Werner Kettunen

Web-sovelluskehityksen tekniikat

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

30.11.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Werner Kettunen Web-sovelluskehityksen tekniikat 30 sivua 30.11.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Ohjaaja	Lehtori Kimmo Saurén

Web-sovelluskehitykseen käytettäviä tekniikoita, työkaluja ja ohjelmakirjastoja on olemassa useita erilaisia ja niiden lähestymistapa web-sovelluskehitykseen poikkeaa jonkin verran toisistaan. Opinnäytetyössä selvitetään teoriassa ja käytännön esimerkkiprojektin avulla yleisimmin web-sovelluskehityksessä käytettyjä tekniikoita ja kirjastoja.

Työssä esimerkkinä luodussa web-sovelluksessa käytettiin Laravel-ohjelmakehystä ja alkuosassa käsiteltyjä työkaluja ja kirjastoja, kuten Bootstrap ja Git. Työssä tehdyssä web-sovelluksessa tutustuttiin käytännön kautta tietokannan tiedon käsittelemiseen ja selaimelta tulevien pyyntöjen reitittämiseen ja näkymien näyttämiseen Laravel-ohjelmakehyksen oman sapluunamoottorin avulla. Palvelimen puolella olevat toiminnot tehtiin PHP:llä ja Laravelillä. Selaimessa käyttäjälle näkyvässä osassa käytettiin yksinkertaista HTML-merkintäkieltä, CSS-tyylityskieltä ja Bootstrap-kirjastoa.

Esimerkkinä luotua yksinkertaista verkkosivua voidaan laajentaa muun muassa Laravel-ohjelmakehyksen tarjoamien tiedostonhallintaan liittyvien kirjastojen sekä ohjelmakehyksen ulkoisten kirjastojen ja työkalujen avulla.

Avainsanat	web, ohjelmakehys, php, kirjasto

Author Title Number of Pages Date	Werner Kettunen Technologies used in web development 30 pages 30 November 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Instructor	Kimmo Saurén, Senior Lecturer

There are several technologies, tools and libraries used in web development, each of them having a slightly different approach. These generally used technologies libraries and tools are reviewed in this thesis in theory and through describing a practical example project.

The practical example project was made using the Laravel framework and with tools and libraries mentioned in the theory section. Topics that are covered in the making of the example project include database actions and routing of requests and building site views using Laravel's own templating engine. Back end features were implemented using PHP and Laravel. Front end features were implemented using basic HTML markup, CSS styling and the Bootstrap library.

Expansion of the practical example project can be done by using different libraries provided by the Laravel framework such as the filesystem library and also by using external libraries and tools.

Keywords	web, framework, php, library

Sisällys

Lyhenteet

PHP	PHP: Hypertext Preprocessor. Palvelinpuolella toimiva skriptikieli.
SQL	Structured Query Language. Tietokantakyselykieli.
MySQL	SQL-kyselyillä toimiva tietokantaohjelmisto.
HTML	HyperText Markup Language. Merkintäkieli, käytetään verkkosivujen muotoilussa.
CSS	Cascading Style Sheets. Verkkosivujen tyylityksen määrittelykieli.
JS	JavaScript. Selaimessa suoritettava skriptikieli.
Git	Versionhallintaohjelmisto. Hajautetun versionhallintamallin mukaan toimiva versionhallintaohjelma.

1 Johdanto

Insinöörityön tavoitteena on tutkia web-kehityksessä yleisesti käytettävissä olevia työkaluja ja ohjelmakirjastoja. Työssä perehdytään tekniikoihin ja sen pohjalta tehdään yksinkertainen verkkosivu käyttäen osaa työn alkuosassa käsitellyistä tekniikoista.

Tekniikat, joihin työssä perehdytään, on jaoteltu palvelimen puolella toimiviin eli back end -tekniikoihin, joihin kuuluvat muun muassa PHP, MySQL ja Nodejs, sekä käyttäjälle näkyviin eli front end –tekniikoihin, kuten HTML, CSS ja JavaScript. Eri tekniikat käydään läpi selvittämällä niiden perustoiminnallisuuksia ja niiden erilasia käyttötarkoituksia. Työssä selvitetään myös yleisesti ohjelmistokehityksessä käytössä olevan Git-versionhallintaohjelmiston perusteita. Myös verkkosivujen kehityksessä käytettäviä ohjelmakehyksiä ja -kirjastoja, kuten Laravel, jQuery ja Bootstrap, tutkitaan työn alkuosassa. Laravel-ohjelmakehys sekä Bootstrap- ja jQuery-kirjastot on tehty työn alkuosassa esiteltäviä tekniikoita käyttäen.

Työssä tehdään yksinkertainen verkkosivu käyttämällä osaa työn alkuosassa käsiteltävistä tekniikoista ja web-kehityksessä käytetyistä työkaluista. Työn lopussa pohditaan yksinkertaisen verkkosivun kehittämistä ja laajentamista eri kirjastoja hyödyntämällä.

2 Käytetyt tekniikat

Web-kehityksessä käytettäviä tekniikoita on monia. Alkeelliseen staattiseen verkkosivuun ei tarvitse kuin HTML-rakenteen ja elementtien tyylisääntöjä CSS-tyylikielellä. Kun web-palveluun halutaan lisätä vuorovaikutusta, dynaamisuutta ja tietokannan käyttöä, tarvitaan pelkän HTML-rakenteen ja CSS-tyylisääntöjen lisäksi PHP-skriptikieltä palvelimen puolella ja JavaScript-skriptikieltä asiakasohjelman puolella. Jos halutaan useita samanlaisia toiminnoiltaan rikkaampia web-palveluita, tulee PHP-ohjelmakehyksen käyttäminen ajankohtaiseksi. Myös JavaScript-kirjastojen kuten jQuery-kirjaston käyttäminen kannattaa, sillä se tarjoaa selkeitä funktioita dynaamisuuden lisäämiseksi web-palveluun ja toimii myös eräänlaisena yhteensopivuuskerroksena eri selaimille.

2.1 Palvelinpuolen tekniikat

Palvelinpuoli eli back end on käyttäjälle näkymätön ohjelman osa. Back endia voi ohjailla erillisellä rajapinnalla käyttäjän toimintojen kautta. Back endissa ovat tietokannat ja tiedon käsittelyä hoitavat ohjelman osat.

PHP-skriptikieli

PHP (PHP: Hypertext Preprocessor) on avoimen lähdekoodin yleiskäyttöinen web-kehityksessä käytettävä skriptauskieli. PHP-koodi suoritetaan palvelimella eli back endissä ja lähetetään suorituksen jälkeinen tulkattu HTML-sisältö selaimelle. PHP-koodi ei näy käyttäjälle, koska se suoritetaan palvelimen puolella ja asiakaspuolelle näkyy vain tulkatun PHP-koodin lopputulos. [1.]

Vaikka PHP:n yleisin käyttötarkoitus on web-palvelimessa käytetty dynaamisten verkkosivujen kehittäminen, voi PHP:tä suorittaa myös paikallisesti komentorivin kautta tavallisen skriptauskielen tavoin [2]. PHP:n komentorivillä suorittamista käytetään esimerkiksi Laravel-ohjelmakehyksen omassa komentorivityökalussa.

PHP toimii usealla käyttöjärjestelmällä ja web-palvelinsovelluksella. PHP:llä voi generoida muun muassa kuvia, PDF-tiedostoja ja XML-tiedostoja, ja ne voi tallentaa levyjärjestelmään myöhempää käyttöä varten. PHP tukee myös usean eri sähköposti- ja tietokantajärjestelmän käyttöä. [2.]

MySQL-tietokantasovellus

MySQL on yleisesti käytössä oleva avoimen lähdekoodin tietokantasovellus. Nimen MySQL My-osa tulee kehittäjän tyttären nimestä My ja SQL-osa lyhenteestä Structured Query Language. SQL on yleisin standardoitu kieli tietokantayhteyksiä varten. [3.]

Tietokantamallina MySQL:ssä toimii relaatiomallinen tietokanta, jossa tietokanta on jaettu useaan tauluun yhden ison taulun sijaan, ja näiden pienempien taulujen välille luodaan yhteyksiä eli relaatioita, joiden mukaan tietoa haetaan. MySQL-tietokannan hallintaan käytetään tietokantahallintajärjestelmä MySQL Serveriä. [3.]

Tietoa haetaan ja käsitellään tietokannasta SQL-kyselyillä. Ennen kyselyjen tekemistä tietokantaan luodaan yhteys. Kyselyjen oikeanlainen muotoilu on nopeampaa ja vie vähemmän resursseja, kuin esimerkiksi kokonaisen taulun tuominen PHP-muuttujaan ja sen käsitteleminen koodissa.

Nodejs-JavaScript-ympäristö

Nodejs on palvelimen puolella toimiva JavaScript-koodia suorittava ohjelma. Nodejs:n pohjana toimii muunneltu Googlen Chrome-selaimen V8 JavaScript -moottori. Nodejs suorittaa JavaScriptiä selaimen tapaan. Toimintaperiaatteeltaan Nodejs on lähellä PHP:tä: ensin JavaScript-tiedosto tulkataan, minkä jälkeen tulkattu sisältö lähetetään asiakasohjelmalle tai suoritetaan paikallisesti komentorivillä. Nodejs:n pohjalta on myös tehty paketinhallintaohjelma NPM (Node Package Manager), jolla Nodejs:lle tehtyjä JavaScript-pohjaisia kirjastoja, ohjelmia ja työkaluja saa haettua, asennettua ja hallittua.

2.2 Selainpuolen tekniikat

Front end on käyttäjälle näkyvä ja käyttäjän hallitsema ohjelman osa. Käyttäjän ohjaama front end voi olla yhteydessä rajapinnan kautta back endiin, mikä mahdollistaa palvelinpuolen ominaisuuksien käyttämisen, kuten tietokantakutsut ja palvelimella sijaitsevien

tiedostojen hallinta. Front endiin luetaan käyttäjälle esimerkiksi selaimessa näkyvä HTML-merkintäkieltä, CSS-tyylikieltä ja JavaScript-koodia sisältävä verkkosivu.

HTML-merkintäkieli

HTML (HyperText Markup Language) on merkintäkieli, jota käytetään web-sivun rakenteen määrittelemiseen. Web-sivulle saavuttaessa selain lähtee ensin rakentamaan sivun sisältöä ja rakennetta vastaanotettujen HTML-määritysten perusteella. HTML:llä määritellään, miten sisältöä esitetään ja miten esimerkiksi teksti asetellaan. Web-sivuilla on monta erilaista elementtiä, kuten kuvia, videota, ääntä, valikoita, syöttökenttiä ja hyperlinkkejä muille web-sivuille. Peruselementtien määrittely onnistuu HTML:llä, mutta niiden ulkonäköön ja toiminnallisuuteen tarvitaan tyylitiedostoja ja skriptikieltä. [4.]

CSS-tyylikieli

CSS (Cascading Style Sheets) on esityskieli, joka määrittelee, miten HTML-merkintäkielellä määritetty web-sivuna esitettävä dokumentti esitetään selaimessa [5]. Selain luo web-sivun rakenteen ensin HTML:n perusteella, minkä jälkeen se ryhtyy lisäämään web-sivun eri elementeille niille määritettyjä esitystyylejä [5]. CSS:ää käytetään muun muassa elementtien värin, koon, reunusten, esitysjärjestyksen, tekstin koon ja tekstin fontin määrittelyyn.

JavaScript-skriptauskieli

JavaScript (lyhennettynä JS) on verkkosivuilla käytetty tulkattu, kevyt olio-pohjainen skriptauskieli. Se toimii pääasiassa selaimessa, mutta sitä näkee myös muualla, kuten palvelinpuolella toimivalla Nodejs-alustalla. JavaScript suoritetaan front endissä asiakaspuolella selaimessa, jossa sitä käytetään muun muassa web-sivujen käytöksen määrittämiseen eri tapahtumien esiintyessä ja yleisen käytöksen määrittämiseen. JavaScript voi toimia sekä proseduraalisena että olio-pohjaisena ohjelmointikielenä. Kaikki JavaScriptissä käytettävät metodit, muuttujat ja oliot luodaan ajon aikana. [6.]

Selaimissa on käytössä selainten omat JavaScript-moottorit, jotka toteuttavat JavaScriptiä ECMAScript-standardin mukaisesti. Mozillan selaimessa on käytössä C++:lla toteutettu SpiderMonkey-moottori ja Javalla toteutettu Rhino-moottori. Googlen selaimessa käytetään V8 JavaScript -moottoria, johon myös Nodejs perustuu. Internet Explorer -selaimissa on käytössä Chakra-niminen JavaScript-moottori. Opera-selaimessa on ollut

oma JavaScript-moottorin toteutus Carakan, mutta uudemmissa versioissa Opera on siirtynyt käyttämään Googlen selaimissa käytettävää V8-moottoria. [6.]

2.3 Versionhallinta Git-versionhallintaohjelmistolla

Git on avoimella lähdekoodilla toteutettu hajautetun versionhallintamallin mukaan toimiva versionhallintaohjelmisto. Versionhallinnan käyttämisen hyödyt korostuvat, kun projektin koko kasvaa vaikeasti hallittavaksi. Usean henkilön tullessa mukaan projektiin versionhallinnan käyttäminen tuo useita etuja koodin jakamisen ja versioiden hallinnan kanssa.

Versionhallintajärjestelmä tallentaa ajan myötä tiedostoihin tehdyt muutokset rekisteriin. Muutosten tallentaminen rekisteriin mahdollistaa muutosten kumoamisen, yksittäisen tiedoston tai koko projektin palauttamisen vanhempaan versioon, muutosten vertailua ja seuraamista. [7, s. 1.]

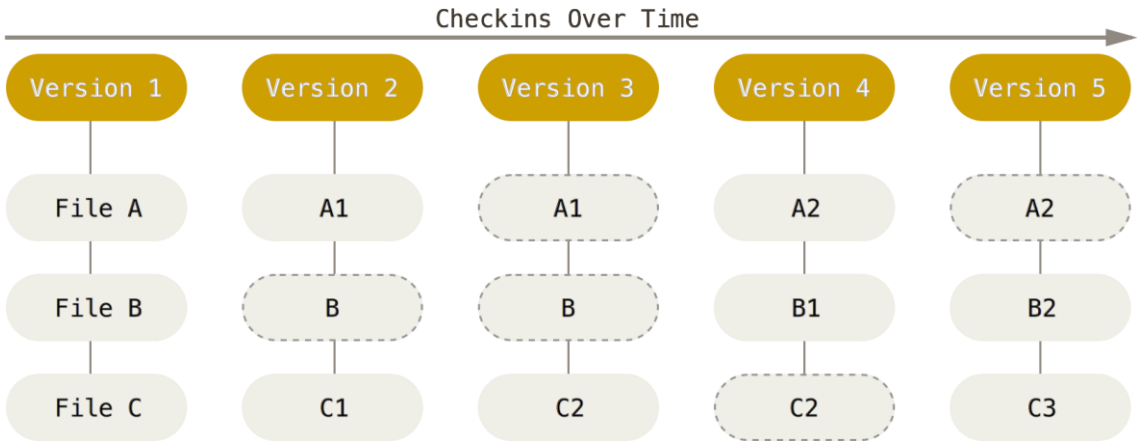
Versionhallintamalleja on useita. Paikallinen versionhallinta voi tarkoittaa kansioiden uudelleennimeämistä, esimerkiksi päivämäärän mukaan, eri versioiden erottamisen helpottamiseksi. Pelkkä tiedostojen tai kansioiden nimeäminen versioiden erottamiseksi on kuitenkin hankalasti ylläpidettävä ja vahinkoaltis versiointitapa. Keskitetyssä versionhallintamallissa projektin versiotiedot tallennetaan keskitetysti palvelimelle, josta käyttäjät voivat projektia kehittää. Keskitetyssä versionhallintamallissa, kuten muissakin keskitetyissä palveluissa, voi tulla ongelmia, jos ainoa palvelin lakkaa toimimasta, jolloin projektin kehitys saattaa pysähtyä kokonaan. [7, s. 1-4.]

Hajautettu versionhallintamalli toimii kuten keskitetty versionhallintamalli, mutta hajautetussa mallissa jokainen käyttäjä kopioi palvelimelta koko versiohistorian paikalliseksi kopioksi. Versiohistorian kopioiminen tekee palvelimen toiminnan lakkaamisesta vähemmän haitallista kuin esimerkiksi keskitetyssä versionhallintamallissa, jossa kehitys saattaa pysähtyä kokonaan. Versiohistorian kopiosta voi jatkaa kehitystä muistakin kuin alkuperäisestä lähteestä. Eri lähteestä kehityksen jatkaminen voi myös olla tahallista, jolloin sitä kutsutaan branchaamiseksi eli haarautumiseksi. [7, s. 1-4.]

Git pitää versioista kirjaa ottamalla joka kommitin yhteydessä tilannevedoksen kaikista versioinnin alaisista tiedostoista kommitin aikana. Suorituskyvyn parantamiseksi Git ei

kuitenkaan sisällytä tilannevedokseen muuttumattomia tiedostoja, vaan lisää muuttumattoman tiedoston viittauksen aikaisempaan tilannevedokseen. [7, s. 6.]

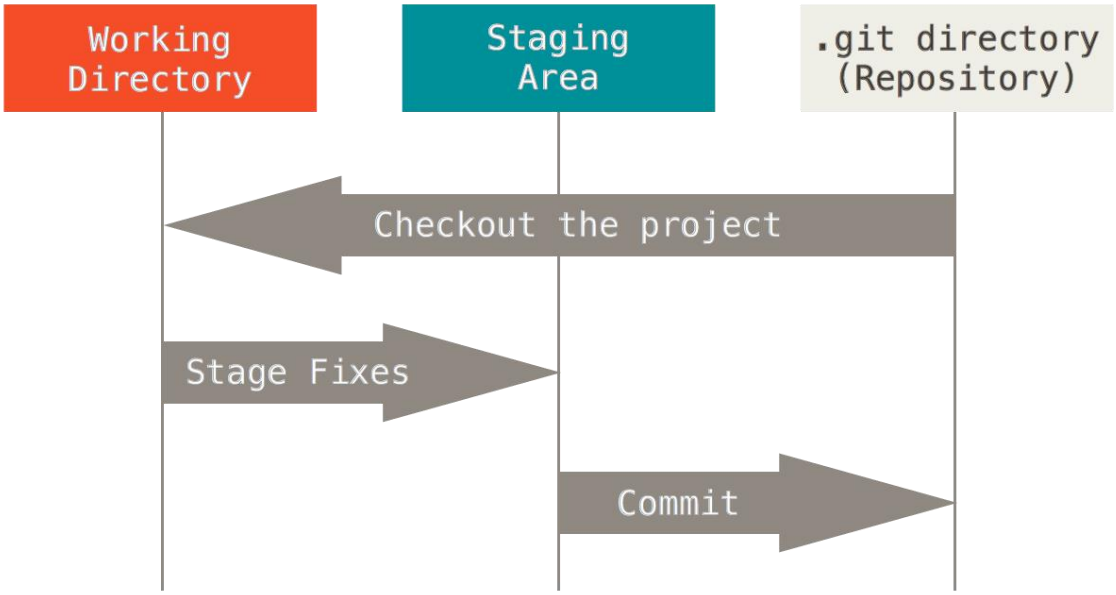
Kuvassa 1 on havainnoillistus Gitin tilannevedoksista, katkoviivoin merkityt tiedostot pysyvät tilannevedoksissa mukana, mutta viittaavat aikaisempaan muuttumattomaan versioon.



Kuva 1. Havainnollistus Gitin tilannevedoksista [8].

Gitissä versiohistoria tallentuu paikalliseen versiotietokantaan, kunnes tehdyt muutokset työnnetään etäiseen repositorioon eli säilytyspaikkaan. Työskentely ei siis keskeydy pitkäänkään aikaan, vaan kehitys voi jatkua monessa kehityshaarassa, kunnes koko projektin saa sulautettua yhteen. Git ottaa versioista 40-merkkisen heksadesimaaleja sisältävän SHA-1-tarkistussumman, jolla se merkitsee kommitit ja varmistaa tarkistussummalla, ettei mikään muutu Gitin sitä huomaamatta. [7, s. 7.]

Gitissä tiedostoilla voi olla kolme tilaa, kommitoitu, muokattu ja vaiheistettu. Kommitoitu tila tarkoittaa, että tieto on tallennettu paikalliseen tietokantaan tilannevedokseen. Muokattu-tila tarkoittaa, että tietoa on muokattu, mutta sitä ei ole kommitoitu eli tallennettu tehtyjä muutoksia paikalliseen tietokantaan. Vaiheistettu-tila tarkoittaa, että muokattu-tilan tiedosto on merkitty nykyisellään seuraavaan paikallisen tietokannan tilannevedokseen. [7, s. 8.] Kuvassa 2 näkyy tiedoston siirtyminen kolmen eri tilan välillä.



Kuva 2. Gitissä olevien tiedostojen kolme eri tilaa ja niiden välillä siirtyminen [8].

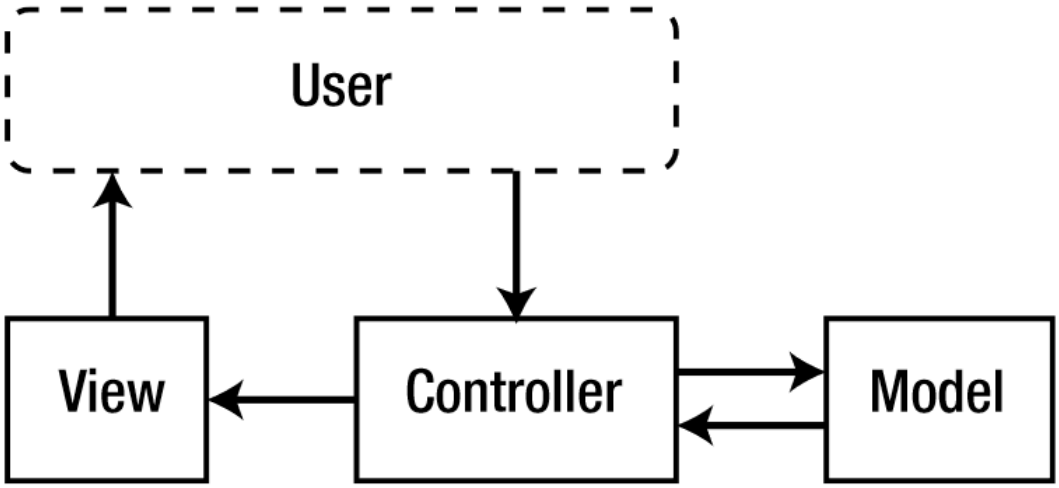
Git-projektissa on kolme eri kohtaa: *.git*-kansio, työskentelykansio ja vaiheistusalue. Tiedostojen metadata ja tilannevedokset sisältävä tietokanta sijaitsee *.git*-kansiossa eli repositoriossa. Työskentelykansio on yksittäinen ulosotettu versio repositorion kloonista. Vaiheistusalue on *.git*-kansion tiedosto, jossa näkyy, mitä seuraava kommitointi pitää sisällään. Työvaiheet menevät järjestyksessä seuraavasti: projektin tiedostoja muutetaan ja ne lisätään vaiheistusalueeseen ja niistä otetaan tilannevedokset, tilannevedos kommitoidaan *.git*-kansioon. [7, s. 8.]

2.4 Ohjelmakehykset ja -kirjastot

Ohjelmakehysten ja -kirjastojen käyttö tuo monia etuja web-palvelujen kehityksessä ja ylläpidossa: ne vähentävät kirjoitettavan koodin määrää ja helpottavat projektin eteenpäin viemistä, kun osa toiminnoista löytyy ohjelmakehysten ja kirjastojen valmiista funktioista ja rakenteista. Ohjelmakehysten ja kirjastojen perehtymiseen kuitenkin menee aina aikaa, joten yksinkertaisissa projekteissa ei kannata väkisin käyttää useampaa kirjastoa kuin on pakko. PHP-ohjelmakehyksissä on yleisesti käytössä MVC-toimintaperiaate (Model View Controller).

MVC-toimintaperiaate perustuu kolmen pääkomponentin yhteyteen. Kolme komponenttia on nimetty *malli*-, *kontrolleri*- ja *view*-osiksi. *Malli*-osassa on ohjelman logiikka, miten

tietoa tallennetaan ja mitä palveluita taustalla käytetään. Tietokantaan liittyvät osat sisällytetään yleensä *malli*-osaan. *View*-osassa on käyttäjälle näkyvät web-ohjelman palat, kuten web-sivu, johon on sisällytetty HTML-, CSS- ja JavaScript-tiedostoja. Kaikki käyttäjälle näkyvä sisällytetään yleensä *view*-osaan. *Kontrolleri*-osa niputtaa *malli*- ja *view*-osat yhteen. *Kontrolleri*-osa eristää *malli*-osan sisällön *view*-osan käyttöliittymästä ja käsittelee, miten verkkosivu vastaa käyttäjän toimintaan *view*-osassa. *Kontrolleri*-osa on ensimmäinen komponentti, mihin käyttäjän suunnasta tulevat pyynnöt tulevat, ja se käyttää hyväkseen *view*- ja *malli*-osan toimintoja toteuttaakseen pyynnöt. [9, s. 1.] Kuva 3 kuvaa MVC-toimintaperiaatteen osia.



Kuva 3. MVC-toimintaperiaatteen osat ja niiden väliset suhteet [9, s. 2].

Laravel-ohjelmakehys

Laravel on PHP:lla toimiva ohjelmakehys, jossa on valmiita työkaluja ja toiminnallisuuksia web-sivuston tekemiseen. Laravel toimii yleisesti käytössä olevalla MVC-toimintaperiaatteella (Model-View-Controller).

Laravelin perusasennuksen juurikansiossa on seuraavat hakemistot:

- App-kansiossa on ohjelmakehyksen ydinosia. Kontrollerit, routet sekä itse konsolikomennot sijaitsevat app-kansiossa.
- *Bootstrap*-kansiossa on ohjelmakehyksen alustamisen kannalta tärkeitä tiedostoja ja konfiguraatiotiedostojen automaattiseen lataamiseen liittyviä tiedostoja. Kansiossa on myös *cache*-kansio, johon ohjelmakehys varastoi

luomiaan välimuistitiedostoja ohjelmakehyksen käynnistämisen nopeuttamista varten.

- *Config*-kansiossa ovat ohjelmakehyksen asetustiedostot.
- *Database*-kansiossa ovat tietokantaan liittyvät *migraatio*- ja *seeder*-tiedostot. Migraatiotiedostoilla voi luoda tietokantaan määritetynlaisia tauluja. *Seeder*-tiedostoilla tietokannan tauluihin voi seedata eli automaattisesti syöttää tietoa, joko satunnaista tai harkittua.
- *Public*-kansio sisältää muun muassa web-sivulla käytettäviä kuvia sekä JavaScript- ja CSS-tiedostoja.
- *Resources*-kansiossa on view't käännettävät ja tulkattavat tiedostot (TypeScript, CoffeeScript, LESS, SASS) sekä lokalisaatiotiedostot.
- *Storage*-kansiossa ovat käännetyt blade-templaatit ja ohjelmakehyksen generoimia tiedostoja. Kansion alakansioita ovat *app*, *framework* ja *log.* Alakansioista löytyy verkkosivun käyttämiä tiedostoja, ohjelmakehyksen luomia tiedostoja ja välimuistitiedostoja sekä web-ohjelman lokitiedostoja.
- *Tests*-kansiossa on automatisoituja testejä.
- *Vendor*-kansio sisältää Composer-työkalun riippuvuuksia. [10.]

jQuery-kirjasto

jQuery-kirjasto tarjoaa suuren kokoelman JavaScript-funktioita. JQuery helpottaa elementtien välillä liikkumista käyttämällä CSS-valitsimia elementtien hallintaa varten, tapahtumien hallintaa, elementtien animointia ja manipulointia sekä toimii eräänlaisena yhteensopivuuskerroksena eri selainversioiden välillä.

Kun vakio-JavaScriptillä elementin valinta tehdään koodilla *document.getElementById('elementti'),* voi jQueryllä tehdä saman koodilla *$("#elementti")*. Elementtejä voi myös valita elementille asetetun CSS-luokan mukaan, esimerkiksi *$(".luokka")*. Elementtien attribuuttien lisäämiselle, tapahtumien kaappaamiselle, elementtien välillä liikkumiselle ja useamman samannimisen elementin käsittelylle on valmiit ja selkeät funktiot.

Bootstrap-kirjasto

Bootstrap on CSS-kirjasto, joka on suunniteltu ensisijaisesti mobiililaitteita varten ja käsittää myös suurempien näyttöjen koot. Bootstrap käyttää globaaleja CSS-asetuksia ja näyttöruudun kokoon sopeutuvan ruudukkomallin. Bootstrapin käyttämä ruudukkomalli on kahteentoista sarakkeeseen jaettu pohja, jossa elementtejä voi määrittää käyttämään tietyn määrän sarakkeita ja asetella elementtejä sarakemitoissa. Sisäkkäin olevat

elementit voi myös jakaa sarakkeisiin, jotka jakautuvat kahteentoista. Bootstrap tarjoaa myös valmiita tyylejä normaalille tekstille, lainauksille, listoille, taulukoille, lomakkeille, painikkeille ja kuville.

3 Esimerkkiverkkosivu

Opinnäytetyössä tehtiin yksinkertainen esimerkkiverkkosivusto, jossa käytettiin Laravel-ohjelmakehyksen osia sekä CSS- ja JavaScript-kirjastoja. Verkkosivua on mahdollista laajentaa käyttämään monimutkaisempia ohjelmakehyksen ulkopuolisia kirjastoja ja komponentteja.

3.1 Valmistelut

Web-palvelin, jolle esimerkkiverkkosivua lähdettiin tekemään, oli valmiiksi asennettu web-palvelin, jossa oli asennettuna Apache-, MySQL- ja PHP-palvelut. Esimerkkiverkkosivua kehitettiin SSH-yhteyden kautta. Verkkosivun koodin editoimiseen käytettiin lisäosilla laajennettua Vim-editoria. Web-palvelimeen saatiin selaimella yhteys lähiverkon kautta.

Palvelimella oli valmiiksi luotuna oma git-käyttäjä, jonka kotikansiossa on projektissa käytetty Git-repositorio eli Git-säilytyspaikka. Repositorion voi kloonata palvelimelta mille tahansa toiselle koneelle ja jatkaa kehitystä esimerkiksi Windowsilla XAMPP-web-palvelimen avulla.

Aluksi Git-repositorio sisälsi vain tyhjän kansion, johon Laravel-ohjelmakehys voitiin asentaa. Laravel-ohjelmakehyksen asentamiseen käytettiin Composer-työkalua, joka toimii kirjastojen ja työkalujen riippuvuudenhallintatyökaluna. Kun asennettiin Laravel-ohjelmakehyksen asennusohjelma, se asentui paikallisesti käyttäjän kotikansiossa sijaitsevaan *.composer/vendor/bin*-kansioon. Laravelia voi kutsua suoraan kansiosta *.composer/vendor/bin/laravel*-komennolla tai lisäämällä *.composer/vendor/bin/laravel*-kansio *~/.bashrc* tiedoston *PATH*-muuttujaan. Kun Laravel-ohjelman sisältävä kansio on lisätty PATH-muuttujaan, voi *laravel*-komentoa käyttää ilman, että koko komentoa edeltävää kansiorakennetta tarvitsee kirjoittaa, eli komentona riittää pelkkä *laravel*.

3.2 Laravel-ohjelmakehyksen alustaminen ja käyttöönotto

Laravel-ohjelmakehys asennettiin *laravel*-komennolla kloonatun git-repositorion kansiossa komennolla *$ laravel new testi*. Ohjelmakehyksen alustava komento luo testi-kansion ja alustaa sinne Laravel-ohjelmakehyksen pohjan. [11.]

Ennen kuin ohjelmakehystä voitiin käyttää, täytyi ohjelmakehyksen kansion juuressa antaa *$ chmod –R 777*-komennolla täydet käyttöoikeudet *storage*- ja *bootstrap/cache*-kansioille. Tietokantaa varten muokattiin ohjelmakehyksen kansion juuressa sijaitsevaa *.env*-tiedostoa, josta löytyy ohjelmakehyksen ympäristömuuttujia. Ympäristömuuttujiin kuuluu muun muassa tietokannan nimi ja tietokantakäyttäjän kirjautumistiedot. Projektia varten luotiin paikallinen MySQL-tietokanta nimellä *testiprojdb*. Tietokannan käyttöä varten luotiin käyttäjä nimellä *testiprojuser*, ja sille asetettiin salasanaksi *testiprojsala*. Tietokanta oli tyhjä eikä sisältänyt tauluja. Tietokannan käytön mahdollistavat tiedot lisättiin *.env*-tiedostoon. Kun Laravelin *DB*-tietokantaluokkaa käyttää, osaa Laravel käyttää ympäristömuuttujiin asetettuja tietokannan tietoja, jolloin ei tietokantaan liittyviin kirjautumisiin ja yhdistämisiin tarvitse kiinnittää huomiota. [11.]

Laravel-ohjelmakehyksen alustamisen jälkeen projektikansio työnnettiin (push) Gitiin. Aluksi Gitille kerrotaan komennolla *$ git add –all,* mitä tiedostoja halutaan sisällyttää repositorioon työnnettäväksi (push). *$ git add –all* -komennolla lisätään koko projektin juurihakemisto ja jatketaan rekursiivisesti kansiorakennetta syvemmälle sekä sisällytetään kaikki juuresta eteenpäin repositorioon. Tämän jälkeen täytyy muutokset kommitoida repositorioon komennolla *$ git commit –m ”ensimmäinen commit”*, jolloin kaikki kommittiin lisätyt tiedostot lähtevät seuraavan pushin eli työnnön yhteydessä *git@localhost:repositories/projekti.git*-repositorioon. Työnnön jälkeen myös muut repositorion käyttäjät voivat hakea projektin uusimpia versioita.

Tyhjän repositorion kloonauksen jälkeen on ensimmäisen kommitin viemisen yhteydessä lisättävä *--set-upstream origin master* -lippu, jotta Git tunnistaa käytettäväksi haaraksi *master*-haaran johon kommitti viedään. Ensimmäinen Gitin versiorekisteriin vienti eli työntö onnistuu siis komennolla *$ git push --set-upstream origin master*.

Laravelin artisan-komentorivityökalu

Artisan on Laravelin mukana tuleva komentorivityökalu, jolla voi suorittaa useita eri valmiiksi määriteltyjä toimintoja. Valmiiksi määritetyillä toiminnoilla voi luoda pohjat kontrollereille, malleille, tietokantamigraatioille ja tietokannan seedereille. *Artisan*-komentojen määrää voi myös itse laajentaa tekemällä omia komentoja. Komentoja voi myös suorittaa ohjelmakoodin seasta. [12.]

Listan komentorivillä suoritettavista *artisan*-komennoista saa komennolla *$ php artisan list*, joka tulostaa listan komennoista ja kuvauksen komennon toiminnasta.

Artisan-komentoa tehdessä voi suorittaa *$ php artisan make:console Komento* -komennon, jolloin artisan luo *app/Console/Commands*-kansioon *Komento.php*-nimisen tiedoston. *Komento.php*-tiedostossa on *Komento*-komennon suorittamista varten tarvittavat tiedot [11.] Esimerkkikoodin 1 tapauksessa komento ottaa vastaan tekstiä parametrinä. Jos parametriä ei anneta, on muuttujan oletusarvona ”*tyhjä*”.

```
protected $signature = 'komento:testi {testi=tyhjä}';
```

Esimerkkikoodi 1. *Komento*-komennon kutsun ja parametrin määrittävä osa komennon määrittävässä *Komento*.php-tiedostossa.

Parametrejä voi hakea funktiokutsulla *$this->parameter(’testi’)*. Parametriä voi käsitellä aivan tavallisen *string*-muuttujan tapaan koodissa. Komennon suoritettava osa on *handle*-funktiossa, jossa parametrejä ja käyttäjän syötteitä voidaan käsitellä.

Käyttäjältä voidaan kysyä syötteitä muun muassa tavallisella syötteellä, kyllä/ei-syötteellä, ennakoivalla syötteellä ja monivalintasyötteellä. Kuvassa 4 näkyy *Komento.php*-tiedoston sisällä olevan *handle*-funktion sisältämä koodi.

```
38    public function handle(){
39        $this->info($this->argument('testi'));
40
41        $tulosta = $this->ask("kirjoita tulostettava: ");
42        $this->line($tulosta);
43
44        if($this->confirm('kyllä/ei?')){
45            $this->line('Valitsit kyllä');
46        }else{
47            $this->line('Valitsit ei');
48        }
49
50        $kirjoita = $this->anticipate('Anna maa', ['Suomi', 'Ruotsi'], false
);
51
52        $valinta = $this->choice('Valitse', ['Eka', 'Toka', 'Vika']);
53        $this->line('Valitsit ', $valinta);
54
55    }
```

Kuva 4. Komentoa käsittelevän *handle*-funktion sisällä oleva koodi.

Kuvan 4 rivillä 39 tulostetaan parametrinä saatu muuttuja. Riveillä 41 ja 42 pyydetään syötettä, joka tulostetaan uudelle riville. Riveillä 44-10 nähdään, miten kyllä/ei-syöte toimii, ja komentoriviltä pyydetään syötettä. Kyllä/ei-syötteessä käytetty funktio *confirm()* palauttaa arvoksi toden tai epätoden, jonka mukaan *if*-lausekkeen lohkot suoritetaan.

Rivillä 50 nähdään ennakoivan syötteen esimerkki, jossa komentorivillä pyydetään syötettä ja valmiita vaihtoehtoja ehdotetaan annetun syötteen mukaan. Rivillä 52 näkyy monivalintasyöte, jossa määritetään taulukon valinnat, jotka annetaan komentorivillä käyttäjälle ja jotka voi valita joko annetun indeksin tai annetun syötteen mukaan.

Uuden komennon luomisen jälkeen luotu komento täytyy lisätä *app/Console/Kernel.php*-tiedostoon *$commands*-muuttujaan. Kuvassa 5 näkyy uuden komennon lisääminen käytössä oleviin komentoihin.

```
15      protected $commands = [
16          \App\Console\Commands\Inspire::class,
17          'App\Console\Commands\Komento'
18      ];
```

Kuva 5. Uuden komennon lisääminen käytössä olevien komentojen joukkoon.

Tämän jälkeen komento on käytettävissä komentorivin kautta komennolla *$ php artisan komento*.

Luoduilla *artisan*-komennoilla voi suorittaa eri toimintoja projektin ylläpitoa tai kehitystä varten. Komennoilla voi esimerkiksi käsitellä tietokantaa tai tehdä projektin pystytysprosessin, jossa tiettyä projektipohjaa monistetaan eri käyttöön.

Tietokantamigraatiota varten tehtävän tiedoston luominen onnistuu käyttämällä *artisan*-komentoa *make:migration*, joka luo tietokantamigraatiotiedoston. Tietokantamigraatiotiedostoon voi lisätä haluamansa taulun nimen ja rakenteen. Komennon perään voi lisätä *--create=pages*-lipun, joka luo tiedostoon valmiin *Schema::create*-funktiolla luodun rakenteen *pages*-taulua varten. Tietokantamigraatiotiedoston luominen *pages*-taulua varten onnistuu komennolla *$ php artisan make:migration create_pages_table –create=pages*, joka luo *database/migrations*-kansioon *CreatePagesTable.php*-tiedoston. CreatePagesTable.php-tiedostoon voi lisätä halutun taulun rakenteen. Kuvassa 6 näkyy *pages*-taulun haluttu rakenne.

```
+-------------+------------------+------+-----+---------------------+----------------+
| Field       | Type             | Null | Key | Default             | Extra          |
+-------------+------------------+------+-----+---------------------+----------------+
| id          | int(10) unsigned | NO   | PRI | NULL                | auto_increment |
| url         | char(100)        | NO   |     | NULL                |                |
| name        | char(100)        | NO   |     | NULL                |                |
| description | longtext         | YES  |     | NULL                |                |
| settings    | text             | YES  |     | NULL                |                |
| created_at  | timestamp        | NO   |     | 0000-00-00 00:00:00 |                |
| updated_at  | timestamp        | NO   |     | 0000-00-00 00:00:00 |                |
+-------------+------------------+------+-----+---------------------+----------------+
```

Kuva 6. *Pages*-taulun rakenne.

Kuvassa 7 olevan taulun luomista varten täytyy *CreatePagesTable.php*-tiedoston *up*-funktiossa olevaan automaattisesti luotuun *Schema::create*-funktiokutsuun lisätä kuvassa 7 näkyvien rivien 16-21 mukaiset komennot. *Schema::create*-funktiokutsussa on automaattisesti lisättyjä osia, kuten *increments('id')* ja *timestamps()*. Nämä automaattisesti lisätyt kohdat lisäävät tauluun automaattisesti inkrementoituvan *id*-sarakkeen ja *timestamps*-funktio sarakkeet *created_at* ja *updated_at*.

```
13      public function up()
14      {
15          Schema::create('pages', function (Blueprint $table) {
16              $table->increments('id');
17              $table->char('url', 100);
18              $table->char('name', 100);
19              $table->longtext('description')->nullable();
20              $table->json('settings')->nullable();
21              $table->timestamps();
22          });
23      }
```

Kuva 7. Migraatiotiedoston määritys taulun luomista varten.

Kuvassa 7 rivinumeroilla 17-20 asetetaan *Blueprint*-luokan funktioita käyttäen sarakkeille *url*, *name*, *description* ja *settings* halutut määritykset.

Kun migraatiotiedosto on luotu, voidaan ajaa migraatiokomento. Migraatiokomento luo migraatiotiedostojen mukaiset taulut *$ php artisan migrate* -komennolla. Kun migraatiokomento on suoritettu ilman virheitä, näkyy *.env*-tiedostossa määritetyssä tietokannassa aikaisemmin kuvassa 6 määritetty taulu ja taulun rakenne, jolloin taulu on valmis käytettäväksi.

Tietokannan seedaaminen tilankäyttötiedoilla onnistuu *database/seeders*-kansiossa sijaitsevilla *seeder*-tiedostoilla. *Seeder*-tiedostot lisäävät tietokantoihin testitietoa tai lopullista käytettävää tietoa, jos tietokantoihin halutaan joitakin pysyvämpää perustietoa mukaan. *Seeder*-tiedostojen luominen onnistuu käyttämällä *artisan*-komentoa *make:seeder*. *Seeder*-tiedoston luominen aikaisemmin määriteltyyn *pages*-taulun täyttämistä varten onnistuu käyttämällä komentoa *$ php artisan make:seeder PagesTableSeeder*. Komento luo lähes tyhjän *PagesTableSeeder.php*-tiedoston *database/seeder*-kansioon. Luotu tiedosto sisältää kehyksen, jonka sisällä voi määritellä minkälaista tilankäyttötietoa tietokantaan halutaan lisätä. Kuvassa 8 näkyy *artisan*-komennolla luodun *seeder*-tiedoston sisältö.

```
12      public function run()
13      {
14          DB::table('pages')->insert([
15              'url' => str_random(6),
16              'name' => 'Test page '.str_random(5),
17              'description' => 'Test description '.str_random(5),
18              'settings' => '{}',
19          ]);
20      }
```

Kuva 8. *Seeder*-tiedoston suoritettava tietokannan *seeder*-koodi.

Kuvassa 8 kutsutaan *DB*-luokan *table*-funktion kautta *insert*-funktiota, jolle annetaan taulukko. Annetussa taulukossa on määritetty, mitä tauluun syötetään. *PagesTableSeeder.php*-tiedostolla tauluun lisätään *url*-, *name*-, *description*- ja *settings*-sarakkeisiin vakiolaatuista ja satunnaista *string*-tyyppistä tietoa, ja muihin sarakkeisiin tulee automaattista tietoa tietokannan puolesta. Jotta juuri luotua *seeder*-tiedostoa voi käyttää *$ php artisan db:seed*-komennon yhteydessä, on tiedosto lisättävä *database/seeds/DatabaseSeeder.php*-tiedostoon. *DatabaseSeeder.php*-tiedostolla voi kootusti suorittaa monia eri *seeder*-tiedostoja. Yksittäisiä *seeder*-tiedostoja voi kuitenkin suorittaa lisäämällä *$ php artisan db:seed* -komentoon lipun. Komentoon lisättävä lippu voi olla esimerkiksi *–class=PagesTableSeeder*, jolloin vain *PagesTableSeeder.php*-tiedostoon määritetyt tietokantaseedaukset ajetaan.

Pages-taulun käsittelyä varten on myös hyvä luoda *malli*, jolla tietokantaa hallitaan *kontrolleri*-osan kautta. Pagesin käsittelyä varten luodaan malli komennolla *$ php artisan make:model Page*. Tämä luo *app*-kansioon *Page.php*-tiedoston, joka toimii Pagen mallina. Artisan-komennossa *make:model* voi myös käyttää *–migration*-lippua, jolloin *artisan*

luo mallille samalla migraatiotiedoston. Migraatiotiedosto kuitenkin luotiin jo aikaisemmin, joten lippua ei tarvittu.

Tiedon käsittely, toiminnallisuuksien lisääminen ja view'ien näyttäminen vaatii kontrollerin. Kontrollerit sijaitsevat *app/Http/Controllers*-kansiossa. Uuden kontrollerin luominen onnistuu *artisan*-komennolla *$ php artisan make:controller MainController,* joka luo *MainController.php*-tiedoston Controllerit sisältävään *app/Http/Controllers*-kansioon. Jotta aikaisemmin luotua *Page*-mallia voi käyttää ja etsiä *pages*-taulusta tietoa, täytyy kontrolleria ohjeistaa käyttämään *Page*-mallia lisäämällä *MainController.php*-tiedoston alkuun koodi *use App\Page.* Tämän jälkeen *Page*-mallia voi käyttää kontrollerikoodissa kutsumalla *Pagea*. Kuvassa 9 näkyy *artisan*-komennolla luotuun *MainController.php*-tiedostoon lisätty *showPage*-funktio ja funktion sisältämä koodi.

```
32      public function showPage(Request $request, $url){
33          $page = Page::where('url', $url)->first();
34
35          if(!$page){
36              return redirect('/home');
37          }
38
39          $paged = (object) array(
40              'url' => $url,
41              'title' => $page->name,
42              'description' => $page->description
43          );
44
45          return view('page', ['paged' => $paged]);
46      }
```

Kuva 9. MainController.php-tiedostossa sijaitseva *showPage*-funktio

Kuvassa 9 näkyy, miten funktio *showPage* ottaa vastaan parametreina pyynnön *$request*-muuttujana ja *$url*-muuttujan. Funktio hakee Page-mallia käyttäen *pages*-taulusta ensimmäisen rivin, jossa *url*-sarake vastaa parametrinä saadun *$url*-muuttujan sisältöä. Jos tietokannasta ei löydy haettua *url*-riviä, uudelleenohjataan käyttäjä */home*-urliin. Kun tietokannasta löytyy haettu rivi, laitetaan sen tuloksina saadut kohdat *$paged*-muuttujaan. Käyttäjä ohjataan *page*-view'hun ja view'lle annetaan kontrollerin *$paged*-muuttuja view'lle käytettäväksi *paged*-muuttujanimellä. *View*-funktion toisena parametrinä voi antaa myös useampia muuttujia eri nimillä, jolloin ne täytyy kuvan 9 rivin 44 mukaisesti lisätä taulukkoon.

Ennen kuin luotua kontrolleria voi käyttää, on luotava route eli reitti kontrolleriin. Reitti tehdään muokkaamalla jo olemassa olevaa *app/Http*-kansiossa sijaitsevaa *routes.php*-

tiedostoa, jossa kaikki reitit sijaitsevat. Oletuksena *routes.php*-tiedostossa on vain ohjaus *resources/views/welcome.blade.php*-view'hun. Kuvassa 10 *routes.php*-tiedostoon lisätään reitit *MainControllerin index*- ja *showPage*-funktioihin.

```
14 Route::get('/', function () {
15     return view('welcome');
16 });
17
18 Route::get('/home', 'MainController@index');
19
20 Route::get('/page/{url}', 'MainController@showPage')
21     ->where(['url' => '[a-zA-Z0-9_]+']);
```

Kuva 10. Määritettyjen URLien ohjaaminen MainControllerin funktioihin.

Kuvassa 10 on lisättynä reitit, jotka ohjaavat */home* URL:iin tulleet pyynnöt *MainControllerin index*-funktioon ja */page/{url}* URL:iin tulleet pyynnöt *MainControllerin showPage*-funktioon. Reitti *MainController.php*:n *showPage*-funktioon toteutuu vain, kun *where*-funktion ehto toteutuu ja *{url}* sisältää isoja ja pieniä kirjaimia ja numeroita. Jos käyttäjä saapuu URL:iin, jota ei löydy *routes.php*-tiedostosta, antaa Laravel 404-ilmoituksen ja näyttää tyhjän sivun. Sivulle sisällytetään myös enemmän virheilmoituksia, jos *.env*-tiedostossa olevan *APP_DEBUG*-ympäristömuuttujan arvo on asetettu todeksi.

Kun kontrollerit ja kontrollereihin vievät reitit on luotu, luodaan kontrollerien käyttämät view't. View't sijaitsevat *resources/views*-kansiossa ja niihin ohjataan *views*-kansion mukaisesti. Esimerkiksi *resources/views/welcome.blade.php*-view'hun ohjataan kontrollerista tai *routes.php*-tiedoston reiteistä koodilla *return view('welcome').*

Routes.php-tiedostossa on määritetty */home*-reitti *MainControllerin index*-funktioon, joka ohjaa yksinkertaiseen *home*-view'hun. View-tiedoston luominen onnistuu luomalla *home.blade.php*-tiedosto *resources/views*-kansioon. Home-view käyttää pohjana kuvassa 11 näkyvää *master.blade.php*-tiedostoa.

```
1  <!doctype html>
2  <html>
3  <head>
4      @include('shared.head')
5  </head>
6  <body>
7  <div class="container">
8
9      @section('navbar')
10     <div class="navbar">
11         @include('shared.navbar')
12     </div>
13     @show
14
15     @section('content')
16     <div id="main" class="content">
17         <h1>Home</h1>
18     </div>
19     @show
20
21 </div>
22
23 </body>
24 </html>
```

Kuva 11. *Master.blade.php*-tiedoston sisältö.

Kun tiedostonimi loppuu *blade.php*:hen, käytetään Laravelin Blade-sapluunamoottoria. View'ssa on käytetty Bladea useassa kohdassa, ja sisältö on yksinkertainen HTML-rakenne *head*- ja *body*-osilla. Kuvan 11 riveillä 4 ja 10 sisällytetään *resources/views/shared*-kansiosta *head.blade.php*- ja *navbar.blade.php*-tiedostot. Kuvan 11 riveillä 9 ja 15 aloitetaan *navbar*- ja *content*-blade osiot, joihin on määritetty oletussisältöä, jota muut view't voivat käyttää tai laajentaa.

Master.bladeen sisällytetty *head.blade.php* sisältää muun muassa kuvan 12 mukaisen merkistökoodauksen ja sisällön venyttämisen laitteen näytön leveyden mukaisesti määrittävät lauseet sekä CSS- ja JavaScript-tiedostojen sisällyttämistä *public/assets*-kansiosta.

```
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title>@yield('pagetitle', '')</title>

<link rel="stylesheet" type="text/css" href="{{ URL::asset('assets/css
/bootstrap.min.css') }}" />
<link rel="stylesheet" type="text/css" href="{{ URL::asset('assets/css
/main.css') }}" />

<script src="{{ URL::asset('assets/js/lib/jquery-2.1.4.min.js') }}" ><
/script>
<script src="{{ URL::asset('assets/js/main.js') }}" ></script>
```

Kuva 12. *Head.php*-tiedosto blade-osilla.

Tyylitiedostojen ja skriptitiedostojen sisällyttäminen tehdään lisäämällä *link*- ja *script*-tageihin tiedostojen sijainnit URL-luokan *asset*-funktiolla, joka luo tiedostoihin URL:n, jossa on verkkosivun URL. Web-sivun URL:ään on lisätty tiedostojen sijainnit siten, että ne latautuvat oikein. Tyylitiedostot ja skriptitiedostot sekä mahdolliset yleisesti käytetyt kuvat sisältävän public/assets-kansion rakenne näkyy kuvassa 13.

```
▾ assets/
  ▾ css/
      bootstrap.min.css
      main.css
  ▸ fonts/
  ▾ js/
    ▾ lib/
        jquery-2.1.4.min.js
      main.js
```

Kuva 13. *Public/assets*-kansion rakenne.

Main view'hun sisällytetty *navbar.blade.php*-tiedosto sisältää Bootstrap-kirjastolla toteutetun navigaatiopalkin, joka pysyy näkyvillä, vaikka sivun sisältö jatkuu selattavana yli näytön alareunan. Kuvassa 14 näkyy Bootstrap-navigaatiopalkin HTML-määritys.

```
<!-- Fixed navbar -->
<nav class="navbar navbar-default navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="#">Project</a>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
          <ul class="nav navbar-nav">
          </ul>
          <ul class="nav navbar-nav navbar-right">
          </ul>
        </div>
      </div>
</nav>
```

Kuva 14. Bootstrap-kirjastolla toteutettu navigaatiopalkki.

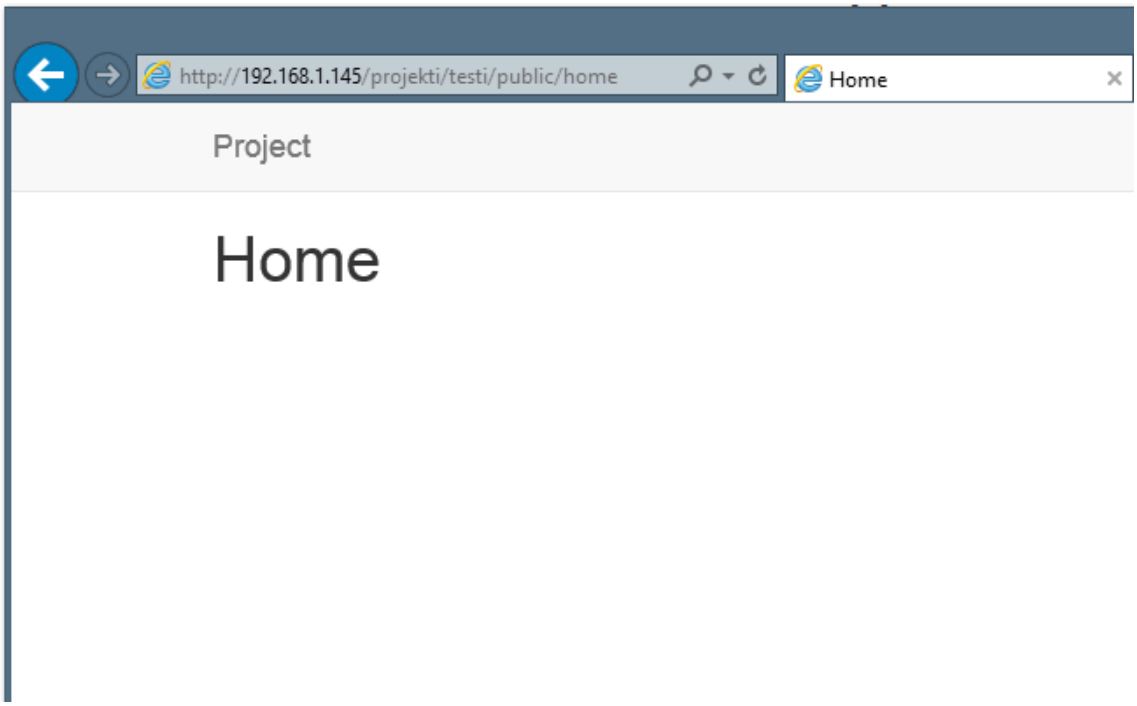
Navigaatiopalkki on yksinkertainen esimerkki, mutta sitä voi laajentaa tarpeen tullen Bootstrapissa määritetyillä elementtien tyyleillä. Eri view'seissä voi käyttää eri navigaatiopalkkeja, ja navigaation saa halutessa mukautumaan kontrollerista tulevan tiedon mukaan.

Home view'ssa käytetään ja laajennetaan *master.blade.php*:tä. Home view'ssa määritetään *master.blade.php*:hen sisällytetyn *head.blade.php*-tiedoston *title*-osaan oma title 'Home'. Home view'n *navbar*-osa sisällytetään muuttumattomana *master.blade.php*:stä ja *content*-osaa ei käytetä vaan määritetään oma otsikkoelementti. Kuvassa 15 nähdään, miten *master.blade.php*-tiedostoa on käytetty home view'n määrittelevässä *home.blade.php*-tiedostossa.

```
@extends('layouts.master')

@section('pagetitle', 'Home')

@section('navbar')
    @parent
@endsection


@section('content')
    <div id="main" class="content">
        <h1>Home</h1>
    </div>
@endsection
```

Kuva 15. *Home.blade.php*-tiedoston sisältö.

Selaimella */home* URL:iin tullessaan käyttäjä ohjautuu kuvan 16 mukaiselle sivulle, jossa on navigaatiopalkki ja ensimmäisen tason otsikko.



Kuva 16. Yksinkertainen Bootstrap-sivu.

Blade-sapluunamoottorin käyttäminen view'eissä mahdollistaa view'ien laajentamisen, kun käytetään ja laajennetaan aikaisemmin määritettyjä view'ejä. Laajennettavaksi pääsapluunaksi voidaan siis tehdä lähes tyhjä *master.blade.php,* jossa on määriteltynä HTML-rakenne *head*- ja *body*-elementeillä. Esimerkiksi *body*-elementtiin voi lisätä blademäärityksiä, joita voi käyttää ja laajentaa muissa sapluunoissa ja jotka jatkavat master.bladea.

Seuraavaksi lisätään tiedosto kuvassa 17 näkyvälle *page*-view'lle. *MainController.php*-tiedoston *MainController*-kontrollerissa haettiin tietokannasta URL:stä saadun parametrin mukainen tietue, jonka tiedot lähetetään edelleen *page*-view'lle. *Page*-view laajentaa home view'n tavoin *master.blade.php*:tä kuvan 17 mukaan.

```
1  @extends('layouts.master')
2
3  @section('pagetitle', $paged->title)
4
5  @section('navbar')
6  @endsection
7
8
9  @section('content')
10     <div id="main" class="content">
11         <div class="jumbotron">
12             <h1>{{ $paged->url }}</h1>
13
14             <p>{{ $paged->description }}</p>
15         </div>
16     </div>
17 @endsection
```

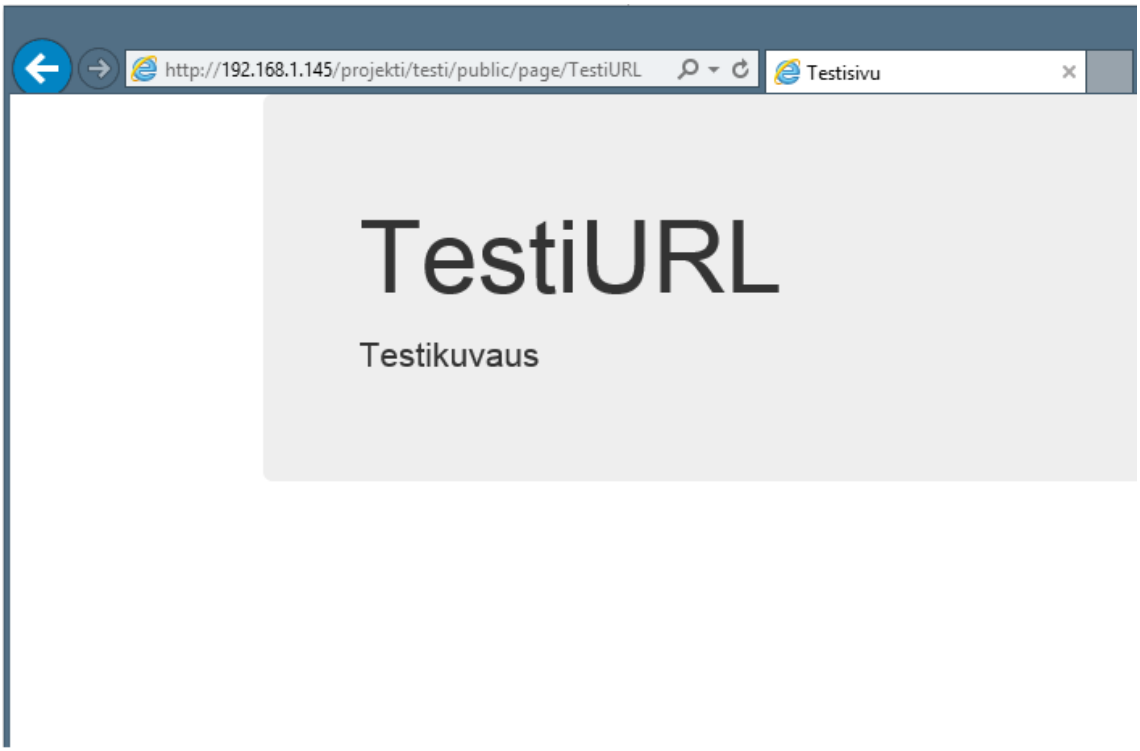
Kuva 17. *Page.blade.php*-tiedoston sisältö.

Page view'ssa asetetaan head.blade.php:n pagetitle-kohtaan kontrollerista saatu muuttuja. Navigointipalkki jätetään pois sisällyttämällä navbar-osio ilman *@parent*-komentoa, jolloin navbar-osio kirjoitetaan yli tyhjänä. Page-view'n content-osio korvataan page.blade.php:n omalla sisällöllä, jossa näytetään kontrollerilta saadut tiedot. Kun selaimen osoitekenttään syöttää /page/TestiURL, haetaan tietokannasta kuvassa 18 näkyvä rivi.

id	url	name	description	settings	created_at	updated_at
4	TestiURL	Testisivu	Testikuvaus	{}	0000-00-00 00:00:00	0000-00-00 00:00:00

Kuva 18. MainControllerin showPage-funktiossa haettava rivi.

MainController-kontrolleri hakee rivin, asettaa arvot muuttujiin ja palauttaa selaimelle vastauksena *page*-view'n sekä antaa view'lle kontrollerikoodissa määritetyt muuttujat. View'ssa haetaan *master.blade.php*-tiedostosta HTML-perusrakenne ja asetetaan kontrollerilta saadut muuttujat paikoilleen bladen kaarisulkumerkintää käyttämällä. Kuvassa 19 näkyy *page*-view selainikkunassa.



Kuva 19. *Page*-view selainikkunassa.

Tässä esimerkissä on pelkästään yksinkertainen sivu. Tehdyt muutokset voi nyt kommitoida paikalliseen Git-versiorekisteriin komennolla *$ git commit -am "pohja alulle"* ja syöttää kommitoinnin yhteydessä paikalliseen versiorekisteriin luotu tilannevedos *git@localhost*-käyttäjän repositorioon *$ git push*-komennolla.

3.3 Toiminnallisuuksien lisääminen

Esimerkkiprojektia laajennettiin tutustumalla tiedon lisäämiseen tietokantaan ja JavaScript-koodin käyttöön *page*-view'ssa. Lisätään toiminta, jolla sivuja saa lisättyä tietokantaan. Tätä varten lisätään uusi reitti *routes.php*-tiedostoon. Kuvassa 20 näkyy *routes.php*:hen lisätty reitti *MainController.php*-tiedoston funktioon *createNewPage*.

```
23 Route::get('/home/addpage', 'MainController@addPage');
24 Route::post('/home/addpage', 'MainController@createPage');
```

Kuva 20. *Routes.php*:hen lisätyt reitit sivun lisäämistä käsitteleviin funktioihin.

Kuvassa 20 näkyy kaksi reittiä kahteen eri MainControllerin funktioon. */home/addpage*-urliin saapuessa ohjaudutaan *addPage*-funktioon, joka vuorostaan ohjaa addpage-view'hun. View'ssa on lomake, jossa on viisi syöttökenttää. Syöttökentät ja kentät sisältävä lomake toteutettiin käyttäen Bootstrapin luokkia. Kuvassa 21 näkyy *addpage*-view'n rakenne.

```
@extends('layouts.master')

@section('pagetitle', 'Add new page')

@section('navbar')
    @parent
@endsection

@section('content')
    <div id="main" class="content">
    <div class="col-md-6 col-offset-3">
        <h1>Add page</h1>

        <form action="{{url('/home/addpage')}}" method="POST">
          {{ csrf_field() }}
          <div class="form-group">
            <label for="pageurl"></label>
            <input type="text" class="form-control" name="pageurl" placeholder="Page URL">
          </div>
          <div class="form-group">
            <label for="pagename">Page name</label>
            <input type="text" class="form-control" name="pagename" placeholder="Page name">
          </div>
          <div class="form-group">
            <label for="pagedescription">Page description</label>
            <input type="text" class="form-control" name="pagedescription" placeholder="Page description">
          </div>
          <div class="form-group">
            <label for="pagesettings">Page settings</label>
            <input type="text" class="form-control" name="pagesettings" placeholder="Page settings">
          </div>

          <button type="submit" class="btn btn-default">Add new page</button>
        </form>
    </div>
    </div>
@endsection
```

Kuva 21. *Addpage*-view'n sisältö *addpage.blade.php*-tiedostossa.

Kuvassa 21 näkyy, kuinka *addpage*-view'ssa laajennetaan luvussa 3.2 luotua *master*-view'tä, määritellään sivun otsikoksi *'Add new page'*, käytetään *master*-view'n mukaisesti *navbar*-osiossa navigointipalkkia sekä määritetään lomakkeen rakenne *content*-osiossa.

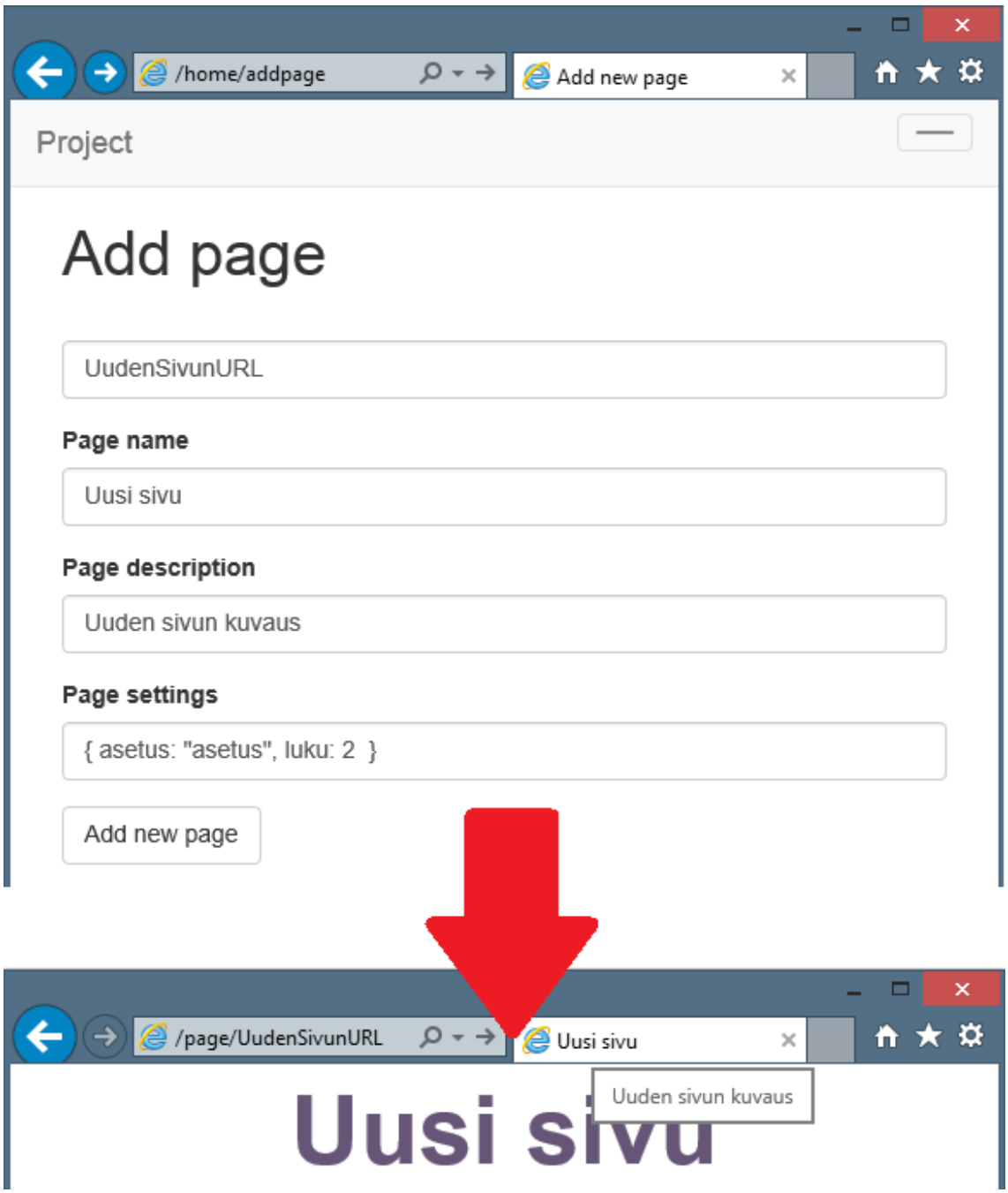
Lomakkeen parametreihin on lisätty *action*-attribuutin sisään bladen kaarisulkeisiin *url*-funktio, jolle annetaan parametriksi *'/home/addpage'. Method*-parametrille on annettu *post*-arvo. Lomakkeen nappia painaessa ohjataan lomakkeen sisältö *POST*-muuttujassa pyynnön mukana */home/addpage*-urliin. Tämä ohjataan kuvan 20 mukaisen *routes.php*-tiedoston reitin mukaisesti kuvan 22 mukaiseen *MainController*-kontrollerin *createPage*-funktioon.

```
49    public function createPage(Request $request){
50        Page::create([
51            'url' => $request->input('pageurl'),
52            'name' => $request->input('pagename'),
53            'description' => $request->input('pagedescription'),
54            'settings' => $request->input('pagesettings'),
55        ]);
56
57        return redirect('page/'.$request->input('pageurl'));
58    }
```

Kuva 22. *MainController*-kontrollerin *createPage*-funktio.

Kuvassa 22 olevan *createPage*-funktion parametriksi on määritetty *$request*-muuttuja. Luodaan *Page*-mallin kautta funktioilla uusi tietue tietokantaan ja palautetaan uudelleenohjaus *redirect*-funktiota käyttäen. Tietokantaan lisätään tietoa funktioon *$request*-muuttujan kautta *addPage*-view'n lomakkeen syöttökenttien sisältöä. Sivunluontifunktioon voidaan lisätä vielä varmistus ennen uudelleenohjausta, ettei kannassa jo ole syötetyllä urlilla olevaa riviä.

Kun osoitteeseen */home/addpage* siirrytään, täytetään *view*-tiedostossa olevan lomakkeen syöttökentät. Tiedot lähetetään lomakkeen painikkeella eteenpäin kontrolleriin. Kontrollerissa lisätään tiedot *Page*-mallia käyttäen tietokantaan ja uudelleenohjataan selain sivulle, joka ohjaa takaisin kontrolleriin, jossa funktio ohjaa view'hun. View'ssa näytetään tietokantaan lisätyt tiedot. Kuvassa 23 on havainnollistettu, mitä selainikkunassa tapahtuu edellä mainitun ketjun aikana.



Kuva 23. Selaimessa nähtävät vaiheet uutta sivua lisättäessä.

Kuvasta 23 näkee tapahtumien kulun selainikkunassa. Selaimessa havaittavat asiat ja palvelimessa tapahtuvat asiat eroavat usein toisistaan vaiheiden määrän ja esitystavan osalta. Tehdyt vaiheet voi jälleen kommitoida ja lähettää edelleen Git-versiorekisteriin.

Esimerkkiverkkosivu tarvitsee vielä toiminnallisuuksia, jotka voi toteuttaa esimerkiksi JavaScriptiä käyttämällä. Sivun asetuksien mukana voi tallentaa asetuksia esimerkiksi

JSON-muotoisena, jolloin asetusten ja toiminnallisuuksien käyttöönotto JavaScript-koodissa käy käden käänteessä.

3.4 Esimerkkiverkkosivun jatkokehityksestä

Esimerkkiverkkosivua voi laajentaa lähes mihin suuntaan tahansa käyttämällä erilaisia tarjolla olevia kirjastoja ja työkaluja. Verkkosivuun voi lisätä käyttäjäntunnistuksen ja käyttäjän luonnin sekä käyttää Laravel-ohjelmakehyksen middleware-toimintoa, jolla käyttäjän tunnistuksen saa yhdistettyä ennen reittien uudelleenohjautumistoimintoa suoritettavaksi.

Verkkosivun kehitystä uutis-, video- tai kuvapalveluksi voidaan jatkaa ottamalla käyttöön ohjelmakehyksen tarjoamia tiedostonhallintaan liittyviä kirjastoja sekä käyttämällä ohjelmakehyksen ulkoisia kirjastoja ja työkaluja. Verkkosivulle voidaan ottaa käyttöön Laravelin Nodejs:llä toimiva Elixir-järjestelmä, jolla voi muun muassa kääntää CoffeeScript-skripti selaimen ymmärtämäksi JavaScript-skriptiksi sekä tiivistää skripti pienempään kokoon ja yhdistää useammat skriptitiedostot yhdeksi ladattavaksi tiedostoksi. Tämä vähentää tiedonsiirron ja pyyntöjen määrää.

4 Yhteenveto

Opinnäytetyössä perehdyttiin web-kehitykseen käytettäviin tekniikoihin, työkaluhin ja kirjastoihin. Työssä oli myös tarkoitus luoda yksinkertainen verkkosivu käyttämällä osaa tekniikoita, työkaluja ja kirjastoja. Peruspohjaa luodessa oli myös tarkoitus tutustua Git-versionhallintaohjelmaan, jonka avulla useampi tekijä voi laajentaa ja kehittää samaa peruspohjaa.

Opinnäytetyössä luotu yksinkertainen verkkosivu toimii Laravel-ohjelmakehyksen päällä, ja valtaosa web-sivun toiminnoista tuli Laravelin mukana. Palvelimen puolella toimivat toiminnot toetutettiin työssä esitetyssä esimerkissä PHP:llä ja Laravelillä toteutettu. Selaimessa käyttäjälle näkyvässä osassa käytettiin yksinkertaista HTML-merkintäkieltä, CSS-tyylikieltä ja Bootstrap-kirjastoa.

Työssä esimerkkinä tehdyn verkkosivun kehitystä voi jatkaa esimerkiksi uutis-, video- tai kuvapalveluksi. Yksinkertaista verkkosivua voidaan laajentaa ottamalla käyttöön Laravel-ohjelmakehyksen tarjoamia tiedostonhallintaan liittyviä kirjastoja ja käyttämällä ohjelmakehyksen ulkoisia kirjastoja ja työkaluja.

Lähteet

1 PHP: What is PHP? - Manual. Verkkodokumentti. PHP. <https://secure.php.net/manual/en/intro-whatis.php>. Luettu 7.11.2015.

2 PHP: What can PHP do? - Manual. Verkkodokumentti. PHP. <https://secure.php.net/manual/en/intro-whatcando.php>. Luettu 7.11.2015.

3 MySQL 5.1 Reference Manual 1.3.1 What is MySQL?. Verkkodokumentti. MySQL. <http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html>. Luettu 8.11.2015.

4 Introduction to HTML - Web developer guide | MDN. Verkkodokumentti. Mozilla Foundation. <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Introduction>. Luettu 8.11.2015.

5 What is CSS - Web developer guide | MDN. Verkkodokumentti. Mozilla Foundation. <https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_Started/What_is_CSS>. Luettu 8.11.2015.

6 About JavaScript - JavaScript | MDN. Verkkodokumentti. Mozilla Foundation. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>. Luettu 8.11.2015.

7 Chacon, Scott & Straub, Ben. 2014. Pro Git. Spring Science+Business Media.

8 Git - Git Basics. Verkkodokumentti. Git. <http://git-scm.com/book/en/v2/Getting-Started-Git-Basics>. Luettu 7.11.2015.

9 Pitt, Chris & Mostrey, Wim. 2012. Pro PHP MVC. Apress.

10 Application Structure - Laravel - The PHP Framework For Web Artisans. Verkkodokumentti. Laravel. <http://laravel.com/docs/5.1/structure>. Luettu 8.11.2015.

11 Installation - Laravel - The PHP Framework For Web Artisans. Verkkodokumentti. Laravel. <http://laravel.com/docs/5.1>. Luettu 31.10.2015.

12 Artisan Console. Verkkodokumentti. Laravel. <http://laravel.com/docs/5.1/artisan>. Luettu 5.11.2015.