LAB University of Applied Sciences

SaaS-ohjelmiston luominen Angularilla

LAB-ammattikorkeakoulu
Tradenomi (AMK)
2022
Teemu Heikkinen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Heikkinen, Teemu	Opinnäytetyö, AMK	2022
	Sivumäärä	
	28	

Työn nimi

SaaS-ohjelmiston luominen Angularilla

Tutkinto

Tradenomi (AMK)

Toimeksiantajan nimi, titteli ja organisaatio

Aatu Suihkonen, Yrittäjä, Consulting Suihkonen Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli tarkastella SaaS-ohjelmiston kehittämistä Angular-ohjelmistokehyksen sekä Microsoftin ohjelmistokehitystyökalujen avulla. Tavoitteena oli myös selvittää, miten Angular ja Microsoftin työkalut toimivat yhdessä, ja saadaanko näitä yhdessä käyttämällä SaaS-ohjelmistolle hyvä lopputulos.

Opinnäytetyön oheksi luotiin yksinkertainen SaaS-sovellus, jonka avulla komponenttien yhdistämistä käytiin läpi. Esimerkkisovellus sisälsi tietokannan, frontendin sekä backendin, mitkä yhdistettiin toisiinsa.

Opinnäytetyön tuloksena saatiin selville, että Angularin ja Microsoftin ohjelmistokehitystyökaluja käyttämällä saadaan luotua hyvä SaaS-ohjelmisto. Saadaan myös selville, millaisiin asioihin sovelluksen suunnitteluvaiheessa kannattaa kiinnittää huomiota.

Asiasanat

Angular, .Net Core, SQL Server, SaaS, sovelluskehitys

Abstract

Author(s)	Type of Publication	Published
Heikkinen, Teemu	Thesis, UAS	2022
	Number of Pages	
	28	

Title of Publication

Building a SaaS-application with Angular

Name of Degree

Bachelor of Business Administration (UAS)

Name, title and organization of the client

Aatu Suihkonen, Entrepreneur, Consulting Suihkonen Oy

Abstract

The purpose of this thesis was to examine the development of SaaS software using Angular framework and software development tools by Microsoft. The goal is to clarify how Angular and Microsoft's tools work together and whether using these together can create a good result for the SaaS software.

Along with this thesis was created a simple SaaS-application that was used review the integration of the components. The example application contained a database, frontend and backend, which were all connected to each other.

As a result of the thesis, a good SaaS software can be created by using Angular and Microsoft's software development tools. The thesis addresses also what kind of things should be paid attention to during designing the application.

Keywords

Angular, .Net Core, SQL Server, SaaS, software development

Sisällys

1 Johdanto

Opinnäytetyöni aiheena on Angular-ohjelmistokehyksen ja Microsoftin ohjelmointityökalujen yhteistyön toimiminen ohjelmistokehityksessä. Tämän oheksi luon SaaS-ohjelmistosta valmiin MVP-ohjelmiston, eli Minimum Viable Productin, joka tarkoittaa vain tarvittavimmat ja pakollisimmat ominaisuudet sisältävää ohjelmistoa. MVP-ohjelmiston seurauksena ohjelmisto voidaan julkaista markkinoille ja sen käyttäjille. (Becker, 2020.) SaaS-ohjelmisto, englanniksi Software as a Service, on pilvessä sijaitseva ohjelmisto, johon se on välitetty applikaationa tai verkkoselaimen kautta. (Short & Spiller, 2022) Opinnäytetyössäni en voi toimeksiantajan pyynnöstä kertoa asiakasyrityksen liiketoiminta- tai palvelumalleista, tai mitä sovellus tulee tekemään ja millainen sen sisältö on. Luon opinnäytetyöhöni yksinkertaiset komponentit sovelluksesta, jotka yhdistän toisiinsa samalla tavalla kuten luotavassa MVP-ohjelmistossa.

Valitsin opinnäytetyöaiheeni, kun toimeksiantajallani oli tarve SaaS-sovellukselle, ja hänellä oli valmis idea siitä, millainen sovellus tulisi tulevaisuudessa olemaan. Opinnäytetyöni toimeksiantaja on Aatu Suihkonen yrityksestä Constulting Suihkonen Oy. Hän otti minuun ensimmäiseksi yhteyttä, jolloin hän kertoi minulle tarpeestaan sekä sovellusideastaan. Hän kertoi minulle suunnitelmastaan, ja aloin jo visioimiaan mielessäni erilaisia mahdollisia malleja ohjelmiston toteutuksesta. Hän osasi kertoa minulle selkeästi suunnitelmiaan ja ajatuksiaan siitä, millainen sovellus tulisi olemaan. Mielestäni hänen kuvailemansa sovellus tulisi helpottamaan sovelluksen käyttäjän työntekoa sekä arkea huomattavasti, joten näin aiheen hyödyllisenä ja mielenkiintoisena. Tämä on todella tärkeä asia sovelluskehityksessä.

Toimeksiantajani oli omassa selaintyöskentelyssään huomannut ongelman, jossa hänellä oli aivan liian monta eri välilehteä auki omalla tietokoneellaan, ja niiden välillä navigoiminen oli vaikeaa ja aikaavievää. Tähän ongelmaan luotavalla SaaS-ohjelmistolla pyritään saamaan ratkaisu.

Olen kiinnostunut sovellusten kehittämisestä, joten kokonaan uuden SaaS-sovelluksen luominen on hyvä aihe kehittää omaa osaamistaan sovelluskehityksen parissa eri sovelluskehityksen osa-alueella. Sovelluskehittäjän työnkuva on sellaista, jota haluaisin itse tulevaisuudessa tehdä. Opinnäytetyössäni keskityn sovelluksen toimintaan, sovelluksen eri komponenttien yhdistämiseen toimimaan yhteydessä, miten sovelluksessa käytettävää dataa hallitaan ja liikutellaan sekä sen ulkoasuun, eli millaisia komponentteja sovelluksen luomisessa käytetään, ja miten eri komponentteja muotoillaan. Ulkoasun tulee miellyttää käyttäjän silmää, ja sovelluksen käyttäminen tulee olla helppoa, jotta käyttäjä ei turhautuisi sovellusta käyttäessä.

Luotava SaaS-ohjemisto on opinnäytetyön toimeksiantajan toiveiden mukaisesti pidettävä salassa, jolloin sovelluskehityksen sisällöstä tai tarkemmin kerrotuista työvaiheista ei tuoda aineistoa opinnäytetyöhön.

Sovellus tullaan kehittämään aivan alusta full stack -kokonaisuudella, joka tarkoittaa frontendin ja backendin yhdistelmää. Minulle annettiin vastuu päättää, millaisilla teknologioilla tulen sovelluksen MVP-version luomaan. Valitsin frontendin tehtäväksi Angular-ohjelmistokehyksellä, backendin ASP.NET Corella sekä tietokannan SQL Serverillä. Sekä ASP.NET Core että SQL Server ovat Microsoftin kehittämiä ohjelmistoja, jolloin backendin ja tietokannan yhdistäminen on yksinkertaista. Tavoitteena opinnäytetyssä on selvittää, miten Angular-ohjelmistokehyksen yhdistäminen Microsoftin ohjelmistoihin, eli backendiin ja tietokantaan, tapahtuu.

2 SaaS

2.1 Software as a Service

SaaS, eli Software as a Service, on suomennettuna Sovellus palveluna. SaaS-ohjelmisto on palveluntarjoajan ylläpitämä ohjelmisto, jota tarjotaan asiakkaille internetin välityksellä. Ohjelmistoa ei tarvitse ladata mistään, vaan asiakas käyttää sovellusta nettiselaimellaan. Asiakas, joka on yleensä organisaatio, maksaa palveluntarjoajalle sovelluksen käytöstä, yleensä joko vuosittain tai kuukausittain. SaaS-ohjelmisto on eri käyttäjille samoja, joita käyttäjät käyttävät omilla käyttäjillään. Käyttäjät tunnistautuvat itse sovellukseen, jolloin he voivat käyttää sovellusta. Samalla sovellus tarkastaa, minkä asiakasorganisaation käyttäjä kyseessä on. Asiakkaiden yksilöinti on SaaS-ohjelmiston vaatimus. Eri asiakkaat voidaan yksilöidä esimerkiksi palvelinten mukaan, jolloin jokaisella asiakkaalla on oma palvelin. (Vanninen, P. 2013) Tämä ei kuitenkaan ole välttämätöntä, sillä eri asiakasorganisaatioiden yksilöimiseen on muitakin vaihtoehtoja, kuten tallennetun tiedon yhdistäminen kyseiseen asiakkaaseen, ja jopa vaihtoehtoisesti asiakkaan yksittäiseen käyttäjään.

SaaS-ohjelmistoilla on useita eri hyviä puolia. SaaS-ohjelmistoja ei tarvitse asentaa eikä niitä tarvitse päivittää. Asiakas voi myös käyttää sovellusta mistä haluaa, millä laitteellaan tahansa ja milloin tahansa. Helposti skaalautuvan ohjelmiston muokkaaminen tarpeiden mukaan on helppoa. Tarvittaessa tietokannan laajuutta voidaan kasvattaa juuri sellaiseksi kuin on tarve (Cloudflare.).

Yleisiä SaaS-ohjelmistoja ovat muun muassa Googlen palvelut, Slack, Netflix ja Zoom (Vyas, I. 2022). Luomani SaaS-ohjelmisto tulee olemaan samanlainen, eli yritysten on mahdollista ostaa ohjelmisto käyttöönsä ja käyttää sitä selaimellaan.

SaaS-palvelut mahdollistavat ohjelmiston samanaikaisen käytön usean eri asiakkaan toimesta. Sovelluksen asiakkaat ovat yleensä organisaatioita, ja kun yhdessä organisaatiossa on useita eri työntekijöitä, voi SaaS-palvelulla olla samanaikaisesti satoja käyttäjiä. (Chai, W. 2021)

Sovelluksen käyttäjänhallintaa varten luomassani sovelluksessa tullaan käyttämään ulkopuolisen palveluntarjoajan kirjautumis- sekä asiakkuudenhallintapalvelua. Asiakkuuksien hallinta onnistuu tällä sovelluksella hyvin, ja tietoja voidaan yksilöidä eri organisaatioihin tai työntekijöihin. Organisaation sisällä toimivia työntekijöitä voidaan myös jaotella eri ryhmiin esimerkiksi työnkuvansa mukaan.

2.2 SaaS-ohjelmiston luominen

SaaS-ohjelmisto koostuu frontendistä, backendistä sekä tietokannasta. Käyttäjälle näkyvät komponentit näytöllä tehdään frontendissä, kun taas backendissä tehdään käyttäjälle näkymättömiä toimintoja. Tietokantaan tallennetaan ohjelmistossa käsiteltäviä tietoja.

Sovelluksen eri osia kannattaa suunnitella huolella, ennen kuin alkaa rakentamaan sovelluksen eri osia. Myös ohjelmointikielet kannattaa miettiä huolella. Sovelluksen eri osia voidaan rakentaa erilaisilla ja eri sovelluskehitysympäristöillä. Eri osien yhdistäminen toisiinsa, jotta sovellus tulisi toimimaan jokaisella osa-alueella kuten haluaa, kannattaa suunnitella.

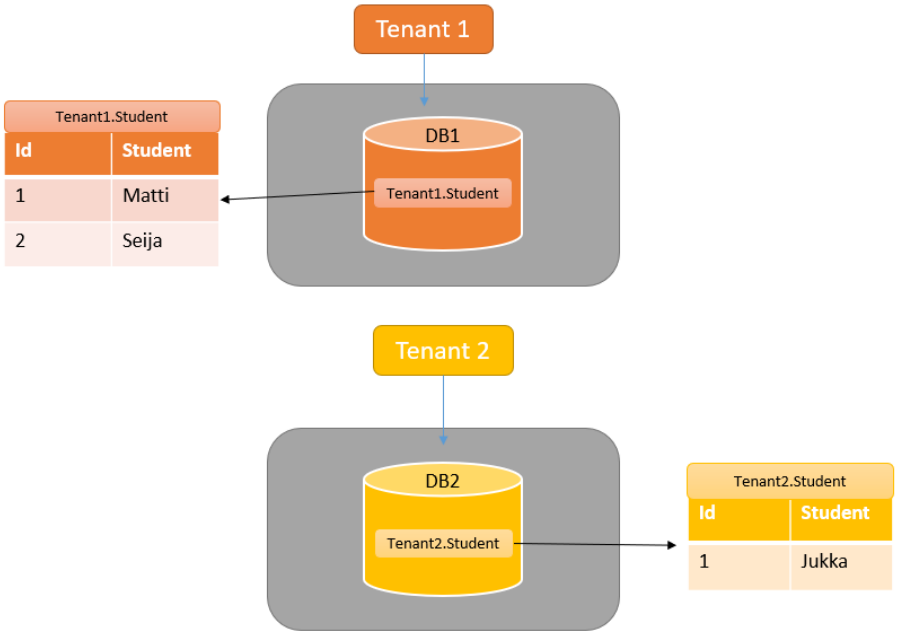
Eri osien yhdistäminen toisiinsa on helpompaa, kun osat luodaan saman palveluntarjoajan palveluilla, kuten esimerkiksi Microsoftin SQL Server -tietokanta sekä ASP .Net Core -backend.

2.3 Tiedon yksilöinti

Satojen eri käyttäjien tallentamia tietoja käsitellessä tietoja täytyy yksilöidä käyttäjäkohtaisesti. Käyttäjäkohtainen yksilöinti voi olla joko organisaatio- tai työntekijätasoista.

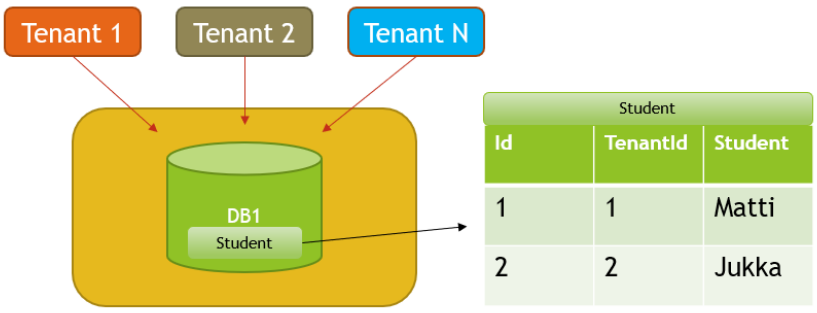
Käyttäjien tallentamia tietoja voidaan tallentaa eri tavoin, jolloin käyttäjien tiedonhallinta onnistuu. Kuvassa 1 jokaiselle asiakkaalle, eli organisaatiolle, luodaan oma tietokanta, johon tallentaa tietoja. Tässä jokaisen yksittäisen työntekijän tiedoille tulee asettaa omat yksilöivät indeksit, jotta henkilökohtaiset tiedot eivät voi olla saatavilla koko organisaatiolle.

Tässä vaihtoehdossa ei tarvitse ottaa huomioon eri organisaatioiden yksilöiviä indeksejä. (Hills, 2020)



Kuva 1. Tiedon tallentaminen useaan tietokantaan

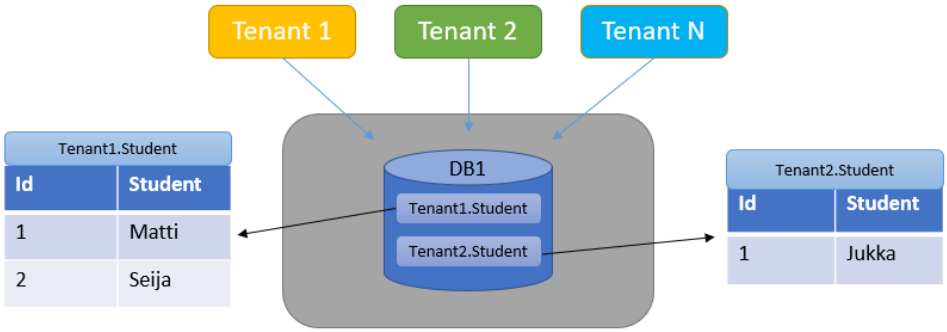
Kaiken tiedon voi myös tallentaa samaan tietokantaan, kuten kuvassa 2. Tällöin jokainen tieto tulee yksilöidä organisaation ja yksittäisen työntekijän kohdalla. Myös HTTP-kutsut vaativat yksityiskohtaisempia hakemisen kriteereitä, jotta halutun organisaation ja halutun työntekijän tietoja saadaan haettua. Tällaisessa tilanteessa riski paljastaa jotain salaista tietoa muille organisaatioille kasvaa. (Hills, A. 2020)



Kuva 2. Tietojen tallentaminen samaan tietokantaan

Yhteen tietokantaan voidaan tallentaa dataa myös toisella tavalla. Kuvassa 3 jokaiselle organisaatiolle luodaan oma taulu, jonne tietoa tallennetaan. Tietokannan tietoja on eristetty paremmin toisistaan, jolloin tietovuotojen riski pienenee. Organisaatioiden määrän

kasvaessa tietokantaan kerääntyy paljon samanlaisia tauluja ja tietokanta saattaa kasvaa suunnattoman suureksi. Tietokantakyselyt muuttuvat monimutkaisemmiksi, kun useita samanlaisia tietokantakyselyitä joudutaan tekemään monta kertaa. (Hills, A. 2020)



Kuva 3. Tietojen tallentaminen saman tietokannan eri tauluihin

Ulkopuolisen asiakkuudenhallintasovelluksen avulla jokaiselle organisaatiolle sekä sen työntekijälle luodaan omat indeksit, joiden avulla tallennettua tietoa voidaan suodattaa. Tietokantaan tulee tietenkin lisätä tallennetun tiedon lisäksi käyttäjäkohtainen indeksi, jolla tieto yhdistetään tiettyyn organisaatioon tai työntekijään.

2.4 Sovelluksen julkaiseminen

SaaS-ohjelmisto ostetaan palveluna, joka toimii verkossa. Asiakas ostaa palvelun, jonka jälkeen hän pystyy alkaa käyttämään sovellusta missä itse haluaakaan.

Sovellus julkaistaan palvelimella, jolloin sovelluksen käyttäjä ei ole riippuvainen mistään muusta, kuin omasta verkkoyhteydestään. Erilaisia pilvipalveluiden tarjoajia on tarjolla monia, joista yleisiä ovat Amazon Web Services, Microsoft Azure sekä Google Cloud (Chand, M. 2021).

3 Sovelluksen suunnittelu

Sovelluksen perusidea on kerätä käyttäjältä hänen tallentamaansa tietoa, ja hänen tulee voida lukea tallennettuja tietoja visuaalisesti näyttävältä käyttöliittymältä. Sovellus vaatii toimiakseen kolme eri osaa. Sovellus kerää suuria määriä käyttäjien syöttämiä ja tallentamia tietoja tietokantaan. Tallennettua tietoa tulee voida lukea ja käyttää jatkossa, jolloin tietokanta on välttämätön sovelluksen rakenteessa. Tietokantaan tallennetaan erimuotoista tietoa, jolloin tallennettavat tietotyypit tulee miettiä tarkasti. Tietoa pitää voida jaotella, järjestellä sekä muokata käyttäjän toimesta eri tavoin. Tallennetut tiedot tulee nimetä selkeästi ja tietoa hyvin kuvaavasti, jotta tulevaisuudessa mahdollisesti muilla samojen ohjelmistojen kanssa tekemisissä olevien on helpompi saada sovelluksen eri osien rakenteista ja sisällöistä kiinni.

Käyttäjien tallentamaa tietoa pitää pystyä yksilöimään ja yhdistämään käyttäjään itseensä sekä toisiin käyttäjiin tarpeen mukaan käyttäjäkohtaisesti. Kaikkea tietokantaan tallennettua tietoa ei tarjota kaikille käyttäjille näkyväksi, vaan tiedot yksilöidään käyttäjäkohtaisiksi. Tietokannan sisällön hallintaan luodaan CRUD-toimintoja, joita asiakas voi oman toimintansa mukaan käyttää.

Backendissä eri komennot hallitsevat tietoja nimiensä mukaan eri tavoin. Tiedon hallinnan perustoimintoja ovat Create, Read, Update ja Delete, joista käytetään yhteisnimitystä CRUD. CRUD-toiminnot tulevat sovelluksen keskeisten tietojen hallintaan.

Käyttäjienhallinta tulee olemaan suuressa osassa sovellusta. Itse kehitettynä käyttäjienhallintaohjelmisto tulisi viemään todella paljon resursseja, jolloin valmiin käyttäjienhallintaohjelmiston yhdistäminen luotavaan sovellukseen tekisi sovelluksen luomisen paljon helpommaksi, vaikka hyvä käyttäjienhallintaohjelmisto voi maksaa sovelluksen kehitysvaiheessa paljon. Hyvä käyttäjienhallintaohjelmisto tähän SaaS-ohjelmiston luomiseen sisältäisi myös kirjautumismahdollisuuden, jonka sovellus vaatii. Käyttäjänhallintajärjestelmällä voidaan myös hallita käyttäjiä, ja heidän oikeuksiaan käyttää SaaS-ohjelmaa.

Eri tiedot tulee jaotella omiksi tietokantatauluikseen, jolloin tauluja voidaan yhdistellä toisiinsa. Taulujen sisältöjä yhdistetään keskenään, jolloin ne rakentavat täyden kokonaisuuden. Sovelluksen kehittyessä tietokantaa tulee voida muokata ja jatkokehittää suuremmaksi. Tietokantaa suunnitellessa ja rakentaessa täytyy ottaa huomioon muuttujien nimeäminen ja muuttujiin tallennettavien tietojen tietotyypit.

Sovelluksen yksi tärkeimmistä kriteereistä on upea ulkoasu. Sovelluksen ulkoasun suunnittelu on siis todella tärkeä osa sovelluksen suunnittelua. Sovellus tulee olla helppo ja

yksinkertainen käytettävä, jotta uusi käyttäjä ei turhautuisi sovellusta käyttäessä. Sovelluksen ulkoasun rakentamiseen ja viimeistelyyn käytetään ohjelmistokehystä, jonka avulla ulkoasun muokkaminen ja rakentaminen onnituu hyvin.

Tiivistettynä sovellus on luotu full-stack -ohjelmointitavalla, joka sisältää tietokannan, frontendin sekä backendin. Full-stack -ohjelmointi on yleistä, sillä yli puolet ohjelmoijista identifioi itsensä full-stack -ohjelmoijiksi. (University Of Toronto.)

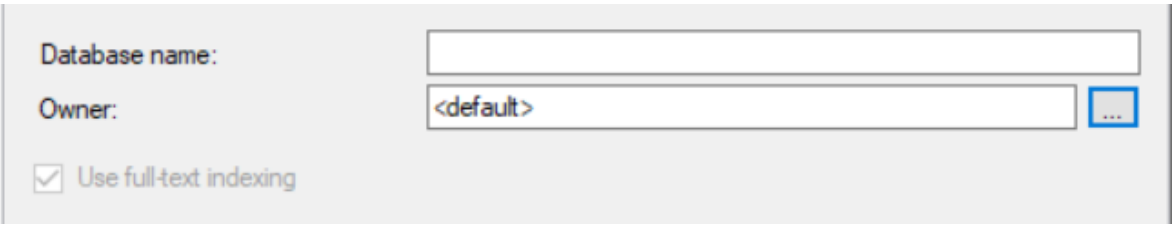
Frontend ja backend ohjelmia kehitetään eri ohjelmointikielillä, joten full-stack -ohjelmoijan tulee osata kehittää ohjelmia useammalla kuin vain yhdellä kielellä. Tottakai myös frontend- ja backend -kehittäjän on hyvä osata ohjelmoida useilla eri kielillä. Projektien vaihdellessa myös ohjelmointitavat ja -kokonaisuudet vaihtelevat.

Sovellus tullaan julkaisemaan pilvessä, jotta sen käyttäjät pääsevät käsiksi sovelluksen toimintoihin. Sovellus julkaistaan Azuressa, johon jokaiselle eri komponentille luodaan oma palvelin. Palvelimilla olevat komponentit kommunikoivat ja toimivat keskenään. Asiakas on tämän jälkeen itse vastuussa oman sovelluksensa päivittämisestä, kun SaaS-ohjelmiston uusimmat päivitykset on päivitetty palvelimille. Seuraavissa kolmessa kappaleessa käsitellään full-stack -ohjelmointitavalla luotavia eri komponentteja, joista SaaS-ohjelmisto koostuu.

4 Tietokanta

4.1 SQL Server

Sovelluksen tietokantana toimii Microsoftin SQL Server -tietokanta, jonka toimintaympäristönä toimii Microsoft SQL Server Management Studio. Uusi tietokanta luodaan kohdasta New database, joka löytyy vasemman palkin Databases-kohdan alta hiiren oikealla näppäimellä. Kuvassa 4 tietokannalle annetaan haluttu nimi, jonka jälkeen se saadaan luotua.



Kuva 4. Tietokannan luominen

SaaS-ohjelmiston luomisvaiheessa loin tietokannasta sellaisen version, joka tulisi olemaan mahdollisimman lähellä lopullisen ohjelmiston tietokantaa. Tässä vaiheessa mahdollisten muutosten tekeminen on vielä helpompaa kuin sellaisen tietokannan muokkaaminen, jonka ympärille on rakennettu kokonainen sovellus.

Tietokannan rakenteeseen tuli muutoksia vaihtelevasti. Välillä muutokset olivat tiedostotyypin muutoksia, ja välillä tietokantaan lisättiin aivan kokonaan uusia tauluja, kun asiakkaan kanssa keskustellessa tuli ilmi jatkokehityksessä tulevista toiveista ja mahdollisuuksista. Taulujen tietojen nimeämisissä kannattaa miettiä, miten isot ja pienet kirjaimet asetellaan.

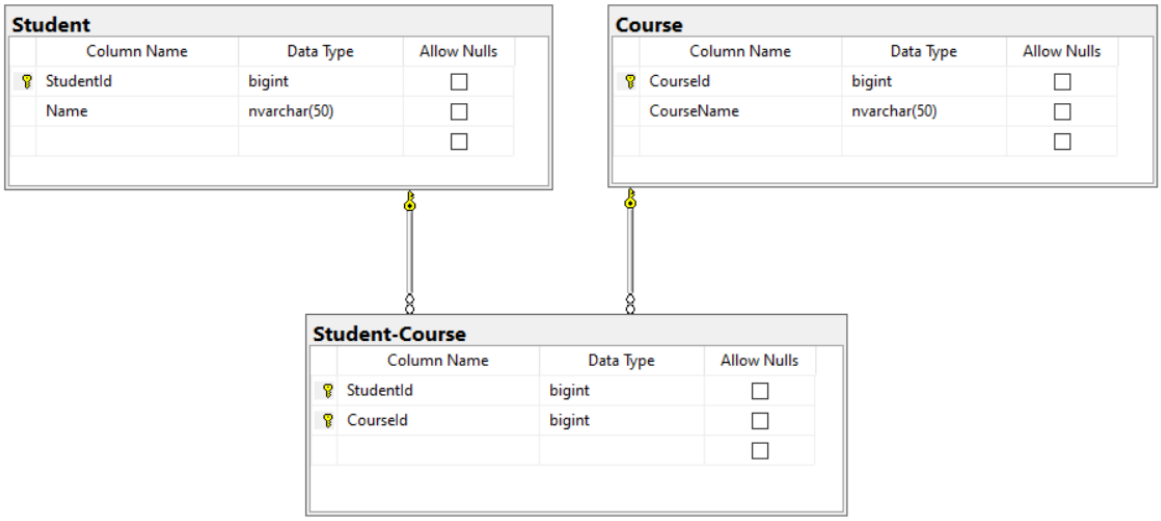
4.2 Tietokantataulujen suhteet

Tietokannassa eri taulut on liitetty toisiinsa erilaisten suhteiden avulla. Tauluja on yhdistetty toisiinsa one-to-many- sekä many-to-many -suhtein. One-to-many -suhde tarkoittaa taulujen välillä olevan sellainen suhde, jossa taulun muuttujalla voi olla suhde useampaan yhden taulun muuttujiin. Many-to-many -suhteet taas antavat mahdollisuuden molempien taulujen muuttujilla olevan suhteessa moneen eri toisen taulun muuttujaan.

Esimerkkikuvassa 5 on esitetty oppilaan ja kurssin yhteyden luominen. Samalla kurssilla voi olla monta oppilasta, ja samaan aikaan sama oppilas voi olla monella eri kurssilla. Student- sekä Course-taulujen välille luodaan uusi taulu, Student-Course, joka yhdistää nämä taulut. Yhdistävän taulun tehtävänä on tallentaa haluttujen taulujen perusavaimet,

primary key. Tällöin perusavaimet muodostavat parin, joiden avulla saadaan luotua useita eri yhdistelmiä näiden kahden halutun tietokantataulun välille.

Primary key eli perusavain yksilöi taulun jokaisen tiedon omakseen. Kuvassa 5 nähdään taulujen perusavaimet, jotka on merkitty keltaisilla avaimilla. Perusavain on jokaisella tiedolla uniikki, eikä se voi olla tyhjä. Secondary keyn eli vierasavaimen tarkoituksena on viitata toisen taulun tiedoissa olevaan perusavaimeen. Toisin kuin perusavain, voi vierasavaimen arvoksi asettaa tyhjän arvon eikä vierasavaimen arvon tarvitse olla uniikki. (GeeksForGeeks, 2022.)

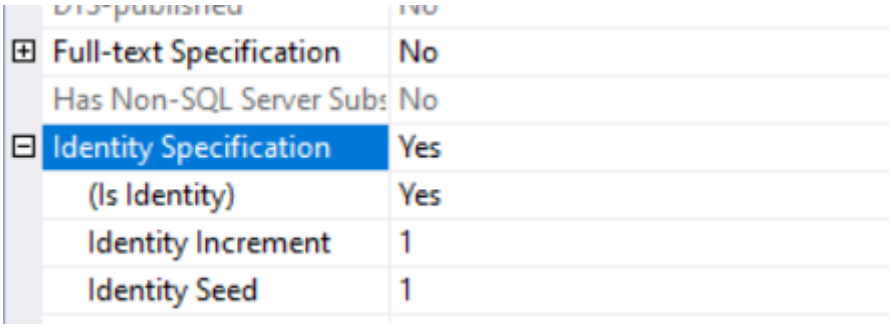


Kuva 5. Tietokantataulujen yhteydet

Taulujen sisällön yksilöinti tapahtuu perusavainten eli ID:n avulla. ID:t asetetaan olevan alla olevan kuvan 6 mukaisesti auto-incrementeiksi. Auto-increment -ominaisuus asettaa halutulle tiedolle, joka on tässä tilanteessa ID, automaattisesti int-arvon siinä järjestyksessä, kun tietoa lisätään. Int-tietotyyppi tarkoittaa kokonaislukua. Tämä ominaisuus antaa tiedon yksilöllistämiseen yksinkertaisen identifioimistavan, jolloin pelkän yksilöivän ID:n mukaan saadaan tarkan muuttujan tiedot.

Many-to-many -suhteessa luodun taulun sisällössä on molempien siihen kytköksissä olevien taulujen ID:t. Tällaiseen tauluun voidaan helposti lisätä esimerkkikuvan perusteella monta eri oppilaan ID:t sekä niille pariksi kurssien ID-arvot. Samalla kurssilla voi siis olla useita oppilaita ja yksi oppilas voi olla usealla eri kurssilla.

Tietokantaan luoduille uusille tiedoille asetetaan ID:t, joiden avulla tietoja tunnistetaan. Kuvan 6 mukaan numeroarvoinen ID asetetaan automaattisesti kasvavaksi. Käyttäjän ei tarvitse tällöin asettaa itse ID-arvoa tallennetulle tiedolle.



Kuva 6. Tallennetun tiedon ID:n automatisointi

5 Backend

5.1 Backend

Backend on sovelluksen käyttäjälle näkymätön koodi, joka toimii tiedonliikuttajana tietokannan ja käyttäjän näkymän, frontendin, välillä. Backendissä luotava koodi on käyttäjälleen huomaamatonta, mutta backendin tekemien eri komentojen nopeudet välittyvät käyttäjälle. Suurten tietomassojen lataaminen voi olla hidasta, jolloin käyttäjän näytöllä näkyvä näkymä voi joutua lataamaan joitakin tietoja pitkäänkin, mikä on sekä sovelluskehittäjän, että sovelluksen käyttäjän näkökulmasta haitallista.

Tämän sovelluksen backend luodaan .Net Frameworkillä, joka on Microsoftin ohjelmistokomponenttikirjasto. Verkkokehyksenä toimii ASP.Net Core, ja kehitysympäristönä Visual Studio 2019. Ohjelmointikielenä toimii C#.

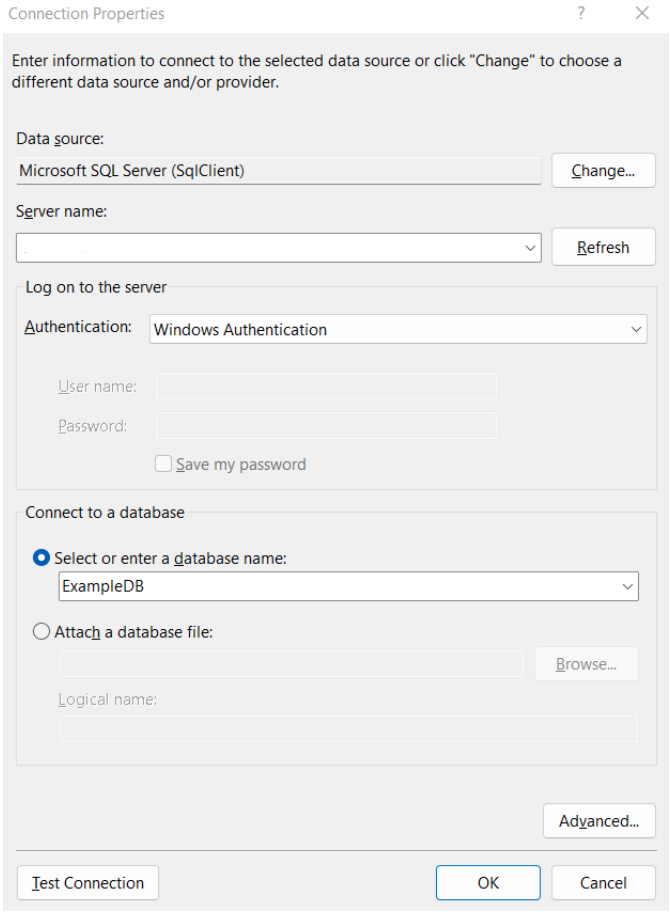
Visual Studiolla projektia aloittaessa valitsin sovelluskehitykseen ASP .Net Core Web API -sovelluskehityspohjan. Kyseinen alusta luo kontrollerin, johon tehdään REST HTTP-kutsuja tiedonhakua varten.

5.2 Tietokannan yhdistäminen backendiin

Tietokantana toimiva SQL Server sekä backendinä toimiva ASP .Net Core ovat molemmat Microsoftin ohjelmistoja, jolloin kyseiset ohjelmistot on suunniteltu siten, että niiden yhdistäminen on tehty helpoksi.

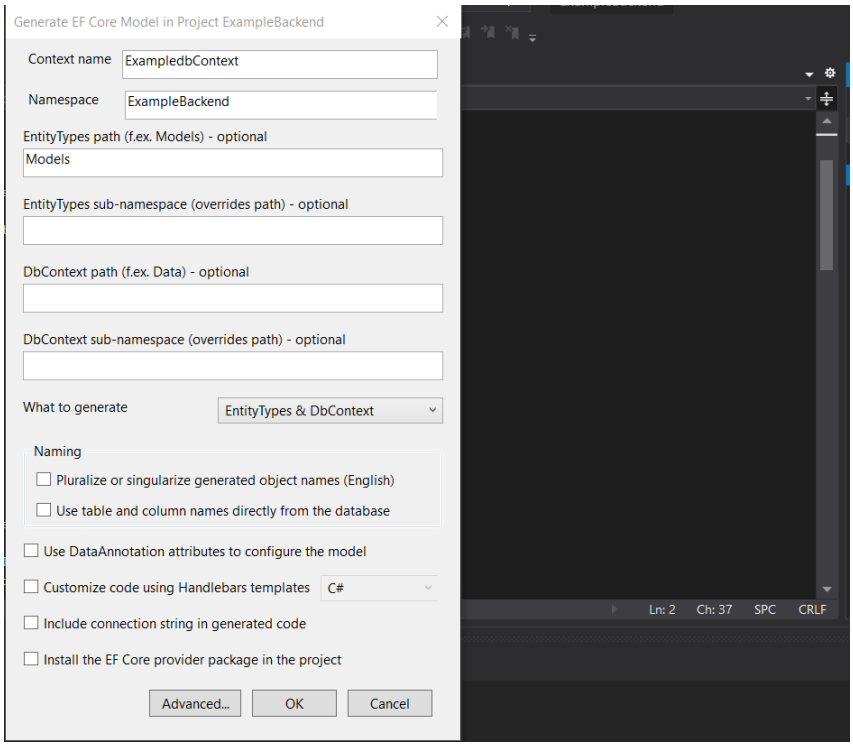
Tietokannan taulut ja niiden tiedot voidaan lisätä backendiin Entity Framework Power toolsin Reverse Engineerin avulla, kuten kuvassa 7 on tehty. Haluttu tietokanta valitaan

omalta tietokoneelta, jonka jälkeen valitaan halutut taulut, jotka tuodaan backendiin.



Kuva 7. Tietokannan yhdistäminen backendiin

Contextille annetaan haluttu nimi ja samalla luodaan kansio eri tietokannan tauluille, kuten kuvassa 8 on tehty, ja kansion nimeksi on asetettu Models. Context on luokka, jonka avulla pystytään hallitsemaan tietokannan tietojen eri CRUD-toimintojen sisältöjä. Context-luokka yhdistetään tietokantaan, jonka tietoihin halutaan päästä käsiksi. Context-luokkaan tallennetaan arvoja, joita välitetään eteenpäin. Contextiin voidaan tallentaa erilaisia tietokokonaisuuksia, jotka eivät välttämättä ole samanlaisia keskenään. Tänne on yhdistetty eri luokkien rajapinnat, joiden avulla voidaan luoda eri luokkia, kuten opiskelijaluokka.



Kuva 8. Tietokantataulujen yhdistäminen backendin luokiksi

Sovelluksen luomisvaiheessa käytetään omalla tietokoneella olevaa tietokantaa, joka on luotu SQL Serverillä. Kyseinen tietokanta yhdistetään backendiin appsettings.json tiedostossa, johon tehdään uusi ConnectionString. Se yhdistetään omaan tietokantaan ja tietokannan omien tietojen mukaan, kuten kuvassa 9 tehdään. Tietokantaan tehtyjen uusien muutosten jälkeen, uuden tietokannan version voi helposti päivittää myös backendiin EF Core Power Toolsin Reverse Engineer -komennolla. Kuvassa 10 otetaan käyttöön haluttu tietokanta, jonka ConnectionString aiemmin luotiin. ConnectionStringin avulla yhdistetään tietokanta sekä sovelluksen backend.

ConnectionString on normaali merkkijono, joka sisältää tietoja yhdistettävästä tietokannasta. SQL Serverin ConnectionString sisältää tiedonlähteen eli palvelimen, josta tietokanta haetaan. Tämän lisäksi ConnectionStringiin kuuluu palvelimella sijaitsevan tietokannan nimi ja käyttäjän tiedot, joilla kirjaudutaan palvelimelle eli käyttäjätunnuksen sekä salasanan. Tietokannan vaihtuessa uudelle tietokannalle tulee tehdä uusi ConnectionString, johon liitetään tarvittavat tiedot. Tietokannan nimen tai palvelimen tietojen muuttuessa tulee muutokset tehdä ConnectionStringiin, jotta yhteys tietokantaan on toimiva.

```
"ConnectionStrings": {
  "ExampledbContext": "Data Source=[redacted];Initial Catalog=ExampleDB;Integrated Security=True"
},
```

Kuva 9. Tietokannan yhdistäminen

Tietokannan yhdistämisen jälkeen Models-kansioon on luotu luokat jokaiselle tietokannan taululle. Tämä on todella helppoa ja nopeaa Microsoftin sovellusten avulla.

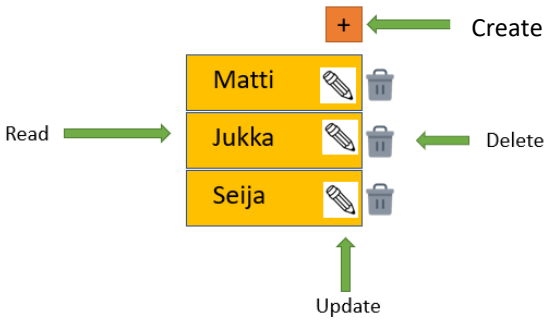
Samalla voidaan lisätä uusi NuGet-package, Microsoft.EntityFrameworkCore sekä Microsoft.EntityFrameworkCore.SqlServer. Tämän jälkeen uusi nimetty muuttuja lisätään Start.up-tiedostoon, joka yhdistää tietokannan backendiin.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ExampledbContext>(opt =>
    {
        opt.UseSqlServer(Configuration.GetConnectionString("ExampledbContext"));
    });
```

Kuva 10. Halutun tietokannan käyttämisen asettaminen

5.3 Tiedonhallinta

Tietokannan sisältöä hallitaan backendin avulla. Backendissä tehdään erilaisia komentoja, jotka tekevät haluttuja toimintoja tietokannan tiedoilla. Kuvassa 11 on havainnollistettu CRUD-toimintoja listalle.



Kuva 11. CRUD-toimintojen havainnollistaminen sovellusmallissa

Sovelluksen kannalta tärkein CRUD-toiminto on uuden tiedon lisääminen, eli create, jotta muita CRUD-komentoja voitaisiin käyttää. On kuitenkin hyvä ensimmäiseksi luoda tiedon lukemiseen tarkoitettu, read, komento. Tällä pystytään myös ensimmäiseksi varmistamaan, että oikea tietokanta on yhdistetty backendiin.

CRUD-komentojen luominen aloitetaan luomalla Controller, joka vastaa sovelluksessa liikkuvan tiedon sisällöstä. Kuvassa 12 on esimerkki controllerista. Komennot hoitavat tiedonhallintaa tietokannassa. Controlleriin asetetaan haluttu rajapinnan URL-osoite, josta tietokannan tietoa voidaan hakea. Yleensä controllerin nimi on sama kuin haluttu rajapintaosoite. Kuvan 12 controlerissa URL-osoite on muodossa ”api/student”.

```
namespace ExampleBackend.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class StudentController : ControllerBase
    {
        private readonly IStudentRepository _studentRepository;

        0 references
        public StudentController(IStudentRepository studentRepository)
        {
            _studentRepository = studentRepository;
        }

        //POST: api/Student
        [HttpPost]
        0 references
        public ActionResult Post([FromBody] Student student)
        {
            var result = _studentRepository.Create(student);
            return new JsonResult(result);
        }

        //GET: api/Student
        [HttpGet]
        0 references
        public ActionResult<string> Get()
        {
            var result = _studentRepository.Read();
            return new JsonResult(result);
        }
```

Kuva 12. Esimerkki controllerista

Controlleriin luodaan funktioita, joilla tehdään haluttuja tehtäviä tietokannan datalle. Kutsut kulkevat luotuun repositoryyn sen oman rajapinnan kautta. Rajapinta on merkitty sen nimen alussa I-kirjaimella. Kuvassa 13 on esimerkki luodusta rajapintarepositorysta.

Rajapinta eli interface on abstrakti- eli tyhjäluokka, johon arvoja tallentamalla saadaan luotua sovellukselle tarvittavia kokonaisuuksia. Uuden luokan luominen Create-komennolla vaatii arvojen asettamisen luokan muuttujille, kuten kuvassa 13 nähdään vaadittavien muuttujien olevan opiskelijan ID sekä nimi. ID on asetettu tietokannassa automaattisesti luotavaksi, joten tässä tilanteessa tarvitsee lisätä vain ainoastaan opiskelijan nimi. Kuvassa 14 esitetään opiskelijaluokan komennot.

Opiskelijaluokan ensimmäinen komento on uuden opiskelija luomista varten. Toisella komennolla voidaan hakea lista opiskelijoista ja kolmannen komennon tehtävänä on hakea vain yksi opiskelija opiskelijan ID:n mukaan. Neljäs komento on opiskelijaluokan muokkaamista varten ja viimeisen, viidennen komennon avulla voidaan poistaa opiskelija tiedot tietokannasta.

```
public partial class Student
{
    – references
    public Student()
    {
        Student_Course = new HashSet<Student_Course>();
    }

    – references
    public long StudentId { get; set; }
    – references
    public string Name { get; set; }

    – references
    public virtual ICollection<Student_Course> Student_Course { get; set; }
}
```

Kuva 13. Opiskelijaluokka

```
7   namespace ExampleBackend.Repositories
8   {
        4 references
9       public interface IStudentRepository
10      {
            2 references
11          Student Create(Student student);
            2 references
12          List<Student> Read();
            3 references
13          Student Read(long studentId);
            2 references
14          Student Update(Student student, long studentId);
            2 references
15          void Delete(Student student, long studentId);
16      }
17  }
18
```

Kuva 14. Esimerkki rajapinta repositorysta

Rajapintarepositoryyn on luotu CRUD-toiminnot, jotka tehdään repositoryssä halutun funktion aikana. Repositoryissä tapahtuu tiedon lukeminen ja hallinta. Rajapinta välittää tietoa eri funktioiden välillä. Kuvassa 15 on esitetty repositoryssa olevia CRUD-toimintoja.

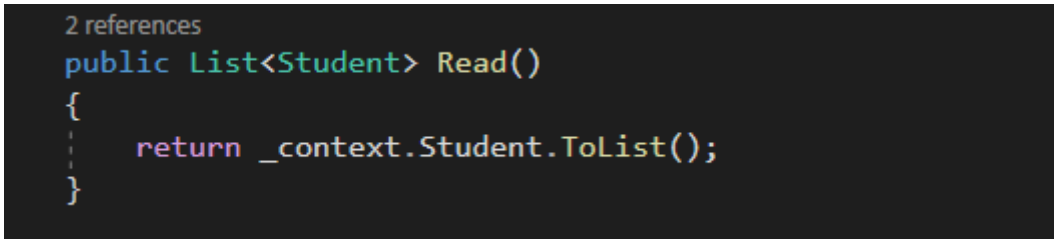
```
namespace ExampleBackend.Repositories
{
    2 references
    public class StudentRepository : IStudentRepository
    {
        private readonly ExampledbContext _context;
        0 references
        public StudentRepository(ExampledbContext context)
        {
            _context = context;
        }

        2 references
        public Student Create(Student student)
        {
            _context.Add(student);
            _context.SaveChanges();
            return student;
        }

        2 references
        public void Delete(Student student, long studentId)
        {
            Student removedStudent = Read(studentId);
            removedStudent = _context.Student.AsNoTracking().FirstOrDefault(s => s.StudentId == studentId);
            if(removedStudent != null)
            {
                _context.Remove(removedStudent);
                _context.SaveChanges();
            }
        }
```

Kuva 15. Esimerkki repositorysta, jossa luotuja CRUD-toimintoja

Eri funktioilla on omia erilaisia toimintoja, joita voi tarkoituksenmukaisesti muokata halutunlaiseksi. Repositoryssä luodaan ensin Context-objekti, jonka tehtävänä tallettaa arvoja ja lähettää ne eteenpäin. Eri funktioissa käsitellään juuri luotua Context-objektia. Kaikkein yksinkertaisimmillaan Context-objekti lukee halutun luokan sisällön tietokannasta ja luo niistä listan, kuten kuvan 16 komento tekee.



```
2 references
public List<Student> Read()
{
    return _context.Student.ToList();
}
```

Kuva 16. Read-toiminto opiskelijalistan hakemiseen

Tässä on esimerkki repositoryssä olevasta funktiosta, jonka tehtävänä on hakea Student-luokan oliot, ja luoda niistä lista järjestyksessä Student olioiden ID:iden mukaan. Lopulta Controllerissa saatu tieto muutetaan JSON-tyyppiseksi tiedoksi.

Backendissä repositoryihin luodut funktiot palauttavat HTTP-kutsujen kautta halutun datan JSON-muotoisena tietona. Tätä tietoa voidaan viedä frontendille, jossa käyttäjä näkee muokattua ja mahdollisesti suodatettua JSON-muotoista tietoa omalla näytöllään. JSON-muotoon tulostettavaa tietoa muokataan halutuksi backendin repositoryn koodissa. Tietoa voidaan järjestellä eri järjestykseen tai siitä voidaan halutessaan tulostaa vain joku tietty osa.

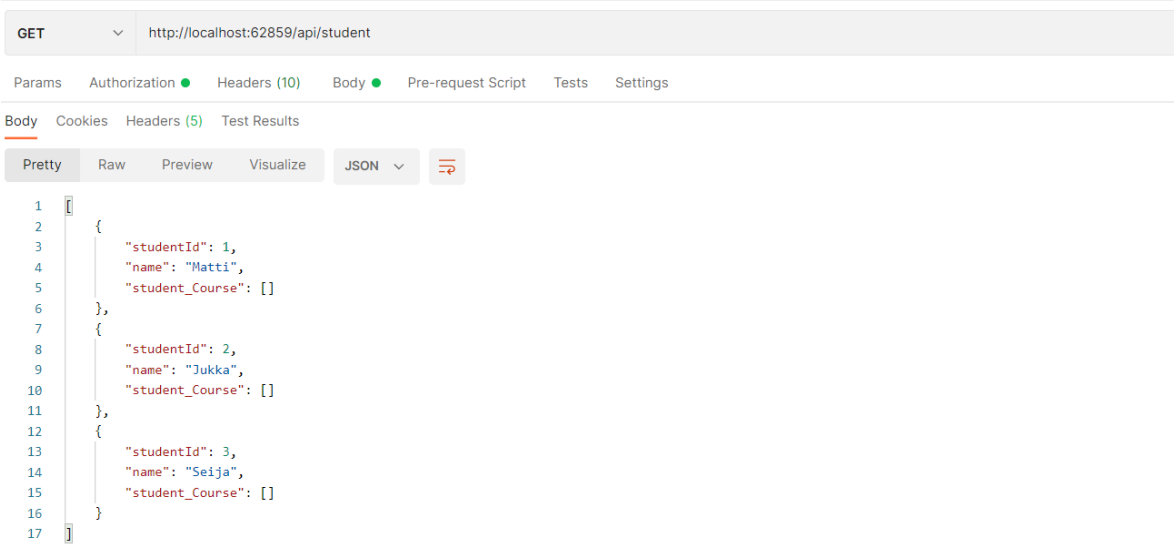
5.4 Backendin toiminnan testaaminen

Backendin eri funktioiden toimivuutta kannattaa testata erilaisilla siihen tarkoitetuilla sovelluksilla. API-liittymien testaamisessa käytin itse Postman-sovellusta. Postmanilla voidaan testata käyttäjän itsetekemiä kutsuja ja niiden toimintaa.

Backendin päällä ollessa Postmanilla luetaan yksinkertaisen GET-komennon, jolla haetaan kaikki oppilaat tietokannasta. Kyseinen GET-komento on esitetty kuvassa 17. Postmanilla asetetaan toiminnoksi GET, ja URL-osoitteeksi osoite, jonka avulla tehdään haluttu funktio. Osoitteen lopussa on polku, jonka avulla esimerkin lista oppilaista haetaan. Kuvassa 18 on näkymä, joka nähdään Postman-sovelluksessa, kun haluttu lista on tulostettu.

```
//GET: api/Student
[HttpGet]
0 references
public ActionResult<string> Get()
{
    var result = _studentRepository.Read();
    return new JsonResult(result);
}
```

Kuva 17. Valmis ja toimiva esimerkki Read-toiminnosta controllerissa



Kuva 18. Tulostettu lista halutusta luokasta JSON-muodossa

5.5 REST

REST eli Representational State Transfer on arkkitehtuurimalli rajapinnoissa, jonka avulla saadaan liikuteltua tietoa tietokannan ja JSON-muotoisen syötteen välillä. Eri HTTP-kutsuilla on omat polkunsa, joista HTTP-kutsu kutsutaan ja ajetaan läpi. Kuvassa 19 on esitetty HTTP-kutsu student-repositoryssa, jolloin student tulee polkuun kertomaan controllerin nimen mukaan, mitä controlleria käytetään.

Kuvan 19 HTTP-kutsussa tehdään Read-komento, joka lukee tietoa. REST-kutsuja kohdistetaan syöttämällä polku URL-osoitteeseen, johon sisällytetään oikean controllerin nimi, tarvittavat merkkijonot sekä indeksit. Haettava tieto tallennetaan muuttujalle result ja tieto haetaan studentRepositoryn rajapinnasta, joka on merkitty _-merkillä. Rajapintarepositoryssa on funktio Read, johon syötetään URL-osoitteessa oleva opiskelijan indeksi. Rajapintarepository on esitetty kuvassa 14 ja rajapintarepositoryn kautta siirrytään tekemään toiminto studentrepositoryyn. Kuvan 20 komento hakee halutun ID:n mukaan opiskelijan tiedot ja asettaa ne tyhjään luokkaan, jonka se palauttaa controllerille. Kun result-muuttuja on saanut oman arvonsa controllerissa, palautetaan se uutena JSON-muotoisena vastauksena, joka näkyy kuvassa 21. Opiskelijasta syötetään kaikki tiedot, joita opiskelijaluokka sisältää.

```
//GET: api/student/1
[HttpGet ("{studentId}", Name = "Get")]
0 references
public ActionResult Get(long studentId)
{
    var result = _studentRepository.Read(studentId);
    return new JsonResult(result);
}
```

Kuva 19. HTTP-kutsun polku

```
public Student Read(long studentId)
{
    return _context.Student.AsNoTracking().FirstOrDefault(s => s.StudentId == studentId);
}
```

Kuva 20. Student-luokan lukeminen ID:n avulla

```
{
    studentId: 2,
    name: "Jukka",
    studentCourse: [ ]
}
```

Kuva 21. Opiskelijaluokan tuloste

5.6 HTTP-kutsujen oikeudet

Sovelluksen HTTP-kutsujen tulee olla salattuja, jotta sovellukseen kirjautumaton käyttäjä ei voi tehdä kutsuja, jolloin tietokannan tieto ei mene ulkopuolisille. Ulkopuolisen palveluntarjoan kirjautumispalvelulla voidaan asettaa organisaation käyttäjälle rooli, joka voi olla esimerkiksi ryhmänjohtaja, myyntityöntekijä tai admin-käyttäjä. HTTP-kutsuihin

tehdään käyttäjän roolin tarkistus, joka määrittää, onko käyttäjällä oikeus tehdä kyseistä kutsua. Kutsun oikeus määritellään ennen HTTP-kutsua luokan controllerissa. Kuvassa 19 on määritetty, että roolilla Teacher saadaan ajettua HTTP-kutsu.

Käyttäjän rooli saadaan haettua kirjautumisen yhteydessä JWT-tokenin avulla. JWT-tokenin, JSON Web Token, avulla pystytään hallitsemaan käyttöoikeustietueita ja sen avulla varmistetaan, onko käyttäjällä oikeutta kyseiseen kohteeseen. JWT-tokenin avulla saadaan selville, mikä rooli käyttäjälle on asetettu ja onko hänellä oikeuksia tehdä kyseisiä kutsuja. Esimerkki kuvassa 22 käyttäjällä tulee olla asetettu Teacher-rooli, jotta HTTP-kutsu voidaan suorittaa.

```
//GET: api/student/1
[HttpGet ("{studentId}", Name = "Get")]
[Authorize("Teacher")]

public ActionResult Get(long studentId)
{
    var result = _studentRepository.Read(studentId);
    return new JsonResult(result);
}
```

Kuva 22. Get-toiminto, jonka läpikäymiseen vaaditaan Teacher-rooli

Käyttäjän rooli tarkastetaan roolin käsittelijällä kuvassa 23, jonka tehtävänä on suorittaa haluttu tehtävä, mikäli käyttäjällä on siihen sopiva rooli. Tämä komento tarkastaa myös, onko käyttäjä kirjautunut sovellukseen sisälle, vai onko kyseessä jokin ulkopuolinen käyttäjä.

```
if (!context.User.HasClaim(c => c.Type == ClaimTypes.Role && c.Issuer == requirement.Issuer))
    return Task.CompletedTask;

var roles = context.User.FindFirst(c => c.Type == ClaimTypes.Role && c.Issuer == requirement.Issuer).Value.Split(' ');

if (roles.Any(s => s == requirement.Role))
    context.Succeed(requirement);

return Task.CompletedTask;
```

Kuva 23. Roolintarkastaja

6 Frontend

6.1 Frontend

SaaS-sovelluksen frontend on luotu Angular ohjelmointikehyksellä, jossa ohjelmointikielenä on Typescript. Angular-ohjelmointikehyksellä voidaan kehittää sovelluksia niin verkkoselaimille, tietokoneille sekä mobiililaitteille, jotka ovat nykypäivänä entistä yleisempiä. Angularista käytettävä sovellusversio on 12.2.8. Kehitysympäristönä käytetään WebStorm-sovellusta.

Angularin oma komponenttikirjasto tarjoaa sovelluksen luontiin paljon erilaisia komponentteja, joiden avulla sovelluksen ulkoasu voidaan luoda käyttäjän silmälle miellyttäväksi. SaaS-sovelluksen yksi päävaatimuksista on ulkoasun näyttävyys. Saman teeman säilyttäminen sovelluksen eri näkymissä luo sovellukselle yhtenäisyyttä ja asiakkaan käyttökokemus parantuu.

6.2 TypeScript

Angular-ohjelmistokehyksen avulla ohjelmistoja voidaan luoda TypeScriptillä. TypeScript perustuu JavaScriptiin, ja se on Microsoftin vuonna 2012 kehittämä ohjelmointikieli, joka toimii suurempien ja monimutkaisempien sovellusten luomisessa todella hyvin.

TypeScriptissä on kaikki JavaScriptin toiminnallisuudet, sekä se on entistä tarkempi virheen tarkistamisessa. TypeScriptin on siis kuin kehittyneempi JavaScript, ja se kehittyy kokoajan.

6.3 Angular-projektin luonti

Uusi projekti luodaan Angular CLI:llä ja projektille annetaan nimi. Uusi projekti luo valmiiksi TypeScript-, yksikkötestaus spec.ts-, HTML- ja CSS-tiedostot sekä app.module.ts -tiedoston. App.module.ts -tiedostossa käynnistetään applikaatio sekä siellä asennetaan moduulit, joita eri komponenteissa käytetään. Jokaiselle komponentille tulee komponentin luodessaan edellä mainitut neljä tiedostoa, joista jokaisella on oma tehtävänsä. Kuvassa 24 ensimmäisenä student-komponentin kansiossa on CSS-tiedosto, jonka tehtävänä on muokata ulkoasun tyylejä. Kaikki ulkoasun muokkaaminen tekstien fontista erilaisiin animaatioihin tehdään CSS-tiedostoissa. HTML-tiedosto on verkkosivun näkymä, joka näkyy selaimessa. Sen avulla luodaan näkymään kaikki näkyvät elementit, kuten tekstiä tai kuvia. Spec.ts -tiedostot ovat yksikkötestausta varten. Yksiköt ovat sovelluksen osia, funktioita tai luokkia. Yksikkötestauksessa varmistetaan ettei ohjelmoidussa koodissa ole virheitä.

Kuva 24. Angular-komponentin sisältö

Yksikkötestit siis tarkistavat ohjelmiston toimivuuden. Sovelluksen virheiden kartoittaminen ja korjaaminen on paljon helpompaa ja halvempaa sovelluksen kehitysvaiheen alussa, kuin pidemmällä sovelluskehityksessä. TypeScript-tiedostoihin luodaan verkkosivun toiminnot. Tiedon liikkuminen, nettisivuilla navigoiminen, eri nappien ja komponenttien toiminnot tapahtuvat tässä tiedostossa.

6.4 Backendin kanssa kommunikoiminen

Jotta frontendin ja backendin välinen kommunikoiminen, eli tiedon siirtäminen onnistuu, tarvitsee Angular-projektiin lisätä erilaisia moduleita. Modulet ovat mekanismeja, joiden avulla erilaiset toiminnot voidaan tehdä. Angularin modulet lisätään app.module-tiedostoon kuvan 25 esittämällä tavalla. Tiedon hakemiseen tietokannasta vaaditaan HttpClientModule, jonka avulla backendin kanssa kommunikointi onnistuu. Modulen lisäämisen jälkeen module tulee lisätä myös komponenttiin, jossa sitä käytetään. Kuvassa 26 on esimerkki modulen lisäyksestä komponenttiin.

```
imports: [
  BrowserModule,
  HttpClientModule
],
```

Kuva 25. Projektiin lisätyt moduulit

```
constructor(private http: HttpClient) { }
```

Kuva 26. HttpClient-modulen lisääminen TypeScript-tiedostoon

Sovelluksen frontend sekä backend toimivat eri verkkosivuilla, jolloin frontend hakee toiselta verkkosivulta tietoa, joka on tässä tilanteessa backend. Tämän vuoksi CORS-mekanismi tekee rajoituksia tiedon jakamisesta verkkosivulta toiselle. CORS määrittää eri HTTP-osoitteita, joihin yhdistäminen on sallittua. CORS määrittää myös, millä eri metodeilla

yhteyden ottaminen eri HTTP-osoitteisiin sallitaan. Kuvassa 27 on annettu lupa käyttää kaikkia verkkosivuja yhteydessä sovelluksen verkkosivun kanssa. Kaikille metodeille on myös annettu oikeudet tehdä API-rajapintakutsuja. Tilanteiden vaihtuessa voidaan asettaa sallituiksi metodeiksi pelkästään haluttuja komentoja, kuten vain GET-komento.

```
services.AddCors(options =>
{
    options.AddPolicy(
        name: "AllowOrigin",
        builder =>
        {
            builder.AllowAnyOrigin().AllowAnyMethod();
        });
});
```

Kuva 27. Eri verkkosivujen kanssa kommunikoimisen salliminen

Projektin HTML-tiedostoon on luotu yksinkertainen komento, jolla kaikkien opiskelijoiden nimet tulostetaan listana näytölle. Kuvassa 28 on kuva HTML-tiedostosta, jossa lista tulostetaan, ja kuvassa 29 on kuva tulostetusta listasta. Angularin erilaisten komponenttien avulla listaa voi muokata sellaiseksi kuin itse haluaa. Angularin avulla myös tulostettavan listan ulkoasua on helppo muokata haluamansalaiseksi käyttäjän näytölle.

```
1  <div>
2    <div>Example</div>
3    <tr *ngFor="let student of studentList">
4      <td>{{student.name}}</td>
5    </tr>
6  </div>
7
```

Kuva 28. Esimerkki HTML-tiedostosta, jolla tulostetaan esimerkkilista

Example
Matti
Jukka
Seija

Kuva 29. Kuva tulostetusta listasta näytöllä

7 Sovelluksen julkaisu

7.1 Julkaisu

Valmis MVP-sovellus julkaistaan Azure-palvelussa. Azure on Microsoftin pilvipalvelu, joka tunnettiin ennen nimellä Windows Azure. Azuressa voidaan rakentaa ja julkaista sovelluksia virtuaalipalvelimille. Luodun sovelluksen tietokanta, frontend sekä backend julkaistaan Azuren pilvipalvelussa, jolloin sovellusta voidaan käyttää verkossa paikasta riippumatta. Sovelluksen tietokanta, Microsoftin SQL Server sekä backend, Visual Studiolla tehty ASP .Net Core, ovat molemmat Microsoftin palveluita. Näiden yhdistäminen Azureen on tehty Microsoftin työkalujen ansiosta helpoksi. SQL Server Management Studiolla sekä Visual Studiolla pystytään julkaisemaan komponentti Azureen omien toimintojensa kautta.

Sovelluksessa on kolme eri julkaistavaa osaa, frontend, backend sekä tietokanta. Azureen jokaiselle osalle on luotu omat palvelimen. Jotta jokainen palvelin pystyy kommunikoimaan keskenään, asetetaan oikeat URL-osoitteet, jotta tiedon hakeminen ja sen tulostaminen toimii julkaistavalla verkkosivulla. Kun sovellusta kehitettiin, ja ohjelmaa ajettiin omalla paikallisella koneella, tulee backendiin muuttaa tietokannan tiedot, jotka ovat nyt palvelimen tietoja. Myös frontendiin asetettu osoite, jonka avulla haetaan backendin tietoja, tulee muuttaa Azure-palvelimen osoitteeksi. Azuressa luoduille palvelimille tulee URL-osoite, jossa sovelluksen osa pyörii.

Azuressa tietokannan palvelimelta löytää tietokannan Connection stringin, jonka avulla tietokanta yhdistetään backendiin. Connection string löytyy asetuksista kohdasta Connection strings. Tältä sivulta löytyvä merkkijono kopioidaan ja liitetään backendin appsettings.json -tiedostoon, kuten aiemmin kuvassa 9 on tehty.

7.2 Tulokset

Tavoitteena oli luoda MVP-sovellus, jolla on keskeisimmät toiminnallisuudet sovelluksen käyttämistä varten. Keskeisimpiin toiminnalliusuuksiin päästiin ja toiminnallisuudet saatiin luotua toimiviksi. Sovelluksen miellyttävän ulkoasun tärkeyttä painotettiin koko kehitysaika ja sovelluksen ulkoasua kehitettiin jatkuvasti. Miellyttävään ulkoasuun päästiin, mutta sovelluksen jatkokehityksen mukana sovelluksen ulkoasua tullaan kehittämään jatkossa vielä modernimpaan ja käyttäjäystävällisempään suuntaan.

Sovelluksen frontend, backend sekä tietokanta saatiin luotua ja niiden kehittäminen yhdessä toimeksiantajan kanssa onnistui sujuvasti. Eri komponentiti toimivat keskenään yhdessä hyvin ja komponenttien jatkokehitystä on helppo jatkaa.

Sovelluksesta julkaistiin MVP-versio, jonka frontend, backend sekä tietokanta julkaistiin Azuren pilvipalvelussa. Luodun sovelluksen julkaisemisen jälkeen sovelluksella oli ensimmäisiä testikäyttäjiä, joilta saatiin palautetta ja mielipiteitä sovelluksen toiminnasta ja sen hyödyllisyydestä. MVP-versio sisältää keskeisimmät toiminnallisuudet, joten toiveita jatkokehitystä varten luonnollisesti tuli ilmi. Sovelluksen MVP-version kehittäminen onnistui ja sovelluksen ensimmäisellä versiolla luotiin pohja tulevaa jatkokehitystä varten.

8 Päätelmiä

SaaS-ohjelmiston luominen alusta alkaen on monivaiheinen tehtävä. Eri komponenttien suunnittelu, rakentaminen ja toisiinsa yhdistäminen ovat jokainen isoja tehtäviään. Aivan uuden tietokannan suunnitteleminen ja toteutus vaatii paljon aikaa sekä eri näkökulmia projektissa toimivilta henkilöiltä. Näin tietokantaan voidaan rakentaa mahdollisimman hyvät ja laajat toiminnot alusta pitäen.

Sovelluksen komponenttien suunnittelu pätee myös backendiin sekä frontendiin. Suunnitellaan halutut HTTP-kutsut, ja luodaan niitä. Samoin suunnittelu sovelluksen ulkoasusta ja frontendin toiminnoista pitää suunnitella. Valmiiden komponenttien yhdistäminen toisiinsa oli minun mielestäni selkeää. Microsoftin tuotteiden, SQL Server -tietokannan sekä ASP .Net Core -backendin, yhdistäminen toisiinsa on helppoa, sillä ne ovat suunniteltu toimimaan keskenään. Visual Studiolla luotava backend on helppo yhdistää tietokantaan Visual Studiossa olevien toimintojen avulla. Toiminnot, jotka yhdistävät halutut luokat suoraan backendiin, ovat mielestäni todella hyviä ja helppoja käyttää. Tiedon liikuttaminen ja sen hallinnointi on mielestäni tehty todella selkeäksi. Frontendin ja backendin yhdistäminen oli mielestäni kätevää ja selkeää. Eri palvelimilla toimivat sovelluksen komponentit pystyttiin yksinkertaisilla tavoilla yhdistämään toisiinsa. Se, että frontendin on helppo hakea eri HTTP-kutsuja pelkän URL-osoitteen avulla, on mielestäni helppoa. Angularilla luodun frontendin lopputulos on mielestäni hyvä. Angularin avulla voi tehdä visuaalisesti näyttäviä komponentteja, joiden avulla voidaan rakentaa kokonainen SaaS-ohjelmisto. Azureen yhdistäminen oli mielestäni hankalin vaihe sovelluksen kehittämisvaiheessa. Microsoftin tietokannan yhdistäminen Azureen onnistui SQL Serverin kautta. Samoin Visual Studiolla on omat ohjelmat Azureen julkaisemista varten. Angular-sovelluksen julkaiseminen Azureen tapahtui Github-repositoryn julkaisemisen avulla. Github-repository julkaistaan Azuressa, jolloin sovelluksen frontend toimii omalla palvelimellaan.

Verkkosovelluksen luominen Angularin sekä Microsoftin työkalujen avulla onnistuu mielestäni hyvin. Eri komponenttien yhdistäminen toisiinsa on helppoa myös niiden muokkaamisen jälkeen.

MVP-ohjelmiston luomisen tavoitteeseen päästiin, ja sovelluskehitys ohjelmiston parissa jatkuu. Sovellusta tullaan kehittämään käyttäjien toiveiden mukaan ja ohjelmistopäivityksiä tullaan tekemään jatkossa eri sovelluskehityksen osa-alueilla.

Lähteet

Becker, R. 2020. Minimum Viable Product (MVP). Viitattu 30.9.2022. Saatavissa https://www.techopedia.com/definition/27809/minimum-viable-product-mvp

Chai, W. 2021. Software as a Service (SaaS). Viitattu 27.9.2022. Saatavissa https://www.techtarget.com/searchcloudcomputing/definition/Software-as-a-Service

Chand, M. 2021. Top 10 Cloud Service Providers In 2021. Viitattu 28.9.2022. Saatavissa https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/

Cloudflare. What is SaaS | SaaS definition. Viitattu 30.9.2022. Saatavissa https://www.cloudflare.com/learning/cloud/what-is-saas/

Geeks for Geeks, 2022. Difference between Primary and Foreign Key. Viitattu 1.11.2022. Saatavissa https://www.geeksforgeeks.org/difference-between-primary-key-and-foreign-key/

Hills, A. 2020. Multi-Tenancy Database Design Approaches with SQL Server (Part 2). Viitattu 27.9.2022. Saatavissa https://www.sentryone.com/blog/multi-tenancy-with-sqlserver-part-2-approaches

Short, T. & Spiller L. 2022. What is SaaS? 10 FAQs About Software-as-a-Service. Viitattu 30.9.2022. Saatavissa https://www.softwareadvice.com/resources/saas-10-faqs-softwareservice/

University Of Toronto, What Is a Full Stack Developer & What Do They Do? Viitattu 27.9.2022. Saatavissa https://bootcamp.learn.utoronto.ca/blog/what-is-a-full-stackdeveloper/

Vanninen, P. 2013. SaaS, multi-tenant ja vahinkovakuutusohjelmiston kehitys. Viitattu 27.9.2022. Saatavissa https://www.cs.helsinki.fi/u/paakki/Semik13-Vanninen.pdf

Vyas, I. 2022. 10 Popular Examples Of SaaS Applications [Proven Strategies]. Viitattu 30.9.2022. Saatavissa https://citrusbug.com/blog/saas-application-example