SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

SIVUSTON HALLINTAPANEELIN TOTEUTUS LARAVEL 8 -OHJELMISTOKEHYKSELLÄ

TEKIJÄ/T: Jimi Holopainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Jimi Holopainen	
Työn nimi Sivuston hallintapaneelin toteutus Laravel 8 -ohjelmistokehyksellä	
Päiväys 1.6.2021	Sivumäärä/Liitteet 30
Toimeksiantaja/Yhteistyökumppani(t) Traakkoni Oy	

Tiivistelmä

Tässä opinnäytetyössä toteutettiin käyttöliittymä ratsastuskoulun kotisivujen sisällön muokkausta varten tiettyjen reunaehtojen mukaisesti. Sivuston käyttäjät ovat pääasiassa henkilöitä, jotka tietävät tietotekniikasta vähän, joten käyttöliittymä tehtiin mahdollisimman yksinkertaiseksi.

Käyttöliittymä toteutettiin hyödyntäen Laravel 8 -ohjelmistokehystä ja sen tarjoamia työkaluja. Ohjelmointikielenä käytettiin PHP:tä. Lisäksi opinnäytetyössä täytyi ottaa huomioon erilaisia käyttöliittymään, käytettävyyteen ja käyttäjiin liittyviä asioita ja ohjeita ja pohtia niiden vaikutusta sekä yleisellä tasolla, että myös tässä projektissa.

Lopputuloksena syntyi käyttöliittymä, jolla muokataan Single Page Application -tyylillä toimivia ratsastuskoulun kotisivuja. Käyttöliittymässä on huomioitu sivuston pääasiallinen käyttäjäryhmä hyödyntäen erilaisia käytettävyyteen liittyviä ohjeita ja sääntöjä. Toteutuksen eri vaiheet dokumentoitiin ja ne esitellään tässä työssä.

Avainsanat

Käyttöliittymä, käytettävyys, Laravel, ohjelmistokehys, PHP, REST

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author(s) Jimi Holopainen	
Title of Thesis Implementation of Website's Control Panel with Laravel 8	
Date 1 June 2021	Pages/Appendices 30
Client Organisation /Partners Traakkoni Oy	

Abstract

The purpose of this thesis was to implement a user interface (UI) with some specific conditions to edit the contents of the home page of a riding school. The website is aimed for users that know little about information technology, so the user interface was created as simple as possible.

The user interface was implemented by using Laravel 8 software framework and the tools it offered. The used programming language was PHP. In addition, some guides and rules regarding the user interfaces, usability and users were taken into consideration in this thesis, both in general and how they affected specifically this project.

The final outcome was a user interface for editing the contents of the homepage of riding school that works as a Single Page Application. The main group of users was noted by using the guides and rules of usability in the user interface. Different phases of implementation are documented and reviewed in this thesis.

Keywords
User interface, usability, Laravel, framework, PHP, REST

SISÄLLYS

LYHENTEET JA KÄSITTEET

Artisan = Laravelin komentorivi, toimii apuna sovelluskehityksessä

Asiakas (client) = Sovellus, jolla otetaan yhteys palvelimeen

Blade = Laravelin näkymäkomponentti MVC-arkkitehtuurissa, sisältää käyttöliittymän logiikan

CRUD (Create, Read, Update, Delete) = Neljä tavallisinta operaatiota REST-arkkitehtuurissa

Composer = PHP:n paketinhallintatyökalu

Fortify = Laravelin käyttämä autentikointityökalu

Git = Komentorivi, jonka avulla versionhallinta suoritetaan

GitHub = Verkkosivusto, johon Git-versionhallintaa käyttävät projektit tallennetaan

Graafinen käyttölittymä (GUI, Graphical User Interface) = laitteen käyttö kuvan ja tekstin avulla

Heuristiikka = Oikeaksi todettu menetelmä ongelmanratkaisuun

HTML (Hypertext Markup Language) = Ohjelmointikieli, jolla nettisivut kirjoitetaan

HTTP (Hypertext Transfer Protocol) = Tiedonsiirtoon käytettävä protokolla

Käytettävyys = Tuotteen tai palvelun helppokäyttöisyyden määrittävä ominaisuus

Käyttöliittymä (UI, User Interface) = Laitteen osa, jonka avulla käyttäjä ja laite ovat vuorovaikutuksessa

Laravel = PHP:n web-ohjelmistokehys

Migraatio = Laravelin tietokantojen versionhallinta

MVC (Model-View-Controller) = Arkkitehtuurimalli, joka jakaa sovelluksen malliin, näkymään ja kontrolleriin

npm (Node Package Manager) = JavaScriptin paketinhallintatyökalu

Ohjelmistokehys = Tuote tai abstraktio, joka muodostaa tietokone- tai web-ohjelman rungon

Palvelin (server) = Ohjelmisto, joka tarjoaa palveluja asiakkaille

PHP (PHP: Hypertext Preprocessor) = HTML-upotettu komentokieli

REST (REpresentational State Transfer) = Arkkitehtuurimalli, perustuu HTTP-protokollaan

Visual Studio Code = Avoimen lähdekoodin ohjelmointiin tarkoitettu tekstieditori

WAMP (Windows, Apache, MySQL, PHP) = Kokoelma ohjelmistoja, joilla web-sovelluskehitys voidaan toteuttaa

1 JOHDANTO

Tämän opinnäytetyön aiheena oli toteuttaa käyttöliittymä, jolla pystytään muokkaamaan verkkosivun sisältöä, eli toteutetaan hallintapaneeli, josta muokkaukset voidaan tehdä suoraan. Aiheen tarjosi Traakkoni Oy ja hallintapaneeli tehdään Savisaari-ratsastuskoulun kotisivuille, joka toimii Single Page Application -tyylillä. Hallintapaneelissa oli tarkoitus pystyä muokkaamaan hinnastoa, hevosten tietoja sekä kesän kursseja ja muodostaa kursseista PDF-sivu.

Sivuston pääasiallinen käyttäjäryhmä koostuu henkilöistä, jotka eivät käytä tietotekniikkaa kovinkaan paljon. Tämä asettaa merkittäviä rajoituksia käyttöliittymän toteutukseen, joten suunnittelussa ja toteutuksessa joudutaan huomioimaan erilaisia käyttöliittymään, käyttäjiin ja käytettävyyteen liittyviä ohjeita ja sääntöjä. Kehityksessä joutuu koko ajan pohtimaan kyseisen käyttäjäryhmän näkökulmaa. Käytännössä se tarkoittaa mahdollisimman yksinkertaista ulkoasua mahdollisimman selkeillä viesteillä, elementtien asetteluilla ja vähäisillä toiminnoilla. Käytettävyys on äärimmäisen tärkeä ominaisuus mille tahansa sovellukselle tai palvelulle, koska se toimii helppokäyttöisyyden mittarina ja määrittää hyvin pitkälti sen, tullaanko kyseistä sovellusta tai palvelua käyttämään jatkossakin.

Opinnäytetyössä on teoriaosuus, jossa pohditaan käyttöliittymiä ja käytettävyyden eri näkökulmia sekä esitellään käytettyjä tekniikoita itse toteutuksen osalta. Itse toteutus käydään läpi tekstien, kuvakaappausten ja koodiesimerkkien avulla ja pohditaan erilaisten käytettävyysohjeiden hyödyntämistä. Ohjelmoinnissa käytettiin Laravel 8 -ohjelmistokehystä, joka toimii PHP-ohjelmointikielellä, ja sen eri ominaisuuksia tarkastellaan.

2 TEORIA

2.1 Käyttöliittymä

Käyttöliittymä (UI) on laitteen osa, jonka avulla ihminen ja tietokone ovat vuorovaikutuksessa ja kommunikoinnissa keskenään. Nämä osat voivat olla esimerkiksi näyttöjä, näppäimistöjä, hiiriä ja työpöytien esiintymisiä näytöllä. Käyttöliittymä on myös väline, jonka avulla käyttäjä on vuorovaikutuksessa sovelluksen tai verkkosivuston kanssa. Kasvava tarve liiketoiminnalle web- ja mobiilisovelluksissa on saanut monet yritykset keskittymään käyttöliittymään saadakseen parannettua käyttäjien kokonaisvaltaista käyttäjäkokemusta. (Churchville, 2019)

Erilaisia käyttöliittymätyyppejä ovat esimerkiksi:

- Graafinen käyttöliittymä (GUI)
- Komentoliittymä eli komentorivi (CLI)
- Puhekäyttöliittymä (VUI)
- Kosketuskäyttöliittymä

Käyttöliittymät ovat kehittyneet vuosien saatossa hyvinkin paljon. Ensimmäisissä tietokoneissa käyttöliittymä rajoittui vain muutamaan nappiin käyttäjän konsolilla. Sitten kehitettiin komentorivi, joka oli aluksi vain yksi rivi käyttäjän syötteille, jonka jälkeen tulivat valikkotyyliset käyttöliittymät, jossa vaihtoehdot olivat listattuina. Lopulta saapuivat graafiset käyttöliittymät, jotka etenkin Microsoft on standardoinut Windows-käyttöjärjestelmissä. Graafiseen käyttöliittymään kuuluvat esimerkiksi ikkunat, painikkeet, vierityspalkit ja kuvakkeet. Erilaisen datan ja multimedian lisääntyessä käyttöliittymissä äänestä, puheesta, liikkeestä ja virtuaalitodellisuudesta on tullut graafinen käyttöliittymä useissa sovelluksissa. Lisäksi mobiilisovellusten suosio on vaikuttanut käyttöliittymiin siten, että älypuhelinten ja tablettien näytöillä toimivat interaktiiviset rajapinnat ja jotkin erityistoiminnot, kuten kosketusohjaus. (Churchville, 2019)

2.2 Käyttäjäkeskeinen suunnittelu

Käyttäjäkeskeinen suunnittelu (UCD) on toistuva suunnitteluprosessi, jossa suunnittelijat keskittyvät käyttäjiin ja heidän tarpeisiinsa jokaisessa suunnittelun vaiheessa. Käyttäjät ovat mukana suunnitteluprosessissa esimerkiksi tutkimusten ja erilaisten suunnittelutekniikoiden kautta, jotta pystytään kehittämään helposti käytettäviä tuotteita. (Interaction Design Foundation)

Käyttäjätutkimukseen kuuluvat loppukäyttäjien tarpeiden, ajatusten ja tunnetilojen ymmärtäminen, minkä lisäksi täytyy tiedostaa todelliset olosuhteet tuotteen tai palvelun käytölle. Näiden seikkojen selvittäminen tehostaa kehitystyön ohjausta ja priorisointia, eli kehitystyöstä voidaan pudottaa pois käyttäjien mielestä tarpeettomia ominaisuuksia, mikä puolestaan vauhdittaa projektin aikataulua sekä pienentää budjettia. Kun kehitystyö keskittyy käyttäjien haluamiin ominaisuuksiin, he ovat huomattavasti tyytyväisempiä lopputulokseen, mikä puolestaan lisää tuotteen tai brändin asiakasuskollisuutta sekä johtaa myynnin kasvuun. (Vaittinen, 2019)

Suunnitteluprosessille ei ole yhtä oikeaa tai tiettyä kaavaa, vaan kaikki riippuu tilanteesta ja käyttäjän haluista. Prosessissa on kuitenkin joitain yleisiä vaiheita, jotka ainakin jossain määrin käydään

läpi prosessin aikana, mutta tilanteesta riippuen vaihtelevassa järjestyksessä. Tällaisia vaiheita voivat olla

- Käyttökontekstin määrittäminen. Ketkä käyttävät tuotetta tai palvelua, mitä asiaa varten ja millaisissa olosuhteissa?
- Vaatimusten määrittäminen. Liiketoiminnan vaatimukset tai käyttäjävaatimukset, jotka täytyy saavuttaa, että tuote olisi onnistunut.
- Suunnitteluratkaisujen toteutus. Tämä voidaan toteuttaa useassa eri vaiheessa, kun kehitystyö etenee karkeasta ideasta valmiiseen malliin.
- Arviointi. Toteutetaan käytettävyystestinä loppukäyttäjillä, jonka jälkeen voidaan tarpeen vaatiessa siirtyä tai palata johonkin muuhun suunnittelun vaiheeseen. (Usability.gov)

Koko suunnitteluprosessi tai sen tietyt vaiheet voidaan myös sisällyttää osana muita kehitystyömetodeja, kuten vesiputousmalli tai ketterä ohjelmistokehitys. Itse kehittäjä huomioi loppukäyttäjien vaatimusten lisäksi kehitystyön aiheen, projektitiimin, aikataulun sekä kehitystyöympäristön. Näiden avulla pystytään tekemään johtopäätökset suoritettavista tehtävistä ja niiden järjestyksestä. (Usability.gov)

Tässä opinnäytetyössä loppukäyttäjät eivät olleet niinkään oleellisesti mukana itse kehitystyötä, mutta projektin speksit määriteltiin siten, että käyttöliittymästä tulee mahdollisimman yksinkertainen ja loppukäyttäjät riippumatta heidän tietoteknisen osaamisen tasosta pystyvät tekemään halutut toimenpiteet käyttöliittymässä ilman suurempia ongelmia. Kehitystyössä täytyi pitää mielessä ja pohtia, että pystyykö lähestulkoon kuka tahansa tekemään halutun toiminnon vaivattomasti.

2.3 Käyttäjäkokemus

Käyttäjäkokemus (UX) viittaa mihin tahansa vuorovaikutukseen käyttäjän ja tuotteen tai palvelun välillä. Käyttäjäkokemusta pohdittaessa tulee huomioida sitä muovaavat elementit, eli miltä loppukäyttäjästä tuntuu ja miten helppoa hänen on suorittaa halutut tehtävät. Tämä voi olla mitä tahansa esimerkiksi tuotteen tuntumisessa kädessä ja verkkokaupan ostotapahtuman välillä. (Stevens, 2021)

Käyttäjäkokemuksen suunnittelu liitetään usein käyttöliittymän suunnitteluun ja käytettävyyteen. Ne ovat kuitenkin osa käyttäjäkokemusta, joka kattaa myös muita osa-alueita. Suunnittelussa ei oteta huomioon pelkästään tuotteen tai palvelun käyttöä, vaan myös tehokkuutta ja sitä, että käyttäjälle jää positiivinen tunne. Ei ole yhtä määritelmää hyvälle käyttäjäkokemukselle, vaan se riippuu tietyn käyttäjän tarpeista tietyssä asiayhteydessä, jossa hän käyttää tuotetta. Käyttäjäkokemuksen suunnittelu on käyttäjäkeskeistä, mutta laajemmassa mittakaavassa siten, että keskitytään myös loppukäyttäjien tunnetasoon pelkän teknisen pohdinnan sijaan. Suunnittelun vaiheet voivat myös edetä vastaavalla tavalla kuin käyttäjäkeskeisessä suunnittelussa (Interaction Design Foundation). Käyttäjäkokemuksen eri näkökulmat tulevat esille, kun pohditaan käytettävyyteen liittyviä asioita.

2.4 Käytettävyys

Käytettävyys on ominaisuus, joka määrittelee, kuinka helppoa käyttöliittymän käyttö on. Se viittaa myös tapoihin parantaa helppokäyttöisyyttä suunnitteluprosessin aikana. Käytettävyys voidaan puolestaan määritellä viiden laatuominaisuuden avulla:

- Oppimiskyky. Miten helposti käyttäjät selviävät perustoiminnoista, kun he käyttävät tuotetta tai palvelua ensimmäisen kerran?
- Tehokkuus. Käyttäjien opittua tuotteen tai palvelun, kuinka nopeasti he pystyvät suorittamaan tehtävät?
- Muistettavuus. Käyttäjien palatessa käyttämään tuotetta tai palvelua tietyn ajan jälkeen, kuinka helposti he kykenevät palauttamaan toiminallisuuden mieleen?
- Virheet. Kuinka paljon virheitä käyttäjät tekevät, kuinka vakavia virheet ovat, ja kuinka helposti käyttäjät voivat palautua virheistä?
- Tyytyväisyys. Kuinka miellyttävää tuotteen tai palvelun käyttö on? (Nielsen, Nielsen Norman Group, 2012)

On myös muita tärkeitä laatuominaisuuksia, kuten hyöty, jonka avulla pohditaan tuotteen tai palvelun toimivuutta, eli tekeekö se, mitä käyttäjä haluaa. Käytettävyys ja hyöty yhdessä määrittävät, mikäli jokin on hyödyllistä. Jos tuote ei tee sitä mitä halutaan, sen helppokäyttöisyydellä ei ole merkitystä. Ei ole myöskään hyvä, että tuote voi teoriassa tehdä halutut asiat, mutta käytännössä se on mahdotonta johtuen käyttöliittymän monimutkaisuudesta. (Nielsen, Nielsen Norman Group, 2012)

Verkossa käytettävyys on ehdoton edellytys. Mikäli sivustoa on vaikea käyttää, kotisivut eivät kerro yrityksen tarjonnasta, sivustolle eksytään tai tiedon lukeminen ja vastausten saanti on vaikeaa, käyttäjät etsivät korvaavia vaihtoehtoja. Ihmiset haluavat suorittaa halutut tehtävät ja toiminnot nopeasti, eikä heillä ole runsaasti aikaa lukea sivuston käyttöohjeita tai ymmärtää rajapintaa. Sivustolta poistuminen on nopein ratkaisutapa. Yrityksen sisäisessä intranetissä käytettävyys liittyy suoraan työntekijöiden tuottavuuteen; jokainen hetki, jonka työntekijä viettää ihmetellen intranetin toimintaa tai vaikeita ohjeita, on käytännössä hukkaan heitettyä rahaa, jonka työntekijät saavat olemalla töissä saamatta töitä tehtyä. (Nielsen, Nielsen Norman Group, 2012)

Käytettävyyden parantaminen liittyy läheistesti käyttäjäkeskeiseen suunnitteluun, joten sen suunnitteluprosessin vaiheita voi hyödyntää, mutta etenkin arviointi on oleellinen osa käytettävyyden kannalta. Arviointi on käytettävyystesti, jonka suorittavat käyttäjät, ja se jaetaan kolmeen eri komponenttiin:

- Etsitään sopivat käyttäjät testiä varten, esimerkiksi verkkokaupan asiakkaat tai työntekijät intranettiä varten.
- Pyydetään käyttäjiä suorittamaan tiettyjä tehtäviä.
- Tehdään havaintoja, mitä käyttäjät tekevät, missä he onnistuvat ja missä heillä on vaikeuksia käyttöliittymän kanssa. Kerätään palaute käyttäjiltä. (Nielsen, Nielsen Norman Group, 2012)

On tärkeää antaa käyttäjien tehdä testi itsenäisesti ja antaa heidän ratkaista ongelmat itse, koska testin tulokset vääristyvät, mikäli kehittäjät antavat mitään ohjeita, esimerkiksi huomion kiinnittäminen johonkin kohtaan näytöllä. Kehittäjien ja suunnittelijoiden tulee kuitenkin seurata käyttäjien tekemistä tarkasti, sillä omat näköhavainnot ovat aina varmempia kuin käyttäjien sanat, vaikka myös käyttäjien palautteella on merkitystä. Yleensä viiden käyttäjän testi on riittävä havaitsemaan mahdolliset käytettävyysongelmat. Resurssien kannalta on parempi ajaa useita pieniä testejä kuin isoja ja kalliita tutkimuksia, jotta testien välissä voidaan kartoittaa ja ratkaista mahdolliset käytettävyysongelmat. Tarpeen vaatiessa voidaan palata muuhun vaiheeseen käyttäjäkeskeisessä suunnittelussa. Tällainen iteratiivinen suunnittelu on paras tapa parantaa käyttäjäkokemusta. Mitä enemmän versioita testataan käyttäjillä, sen parempi. (Nielsen, Nielsen Norman Group, 2012)

2.4.1 Käytettävyyden heuristiikat

Heuristinen arviointi on käytettävyyden tutkimisen tekniikka, jossa yksi tai useampi käytettävyyden asiantuntija arvioivat tuotteen tai palvelun käyttöliittymää hyödyntäen heuristiikkoja, jotka puolestaan ovat vakiintuneita, oikeaksi todettuja suuntaviivoja, jotka yleensä johtavat hyvään käyttöliittymän malliin. Heuristiikat eivät ole täysin kiveen hakattuja, vaan niistä voidaan tehdä erilaista tulkintaa tilanteesta riippuen. (Muniz)

Heuristisen arvioinnin hyödyntäminen voi olla nopeampi ja halvempi tapa parantaa tuotteen tai palvelun käytettävyyttä ennen käytettävyystestien toteuttamista. Näin saadaan paikallistettua jo joitain käytettävyysongelmia ennen testejä. Heuristinen arviointi löytää arvioilta 30 - 50 % kaikista käytettävyysongelmista, kun verrataan käyttäjätestiin. Heuristinen arviointi ei kuitenkaan ole tae täydelliseen käyttöliittymään. Asiantuntijatkin ovat vain ihmisiä, eivätkä välttämättä paikallista kaikkia käytettävyysongelmia, mitä loppukäyttäjät saattaisivat löytää käytettävyystestin aikana. (Muniz)

2.4.2 Nielsenin säännöt

Jakob Nielsen on kehittänyt kymmenen pääsääntöä, joita voidaan hyödyntää heuristisessa arvioinnissa. Ne ovat

1. Tuotteen tilan näkyvyys. Tuotteen tulisi aina pitää käyttäjät ajan tasalla siitä, mitä tapahtuu, asianmukaisen palautteen kanssa sopivassa ajassa. Kun käyttäjät tietävät tuotteen nykyisen tilan, he oppivat tulokset edellisestä vuorovaikutuksesta ja voivat määritellä seuraavat toimenpiteet. Käyttäjät luottavat tuotteeseen ja brändiin enemmän. (Nielsen, Nielsen Norman Group, 1994)
2. Tuotteen ja oikean elämän vastaavuus. Tuotteen tulisi käyttää samaa kieltä kuin käyttäjien, eli käytetään sanoja, fraaseja ja käsitteitä, jotka ovat tuttuja käyttäjille sisäisen jargonin sijaan. Informaation tulisi esiintyä luonnollisessa ja loogisessa järjestyksessä, mutta kuitenkin riippuen käyttäjistä. Esimerkiksi jokin käsite voi olla työntekijöille tuttu, mutta asiakkaat eivät ymmärrä sitä. (Nielsen, Nielsen Norman Group, 1994)
3. Käyttäjän kontrolli ja vapaus. Käyttäjät usein tekevät virheitä, joten he tarvitsevat selvästi merkityn poistumisen toiminnosta, jota ei haluttu tehdä. Kun tällainen peruuttaminen on mahdollista, käyttäjät pystyvät jatkamaan tuotteen hallintaa, eivätkä jää jumiin ja turhaudu. Lisäksi käyttäjät saavat lisää vapaudentunnetta ja itseluottamusta. (Nielsen, Nielsen Norman Group, 1994)

4. Yhteneväisyys ja standardit. Käyttäjien ei pitäisi ihmetellä, että tarkoittavatko eri sanat, tilanteet tai toiminnot samaa asiaa, vaan noudatetaan olemassa olevia käyttöstandardeja riippuen alustasta ja toimialasta. Yhteneväisyyden ylläpitämisen epäonnistuminen voi lisätä käyttäjän kognitiivista kuormitusta, koska heidän olisi pakko opetella jotain uutta. (Nielsen, Nielsen Norman Group, 1994)
5. Virheiden estäminen. Hyvät virheviestit ovat tärkeitä, mutta parhaat tuotteet estävät ongelmia tapahtumasta ollenkaan. Joko eliminoidaan virheherkät olosuhteet, tai annetaan palaute käyttäjälle ennen kuin he tekevät tarvittavat toimenpiteet. Virheitä voivat olla tahattomat lipsahdukset tai tietoiset virheet, jossa käyttäjä ei osaa tulkita tuotetta. (Nielsen, Nielsen Norman Group, 1994)
6. Tunnistaminen muistamisen sijaan. Minimoidaan käyttäjien muistamisen tarve tekemällä elementit, toiminnot ja vaihtoehdot näkyviksi. Käyttäjien ei pitäisi muistaa tuotteen käyttöä, kun siirrytään tuotteen osasta toiseen. Informaatio tuotteen käytölle tulisi olla näkyvillä ja saatavissa, kun sitä tarvitaan. Ihmisillä on rajoitettu lyhytkestoinen muisti, joten tunnistamista suosivat tuotteet helpottavat lähimuistin kuormaa. (Nielsen, Nielsen Norman Group, 1994)
7. Käytön joustavuus ja tehokkuus. Kokeneemmat käyttäjät voivat hyödyntää käyttöä nopeuttavia toimenpiteitä, joita uudet käyttävät eivät ole vielä huomanneet. Käyttäjät voivat räätälöidä heille sopivan tavan käyttää tuotetta. (Nielsen, Nielsen Norman Group, 1994)
8. Esteettinen ja minimalistinen ulkoasu. Tuotteiden ei pitäisi sisältää asiaankuulumatonta tai harvoin käytettyä tietoa. Jokainen ylimääräinen tiedon jyvänen tuotteessa kilpailee hyödyllisen tiedon kanssa ja näin tiedon suhteellinen näkyvyys vähenee. Tärkeää on keskittyä tuotteen olennaisiin tuotteen sisältöihin ja visuaalisiin elementteihin, jotka palvelevat käyttäjien päämääriä. (Nielsen, Nielsen Norman Group, 1994)
9. Käyttäjien auttaminen tunnistamaan, analysoimaan ja palautumaan virhetilanteista. Virheviestien tulisi esiintyä tavallisella kielellä virhekoodien sijaan, tarkasti ilmoittaen ongelman ja rakentavasti ehdottaen ratkaisua. Virheviestien tulisi myös visuaalisesti erottua, jotta käyttäjät huomaavat ne. (Nielsen, Nielsen Norman Group, 1994)
10. Opastus ja dokumentaatio. Paras tilanne olisi, kun tuote ei tarvitse lisäselityksiä. Voi kuitenkin olla tarpeellista tarjota dokumentaatio, jotta käyttäjät ymmärtävät haluttujen toimenpiteiden suorittamisen. Ohjeiden tulisi olla helposti löydettävissä ja keskittyä käyttäjien tehtäviin, lyhyesti ja ytimekkäästi. (Nielsen, Nielsen Norman Group, 1994)

Heuristiikat kehitettiin alun perin vuonna 1990, joita paranneltiin 1994. Nämä heuristiikat ovat täysin päteviä tänä päivänä ja todennäköisesti tulevat olemaan olennainen osa myös tulevaisuuden käyttöliittymiä. (Nielsen, Nielsen Norman Group, 1994)

2.4.3 Shneidermanin säännöt

Ben Shneiderman on kehittänyt kahdeksan kultaista sääntöä käyttöliittymän suunnitteluun. Nämä säännöt ovat yhtä lailla päteviä ja hyvin samankaltaisia kuin Nielsenin säännöt. Shneidermanin kahdeksan kultaista sääntöä ovat seuraavat:

1. Yhteneväisyyteen pyrkiminen. Käytetään tuttuja kuvakkeita, värejä, valikon rakenteita ja toimintoja, kun tehdään samantapainen tilanne tai toimenpide. Tiedon välityksen standardoiminen varmistaa, että käyttäjien ei tarvitse opetella uutta tapaa samalle toimenpiteelle. (Wong, 2020)
2. Oikotien tarjoaminen kokeneemmille käyttäjille. Kun tuotetta käytetään enemmän, tulee tarve suorittaa toimenpiteet nopeammin, joten tuotetta enemmän käyttäneet voivat operoida nopeammin ja vaivattomammin. (Wong, 2020)
3. Informatiivisen palautteen tarjoaminen. Käyttäjien tulisi tietää, missä he ovat ja mitä tapahtuu koko ajan. Jokaiselle toimenpiteelle pitäisi olla asiaankuuluva, helposti luettava palaute riittävän pienessä ajassa. Ei virhekoodia, vaan selkokieliset virheviestit. (Wong, 2020)
4. Dialogien suunnitteleminen siten, että ne johtavat lopputulokseen. Käyttäjiä ei saa pitää pimennossa, vaan heidän kuuluu saada tietää, mihin tehty toimenpide on johtanut. Käyttäjät arvostavat esimerkiksi ”Kiitos”-viestiä ja todistetta ostotapahtumasta, kun asiointi verkkokaupassa on suoritettu. (Wong, 2020)
5. Yksinkertainen virheenkäsittely. Järjestelmien tulisi olla niin virheettömiä kuin mahdollista, mutta mikäli väistämättömiä virheitä ilmaantuu, niin on tärkeää tarjota käyttäjille yksinkertaiset ja tarkat ohjeet, jotta ongelma saadaan ratkaistua mahdollisimman nopeasti ja vaivattomasti. (Wong, 2020)
6. Sallitaan toimintojen helppo peruminen. Käyttäjille tulee tarjota selvä tapa peruuttaa tehty toiminto. Perumisen tulisi olla mahdollista eri tilanteissa, kuten yksittäisen tai usean toimenpiteen jälkeen. (Wong, 2020)
7. Käyttäjän kontrollin tunteen tukeminen. Sallitaan käyttäjien olevan aloitteen tekijöitä ja annettaan tunne, että heillä on täysi hallinta tapahtumiin. Tämä saavutetaan, kun tuote käyttäytyy, kuten käyttäjät ovat odottaneet. (Wong, 2020)
8. Vähennetään käyttäjien lyhytkestoisen muistin kuormaa. Koska ihmisten lyhytkestoinen muisti on hyvin rajattua, käyttöliittymien tulee olla mahdollisimman yksinkertaisia ja oikeanlaisella tiedon järjestyksellä. Asian tunnistaminen on aina helpompaa kuin asian muistaminen, koska tunnistamiseen kuuluu vihjeiden havaitseminen, joka puolestaan auttaa merkityksellisen tiedon hyödyntämistä. Esimerkiksi koetilanteessa monivalintakysymykset ovat helpompia kuin avoimet kysymykset, koska monivalinnassa tarvitaan vain vastauksen tunnistaminen sen sijaan, että se pitäisi kaivaa muistin perukoista. (Wong, 2020)

3 KÄYTETYT MENETELMÄT JA TEKNIIKAT

3.1 WampServer

WampServer on Windowsin web-kehitysympäristö. Se sisältää kokoelman ohjelmistoja, joiden avulla kehitystyö toteutetaan. Kokoelmaan kuuluvat Apache, MySQL ja PHP. Apache on HTTP-palvelinohjelma, jonka avulla haluttu sivusto esitetään käyttäjälle. MySQL on puolestaan tietokanta, johon talletetaan haluttuja tietoja sivustolta, esimerkiksi käyttäjät ja aikaleimat tietyille tapahtumille. Hallinta tapahtuu kokoelman mukana tulevan graafisen käyttöliittymän kautta, eli tässä tapauksessa PhpMyAdminin. (WPBeginner)

WampServer on erinomainen kehitysympäristö paikalliseen testaamiseen, sillä tiedon siirtoa julkiseen verkkosivustoon ei tarvita. Tämä myös nopeuttaa kehitystyötä. Aloittelijat voivat myös opetella ja testata vaivattomammin. WampServeristä on myös muita versioita, kuten Linuxille tarkoitettu LAMP, joka sisältää samat työkalut kuin Windowsin versio. (WPBeginner)

3.2 Visual Studio Code

Visual Studio Code on kevyt, mutta tehokas lähdekoodin tekstieditori, joka toimii työpöytäsovelluksena ja on saatavilla Windowsille, MacOS:lle ja Linuxille. Mukaan kuuluu sisäänrakennettu tuki JavaScriptille, TypeScriptille ja Node.js:lle. Lisäksi usea ohjelmointikieli on tuettu, esimerkiksi C++, C#, Java, Python ja PHP. (Visual Studio Code)

3.3 Git

Git on versionhallintatyökalu, jonka avulla projektiin, esimerkiksi ohjelmistoon, tehdyt muutokset tallennetaan sille varattuun paikkaan, eli repositorioon. Näin kehittäjät pystyvät tekemään yhteistyötä lataamalla uusiman version ohjelmistosta, tekemällä halutut muutokset ja päivittämällä jälleen muokatun version repositorioon. Git toimii komentorivillä ja sen avulla tehdään itse toimenpiteet, kun taas GitHub on se paikka, johon projektit ladataan ja jossa repositoriot sijaitsevat. Jokaiselle projektille on oma repositorio, jolle puolestaan on oma linkki, jonka kautta repositoriota voi tarkastella. (Brown, 2019)

3.4 Laravel

Laravel on web-ohjelmistokehys tyylikkäällä syntaksilla. Laravel tarjoaa perusrakenteen ja lähtötilanteen sovelluksen kehittämiselle, joten kehittäjä voi keskittyä olennaiseen Laravelin hoitaessa yksityiskohdat taustalla. Laravel soveltuu sekä aloitteleville koodareille että kokeneille kehittäjille, johtuen sen monipuolisesta tarjonnasta ja laajasta dokumentaatiosta. Laravel on myös erittäin skaalautuva, siitä kertoo erilaisten Laravel-sovellusten pystyvän käsittelemään arviolta satoja miljoonia pyyntöjä kuukaudessa. (Otwell)

3.4.1 PHP

PHP (PHP: Hypertext Preprocessor) on HTML-upotettu komentokieli, jonka tavoitteena on mahdollistaa dynaamisesti generoitujen verkkosivujen toteutuksen nopeasti. Suuri osa PHP:n syntaksista on lainattu muista ohjelmointikielistä, kuten C:stä ja Javasta. (PHP)

3.4.2 REST

REST (REpresentational State Transfer) on arkkitehtuurimalli, jonka avulla eri tietokonejärjestelmät voivat kommunikoida keskenään verkossa. REST:ssä toteutetaan sekä palvelin (server), että asiakas (client), mutta toteutus voidaan hoitaa täysin erillisesti. Niiden ei siis tarvitse edes tietää toisistaan, joten tarvittavat muutokset voidaan tehdä jompaankumpaan ilman, että ne vaikuttaisivat toisen toimintaan. Käyttöliittymän erottaminen tiedon tallentamisen haasteista parantaa joustavuutta eri alustoilla ja skaalautuvuutta yksinkertaistamalla palvelimen komponentteja. (Codecademy)

Tilattomuus on oleellinen osa REST-arkkitehtuuria. Se tarkoittaa sitä, että palvelimen ei tarvitse tietää mitään asiakkaan tilasta ja toisinpäin. Näin molemmat pystyvät ymmärtämään mitä tahansa saapuvia viestejä. Tilattomuus saavutetaan hyödyntämällä resursseja komentojen sijaan. Resurssit kuvaavat mitä tahansa objektia, dokumenttia tai asiaa, joka voidaan tallentaa tai lähettää muihin palveluihin. Tästä johtuen REST ei riipu käyttöliittymien toteutuksesta ja eri komponentteja voidaan päivittää vaikuttamatta koko järjestelmän toimivuuteen. (Codecademy)

REST-arkkitehtuurissa asiakkaat lähettävät palvelimelle pyyntöjä noutaa tai muokata resursseja, ja palvelin vastaa näihin pyyntöihin. Nämä pyynnöt koostuvat yleensä HTTP-verbistä, otsikosta, resurssin polusta ja vaihtoehtoisesta dataa sisältävästä viestistä. HTTP-verbejä voivat olla esimerkiksi neljä tavallisinta toimintoa, joita kutsutaan CRUD-operaatioiksi. CRUD tulee seuraavista sanoista:

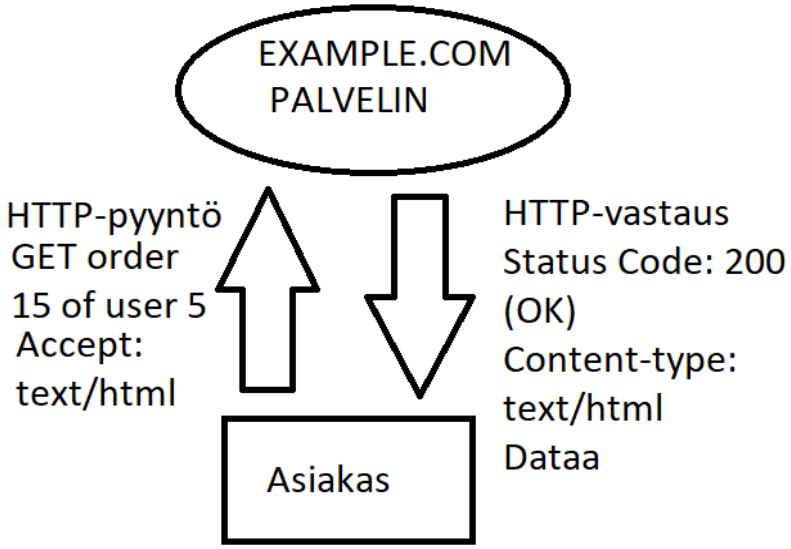
- Create. Luodaan uusi resurssi. Toteutetaan HTTP-verbillä POST.
- Read. Noudetaan tietty resurssi (yleensä id:n mukaan) tai kokoelma resursseja. Toteutetaan HTTP-verbillä GET.
- Update. Päivitetään tietty resurssi. Toteutetaan HTTP-verbillä PUT.
- Delete. Poistetaan tietty resurssi. Toteutetaan HTTP-verbillä DELETE.

Pyyntöjen otsikot sisältävät noudetun sisällön tyypin. Näin voidaan varmistaa, että palvelin ei lähetä dataa, jota asiakas ei pysty käsittelemään. Varmistus tehdään Accept-kentässä. Esimerkiksi HTML:ää sisältävä tekstitiedosto määriteltäisiin tyypillä text/html. Muita tyyppejä ovat esimerkiksi kuvat ja ääni- ja videotiedostot. Polut puolestaan määrittävät, missä resurssissa haluttu toimenpide suoritetaan ja auttavat asiakasta ymmärtämään, missä ollaan ja mitä tapahtuu. Esimerkkipolku

```
example.com/users/5/orders/15
```

kertoo selvästi, mihin se viittaa, eli tässä tapauksessa käyttäjään, jonka id on 5 ja hänen tilaukseensa id:llä 15. Tämä helpottaa huomattavasti uusien käyttäjien navigoimista sivustolla ja pysymistä ajan tasalla. (Codecademy)

Palvelin puolestaan vastaa pyyntöihin lähettämällä vastaavan tyyppisen kentän kuin asiakkaan pyynnössä, esimerkiksi text/html. Lisäksi palvelin lähettää asiakkaalle tilannekoodin, joka kertoo operaation onnistumisesta. Esimerkiksi koodi ”200 (OK)” kertoo onnistuneesta HTTP-pyynnöstä, kuten tiedon lukemisesta GET-verbillä. (Codecademy)



Kuva 1 Esimerkki REST:n toimintaperiaatteesta. (Holopainen 2021)

3.4.3 MVC

MVC (Model-View-Controller) on arkkitehtuurimalli, joka jakaa sovelluksen kolmeen loogiseen komponenttiin: malliin (model), näkymään (view) ja kontrolleriin eli ohjaimeen (controller). Jokainen näistä komponenteista on rakennettu käsittelemään sovelluksen kehitysnäkökulmaa. MVC on yksi eniten käytetyistä web-kehitystyökaluista, koska sen avulla projekteista saadaan skaalautuvia ja helposti laajennettavia. (Tutorialspoint)

Malli-komponentti vastaa kaikesta dataan liittyvästä logiikasta, jonka kanssa käyttäjä on tekemisissä. Tämä voi kuvata esimerkiksi dataa, jota siirrellään näkymän ja kontrollerin välillä, tai se voi olla myös muuhun sovelluksen logiikkaan liittyvää dataa. Esimerkiksi User-malli hakee käyttäjän tiedot tietokannasta, muokkaa tietoja tarvittaessa ja päivittää tiedot takaisin kantaan tai käyttää tietoja sen esittämiseen. (Tutorialspoint)

Näkymä-komponenttia käytetään kaikkeen sovelluksen käyttöliittymään kuuluvaan logiikkaan. Esimerkiksi User-näkymässä ovat kaikki käyttöliittymään kuuluvat omat komponentit, kuten tekstikentät, pudotusvalikot, painikkeet ja muut tekstit. Loppukäyttäjä on vuorovaikutuksessa näiden osien kanssa. (Tutorialspoint)

Kontrolleri toimii rajapintana mallin ja näkymän välillä, jossa hallitaan saapuvia pyyntöjä, muokataan dataa mallia hyödyntäen ja ollaan vuorovaikutuksessa näkymän kanssa, jotta lopputulos saadaan esitettyä. Esimerkiksi User-kontrolleri käsittelee kaikki tapahtumat ja syötteet User-näkymästä ja päivittää tietokantaa käyttäen User-mallia. Samaa kontrolleria käytetään tiedon tarkasteluun. (Tutorialspoint)

MVC:tä käytetään erityisesti web- ja mobiilisovellusten suunnittelussa ja sitä voi useiden ohjelmointikielten eri kehyksissä. MVC:n avulla kehitystyö voidaan helpommin jakaa eri osiin ja sovellusta on myös helpompi päivittää ja muokata. (Doherty, 2020)

3.4.4 Migraatio

Migraatio on Laravelin tapa toteuttaa tietokantojen oma versionhallinta. Mikäli tietokannan tai tietyn tietokannan taulun rakenteeseen täytyy tehdä muutoksia, migraatio ratkaisee tämän ongelman. Migraatio toteutetaan komentorivillä ajamalla komento halutulle migraatiolle (esimerkiksi uuden taulun luonti), muokkaamalla sopivasti Laravelin luomaa migraatiotiedostoa ja suorittamalla itse migraation (Otwell). Koko migraation prosessi käydään läpi kappaleessa 4.3.

3.4.5 Blade

Blade on yksikertainen, mutta tehokas ”mallimoottori” Laravel-ohjelmistokehyksessä. Blade-mallitiedostot käyttävät blade.php-päätettä ja toimivat MVC:n näkymä-komponentteina (ei pidä sekoittaa MVC:n malli-komponenttiin), eli niissä toteutetaan käyttöliittymän logiikka. Blade tarjoaa lisäksi oikoreittejä yleisimpiin PHP:n peruslogiikoihin, kuten ehtolauseisiin ja silmukoihin eli loopeihin (Otwell). Blade-tiedostoja tarkastellaan tarkemmin kappaleessa 4.5.3.

4 TYÖN TOTEUTUS

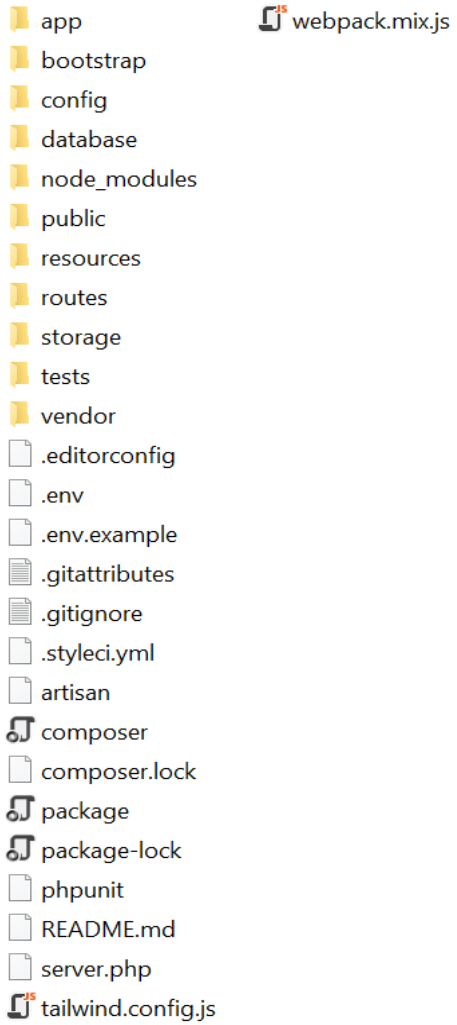
4.1 Alkuvalmistelut

Aloituksen yhteydessä tehtiin tarvittavat asennukset ja luotiin uusi Laravel-projekti. Itse työkalujen asennukseen ei tarvinnut käyttää aikaa, sillä kaikki projektissa tarvittavat työkalut (WampServer, Visual Studio Code, PHP) olivat jo valmiiksi asennettuina ja niistä myös oikeat versiot. Joskus voi käydä niin, että PHP:n versio ei ole riittävän uusi Laravel 8:n käyttöä varten, joten PHP joudutaan päivittämään ja sen myötä myös WampServer. Tässä projektissa käytettiin PHP:n versiota 7.4.14.

Uusi Laravel-projekti luodaan käyttäen Composeria, jonka avulla tehdään PHP:n paketinhallinta sekä pidetään projektin riippuvuudet ajan tasalla. Uuden projektin luonti tapahtuu komentorivillä valitsemalla haluttu polku ja suorittamalla seuraava komento:

```
composer create-project laravel/laravel saviska
```

Laravel luo selkeän kansiorakenteen ja generoi valmiiksi osiot MVC-mallille sekä joitain perustoimintoja.



Kuva 2 Laravel-projektin kansiorakenne. (Holopainen 2021)

Jotta sivusto saataisiin näkymään paikallisesti, WampServerin on tiedettävä, missä projektin aloitussivu sijaitsee. Tämä saadaan tehtyä määrittämällä projektin polku rootiksi (juureksi) Apachen asetuksiin, jotka puolestaan löytyvät tiedostosta

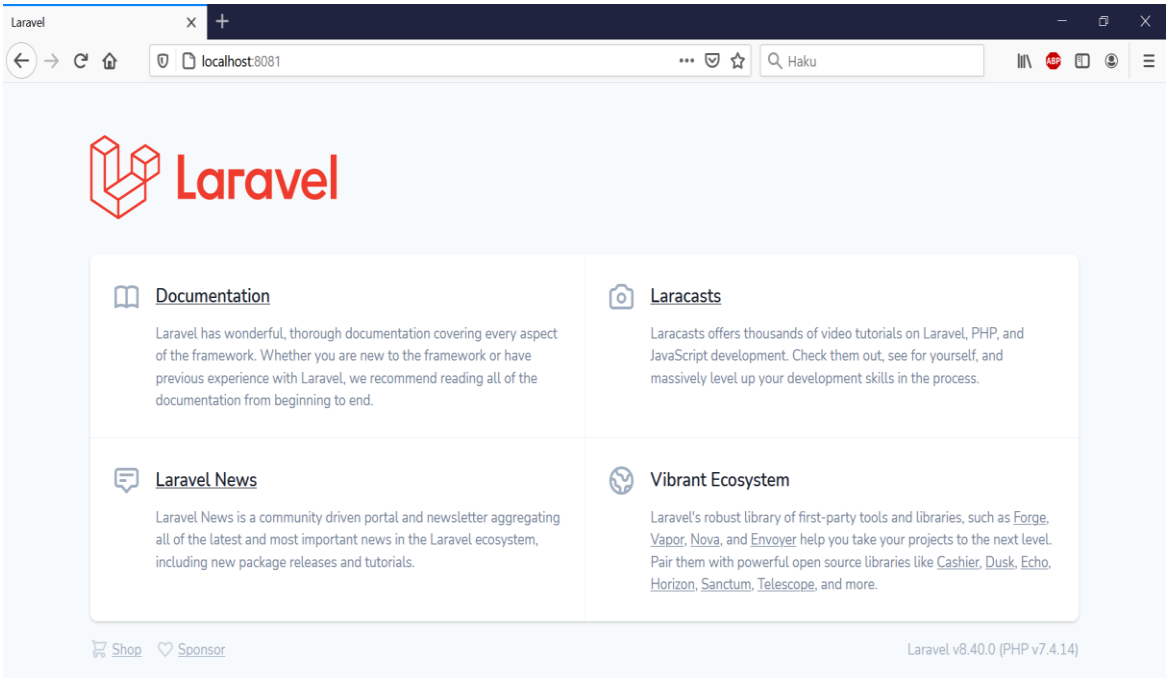
```
C:\wamp64\bin\apache\apache2.4.35\conf\httpd.conf
```

eli WampServerin sijainnin alta, joka voi vaihdella riippuen, mihin WampServer on asennettu. Httpd.conf-tiedostoon määritetään projektin sijainti rootiksi.

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "C:\Oppari\saviska\public"
<Directory "C:\Oppari\saviska\public">
```

Kuva 3 Projektin sijainnin määrittäminen rootiksi WampServeriin. (Holopainen 2021)

Apachen asetuksiin on myös määritelty, että sivusto aukeaa portissa 8081. Testataan sivuston toimivuus paikallisesti.



Kuva 4 Sivuston testaaminen. (Holopainen 2021)

Sivusto on käytännössä Laravelin generoima oma aloitussivu, koska projektiin ei ole tässä vaiheessa tehty vielä muita toiminnallisuuksia. Pääasia eli projektin toimivuuden testaaminen kuitenkin suoritettiin onnistuneesti.

Projektiin asennettiin vielä lisäksi Jetstream- ja Livewire-lisäosat, jotka tarjoavat toteutukset esimerkiksi kirjautumiselle, rekisteröinnille ja kirjastot Blade-mallimoottoria varten. Jetstream asennetaan

seuraavasti:

```
composer require laravel/jetstream
```

Livewiren asennukseen hyödynnetään seuraavaa Laravelin Artisan-komentoa:

```
php artisan jetstream:install livewire
```

Tämän jälkeen täytyy vielä päivittää projektin kirjastojen riippuvuudet npm-komennoilla:

```
npm install

npm run dev
```

Ajetaan myös migraatio, joka luo valmiiksi oletustauluja tietokantaa varten (migraatioista tarkemmin kappaleessa 4.3):

```
php artisan migrate
```

4.2 Git-versionhallinnan käyttöönotto

Git-komentorivillä määritetään yhteys jo valmiiksi luotuun GitHub-repositorioon, johon tarvitaan oikeudet GitHub-käyttäjälle, joka on puolestaan jo valmiiksi yhdistetty Git-komentoriviin. Yhteyden muodostamiseen käytetään seuraavia komentoja:

```
git init
git add .
git commit -m “first commit”
git commit -M master
git remote add origin https://github.com/Traakkoni/saviska.git
git push -u origin master
```

Ensimmäinen komento luo paikallisen repositorion, ja se tehdään vain kerran projektin alussa. Muuten samoja komentoja käytetään, kun projektin uusin versio halutaan päivittää repositorioon. Projektin tilanteen voi tarkistaa komennolla

```
git status
```

joka näyttää, onko projektin tiedostoja esimerkiksi muokattu, poistettu tai lisätty uusia tiedostoja. Yleensä GitHub-repositorioissa on useita branchejä (haaroja), joiden avulla projektin eri osa-alueiden kehitys voidaan eristää toisistaan, eikä master-branchiä käytetä kuin vasta valmiin projektin päivittämiseen. Tässä projektissa kuitenkin käytettiin pelkkää master-branchiä johtuen siitä, että kehittäjiä oli vain yksi ja MVC-malli käytännössä tekee osa-aluejaon jo valmiiksi, joten tarvetta muille brancheille ei ollut.

4.3 Tietokanta ja migraation toteutus

Tietokannan hallinta tapahtuu WampServeriin kuuluvan PhpMyAdminin avulla. Sinne luodaan uusi tietokanta, joka yhdistetään Laravel-projektiin. Yhdistäminen tapahtuu projektista löytyvässä .env-tiedostossa, jossa ovat parametrit sovelluksen ja tietokannan tiedoille sekä tietokannan käyttäjä ja salasana. Tätä .env-tiedostoa ei oteta mukaan versionhallintaan, sillä eri kehittäjät tai palvelimet voivat vaatia erilaisen konfiguraation, minkä lisäksi .env-tiedosto väärissä käsissä voi olla iso tietoturvariski, koska kyseisessä tiedostossa on esimerkiksi salasana selkokielellä. Näistä seikoista johtuen .env-tiedosto pysyy ainoastaan paikallisessa käytössä, mutta .env-example-tiedosto voidaan ottaa mukaan versionhallintaan, koska siinä ovat nimensä mukaisesti vain esimerkit vaaditusta .env-tiedoston rakenteesta.

Muutokset tietokantaan toteutetaan migraation avulla, esimerkiksi uuden taulun lisääminen. Migraatioita varten käytetään Laravelin Artisan-komentoja. Uuden migraation luonti voidaan suorittaa seuraavasti:

```
php artisan make:migration [create_tablename_table]
```

Migraatio luodaan projektin kansioon databases/migrations aikaleiman kanssa, jotta Laravel pystyy määrittämään migraatioiden järjestyksen. Migraation nimi kannattaa valita sopivasti siten, että Laravel pystyy täyttämään taulun nimen jo etukäteen migraatiotiedostoon. Kyseistä tiedostoa voidaan sitten muokata sellaiseksi, kun kyseisen taulun rakenne halutaan.

```php
class CreateCoursesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('courses', function (Blueprint $table) {
            $table->id();
            $table->string('date');
            $table->string('name');
            $table->string('price');
            $table->string('extra_info1');
            $table->string('extra_info2');
        });
    }
```

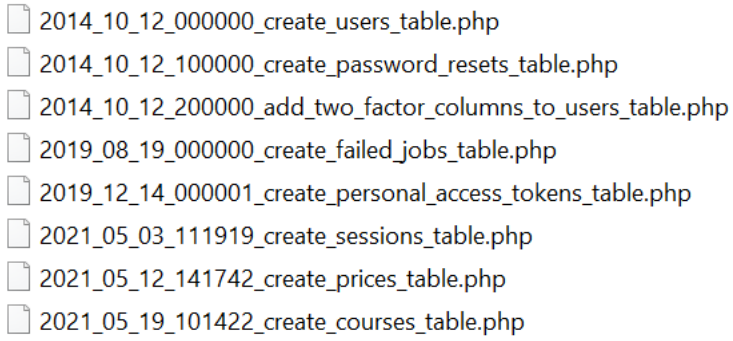
Kuva 5 Migraatiotiedosto courses-taulun luontia varten. (Holopainen 2021)

Projektin spekseihin kuului, että kurssiin liittyviä muokattavia tietoja ovat päivämäärä, nimi, hinta ja kaksi lisätietokenttää. Nämä määritellään migraatiotiedostoon, jonka jälkeen migraatio suoritetaan.

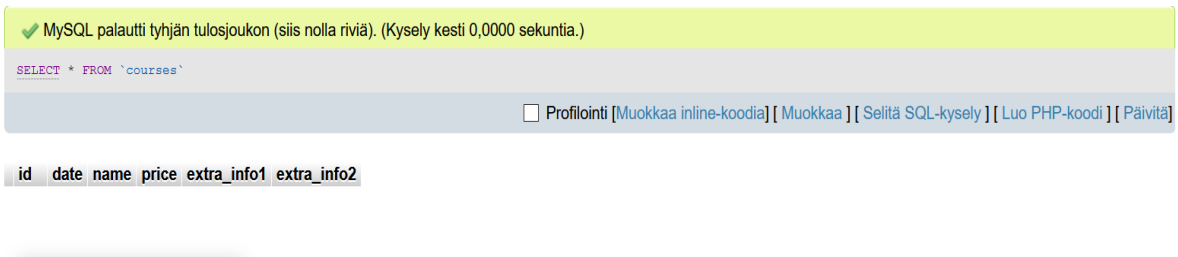
```
C:\Oppari\saviska>php artisan make:migration create_courses_table
Created Migration: 2021_05_19_101422_create_courses_table

C:\Oppari\saviska>php artisan migrate
Migrating: 2021_05_19_101422_create_courses_table
Migrated:  2021_05_19_101422_create_courses_table (41.45ms)

C:\Oppari\saviska>
```

Kuva 6 Migraation suorittaminen kokonaisuudessaan. (Holopainen 2021)

Projektin muiden taulujen luonti ja rakenteen muokkaaminen toteutettiin täysin vastaavalla tavalla kuin courses-taulun kanssa, toki jokaiselle taululle tuli omat muuttujat, mutta muuten idea oli täysin sama: luodaan migraatiotiedosto, muokataan tiedostoa sopivasti ja ajetaan migraatio.



Kuva 7 Migrations-kansion sisältöä. (Holopainen 2021)



Kuva 8 Courses-taulu PhpMyAdminissa. (Holopainen 2021)

4.4 Autentikointi

Yhtenä ehtona hallintapaneelille oli, että sitä voivat käyttää vain tietyt käyttäjät, eli sinne ei pysty itse rekisteröitymään, vaan pelkästään kirjautunut käyttäjä voi lisätä uusia käyttäjiä. Tämä pystytään toteuttamaan Fortify-autentikointityökalun avulla. Oletuksena se tekee automaattisesti kaiken backend-toiminnallisuuden tavalliselle rekisteröinnille, mutta koska uuden käyttäjän lisääminen haluttiin toteuttaa vain käyttäjän ollessa kirjautuneena hallinpaneeliin, Fortifyn toimintaa oli hieman muutettava. Se tapahtui siten, että Fortifyn omia reittejä (routes) ei huomioida, vaan ne kopioidaan ja muokataan siten, että vain kirjautunut käyttäjä pääsee rekisteröintinäkymään.

Fortifyn omien reittien huomioimatta jättäminen tehdään konfiguraatiotiedostossa

```
app\Providers\FortifyServiceProvider.php
```

ja siellä boot-funktioon lisätään seuraava rivi:

```
Fortify::ignoreRoutes();
```

Fortifyn reitit löytyvät sen omasta routes.php-tiedostosta ja sieltä kopioidaan rekisteröintiä varten tarkoitetut reitit ja funktiot. Ne lisätään routes/web.php-tiedostoon, joka vastaa web-sovelluksen REST-rajapinnasta ja niihin pääsee kirjoittamalla reitin URL:n selaimeen (mikäli käyttäjällä on oikeudet kyseiseen reittiin). Rekisteröinti vain kirjautuneella käyttäjällä onnistuu seuraavasti:

```php
Route::group(['middleware' => config('fortify.middleware', ['web'])], function () {
    $enableViews = config('fortify.views', true);

if (Features::enabled(Features::registration())) {
        if ($enableViews) {
            Route::get('/register', [RegisteredUserController::class, 'create'])
                ->middleware(['auth:sanctum', 'verified'])
                ->name('register');
        }

        Route::post('/register', [RegisteredUserController::class, 'store'])
            ->middleware(['auth:sanctum', 'verified']);
    }
});
```

Kuva 9 web.php-tiedoston sisältöä rekisteröinnin osalta. (Holopainen 2021)

4.5 MVC-komponentit

Kuten jo projektin luonnin yhteydessä todettiin, Laravel generoi automaattisesti kansiorakenteen MVC-mallille, joten eri osien toteutus oli helppo aloittaa. MVC:n osista malli ja kontrolleri löytyvät app-kansion alta ja näkymä sijaitsee resources-kansiossa.

MVC-komponenttien toteutusjärjestyksellä ei sinänsä ole suurta merkitystä, mutta tässä työssä aloitettiin mallin luomisella, jonka jälkeen muodostettiin kontrolleri ja viimeisenä näkymä kyseiselle sivulle. Tämä etenemisjärjestys myös tuntui kaikista loogisimmalta ja yksinkertaisimmalta toteuttaa. Lisäksi täytyi muodostaa sopivat reitit näkymille ja mahdollinen datan siirtely.

Toteutettavia MVC-kokonaisuuksia oli käytännössä yhtä monta kuin hallintapaneelissa on sivuja tiedon muokkausta varten, eli yhteensä kolme kokonaisuutta. Niiden luontiprosessi oli tismalleen sama joka kerta, toki jokainen malli, näkymä ja kontrolleri tehtiin halutulle toiminnallisuudelle sopivaksi.

4.5.1 Malli

Malli-komponentin sisältö tehtiin samanlaiseksi kuin kyseistä mallia vastaavan tietokannan taulun sisältö. Esimerkiksi course-malliin tulivat samat attribuutit kuin mitä courses-taulussa oli.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Course extends Model
{
    use HasFactory;
    public $timestamps = false;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'date',
        'name',
        'price',
        'extra_info1',
        'extra_info2',

    ];
```

Kuva 10 Course.php-tiedoston sisältöä. (Holopainen 2021)

4.5.2 Kontrolleri

Kontrolleri kannattaa nimetä mallia vastaavaksi, esimerkiksi course-mallia vastaavaksi kontrolleriksi nimettiin CourseController. Kontrollerissa on funktioita, jotka voivat esimerkiksi muokata dataa tai

palauttaa muita näkymiä. Dataa muokkaavien funktioiden ideana on, että kontrolleri kuuntelee saapuvia pyyntöjä (tässä tapauksessa pyynnöt tulevat näkymältä) ja dataa muokataan pyynnön mukaisesti.

```
public function AddNewCourse (Request $request)
{
    Validator::make($request->toArray(), [
        'date' => ['required'],
        'name' => ['required'],
        'price' => ['required']
    ])->validate();
    try {
        $course = new Course();
        $course->date = $request['date'];
        $course->name = $request['name'];
        $course->price = $request['price'];
        $course->extra_info1 = $request['extra_info1'];
        $course->extra_info2 = $request['extra_info2'];
        $course->save();
        $status['status'] = "Kurssin lisäys onnistui!";
        return Redirect::route('courses')->with($status);
    } catch(Exception $e) {
        $status['status'] = "Kurssin lisäys ei onnistunut!";
        Logger()->error('Lisäys ei onnistunut!');
        return Redirect::route('courses')->withErrors($status);
    }
}
```

Kuva 11 Uuden kurssin lisääminen CourseControllerissa. (Holopainen 2021)

Kontrolleri ottaa vastaan dataa näkymältä ja vaatii, että kentissä date, name ja price on jotain tietoa. Sitten luodaan uusi kurssi sitä vastaavasta näkymästä ja lisätään pyynnössä tulleet tiedot niitä vastaaviin attribuutteihin, jonka jälkeen se tallennetaan tietokantaan ja annettaan käytettävyyden parantamiseksi viesti käyttäjälle toimenpiteestä ja sen mahdollisesta epäonnistumisesta.

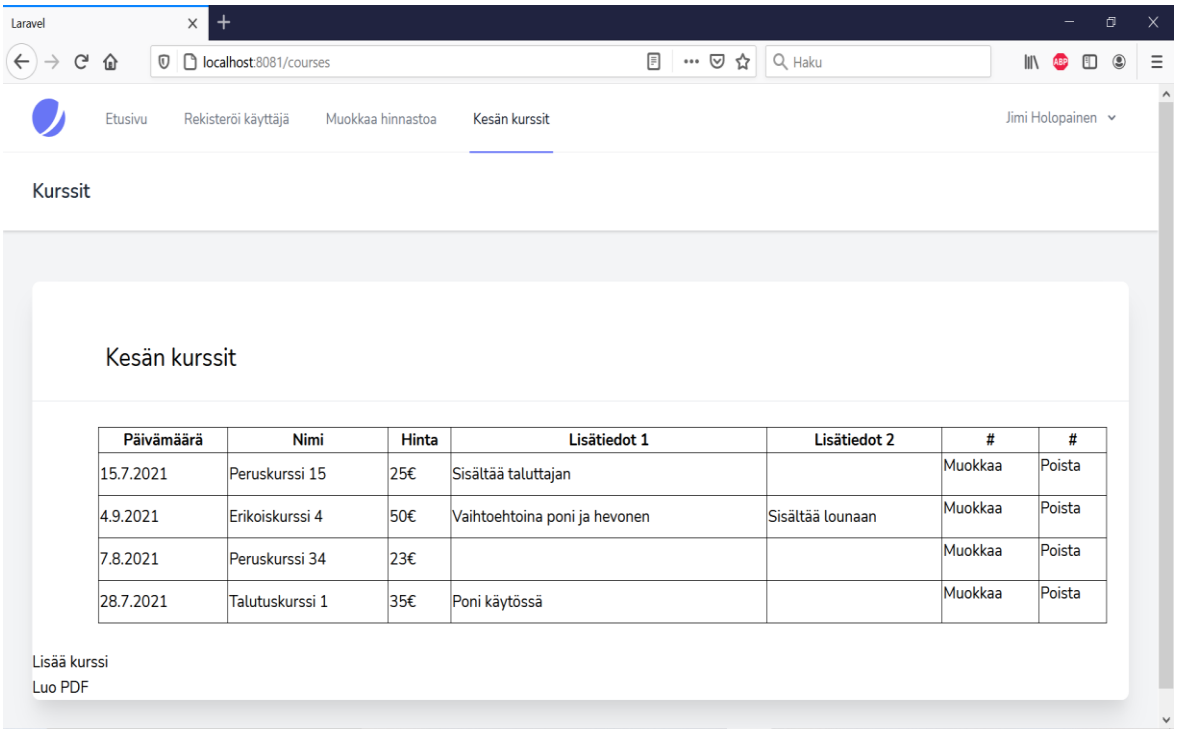
4.5.3 Näkymä

Blade-tiedostoissa toteutetaan itse käyttöliittymä. Nimeäminen tehdään loogisesti sen mukaan, mitä tietoa kyseisessä näkymässä on tarkoitus käsitellä. Tiedostot voivat sisältää HTML- tai PHP-koodia sekä itse tehtyjä Blade-komponentteja. Esimerkiksi kurssien listaamisessa käytettiin HTML:n table-elementtiä ja haluttu data saatiin näkyviin käyttäen PHP:n ehtolauseita.

```
<div>
    <div class="p-6">
        <div class="ml-12">
            <table style="width:100%">
                <tr>
                    <th>Päivämäärä</th>
                    <th>Nimi</th>
                    <th>Hinta</th>
                    <th>Lisätiedot 1</th>
                    <th>Lisätiedot 2</th>
                    <th>#</th>
                    <th>#</th>
                </tr>
                @forelse($courses as $course)
                <tr>
                    <td>{{$course->date}}</td>
                    <td>{{$course->name}}</td>
                    <td>{{$course->price}}</td>
                    <td>{{$course->extra_info1}}</td>
                    <td>{{$course->extra_info2}}</td>
                    <td><form action="{{ route('editcourse', $course->id)}}" method="get">
                        @csrf
                        @method('GET')
```

Kuva 12 Courses.blade.php-tiedoston sisältöä. (Holopainen 2021)

Selaimessa näkymä näyttää seuraavalta esimerkkidatan kanssa:



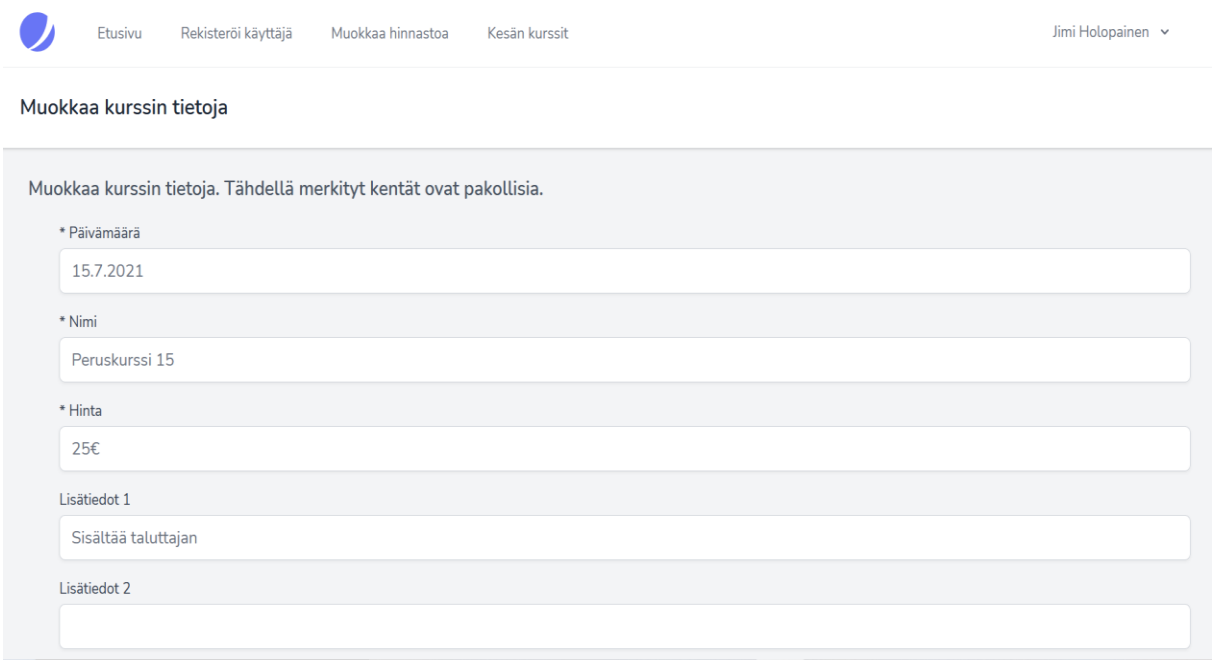
Kuva 13 Courses-näkymä selaimessa. (Holopainen 2021)

Elementtien asettelu tehtiin mahdollisimman yhtenevästi ja siten, että tiedon lukeminen ja navigointi onnistuisi helposti käyttökokemuksen parantamiseksi. Ulkoasu sivustolla on yksinkertainen johtuen siitä, että spekseihin ei kuulunut ulkoasulliset seikat muuten, kuin elementtien sijoittelun osalta. Siksi esimerkiksi painikkeissa on haluttu toiminnallisuus, mutta ei vielä tyylimäärittelyä.

Datan muokkauksessa tiedot täytettiin valmiiksi tekstikenttiin, jotta käyttäjän ei tarvitsisi muistella, mitä tietoja kyseisille attribuuteille aiemmin oli.

```
<div class="mt-4">
    <x-jet-label for="name" value="* {{ __('Nimi') }}" />
    <x-jet-input id="name" class="block mt-1 w-full @error('name') is-invalid @enderror" type="text" name="name"
    value="{{old('name') ?? $course['name']}}" required  />
    @error('name')<span style="color: red">{{ $message }}</span>@enderror
</div>
```

Kuva 14 Editcourse.blade.php-tiedoston sisältöä. (Holopainen 2021)



Kuva 15 Editcourse-näkymä selaimessa. (Holopainen 2021)

4.6 Reitit

Kuten autentikoinnin yhteydessä todettiin, reitit löytyvät routes/web.php-tiedostosta ja siellä on kaikki sovelluksen REST-rajapintaan liittyvä toiminnallisuus. Reitit ohjaavat tietyn URL:n mukaisesti käyttäjän haluttuun näkymään tai pyytävät kontrolleria suorittamaan halutun toimenpiteen. Näihin käytetään CRUD-operaatioita.

```
Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {
    return view('dashboard');
})->name('dashboard');
Route::middleware(['auth:sanctum', 'verified'])->get('/', function () {
    return view('dashboard');
})->name('dashboard');
Route::middleware(['auth:sanctum', 'verified'])->get('/courses', function () {
    $courses['courses'] = Course::all();
    return view('courses')->with($courses);
})->name('courses');
Route::middleware(['auth:sanctum', 'verified'])->get('/addcourse', function () {
    return view('addcourse');
})->name('addcourse');
```

Kuva 16 Web.php-tiedoston sisältöä. (Holopainen 2021)

```
Route::put('updateprices/{price}', [PriceController::class, 'UpdatePrices'])->name('updateprices')->middleware(['auth:sanctum', 'verified']

Route::post('addnewcourse', [CourseController::class, 'AddNewCourse'])->name('addnewcourse')->middleware(['auth:sanctum', 'verified']);

Route::get('editcourse/{course}', [CourseController::class, 'EditCourse'])->name('editcourse')->middleware(['auth:sanctum', 'verified']);

Route::put('updatecourse/{course}', [CourseController::class, 'UpdateCourse'])->name('updatecourse')->middleware(['auth:sanctum', 'verified

Route::delete('deletecourse/{course}', [CourseController::class, 'DeleteCourse'])->name('deletecourse')->middleware(['auth:sanctum', 'verif


Route::get('/pdf', [CourseController::class, 'createPDF'])->name('pdf')->middleware(['auth:sanctum', 'verified']);
```

Kuva 17 Web.php-tiedoston sisältöä. (Holopainen 2021)

Reittien oikein nimeäminen on erittäin tärkeää, sillä sen tulisi vastata sitä, mitä toimintoja kyseisen reitin kautta tulee tapahtumaan. Tämä myös helpottaa käyttäjän navigointia, sillä jos reitti ohjaa tiettyyn näkymään, niin URL:n olisi hyvä vastata kyseisessä näkymässä näytettävää dataa tai toiminnallisuuksia. Lisäksi hyvin suunniteltu reititys helpottaa sovelluksen päivittämistä.

Tässä työssä jokaiseen reittiin lisättiin autentikointi, koska rekisteröimättömien käyttäjien pääsy hallintapaneeliin ei ole sallittua. Reitit nimettiin näkymiä tai kontrollereiden funktioita vastaaviksi.

5 POHDINTA JA JATKOKEHITYSIDEAT

Opinnäytetyö onnistui mielestäni erittäin hyvin siihen nähden, että käytettävissä ollut aika oli melko rajallinen. Erittäin iso etu oli siinä, että Laravelia oli tullut käytettyä jo aiemmin, joten sen perusperiaatteiden opetteluun ei tarvinnut käyttää ylimääräistä aikaa. Työn speksit olivat selvät ja loogiset, joten eteneminen onnistui vaivattomasti. Laravelin kattava dokumentaatio auttoi myös tarvittaessa. Lisäksi oma ohjelmointiosaaminen kehittyi työn aikana.

Työn spekseissä määritellyt osa-alueet saatiin toteutettua ja niihin halutut toiminnallisuudet. Se ei kuitenkaan tarkoita sitä, että käyttöliittymää ei voisi parantaa. Siihen jäi vielä joitain käytettävyysongelmia, mutta nämä eivät ole kovin vakavia, esimerkiksi palautteen antaminen toimenpiteestä käyttäjälle ei onnistu jokaisessa tilanteessa. Lisäksi ulkoasullisia seikkoja ei juurikaan tehty työssä, mutta ne eivät toisaalta olleet osa työn speksejä. Perustoiminnot kuitenkin toimivat ongelmitta eli käyttäjien lisääminen ja hinnaston, kurssien ja hevosten tietojen muokkaus. Myös käytettävyys saatiin hyvälle tasolle pieniä edellä mainittuja yksityiskohtia lukuun ottamatta. Suuremman mittakaavan jatkokehitysideoita voisivat olla esimerkiksi sivuston osioiden lisääminen ja muokkaaminen, koska nyt sivuston sisältö ja sitä kautta myös hallintapaneelin toiminta on ennalta määritelty tiettyihin osioihin. Lisäksi joitain pienempiä jo tehtyjä ominaisuuksia voisi vielä kehittää, esimerkiksi kurssien listaaminen päivämäärän mukaan.

Kokonaisuutena voin kuitenkin olla tyytyväinen työn tuloksiin etenkin tiukan aikataulun aiheuttamien haasteiden takia. Suurempia aikatauluun vaikuttavia ongelmia ei kuitenkaan onneksi ilmennyt työn aikana, vaan työ eteni vaihe vaiheelta loogisesti ja lopulta työn alussa asetettuihin tavoitteisiin päästiin.

LÄHTEET

Brown, K. (13. 11. 2019). *How-To Geek*. Haettu 9. 5. 2021 osoitteesta https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/

Churchville, F. (2019). *user interface (UI)*. Haettu 3. 5. 2021 osoitteesta https://searchapparchitecture.techtarget.com/definition/user-interface-UI

Codecademy. Haettu 12. 5. 2021 osoitteesta https://www.codecademy.com/articles/what-is-rest

Doherty, E. (11. 5. 2020). *Educative*. Haettu 12. 5. 2021 osoitteesta https://www.educative.io/blog/mvc-tutorial

Interaction Design Foundation. Haettu 4. 5. 2021 osoitteesta https://www.interaction-design.org/literature/topics/user-centered-design

Interaction Design Foundation. Haettu 7. 5. 2021 osoitteesta https://www.interaction-design.org/literature/topics/ux-design

Muniz, F. *Usability Geek*. Haettu 7. 5. 2021 osoitteesta https://usabilitygeek.com/heuristic-evaluation-introduction/

Nielsen, J. (24. 4. 1994). *Nielsen Norman Group*. Haettu 7. 5. 2021 osoitteesta https://www.nngroup.com/articles/ten-usability-heuristics/

Nielsen, J. (3. 1. 2012). *Nielsen Norman Group*. Haettu 7. 5. 2021 osoitteesta https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Otwell, T. *Laravel*. Haettu 9. 5. 2021 osoitteesta https://laravel.com/docs/8.x

PHP. Haettu 9. 5. 2021 osoitteesta https://www.php.net/manual/en/faq.general.php

Stevens, E. (19. 3. 2021). *Careerfoundry*. Haettu 7. 5. 2021 osoitteesta https://careerfoundry.com/en/blog/ux-design/what-is-user-experience-ux-design-everything-you-need-to-know-to-get-started/

Tutorialspoint. Haettu 12. 5. 2021 osoitteesta https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

Usability.gov. Haettu 5. 5. 2021 osoitteesta https://www.usability.gov/what-and-why/user-centered-design.html

Vaittinen, J. (23. 10. 2019). *Codemate*. Haettu 5. 5. 2021 osoitteesta https://www.codemate.com/fi/kayttajakeskeinen-suunnittelu/

Visual Studio Code. Haettu 9. 5. 2021 osoitteesta https://code.visualstudio.com/docs

Wong, E. (2020). *Interaction Design Foundation*. Haettu 7. 5. 2021 osoitteesta https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces

WPBeginner. Haettu 9. 5. 2021 osoitteesta https://www.wpbeginner.com/glossary/wamp/