter, most common threats for web application were walked through and how to prevent from them.

In this thesis ASP.NET MVC -application and API were successfully programmed. Connection from application to Microsoft SQL Server was made with security. The result will be possibly used as a technology demonstration in the commissioner's future projects.

Keywords ASP.NET MVC, Web API, Application Programming Interfaces

Pages 36 pages

SISÄLLYS

1 JOHDANTO

Internet ja sen kautta saatavilla olevat palvelut ovat arkipäivää suurelle osalle ihmisistä. Niin työ- kuin yksityiselämässä käytetään yhä enemmän palveluita ja ohjelmistoja Internetin välityksellä. Halutaan käyttää jotain tiettyä tietoa (joka sijaitsee usein jonkinlaisessa tietokannassa), ehkä muokata tietoja tai lisätä uusia ja tämä onnistuu mistä vain, kunhan ollaan Internet-yhteyden päässä. Tietojen käyttö tapahtuu yleensä jonkinlaisen rajapinnan kautta.

Tämän opinnäytetyön tavoitteena on ohjelmoida toimeksiantajayritykselle ASP.NET-pohjainen sovellus sekä ohjelmointirajapinta eli API (Application Programming Interface). Työ on toiminnallinen opinnäytetyö. ASP.NET-sovelluksesta luodaan yhteys Microsoftin SQL Serverissä olevaan tietokantaan. Sovelluksen avulla tulee voida käsitellä tietokannassa olevia tietoja tietoturvallisesti ohjelmointirajapinnan kautta. Tietokannassa voisi olla esimerkiksi kuljetuksissa käytettävien ajoneuvojen tietoja, joita sovelluksen kautta lisätään, poistetaan tai muokataan. Sovellusta voidaan käyttää yrityksen sisäisesti tai sen ulkopuolella. Tämän vuoksi tietoturva-asiat ovat tärkeä osa työtä ja sovellukseen luodaan myös kaksi käyttäjäryhmää, joilla molemmilla on eri tasoiset oikeudet toimintojen suorittamiseen.

Toimeksiantajayrityksellä ei vielä ole tämänkaltaista sovellusta ja opinnäytetyön tulosta tullaan mahdollisesti käyttämään tulevaisuuden projekteissa. Työn tekijää kiinnosti erityisesti ohjelmointitaitojen kehittäminen ja siltä pohjalta toimeksiantaja laati sopivan aiheen, josta olisi hyötyä molemmille osapuolille.

Tutkimuskysymyksiksi laadittiin kaksi kokonaisuutta:

- Mikä on järkevä ja moderni tapa toteuttaa ohjelmointirajapinta .NET-ympäristössä?
- Miten varmistetaan sovelluksen tietoturvallisuus?

2 MICROSOFTIN .NET-YMPÄRISTÖ

Työn toimeksiantajan jo olemassa olevat ohjelmistot on toteutettu Microsoftin .NET-teknologialla. Tämän vuoksi vaatimuksena oli käyttää myös opinnäytetyössä .NET-ympäristöön sopivia teknologioita. Tässä luvussa käydään läpi yleisesti Microsoftin .NET-ympäristöä työn rajauksen puitteissa.

2.1 .NET-ympäristö yleisesti

.NET-alusta eli .NET platform on Microsoftin kehittämä avoimen lähdekoodin ympäristö ohjelmistokehitykseen ja sitä voidaan käyttää monella eri ohjelmointikielellä. Alusta tarjoaa luokkakirjastoja kaikenlaisten sovellusten tekoon, jotka toimivat useilla eri käyttöjärjestelmillä (esim. Windows, iOS, Android, Linux). .NET-alustaa käyttämällä voidaan tehdä esimerkiksi web-sovelluksia, ohjelmia tietokoneelle ja älypuhelimelle tai vaikkapa IoT-laitteelle. Kehitystyökalujakin on saatavilla useampi vaihtoehto, esimerkkinä mainittakoon Visual Studio. (Microsoft, 2019h; Microsoft, 2019k)

Visual Studio on Microsoftin kehitysympäristö ohjelmistokehitykseen ja se on osa .NET-tuoteperhettä. Ohjelman avulla kehittäjä voi helposti luoda uutta koodia (eri kielillä), testata ja muokata sitä. Myöskin omien sovellusten julkaisu onnistuu Visual Studion kautta. Ohjelmassa on mm. automaattiset ehdotukset koodin kirjoittamiseen, virheenluku, mahdollisuus graafisten sovellusten luontiin ja paljon muuta. Visual Studio on saatavissa Windows- ja Mac-käyttöjärjestelmille. (Microsoft, 2019f)

.NET-ohjelmistokehys eli .NET Framework on osa .NET-alustaa ja se on kehitetty Windows-ympäristön sovelluskehitykseen. Sillä voidaan tehdä muun muassa internetsivustoja, palveluita ja työpöytäohjelmia. Ohjelmistokehyksen kaksi tärkeintä osaa ovat Common Language Runtime (CLR) ja .NET Framework Class Library (FCL). CLR tarjoaa palvelut, jotka ajetaan suoritettavien ohjelmien taustalla. FCL sisältää yleiskäyttöiset luokat toiminnallisuuksille sekä ohjelmointirajapinnan eli API:n. (Microsoft, 2019i)

2.2 ASP ja ASP.NET

ASP on lyhenne sanoista Active Server Pages ja Microsoft suunnitteli sen toiminnallisten verkkosivujen luomiseen. Aiemmin yleisessä käytössä olivat staattiset verkkosivut, joilla ei ollut toiminnallisuuksia. Käytännössä staattisella verkkosivulla näkyi vain tietty, muuttumaton tieto (vertaa esim. multimediaelementit tai tietokannasta tietoja hakeva sivusto). Sivustoille kaivattiin vuorovaikutusta ja näin syntyi ASP. Ajan myötä ohjelmoijien tarpeet kuitenkin kasvoivat ja ASP ei enää kattanut kaikkia ohjelmoijien tarpeita. Tämän vuoksi kehitettiin ASP.NET-ohjelmistokehys web-sovellusten kehitysympäristöksi. (Martin & Tomson, 2002, s. 6; Inkinen, 2003, s. 36)

ASP.NET-ohjelmistokehys julkaistiin vuonna 2002 ja se tarjosi kehittäjille parempia työkaluja web-sovellusten luomiseen kuin ASP. Tämä uusi ohjelmistokehys antoi kehittäjille käyttöön .NET-arkkitehtuurin ominaisuudet. Suurimmat erot ASP:n ja ASP.NET:in välillä ovat ohjelmointikielissä ja arkkitehtuurissa. ASP.NET tukee .NET-ympäristön monipuolisempia ohjelmointikieliä ja uudistuneen arkkitehtuurin myötä ohjelman suorittaminen on nopeutunut. (Galloway, Wilson, Allen, & Matson, 2014, ss. 1-2; Inkinen, 2003, ss. 36 & 40-55)

ASP.NET voidaan jakaa kahteen ryhmään: web-lomakkeet (Web Forms) ja verkkopalvelut (Web Services). Web Forms -sovellukset ovat käyttöliittymäpohjaisia ja vuorovaikutteisia. Web Services -palvelut taas käsittävät toiminnalliset palvelut, joita muut sovellusympäristöt voivat käyttää. (Galloway ym., 2014, ss. 1-2; Inkinen, 2003, ss. 36 & 58)

ASP.NET toimii useilla erilaisilla laitteilla: esimerkiksi tietokoneella, tabletilla ja matkapuhelimella. Ja käytetyn tyylin mukaan yhdellä (Windows) tai useammalla alustalla (Windowsin lisäksi Linux, macOS sekä Docker). .NET-ympäristö tarjoaa ohjelmoijille monipuolisia työkaluja sovelluskehitykseen. ASP.NET sisältää kaikki .NET:in työkalut ja vielä enemmän. Avoimen lähdekoodin ASP.NET on laadittu erityisesti Web-sovellusten tekemiseen. (Microsoft, 2019j)

ASP.NET-sovelluksia voidaan tehdä kahdella tapaa, käyttämällä .NET Frameworkia tai uudempaa .NET Corea. ASP.NET:istä puhuttaessa tarkoitetaan .NET Frameworkilla tehtyä sovellusta ja ASP.NET Core on taas uudempi versio ASP.NET:istä sovellusten luomiselle. Suurimmat erot näiden kahden välillä ovat suorituskyvyssä ja siinä, millä käyttöjärjestelmällä sovellus toimii. ASP.NET Framework toimii ainoastaan Windowsilla ja suorituskyky on hyvä, kun taas ASP.NET Core toimii useammilla käyttöjärjestelmillä ja suorituskyky on erinomainen. (Microsoft, 2018a; Microsoft, 2019b; Microsoft, 2019j)

2.3 **ASP.NET MVC**

MVC (Model-View-Controller) on arkkitehtuurityyli, jota voidaan käyttää hyödyksi ASP.NET-sovelluksissa. Välttämätöntä se ei ole. Suomeksi kyseistä tyyliä kutsutaan malli-näkymä-ohjain-arkkitehtuuriksi. ASP.NET MVC on ohjelmistokehys web-sovellusten rakentamiseen MVC-mallia käyttäen. Models-osio käsittää luokat (classes), joita käsitellään. View määrittää käyttäjänäkymän (User Interface = UI) eli sovelluksen ulkoasun. Controller määrittää sovelluksen toiminnallisuuden eli sieltä löytyy esimerkiksi metodeita, miten tietyn funktiokutsun tullessa toimitaan. Controller toimii eräänlaisena yhdistävänä tekijänä Model ja View -osioiden välillä. MVC-mallissa siis käsiteltävä tieto, käyttäjänäkymä ja sovelluksen toiminnallisuus ovat eroteltuna omiin paikkoihinsa, joka helpottaa koodin käsittelyä. (Galloway ym., 2014, ss. 1-3; ks. myös Koskimies & Mikkonen, 2005, s. 142)

Yksi ASP.NET MVC:n ominaisuuksia on koodin automaattinen luonti valmiin mallin pohjalta. Tätä kutsutaan nimellä scaffolding. Esimerkiksi uutta näkymää (View) luotaessa on mahdollista valita valmiita pohjia aiemmin luodun Model-luokan pohjalta. Voidaan luoda esimerkiksi lista tai muokkaussivu. Myös Controlleria luotaessa voidaan käyttää automaattista koodin luontia, jolloin Controlleriin tulee automaattisesti vaikkapa CRUD-toiminnot johonkin Model-luokkaan liittyen. (Galloway ym., 2014, ss. 80-81) CRUD-operaatioista kerrotaan tarkemmin luvussa 3.5

Näkymien sisällä käytetään Razor-syntaksia, joka on laadittu helpottamaan kehittäjien työtä. Syntaksin avulla on mahdollista kirjoittaa helposti ja nopeasti vaikkapa html-, C# ja JavaScript-kieltä sekaisin samalle sivulle. Vaihdettaessa html-kielestä C#-kieleen, tarvitsee vain kirjoittaa @ symboli koodin alkuun ja ohjelma osaa lukea sen C#-kieleksi. Esimerkiksi foreach kirjoitettaisiin View-sivulle @foreach. (Galloway ym., 2014, ss. 63-64)

3 WEB SERVICET JA OHJELMOINTIRAJAPINNAT

Web Servicellä tarkoitetaan yksinkertaisimmillaan resurssia, joka on saatavilla internetin kautta. Web Service on palvelu, jonka päätoimena on toimia välittäjänä kahden yhteensopivan tietokoneen välillä ja tämä kommunikaatio tapahtuu tietoverkon kautta. Web Service toimii erilaisilla alustoilla ja voidaan kirjoittaa monella eri ohjelmointikielellä. Sen kehittämiseen voidaan sanoa olevan kaksi pääasiallista lähestymistapaa: SOAP-protokollan tai REST-arkkitehtuurin käyttäminen. Näistä kahdesta tarkemmin tämän kappaleen alaluvuissa. (Balani & Rajeev, 2009, ss. 33 & 193; Bush, 2019)

Web Service -termiä käytetiin alun perin palvelukeskeisen arkkitehtuurin (Service Oriented Architecture) eli SOA:n yhteydessä. Palvelukeskeisen arkkitehtuurin ytimenä on jakaa sovelluksen toiminnot itsenäisiksi osiksi ja asettaa ne saataville palveluina internetin välityksellä. Web Servicet käyttivät aiemmin eniten SOAP-protokollaa, joka välittää XML-muodossa olevia tietoja verkon ylitse HTTP:n avulla. Nykyään yhä yleisempi tapa on käyttää REST-arkkitehtuuriin perustuvaa rajapintaa sovellusten välisessä tiedonsiirrossa. (Freeman, 2019; ks. myös Bush, 2019)

API eli ohjelmointirajapinta on liittymä, jonka avulla voidaan käyttää jo olemassa olevan, toisen ohjelman, tietoja ja toiminnallisuuksia. Ohjelmointirajapinnan arkkitehtuuria tai protokollia ei ole tarkasti määritelty. Verrattuna Web Serviceen, API on kahden sovelluksen välinen tiedonsiirtokanava. Voidaan sanoa, että jokainen Web Service on API, mutta jokainen API ei ole Web Service. (Bush, 2019)

3.1 SOAP Web Service

Web Servicet suunniteltiin alun perin käyttämään SOAP-protokollaa (Simple Object Access Protocol). SOAP-protokollaa käytettäessä tietoja lähetetään verkon yli useimmiten HTTP-pyyntöjen avulla XML-muodossa. SOAP-protokolla lähettää tiedot ns. SOAP-kirjekuoren (SOAP Envelope) muodossa ja tähän kirjekuoreen on suljettu kaikki siirrettävä tieto. Kirjekuoren sisältö koostuu otsikosta (header) ja sisällöstä (body). Otsikkotason tiedot ovat vapaaehtoisia, niissä voidaan kertoa esim. tapahtuman laatu tai turvallisuuteen liittyviä tietoja. Varsinainen pääsisältö kerrotaan body-osiossa. (Balani & Rajeev, 2009, s. 28; Freeman, 2019)

SOAP Web Servicessä palveluntarjoaja julkaisee palvelun tiedot WSDL-tiedostoa käyttäen. WSDL (Web Services Description language) on XML-pohjainen kieli, jolla kuvataan Web Services-palveluita. Siinä kerrotaan, millä tyylillä viestejä lähetetään palvelimen ja asiakkaan välillä.
(Balani & Rajeev, 2009, ss. 28-35)

SOAP Web Servicessä voidaan käyttää kahta eri viestintätyyliä: etätoimintosarjakutsumuotoa (Remote Procedure Call eli RPC) tai sanomaperusteista (Document) muotoa. Käytettävä viestintätyyli on määritelty WSDL-tiedostossa. (Balani & Rajeev, 2009, s. 35)

3.2 **RESTful Web Service**

REST (Representational State Transfer) on ohjelmistoarkkitehtuuri, joka on yksi vaihtoehto aiemmin esitellylle SOAP-protokollalle. Alkuperäisen määritelmän REST arkkitehtuurille laati Roy Fielding väitöskirjassaan vuonna 2000. REST-arkkitehtuuri on saanut paljon suosiota yksinkertaisuutensa vuoksi, verraten SOAP-perusteisiin Web Serviceihin. REST antaa ehtoja rajapinnalle, mutta ei määrittele formaattia tiedon siirtämiseen. Arkkitehtuurin perusajatuksena on toimia tiedon välittäjänä erilaisten sovellusten välillä käyttäen tilatonta HTTP-protokollaa. Tilattomuudella (stateless) tarkoitetaan, että jokainen palvelimelle lähetettävä pyyntö on oma kokonaisuutensa eli pyynnöt eivät tiedä mitään toisista pyynnöistä. Palvelimelle ei siis tallenneta mitään, mutta palvelimen vastauksia voidaan tallentaa asiakkaan puolelle. (Kanjilal, 2013, ss. 24-28; Mikkonen, 2017)

REST käyttää asiakas-palvelin-arkkitehtuuria (client-server), jonka toimii istuntojen eli sessioiden avulla. Istunto alkaa, kun asiakas ottaa yhteyden palvelimeen ja päättyy kun pyydetyt tehtävät on hoidettu. Asiakas-palvelin-arkkitehtuurin mukaisesti käyttäjä lähettää pyynnön, jonka palvelin sitten käsittelee. Palvelin huolehtii pyyntöön liittyvän resurssin mahdollisista virheistä. Resurssin hallinta (palvelin) on siten kapseloitu eikä käyttäjän tarvitse huolehtia teknisistä ongelmista. (Kanjilal, 2013, ss. 24-26; Koskimies & Mikkonen, 2005, ss. 136-138)

Tilattomuuden ja asiakas-palvelin-arkkitehtuurin lisäksi REST-sovellusta määrittävät seuraavat: välimuisti (cacheable), yhtenäinen rajapinta (uniform interface), resurssien hallinta (resource management) ja ladattava koodi (code on demand). Välimuistin osalta edellytetään, että palvelin kertoo vastauksessaan asiakkaalle, saako vastausta tallentaa välimuistiin vai ei. Tämä lisää mm. tehokkuutta, sillä asiakkaan ei välttämättä tarvitse lähettää samaa pyyntöä uudelleen, koska vastaus on jo tallessa välimuistissa. (Kanjilal, 2013, ss. 24-28)

Yhtenäisellä rajapinnalla tarkoitetaan sitä, että sekä käyttäjä että palvelin käyttävät tiettyjä toimintoja, tässä tapauksessa HTTP-metodeita, joista yleisimmät ovat GET, POST, PUT, ja DELETE. HTTP-metodeita voidaan verrata CRUD-operaatioihin, joita käsitellään tarkemmin luvussa 3.5. Resurssien hallinta tarkoittaa, että jokaisella resurssilla on yksilöity URI-osoitteensa (Uniform Resource Identifier) ja resurssi voi olla missä tahansa digitaalisessa muodossa. Ladattava koodi ei ole pakollinen osa REST-sovellusta. Sillä tarkoitetaan, että sovelluksen koodia voidaan muokata tai laajentaa lataamalla koodia paikallisesti palvelimelta. (Kanjilal, 2013, ss. 24-28)

REST-arkkitehtuuriin pohjautuvista palveluista käytetään nimeä RESTful (Web) Service tai RESTful Web API. Näillä palveluilla ei ole tarkkaa laadintakriteeriä (vrt. SOAP), ne koostuvat resurssikokoelmista ja käyttö onnistuu millä tahansa alustalla tai ohjelmointikielellä. (Kanjilal, 2013, ss. 40-41)

3.3 **SOAP vai REST?**

Edellä käsiteltiin yleisesti SOAP:in ja REST:in ominaisuuksia. Seuraavaksi vertaillaan kevyesti näiden eroja ja sitä, kumpi tyyli sopii tämän työn tarkoitukseen. Jos tarkoituksena on siirtää ja vastaanottaa yksinkertaista tietoa esim. XML-muodossa, niin silloin RESTful Service on hyvä valinta. Jos asiakkaan ja palveluntarjoajan välillä on monimutkaista siirrettävää tietoa tai spesifikaatioiden käyttöä niin silloin SOAP-protokollan käyttö on perusteltua. (Balani & Rajeev, 2009, ss. 33-34)

SOAP vaatii tiukkojen sääntöjen noudattamista ja sillä on tarkasti määritellyt kriteerit. REST-arkkitehtuurityyli taas antaa kehittäjälle vapaammat kädet toteuttaa Web Service tai API. (Bush, 2019) (Balani & Rajeev, 2009, ss. 33-35) Vaikuttaa siltä, että REST sopii ensikertalaiselle SOAP-protokollaa paremmin rajapinnan luomiseen. Yksi tapa laatia REST-rajapinta on käyttää Web API:a, jota käsitellään tarkemmin seuraavassa kappaleessa 3.4 (Microsoft, 2019a). Web API:a pidetään hyvänä vaihtoehtona .NET-ympäristön REST-rajapinnaksi (Kanjilal, 2013, s. 208).

3.4 **Web API ASP.NET-sovelluksessa**

Vuonna 2011 julkaistun ASP.NET MVC 4 -mallin mukana tuli uusi ominaisuus helpottamaan kehittäjiä ohjelmointirajapinnan tekoon: ASP.NET Web API. Tämä kyseinen Web API on ohjelmistokehys, joka on suunniteltu erityisesti HTTP-palveluita varten ja se toimii osittain MVC-mallin mukaisesti. ASP.NET Web API mahdollistaa internetsivustojen sekä niihin liittyvien palveluiden kehittämisen yhdessä ja samassa projektissa, rinnakkain. Vaikka Web API onkin julkaistu osana ASP.NET MVC:tä, sitä voidaan käyttää myös erillään ASP.NET:istä. Käyttö onnistuu minkä tahansa .NET-sovelluksen kanssa. Web API noudattaa RESTful arkkitehtuuria ja tämän vuoksi siitä käytetään myös nimitystä REST API. (Galloway ym., 2014, ss. 7-9 & 334; ks. myös Microsoft, 2019a)

Web API:lla voidaan palauttaa tietoa JSON tai XML-muodossa (Kanjilal, 2013, s. 158). JSON (JavaScript Object Notation) on kevyt tekstimuotoinen standardi tiedon välitykseen ja XML (Extensible Markup Language) tekstiperustainen merkintäkieli, jolla on hieman enemmän muotosääntöjä kuin JSON:illa (Kanjilal, 2013, ss. 121-122).

Web API käyttää ohjainta (controller), mutta hieman eri tavalla kuin MVC-malli. MVC-mallin mukainen ohjain periytyy Controller-luokasta ja sille

tulevat pyynnöt ohjataan toimintojen nimien mukaan, kun taas Web API periytyy ApiController-luokasta ja sille tulevat pyynnöt ohjataan HTTP-metodien mukaan (avataan seuraavassa kappaleessa 3.5). Pyynnön tullessa ohjaimelle, Web API lähettää vastauksen saman tien yleensä luokka-objektina (model object) tai HTTP-vastauksena. Ei välitetä näkymiä, vaan ainoastaan tietoja. HTTP-vastauksella tarkoitetaan esimerkiksi koodia 201 ”Created”, joka annetaan, kun uusi objekti on luotu POST-pyynnöllä. Kun taas MVC-mallin ohjaimen tapauksessa vastaukseen liitetään model ja view-osiot eli vastauksena saadaan tiedon lisäksi myös näkymiä. (Galloway ym., 2014, ss. 334-336 & 340)

Koodissa 1 nähdään Visual Studion generoima valmis API Controller C#-ohjelmointikielellä. Controllerilla on HTTP-metodien mukaiset toiminnot GET, POST, PUT, ja DELETE. Kun API Controller saa GET-pyynnön osoitteeseen api/values/1, se palauttaa esimerkkikuvan tapauksessa id:n 1 arvon, joka tässä on ”value1”.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace WebApplication.Controllers
{
    [Authorize]
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        public string Get(int id)
        {
            return "value";
        }

        // POST api/values
        public void Post([FromBody]string value)
        {
        }

        // PUT api/values/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE api/values/5
        public void Delete(int id)
        {
        }
    }
}
```

Koodi 1. Esimerkki Web API Controllerista.

3.5 HTTP-metodit ja CRUD-operaatiot

Tietokantaa voidaan käyttää ohjelmointirajapinnan kautta HTTP-metodien avulla. Yleisimmät HTTP-metodit ovat GET, POST, PUT, ja DELETE. Kullekin HTTP-metodille voidaan löytää vastine CRUD-operaatioista. (Kanjilal, 2013, ss. 40-41) CRUD-operaatioita käytetään yleisesti monessa paikassa, muun muassa ohjelmointirajapinnoissa ja tietokannoissa. CRUD-operaatioilla tarkoitetaan CREATE, READ, UPDATE, ja DELETE toimintoja eli suomeksi tietojen luonti, lukeminen, päivittäminen ja poistaminen. (Kanjilal, 2013, ss. 40-41; Salminen, 2018) Yleisimpiä HTTP-metodeja vastaavat CRUD-operaatiot on esitetty taulukossa 1.

Taulukko 1. CRUD-operaatioita vastaavat HTTP-metodit (Kanjilal, 2013, s. 41).

CRUD-operaatio	**HTTP-metodi**	**Suomeksi**
Read	GET	Tietojen lukeminen
Create	POST	Uuden tiedon luominen
Update	PUT	Tietojen päivittäminen
Delete	DELETE	Tietojen poistaminen

HTTP-metodien toimivuutta voidaan testata esimerkiksi Postman-ohjelman avulla. Ohjelmalla pystyy lähettämään haluamansa pyynnön (HTTP-metodit) valitsemaansa osoitteeseen ja katsomaan vastauksen. Pyyntöjä on mahdollista tallentaa ja lähetettävän tiedon formaatin voi asettaa haluamakseen. (Postman Inc, 2019) Voidaan lähettää esimerkiksi GET-pyyntö API:n osoitteeseen ja saada vastaukseksi listaus tietokannan tiedoista JSON-muodossa. Koodin 1 (sivu 8) tapauksessa Postman-ohjelman vastaus GET-pyyntöön olisi seuraavanlainen: { "value1", "value2" }.

3.6 Ohjelmointirajapinnan käyttö JavaScript-teknologioiden avulla

JavaScript on ohjelmointikieli ja sitä voidaan käyttää muun muassa kommunikointiin ohjelmointirajapinnan kanssa (Mozilla, n.d.-b). Ajax (Asynchronous JavaScript and XML) on nimitys joukolle web-teknologioita (joihin myös JavaScript kuuluu), jotka on koottu niin sanottuun Ajax-malliin. Sen avulla voidaan toteuttaa toisistaan riippumattomia toimintoja ja ladata tietoja päivittämättä sivua. Ajaxia käytetään nykyään lähes kaikessa internet-kehityksessä. (Galloway ym., 2014, ss. 213-214; Mozilla, n.d.-a)

JQuery on yksi suosituimpia JavaScript-kirjastoja ja siihen on saatavilla useita lisäosia. Microsoft tukee jQuerya ja jokaisesta luodusta ASP.NET MVC -projektista löytyy Scripts-kansio, jossa on jo valmiina jQuery-tiedostoja. JQuery on kevyt, mutta tehokas ja toimii yleisimmin käytetyillä internetselaimilla. (Galloway ym., 2014, s. 214) JQuery.DataTables on erikseen

asennettava lisäosa erilaisten taulukoiden hallintaan. Työkalussa on valmiina muun muassa sivunumerointi ja hakukenttä. (SpryMedia Ltd, n.d.)

Axios on JavaScript-kirjasto HTTP-pyyntöjen käsittelyyn ja sitä voidaan käyttää myös ohjelmointirajapinnan kanssa kommunikointiin eli API-kutsuihin. Työkalun avulla on helppo käsitellä JSON-muotoista tietoa ja siinä on myös ominaisuuksia, jotka auttavat suojautumaan Cross-Site Request Forgery -hyökkäyksiltä. Cross-Site Request Forgery -hyökkäyksiä käsitellään tarkemmin luvussa 5.3.2. (Zabriskie, n.d.)

4 TIETOKANNAT

Tietokannoilla tarkoitetaan yleiskielessä paikkaa, johon tietoa voi tallentaa. Tietokannoissa tiedot on tallennettu tauluihin ja tauluilla voi olla yhteyksiä (relaatiot) toisiinsa. Yleensä tietokannasta voidaan myös etsiä tietoa. Tietokantojen päätehtävänä on tiedon tallentaminen ja tallennetun tiedon hakeminen. Tietokanta luodaan tarvetta varten ja se on kokoelma tietyn aihepiirin tietoja. Tietokantoihin tallentamiseen ja sieltä hakemiseen liittyy muutamia haasteita, joita ovat tietoturva, suorituskyky, eheys ja pysyvyys. (Salminen, 2018)

Tietoturvaan liittyen on pohdittava mm. kuka tietoon pääsee käsiksi ja minne tiedot on fyysisesti tallennettu. Palvelimen suorituskyvyn on oltava tarpeeksi tehokas. Suorituskykyongelmia voi tulla, jos tietoa on miljoonia rivejä. Miten varmistetaan tehokas tiedon tallentaminen ja hakeminen? Eheydellä tarkoitetaan sitä, että tietokantaa rakennettaessa on muistettava noudattaa kyseiselle tietokannalle määriteltyjä sääntöjä. On esimerkiksi voitu määritellä, että jonkin taulun tietyille käsitteille annetaan aina numeroarvoja. Emme myöskään voi tallentaa tauluun mitään äkkiseltään keksittyä tietoa, sillä sille ei ole paikkaa. Yhteen tauluun tallennetaan aina tietyt tiedot. Sääntöjä noudattamalla tietokannan eheys säilyy. Pysyvyydellä tarkoitetaan tallennetun tiedon ja muokkausten säilymistä virhetilanteissa, jos vaikkapa sähkökatko katkaisee internetyhteyden. (Salminen, 2018)

Tietokantoja on perinteisesti kahta tyyppiä: relaatiotietokannat ja ei-relaationaaliset tietokannat. Relaatiotietokannat käyttävät rakenteista SQL-kyselykieltä (Structured Query Languege) tiedon käsittelyyn. Relaationaaliset SQL-tietokannat ovat hyvin suosittuja ja sopivat monipuoliseen tiedon käsittelyyn. Tosin niihin tiedon tallentaminen on tehtävä aina ennalta määritettyjen normien mukaan. Ei-relaationaalisia tietokantoja kutsutaan myös NoSQL (Not Only SQL)-tietokannoiksi. NoSQL-tietokannoissa tiedon tallentamiseen ei ole tiettyjä sääntöjä, vaan sinne voidaan vapaamuotoisesti tallentaa tekstin lisäksi vaikkapa dokumentteja. (Smallcombe, 2019) Nykyään markkinoille on tullut näiden kahden perinteisen tietokantatyypin lisäksi NewSQL-tietokannat, joka yhdistää relaatiotietokantojen monipuoliset ominaisuudet NoSQL-tietokantojen laajennettavuuteen (Simsek, 2019).

4.1 Tietokannan hallintajärjestelmät

Tietokannan hallintajärjestelmä (DBMS = Database Management System) on ohjelma, joka auttaa käyttäjää hallitsemaan tietokantaa. Hallintajärjestelmässä esimerkiksi tietokantojen luominen ja ylläpito on tehty helpoiksi. (Salminen, 2018) Tunnettuja tietokannan hallintajärjestelmiä ovat esimerkiksi Microsoftin SQL Server, Oracle, IBM:n DB2 ja MySQL. Seuraavassa

kappaleessa kerrotaan hieman lisää Microsoftin SQL Serveristä, jota käytettiin tämän työn käytännön osassa.

SQL Server on Microsoftin kehittämä operationaalisen tietokannan hallintajärjestelmä (ODBMS eli Operational Database Management System). Se on saatavilla Windowsille ja Macille. (Microsoft, 2019g) Microsoft SQL Server Management Studio (SSMS) on ohjelma SQL ympäristöjen hallintaan ja sen avulla onnistuu mm. tietokantojen luominen, muokkaus ja seuranta. SSMS:llä voidaan hallita useita eri ympäristöjä (muitakin kuin Microsoftin) eli sitä käytettäessä tietokannan ei tarvitse välttämättä sijaita SQL Serverillä. (Microsoft, 2019d)

4.2 **Entity Framework Code First**

Entity Framework on .NET-työkalu, jonka avulla hoidetaan sovelluksen olioiden ja tietokannan välinen kommunikaatio. Tätä kutsutaan olio-relaatiomallinnukseksi (Object-Relational Mapper = O/RM). (Microsoft, 2016c) Entity Frameworkin käyttöön tietokantojen suhteen on kaksi lähestymistapaa, riippuen onko tietokanta jo olemassa vai ei. Jos halutaan luoda kokonaan uusi tietokanta, käytetään joko Code First tai Model First -tyyliä. Code First tarkoittaa, että Entity Framework luokkamalli (model) laaditaan kehittäjän kirjoittaman ohjelmointikoodin pohjalta. Toinen tapa, Model First, käyttää Entity Framework Designeria, jossa piirretään taulut perinteiseen tietokantatyyliin. Kun taas tietokanta on jo olemassa, niin silloin voidaan käyttää jo edellä mainittua Code First -tyyliä tai sitten Database First eli tietokanta ensin -tyyliä. Database First -lähestymistavassa luodaan yhteys jo olemassa olevaan tietokantaan Entity Framework Designerin avulla. (Microsoft, 2018b) Tässä työssä tullaan käyttämään Code First -lähestymistapaa uuden tietokannan luomiseen, sillä työn tekijä kokee koodin kirjoittamisen itselleen sopivammaksi kuin graafisen työkalun käytön. Työn käytännön osassa tullaan luomaan tietokantataulut Code First -tyylillä Microsoftin SQL Serveriin.

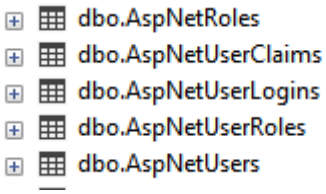
Code First -tyyliä käytettäessä luodaan sovellukseen ensin luokkamalli (model class) ja annetaan sille tarvittavat ominaisuudet. Tässä kohtaa on huomioitava, että NuGet Package Manager, Visual Studion lisäosa, on asennettuna. NuGet Package Managerin avulla asennetaan EntityFramework -paketti, jotta voidaan käyttää työkalun kirjastoja. Seuraavaksi lisätään tehty luokka DataContext:iin DbSet-komennon avulla. Tämän jälkeen kaikki on valmista tietokannan luomista varten. Lopuksi otetaan Package Manager Consolen (Visual Studion toiminnallisuus) kautta migraatiot käyttöön (komento *Enable-Migrations*) ja niiden avulla ajetaan tiedot tietokantaan (komennot *Add-Migration MigraationNimi* ja *Update-Database*). Yksinkertaisuudessaan Code First on sitä, että kirjoitetaan luokat esimerkiksi C#-ohjelmointikielellä ja Entity Framework luo niiden perusteella taulut tietokantaan. (Microsoft, 2016a; Microsoft, 2016b)

5 ASP.NET-SOVELLUKSEN TIETOTURVA

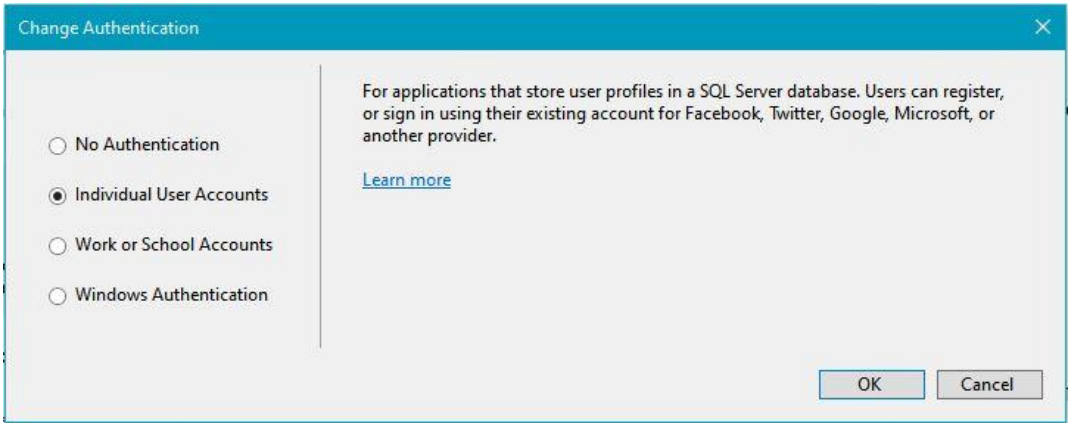
Tietoturvallisuus on tärkeä osa web-sovelluksia. Salaisiin tietoihin ei haluta kenen tahansa pääsevän käsiksi. Kappaleessa käydään läpi turvallisuuteen liittyviä asioita erityisesti ASP.NET-sovelluksen näkökulmasta.

5.1 ASP.NET Identity

ASP.NET Identity on ohjelmistokehys sovelluksen käyttäjien hallintaan ja tunnistamiseen. Sitä voidaan käyttää kaikissa ASP.NET-tuoteperheen ohjelmistokehyksissä eli esimerkiksi ASP.NET MVC:ssä tai Web API:ssa. ASP.NET Identity mahdollistaa käyttäjien rekisteröitymisen sosiaalisen median tilien (esim. Facebook) kautta tai sitten perinteisen kirjautumislomakkeen avulla. Kehittäjän on helppo hallita käyttäjien tietoja ja lisätä haluamiaan ominaisuuksia, esimerkiksi vaatia käyttäjän osoitetta rekisteröityessä. Oletuksena ASP.NET Identity lisää käyttäjien tiedot Entity Framework Code Firstin avulla tietokantaan, luomalla sinne viisi taulua (Kuva 1). Muitakin lähestymistapoja voidaan käyttää, kuten jo olemassa olevaa tietokantaa. Erilaisten käyttäjäryhmien teko on myös mahdollista, esimerkiksi Administrator eli pääkäyttäjäryhmä ja User eli peruskäyttäjäryhmä. Pääkäyttäjälleryhmälle voidaan asettaa suuremmat oikeudet sovelluksen käyttöön ja peruskäyttäjä voidaan rajata pois tietyistä toiminnallisuuksista. Käyttäjäryhmät tallentuvat AspNetRoles-tauluun ja tietyt käyttäjät linkitetään tiettyihin ryhmiin taulussa AspNetUserRoles. (Galloway ym., 2014, s. 12; Microsoft, 2019e)



Kuva 1. ASP.NET Identityyn liittyvät tietokantataulut.



Kuva 2. Käyttäjien todennustyypin valinta.

Käyttäjän todennuksella (user authentication) tarkoitetaan sitä, onko käyttäjä se kuka hän väittää olevansa. Todennus suoritetaan esimerkiksi käyttäjänimen ja salasanan avulla. (Galloway ym., 2014, s. 162) ASP.NET Identity voidaan ottaa käyttöön projektia luodessa, valitsemalla sopiva käyttäjien todennustyyppi (Kuva 2). Tällöin projektiin tuodaan automaattisesti kolme kirjastoa käyttäjien todennuksen hallintaan, jotka esitetään taulukossa 2. (Microsoft, 2019e)

Taulukko 2. ASP.NET Identityyn liittyvät kirjastot ja niiden päätehtävät.

Kirjaston nimi	**Tehtävä**
Microsoft.AspNet.Identity.Core	ASP.NET Identityn perusta.
Microsoft.AspNet.Identity.Entity-Framework	Tietokannan kanssa kommunikointi. Esimerkiksi käyttäjän tietojen tallennus tietokantaan.
Microsoft.AspNet.Identity.OWIN	Mahdollistaa OWIN-perusteisen todennuksen. Tarvitaan esimerkiksi sisäänkirjautuessa sovellukseen, kun luodaan eväste OWIN todennuksen avulla.

Todennustyyppejä (Kuva 2) on neljä erilaista: ei todennusta, yksittäiset käyttäjätilit, organisaatiotilit ja Windows-todennus. Valitessa yksittäiset käyttäjätilit rekisteröinti onnistuu lomakkeella tai vaihtoehtoisesti jo olemassa olevan sosiaalisen median käyttäjätilin avulla. Organisaatiotilit-vaihtoehto sopii yrityksille, joissa käyttäjien todennus tehdään Active Directoryn tai Officen kautta. Windows-todennusta käytetään yrityksen sisäisissä palveluissa.

5.2 **Authorize-attribuutti**

Käyttäjän valtuutuksella (authorization) tarkoitetaan tietoa siitä, onko kyseisellä käyttäjällä lupa johonkin toimintoon sivustolla. (Microsoft, 2012) ASP.NET MVC:n ominaisuuksiin kuuluva AuthorizeAttribute on helpoin tapa lähteä liikkeelle sovelluksen suojaamisessa. Tällöin sallitaan käyttö vain rekisteröityneille käyttäjille. Authorize-attribuuttia voidaan käyttää paikallisesti, esimerkiksi yhden Controllerin sisällä, tai sitten kattamaan koko sovelluksen. Koko sovellukseen vaikuttava Authorize-suodatin lisätään App_Start -kansion FilterConfig-tiedostoon. Tämä on kannattavaa silloin, kun lähes koko sovellus on vain rekisteröityneille käyttäjille. Rekisteröitymättömille käyttäjille sallitut, muutamat sivut, voidaan yksittäin sallia anonyymien käyttöön lisäämällä attribuutti AllowAnonymous. (Galloway ym., 2014, ss. 162 & 170-171)

Authorize-attribuutilla voidaan sallia toiminnot tietyille käyttäjille (users) tai/ja käyttäjäryhmille (roles). Käyttäjäryhmille rajaamista pidetään yksittäistä käyttäjää parempana vaihtoehtona, sillä käyttäjänimet voivat

muuttua. Uusi käyttäjä on myös helppo lisätä tiettyyn käyttäjäryhmään, eikä koodiin tarvitse lisätä uuden käyttäjän nimeä manuaalisesti. (Galloway ym., 2014, ss. 172-173) Authorize-attribuuttia kutsutaan myös valtuutussuodattimeksi (authorization filter). (Microsoft, 2012)

Authorize-attribuuttia voidaan käyttää myös Web API:n suojaamisessa. Attribuutin avulla voidaan sallia API:n toimintojen käyttö vain tietyille henkilöille tai ryhmille. Käytettäessä Authorize-attribuuttia Web API:ssa on huomioitava, että käyttää nimiavaruutta System.Web.Http, sillä System.Web.Mvc ei ole yhteensopiva API:n kanssa. (Microsoft, 2012)

5.3 **Tavallisimmat uhat ja niiltä suojautuminen**

Seuraavissa luvuissa käsitellään tavallisimpia uhkia, joita verkkosivuille kohdistuu ja käydään lyhyesti läpi, miten niiltä voi suojautua.

5.3.1 Cross-Site Scripting (XSS)

Cross-Site Scripting tarkoittaa sitä, että ulkopuolinen henkilö voi syöttää verkkosivustolle omaa koodiaan esimerkiksi tekstilaatikon tai hakukentän kautta. Tästä käytetään myös nimitystä XSS ja se on hyvin yleinen tapa hyökätä verkkosivustolle. Hyökkääjä voi käyttää passiivista tai aktiivista injektiota. Passiivinen injektio voi tapahtua esimerkiksi tekstilaatikon kautta, jolloin hyökkääjä lähettää tekstilaatikon kautta koodia, joka tallentuu sivuston tietokantaan eikä kukaan välttämättä huomaa mitään. Aktiivisessa injektiossa tieto ei tallennu tietokantaan, vaan näkyy vain sivustolla. Esimerkiksi suojaamattomaan hakukenttään voidaan syöttää koodia, jolla luodaan kirjautumisikkunalta näyttävä kohta sivustolle ja tähän tietoja syöttämällä hyökkääjä saa haluamansa. (Galloway ym., 2014, ss. 183-187)

Cross-Site Scriptingiltä voidaan suurimmilta osin suojautua muuntamalla kaikki koodikielelle. Tämä tarkoittaa, että käyttäjän mahdollisesti syöttämät merkit muunnetaan tekstiksi. Eli jos käyttäjä voi syöttää tietoja johonkin kenttään niin ennen sivustolle lähetystä koodataan kentän sisältö, ettei se lähde sellaisenaan eteenpäin. MVC-mallia käytettäessä koodikielelle muuntaminen onnistuu lisäämällä View-näkymään Html.Encode tai Html.AttributeEncode -attribuutit. Jos käytössä on Razor-syntaksin mukaiset ilmaisut niin sitten lisätoimia ei tarvita, sillä Razor-syntaksi huolehtii koodauksesta. (Galloway ym., 2014, ss. 187-188)

5.3.2 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery voi olla XSS-hyökkäystä voimakkaampi. Tähän liittyy myös termi confused deputy, jolla tarkoitetaan ymmällään olevaa internetselainta. CSRF-hyökkäyksessä huijataan internetselainta suorittamaan väärä pyyntö väärään osoitteeseen, jolloin selain menee sekaisin. Esimerkiksi hyökkääjä voi lähettää keskustelupalstalle linkin kuva-tagien

sisällä ja tämä linkki johtaa vaikkapa sivulle, joka tekee pankkisiirron hyökkääjän tilille. Jos keskustelupalstan käyttäjä on äskettäin ollut sisäänkirjautuneena kyseisen pankin sivulla ja erehtyy klikkaamaan linkkiä, hänen selaimensa tekee palvelimelle GET-pyynnön, joka hakee pankin sisäänkirjautumistiedot sisältävän evästeen ja linkin kohdesivusto voi täten tehdä pankkisiirron käyttäjän tililtä hyökkääjän tilille. (Galloway ym., 2014, ss. 193-196)

Enimmiltä CSRF-hyökkäyksiltä suojautumaan auttaa ASP.NET MVC:n Token Verification, joka vapaasti suomennettuna tarkoittaa symbolista tarkistusta. Tämä tapahtuu lisäämällä lähetettäviin lomakkeisiin piilotettu kenttä, johon luodaan symbolinen koodi. Tätä koodia verrataan käyttäjän koneelle tallennetun evästeen koodiin ja jos se täsmäävät niin tietojen lähettäminen sallitaan. (Galloway ym., 2014, s. 196)

5.3.3 Evästeiden varastaminen (Cookie-Stealing)

Monet verkkosivustot käyttävät evästeitä ja niihin tallennetaan esimerkiksi käyttäjän sisäänkirjautumistiedot tai selaushistoria. Evästeitä on kahdenlaisia, istunnon eli session ajan selaimessa tallessa pidettäviä tai pysyvästi tietokoneelle tallennettuja. Session päättyessä kyseiset evästeet unohdetaan. Pysyvät evästeet taasen auttavat seuraavalla vierailukerralla internetsivua muistamaan käyttäjän. XSS-haavoittuvuudet mahdollistavat evästeiden varastamisen eli hyökkääjän tarvitsee vain kirjoittaa tarvittava koodinpätkä ja hän voi saada varomattoman käyttäjän evästeen itselleen ja esiintyä hänenä. (Galloway ym., 2014, ss. 197-199)

Useimmissa tapauksissa evästevarkauksilta voidaan suojautua HttpOnly-merkinnällä, joka lisätään sovelluksen sisäiseen Web.config-tiedostoon. Tämä mitätöi evästeen, jos jokin muu kuin palvelin yrittää tehdä sille jotakin. (Galloway ym., 2014, s. 199)

5.3.4 Over-Posting

Over-Postingilla tarkoitetaan sitä, että hyökkääjä lähettää sivustolle tietoja, joita hänen ei ole sallittu lähetettävän. Tämä tapahtuu silloin, kun sivuston käyttäjälle on vahingossa paljastettu tietoja, joita hänen ei kuulu tietää. Sivustolla voi olla esimerkiksi luokka nimeltä Review (Koodi 2) ja lomake, jolla käyttäjä voi lähettää tiedot Name ja Comment-ominaisuuksiin. Selaimen kehittäjätyökaluja käyttämällä hyökkääjä pääsee helposti käsiksi kaikkiin muihin kyseisen luokan ominaisuuksiin ja jopa Product-luokkaan, joka on viiteavaimena. Eli hyökkääjä voi lähettää tietoja näihin kaikkiin, vaikka sallittuna oli vain Name ja Comment -kohdat. (Galloway ym., 2014, s. 200)

```
public class Review {
 public int ReviewID { get; set; } // Primary key
 public int ProductID { get; set; } // Foreign key
```

```
 public Product Product { get; set; } // Foreign entity
 public string Name { get; set; }
 public string Comment { get; set; }
 public bool Approved { get; set; }
}
```

Koodi 2. Review-luokka. (Galloway ym., 2014, s. 200)

Over-postingilta suojautumiseen on useita vaihtoehtoja. Jos käytetään ASP.NET MVC:n automaattista View-näkymän luontia mallin pohjalta niin yksi vaihtoehto on lisätä luokalle Bind-attribuutti ja sisällyttää siihen ominaisuudet, jotka annetaan näkymän käyttöön. Esimerkkitapauksessa Review-luokan yläpuolelle tulisi koodi [Bind(Include="Name, Comment")]. Toinen vaihtoehto on luoda ViewModel, jolle annetaan käyttöön vain sallitut ominaisuudet. (Galloway ym., 2014, ss. 201-202) API:n suojaamiseen over-postingilta voidaan käyttää tiedonsiirto-objekteja (Microsoft, 2019c).

Tiedonsiirto-objektien (Data Transfer Objects) eli DTO:n päätehtävänä on eriyttää palvelinobjektit (domain objects) ja käyttäjälle palautettavat objektit toisistaan. Sitä siis käytetään tiedon siirtoon asiakkaan ja palvelimen välillä. Ohjelman funktioiden ei pitäisi koskaan vastaanottaa tai palauttaa palvelinobjektia. Ei ole tietoturvallista, jos käyttäjälle palautettava tieto haetaan suoraan tietokannasta. Joku voi hyvinkin päästä väliin ja lähettää omia tietojaan tietokantaan. Tähän väliin tarvitaan tiedonsiirto-objekti. (Microsoft, 2019c; Volosoft, 2019)

DTO:n avulla voidaan piilottaa halutut asiat eli rajata tiedot, joihin käyttäjä pääsee käsiksi. Otetaan esimerkki: Meillä on tietokannassa taulu ”Käyttäjät”, jossa kirjattuna käyttäjän nimi ja salasana. Käyttäjälle halutaan näyttää listaus muista käyttäjistä. Jos käyttäjien nimet haetaan suoraan tietokannasta, niin vaarana on se, että joku käyttää tätä polkua myös salasanojen hakemiseen. Ratkaisuna on tehdä uusi luokka ”KäyttäjätDTO”, jossa on tiedot vain käyttäjänimistä, jotka se hakee tietokannasta. Sitten listaus muista käyttäjistä haetaankin DTO:sta, eikä ole enää vaaraa, että salasanoihin päästään tämän kautta käsiksi. (Microsoft, 2019c; Volosoft, 2019)

Tiedonsiirto-objektien käyttö voidaan toteuttaa manuaalisesti koodia kirjoittamalla tai jonkin työkalun avulla. Tällainen työkalu on esimerkiksi AutoMapper. (Microsoft, 2019c) AutoMapper on kirjasto, jolla voidaan helposti mapata objekti toiseen objektiin. Objektin mappaus toiseksi objektiksi tarkoittaa sitä, että sisään tuleva objekti muunnetaan toisentyyppiseksi objektiksi ulostulossa. (Jimmy Bogard Revision, 2017) Otetaan esimerkiksi sisään tuleva objekti ”KäyttäjäDTO”, kun halutaan tallentaan uusi käyttäjä tietokantaan. ”KäyttäjäDTO” mapataan ”Käyttäjä”-objektiksi, joka sitten tallentuu tietokantaan. Näin sovellus ei anna käyttäjälle suoraa yhteyttä tietokannan ”Käyttäjä”-objektiin.

5.3.5 Uudelleenohjaus osoitteeseen (Open Redirection)

Aina kun jokin sivuston toiminto vastaanottaa parametrina internetosoitteen, on huolehdittava, ettei kolmas osapuoli pysty syöttämään osoitteeksi mitä tahansa. Esimerkiksi sisäänkirjautumistoiminnon ollessa haavoittuva, käyttäjän syötettyä tunnuksensa ja salasanansa saattaa hän ohjautua hyökkääjän sivulle, jonne myöskin erehtyy syöttämään tietonsa. Tämän jälkeen hyökkääjän sivusto ehkä ohjaa käyttäjän takaisin oikealle sivustolle eikä käyttäjä huomaa, että hänen tietonsa on siepattu. (Galloway ym., 2014, ss. 202-205)

Haitalliset uudelleenohjaukset voidaan estää lisäämällä toimintoon tarkistus internetosoitetta varten. Eli ennen ohjausta mihinkään osoitteeseen, ohjelma tarkistaa onko kyseinen internetosoite kehittäjän kyseiseen toimintoon tarkoittama vai ei. Jos internetosoite on sopiva, niin ohjataan käyttäjä kyseiseen osoitteeseen. Jos ei, niin voidaan vaikkapa antaa jokin virheilmoitus, josta käyttäjä huomaa, että joku yrittää ohjata häntä väärään paikkaan. (Galloway ym., 2014, ss. 205-207)

6 ASP.NET-SOVELLUKSEN TOTEUTUS KÄYTÄNNÖSSÄ

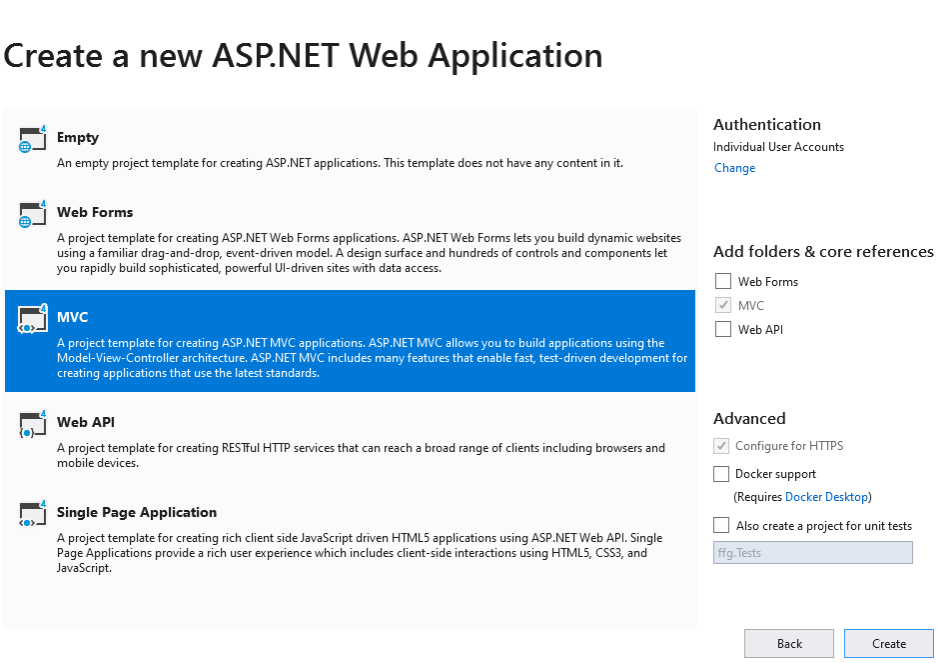
Opinnäytetyön tarkoituksena oli luoda toimeksiantajalle ASP.NET-pohjainen käyttöliittymä ja ohjelmointirajapinta, jonka kautta voitaisiin tietoturvallisesti käyttää Microsoftin SQL Serverissä sijaitsevaa tietokantaa. Ohjelmointikielen ja rajapinnan suhteen annettiin vapaat kädet. Visual Studio valikoitui kehitysympäristöksi sen tuttuuden ja hyvien ASP.NET-sovelluskehitysominaisuuksien vuoksi.

Käyttöliittymä toteutettiin ASP.NET MVC-mallia käyttäen, sillä se vaikutti selkeältä kokeilujen jälkeen. Alun perin ajatuksena oli ohjelmoida käyttöliittymä ASP.NET MVC Core:lla, mutta ajatuksesta luovuttiin, sillä koko ASP.NET oli työn tekijälle uusi asia. Päädyttiin ohjelmoimaan ASP.NET MVC -mallilla .NET Framework-ohjelmistokehystä käyttäen.

Ohjelmointikieleksi valikoitui C#, koska se oli jo ennestään tuttu tekijälle. Ohjelmointirajapinnaksi valittiin REST API toteutettuna Web API:n avulla, sillä tarkoitus on toimia .NET-ympäristössä ja välittää yksinkertaista tietoa JSON-muodossa. Monimutkaisia toimintoja ei ole tarpeen toteuttaa. Rajapinta luotiin, jotta tietokantaa voidaan sen avulla käyttää tarpeen mukaan yrityksen ulkopuoleltakin. Tulevaisuudessa rajapintaa voidaan hyödyntää työssä tehdyn ASP.NET-sovelluksen lisäksi myös vaikkapa mobiilisovelluksessa. Toimeksiantajan toiveesta työssä käytettiin Microsoftin SQL Serveriä tietokannan palvelimena. Tietokantaan luotiin esimerkin vuoksi helppolukuinen autotietokanta Entity Framework Code Firstin avulla.

6.1 Projektin luominen

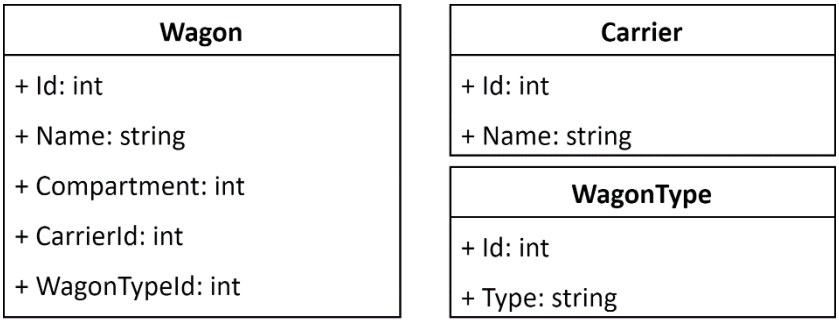
Sovelluksen toteuttaminen aloitettiin luomalla Visual Studiossa uusi projekti, jonka tyyppi oli ASP.NET Web Application (.NET Framework). Projektin luomisen yhteydessä valittiin käyttöön MVC-malli ja käyttäjien todennustyyppi ”Individual User Accounts” (Kuva 3). MVC-pohja loi projektiin automaattisesti Models, Views ja Controllers -kansiot. Käyttäjien todennus -pohja loi sovellukseen valmiit pohjat rekisteröitymistä ja sisäänkirjautumista varten. Käyttäjien todennus on osa ASP.NET Identityä.



Kuva 3. Projektin luominen

6.2 Models (luokat)

Luotiin Models-kansioon luokat, joita sovellus tarvitsee. Luokat olivat Wagon, WagonType ja Carrier (Kuva 4). Nämä luokat vastaavat tietokannan tauluja ja Entity Framework käyttää niitä myöhemmin, ensimmäisen migraation kohdalla, pohjana tietokantataulujen luomiselle Microsoftin SQL Serveriin. Wagon-luokan sisältö esitetään esimerkkinä koodissa 3. Luokan ominaisuuksille asetettiin myös joitakin vaatimuksia, esimerkiksi ”Required” tarkoittaa ominaisuuden olevan pakollinen.



Kuva 4. Luokat.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace UpeaTekniikkademo3.Models
{
```

```
    public class Wagon
    {
        public int Id { get; set; }

        [Required]
        public string Name { get; set; }

        [Range(1, 10)]
        public int Compartment { get; set; }

        // Foreign keys & navigation properties
        public int WagonTypeId { get; set; }
        public WagonType WagonType { get; set; }

        public int CarrierId { get; set; }
        public Carrier Carrier { get; set; }

    }
}
```

Koodi 3. Wagon-luokan koodi.

6.3 **Tiedonsiirto-objektit**

Luokkien lisäksi Models-kansioon luotiin tiedonsiirto-objektit eli DTO:t kullekin luokalle. Toinen tapa olisi ollut luoda niille oma kansio, esimerkiksi ”DTOS”, vaikkapa projektin pääkansion alle. DTO:t olivat muuten samankaltaiset kuin niiden alkuperäiset luokat, mutta nimen perässä pääte ”DTO”. Eli esimerkiksi WagonDTO, jonka sisältö havainnollistetaan koodissa 4. Jos luokassa oli viittauksia muihin luokkiin niin ne korvattiin kyseisten luokkien DTO-luokilla.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace UpeaTekniikkademo3.Models
{
    public class WagonDTO
    {
        public int Id { get; set; }

        [Required]
        public string Name { get; set; }

        [Range(1, 10)]
        public int Compartment { get; set; }

        // Foreign keys & navigation properties
        public int WagonTypeId { get; set; }
        public WagonTypeDTO WagonType { get; set; }

        public int CarrierId { get; set; }
        public CarrierDTO Carrier { get; set; }

    }
```

```
}
```

Koodi 4. WagonDTO-luokan koodi.

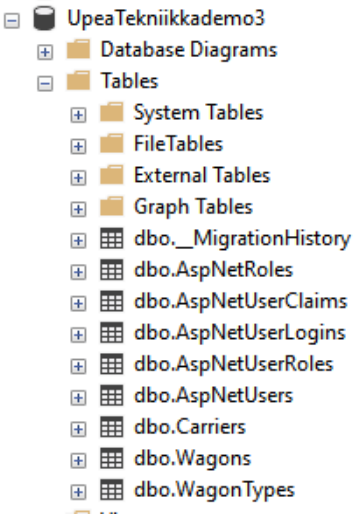
6.4 Tietokantayhteys ja tietokannan luominen

Sovelluksen tietokantayhteys asetettiin siten, että tietokanta luotiin työn tekijän koneella olevaan Microsoftin SQL Serveriin. Muokattiin connectionString-muuttujaa Web.config-tiedostossa. Tietokantayhteyden asettamisen jälkeen luotiin tietokanta ja sen taulut Entity Framework Code First -migraation avulla.

```
public DbSet<Wagon> Wagons { get; set; }
public DbSet<WagonType> WagonTypes { get; set; }
public DbSet<Carrier> Carriers { get; set; }
```

Koodi 5. Luokkien lisääminen DataContextiin.

Ennen ensimmäistä migraatiota lisättiin Models-kansioon luodut luokat DataContextiin DbSet-komentojen avulla (Koodi 5). Siten luokat saadaan mukaan migraatioon. Tämän jälkeen luotiin ensimmäinen migraatio nimeltään ”Initial” komennolla *Add-Migration Initial.* Sisäänajo tapahtui komennolla *Update-Database*. Tämän ensimmäisen migraation sisältönä oli Models-kansioon luodut luokat sekä käyttäjärekisteriin liittyvät taulut (mm. AspNetRoles). Käyttäjärekisterin taulut ovat osa ASP.NET Identityä. Kun migraatio oli ajettu, Entity Framework loi taulut tietokantaan. Asia voitiin todentaa avaamalla Microsoftin SQL Server Management Studio ja avaamalla sieltä projektin nimellä löytyvät taulut (Kuva 5).



Kuva 5. Code First -migraation luomat taulut.

Toisena luotiin migraatio nimeltä ”TestDataToDb”, jossa syötettiin SQL-kielellä tietoja tietokantaan testausta varten. Koodissa 6 nähdään esimerkkinä, kuinka yhden ajoneuvon tiedot syötettiin tietokantaan Code First -migraation avulla. Koodi oli lisättävä migraation Up()-toiminnon sisälle.

```
Sql("INSERT INTO Wagons (Name, Compartment, WagonTypeId, Car-
rierId) VALUES ('ABC-123', 5, 1, 1)");
```

Koodi 6. Tietojen syöttö migraatioon SQL-kielellä.

6.5 Controllers (ohjaimet)

Controlleria luotaessa luodaan myös automaattisesti samannimiset kansiot Views-kansion alle (ilman Controller-päätettä). Sovellukseen luotiin oletuksena jo projektin luomisvaiheessa kolme Controlleria, jotka olivat HomeController, AccountController ja ManageController. Controllerit luotiin Controllers-kansioon ja kutakin muokattiin sovelluksen tarpeisiin sopiviksi. Views-kansiosta löytyy vastineet näille Controllereilla eli kansiot Home ja Account. HomeController määrää toiminnot, jotka tapahtuvat navigoitaessa osoitteeseen ”sivuston_osoite/home”. Kyseinen osoite vie käyttäjän HomeControllerin Index-toimintoon (Kuva 6), joka ohjaa käyttäjän Index-nimiseen näkymään (Views-kansiossa). Account- ja ManageController liittyvät ASP.NET Identityyn ja se sisältävät käyttäjien hallintaan liittyviä toimintoja.

```
 9    public class HomeController : Controller
10    {
11        [AllowAnonymous]
12        public ActionResult Index()
13        {
14            return View();
15        }
16
17    }
```

Kuva 6. Osa HomeControllerin koodista.

Näiden kahden Controllerin lisäksi sovellukseen luotiin neljä uutta Controlleria. Luokkiin liittyvät WagonsController, WagonTypesController ja CarriersController sekä käyttäjäroolien hallintaan liittyvä RolesController. Samalla Views-kansioon luotiin automaattisesti vastaavat kansiot. Controllereihin liittyvistä näkymistä kerrotaan lisää luvussa 6.7. Kunkin Controllerin tehtävä on hallita erilaisia toimintoja. RolesControlleriin luotiin toimintoja liittyen uusien käyttäjäroolien luomiseen. Luokkiin liittyviin Controllereihin luotiin toiminnot Index, Edit ja New. Edit ja New -toiminnot ohjaavat käyttäjän samannimiseen näkymään (View). Tosin kyseiset toiminnot vaativat pääkäyttäjän roolin eli peruskäyttäjä ei pysty käyttämään näitä toimintoja. Index-toiminto ohjaa käyttäjän, roolista riippuen (Kuva 7), sopivaan näkymään. Esimerkiksi pääkäyttäjä ohjataan näkymään, jossa tietoja voidaan muokata ja poistaa.

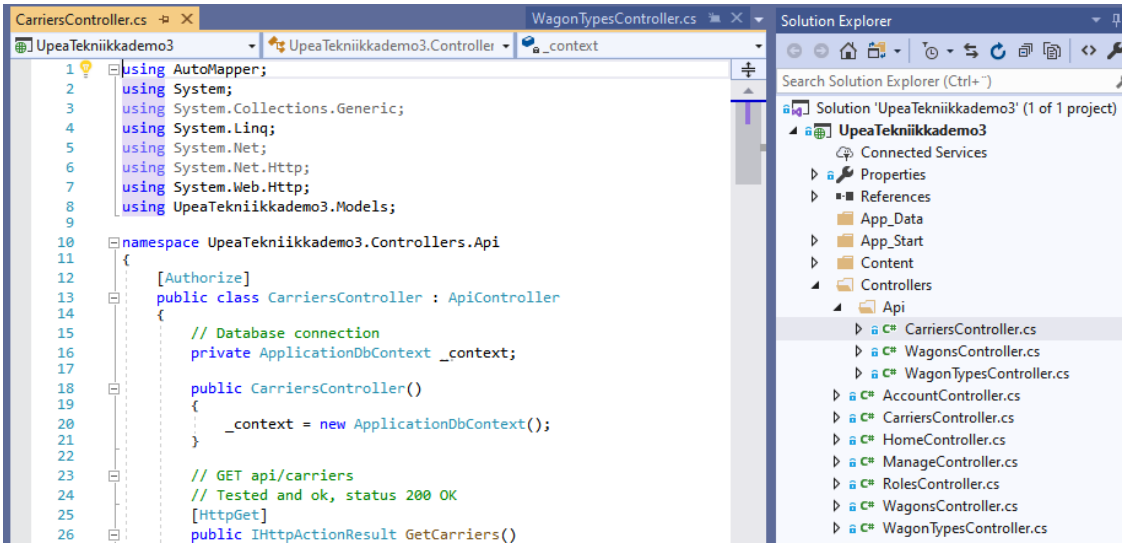
```
12      public class WagonsController : Controller
13      {
14          public ActionResult Index()
15          {
16              if (User.IsInRole("Administrator"))
17                  return View("AdminIndex");
18
19              return View("UserIndex");
20          }
21
```

Kuva 7. WagonsControllerin Index-toiminto.

6.6 **Web API:t**

Web API:t eli Web API Controllerit luotiin Controllers-kansion alikansioon nimeltä Api. Näitä Web API Controllereita luotiin sovellukseen kolme kappaletta, yksi kullekin luokalle. Nimet ovat samat kuin aiemmassa luvussa käsitellyillä Controllereilla. Web API Controllereilla ei ole vastinetta Views-kansiossa, sillä niiden tehtävä on välittää tietoa, ei näkymiä.

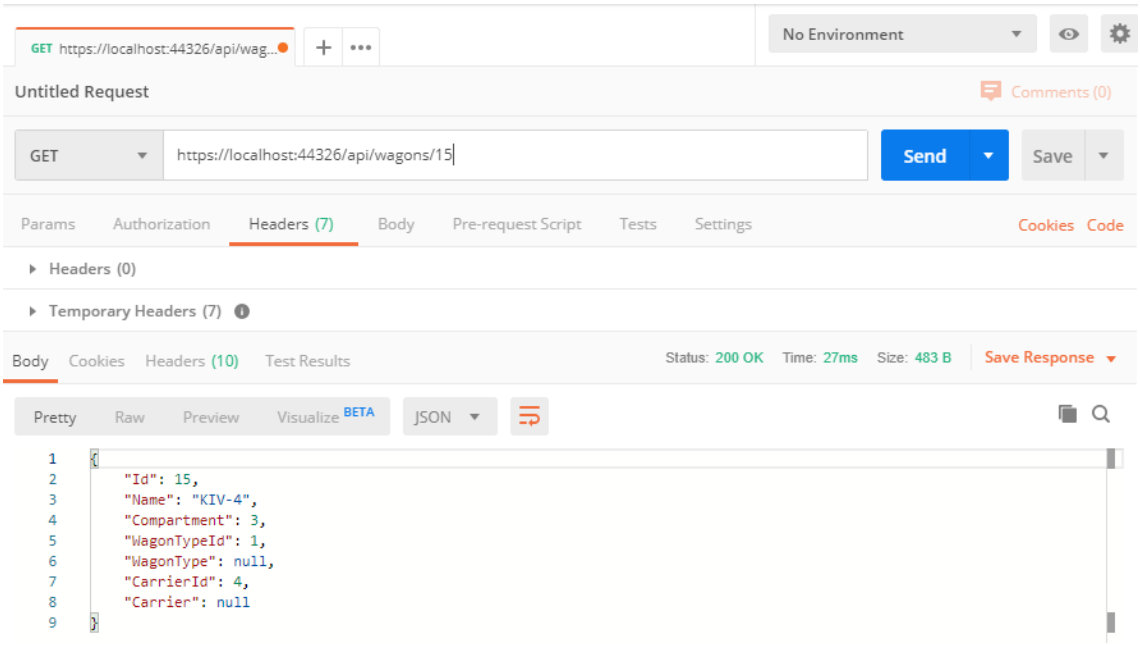
API:t luotiin tyhjinä, kuten äskeisessä luvussa mainitut uudet Controllerit-kin, ilman automaattista koodia, jotta voitiin olla tietoisia kaikesta sisällöstä. Luomisvaiheessa API:en niiden tyypiksi määriteltiin *Web API 2 Controller - Empty*. Kuvassa 8 vasemmalla nähdään CarriersController, joka periytyy ApiController-luokasta sekä oikealla projektin kansiorakenne.



Kuva 8. CarriersController, joka on ApiController.

Web API Controllereihin luotiin HTTP-metodien mukaiset toiminnot GET, POST, PUT ja DELETE, jotka vastaavat tietokannan CRUD-operaatioita. Toiminnot eivät saa tietoturvasyistä vastaanottaa tai palauttaa suoraan tietokannan objektia eli toiminnot toteutettiin tiedonsiirto-objekteja apuna käyttäen. Toiminnoissa käytettiin AutoMapper-työkalua tietokannan ja tiedonsiirto-objektien väliseen kommunikaatioon. API:en toimintojen

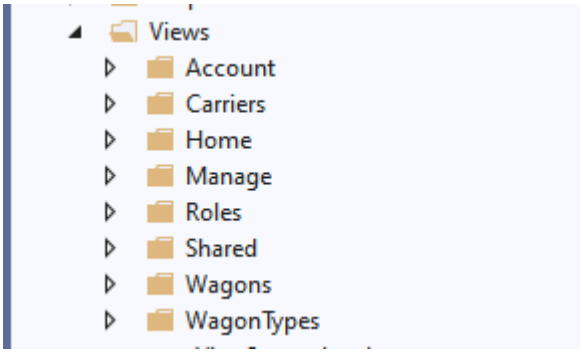
kirjoittamisen jälkeen testattiin API-kutsuja Postman-ohjelman avulla. Kuvassa 9 esimerkkinä vastaus GET-pyyntöön.



Kuva 9. GET-pyynnön vastaus Postman-sovelluksessa.

6.7 Views (näkymät)

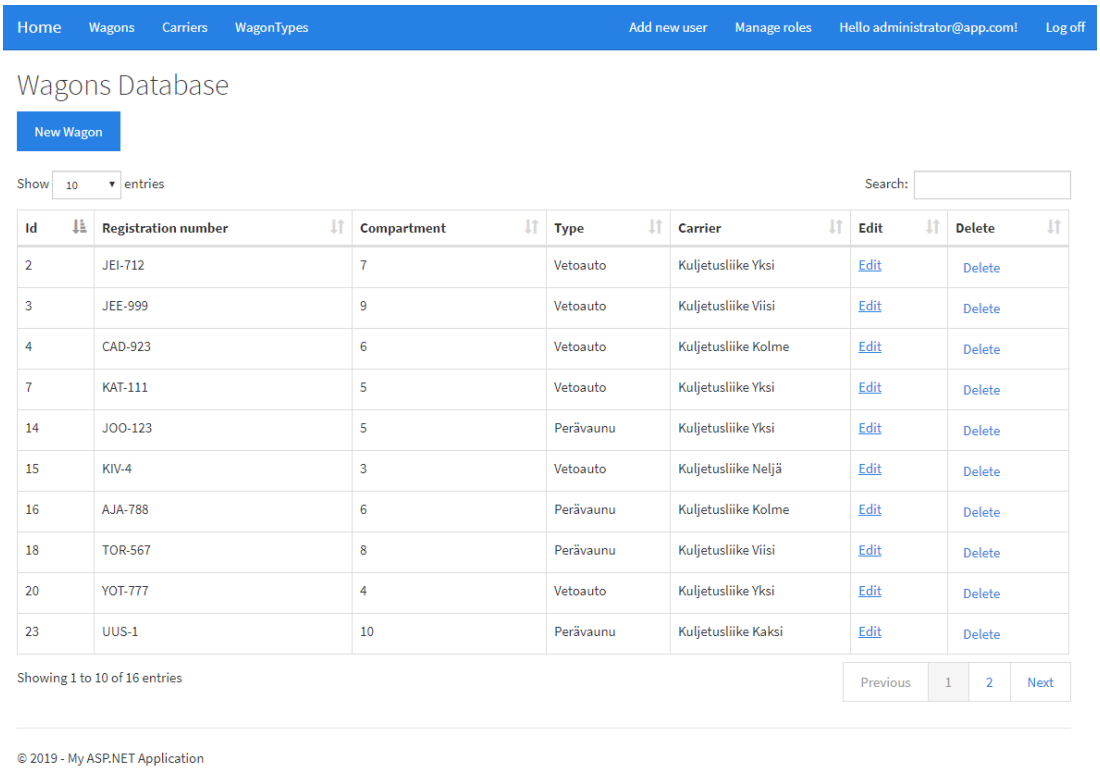
Kuten Controllereista kertovassa kappaleessa mainittiin, ohjelma loi Views-kansion alle automaattisesti kansiot kunkin Controllerin nimen mukaan (Kuva 10). Näiden lisäksi Views-kansiosta löytyy Shared-kansio, joka nimensä mukaisesta sisältää näkymiä, joita voidaan käyttää missä vain sovelluksen sisällä (esimerkiksi ulkoasu). Account-ja Manage-kansioissa sijaitsevat ASP.NET Identityn käyttäjienhallintaan liittyvät näkymät, joista osaa on muokattu sovelluksen tarpeisiin sopiviksi. Roles-kansiosta löytyy näkymät käyttäjäroolien selailuun ja lisäämiseen. Esimerkkinä käydään läpi Wagons-kansion sisällä olevat näkymät. Vastaavat löytyvät myös WagonTypes- ja Carriers-kansioista.



Kuva 10. Views-kansiorakenne.

Esimerkiksi Wagons-kansiosta löytyy näkymät, joita käytetään Wagons-Controllerin toiminnoissa. Näitä ovat AdminIndex, UserIndex, New ja Edit. Molemmilla Index-sivulla näytetään listaus kaikista tietokannan objekteista. Käyttäjäryhmän mukaan näytetään joko AdminIndex tai UserIndex-sivu. Pääkäyttäjä ohjataan AdminIndex-sivulle ja hänellä näkyy tietojen lisäys-, muokkaus- ja poistopainikkeet. Peruskäyttäjä ei pysty tekemään muuta kuin katsomaan tietoja eli hänellä näkyy pelkistetty UserIndex-sivu ilman lisäys-, muokkaus- ja poistopainikkeita.

Kuvassa 11 esimerkkinä näkymä pääkäyttäjän Index-sivusta, joka löytyy navigoitaessa osoitteeseen ”sivuston_osoite/wagons”. Näkymän toteutuksessa käytettiin apuna jQueryn DataTables-lisäosaa. Sovellukseen tehtiin tarvittavat määritykset lisäosan toimintakuntoon saattamiseksi. Tämän jälkeen Index-näkymään kirjoitettiin JavaScript-koodia, joka Ajaxia apuna käyttäen hakee tietokannan tiedot taulukkoon WagonsController-nimistä Web API:a hyödyntäen.



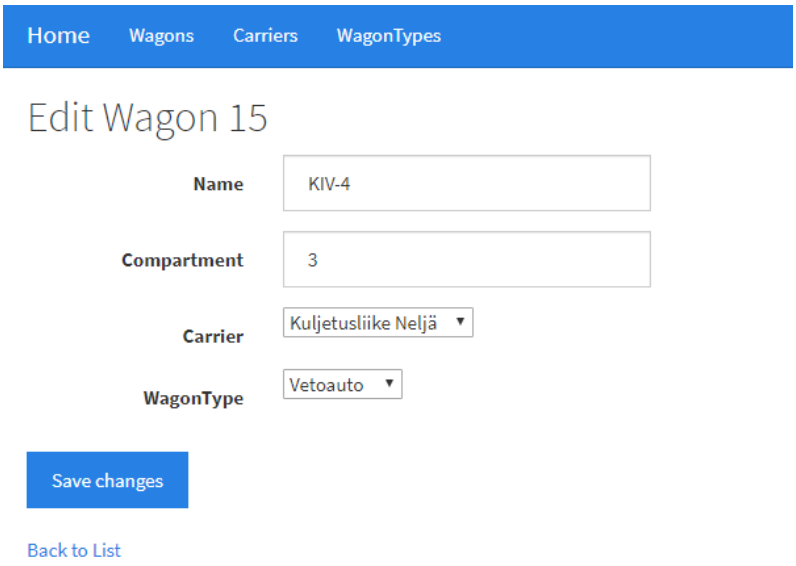
Kuva 11. Wagons-luokan objektit haettu WagonsController Web API:n kautta tietokannasta.

Tietojen poistaminen toteutettiin myöskin Ajaxia hyödyntäen (Kuva 12) siten, että Delete-linkkiä klikkaamalla käyttäjälle tulee ponnahdusikkuna, joka kysyy, haluatko poistaa objektin ja vahvistamalla tämän sekä sivu että tietokanta päivittyvät heti.

```
$("#table_for_db").on("click", ".deleteItem", function () {
    var button = $(this);
    if (confirm("Are you sure you want to permanently remove this item?")) {
        $.ajax({
            url: "/api/wagons/" + button.attr("data-wagon-id"),
            method: "DELETE",
            success: function () {
                table.row(button.parents("tr")).remove().draw();
            }
        });
    }
});
```

Kuva 12. Objektin poistaminen DataTables-taulukosta ja tietokannasta.

Uuden objektin lisääminen vie erilliselle New-sivulle ja tietojen päivitys tapahtuu Edit-sivulla. New-näkymään tehtiin lomake, jossa on kentät uuden objektin luomiselle tietokantaan. Kun lomake lähetetään, lähtee samalla POST-pyyntö WagonsController Web API:lle toteutettuna Axios-kirjaston avulla. Edit-näkymään toteutettiin samankaltainen lomake, mutta täytettynä muokattavan objektin nykyisillä tiedoilla (Kuva 13). Tämäkin toteutettiin Axios-kirjaston avulla. Edit-näkymän lomakkeen lähetys lähettää Web API:lle PUT-pyynnön muokattavan objektin osoitteeseen.



Kuva 13. Wagon-luokan Edit-näkymä.

6.8 Tietoturvan toteutus

Sivustolle luotiin kaksi käyttäjäryhmää ASP.NET Identityn avulla: Administrator ja User. Administrator-käyttäjäryhmä on tässä tapauksessa niin sanottu pääkäyttäjäryhmä ja siihen kuuluvilla henkilöillä on oikeus tehdä sivustolla kaikkia toimintoja. Administrator-käyttäjäryhmälle luotiin User-ryhmästä eroava navigaatiopalkki ja sitä kautta he voivat lisätä uusia käyttäjiä tai käyttäjäryhmiä sivustolle. User-käyttäjäryhmään kuuluvat niin kutsutut peruskäyttäjät voivat ainoastaan tarkastella tietokannassa olevia tietoja sivuston avulla. Heillä ei ole oikeuksia lisätä, muokata tai poistaa

tietoja. Käyttäjäryhmät tallennettiin Entity Frameworkin Migraation avulla tietokannan AspNetRoles-tauluun. AspNetUserRoles-tauluun taasen tallentuu tieto, kuka käyttäjä kuuluu mihinkin ryhmään. Käyttäjät löytyvät taulusta AspNetUsers.

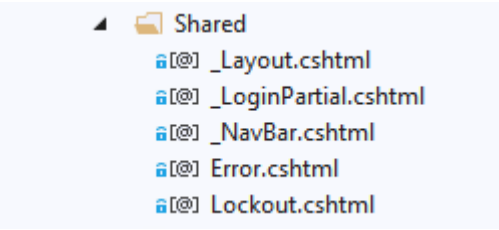
App_Start-kansion FilterConfig-tiedostoon lisättiin Authorize-suodatin, jolla koko sivusto saatiin näkymään vain rekisteröityneille käyttäjille. Poikkeuksena Home-sivu, jonka Controlleriin lisättiin suodatin [AllowAnonymous] (Kuva 6, s. 23). Suodattimia lisättiin myös muihin Controllereihin, jotta voitiin määritellä Administrator-käyttäjäryhmän sekä siihen kuulumattomien toiminnot.

Cross-Site Scriptingiltä suojautuminen hoidettiin käyttämällä näkymissä mahdollisuuksien mukaan Razor-syntaksin mukaisia ilmaisuja, sillä ne muunnetaan automaattisesti koodikielelle. Näkymissä, joissa tietoja lähetettiin JavaScriptin avulla, koodikielelle muuntaminen hoidettiin JavaScriptin escape-attribuutin avulla. Tosin escape-attribuutin käyttö havaittiin myöhemmin ongelmalliseksi suomen kielen ääkkösten kohdalla, sillä niitä ei olisi tarpeen muuntaa erikoismerkeiksi.

Sivuston käyttäjien hallintaan liittyvä rekisteröintisivu on jo valmiiksi suojattu Cross-Site Request Forgery -hyökkäyksiltä Token Verificationin avulla. Tietokantaan liittyvät tietojen päivitykset on suojattu Authorize-attribuutilla eli kukaan ulkopuolinen ei pääse kyseisiin toimintoihin käsiksi. Näissä ei voitu käyttää samanakaltaista Token-syntaksia, sillä tiedot menevät Web API:n kautta. Evästeiden varastamiselta suojauduttiin lisäämällä Web.config-tiedostoon system.webin sisälle HttpOnly-merkintä. Overpostingilta tietokantaa suojataan tiedonsiirto-objektien avulla eli käyttäjälle ei paljasteta ylimääräistä tietoa. Uudelleenohjausuhkia sovelluksessa ei juurikaan ole, sillä olemassa olevat uudelleenohjaukset osoitteeseen liittyvät ASP.NET Identityn toimintoihin ja niiden suojaus on automaattisesti hoidettu AccountControllerissa.

6.9 **Sivuston asettelu ja ulkoasu**

Sivuston asetteluun liittyvät näkymät löytyvät Views-kansion alikansiosta Shared (Kuva 14). Navigaatiopalkki löytyy ASP.NET-sovelluksessa oletuksena _Layout-tiedostosta, mutta sovellukseen luotiin navigaatiopalkki omaan tiedostoonsa _NavBar ja sisällytettiin se _Layout-näkymään. Näin navigaatiopalkkia ja sivuston rakennetta on helpompi muokata. Sivuston ulkoasuun liittyvää tiedostoa _Layout ei muokattu muuten kuin navigaation osalta.



Kuva 14. Ulkoasuun liittyvät näkymät.

Käyttäjälle näkyvä ulkoasu muokkautuu css-tyylitiedostojen avulla. Sivustolle haettiin sopiva Bootstrap-teema eli valmis ulkoasu Bootswatch.com-internetsivustolta. Teema ladattiin ASP.NET-sovellluksen Content-kansioon ja asetettiin käyttöön App_Start-kansion BundleConfig-tiedostossa. Sivuston ulkoasun ja tyylin voi nähdä aiemmista kuvista 11 ja 13 (sivut 26-27).

7 TULOKSET

ASP.NET MVC -sovelluksen ohjelmointi onnistui hyvin. MVC-malli koettiin selkeäksi käyttää ja sen koettiin myös helpottavan koodin kirjoittamista sekä lukemista. Projektin luomisvaiheessa oli muistettava ottaa ASP.NET Identity -ohjelmistokehyksen mukainen käyttäjien tunnistus käyttöön, sillä sitä ei ilmeisesti pysty lisäämään myöhemmin Visual Studion 2019 -versiossa. Sovellukseen saatiin luotua onnistuneesti kaikki halutut toiminnallisuudet eli sivu tietokannan kaikkien tietojen näyttämiselle ja myös uuden tiedon luonti sekä tietojen muokkaus -sivut.

Ohjelmointirajapintojen luominen Web API:en avulla onnistui myöskin odotetusti. Web API:t sisältävät CRUD-operaatioita vastaavat HTTP-metodien mukaiset toiminnot GET, POST, PUT ja DELETE. Toiminnot toimivat kuten pitääkin, eikä ongelmia toimivuuden suhteen ole havaittu. Toimintoja testattiin Postman-ohjelman avulla ja lopulta ASP.NET-sovelluksen kautta.

Yhteys ASP.NET MVC -sovelluksen käyttäjänäkymästä ohjelmointirajapintaan oli haastavampi asia. Tähänkin löytyi useita eri ratkaisuja. Päädyttiin toteuttamaan API-kutsut JavaScript-kirjastojen avulla. Tässä työvaiheessa meni ehdottomasti eniten aikaa. Kommunikointi API:n ja ASP.NET MVC -sovelluksen käyttäjänäkymän välillä onnistui, joskin pieniä kehityskohtia havaittiin. Esimerkiksi käsittelyssä olevan luokan ominaisuuksien validointeja ei saatu API:n kautta kulkemaan, kun kommunikointivälineenä käytettiin JavaScriptia. Myös suomen kielen ääkkösten koodikielelle muunto aiheutti haasteita ja jäänee jatkokehitykseen.

Yhteys Microsoftin SQL Serveriin luotiin onnistuneesti. Tietokannan taulut luotiin SQL Serveriin Entity Frameworkin Code First -migraatioita apuna käyttäen. Jos toimeksiantaja haluaa käyttää sovelluksessa Database First -tyyliä, niin sekin on mahdollista. Onnistuttiin luomaan toimeksiantajan toivoma ASP.NET-sovellus, joka on ohjelmointirajapinnan kautta yhteydessä Microsoftin SQL Serveriin.

8 YHTEENVETO JA POHDINTA

Yleisesti opinnäytetyönprosessin aikana opittiin paljon uutta tietoa. Tapoja toimeksiantajan toivoman sovelluksen luomiseen löytyy maailmasta vähintään yhtä monta kuin tekijöitäkin. Esimerkiksi ASP.NET MVC -sovelluksen ja Web API:t olisi voinut erottaa omiksi projekteikseen. Ja tietojen haku ohjelmointirajapinnasta olisi voitu toteuttaa muullakin keinolla kuin JavaScript-kirjastoja apuna käyttäen. Tämä muu tapa olisi ollut esimerkiksi erilaisten Web API Client -kirjastojen käyttäminen. JavaScript-kirjastojen avulla toteutetut API-kutsut koettiin aloittelijaystävällisemmäksi.

Ensimmäisessä tutkimuskysymyksessä pohdittiin, mikä olisi järkevä ja moderni tapa toteuttaa ohjelmointirajapinta .NET-ympäristössä. Tutkittiin SOAP- ja REST Web Servicen eroja ja päädyttiin lopulta toteuttamaan Web API, joka noudattaa REST-arkkitehtuuria. REST-arkkitehtuurin mukaiset rajapinnat ovat nykyään yhä yleisempiä, joten se on moderni tapa luoda ohjelmointirajapinta. Web API on myös järkevä valinta .NET-ympäristöön, sillä se on luotu .NET-ympäristön mukana ja toimii minkä tahansa .NET-sovelluksen kanssa.

Toinen tutkimuskysymys liittyi tietoturvallisuuteen, miten varmistetaan, ettei kukaan ulkopuolinen pääse käsiksi tietoihin. Sovellus toteutettiin ASP.NET MVC -mallia käyttäen ja käyttäjien hallinnassa hyödynnettiin ASP.NET Identityä. Molemmissa on paljon sisäänrakennettuja ominaisuuksia tietoturvallisuuteen liittyen. Esimerkiksi jos käyttää ASP.NET MVC -mallin näkymissä, syötettävän tekstikentän kohdalla Razor-syntaksin mukaisia ilmaisuja niin syötettävä teksti muunnetaan koodikielelle ennen kuin se lähetetään eteenpäin esimerkiksi tietokantaan. Tosin tämä ei toimi, jos tietoja lähetetään JavaScriptin avulla. Työssä hoidettiin kyseiset tapaukset JavaScriptin encode-attribuutin avulla, mutta myöhemmin huomattiin, että myös suomen kielen ääkköset muuntuvat erikoismerkeiksi. Tämä asia vaatinee lisäselvitystä jatkokehitysvaiheessa. ASP.NET Identity huolehtii käyttäjien tunnistuksesta ja valtuutuksesta. Näiden sisäänrakennettujen ominaisuuksien lisäksi sovelluksen suojaamisessa käytettiin muun muassa Authorize-attribuuttia, joka estää tuntemattomia tai tiettyyn käyttäjäryhmään kuulumattomia suorittamasta tiettyjä toimintoja. Ohjelmointirajapinnat eli Web API:t suojattiin myös samaa attribuuttia käyttäen.

Jatkopohdintana on mietittävä luotuja Web API -rajapintoja ja niiden turvallisuutta. Riittääkö Authorize-attribuutin käyttö vai vaatiiko niiden suojaus enempiä toimia. Microsoft kertoo sivuillaan, että ei ole kätevää tapaa lähettää Anti-Forgery Tokenia käyttäjälle, kun tietoja noudetaan API:n kautta. Toisaalta, Authorize-attribuutti pitää huolen, että tunnistautumaton käyttäjä ei pääse Web API:in käsiksi.

Aiheeseen liittyviä jatkokehitysideoita kertyi työtä tehdessä. Sovelluksen käyttäjien hallintaa voitaisiin kehittää ja ottaa esimerkiksi kaksivaiheinen

tunnistautuminen tai sähköpostivahvistukset käyttöön. Nämä ominaisuudet on mahdollista toteuttaa ASP.NET Identityn avulla. Työssä luodut ohjelmointirajapinnat (Web API:t) voidaan ottaa käyttöön muissakin sovelluksissa. Jos esimerkiksi tulevaisuudessa toimeksiantajayritys kokee tarpeelliseksi käyttää Microsoftin SQL Serverin tietokantaa uuden mobiilisovelluksen kautta, sieltä voidaan ottaa yhteys tässä työssä luotuihin Web API -rajapintoihin. Jatkokehityksenä voisi myös selvittää, kuinka saada luokkien ominaisuuksiin liittyvät validoinnit näkymään ASP.NET-sovelluksessa, kun tietokannan tiedot noudetaan Web API:n kautta JavaScriptin avulla.

Kaiken kaikkiaan opinnäytetyö oli tekijälleen laaja ja antoisa projekti. Lisäinspiraatiota työn tekemiseen saatiin siitä, että sitä tai sen osia tullaan mahdollisesti käyttämään tulevaisuudessa. Työn tekijän ohjelmointitaidot kasvoivat ja toimeksiantaja sai haluamansa sovelluksen rajapintoineen jatkokehitysideoiden kera.

LÄHTEET

Balani, N. & Rajeev, H. (2009). *Apache CXF Web Service Development : Develop and Deploy SOAP and RESTful Web Services*. Birmingham, UK: Packt Publishing Ltd. Haettu 30.9.2019. Ebook Central -tietokanta.

Bush, T. (2019). What Is The Difference Between Web Services and APIs? Blogijulkaisu 16.7.2019. Haettu 28.9.2019 osoitteesta https://nordica-pis.com/what-is-the-difference-between-web-services-and-apis/

Freeman, J. (2019). What is an API? Application programming interfaces explained. Blogijulkaisu 8.8.2019. Haettu 28.9.2019 osoitteesta https://www.infoworld.com/article/3269878/apis/what-is-an-api-application-programming-interfaces-explained.html

Galloway, J., Wilson, B., Allen, K. S., & Matson, D. (2014). *Professional ASP.NET MVC 5*. Indianapolis, Indiana: John Wiley & Sons, Inc. Haettu 18.9.2019. Ebook Central -tietokanta.

Inkinen, V. (2003). *ASP.NET.* Jyväskylä: Dark Oy.

Jimmy Bogard Revision. (2017). Getting Started Guide — AutoMapper documentation. Haettu 21.10.2019 osoitteesta https://auto-mapper.readthedocs.io/en/latest/Getting-started.html

Kanjilal, J. (2013). *ASP.NET Web API : Build RESTful web applications and services on the .NET framework*. Birmingham, UK: Packt Publishing Ltd. Haettu 26.9.2019. Ebook Central -tietokanta.

Koskimies, K. & Mikkonen, T. (2005). *Ohjelmistoarkkitehtuurit*. Jyväskylä: Gummerus Kirjapaino Oy.

Martin, J. & Tomson, B. (2002). *ASP.NET*. Helsinki: Edita Prima Oy.

Microsoft. (2012). Authentication and Authorization in ASP.NET Web API | Microsoft Docs. Haettu 22.10.2019 osoitteesta https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api

Microsoft. (2016a). Code First Migrations - EF6 | Microsoft Docs. Haettu 14.10.2019 osoitteesta https://docs.microsoft.com/fi-fi/ef/ef6/modeling/code-first/migrations/

Microsoft. (2016b). Code First to a New Database - EF6 | Microsoft Docs. Haettu 14.10.2019 osoitteesta https://docs.microsoft.com/fi-fi/ef/ef6/modeling/code-first/workflows/new-database

Microsoft. (2016c). Overview of Entity Framework 6 - EF6 | Microsoft Docs. Haettu 14.10.2019 osoitteesta https://docs.microsoft.com/fi-fi/ef/ef6/

Microsoft. (2018a). Choose between .NET Core and .NET Framework for server apps | Microsoft Docs. Haettu 1.10.2019 osoitteesta https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server?toc=%2Faspnet%2Fcore%2Ftoc.json&bc=%2Faspnet%2Fcore%2Fbreadcrumb%2Ftoc.json&view=aspnetcore-3.0

Microsoft. (2018b). Creating a Model - EF6 | Microsoft Docs. Haettu 14.10.2019 osoitteesta https://docs.microsoft.com/fi-fi/ef/ef6/modeling/

Microsoft. (2019a). ASP.NET Web APIs | Rest API's with .NET and C#. Haettu 18.9.2019 osoitteesta https://dotnet.microsoft.com/apps/aspnet/apis

Microsoft. (2019b). Choose between ASP.NET 4.x and ASP.NET Core | Microsoft Docs. Haettu 1.10.2019 osoitteesta https://docs.microsoft.com/en-us/aspnet/core/fundamentals/choose-aspnet-framework?view=aspnetcore-3.0

Microsoft. (2019c). Create Data Transfer Objects (DTOs) | Microsoft Docs. Haettu 26.9.2019 osoitteesta https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

Microsoft. (2019d). Download SQL Server Management Studio (SSMS) - SQL Server | Microsoft Docs. Haettu 30.9.2019 osoitteesta https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

Microsoft. (2019e). Introduction to ASP.NET Identity - ASP.NET 4.x | Microsoft Docs. Haettu 21.10.2019 osoitteesta https://docs.microsoft.com/fi-fi/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

Microsoft. (2019f). Overview of Visual Studio | Microsoft Docs. Haettu 30.9.2019 osoitteesta https://docs.microsoft.com/fi-fi/visualstudio/get-started/visual-studio-ide?view=vs-2019

Microsoft. (2019g). SQL Server Documentation - SQL Server | Microsoft Docs. Haettu 30.9.2019 osoitteesta https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-ver15

Microsoft. (2019h). What is .NET? An open-source developer platform. Haettu 22.8.2019 osoitteesta https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

Microsoft. (2019i). What is .NET Framework? A software development framework. Haettu 17.9.2019 osoitteesta https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

Microsoft. (2019j). What is ASP.NET? | .NET. Haettu 22.8.2019 osoitteesta https://dotnet.microsoft.com/learn/web/what-is-aspnet

Microsoft. (2019k). Why choose the .NET developer platform? Haettu 17.9.2019 osoitteesta https://dotnet.microsoft.com/platform/why-choose-dotnet

Mikkonen, J. (2017). Rest on nettipalveluiden yhteinen kieli. Blogijulkaisu 27.5.2017. Haettu 28.9.2019 osoitteesta https://www.tivi.fi/uutiset/rest-on-nettipalveluiden-yhteinen-kieli/23703ab5-dd19-383e-a422-ebfc3d910583

Mozilla. (n.d.-a). Ajax - Developer guides | MDN. Haettu 11.11.2019 osoitteesta https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Mozilla. (n.d.-b). Web APIs | MDN. Haettu 11.11.2019 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API

Postman Inc. (2019). How to Use Postman API Client: GraphQL, REST, & SOAP Supported. Haettu 21.10.2019 osoitteesta https://www.getpostman.com/product/api-client

Salminen, L. (2018). Tietokannat-kurssin verkkoaineisto, Moodle. Hämeen ammattikorkeakoulu. Haettu 2.9.2019 osoitteesta https://moodle.hamk.fi

Simsek, G. (2019). What Is New About NewSQL? - Software Engineering Daily. Blogijulkaisu 24.2.2019. Haettu 1.10.2019 osoitteesta https://softwareengineeringdaily.com/2019/02/24/what-is-new-about-newsql/

Smallcombe, M. (2019). SQL vs. NoSQL: How Are They Different and What Are the Best SQL and NoSQL Database Systems? Blogijulkaisu 23.8.2019. Haettu 3.9.2019 osoitteesta https://www.xplenty.com/blog/the-sql-vs-nosql-difference/

SpryMedia Ltd. (n.d.). DataTables | Table plug-in for jQuery. Haettu 15.10.2019 osoitteesta https://datatables.net/

Volosoft. (2019). ASP.NET Boilerplate | Data Transfer Objects. Haettu 26.9.2019 osoitteesta https://aspnetboilerplate.com/Pages/Documents/Data-Transfer-Objects

Zabriskie, M. (n.d.). GitHub - axios/axios: Promise based HTTP client for the browser and node.js. Haettu 31.10.2019 osoitteesta https://github.com/axios/axios