jamk.fi

Betonin lujuudenkehityksen seuranta-palvelu

Valtteri Lähdesmäki

Opinnäytetyö
Joulukuu 2020
Tekniikan ala
Insinööri (AMK), sähkö- ja automaatiotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Lähdesmäki, Valtteri	Opinnäytetyö, AMK	Joulukuu 2020
	Sivumäärä 60	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Betonin lujuudenkehityksen seurantapalvelu

Tutkinto-ohjelma
Insinööri (AMK), sähkö- ja automaatiotekniikka

Työn ohjaaja(t)
Riekkinen, Juho; Häkkinen, Veli-Matti

Toimeksiantaja(t)
Lähdesmäki, Pekka

Tiivistelmä

Opinnäytetyössä luotiin toiminnoiltaan valmis selainpohjainen sovellus betonin lujuudenkehityksen seurantaan rakennustyömaille. Opinnäytetyöprojektissa käytettiin nykyaikaisia ohjelmistoja sovelluksen kehittämiseen. Työkalujen ja ohjelmistojen vaatimus oli, että ne soveltuivat testiyrityksen palvelimien kanssa yhteen, jotta testaaminen pystyttiin suorittamaan heidän suojatuilla palvelimillaan. Selainpohjaisen sovelluksen piti sisältää toiminnoilta käyttäjien, yritysten ja projektien hallinnan. Lisäksi sovelluksen piti sisältää myös matemaattiset algoritmit, joilla pystyttiin arvioimaan betonin lujuudenkehitys ja esittämään sitä kuvaava käyrästö.

Työn toteutusmenetelmäksi valittiin kehittämistutkimus, koska opinnäytetyössä kehitettiin sovellus. Projektin pohjana toimi vanhat olemassa olevat betonin lujuudenkehityksen seurantasovellukset. Vanhoista sovelluksista otetiin mallia tähän projektiin. Uutta sovellusta kehitettäessä huomioitiin testiyrityksen tarpeet. Työssä käytiin läpi sovelluksen luonnin perusmenetelmiä ja kuinka nykyaikaiset ohjelmistokirjastot nopeuttavat ohjelmointia ja sen myötä säästävät rahaa niin asiakkaalta kuin toimittajalta.

Sovellusta kehitettiin etsimällä tehokkaita sovelluskehitystyökaluja. Sopivimman ohjelmistotyökalun löydettyä, tutkittiin laajasti työkalun toiminnallisia menetelmiä ja kirjastoja, mitä hyödynnettiin sovelluksen kehittämisessä. Sovelluksen kehittämisessä käytettiin mahdollisimman paljon kirjastoja ja valmiita komponentteja. Opinnäytetyössä esitetään, miten samanlainen projekti pystyttäisiin toistamaan.

Avainsanat (asiasanat)

ASP.NET, MVC, Betonin lujuuden kehitys, SQL, API

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Lähdesmäki, Valtteri	Bachelor's thesis	December 2020
		Language of publication: Finnish
	Number of pages: 60	Permission for web publication: x

Title of publication

Concrete strength monitoring service

Degree programme

Electrical- and automation technology

Supervisor(s)

Riekkinen, Juho; Häkkinen, Veli-Matti

Assigned by

Lähdesmäki Pekka

Abstract

In the thesis, a web-based application for monitoring the concrete strength was developed. The project were developed using modern software approach. The requirements for the tools and software were that they should be compatible with the test company's servers so that testing could be performed on their secure servers. The web-based application was supposed to have functions for user, company, and project management. In addition, the application had been included mathematical calculations that were able to verify the development of the strength of the concrete and the diagrams describing it.

Development research was chosen as the method of implementing the work, because the application was developed in the thesis. The project was based on old existing concrete strength monitoring applications. The opinions of the test company were considered when developing the new application. The work was covered the basic methods of application creation and modern software libraries which speed up programming and save money for both the customer and the supplier.

The objects were developed the application by investigating and using the powerful application development tools. After finding the most suitable software tool, the functional methods and libraries of the tool were extensively studied and were utilized in the development of the application. The development of the application was aimed for using as many libraries and ready-made components as possible. The thesis shows how a similar project could be repeated.

Keywords/tags (subjects)

ASP.NET, MVC, Strength development of concrete, SQL, API

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

1 Johdanto

Rakennusteollisuus on digitalisoitumassa koko ajan enemmän. Digitalisoitumisessa uusien ohjelmistojen tekeminen tulee tärkeäksi ja tarpeelliseksi. Ohjelmistojen avulla rakennusteknologian yritykset pysyvät kehityksessä mukana ja pystyvät tarjoamaan asiakkaalle parempia ja turvallisempia tuotteita. Uusien ohjelmien avulla pyritään säästämään aikaa ja sitä kautta myös rahaa.

Tämän opinnäytetyön tavoite on esittää, kuinka pystytään nykyaikaisilla menetelmillä ja työkaluilla ohjelmoimaan ja suunnittelemaan toimiva sovellus rakennusteknologiaan. Nykyaikaisilla ohjelmistoilla pystytään vähentämään ohjelmistokehityksen kustannuksia. Opinnäytetyössä esitetään hyviä käytäntöjä nettipohjaisen palvelun tekemisen vaiheista ja opetetaan tekemään samankaltaisia sovelluksia. Opinnäytetyössä opetetaan käyttämään ASP.NET -ohjelmointikehystä ja Visual Studion tavallisimpia lisäosia, joita ohjelmisto kehitykseen tarvitaan. Opinnäytetyössä selvitetään betonin lujuudenkehityksen ohjelman kehittämisessä tekemiäni virheitä.

2 Tavoitteet ja rajaukset

Opinnäytetyön tavoitteena on luoda valmis tai toimivuudeltaan valmis selain pohjainen ohjelmisto betonin lujuudenkehityksen seurantaan. Työssä kehitetään yleinen rakennusteollisuudelle soveltuva palvelu, jota myös betoniteollisuuden yritys tulee mahdollisesti käyttämään. Toimivalla lujuudenkehityksen seurantajärjestelmällä rakennusteollisuus voi säästää merkittävästi kustannuksia aikaa säästämällä. Opinnäytetyön tavoitteena on kehittää merkittävästi toimeksiantajan ohjelmointi-, laite-, sovellusarkkitehtuuri- ja algoritmiosaamista. Opinnäytetyö opettaa kommunikoimaan asiakasrajapinnan kanssa ja määrittelemään sovelluksen vaatimuksia.

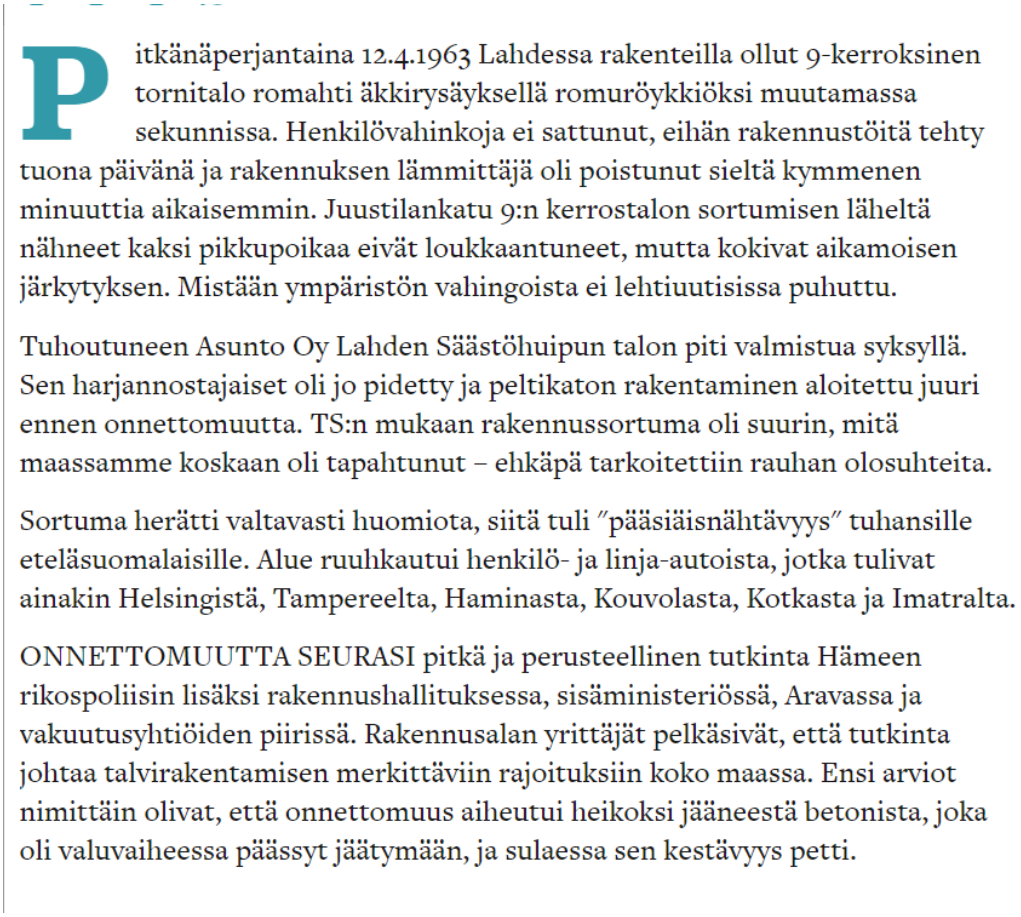
Opinnäytetyön tavoitteena on saada aikaan mahdollisimman luotettava ja tehokas selainpohjainen sovellus betonin lujuudenkehityksen laskentaan. Sovelluksen toimin-

nallisuuden tavoite on hallita organisaatioita, käyttäjiä ja mittareita. Sovellusta kehitetään tekemällä monia testiversioita. Testiversioiden avulla pystytään löytämään kaikki tarvittavat toiminallisuudet sovellukseen, joita projektin alussa ei välttämättä tiedetä. Aiheita, joita opinnäytetyö käsittelee ovat ASP.net, SQL, MVC, Microsoft SQL Server, Visual Studio, betonin lujuudenkehitys ja kolmannen osapuolen dataloggerin rajapinta.

3 Hankkeen tausta

Projekti alkoi kesällä 2019, kun toimeksiantajalta kysyttiin webpohjaista sovellusta betonin lujuudenkehityksen seurantaan. Sovellusta on kehitetty yhdessä toimeksiantajan kanssa ketterällä menetelmällä. Kehitystyöhön on osallistunut myös tuleva asiakas.

Betonirakentamisessa on tärkeää saada tietoa betonin lujuudenkehityksestä reaaliajassa, jotta tarvittavia toimenpiteitä voidaan tehdä rakentamisessa. Tärkeää on myös pystyä todentamaan betonivalun lopullinen lujuus. Uutta ohjelmaa lähdettiin toteuttamaan tarpeesta saada edullisempi ja toimivampi ohjelma, jolla betonialan yritykset voisivat parantaa kilpailukykyään ja varmistaa rakennusturvallisuutta. Esimerkki epäonnistuneesta betonivalusta on tornitalosta Lahdessa.



Pitkänäperjantaina 12.4.1963 Lahdessa rakenteilla ollut 9-kerroksinen tornitalo romahti äkkirysäyksellä romuröykkiöksi muutamassa sekunnissa. Henkilövahinkoja ei sattunut, eihän rakennustöitä tehty tuona päivänä ja rakennuksen lämmittäjä oli poistunut sieltä kymmenen minuuttia aikaisemmin. Juustilankatu 9:n kerrostalon sortumisen läheltä nähneet kaksi pikkupoikaa eivät loukkaantuneet, mutta kokivat aikamoisen järkytyksen. Mistään ympäristön vahingoista ei lehtiuutisissa puhuttu.

Tuhoutuneen Asunto Oy Lahden Säästöhuipun talon piti valmistua syksyllä. Sen harjannostajaiset oli jo pidetty ja peltikaton rakentaminen aloitettu juuri ennen onnettomuutta. TS:n mukaan rakennussortuma oli suurin, mitä maassamme koskaan oli tapahtunut – ehkäpä tarkoitettiin rauhan olosuhteita.

Sortuma herätti valtavasti huomiota, siitä tuli ″pääsiäisnähtävyys″ tuhansille eteläsuomalaisille. Alue ruuhkautui henkilö- ja linja-autoista, jotka tulivat ainakin Helsingistä, Tampereelta, Haminasta, Kouvolasta, Kotkasta ja Imatralta.

ONNETTOMUUTTA SEURASI pitkä ja perusteellinen tutkinta Hämeen rikospoliisin lisäksi rakennushallituksessa, sisäministeriössä, Aravassa ja vakuutusyhtiöiden piirissä. Rakennusalan yrittäjät pelkäsivät, että tutkinta johtaa talvirakentamisen merkittäviin rajoituksiin koko maassa. Ensi arviot nimittäin olivat, että onnettomuus aiheutui heikoksi jääneestä betonista, joka oli valuvaiheessa päässyt jäätymään, ja sulaessa sen kestävyys petti.

Kuvio 1 Uutinen heikoksi jääneestä betonista(Junttila 2013.)

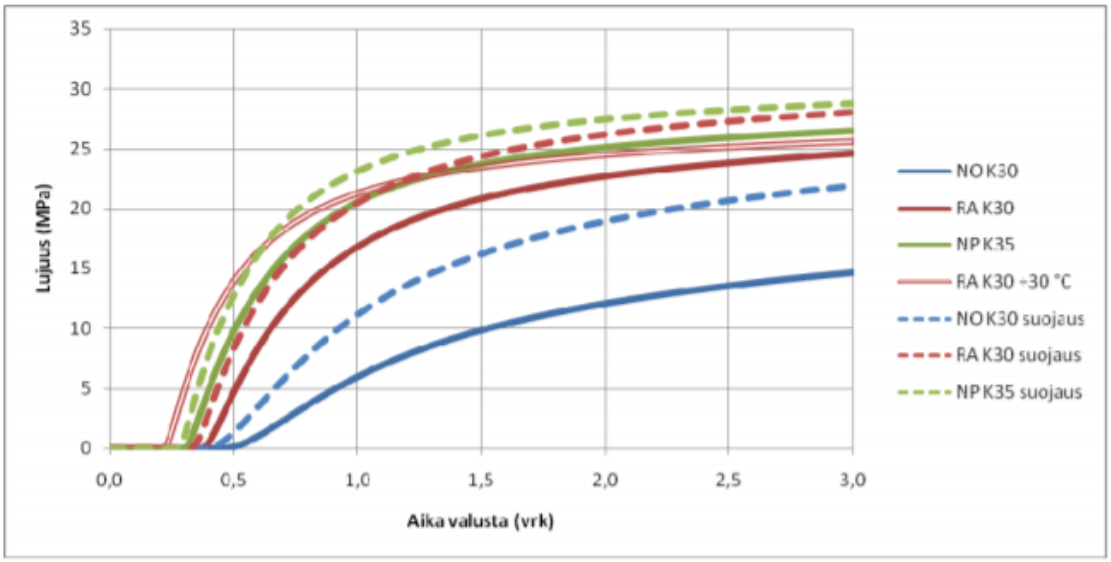
4 Rakennustekniikka

Betoni on maailman eniten käytetty rakennusmateriaali. Betonia käytetään rakennustekniikassa niin pientalojen-, kerrostalojen-, siltojen-, tunneleiden-, pato- ja voimalaitosrakenteiden rakentamiseen. Betonirakentamista voidaan tehdä valamalla paikan päällä tai tuoda valmiiksi tehtyjä elementtejä, esimerkiksi seiniä ja putkia. (Betonirakentaminen N.d.) Betonin lujuuden seuraaminen on tärkeä asia, koska se parantaa betonirakenteen turvallisuutta ja käyttöikää. Ne ovat osittain riippuvaisia betonin puristuslujuudesta. (Betoni on luja rakennusmateriaali, mutta mikä on betonin lujuus? N.d). Betoni rakentamisesta on monia hyötyjä moneen muuhun rakennusmateriaaliin ja sen suosio rakentamisessa perustuu betonin ominaisuuksiin. Betonin ominaisuuksia ovat kosteuden kesto, lujuus, jäykkyys, turvallisuus ja muokattavuus.

Lujuus pystytään määrittämään sekä valmiista rakenteesta että valusta. Tässä opinnäytetyössä keskitytään betonivalun aikaisen lujuuden määrittämiseen. (Betonin ominaisuudet ja käyttö N.d.)

Betonin lujuudella tarkoitetaan tietyn betonin lujuusluokkaa, puristuslujuutta koekappaleessa tai rakenteessa, säilyvyyttä tai betonirakenteen lujuutta. Betonin lujuuden määritelmälle ei ole yhtä kaavaa vaan sitä voi arvioida tilastollisilla menetelmillä. Betonin lujuudenkehitykseen vaikuttaa kovettumisaika ja lämpötila. Betonin kastelu parantaa lujuudenkehitystä. Lujuus kehittyy nopeammin kuivumisen ensimmäisinä päivinä ja hidastuu ajan kuluessa. Lujuudenkehityksen nopeuteen vaikuttaa myös ratkaisevasti lämpötila. Lämpötilaan betonissa vaikuttaa ulkoinen lämpötila eli sää ja betonin sisältä tuleva lämpötila eli reaktiolämpötila. (Betoni on luja rakennusmateriaali, mutta mikä on betonin lujuus? N.d.) Mitä lähemmäksi betonin lämpötila menee nollaan, kuivuminen hidastuu ja lähellä nollaa lopulta pysähtyy. Liian korkea tai matala lämpötila alentaa betonin lujuutta.

Jokaisella betonilla on oma lujuusluokka. Lujuusluokka määritellään betonille koekappaletta puristamalla. Puristuslujuutta arvioidaan valmistuksessa EN206-standardin mukaan. (Betoni on luja rakennusmateriaali, mutta mikä on betonin lujuus? N.d.) Betonin lujuus riippuu sen sisältämästä vesisementtisuhteesta. Vesisementtisuhteella tarkoitetaan betonin sisältämän vesimäärä ja sementin painonsuhdetta. Mitä enemmän betoni sisältää vettä, sitä enemmän sen lujuus pienenee.



Kuvio 2 Betonin lujuuden kehitys suojauksella ja ilman. (BETONIN VALINTA RAKENTEISIIN - OLOSUHDEHALLINTA N.d.)

Betonin lujuus voidaan arvioida Sadgroven menetelmällä. Sadgroven menetelmää käytettäessä tarvitaan tiedot betonin lämpötilasta ja betonin kovettumiseen kuluneesta ajasta. Kypsyys t20 esitetään kaavalla.

t20 = ((T+16 C)/36 C)2 * t

Kaavassa T[°C] on betonin lämpötila ja t[d] on kovettumisaika. Kypsyydestä ja tunnetusta betoniseoksesta voidaan johtaa arviolujuudelle.

(by 201 betonitekniikan oppikirja 2018, 92.)

5 Ohjelmistotyökalut ja tavat

Palvelu kehitettiin ISS palvelimelle ja SQL tietokannalle. Selvitystyön perusteella ohjelman kehitysympäristöksi valittiin Visual Studio. Visual Studiossa pystyy luomaan projektin, johon saa liitettyä SQL tietokannan ja Visual Studiossa pystyy hyödyntämään GitHub versionhallintasovellusta lisäosalla. Ohjelman ja tietokannan julkaiseminen on myös tehty helpoksi Visual Studiolla, mikäli palvelimet ovat jo valmiina ja konfiguroitu. Visual Studio valittiin myös siksi, että sillä pystyy ohjelmoimaan ASP.NET:illä, joka on nykyaikainen ja paljon käytetty ohjelmointikehys. ASP.NETissä on useita eri ohjelmointikieli- ja rakennevaihtoehtoa. Tätä projektia varten valittiin ASP.NET MVC ohjelmointikehys, RAZOR syntaksi ja ohjelmointikieleksi HTML ja C#.

5.1 Asp.net

ASP.NET on Microsoftin kehittämä webohjelmointikehys. ASP.NET on kehitetty tuomaan lisää työkaluja ja kirjastoja verkkosovellusten rakentamiseen .NET alustaan. ASP.NET on avoimen lähdekoodin ohjelmointikehys. ASP.NET on kehitetty alustariippumattomaksi ja sitä pystyy pyörittämään monilla eri alustoilla kuten Windowsilla, Linuxilla, MacOS ja Dockerilla. ASP.NET:in backendissä eli palvelimessa pyörivällä ohjelmistolla käsitellään verkkopyyntöjä C#- tai F# ohjelmointikielillä. Käyttöliittymä eli frontend tehdään Razor-syntaksilla, joka on dynaaminen tapa luoda verkkosivuja C#:lla. ASP.NET:ssä on monia eri kirjastoja sovellusmalleille kuten FORMS, MVC, AJAX, Web Pages, Web API, WebHooks ja SignaIR. Edellä mainituilla eri verkkosovellusmalleilla saadaan ohjelmistojen kehitykseen tehokkuutta. ASP.NET:issä on myös valmiina sisäänkirjautumisen todennusjärjestelmä, jolla pystytään käsittelemään kirjautumisia ja todentamaan käyttäjiä niin sisäisesti kuin kolmannen osapuolten kanssa kuten Google ja Twitter. (What is ASP.NET? N.d.)

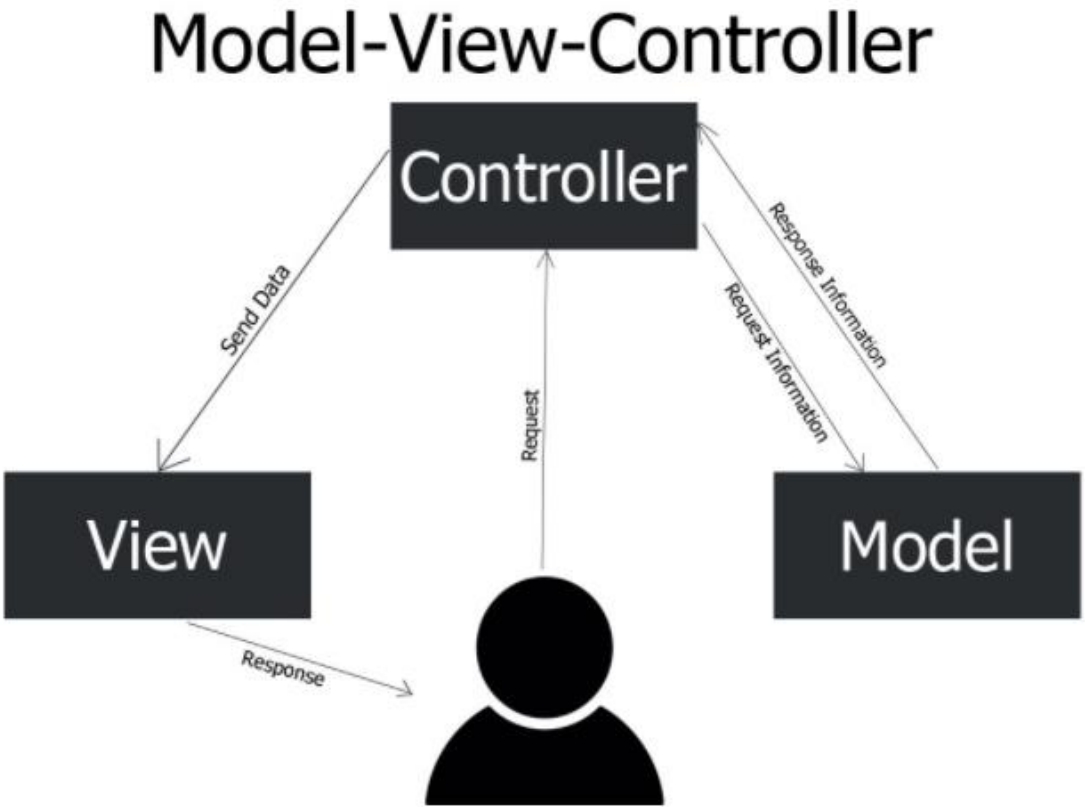
Backend on palvelimessa oleva taustakoodi, mikä lukee, tallentaa, päivittää, hakee ja käsittelee tietoa niin tietokannasta kuin frontendiltä. ASP.NET:ssä backend pystytään

kirjoittamaan kolmella eri ohjelmointikielellä, mitkä ovat C#, F# ja Visual Basic. ASP.NET on laajennus .NET ohjelmistokomponentti kirjastosta. ASP.NET:ssä pystyy käyttämään samoja kirjastoja kuin .NET. Valmiiden pakettien ja kirjastojen kirjo on suuri, koska .NET ja ASP.NET ovat olleet käytössä pitkään ja niiden kehittäjiä on paljon. Lisäksi omien kirjastojen ja pakettien kirjoittaminen on mahdollista. (What is ASP.NET? N.d.)

Frontend on käyttäjälle näkyvä osa päätelaitteella. Frontend näyttää ja hallitsee syötettyä tietoa käyttäjälle. ASP.NET:ssä frontend on dynaaminen sivu, mikä ohjelmoidaan C#, HTML, CSS ja JavaScriptillä. ASP.NET käyttää Razor syntaksia dynaamisten nettisivujen luomiseen. Razor tarjoaa minimalistisen syntaksin frontendin luomiseksi ja siten esimerkiksi ei tarvita aloitus- ja lopetustageja sekä HTML ja C# pystytään yhdistämään sujuvasti. (Weil 2015 20.)

5.2 MVC framework

MVC on ASP.NET:n sisältyvä ohjelmistonsuunnittelumalli. MVC:n avulla pystytään tekemään dynaamisia verkkosivustoja ja se mahdollistaa toiminnallisuuksien erittelyn. MVC on ohjelmiston suunnittelumalli, joka tulee sanoista model, view ja controller. Ohjelmiston suunnittelumallissa erotetaan ohjelmiston data (model), käyttöliittymän näkymä eli käyttöliittymä (view) ja sovelluksen logiikka (controller). Tämän mallin avulla pystytään yhdistämään helpommin sovelluksen tietoa ja dataa sekä helpottamaan projektin kehitystä selkeyttämällä ohjelman osia. Virheen korjaaminen ja tietyn komponentin testaus on myös helpompaa erottelun myötä. MVC -malli tekee helpoksi päivittämisen ohjelman kehityksen aikana. Jokaisella ohjelmistomallin pääryhmällä on oma päätehtävänsä, mutta niitä pystyy myös käyttämään jossain määrin ristikkäin. MVC -mallin avulla käyttäjän pyynnöt ohjataan sovelluksen logiikalle, mikä on vastuussa mallin tehtävän suorittamisesta tai kyselyn noutamisesta. Ohjaimen tehtävä tässä mallissa on valita oikea näkymä käyttäjälle ja tuoda näkymälle kaikki tarvittavat tiedot. (ASP.NET MVC Pattern N.d.)



Kuvio 3 MVC -osien yhteydet (Spinelli 2018.)

Mallin (model) tehtävä ohjelmistossa on olla dynaaminen tietorakenne ja näyttää sovelluksen tila sekä ohjata logiikoita ja toimintoja. Malli on siis järjestelmän osa, mikä käsittelee ja varastoi ohjelmiston sisällä liikkuvaa tietoa. Logiikat ja sovelluksen toteutukset olisi hyvä sisällyttää malliin sovelluksen tilan säilyttämiseksi. Tietoa hallinnoivat mallit kytketään tietovarastoon eli tietokantaan. (MVC for dummies: malli, näkymä ja ohjain -arkkitehtuuri web-sovelluksissa 2020.)

Näkymän (view) tehtävä on näyttää käyttäjälle tietoa oikeassa muodossa. Tietoa pystytään näyttämään HTML, JSON tai XML -muodossa. Näkymä antaa käyttäjälle näytettävän kuvan lisäksi myös dataa käyttöliittymän käsiteltäväksi. Razor tarjoaa näkymälle yksinkertaisen ja vähän kuormittavan tavan esittää HTML -sisältöä. Näkymässä pystytään myös Razorin ansiosta kirjoittamaan C# ohjelmointikieltä. (MVC for dummies: malli, näkymä ja ohjain -arkkitehtuuri web-sovelluksissa 2020.)

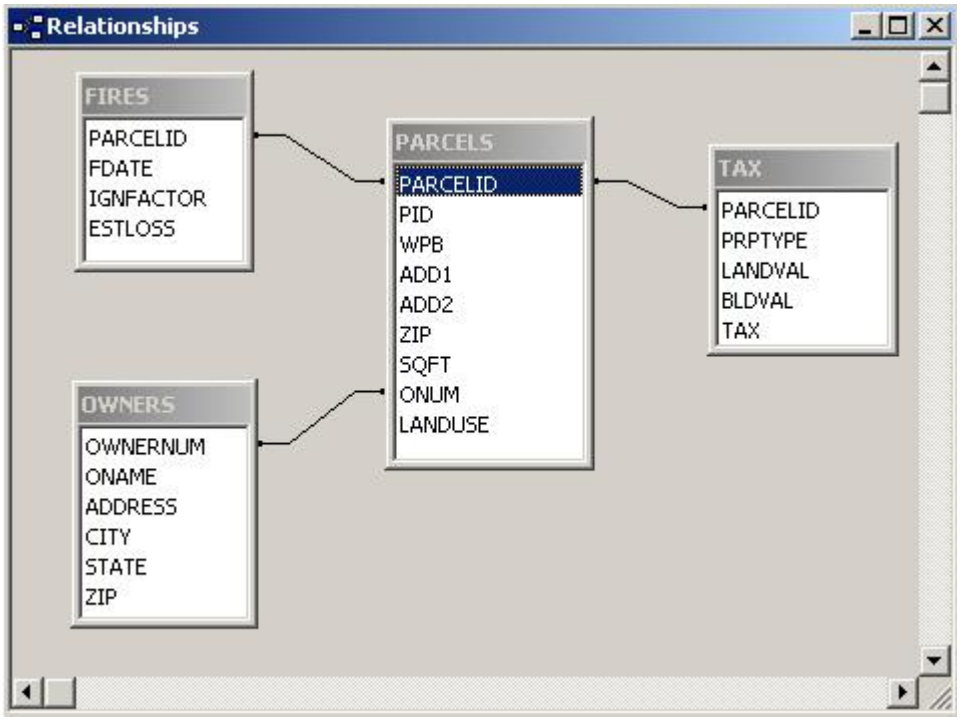
Ohjain (controller) työskentelee koko ajan näkymän ja mallin välillä. Ohjain ottaa vastaan komentoja mallilta ja näkymältä. Ohjain siirtää myös tietoa mallin ja näkymän välillä. Ohjaimen tehtävä on hakea, kirjoittaa tai poistaa dataa mallista ja lopuksi palauttaa uusi näkymä ohjeiden mukaan. Ohjain kirjoitetaan C# kielellä. Tieto kyselypolusta, merkkijonosta ja pyynnön rakenteesta voidaan suoraan sitoa ohjaimen parametreihin. (Smith & Addie 2019.)

5.3 Tietokanta

Tietokanta on tietokoneella oleva kokoelma tietoa. Tietokantoja tarvitaan tässä projektissa tallentamaan ja luovuttamaan tietoa. Projektissa tietokantoihin tallennetaan kaikki ohjelmaan sisältyvät tiedot kuten tunnukset, salasanat, projektit ja niin edelleen. Tietokantoihin suoritetaan hakuja ja ne keskustelevat ohjelman kanssa luovuttaen ja tallentaen tietoa. Tietokantaan pystytään suorittamaan hakuja, jotta oikea tieto löydettäisiin. Erilaisia tietokantoja on nykyään paljon. Tässä projektissa tietokantana toimii Microsoft SQL.

SQL on kieli relaatiotietokantojen tietojen käsittelemiseksi. Relaatiotietokanta rakentuu tauluista. Tauluissa tiedot esitetään riveillä ja sarakkeilla. Yksittäinen tieto tallennetaan vain yhteen paikkaan. Relaatiotietokantaan tallennetaan myös tieto, miten eri taulut liittyvät toisiinsa. SQL kieltä ei käytetä mihinkään muuhun kuin relaatiotietokantojen käsittelyyn. SQL:n avulla pystytään luomaan tietokantoja, jotka sisältävät tauluja ja taulujen väliset suhteet. Kielen avulla pystyy täyttämään tietokantatauluja tiedolla ja muuttamaan, poistamaan sekä hakemaan tietoja tietokantataulusta. Lisäksi myöntämään lupia tietokantaan ja suojamaan tietokantatauluja ristiriitojen sekä käyttäjien virheiden aiheuttamilta vioilta. Tässä projektissa jokainen tietokannan taulu edustaa yhtä pääryhmää esim. käyttäjää tai projektia. Taulut luodaan projektin alussa ja niihin laitetaan projektin alussa arvioidut sarakkeet. Sarakkeet ja tau-

lut voivat muuttua kehityksen myötä. SQL:n kielellä ja ASP.NET:llä on helppoa kehityksen aikana poistaa tai lisätä uusia tauluja tai sarakkeita. (Pieni SQL-opas N.d)



Kuvio 4 Relaatiotietokanta ja taulujen väliset suhteet. (Grayson 2001)

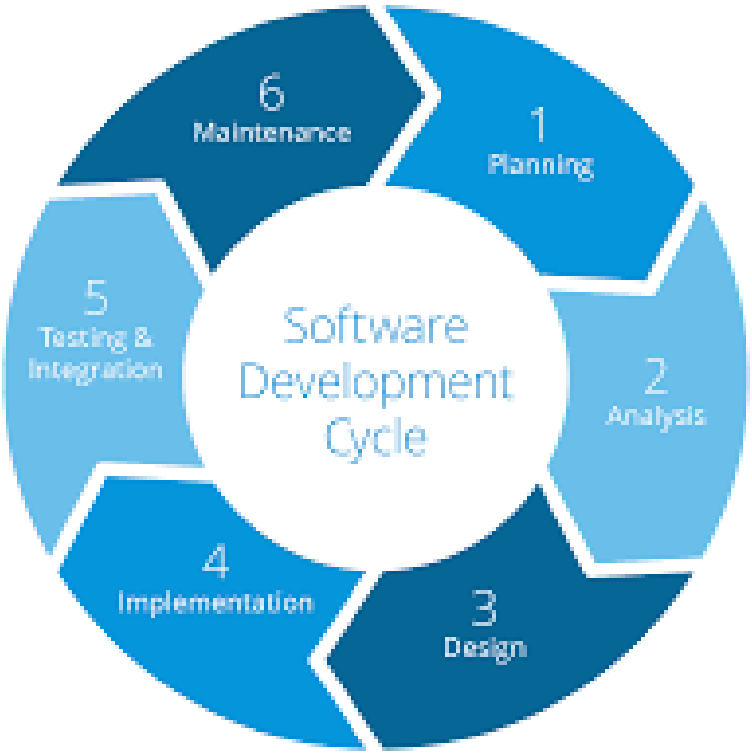
6 Sovellusten kehitysmenetelmät

Ohjelmistotuotanto on nimitys työnteon menetelmille, joita käytetään ohjelmistojen kehityksessä. Ohjelmistotuotanto kattaa ohjelmiston prosessihallinnan ja valmistamisen menetelmät. Siihen sisältyy kaikki ohjelmistonkehityksen vaiheet. Ohjelmistotuotantoon on kehitetty erilaisia menetelmiä, jotta ohjelmia pystyttäisiin kehittämään järjestelmällisesti. Ohjelmistokehityksessä etsitään vastauksia asiakkaiden tarpeisiin, kuinka saadaan aikaan luotettavia järjestelmiä, miten hallitaan kehittämisen eri vaiheita ja kuinka sovellukset tehdään aikataulun mukaisesti.

Tämän projektin ohjelmistonkehitykseen haluttiin löytää mahdollisimman toimiva ja käytännöllinen menetelmä helpottamaan ohjelmointia ja kehitystä. Oikean menetelmän löytäminen oli tässä projektissa erittäin tärkeää, koska uutta sovellusta ei tehty

täysin vanhalle pohjalle vaan ohjelmaan haluttiin löytää uusia toimintoja. Uudesta ohjelmistosta haluttiin myös poistaa mahdollisimman paljon turhia toimintoja tai tehdä niistä automaattisia, jottei käyttäjän tarvitse niitä itse tehdä, kuten aiemmin. Tähän projektiin menetelmäksi valittiin ketterä kehitys, koska ohjelman toimintojen toimivuudesta haluttiin saada mahdollisimman paljon tietoa käyttäjältä. Projektin alussa ei määritellä ohjelmiston lopullisia ominaisuuksia vaan niitä kehitetään projektin edistyessä. Ketterä kehitys sopi myös tähän projektiin, koska uusista toiminnallisuuksista haluttiin saada mahdollisimman nopeasti esiteltäviä versioita ennen kuin niitä jatkokehitettäisiin.

Monissa eri kehitysmenetelmissä järjestelmä määritellään ennen projektin aloittamista, mutta ketterä kehityksessä näin ei tarvitse tehdä. Ohjelmoinnissa ketterä kehitys on mahdollista, koska toiminnallisuuteen voidaan tehdä suuria muutoksia projektin kehityksen aikana. (Järvenpää 2020.) Ketterä kehityksessä tärkeää on kommunikointi. Kommunikoinnin avulla pystytään ottamaan kantaa toteutukseen, menetelmään ja sitä kautta lopputulokseen. Hyvän kommunikoinnin avulla säästytään ylimääräiseltä työltä tilanteissa, joissa tehdään muutoksia projektiin. Työ paloitellaan ketterässä kehityksessä pieniin osiin. Jokainen projektin osa testataan ennen kuin se liitetään lopulliseen työhön. Asiakas on mukana projektissa koko kehityksen ajan. Ohjelman kehittäjät kommunikoivat koko ajan asiakkaan kanssa tuloksista ja pyrkivät kohti samaa päämäärää. Projektin tavoite muokkaantuu koko kehityskaaren aikana asiakkaan (käytetään myös nimitystä product owner) ja kehittäjän kanssa. Asiakas ja kehittäjä yhdessä ymmärtävät kehitettävän tuotteen päämäärän selvemmin ja paremmin. Ketterän kehityksen asiakkaalle pystytään luomaan hyvä tuote, vaikka asiakkaalla ei olisi selvää kuvaa tuotteesta projektin alkaessa. (Ketterä kehitys 2015.) Ketterä kehityksessä on erilaisia prosessikuvauksia, joilla esitetään prosessikuvaukseen liittyviä elementtejä. Erilaisia prosessikuvaus malleja on esimerkiksi Scrum-menetelmät ja extreme programming.



Kuvio 5 Ketterä kehityksen jakso. (Van Der Hoek 2018.)

Scrum on toimintamalli, jonka mukaan projektia ohjataan. Scrum -menetelmä keskittyy projektin vaiheistamiseen ja kontrollointiin. Scrum -menetelmässä määritellään tietyn pituiset kehitysjaksot (sprint). Kehitysjaksot ovat pienimuotoisia projekteja, jonka aikana tehdään sovitut tehtävät. Scrum -menetelmässä henkilöillä on tietty roolitus. Erilaisia rooleja on product owner, kehitystiimi, scrum master, backlog ja sprint backlog. Product owner on tiimissä asiakkaan ääni, jonka tehtävä on maksimoida tiimin arvo asiakkaalle. Kehitystiimi nimensä mukaan vastaa tuotteen kehityksestä. Scrum masterin tehtävä on hallita tiimin tapaamisia, valmentaa ja pitää yllä tiimin ilmapiiriä. Backlogin tehtäviin kuuluu uusien ominaisuuksien ja bugikorjausten määrittely. Sprint backlog on tiimi, joka suorittaa tehtäviä yksilöllisten taitojen mukaan. (Tolvanen N.d.)

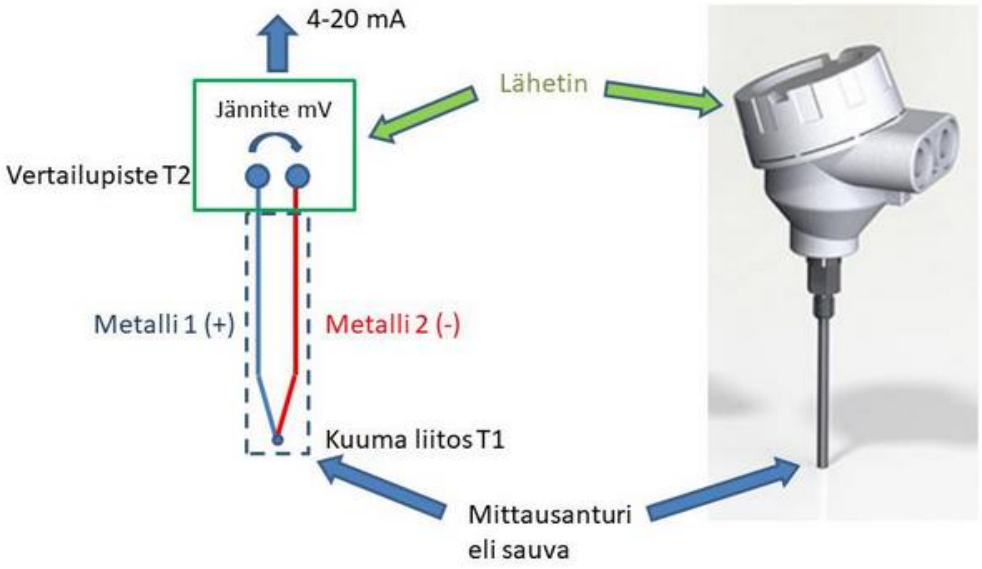
Extreme programming on yksi ketterä kehityksen prosessikuvausmalleista. Exteme programmingissa on neljä tärkeää arvoa kommunikointi, yksinkertaisuus, palaute ja

rohkeus. Extreme programming sisältää neljä eri toimintaa: ohjelmointi, testaus, kuunteleminen ja virheiden etsintä. Extreme programmingissä ohjelmiston tärkeät ominaisuudet tehdään ensimmäiseksi. Uusia ominaisuuksia tuodaan lisää kehityksen aikana. Kehityksen aikana kehittäjät tekevät yhden ominaisuuden kerrallaan. Kehityksen aikana ohjelmaa testataan koko ajan toimivuuden varmistamiseksi. (Pearman & Goodwill 2006.)

7 Betonin lämpötilan mittaus

Tämän sovelluksen toiminnan kannalta on tärkeää betonivalun lämpötilan kerääminen kentältä ohjelmistoon. Lämpötilatiedon lisäksi tarvitaan päivä ja kellonaika, jotta betonin lujuus pystytään laskemaan. Lämpötilan mittaamisessa ja tiedon hakemisessa ei saa olla virheitä lujuuden varmistamiseksi. Tärkeimpiä kohtia toimivuuden kannalta on anturin oikea sijoittaminen betonivaluun, lämpötilakantojen ohjelmiston toimivuus ja lämpötilamittarin toimivuus.

Betonin lämpötilan mittaus toteutetaan etäluettavalla dataloggerilla. Dataloggerissa on termolankapari, jonka avulla saadaan betonivalusta lämpötilatieto. Termolankaparissa on kahdesta eri metallista tai metalliseoksesta valmistettua metallilankaa. (Miten toimii TE-anturi (termoelementti)? N.d.) Kun langat yhdistetään päistä, syntyy termopari. Yhteen laitetun pään lämpötila on eri kuin vertailupisteen. Lämpötilan ollessa erisuuri lämpösähköinen ilmiö aiheuttaa langanpäihin jännite-eron. Lämpösähköstä syntyvä jännite on todella heikko, joten sitä pitää vahvistaa lähettimessä. (Termoelementti eli termopari 2019.) Erilaisia termolankapareja on monia, joista jokaisella on oma mittausalue. Mittauksen alkaessa termopari työnnetään betonivaluun. Matalalämpöiset termoparit ovat halpoja ja termopari on helppo itse liittää, joten mittauksen päätyttyä termoparin langat voidaan katkaista betonin sisään.



Kuvio 6 Kuva termolankaparista dataloggerisssa. (Termoelementti eli termopari 2019.)

Dataloggeri on työkalu, jolla pystytään seuramaan ja tallentamaan mitattavaa muutosta ajan kuluessa. Dataloggerin avulla pystytään mittamaan erilaisia muutoksia kuten lämpötila, kosteus sekä säteily riippuen anturista ja toimivuudesta. Dataloggerissa on mikroprosessori, muisti, ulkoinen tai sisäänrakennettu sensori ja virtalähde. Dataloggerit tallentavat tietoa joko sisäiseen muistiin tai pystyvät lähettämään sen verkon tai muun ulkoisen lähteen kautta palvelimelle. Dataloggereissa pystyy säätämään mittauksen ajanjaksoa. (Data Loggers 2018.)

8 Projektin Lähtökohdat

Projektin tavoitteena oli saada verkkopohjainen sovellus, jossa pystyttäisiin hallinnoimaan käyttäjiä, mittareita, yrityksiä, projekteja ja mittauspisteitä. Projektia lähdettäessä suunnittelemaan tehtiin käyttäjätarinoita sovelluksen lopputuloksen esille tuo-

misen helpottamiseksi asiakkaalle kuin myös ohjelmoijille. Projektia aloitettaessa valmiina olivat tarvittavat työkalut: tietokone, ohjelmointialustat, palvelimet sekä dokumentaatiot tarvittavista työkaluista ja fyysisistä laitteista kuten dataloggerit.

Vimma = ohjelmistotoimittaja ja ohjelmistotuki, "Superkayttaja"
Betsku Oy = Valmisbetonitehdas, tarjoaa ohjelmistoa asiakkaille
Pete Betsku = Valmisbetonitehtaan betoniasiantuntija, yritystuki, "Yllapitaja"
TahtiRaksa Ab = Rakennusliike/rakennusurakoitsija/betonointiyrittäjä
Jaakko Tahti = Rakennusliikkeen it-tuki, vastaa rakennusliikkeen järjestelmistä, "YritysPitaja"
Peku Tahti = Rakennusliikkeen työmaainsinööri/projektin/valujen vastaava, "ProjektiPitaja"
Kampus rakennus = rakennustyömaa, jossa Peku
Maalattia=Betonivalu, josta tietoja mitataan

Ylemmällä roolilla on aina myös alemman roolin oikeudet

1. Pete soittaa Vimma: “pitäisi saada se teidän lujuudenkehitys softa”
2. Vimma pyytää Petea rekisteröitymään sovellukseen netissä
3. Pete tekee työtä käskettyä ja rekisteröityy järjestelmään
4. Pete ei näe, kun info sivun koska ei ole muita oikeuksia
5. Pete lähettää kyselysivulta ikkunasta viestin, että hoidettu
6. Vimma lisää yritystiedot, ja Petelle oikeuden “Yllapitaja” ja yrityksen tiedot “Betsku Oy”
7. Peku soittaa Petelle, että meillä on tämä valu lähdössä, pitäisi mitata dataloggereilla lujuuden kehitystä.
8. Pete tsekkaa, ettei TahtiRaksalla ole vielä tietoja järjestelmässä
9. Pete pyytää Pekua rekisteröitymään järjestelmään ja kysyy, kuka on Tahtiraksan järjestelmävastaava
10. Peku kertoo Tahti Raksa It vastaava on, Jaakko Tahti, joten Pete soittaa Jaakolle, kertoo mikä on tilanne, sopii asiat ja pyytää Jaakkoa rekisteröitymään järjestelmään
11. Jaakon rekisteröidyttyä, Pete lisää yritystiedot ”TahtiRaksa”, Jaakolle yritystiedot ”TahtiRaksa” ja lisää rooliksi “yritysylläpitäjä”
12. Nyt kun yritys lisätty, lisätään myös Pekulle yritystiedot ”TahtiRaksa” ja rooli ProjektiPitaja
13. Nyt Jaakko voi lisätä omalle yritykselle projektin “KampusValu” jossa Peku on vastaava, Lisätä Pekulle tuon projektin (Pekulla voi olla myös muita projekteja).
14. Kun valu on alkamassa, Peku kilauttaa Petelle ja pyytää dataloggeria.
15. Pete toimittaa sen ja Lisää dataloggerin järjestelmään ja kiinnittää sen “TahtiRaksalle” sovituksia ajaksi
16. Kun valu on alkamassa, Peku käy lisäämä uuden valun ”Maalattia” (voi olla useita valuja) ja käy liittämässä siihen dataloggereita 1-n kappaletta (nämä lisätty Tahtiraksa Ab”:lle ajaksi,)
17. Peku Määrittelee Valulle aloitusajan (mietitään myöhemmin betoni)
18. Peku seuraa dataloggeri mittausten antamia kauniita lämpötila käyriä valun aloitushetkestä alkaen
19. 3 päivän ja 4h jälkeen Peku huomaa, että betonille on tullut riittävästi kypsyyttä, tulostaa sivun ja sulkee lehden

Kuvio 7 Käyttäjäkuvaus ohjelmasta

Projektille raakana mallina toimi vanha sovellus, josta otettiin mallia sovelluksen rakenteista ja toiminnoista. Sovellus tehtiin uudestaan alusta loppuun ja vanhan sovel-

luksen koodeja ei pystytty hyödyntämään tässä projektissa. Vaikka sovellusta lähdettiin kehittämään vanhan ulkoasun ja toimintojen mukaan, pyrittiin uutta sovellusta tekemään käyttäjäystävällisemmäksi ja tehokkaammaksi. Uuteen sovellukseen lisättiin tarvittaessa uusia toimintoja sekä vanhoja toimintoja poistettiin tai parannetiin riippuen niiden toimivuudesta ja tarpeellisuudesta.

Projektiin valittiin Visual Studio ohjelmistokehitystyökalu ja sieltä löytyvä ASP.NET -ohjelmistokehys. Visual Studiossa saatiin käyttöön kaikki projektissa tarvittavat työkalut yhdeksi kokonaisuudeksi. Visual Studiosta ja ASP.NET:stä oli myös laajat dokumentoinnit ja yhteisön keskustelupalstat, joista sai helposti tarvittaessa apua projektin ongelmiin.

8.1 Ohjelmointikielet

Ohjelmointikielinä toimi ASP.NET:ssä C#, HTML, CSS ja Javascript. ASP.NET:n hyviä puolia on sen tehokkuus ja nopeus suorituskyvyssä. Nopeutta ASP.NET:lle tuo sen kääntäminen, joka on tehtävä vain kerran toisin kuin tulkituissa koodeissa missä se suoritetaan joka kerta. ASP.NET käyttää käännettyä koodia mikä on skaalautuvampi ja johdonmukaisempi. (https://existek.com/blog/aspnet-vs-php-which-is-better-for-web-development/) Kehittäminen on tehty mallien avulla pieniksi osiksi, minkä myötä projektia on helpompi hallita ja yhdistellä. Välimuistijärjestelmä on helppokäyttöinen ja tehokas. Välimuistijärjestelmän avulla pystytään parantamaan palvelin sovelluksen suorituskykyä ja skaalautuvuutta. (The Good and the Bad of .NET Framework Programming 2020.) ASP.NET:ä pystytään kehittämään integroidussa kehitysympäristössä. Visual Studio IDE on integroitu kehitysympäristö, joka tarjoaa kaikki työkalut ohjelman kirjoittamiseen, julkaisemiseen, testaamiseen ja hallitsemiseen. ASP.NET sisältää paljon eri kirjastoja ja paketteja, joilla pystytään tekemään toiminnallisuuksia ohjelmistoon helposti ja vaivattomasti.

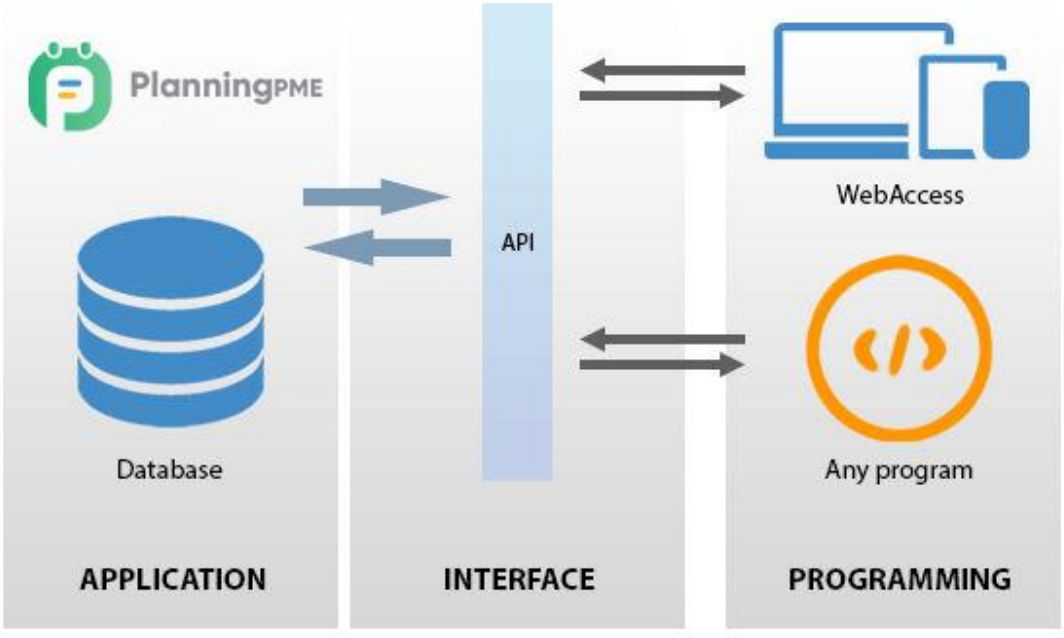
ASP.NET:n kehityksen huonoja puolia on sen kulut. ASP.NET käyttää SQL -palvelinta mikä on tehokas, mutta kallis tapa tietokannan ylläpitoon. Microsoftin kehittämä ISS -palvelin, mitä ASP.NET tukee on kallis. Frontendin tekeminen hienoksi on hieman vaikeampaa kirjoittaa koodilla, kuin graafisesti vain naputella palikoita paikoilleen.

Virheiden määrä ohjelman koodissa kasvaa helposti isommaksi, kun ASP.NET:ssä pitää kirjoittaa itse koodia eikä vain vetää ja pudottaa komponentteja ja toiminnallisuuksia. (ASP.NET Is Almost Certainly The Wrong Technology For Your Project N.a.)

ASP.NET:n etu ovat valmiina olevat palvelimet, mitkä ovat jo käytössä, joten siitä ei tule lisäkustannuksia. Kehittäminen nettiohjelmistossa olisikin helpompaa liimaa ja leikkaa tavalla esim. Wordpress:n pienissä ohjelmistoissa, missä tärkein on ulkoasu mutta ei toiminallisuus. ASP.NET soveltuu paremmin tähän projektiin sen kokoluokan ja monipuolisuuden takia. ASP.NET:n avulla pystytään tekemään enemmän monimutkaisia laskentoja ja parantamaan tehokkuutta.

8.2 Rajapinnat

Sovelluksessa haetaan lämpötilatietoa järjestelmien rajapinnan kautta. Rajapinta on kahden eri sovelluksen välinen keino saada kyselyt onnistumaan. Rajapinta on tehokas ja yksinkertainen tapa kysellä tietoa kolmannelta osapuolelta. Rajapinta pitää olla aina määritelty sovelluksessa, mistä tietoa haetaan.



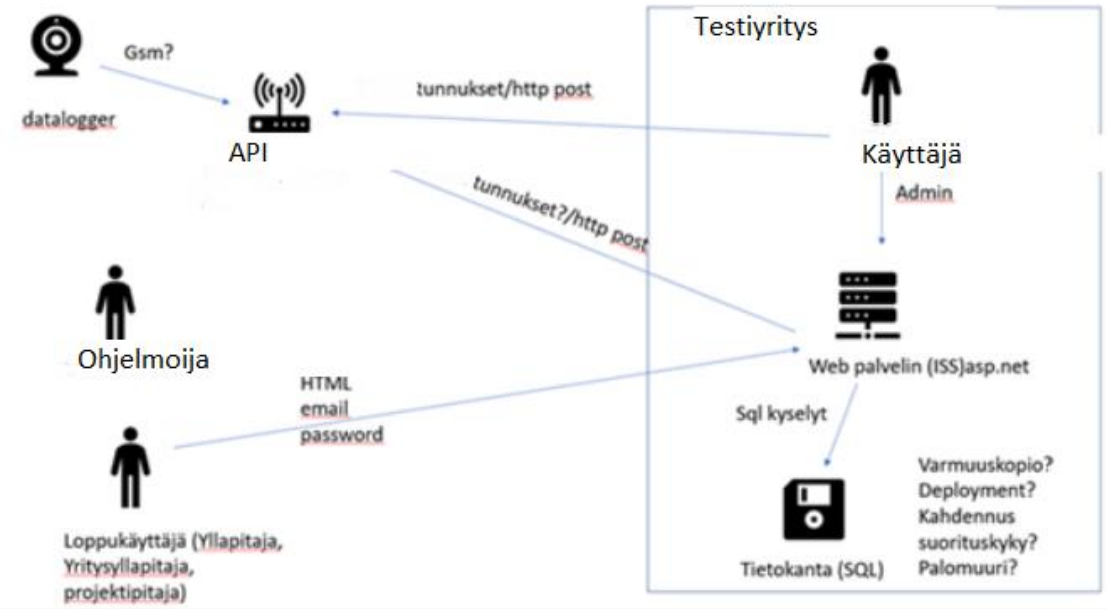
Kuvio 8 Apin toiminta. (Definition and role: what is an API? N.d.)

Projektissa on käytössä etäluettavat dataloggerit, jotka lähettävät tietoa kolmannen osapuolen palvelimille. Kolmannen osapuolen palvelimelta saadaan tieto mittauspisteistä API:n avulla. API on ohjelmointirajapinta, joka sallii eri ohjelmistojen välisen kommunikoinnin. API mahdollistaa tiedon siirtämisen automaattisesti eri ohjelmien välillä. (API – Mikä on API? N.d.) API keskustelee http protokollalla. Http protokollassa lähetetään pyyntö palvelimelle, joka lähettää vastauksen asiakkaalle. Pyyntökutsussa on neljä erilaista päätyyliä: GET, POST, PUT ja DELETE. GET:n avulla pystytään saamaan tietoa toisesta sovelluksesta. POST:n avulla halutaan luoda tietoja ja lisätä ne toiseen sovellukseen. PUT:lla muutetaan tietoja toisessa sovelluksessa. DELETE:n avulla poistetaan tietoja toisesta sovelluksesta. Pyynnössä pitää tietää tarkasti mitä tietoja vastaanottava sovellus haluaa, jotta pyyntö onnistuu. Pyynnön onnistuessa toinen sovellus lähettää joko halutun datan tai lisätietoa, mistä haluttu data voidaan löytää. (Macoveiciuc 2020.)

8.3 Palvelimet

Projektissa palvelimet olivat valmiiksi määritellyt. Sovelluksen omalle tietokannalle palvelimeksi oli määrätty Microsoft SQL server ja ohjelmalle palvelimeksi oli ISS server. Sovelluksenkehityksessä palvelut ja työkalut olivat Microsoftin, joten yhteensopivuus oli hyvä.

Visual Studiossa pystyy tekemään oman SQL serverin vaivattomasti sovelluksen kehittämiseksi ja saman tietokannan pystyy helposti siirtämään oikealla palvelimelle. Myös sovellusprojektin lataaminen ISS palvelimelle on yksinkertaista graafisen liittymän avulla, mikä on rakennettu myös Visual studion sisään.



Kuvio 9 Arkkitehtuurikuva sovelluksesta.

8.4 Muutoksen hallinta ja varmentaminen

Muutoksen hallinta ja varmentaminen on tärkeää sovellusta tehtäessä. Muutoksen hallinnan ja varmentamisen avulla pystytään seuraamaan projektin vaiheita ja luotettavasti palaamaan projektissa vanhoihin muutoksiin, mikäli uudet muutokset sovelluksessa eivät ole toimivia tai tarpeellisia. Varmentamisen avulla pystytään todentamaan projektin versioiden tallennus ja projektin häviämisen estäminen, mikäli kehityksessä käytetty tietokone tai muut työkalut hajoavat tai menevät jumiin.

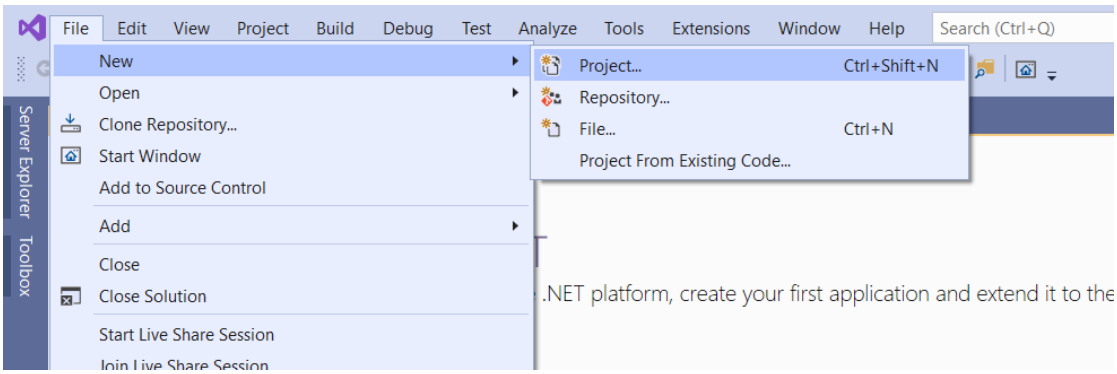
Muutoksen hallinta ja varmentaminen hoidettiin tässä projektissa GitHub:n avulla. GitHub on ilmainen verkkosivu, joka tarjoaa palvelua versionhallintaan. GitHub:ssa pystyy versionhallinnan lisäksi seurata bugeja, tehtäviä, integraatiota ja projektin viestittämistä. GitHub:ssa pystyy tekemään yksityisiä ohjelmavarastoja, johon pystyy lisäämään muita hallinnoijia. Lisäämällä ohjelmavarastoon muita hallinnoijia, pystyy projektin suorittamaan helposti myös ryhmänä. (The tools you need to build what you want. N.d.)

GitHub valittiin tähän projektiin versionhallintatyökaluksi sen helppokäyttöisyyden ja monipuolisuuden takia. Visual Studioon saa GitHub:n lisäosan, jonka avulla on helppo ladata uusia versioita GitHub:lle. GitHub:sta on myös helppo ladata viimeisin versio sovelluksesta mille vain tietokoneelle missä on Visual Studio.

9 Projektin ohjelmointivaiheet

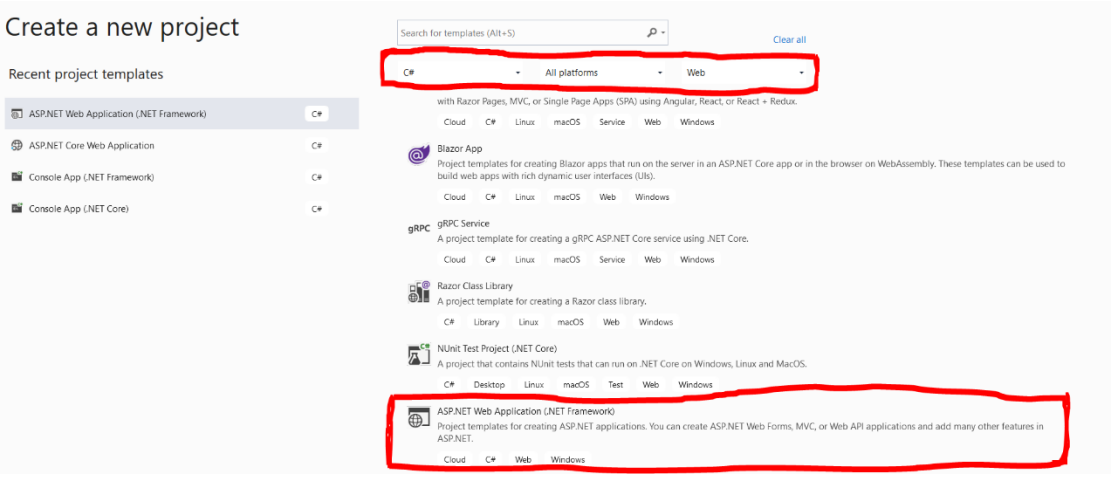
9.1 Ohjelman luominen ja versionhallinta

ASP.NET MVC -projekti luodaan ja perusasetukset laitetaan kuntoon Visual Studiossa. Esimerkit on tehty tyhjällä projektilla. ASP.NET -projekti alustetaan luomalla uusi projekti. Projekti luodaan VisualStudiossa File->New->Project, jonka jälkeen avautuu New Project -ikkuna.



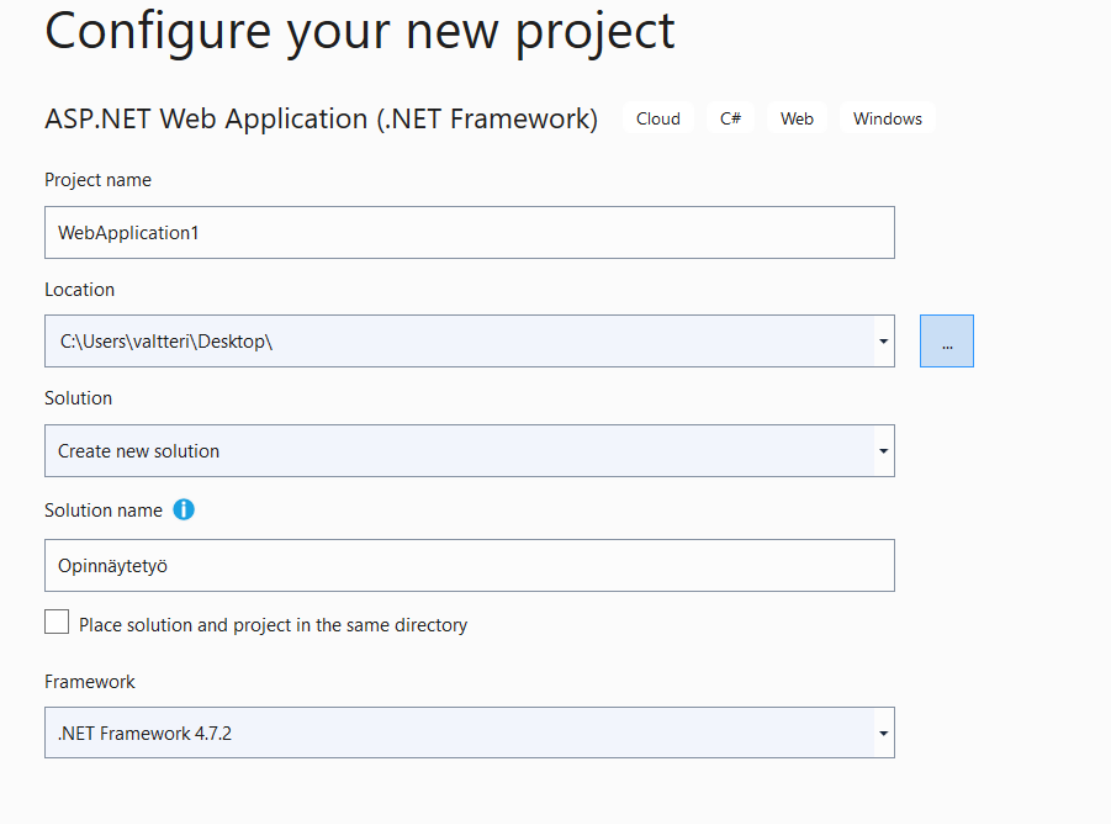
Kuvio 10 Uuden projektin luonti.

New Project -ikkunasta valitaan ohjelmointikieleksi joko C#, VisualBasic tai F#. Valitaan viereiseen alasvetolistaan Web, jotta näkyviin saadaan vain nettipohjaiseen sovelluskehitykseen tarkoitetut mallit. ASP.NET Web Application (.NET Framework) on tämän sovelluksen tekemiseen valittu työkalu.



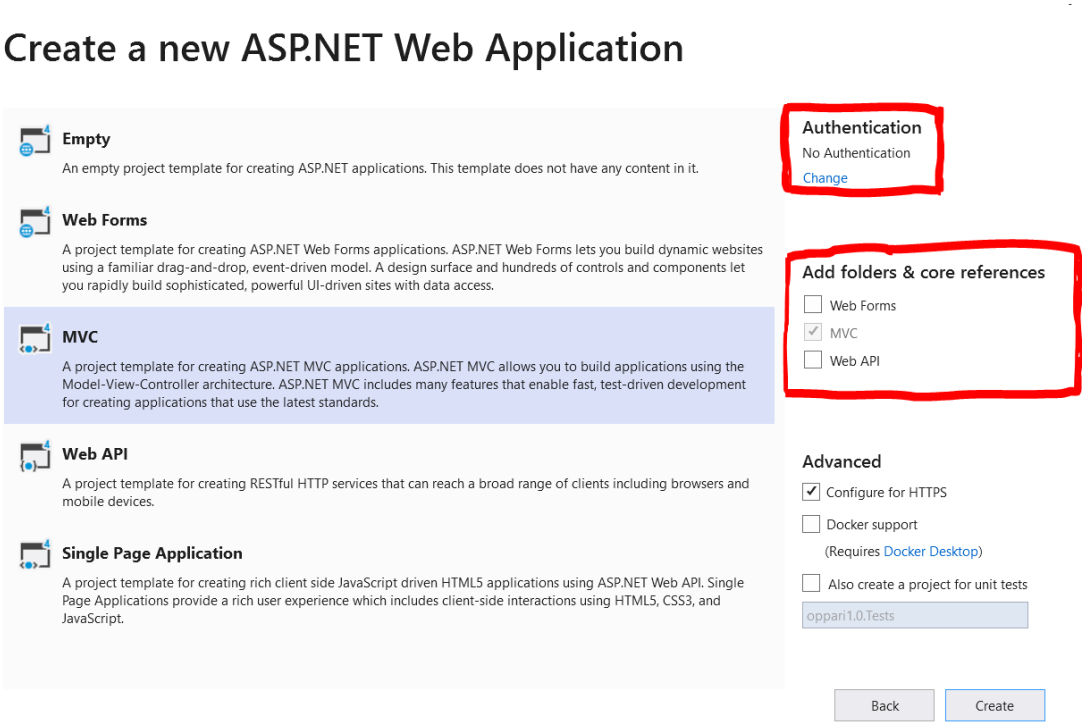
Kuvio 11 Projektin mallin valinta.

Projekti nimetään Project Name kohdasta. Projektin tallennuspaikkaa voidaan muuttaa Locationista. Create luo projektin.

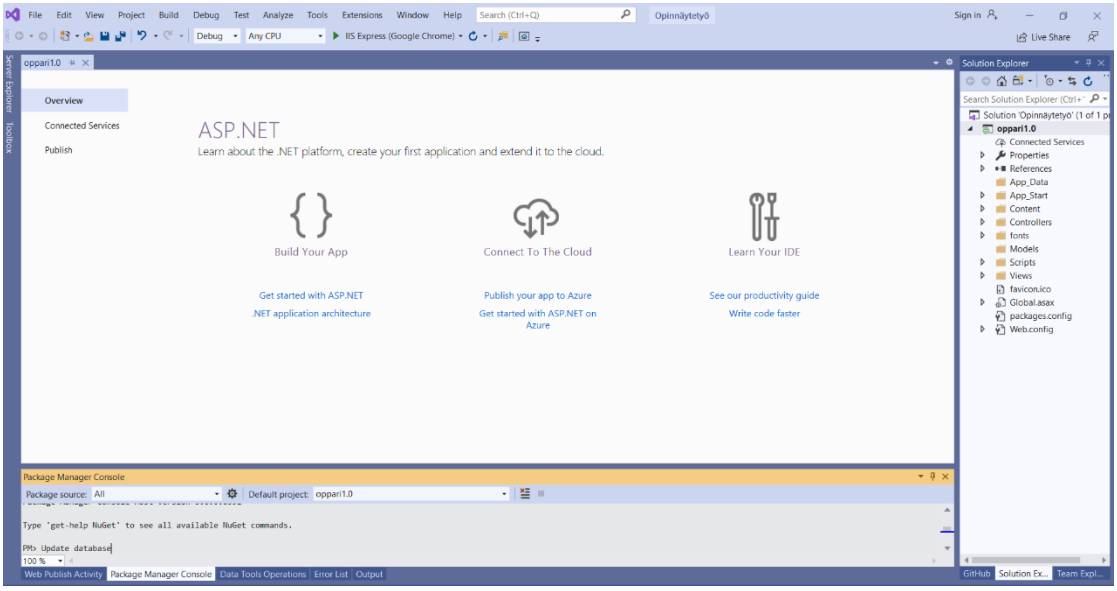


Kuvio 12 Projektin konfigurointi.

New ASP.NET Web Application -valintaikkunasta saadaan MVC arkkitehtuuri käyttöön. Authenticationista voidaan automaattisesti luoda projektiin käyttäjän kirjautuminen, ulos kirjautuminen ja käyttäjänhallinta valitsemalla invidual users.

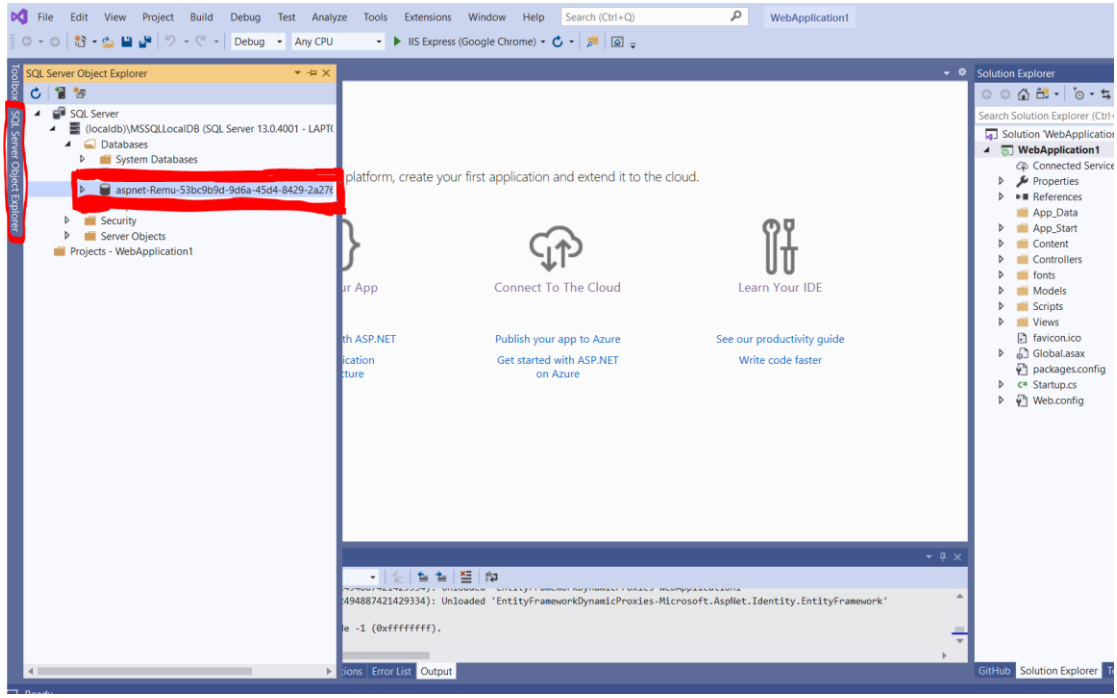


Kuvio 13 MVC -mallin ja tunnistamisen luominen



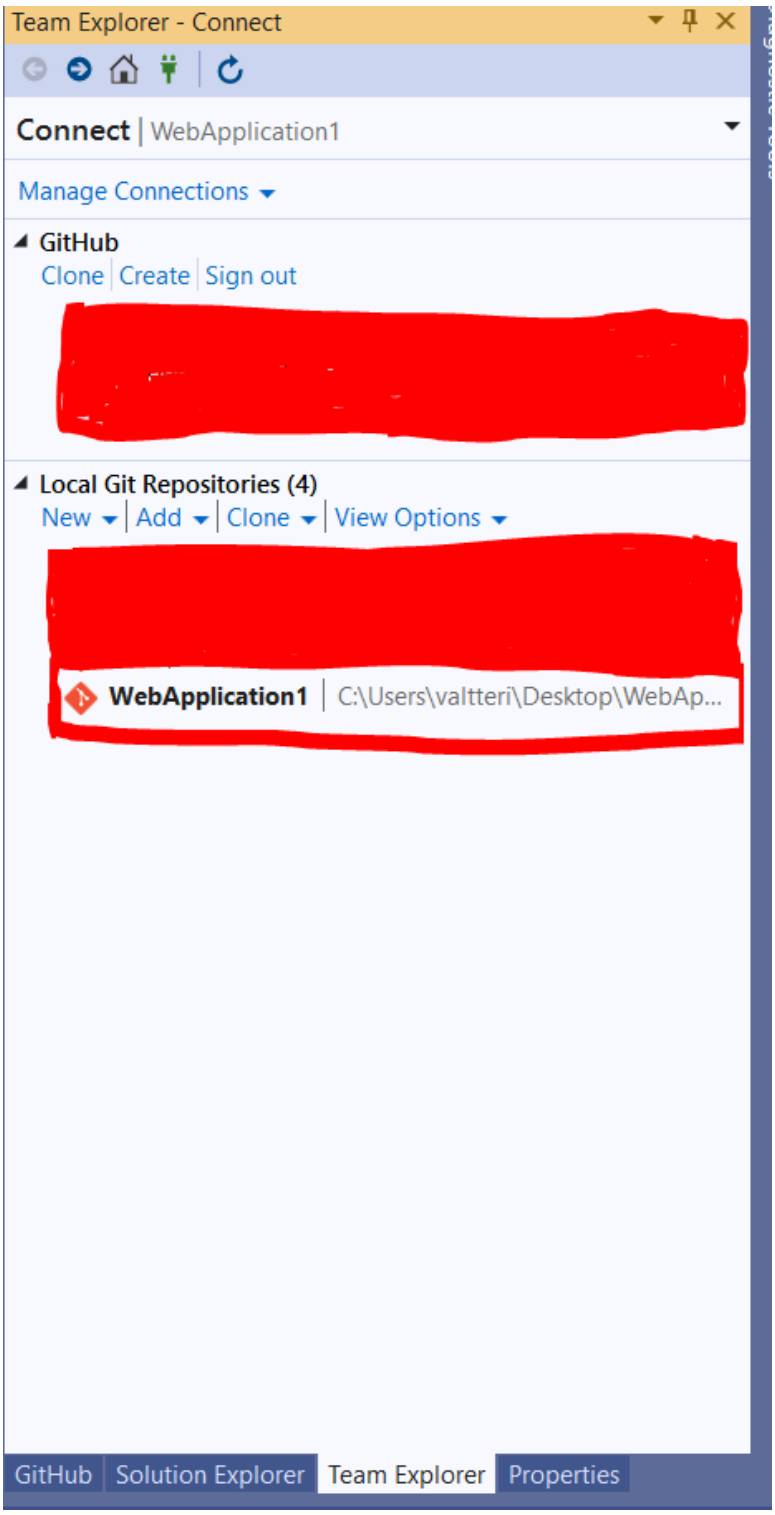
Kuvio 14 Sovelluksen käynnistys onnistuu play -painikkeella.

Ohjelmaa ajettaessa ensimmäistä kertaa, projekti luo itselleen SQL -tietokannan. Tietokannan voi nähdä View -> SQL Server Object Explorer -ikkunasta. Tietokannan avaamalla pystyy näkemään sinne tallennetut taulut.



Kuvio 15 Projektin tietokantatyökalu.

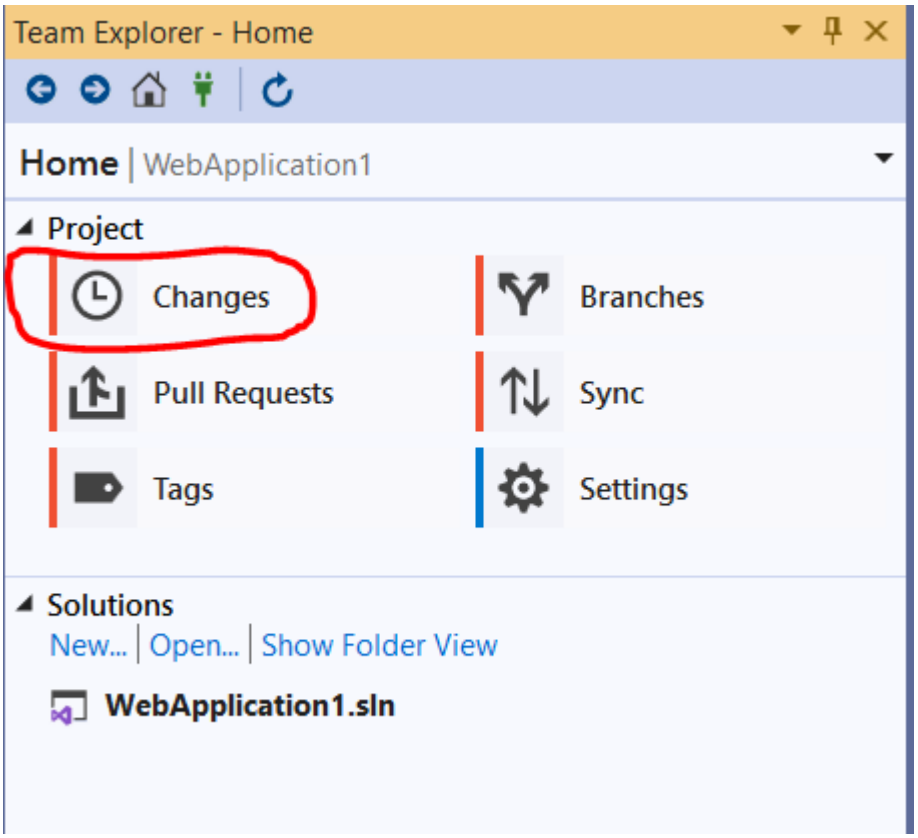
GitHub -sovellukseen täytyy olla luotu käyttäjä, jotta sen pystyy ottamaan käyttöön Visual Studiossa. Visual Studiossa Github:n laajennus lisätään Add to Source Control -painikkeesta. Team Explorerista ladataan ja haetaan projektin versioita Git Hub:sta. Team Explorerista Sync -painikkeella voi valita version hallinta työkaluksi, joko GitHub tai Azure. Valittaessa GitHub sinne pitää kirjautua tunnuksilla. Private Repository valintaruudulla pystyy tekemään arkistosta yksityisen, jolloin vain oikeuden saaneet pystyvät näkemään ohjelman ja lataamaan tai viemään uusia versioita. Asetusten ollessa oikein, tuplaklikkaamalla ohjelman nimeä aukeaa ohjelman versionhallinta.



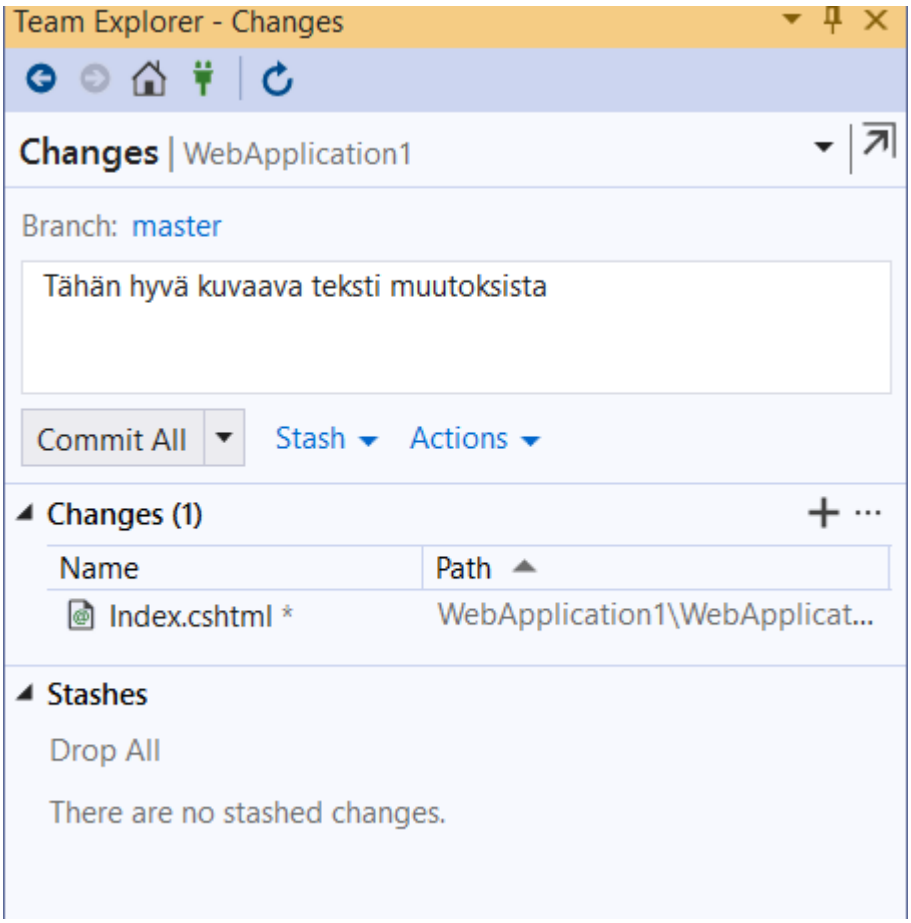
Kuvio 16 Team explorer

Projektin versionhallinta on toiminnassa GitHub:ssa. Seuraavien versioiden valmistuessa Team Exploreriin Changes -napista voi nähdä onko ohjelmaa muutettu. Ohjelman muutokset pystytään viemään GitHubiin Commit All -napista. Tekstikenttään kannattaa kirjoittaa selkeä teksti muutoksista, jotta itselle ja projektinjäsenille tulee

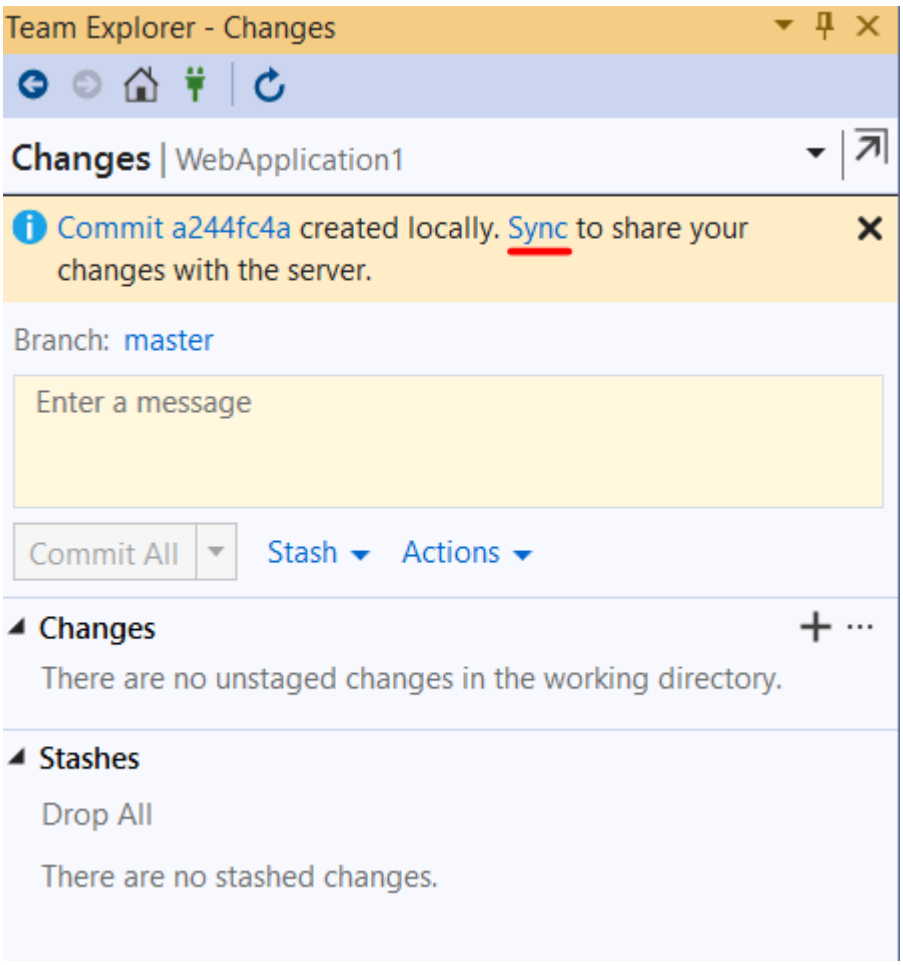
selkeä kuva tehdyistä muutoksista. Sync -napista muutokset synkronoidaan kommentin kanssa. Push -nappista muutokset viedään version hallintaan. Haettaessa GitHubista projektiin jonkun muun tekemiä muutoksia ne haetaan Pull-napilla.



Kuvio 17 Versionhallinnan etusivu.



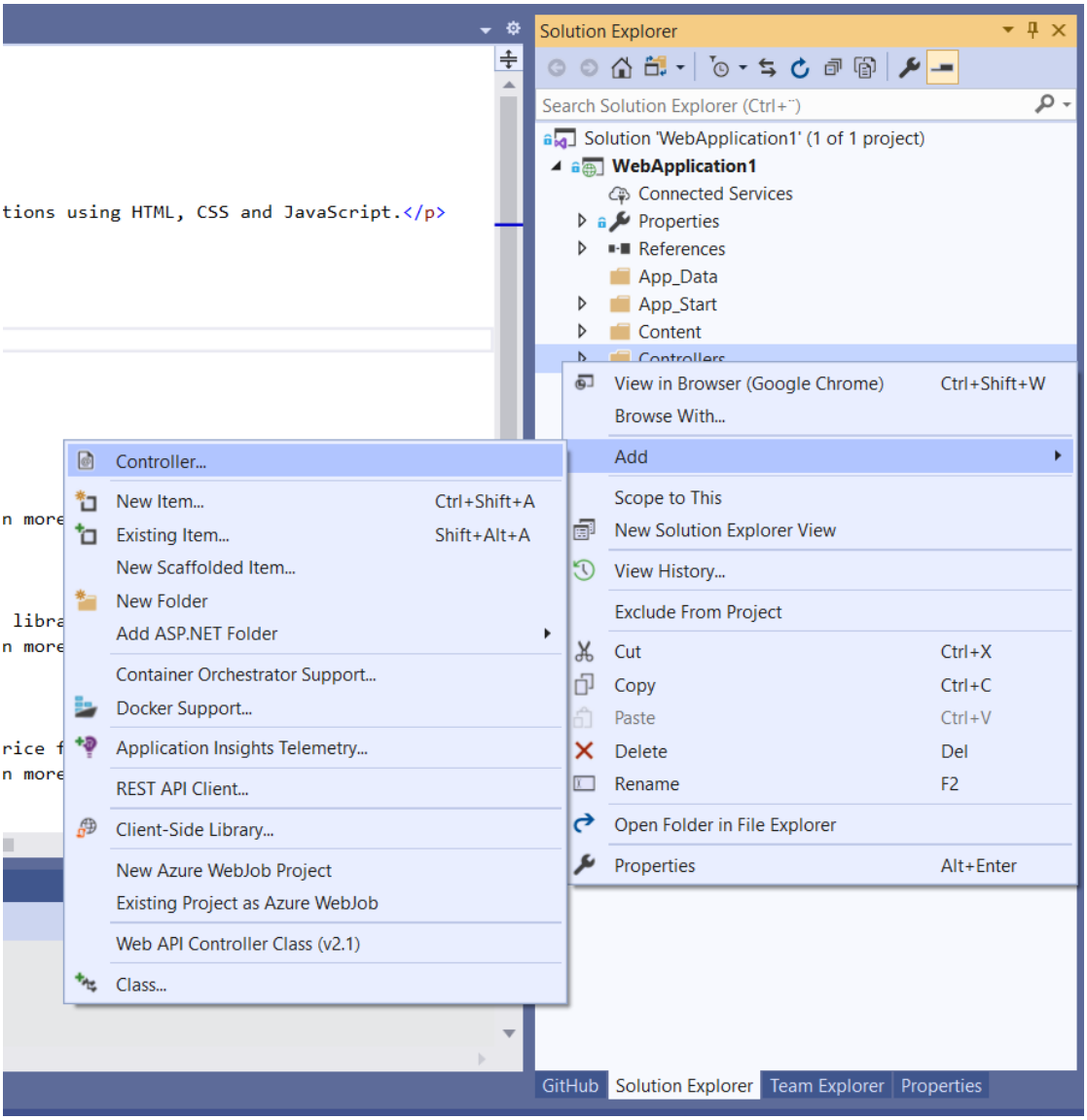
Kuvio 18 Muutoksien tallentaminen.



Kuvio 19 Muutoksien synkronointi palveluun.

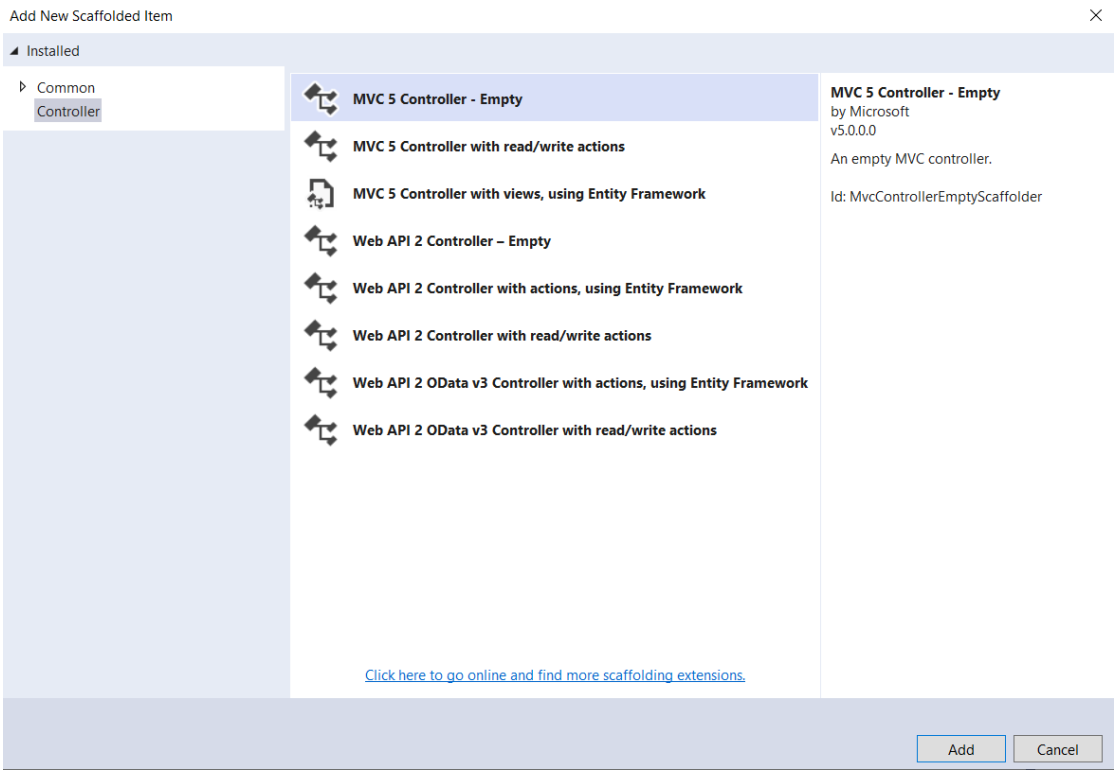
9.2 ASP.NET MVC rakenne ja sen käyttäminen

ASP.NET MVC -mallin rakenne sisältää mallin, kontrollerin ja näkymän. Kontrolleri luodaan painamalla Solution Explorerista Controller -kansiota oikealla, jonka jälkeen avautuu ikkuna, josta Add -> Controller valinnalla pystytään valitsemaan haluttu kontrolleri.



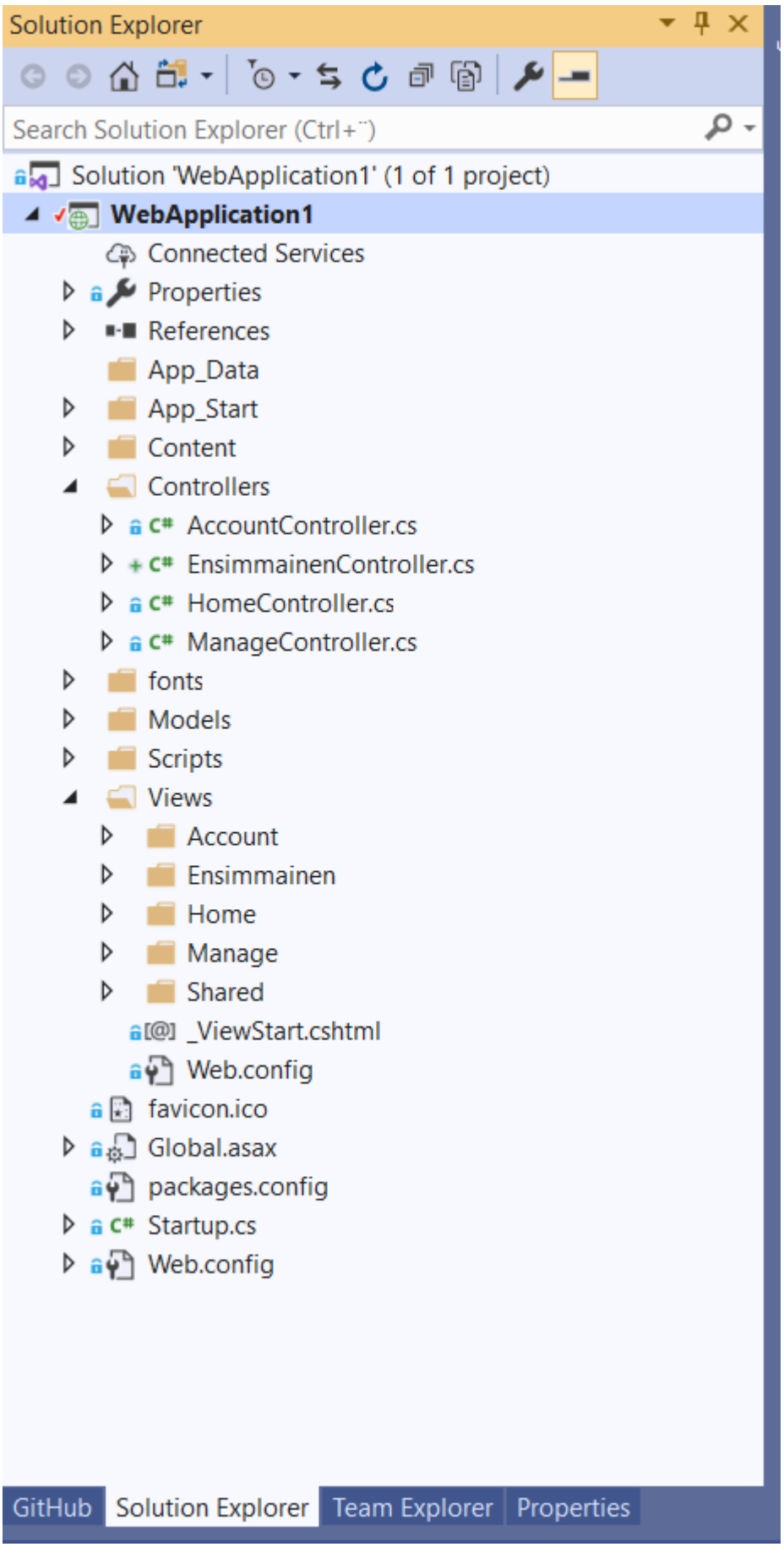
Kuvio 20 Kontrollerin lisääminen.

Seuraavaksi avautuu ikkuna, josta voi valita sopivan kontrollerin. Erilaisia kontrolle-reita on Tyhjä, Kontrolleri luku- ja kirjoitusmetodeilla ja kontrolleri Razor näkymällä, joka sisältää luo, lue, kirjoita ja päivitä metodit. Web API:n kontrollerit hallinnoivat http kutsuja ja pyyntöjä. Nimensä mukaan ne soveltuvat API:n tekemiseen.



Kuvio 21 MVC:n erillaiset kontrollerit.

Ensimmäinen kontrolleri on luotu. Visual Studio luo automaattisesti Views -kansion kontrollerille. Views -kansio on tyhjä sinne ei automaattisesti generoidu tietoa.



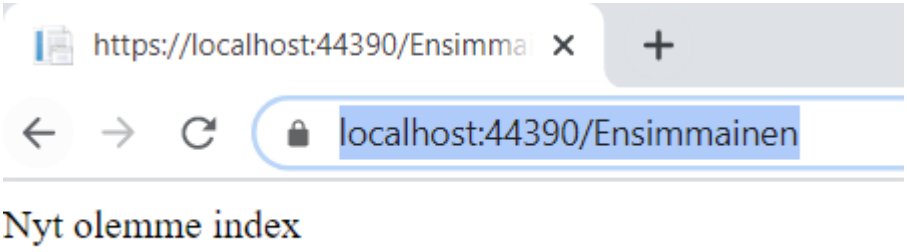
Kuvio 22 Projektin tiedostopuu.

Kuvio 23 Esimerkki kontrollerin valmiista toteutuksesta.

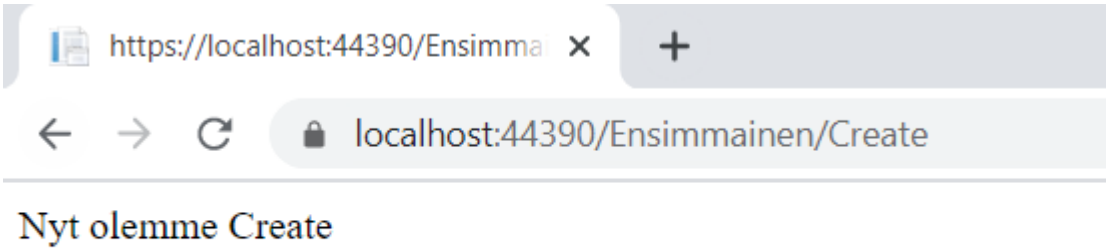
```
public class EnsimmainenController : Controller
{
    // GET: Ensimmäinenkontrolleri
    0 references
    public string Index()
    {
        return "Nyt olemme index";
    }

    // GET: Ensimmäinenkontrolleri/Details/5
    0 references
    public ActionResult Details(int id)
    {
        return View();
    }

    // GET: Ensimmäinenkontrolleri/Create
    0 references
    public string Create()
    {
        return "Nyt olemme Create";
    }
```

Kuvio 24 Metodeihin pystyy muuttamaan tehtäviä.

Kutsuessa tätä kontrolleria käsky menee suoraan Index kohtaan https://local-host:446380/Ensimmainen. Halutessa kutsua Create kutsua se pitää lisätä kontrolle-rin nimen perään. https://localhost:44760/Ensimmainen/Create. Esimerkissä kontrol-leri palauttaa string -merkkijonon eikä kutsu näkymää. Muuttamalla kontrollerilta metodiin Index ja Create return View(), ne kutsuvat Views-kansion näkymiä.

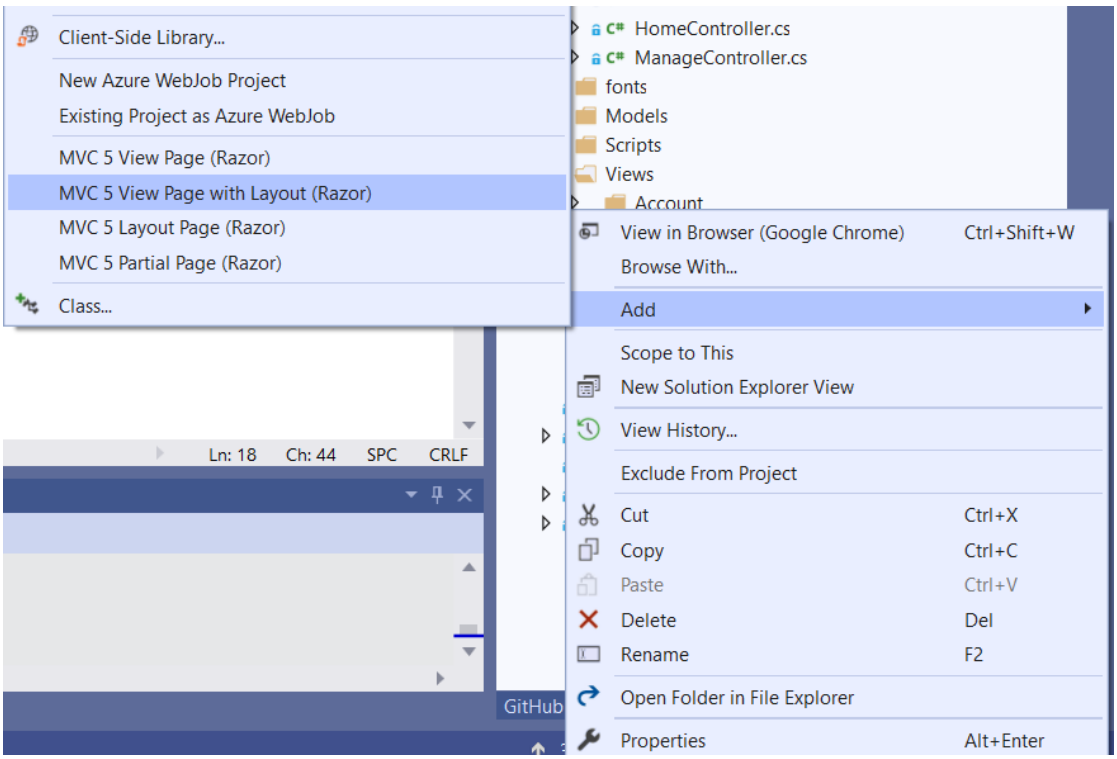


Kuvio 25 Index käyttöliittymänäkymä.

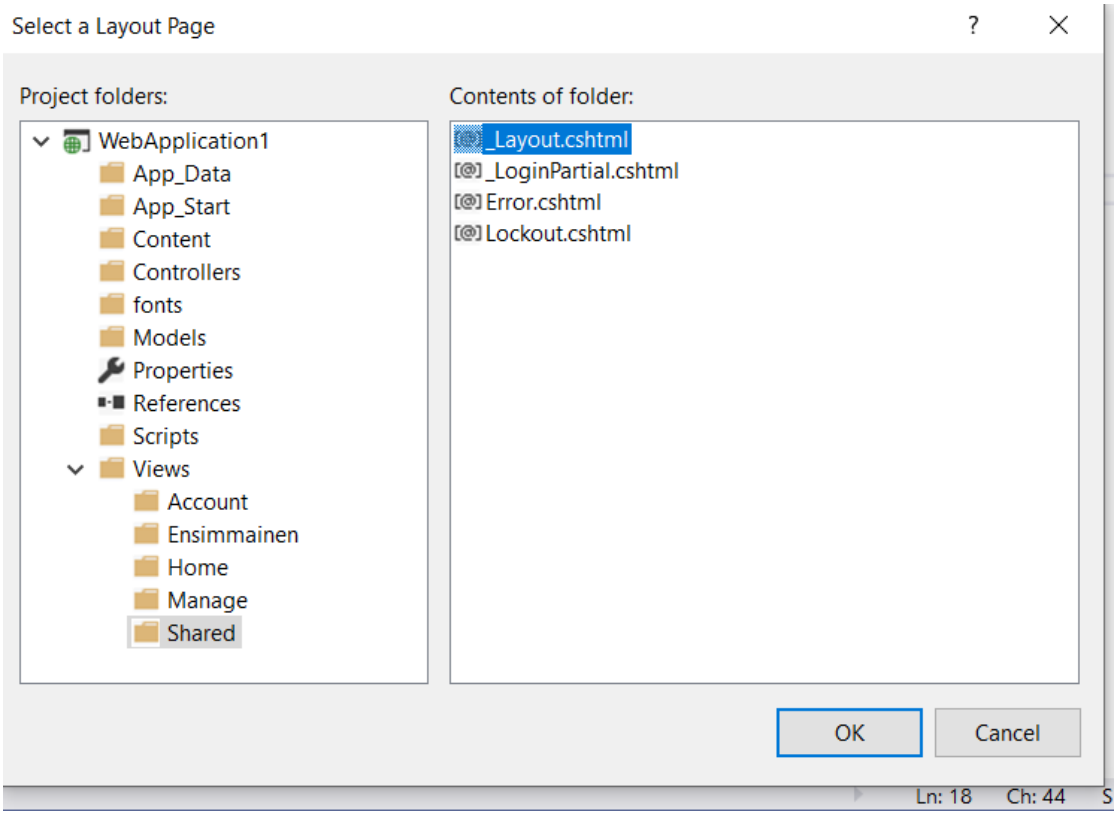


Kuvio 26 Create käyttöliittymänäkymä.

Views -kansioon Index-osoitteelle view:n tekeminen tapahtuu klikkaamalla hiiren oikealla kansiota, jolloin polkua Add-> MVC5 View Page with Layout(Razor) -näkymä luodaan. Nimeksi annetaan sama kuin mikä on kontrollerilla eli tässä tapauksessa Index ja Valitaan Shared -> _layout.cshtml. Tämä views tulee nyt malliksi Index Viewsille



Kuvio 27 Näkymän luonti.

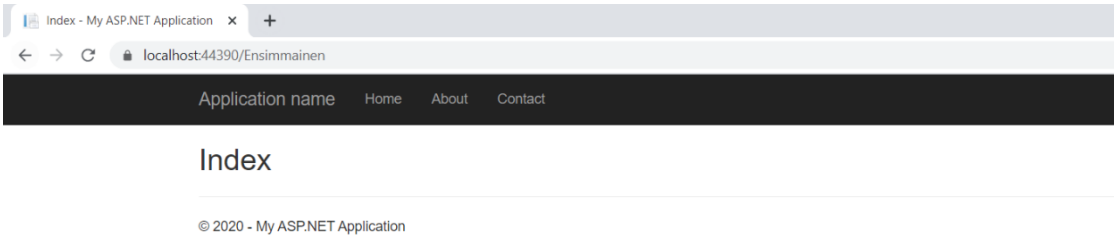


Kuvio 28 Näkymälle valmiiksi tehdyt pohjat.

```
public ActionResult Index()
{
    return View();
}
```

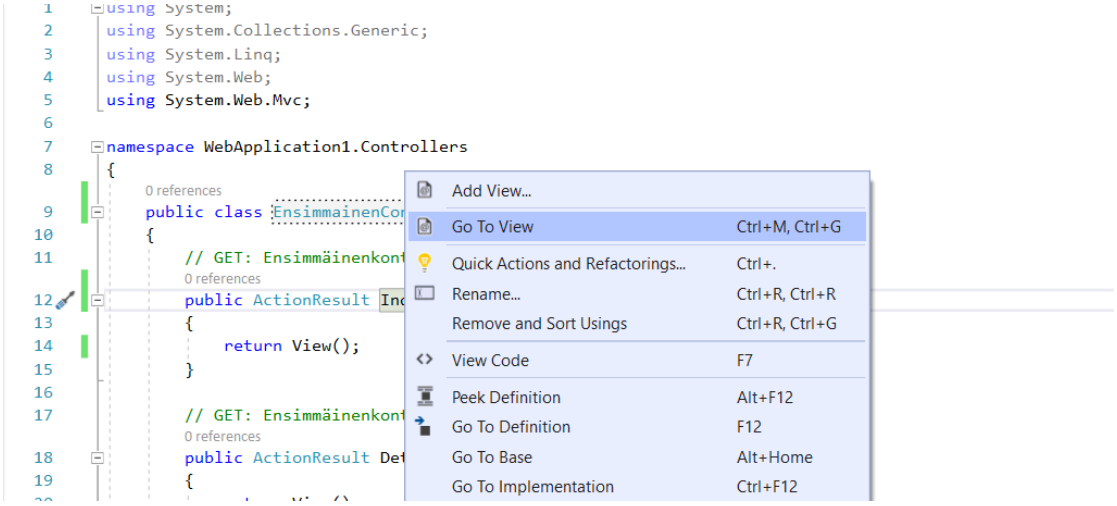
Kuvio 29 Krontrollerin kutsun muuttaminen oikean näkymän palauttamiseksi.

Kutsumalla Ensimmaista se hakee tietoa Ensimmainen/Index -näkymästä. Näkymä on muuttunut, koska pohjana on käytetty Layouttia. Näkymää luodessa siihen voi valita malliksi myös muita valmiita pohjia. Projektin sivujen yhtenäisuuden kannalta Layout pohja on paras valinta, koska sisään kirjautuminen ja muut projektin luomisessa valmiiksi tehdyt sivut ovat luotu Layout -pohjalle.



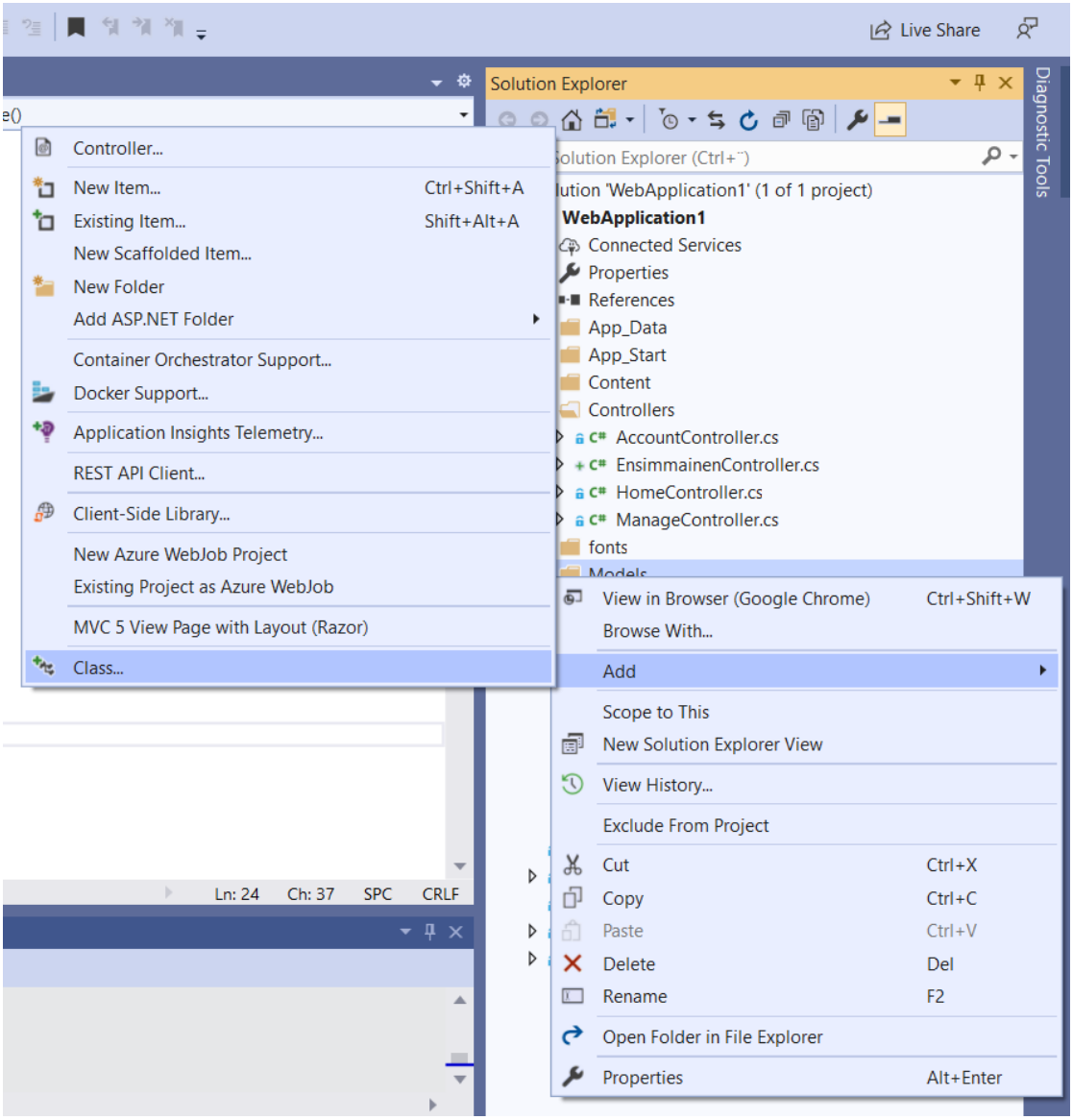
Kuvio 30 Index näkymä Layout -pohjalla.

Kontrollerista pystyy tarkistamaan mihin näkymään se menee painamalla Index-metodia hiiren oikealla ja sitten valitsemalla Go To View, jolloin näkymä johon kontrolleri on liitetty, tulee näkyviin. Tällä keinolla pystytään varmistamaan kontrollerin ja näkymän välinen linkki ilman sovelluksen ajamista.



Kuvio 31 Kontrollerin ohjautumisen tarkistus oikeaan näkymään.

Model luodaan hiiren oikeaa painamalla Model -kansion päältä, jolloin tulee ikkuna, josta valitaan Add-> Class... Mallille annetaan kuvaava nimi ja luodaan se. Nyt mallille voidaan lisätä attribuutteja. KoiraDBContext -luokka hallitsee objektien noutamista, tallentamista ja päivittämistä tietokantaan.

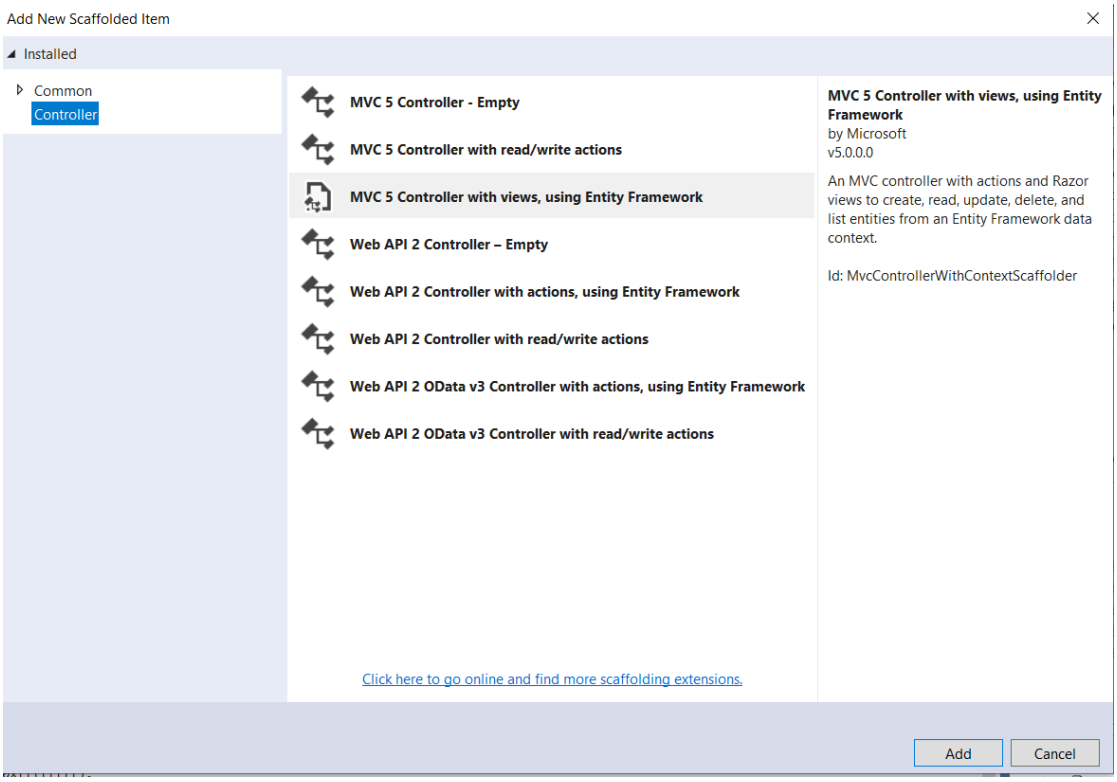


Kuvio 32 Mallin luonti.

```
6
7   namespace WebApplication1.Models
8   {
        1 reference
9       public class Koira
10      {
            0 references
11          public int ID { get; set; }
            0 references
12          public string Nimi { get; set; }
            0 references
13          public double Paino { get; set; }
            0 references
14          public class KoiraDBContext : DbContext
15          {
                0 references
16              public DbSet<Koira> Koiras { get; set; }
17          }
18
19      }
20  }
```

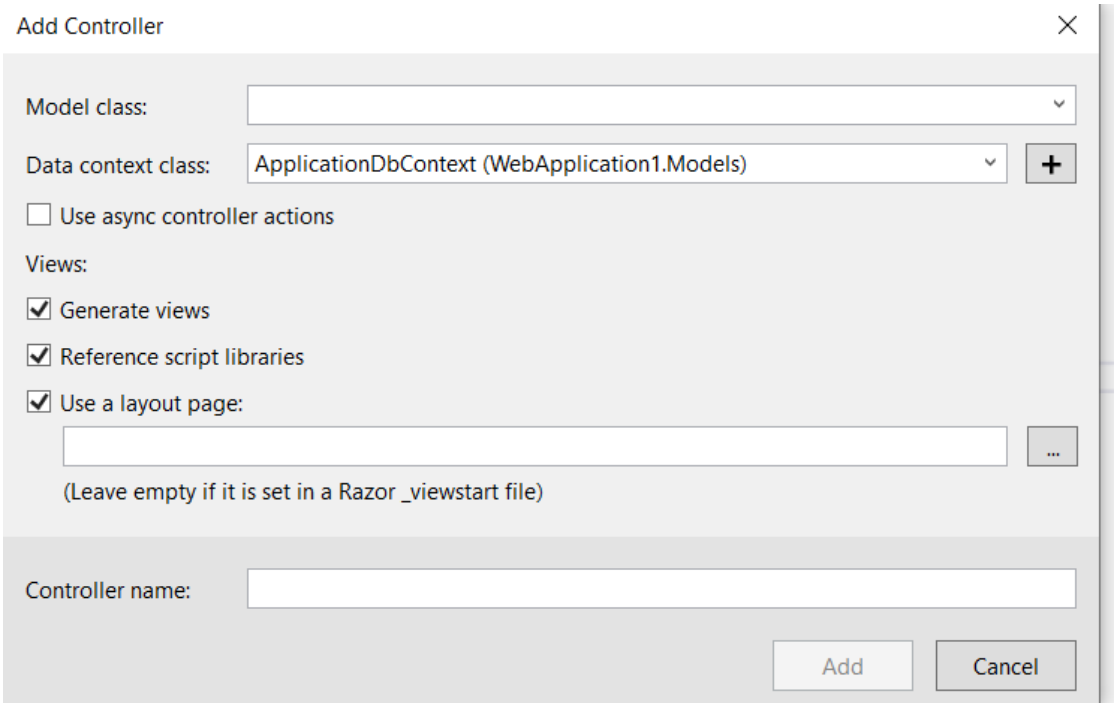
Kuvio 33 Esimerkki mallin metodeista.

Koira -mallille pystytään automaattisesti generoimaan metodit: luo uusi, poista, muokkaa ja Tiedot. Metodit voidaan generoida automaattisesti luomalla uusi kontrolleri MVC 5 Controller With views, using Entity Framework. Kontrolleri luo automaattisesti myös näkymät eri toiminnoille.



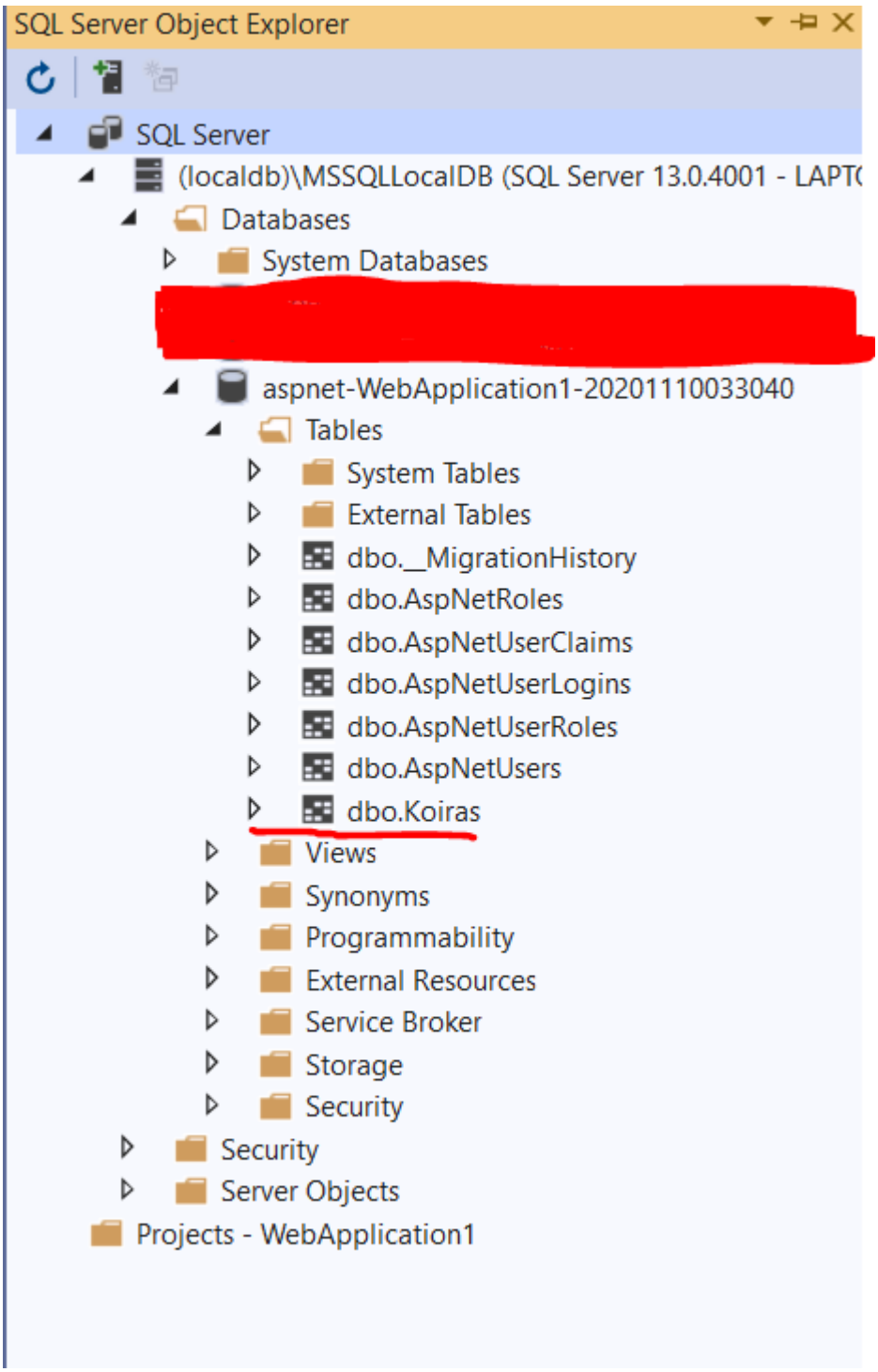
Kuvio 34 Kontrollereiden ja näkymien luonti mallin pohjalta.

MVC 5 Controller With views, using Entity Frameworkin asetuksiin Model Class kohtaan valitaan haluttu malli, jolle generoidaan metodit. Data Context class kohtaan valitaan projektissa käytetty tietokanta. Pohjaksi näkymille pystytään taas valitsemaan Layout, jotta yhtenäisyys projektin ulkoasussa säilyy.



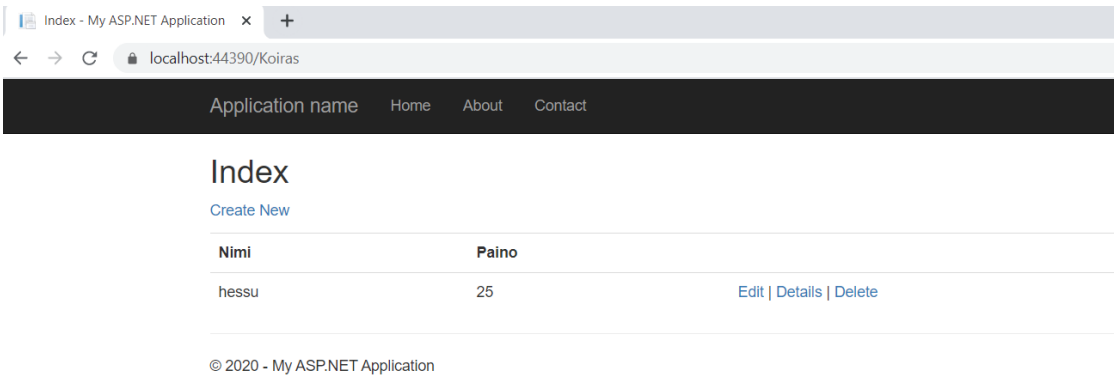
Kuvio 35 Mallin ja asetusten valinta.

Controllers ja Views kohtaan on automaattisesti luotu Koiralle kontrolleri ja eri näkymät metodeille. Nyt kantaan voidaan päivittää koirataulu Package Manage Consolesta Migration -komennolla. Enable-migrations komennolla otetaan muutokset käyttöön. Add-migration -komennolla voidaan lisätä uudet muutokset tietokantaa. Update-database -komennolla pystytään päivittämään tietokanta. Tietokantoihin ilmestyy Koira -taulu, johon ohjelma pystyy lisäämään ja poistamaan tietoa.



Kuvio 36 Mallin pohjalta tuotu taulu tietokantaan.

Ajamalla ohjelmaa Koiras -polusta avautuu sivu, jossa voidaan luoda uusi koira, muokata sitä tai poistaa se. Tässäkin polussa ohjelma menee aina ensimmäiseksi Index -sivulle. Muilla toiminnoilla kuin Index on tärkeää, että tietokantaan on Koira -taulu luotu automaattisesti oikein, koska ne hakevat tai tallentavat tietoa tietokantaan.



Kuvio 37 Esimerkki automaattisesti luodusta Koiras -näkymästä.

Kuvio 38 Esimerkki Koiras luo uusi -kontrollerista.

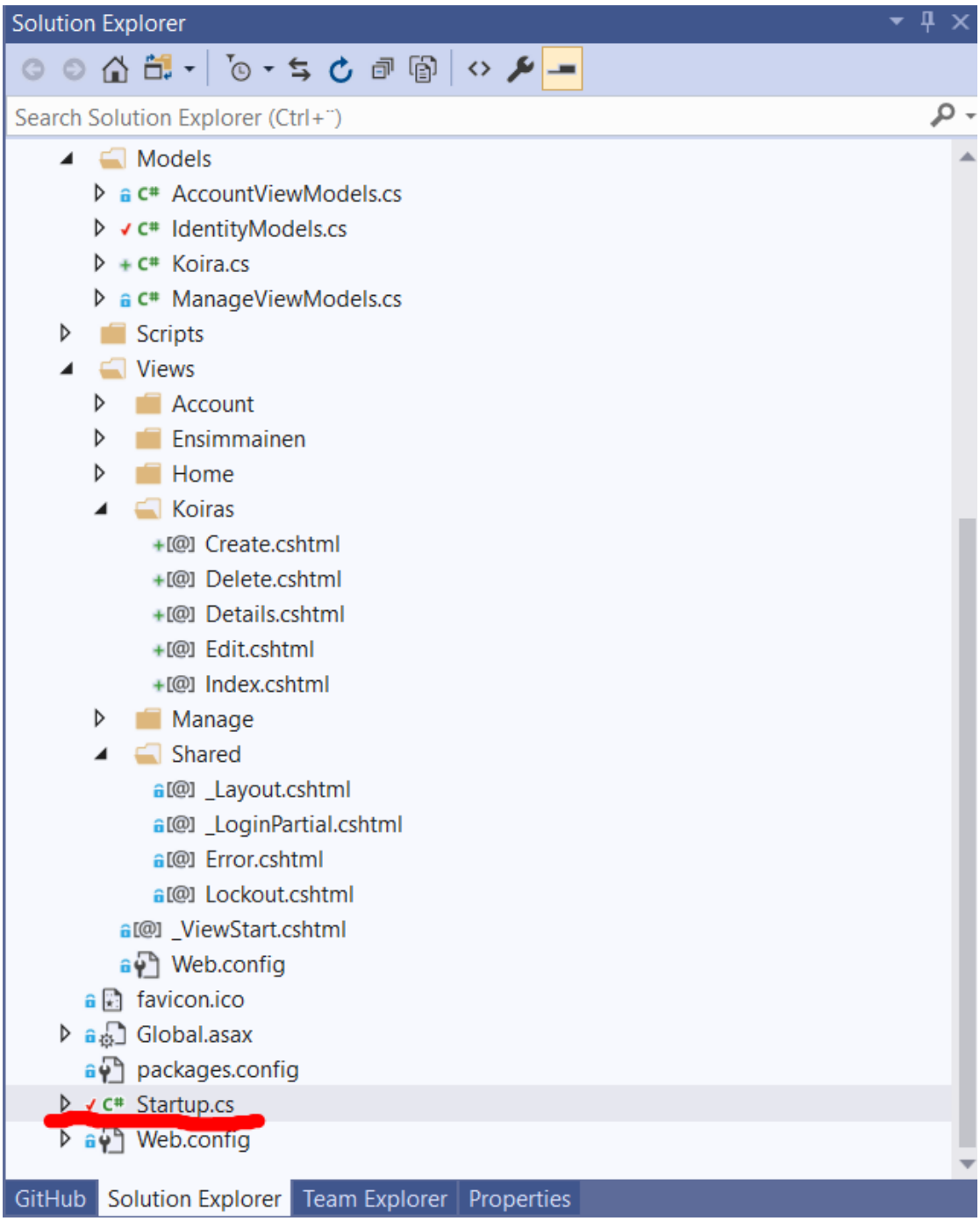
Html.LabelFor on otsikko, joka saadaan suoraan Koiramallista. Tässä tapauksessa otsikossa lukee nimi. Html.EditorFor luo tyhjän tekstilaatikon, johon voidaan kirjoittaa haluttu nimi uudelle koiramuuttujalle ja mallista haetaan siihen oikea tieto. Html.ValidationMessageFor tarkistaa, onko kirjoitetulle nimelle kaikki oikein esim. kenttä ei ole tyhjä tai kenttää ei ole täytetty väärin. <input type="submit" value="Create" class="btn btn-default" /> submittiä painettaessa ohjelma menee kontrollerin 2 kohtaan, johon se hakee ID, Nimi ja Paino. Tässä ohjelmassa tarkistetaan, onko malli oikein luotu. Jos malli on oikein, se lisätään tietokantaan ja tallennetaan muutokset ja lopuksi palataan Koira Index näytölle.

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Koira</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Nimi, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Nimi, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Nimi, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Paino, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Paino, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Paino, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```


Kuvio 39 Esimerkki Koiras luo uusi näkymästä.

9.3 Lisäosien käyttäminen

Roolien avulla pystytään antamaan käyttäjille haluttuja oikeuksia. Pystymme estämään tiettyjä näkymiä tai kokonaisuuksia tietyiltä rooleilta. Projektia tehdessä on hyvä luoda automaattisesti yksi käyttäjä Super käyttäjä, jolle annetaan vahvimmat oikeudet ohjelman käynnistyessä, jotta pääsemme varmasti kiinni myös estettyihin näkymiin, kun sovellus ajetaan ensimmäisen kerran.



Kuvio 40 Roolien luonti tapahtuu Startup.cs.

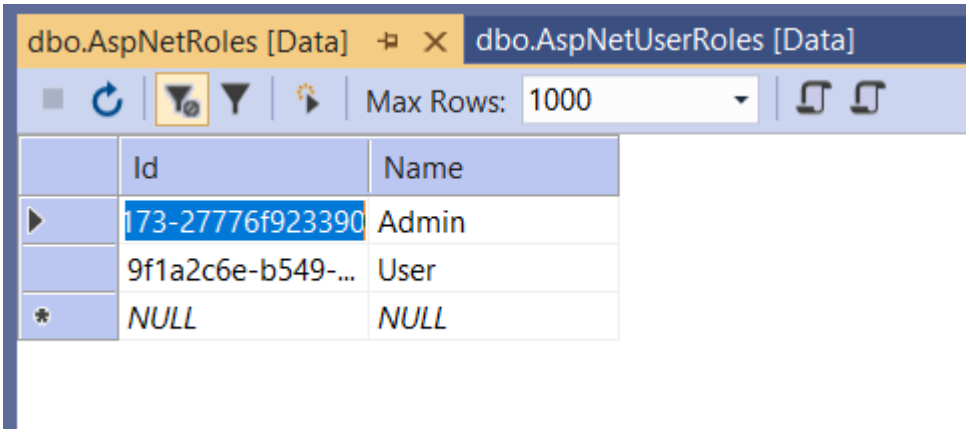
Aliohjelma kutsu roolien tekemiseen luodaan pääohjelma Configuration:lla. Configuration ajetaan aina kun ohjelma käynnistetään. Configuration:sta ohjelma menee aliohjelmaan luoRoolitjaKayttajat(). Aliohjelmassa on kommentoitu, mitä missäkin tapahtuu. Roolit luodaan ensimmäisellä ohjelman ajokerralla. Seuraavilla ajoilla rooleja ei enää luoda uusiksi.

```
        1 reference
        private void luoRoolitjaKayttajat()
        {
            //Tässä yhdistetään tietokanta ja datan malli luokat.
            ApplicationDbContext context = new ApplicationDbContext();
            //Tässä luodaan uusi roolimanager
            var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(context));
            //Tässä luodaan uusi käyttäjämanager
            var UserManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(context));


             //Tutkitaan onko Admin rooli jo olemassa
            if (!roleManager.RoleExists("Admin"))
            {

                   //Luodaan roolille pohja
                var role = new Microsoft.AspNet.Identity.EntityFramework.IdentityRole();
                //Annetaan roolille nimi
                role.Name = "Admin";
                //Luodaan rooli tietokantaan
                roleManager.Create(role);


                //Luodaan uusi käyttäjä
                var user = new ApplicationUser();
                //Annetaan käyttäjälle nimi
                user.UserName = "Super";
                //Annetaan käyttäjälle sposti
                user.Email = "super@gmail.com";
                //Annetaan käyttäjälle salasana
                string userSala = "Sup3r!";
                //Luodaaan käyttäjä
                var kayttaja = UserManager.Create(user, userSala);
```

```
                string userSala = "Sup3r!";
                //Luodaaan käyttäjä
                var kayttaja = UserManager.Create(user, userSala);

                   //Jos kaikki oikein lisätään käyttäjälle rooli
                if (kayttaja.Succeeded)
                {

                    var result1 = UserManager.AddToRole(user.Id, "Admin");

                }
            }


            //Tutkitaan onko User rooli jo olemassa
            if (!roleManager.RoleExists("User"))
            {
                //Luodaan rooli
                var role = new Microsoft.AspNet.Identity.EntityFramework.IdentityRole();
                //Annetaan roolille nimi
                role.Name = "User";
                //tehdään rooli tietokantaan
                roleManager.Create(role);

            }
        }
    }
```

Kuvio 41 Esimerkki roolien luonnista.

Nyt projektiin on luotu roolit User ja Admin. Olemme myös luoneet automaattisesti käyttäjän Super, jolla ohjelman käynnistyessä on Admin rooli. Ajamalla ohjelman tietokantaan ilmestyy rooleille taulut.



Kuvio 42 Tietokantaan tallentuneet roolit.

Yhteys kolmannen osapuolen sovellukseen otetaan rajapinta pyynnöllä request, joka sisältää webpyynnön. Pyyntö sisältää sivuston osoitteen ja halutut parametrit, jotka toisen sovelluksen luoja on valinnut käyttäjän todentamiseksi ja kutsun varmentamiseksi. Webrequestin ominaisuuksia on monia ja ne löytävät sivustolta: https://docs.microsoft.com/en-us/dotnet/api/system.net.httpwebrequest?view=net-5.0. Tässä sovelluksessa tarvitaan Accept ja Headers ominaisuuksia. Pyynnön ollessa oikein vastaus sivustolta voidaan ottaa kiinni (HttpWebResponse)request2.GetResponse();. Pyynnön ollessa väärin, sivusto palauttaa normaalin http erorin. Pyynnön ollessa oikein saamme halutun datan responseen, jolloin voimme käyttää tai muokata dataa sovelluksessamme. Virhe on helpoin ottaa kiinni try catch operaatiolla.

```
var request2 = (HttpWebRequest)WebRequest.Create("https://sivustojohonotetaanyhteyttä/" + "tekstiä" + parametrit);
                    request2.Accept = "application/json";
                    request2.Headers["x-auth-token"] = kayttajavalintatoken;

                    try
                    {
                        response2 = (HttpWebResponse)request2.GetResponse();
                        if (response2 == null || response2.ContentLength < 1) { throw new Exception("Ei vastausta"); }
                        Debug.WriteLine("kanavan" + o.ToString() + "data loppu" + DateTime.UtcNow.ToShortTimeString());
                    }
                    catch (Exception e)
                    {
                        ViewData["HUOM"] = "8";
                        _logger.LogError(e, "ei lötynyt dataa ", DateTime.UtcNow);
                        return NotFound();
                    }
```

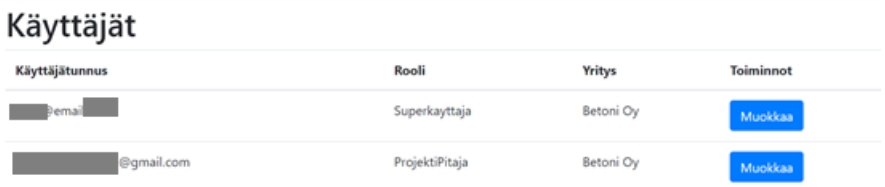
Kuvio 43 Esimerkki request -pyynnöstä.

10 Yhteenveto

Kehityshankkeen tuloksena saatiin toimiva sovellus. Sovelluksessa toimii käyttäjien-, mittareiden- ja yritystenhallinta. Sovellus ottaa yhteyden kolmannen osapuolen sovellukseen ja matemaattiset kaavat toimivat lujuuden laskemiseksi ja ne pystytään esittämään käyrästössä. Kehityshankkeessa kehitettäväksi jäi ulkoasu, mitä tullaan jatkokehittämään yhdessä testiyrityksen kanssa käyttäjälle mieluisaksi.

Opinnäytetyössä onnistuttiin kertomaan tarkasti ja havainnollisesti, kuinka nykyaikaisilla menetelmillä ja työkaluilla pystytään ohjelmoimaan ja suunnittelemaan toimiva sovellus esimerkiksi rakennusteknologiaan. Opinnäytetyössä kerrotaan hyviä käytäntöjä nettipohjaisen palvelun tekemisen vaiheista ja niiden avulla lukija pystyy tekemään samankaltaisia sovelluksia. Opinnäytetyö opettaa lukijaa käyttämään ASP.NET:n ohjelmointikehystä ja Visual Studion tavallisimpia lisäosia, joita ohjelmistokehitykseen tarvitaan. Sovellusta kehitettäessä testiversioiden avulla käyttäjäystävällisyys parani ja sovellukseen saatiin tarvittavat toiminnalliset komponentit.

Käyttäjien hallinta suoritetaan suojatussa näkymässä, johon pääsee vain käyttäjät, joiden roolit ovat joko Ylläpitäjä tai Superkäyttäjä. Haasteita käyttäjien hallinnoimisessa oli aluksi käyttäjän poistaminen, jolloin viitteitä jäi muihin tietokantatauluihin. Roolin asettaminen käyttäjälle ja käyttäjien tietojen näyttäminen valmiina olevassa ASP.NET:n toiminnossa oli yksinkertaista.

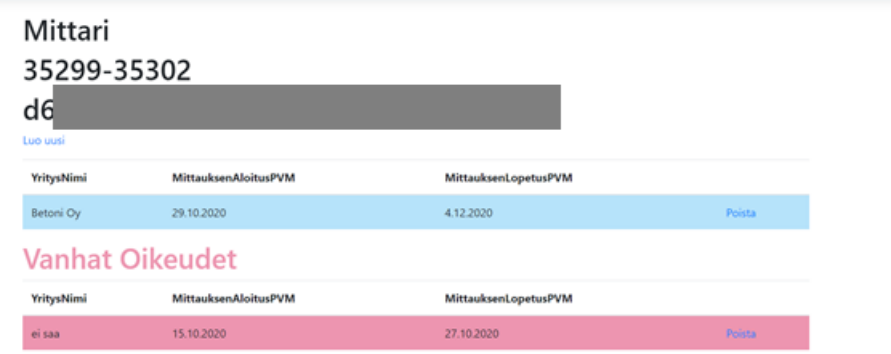


Kuvio 44 Käyttäjien hallinnointi.

Mittareiden hallinta tehtiin hakemalla tieto kolmannen osapuolen tietokannoista, jolloin samat tiedot lisättiin järjestelmän omaan SQL -tietokantaan. Sovelluksessa mittareiden tietoa hyväksikäyttämällä haetaan rajapinnasta tietylle mittarille osoitetut mittausarvot. Mittarit on suojattu näkymä, johon oikeudet saa vain mittareiden

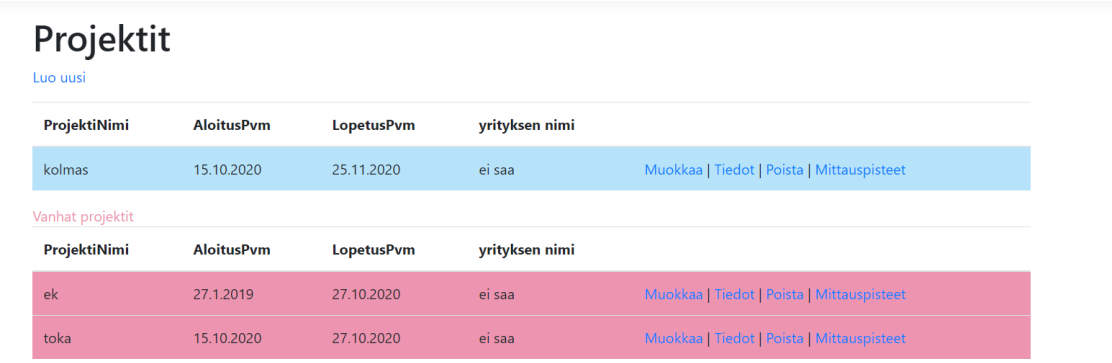
omistajat. Mittareiden omistajat pystyvät lainaamaan mittareita yrityksille tietylle aikajaksolle.

Mittareiden hallinnassa toimii mittareiden näyttäminen ja tallentaminen. Näyttäminen tehtiin valmiilla pohjilla, johon haetttin tiedot tietokannasta. Vaikeaksi osoittautui rajapinnan käyttäminen. Apin käyttämisessä vaikeaa oli virhe ilmoitusten luku, koska vastaukset olivat vain yleisiä kommunikointivirheitä kuten error 404 eikä tarkkoja virheilmoituksia ole saatavilla.



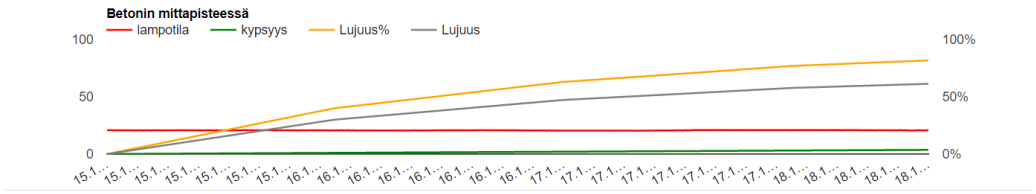
Kuvio 45 Mittareiden hallinnointi.

Projektien hallinnassa esitettiin näkymää vain käyttäjän oman yrityksen projekteista. Projektien luonti ja muokkaus tehtiin valmiilla olevilla pohjilla, joita muokattiin tarkoituksen mukaisiksi. Projekteihin yhdistettiin mittauspisteet tietokannasta ja käyttöliittymästä. Projektin sisällön määrittelemisen teki vaikeaksi häilyvä rajapinta mittauspisteen ja projektin ominaisuuksien välillä. Lopputuloksesta saatiin kuitenkin toimiva yhteistyössä asiakkaan kanssa.



Kuvio 46 Projektien hallinointi.

Projektin matemaattinen laskenta ja käyrän näyttäminen oli yksi vaikeimmista osuuksista ohjelman luomisessa. Käyrästöt tehtiin Google charts:n verkkopalvelua käyttäen. Verkkopalvelun käyttämisessä ongelmaksi tuli sen laajuus, kun Google charts:lla pystyy tekemään paljon erilaisia käyrästöjä ja sen vuoksi dokumentaatiosta oli haastavaa löytää yksityiskohtia toteuttamisen tueksi. Google charts:lla käyrät tehtiin jsp script:llä. Matemaattisessa laskennassa oli vaikeaa suuren tietomäärän laskeminen tehokkaasti. Tehokkuutta laskentaan lisättiin käyttämällä valmiina olevia taulukoiden järjestämis-funktioita.



Kuvio 47 Betonin lujuuden kehityskäyrä.

Ketterä kehittäminen oli hyvä valinta tähän projektiin, koska ohjelma pystyttiin helposti paloittelemaan osiin. Projektin yksittäisten osien ollessa valmiita ne testattiin yhdessä toimeksiantajan ja testiyrityksen kanssa. Kommunikointi onnistui hyvin osapuolilta ja tarvittavat ohjelmiston lisäykset ja komponenttien poistot saatiin tietoon ajoissa ja ylimääräiseltä työltä vältyttiin.

Tietokannan hallinta ja luominen onnistui vaivattomasti. Projektissa ei pystytty tietokantaa luomaan kokonaan ohjelman kehittämisen alussa, koska kaikkia tietoja ei ollut

vielä tiedossa. Tietokannan käyttäminen oli helppoa Visual Studion lisäosalla. Ongelmia tietokannan päivittämisessä oli sen yhteensopivuus nykyisiin toimintoihin. Tietokantatauluun uusia tietueita päivitettäessä ohjelma hajosi osittain, koska ohjelman ja tietokannan välillä tiedon hakemisessa ja tallentamisessa oli risteyksiä. Taulujen muuttuneiden tietueiden lisääminen oli helppoa ohjelma koodiin, koska Visual Studio näyttää tarkasti virheen paikan. Ohjelman kehittämisessä olisi päästy helpommalla, jos tarvittavat tiedot olisi tiedetty projektin aloittamisessa. Kaikkien taulujen sisältämät tietueet ovat käytännössä kuitenkin mahdotonta tietää projektia aloitettaessa, mikäli ohjelmaan tehdään muutoksia kehityksen aikana.

11 Pohdinta

Sovellus onnistui lähes toivotulla tavalla. Kehittämisessä pääsääntöisesti oli ongelmia liittyen ASP.NET MVC -ohjelmointikehyksen erilaisiin toimintoihin. Ongelmat ASP.NET MVC:ssä liittyivät kokemattomuuteen ja siihen mitä kaikkea pystyy automaattisesti tekemään ja luomaan käytössä olevilla kirjastoilla. Ohjeita oli haastava löytää, koska ne ovat hajallaan eri lähteissä, joiden luotettavuus vaihteli merkittävästi. Ohjelman kehityksen aikana opin paljon uusia asioita ASP.NET MVC:ssä olevista toiminnoista. Ohjelmoinnin aloittamisen alkuvaiheessa olisi pitänyt perehtyä syvällisemmin ASP.NET MVC:n erilaisiin toimintoihin, koska suurimmat ongelmat olivat ratkaistavissa valmiina olevilla funktioilla.

Jatkokehitettävää ohjelmassa on uusien toimintojen ja laskenta-algoritmien lisääminen. Kehitettävää on myös paljon ulkoasussa, jotta ohjelmisto näyttäisi mahdollisimman ammattimaiselta käyttäjän mielestä. Tähän sovellukseen pystyy tulevaisuudessa lisäämään helposti eri toimijoiden dataloggereita ja betonivalmistajien erilaisia lujuusalgoritmeja. Tätä sovellusta voi käyttää myös pohjana tulevaisuudessa tehtäville sovelluksille, kosks käyttäjien, yritysten ja projektien hallinta on suoraan uudelleenkäytettävissä.

Lähteet

API – Mikä on API?. N.d. Artikkeli Visma sivustolla. Viitattu 14.11.2020. https://www.visma.fi/epasseli/kirjanpidon-sanakirja/a/api/

ASP.NET Is Almost Certainly The Wrong Technology For Your Project. N.a. Artikkeli Jimmyweb sivustolla. Viitattu 14.11.2020. https://jimmyweb.net/insights/aspnet-is-almost-certainly-the-wrong-technology-for-your-website-project/.

ASP.NET MVC Pattern. N.d. Artikkeli Microsoftin sivuilla. Viitattu 16.11.2020. https://dotnet.microsoft.com/apps/aspnet/mvc.

Betoni on luja rakennusmateriaali, mutta mikä on betonin lujuus? N.d. Artikkeli Finnsementin sivustolla. Viitattu 18.11.2020. https://finnsementti.fi/palvelut/tietoa-betonista/betonin-lujuus/.

Betonin ominaisuudet ja käyttö. N.d. Artikkeli betoni.com sivustolla. Viitattu 18.11.2020. https://betoni.com/tietoa-betonista/perustietopaketti/betoni-rakennusmateriaalina/betonin-ominaisuudet-ja-kaytto/.

BETONIN VALINTA RAKENTEISIIN – OLOSUHDEHALLINTA. N.d. 300 mm paksun paikallavaluholvin lujuudenkehitys, kun ilman lämpötila on -5 °C, muottivaneri 22 mm. Betonimassan lämpötila +20 °C (lukuun ottamatta RA K30 +30 °C). Suojausvaihtoehdossa 6 mm eristepeite holvin yläpinnalla, holvin alapuolinen tila lämmitetty +15 °C. Kuva Ruduksen julkaisussa. Viitattu 20.11.2020. https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.rudus.fi%2FDownload%2F23940%2FBetonin%2520valinta%2520rakenteisiin%2520-%2520olosuhdehallinta.pdf&psig=AOvVaw1jk3a_BzeP5H5ojOvlPQq&ust=1605980380884000&source=images&cd=vfe&ved=0CAMQjB1qFwoTCPiPq6TVke0CFQAAAAAdAAAAABAD.

Betonirakentaminen. N.d. Artikkeli betoni.com sivustolla. Viitattu 18.11.2020. https://betoni.com/betonirakentaminen/.

by 201 Betonitekniikan oppikirja 2018. 2018. BY-koulutus.

Data Loggers. 2019. Artikkeli Omega sivustolla. Viitattu. 16.11.2020. https://www.omega.com/en-us/resources/data-loggers.

Definition and role: what is an API?. N.d. Artikkeli planningpme sivustolla. viitattu 20.11.2020. https://www.planningpme.com/planningpme-api.htm.

Grayson, T. 2001. Introduction to Relational Database Design. Artikkeli MIT sivustolla. Viitattu 20.11.2020. http://web.mit.edu/11.521/www/lectures/lecture10/lec_data_design.html.

Junttila, V.2013. Tornitalo romahti Lahdessa. Turun Sanomat. Viitattu 20.11.20. https://www.ts.fi/puheenvuorot/469561/Tornitalo+romahti+Lahdessa.

Järvenpää, J. 2020. Ketterä kehittäminen – miten ja miksi?. Blogi Vincit sivustolla. Viitattu 16.11.2020. https://www.vincit.fi/fi/kettera-kehittaminen-miten-ja-miksi/.

Ketterä kehitys. 2015. Blogi Turun yliopiston sivuilla. Viitattu 16.11.2020. https://tt.utu.fi/embedded_kasikirja/1/1/index.html.

Macoveiciuc, A. 2020. Beginner’s Guide to APIs, Protocols and Formats. Artikkeli medium sivustolta. Viitattu 14.11.2020. https://medium.com/frontend-digest/beginners-guide-to-apis-protocols-and-data-formats-f80cf7f30425.

Miten toimii TE-anturi (termoelementti)?. N.d. Artikkeli Epic sensors sivulla. Viitattu 16.11.2020. https://www.epicsensors.fi/faq/miten-te-anturi-toimii/.

MVC for dummies: malli, näkymä ja ohjain -arkkitehtuuri web-sovelluksissa. 2020. Blogi hurja sivustolla. Viitattu 16.11.2020. https://www.hurja.fi/blogi/mvc-for-dummies-malli-nakyma-ja-ohjain-arkkitehtuuri-web-sovelluksissa/.

Pearman, G & Goodwill, J. 2006. Pro .NET 2.0 Extreme Programming. Apress.

Räsänen, S. Pieni SQL-opas. Opetus opas ammattikorkeakoulu savonia sivuilla. Viitattu 16.11.2020. http://webd.savonia.fi/home/ktrasse/muut/visbas/kappale16.htm.

Smith, S & Addie, S. 2019. Handle requests with controllers in ASP.NET Core MVC. Artikkeli Microsoftin sivuilla. Viitattu 16.11.2020. https://docs.microsoft.com/en-gb/aspnet/core/mvc/controllers/actions?view=aspnetcore-3.1.

Spinelli, J. 2018. MVC Overview. Artikkeli Medium sivustolla. Viitattu 20.11.2020. https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5.

Termoelementti eli termopari. 2019. Oppimateriaali Keuda sivustolla. Viitattu 16.11.2020. https://keuda.moodle.fi/mod/page/view.php?id=431387.

The Good and the Bad of .NET Framework Programming. 2020. Artikkeli Altexsoft sivustolla. Viitattu 14.11.2020. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming/.

The tools you need to build what you want. N.d. Esittely sivu työkalusta GitHubin etusivulla. Viitattu 14.11.2020. https://github.com/features#team-management.

Tolvanen, P. N.d. Ketteryys haltuun: Scrum pähkinänkuoressa. Artikkeli Sininen meteoriitti sivustolla. Viitattu 16.11.2020. https://meteoriitti.com/2013/06/06/ketteryys-haltuun-scrum-pahkinankuoressa/.

Van Der Hoek, J. 2018. Pursuing a Full Agile Software Development Life Cycle. Artikkeli Mendix sivustolla. Viitattu 20.11.2020. https://www.mendix.com/blog/pursuing-a-full-agile-software-lifecycle/.

Weil, A. 2015. Learn ASP.NET MVC. Leanpub.

What is ASP.NET?. N.d. Artikkeli Microsoftin sivuilla. Viitattu 16.11.2020. https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet.

Liitteet