OPPIMISPORTAALIN TAUSTAJÄRJESTELMÄN TOTEUTUS

LAB-AMMATTIKORKEAKOULU
Tekniikan ala
Tieto- ja viestintätekniikka
Ohjelmistotekniikka
Kevät/Syksy 2020
Otto Kyllönen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Kyllönen, Otto	Opinnäytetyö, AMK	kevät 2021
	Sivumäärä	
	29	

Työn nimi

Oppimisportaalin taustajärjestelmän toteutus

Tutkinto

Tieto- ja viestintätekniikka (AMK)

Tiivistelmä

Opinnäytetyön aiheena oli oppimisportaalin back-end:in toteutus ASP.NET Core-ohjelmistokehyksellä. Tavoitteena oli saada aikaan pilottiversio. Tarkempina tavoitteina oli sovelluksen tietokannan kehittäminen sekä autentikoinnin toteutus. Sovellus toteutettiin Piilaakso Academy Oy:lle.

ASP.NET Core on .NET-alustan ohjelmistokehys, joka toimii usealla alustalla. Core-sovelluksissa voi hyödyntää .NET-alustan teknologioita kuten LINQ-teknologiaa, joka lisä kantakyselyt suoraan koodikieleen tai Entity Frameworkia, joka mahdollistaa tietokannan käsittelyn käyttäen .NET-objekteja.

Oppimisportaalin tavoitteena on auttaa käyttäjiä kehittymään ja seuraamaan kehitystä. Sovelluksessa on taidot ja sertifikaatit, joilla ilmaistaan käyttäjän osaamista. Lisäksi on ryhmiä ja kursseja, joiden tarkoituksena on ryhmitellä käyttäjiä sekä antaa tavoitteita. Yhtiöt ovat kuin ryhmiä, joita voidaan jakaa yhtiöryhmiin, joilla on omat taidot ja sertifikaatit.

Sovelluksessa käytettiin tietokantaa, johon vaikutettiin koodissa käyttäen Entity Framework Corea sekä LINQ-teknologiaa. Lähes kaikki resurssit ovat pääpiirteittäin hyvin samanlaisia keskenään tietokannan näkökulmasta, mutta muuten eroavaisuuksia löytyy. Valtuuttaminen on roolipohjaista ja määrittely on ohjain tai metodikohtaista.

Sovelluksesta valmistui pilottiversio, jossa toimii suurin osa ominaisuuksista. Lisää ominaisuuksia on suunnitteilla ja niiden mukaan voi olla tarpeen muuttaa jo olemassa olevaa pohjaa.

Asiasanat

.NET, ASP.NET Core, Back-end, Entity Framework, LINQ

Abstract

Author(s)	Type of publication	Published
Kyllönen, Otto	Bachelor's thesis	Spring 2021
	Number of pages	
	29	

Title of publication

Development of learning portal Back-End

Name of Degree

Bachelor of Information technology

Abstract

Topic of the thesis was developing Learning portal applications back-end using ASP.NET Core-framework. Goal was to develop a pilot version of the application. Application was made for Piilaakso Academy Oy.

ASP.NET Core is a .NET-platforms multiplatform framework. Core applications can take advantage of .NET-platforms technologies such as LINQ, which integrates query capabilities to programming language or Entity Framework that makes it possible for developers to work with databases using .NET objects.

Goal of the Learning portal was to help users to improve user's skills and keep track of their improvement. The application has skills and certificates to express user's skills. There are also groups and courses that are meant to group users and provide goals to them. Companies are like groups that can also be divided into company groups each with their own skills and certificates.

Application used a database and Entity Framework Core and LINQ to work with the said database. Most of the resources where similar from the database perspective, but there where notable differences. Authorization was done with role-based authorization and defined on the method or controller basis.

Pilot version was carried out with mostly functional features. More features are on their way to be implemented, which may require to apply required changes to the current groundwork.

Keywords

.NET, ASP.NET Core, Back-end, Entity Framework, LINQ

SISÄLLYS

1 JOHDANTO 1
2 TEKNOLOGIAT 2
2.1 .NET 2
2.1.1 NuGet 2
2.1.2 CLR 3
2.2 ASP.NET Core 3
2.2.1 Identity 4
2.2.2 Valtuuttaminen 5
2.3 Entity Framework Core 6
2.3.1 Malli 8
2.3.2 Kyselyt 10
2.4 LINQ 10
2.5 Reititys 13
3 PROJEKTI/OPPIMISPORTAALI 15
3.1 Tavoite 15
3.2 Toiminnan kuvaus 16
3.3 Roolit 16
3.4 Tavoitetasot 17
3.5 Tietokannan kuvaus 17
4 BACKENDIN TOTEUTUS 19
4.1 Tietokanta 19
4.1.1 Yritysryhmät ja yritykset 19
4.1.2 Kurssit 20
4.1.3 Ryhmät (Groups) 20
4.2 Käyttäjä 20
4.3 Kieli 21
4.4 Reititys 21
4.5 Rajapintojen testaus 22
5 YHTEENVETO 26
6 LÄHTEET 27

1 JOHDANTO

Opiskeleminen ja oppiminen ovat murroksessa. Oppiminen vauhdittuu digitalisaation myötä, kun aineistoja ja opetusta on saatavilla eri muodoissa. Mikäli on halu oppia, ei tarvitse välttämättä hakeutua koulutukseen tai pyytää toista henkilöä opettamaan, vaan voi etsiä materiaalia itse verkosta. Tämä myös mahdollistaa oppimisen ajasta ja paikasta riippumatta. Oppimista ja sen kehitystä on mielenkiintoista, mutta myös hyödyllistä, seurata. Kehitystä seuraamalla voi suunnata jatkon kannalta tarvittavien taitojen kehitystä, josta hyötyy sekä henkilö itse, että mahdollinen työnantaja tai vastaava.

Piilaakso Academy on lahtelainen digitaalisen osaamisen kehittämiseen ja työllistymiseen keskittyvä yritys, joka on perustettu vuonna 2018. Piilaakso auttaa yrityksiä rekrytoimaan vaatimuksia vastaavia työntekijöitä, sekä toteuttaa koulutuksia.

Tämän opinnäytetyön tavoitteena on oppimisportaalin kehittäminen ja pilottiversion luonti käyttäen ASP.NET Corea ja sen versiota 3.1. Lisäksi tavoitteena on kehittää käyttäjien autentikointia sekä toimiva tietokantarakenne. Tämä opinnäytetyö keskittyy projektin backend-toteutukseen.

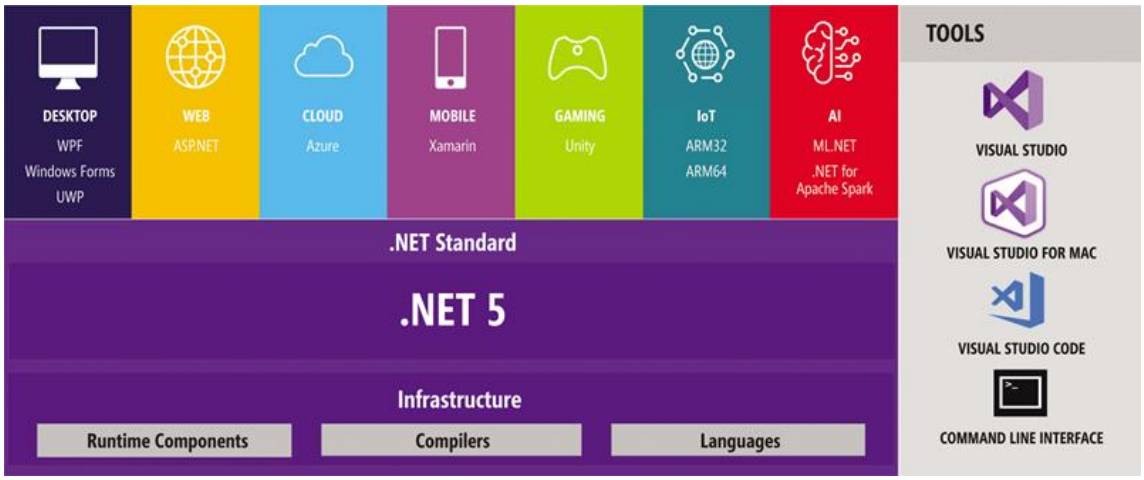
Tässä opinnäytetyössä on teoriaosuus, johon kuuluvat kappale 2, ja käytännön osuus, johon kuuluu kappaleet 3, 4 ja 5. Teoriaosuudessa käydään läpi ensin .NET -ympäristöä ja sitten siihen liittyviä teknologioita Entity Frameworkia sekä LINQ:a. Lopuksi käsitellään työssä käytetyn ASP.NET Coren perusteita. Käytännön osuudessa ensin tutustutaan oppimisportaalin tavoitteisiin, jonka jälkeen käydään läpi toteutusta. Toteutuksessa pääosin käsitellään tietokantaan liittyviä keskeisiä asioita.

2 TEKNOLOGIAT

2.1 .NET

.NET on Microsoftin kehittämä avoimen lähdekoodin alusta, joka sisältää useita eri toteutuksia. Tunnetuimmat toteutukset ovat: .NET Core, .NET Framework ja Xamarin. .Net Core on monelle alustalle suunniteltu kehitysalusta, .Net Framework on vain Microsoftin Windows -käyttöjärjestelmillä toimiva kehitysalusta ja Xamarin on mobiilikäyttöjärjestelmille suunnattu versio. (Microsoft t, 2020.) Ohjelmointikielinä käytettävissä on C# (C-Sharp), F# (F-Sharp) ja Visual Basic. IDE:nä (Integrated Development Environment) Microsoft suosittelee Visual Studiota Windowsilla sekä MacOS:llä, Visual Studio Codea, CLI:tä (Command Line Interface) sekä GitHub Codespacea. (Microsoft p, 2020.) Kuvassa 1 on yhteenveto .NET-alustasta. Kuvassa on lajiteltu käyttökohteet, niihin tarkoitetut toteutukset sekä suositeltuja ohjelmankehitysympäristöjä ja työkaluja. Käyttökohteiden alla on .NET-standardi, joka sisältää alustakohtaiset rajapintamääritelmät, kirjastot ja luokat. Infrastruktuuriin kuuluu ajonaikana olevat komponentit, kääntäjät sekä ohjelmointikielet. (Microsoft t, 2020.)

Kehityksen kannalta kaikki .NET-kehitysalustan toteutukset mahdollistavat samojen ohjelmointikielien ja usein myös samojen työkalujen käytön. Yhdistetyn standardisoidun kirjaston avulla kehittäjät voivat käyttää samoja työkaluja alustasta huolimatta. (Microsoft t. 2020)



Kuva 1. .NET alusta (Microsoft .Net Blog, 2019)

2.1.1 NuGet

NuGet-paketit ovat .NET-ympäristön käyttämiä koodipaketteja, joissa on valmiiksi käännetty koodi .dll-muotoisessa tiedostossa, manifesti, jossa kerrotaan paketin versionumero

ja vastaavia tietoja, sekä loput koodin käyttämisen kannalta tärkeät tiedostot. Käyttäjät voivat valita tarpeidensa mukaan NuGet-paketteja kirjastosta ja ottaa niitä käyttöön projektissaan. (Microsoft a, 2019.)

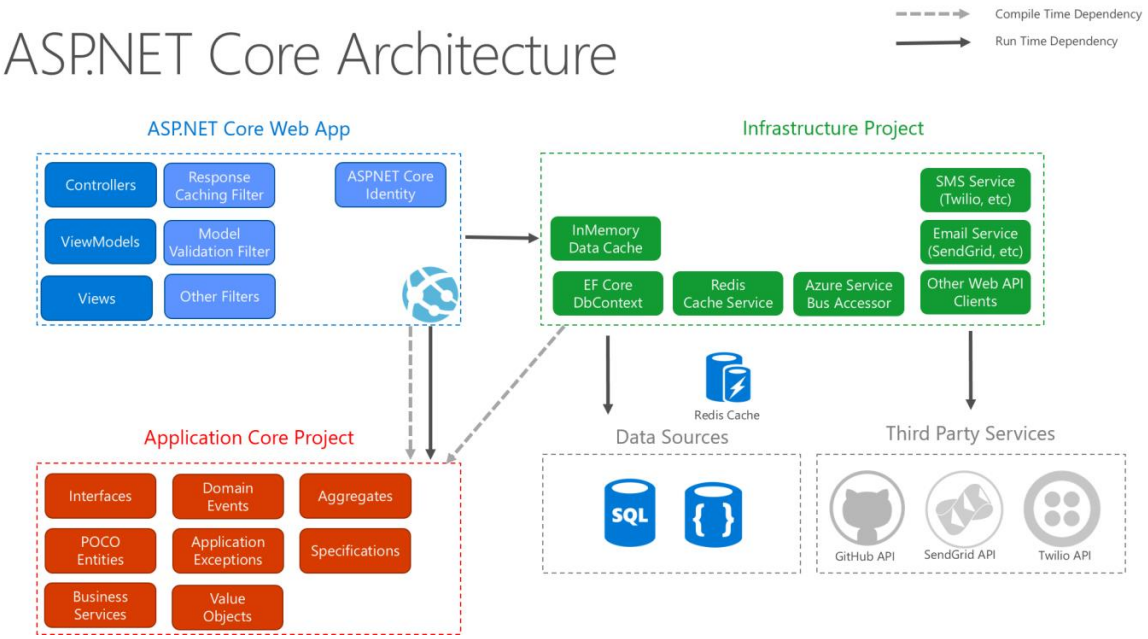
2.1.2 CLR

CLR (Common Language Runtime) on virtuaalikone, joka vastaa muistinhallinnasta sekä toimii kääntöympäristönä välitason koodille. IL-koodi on osittain käännettyä koodia, joka käännetään koneelle ymmärrettäväksi koodiksi ympäristön toimesta. Koodi on osittain käännettyä, koska ei ole varmuutta siitä, minkälaisessa ympäristössä sitä suoritetaan, joten loppuun kompilointi tapahtuu vasta kohdelaitteessa. Ennen suoritusta käytetään JIT-kääntäjää (Just-In-Time), jolla IL-koodi käännetään konekieleksi. IL tunnetaan myös lyhenteellä MSIL (Microsoft Intermediate Language) Microsoftin ympäristöissä. CLR implementaatio on eri jokaiselle .NET toteutukselle. (Microsoft f, 2020.)

2.2 ASP.NET Core

ASP.NET Core on Microsoftin kehittämä avoimen lähdekoodin .NET ohjelmistokehys, jonka tarkoituksena on monialustaisen verkkosovellusten tuottaminen (Tutorialsteacher, 2020). Coren ensimmäinen versio julkaistiin vuonna 2016. Ohjelmistokehyksestä on tähän mennessä tullut useita eri versiota, joista uusin on 5.0, joka julkaistiin vuoden 2020 marraskuussa. Coressa yhdistyy MVC (Model-View-Controller), Web API, sekä Web Pages. ASP.NET Coressa ohjelmointikielissä vaihtoehtoina on C# ja F#. Ajonaikaisena ympäristönä ASP.NET Core käyttää .NET Corea. (Chowdhuri 2016, 8-; Microsoft u. 2021.)

Kuvassa 2 näkyy ASP.NET Coren arkkitehtuuri. Coressa eri osat on jaettu kerroksiin. Sovelluskerroksessa on logiikkaan ja käyttöliittymään liittyvät osat kuten ohjaimet ja näkymät. Infrastruktuurikerroksessa on tietokannan käsittelyyn, verkkorajapintojen käsittelyyn sekä muistiin varastointiin liittyvät osat. Ohjelmanydinkerroksessa on esimerkiksi objektit ja rajapinnat. Sovellukseen voi myös kytkeä kolmannen osapuolen palveluita sekä tietokantoja.



Kuva 2. Asp.NET Core arkkitehtuuri (c# Corner 2020)

ASP.NET Core sovelluksissa HTTP-pyynnöt eivät tule suoraan sovellukselle, vaan erilliselle HTTP-palvelimelle, josta pyynnöstä muodostetaan erilaisista rajapinnoista koottu ”HttpContext”-objekti. Coressa tulee mukana kaksi verkkopalvelinta: Kestrel sekä WebListener. Erona on käyttöalustat, joista Kestrel toimii monella alustalla ja WebListener vain Microsoftin alustoilla. (Smith, 2017.)

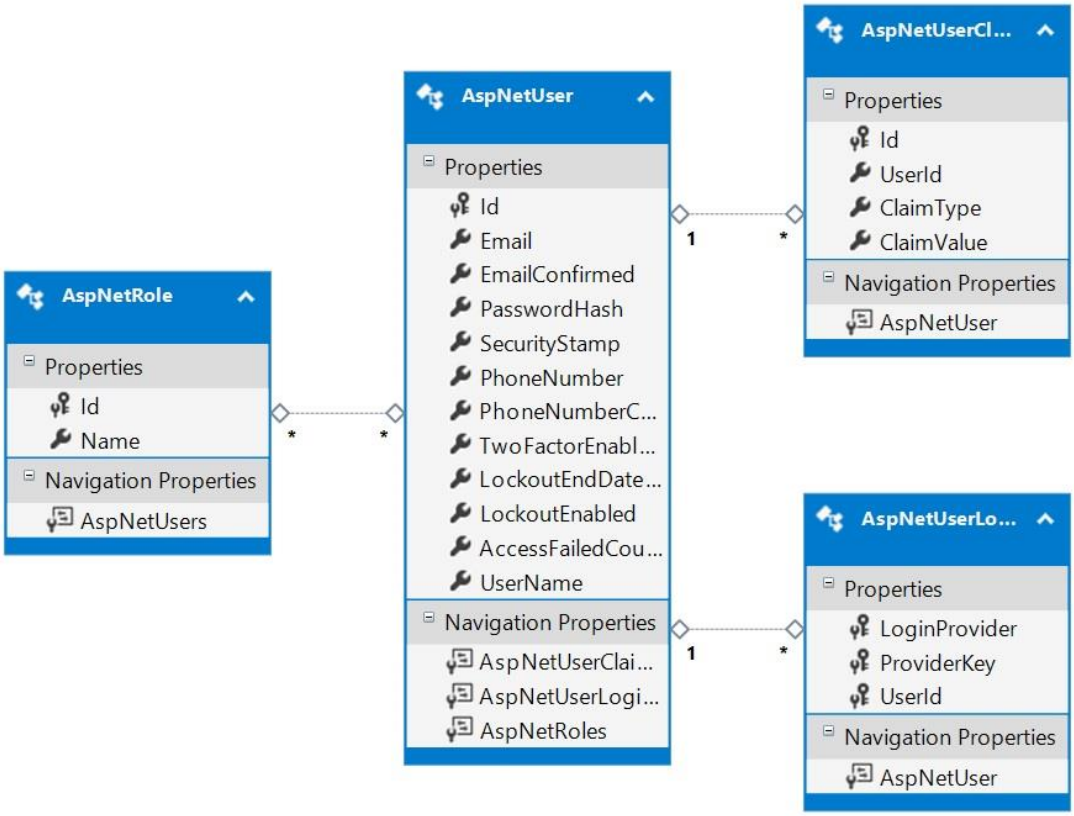
ASP.NET Core tarjoaa näkymien kehittämiseen kaksi työkalua: Blazor ja Razor Pages. Razor Pagesissa on mahdollista upottaa koodia suoraan näkymään tai erilliseen tiedostoon, joka luodaan samalla kun luo Razor Pages-sivun. Blazor mahdollistaa C#:in käytön JavaScriptin sijaan. (Microsoft k, 2020.)

2.2.1 Identity

Identity on Microsoftin kehittämä käyttäjienhallintajärjestelmä, joka vastaa käyttäjien autentikoinnista ja valtuuttamisesta. Identity mahdollistaa omien kirjautumismetodien luonnin ja hallinnan. Tuki löytyy myös ulkopuoliselle autentikoinnille, kuten esimerkiksi Facebook-tunnuksille. Autentikoinnissa palvelin tarkistaa käyttäjän tiedot ja käyttäjältä vaaditaan tunnistautuminen. Valtuuttamisessa palvelin tarkistaa, onko käyttäjällä oikeus hänen hakemaansa resurssiin tai tiedostoon. (Microsoft l, 2019.)

Käyttäjän tiedot tallennetaan oletuksena tietokantaan, jolloin Identity käyttää oletuksena Entity Frameworkkia (EF). Taulut ovat ennalta luotuja, mutta ovat muokattavissa migraatioiden avulla. Kuvassa 3 näkyy vakio rakenne käyttäjäntiedoille. Taulut ovat rooleille, käyttäjille, väitteille sekä kirjautumisille. Kuvasta näkyy myös taulujen väliset suhteet.

Käyttäjällä voi olla useita rooleja ja roolilla voi olla useita käyttäjiä. Yhdellä käyttäjällä voi olla useita kirjautumisia sekä väitteitä (engl. claims), mutta väite ja kirjautuminen on käyttäjäkohtainen. Migraatiot lisätään, kun mallia muutetaan, ja niistä vastaa Entity Framework / Core. EF vertaa sen hetkistä mallia tilannekuvaan vanhasta mallista ja päivittää muutokset, mikäli niitä on. Koodissa on mahdollista määrittää mikä jää migraation ulkopuolelle. (Microsoft i, 2019.)



Kuva 3. Identity Frameworkin käyttäjätaulut (Gazzo, 2014)

Rajapinnat käyttäjien sekä roolien hallintaan ovat nimeltään managereja. Käyttäjille ja rooleille on molemmille omansa. Molemmissa on metodeita resurssien hakuun, lisäämiseen, muokkaukseen sekä poistoon. Sisään kirjautumisen API on puolestaan ”SignInManager” (Microsoft k, 2020.)

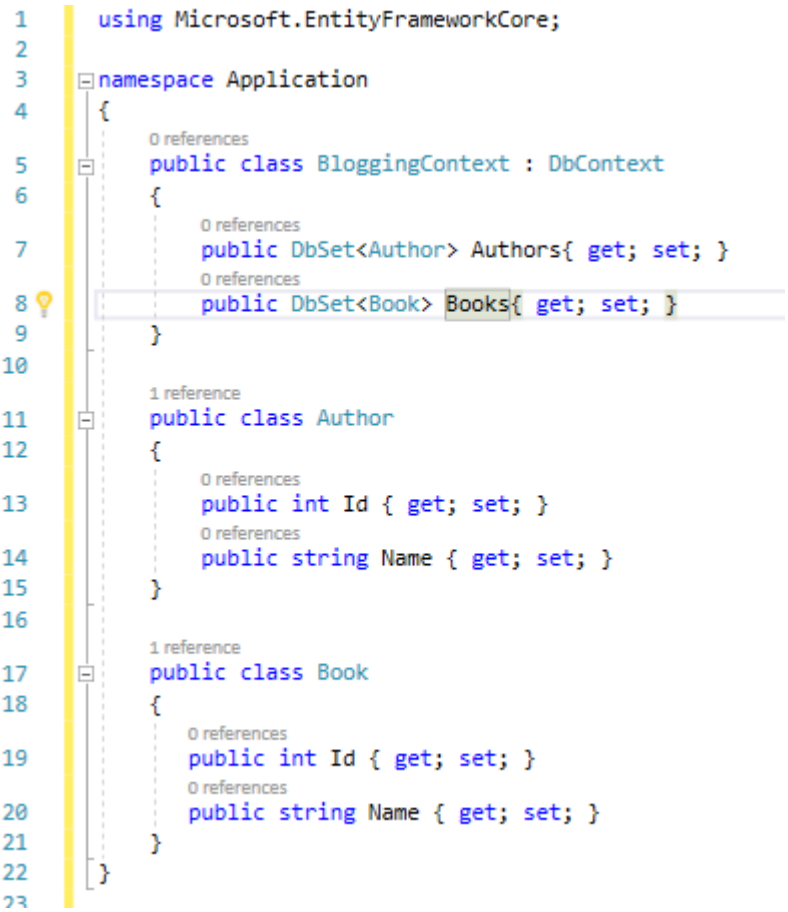
2.2.2 Valtuuttaminen

Valtuuttamiseen on useita tapoja. Sellaisia ovat esimerkiksi: rooli- sekä väitepohjainenvaltuutus. (Claims). Roolipohjaisessa valtuuttamisessa määritetään sallitut roolit, jotka voivat

kutsua päätepistettä. Pyynnön saapuessa tarkistetaan käyttäjärooli-taulusta rooli, jonka jälkeen joko jatketaan pyynnön suoritusta tai evätään käyttäjältä pääsy. (Microsoft r, 2016.) Väitepohjaisessa valtuuttamisessa väitteet ovat nimi-arvo-pari. Käyttäjää luotaessa annetaan hänelle pari määrittäen käyttäjän oikeudet. Väitteillä voi tarkemmin määritellä oikeuksia, kuin mitä rooleilla on mahdollista. (Microsoft e, 2016.)

2.3 Entity Framework Core

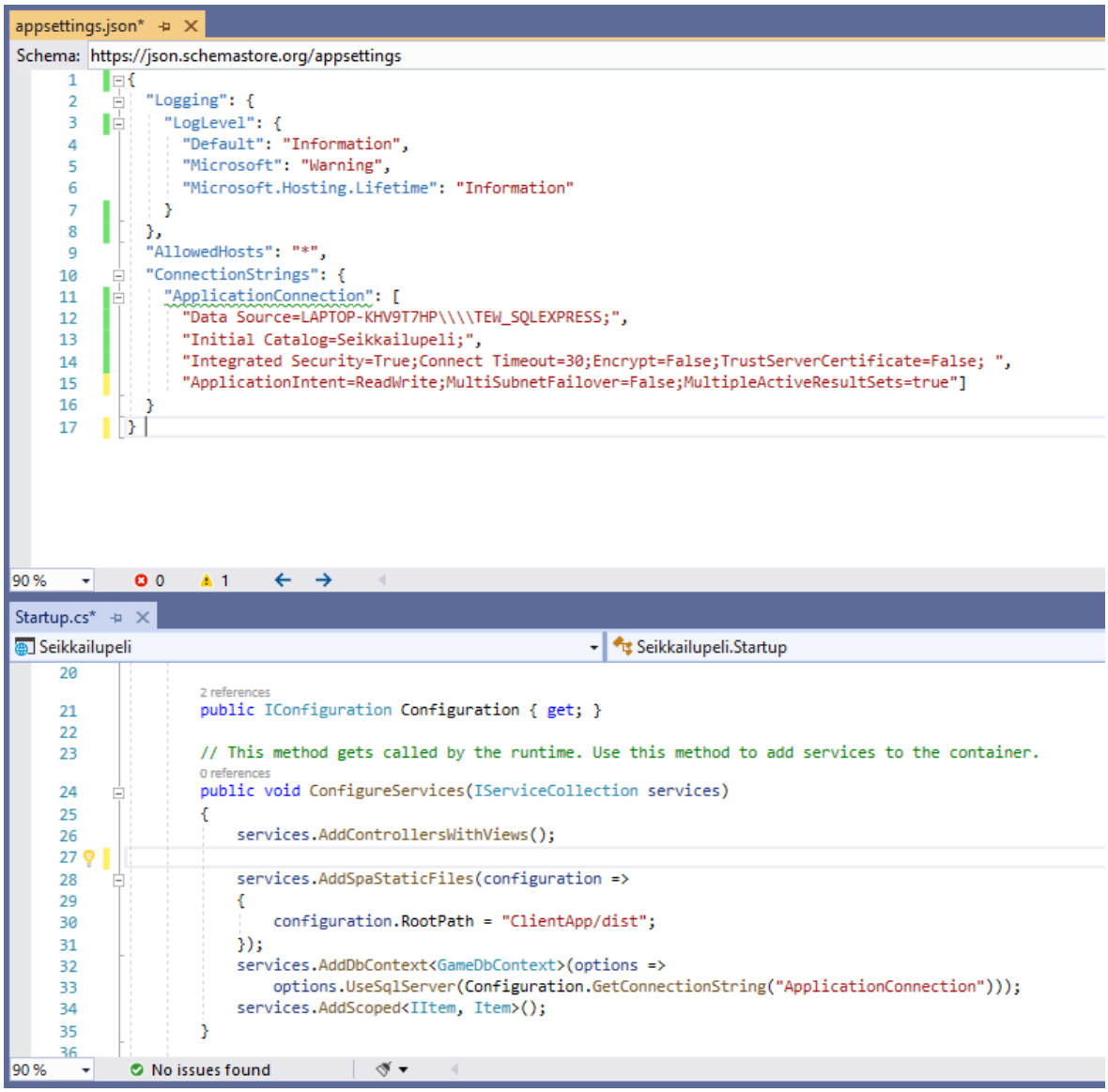
Entity Framework on avoimen lähdekoodin järjestelmäriippumaton .NET ympäristön ORM-kehys (Object-Relational Mapping), jonka avulla tietokantojen dataa voidaan käsitellä käyttäen .NET-objekteja. Entiteetti on luokka, joka kuvaa tietokannan taulua. Entiteetillä viitataan johonkin olemassa olevaan resurssiin, kuten käyttäjään. Käyttäjätaulussa oleva rivi viittaa luultavasti olemassa olevaan henkilöön. Esimerkiksi kuvassa 4 rivillä 7 on määritetty Author-luokan kokoelmaa kuvaava muuttuja, jota käytettäessä se yhdistää tietokannan Authors -nimiseen tauluun ja tieto, joka palautetaan haun yhteydessä, on Author-tyyppistä. Entity Framework tukee usean valmistajan tietokantamoottoreita. Datan sitominen tyyppiin tapahtuu mallin (model) kautta. (Microsoft h, 2020.) Kuvassa näkyy myös tietokantakonteksti-luokka, jota käytetään luokkien sekä tietokannan yhdistäjänä. Tietokantakontekstiluokka vastaa tietokantaan kohdistuvista kyselyistä sekä tulosten käännöstä objekteiksi.



```
using Microsoft.EntityFrameworkCore;

namespace Application
{
    0 references
    public class BloggingContext : DbContext
    {
        0 references
        public DbSet<Author> Authors{ get; set; }
        0 references
        public DbSet<Book> Books{ get; set; }
    }

    1 reference
    public class Author
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
    }

    1 reference
    public class Book
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
    }
}
```

Kuva 4. Entity Frameworkin mukainen olion sidonta tietokantaan

Tietokantayhteydet määritetään Asp.Net Coressa ”Startup.cs”-tiedostossa sekä ”appsettings.json”-tiedostossa. (Microsoft h, 2020.)



Kuva 5: Tietokantayhteyden määritys

Entity Framework sisältää asynkroonisen version jokaiselle I/O toiminnoille. Samalle tietokantakontekstille ei voi lähettää useita kyselyitä samanaikaisesti. LINQ-komennoille, kuten where, ei kuitenkaan ole asynkronista vastinetta, sillä vain suoritusta aiheuttavilla komennoilla on sellainen. (Microsoft b, 2020.)

2.3.1 Malli

Entity Frameworkin mallit koostuvat entiteettiluokista sekä kontekstiobjektista. Kontekstiobjekti mahdollistaa tiedon hakemisen ja muokkaamisen. Mallin luonnissa määritetään muuttujat ja niiden tyypit. Käyttäen attribuutteja (data annotaatio) on mahdollista esimerkiksi tehdä ominaisuudesta pakollinen, rajoittaa arvoa, sitä minkälaisena se näytetään näkymässä ja onko ominaisuus vierasavain vai pääavain (Microsoft g, 2020.) Muuttujan

tyyppi voi vaikuttaa myös sen pakollisuuteen. Jos NRT-attribuutti (Nullable Reference Types) on päällä, nollattaville (nullable) muuttujille ei tarvitse määrittää arvoa, kun taas ei nollattaville muuttujille on määritettävä. Käytännössä arvo on pakollinen, mikäli se ei ole nollattava. Oletuksena NRT on kuitenkin pois päältä, eli mikäli haluaa muuttujan olevan pakollinen, tarvitsee muuttujalle asettaa attribuutilla pakollisuus. (Microsoft v, 2020.)

Mallit voidaan jakaa myös kahteen eri tyyppiin: näkymämalleihin (View Model) ja tietokantamalleihin (Database Model). Mallissa voi tarvita ominaisuuksia, joita ei ole tietokannassa. Tämänlainen muuttuja tulee merkata NotMapped- attribuutilla, sillä muuten tietokantaa käytettäessä tulee virheitä ja sovellus joko kaatuu kokonaan tai ei päästä ohjelmaa etenemään. Kuvassa 6 on esimerkki id-muuttujasta, jolle on annettu Required-attribuutti. Eli Id on tässä tapauksessa annettava objektia luodessa.

```
[Required]
4 references
public int Id { get; set; }
```

Kuva 6. Esimerkki attribuutista

Attribuuttien ohella voi käyttää Fluent API:a määrittämään mallin ominaisuuksia. OnModelCreating metodissa voi asettaa esimerkiksi jonkun ominaisuuden pakolliseksi (Kuva 7). Fluent API:lla tehdyt määritykset yliajavat attribuuteilla tehdyt muutokset. (Microsoft g, 2020.)

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Item>()
        .Property(b => b.Id)
        .IsRequired();
}
```

Kuva 7. Fluent API:lla tehty määritys ”Item”-tyyppiselle objektille

Fluent API:lla on mahdollista vaikuttaa taulujen suhteisiin esimerkiksi niin, että on yksi oppilas, jolla monta arvosanaa. Mikäli oppilas poistetaan, poistetaan hänen arvosanansakin. Mahdollisuuksia vaikuttaa tietokantoihin on muitakin edellisen ollessa vain esimerkki.

Tietokantamallit kuvastavat tietokannassa olevan tiedon rakennetta. Muuttujat ovat samannimisiä sekä taulussa, että mallissa. Tietokantamallin on vastattava tietokannan taulun skeemaa ja mallissa on määriteltävä muuttuja vähintään jokaiselle taulussa vaaditulle kentälle. Näkymämallissa otetaan huomioon näkymien tarvitsemat ominaisuudet ja tiedot. Näitä malleja käytetään nimensä mukaisesti näkymissä, mutta myös tietokantamallia on mahdollista käyttää näkymän mallina.

Entity Frameworkissa on käytäntö, jossa on määritetty ominaisuuden nimeltä Id, olevan pääavain. Mikäli sen nimistä ominaisuutta ei löydy, on käyttäjän vastuulla määrittää pääavain. (Microsoft n, 2019.)

2.3.2 Kyselyt

Kyselyt tietokantoihin voi tehdä käyttäen Microsoftin LINQ-teknologioita (Language Integrated Query).. Saadessaan kyselyn, Entity Framework Core lähettää sitä vastaavan kyselyn tietokantaan. LINQ käyttää tietokantakontekstia sekä entiteettejä, jotka viittaavat tietokannan tauluihin ja riveihin. Tietokantamoottori kääntää kyselyn tietokannan omalle kielelle. Tämä kysely suoritetaan, ja palautuksena tulee arvot, jotka vastaavat kyselyä. Mikäli kysely on seurantakysely, tarkistaa EF Core yksitellen, onko jotkin entiteeteistä merkattu muutosseurantaan. Entiteetin ollessa muutosseurannassa, palautetaan jo olemassa olevan entiteetti, muuten luodaan uusi ja kirjataan se muutosseurantaan. Jos taas kysely ei ole seurantakysely, luodaan uusi entiteetti jokaisesta. Tulokset varastoidaan muistiin, jotta ei tarvitse lähettää samaa kyselyä tietokantaan useaan kertaan, tehostaen sovelluksen nopeutta. (Microsoft j, 2019.)

Tiedon tallennus tai muokkaus tapahtuu käyttäen luotua tietokantakontekstia. Jokaisella konteksti-instanssilla on muutosseuranta, joka seuraa muutoksia ja pitää ne muistissa, kunnes ne tallennetaan erikseen kutsuttavalla tallennuskomennolla. Tietokanta vastaa muutoksen tulkkauksesta ja suorittaa oikeat operaatiot. (Microsoft d, 2020.)

2.4 LINQ

LINQ on Microsoftin teknologia, jonka tarkoituksena on tietokantakyselyiden kytkeminen koodiin. Täten kyselyissä voi hyödyntää esimerkiksi C# ominaista vahvaa tyypitystä. Käytössä on myös C# avainsanat sekä operaattorit. LINQ on tuettuna C#:ssa sekä Visual Basicissa. LINQ-komponentteihin kuuluu LINQ to SQL, LINQ to XML, LINQ to Entities, joka on tarkoitettu Entity Frameworkin kanssa työskentelyyn sekä LINQ to Objects, joka on muistissa olevien kokoelmien, kuten listojen ja taulukoiden, kanssa toimiva komponentti. (Microsoft m, 2015.)

Suurin osa LINQ:n metodeista suoritetaan kokoelmille, jotka implementoivat tyyppiä IEnumerable tai IQueryable. Ero edellä mainittujen välillä on tietokantakyselyissä suorituksen ajankohdassa ja tavassa. IQueryable-tyyppisessä kyselyssä suodatus ja kysely suoritetaan tietokannassa**.** IEnumerable-tyyppisessä kyselyssä lähetetään kysely tietokantaan, joka palauttaa kaikki tiedot tauluista, ja suorittaa kyselyn mukaiset suodatukset

palautettuun tauluun. IQueryable perii IEnumerable luokan, joten IQueryablella voi tehdä samoja asioita kuin IEnumerablella. Visual studio automaattisesti valitsee tietokantakyselyihin IQueryablen, ja muistissa oleviin kokoelmiin IEnumerable-tyyppisen muuttujan. (Siddiqui 2020.)

Kyselyt LINQ:ssa koostuvat kolmesta eri kohdasta: datalähteen hankinnasta, kyselyn luonnista sekä kyselyn suorituksesta. Datalähteen hankinnassa määritetään haun kohde. Haun voi kohdistaa kokoelmiin, jotka käyttävät IEnumerable-rajapintaa, kuten listoihin, tietokantoihin ja XML-dokumentteihin. IEnumerable-rajapintaa tarvitaan LINQ-metodien kanssa, jotka ovat jatkeita IEnumerablessa muodostetuille metodeille, kuten ”OrderBy”. Kyselyn luonnissa kirjataan itse kysely, määrittäen mitä, mistä ja millä ehdoilla haetaan. Viimeisenä on kyselyn suoritus, joka tapahtuu joko viivästytetysti tai heti. Viivästetysti suorittaessa vasta kyselymuuttujaa iteroitaessa kysely suoritetaan. Iterointi tapahtuu esimerkiksi foreach-komennossa, kuten kuvassa 8. Jokainen tulos asetetaan item-muuttujaan yksi kerrallaan ja lisätään rivillä 42 alustettuun listaan, kunnes kaikki tulokset on käyty läpi. Toinen vaihtoehto on pakottaa kyselyn suoritus heti. Tämä on toteutettavissa esimerkiksi Count, Max, Average ja First-komennoilla. Nämä komennot suorittavat foreach-komennon, koska niitä käyttävän kyselyn on iteroitava tulokset palautusta varten. Palautuksena tulee yksittäinen arvo komennon mukaan. Myös listaan tai taulukkoon tulosten asettaminen pakottaa kyselyn suorittamisen heti. (Microsoft o, 2015.)

Tulosten suodattamiseen voi käyttää useita suodattimia kerralla. Suodattimet asetetaan where- tai vastaavaan lauseeseen. Valinta tapahtuu select-lauseessa. Tulokset voi valita kokonaisina objekteina, tai ottaa vain tietyn ominaisuuden tuloksista, esimerkiksi nimen. Kuvassa 8 tämä on määritelty riviltä 43 alkaen ”yhtä suuri kuin” -merkin jälkeen riville 45 asti. _dbContext-datakontekstissa on määritetty entiteetti nimeltä Items, josta haetaan kaikki esineet, joiden nimi on ”Key”, ja asetetaan ne joukkomuuttujaan ”chosenItem”. Muuttujassa ”query” on nyt edellä luetellun kyselyn lopputulos. Uuden objektin luominen ”select”-komennossa on mahdollista, ja siihen voi valita useita ominaisuuksia.

```
42          List<Item> listOfItems = new List<Item>();
43          var query = from chosenItem in _dbContext.Items
44                      where chosenItem.Name == "Key"
45                      select chosenItem;
46
47          foreach (var item in query)
48          {
49              listOfItems.Add(item);
50          }
51
```

Kuva 8. Esimerkki LINQ-kyselystä ja sen suorituksesta

LINQ:ssa on useita funktioita datan hakemiseen, muokkaamiseen, sekä järjestämiseen. Any, All ja Contain-operaatiot palauttavat kaikki tulokset, jotka mahtuvat kriteerien sisään, esimerkiksi, jos on lista nimistä ja etsitään tiettyä kirjainta. Any-operaatio palauttaa tuloksen, jos yksikin kirjain täsmää, All-operaatio palauttaa kaikkien kirjaimien täsmätessä ja Contain-operaatio, jos nimi sisältää kirjaimen. First-operaatio hakee ensimmäisen tuloksen, paitsi, jos haun kohteena oleva taulu on tyhjä, jolloin sovellus aiheuttaa poikkeuksen, mutta FirstOrDefault-operaatio palauttaa null: in. Single- sekä SingleorDefault-operaatiot toimivat taas siten, että ne palauttavat ainoan tuloksen, joka vastaa kyselyä. Mikäli tuloksia on enemmän kuin yksi tai ei yhtään, Single-komento aiheuttaa poikkeuksen ja SingleorDefault-operaatio aiheuttaa poikkeuksen vain, jos tuloksia on useampia. (Microsoft c, 2015.)

Lajitella voi yhden tai useamman ominaisuuden perusteella. Esimerkiksi tuloksia voi lajitella ensisijaisesti sukunimen perusteella ja toissijaisesti iän perusteella. Järjestykseen voi vaikuttaa joko orderby-operaatiolla tai järjestämällä tulokset jonkun omaisuuden mukaan. Groupby-operaatiolla voi luokitella tulokset eri ryhmiin ominaisuuden perusteella, kuten asuinpaikan tai vastaavan. Kyselyn tulokset voi asettaa muuttujaan ”into”-komennolla, jonka jälkeen voi kohdistaa jatkokyselyt muuttujaan tallennettuihin tuloksiin. Lopputulos voidaan myös järjestää nousevassa tai laskevassa järjestyksessä. (Microsoft s, 2015.) Liitoshaut suoritetaan ”join”-komennolla, jolloin tietoa voidaan hakea useammasta taulusta samalla kertaa. Koska kyselyt voidaan tallentaa muuttujaan, on mahdollista suorittaa sama kysely uudestaan tekemättä uutta erillistä muuttujaa, tai kyselyä, käyttäen jo olemassa olevaa kyselymuuttujaa uudestaan tai muokattuna. (Microsoft c, 2015.)

Syntaksillisesti LINQ:sta on kaksi eri vaihtoehtoa: metodi- sekä kyselysyntaksi. Kyselysyntaksi on visuaalisesti lähellä perinteistä SQL-kieltä ja metodisyntaksi näyttää C#-koodilta. Vaikka vaihtoehdoista voi valita preferenssin mukaan mieleisensä, on molemmilla syntakseilla omat etunsa tietyissä tilanteissa. Kyselysyntaksilla on mahdollista hakea monesta lähteestä yhdessä lauseessa. Kyselysyntaksilla tehdyt haut kuitenkin käännetään suorittaessa metodikutsuiksi, jotta CLR voi suorittaa kyselyt. Metodisyntaksilla on mahdollista käyttää LINQ:n metodeita, joita kyselysyntaksilla ei voi käyttää, kuten Count- tai Max-operaatioita. Kyselyt on mahdollista suorittaa synkronisesti sekä asynkronisesti. (Microsoft q, 2015).

Where-lausekkeen kanssa metodisyntaksia käytettäessä on joskus tarpeen käyttää lambda-lausekkeita. C#-kielessä lambda-lauseke alkaa ”=>”-merkin jälkeen. Esimerkiksi kuvassa 9 on metodisyntaksissa käytetty lambda-lauseketta rajoittamaan tulokset vain, mikäli verrattava data vastaa muuttujaan asetettua arvoa. (Microsoft q, 2015.)

```
var skillQuerySyntax = from m in _context.UserSkills
                       where m.SkillName == skillName
                       select m;

var skillMethodSyntax = _context.UserSkills.Where(x => x.SkillName == skillName);
```

Kuva 9. LINQ-syntaksia

2.5 Reititys

Reitityksessä HTTP-pyynnöt ohjataan sovelluksessa suoritettaviin päätepisteisiin. Päätepisteet ovat sovelluksessa suoritettavaa koodia, joka suoritetaan pyynnöstä. Reitityksen ASP.NET Core -sovelluksissa voi toteuttaa monella eri tavalla esimerkiksi ohjaimissa, Razor-sivuilla tai delegaateilla. (Ryan ym, 2020.)

```
58          app.UseRouting();
59
60          app.UseEndpoints(endpoints =>
61          {
62              endpoints.MapControllerRoute(
63                  name: "default",
64                  pattern: "{controller}/{action=Index}/{id?}");
65          });
```

Kuva 10. Reitityksen määritelmä ASP.NET Core sovelluksen Startup.cs-tiedostossa

Kuvassa 10 on käytetty IApplicationBuilder-objektin funktiota UseRouting, joka lisää väliohjelmiston reittivertailuun. Ideana on, että tutkitaan tullutta pyyntöä, ja etsitään pyyntöä vastaavin päätepiste. UseEndpoint-funktio lisää päätepisteiden suorituksen väliohjelmistoksi. Vielä MapControllerRoute-funktiossa määritetään ohjainten funktiot päätepisteiksi sekä reitin rakenne, kuten kuvan 10 esimerkissä on määritetty nimi: default ja UR-osoitteen rakenne (pattern). URL-osoitteen rakennetta voi myös määrittää muuttuja-arvoparelle vaadittavia ominaisuuksia, kuten tietotyyppiä tai enimmäispituutta. Määrittäminen tapahtuu joko käyttäen säännöllisiä lausekkeita (Regular expressions) tai rajoituksia, jotka asetetaan attribuuteissa. (Nowak ym. 2020.)

```
60          [Route("Game/Get/{id}")]
            0 references
61          public async Task<string> Get(int itemId)
```

Kuva 11. Reitityksen ohjaimessa.

Kuvassa 11 on määritettynä reititys ohjaimen funktiossa. Mikäli URL vastaa attribuuttiin syötettyä määritelmää, ohjataan pyyntö sille funktiolle. Esimerkissä on ohjain, sen funktio, sekä id-muuttuja, jota tarvitaan suorituksessa.

Oletuksena URL-osoitteessä on ensin ohjaimen nimi, sitten metodin nimi, jota seuraa metodin tarvitsemat muuttujat, kuten Id. Päätepisteen muoto on kehittäjän muokattavissa. Tilanteissa, joissa reitittäminen voi olla moniselitteistä, esimerkiksi on kaksi samannimistä metodia, ”get” ja ”post”-attribuuteilla erotteleminen on mahdollista. Näin ollen pyynnön tyypin mukaan suoritetaan toinen metodeista. (Nowak, Ryan yms. 2016.)

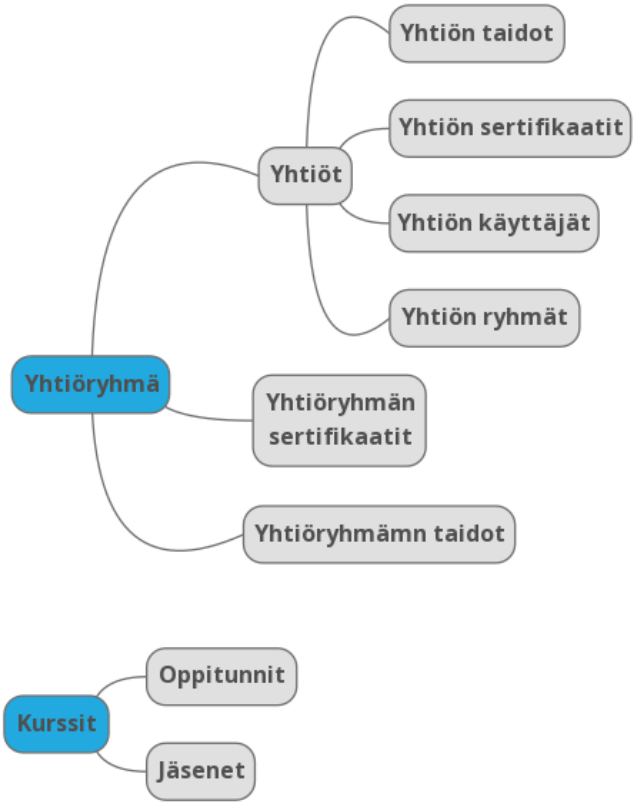
3 PROJEKTI/OPPIMISPORTAALI

3.1 Tavoite

Sovelluksen tarkoitus on tukea ja ylläpitää käyttäjien taitoja ja niiden kehitystä. Käyttäjiä voidaan seuloa ja ilmoittaa kursseille tarpeiden mukaan. Sovelluksessa on neljä roolia: superadmin, yhtiöryhmänvalvoja, järjestelmänvalvoja sekä käyttäjä. Superadminilla on oikeudet kaikkialle sovelluksessa. Yhtiöryhmänvalvojan oikeudet ulottuvat yhtiöryhmää koskeviin asioihin. Järjestelmänvalvojalla oikeudet ovat yhtiön sisällä. Käyttäjän oikeudet on rajattu omiin tietoihin.

Sovelluksessa on erilaisia taitoja ja sertifikaatteja, joita voidaan myöntää käyttäjille. Taitojen ja sertifikaattien lisäksi sovelluksessa on mahdollista etsiä ja ilmoittautua kursseille kehittämään haluttua osaamista. Kursseille määritetään erikseen taidot, joita niiden on tarkoitus kehittää, mahdollinen tavoitetaso sekä suositeltava lähtötaso. Kurssit eivät ole yhtiökohtaisia.

Käyttäjiä on mahdollista hakea suodattimia käyttäen. Suodattimia ovat taidot, sertifikaatit, yritykset ja ryhmät. Roolista riippuen henkilöistä näytetään eri määrä tietoa. Järjestelmänvalvojat näkevät oman yhtiönsä sisältä henkilön kaikki tiedot, mutta yhtiöryhmän muista yhtiöistä löytyvät näkyvät anonyymisti. Yhtiöryhmänvalvojat näkevät yhtiöryhmän käyttäjät ja kaikki heidän tietonsa. Kuten kuvassa 12 näkyy, yhtiöryhmässä on yhtiöt, taidot sekä sertifikaatit. Yhtiöillä on yhtiöryhmän taidot ja sertifikaatit, mutta myös mahdollisuus olla omia yrityskohtaisia taitoja sekä sertifikaatteja. Yhtiöiden sisällä on mahdollista jakaa käyttäjiä myös ryhmiin.



Kuva 12. Yhtiöryhmäkuvaaja

3.2 Toiminnan kuvaus

Sovelluksessa yhtiöt voivat kuulua yhtiöryhmiin. Yhtiöryhmässä voi määritellä yhteiset taidot ja sertifikaatit, jotka ovat käytössä jokaisessa yhtiöryhmään kuuluvassa yhtiössä. Toisten yhtiöryhmien tietoja ei näe missään tapauksessa. Yhtiön sisällä on käyttäjiä, sekä ryhmiä, joihin voi lisätä käyttäjiä. Ryhmille asetetaan taitoja ja niille tavoitteita. Yhtiön järjestelmänvalvojat voivat lisätä yhtiössä olevan käyttäjän ryhmään tai ilmoittaa hänet kurssille.

3.3 Roolit

Rooli määritetään käyttäjän luonnissa, mutta sitä voi myöhemminkin muokata muiden käyttäjätietojen ohella. Käyttäjällä voi olla eri rooli eri yhtiössä, sekä se voi kuulua useisiin yhtiöryhmän yhtiöihin.

Superadminin oikeudet yltävät kaikkialle sovelluksessa. Jokaisen yhtiöryhmän ja yhtiön tiedot ovat superadminien muokattavissa sekä saatavissa. Vain ja ainoastaan superadminit voivat luoda yhtiöryhmiä.

Järjestelmänvalvoja voi lisätä yhtiökohtaisia taitoja ja sertifikaatteja, jotka näkyvät vain yhtiössä, johon ne on luotu. Jotta näitä voi käyttää yhtiöryhmässä, pitää ne siirtää yhtiötasolle superadminin tai yhtiöryhmän järjestelmänvalvojan toimesta. Uusien käyttäjien lisääminen yhtiöön tapahtuu myös järjestelmänvalvojan toimesta. Lisätessä käyttäjää, järjestelmänvalvoja voi valita, mihin kuulumistaan yhtiöistä käyttäjä laitetaan. Tarkennettu haku on järjestelmänvalvojan käytettävissä, mutta hän näkee vain oman yhtiön käyttäjät.

Yhtiöryhmän järjestelmänvalvoja on oikeuksiltaan lähes samanlainen järjestelmänvalvojan kanssa, mutta voi sen lisäksi muokata myös yhtiöryhmän tasolla taitoja, sertifikaatteja sekä nimeä. Tarkennetussa haussa tämän roolin edustaja näkee tulokset kaikista yhtiöryhmän yhtiöistä.

Peruskäyttäjä voi arvioida omia taitotavoitteitaan sekä osaamistaan. Käyttäjä voi merkata itselleen sertifikaatteja, mikäli hänellä on sellaisia muualta hankittuna. Samalla luodaan tietokantaan uusi sertifikaatti käytettäväksi tilanteissa, joissa jollain muullakin henkilöllä on sama sertifikaatti. Käyttäjä ei voi kuitenkaan antaa itselleen jo olemassa olevia sertifikaatteja. Käyttäjältä on piilotettu muiden, sekä oman yhtiön käyttäjien tiedot, omat pois lukien.

3.4 Tavoitetasot

Tavoitteita taidoille voi tulla kolmesta eri paikasta: käyttäjältä itseltään, yhtiöltä, sekä yhtiössä olevilta ryhmiltä, joihin käyttäjä kuuluu. Käyttäjän asettama tavoite on kuitenkin yliajettavissa, mikäli yhtiö tai ryhmä, johon käyttäjä kuuluu, vaatii enemmän. Kaikki taitotavoitteet käyttävät samaa 0–5 -asteikkoa kuin taidotkin.

Myös kursseilla on taitotavoitteet, joihin käyttäjä pyritään nostamaan. Tavoitteiden lisäksi on myös lähtötaso taidoille, jotka kuvastavat suositeltua tasoa, jolla taidon tulisi olla ennen kurssille osallistumista. Lähtötavoitetta ei ole pakko määrittää.

3.5 Tietokannan kuvaus

Sovelluksessa on käytössä paljon resursseja, joilla on keskenään monen suhde moneen-yhteys, joten tietokannan tulee vastata tähän tarpeeseen. Esimerkiksi yhtiönkäyttäjillä on oma taulu, johon laitetaan id-arvot yhtiöltä sekä käyttäjältä.

Käyttäjien ja roolien taulut on luotu ASP.NET Identityn pohjalta, joskin pienillä muutoksilla. AspNetUsers-tauluun, joka sisältää käyttäjien tiedot, on lisätty Company-sarake, joka sisältää käyttäjän aktiivisen yrityksen id:n, sarake käyttäjän läsnäololle, sekä etu- ja sukunimille omat sarakkeet. Loput tauluista ovat itse luotuja.

Sovelluksessa olevat yhtiöryhmävalvojat on tehty superadminista siten, että ”Company Members”-taulussa on erillinen rivi: ”CompanyGroupAdmin”, joka on binääriarvo. Mikäli luku on 1, tarkoittaa että käyttäjä on yhtiöryhmänvalvoja.

4 BACKENDIN TOTEUTUS

4.1 Tietokanta

Sovelluksessa on kaksi eri tietokantakontekstiluokka. Toiseen osaan kuuluu kaikki ASP.NET Identityn omat, nimeltään IdentityDataContext ja toiseen loput, itse luodut taulut, GeneralDataContext. Kaikki tietokantoihin kohdistuvat pyynnöt suoritetaan LINQ:lla.

4.1.1 Yritysryhmät ja yritykset

Yhtiöryhmää luodessa voi määrittää siihen kuuluvia yhtiöitä, mutta se ei ole pakollista, kuten ei myöskään ole pakko määrittää yhtiölle yhtiöryhmää yhtiön luonnissa. Yhtiöryhmän muokkaaminen on sallittua vain yhtiöryhmänjärjestelmänvalvojalle tai superjärjestelmänvalvojalle, mutta yhtiön tietoja voi muokata myös yhtiön järjestelmänvalvojat. Mikäli yritysryhmä poistetaan, poistuu poistetun yhtiöryhmän jäsenyydet.

Kuten edellisessä kappaleessa mainittiin, yhtiöryhmillä on omat taidot ja sertifikaatit, jotka ovat käytössä ryhmään kuuluvilla yhtiöillä. Yhtiöillä voi olla myös omia taitoja tai sertifikaatteja, jotka ovat vain käytössä siinä yhtiössä. Jos yhtiön järjestelmänvalvoja luo taidon tai sertifikaatin, sidotaan taito suoraan siihen yhtiöön, eikä yhtiöryhmään. Eli käytännössä ”Company Group Skills”-taulussa olevaan ”Company Id”-sarakkeeseen merkataan sen yrityksen id, mihin resurssin luoja kuuluu luomishetkellä. Mikäli resurssin luoja on superadmin tai yhtiöryhmänjärjestelmänvalvoja, tulee resurssista suoraan yhtiöryhmän resurssi. Sertifikaatit ovat poikkeus, sillä peruskäyttäjä voi luoda sertifikaatin. Käyttäjän luoma sertifikaatti myönnetään sekä käyttäjälle, että yhtiön käyttöön. Resurssin voi siirtää myöhemmin koko yhtiöryhmän käyttöön tarvittaessa.

Käyttäjien on mahdollista kuulua useampaan yritykseen ja jokaisessa yrityksessä käyttäjällä ei välttämättä ole sama rooli. Käyttäjän rooli on kuitenkin tietokannassa vain yhdellä rivillä. Yhtiötä vaihtaessa katsotaan ”Company Members”-taulusta käyttäjän rooli Id, joka asetetaan myös ”User Roles”-taulussa käyttäjän rooliksi. Joka kerta kun yhtiö vaihtuu, tehdään tämä prosessi. Käyttäjän kirjautuessa sisään, tarkistetaan rooli.

Käyttäjän on kuuluttava johonkin yritykseen. Käyttäjää luodessa, superadmin voi asettaa käyttäjän haluamaansa yritykseen. Yhtiöryhmäadmin voi asettaa käyttäjän vain kuulumiensa yhtiöryhmien yrityksiin ja yhtiön järjestelmänvalvojan luoma käyttäjä lisätään automaattisesti hänen aktiiviseen yritykseensä.

Yritysryhmää poistettaessa tarkastetaan tietokannasta, kuuluuko yritysryhmään yrityksiä ja poisto estetään, mikäli yritysryhmään kuuluu yhtään yritystä. Yhtiöillä on samanlainen toimintamalli, mutta koskee käyttäjiä.

4.1.2 Kurssit

Kurssit ovat ryhmäkohtainen resurssi, joiden tarkoituksena on kehittää niihin osallistuvien käyttäjien taitoja. Kurssien suorituksen jälkeen voi järjestelmänvalvoja käydä antamassa arvosanan kurssin suorituksesta. Kurssin suoritus ei kuitenkaan nosta käyttäjän taitoa, vaan se on muutettava erikseen.

Kursseilla on oppitunnit, jotka lisätään kurssin luonnin jälkeen. Oppitunneille voi määrittää ajan, paikan sekä aiheen. Oppitunnit ovat liitetty kursseihin kurssin Id-arvolla.

4.1.3 Ryhmät (Groups)

Ryhmät ovat yrityskohtaisia ja ne asettavat tavoitteita käyttäjille taitojen suhteen. Ryhmällä on taidot sekä niille omat taitotavoitteet, joita seurataan ryhmään kuuluvien käyttäjien taitotason keskiarvolla. Tietokannasta otetaan jokaisen ryhmään kuuluvan käyttäjän uusin merkkaus ja lasketaan niiden keskiarvo.

4.2 Käyttäjä

Käyttäjällä on kytköksiä sertifikaatteihin, taitoihin, ryhmiin, yhtiöihin, kursseihin sekä oppitunteihin. Jokaisessa edellä mainituista ominaisuuksista, on monen suhde moneen tilanne. Käyttäjän oikeuksien hallintaan on käytetty Entity Frameworkin Identityn vakio asetuksia, joka sisältää vakio tokenit, vakio UI:n (User Interface) sekä evästeautentikaation.

Autentikointi tapahtuu Microsoftin tarjoamalla ohjelmistolla, joka lisätään IApplicationBuilder-interfaceen ”startup.cs”-tiedostossa. Sovelluksessa on autentikaatio-eväste, johon annetaan käyttäjän identiteetti kirjautumisen yhteydessä.

Valtuuttaminen on roolipohjaista ja on määritelty ohjaimissa metodikohtaisesti. Suurinta osaa metodeista ei voi käyttää, ellei ole kirjautunut sisään ja vähintään user-tason käyttäjä pois lukien sisäänkirjautumiseen tarvittavat metodit sekä etusivu. Erityistapauksia on, missä adminit eivät voi käyttää metodeita mutta Company Group adminit voivat, kuten Company Groupin taitojen tai sertifikaattien muokkaus.

Mikäli käyttäjä kuuluu useampaan yritykseen, voi vain yksi olla aktiivisena kerrallaan. Yrityksen voi vaihtaa halutessaan missä tahansa. Aktiivinen yritys on merkattu

käyttäjätauluun. Käyttäjän aktiivisuus on merkattu tietokantaan, jota säädetään kirjautuessa sisään sekä ulos.

Tarkennetussa haussa voi hakea taitojen, sertifikaattien, yhtiöiden ja ryhmien pohjalta ja yhdistää eri kriteereitä keskenään. Jokaista kriteeriä voi olla useampi kerralla esimerkiksi käyttäjää voi hakea kahden taidon ja sertifikaatin perusteella. Ohjaimessa jokaisella kriteerillä on oma suodatin, jossa tietokannasta haetaan id:arvon perusteella. Taidoissa on myös mahdollista antaa niiden minimi- sekä maksimiarvo. Ohjaimen puolella hakutoiminto on ketjutettu niin, että ensimmäisen suodattimen jälkeen otetaan lista talteen ja siitä seuraavissa suodattimissa karsitaan listasta pois käyttäjät, jotka eivät täytä ehtoja. Tämä jatkuu, kunnes viimeisen suodattimen jälkeen on hakuehdot täyttäneet käyttäjät selvillä. Tietokannasta otetaan kaikki suodattimen ominaisuuksiin täsmäävät käyttäjät, jonka jälkeen tulokset limitetään jo olevassa olevan listan kanssa, mikäli sellainen on. Jos ei ole, viedään lista sellaisenaan seuraavalle suodattimelle, jossa suoritetaan haut ja limitetään sen haun tulokset. Ohjaimen GET-metodissa tarkastetaan käyttäjän yhtiö sekä rooli, jotta ladataan käyttäjälle kuuluvat vaihtoehdot valittavaksi.

Käyttäjän taidot arvioidaan asteikolla 0–5. Taidoista tallennetaan jokainen arviointi, jotta voidaan seurata kehitystä. Koska taitojen kohdalla tallennetaan useat tulokset samalle käyttäjälle, pitää valita jokaisen käyttäjän kohdalta uusin taitoarviointi halutusta taidosta. Taidoista tallennetaan myös päivämäärä, joten käyttäjän nykyinen taitotaso on mahdollista hakea tuoreimmalla päivällä.

4.3 Kieli

Oppimisportaali on tämän työn aikana kaksikielinen. Vaihtoehtoina on suomen sekä englannin kielet. Kielten välillä voi vaihtaa missä tahansa sovelluksen osassa.

Kielen määritys on toteutettu evästeenä, joka muutetaan valinnan mukaan. Tästä syystä kieli pysyy samana myöhemmälläkin käyttökerralla. HTTP-vastauksena lähtee evästeen nimi, arvo sekä asetukset, joissa on määritetty evästeen kesto. Kieltä vaihtaessa lähetetään ohjaimelle sen hetkinen URL-osoite, joka ladataan uudella kielellä. Molemmilla kielillä on omat resurssit jokaiselle ohjaimelle.

4.4 Reititys

Sovelluksen reititys on määritelty startup.cs tiedostossa. Applikaatiossa on yksi reititys tyyppi, jonka muoto on: kulttuuri, ohjain, toiminto, muuttujat. Kultturi osa määrittää applikaatiossa käytetyn kielen. URL-osoitetta manuaalisesti muokkaamalla voi vaihtaa kieltä,

mutta se ei muutu, mikäli vaihtaa kieltä sovelluksen sisällä. Sovelluksessa ei käytetä Ohjaimensisäistä reititystä ollenkaan.

URL-osoitteessa on suoritettava ohjain sekä funktio, johon suoritus ohjataan sekä tarpeelliset muuttujat. Funktiossa tehdään tarvitut toiminnot, jonka jälkeen palautuksena lähtee näkymämallin tyyppinen objekti, johon on asetettu halutut arvot, joita tarvitaan sivun näytössä. Useimmilla funktioilla on GET sekä POST-variantti, joissa yleensä GET-metodit lataavat toivotun sivun ja POST-metodit vaikuttavat tietokantaan käyttäjän antamilla tiedoilla.

4.5 Rajapintojen testaus

Verkkorajapintojen testaus on toteutettu siihen tarkoitetulla Postman-sovelluksella. Testaamisessa pitää ottaa huomioon autentikointievästeet, joita sovellus käyttää. Nämä evästeet täytyy antaa Postmanille, jotta HTTP-pyyntöjä voi suorittaa. Evästeet voidaan manuaalisesti ottaa selaimen kehittäjän työkaluista tai käyttäen ulkopuolisia sovelluksia kuten Postmanilla Interceptor-laajennusta.

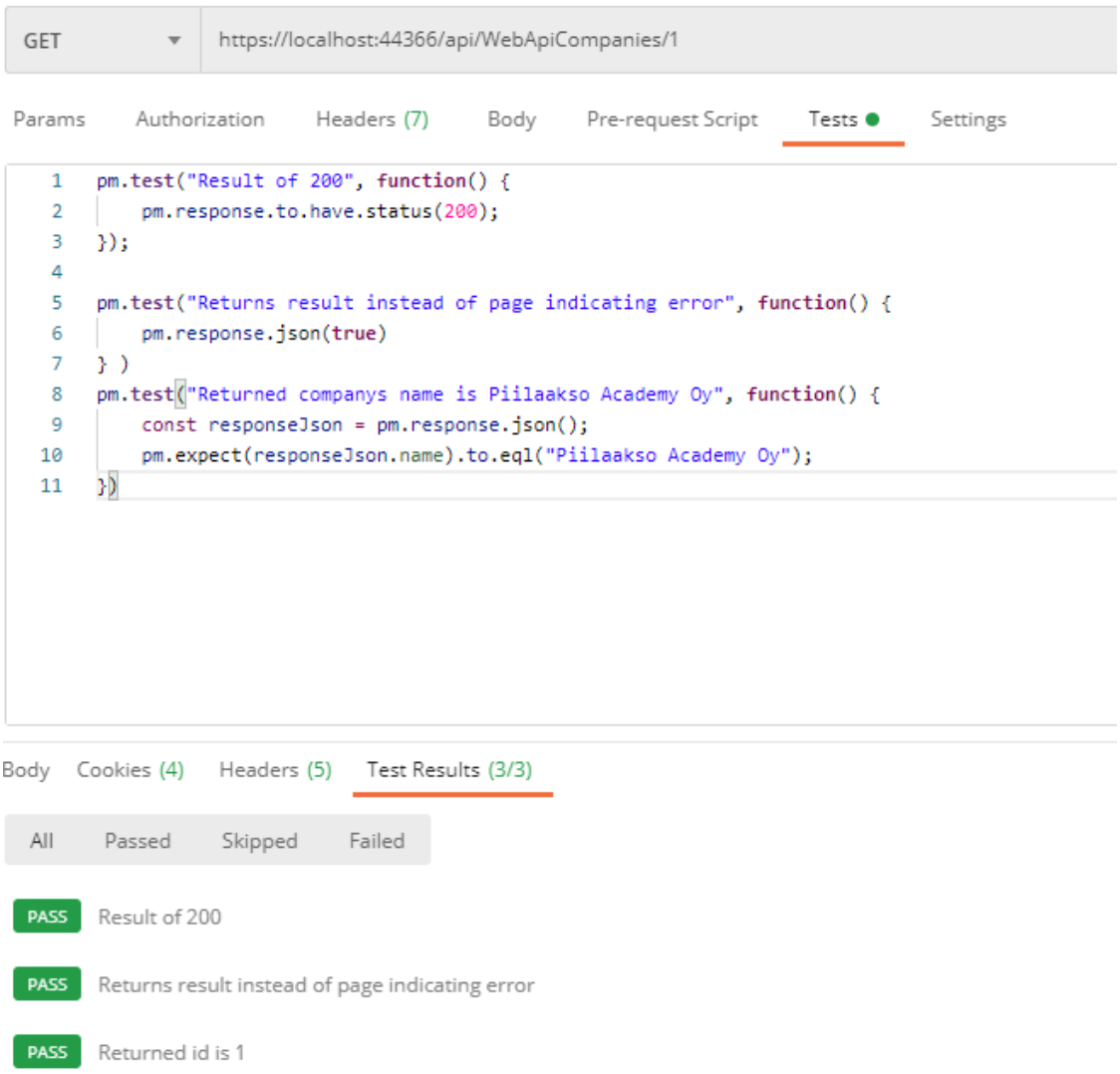
Ohjaimien suoraan testaaminen Postmanilla palauttaa pyydetyn verkkosivun. Tämä on tehty ohjaintasolla niin, että tarvittavien toimenpiteiden jälkeen palautetaan näkymä, ja näkymän mallin mukainen muuttuja sen kanssa, mikäli tarpeellista. Tietokannan toimintojen testaaminen voidaan suorittaa myös erillisillä WebApi-ohjaimille osoitetuilla kutsuilla. WebApi ohjaimelta onnistuneessa kutsussa tulee palautuksena JSON-muotoinen objekti.

GET-pyynnöissä haetaan halutut resurssit. Evästeissä on määritetty käyttäjä, jonka mukaan määritetään ehdot haulle. Esimerkiksi käyttäjiä hakiessa haetaan saman yhtiön, yhtiöryhmässä vain yhtiöryhmänvalvojat tai järjestelmänvalvoja tason käyttäjät saavat palautuksena listan yhtiöistä ja niiden tiedoista. GET-pyynnöistä on kahta eri tyyppiä, toinen hakee kaikki johonkin ryhmään tai vastaavaan kuuluvat ja toinen hakee yksittäinen käyttäjän tai ryhmän. Jälkimmäisen mukana annetaan halutun resurssin Id-arvo. DELETE-pyyntö toimii samalla tavalla kuin jälkimmäinen GET-pyyntö, mutta nimensä mukaisesti poistaa haetun resurssin.

POST-pyynnössä luodaan resurssi, joten on tilannekohtaista, mitä tarvitsee antaa tiedoksi. Postmanissa pitää antaa resurssikohtaiset tiedot body-välilehteen JSON-muodossa. Esimerkiksi yhtiötä määrittäessä tarvitaan nimi. Id annetaan tietokannassa, joten sitä ei tarvitse määrittää millekään resurssille erikseen. Luomisen jälkeen palautuksesta kopioidaan luodun resurssin Id postmanin ympäristömuuttujiin, jotta sitä voidaan käyttää muissakin pyynnöissä.

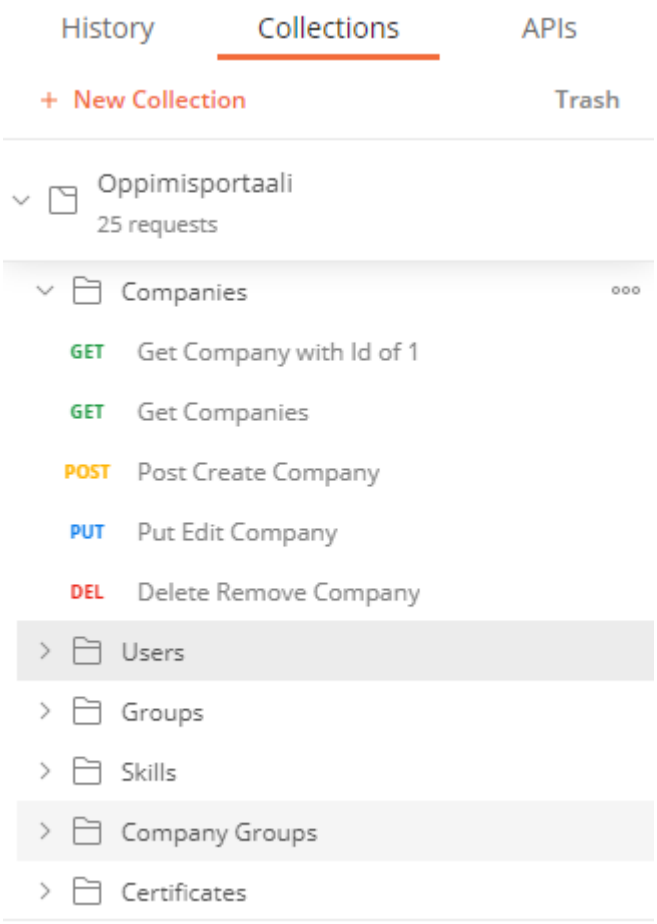
Muokkaamisen testaukseen PUT-pyynnössä tarvitsee antaa Id URL-osoitteessa sekä resurssin tarvitsemat tiedot body-välilehdessä. Myös Id tulee antaa, jotta tietokantaan muokataan oikeaa riviä. Ohjaimessa verrataan annettuja id-arvoja, ja mikäli ne täsmäävät ja tietokannasta löytyy sitä vastaava tulos, muokataan resurssia. Muussa tapauksessa tulee virhe numero 400 eli vääränlainen pyyntö.

Jokaiseen eri pyyntöön on myös asetettu eri testejä varmistamaan, että jokainen pyyntö ei pelkästään mene perille, vaan saa myös aikaan toivotun tuloksen. Testit on kirjoitettu JavaScriptillä. Testille annetaan ensin nimi, jonka jälkeen määritellään läpäisykriteeri. Kuvassa 13 on esitelty yhtiön GET-pyynnön testit. Riviltä 1 asti on märitelty kolme testiä, joiden on tarkoitus testata oikeaa vastauskoodia, palautuksen tyyppiä, onko palautuksessa olevan yrityksen nimi oikea.



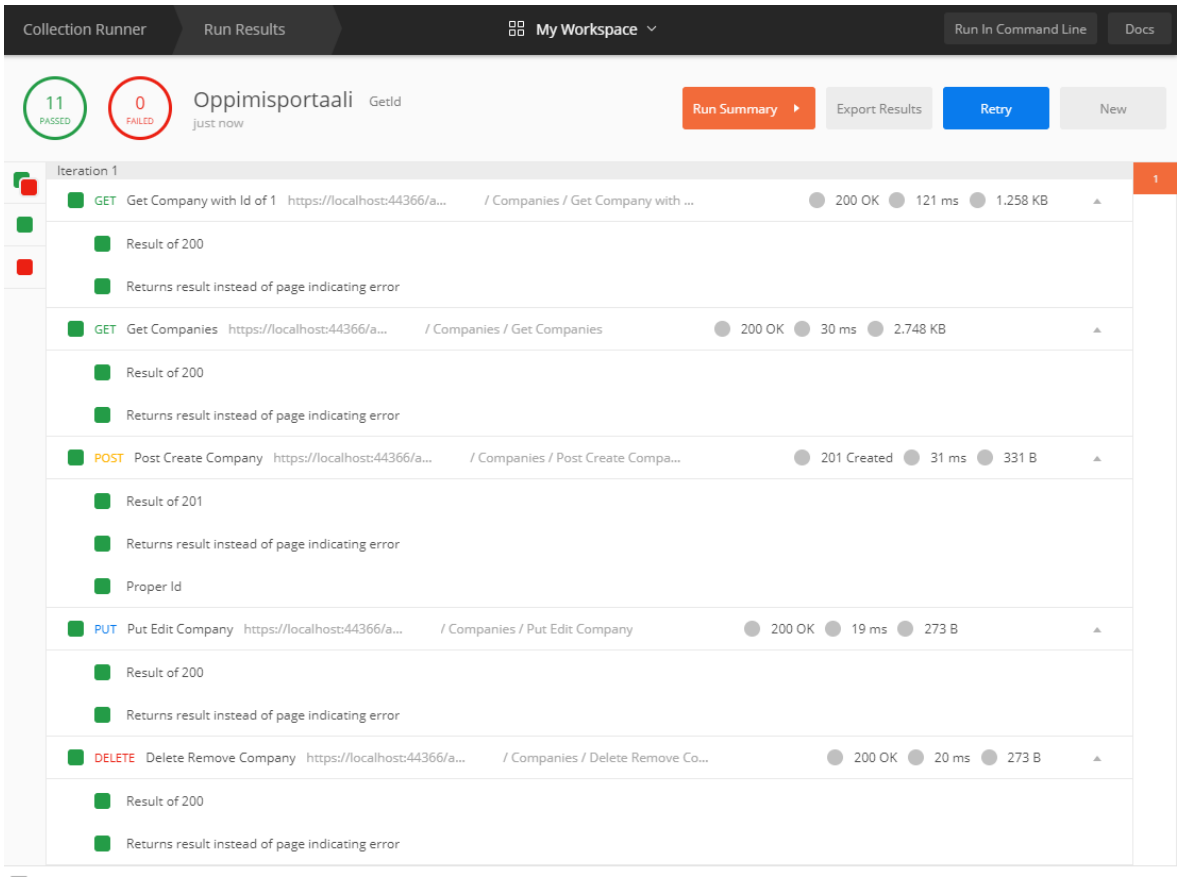
Kuva 13. GET-pyynnön testit

Testit suoritetaan Postmanin Collection Runnerissa, jossa on mahdollista suorittaa useita pyyntöjä kerralla. Pyynnöt on jaettu kansioittain sen mukaan, mihin resurssityyppiin niillä vaikutetaan.



Kuva 14. Pyyntö kokoelmat resursseittain

Collection Runnerin käytyä läpi jokainen pyyntö, näkyy tulokset pyyntöjen onnistumisista, sekä mahdollisten testien tuloksista. Kuvassa 14 näkyy kokoelma pyynnöistä, jotka on jaoteltu rajapintojen mukaan eri kansioihin, joiden sisällä on pyynnöt. Testauksen jälkeen Collection runner näyttää koosteen onnistuneista ja epäonnistuneista testeistä, sekä syyt miksi ne epäonnistuivat (Kuva 15).



Kuva 15 Postmanin Collection Runner yhden iteraation jälkeen

5 YHTEENVETO

Tavoitteena oli kehittää oppimisportaalista pilottiversio käyttäen ASP.NET Core 3.1. Tietokannan rakentaminen ja autentikoinnin kehittäminen sovellukselle olivat keskipisteenä.

Oppimisportaalin pilottiversiossa toimii lähes kaikki ominaisuudet sovelluksen vaatimalla tavalla. Lisättäviä ominaisuuksia kuitenkin on vielä jäljellä, joten nykyisiäkin ominaisuuksia voi olla tarpeen muokata. Tietokanta on myös tähän hetkiseen tilanteeseen toimiva kattaen kaikki tarpeet, mitä sovelluksella on. Jokaiselle resurssille löytyy taulut ja käyttäjillä sekä rooleilla on tarvittavat tiedot merkattu omissa tauluissaan. Autentikointi toimii kaikkialla sovelluksessa pois lukien yhtiöryhmänjärjestelmänvalvojan, joka tulee ottaa vielä huomioon kaikissa sovelluksen ohjaimissa ja metodeissa.

Valtuuttaminen olisi ehkä voitu toteuttaa käyttäen väitepohjaistavaltuuttamista. Sen avulla olisi myös voinut tarkemmin määritellä rajoituksia ohjaimille ja metodeille. Toki sovellus oli jo tehty pitkälle rooli pohjaisella autentikoinnilla, joten vaihtaminen ei ole välttämättä sen arvoista.

Nykyisen toiminnallisuuden hiominen ja loppujen ominaisuuksien tuominen on seuraavat askeleet sovelluksen kehittämisessä. Tällä hetkellä yhtiöryhmänjärjestelmänvalvojan rajoituksia ei ole otettu käyttöön kaikissa sovelluksen ohjaimissa. Sovelluksessa on myös muutama ongelma, jotka tulee ratkaista. Esimerkiksi käyttäjät voivat tarkastella toisten tietoja käyttämällä toisen henkilön id-arvoa URL-osoitteessa. Tämä on suuri tietoturvariski, joten sille pitää keksiä ratkaisu joko ohjain tasolla tai jollain toisella tavalla.

LÄHTEET

Arora, A. 2014. Introduction to ASP.NET Identity 2.0. Viitattu 12.12.2020. Saatavissa: https://www.c-sharpcorner.com/UploadFile/16101a/introduction-to-Asp-Net-identity-2-0/

Chowdhuri, S. 2016. ASP.NET Core Essentials. Viitattu 20.11.2020. Saatavilla: https://books.google.fi/books?hl=fi&lr=&id=oIFcDgAAQ-BAJ&oi=fnd&pg=PP1&dq=asp.net+core&ots=eBbIEBx2D-&sig=4XvU62FtV9N-s_XYjSS2j7epwOE&redir_esc=y#v=onepage&q=asp.net%20core&f=false

C# Corner. 2020. ASP.NET Core 2 – Architecture And Design Pattern Ideology. Viitattu 20.10.2020. Saatavissa: https://www.c-sharpcorner.com/article/asp-net-core-2-architecture-design-pattern-ideology/

Gazzo, J. 2014. ASP.NET Identity Framework. Viitattu 20.10.2020. Saatavissa https://www.teamscs.com/2014/11/asp-net-identity-framework/

Microsoft .NET Blog. 2019. .NET - A Unified Platform. Viitattu 20.10.2020. Saatavissa: https://devblogs.microsoft.com/dotnet/introducing-net-5/

Microsoft a. 2019. An introduction to NuGet. Viitattu 10.12.2020. Saatavissa: https://docs.microsoft.com/en-us/nuget/what-is-nuget

Microsoft b. 2020. Asynchronous Programming. Viitattu 25.01.2021. Saatavissa https://docs.microsoft.com/en-us/ef/core/miscellaneous/async

Microsoft c. 2015. Basic LINQ Query Operators (C#) Viitattu 1.12.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations

Microsoft d. 2020. Change Tracking in EF Core. Viitattu 03.02.20201. Saatavissa: https://docs.microsoft.com/en-us/ef/core/change-tracking/

Microsoft e. 2016. Claims-based authorization in ASP:NET Core. Viitattu 10.12.2020. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-3.1

Microsoft f. 2020. Common Language Runtime (CLR). Viitattu 25.11.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/standard/clr

Microsoft g. 2020. Creating and configuring a model. Viitattu 5.1.2021. Saatavissa: https://docs.microsoft.com/en-us/ef/core/modeling/

Microsoft h. 2020. Entity Framework Core. Viitattu 10.12.2020. Saatavissa: https://docs.microsoft.com/en-us/ef/core/

Microsoft i. 2019. Getting Started with EF Core. Viitattu 20.11.2020. Saatavissa: https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli

Microsoft j. 2020. How Queries Work. Viitattu 10.12.2020. Saatavissa: https://docs.microsoft.com/en-us/ef/core/querying/how-query-works

Microsoft k. 2020. Introduction to ASP.NET Core. Viitattu 20.01.2021. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0

Microsoft l. 2019. Introduction to ASP.NET Identity. Viitattu 10.12.2020. Saatavissa: https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

Microsoft m. 2015. Introduction to LINQ Queries (C#) Viitattu 20.11.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries

Microsoft n. 2019. Keys. Viitattu 25.01.2021. Saatavilla: https://docs.microsoft.com/en-us/ef/core/modeling/keys?tabs=data-annotations

Microsoft o. 2015. Language Integrated Queries (LINQ) Viitattu 20.11.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

Microsoft p. Overview of how .NET is versioned. Viitattu 20.11.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/core/versions/

Microsoft q. 2015. Query Syntax and Method Syntax in LINQ (C#). Viitattu 10.12.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/query-syntax-and-method-syntax-in-linq

Microsoft r. 2016. Role-based Authorization in ASP.NET Core. Viitattu 20.11. 2020. Saatavilla: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-3.1

Microsoft s. 2015. Walkthrough: Writing Queries in C# (LINQ). Viitattu 30.01.2021. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/walkthrough-writing-queries-linq

Microsoft t. 2020. What is .NET. Viitattu 25.11.2020. Saatavissa: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

Microsoft u. 2021. What is ASP.NET Core. Viitattu 07.02.2021. Saatavissa https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core

Microsoft v. 2020. Working with Nullable Reference Types. Viitattu 25.01.2021. Saatavissa: https://docs.microsoft.com/en-us/ef/core/miscellaneous/nullable-reference-types

Nowak R., Larkin K., Anderson R. 2020. Routing in ASP.NET Core. Viitattu 20.1.2021. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-3.1

Siddiqui. 2020. What is the difference between IEnumerable and IQueryable. Viitattu 20.01.2021. Saatavissa: https://www.tutorialspoint.com/what-is-the-difference-between-ienumerable-and-iqueryable-in-chash

Smith, S. 2017. Servers. Viitattu 7.12.2020 https://aspnetcore.readthedocs.io/en/stable/fundamentals/servers.html#

TutorialsTeacher. ASP.NET Core Overview. Viitattu 20.11.2020 Saatavissa: https://www.tutorialsteacher.com/core/aspnet-core-introduction