Henry Hoipo

VARAUSJÄRJESTELMÄ PORIN KAUPUNKIKESKUSTA RY:LLE

Tietojenkäsittelyn koulutusohjelma

2020

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

VARAUSJÄRJESTELMÄ PORIN KAUPUNKIKESKUSTA RY:LLE

Hoipo Henry
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Lokakuu 2019
Sivumäärä: 29

Asiasanat: php, laravel, mvc, varausjärjestelmä

Opinnäytetyössä kerrotaan varausjärjestelmän suunnittelusta ja rakentamisesta Porin Kaupunkikeskusta Ry:lle. Järjestelmän tarkoituksena oli tuoda käyttöön uusi hallintatyökalu asiakasrekisterin, myyntialueiden, niiden myyntipaikkojen sekä varausten ylläpitämiseksi.

Työssä kerrotaan enimmäkseen Laravel nimisen PHP-ohjelmistokehyksen toiminnasta ja käytettävyydestä, sekä hieman tietokannan rakentamisesta ja ohjelmistotuotannosta teoriassa ja käytännössä.

Käytännön osuudessa kerrotaan ensin hieman ohjelmistotuotannosta, jonka jälkeen tietokannan luomisesta ja suunnittelusta. Tämän jälkeen kerrotaan itse sovelluksen ohjelmoinnista vaiheittain.

Sovelluksesta saatiin hyvin perustarpeisiin vastaava järjestelmä, jonka avulla voitiin luoda ja ylläpitää asiakasrekisteriä, varauksia, myyntialueita ja niiden myyntipaikkoja. Järjestelmän päivittämistä ja jatkokehittämistä ei jatkettu ensimmäistä versiota pidemmälle yhteistyön tultua päätökseen.

RESERVATION SYSTEM FOR PORIN KAUPUNKIKESKUSTA RY

Hoipo Henry
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Business Information Technology
October 2019
Number of pages: 29

Keywords: php, laravel, mvc, reservation system

The purpose of this thesis was to tell about planning and programming a reservation system for Pori Kaupunkikeskusta Ry. The system was designed to bring a new management tool for customer database, sales region and outlets, as well as reservation maintenance.

The thesis mainly describes how the PHP software framework called Laravel operates. But also tells a bit about planning and creating a database and application around it, as well as software engineering in theory and in use of a real project.

The practical section starts with software engineering in use. Then continues with database planning and constructing. And lastly about building the software itself step-by-step.

The application was built to be a very basic maintaining system for customer register, reservations, sales areas and their outlets. Further development for the application was stopped due to co-operation cancelling out.

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä kerron Laravel-nimisestä PHP-ohjelmistokehyksestä, sen toiminnasta ja miten sen käyttö sovelluksen rakentamisessa käytännössä tapahtuu. Lisäksi avaan myös hieman tietokannan rakennetta ja sen taulujen suhteita toisiinsa, sekä kerron ohjelmistotuotannosta teoriassa ja käytännössä.

Tämän opinnäytetyön aiheen sain sen hetkisestä työharjoittelupaikastani Aline Creative Technology Studio Oy:ltä. Sovellus oli toinen projektini työharjoittelun alkamisesta ja ensimmäinen laatuaan, jossa pääsin suunnittelemaan ja rakentamaan sovelluksen alusta loppuun. Aiempaa kokemusta PHP-ohjelmointikielestä minulla oli hyvin vähän, sekä käytännön kokemusta MVC-mallin toteuttamisesta ei yhtään. Projektissa pääsin myös suunnittelemaan ensimmäistä kertaa tietokannan rakenteen ja sen yhteydet. Käytettyjen tekniikoiden opiskeluun käytin verkosta löydettävissä olevia dokumentaatioita ja keskustelufoorumeita, sekä kollegani tietotaitoa ja käytettävissä olleiden vastaavien ohjelmistojen koodia.

Opinnäytetyössä kerrottavan projektin tavoite oli saada Porin Kaupunkikeskusta Ry:n sisäiseen käyttöön järjestelmä, jonka avulla he voivat hallita asiakastietojaan sekä varauksia Porissa sijaitsevien Kauppatorin ja Eetunaukion myyntipaikoille.

2 KÄYTETYT TEKNIIKAT

2.1 PHP

PHP (PHP: Hypertext Preprocessor) on hyvin suosittu ja moneen tarkoitukseen sopiva skriptikieli, jonka kehittämisen aloitti vuonna 1994 tanskalainen Rasmus Lerdorf. Sen alkuperäinen tarkoitus oli pitää lukua verkossa olleen tekijänsä ansioluettelon lukijoista. Vuonna 1995 hän julkaisi työkalunsa lähdekoodin julkiseen käyttöön. Opinnäytetyön kirjoitushetkellä viimeisin julkaistu versio PHP:stä on 7.4 ja se julkaistiin marraskuussa 2019. (PHP www-sivut 2020) Maaliskuussa 2020 noin 79% kaikista verkkosivustoista, joiden palvelinpuolen ohjelmistokieli tunnetaan, käyttää PHP:tä. (W3Techs www-sivut 2020) PHP-skriptit suoritetaan palvelimen puolella, ja sen tiedostot voivat sisältää tekstiä, HTML-, CSS-, JavaScript-, ja PHP-koodia.

2.2 Laravel

Laravel on ilmainen, avoimeen lähdekoodiin perustuva PHP-ohjelmistokehys. Sen on kehittänyt Taylor Otwell, ja sen ensimmäinen versio 1.0 julkaistiin kesäkuussa 2011. Tuosta ajankohdasta lähtien päivitettyjä versioita on julkaistu kahdesta kolmeen vuosittain. Kirjoitushetkellä viimeisin vakaa julkaistu versio on 6.2, joka jaettiin yhteisön käyttöön lokakuussa 2019. (Surguy 2013)

Laravel-ohjelmistokehys on suunnattu niille verkkosovelluksille, jotka käyttävät hyväkseen MVC-sovellusarkkitehtuurityyliä. Tämän ohjelmistoarkkitehtuurityylin tarkoituksena on erotella sovelluksen käyttöliittymä, data ja prosessien käsittely toisistaan. MVC-tyylin käyttö mahdollistaa hyvin rakennetun ja suunnitellun sovelluspohjan käyttämistä monissa eri sovelluksissa tai verkkototeutuksissa.

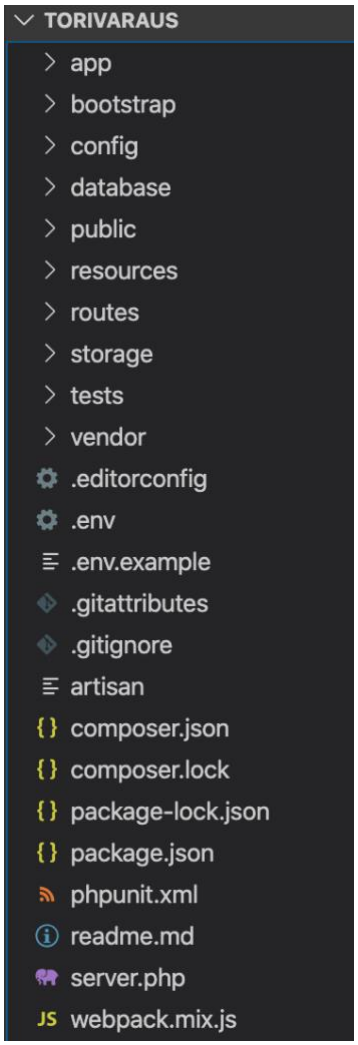
Projektissa käytetyn version 5.8 toimiminen palvelimella edellyttää sen täyttävän seuraavat vaatimukset: PHP versio 7.1.3, PDO PHP -lisäosa, JSON PHP -lisäosa, OpenSSL PHP -lisäosa, Tokenizer PHP -lisäosa, XML PHP -lisäosa, MBstring PHP -lisäosa, Ctype PHP -lisäosa, BCMath PHP -lisäosa sekä Composer PHP -työkalu. (Laravel www-sivut 2020)

Opinnäytetyöprojektin toteutukseen valittiin Laravel sen monimuotoisuuden ja perusteellisen dokumentaation vuoksi. Laravelilla on hyvin laaja käyttäjäkunta, ja monissa PHP-ohjelmistokehys vertailuartikkeleissa se asetettiin ykkösvalinnaksi (Patil 2019) (Walley 2019). Sen avulla voidaan hallinnoida vaativia ja monimutkaisia verkkosovelluksia turvallisesti, nopeasti sekä tehokkaasti. Kehitysprosessit on yksinkertaistettu siten, että tuoreeltaan aloittelevakin ohjelmoija pystyy rakentamaan toimivan ja turvallisen verkkosovelluksen dokumentaatioita ja artikkeleita seuraamalla. Esimerkiksi reititys, istunnot, välimuisti ja käyttäjän varmentaminen on tehty erittäin helpoksi.

Laravelin tiiviin sekä laajan yhteisön ansiosta löytää tähän ohjelmistokehykseen myös monia valmiita paketteja, moduuleita, liitännäisiä (eng. plug-in) sekä komponentteja. Näistä paketeista yksi hyvä esimerkki on itse projektissakin käytetty Laravel Authentication. Artisan-konsolin ja kahden komennon avulla saadaan luotua tietokantataulut, näkymät, mallit ja käsittelijät käyttäjän tunnistautumista varten.

2.2.1 Rakenne

Laravel sisältää käytännössä rakenteen kansioista ja alakansioista, sekä tiedostoista, jotka projektiin liittyvät (Kuva 1). Projektin kannalta eniten käytetyt kansiot olivat app, config, database, resources sekä routes. Tiedosto nimeltä .env taas puolestaan on suuressa osassa koko sovelluksen toiminnassa.



Kuva 1. Laravel-sovelluksen kansiorakenne.

App-kansio sisältää 8 alakansiota, jotka sisältävät koko projektin lähdekoodiin liittyvät tiedostot. Esimerkiksi tapahtumia (eng. events), väliohjelmistoja (eng. middleware), poikkeuksia (eng. exceptions) sekä oikeuksiin ja konsolin toimintaan liittyviä asioita. App-kansio sisältää myös sovelluksen luonnissa suurta roolia esittävät malli- sekä käsittelijätiedostot.

Config-kansio sisältää tiedostoja, jotka sisältävät erilaisten palveluiden konfiguraatioita sekä edistävät Laravel-sovelluksen toiminnallisuutta. Jokainen tiedosto on nimetty siihen liittyvän toiminnallisuuden mukaan, ja sisältää siihen liittyviä parametrejä ja arvoja. Esimerkiksi app.php-tiedosto sisältää sovelluksen nimen, tuotannon vaiheen (kuten suunnitteluvaihe, tuotantovaihe), aikavyöhyke- ja kieliasetuksia.

Database-kansiossa on kolme alakansiota: Seeds, Migrations ja Factories. Näistä Seeds ja Factories sisältävät testidatan luomiseen liittyviä tiedostoja, jolla projektin testaamisesta voidaan tehdä helpompaa valmiiksi ohjelmiston luomalla datalla. Migrations-kansiossa on tietokannan luomiseen, päivittämiseen ja poistamiseen tarkoitettuja tiedostoja (Kuva 2). Näiden avulla tietokannan rakentaminen ja taulujen toiseensa liittäminen tapahtuu hyvinkin nopeasti ja helposti.

```
14      public function up() {
15          Schema::create('myyntialue', function (Blueprint $table) {
16              $table->increments('id')->unsigned();
17              $table->string('nimi');
18              $table->timestamps();
19          });
20      }
21      public function down() {
22          Schema::dropIfExists('myyntialue');
23      }
```

Kuva 2. Myyntialue tietokantataulun luomisen migraatiotiedoston funktiot.

Resource-kansio sisältää alakansioita ja tiedostoja, joilla sovelluksen näkymää hallitaan. Monikielisen sivuston luomisen avuksi sovellus voidaan rakentaa niin, että sisällöt kirjoitetaan muuttujien sisään, ja jokaiselle muuttujalle luodaan kielen mukainen käännös. Näiden muuttujien käännökset löytyvät alakansiosta kyseisen kielen tunnuksella. Nämä tiedostot sisältävät jo valmiiksi kirjoitetut virheilmoitukset, joiden kääntäminen onnistuisi pienellä työllä. Näkymät alakansioon luodaan sivujen HTML-tiedostot tai blade-pohjat. Nämä toimivat MVC-rakenteen Views-osana, sillä tänne luodaan tiedostot, joiden avulla saadaan käsittelijästä tuotu tieto näkyviin.

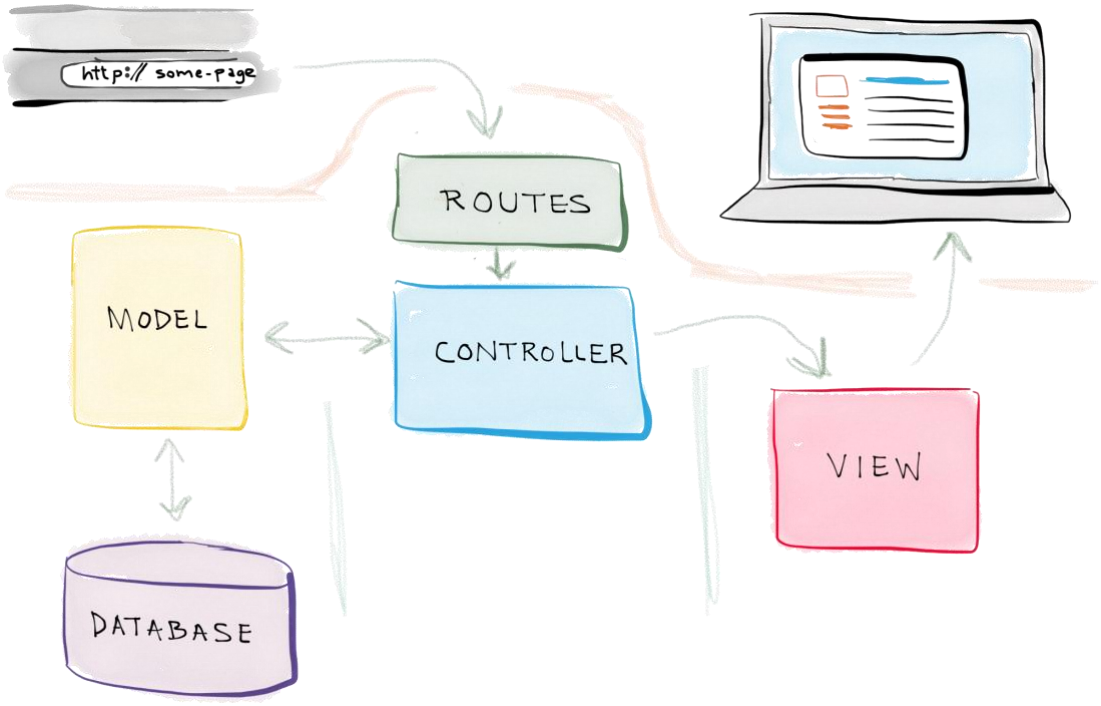
Kaikki Laravelin reititykseen liittyvät tiedostot löytyvät kansiosta Routes. Tätä kautta ohjelmistokehys lataa reitit automaattisesti. Routes-kansion web.php-tiedostossa määritellään verkkosovelluksen reitit. Reitille määritetään näkymän kansio ja tiedosto sekä määritetään, mitä käsittelijää ja käsittelijän funktiota kutsutaan. Samassa tiedostossa voidaan määritellä middleware (suom. väliohjelmisto), joka auttaa sovellusta suodattamaan http-pyyntöjä.

Env (enviroment configuration) -tiedosto sisältää listan verkkopalveluista ja sovelluksen konfiguraatioista, jotka on liitetty Laravel-sovellukseen. Lista koostuu

erinäköisistä muuttujista ja parametreistä, jotka liittyvät sovellukseen itseensä tai ulkopuoliseen palveluun. Näitä ovat esimerkiksi sovelluksen sovellusavain, URL-osoite, tietokannan yhteyden eri muuttujat sekä sähköpostipalvelimen yhteyden tiedot.

2.2.2 MVC

Laravel käyttää hyödykseen ohjelmistotekniikan suunnittelumallia, MVC-arkkitehtuuria. Lyhenne MVC tulee sanoista Model (suom. Malli), View (suom. Näkymä) ja Controller (suom. Käsittelijä). Sen päätarkoituksena on auttaa ohjelmiston kehittäjää pilkkomaan ohjelmisto dataksi, sen käsittelyksi ja käyttöliittymäksi. MVC-mallissa käsittelijä tuo näkymälle datan, jonka se on saanut mallilta tietokannan kautta (Kuva 3).



Kuva 3. MVC-arkkitehtuurin toiminta kuvattuna. (Coleman 2020)

Malli on keskinäinen osa tätä arkkitehtuurityyliä. Sen avulla rakennetaan ja hallitaan sovelluksen dataa, logiikoita sekä erilaisia ohjelmistosääntöjä. Kaikki data, jonka kanssa käyttäjä pääsee työskentelemään, tulee mallin kautta.

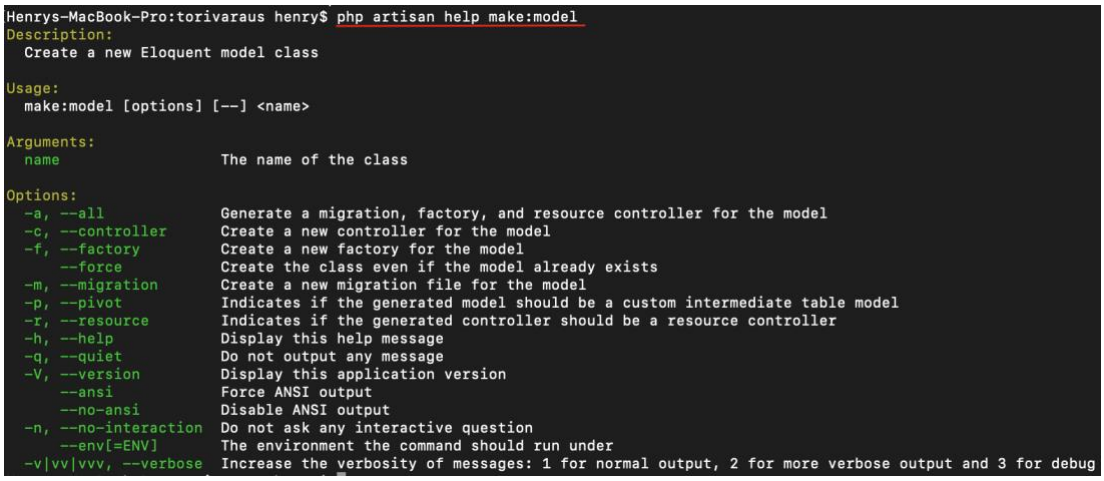
Näkymä on sovelluksen osa, jolla saadaan Mallin ja Käsittelijän sille tuomat tiedot näkyviin käyttäjälle. Näkymä määrittelee sovelluksen ulkoasun, rakenteen sekä paikat, jossa saadut tiedot näytetään. Näkymässä voidaan tehdä muutakin kuin vain näyttää Mallin ja Käsittelijän sille antamia tietoja. Se voi sisältää JavaScriptillä tehtyjä toimintoja tai pieniä viilauksia saatuihin toimintoihin yksittäisillä sivuilla.

Käsittelijä toimii sovelluksen näkymien ja mallien välillä käsitelläkseen kaikki saapuvat pyynnöt (eng. request), suorittaakseen toimintalogiikat sekä käsitelläkseen saadun datan ennen sen eteenpäin siirtämistä. Samassa käsittelijätiedostossa voi monta käsittelijää monelle eri näkymälle.

Hyvin rakennetun ohjelmiston MVC-rakenteesta voidaan saada hyötykäytettyä monia valmiita komponentteja toisiin projekteihin tai sovelluksiin. Tietyllä tavalla rakennetuissa ohjelmistoissa voidaan jopa liittää erillisiä malleja ja kontrollereita toisiinsa vain pienilläkin muutoksilla. Fiksusti ja toiminnallisesti kerran rakennettu MVC-sovellus voi siis toimia ”koodipankkina” moniin samankaltaisiin sovelluksiin.

2.2.3 Artisan

Artisan on komentoriviliittymä (CLI, Command-line interface) joka sisältyy Laraveliin. Se tarjoaa monia sovelluksen kehittämistä helpottavia komentoja, kuten tietokantojen, käsittelijöiden ja mallien luomiseen. Artisan tarjoaa komennolla ”php artisan list” listan kaikista sen komennoista, ja jokaisesta komennosta saa lisätietoa lisäämällä sen eteen ”help”. Esimerkiksi mallin luomisessa ”help”-komennon lisäämisellä saadaan seuraavanlainen lista apukomennoista (Kuva 4).



Kuva 4. Artisan liittymän ”help” komento uuden mallin luomiselle.

Artisanin lisäksi kaikki Laravel-sovellukset sisältävät REPL(Read-Eval-Print-Loop) -konsolin nimeltä Tinker. Se aktivoidaan komennolla ”php artisan tinker”. Tinkerin avulla voidaan hallita Eloquent ORM:ää, tehtäviä sekä tapahtumia. Tinkerin avulla pääsee esimerkiksi näkemään nopeasti halutun tietokantataulun sisällön.

2.2.4 Eloquent ORM

Eloquent ORM (Object Relation Mapper) joka sisältyy Laraveliin, tarjoaa yksinkertaisen ja helpon ”Active Record” -toteutus, joka auttaa tietokannan kanssa työskentelyssä. Jokaista tietokantataulua kohden on sitä vastaava malli, jonka avulla kyseistä taulua hallinnoidaan. Mallin avulla tauluihin voidaan luoda uusia rivejä, hakea, poistaa tai syöttää sinne dataa.

Esimerkiksi olkoon olemassa malli ’Customer’ eli asiakas, ja sille vastaava tietokantataulu ’customers’ eli asiakkaat. Laravelin Eloquentin avulla voidaan hakea dataa asiakkaat tietokantataulusta pelkän mallin avulla, ilman itse kirjoitettua SQL-koodia. Alla esimerkkejä ero hakutyyleistä (Taulukko 1).

Taulukko 1. Eloquentin komentoja Customer-mallin kokoelmalle.

Etsi kaikki asiakkaat	Customer::all()
Etsi asiakas tietyllä id:llä	Customer::find(id)
Poista tietty asiakas	Customer::delete(id)

Eloquent tarjoaa myös eri tietokantataulujen välisten suhteiden hallintaa. Yhden-suhde-yhteen, yhden-suhde-moneen ja monen-suhde-moneen -suhteiden määrittely tapahtuu mallin sisällä yksinkertaisella komennolla.

Yhden-suhde-yhteen voisi olla esimerkiksi aiemmin käytetyn asiakas mallin ja ’CustomersDetails’ eli asiakkaan tiedot -mallin välinen yhteys. Toki asiakkaan tiedot voisi tallentaa asiakas tauluunkin, mutta tietokannan koon hallitsemiseksi etenkin suuremmissa ohjelmistoissa on parempi pilkkoa tietokantaa pienempiin, mutta yksinkertaisiin tauluihin. Kuten esimerkki taulukko 3:ssa on tehty (Taulukko 3).

Jatketaan aiempaa esimerkkiä, mutta lisätään mukaan malli ’Reservation’ eli varaus sekä ’Room’ eli huone. Yhden-suhde-moneen taas olisi asiakkaan ja varauksen välinen yhteys (Taulukko 4). Asiakkaalla voi olla monta varausta, mutta varauksella on vain yksi varaaja eli asiakas. Monen suhde moneen yhteys toteutuu varauksen ja huoneen välillä (Taulukko 5). Yhdessä varauksessa voi olla monta huonetta, ja huone voi kuulua moneen varaukseen.

Taulukko 3. Yhden-suhde-yhteen -funktiot malleissa.

’Customer’ -malli	public function customerDetails() {
	return $this->hasOne(CustomerDetails::class);
	}

’CustomerDetails’ -malli	public function customer() {
	return $this->belongsTo(Customer::class);
	}

Taulukko 4. Yhden-suhde-moneen -funktiot malleissa.

’Customer’ -malli	public function reservations() {
	return $this->hasMany(Reservation::class);
	}

’Reservation’ -malli	public function customer() {
	return $this->belongsTo(Customer::class);
	}

Taulukko 5. Monen-suhde-moneen -funktiot malleissa.

’Reservation’ -malli	public function rooms() {
	return $this->belongsToMany(Room::class);
	}

’Room’ -malli	public function reservations() {
	return $this->belongsToMany(Reservation::class);
	}

Monen-suhde-moneen poikkeaa kahdesta aiemmasta siten, että se vaatii kolmannen taulun toimiakseen. Tämänlaista taulua kutsutaan ’pivot’ -tauluksi, sillä siinä ristiintaulukoidaan kaksi muuttujaa. Pivot-taulun sisällöksi tulevat kaksi saraketta ovat ensimmäisen mallin id- ja toisen mallin id-sarakkeet.

Esimerkiksi kun liitetään huone, jonka id on 12 varaukseen, jonka id on 34, näyttää pivot taulu seuraavalta (Taulukko 6, 7 ja 8).

Taulukko 6. Monen-suhde-moneen -yhteyden pivot taulu.

Room_id	Reservation_id
12	34

Taulukko 7. Monen-suhde-moneen -yhteyden pivot taulu, jossa varaus sisältää useamman kuin yhden huoneen.

Room_id	Reservation_id
22	4
23	4

Taulukko 8. Monen-suhde-moneen yhteyden pivot taulu, jossa huone sisältyy useampaan, kuin yhteen varaukseen.

Room_id	Reservation_id
8	74
8	102

Vaikka huone voi kuulua useaan eri varaukseen, ei se tarkoita, että huone olisi varattuna kahdelle eri varaajalle samaan aikaan.

2.3 Selainkäyttöliittymän määrittely

Laravel käyttää sivujensa ulkonäön määrittelyssä apunaan Bootstrap- ja Vue-tekniikoita, ja tiedon esittämisessä Blade-nimistä PHP-mallimoottoria.

Sivuston päänäkymä ja asettelumalli määritetään “app.blade.php” -nimisessä tiedostossa (Kuva 5), jonne voidaan sisällyttää Bootstrap-versiohakuscriptit tai muut sivuston käyttämät JavaScriptit.

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

<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
        @show

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

Kuva 5. App.blade.php -tyyliasettelutiedoston rakenne. (Laravel www-sivut 2020)

Blade on Laravelin mukana toimitettu yksinkertainen ja tehokas PHP-mallimoottori, joka ei rajoita kirjoittamaan näkymiin vain PHP-koodia. Se poikkeaa muista sillä, että se kokoaa ja välimuistittaa sivunsa, kunnes niitä muutetaan. Blade-syntaksissa voidaan kirjoittaa muuttujia ja muita PHP-toiminteita kaksoisaalotosulkeiden väliin (Kuva 6). Laravel tarjoaa valmiit kokoelman sivutukset, kuten alla olevan kuvan rivillä 9. Sivutuksen määrän voi asettaa käsittelijässä, esimerkiksi 20 asiakasta per. sivu.

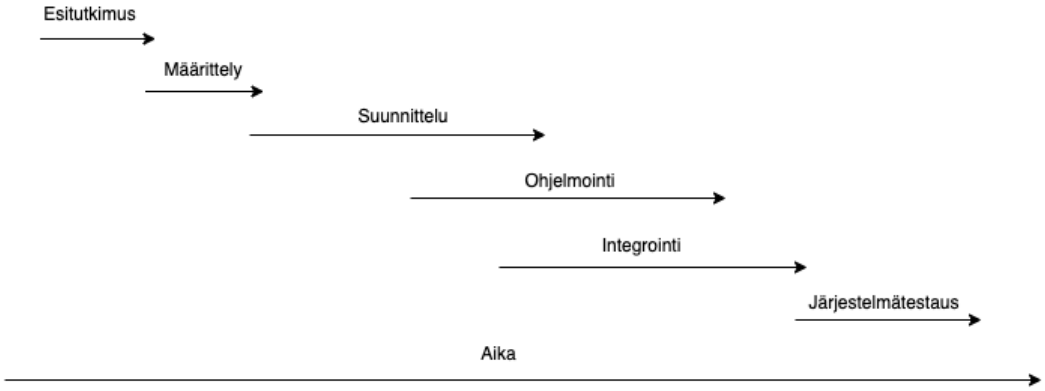
```
1   @extends('layouts.app')
2   @section('content')
3   <div>
4       <ul>
5           @foreach($asiakkaat as $asiakas)
6               <li>{{ $asiakas->nimi }}</li>
7           @endforeach
8       </ul>
9       {{ $asiakkaat->links() }}
10  </div>
11  @endsection
```

Kuva 6. Lista asiakkaista Blade-syntaksin foreach-loopilla.

3 OHJELMISTOTUOTANTO

3.1 Teoriaa

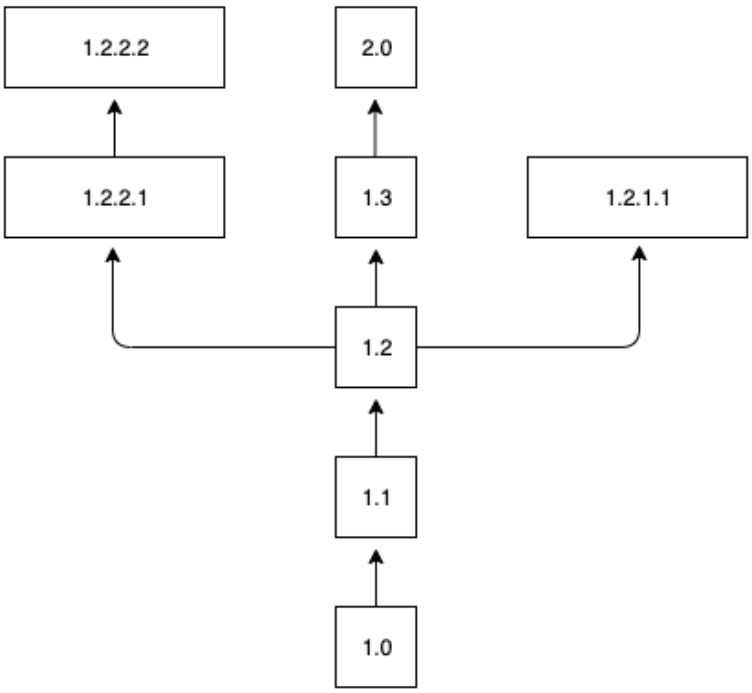
Ohjelmistotuotanto (eng. Software Engineering) tarkoittaa erilaisten tietokoneohjelmistojen tuottamista. Sen eri mallien avulla voidaan luoda ohjelmiston tuottamiselle työkalut, joiden avulla saadaan rakennettua laadukasta ja tehokasta ohjelmistoa mahdollisimmat edullisesti. Sen avulla voidaan seurata ja määritellä hyvinkin tarkasti työprosessin tehtäväjaot sekä työvaiheet projektikohtaisesti (Kuva 6). Osana ohjelmistotuotantoa ovat myös projektinhallinta, ohjelmiston dokumentointi sekä sen versionhallinta.



Kuva 6. Ohjelmistotuotannon projektin kulku.

Käsite ohjelmistotuotannosta nousi pintaan ensimmäisen kerran vuonna 1968 NATOn järjestämässä konferenssikokouksessa, sekä myöhemmin vuonna 1969. Tuolloin aiheena oli ns. ohjelmistokriisi kasvavien ja monimutkaistuvimpien ohjelmistojen vuoksi. Tästä ohjelmistotuotanto alkoi nostamaan päätään. 1970-luvun lopulla kehiteltiin ensimmäisiä ohjelmistoympäristöjä. 1980-luvun alussa julkaistiin ensimmäinen aiheeseen liittyvä oppikirja. 1980-luvun loppupuolella olio-ohjelmointikielten käyttö lisääntyi. 1990-luvun lopulla UML-kielen käyttöönottoa ehdotettiin ja 2000-luvun alussa se levisi jo laajasti. (Sommerville 2008)

Versionhallinnan tarkoituksena on tarjota ohjelmoijille ympäristö, jossa he voivat työstää koodia yhtaikaisesti. Versionhallinnan työkalujen avulla jokainen ohjelmoija voi työstää koodia samaan aikaan ilman, että heidän tekemänsä muutokset vaikuttavat vielä toisen ohjelmoijan koodiin. Täten koodien yhdistämisessä voidaan tarkkailla, tuleeko jotain muuttaa siten, että koodit toimivat toistensa kanssa. Versionhallinta mahdollistaa myös aiempien versioiden tarkkailua, mikäli uusin versio ohjelmistosta aiheuttaa ongelmia tai antaa virheilmoituksia. Ohjelmiston versio voi sisältää eri haaroja, näiden tarkoituksena on toimia esimerkiksi eri alustoille samassa pisteessä olevalla ohjelmistolle selvemmin omat kehitysympäristöt (Kuva 7). (Haikala & Märijärvi 2004, 260)



Kuva 7. Ohjelmiston versiopuu.

Ohjelmistotuotannon dokumentointiin voi kuulua projektisuunnitelma, määrittelydokumentti, suunnitteludokumentti ja testaussuunnitelma. Dokumentointia on myös itse sovelluksen koodiin kirjoitettavat ns. kommentit, joilla monimutkaista ohjelmistoa ja sen komponentteja voidaan avata koodia tarkastelevalle. Kaikkea ohjelmistossa ei tulekaan avata, sillä monet tapahtumat ovat yksiselitteisiä ja koodista itsestään luettavissa. Esimerkiksi monikohtaisessa ehtolauseessa on hyvä avata, että mikä ehto sisältää mitäkin. Kokenut ohjelmoija kykenee varmaan koodia tutkailemalla ymmärtämään, mitä ehtolauseke tekee. Nopeammin se kuitenkin onnistuu lukemalla kommentiksi kirjoitetun ohjeen. (Haikala & Märijärvi 2004, 51)

3.2 Projektissa käytännössä

Projektia tehdessä ei käytetty ketteriä kehitysmenetelmiä. Koska projekti oli ensimmäiseni työelämässä, keskityin ohjelmiston luomiseen kaikesta eniten. Samalla tuli tutuksi versionhallinta sekä ohjelmiston dokumentointi.

Ohjelmiston kehityksessä käytettiin seuraavia työkaluja: Trello-sivustoa projektin tehtävien hallintaan ja seurantaan, Sourcetree-sovellusta versionhallintaan sekä koodin kirjoittamiseen Visual Studio Code 2 -ohjelmaa.

Trellon avulla saatiin listattua projektin eri vaiheita, osa-alueita ja määrityksiä ohjelmistoa varten. Projektissa käytettävät kuvat saatiin haltuun tätä kautta, muun materiaalin kanssa.

Sourcetree mahdollisti usean haaran luomisen ja versioiden hallitsemisen. Jokainen omalla työasemalla luotu muutos piti ”puskea” live-palveluun, jotta muut näkisivät sen. Jokaisesta tämänlaisesta tapahtumasta jäi tarkasteltava sivu, jonka avulla pääsi näkemään edellisten puskujen tehdyt muutokset. Projektissa käytettiin kahta eri haaraa, toista kehityksessä ja toista itse tuotannossa. Tämä mahdollisti luotujen muutosten testaamisen useammalta kuin yhdeltä työasemalta, ennen kuin niitä annettiin asiakkaalle testattavaksi

Visual Studio Code 2 tarjoaa lisäosia, joiden avulla koodin luettavuus helpottuu sitä kirjoittaessa ja korjatessa. Sen virheilmoituksien avulla näkee virheellisesti kirjoitetut syntaksit jo ennen kuin muutokset ehtii tallentamaan ja sivua tarkastelemaan.

4 SOVELLUS

Sovelluksen tarkoituksena oli luoda hallintapuolen käyttöliittymä, jolla voitaisiin luoda, lukea, päivittää ja muokata varauksia, asiakkaita ja tarjolla olevia varattavia alueita ja niiden paikkoja. Sovelluksen luomisen ensiaskeleet olivat siis tietokannan

suunnittelu ja sen rakentaminen. Tämä tuli tehdä huolellisesta ja ajatuksella, sillä koko ohjelman toiminta perustuu käytännössä tietokannan tietojen jatkuvasta käyttämisestä.

Tietokannan pohjan valmistumisen jälkeen päästiin aloittamaan palvelinkoodin kirjoitus, josta seuraava askel sovelluksen kehittämiseen oli mallien luominen vastaamaan tietokannan taulujen rakennetta. Kun mallit oli saatu pohjustettua, voitiin aloittaa käsittelijöiden ja näkymien koodaaminen.

Ennen varsinaista käsittelijöiden koodaamista, tuli tietää, mitä näkymiä sovellukseen tulee. Käsittelijää on turha luoda ja rakentaa sen enempää, ellei tiedä mitä tietoa sen avulla tahdotaan näyttää. Joten aloimme näkymä kerrallaan luomaan niiden käyttöön soveltuvat käsittelijät. Tässä kohtaa sovelluksen kehys onkin jo ”valmis” ja enää jää jäljelle sen viimeistely luomalla asiakkaan tarpeisiin soveltuvat loput toiminnot.

4.1 Tietokanta

Tietokannan rakentaminen alkoi ensin perustarpeiden määrittelystä. Aiemmin asiakkaalta kerättyjen tietojen perusteella loimme taulut asiakkaille, varauksille, myyntialueille sekä myyntipaikoille. Taulujen kentiksi syötimme aluksi perustiedot, joihin projektin edetessä lisäsimme kenttiä sitä mukaan, kun asiakas niitä koki tarpeellisiksi (Kuva 8).

```
14      public function up()
15      {
16          Schema::table('myyntipaikka', function (Blueprint $table) {
17              $table->text('lisatiedot')->nullable();
18          });
19      }
20      /**
21       * Reverse the migrations.
22       *
23       * @return void
24       */
25      public function down()
26      {
27          Schema::table('myyntipaikka', function (Blueprint $table) {
28              $table->dropColumn('lisatiedot');
29          });
30      }
```

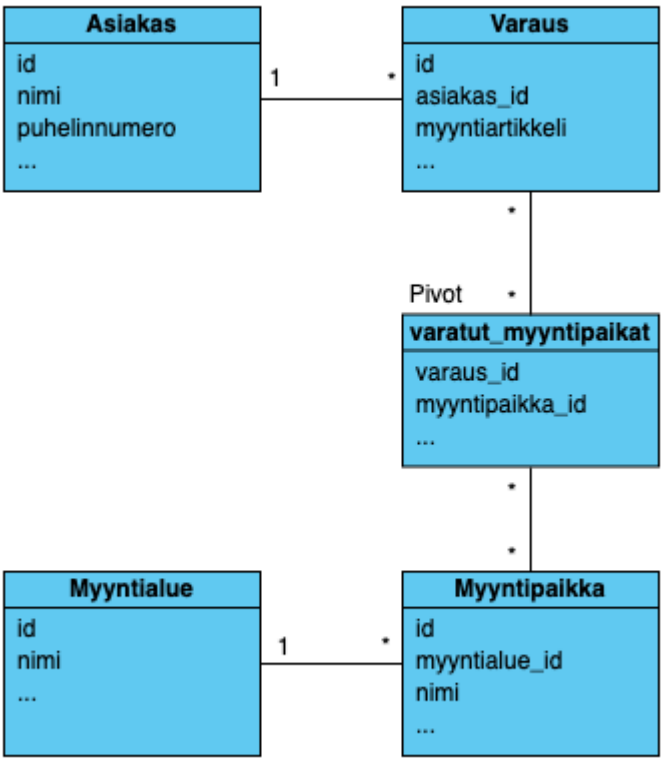
Kuva 8. Olemassa olevan tietokannan tauluun kentän lisääminen jälkikäteen.

Näiden tietokannan taulujen lisäksi sovelluksen toiminnallisuuden kannalta Laravel auttoi meitä luomaan käyttäjien tunnistautumisen hallintaa helpottavat taulut, josta aiemmin mainittiin kappaleessa 2.1. Mahdollisten virhetilanteiden varalta loimme myös taulun, jonne Laravel tallentaa sovelluksessa tapahtuneiden muutosten tiedot. Esimerkiksi asiakastietojen muokkaus tai poisto. Virheen ilmaantuessa, voisimme käydä tietokannasta katsomassa tapahtuman tiedot annetulla päivämäärällä ja kellonajalla. Kokonaisuudessaan tietokantaan tuli siis yhdeksän taulua (Kuva 9).

Table	Rows	Type	Collation	Size	Overhead
activity_log	153	InnoDB	utf8mb4_unicode_ci	112 KiB	-
asiakas	107	InnoDB	utf8mb4_unicode_ci	48 KiB	-
migrations	15	InnoDB	utf8mb4_unicode_ci	16 KiB	-
myyntialue	3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
myyntipaikka	141	InnoDB	utf8mb4_unicode_ci	16 KiB	-
password_resets	1	InnoDB	utf8mb4_unicode_ci	32 KiB	-
users	9	InnoDB	utf8mb4_unicode_ci	32 KiB	-
varatut_myyntipaikat	82	InnoDB	utf8mb4_unicode_ci	16 KiB	-
varaus	46	InnoDB	utf8mb4_unicode_ci	16 KiB	-
9 table(s)	557	InnoDB	latin1_swedish_ci	304 KiB	0 B

Kuva 9. Tietokannan taulut.

Kun tietokannan taulut ja kentät oli saatu alulleen, tuli seuraavaksi miettiä, mitä riippuvuuksia tauluihin tulee luoda. Yhteyksiä taulujen välille tulisi ainakin seuraaviin tauluihin: asiakas ja varaus, myyntialue ja myyntipaikka sekä varaus ja myyntipaikka. Taulujen yhteyksien hahmottamisen helpottamiseksi loin luokkakaavion tauluista ja niiden yhteyksistä (Kuva 10). Kun kollegan kanssa totesimme yhteyksien olevan riittävät, pääsimme toteuttamaan itse tietokannan koodaamisen ja siirtymisen palvelinkoodin tuottamiseen.



Kuva 10. Luokkakaavio tietokannan yhteyksistä.

4.2 Palvelinkoodi

4.2.1 Mallien rakentaminen

Tietokannan taulujen luomisen jälkeen voitiin aloittaa mallien koodin kirjoittaminen. Laravelin toimintamallin mukaan jokaisella tietokannan taululla tulee olla sitä vastaava malli, joten niiden rakentaminen tehtiin sen mukaan.

Mallien koodiin määriteltiin niitä vastaavien tietokannan taulujen nimet, täytettävät kentät, sekä yhteydet toisiin tauluihin ja malleihin (Kuva 11). Malleihin voidaan myös kirjoittaa muita metodeja, joilla voidaan hakea vaikkapa kyseisestä tietokannan taulusta tiettyjä rivejä tietoa, jos useassa tapauksessa näitä tietoja tarvittaisiin (Kuva 12).

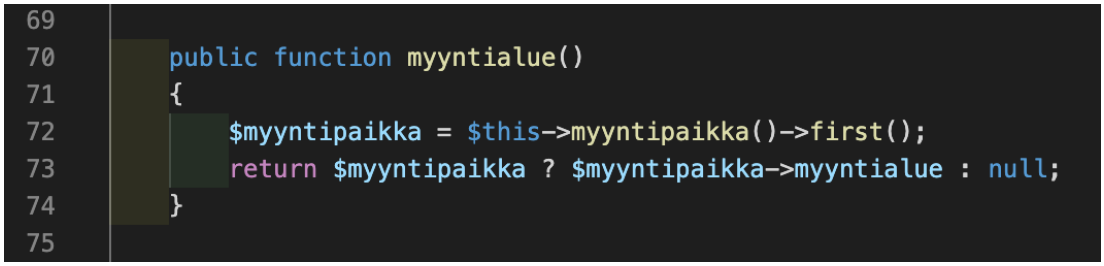
```php
class Myyntipaikka extends Model
{
    protected $fillable = [
        'id',
        'nimi',
        'myyntialue_id',
        'tyyppi',
    ];

    protected $table = 'myyntipaikka';

    public function myyntialue()
    {
        return $this->belongsTo(Myyntialue::class);
    }

    public function varaukset()
    {
        return $this->belongsToMany(Varaus::class, 'varatut_myyntipaikat');
    }
}
```

Kuva 11. Myyntipaikka-malli. Täytettävät kentät, taulun nimi sekä yhteyksien määritys metodit muihin malleihin / tauluihin.



```php

    public function myyntialue()
    {
        $myyntipaikka = $this->myyntipaikka()->first();
        return $myyntipaikka ? $myyntipaikka->myyntialue : null;
    }

```

Kuva 12. Varaus-malliin luotu metodi, joka hakee varaukseen merkityn myyntipaikan myyntialueen.

4.2.2 Käsittelijöiden ja näkymien rakentaminen

Sovelluksen kehittämisen seuraavat vaiheet mallien rakentamisen jälkeen olivat käsittelijöiden, näkymien sekä niiden yhteyksien määrittelyt. Ennen ensimmäisenkään varauksen luomista tuli ohjelmoida asiakastietojen, myyntialueiden sekä myyntipaikkojen hallintaa varten käsittelijät ja näkymät. Ennen kun näitä tietoja päästään hallitsemaan, on turha luoda varausjärjestelmää itsessään.

Näiden kolmen mallin käsittelijöiden ja näkymien ohjelmointi tapahtui samalla kaavalla. Ensin luodaan funktio tietojen luomiseen sekä muokkaamiseen. Nämä toiminnot voidaan kirjoittaa yhteen funktioon koodin selkeyttämiseksi (Kuva 13). Tietojen luomisen ja muokkaamisen hallinnan rakennus yhdelle sivulle säästää aikaa, mikäli taulun rakenteeseen tulisi muutoksia. Tällöin muutoksia lomakkeisiin ja funktioihin ei tarvitse kirjoittaa kahteen kertaan, ja mahdollisilta virheiltä säästytään.

```
90      public function muokkaa(Request $request, Asiakas $asiakas = null)
91      {
92          if (is_null($asiakas)) {
93              $asiakas = new Asiakas();
94          }
95          if ($request->isMethod('post')) {
96              if ($request->has('delete')) {
97                  $asiakas->delete();
98                  return redirect()->to('/yritykset');
99              }
100             $request->validate([
101                 'ynimi' => 'nullable|string|max:255', …
122             ]);
123             $asiakas->ynimi = $request->input('ynimi'); …
145             $asiakas->save();
146             return redirect()->to('/yritykset');
147         }
148         return view('yritys.muokkaa', ['asiakas' => $asiakas]);
149     }
```

Kuva 13. Asiakkaan luominen ja olemassa olevan asiakkaan tietojen päivittäminen samassa funktiossa.

Kun käsittelijään on kirjoitettu funktio tietojen luomista ja muokkausta varten, voitiin luoda näkymätiedosto, jonka avulla tiedot saataisiin tälle käsittelijälle (Kuva 14). Ennen kuin näkymään itsessään päästään käsiksi, tulee määrittää reititystiedostoon reitti käsittelijän ja näkymätiedoston välille (Kuva 15).

```
<div class="card-header">Asiakkaan {{ $asiakas->exists ? 'muokkaaminen' : 'lisääminen' }}</div>
<div class="card-body">
    <form action="{{ url()->current() }}" method="post">
        @if ($errors->any())
            <div class="alert alert-danger" role="alert">
                Please fix the following errors
            </div>
        @endif
        @csrf
        <div class="row">
            <div class="form-group{{ $errors->has('ynimi') ? ' has-error' : '' }}">
                <label for="ynimi">Yrityksen nimi</label>
                <input  type="text" class="form-control" id="ynimi"
                        name="ynimi" value="{{ old('ynimi') ?? $asiakas->ynimi }}">
                @if($errors->has('ynimi'))
                    <span class="help-block">{{ $errors->first('ynimi') }}</span>
                @endif
            </div>
            <div class="form-group{{ $errors->has('yhenkilo') ? ' has-error' : '' }}">…
        </div>
        <button type="submit" class="btn btn-md btn-primary">Tallenna</button>
        <input  type="submit" name="delete" onclick="return confirm('Haluatko varmasti poistaa asiakkaan?');"
                value="Poista" class="btn btn-danger" />
    </form>
</div>
```

Kuva 14. HTML-näkymä asiakkaan tietojen luomiselle / muokkaamiselle.

```
Route::any('/yritykset/lisaa', 'YritysController@lisaa');
Route::get('/yritykset/{yritys}', 'YritysController@details');
Route::any('/yritykset/{yritys}/muokkaa', 'YritysController@muokkaa');
```

Kuva 15. Reititystiedoston reittimäärityksen asiakkaan hallintaan.

Kun mallin tietojen luominen ja muokkaaminen on kunnossa, luodaan käsittelijään funktiot sekä näkymät olemassa olevan tiedon listaukseen sekä tarkasteluun (Kuva 16).

```
39      public function list(Request $request)
40      {
41          // Haetaan kaikki tietokannasta löytyvät asiakkaat ja järjestetään nimen perusteella.
42          $asiakkaat = Asiakas::orderBy('ynimi', 'ASC');
43
44          // Palautetaan näkymän tiedosto, sekä lista asiakkaista.
45          // Käytetään Laravelin tarjoamaa 'paginate' metodia, joka sivuttaa tietokannasta haetut tiedot.
46          // Määritetään metodi näyttämään 25 asiakasta / sivu.
47          return view('yritys.list', ['asiakkaat' => $asiakkaat->paginate(25) ]);
48      }
49      public function details(Request $request, Asiakas $asiakas)
50      {
51          return view('yritys.details', [ 'asiakas' => $asiakas ]);
52      }
53      public function muokkaa(Request $request, Asiakas $asiakas = null)
54  >   {…
112     }
```

Kuva 16. Asiakas-mallin tietojen käsittelijän funktiot: asiakkaiden listaus, asiakkaan tietojen esittäminen sekä asiakkaan tietojen luominen / muokkaaminen.

Kun asiakkaiden, myyntialueiden ja myyntipaikkojen hallintaan liittyvät toiminteet olivat kunnossa, pääsimme kehittämään varausten tekemistä. Yksi varaus ikään kuin solmii nämä kolme muuta mallia yhteen itsensä kanssa.

Varauksen käsittelijään luotiin samat kolme funktiota, mitä aiempiin malleihin. Periaate oli jälleen sama, mutta muutamalla lisätoiminnolla. Varauksia luodessa ja muokatessa tuli huomioida kaikki muutkin järjestelmässä olevat varaukset, ettei yhdelle myyntipaikalle tulisi tehtyä päällekkäisiä varauksia. Koska varaukselle tuli määritellä alkamis- ja loppumisaika, piti huolehtia myös siitä, ettei alkava varaus voi loppua ennen alkamispäivämäärää. Päällekkäin tehtävien varausten tarkistus toteutettiin ehtolauseilla, jossa luodaan lista varauksista, joiden kanssa valitut päivämäärät menevät päällekkäin (Kuva 17).

```php
if (empty(old('alkaa')) && empty($varaus->alkaa)) {
    $varatutPaikat = [];
} else {
    if (!empty(old('alkaa') && !empty(old('loppuu')))) {
        $alkaa = Carbon::createFromFormat('j.n.Y G:i', old('alkaa'));
        $loppuu = Carbon::createFromFormat('j.n.Y G:i', old('loppuu'));
    } else {
        $alkaa = $varaus->alkaa;
        $loppuu = $varaus->loppuu;
    }
    $varatutPaikat = Myyntipaikka::whereHas('varaukset', function ($q) use ($varaus, $alkaa, $loppuu) {
        $q->where('varaus.id', '!=', $varaus->id)
        ->where('varaus.alkaa', '<', $loppuu)
        ->where('varaus.loppuu', '>', $alkaa);
    })->select([ 'id' ])->get()->map(function ($item) {
        return $item->id;
    })->all();
}
```

Kuva 17. Myyntipaikkojen hakeminen tietokannasta, joiden päivämäärät osuvat näkymäpuolella valittuihin.

Näkymäpuolella varattu myyntipaikka merkittiin ehtolauseella, jos kyseinen paikka olisi käsittelijässä luodussa listassa ‘varatutPaikat’, estettäisiin paikan valinta kyseisillä päivämäärävalinnoilla varattuna (Kuva 18).

```
<label for="paikkanumero">Paikkanumero</label>
<div class="form-group" id="paikkanumero">
    @foreach($myyntialueet AS $myyntialue)
        @foreach($myyntialue->myyntipaikat AS $myyntipaikka)
            <label>
                <input ⋯
                />
                {{ $myyntipaikka->nimi }}
                @if(in_array($myyntipaikka->id, $varatut_paikat))
                    <span class="badge badge-danger">varattu ⋯
                    </span>
                @endif
            </label>
        @endforeach
    @endforeach
</div>
```

Kuva 18. Varauksen luomisen / muokkaamisen näkymän ehtolauseke varatulle myyntipaikalle.

Päivämäärien valitsemiseen käytimme JavaScript-kirjastoa nimeltä ’Date Range Picker’. Tämän kirjaston käyttö mahdollisti visuaalisesti ja käytännöllisesti näyttävän ja yksinkertaisen, mutta hyvin hallittavan kalenterinäkymän käytön. Jokaisen päivämäärävalinnan jälkeen ajettiin käsittelijän funktio uudelleen läpi, jossa se tarkisti mahdolliset päällekkäisyydet.

4.3 Käyttöliittymä

Varausjärjestelmän tarkoituksena oli luoda hallintapuolen järjestelmä, täten käyttöliittymän tuli olla selkeä ja tarpeeksi yksinkertainen. Visuaalista puolta ei tähän versioon lähdetty tekemään, koska varausten tekemisen avaus yleisölle oli suunniteltu seuraavaan versioon.

Hallintapuolen tuli olla kuitenkin vuorovaikutteinen ja selkeä. Bootstrap, joka on avoimen lähdekoodin CSS-ohjelmistokehys, tarjoaa käyttöliittymän visuaalisen puolen kehittämiseksi tarpeelliset työkalut. Bootstap mahdollistaa myös vuorovaikutteinen sisällytyksen ulkonäköön ilman, että sen eteen tarvitsi kirjoittaa ylimääräistä koodia.

Visuaalinen puoli ei lisännyt sovelluksen kehittämisen tuntimäärää paljoakaan. Käsittelijöiden kanssa samaan aikaan rakennetut näkymät kirjoitettiin siten, että tiedot luettavuudessa käytetyt elementit käyttivät Bootstrap-luokkia.

5 YHTEENVETO

Projektin tavoitteena oli luoda yksinkertainen hallintajärjestelmä erilaisten tietojen ylläpitoon ja se tavoite saavutettiin. Tietotaitoni sovelluksen rakentamiseen käytetyistä eri asioista oli hyvinkin vähäiset, mutta projektin edetessä opin todella paljon uutta kaikkeen ohjelmiston tuottamiseen liittyvään tekemiseen. Lopputulokseen olen samaan aikaan tyytyväinen sekä tyytymätön. Tyytyväinen siksi, että sovellus täytti odotetut vaatimukset. Mutta tyytymätön siksi, että ohjelmiston kehitys ei saanut jatkoa, enkä täten saanut syytä päästä parantelemaan aiemmin kirjoittamaani koodia.

Jos oppimiskäyrääni projektin edetessä olisi mitattu, olisi siinä näkynyt huima nousukäyrä. Opin uutta ohjelmiston kehittämisen jokaisesta näkökulmasta, ja olen näillä opeilla päässyt rakentamaan uusia ohjelmistoja kyseisen projektin jälkeen. Jos rakentaisin tämän varausjärjestelmän uudestaan tällä tietotaidolla, joka projektin myötä karttui, tulisi luettavasta koodista hyvin erinäköistä ja selkeämpää.

Työtä tehdessäni huomasin myös omasta oppimisestani sen, että parhaiten opin tekemällä. Omasta mielestäni projekti sattui olemaan juuri sopiva ensimmäiseksi ohjelmistoprojektikseni. Se sisälsi paljon eri työalueen tehtäviä ja haastoi minut oppimaan, muttei ollut ylitsepääsemättömän vaikeaa.

Vaikka projekti keskeytyi yhteystyön päättymisen vuoksi, olen tyytyväinen siihen, että projekti osui kohdalleni. Sen avulla pääsin ohjelmiston kehittämisen maailmaan käsiksi ja hyödyntämään matkan varrella saatujani oppeja.

LÄHTEET

Akshay Patil 2019. ’10 reasons why Laravel is the best PHP framework for 2019’. Blog. 2019. Viitattu 28.12.2019. https://www.clariontech.com/blog/10-reasons-why-laravel-is-the-best-php-framework-for-2019

Alex Coleman. ’Creating a basic Laravel 5 MVC application in 10 minutes’. Blog. 2020. Viitattu 12.1.2020. https://selftaughtcoders.com/from-idea-to-launch/lesson-17/laravel-5-mvc-application-in-10-minutes/

Haikala, I. & Märijärvi J. 2004. Ohjelmistotuotanto. Helsinki: Talentum.

Ian Sommerville. ‘Software engineering history’. Blog. 2008. Viitattu 3.2.2020. https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/History/index.html

Kenneth Walley. ‘Top 6 most used PHP frameworks for web development 2020’. Blog. 12.11.2019. Viitattu 28.12.2019. https://dev.to/websitedesignnj/top-6-most-used-php-frameworks-for-web-development-2020-46o5

Laravel www-sivut 2020. Viitattu 19.12.2019. https://laravel.com/

Maks Surguy 2013. ‘History of Laravel PHP framework, Eloquence emerging’. Code Blog. 27.7.2013. Viitattu 6.1.2020. https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/

PHP www-sivut 2020. Viitattu 14.3.2020. https://www.php.net/manual/en/history.php.php

W3Techs www-sivut 2020. Viitattu 14.3.2020. https://w3techs.com/technologies/details/pl-php