Anh-Vu Nguyen

Verkkosovelluskehitys Laravel 5.3:lla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

5.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Anh-Vu Nguyen Verkkosovelluskehitys Laravel 5.3:lla 71 sivua + 4 liitettä 5.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Outi Grotenfelt

Insinöörityössä tutustuttiin Laravel 5.3 Framework -sovelluskehykseen. Työn tavoitteena oli oppia Laravelin käyttöä, ja sitä käyttämällä nopeuttaa monimutkaisen ja turvallisen verkkosovelluksen luomista ja kehitystä. Laravel on avoimen lähdekoodin PHP-sovelluskehys, joka perustuu MVC-malliin. Sen tarkoituksena on helpottaa ja nopeuttaa verkkosovelluskehittäjän työtä.

Samalla haluttiin luoda kaksi päiväkodeille suunnattua verkkosovellusta kyseistä sovelluskehystä käyttämällä. Halutuista verkkosovelluksista ensimmäinen oli tarkoitettu päiväkodin työntekijöille ja toinen päiväkodin lasten huoltajille. Verkkosovellusten tarkoituksena oli helpottaa lasten huoltajien ja päiväkodin työntekijöiden välistä kommunikaatiota,
läsnäolojen ja ruokailujen merkintää, sekä näyttää huoltajille lastensa päivän kuulumiset.

Työssä kerrottiin aluksi yleisesti verkko-ohjelmoinnin perustasta, jotka lukijan piti ymmärtää ja osata ennen kuin syvennyttiin Laravel-sovelluskehykseen. Sitten käytiin Laravelin asennus läpi askel askeleelta, jonka jälkeen syvennyttiin Laravelin dokumentaatioon.

Laravelin dokumentaatio-osiossa käytiin läpi kaikki haluttujen verkkosovellusten kannalta tarpeelliset Laravelin tarjoamat toiminnallisuudet. Omilla koodiesimerkeillä havainnollistettiin Laravelin tarjoamien toiminnallisuuksien käyttämistä, ja samalla kerrottiin niiden merkityksistä halutuissa päiväkotisovelluksissa. Laravelin toiminallisuudet ja dokumentaatio käytiin läpi myös siinä järjestyksessä kuin niitä käytettiin verkkosovelluskehityksen aikana.

Lopputuloksena syntyi kaksi turvallista ja tavoitteiden mukaista päiväkodeille tarkoitettua verkkosovellusta. Sovellukset on tarkoitus myydä kunnille päiväkodin nykyisen järjestelmän digitalisoinnin tueksi.

Avainsanat	Laravel, verkkosovellus, sovelluskehys, päiväkoti, digitalisaatio

Author(s) Title Number of Pages Date	Anh-Vu Nguyen Web Application Development with Laravel 5.3 71 pages + 4 appendices 5 May 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Outi Grotenfelt, Senior Lecturer

This thesis focuses on Laravel 5.3 Framework. The goal of the thesis was to learn to use Laravel, and with it hasten complex web application development.
Laravel is an open-source PHP-Framework based on MVC-architecture. It was created to support web developers in their work.

There was also an additional goal to the study, where two complex web applications were to be created for kindergartens with the help of the Laravel Framework. The first web application was meant for the kindergarten employees, and the second for the guardians of the kindergartens' children. The purpose of the web applications was to help the communication between the guardians of the kindergarten's children and the kindergarten's employees, mark presences and meals, and also show the guardians how their childrens' days have proceeded.

In the beginning of the study the general basics of web application development are explained. Next, the Laravel Installation is explained step by step, and after that the focus shifts to Laravel's documentation.

In the Laravel's documentation section all the necessary functionalities for the kindergarten's web applications are explained. The usage of Laravel's functionalities, and their purpose in the kindergarten's web applications, are also explained. The order of going through the functionalities also matches the web development flow itself.

The result consists of two secured kindergarten web applications that met the additional goals of the study. The web applications are meant to be sold to the government to support the digitalization of the current kindergarten systems.

Keywords	Laravel, web application, framework, kindergarten, digitalisation

Sisällys

Liitteet

Lyhenteet

CLI	Command Line Interface. Komentorivikäyttöliittymä.
CRUD	Create, Read, Update and Delete. Neljä perusfunktiota jatkuvaan tiedon tallentamiseen.
CSRF	Cross-Site Request Forgery. Istuntoa hyväksi käyttävä hakkerointimenetelmä.
CSS	Cascading Style Sheets. Tyyliohjeiden laji, jossa määritellään nettisivun ulkonäkö.
HTML	Hypertext Markup Language. Kuvauskieli, jolla nettisivut on kirjoitettu.
HTTP	Hypertext Transfer Protocol. Selainten käyttämä tiedonsiirtoprotokolla.
IDE	Integrated Development Environment. Ohjelmointiympäristö sovelluskehitykseen.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuurimuoto.
OS	Operating System. Käyttöjärjestelmä.
PHP	PHP: Hypertext Preprocessor. Ohjelmointikieli.
SQL	Structured Query Language. Standardoitu kyselykieli, jota käytetään relaatiotietokannoissa.

1 Johdanto

Verkkosovelluksista on tullut nykypäivänä välttämätön osa liiketoimintaa, jolloin niille on olemassa myös paljon kysyntää (1). Niiden avulla voidaan hankkia julkisuutta ja kuuluvuutta erilaisille yrityksille, tavoittaa asiakkaita helpommin, sekä digitalisoida ja nopeuttaa palveluita. Runsaan kysynnän takia verkkosovellusten luomisessa ja kehittämisessä on keskeisintä sovelluskehyksien käyttäminen.

Sovelluskehyksien ideana on nopeuttaa verkkosovelluskehityksen kulkua tarjoamalla valmiita komponentteja tai toimintoja, joita ohjelmoija voi lisätä sovellukseen tai käyttää tarvittaessa. Sovelluskehys voi olla avointa tai suljettua lähdekoodia, millä viitataan lähdekoodin saatavuuteen. Avoimessa lähdekoodissa kaikki on ilmaista, ja kuka vain pääsee käyttämään ja kehittämään sitä, toisin kuin suljetussa lähdekoodissa, joka on rajoittunut vain tietyn tiimin kehitettäväksi ja käytettäväksi. Avoimessa lähdekoodissa jatkokehitys tapahtuu myös pääosin vapaa-ajan harrastuksena, ja koska kehittäjä voi siinä olla kuka vain, se ei ole myöskään rajattu tiettyyn aikaan tai paikkaan, jolloin sitä päivitetään nopeammin ja useammin kuin suljettua lähdekoodia.

Insinöörityön aiheena on tutustua tämän hetken suosituimpaan palvelinpuolella käytettävään avoimen lähdekoodin PHP-sovelluskehyksen uusimpaan versioon, Laravel 5.3:een (liite 2). Työn tavoitteena on Laravel-sovelluskehystä käyttämällä kehittää kaksi turvallista autentikaatiojärjestelmää sisältävää päiväkotisovellusta Suomen päiväkotijärjestelmän digitalisaation tueksi.

Luvussa 2 selitetään kaikki verkkosovelluskehityksen kannalta vaadittavat esitiedot kevyenä pintaraapaisuna. Luvussa 3 käydään askel askeleelta Laravelin asentaminen sekä uuden Laravel-projektin luonti Windows-käyttöjärjestelmässä. Luku 4 on hyvin teoriapainotteinen, ja siinä käydään läpi päiväkotiverkkosovelluksissa tarvittavat Laravel-sovelluskehyksen tarjoamat komponentit. Komponenttien läpikäyminen tapahtuu kronologisesti verkkosovelluskehityksen mukaisessa järjestyksessä, ja niiden käyttämistä havainnollistetaan omien koodiesimerkkien avulla. Samalla selitetään, mikä on niiden merkitys päiväkotiverkkosovelluksissa.

2 Yleistä verkko-ohjelmoinnista

2.1 Front-end

Front-end eli selainpuoli kattaa kaiken käyttäjälle näkyvän osuuden verkkosovelluksessa (2). Siihen kuuluu HTML-, CSS- ja JavaScript-tiedostot, jotka voidaan ajatella sivuston rakennusohjeina. Kyseiset tiedostot määräävät sivuston ulkoasun, joka saattaa vaihdella hieman eri selaintyyppien välillä. Tämä johtuu siitä, että jokainen selaintyyppi käsittelee front-end-tiedostoja omalla tavallaan. Esimerkiksi vanhat Internet Explorer-selaimet eivät pysty ymmärtämään modernimpia front-end-ominaisuuksia, ja siksi sillä avatut sivut näyttävät tylsemmiltä tai rikkoutuneilta. Tämän takia verkkosivua pitääkin testata useammalla eri selaintyypillä.

Vanhempien selaimien tukeminen lisää verkko-ohjelmoijan työtä, sillä siinä pitää usein keksiä toinen tapa jonkin modernin ominaisuuden toimimiseksi, mikä voi olla joissakin tapauksissa mahdotonta. Toisaalta 12. tammikuuta 2016 Microsoft julkaisi tuen päättymisen vanhemmille IE10, IE9, ja IE8-selainversioille joissakin Windows-käyttöjärjestelmissä. Se tarkoittaa sitä, että kyseiset selaimet eivät saisi enää turvallisuuspäivityksiä tai teknistä tukea. Lubos Kmetkon mielestä kyseisiä selaimia varten ei siis kannata ohjelmoida, koska niitä selaimia käyttävät ihmiset olisivat myös turvallisuusriskin kohteena (3).

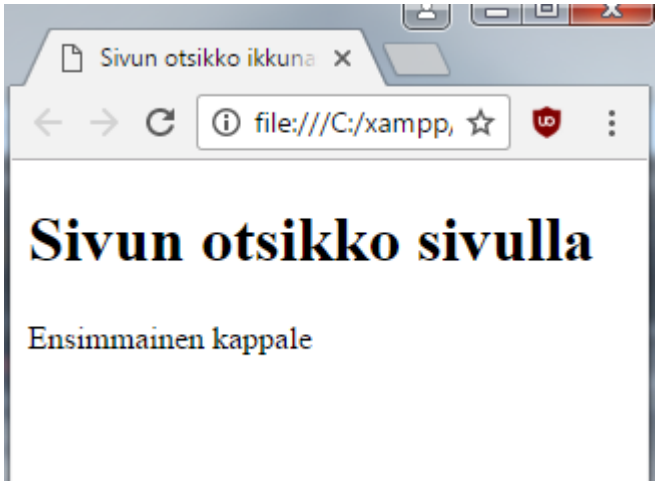
2.1.1 HTML5

HTML eli Hypertext Markup Language on tunnettu erityisesti kielenä, jolla verkkosivut on kirjoitettu. HTML-tiedoston tunnistaa .html-tiedostopäätteestä, ja se sisältää rakenteellista tekstiä, joka muodostuu sisäkkäisistä elementeistä, verkkosivujen rakennuspalikoista. Elementit kuvataan koodissa avaus- ja sulkutägeillä eli tunnisteilla. Niiden tehtävänä on ohjeistaa selaimia, miten niiden välissä oleva sisältö tulisi jäsentää.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Sivun otsikko ikkunassa</title>
    </head>
    <body>
        <h1>Sivun otsikko sivulla</h1>
        <p>Ensimmainen kappale</p>
    </body>
</html>
```

Kuva 1. HTML-dokumentti

Kuvassa 1 on yksinkertainen esimerkki HTML-koodista, jossa esimerkiksi *Sivun otsikko sivulla* -teksti on h1-tägien sisällä. Tämä kertoo selaimelle, että *Sivun otsikko sivulla* -teksti tullaan näyttämään h1-kuvauksen mukaisesti, ja h1 kuvaa isoa otsikkoa. Tällöin HTML-sivu eli HTML-koodi verkkoselaimella tulisi näyttämään tältä:



Kuva 2. HTML-sivu verkkoselaimella.

Tässä projektissa käytettiin uudempaa HTML5-versiota HTML-sivujen luomiseen.

2.1.2 CSS3

CSS eli Cascading Style Sheets on tyylikieli, joka vastaa HTML-elementtien tyylityksestä, kuten fontista, väreistä, sisältöjen sijainneista ja niin edelleen. Verkkoselain rakentaa CSS-tyyliehdotusten pohjalta sivun graafisen ulkoasun, ääntämisen tai tulostuksen.

CSS-koodia sisältävää tiedostoa, jonka tunnistaa .css-tiedostopäätteestä, kutsutaan tyyliehdotukseksi, joka taas koostuu vähintään yhdestä tyylisäännöstä. Tyylisääntö koostuu valitsimesta ja ominaisuus-arvo-pareista seuraavalla tavalla:

```
valitsin {
    ominaisuus1: arvo;
    ominaisuus2: arvo;
}
```

Kuva 3. CSS-syntaksi

Valitsimia on monenlaisia, mutta yksinkertaisemmilla valitsimilla tarkoitetaan HTML-koodin tägien nimiä, esimerkiksi kuvan 1 *Sivun otsikko sivulla* -tekstin valitsin olisi h1. Jos h1-tägille haluttaisiin antaa punainen väri, CSS-koodi näyttäisi tällaiselta:

```
h1 {
    color: red;
}
```

Kuva 4. CSS-syntaksin mukainen tyylisääntö, jossa h1-tägien sisältö värjätään punaiseksi

Ennen kuin voidaan käyttää CSS:ää, se pitää liittää HTML-sivuun, jonka voi tehdä kahdella eri tavalla. Ensimmäinen tapa on upottaa CSS-koodi style-tägien sisään tällä tavalla:

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            h1 {
                color: red;
            }
        </style>
        <title>Sivun otsikko ikkunassa</title>
    </head>
    <body>
        <h1>Sivun otsikko sivulla</h1>
        <p>Ensimmainen kappale</p>
    </body>
</html>
```

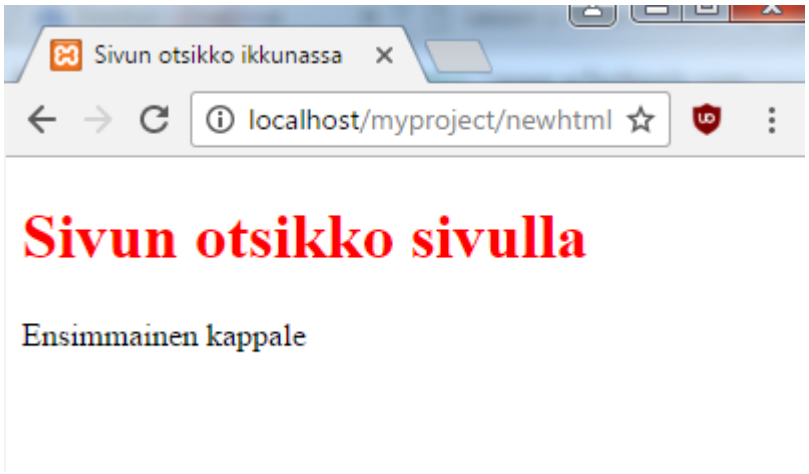
Kuva 5. HTML-dokumentti, johon on upotettu CSS-koodia

Paras käytäntö on kuitenkin se, että HTML-dokumentti olisi mahdollisimman pieni, ja se tapahtuu toisella tavalla. Siinä viitataan erilliseen css-tiedostoon HTML-dokumentissa link-tägien avulla, jossa href-attribuutin sisään tulee polku tyylitiedostoon:

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="csstiedostonnimi.css">
        <title>Sivun otsikko ikkunassa</title>
    </head>
    <body>
        <h1>Sivun otsikko sivulla</h1>
        <p>Ensimmainen kappale</p>
    </body>
</html>
```

Kuva 6. HTML-dokumentti, jossa on viitattu CSS-tiedostoon.

Molemmissa tapauksissa kuvan 2 sivu näyttäisi sen jälkeen tällaiselta:



Kuva 7. HTML-sivu, johon on lisätty CSS-koodia

Tässä insinöörityössä käytettiin CSS3-versiota, joka on uusin CSS-standardi.

2.1.3 JavaScript

JavaScript on ohjelmointikieli, minkä avulla tehdään verkkosivuista interaktiivisia, eikä sitä tule sekoittaa Java-kieleen. JavaScript-tiedoston tunnistaa .js-tiedostopäätteestä. Sen avulla pystytään lisäämään HTML-elementteihin toiminnallisuuksia tai muuttamaan niiden sisältöä, kuten elementtien piilottaminen ja näyttäminen tiettyä nappia painaessa ja niin edelleen. Kuten CSS- ja HTML-koodit, JavaScript ajetaan myös selaimessa, ja jokainen selain ymmärtää sen hieman eri tavalla.

```
var x = document.getElementsByTagName("h1");
x[0].innerHTML = "Otsikko muutettu.";
```

Kuva 8. JavaScript-syntaksi

Kuvan 8 koodissa asetetaan muuttuja x viittaamaan HTML-dokumentin kaikkiin h1-tägeihin. Sen jälkeen innerHTML-funktion avulla muutetaan muuttuja x:n ensimmäinen alkio eli toisin sanoen dokumentin ensimmäisen h1-tägin sisältö *Otsikko muutettu* -tekstiksi. Yksinkertaisimmillaan JavaScript on tällaista. Kuten CSS:n tapauksessa JavaScript pitää liittää HTML-sivuun, jotta se saadaan toimimaan. Käytetään esimerkkinä kuvan 1 HTML-dokumenttia. Liittämisen voi tehdä kahdella eri tavalla, joista yksi on upottaa JavaScript-koodi script-tägien sisään:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Sivun otsikko ikkunassa</title>
    </head>
    <body>
        <h1>Sivun otsikko sivulla</h1>
        <p>Ensimmainen kappale</p>
        <script>
            var x = document.getElementsByTagName("h1");
            x[0].innerHTML = "Otsikko muutettu.";
        </script>
    </body>
</html>
```

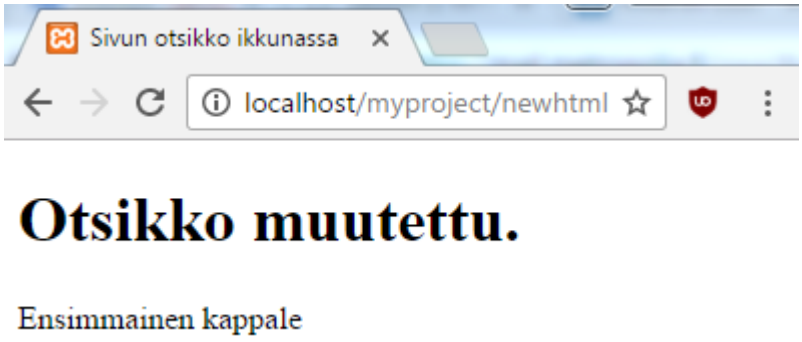
Kuva 9. HTML-dokumentti, johon on upotettu JavaScript-koodia

Myös JavaScriptin tapauksessa paras käytäntö on saada HTML-dokumentista mahdollisimman pieni, joka toteutuu toisella tavalla. Siinä viitataan erilliseen js-tiedostoon HTML-dokumentissa laittamalla polku js-tiedostoon script-tägin src-attribuuttiin seuraavasti:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Sivun otsikko ikkunassa</title>
    </head>
    <body>
        <h1>Sivun otsikko sivulla</h1>
        <p>Ensimmainen kappale</p>
        <script src="jstiedostonnimi.js"></script>
    </body>
</html>
```

Kuva 10. HTML-dokumentti, jossa on viitattu JavaScript-tiedostoon

Molemmissa tapauksissa kuvan 2 sivu näyttäisi sen jälkeen tällaiselta:



Kuva 11. HTML-sivu, jossa otsikko on muutettu JavaScript-koodilla

Parhaan käytännön mukaan kaikki script-tägit kannattaa lisätä body-tägin loppuun. Syynä tähän on se, että selaimet lukevat HTML-dokumenttia ylhäältä alas. Jos scripti luetaan ensiksi, saattaa vastaan tulla tilanne, missä JavaScript-koodin muuttuja viittaa elementtiin, mitä ei ole vielä ehditty käydä läpi. Siinä tapauksessa muuttuja ei viittaa mihinkään, jolloin siihen liittyvä JavaScript-koodi ei myöskään tee mitään.

2.1.4 JQuery & Bootstrap

Tässä työssä käytettiin front-end-ohjelmoinnin nopeuttamiseksi ja helpottamiseksi avoimen lähdekoodin uusinta jQuery-versiota ja Bootstrap 3 -sovelluskehystä.

JQuery on JavaScript-kirjasto, joka yksinkertaistaa ja lyhentää JavaScript-syntaksia, ja se on suunniteltu myös tukemaan paremmin Cross-browseria. Cross-browser-tuella viitataan siihen, että samalla koodilla saataisiin sama visuaalinen ilme ja toiminnallisuus kaikkiin selaimiin, jonka vuoksi ei tarvitse ohjelmoida omaa koodia jokaista selainta varten. JQuery-tuen voi lisätä lataamalla jquery.js-nimisen tiedoston jQueryn omilta sivuilta, ja viitata siihen kuvan 10 mukaisesti, sillä se on JavaScript-tiedosto.

Bootstrap on suosituin front-end-sovelluskehys, jonka pääpainona on verkkosivujen responsiivinen suunnittelu. Responsiivisella suunnittelulla tarkoitetaan sitä, että verkkosivun sisältö muuttuu selainikkunan koosta riippuen. Sivu on responsiivinen, kun pienimmälläkin mobiililaitteella samalle sivulle mentäessä käyttäjä ei pysty skrollaamaan sivua sivusuunnassa, ja kaikki sivun sisältö mahtuisi selainikkunan vasemman ja oikean ikkunarajan väliin. Tämä helpottaa käyttäjää niin, että hän pääsee mobiililaitteilla käymään koko sivun läpi vain skrollaamalla alaspäin ja olemaan huolehtimatta siitä, jos joitakin verkkosivun elementtejä olisi piiloutunut mobiiliselainikkunan ulkopuolelle.

Bootstrapin omilta sivuilta voi ladata bootstrap.css- ja bootstrap.js-nimiset tiedostot ja linkittää ne kuvien 6 ja 8 mukaisesti HTML-dokumenttiin riippuen siitä, onko kyseessä .css- vai .js-tiedosto. Bootstrapin toiminnallisuuden edellytyksenä on kuitenkin jQuery, sillä bootstrap.js-tiedosto on rakennettu jQuery-syntaksilla. HTML-dokumentissa on sen takia viitattava jquery.js-tiedostoon ennen bootstrap.js-tiedostoa.

2.2 Back-end

Back-endillä tarkoitetaan kaikkea käyttäjälle näkymätöntä osuutta verkkosovelluksessa, ja se sisältää kaikki verkkosovelluksen takana olevan logiikan (2). Toisin sanoen se kattaa kaiken koodin, joka ajetaan sivuston palvelimella, esimerkiksi lomakkeiden käsittelyt, kirjautuminen, salanojen tarkistaminen, järjestelmäintegraatiot ja tietokantojen käsittely. Koska back-end suoritetaan sivuston palvelimella, se on toiminnaltaan täysin selaimesta riippumaton. Back-end-ohjelmointikieliä on monta, mutta tässä työssä keskitytään niistä vain Laravelin käyttämään PHP-ohjelmointikieleen.

2.2.1 PHP

PHP, koko nimeltään PHP: Hypertext Pre-processor, on tämän hetken käytetyin palvelinpuolen ohjelmointikieli (liite 1). PHP-tiedoston tunnistaa .php-tiedostopäätteestä. Mikä tekee PHP-kielestä erikoisen, on se, että PHP pystyy sisältämään itsensä lisäksi kaiken, mitä HTML-dokumenttiin kuuluu: HTML-, CSS-, ja JavaScript-koodit. PHP-koodiosuuden voi laittaa mihin kohtaan dokumenttia tahansa, kunhan se erotetaan PHP:n alkumerkillä *<?php* ja lopetusmerkillä ?>, jonka väliin tulee PHP-koodiosuus seuraavalla tavalla:

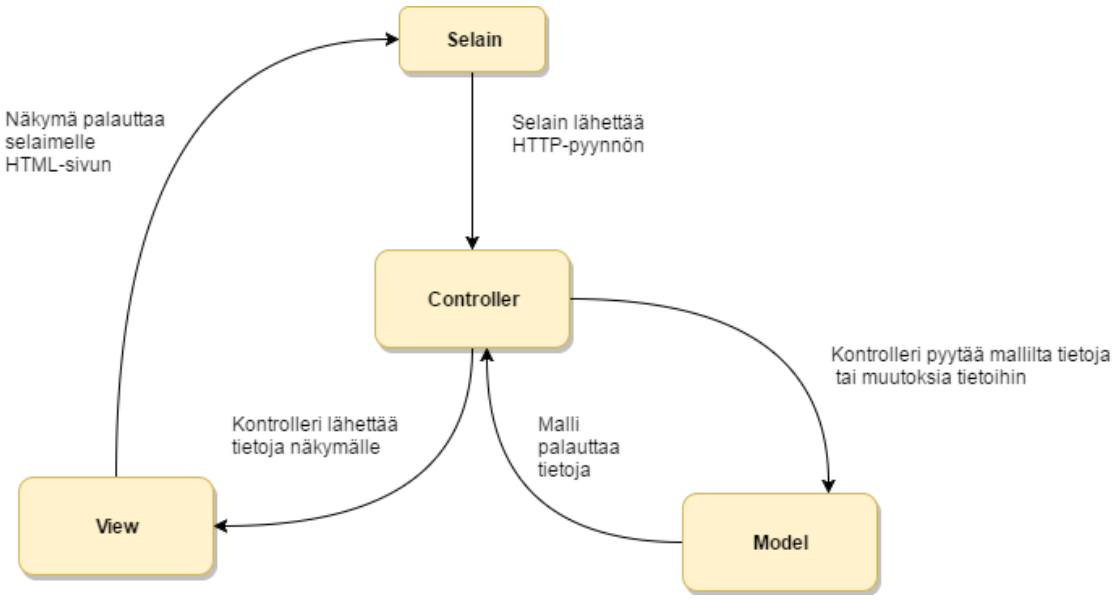
```
<!DOCTYPE html>
<html>
    <head>
        <title>Sivun otsikko ikkunassa</title>
    </head>
    <body>
        <h1>Sivun otsikko sivulla</h1>
        <p>
            <?php
            // PHP koodi tulee tähän
            ?>
        </p>
    </body>
</html>
```

Kuva 12. PHP-tiedosto, joka sisältää HTML- ja PHP-koodia p-tägien sisällä

Kuvan 12 koodiesimerkistä huomataan, että PHP-koodiosuus on haluttu laittaa p-tägien sisään, mutta mikään ei myöskään estä laittamasta sitä osaksi p-tägiä. Laravelin kaikki logiikkaa sisältävät tiedostot on kirjoitettu PHP-kielellä.

2.2.2 MVC-malli verkko-ohjelmoinnissa

Laravel sovelluskehys perustuu MVC-malliin. MVC tulee sanoista Model, Controller ja View. Se on verkko-ohjelmoinnissa sovellusarkkitehtuuri, jossa koko sovelluslogiikka on jaettu kolmeen pääkomponenttiin (4). Sovelluslogiikan jakaminen eri komponentteihin jäsentelee ja siistii sovelluskoodia, jota on myöhemmin helppo ylläpitää. Sovelluskehityksen näkökulmasta koodia on myös helppo käyttää uudelleen, jos uusia ominaisuuksia halutaan lisätä sovellukseen. Yhdessä sovelluksessa voi olla useampaa samaa pääkomponenttia.



Kuva 13. MVC-mallin vastuunjako

Model eli malli käyttää usein tietokantaa, ja sen ainoana tehtävänä on tiedon käsitteleminen ja abstraktointi. Se toteutetaan keräämällä eri tietokohteisiin liittyvät toiminnot luokiksi ja funktioiksi. Tämä taas luo siistin abstraktiotason, joka piilottaa tietokohteen rumat yksityiskohdat itsensä taakse.

Controller eli kontrolleri vastaanottaa käyttäjän selaimelta tulevat HTTP-pyynnöt, ja käsittelee sen. Se käyttää mallia halutun tiedon hakemiseen ja muuttamiseen, ja lopuksi antaa yleensä suorituksen näkymälle. Samalla kontrolleri voi myös välittää muuttujien avulla näkymälle tietoa näytettäväksi.

View eli näkymä ainoastaan määrittää käyttäjälle näkyvän sivun ulkoasun ja kontrollerilta saatujen muuttujien avulla tietojen esityksen. Tietokantasovelluksen tapauksessa näkymät ovat tiedostoja, joissa määritellään vain sivulle tuleva HTML-koodi, ja sen sekaan oikeisiin kohtiin sivua tulevat muuttujat, jotka sisältävät halutut tiedot. Näkymien ei tarvitse tietää, mistä muuttujat saadaan, kunhan ne ovat olemassa.

3 Laravelin asennus ja käyttöönotto

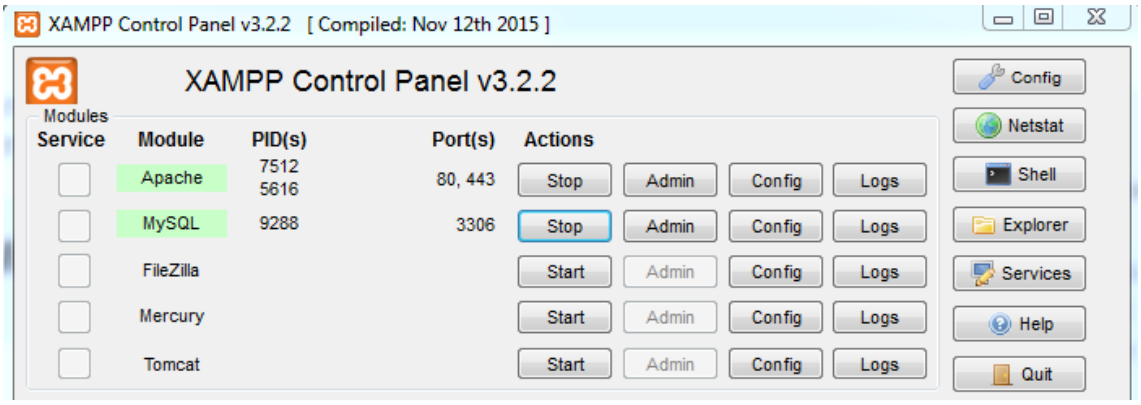
3.1 Palvelinpuolen vaatimukset ja XAMPP:n asennus

Laravel 5.3 vaatii toimiakseen palvelimelta (5, s. 82)

- PHP >= 5.6.4
- OpenSSL PHP-liitännäisen
- PDO PHP-liitännäisen
- Mbstring PHP-liitännäisen
- Tokenizer PHP-liitännäisen
- XML PHP-liitännäisen.

Laravel tarjoaa tätä varten oman Homestead-virtuaalikoneen, mutta sitä ei ole pakko käyttää. Kyseiset kriteerit pystytään myös täyttämään esimerkiksi asentamalla paikallinen palvelinohjelmisto, kuten XAMPP tai WAMPP. Paikallinen palvelinohjelmisto luo omalle koneelle verkkopalvelimen, jota kehittäjät pystyvät käyttämään muun muassa tietokannan, verkkoapplikaation tai verkkosivujen testaamisessa ja käyttöönotossa.

Tässä insinöörityössä asennettiin XAMPP Windows-käyttöjärjestelmään. Kun XAMPP oli asennettu, piti käynnistää Apache ja MySQL, jotta paikallista verkkopalvelinta ja tietokantaa päästäisiin käyttämään.



Kuva 14. XAMPP-ikkuna.

XAMPP vaati myös sen, että kaikki paikallista verkkopalvelinta käyttävät projektit asennettiin xampp-asennussijainnin htdocs-kansion alle (oletuksena Windows-käyttöjärjestelmissä C:\xampp\htdocs). MySQL-rivin Admin-napista päästiin paikalliseen tietokantapalvelimeen käsiksi, jossa pystyi luomaan testitietokannan Laravel-projektia varten, jos sitä tarvittiin.

3.2 Composer

Composer on työkalu PHP:ssa riippuvuuksien hallintaan (6). Sen avulla pystytään projektikohtaisesti asentamaan ja päivittämään kaikki tarvittavat riippuvuudet, joista projekti on itse riippuvainen. Laravelin asennus vaati composerin.

Esimerkiksi sovellukseen halutaan autentikaatiojärjestelmä, joka on jo valmiiksi keksitty jossakin sovelluskehyksessä. Sitten halutaan jokin aikataulujärjestelmä, joka on myös keksitty jossakin toisessa sovelluskehyksessä. Halutut järjestelmät voivat kuitenkin riippua useammasta kuin yhdestä sovelluskehyksestä, jotka myös voisivat riippua toisista sovelluskehyksistä. Siinä composer katsoisi kaikki tarvittavat riippuvuudet jokaiselle, ja asentaisi tai päivittäisi ne omaan projektiin. Tällöin kehittäjän ei tarvitsisi pitää kirjaa siitä, mitä osia projekti ja projektin osat tarvitsevat, sillä composer hoitaisi sen itse. Samalla composer varmistaisi sen, ettei vääriä versioita tarvittavista kirjastoista tulisi vahingossa projektiin, jolloin projekti menisi rikki.

3.2.1 Composerin asennus

Composerin asennus oli helppoa, sillä asennuspaketti oli Windows-käyttöjärjestelmiä varten olemassa. Asennuksen jälkeen pystyi varmistamaan, oliko composer asentunut onnistuneesti, kirjoittamalla komentoriville *composer –v*. Silloin piti tulla näkyviin composerin versionumero:

Kuva 15. Composer-versionumero komentoikkunassa

3.2.2 Laravel-projektin luonti composerin avulla

Kun composer oli asennettu, päästiin luomaan tyhjä Laravel-projektipohja versionumerolla 5.3. Ensiksi kannatti navigoida komentorivillä xampp/htdocs-kansion sisään, sillä XAMPP:in käyttö vaati projektien sijainnin olevan siellä. Laravel-projektipohja luotiin sitten komentorivillä komennolla *composer create-project --prefer-dist laravel/laravel projektinnimi "5.3.*"* . Huomattiin, että uuden Laravel-projektipohjan luonti latasi siihen myös

tarvittavat riippuvuudet Composerin avulla ja generoi samalla oman applikaatioavaimen. Ilman applikaatioavainta käyttäjien istunnot ja muut salatut tiedot eivät olisi turvassa. Sen jälkeen projektipohja oli valmis käytettäväksi. Esimerkissä luodun projektin nimi oli myproject:

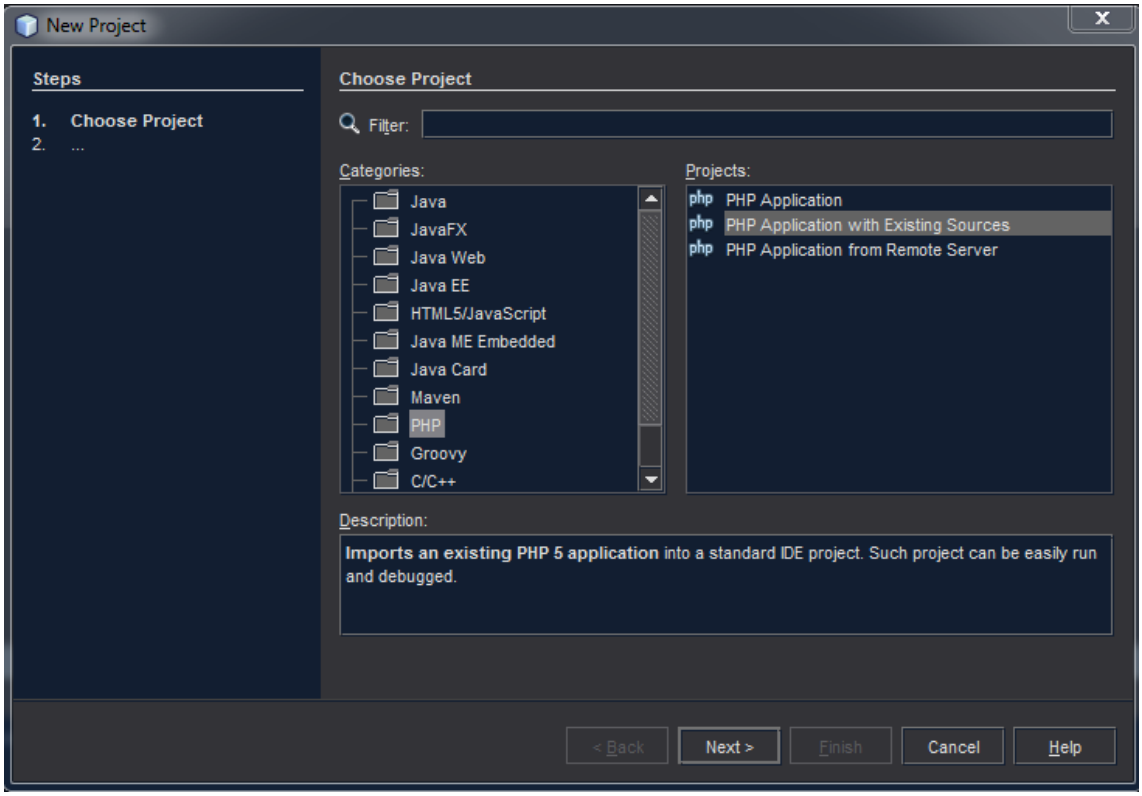
```
C:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel myproject
"5.3.*"
```

Kuva 16. Uuden Laravel-projektin luonti komentoikkunassa composerin avulla

3.3 Ohjelmointiympäristö Netbeans IDE

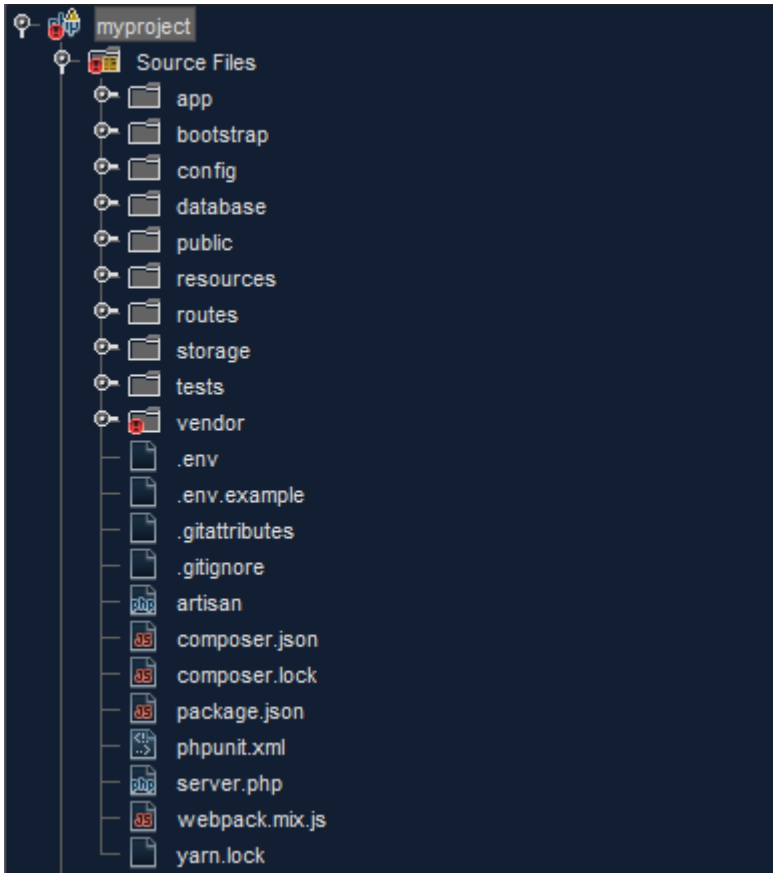
Työssä käytettiin Netbeans IDE:ä (8.2 All-versio) ohjelmointiympäristönä, jonka pystyi lataamaan Netbeansin omilta sivuilta. Kyseinen versio vaati myös JDK 8 tai suuremman version esiasennettuna omalle koneelle. Halutun JDK-version pystyi lataamaan Oraclen omilta sivuilta.

Kun Netbeans oli asennettu ja käynnistetty, piti luoda uusi Netbeans PHP -projekti jo olemassa olevasta Laravel-projektipohjasta:



Kuva 17. Netbeans-projektin luominen Laravel-projektista

Sitten valittiin sijainniksi Laravel-pohjainen projekti, joka luotiin alussa xampp/htdocs-kansioon ja painettiin Finish. Tämän jälkeen Laravel-projekti oli valmis muokattavaksi Netbeansissä, ja projektin rakenne näytti seuraavalta:



Kuva 18. Laravel-projektin tiedostorakenne

3.4 Versionhallinta Git

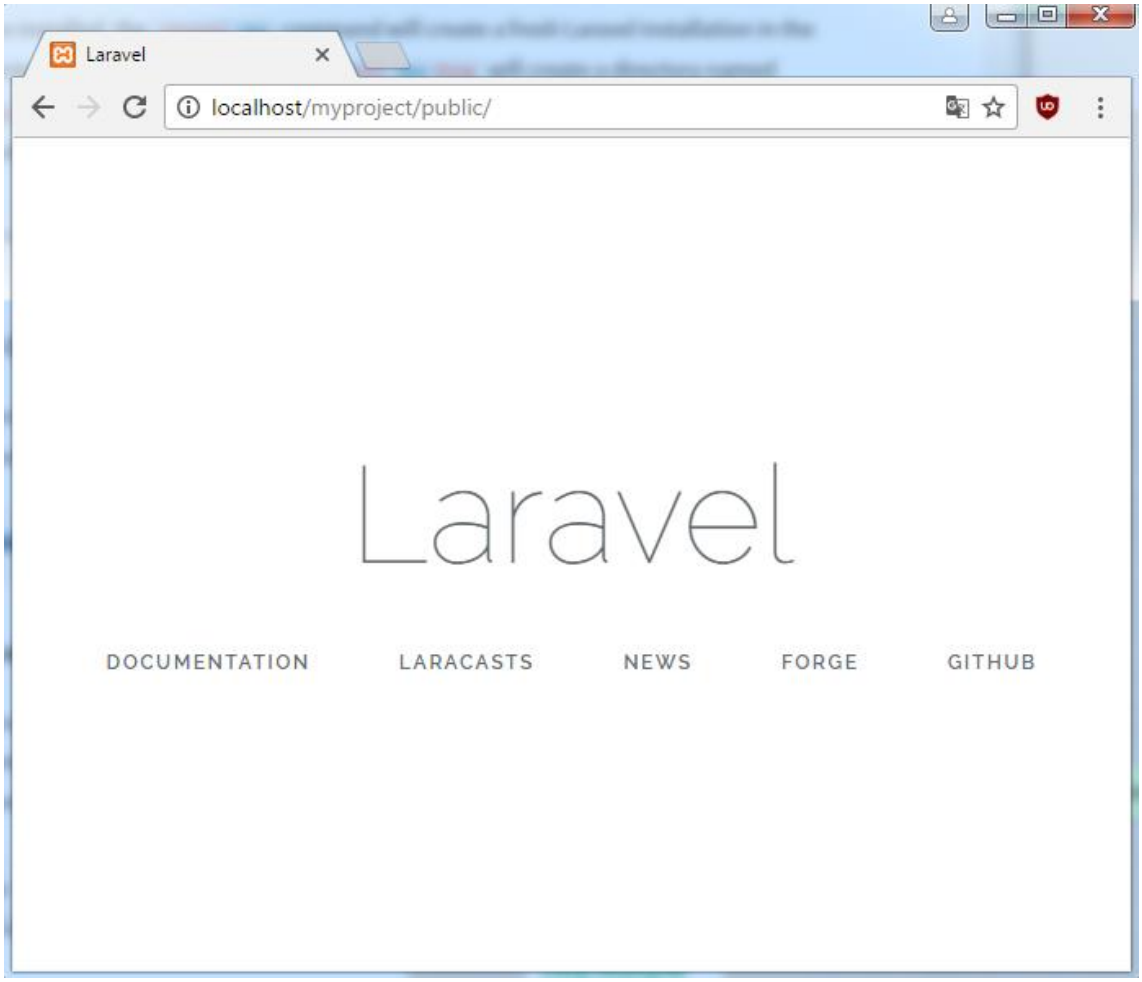
Versiohallinta on tekniikka, jolla pidetään kirjaa muutoksista tiedostotasolla. Se ei ole millään tavalla pakollinen, mutta isojen sovellusten kehittämisessä se on Stuart Yeatesin mielestä välttämätön työkalu (7). Sillä pystytään muun muassa katsomaan, mitä on mihin aikaan tehty, mitä muutos koskee ja kuka sen on tehnyt. Versionhallinnasta on myös hyötyä tiimitasolla ohjelmoitaessa, sillä muilla kehittäjillä on siihen pääsy, ja päällekkäisten muutosten mahdollisuus vähenee. Versionhallinnasta on myös apua, jos halutaan päästä tiedoston vanhempaan versioon käsiksi esimerkiksi, jos jokin uusi muutos tiedostossa on rikkonut sovelluksen.

Hyvän toimintatavan mukaan sovelluksen lähdekoodit pidetään versionhallinnassa. Suosituin versionhallintajärjestelmä on tällä hetkellä Git. Laravel-projektipohja sisältää Gitiä hyödyntävän .gitignore-tiedoston (kuva 18), joka määrittelee valmiiksi Gitille, mitä tiedostoja ei lisätä ja mitä lisätään versionhallintaan. Yleissääntönä on, että vain kaikki tarvittavat riippuvuudet ja salaista tietoa sisältävät tiedot jätetään versionhallinnasta pois. Laravelin .gitignore on jo valmiiksi määritelty sellaiseksi. Tarvittavat riippuvuudet voivat olla hyvin isoja, yli kymmenientuhansien tiedostojen kokoisia, eikä niihin tehdä itse ohjelmallisia muutoksia, jolloin niitä ei ole järkeä pitää versionhallinnassa. Salaista tietoa esimerkiksi salasanoja sisältävät tiedostot myös jätetään pois, koska muuten ne näkyisivät kaikille, joilla on pääsy projektin versionhallintaan. Laravel-projektin tapauksessa se tarkoittaa applikaation ympäristöä käsittelevää .env-tiedostoa (kuva 18), joka sisältää muun muassa salasanat tietokantaan ja tarvittavaan sähköpostitiliin.

Kun projektipohja on asetettu versionhallintaan, toinen ohjelmoija, jolle on annettu oikeudet kyseisen projektin versionhallintaan, voi kopioida kaikki tiedostot sieltä omalle koneelleen. Huomataan kuitenkin, että toisen ohjelmoijan kopiosta puuttuu kaikki riippuvaisuustiedostot sekä .env-tiedosto, sillä ne oli jätetty pois versionhallinnasta. Composerin omat muistitiedostot: composer.json ja composer.lock (kuva 18) ovat kuitenkin versionhallinnassa. Ne sisältävät tietoa tarvittavista riippuvuuksista, joiden avulla avulla composer pystyy palauttamaan/asentamaan kaikki tarvittavat riippuvaisuustiedostot, jotka oli jätetty pois versionhallinnasta omalle kopioprojektille. Pitää aluksi mennä komentorivillä projektin sijaintiin, ja sitten suorittaa komento *composer install,* jonka jälkeen projektissa on samat riippuvaisuustiedostot kuin alkuperäisessä projektissa. .env-tiedosto pitää sen sijaan antaa käsin toiselle ohjelmoijalle versionhallinnan turvallisuusriskien vuoksi.

3.5 Laravel-projektin alkunäkymä

Kun edelliset vaiheet oli tehty, Laravel-projektin nettiselainnäkymään pääsi kirjoittamalla selaimen osoitekenttään *http://localhost/projektinnimi/public*. Edellisen esimerkin mukaisesti projektin nimi oli myproject, jolloin selaimen osoitekenttään olisi pitänyt kirjoittaa *http://localhost/myproject/public* . Silloin näkymän pitäisi näyttää tältä:



Kuva 19. Laravel-projektipohjan näkymä nettiselaimella

Osoitekentän localhost viittasi xampp/htdocs-kansioon ja myproject viittasi laravelin projektikansion nimeen. Public vaadittiin vielä siksi, koska se on projektikansion sisällä oleva kansio, joka sisältää index.php-tiedoston. Laravelin dokumentaation mukaan index-php-tiedosto toimii etupään kontrollerina kaikkia sovellukseen meneviä HTTP-pyyntöjä varten. Se siis määrittää kotisivun sijainnin.

4 Verkkosovelluskehitys

4.1 PHP artisan

Artisan on komentorivikäyttöliittymä (5, s. 556), joka tulee Laravel-projektin mukana. Se sisältää paljon projektin kehitystä nopeuttavia komentoja. Kaikki Artisan-komennot voi

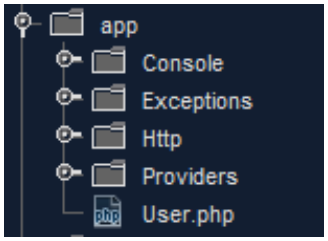
nähdä kirjoittamalla komentoriville *php artisan list*, kun ollaan projektikansiossa (kts. liite 1).

Esimerkiksi halutaan luoda Controller, joka hoidetaan Artisan komennolla *php artisan make:controller tiedostonnimi*. Artisanin avulla Controller luodaan oikeaan paikkaan hakemistorakenteessa ja siihen liitetään kaikki tarvittavat pohjakoodit, jotka se tarvitsee toimiakseen. Tällä tavoin vältytään helpommin inhimillisiltä virheiltä, kuten väärältä tiedostosijainnilta. Artisan sallii myös omien komentojen luomisen.

Tässä insinöörityössä tullaan pääasiassa käyttämään Artisania tiedostojen ja ominaisuuksien luomiseen.

4.2 Sovelluksen infastruktuuri lyhyesti

4.2.1 App-hakemisto



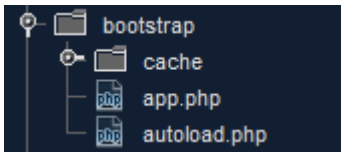
Kuva 20. App-hakemisto projektipohjassa

App-hakemistossa (5, s. 89) pidetään suurin osa sovelluslogiikasta. Hakemiston alla oleva Console-hakemisto sisältää tietoa omista Artisan-komennoista, jos niitä on luotu. Exception-hakemistossa määritellään miten käsitellään mitkäkin ajonaikaiset poikkeukset. Http-hakemisto sisältää kaikki Controller-, Middleware- ja Form Request-tiedostot. Niiden tehtävänä on käsitellä kaikki sovellukseen kohdistuvat HTTP-pyynnöt, ja ne sisältävät suurimman osan sovelluslogiikasta. Providers-hakemisto vastaa tiedostojen rekisteröimisestä sovellukseen, jotta sovellus tietää, mitä tiedostoja se tulee käyttämään.

4.2.2 Bootstrap-hakemisto

Bootstrap-hakemisto (5, s. 90) sisältää kaikki sovelluskehyksen esilataustiedostot, eikä pidä sekoittaa jo edellämainittuun Bootstrap-sovelluskehykseen. Se sisältää myös

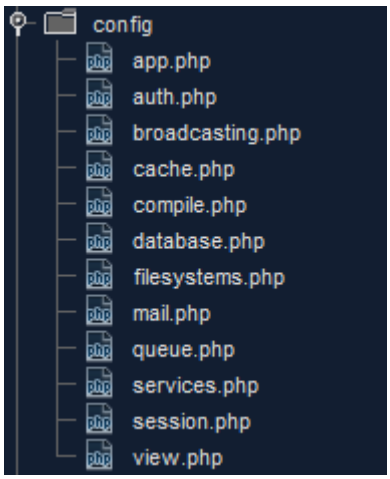
cache-hakemiston, jossa pidetään vaihtoehtoisia sovelluskehyksen generoimia välimuistitiedostoja. Niiden tarkoituksena nopeuttaa itse sovellusta. Sovellusta nopeuttavien tiedostojen generoinnit ja poistamiset hoidetaan Artisanin avulla.



Kuva 21. Bootstrap-hakemisto projektipohjassa

4.2.3 Config-hakemisto

Config-hakemisto (5, s. 90) sisältää kaikki sovelluksen asetusten kannalta tarpeelliset tiedot, esimerkiksi lokalisaatio-asetukset, istunnon elinajan ja niin edelleen. Lokalisaatio-asetuksia muuttamalla voidaan valita ensisijainen ja toissijainen kieli sovellukselle, jonka oletus on englanti. Istunnon elinaika määrittelee ajan, jolloin käyttäjä saa olla sovelluksessa kirjautuneena ja toimettomana, ennen kuin hänet potkitaan ulos. Oletuksena se on määritelty kahteen tuntiin.



Kuva 22. Config-hakemisto projektipohjassa

4.2.4 Database-hakemisto

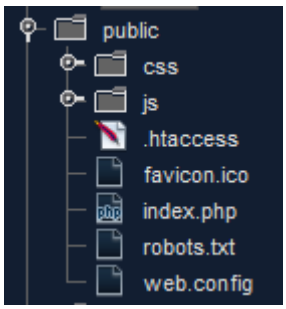
Database-hakemisto (5, s. 90) sisältää tietoa tietokannasta. Siellä sisältävien Migrationsien ansiosta pystytään pitämään tietokannan luonti ikään kuin versionhallinnassa. Sen avulla tietokannan testaaminen on helpompaa, sillä tietokannan luontitieto kulkee sovelluksen sisällä mukana. Tällä tavoin toisten ohjelmoijien ei tarvitse erikseen pyytää ko-

piota tietokannasta, joka jouduttaisiin lähettämään ja päivittämään joka kerta, kun tietokanta muuttuu. Database-hakemisto sisältää myös Seeder-tiedostot, joiden tehtävänä on täyttää luotu tietokanta testidatalla ja Factories-tiedostot, jotka hoitavat saman asian vaihtoehtoisella tavalla.

Kuva 23. Database-hakemisto projektipohjassa

4.2.5 Public-hakemisto

Public-hakemisto (5, s. 90) on kaikkien HTTP-pyyntöjen sisääntulokohta siinä olevan index.php-tiedoston vuoksi. Siksi pitää varmistaa, että public-hakemisto on palvelinsivun juuressa, kun projekti viedään tuotantoon oikealle palvelimelle. Public-hakemisto sisältää myös kaikki sovelluksessa tarvittavat kuvat, CSS- ja JavaScript-tiedostot.



Kuva 24. Public-hakemisto projektipohjassa

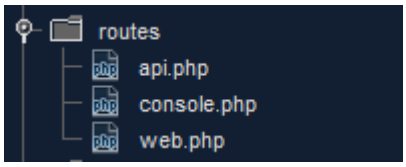
4.2.6 Resources-hakemisto

Resource-hakemisto (5, s. 90) vastaa kaikista näkymistä ja niihin liittyvistä kielitiedostoista. Sen lisäksi se voi sisältää assets-kansiossa CSS- ja JavaScript-tiedostoja, jos ne halutaan yhdistää. Yhdistämisellä tarkoitetaan sitä, että useampaa CSS- ja JavaScript-tiedostoa linkitetään yhteen yhdeksi yhteiseksi CSS- ja JavaScript-tiedostoksi, joihin on helppo viitata näkymissä. Tämän käyttö ei ole kuitenkaan pakollista, ja suoritetaan yleensä vasta tuotantovaiheessa. Lang-hakemisto sisältää nimensä mukaisesti kaikki halutut käännökset eri kielille. Views-hakemisto sisältää sen sijaan vain näkymiä eli tiedostoja, joissa määritellään vain sivulle tuleva HTML-koodi, ja sen sekaan tulevat muuttujat, jotka sisältävät halutut tiedot.

Kuva 25. Resource-hakemisto projektipohjassa

4.2.7 Routes-hakemisto

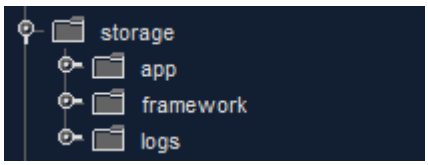
Routes-hakemisto (5, s. 90) sisältää kaikki sovelluspolkujen määrittelyt. Sovelluspoluilla tarkoitetaan URL:n jatko-osuutta, esimerkiksi URL:ssä http://www.hs.fi/uutiset se olisi /uutiset. Routes-hakemiston tehtävänä on linkittää sovelluspolut haluttuihin kontrollerin funktioihin.



Kuva 26. Routes-hakemisto projektipohjassa

4.2.8 Storage-hakemisto

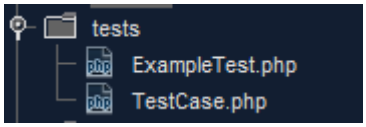
Storage-hakemisto (5, s. 91) sisältää kaikki sovelluksen ja sovelluskehyksen generoimat tiedostot lukuunottamatta bootstrap-hakemiston välimuistia nopeuttavia tiedostoja. Se on jaettu kolmeen kansioon. App-hakemistoon tallennetaan kaikki tiedostot, jotka sovellus luo itse. Framework-hakemistoon tallennetaan sovelluskehyksen generoimat tiedostot. Logs-hakemistoon on tarkoitus tallentaa sovelluksen lokitiedostot. Sovelluskehityksen kannalta storage-hakemistoon ei tarvitse koskea, sillä Laravel on automatisoitu pitämään storage-tiedostoja aina ajan tasalla.



Kuva 27. Storage-hakemisto projektipohjassa

4.2.9 Tests-hakemisto

Test-hakemistoon (5, s. 91) lisätään omat automatisoidut yksikkötestaukset. Yksikkötestauksilla tarkoitetaan sovelluskoodiin kuuluvien yksittäisten osien testaamista. Sen tarkoituksena on pilkkoa sovelluskoodit pieniin testattaviin yksiköihin ja varmistaa samalla niiden oikeellisuus.



Kuva 28. Test-hakemisto projektipohjassa

4.2.10 Vendor-hakemisto

Vendor-hakemisto (5, s. 91) sisältää kaikki composerin avulla asennetut riippuvuustiedostot, ja on kooltaan hyvin suuri. Vendor-kansioon ei myöskään tehdä omia muokkauksia, sillä ne häviävät, kun composer päivittää tai lataa riippuvaisuustiedostot. Syynä tähän on se, että omat muutokset eivät tallennu alkuperäisiin tiedostoihin, josta ne ovat peräisin. Näistä syistä johtuen vendor-hakemisto on jätetty valmiiksi pois versionhallinnasta.

4.3 Autentikaatiojärjestelmän luonti

Laravel sisältää helppokäyttötoiminnot turvallisen autentikaatiojärjestelmän luomiseen (5, s. 275). Laravelin autentikaatio koostuu guards ja providers-olioista. Guards-olioiden määrittää, millä tavalla käyttäjä autentikoidaan. Providersien tehtävänä on sen sijaan määrittää, miten haluttu käyttäjä saadaan tietokannasta. Näitä ei kuitenkaan tarvitse muokata, sillä Laravel on jo valmiiksi konfiguroinut ne toimivaksi kokonaisuudeksi. Laravelin omaa guards-oliota kutsutaan session guardiksi, jonka tehtävänä on ylläpitää istunnon tilaa storage-hakemiston istuntotiedostojen ja evästeiden avulla. Laravelin Provider on asetettu hakemaan käyttäjä tietokannasta Eloquentin ja Database Query Builderin avulla, joista kerrotaan lisää myöhemmin.

Laravelin autentikaatiojärjestelmän luomisella oli kaksi vaatimusta. Ensimmäinen niistä oli se, että tietokanta olisi luotu. Toinen vaatimuksista taas oli sähköpostitilin tiedot. Tietokantaa tarvittiin siihen, että siihen tallennettaisiin kaikki käyttäjätiedot, kuten sähköpostiosoitteet ja salatut salasanat. Sen luomiseen käytettiin XAMPP-ohjelmistoa.

Sähköpostitiliä tarvittiin ”unohdin salasanani” -toimintoon, sillä Laravelin autentikaatiojärjestelmässä salasananvaihtolinkki lähetettäisiin käyttäjän rekisteröimään sähköpostiosoitteeseen, jonka lähettäjänä oli Laravel-sovelluksessa määritelty sähköpostiosoite. Tässä työssä tehtiin dummy gmail -sähköpostitili, jota käytettiin vain salasananvaihtolinkin lähettämiseen.

Kun tietokanta ja sähköpostitili on luotu, ne pitää laittaa projektin juuressa olevaan .env-tiedostoon, joka näyttää projektipohjassa tältä:

```
APP_ENV=local
APP_KEY=base64:fQOYqmcUe4zeCJEGIcGXUIUJmRrfQRUvuzamFoFgmQI=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

PUSHER_APP_ID=
PUSHER_KEY=
PUSHER_SECRET=
```

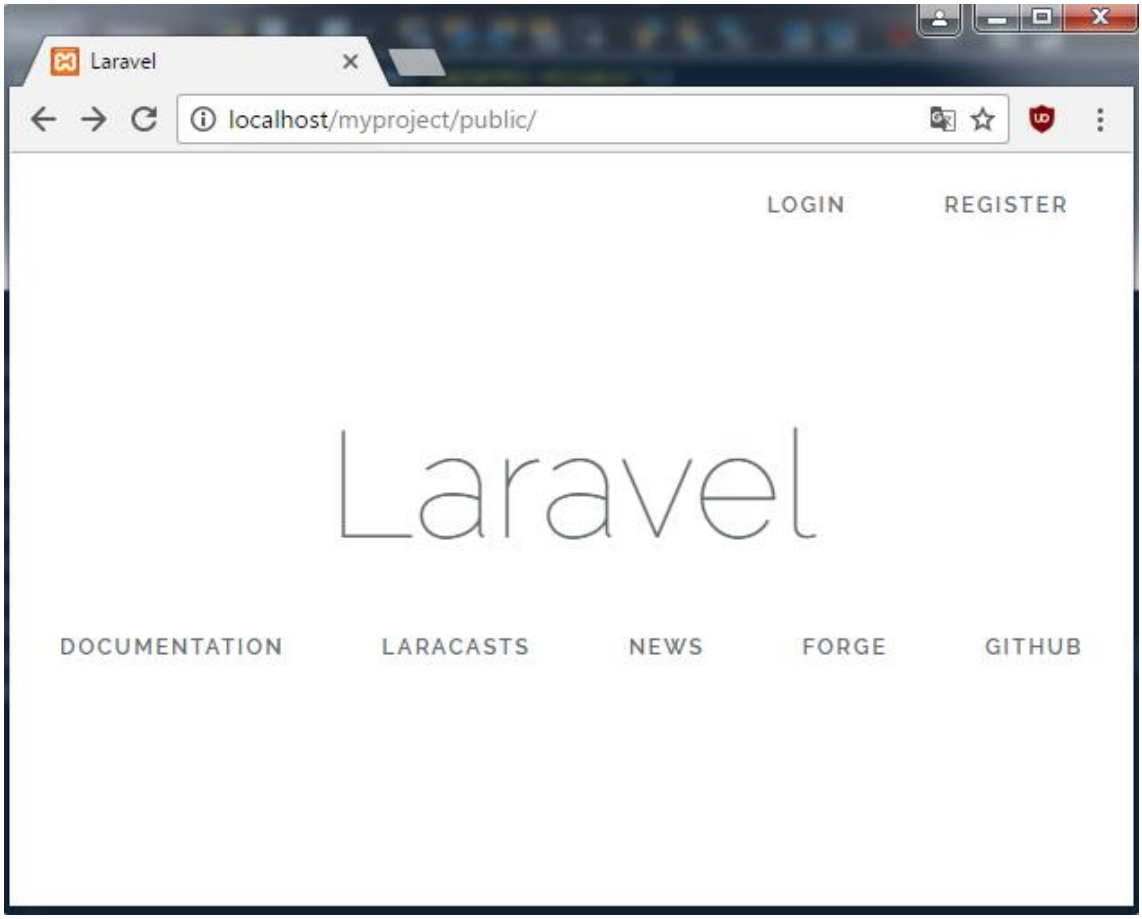
Kuva 29. Projektipohjan juuressa olevan .env-tiedoston sisältö

Tietokantaa koskevat muuttujat olivat DB-alkuisia. Laravelin .env-tiedoston sisältö oli konfiguroitu valmiiksi paikallista palvelinohjelmistoa varten, jolloin DB_HOST ja DB_PORT-arvoja ei tarvinnut muuttaa. Koska tietokanta luotiin tässä työssä XAMPP:in

oletusarvojen avulla, DB_USERNAME-arvoksi asetettiin root, ja DB_PASSWORD-arvo jätettiin tyhjäksi. DB_DATABASE-arvoon laitettiin kuitenkin luodun tietokannan nimi.

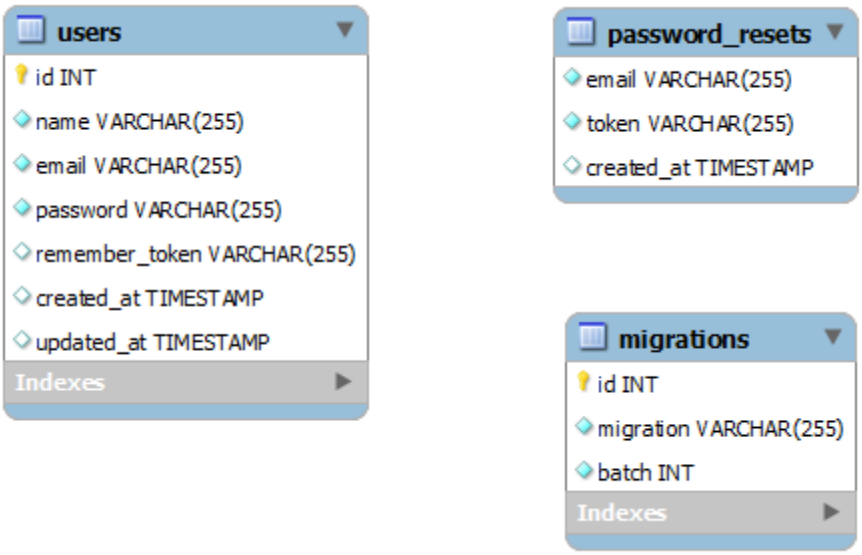
MAIL-alkuiset muuttujat koskivat sähköpostitilin arvoja. MAIL_USERNAME ja MAIL_PASSWORD -arvoiksi asetettiin halutun sähköpostitilin käyttäjätunnus ja salasana. Muut MAIL-muuttujien arvot täytettiin sähköpostitarjoajan suositusten mukaisesti.

Tämän jälkeen käytettiin Laravelin PHP Artisania autentikaatiojärjestelmän luomiseen menemällä komentorivillä projektikansioon ja kirjoittamalla sinne *php artisan make:auth*. Artisan-komento make:auth luo kaikki tarvittavat kontrollerit, näkymät ja sovelluspolkumäärittelyt autentikaatiojärjestelmää varten oikeisiin paikkoihin sovelluksen hakemistorakenteessa. Sen jälkeen piti vielä luoda käyttäjiä varten tarkoitetut taulut tietokantaan tarvittavine attribuutteineen, ja se tapahtui komennolla *php artisan migrate*. Kun autentikaatiojärjestelmä oli lisätty, sovelluksen alkunäkymässä näkyi navigaatiopalkin oikealla puolella kirjautumis- ja rekisteröitymismahdollisuudet:



Kuva 30. Laravel-projektin alkunäkymä selaimella autentikaatiojärjestelmän lisäyksen jälkeen

Tietokannan puolella tuli luotua kyseiset taulut:



Kuva 31. Laravel-autentikaatiojärjestelmän luomat taulut tietokannassa

Laravel-autentikaatiojärjestelmässä käyttäjän salasana enkryptataan eli salataan ennen kuin se tallennetaan tietokantaan. Käytännössä se tarkoittaisi sitä, että jos hakkeri pääsisi jollain tavalla murtautumaan tietokantaan ja hakemaan sieltä salasanatiedot ulos, oikeaa salasanaa ei saataisi selville, koska salasana oli muutettu salauksen mukaisesti joksikin toiseksi. Laravelissa enkryptaus tapahtuu OpenSSL-tekniikan avulla, joka perustuu AES-256- tai AES-128-salaukseen, ja käyttää .env-tiedostossa olevaa APP_KEY-kentän arvoa (ks. kuva 29). APP_KEY-kentän arvo generoidaan jokaiselle Laravel-projektipohjalle erikseen, jolloin enkryptaus ei ole Laravel-sovellusten välillä koskaan täysin sama, mikä lisää turvallisuutta.

4.4 Eloquent

Ennen kuin tietokannasta pystyy lukemaan tai muuttamaan tietoa, täytyy jokaista tietokanta-taulua vastaan olla oma Model-luokkansa MVC-arkkitehtuurin mukaisesti (ks. kuva 13). Laravel käyttää Eloquent Model-luokkia MVC-arkkitehtuurin Model-osana (5, s. 487). Eloquent Model eroaa tyypillisestä Model-luokasta niin, että siinä ei tarvitse määritellä taulun attribuutteja. Autentikaatiojärjestelmän mukana Laravel oli luonut User-nimisen Eloquent Model-luokan valmiiksi, mutta omia Eloquent Model-luokkia luotiin Artisan komennolla *php artisan make:model Mallinnimi*.

4.4.1 Tietokantatauluun viittaaminen

Havainnollistetaan tätä kindergartens-taululla, joka on luotu tietokantaan. Parhaan käytännön mukaisesti kaikki ohjelmointiin liittyvät attribuutit ja muuttujat tulisi nimetä englanniksi sen kieliasemansa vuoksi. kindergartens-taululla voisi olla attribuutteina esimerkiksi id, name ja notes. Luodaan Kindergarten-niminen Eloquent Model-luokka Artisan komennolla: *php artisan make:model Kindergarten*. Sen jälkeen huomataan, että heti apphakemiston alle on generoitu Eloquent Model luokkaa vastaava Kindergarten.php, johon pitäisi lisätä vielä viittaus tietokantataulun nimeen, joka on tässä tapauksessa kindergartens, table-muuttujan avulla seuraavasti:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Kindergarten extends Model {

    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'kindergartens';

}
```

Kuva 32. Kindergarten Eloquent Model -luokka, jossa on viittaus kindergartens-tauluun

Tämän jälkeen kaikkiin kindergartens-taulun attribuutteihin päästään käsiksi Kindergarten Eloquent Model -luokan avulla. Eloquent Model -luokka kuitenkin olettaa, että viitattavassa tietokantataulussa on attribuutit created_at ja updated_at, joita ei kindergartens-tietokantataulussa ole. Siinä tapauksessa pitää vielä asettaa timestamps-muuttuja epätodeksi, joka tapahtuu lisäämällä seuraava koodi table-muuttujan alle:

```php
/**
 * Indicates if the model should be timestamped.
 *
 * @var bool
 */
public $timestamps = false;
```

Kuva 33. Timestamp-muuttuja, joka asetetaan epätodeksi

Eloquent Model-luokkaan voi myös määrittää salaisia attribuutteja, joita ei pitäisi koskaan näyttää, kun haetaan tietoa Eloquentin avulla. Kyseiset attribuutit ovat yleensä salasanoja. Salaiset attribuutit pilkuilla erotettuina asetetaan hidden-muuttujan arvoiksi lisäämällä seuraava koodi Eloquent Model -luokan sisälle:

```
/**
 * The attributes that should be hidden for arrays.
 *
 * @var array
 */
protected $hidden = [
    'password',
];
```

Kuva 34. Hidden-muuttuja, johon asetetaan piilotettavaksi attribuutiksi taulun salasanat

4.4.2 Tietokantataulujen välinen suhde

Yhdessä tietokannassa voi olla monta taulua, jotka ovat liittyneet toisiinsa jollakin tavalla. Ne pitää silloin myös ilmaista Eloquent Model-luokissa funktioiden avulla. Esimerkiksi edellä mainitun kindergartens-taulun lisäksi voisi olla olemassa groups-taulu, jossa olisi attribuutteina id, name ja max_age. Tietokannan jokaista taulua varten piti tehdä Eloquent Model-luokka, joten piti luoda uusi Eloquent Model-luokka Group-nimellä, ja se luodaan Artisan komennolla: *php artisan make:model Group*.

Yhdellä päiväkodilla voi olla monta ryhmää, jolloin kindergartens-taulun suhde groups-tauluun olisi yhden suhde moneen. Silloin Kindergarten Eloquent Model-luokassa määriteltäisiin omistussuhde moneen groupiin seuraavalla funktiolla:

```
/**
 * Get the groups for the kindergarten.
 */
public function groups()
{
    return $this->hasMany('App\Group');
}
```

Kuva 35. Kindergarten Eloquent Model -luokan funktio, jossa ilmaistaan omistussuhdetta moneen groupiin

Huomataan, että hasMany-funktio ottaa parametriksi App\Groupin, joka viittaa Eloquent Model -luokan Groupiin. Syy siihen, miksi App-polku pitää olla Groupin edessä johtuu

siitä, että kaikki Eloquent Model -luokat sijaitsevat App-nimiavaruudessa, jolloin niihin viitataan App-nimiavaruuden kautta.

Group Eloquent Model -luokassa sen sijaan pitää määrittää kuuluvuussuhde päiväkotiin eli Kindergarten Eloquent Model -luokkaan. Ryhmä voi kuulua vain yhteen päiväkotiin eli pitää määrittää kuuluvuussuhde yhteen seuraavalla funktiolla:

```
/**
 * Get the kindergarten that owns the group.
 */
public function kindergarten()
{
    return $this->belongsTo('App\Kindergarten');
}
```

Kuva 36. Group Eloquent Model -luokan funktio, jossa ilmaistaan kuuluvuussuhdetta yhteen kindergarteniin

Kaikki suhdemäärittelyfunktiot, kuten hasMany ja belongsTo vaativat aina Eloquent Model -luokan, joihin päästään käsiksi App-nimiavaruuden avulla.

Jos jokaisella päiväkodilla saisi olla vain yksi ryhmä, niin silloin olisi kyseessä yhden suhde yhteen. Siinä tapauksessa kuvan 35 hasMany-funktio muutettaisiin hasOne-funktioksi. Kuvan 36 funktiota ei tarvitsisi muuttaa, sillä ryhmä voi edelleen kuulua vain yhteen päiväkotiin.

Joskus on kuitenkin kyse monen suhde moneen -tapauksesta, jossa esimerkiksi yhdellä päiväkodilla voi olla monta ryhmää, mutta samoja ryhmiä pitää myös jakaa monen päiväkodin kesken. Tällöin sekä Kindergarten että Group Eloquent Model -luokkiin pitää määritellä kuuluvuus moneen seuraavasti:

```
/**
 * The groups that belong to the kindergarten.
 */
public function groups()
{
    return $this->belongsToMany('App\Group');
}
```

Kuva 37. Kindergarten Eloquent Model -luokan funktio, jossa ilmaistaan kuuluvuussuhdetta moneen groupiin

```php
/**
 * The kindergartens that belong to the group.
 */
public function kindergartens()
{
    return $this->belongsToMany('App\Kindergarten');
}
```

Kuva 38. Group Eloquent Model -luokan funktio, jossa ilmaistaan kuuluvuussuhdetta moneen kindergarteniin

4.4.3 Datan hakeminen tietokannasta

Kun Eloquent Model -luokat ja niiden väliset suhteet on määritelty tietokannan mukaisiksi, voidaan alkaa hakemaan dataa tietokannasta niiden avulla. Dataa voidaan hakea kahdella tavalla, joista yksi perustuu Eloquentiin. Eloquentiin perustuvassa datahaussa voidaan ajatella Eloquent Model -luokkaa tietokannassa tapahtuvan kyselyn rakentajana.

Edellä määriteltiin päiväkodin ja ryhmän suhde yhden suhteena moneen. Seuraavassa koodissa havainnollistetaan kyseisessä tapauksessa päiväkodin ja ryhmien datan hakua Eloquent Model -luokan tarjoavien metodien avulla:

```php
<?php

use App\Kindergarten;

// Retrieve all kindergartens...
$kindergartens = Kindergarten::all();

// Retrieve a kindergarten model by its primary key 1...
$kindergarten = Kindergarten::find(1);

// Retrieve groups that belong to the kindergarten...
$groups = $kindergarten->groups;

//iterate through all groups and print each one's name and max age
foreach ($groups as $group) {
    echo $group->name;
    echo $group->max_age;
}
```

Kuva 39. Koodiesimerkki datan hakemisesta Eloquentin avulla

Kuvan 39 koodissa käytettiin App-nimiavaruudessa olevaa Kindergarten Eloquent Model -luokkaa, jolloin se piti määrittää käytettäväksi alussa use-määrittelyllä. Sen jälkeen Kindergarten Eloquent Model -luokkaan pystyi viittaamaan koodissa suoraan Kindergarten-nimellä. Eloquent Model -metodeja kutsuessa suoraan Eloquent Model -instanssin kautta piti käyttää kahta kaksoispistettä. Syynä siihen on se, että PHP-kielessä kutsutaan staattisia metodeja toisesta luokasta kahden kaksoispisteen avulla.

Alussa kindergartens-muuttujaan asetettiin kaikki tietokannan päiväkodit Eloquent-luokan all-metodin avulla. Sen jälkeen kindergarten-muuttujaan asetettiin halutulla id-pääavaimella sen omaava päiväkoti tietokannassa find-metodilla. Find-metodi vaati parametriksi pääavaimen arvon, joka oli tässä tapauksessa yksi. Eloquent Model -metodeja on enemmänkin, mutta edellä mainitut ovat yleisimmin käytetyt metodit.

Eloquent Model -metodien palauttama data sisältää myös kyseisessä Eloquent Model -luokassa määritellyt suhteet. Sekä kindergartens- että kindergarten-muuttujat sisältävät siis kaikki Kindergarten Eloquent Model -luokassa määritellyt suhteet muihin Eloquent Model -luokkiin. Kuvassa 21 Kindergarten Eloquent Model -luokassa määriteltiin suhde Group Eloquent Model -luokkaan groups-funktiolla, joten kaikki päiväkotiin liittyvät ryhmät saatiin sitä kutsumalla, kuten huomataan kuvan 39 groups-muuttujan asettamisessa. Lopussa käytiin kaikki ryhmät läpi iteroimalla groups-muuttuja, ja tulostettiin jokaisen group-muuttujan name-attribuutti eli ryhmän nimi, ja max_age-attribuutti eli maksimi-ikäraja, echo-funktion avulla.

Eloquent Model -metodeihin voidaan myös lisätä lisämäärityksiä tai rajoitteita Collection-luokan avulla, jos kyseiset metodit palauttavat useamman kuin yhden objektin. Esimerkiksi kuvan 39 tapauksessa halutaankin rajoittaa groups-muuttujaan vain ne päiväkodin ryhmät, joiden maksimi-ikäraja on seitsemän. Siihen käytetään Collection-luokan where-metodia, jolloin kuvan 39 koodi muuttuu seuraavanlaiseksi:

```php
<?php

use App\Kindergarten;

// Retrieve all kindergartens...
$kindergartens = Kindergarten::all();

// Retrieve a kindergarten model by its primary key 1...
$kindergarten = Kindergarten::find(1);

// Retrieve groups where that belong to the kindergarten
// and group's max age is 7 ...
$groups = $kindergarten->groups->where('max_age', 7)->all();

//iterate through all groups and print each one's name and max age
foreach ($groups as $group) {
    echo $group->name;
    echo $group->max_age;
}
```

Kuva 40. Koodiesimerkki datan hakemisesta Eloquentin avulla, johon on lisätty rajoite

Huomataan, että ainoa muutos koodissa on where-metodin kutsumisen lisäys groups-muuttujan määrittelyssä, jonka jälkeen kutsutaan all-metodia. Where-metodi vaatii ensimmäiseksi parametriksi attribuutin nimen, mikä oli tässä tapauksessa max_age eli maksimi-ikäraja. Toiseksi parametriksi vaadittiin sille haluttu arvo, joka oli tässä tapauksessa seitsemän. All-metodia kutsuttiin vielä lopuksi, sillä sen avulla kerrottiin lisämääritysten lopetuskohta. Samalla se palautti kaikki ryhmät, jotka täyttivät sitä ennen määritetyt lisämääritykset tai rajoitteet. Tämän jälkeen groups-muuttuja sisälsi vain ne ryhmät, joiden maksimi-ikäraja oli seitsemän. Collection-luokan metodeja on paljon enemmänkin, mutta ne voi katsoa Laravelin dokumentaatiosta.

4.4.4 Datan luominen, muokkaaminen ja poistaminen

Tässä osiossa käytetään havainnollistavana esimerkkinä edellisissä osioissa määriteltyä Kindergarten Eloquent Model -luokkaa ja siihen liittyvää kindergarten-tietuetta tietokannassa.

Uuden tietueen luominen tietokantaan (vrt. tietokannan INSERT-kysely) tapahtuu luomalla uusi Model-instanssi, jonka jälkeen siihen asetetaan halutut attribuutit ja kutsutaan save-metodia. Seuraavassa koodissa luotiin uusi Kindergarten Eloquent Model -luokan

instanssi, jonka nimeksi asetettiin Päiväkoti Viskurikuja. Sen jälkeen kutsuttiin save-metodia, joka tallensi kyseisen Model-luokan instanssin tietokantaan uutena tietueena.

```php
<?php
use App\Kindergarten;

$kindergarten = new Kindergarten;

$kindergarten->name = 'Päiväkoti Viskurikuja';

$kindergarten->save();
```

Kuva 41. Koodiesimerkki uuden tietueen luomisesta tietokantaan

Tietokannassa olevan tietueen muokkaamisessa (vrt. tietokannan UPDATE-kysely) pitää ensiksi hakea muokattava tietue esimerkiksi Eloquent Model -luokan find-metodilla (ks. kuva 40). Sen jälkeen muokattavalle attribuutille voidaan asettaa uusi arvo, ja tallentaa se samalla tavalla kuin kuvan 40 koodissa. Seuraavassa koodissa haettiin päiväkoti pääavaimen arvolla 1, jonka jälkeen sen nimi muutettiin Päiväkoti Jäkäläksi. Sen jälkeen tallennettiin muutokset tietokantaan save-metodilla.

```php
<?php

use App\Kindergarten;

$kindergarten = Kindergarten::find(1);

$kindergarten->name = 'Päiväkoti Jäkälä';

$kindergarten->save();
```

Kuva 42. Koodiesimerkki tietokannan tietueen muokkaamisesta

Tietueen poistaminen (vrt. tietokannan DELETE-kysely) tietokannasta tapahtui hakemalla poistettava tietue esimerkiksi find-metodilla johonkin muuttujaan, kuten edellisessä tapauksessa. Sen jälkeen muuttujasta piti kutsua delete-metodia, jolloin haettu tietue poistettiin myös tietokannasta. Tietueen poiston voi tehdä myös lyhyemmin, jos tietää sen pääavaimen arvon kutsumalla Eloquent Model -luokan destroy-metodia ja asettamalla sille parametriksi pääavaimen arvon. Destroy-metodi voi ottaa parametriksi myös useamman pääavaimen arvon, jolloin se poistaa useampia tietueita samalla kertaa. Seuraavassa koodiesimerkissä havainnollistetaan molempia tapoja:

```php
<?php

use App\Kindergarten;

/**
 * First method for deleting kindergarten with primary key 1
 */
$kindergarten = Kindergarten::find(1);
$kindergarten->delete();

/**
 * Second method for deleting kindergarten with primary key 1
 */
Kindergarten::destroy(1);

/**
 * Second method for deleting kindergartens with primary keys 1 and 2
 */
Kindergarten::destroy(1, 2);
```

Kuva 43. Koodiesimerkki tietokannan tietueen tai tietueiden poistamisesta

4.4.5 DB Query Builder vaihtoehtona Eloquentille

Kun on kyse hyvin monimutkaisista kyselyistä, kannattaa käyttää Laravelin DB Query Builderia. DB Query Builder ei kuitenkaan kuulu Eloquentiin, jolloin sen avulla ei pääse hyödyntämään Eloquent Model-luokassa määritelyjä suhteita. Sen takia kannattaa käyttää Eloquentia niin pitkälle kuin on mahdollista. DB Queryllä on kuitenkin joitakin etuja Eloquentiin verrattuna. Se tarjoaa esimerkiksi enemmän vaihtoehtoja kyselyn muokkaamiselle ja lisärajoitteille, ja muistuttaa syntaksiltaan enemmän SQL:ää.

DB Query Builderin metodeihin pääsi käsiksi määrittämällä tiedoston alussa use-määrittelyllä sen omaavan Facaden. Facade on nimitys Laravelin staattisille rajapinnoille, ja kaikki Facadet sijaitsevat Illuminate\Support\Facades-nimiavaruudessa. DB Query Builderin Facade on nimeltään DB, jolloin sen use-määrittely on Illuminate\Support\Facades\DB.

DB Query Builderin metodit eroavat Eloquentista hieman, mutta suurin ero DB Query Builderissa on se, että alussa pitää table-metodin avulla määrittää aina halutun taulun nimi tietokannassa, jonka jälkeen vasta voidaan lisätä rajoitteita. Kun haetaan tietoa tietokannasta, pitää muistaa kutsua get-metodia lopuksi. Seuraavassa esimerkkikoodissa

havainnollistetaan tiedon hakemista edellä mainitusta kindergartens-tietokantataulusta DB Queryn avulla:

```php
<?php

use Illuminate\Support\Facades\DB;

/**
 * Get all kindergartens from kindergarten-table
 */
$kindergartens = DB::table('kindergartens')->get();

/**
 * Get kindergartens with wanted name from kindergarten-table
 */
$wantedKindergartens = DB::table('kindergartens')
        ->where('name', 'Päiväkoti Viskurikuja')->get();
```

Kuva 44. Koodiesimerkki datan hakemisesta DB Queryn avulla

Kuvan 44 koodissa määriteltiin aluksi käytettäväksi DB Queryn DB-Facadea. Sen jälkeen haettiin kaikki kindergartens-taulun tietueet eli päiväkodit ja tallennettiin ne kindergartens-muuttujaan. WantedKindergartens-muuttujaan tallennettiin vain ne päiväkodit, joiden nimi oli Päiväkoti Viskurikuja.

Uuden tietueen luomisessa pitää kutsua insert-metodia, joka ottaa parametrikseen attribuutti-arvo pareja, ja sallii myös useamman tietueen luomisen samalla kerralla. Tietueen muokkaamisessa käytetään update-metodia, joka ottaa parametriksi muokattavat attribuutit, ja sille asetettavat arvot pareina. Poistamisessa käytetään delete-metodia. Seuraavassa koodissa havainnollistetaan uuden tietueen luomista, muokkaamista ja poistamista jo edellä mainittua groups-tietokantataulua käyttäen DB Query:n avulla:

```php
<?php

use Illuminate\Support\Facades\DB;

/**
 * Insert multiple new records into the groups-database table
 */
DB::table('groups')->insert([
        ['name' => 'Vehnä', 'max_age' => 4],
        ['name' => 'Ruis', 'max_age' => 5]
]);

/**
 * Update the name of the group which has id as 1
 */
DB::table('groups')->where('id', 1)->update(['name' => 'Ohra']);

/**
 * Destroy a group which has id as 1
 */
DB::table('groups')->where('id', 1)->delete();
```

Kuva 45. Koodiesimerkki tietueiden luomisesta, muokkaamisesta ja poistamisesta DB Queryn avulla

Kuvan 45 koodissa piti kuvan 44 koodin mukaisesti määritellä DB-Facade käytettäväksi. Alussa luotiin kaksi uutta tietuetta eli ryhmää groups-tietokantatauluun insert-metodin ja haluttujen nimien sekä maksimi-iän avulla. Sen jälkeen muokattiin id-arvolla yksi vastaavan ryhmän nimeä update-metodin avulla. Lopussa poistettiin kyseinen ryhmä delete-metodilla.

4.4.6 Transaktiot

Kun muokataan useampaa erilaista tietokannan tietuetta samaan aikaan, voi tapahtua virheitä, jossa jokin tietueista ei tallennu. Esimerkiksi käyttäjä haluaa maksaa laskun. On olemassa käyttäjän tiliä vastaava tietue sekä laskua vastaava tietue. Käyttäjä maksaa laskun, ja rahat vähennetään käyttäjän tiliä vastaavasta tietueesta, jolloin lasku pitäisi merkata maksetuksi. Voi kuitenkin olla, että ohjelma kaatuu juuri ennen kuin lasku on merkattu maksetuksi, jolloin lasku on edelleen tietokannan mukaan maksamatta. Käyttäjän tililtä on kuitenkin jo vähennetty rahaa, sillä ohjelma toimi silloin vielä normaalisti. Tämän tyyppisiä tilanteita varten on olemassa transaktioita, jotka toimivat kaikki tai ei mitään -periaatteella.

Transaktiot pitävät kirjaa sen sisällä määritetyistä muutoksista, kun ne vaikuttavat useampaan tietueeseen samaan aikaan, ja muutokset suoritetaan tietokannassa yhtenä SQL-kyselynä. Jos yksikin määritetyistä muutoksista epäonnistuisi, koko SQL-kyselyä ei tietokannan puolella edes suoritettaisi, jolloin mikään tietokannan tietueista ei muuttuisi.

Laravelissa ainoastaan DB Queryn DB-Facade tarjoaa transaktiotoiminnallisuuden. Se määritellään kutsumalla transaction-metodia DB-Facade-luokan instanssista, jonka sisään tulee kaikki tietokantaa muuttavat metodit. Seuraavassa esimerkissä havainnollistetaan transaction-metodin käyttöä muuttamalla kahden eri päiväkodin nimeä samaan aikaan:

```php
<?php

use App\Kindergarten;
use Illuminate\Support\Facades\DB;

DB::transaction(function () {
    $kindergarten1 = Kindergarten::find(1);
    $kindergarten2 = Kindergarten::find(2);

    $kindergarten1->name = 'Muutettu nimi 1';
    $kindergarten2->name = 'Muutettu nimi 2';

    $kindergarten1->save();
    $kindergarten2->save();
});
```

Kuva 46. Transaktioiden käyttäminen

Kuvan 46 koodissa huomataan, että se ei poikkea paljoa kuvan 42 koodista. Ainoana erona on se, että muutettavia päiväkoteja on kaksi, ja ne on laitettu transaction-metodin sisään. Tämä tarkoittaisi sitä, että jos yksikin päiväkotien nimistä ei muuttuisi, niin ei myös toinenkaan.

Molemmissa päiväkotisovelluksissa kaikki tietokannan tietueita muuttavat metodit laitettiin transaction-metodin sisälle. Vaikka kyseessä olisikin vain yhtä tietuetta muuttava metodi, joka ei hyötyisi transaktioista millään tavalla, se asetettiin transaction-metodin sisälle. Syynä siihen on se, että transaction-metodi ei ole myöskään millään tavalla haitaksi, ja jatkokehitystä ajatellen se on hyvä olla valmiina, jos metodit muuttuvat useita tietueita käsitteleviksi tulevaisuudessa.

4.4.7 SQL-injektioilta suojaaminen

SQL-injektio on yksi yleisimmistä web-hakkerointimenetelmistä, jonka avulla hakkeri pystyy aiheuttamaan tuhoa tietokantaan tai saamaan kaikki tiedot ulos sieltä. SQL-injektiossa hakkeri injektoi eli lisää sovelluksen käyttämään SQL-kyselyyn oman jatko-osan, jolloin tietokannassa tapahtuva SQL-kysely muuttuu hakkerin haluamaksi. Pahimmassa tapauksessa jatko-osa voi olla käsky tuhota jokin taulu tietokannassa kokonaan.

Laravelin Eloquent ja DB Query Builder käyttävät PDO-parametrisointia, joka suojaa tälläisilta hyökkäyksiltä. Gilmoiren mukaan Laravel muuttaa tietokannan SQL-syntaksin aina sellaiseksi, ettei SQL-kyselyihin pysty lisäämään jatko-osia (8).

4.4.8 Eloquent Model -luokkien merkitys päiväkotisovelluksessa

Molemmat päiväkotisovellukset käyttävät yhteistä tietokantaa, jolloin niihin täytyy luoda samat Eloquent Model -luokat. Halutuissa päiväkotisovelluksissa on olemassa käyttäjiä, jotka voidaan ajatella työntekijöinä tai lasten huoltajina. Työntekijä voi kuulua vain yhteen päiväkotiin, jolloin työntekijän suhde päiväkotiin on yhden suhde yhteen. Lasten huoltajalla voi olla monta lasta, jolloin huoltajan suhde lapseen on yhden suhde moneen. Päiväkodissa on myös ryhmiä, ja ryhmän sisällä lapsia, jolloin molemmissa tapauksissa olisi kyseessä yhden suhde moneen. Jokaisella ryhmällä ja lapsella on myös määritetty omia päiviä, jotka sisältävät kaiken päivään liittyvät tiedot. Näiden tietojen perusteella verkkosovelluksille luotiin tarvittavat Eloquent Model -luokat.

4.5 Controller

Laravelin kontrollerin (5, s. 166) tehtävänä oli vastaanottaa käyttäjän selaimelta tulevat HTTP-pyynnöt, ja käsitellä ne käyttäen Eloquent Model -luokkaa halutun tiedon hakemiseen ja muuttamiseen. Lopuksi kontrolleri saattaa välittää tietoa näkymälle antamalla sille suoritusajan.

Kun kaikki tietokannan tauluja vastaavat Eloquent Model -luokat on määritelty, voidaan luoda Controller-luokat Artisan komennolla *php artisan make:controller Kontrollerinnimi*. Komento luo tyhjän Controller-pohjatiedoston app/http/Controllers-kansion alle. Artisan tarjoaa kuitenkin myös CRUD-operaatioilla varustetun Controller-pohjatiedoston, jonka

voi luoda komennolla *php artisan make:controller Kontrollerinnimi --resource*. CRUD-operaatioilla (Create, Read, Update, Delete) tarkoitetaan tietokannan datan luomista, näkemistä, muokkaamista ja poistamista. Seuraavassa esimerkissä luodaan KindergartenController-niminen Controller-luokka, joka on varustettu CRUD-operaatioden funktioilla, käyttämällä Artisan komentoa php artisan make:controller KindergartenController --resource:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class KindergartenController extends Controller {

    public function index() {
        //
    }

    public function create() {
        //
    }

    public function store(Request $request) {
        //
    }

    public function show($id) {
        //
    }

    public function edit($id) {
        //
    }

    public function update(Request $request, $id) {
        //
    }

    public function destroy($id) {
        //
    }

}
```

Kuva 47. Kontrolleri, joka on varustettu tyhjillä CRUD-operaatiometodeilla

Kuvasta 47 huomataan, että luodussa Kontrollerissa on jo valmiiksi nimetty tietyt metodit tietyillä parametreilla, mutta logiikkaosuus on vielä tyhjä.

4.5.1 CRUD: Create

CRUD-operaation Create-osuutta vastaavat metodit ovat create, ja store. create-metodin tehtävänä on näyttää käyttäjälle uuden tietueen luomisnäkymä, esimerkiksi uuden päiväkodin luomista vastaava lomakenäkymä. Kyseisen näkymään tallennetut tiedot siirtyvät request-muuttujan avulla store-metodiin, jossa suoritetaan uuden tietueen luominen tietokantaan annetuilla tiedoilla. Seuraavassa esimerkissä oletetaan, että uuden päiväkodin luomisnäkymä sijaitsee näkymille tarkoitetussa paikassa resource-hakemiston view-kansion alla (ks. kuva 25) nimellä create-kindergarten.blade.php. Näkymistä selitetään enemmän seuraavassa osiossa. Oletetaan myös, että näkymän HTML-dokumentissa on input-tägi eli kenttä, jonka name-attribuutti on kindergartenName. Halutaan luoda uusi päiväkoti tietokantaan, jolloin create- ja store-metodit näyttäisivät kontrollerissa seuraavalta:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Kindergarten;

class KindergartenController extends Controller {

    public function create() {
        return view('create-kindergarten');
    }

    public function store(Request $request) {
        $kindergarten = new Kindergarten;
        $kindergarten->name = $request->kindergartenName;
        $kindergarten->save();
    }

}
```

Kuva 48. Koodiesimerkki tietueen luomiseen liittyvistä kontrollerin metodeista

Kuvan 48 koodissa huomataan, että create-metodi palauttaa view-metodin avulla käyttäjälle päiväkodin luomisnäkymän. view-metodi ottaa parametriksi näkymän nimen merkkijonona ilman tiedostopäätteitä, eli .blade.php-osuus jätetään pois. Näkymän pitää myös olla resource/view-kansion alla, jos siihen halutaan viitata suoraan, kuten tässä kooditapauksessa tehtiin. Syvempään viittaukseen hakemistorakenteessa käytetään merkkijonossa pistettä, esimerkiksi jos create-kindergarten-näkymä olisikin vielä kindergarten-kansion alla, merkkijonoparametri olisi kindergarten.create-kindergarten.

Kun käyttäjä on täyttänyt luomisnäkymässä input-tägin, jonka name-attribuutti on kindergartenName, ja lähettänyt pyynnön tietueen luomiseksi, päästään store-metodin sisälle. store-metodi ottaa parametriksi request-olion, jonka sisällä on käyttäjän täyttämä data, joka on jäsennelty input-tägin name-attribuutin mukaisesti. Tässä tapauksessa se olisi kindergartenName, jolloin käyttäjän syöttämään dataan päästään käsiksi kutsumalla sitä request-oliosta. Muulla tavalla store-metodin sisältö ei eroa kuvan 41 koodista. Täytyy kuitenkin muistaa lisätä use-määritys Kindergarten Eloquent -luokkaan, sillä sitä käytetään, kuten kuvan 41 koodissa tehtiin.

4.5.2 CRUD: Read

CRUD-operaation Read-osuutta vastaavat metodit ovat index ja show. Metodien tehtävänä on hakea tietokannasta tarvittavat tiedot, ja palauttaa sopiva näkymä kyseisillä tiedoilla varustettuina, jotta ne voidaan näyttää käyttäjälle. index-metodi hakee perussivuun tarvittavat tiedot, jotka voisivat olla esimerkiksi kaikki päiväkodit. show-metodi ottaa parametriksi id-pääavaimen arvon, sillä sen avulla haetaan vain yhteen tietueeseen liittyvää tietoa, esimerkiksi vain jokin tietty päiväkoti. Seuraavassa koodissa havainnollistetaan index- ja show-metodeja. Oletetaan, että näkymä kindergartens-index.blade.php vastaa kaikkia päiväkoteja sisältävää perussivua, ja näkymä certain-kindergarten.blade.php vastaa yhdelle päiväkodille tarkoitettua sivua. Silloin koodi näyttäisi tältä:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Kindergarten;

class KindergartenController extends Controller {

    public function index() {
        $kindergartens = Kindergarten::all();

        return view('kindergartens-index', [
            'kindergartens' => $kindergartens,
        ]);
    }

    public function show($id) {
        $kindergarten = Kindergarten::find($id);

        return view('certain-kindergarten', [
            'kindergarten' => $kindergarten,
        ]);
    }

}
```

Kuva 49. Koodiesimerkki tiedon näyttämiseen liittyvistä kontrollerin metodeista

Kuvan 49 koodin index-metodissa haettiin kaikki tietokannan päiväkodit ja asetettiin ne kindergarten-muuttujaan. Sen jälkeen palautettiin view-metodin avulla kindergartens-index-näkymä, johon oli lisätty toisena parametrina taulukko. Taulukko koostui näkymälle annetun muuttujan nimestä merkkijonona ja sille asetettavasta arvosta. Toisin sanoen näkymälle annettiin kindergartens-niminen muuttuja, jonka arvoksi annettiin kontrollerin kindergartens-muuttuja eli kaikki päiväkodit. Show-metodissa sen sijaan rajoituttiin vain yhteen päiväkotiin käyttämällä metodin parametrina olevaa id-muuttujaa, jonka avulla haettiin haluttu päiväkoti. Sen jälkeen palautettiin certain-kindergarten-näkymä halutun päiväkodin tiedoilla varustettuna. Hakumetodit olivat samanlaisia kuin kuvan 40 koodissa.

4.5.3 CRUD: Update

CRUD-operaation Update-osuutta vastaavat metodit ovat edit, ja update. Edit-metodi toimii samalla tavalla kuin edellä mainittu show-metodi, mutta palauttaa kuitenkin tietojen muokkaamiseen tarkoitetun näkymän tietojen näyttämisen sijaan. Siksi edit-metodi sisällytetään tähän osuuteen. Update-metodi toimii kuten create-metodi ainoana erona kuitenkin se, että uuden Model-instanssin luomisen sijaan haetaan tietokannasta muokattava tietue metodille annetun id-parametrin avulla. Seuraavassa esimerkissä oletetaan, että muokkausnäkymä tietylle päiväkodille olisi nimeltään edit-kindergarten.blade.php. Silloin koodi näyttäisi seuraavanlaiselta:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Kindergarten;

class KindergartenController extends Controller {

    public function edit($id) {
        $kindergarten = Kindergarten::find($id);

        return view('edit-kindergarten', [
            'kindergarten' => $kindergarten,
        ]);
    }

    public function update(Request $request, $id) {
        $kindergarten = Kindergarten::find($id);
        $kindergarten->name = $request->kindergartenName;
        $kindergarten->save();
    }

}
```

Kuva 50. Koodiesimerkki tietueen muokkaamiseen liittyvistä kontrollerin metodeista

Kuten huomataan, edit-metodi muistuttaa kuvan 49 show-metodia. Siinä määritellään haettava päiväkoti metodille annetun id-parametrin avulla, jonka jälkeen palautetaan haetun päiväkodin tiedoilla varustettu muokkausnäkymä käyttäjälle. Kun käyttäjä on syöttänyt uuden nimen päiväkodille input-tägiin, jonka name-attribuutti on kindergartenName, päästään käyttäjän syöttämään nimeen käsiksi request-olion avulla, kuten store-metodin tapauksessa. Sen jälkeen kutsutaan Eloquent Model -luokan tarjoamaa save-metodia, kuten kuvassa 42 on tehty.

4.5.4 CRUD: Delete

CRUD-operaation Delete-osuutta vastaa destroy-metodi. Ainoa tieto, minkä destroy-metodi tarvitsee, on poistettavan tietueen id. Sen jälkeen kutsutaan Eloquent Model -luokan tarjoamaa destroy-metodia tietueen poistamiseen tietokannassa kuvan 43 mukaisesti. Seuraavassa esimerkissä poistetaan haluttu päiväkoti tietokannasta seuraavalla koodilla:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Kindergarten;

class KindergartenController extends Controller {

    public function destroy($id) {
        Kindergarten::destroy($id);
    }

}
```

Kuva 51. Koodiesimerkki tietueen poistamiseen liittyvästä kontrollerin metodista

4.5.5 Controller-luokkien merkitys päiväkotiverkkosovelluksissa

Molemmille päiväkotisovelluksille luotiin omat Controller-luokat, sillä ne eivät toimi logiikaltaan samalla tavalla. Esimerkiksi työntekijöille tarkoitetulla sovelluksella oli enemmän oikeuksia kuin vanhemmille tarkoitetulla, koska he pystyivät muun muassa muuttamaan lapsen poissaolon lisäksi ryhmiin tai lasten päiviin liittyviä tietoja. Vanhemmat pystyivät muuttamaan ainoastaan omiin lapsiinsa liittyviä poissaoloja tietyn verran etukäteen. Controller-luokat luotiin CRUD-operaatioita sisältävinä, ja yleisenä periaatteena oli, että jokaista Eloquent Model-luokkaa vastasi oma Controller-luokka.

4.6 View

Laravelin näkymät eli viewit (5, s. 239) sijaitsevat resources-hakemiston view-kansion alla (ks. kuva 26). Niiden tehtävänä on määrittää käyttäjälle näkyvän sivun ulkoasu ja kontrollerilta saatujen muuttujien avulla tietojen esitys, eli toisin sanoen presentaatiologiikka. Näkymät sisältävät HTML-koodia, ja Laravel tarjoaa niihin helppokäyttöisen

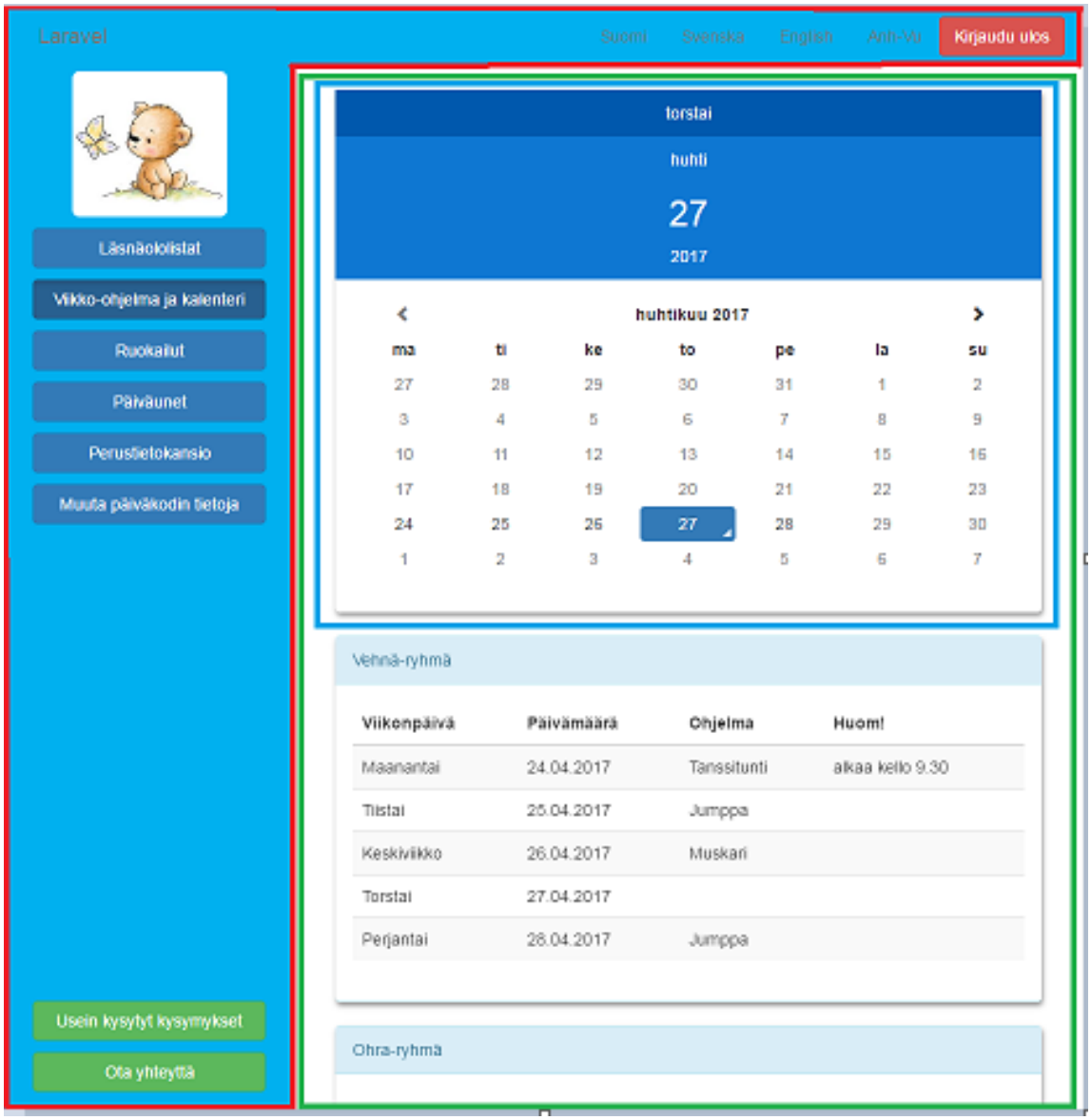
Blade-syntaksin. Näkymien luonti pitää tehdä manuaalisesti resources/views-hakemiston alle, sillä Artisan ei tarjoa komentoa siihen tarkoitukseen. Syynä siihen on se, että näkymissä ei ole tarvetta pohjakoodille. Kontrollereissa palautettavat näkymät on oltava luotu, ennen kuin niitä pystytään käyttämään.

4.6.1 Blade Templates

Blade-syntaksin (5, s. 245) tarkoituksena on korvata monimutkaiselta näyttävä PHP-koodiosuus helposti ymmärrettävän syntaksin taakse, jonka Laravel itse kääntää takaisin puhtaaksi PHP-koodilla varustetuksi näkymäksi. Blade-syntaksin käyttö ei kuitenkaan estä PHP-koodin käyttöä. Blade-syntaksin käyttämiseksi näkymätiedoston pitää olla .blade.php-tiedostopäätteinen.

Näkymät kannattaa jakaa useampaan pieneen näkymäkomponenttiin, sillä Blade tarjoaa tavan yhdistellä niitä haluamalla tavalla. Samalla vältetään parhaan toimintatavan mukaisesti duplikaattikoodia, koska komponentteja voi käyttää uudelleen.

Näkymä voidaan ajatella jakautuvan layout-näkymään, section-näkymään sekä lapsinäkymiin. Layout-näkymään voidaan ajatella kuuluvan kaikki ne HTML-sivun osat, jotka ovat läsnä melkein jokaisella sivulla, kuten logo tai navigointipalkki. Section-näkymä on se osa, jonka kontrollerit palauttavat. Section-näkymä perii myös ympärilleen layout-näkymän. Samalla section-näkymä voi sisällyttää itseensä muita lapsinäkymiä. Seuraavassa esimerkissä havainnollistetaan, miten näkymän voidaan ajatella jakautuvan:



Kuva 52. Näkymä työntekijöille tarkoitetusta päiväkotisovelluksesta, joka on jaettu kolmeen komponenttiin

Tässä työssä luotiin muun muassa kuvan 52 mukainen näkymä työntekijöille tarkoitettuun päiväkotisovellukseen. Näkymän punainen osuus vastaa layout-näkymän osuutta, koska sen voidaan ajatella pysyvän aina samana. Layout-näkymän ulkopuolelle jäävä vihreä osuus vastaa section-näkymää, joka muuttuu erilaiseksi verkkosovelluksen sivusta riippuen. Sininen osuus on vain lapsinäkymä section-näkymän sisällä, joka tässä tapauksessa sisältää kalenteri-osuuden HTML-koodin.

Koska layout-näkymä on melkein kaikissa sivuissa läsnä, siinä on hyvä määrittää kaikki tarvittavat viittaukset front-end-puolen CSS- ja JavaScript-tiedostoihin, kuten Bootstrapiin ja JQueryiin, sekä omiin CSS- ja JavaScript-tiedostoihin kuvien 6 ja 8 mukaisesti.

Silloin kaikki näkymät, jotka käyttävät kyseistä layout-näkymää, saavat myös tarvittavat viittaukset käyttöönsä.

Autentikaatio-järjestelmän lisäämisen jälkeen projektin resources/views-kansion alle oli luotu layouts-kansio, johon kuului nimensä mukaisesti kaikki layout-näkymät. Sen alla oli myös valmiiksi generoitu app.blade.php-niminen layout-näkymä, joka edusti kuvan 30 ylänavigaatiopalkkia. Yksinkertaisimmillaan layout-näkymän koodi on seuraavanlainen:

```
<!-- Stored in resources/views/layouts/app.blade.php -->

<html>
    <head>
        <title>App Name</title>
    </head>
    <body>
        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

Kuva 53. Esimerkki layout-näkymän koodista

Kuvan 53 koodista huomataan, että layout-näkymä on pääasiassa HTML-dokumenttia, mutta sisältää yield-direktiivin. @-merkillä viitataan Blade-syntaksin direktiiviin, ja yield-direktiivin tehtävänä on merkata section-näkymän paikka, jonka ympärille layout-näkymä kääriytyy, kun sitä käytetään. @yield('content') on siis Blade-syntaksiosuutta. Yield-direktiivi ottaa parametrikseen section-näkymän section-direktiivin parametrin nimen, joka on tässä tapauksessa ja seuraavassa koodissa content:

```
<!-- Stored in resources/views/section-content.blade.php -->

@extends('layouts.app')

@section('content')
    @include('calendar');
    <p>This is my body content.</p>
@endsection
```

Kuva 54. Esimerkki section-näkymän koodista

Layout-näkymän käyttämistä varten section-näkymän on sisällytettävä tieto siitä, mitä layout-näkymää halutaan käyttää. Se hoidetaan extends-direktiivillä, joka ottaa parametrina layout-näkymän sijainnin samalla tavalla kuin kuvan 49 view-metodi tekee. Section-

direktiivillä määritellään näkymän section-osuuden nimitys ja alkukohta, ja endsection-direktiivillä kerrotaan section-osuuden lopetuskohta. Kuvan 54 koodissa section-direktiivin parametri vastaa kuvan 53 yield-direktiivin parametria, jolloin ne ovat linkittyneinä toisiinsa. Kuvan 53 layout vaatii section-osuuden, jonka nimi on content, ja jonka kuvan 54 näkymä tarjoaa. Include-direktiivillä voidaan lisätä muita lapsinäkymiä osaksi omaa näkymää, ja se ottaa parametriksi extends-direktiivin tavoin halutun näkymän sijainnin. Kuvan 54 tapauksessa kalenterin lapsinäkymä sijaitsee resources/views/calendar.blade.php-tiedostossa, ja se lisätään osaksi section-näkymää include-direktiivin avulla.

4.6.2 Datan tulostaminen Blade-syntaksin avulla

Blade-varustetuissa näkymissä voi näyttää dataa käyttämällä kahta sisäkkäistä aaltosulkua, joiden sisään näytettävä data tulee. Kahdet aaltosulut kutsuvat niiden sisään tulevalla datalla PHP:n htmlentities-funktiota, joka estää XSS-hyökkäykset.

XSS eli Cross-Site Scripting on Gilmoiren mukaan tapa, jossa hakkeri kirjoittaa pahaa JavaScript-koodia johonkin lomakekenttään ja tallentaa sen tietokantaan (8). Kun tietokannasta haetaan hakkerin tallentama koodi ja tuodaan kontrollerin avulla näkymään selaimen suoritettavaksi, selain ajaa hakkerin JavaScript-koodin läpi, sillä se on upotettu kuvan 9 mukaisesti osaksi näkymää. PHP:n htmlentities-funktio estää XSS-hyökkäykset muuttamalla datan tulostuksen yhteydessä kaiken datan merkkijonoksi, jolloin selain ei tulkitsisi tulostettavaa dataa koodina ja suorittaisi sitä.

Havainnollistetaan datan tulostamista ottamalla käyttöön esimerkiksi kuvan 49 show-metodi, jossa lähetetään halutun päiväkodin tietoja sisältävä kindergarten-muuttuja certain-kindergarten.blade.php-nimiseen näkymään. Silloin kyseinen näkymä pystyy näyttämään esimerkiksi päiväkodin nimen seuraavalla koodilla:

```
Päiväkodin nimi on: {{ $kindergarten->name }}
```

Kuva 55. Esimerkki näkymän koodista, jossa tulostetaan sille annetun muuttujan avulla tietoa

Joissakin tapauksissa voi olla, että näkymälle annettu muuttuja ei ole olemassa, tai se on tyhjä. Siihen tarkoitukseen Laravel tarjoaa lyhyen or-syntaksin kaksoisaaltosulkeiden sisään, jonka oikealle puolelle määritetään mitä tulostetaan, jos kyseiset ehdot täyttyvät. Silloin kuvan 55 esimerkki näyttäisi seuraavalta:

```
Päiväkodin nimi on: {{ $kindergarten->name or 'Ei päiväkotia'}}
```

Kuva 56. Esimerkki näkymän koodista, jossa tulostetaan *Ei päiväkotia*, jos haluttua muuttujaa ei ole olemassa, tai se on tyhjä.

4.6.3 Näkymän rakenteen kontrolloiminen Blade-syntaksin avulla

Muihin PHP-koodia muistuttaviin Blade-direktiiveihin kuuluu muun muassa for-, foreach-, while- ja if-direktiivit. Niitä käytetään PHP-syntaksin tavoin näkymän rakenteen muodostamisessa, mutta samalla ne tarjoavat yksinkertaisen syntaksinsa lisäksi loop-muuttujan toistorakenteissa. Loop-muuttuja sisältää tietoa toistorakenteesta, esimerkiksi monesiteraatio on toistorakenteessa meneillään, onko iteraatio ensimmäinen tai viimeinen ja niin edelleen. Täytyy kuitenkin muistaa lopettaa kyseiset direktiivit end-alkuisella direktiivillä, jotta Blade tietää, missä direktiivin lopetuskohta on. Seuraavassa esimerkkikoodissa havainnollistetaan näkymän rakenteeseen liittyvien direktiivien käyttöä:

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif

@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@while (true)
    <p>
        I'm looping forever and this
        is the {{$loop->iteration}} iteration
    </p>
@endwhile
```

Kuva 57. Esimerkki Blade-rakennedirektiivien käytöstä

Kuvassa 57 huomataan, että Blade-rakennedirektiivit rakentuvat tulostettavan sisällön ympärille, ja sisältö tulostetaan direktiivien ehtojen mukaisesti. Sisältö voi olla HTML-

koodia, merkkijonoa tai niiden yhdistelmää. Rakenteeltaan Blade-direktiivit toimivat samalla tavalla kuin PHP-koodi. Viimeisessä p-tägissä on kuitenkin käytetty Blade-direktiivin tarjoamaa loop-muuttujaa, joka sisältää tietoa sen ympärillä olevasta toistorakenteesta. Kaikki loop-muuttujan tiedot voi katsoa Laravelin dokumentaatiosta.

4.6.4 Näkymien merkitys päiväkotisovelluksissa

Molempiin päiväkotisovelluksiin luotiin omat näkymät, ja ne pyrittiin pilkkomaan mahdollisimman pieniin komponentteihin, sillä joitakin komponentteja esiintyi useammassa sivussa, ja niitä voitiin tällä tavalla käyttää uudelleen helposti. Samalla pyrittiin käyttämään front-endin eli selainpuolen Bootstrap-sovelluskehystä mahdollisimman paljon näkymien luonnin nopeuttamiseksi.

4.7 Routing

Routing (5, s. 145) pitää tehdä ennen kuin päästään kokeilemaan mitään verkkosovelluksen toiminnallisuuksia. Sen tehtävänä on luoda ja yhdistää URL-osoitteet eli polut kontrollereissa määritettyihin metodeihin (ks. kuva 47). Kaikki polut määritetään routes-hakemiston alle (ks. kuva 26). Huomataan, että siellä on jo api.php, console.php, ja web.php-tiedosto valmiina. Api.php-tiedostoa käytetään API-sovellusten tekoon. Console.php-tiedostoa käytetään konsoli-ikkunan kautta määritettyihin polkuihin. Web.php-tiedostossa määritellään polut verkkosovelluksen käyttöliittymälle. Artisan tarjoaa komennon *php artisan route:list*, joka listaa kaikki määritellyt polut ja niihin liittyvät tiedot.

Tässä insinöörityössä tehtiin itsenäisiä verkkosovelluksia, jotka sisälsivät omat käyttöliittymät, jolloin käytettiin web.php-tiedostoa polkujen määrittelemiseksi. Kaikki sen sisällä määritellyt polut on liitetty web-middlewareen, joka hoitaa istunnon tilat ja CSRF-suojauksen. Middlewaresta kerrotaan lisää myöhemmin.

4.7.1 HTTP-verbit ja niihin liittyvät metodit

Ennen kuin voidaan määrittää omia polkuja, on ymmärrettävä HTTP-verbien merkitys. HTTP-verbit kertovat palvelimelle, mitä halutun sivun datalle tai datalla pitäisi tehdä. Niistä yleisimmät ovat POST-, GET-, PUT-, PATCH- ja DELETE-verbit, jotka vastaavat CRUD-operaatiota itsessään.

GET-verbiä käytetään, kun halutaan hakea ja näyttää dataa, jolloin CRUD-operaatioissa se vastaisi Read-osuutta eli kontrollerin index- ja show-metodia (ks. kuva 49). Sen lisäksi GET-verbiä käytetään myös create- ja edit-metodeissa (ks. kuvat 48 ja 50), koska niissä haetaan myös dataa, vaikka ne eivät käyttötarkoitustensa takia Read-osuuteen kuuluisikaan.

POST-, PUT-, PATCH- ja DELETE-verbit ovat sen sijaan tarkoitettu tietokannan muokkaamismetodeihin. POST-verbiä käytetään, kun halutaan luoda uusi tietue tietokantaan, ja vastaa CRUD-operaatioissa Create-osuuden store-metodia (ks. kuva 48). PUT- ja PATCH-verbien avulla muokataan jo olemassa olevaa tietuetta, ja vastaa Update-osuuden update-metodia (ks. kuva 50). DELETE-verbi hoitaa tietueen poiston ja vastaa Delete-osuuden destroy-metodia (ks. kuva 51).

Laravel tarjoaa jokaista edellä mainittua HTTP-verbiä varten oman metodinsa:

```php
<?php

Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
```

Kuva 58. Route-metodien syntaksi polkujen rekisteröimiseen.

Kuvan 58 koodissa huomataan, että metodien nimet ovat samat kuin edellä mainitut HTTP-verbit. Kaikki kyseisen koodin metodit ottavat kuitenkin kaksi parametria vastaan. Ensimmäisen parametrin eli uri-muuttujan tilalle voidaan määritellä polun nimen merkkijonona, jotta tiedetään, mitä URL-osoitetta pitää kutsua. Toisen parametrin eli callback-muuttujan tilalle voidaan määritellä, mitä pitäisi tapahtua, kun URL-osoitetta kutsutaan, eli toisin sanoen palvelinpuolella tapahtuva logiikka. Parhaan toimintatavan mukaisesti routes-hakemiston tiedostot eivät kuitenkaan sisällä logiikkaa itsessään, vaan kutsuvat Controller-luokan metodeja, jossa logiikka sijaitsee.

Oletetaan, että verkkosovellus sijaitsee kuvan 19 mukaisesti osoitteessa http://localhost/myproject/public, ja KindergartenController-luokan CRUD-operaatioden index- ja show-metodit on luotu (ks. kuva 49). Silloin niille voidaan määrittää polut routes-hakemiston web.php-tiedostoon seuraavasti:

```php
<?php

Route::get('/', 'KindergartenController@index');
Route::get('/kindergarten/{id}', 'KindergartenController@show');
```

Kuva 59. Esimerkki GET-verbin metodien polkujen rekisteröimisestä

Kuvan 59 koodissa ensimmäisen get-metodin ensimmäiseksi parametriksi annettiin ’/’, joka lisätään verkkosivun sijainnin loppuun. Tässä tapauksessa osoitteesta tulisi silloin http://localhost/myproject/public/, joka viittaa edelleen kotisivuun. Toiseksi parametriksi annettiin merkkijonoviittaus KindergartenController-tiedostoon, ja @-merkillä kutsuttiin sen sisällä määritettyä metodia, joka oli tässä tapauksessa index. Toisin sanoen, kun käyttäjä menee verkkosovelluksen kotisivulle eli http://localhost/myproject/public/-osoitteeseen, KindergartenController-luokan index-metodia kutsutaan.

Toisessa get-metodissa käytetään KindergartenController-luokan show-metodia (ks. kuva 49). Tiedetään kuitenkin, että show-metodi vaatii parametriksi id-nimisen muuttujan, jonka avulla se hakee halutun päiväkodin tietokannasta. Haluttu parametri show-metodille saadaan get-metodin ensimmäisestä parametrista, eli polusta {id}-osuuden avulla. Aaltosulkeet viittaavat siihen, että kun polkua kutsutaan, sen kohdalle asetettava arvo tallennetaan aaltosulkeiden sisään määritettyyn muuttujanimeen, joka on tässä tapauksessa id. Muuttujanimi voi olla mikä vain, kunhan se havainnollistaa, minkälainen parametri on kyseessä. Toisin sanoen kun käyttäjä menee osoitteeseen http://localhost/myproject/public/kindergarten/1, KindergartenController-luokan show-metodia kutsutaan parametrilla yksi, sillä se on {id}-osuuden kohdalla.

Muut edellämainitut HTTP-verbit merkataan samalla tavalla kuin kuvan 59 koodissa, mutta Laravel pystyy kiteyttämään kaikki Controller-luokan CRUD-operaation metodit yhdelle riville. Seuraavassa esimerkissä oletetaan, että kaikki KindergartenController-luokan CRUD-operaation metodit on määritelty ja luotu, jolloin voidaan kiteyttää metodien polkujen rekisteröiminen seuraavaan lauseeseen:

```php
<?php

Route::resource('kindergartens', 'KindergartenController');
```

Kuva 60. CRUD-operaatioihin liittyvien metodien polkujen rekisteröiminen yhdellä lauseella

Kuvan 60 koodissa huomataan, että resource-metodi ottaa ensimmäiseksi parametriksi kindergartens-merkkijonon, johon pohjautuen kaikki polut luodaan. Toiseksi parametriksi vaaditaan CRUD-operaatioita sisältävän Controller-luokan nimi, joka on tässä tapauksessa KindergartenController. Resource-metodista kuitenkin huomataan, että se ei kerro mitään käytettävistä HTTP-verbeistä ja niihin liitettävistä Controller-luokan metodeista, tai polkujen nimityksistä, sillä ne kaikki on piilotettu yhden rivin resource-metodin taakse. Tämän takia Artisan tarjoaa komennon *php artisan route:list*, joka listaa kaikki verkkosovelluksen polkumäärittelyt resource-metodin taakse piiloutuneet mukaan lukien. Kun kutsutaan kyseistä komentoa kuvan 60 tapauksessa, saadaan seuraava lista:

```
Method    | URI                                | Name                | Action
----------+------------------------------------+---------------------+-------------------------------------------------------
GET|HEAD  | kindergartens                      | kindergartens.index   | App\Http\Controllers\KindergartenController@index
POST      | kindergartens                      | kindergartens.store   | App\Http\Controllers\KindergartenController@store
GET|HEAD  | kindergartens/create               | kindergartens.create  | App\Http\Controllers\KindergartenController@create
GET|HEAD  | kindergartens/{kindergarten}       | kindergartens.show    | App\Http\Controllers\KindergartenController@show
PUT|PATCH | kindergartens/{kindergarten}       | kindergartens.update  | App\Http\Controllers\KindergartenController@update
DELETE    | kindergartens/{kindergarten}       | kindergartens.destroy | App\Http\Controllers\KindergartenController@destroy
GET|HEAD  | kindergartens/{kindergarten}/edit  | kindergartens.edit    | App\Http\Controllers\KindergartenController@edit
```

Kuva 61. Artisanin tulostama lista resource-metodin polkumäärittelyistä

Kuten kuvassa 61 huomataan, kaikki KindergartenController-luokan metodit on määritelty johonkin polkuun. Metodi-sarakkeessa kerrotaan käytetyistä HTTP-verbistä, ja URI-sarakkeessa mitä http://localhost/myproject/public/ -osoitteeseen pitää lisätä, jotta Action-sarakkeen metodia kutsutaan.

4.7.2 HTML-formien yhdistäminen

HTML:n form-tägit hoitavat yleensä kaikki front-endin puolella syötetyn datan lähettämisen back-endiin. Form-tägin method-attribuutissa määritellään HTTP-verbi, jota halutaan käyttää (vrt. kuvan 61 Method-saraketta), ja action-attribuutissa siihen liittyvän metodin polku (vrt. kuvan 61 URI-saraketta). Edellä mainituista HTTP-verbeistä form-tägin method-attribuutti hyväksyy arvoksi vain GET- tai POST-verbin, jolloin PUT-, PATCH-, ja DELETE-verbien määrittelemiseksi on käytettävä toista keinoa, jota kutsutaan spoofaamiseksi.

Spoofaamisessa selainta huijataan luulemaan, että PUT-, PATCH-, ja DELETE-verbit ovat hyväksyttyjä form-tägin method-attribuutin arvoja. Se tehdään lisäämällä Blade-syn-

taksin avulla method_field('haluttu HTTP-verbi') form-tägien sisään. Seuraavassa koodissa havainnollistetaan form-tägiä, jossa halutaan päivittää id-arvoa yksi vastaavan päiväkodin tietoa:

```
<form method="POST" action="/kindergartens/1">
    ...
    {{ method_field('PUT') }}
    {{ csrf_field() }}
</form>
```

Kuva 62. Esimerkki form-tägistä, jonka avulla voi muuttaa tietyn päiväkodin tietoa

Tietyn päiväkodin tietojen päivittämiseen tarvitaan KindergartenController-luokan update-metodia, jonka polku on kindergartens/{kindergarten} (ks. kuva 61). Aaltosulkujen sisään tuleva osuus polussa ilmaisi parametria. Tällöin kuvan 62 action-attribuutin arvoksi tuleekin /kindergartens/1. Huomataan myös, että update-metodiin linkitettävä HTTP-verbi on PUT (ks. kuva 61), jolloin selainta on spoofattava Blade-syntaksin aaltosulkujen välissä olevan method_field('PUT') avulla. Method-attribuutin arvoksi asetetaan POST, koska se on HTML-hyväksytty arvo, jota spoofaamisessa yritetään korvata. Kun formi eli lomake lähetetään, selain lähettää siellä syötetyt tiedot backendiin PUT-pyynnön /kindergartens/1 -polkuun, eli KindergartenController-luokan update-metodiin (ks. kuva 61). Turvallisuuden kannalta on vielä huomattava, että Laravel ei hyväksy POST-, PUT- ja DELETE-pyyntöjä, ellei form-tägin sisälle ole vielä määritelty CSRF-tokenia, kuten kuvan 62 koodissa on tehty lisäämällä Blade-syntaksin aaltosulkujen väliin csrf_field(). CSRF-tokenia käytetään CSRF-suojaukseen.

4.7.3 CSRF-suojaus

CSRF:llä eli Cross-Site Request Forgerylla tarkoitetaan Gilmoiren mukaan hyökkäystä, jossa käytetään hyväksi selaimeen tallennettua istuntoa tai evästettä (8). Kun käyttäjä kirjautuu sisään johonkin verkkosovellukseen, käyttäjän selaimelle annetaan yleensä jokin eväste, joka sisältää enkryptatun istuntoon liittyvän ID:n. Kyseistä ID:tä käyttämällä palvelin yhdistää käyttäjän käynnissä olevaan istuntoon, jolloin hänen ei tarvitse kirjautua sisään uudelleen joka kerta, kun navigoidaan verkkosovelluksessa sivulta toiselle tai kun lähetetään HTTP-pyyntöjä kyseisessä verkkosovelluksessa muun muassa datan tallentamiseen. Ongelmana tässä on kuitenkin se, että eväste tallennetaan selainkohtaisesti,

jolloin kaikki muut kyseisellä selaimella avatut haitalliset sivut voivat käyttää sitä hyväksi. Palvelin ei osaa erottaa, mistä sivusta HTTP-pyyntö tehdään, vaan sille riittää, että selainkohtainen eväste on olemassa, jolloin käyttäjä lasketaan sisäänkirjautuneeksi eli autentikoiduksi. Silloin toinen haitallinen sivu on oikeutettu tekemään evästeen avulla muutoksia verkkosovelluksessa käyttäjän tietämättä.

Esimerkiksi käyttäjä kirjautuu sisään Facebookiin. Oletetaan, että Facebookissa ei ole CSRF-suojausta. Selaimeen tallennetaan istuntoon liittyvä enkryptattu ID, ja palvelimelle sitä vastaava istunto. Uusi haitallinen sivu avataan samalla selaimella, kun käyttäjä on vielä sisäänkirjautuneena Facebookiin, vaikka viruksen takia. Haitallinen sivu haluaa vaihtaa salasanaa ja tietää salasananvaihtopolun, jota se kutsuu. Palvelin yhdistää salasananvaihdon HTTP-pyynnön yhteydessä palvelimessa olevaan istuntoon selaimen evästeen avulla, joka viittaa alussa olevaan Facebook-istuntoon. Palvelin huomaa, että istunto on olemassa, ja sallii salasanan vaihdon. Käyttäjä ei tiedä asiasta mitään.

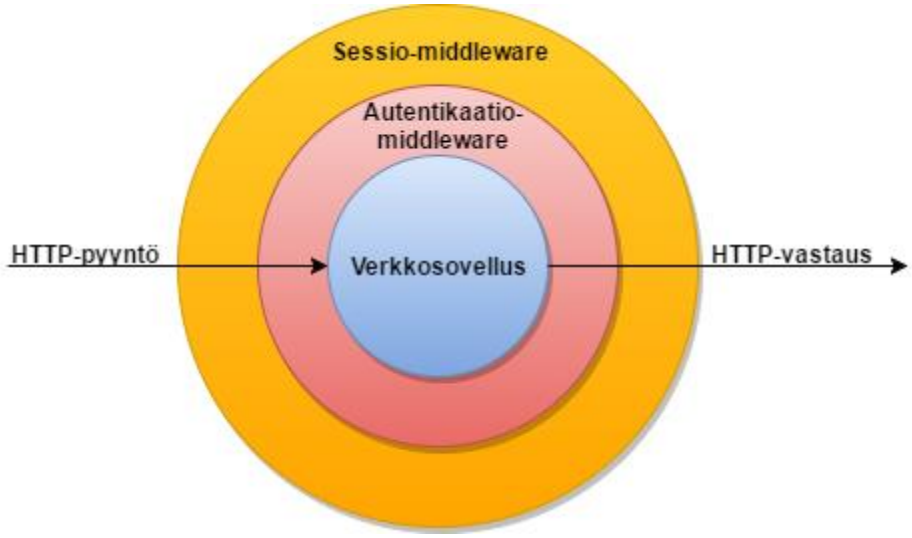
Laravel tarjoaa CSRF-suojauksen tällaisia tilanteita varten CSRF-tokenilla. CSRF-token on suurikokoinen sattumanmukaisesti luotu merkkijono, jonka palvelin lähettää sivulle, kun käyttäjä on kirjautunut sisään. CSRF-tokenin arvo luodaan aina uudelleen, kun uusi sivu kutsuu sitä, jolloin se ei pysy koskaan samana eri sivuissa tai selaimen eri ikkunoissa. Tämä tarkoittaa sitä, että palvelin pystyy CSRF-tokenin avulla erottelemaan, mistä sivusta HTTP-pyyntö tehdään ja onko sen tehnyt oikea sivusto vai ei. Laravel vaatii jokaisessa tietokantaa muuttavissa pyynnöissä eli POST-, PUT- ja DELETE-pyynnöissä CSRF-tokenin mukaan form-tägin sisään kuvan 62 mukaisella tavalla, CSRF-suojauksen takia. CSRF-tokenin tarkistamisen hoitaa VerifyCsrfToken Middleware-luokka.

4.8 Middleware

Middleware-luokkien (5, s. 155) avulla pystytään suodattamaan tai muuttamaan HTTP-pyyntöä ennen kuin saapuvat itse verkkosovellukseen. HTTP-pyynnön pitää siis läpäistä kaikki sovelluksessa määritellyt Middleware-luokat, ennen kuin verkkosovellus ottaa sen vastaan. Kyseisiä Middleware-luokkia kutsutaan Before Middleware -luokiksi, koska ne käsittelevät HTTP-pyynnön ennen verkkosovellusta.

Middleware-luokkien avulla voi myös käsitellä sovelluksen lähettämää HTTP-vastausta jälkikäteen juuri ennen kuin se saapuu käyttäjälle, ja niitä kutsutaan After Middleware -

luokiksi. Middleware-luokkia voi tällöin ajatella suojamuureina, jotka ympäröivät sovellusta. Laravel hoitaa Middleware-luokkien avulla muun muassa käyttäjän autentikaation ja CSRF-suojauksen, jota havainnollistetaan seuraavan kuvan avulla:



Kuva 63. Middlewaren merkitys verkkosovelluksessa

Kuvassa 63 lähetetään HTTP-pyyntö verkkosovellukseen, esimerkiksi käyttäjäkohtaisen näkymän näyttämiseen. Ensiksi HTTP-pyyntö menee Sessio-middlewareen, joka katsoo, onko istuntoa jo olemassa. Jos kyseessä olisi POST-, PUT- tai DELETE-pyyntö, Sessio-middleware tarkistaisi myös CSRF-tokenin oikeellisuuden. Jos istuntoa ei ole olemassa tai CSRF-tokeni on väärä, HTTP-pyyntö hylätään, ja käyttäjä viedään takaisin sisäänkirjautumissivulle. Jos istunto on kuitenkin jo olemassa, HTTP-pyyntö saa mennä eteenpäin Autentikaatio-middlewareen, jossa tarkistetaan, onko käyttäjä jo kirjautunut sisään. Jos käyttäjä ei ole sitä tehnyt, HTTP-pyyntö hylätään myös, ja käyttäjä viedään takaisin sisäänkirjautumissivulle. Vasta kun kaikkien Middleware-luokkien ehdot täyttyvät, pääsee HTTP-pyyntö sovelluksen sisään käsiteltäväksi.

4.8.1 Middleware-luokan luonti

Oman Middleware-luokan voi luoda Artisan-komennolla *php artisan make:middleware Middlewarennimi*. Komento luo Middleware-luokan pohjatiedoston app/Http/Middleware-kansion alle. Middleware-luokan pohjatiedostossa on määritelty vain handle-metodi, jossa HTTP-pyyntöä tai HTTP-vastausta käsitellään riippuen siitä, onko kyseessä Before vai After Middleware. Seuraavassa esimerkissä luodaan uusi CheckRole-niminen Middleware-luokka komennolla *php artisan make:middleware CheckRole*:

```php
<?php

namespace App\Http\Middleware;

use Closure;

class CheckRole
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        return $next($request);
    }
}
```

Kuva 64. CheckRole-niminen Middleware-luokan pohjatiedosto

Kuvan 64 Middleware-luokassa huomataan, että siellä on valmiiksi määritelty handle-metodi. Handle-metodin tehtävänä on käsitellä request-olion mukana tulevaa käyttäjän lähettämää HTTP-pyyntöä, ja lähettää se eteenpäin next-muuttujan parametrina kuvan 63 seuraavana ketjussa olevalle. Huomataan kuitenkin, että CheckRole-Middlewaressa ei ole vielä määritetty ehtoa HTTP-pyynnön keskeyttämiselle. Oletetaan, että tietokannan users-taulussa on role-niminen attribuutti. Halutaan, että ainoastaan role-attribuutin arvolla yksi HTTP-pyyntö hyväksytään ja annetaan seuraavalle ketjussa olevalle. Muissa tapauksissa HTTP-pyyntöä ei hyväksyttäisi, ja käyttäjä palautettaisiin takaisin kirjautumissivulle. Silloin kuvan 64 handle-metodi muuttuisi seuraavanlaiseksi:

```php
/**
 * Handle an incoming request.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \Closure  $next
 * @return mixed
 */
public function handle($request, Closure $next)
{
    if($request->user()->role !== 1){
        return redirect('/login');
    }

    return $next($request);
}
```

Kuva 65. Middleware-luokan handle-metodi, johon on lisätty ehto

Kuvan 65 handle-metodissa tarkistetaan, jos HTTP-pyynnön lähettäneen käyttäjän role-attribuutti ei ole yksi. HTTP-pyyntöön liittyvän käyttäjän saa kutsumalla user()-metodia request-oliosta, jonka jälkeen role-attribuuttiin päästään käsiksi kutsumalla sitä saadusta user()-metodista. Jos role-attribuutti ei ole yksi, käyttäjä palautetaan takaisin kirjautumis-sivulle ja Middleware-luokan suoritus keskeytyy. Muussa tapauksessa HTTP-pyyntö jatkaa matkaansa. Tässä tapauksessa CheckRole-Middleware olisi Before Middleware, koska se käsittelee HTTP-pyyntöjä.

HTTP-vastausta käsittelevänä eli After Middlewarena CheckRole-Middlewaren handle-metodi eroaisi hieman kuvan 64 handle-metodista:

```php
/**
 * Handle an incoming request.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \Closure  $next
 * @return mixed
 */
public function handle($request, Closure $next)
{
    $response = $next($request);

    // Perform action

    return $response;
}
```

Kuva 66. After Middleware -luokan handle-metodin rakenne

4.8.2 Middleware-luokan rekisteröinti

Ennen kuin Middleware-luokkaa pystytään käyttämään, se täytyy rekisteröidä sovellukseen. Middleware-luokan rekisteröiminen tapahtuu valmiiksi määritellyssä app/Http/Kernel.php-tiedostossa. Kernel.php-tiedostossa on kolme muuttujaa, joihin voi lisätä viittauksen haluttuun Middleware-luokkaan sen mukaan, missä sitä haluaa käyttää.

Muuttujaan middleware määritetään kaikki globaalit Middleware-luokat, jotka halutaan ajettavan jokaisen HTTP-pyynnön yhteydessä pilkuilla erotettuina. Laravel on jo valmiiksi määritellyt sinne CheckForMaintenanceMode-Middlewaren, joka katsoo, onko sovellus alhaalla muun muassa huoltokatkon vuoksi:

```
/**
 * The application's global HTTP middleware stack.
 *
 * These middleware are run during every request to your application.
 *
 * @var array
 */
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
];
```

Kuva 67. Globaalin Middleware-luokan määrittäminen app/Http/Kernel.php-tiedostossa

Jos halutaan lisätä Middleware-luokka vain tiettyihin polkuihin, se täytyy lisätä viittauksena routeMiddleware-muuttujaan. Viittaukselle pitää myös määrittää oma merkkijonotunnus. Merkkijonotunnuksen avulla voidaan kutsua haluttua Middleware-luokkaa. Laravelin routeMiddleware-muuttujaan on määritelty valmiiksi yleisimmin käytetyt Middlewaret:

```
/**
 * The application's route middleware.
 *
 * These middleware may be assigned to groups or used individually.
 *
 * @var array
 */
protected $routeMiddleware = [
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
];
```

Kuva 68. Tiettyihin polkuihin määriteltävät Middleware-luokat app/Http/Kernel.php-tiedostossa

Kun tiettyjen polkujen Middleware-luokat on lisätty routeMiddleware-muuttujaan, haluttua Middleware-luokkaa pystyi käyttämään määrittämällä se halutulle polulle routes-hakemiston alla olevassa polkujenmäärittelytiedostossa (ks. kuva 26) middleware-metodin avulla. Middleware-metodi ottaa parametriksi merkkijonotunnuksen, joka oli määritelty routeMiddleware-muuttujaan Middleware-luokan viittauksen yhteydessä. Seuraavassa esimerkissä kuvan 59 polkuihin lisätään kuvassa 68 viitattu Authenticate-Middleware-luokka seuraavalla tavalla:

```
<?php

Route::get('/', 'KindergartenController@index')->middleware('auth');
Route::get('/kindergarten/{id}', 'KindergartenController@show')
        ->middleware('auth');
```

Kuva 69. Esimerkki Middleware-luokkien käyttämisestä tietyissä poluissa

Kolmantena muuttujana app/Http/Kernel.php-tiedostossa on middlewareGroups-muuttuja. Sen avulla voidaan asettaa useampia Middleware-luokkia yhden merkkijonotunnuksen alle, jolloin voidaan liittää useampi Middleware haluttuihin polkuihin kuvan 69 mukaisella tavalla. MiddlewareGroups-muuttuja on määritelty valmiiksi seuraavasti:

```
/**
 * The application's route middleware groups.
 *
 * @var array
 */
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:60,1',
        'bindings',
    ],
];
```

Kuva 70. Tiettyihin ryhmiin määriteltävät Middleware-luokat app/Http/Kernel.php-tiedostossa

Laravelissa kaikki routes/web.php-kansion alla määriteltävät polut ottavat automaattisesti käyttöönsä kaikki web-merkkijonotunnuksen sisällä viitatut Middleware-luokat, jolloin sitä ei tarvitse kutsua siellä erikseen.

4.8.3 Middleware-luokan merkitys päiväkotisovelluksissa

Molempiin päiväkotisovelluksiin käytettiin omia versioita edellämainitusta CheckRole Middleware -luokasta (ks. kuva 64). Työntekijöille tarkoitetussa päiväkotisovelluksessa CheckRole tarkisti, onko käyttäjä työntekijä vai ei. Huoltajille tarkoitetussa päiväkotisovelluksessa CheckRole sen sijaan tarkisti, onko käyttäjä vanhempi vai ei. Tavoitteena oli, että työntekijä ei voinut kirjautua sisään huoltajille tarkoitettuun sovellukseen tai toisinpäin.

Middleware-luokalla hoidettiin myös käyttäjän syöttämän datan turhien välilyöntien karsiminen, ennen kuin se saapuu sovellusten sisälle käsiteltäväksi. Syynä siihen oli se, että esimerkiksi modernin kännykän näppäimistöllä kirjoitettaessa, käyttäjälle näytetään valmiita ehdotuksia useimmin käytetyistä syötteistä. Kun käyttäjä valitsee ehdotetun syötteen, joka voisi tässä tapauksessa olla sähköpostiosoite, näppäimistö lisää ylimääräisen välilyönnin sähköpostiosoitteen loppuun. Ylimääräinen välilyönti on siinä siksi,

ettei käyttäjän tarvitse painaa sitä erikseen, jos hän haluaa jatkaa jonkin lauseen kirjoittamista. Sisäänkirjautuessa ylimääräinen välilyönti on kuitenkin ongelma, sillä se lasketaan osaksi sähköpostiosoitetta, joillei sitä karsita pois.

4.9 Autorisaatio

Autorisaatiolla tarkoitetaan oikeuksien tarkistamista, muun muassa onko HTTP-pyynnöllä oikeutta hakea tai muuttaa tietokannan tietoa. Autorisaation vai Laravelissa hoitaa monella eri tavalla, joista kuvan 69 tapa oli Middlewaren avulla. CRUD-operaatiota sisältävässä sovelluksessa suositellaan kuitenkin Laravelin tarjoamaa Policy-luokkaa.

4.9.1 Policy-luokan luonti

Policy-luokka (5, s. 294) ulkoistaa autorisaatiologiikan tietyn Eloquent Model -luokan ympärille. Jos esimerkiksi halutaan autorisoida kaikkea tietokannan päiväkotiin eli Kindergarten Eloquent Model -luokkaan liittyviä CRUD-operaatioita, jotka on määritelty KindergartenController-luokassa (ks. kuva 47), voidaan luoda KindergartenPolicy-luokka. Policy-luokka halutulle Eloquent Model -luokalle luodaan Artisan-komennolla *php artisan make:policy Policyluokannimi --model=EloquentModelluokannimi*, ja se generoituu app/Policies-kansion alle. Seuraavassa esimerkissä luodaan KindergartenPolicy-luokka Kindergarten Eloquent Model -luokan ympärille Artisan komennolla *php artisan make:policy KindergartenPolicy –model=Kindergarten*:

```php
<?php

namespace App\Policies;

use App\User;
use App\Kindergarten;
use Illuminate\Auth\Access\HandlesAuthorization;

class KindergartenPolicy
{
    use HandlesAuthorization;

    public function view(User $user, Kindergarten $kindergarten)
    {
        //
    }

    public function create(User $user)
    {
        //
    }

    public function update(User $user, Kindergarten $kindergarten)
    {
        //
    }

    public function delete(User $user, Kindergarten $kindergarten)
    {
        //
    }
}
```

Kuva 71. Policy-luokan pohjatiedoston rakenne

Kuvassa 71 huomataan, että Policy-luokalla on neljä metodia, joiden logiikkaa ei ole vielä määritelty. Metodeissa on yhteistä se, että niissä pitää palauttaa joko true tai false riippuen siitä, onko käyttäjällä tarpeeksi oikeuksia. Jos ehto täyttyy ja palautetaan true, ohjelma jatkaa suoritustaan normaalisti. Päinvastaisessa tapauksessa Policy-luokka lopettaa ohjelman suorituksen ja palauttaa käyttäjälle HTTP 403-statuskoodin. HTTP 403-statuskoodi viittaa standardinmukaisesti kiellettyyn toimintoon, ja selain näyttää sen saadessaan valkoisen sivun, jossa lukee ainoastaan HTTP Error 403 Forbidden.

Policy-luokan view-metodi vastaa CRUD-operaatioissa Read-osuutta, jolloin sen sisälle ohjelmoidaan kaikki tietojen hakuun liittyvien oikeuksien tarkistaminen eli autorisaatiologiikka. Create-metodi vastaa CRUD-operaatioissa Create-osuutta, jolloin sen sisälle ohjelmoidaan uuden tietueen luomiseen liittyvä autorisaatiologiikka. Update-metodi vastaa

CRUD-operaation Update-osuutta ja sisältää tietojen päivittämiseen liittyvän autorisaatiologiikan. Delete-metodi sisältää tietueen poistoon liittyvän autorisaatiologiikan ja vastaa CRUD-operaatioissa Delete-osuutta.

Seuraavassa esimerkissä halutaan, että vain käyttäjät, joiden role-attribuutti 3 tai suurempi, saavat poistaa päiväkodin. Koska kyseessä on tietueen poistaminen, muokataan Policy-luokan delete-metodia, joka näyttäisi lopuksi seuraavalta:

```
public function delete(User $user, Kindergarten $kindergarten) {
    if($user->role >= 3){
        return true;
    }
    else{
        return false;
    }
}
```

Kuva 72. Policy-luokan delete-metodi, jossa vain käyttäjät, joiden role-attribuutti on 3 tai suurempi, läpäisevät autorisaation

4.9.2 Policy-luokan rekisteröinti

Ennen kuin Policy-luokkaa voidaan käyttää, se täytyy rekisteröidä app/Providers/AuthServiceProvider.php-tiedostoon valmiiksi luotuun policies-muuttujan sisään. Policies-muuttuja ottaa vastaan Eloquent Model -luokan ja Policy-luokan pareina seuraavalla tavalla:

```
/**
 * The policy mappings for the application.
 *
 * @var array
 */
protected $policies = [
    App\Kindergarten::class => App\Policies\KindergartenPolicy::class,
];
```

Kuva 73. KindergartenPolicy-luokan rekisteröiminen app/Providers/AuthServiceProvider.php-tiedostoon

Kun Policy-luokka on rekisteröity, ja kaikki siinä olevien metodien autorisaatiologiikka luotu, sitä voidaan kutsua Controller-luokan metodeissa. Policy-luokkaa on kuitenkin kutsuttava metodien alkupäässä, koska silloin ei ole vielä päästy aloittamaan Controller-

luokan CRUD-operaatioita. Tämä mahdollistaa ohjelman keskeytyksen heti alussa, jos tarvittavia oikeuksia ei löydy.

Täytyy myös muistaa, mitkä Policy-luokan metodit kuuluvat mihinkin Controller-luokan metodeihin. CRUD-operaatioita tukevassa Controller-luokassa kaikki luomiseen liittyvät metodit (ks. kuva 48) kutsuvat Policy-luokan create-metodia. Kaikki tiedon hakuun liittyvät metodit (ks. kuva 49) kutsuvat Policy-luokan view-metodia. Tietueen muokkaamiseen tarkoitetut Controller-luokan metodit (ks. kuva 50) kutsuvat Policy-luokan update-metodia. Poistamiseen tarkoitetuissa metodeissa (ks. kuva 51) kutsutaan Policy-luokan delete-metodia. Policy-luokkaa kutsutaan Controller-luokassa this-muuttujan authorize-metodin avulla, joka ottaa parametrikseen halutun Policy-luokan metodin nimen merkkijonona, sekä CRUD-operaatioon liittyvän tietueen eli Modelin, jos sitä tarvitaan. Seuraavassa koodissa havainnollistetaan Controller-luokan update-metodin (ks. kuva 50) autorisaatiota Policy-luokan avulla seuraavasti, kun halutaan muokata päiväkodin tietoja:

```
public function update(Request $request, $id) {
    $kindergarten = Kindergarten::find($id);
    $this->authorize('update', $kindergarten);

    return view('edit-kindergarten', [
        'kindergarten' => $kindergarten,
    ]);
}
```

Kuva 74. Controller-luokan update-metodi, jossa on kutsuttu Policy-luokan update-metodia

Kuvan 74 update-metodissa huomataan, että ainoana erona kuvan 50 update-metodiin on Policy-luokan kutsumisen lisääminen this-muuttujan authorize-metodin avulla. Authorize-metodi kutsuu Policy-luokan update-metodia, sillä se on määritelty ensimmäiseksi parametriksi. Authorize-metodin toinen parametri ottaa vastaan halutulla id:llä etsityn päiväkodin, jota tullaan päivittämään. Jos käyttäjällä riittäisi oikeudet, ohjelma jatkaisi suoritustaan ja näyttäisi edit-kindergarten-näkymän. Muussa tapauksessa käyttäjälle palautettaisiin HTTP-403 statuskoodi, ja ohjelman suoritus keskeytyisi.

4.9.3 Policy-luokkien merkitys päiväkotisovelluksessa

Työntekijälle tarkoitetussa päiväkotisovelluksessa on erilaiset toiminnallisuudet, jolloin niihin liittyvät oikeudet ovat myös erilaisia huoltajille tarkoitettuun päiväkotisovellukseen verrattuna. Siksi myös Policy-luokkien pitää olla erilaiset. Työntekijöille tarkoitetun sovelluksen Policy-luokilla varmistettiin, ettei työntekijä pääse näkemään tai muuttamaan mihinkään muuhun päiväkotiin liittyvää dataa. Huoltajien tapauksessa Policy-luokilla varmistettiin se, että huoltaja näkee vain omiin lapsiinsa liittyvän datan, eikä kenenkään toisen.

4.10 Validoinnit

Validoinnilla (5, s. 211) tarkoitetaan käyttäjän syöttämien tietojen tarkistamista ja vastaa CRUD-operaatioissa vain Create- ja Update-osuutta, koska vain niissä käsitellään käyttäjän syöttämää dataa. Yleisenä periaatteena validoinnissa on se, että käyttäjään ei voi koskaan luottaa. Validoinnilla voidaan varmistaa, ettei käyttäjä syötä väärän tyyppistä tai arvoista dataa, ja varoittaa käyttäjää siitä, jos niin tapahtuu. Validointi voidaan hoitaa sekä frontendissa eli selainpuolella, että backendissä eli palvelinpuolella. Havainnollistetaan validointia molemmissa tapauksissa käyttäen esimerkkinä päiväkodin tietojen muokkausnäkymän form-tägiä (ks. kuva 62), jossa voidaan muuttaa päiväkodin nimeä. Näkymän form-tägi ilman validointia näyttäisi silloin tältä:

```
<form method="POST" action="/kindergartens/1">
    <input type="text" name="name" placeholder="Anna nimi...">
    <button type="submit">Tallenna</button>
    {{ method_field('PUT') }}
    {{ csrf_token }}
</form>
```

Kuva 75. Esimerkki form-tägistä, jonka avulla voi muuttaa tietyn päiväkodin nimeä

Kuvan 75 form-tägissä on input-tägi, johon käyttäjän kirjoittamat tiedot menevät. Button-tägi toimii tallennusnappina, joka lähettää tiedon HTTP-pyyntönä palvelinpuolelle. Lopussa on määritelty tietojen päivittämiseen vaadittava PUT-metodin spoofaaminen, ja CSRF-suojausta varten tarvittava csrf-tokeni.

4.10.1 Front-end validointi

Front-endin validointi perustuu selaimessa tapahtuvaan validointiin, ja tapahtuu ennen kuin HTTP-pyyntö lähetetään palvelimelle. HTML-koodi tarjoaa validointiin attribuutteja,

jotka voi lisätä muun muassa input-tägiin, mutta ne ovat toiminnaltaan hyvin rajoitettuja. Kyseiset attribuutit sisältävät validointisääntöjä, joita syötettävän datan on noudatettava. Esimerkiksi halutaan, että päiväkodin nimi-kenttä ei saa olla tyhjä, ja se saa olla enintään 6 merkkiä pitkä. Silloin input-tägiin lisätään required-attribuutti, joka ei salli tyhjän nimen lähettämistä. Enintään kymmenen merkkiä salliva validaatio hoidetaan lisäämällä max-length-niminen attributti, joka ottaa arvokseen sallitun pituuden, eli tässä tapauksessa arvon kymmenen. Silloin kuvan 75 form-tägi muuttuu seuraavaksi:

```
<form method="POST" action="/kindergartens/1">
    <input type="text" name="name" placeholder="Anna nimi..."
           required maxlength="10">
    <button type="submit">Tallenna</button>
    {{ method_field('PUT') }}
    {{ csrf_token }}
</form>
```

Kuva 76. Esimerkki form-tägistä, jossa input-tägiin on lisätty HTML-validaation attribuutteja

Front-endin validaatio ei kuitenkaan riitä, sillä se on helposti kierrettävissä esimerkiksi selaimen kehittäjätyökalun avulla. Sen pääasiallinen tarkoitus onkin karsia pois vain osa turhista HTTP-pyynnöistä ennen kuin ne lähetetään, ja on tarkoitettu vain peruskäyttäjille, joilla ei yleensä ole tietoa front-endin validaation kiertämisestä.

4.10.2 Back-end-validointi Form Request -luokkien avulla

Back-end-validoinnilla tarkastetaan kaikki palvelimeen asti tuleva käyttäjän lähettämä data. Sitä ei voi millään keinolla kiertää, sillä palvelinpuolen suoritus on käyttäjän ulottumattomissa. Sen takia back-end-validointi ajatellaan pakolliseksi validoinniksi, ja front-end-validointi vain täydentää sitä. Laravel tarjoaa back-end-validointiin monia eri tapoja, mutta parhaan käytännön mukaisesti se kannattaa ulkoistaa Form Request -luokkiin, jota on sitten helppo määrittää käytettäväksi Controller-luokan metodeissa, jossa käytetään käyttäjän syöttämää dataa. Ajatuksena on, että jokaista verkkosovelluksessa esiintyvää form-tägiä kohtaan on olemassa oma Form Request -luokkansa, joka hoitaa kyseisen form-tägiin syötetyn datan validoinnin. CRUD-operaatioita tukevassa Controller-luokassa käyttäjän syöttämään dataan pääsee käsiksi request-olion avulla, jolloin se pitää validoida.

Form Request -luokat luodaan Artisan-komennolla *php artisan make:request FormRequestluokannimi*. Haluttu Form Request -luokan pohjatiedosto luodaan app/http/Requests-kansion alle. Luodaan kuvan 75 form-tägille UpdateKindergarten-niminen Form Request -luokka komennolla *php artisan make:request UpdateKindergarten*:

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UpdateKindergarten extends FormRequest {

    public function authorize() {
        return false;
    }

    public function rules() {
        return [
                //
        ];
    }

}
```

Kuva 77. Form Request -luokan pohjatiedosto

Kuvassa 77 huomataan, että Form Request -luokassa on määritelty valmiiksi authorize- ja rules -metodi. Authorize-metodin sisälle voi lisätä omaa autorisaatiologiikkaa, joka toimii samalla tavalla kuin kuvan 72 tapauksessa. Jos autorisaatio päästää läpi, palautetaan true. Muussa tapauksessa palautetaan false, jolloin käyttäjälle annetaan HTTP-403-statuskoodi, ja ohjelman suoritus päättyy siihen. Koska autorisaatiologiikka oli jo aiemmin ulkoistettu kokonaan Policy-luokkaan, Form Request-luokan authorize-metodi voidaan laittaa palauttamaan true, sillä sitä ei käytetä.

Kun authorize-metodissa on palautettu true, ohjelman suoritus siirtyy rules-metodiin, jossa voi määritellä, mitä validaatiosääntöjä käyttäjältä saatavan datan halutaan noudattavan. Rules-metodissa pitää määritellä näkymän puolella syötetyn dataa sisältävän tägin name-attribuutin arvo sekä siihen viitatut validaatiosäännöt. Kuvan 75 tapauksessa input-tägi sisältää käyttäjän syöttämän datan, jolloin sen name-attribuutin arvo olisi name. Seuraavassa esimerkissä halutaan, että päiväkodin nimi-kenttä ei saa olla tyhjä ja on enintään 6 merkin pituinen, kuten front-end validaation esimerkissä. Sen lisäksi

voidaan määrittää, että käyttäjän syöttämä nimi pitää olla ainutlaatuinen koko tietokannan kindergarten-taulussa, mitä front-end-validaatio ei pysty tarjoamaan. Form Request -luokka näyttäisi silloin seuraavalta:

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UpdateKindergarten extends FormRequest {

    public function authorize() {
        return true;
    }

    public function rules() {
        return [
            'name' => 'required|max:10|unique:kindergarten',
        ];
    }

}
```

Kuva 78. Form Request -luokka, johon on lisätty validaatiosääntöjä

Kuvan 78 koodissa huomataan, että rules-metodi palauttaa halutun name-attribuutin arvon, johon on viitattu halutuilla validaatiosäännöillä. Validaatiosäännöt annetaan merkkijonoina ja erotetaan |-merkillä. Required-sääntö katsoo, onko käyttäjän syöttämä data tyhjä vai ei, jonka jälkeen max:10-sääntö tarkistaa, jos annettavan data on yli kymmenen merkkijonon pituinen. Lopuksi unique:kindergarten-sääntö katsoo kindergarten-taulun tietuieista, ettei kyseistä nimeä ole vielä annettu. Laravel tarjoaa ison joukon validointisääntöjä, jotka löytyvät Laravelin dokumentaatiosta.

Jos yksikin sääntö ei toteudu, Laravel palauttaa käyttäjän takaisin päiväkodin tietojen muokkausnäkymään virheviesteillä ja hylätyllä datalla varustettuna. Validaatioon liittyvät virheviestit on määritelty Laravelissa englanninkielellä valmiiksi, mutta niitä voi myös ylikirjoittaa lisäämällä messages-metodi Form Request -luokkaan. Messages-metodissa viitataan haluttuun sääntöön omalla halutulla virheviestillä, joka päätyy näkymälle näytettäväksi, jos haluttu sääntö ei toteudu. Seuraavassa esimerkissä kuvan 78 tapaukseen määritetään omat virheviestit lisäämällä seuraava koodi Form Request -luokan sisälle:

```
public function messages() {
    return [
        'name.required' => 'Päiväkodin nimi on pakollinen.',
        'name.max' => 'Päiväkodin nimi saa olla enintään 10 merkkiä pitkä.',
        'name.unique' => 'Päiväkodin nimi on jo varattu.',
    ];
}
```

Kuva 79. Form Request -luokan messages-metodi, jossa on asetettu omat validaatioon liittyvät virheviestit.

Virheviesteillä käyttäjä saa lisätietoa siitä, mikä syötetyssä datassa on väärin. Validaattorin hylkäämäksi tulleen datan palauttaminen on myös käyttäjän kannalta hyvin hyödyllinen, sillä sen avulla käyttäjän ei tarvitse kirjoittaa kaikkea enää uusiksi, vaan voi muokata tai korjata syötetyn datan virheviestien mukaan. Jotta virheviestejä ja hylättyä dataa voidaan näyttää kuvan 75 näkymässä, ne täytyy tulostaa seuraavasti:

```
<form method="POST" action="/kindergartens/1">

    @if (count($errors) > 0)
    <div>
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
    @endif

    <input type="text" name="name" placeholder="Anna nimi..."
           value="{{ old('name') }}">
    <button type="submit">Tallenna</button>
    {{ method_field('PUT') }}
    {{ csrf_token }}
</form>
```

Kuva 80. Esimerkki form-tägistä, johon on lisätty back-end-validaatioon liittyvät virheviestit

Kaikissa Blade-syntaksilla varustetuissa näkymissä on määritelty errors-muuttuja valmiiksi. Errors-muuttuja on alussa aina tyhjä, mutta täyttyy validointivihreissä niitä vastaavilla virheteksteillä. Kuvan 80 form-tägin alussa on Blade-syntaksilla tulostettu virheviestit, kun errors-muuttujan sisältö ei ole tyhjä eli lukumäärä on suurempi kuin nolla.

Kuvan 80 input-tägin value-attribuuttiin arvoksi asetettiin old-metodi, joka otti parametrikseen name-attribuutin arvon. Value-attribuutin tehtävänä on asettaa valmis arvo näky-

män syötekenttään. Old-metodin avulla päästään käsiksi validaattorin hylkäämäksi asetettuun dataan, jolloin käyttäjän ei tarvitse validaatiovirheen takia kirjoittaa kaikkea uusiksi. Alussa old-metodi tulostaisi aina tyhjää, sillä hylättyä dataa ei ole, jolloin syötekenttä olisi tyhjä. Kun käyttäjän syöttämä data hylätään validoinnissa palvelinpuolella, old-metodiin tallentuisi hylätty arvo, joka tulostuisi syötekenttään valmiiksi käyttäjän palaessa päiväkodin tietojen muokkausnäkymään validaatiovirheen takia.

Ennen kuin Form Request -luokan validaatiomekanismia pystytään käyttämään, se täytyy määritellä Controller-luokassa käytettäväksi, ja tapahtuu muuttamalla Controller-luokan request-muuttujan tyyppiä halutuksi Form Request -luokan tyypiksi. CRUD-operaatioita sisältävässä Controller-luokassa ainoastaan store- ja update-metodit ottavat parametrikseen request-muuttujan, joten vain niihin voidaan liittää Form Request -luokat. Form Request -luokan liittäminen tiettyyn metodiin tapahtuu kyseisen metodin request-parametrin tyyppiä muuttamalla. Alussa request-parametrin tyyppi on määritelty Request-luokaksi, mutta Form Request -luokan liittämiseksi request-parametrin tyyppi on korvattava Form Request -luokalla. Seuraavassa esimerkissä yhdistetään kuvan 77 UpdateKindergarten-niminen Form Request -luokka KindergartenController-luokan update-metodiin (ks. kuva 74), jolloin update-metodi muuttuisi seuraavanlaiseksi:

```
public function update(App\Http\Requests\UpdateKindergarten $request, $id) {
    $kindergarten = Kindergarten::find($id);
    $this->authorize('update', $kindergarten);

    return view('edit-kindergarten', [
        'kindergarten' => $kindergarten,
    ]);
}
```

Kuva 81. Controller-luokan update-metodi, johon on liitetty Form Request -luokan validointi ja Policy-luokan autorisaatio

Kuvassa 81 huomataan, että ainoa ero kuvaan 74 verrattuna on request-parametrin tyypin muuttuminen Request-luokasta UpdateKindergarten Form Request -luokaksi.

4.10.3 Validaation merkitys päiväkotisovelluksissa

Molemmissa päiväkotisovelluksissa käytettiin front-end- ja back-end-validointia. Front-end validaatio tehtiin jokaiselle näkymien form-tägeille erikseen, ja back-endissä luotiin niitä vastaavat Form Request -luokat. Form Request -luokkiin oli määritelty myös omat

virheviestit, sillä Laravelin tarjoamat englanninkieliset virheviestit eivät sopineet suomenkieliseen sovellukseen.

5 Yhteenveto

Insinöörityössä tutustuttiin verkkosovelluskehitykseen ja sovelluskehysten käytön hyötyihin. Nykypäivänä verkkosovelluksista on tullut välttämätön osa liiketoimintaa, jolloin niiden kysyntä on suurta. Suuren kysynnän takia verkkosovelluskehityksen on muututtava koko ajan tehokkaammaksi ja nopeammaksi. Tämän mahdollistamiseksi on kehitetty lukuisia sovelluskehyksiä, jotka komponenteillaan nopeuttavat verkkosovelluskehittäjän työtä.

Tässä työssä oli luontevaa aloittaa selittämällä verkkosovelluskehityksen vaatimista esitiedoista, ennen kuin perehdyttiin sovelluskehyksien ominaisuuksiin. Työssä perehdyttiin erityisesti palvelinpuolella käytettävän Laravel-sovelluskehyksen tarjoamiin komponentteihin.

Työn aiheeksi valittiin verkkosovelluskehitys Laravelilla, koska haluttiin tutustua palvelinpuolen suosituimpaan ja nopeasti kehittyvään PHP-kielen sovelluskehykseen, ja sen avulla kehittää kaksi turvallista päiväkodeille suunnattua verkkosovellusta. Laravel valittiin myös sen takia, koska se tarjosi valmiin turvallisen autentikaatiojärjestelmän, joka oli helposti lisättävissä sovelluksiin. Laravel-sovelluskehyksen oppiminen vaati paljon aikaa, mutta selkeän dokumentaationsa ansiosta se oli helppo omaksua.

Insinöörityön tuotoksena luotiin kaksi päiväkotisovellusta, joista yksi oli tarkoitettu päiväkodin työntekijöille, ja toinen lasten huoltajille. Verkkosovellusten tarkoituksena oli helpottaa lasten huoltajien ja päiväkodin työntekijöiden välistä kommunikaatiota, läsnäolojen ja ruokailujen merkintää, sekä näyttää huoltajille lastensa päivän kuulumiset. Sovellusten piti myös sisältää turvallinen autentikaatiojärjestelmä ja olla suojattu yleisimpiä verkkohyökkäyksiä vastaan. Työlle asetetut tavoitteet toteutuivat, ja luodut tavoitteiden mukaiset sovellukset on tarkoitus myydä kunnille päiväkodin nykyisen järjestelmän digitalisoinnin tueksi.

Lähteet

1 Annie. Why Web Apps Are Important For Businesses Today? Verkkodokumentti. <http://www.vensi.com/why-web-apps-are-important-for-businesses-today>. August 10th, 2016. Luettu 15.09.2016.

2 Tuulikki Laine. Mitä markkinoijan tulee ymmärtää web-ohjelmoinnista? Verkkodokumentti.<http://www.dagmar.fi/uutiset/mita-markkinoijan-tulee-ymmartaa-web-ohjelmoinnista> 23.09.2015. Luettu 10.12.2016.

3 Lubos Kmetko. Why you should stop supporting IE10, IE9 and IE8. Verkkodokumentti. <https://www.xfive.co/blog/stop-supporting-ie10-ie9-ie8>. February 17th, 2016. Luettu 10.12.2016.

4 Alek. [MVC] MVC Design pattern in a nutshell, in the context of a web application. Verkkodokumentti. <http://clubasp.blogspot.fi/2013/05/mvc-mvc-design-pattern-in-nutshell-in.html> May 3rd 2013. Luettu 14.12.2016.

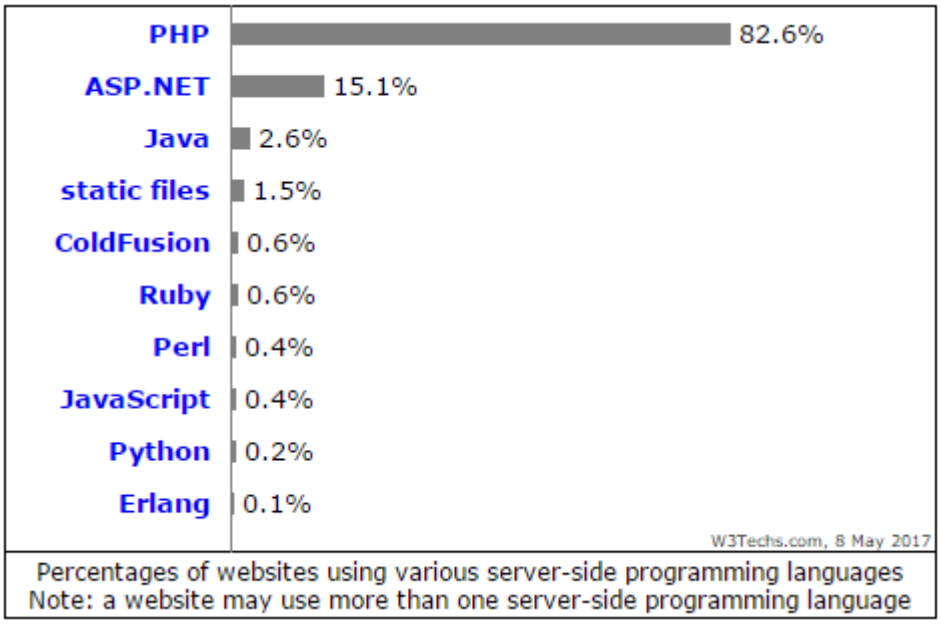
5 Gary Blankenship. Laravel 5 Official Documentation. 2017. Leanpub.

6 Brown, Philip. What is PHP Composer? Verkkodokumentti. <http://culttt.com/2013/01/07/what-is-php-composer>. January 7th 2013. Luettu 01.02.2017.

7 Stuart Yeates. What Is Version Control? Why Is It Important For Due Diligence? Verkkodokumentti. <http://oss-watch.ac.uk/resources/versioncontrol>. May 9th 2013. Luettu 02.02.2017

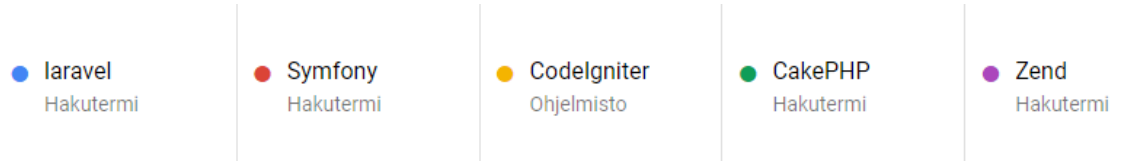
8 W. Jason Gilmoire. How Laravel 5 Prevents SQL Injection, Cross-Site Request Forgery, and Cross-Site Scripting. Verkkodokumentti. <http://www.easylaravelbook.com/blog/2015/07/22/how-laravel-5-prevents-sql-injection-cross-site-request-forgery-and-cross-site-scripting>. July 22nd, 2015. Luettu 4.02.2017.

Palvelinpuolen käytetyimmät ohjelmointikielet

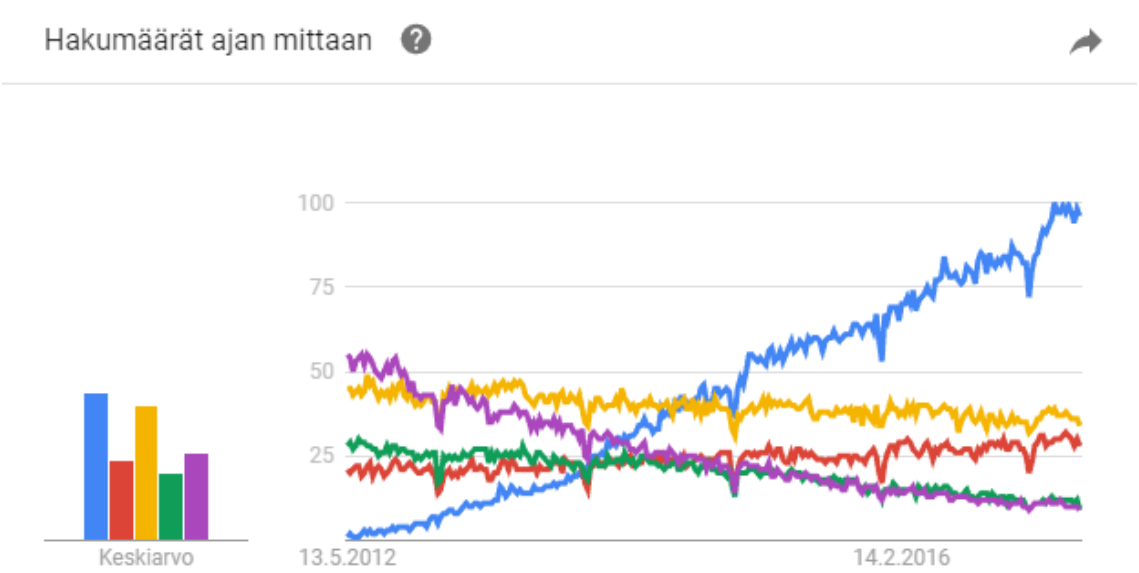


Kuva 1. W3Techsin tilastot käytetyistä palvelinpuolen ohjelmointikielistä, joita on käytetty kymmenessä miljoonassa suosituimmassa verkkosivussa

Google Trends-tilastot suosituimmista PHP-sovelluskehyksistä



Kuva 1. Google Trendsissä käytetyt hakusanat suosituimmista PHP-sovelluskehyksistä. Hakutermit vastaavat tiettyjä sanoja, kun taas aiheet vastaavat saman käsitteen sanoja millä tahansa kielellä.



Kuva 2. Google Trendin kuvaaja kuvan 1 hakusanojen hakumääristä viimeisten viiden vuoden aikana. Numerot esittävät haun suosiota valitulla ajanjaksolla ja alueella suhteutettuna kaavion suurimpaan arvoon. Asteikon arvo 100 on alue, jolla termi oli suosituin, 50 on alue, jolla termillä tehtiin hakuja puolet vähemmän kuin 100 pisteen alueella, ja 0 on alue, jolla hakujen osuus oli alle 1 % suurimmasta osuudesta.

Kuva 3. Google Trendin raportti kuvan 1 hakusanojen suosituimmasta hakusanasta viimeisten viiden vuoden ajanjaksolta maakohtaisesti.

PHP Artisan-komennot

```
Available commands:
  clear-compiled       Remove the compiled class file
  down                 Put the application into maintenance mode
  env                  Display the current framework environment
  help                 Displays help for a command
  inspire              Display an inspiring quote
  list                 Lists commands
  migrate              Run the database migrations
  optimize             Optimize the framework for better performance
  serve                Serve the application on the PHP development server
  tinker               Interact with your application
  up                   Bring the application out of maintenance mode
 app
  app:name             Set the application namespace
 auth
  auth:clear-resets    Flush expired password reset tokens
 cache
  cache:clear          Flush the application cache
  cache:table          Create a migration for the cache database table
 config
  config:cache         Create a cache file for faster configuration loading
  config:clear         Remove the configuration cache file
 db
  db:seed              Seed the database with records
 event
  event:generate       Generate the missing events and listeners based on regist
ration
 key
  key:generate         Set the application key
 make
  make:auth            Scaffold basic login and registration views and routes
  make:command         Create a new Artisan command
  make:controller      Create a new controller class
  make:event           Create a new event class
  make:job             Create a new job class
  make:listener        Create a new event listener class
  make:mail            Create a new email class
  make:middleware      Create a new middleware class
  make:migration       Create a new migration file
  make:model           Create a new Eloquent model class
  make:notification    Create a new notification class
  make:policy          Create a new policy class
  make:provider        Create a new service provider class
  make:request         Create a new form request class
  make:seeder          Create a new seeder class
  make:test            Create a new test class
 migrate
  migrate:install      Create the migration repository
  migrate:refresh      Reset and re-run all migrations
  migrate:reset        Rollback all database migrations
  migrate:rollback     Rollback the last database migration
  migrate:status       Show the status of each migration
 notifications
  notifications:table  Create a migration for the notifications table
 queue
  queue:failed         List all of the failed queue jobs
  queue:failed-table   Create a migration for the failed queue jobs database tab
le
  queue:flush          Flush all of the failed queue jobs
  queue:forget         Delete a failed queue job
  queue:listen         Listen to a given queue
  queue:restart        Restart queue worker daemons after their current job
  queue:retry          Retry a failed queue job
  queue:table          Create a migration for the queue jobs database table
  queue:work           Start processing jobs on the queue as a daemon
 route
  route:cache          Create a route cache file for faster route registration
  route:clear          Remove the route cache file
  route:list           List all registered routes
 schedule
  schedule:run         Run the scheduled commands
 session
  session:table        Create a migration for the session database table
 storage
  storage:link         Create a symbolic link from "public/storage" to "storage/
app/public"
 vendor
  vendor:publish       Publish any publishable assets from vendor packages
 view
  view:clear           Clear all compiled view files
```

Kuva 1. PHP Artisanin tarjoamat komennot