Markus Mäkinen

Yksityisen pilvitallennuspalvelun luominen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan koulutusohjelma

Insinöörityö

9.5.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Markus Mäkinen Yksityisen pilvitallennuspalvelun luominen 35 sivua 9.5.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikan koulutusohjelma
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Outi Grotenfelt

Insinöörityön aiheena oli rakentaa oma pilvitallennustila-sovellus, jota pystyi käyttämään selaimella julkisen verkon yli. Sovellus toteutettiin Amazonin AWS-alustalla ja palvelimeksi valittiin Ubuntu-palvelin.

Itse sovellus tallennustilan ja tiedostojen hallinnointiin tehtiin käyttäen Laravel PHP-sovelluskehystä ja useita javascript-kirjastoja. Dropzone.js:ää käytettiin tiedostojen lataamiseen, Datatablesia käytettiin tiedostojen listaamiseen ja sorttaamiseen ja Justified.js:ää sekä Blueimp-galleryä käytettiin sovelluksen galleriaosuutta ja kuvien selaamista varten.

Insinöörityön lopputuotoksena syntyi verkon yli toimiva tallennustilasovellus, joka toimii AWS-palvelimella.

Avainsanat	AWS, Laravel, pilvipalvelu, sovelluskehitys, Ubuntu

Author	Markus Mäkinen
Title	Creating a private cloud storage service
Number of Pages	35 pages
Date	9 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Software Engineering
Instructors	Outi Grotenfelt, Lecturer

The purpose of this thesis was to build a private cloud storage software and make it accessible to be used outside of the home network. The software was deployed to Amazon AWS platform and the server used was a Ubuntu server.

The software for managing the cloud storage and files was built using Laravel PHP-framework and various javascript libraries. Dropzone.js was used for uploading the files, Datatables was used for listing and sorting the files on the front end and Justified.js and Blueimp gallery were used for the gallery section and browsing the images.

The result of this thesis was a working cloud storage software, installed on AWS, and that was usable over the public network.

Keywords	AWS, cloud service, Laravel, software development, Ubuntu

Sisällys

Lyhenteet

CSRF	Cross-Site Request Forgery, hyökkäys, jossa hyökkääjä saa uhrin selaimen lähettämään http-pyynnön hyökkääjän haluamaan kohdepalveluun.
HTTP	Hypertext Transfer Protocol, protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon.
IaaS	Infrastructure as a service, palvelimien ja palvelinsalien ulkoistaminen yritykselle, joka mahdollistaa palveluiden tuottamisen ja olemassaolon tarjoamalla laskentatehoa ja tallennustilaa.
LTS	Long term support, palvelimen käyttöjärjestelmän versio, jolle on luvattu tehdä tietoturvapäivityksiä vuosiksi eteenpäin.
NPM	Node.js Package Manager, Node.js:n oletuspaketinhallintajärjestelmä.
ORM	Object-relational mapping, oliomallin mukaisen esityksen kuvaus relaatiomallin mukaiseksi esitykseksi.
SSH	Secure Shell, salattuun tietoliikenteeseen tarkoitettu protokolla.
URI	Uniform Resource Identifier, merkkijono, jolla kerrotaan tietyn tiedon paikka (URL) tai yksikäsitteinen nimi (URN).

1 Johdanto

Insinöörityön tarkoituksena on opastaa oikean maailman tarpeisiin vastaavan web-sovelluksen tekoon. Projekti sisältää uusimpien web-teknologioiden työkaluja ja työskentelytapoja ja opastaa niiden asentamisessa sekä käyttämisessä pilvitallennustila-esimerkkiprojektin avulla.

Tässä insinöörityössä käydään läpi Laravel-sovelluksen kehitysympäristön asentaminen, hyödyllisten kirjastojen lisääminen ja esitellään sovelluksen käyttöä. Näiden lisäksi käydään läpi vielä Ubuntu-palvelimen asentaminen ja konfigurointi Amazonin AWS-alustalle.

Itse pilvitallennustila-projektissa voidaan luoda käyttäjiä, ladata tiedostoja ja kuvia palveluun, luoda niille kategorioita sekä muokata, ladata tai poistaa niitä. Tiedostojen näkyvyys toteutetaan siten, että vain tiedostojen lataajalla on oikeus nähdä ja tehdä toimintoja niille, myös kategorioista luodaan käyttäjäkohtaisia. Tässä projektissa tiedostot ladataan palvelimelle samaan kansioon, joten käyttöoikeuksien hallinta ohjelmiston tasolla on tässä projektissa olennaista. Todennukseen (authentication) käytetään sovelluskehykseen sisään rakennettua ominaisuutta, jolla luodaan kaikki käyttäjän rekisteröitymiseen ja kirjautumiseen vaadittava.

Mikäli kaikki suunnitellut ominaisuudet saadaan toimivaksi, tehdään tiedostoille vielä niiden jakomahdollisuus, jolloin käyttäjä voi itse valita, minkä tiedoston haluaa linkillä jakaa ulkomaailmaan.

2 Pilvipalvelut

Pilvipalvelulla tarkoitetaan sitä, kun palvelut tai tiedostot eivät ole fyysisesti omalla tietokoneella tai yrityksen omalla palvelimella, vaan ne ovat pilvipalvelun tarjoavalla yrityksellä. Tiedostoihin ja ohjelmiin pääsee käsiksi internetin kautta lähes kaikkialta ja milloin vain. Pilvi tarkoittaa käytännössä palvelimien verkostoa. [1.] Pilvipalvelut varmuuskopioivat usein niihin ladatut tiedostot automaattisesti osana palvelua ja tarjoavat niiden synkronointimahdollisuutta. Pilvipalvelut antavat usein mahdollisuuden tiedostojen julkiseen jakamiseen joko erillisellä kutsulinkillä tai sähköpostiosoitteen avulla.

Pilvipalveluista ja niiden turvallisuudesta on pitkään käyty keskustelua, mutta nykyään yleinen näkemys tuntuu olevan, että on turvallisempaa säilyttää tiedostoja pilvipalvelussa kuin omassa, suljetussa tietovarastossa. Oman näkemykseni mukaan syitä tähän on, että varmuuskopiointi koetaan liian työlääksi ja oma tekninen tietotaito ei usein riitä siihen, että tiedostojen uskotaan olevan turvassa. Koetaan turvallisemmaksi siirtää vastuu tiedostojen ja palveluiden toiminnassa säilymisestä ulkoiselle toimijalle, tiedostojen hallintaan erikoistuneelle yritykselle ja tästä ollaan myös valmiita maksamaan. Pilvipalveluiden hinnat ovat myös nykyään sillä tasolla, että usein tulee jopa halvemmaksi ulkoistaa tiedostojen säilytys kuin säilyttää niitä omalla palvelimella.

3 Pilvitallennustila

Pilvitallennustilaa tarjoavat suuret teknologiajätit, kuten Microsoft (OneDrive), Google (Drive), Amazon (Drive), Apple (iCloud) ja Dropbox omalla nimikkotallennustilallaan. Amazon poikkeaa hinnoittelupolitiikallaan muista pilvitallennustilan tarjoajista ja se laskuttaa käyttäjiään tallennustilan käytöstä vuosittain, ilman mahdollisuutta kuukausittaiseen laskutukseen. Vertailussa sille on annettu kuukausittaista laskutusta vastaavat arvot ja muut vertailussa olevat palvelut ovat valmiiksi kuukausihinnoiteltuja. Huomionarvoista vertailussa on, että Applen iCloud-palvelu on yhteensopiva ainoastaan Applen laitteiden kanssa.

Taulukko 1. Yksityishenkilöille tarjottavien pilvitallennustilojen hintojen vertailua.

Tallennus-tila	**OneDrive**	**Google Drive**	**iCloud**	**Dropbox**	**Amazon Drive**
Maksuton	5 Gt	15 Gt	5 Gt	2 Gt	5 Gt
Pieni	50 Gt, 2€/kk,	100 Gt, 1,99€/kk	50 Gt, 0,99€/kk,	-	100 Gt, 0,81€/kk
Keskisuuri	1 Tt, 7€/kk	1 Tt, 9,99€/kk	2 Tt, 9,99€/kk	1 Tt, 9,99€/kk	1 Tt, 4,07€/kk

Taulukosta 1 käy ilmi, että Google tarjoaa ilmaiseksi suurimman tallennustilan ja se on lisäksi alustasta riippumaton, vastaavasti Dropbox tarjoaa pienimmän ilmaisen tallennustilan.

Vertailuun otettiin lisäksi oman palvelimen ylläpitämisestä koituneet kustannukset sen käyttämän sähkön osalta, sillä tätä vaihtoehtoa harkittiin ja myös testattiin projektin

aikana. Hinta palvelimen ylläpitämiselle on laskettu niin, että palvelinta pidettäisiin päällä 24 tuntia vuorokaudessa ympäri vuoden. Pidetään sähkön hintana Suomen kotitalous-sähkön keskihintaa 12 senttiä/kWh. Sähkölaitteen käyttämä sähköenergia kW saadaan kertomalla laitteen teho P käytetyllä ajalla t. Energia E on näin ollen = Pt.

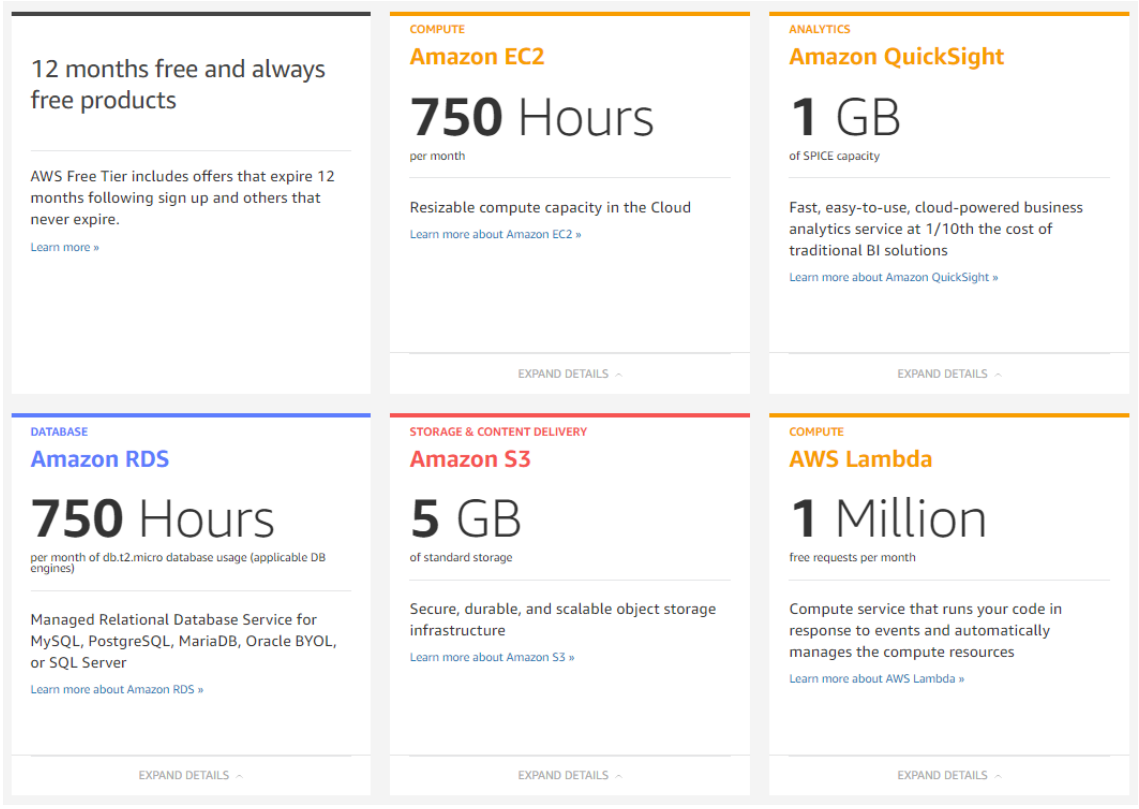
Laitteen teho on kulutusmittarilla mitattuna 110W, eli sen käyttämä sähköenergia tunnin aikana on 0,11 kW. Vuorokausikulutukseksi laitteelle muodostuu 0,11kW * 24h = 2,64kWh. Vuorokaudessa hintaa palvelimen ylläpitämiselle sähkön osalta kertyy tällöin 2,64kWh * 0,12€ = 0,3168€ ja vuodessa 0,3168€ * 365 = 115,632€.

Vertailuun on otettu mukaan Vattenfallin arvio kannettavan tietokoneen käyttämisestä. Hinta-arvio sisältää sähkön myynnin, siirron ja verot. Hinta on 12 snt/kWh, joka vastaa kotitaloussähkön keskihintaa Suomessa. [2.] Voimatorin vastaava arvio on laskettu hinnalla 14 snt/kWh. [3.]

Taulukko 2. Kannettavan tietokoneen sähkönkulutuksesta koituvat kustannusarviot.

Kannettava tietokone	Kulutus kWh/tunti	Hinta snt/tunti	Tuntia/pvä	€/vuosi
Vattenfallin arvio	0,03	0,12	24	31,52
Voimatorin arvio	-	0,14	24	31,89
Oma arvio	0,11	0,12	24	115,6

Huomataan, että vaikka palvelimen eli tässä tapauksessa kannettavan tietokoneen näyttö on pois päältä, on energiankulutus varsin suuri. Tuloksiin vaikuttaa kannettavien tietokoneiden tehoerot, mutta tämän suuruisen kulutuksen omaavasta palvelimesta joutuisi maksamaan yli 115 euroa vuodessa. Projektissa päädyttiin käyttämään Amazonin AWS:ää, joka tarjosi uudelle asiakkaalle vuodeksi ilmaisen palvelin-instanssin ja tallennustilaa.



Kuva 1. Amazonin ilmaisen tason palveluita ja rajoituksia.

Amazon tarjosi ilmaiseen käyttöön EC2 T2-mikro-instanssin, johon oli mahdollista asentaa oma palvelin. Instanssilla on rajoituksena 750 tuntia käyttöaikaa kuukaudessa. Tämä tarkoittaa sitä, että yhtä palvelininstanssia voi pitää päällä 24h vuorokaudessa vuoden ajan.

Projektissa testattiin lisäksi Amazonin S3-tallennustilapalvelua, jossa rajoituksena oli 5Gb datan siirtoa kuukaudessa.

4 Sovelluskehys

Projektissa käytettiin Laravel PHP-sovelluskehystä, joka tarjosi hyvän kehitysalustan tämän kaltaisen web-sovelluksen tarpeisiin. Laravel tarjoaa kevyempiä sovelluskehys-versioita mm. Api-käyttöön (Lumen) sekä MacOS-käyttöön (Valet), mutta tämän projektin tarpeisiin normaali asennus Laravelista oli hyödyllisin, sillä projekti on ns. ”Full-stack” eli sitä varten luotiin tietokanta, mallit, kontrollerit ja näkymät.

Laravel käyttää malleja varten omaa ORM:ää, jota kutsutaan Eloquentiksi. Näkymiä varten sovelluskehys käyttää oletuksena Blade-templateja, joilla voidaan rikastaa html-sivuja ketterästi erinäisin funktioin. Funktioilla voidaan mm. rajata käyttäjän oikeutta nähdä tiettyjä sivuja tai niissä näytettävää dataa, mikä oli tässä projektissa tärkeää. Projektiin käytettiin Laravelin sen hetkistä uusinta versiota 5.6.

4.1 Työympäristön asennus

Ennen työympäristön asentamista täytyi selvittää mitä kaikkea Laravel 5.6 vaati palvelimelta toimiakseen [4.]. Vaatimuksena olivat seuraavat:

- PHP >= 7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension.

Riippuvuuksien hallintaan Laravel hyödyntää Composeria ja myös sen asentaminen oli vaadittua. Composer on sovelluskohtaiseen käyttöön tarkoitettu paketinhallintamanageri PHP:lle. Sovellus luotiin käyttämällä Composerin ”create-project” -komentoa.

```
composer create-project --prefer-dist laravel/laravel <projektin nimi>
```

Esimerkkikoodi 1. Komento Laravel-projektin luomiseen käyttäen Composeria.

Sovelluksen kehitykseen käytettiin käyttöjärjestelminä niin Windowsia, kuin MacOS:ää. Windowsille helpoin ratkaisu oli asentaa Laravel Homestead. Homestead on käytännössä virtuaalinen Ubuntu-palvelin, joka täytti kaikki aiemmin luetellut vaatimukset palvelimen osalta. Toimiakseen Homestead vaati Vagrantin, jolla pystyttiin hallinnoida virtuaalisia kehitysympäristöjä. Jotta virtuaalisen kehitysympäristön sai ylipäätään päälle, tarvittiin vielä lisäksi joko VirtualBox, WMWare, Parallels tai Hyper-V. Tässä projektissa käytettiin Virtualboxia. Vagrantin ja Virtualboxin asentaminen oli niin suoraviivaista, etten kokenut tarpeelliseksi kertoa niiden asentamisesta tarkemmin. Niiden asentamisessa seurattiin Laravelin dokumentaatiota aiheesta. [5.]

Homesteadin mukana tulevat ohjelmat:

- Ubuntu 16.04
- Git
- PHP 7.2
- PHP 7.1
- PHP 7.0
- PHP 5.6
- Nginx
- Apache (Optional)
- MySQL
- MariaDB (Optional)
- Sqlite3
- PostgreSQL
- Composer
- Node (With Yarn, Bower, Grunt, and Gulp)
- Redis
- Memcached
- Beanstalkd
- Mailhog
- Elasticsearch (Optional)
- ngrok
- wp-cli
- Zend Z-Ray
- Go.

Homestead-ympäristö asennettiin kloonaamalla sen repository käyttämällä komentotulkkia ja se asennettiin käyttäjän kotihakemiston juureen.

```
git clone https://github.com/laravel/homestead.git ~/Homestead
```

Esimerkkikoodi 2. Homesteadin kloonaaminen.

Kun repository oli kloonattu, käyttöjärjestelmästä riippuen ajettiin komento, joka loi Homesteadin konfiguraatiotiedoston Homestead.yaml:n. Konfiguraatiotiedosto tallettui juuri luotuun Homestead-kansioon.[6.]

```
// Windows...
init.bat

// Mac / Linux...
bash init.sh
```

Esimerkkikoodi 3. Komento Homestead.yaml-konfiguraatiotiedoston luomiseen.

Kuvassa 2 näkyvässä *Homestead.yaml* -esimerkissä määritellään mm. Vagrant-tarjoaja, kansiot, jotka synkronoidaan Homestead-ympäristön kanssa sekä sivustot, jotka synkronoidaan paikallisesta projektikansiosta Homestead-ympäristöön.

```
---
ip: "192.168.10.10"
memory: 2048
cpus: 1
provider: virtualbox

authorize: ~/.ssh/id_rsa.pub

keys:
    - ~/.ssh/id_rsa

folders:
    - map: ~/Code
      to: /home/vagrant/Code

sites:
    - map: homestead.app
      to: /home/vagrant/Code/Laravel/public
    - map: miniupload.site
      to: /home/vagrant/Code/miniupload/public
```

Kuva 2. Esimerkki Homestead.yaml -konfiguratiotiedostosta.

Kun Homestead-ympäristö oli saatu konfiguroitua, täytyi vielä lisätä käyttöjärjestelmän *hosts*-tiedostoon domain eli verkkotunnus kyseiselle projektille. Tällä saatiin aikaan se, että kun navigoitiin selaimella sille määrättyyn osoitteeseen, löytyi juuri asennettu Laravel-projekti sieltä.

```
// Windows...
C:\Windows\System32\drivers\etc\hosts

// Mac / Linux...
/etc/hosts
```

Esimerkkikoodi 4. Hosts-tiedoston polut eri käyttöjärjestelmissä.

Hosts-tiedostoon lisättiin sama ip-osoite kuin Homestead.yaml-tiedostoon ja sen jälkeen url-osoite, josta projektin haluttiin selaimella löytyvän.

```
192.168.10.10  miniupload.site
```

Esimerkkikoodi 5. Hosts-tiedostoon lisätty rivi.

Halutessaan Homesteadin voi asentaa myös MacOS:lle, mutta sille on olemassa myös kevyempi, vähemmän tilaa vievä Valet, jota tässä projektissa myös käytettiin silloin, kun ohjelmistoa kehitettiin Macillä. Valet ei tarvitse toimiakseen Vagrantia, Homesteadia eikä Virtualboxia vaan se käyttää ainostaan Nginx:ää ja DnsMasq:a, joka ohjaa kaikki *.test-url:iin tehdyt kutsut Valetilla määritettyyn projektikansioon. Näin ollen myöskään hosts-tiedostoon ei tarvinnut koskea.

Laravel-projekteihin luodaan tyypillisesti *.env*-niminen tiedosto, johon määritellään projektikohtaisia muuttujia, kuten sivuston nimi, sivuston tyyppi eli onko se kehityksessä vai tuotannossa, debuggauksen taso, tietokantayhteyden tyyppi ja tiedot sekä tarpeen tullen esim. Sähköposteihin ja Amazonin S3-tallennustilaan liittyvät konfiguraatiot. Tämä on tiedosto jota ei kuulu lisätä versionhallintaan sen sisältämien tunnusten ja salasanojen takia.

```
APP_NAME=Miniupload
APP_ENV=local
APP_KEY=base64:dWA1TDF21NXr321rlasReLC0p84wCnImr6Q=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=192.168.10.10
```

```
DB_PORT=3306
DB_DATABASE=<tietokannan nimi>
DB_USERNAME=homestead
DB_PASSWORD=secret
```

Esimerkkikoodi 6. Projektin .env-tiedoston keskeiset kohdat.

4.2 Artisan

Artisan on Laravelin mukana tuleva komentoliittymä (command line interface), joka tarjosi paljon hyödyllisiä komentoja sovelluksen kehitykseen. Kaikki sen sisältämät komennot sai listattua ”*php artisan list*” -komennolla. Tässä projektissa käytännöllisimpiä komentoja olivat:

```
php artisan route:list
```

- Listasi kaikki ohjelmiston reitit ja niitä vastaavat metodit.

```
php artisan migrate
```

- Teki tietokantamigraation.

```
php artisan migrate:fresh
```

- Tyhjensi tietokannan ja ajoi tietokantamigraatiot uudelleen.

```
php artisan make:auth
```

- Loi sivustolle kirjautumiseen ja rekisteröitymiseen vaaditut näkymät ja reitit.

```
php artisan make:controller
```

- Loi uuden kontrolleriluokan.

```
php artisan storage:link
```

- Loi symbolisen linkin public/storage-kansion ja storage/app/public-kansion välille.

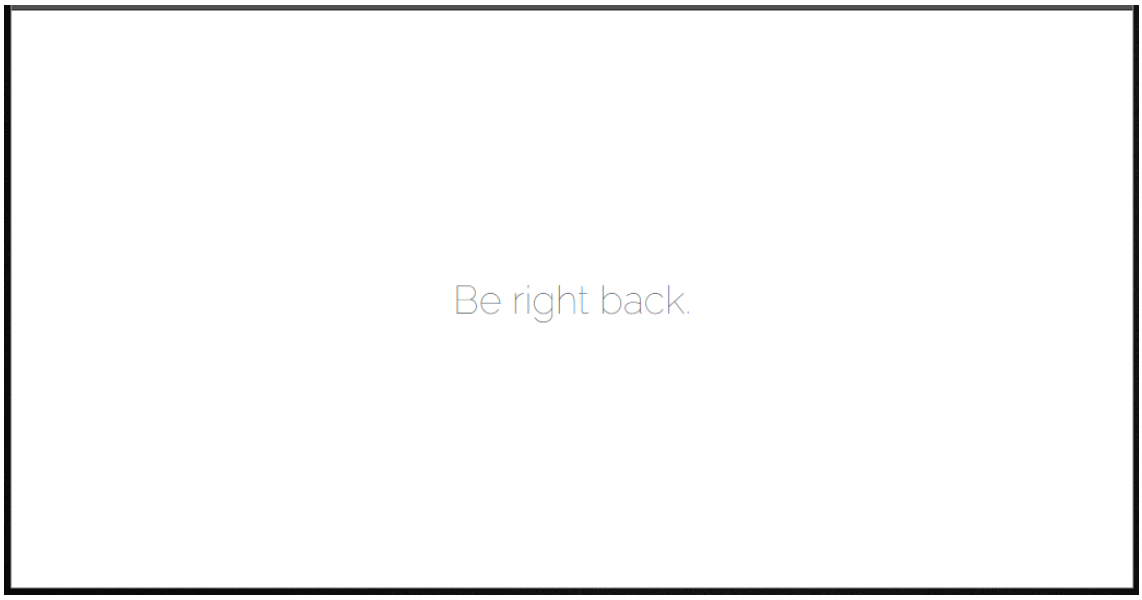
```
php artisan vendor:publish
```

- Julkaisi itse asennetut kirjastot käytettäviksi sovelluksessa.

Mikäli sivusto on jo julkaistu ja tuotannossa, Artisan sisältää komennon myös sivuston huoltokatkolle, jos tilanne sellaista vaatii. Huoltokatkon aikana jokainen sivuston reitti palauttaa vain huoltokatkosivun.

```
php artisan down
```

Esimerkkikoodi 7. Huoltokatkon kytkeminen päälle artisan-komennolla.



Kuva 3. Sivuston huoltokatkonäkymä.

Kun katko on ohi ja halutaan palauttaa sivuston toiminta normaalitilaan, ajetaan komento

```
php artisan up
```

Esimerkkikoodi 8. Artisan-komento, joka palauttaa sivuston normaalitilaan.

4.3 Tietokanta

Tietokantana projektissa käytettiin MySQL:ää, joka on suosittu relaatiotietokantaohjelmisto. MySQL on tämän tyyppiseen sovellukseen mielestäni hyvä valinta ja se oli jo entuudestaan tuttu. MySQL on myös Laravelin oletustietokanta, joten sen valitseminen oli tähän projektiin loogista. Tietokannan hallinnointiin käytettiin HeidiSQL:ää, joka on Windows-ympäristöön kehitetty graafinen tietokannan käyttöliittymä.

Homestead sisältää MySQL:n, joten sen konfiguroimiseen ei vaadittu muuta kuin aiemmin esitetyn *.env*-tiedoston määritellyt kohdat eli tietokantayhteyden tyyppi, IP-osoite, tietokannan portti, tietokannan nimi ja tietokannan käyttäjän nimi sekä salasana.

Laravel käyttää tietokannan hallintaa varten tietokantamigraatioita, jotka ovat eräänlainen versionhallinta tietokantatauluille [7.] Tietokantamigraatiot helpottavat projektin tietokannan jakamista sen kehittäjien kesken. Migraatioita käyttämällä kaikki projektin sisältämät tietokantataulut ja testidata voidaan luoda yhdellä Artisan-komennolla. Projektissa käytetyt Artisan-komennot löytyvät aiemmin esitetystä Artisan-kappaleesta.

Projektia varten luotiin tietokantamigraatiot käyttäjille, tiedostoille, kuville ja kategorioille. Alla näkyvässä koodiesimerkissä tiedostot-tietokantatauluun luotiin siihen tarvittavat kentät ja assosioitiin tiedostot ohjelman käyttäjän id-tietueen kanssa.

```
class CreateFilesTable extends Migration
{
      public function up()
      {
            Schema::create('files', function (Blueprint $table) {
            $table->increments('id');
            $table->string('filename');
            $table->string('original_name');
            $table->integer('size');
            $table->string('extension');
            $table->integer('user_id')->unsigned();
            $table->foreign('user_id')->references('id')->on('users');
            $table->timestamps();
            });
      }

      public function down()
      {
            Schema::dropIfExists('files');
      }
}
```

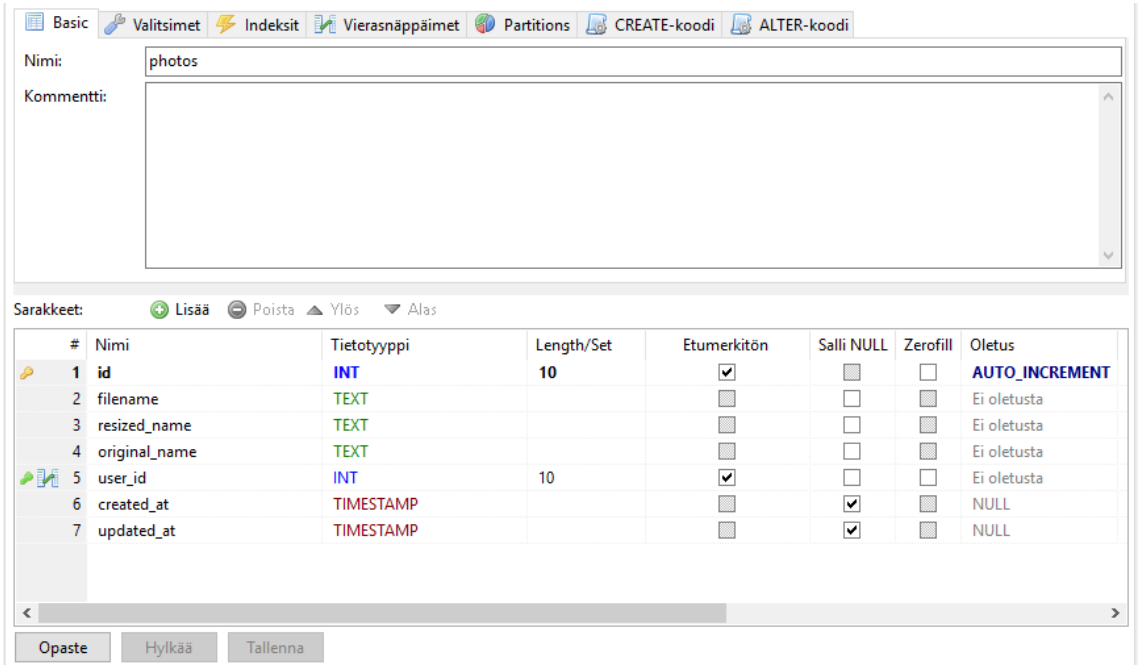
Esimerkkikoodi 9. Projektin tiedostot-tietokantataulun luomiseksi tehty tietokantamigraatio.

Migraatio voitiin tämän jälkeen ajaa komentotulkista ”*migrate*”-komennolla, jonka jälkeen uusi tietokantataulu luotiin tietokantaan.

```
php artisan migrate
```

Esimerkkikoodi 10. Tietokantamigraation ajaminen.

Tietokantayhteys oli tässä vaiheessa konfiguroitu ja tietokantaan voitiin ottaa yhteys HeidiSQL:llä



Kuva 4. HeidiSQL:ssä näkyvät tietokantakentät kuvat-taulussa.

4.4 Mallit

Laravel käyttää Eloquent-malleja, joka on Active record-implementaatio. Active record on arkkitehtuurinen suunnittelumalli ja tarkoittaa sitä, että jokaista tietokantataulua vastaa oma malli, jota käytetään sen kanssa toimimiseen. [8.] Malleja käyttämällä voidaan esimerkiksi hakea tietoja tietokantatauluista tai luoda niihin uusia ilmentymiä.

Projektissa omat mallit luotiin käyttäjille, tiedostoille, kuville sekä kategorioille. Esimerkiksi Tiedostoja vastaavan mallin ”Files” funktio *user()* kuvaa sitä, että käyttäjällä on useita tiedostoja, jotka käyttäjä omistaa.

```
class File extends Model
{
      public function user(){
            return $this->belongsTo('App\User');
      }
}
```

Esimerkkikoodi 11. Tiedostot kuuluvat käyttäjälle.

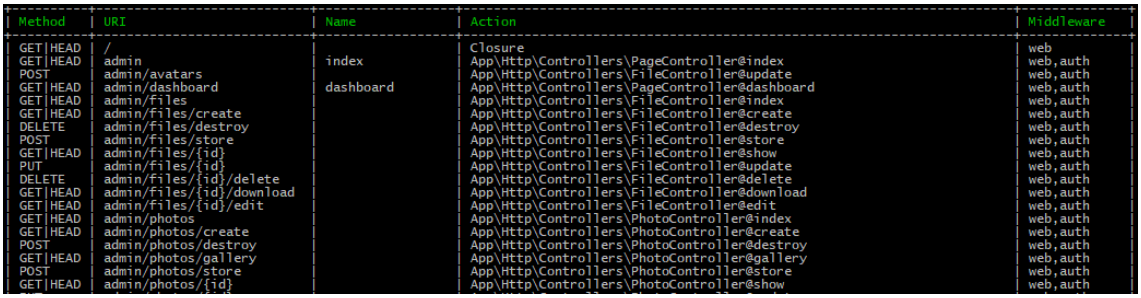
Tästä on hyötyä siten, että jokainen palveluun ladattu tiedosto saadaan assosioitua tiettyyn käyttäjään. Täten voidaan rajata ja säätää kuvien näkymiseen, muokkaamisen, poistamisen ym. vaikuttavia oikeuksia niin, että vain kuvan palveluun ladanneella käyttäjällä on oikeus kyseisiin toimintoihin.

4.5 Reititys

Kaikki Laravel-ohjelmistossa käytettävät reitit listataan *routes*-tiedostoon ja ohjelmistokehys lataa ne sieltä automaattisesti käytettäväksi. Reittitiedosto toimii periaatteessa välittäjänä selaimelta tulevien http-pyyntöjen ja ohjelmiston logiikan välillä. Tyypilliseen web-sovellukseen, missä on käyttöliittymä, listataan reitit *routes*-kansiosta löytyvään *web*.php-tiedostoon. Tähän tiedostoon lisättävät reitit ovat valmiiksi *web-middleware* -ryhmässä, joka tarkoittaa sitä, että niihin tallentuu aktiivisen vierailun tila selaimessa (*session state*) ja että ne ovat CSRF-hyökkäyksiltä suojattuja. [9.]

Routes-tiedostossa määritetään tyypillisesti kullekkin URI:lle siitä vastuussa oleva kontrolleri ja kontrollerin funktio, jotka ovat vastuussa siitä mitä tapahtuu, kun käyttäjä navigoi selaimella kyseiseen reittiin. ”*php artisan route:list”*-komennolla saadaan listattua komentotulkin kautta kaikki reitit ja niitä vastaavat metodit, uri:t, nimet, toiminnot ja middlewaret.

Tämän projektin kaikki reitit ovat konfiguroitu löytymään /admin -routen takaa Laravelin *prefix*-funktiolla, lukuun ottamatta autentikaatioon liittyviä reittejä eli käyttäjän kirjautumista ja rekisteröitymistä. Syynä prefixaukseen on se, että sovellusta ei ole tarkoitus käyttää vierailijana, vaan kaikki toiminnot vaativat admin-oikeuksia.



Method	URI	Name	Action	Middleware
GET|HEAD	/		Closure	web
GET|HEAD	admin	index	App\Http\Controllers\PageController@index	web,auth
POST	admin/avatars		App\Http\Controllers\FileController@update	web,auth
GET|HEAD	admin/dashboard	dashboard	App\Http\Controllers\PageController@dashboard	web,auth
GET|HEAD	admin/files		App\Http\Controllers\FileController@index	web,auth
GET|HEAD	admin/files/create		App\Http\Controllers\FileController@create	web,auth
DELETE	admin/files/destroy		App\Http\Controllers\FileController@destroy	web,auth
POST	admin/files/store		App\Http\Controllers\FileController@store	web,auth
GET|HEAD	admin/files/{id}		App\Http\Controllers\FileController@show	web,auth
PUT	admin/files/{id}		App\Http\Controllers\FileController@update	web,auth
DELETE	admin/files/{id}/delete		App\Http\Controllers\FileController@delete	web,auth
GET|HEAD	admin/files/{id}/download		App\Http\Controllers\FileController@download	web,auth
GET|HEAD	admin/files/{id}/edit		App\Http\Controllers\FileController@edit	web,auth
GET|HEAD	admin/photos		App\Http\Controllers\PhotoController@index	web,auth
GET|HEAD	admin/photos/create		App\Http\Controllers\PhotoController@create	web,auth
POST	admin/photos/destroy		App\Http\Controllers\PhotoController@destroy	web,auth
GET|HEAD	admin/photos/gallery		App\Http\Controllers\PhotoController@gallery	web,auth
POST	admin/photos/store		App\Http\Controllers\PhotoController@store	web,auth
GET|HEAD	admin/photos/{id}		App\Http\Controllers\PhotoController@show	web,auth

Kuva 5. Sovelluksen reitit listattuna komentotulkissa.

```
18    Auth::routes();
19
20     Route::middleware(['auth'])->group(function () {
21
22        Route::prefix('admin')->group(function () {
23
24            //Pages
25            Route::get('/', 'PageController@index')->name('index');
26            Route::get('dashboard', 'PageController@dashboard')->name('dashboard');
27            Route::get('profile', 'ProfileController@index')->name('profile');
28
29            //Files
30            Route::get('files/','FileController@index');
31            Route::get('files/create', 'FileController@create');
32            Route::get('files/{id}/edit', 'FileController@edit');
33            Route::post('files/store','FileController@store');
34            Route::put('files/{id}','FileController@update');
35            Route::delete('files/destroy','FileController@destroy');
36            Route::delete('files/{id}/delete','FileController@delete');
37            Route::get('files/{id}/download','FileController@download');
38            Route::get('files/{id}','FileController@show');
39            Route::post('avatars', 'FileController@update');
40
41        });
42     });
43
```

Kuva 6. Autentikaation reititys, middleware-ryhmä ja admin-paneelin sivujen sekä tiedostojen reititys sovelluksessa.

Esimerkiksi kun käyttäjä menee selaimella sovelluksen polkuun /admin/files, eli tekee get-pyynnön, laukaisee ohjelmistokehys sille reititystiedostossa määrätyn tapahtuman. Kuvasta 6 käy ilmi, että get-pyyntö files-url:iin ohjautuu FileControllerin index-metodiin, joka vuorostaan palauttaa kontrollerilta sille määrätyn näkymän.

```
public function index()
    {
        $files = File::with('user')->get();
        return view('admin.files.index', compact('files'));
    }
```

Esimerkkikoodi 12. Pyyntö ohjautuu *FileControllerin index*-metodiin, joka vie näkymään käyttäjän omistamat tiedostot ja palauttaa näkymäsivun.

4.6 Tietoturva

Tietoturva on olennainen osa sovelluksen kehittämistä ja se tulee ottaa huomioon jo siinä vaiheessa, kun sovellusta suunnitellaan. Laravel tarjoaa valmiiksi paketin käyttäjän rekisteröitymiseen ja sisäänkirjautumiseen. Tämä on hyvä lähtökohta sovelluksen kehittämiselle ja voidaan olla varmoja siitä, että ne on toteutettu tietoturvallisesti. Laravelin autentikaatiopaketissa tietokantamigraatio ja reitit luodaan käyttäjille yhdellä Artisan-komennolla ”*php artisan make:auth*”.

Käyttäjän luoma salasana hashataan automaattisesti Bcryptillä ja sille luodaan palautustoken, jolloin toiminnallisuus käyttäjän unohtuneen salasanan palauttamiseksi on valmiiksi olemassa. Toimiva käyttäjien rekisteröinti, kirjautuminen ja salasanan palautus onnistuvat siis yhdellä ainoalla komennolla, jolloin voitiin keskittyä enemmän itse sovelluksen toiminnallisuuden kehittämiseen. [10.]

Tähän sovellukseen on määritetty, että jos käyttäjä yrittää päästä admin-paneliin käsiksi kirjautumatta sisään, tapahtuu jokaisesta */admin*-polusta uudelleenreititys takaisin sivuston kirjautumisosioon. Uudelleenreitityksestä vastaa *routes*-tiedostossa määritetty middleware ”*auth*”, joka on Laravelin sisään rakennettu ominaisuus. Auth-middlewarella tarkistetaan, että käyttäjä on kirjautuneena sovellukseen. Kuvassa 6 näkyy, kuinka kaikki reitit ovat *auth-middleware* -funktion sisällä.

Laravel suojaa sovellusta cross-site request forgeryltä (CSRF) generoimalla jokaiselle käyttäjälle oman CSRF-tokenin. Aina, kun käyttäjä tekee sivustolla jonkin toiminnon, kuten täyttää lomakkeen ja lähettää sen, varmistetaan lomakkeeseen liitettävällä CSRF-tokenilla, että se vastaa käyttäjän sessioniin annettua tokenia. Näin varmistetaan se, että hyökkääjä ei pysty esiintymään toisena käyttäjänä. Jokaiseen POST-, PUT- tai DELETE -routeen tehtävään http-pyyntöön tulee sisältyä CSRF-token tai muuten pyyntö hylätään. [11.]

4.7 Monitorointi

Sovelluksen tietoturvan ylläpitämiseksi täytyy olla ajan tasalla sovelluksen virheellisestä toiminnasta eli sen bugeista. Laravelin versio 5.6 tarjosi uudistetun virhelokituksen, joka oli konfiguroitavissa käyttötarpeen ja sovelluksessa tapahtuvan virheen vaaratason

mukaan. Sovelluksessa päädyttiin käyttämään Slack-lokitusta ja sovelluksen sisäistä lokitusta. Konfiguraatiota pääsi muokkaamaan *config/logging.php*-tiedostosta, johon määrittelyt tehtiin uudella stack-muuttujalla. Slack-kanavalle tehtävää lokitusta varten luotiin uusi ”incoming webhook”, joka tarkoittaa ulkoisesta lähteestä tulevan http-viestin lisäämistä tietylle Slack-kanavalle. Webhookin url tallennettiin paikallisen tason muuttujatiedostoon *.env*:iin LOG_SLACK_WEBHOOK_URL -muuttujan arvoksi.

```
'channels' => [
      'stack' => [
            'driver' => 'stack',
            'channels' => ['single', 'slack'],
      ],

      'single' => [
            'driver' => 'single',
            'path' => storage_path('logs/laravel.log'),
            'level' => 'debug',
      ],

      'slack' => [
            'driver' => 'slack',
            'url' => env('LOG_SLACK_WEBHOOK_URL'),
            'username' => 'Laravel Log',
            'emoji' => ':boom:',
            'level' => 'error',
      ],
];
```

Esimerkkikoodi 13. Lokituskanaviksi valittiin channels-arrayssä ’single’ eli sovelluksen sisäinen ja ’slack’ eli slack-lokitus

4.8 Validaatio

Laravel tarjoaa useita eri lähestymistapoja sovellukseen syötettävän datan validointiin. Tässä projektissa validoinnit suoritettiin pääosin kontrolleriluokkien funktioissa. Esimerkiksi kategoriaa lisättäessä tarkastetaan sen kontrollerin *store*-metodissa, että sille on annettu nimi, se on uniikki ja sen maksimipituus on 255 merkkiä.

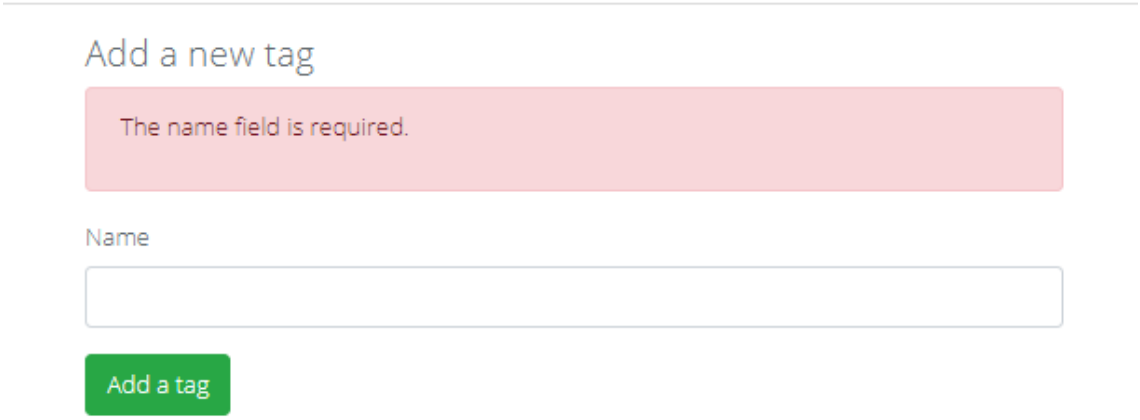
```
$request->validate([
      'name' => 'required|unique:categories|max:255'
]);
```

Esimerkkikoodi 14. Syötetyn kategorian validointi.

Validaatiovirheet voidaan näyttää kaikista web-ryhmään kuuluvista reiteistä yksinkertaisella silmukalla, sillä web-middleware, johon kaikki ohjelman reitit kuuluvat, on kiinnitetty *ShareErrorsFromSession*-middlewareen. Tämä tarkoittaa sitä, että *$error*-muuttujalla on aina jokin arvo. [12.]

```
@if ($errors->any())
<div class="alert alert-danger">
      <ul>
            @foreach ($errors->all() as $error)
                  <li>{{ $error }}</li>
            @endforeach
      </ul>
</div>
@endif
```

Esimerkkikoodi 15. Validointisilmukka.



Kuva 7. Validaatiovirheiden listaaminen näkymässä.

5 Palvelimen asentaminen

Projektia varten luotiin virtuaalisen Homestead-ohjelmiston sisältämän testipalvelimen lisäksi erillinen Ubuntu-palvelin ”tuotantokäyttöön”, johon asennettiin kaikki Laravelin vaatimat ohjelmistot ja joka avattiin niin, että se oli käytettävissä myös sisäverkon ulkopuolelta.

Palvelin asennettiin ja konfigurioitiin ensin omalle tietokoneelle, jonka kanssa palvelimen käyttöä harjoiteltiin. Myöhemmässä vaiheessa siirryttiin käyttämään Amazonin AWS:ää ja sinne asennettua Ubuntu-palvelinta. Asennusprosessi AWS:ään oli hyvin samankaltainen kuin omalle tietokoneelle, joskin Amazonin tarjoamassa palvelimessa oli jo valmiiksi konfiguroitu toiminnallisuus ssh-yhteyksiä varten.

Palvelimien asennuksissa ohjeina käytettiin pääosin Digital Oceanin tekemiä oppaita, joiden mukaan asennuksessa edettiin. Digital Ocean on IaaS-palveluntarjoaja, joka tarjoaa mahdollisuuden luoda virtuaalisia käyttöjärjestelmiä sen tarjoamille palvelimille.

Digital Oceania ei kuitenkaan käytetty tässä projektissa muuhun kuin sen tarjoamien oppaiden seuraamiseen, vaan pitäydyttiin Amazonin palveluissa.

Oman testipalvelimen käyttöjärjestelmä asennettiin muistitikulle käyttäen *Rufus*-nimistä ohjelmaa, jolla saatiin luotua käynnistettävä USB-asema. Käyttöjärjestelmäksi palvelimelle valittiin *Ubuntu server 16.04.4 LTS* ja se asennettiin palvelimelle asennusvaiheessa ruudulle ilmestyvien ohjeiden mukaan. Palvelimelle suositellut minimivaatimukset olivat:

- 2 GHz kaksiytiminen prosessori tai parempi
- 2 GB RAM-muistia
- 25 GB kovalevytilaa
- DVD- tai USB-asema asennusta varten
- Internet-yhteys.

Kun palvelinta alettiin konfiguroimaan, täytyi tietoturvan ja käytettävyyden kannalta heti ottaa muutamia asioita huomioon [13.] Näitä olivat uuden käyttäjän luominen heti alussa, jotta ei tarvitse käyttää root-käyttäjää, ssh-avainparin luominen ja Openssh-serverin asentaminen. Näin palvelimeen saatiin myöhemmin yhteys turvallisesti kotiverkon ulkopuolelta ssh:lla ja salasanalla kirjautuminen voitiin poistaa käytöstä.

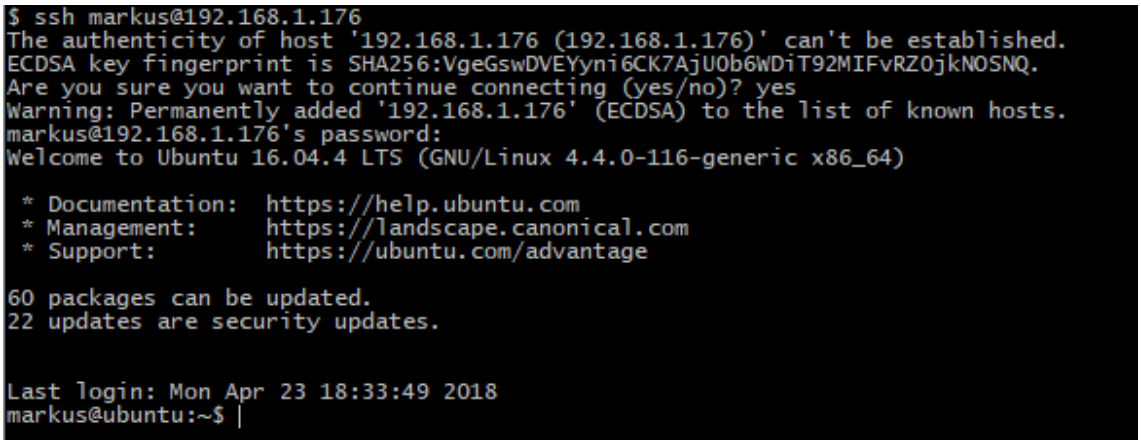
Nämä vaiheet paransivat palvelimen turvallisuustasoa vaatimalla tästä lähin jokaisella yhdistyskerralla yksityistä ssh-avainta palvelimelle kirjautumiseen. Avainpari luotiin komentotulkista komennolla ”*ssh-keygen*”, joka loi yksityisen ja julkisen ssh-avaimen sille käyttäjälle, joka oli sillä hetkellä kirjautuneena palvelimelle sisään.

Yksityistä avainta ei tule jakaa kenenkään kanssa, jonka ei ole tarkoitus päästä palvelimelle. Se jaetaan ainoastaan niille tietokoneille, joista yhteys palvelimeen halutaan muodostaa. Avainparin luomisen jälkeen asennettiin *OpenSSH-server*.

```
sudo apt-get install openssh-server
// käynnistetään palvelu uudelleen
sudo service ssh restart
```

Esimerkkikoodi 16. Openssh-serverin asennuskomento ja uudelleenkäynnistys restart-komennolla.

Näiden vaiheiden jälkeen palvelin oli valmiudessa palvelimen sisäverkon ulkopuolelta tuleviin yhteyspyyntöihin.



Kuva 8. Ensimmäinen yhteys palvelimeen toisesta sisäverkon tietokoneesta.

Seuraava vaihe oli päivittää palvelimen paketit apt-paketinhallintajärjestelmällä.

```
sudo apt-get update
sudo apt-get upgrade
```

Esimerkkikoodi 17. Update-komennolla päivitettiin pakettien lista ja upgrade-komennolla varsinaiset paketit.

Seuraavaksi poistettiin käytöstä pelkällä salasanalla kirjautuminen palvelimelle, sillä ssh-yhteyden konfiguroinnin jälkeen pelkällä salasanalla kirjautumiselle ei ollut tarvetta ja se luokitellaan tietoturvariskiksi. Palvelimen Nano-tekstieditorilla avattiin *ssh*:n *config*-tiedosto ja poistettiin *PasswordAuthentication*-kohdasta sen edessä oleva risuaita ja annettiin uudeksi arvoksi ”no”.

```
sudo nano /etc/ssh/sshd_config
//
#PasswordAuthentication no
```

Esimerkkikoodi 18. Pelkällä salasanalla palvelimelle kirjautumisen poistaminen käytöstä.

Komennon ajamisen jälkeen käynnistettiin ssh-palvelu uudelleen muutosten aktivoimiseksi komennolla ” sudo service ssh restart”.

Edellä läpi käydyt kohdat antoivat hyvän perustan palvelimen tulevaan konfigurointiin ohjelmistokehyksen vaatimuksia varten. Amazonin palvelimelle näitä vaiheita ei tarvinnut tehdä, vaan ssh-avain luotiin AWS:n graafisesta käyttöliittymästä. Yhteys luotuun

palvelimeen saatiin luotua Amazonin automaattisesti luomalla ohjeella, jossa avaimelle annettu nimi ja palvelimen ip-osoite olivat jo valmiiksi konfiguroituina.

Tietokoneella, josta yhteys otettiin palvelimeen, oli Windows-tietokone, jossa käytettiin Git bash:ia. Git bash on komentotulkki, joka tarjoaa mahdollisuuden ladata konfiguraatioita ohjelman käynnistyessä. Tähän tarkoitukseen luotiin *.bashrc*-niminen tiedosto käyttäjän kotihakemistoon, joka latasi käyttäjän yksityisen avaimen automaattisesti komentotulkkiin mahdollistaen yhteyden palvelimeen.

```
eval $(ssh-agent -s)
ssh-add ~/.ssh/<yksityisen_avaimen_nimi>
```

Esimerkkikoodi 1. .bashrc-tiedostoon lisättiin yllä olevat rivit, jotta yksityinen avain latautui automaattisesti.

Riippumatta siitä käytettiinkö omaa tuotantopalvelinta vai Amazonin palvelinta, toimi yhteyden ottaminen palvelimeen samalla tavalla.

```
ssh -i "<yksityisen_avaimen_nimi>" ubuntu@ec2-<palvelimen ip-osoite>.eu-west-
1.compute.amazonaws.com
```

Esimerkkikoodi 2. Palvelimeen otetaan yhteys komentotulkista.

Ensimmäisenä itse sovelluksen vaatimuksista asennettiin palvelimelle PHP:n versio 7.2 ja Laravelin vaatimat php-moduulit.

```
//päivitetään jälleen paketit ja asennetaan ne
sudo apt-get update && apt-get upgrade

//lisätään kolmannen osapuolen repository php:tä varten
sudo apt-get install python-software-properties

//lisätään php repository
sudo add-apt-repository ppa:ondrej/php

//päivitetään pakettilista
sudo apt-get update

//asennetaan php
sudo apt-get install php7.2

//asennetaan yleisesti käytössä olevat php-moduulit
sudo apt-get install php-pear php7.2-curl php7.2-dev php7.2-gd php7.2-mbstring
php7.2-zip php7.2-mysql php7.2-xml
```

Esimerkkikoodi 3. Php:n ja sen moduuleiden asennus.

Php:n asennuksen jälkeen asennettiin riippuvuuksien hallintaa varten Composer.

```
sudo apt install composer
```

Esimerkkikoodi 4. Composerin asennuskomento.

Composerin asentamisen jälkeen kloonattiin sovellus versionhallinnasta palvelimelle. Tässä projektissa versionhallintaan käytettiin Gitlabia. Projekti kloonattiin palvelimen /var/www/html-kansioon, jotta se oli käytettävissä Apache web-palvelimelle.

```
git clone https://gitlab.com/<käyttäjänimi>/<projektin nimi>.git
```

Esimerkkikoodi 5. Projektin repositoryn kloonaus.

Kun repository oli saatu kloonattua, asennettiin projektin vaatimat liitännäiset.

```
sudo composer install
```

Esimerkkikoodi 6. Projektin liitännäisten asennus.

```
"require": {
      "php": ">=7.1.3",
      "enyo/dropzone": "^5.1",
      "fideloper/proxy": "~4.0",
      "intervention/image": "^2.4",
      "laravel/framework": "5.6.*",
      "laravel/tinker": "~1.0",
      "laravelcollective/html": "^5.4.0",
      "league/flysystem-aws-s3-v3": "^1.0"
      },
"require-dev": {
      "filp/whoops": "~2.0",
      "fzaninotto/faker": "~1.4",
      "mockery/mockery": "~1.0",
      "nunomaduro/collision": "~2.0",
      "phpunit/phpunit": "~7.0",
      "symfony/thanks": "^1.0"
      }
```

Esimerkkikoodi 7. Projektin composer.json-tiedostossa olevat ohjelman vaatimat liitännäiset.

Seuraavaksi asennettiin Node.js, jotta saatiin projektin npm-paketit asennettua.

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
npm install npm@latest -g
//asennetaan node-moduulit projektikansiossa
sudo npm install
```

Esimerkkikoodi 8. Nodejs:n ja Noden paketinhallintaohjelma npm:n asennus ja projektin node-moduuleiden asennus.

Seuraavaksi oltiin valmiina asentamaan palvelimelle tietokanta ja tähän tehtävään valittiin Mysql-server.

```
sudo apt-get install mysql-server
mysql_secure_installation
```

Esimerkkikoodi 9. Mysql-serverin asentaminen ja tietoturvallisemman konfiguraation ajaminen.

Mysql:n tarjoama secure-installation kovettaa palvelimen tietoturvaa tietokannan ostalta käyttäjän valintojen perusteella. Secure-installation kysyy mm. halutaanko tietokantaan luotaville käyttäjille asettaa salasanan osalta minimivaatimuksia pituuden ja merkistön suhteen ja asennusohjelma tarjoaa siihen erilaisia vaihtoehtoja. Asennusohjelma kysyy lisäksi, halutaanko tietokantaan automaattisesti luotu testidata ja testikäyttäjä poistaa sekä estää root-käyttäjän pääsy etäyhteydellä kiinni tietokantaan. Kaikilla näillä valinnoilla on tietokannan tietoturvan kannalta positiivinen vaikutus.

Seuraavaksi kirjauduttiin sisään Mysql:ään ja lisättiin uusi Mysql-käyttäjä. Käyttäjän salasanan täytyy tässä vaiheessa täyttää secure-installationin aikana määritetyt minimivaatimukset pituuden ja merkistön osalta.

```
mysql -u root -p
```

Esimerkkikoodi 10. Kirjaudutaan Mysql:ään.

```
CREATE USER '<nimi>'@'localhost' IDENTIFIED BY '<salasana>';
GRANT ALL PRIVILEGES ON * . * TO 'nimi'@'localhost';
```

Esimerkkikoodi 11. Luodaan uusi käyttäjä ja annetaan täydet oikeudet tietokantaan.

```
CREATE DATABASE <tietokannan nimi>;
```

Esimerkkikoodi 12. Luodaan uusi tietokanta.

Seuraavaksi luotiin projektikohtainen .env-tiedosto ja lisättiin siihen tarvittavat tiedot tietokantayhteyttä varten, jonka jälkeen voitiin ajaa tietokantamigraatio.

```
APP_NAME=<projektin nimi>
APP_ENV=production
APP_KEY=<32 merkkiä pitkä merkkijono>
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=<julkinen ip-osoite>

LOG_CHANNEL=stack
LOG_SLACK_WEBHOOK_URL=https://hooks.slack.com/services/<yksilöivä tunniste>
```

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=<tietokannan nimi>
DB_USERNAME=<tietokannan käyttäjän tunnus>
DB_PASSWORD=<tietokannan käyttäjän salasana>
```

Esimerkkikoodi 13. Lisätään .env-tiedostoon tarvittavat kohdat.

Konfiguraatiotiedoston luonnin jälkeen voitiin asettaa sovelluksen .env-tiedostoon kohtaan APP_KEY sen vaatima application key komennolla ”*php artisan key:generate*”. Jos application keytä ei aseta, sovelluksen salattu data ja käyttäjien sessiot eivät toimi turvallisesti. Seuraava vaihe oli antaa web-palvelimelle oikeudet lukea projektikansiota.

```
sudo chown www-data: -R /var/www/html/<projekti>/
```

Esimerkkikoodi 14. Sovelluksen lukuoikeuden antaminen web-palvelimelle.

Jotta Laravel saatiin toimimaan palvelimella oikein, täytyi palvelimella olla oikeudet kirjoittaa *storage*- ja *bootstrap/cache* -kansioihin.

```
sudo chmod -R 775 /var/www/html/<projekti>/storage/
sudo chmod -R 775 /var/www/html/<projekti>/bootstrap/cache
```

Esimerkkikoodi 15. Kansio-oikeuksien säätö.

Seuraavaksi linkitettiin sovelluksen *storage*-kansio *public*-kansioon symbolisen linkin avulla. Artisan tarjosi tähän suoran komennon ja se tehtiin siksi, että web-palvelin pystyi lukemaan ja tallettamaan kansioihin.

```
php artisan storage:link
```

Esimerkkikoodi 16. Symbolisen linkin luominen kansioiden välille.

Apachea käytettäessä palvelimella olevat web-sivut jaotellaan omiin kansioihinsa. Jotta palvelimella olevia sivustoja voidaan konfiguroida erikseen, täytyy niille luoda ”virtual hostit”. [14.] Virtual hostissa määritetään mm. kyseisen sivuston juurikansio. Konfigurointi aloitettiin luomalla laravel.conf-niminen tiedosto Apachen sites-available-kansioon.

```
sudo nano /etc/apache2/sites-available/laravel.conf
```

Esimerkkikoodi 17. Uuden virtual hostin luominen.

Konfiguraatiotiedostoon kirjoitettiin virtual hostin asetukset, eli annettiin projektin juurikansion polku ja määritettiin Apachen virhelokitus.

```
<VirtualHost *:80>
    ServerName yourdomain.tld

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/<projekti>/public

    <Directory /var/www/html/<projekti>>
        AllowOverride All
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Seuraavaksi sammutettiin oletuskonfiguraatio, aktivoitiin uusi konfiguraatio ja käynnistettiin apache uudelleen.

```
sudo a2dissite 000-default.conf
sudo a2ensite laravel.conf
sudo a2enmod rewrite
sudo service apache2 restart
```

Tässä vaiheessa kaikki tarpeellinen oli asennettu ja web-palvelin toiminnassa.

6 Palvelimen hallinta

Tuotantopalvelimeen yhdistäminen toteutettiin niin, että sisäverkon lisäksi myös ulkoverkosta saatiin palvelimeen yhteys toisesta tietokoneesta ssh:ta käyttäen. Yhteyden saamiseksi palvelimeen komentotulkin kautta vaadittiin, että palvelimelle oli luotu tili käyttäjälle. Tämän lisäksi tarvittiin palvelimen julkinen ip-osoite. Yhteyden luomiseksi palvelimeen komentotulkkiin kirjoitettiin yksinkertaisesti:

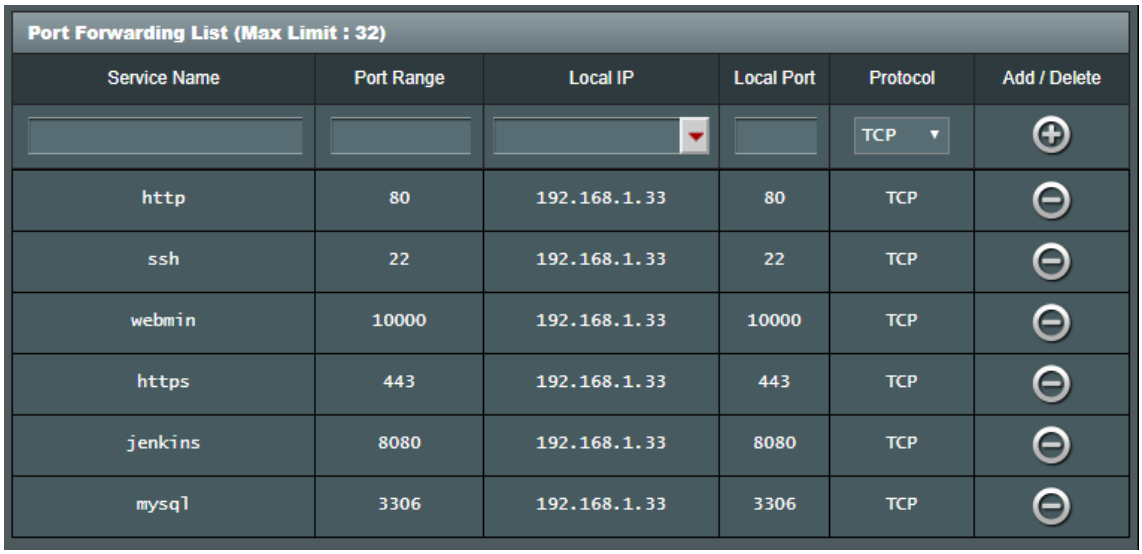
```
ssh<käyttäjätunnus>@<palvelimen ip>
```

Komentotulkin lisäksi omaan tuotantopalvelimeen sai yhteyden siihen asennetun Webmin-lisäosan kautta. Amazonin palvelimeen Webminiä ei asennettu, sillä Amazonin dashboard tarjosi vastaavat ominaisuudet. Webmin on etäyhteyden palvelimeen mahdollistava graafinen, selaimen kautta toimiva käyttöliittymä. Webminin asennus aloitettiin lisäämällä seuraava rivi apt-paketinhallintaohjelmiston lähteisiin. Lähteet löytyvät palvelimella polusta */etc/apt/sources.list.*

```
deb https://download.webmin.com/download/repository sarge contrib

//lisätään Webminin GPG-avain, jotta palvelin luottaa lähteeseen
sudo wget http://www.webmin.com/jcameron-key.asc
sudo apt-key add jcameron-key.asc

//päivitetään pakettilista
sudo apt-get update

//asennetaan Webmin
sudo apt-get install webmin
```

Esimerkkikoodi 18. Webminin asennuskomennot.

Oletusportti Webminille on 10000 ja komentotulkin kautta tehtävälle ssh-yhteydelle 22. Molempiin yhdistämistapoihin vaadittiin oikeiden porttien aukaisua siitä reitittimestä, jonka kautta palvelin oli yhteydessä ulkomaailmaan. Local IP -kohtaan valittiin palvelimen sisäverkon IP-osoite, jonka portit avattiin ulkomaailmaan.



Port Forwarding List (Max Limit : 32)

Service Name	Port Range	Local IP	Local Port	Protocol	Add / Delete
				TCP	⊕
http	80	192.168.1.33	80	TCP	⊖
ssh	22	192.168.1.33	22	TCP	⊖
webmin	10000	192.168.1.33	10000	TCP	⊖
https	443	192.168.1.33	443	TCP	⊖
jenkins	8080	192.168.1.33	8080	TCP	⊖
mysql	3306	192.168.1.33	3306	TCP	⊖

Kuva 9. Porttien avaamista Asus-merkkisen reitittimen käyttöliittymässä.

Asennuksen ja porttien avaamisen jälkeen Webmin-halintapaneeli löytyi selaimesta palvelimen julkisella ip-osoitteella portista 10000.

Kuva 10. Hallintapaneeliin kirjaudutaan palvelimelle luodun käyttäjän tunnuksilla.

Kuva 11. Webmin-käyttöliittymässä nähdään mm. Palvelimen muistin- ja tilankäyttö.

7 Käyttöliittymä

Sivuston ulkoasu toteutettiin Bootstrap-css -kirjastoa avuksi käyttäen. Sovelluksen admin-osio toteutettiin Blade-templateina, joka mahdollisti @yield-komennon käyttämisen. @yield-komennolla saatiin ylikirjoitettua normaalinäkymään sivuston eri sivujen sisältö sille määritettyihin kohtiin. Layout-tiedosto toimi ns. Boilerplatena, jossa ladattiin tyylitiedostoja, fontteja ja skriptitiedostoja, jotka olivat usealle eri sivulle yhteisiä. Koodi pysyi näin siistinä ja helpommin ymmärrettävänä eikä ollut toisteista.

```
<!DOCTYPE html>
      <head>
            <link rel="stylesheet" href="{{ asset('css/app.css') }}">
            @yield('head')
      </head>

      <body>
      @include('admin.dashboard.navbar')
      <div class="container">
            <main>
                  @yield('content')
            </main>
      </div>
      </body>
</html>

<!-- Scripts -->
<script src="/js/app.js"></script>
@yield('js')
```

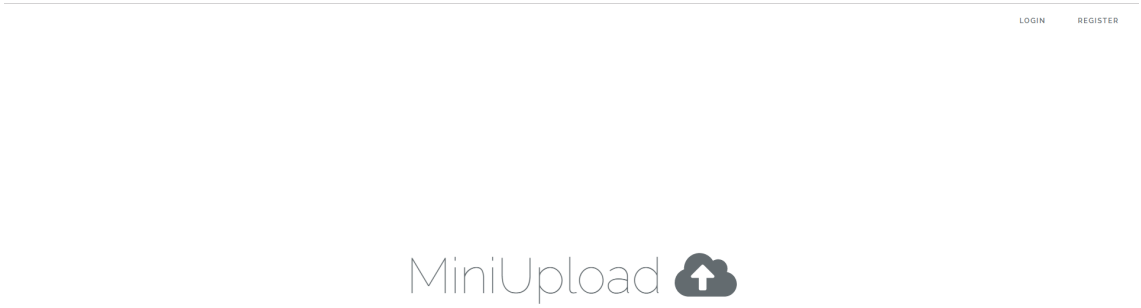
Esimerkkikoodi 19. Sovelluksen admin-osion layout, joka on tehty Blade-templatea hyödyntäen.

Layoutissa ladattava app.css-tiedosto on Webpackia käyttäen Scss-tiedostosta muunnettu ja se yhdistää kaikki sivulla tarvittavat css-tiedostot yhdeksi tiedostoksi. Vastaavasti app.js-tiedostossa on yhdistetty ohjelman tarvitsemat javascriptit.

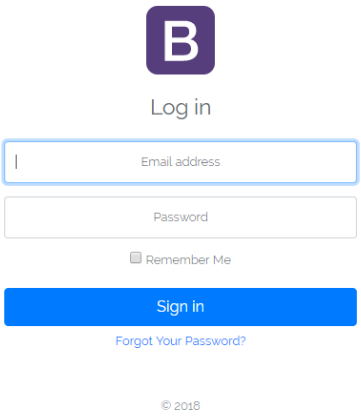
```
let mix = require('laravel-mix');

mix.js('resources/assets/js/app.js', 'public/js')
   .sass('resources/assets/sass/app.scss', 'public/css');
```

Esimerkkikoodi 20. Webpack-konfiguraatiotiedosto, jolla mm. resources-kansion scss-tiedosto muunnetaan css-tiedostoksi ja tallennetaan public-kansioon.

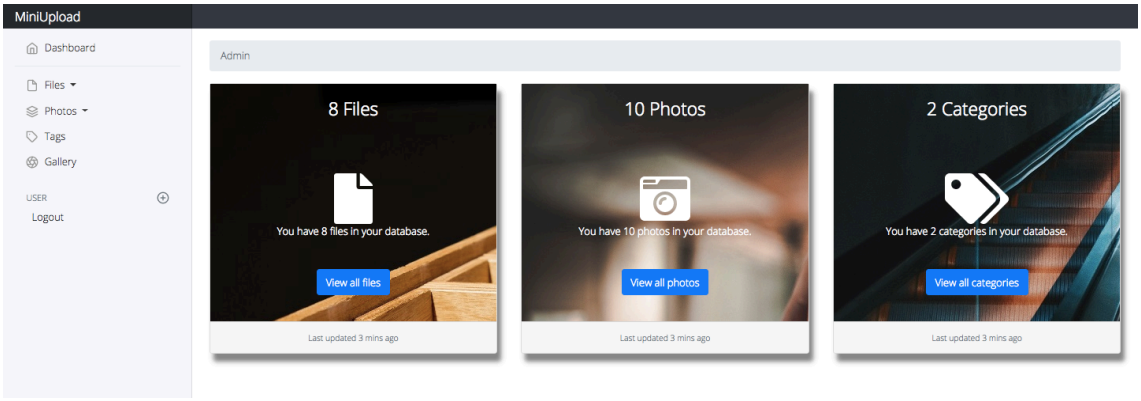
Sivuston aloitusnäkymässä tarjotaan mahdollisuus rekisteröityä tai kirjautua sisään palveluun.



Kuva 12. Sivuston aloitusnäkymä.



Kuva 13. Sivustolle kirjautumisen näkymä.



Kuva 14. Kirjautumisen jälkeen käyttäjä ohjataan dashboard-näkymään, jossa listataan käyttäjän lataamat tiedostot, kuvat ja käyttäjän luomat kategoriat.

8 Kirjastot

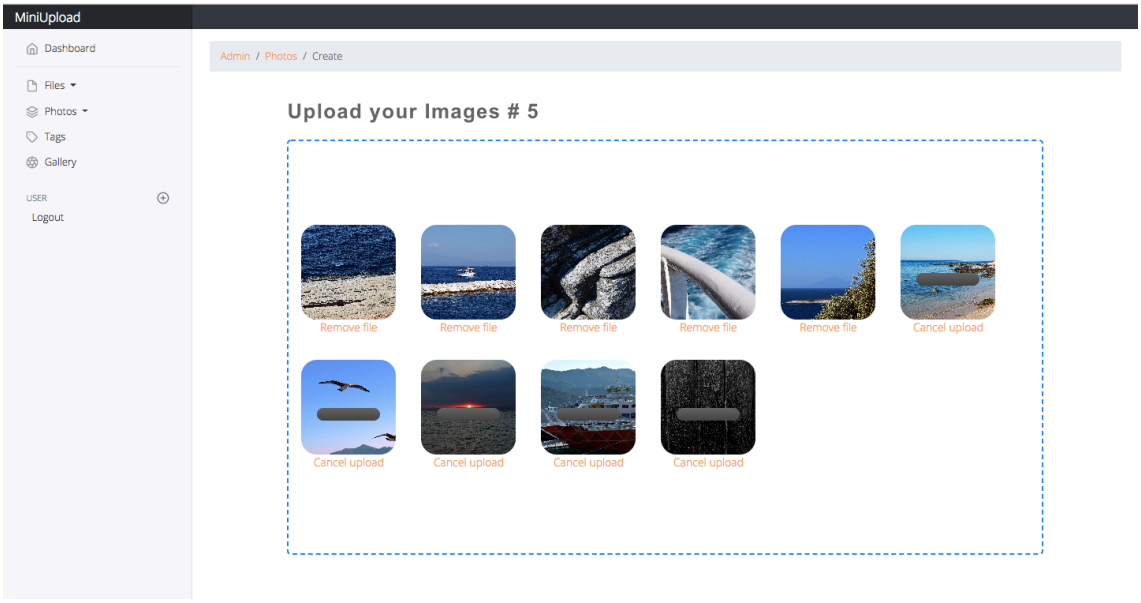
Projektin alkuvaiheessa tiedostojen lataamista palveluun lähdettiin toteuttamaan manuaalisesti, mutta myöhemmässä vaiheessa päätettiin ottaa käyttöön Dropzone.js, jolla kuvien lataaminen palveluun onnistui mutkitta niin yksittäisten kuvien kuin useampien kuvien joukkoina. Hieman Dropzonen konfiguraatiota muuttamalla saatiin myös muun tyyppisten tiedostojen lataus palveluun toimimaan ja näin säästettiin huomattavasti aikaa ja pystyttiin keskittymään olennaisempiin asioihin.

Kirjastoja mietittäessä oli harkittava tarkkaan, oliko niistä saatu hyöty niin suuri, että ne kannatti ottaa käyttöön. Jokaisessa kirjastossa oli haasteena sen käyttämisen harjoittelu, sen sovittaminen olemassa olemaan ohjelmaan ja joissain tapauksissa epäsopivuuksia ohjelmistoversioissa. Mielestäni kirjastoista saatu hyöty etenkin kuvien osalta oli todella merkittävä enkä esimerkiksi galleriaa lähtisi toteuttamaan puhtaalta pöydältä, koska valmiita ja toimivaksi todettuja ratkaisuja löytyi jo niin paljon.

8.1 Dropzone.js

Kuvien tallentamiseen palvelimelle käytettiin Dropzone.js-kirjastoa, jonka avulla oli mahdollista ladata useita tiedostoja kerrallaan joko tavanomaisesti painiketta painamalla tai vaihtoehtoisesti drag&drop-tyylillä. Dropzone näyttää tiedoston latauksen etenemisen sekä tarjoaa tiedoston poistomahdollisuuden heti kuvan lataamisen jälkeen. Dropzonella voi myös suorittaa validointia, kuten rajata tiedoston tyyppiä ja kokoa sekä luoda

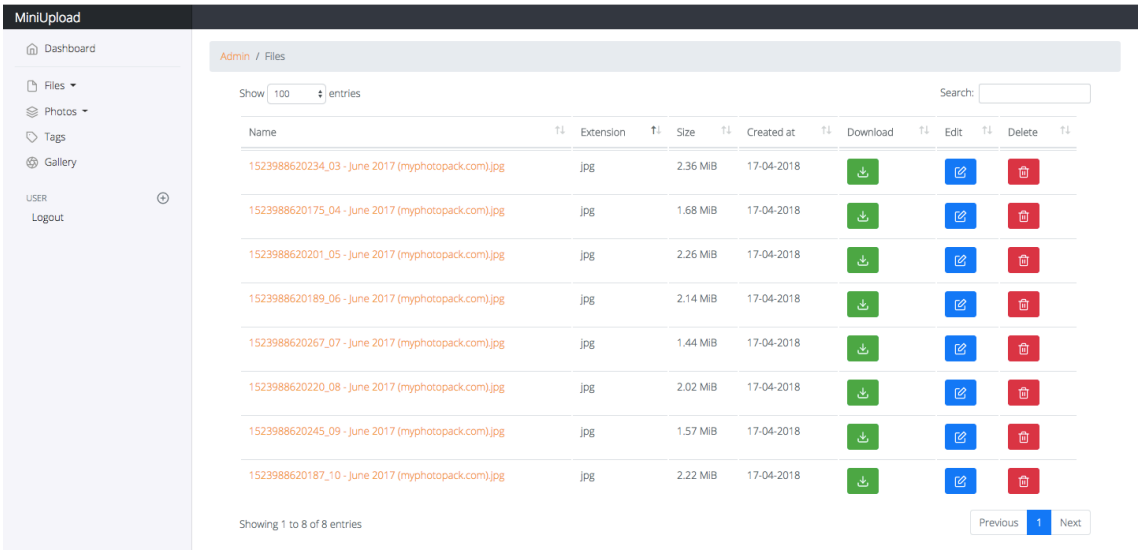
thumbnaileja kuvista jo niiden latausvaiheessa. Dropzone oli kuitenkin vain front-end toteutus tiedostojen lataamiselle ja itse logiikka sille täytyi tehdä kontrolleritasolla.



Kuva 15. Kuvatiedostojen lataaminen palveluun Dropzonella.

8.2 Datatables

Datatables on jquery-liitännäinen, jolla luotiin taulukko tiedostojen ja kuvien listamuotoista selaamista varten. Datatables tarjosi tiedostojen hakutoiminnon ja tiedostojen lajittelemisen niin nimen, koon, tyypin kuin luomispäivän perusteella sekä sivutuksen (pagination) näkymätasolla. Datatables tarjosi myös mahdollisuuden vaihtaa sarakkeiden paikkaa taulukossa, jolloin käyttäjä pystyi muokkaamaan niiden järjestystä mieltymystensä mukaan.



Kuva 16. Tiedostojen listaaminen Datatablesilla.

8.3 Justified.js

Projektin galleriaosuus luotiin käyttämällä Justified.js:ää, joka on jquery-lisäosa. Justified.js mm. laskee automaattisesti sen, miten kuvat sijoitetaan sivulle. Gallerian muotoilua varten voitiin helposti määrittää omia asetuksia, kuten maksimi rivikorkeus ja reunuksien paksuus.

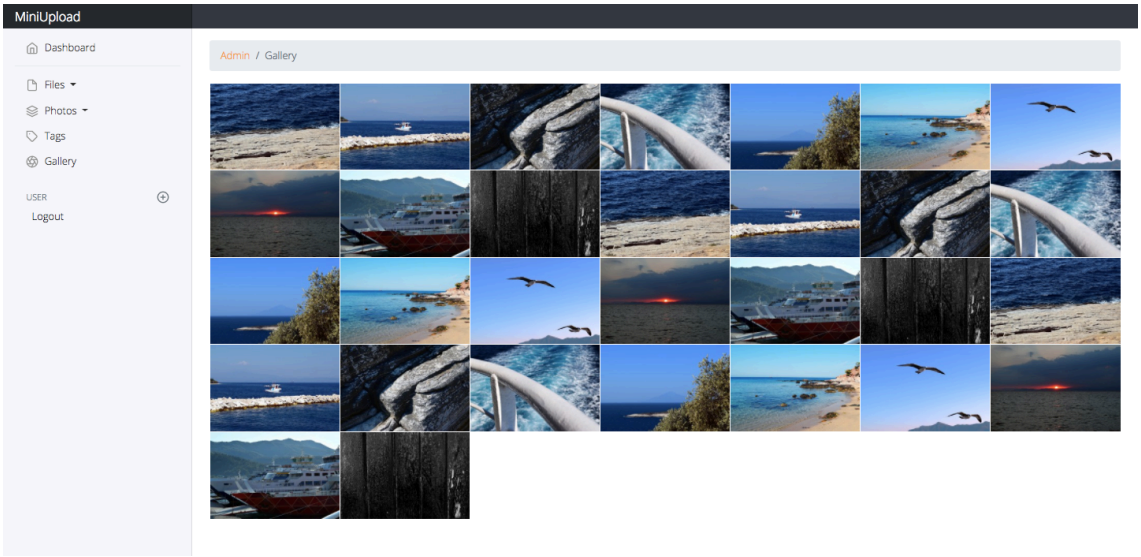
Justified.js:än asentaminen oli varsin suoraviivainen prosessi. Galleria-sivu laitettiin perimään admin-layout, sillä tähän näkymään haluttiin mm. navigointipalkki, jquery, bootstrap ja samat fontit, jotka admin-layoutissa oli jo määritelty. Head-osioon lisättiin Justifiedin tyylitiedosto ja javascript osioon Justifiedin javascript-tiedosto.

Kirjastolle täytyi enää kertoa, minkä id:n omaavaan elementtiin sen tulee kohdistaa galleria ja sen jälkeen iteroida storage-kansion tallennettujen kuvien läpi, jotta galleria saatiin populoitua kuvatiedostoilla.

```
@extends('layouts.admin)

@section('head')
     <link rel="stylesheet" href="{{ url('/css/justified.min.css') }}">
@endsection

@section('js')
    <script src="{{ url('/js/jquery.justifiedGallery.min.js') }}"></script>
@endsection

<script>
```

```
      $('#gallery').justifiedGallery({
            lastRow : 'nojustify',
            rowHeight : 100,
            rel : 'gallery3,
            margins : 1
      });
</script>

<div id="gallery">
      @foreach($photos as $photo)
            <a href="/storage/images/{{ $photo->filename }}">
                  <img src="/storage/images/{{ $photo->resized_name }}">
            </a>
      @endforeach
</div>

@endsection
```

Esimerkkikoodi 21. Justified.js:än konfigurointi.



Kuva 17. Gallerianäkymä, joka on luotu Justified.js:llä.

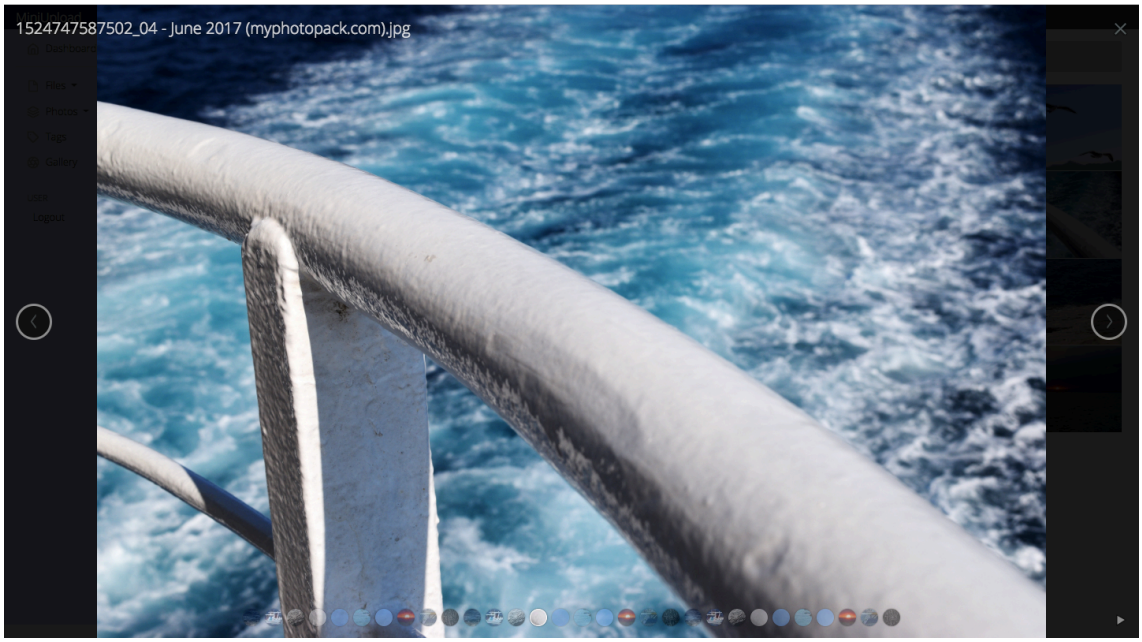
8.4 Blueimp gallery

Justified.js:än lisäksi projektissa käytettiin Blueimp galleryä, joka tarjosi karuselli-ominaisuuden, kuvien esikatselumahdollisuuden ja lightbox-toiminnallisuuden sivuston gallerian kuviin. Galleriassa pystyi Blueimpin ansiosta navigoida hiirtä tai näppäimistöä käyttämällä tai laittamaan halutessaan automatisoidun diaesityksen päälle.

Blueimp-galleria asennettiin samalla tavalla kuin Justified.js galleria. Head-osioon lisättiin liitännäisen vaatimat css-tiedostot ja javascript-osioon sen vaatimat javascriptit. Galleria kohdistettiin samalla tavalla tiettyyn html-elementtiin id:n perusteella.

```
<script>
document.getElementById('gallery').onclick = function (event) {
    event = event || window.event;
    var target = event.target || event.srcElement,
        link = target.src ? target.parentNode : target,
        options = {index: link, event: event},
        links = this.getElementsByTagName('a');
    blueimp.Gallery(links, options);
};
</script>
```

Esimerkkikoodi 22. Blueimp-galleryn kohdistaminen tiettyyn elementtiin.



Kuva 18. Kuvien selausnäkymä, joka toteutettiin Blueimp-galleryllä.

9 Yhteenveto

Tämän insinöörityön tavoitteena oli tutustuttaa lukija kokonaisvaltaisen web-sovelluksen tekoon ja saada aikaan toimiva pilvitallennustila. Mielestäni tavoitteisiin päästiin, sillä sovelluksesta tuli toimiva ja itselle hyödyllinen. Sovelluksen tekeminen oli todella opettavaista, koska se sisälsi niin paljon työkaluja ja tekniikoita, joita oli opiskeltava. Kokemusta tuli eritoten palvelimen konfiguroinnista ja erilaisten kirjastojen integroimisesta sovellukseen.

Sovelluksen kehityksessä käytettiin avuksi Laracastsin videoita ja muita yksittäisiä tutoriaaleja. Palvelimen konfiguroinnissa etenkin Digital Oceanin oppaat ja ongelmien ilmetessä Stack Overflow:sta löytyneet ohjeet helpottivat työtä todella paljon.

Sovelluksessa oli tarkoitus saada pienellä konfiguraatiomuutoksella tiedostot latautumaan Amazonin S3-palveluun ja sovellus saatiinkin toimimaan S3-tallennustilan käyttöliittymänä, mutta ominaisuutta ei keretty kuin testata yhden tiedostotyypin osalta.

Insinöörityössä aikaan saatu sovellus mahdollistaa sen, että pääsen verkon yli käsiksi omiin tiedostoihini ja pystyn käyttämään niitä kaikkialta.

Sovellusta kehittäessä huomioitiin tietoturvallisuus alusta alkaen ja se tulee myös jatkossa olemaan keskeisessä osassa. Sovellus ei vielä tue videotiedostojen toistoa eikä tiedostojen jakamista julkisiksi, mutta kehitystyö jatkuu tämän insinöörityön jälkeen ja sovellusta muokataan vastaamaan entistä paremmin omia tarpeita.

Lähteet

1. Pilvipalvelu. Verkkoaineisto: <https://yksityisille.hub.elisa.fi/mika-on-pilvipalvelu/>. Luettu 5.3.2018.

2. Sähkönkulutus. Verkkoaineisto: <https://www.vattenfall.fi/energianeuvonta/sahkon-kulutus/sahkolaitteiden-energiankulutus/>. Luettu 24.4.2018.

3. Sähkönkulutus. Verkkoaineisto: <http://www.voimatori.fi/energiatietoa/Kulutuslas-kuri/fi_FI/Mita_maksaa/>. Luettu 24.4.2018.

4. Laravelin palvelimen vaatimukset. Verkkoaineisto: <https://laravel.com/docs/5.6/in-stallation#server-requirements>. Luettu 26.4.2018.

5. Laravelin asentaminen. Verkkoaineisto: <https://laravel.com/docs/5.6>. Luettu 26.4.2018.

6. Homestead. Verkkoaineisto: <https://laravel.com/docs/5.6/homestead>. Luettu 26.4.2018.

7. Tietokantamigraatiot. Verkkoaineisto: <https://laravel.com/docs/5.6/migrations>. Lu-ettu 26.4.2018.

8. Eloquent. Verkkoaineisto: <https://laravel.com/docs/5.6/eloquent>. Luettu 26.4.2018.

9. Reititys. Verkkoaineisto: <https://laravel.com/docs/5.6/routing>. Luettu 26.4.2018.

10. Tunnistautuminen. Verkkoaineisto: <https://laravel.com/docs/5.6/authentication>. Luettu 26.4.2018.

11. CSRF. Verkkoaineisto: <https://laravel.com/docs/5.6/csrf>. Luettu 26.4.2018.

12. Validaatio. Verkkoaineisto: <https://laravel.com/docs/5.6/validation>. Luettu 26.4.2018.

13. Palvelimen asentaminen. Verkkoaineisto: <https://www.digitalocean.com/commu-nity/tutorials/initial-server-setup-with-ubuntu-16-04>. Luettu 23.4.2018

14. Laravelin asentaminen palvelimelle. Verkkoaineisto <https://www.howto-forge.com/tutorial/install-laravel-on-ubuntu-for-apache/>. Luettu 26.4.2018