LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Migraatioprosessi

Case: Fun Academy Oy

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2017
Mikael Wahlfors

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

WAHLFORS, MIKAEL:

Migraatioprosessi
Case: Fun Academy Oy

Ohjelmistotekniikan opinnäytetyö, 55 sivua, 7 liitesivua

Kevät 2017

TIIVISTELMÄ

Opinnäytetyössä käydään läpi jo tuotannossa olevan sivuston uudistamisprosessia, jossa koko sivusto uudistetaan kerralla. Työssä käydään pääasiallisesti backend API -rajapinnan siirtoa Lumen-ohjelmistokehykseen. Työ käsittelee Lumen-ohjelmistokehyksen yleisimpiä arkkitehtuurillisiä piirteitä ja lisäosia. Opinnäytetyössä käydään läpi joitakin kryptografian peruskäsitteitä ja yleisimpiä tiivisteitä.

Kohteena uudistusprosessilla oli Fun Academy Oy:n Fun Learning Community -tuote, joka on opettajille tarkoitettu sosiaalinen verkosto, jossa opettajat ympäri maailmaa voivat tehdä yhteistyötä jakamalla ideoita muiden opettajien kanssa.

Uudistamisprosessin tavoitteena oli rakentaa uusi laadukkaampi yhteisöpalvelu vanhasta yhteisöpalvelusta, joka olisi nopeampi, modulaarisempi ja tietoturvallisempi.

Sivuston toteuttamisessa käytettiin Lumen ohjelmistokehyksen lisäksi monia teknologioita, joista mainitsemisen arvoisia ovat esimerkiksi Redis -välimuistitietokanta, joka paransi sivuston suorituskykyä. Lisäksi uuteen toteutukseen rakennettiin kattava yksikkötestaus käyttämällä phpunit -yksikkötestauskehystä.

Ongelmia työn aikana oli, mutta projektin lopptuloksessa päästiin tavoitteeseen. Vanha sivusto voitiin korvata uudella sivustolla, joka täytti kaikki sille annetut kriteerit. Lukijan olisi hyvä tietää olio-ohjelmoinnin perusteita. Lisäksi on lukijan hyödyllistä tietää joitain ohjelmistokehityksessä käytettyjä menetelmiä.

Asiasanat: PHP, Lumen, bcrypt, API, SQL, Redis, MVC, Migraatio

Lahti University of Applied Sciences
Degree Programme in Information Technology

Wahlfors, Mikael:

Migration Processes
Case: Fun Academy Oy

Bachelor's Thesis in Software Engineering, 55 pages, 7 pages of appendices

Spring 2017

ABSTRACT

Object of this thesis was to document the process of building better version of existing web-site that is in production. Thesis will mainly consist of migration of back end API to a Lumen framework. Paper will go through basic architecture of Lumen framework and addons used by it. Thesis briefly go over also basics of cryptrocraphy and common hashing principles.

The target of migration process was Fun Learning Community website that is a social network for teachers. Community will provide place for teachers to collaborate and exchange ideas. Communitys is part of product family of Fun Academy Oy.

Project endgoal was to provide new website to the customers that is faster, more modular and secure.

To reach the goal Lumen framework was used along side multiple independent technologies which include Redis that is a In-memory database that is used to provide faster response times to client. Unit tests were build with phpunit for the new web-site to provide more quality to the product.

There were many problems during the process of making, but end goal was reached. The new site fulfilled the criterias which were made for it in the start of the project. Old site was completely replaced with the new one. Its recommended that the reader of this thesis should know basics of object oriented programming. Also, basic knowledge of software architectures would be advantage for the reader.

Key Words: PHP, Lumen, bcrypt, API, SQL, Redis, MVC, Migration

LYHENTEET JA KÄSITTEET

Apache HTTPD

Apache Foundationin avoimen ja vapaan lähdekoodin WWW-palvelinohjelmisto. Se löytyy tavallisesti asennettuna GNU/Linux-pohjaisista käyttöjärjestelmistä. Apache on maailman eniten käytetyin WWW-palvelinohjelmisto (w3techs 2017).

API — API eli Application programming interface on määritelmä, jonka mukaan eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja eli keskustella keskenään.

bcrypt — bcrypt on Niels Provosin ja David Mazièresin kehittämä yhdensuuntainen tiiviste, joka perustuu Blowfish-salauskirjoitusjärjestelmään.

HTTP — HTTP eli Hypertext Transfer Protocol on protokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon.

HTTPS — HTTPS eli Hypertext Transfer Protocol Secure on HTTP-protokollan ja TLS/SSL-protokollan yhdistelmä, jota käytetään tiedon suojattuun siirtoon webissä.

JSON — JSON (lyhenne sanoista JavaScript Object Notation) on yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.

LAMP — LAMP on kokoelma avoimen lähdekoodin ohjelmia, jotka yhdessä muodostavat WWW-palvelimen, jonka alla voidaan suorittaa dynaamisia websivuja. LAMP koostuu sanoista Linux Apache MySQL ja Perl/PHP/Python.

MD5 — MD5 on niin kutsuttu message-digest-algoritmi, jota käytetään muun muassa kryptografiassa. MD5 on yksi monista Ronald Rivestin kehittämistä tiivistealgoritmeista, ja se perustuu aikaisempaan MD4-algoritmiin

MVC — MVC-arkkitehtuuri (sanoista model-view-controller eli malli–näkymä–käsittelijä) on ohjelmistoarkkitehtuurityyli, jonka tarkoituksena on käyttöliittymän erottaminen sovellusaluetiedosta. MVC-arkkitehtuuria käytetään etenkin graafisten käyttöliittymien suunnittelussa ja ohjelmoinnissa.

PHP — PHP Hypertext Preprocessor on olio-pohjainen ohjelmointi- ja skriptauskieli, joka alun perin kirjoitettiin tietojen hakemiseen, kä- sittelyyn ja esittämiseen suoraan HTML-sivuilla. Nykyään PHP on täysimittainen ohjelmointikieli. Se on löyhästi tyypitetty ja siitä löytyy muun muassa Garbage Collector -toiminto.

TLS	TLS eli Transport Layer Security aiemmin tunnettu nimellä Secure Sockets Layer (SSL), on salausprotokolla, jolla voidaan suojata Internet-sovellusten tietoliikenne IP-verkkojen yli.
XML	XML (Extensible Markup Language) on metakieli, jolla määritellään rakenteellisia merkkauskieliä.

SISÄLLYS

1 JOHDANTO

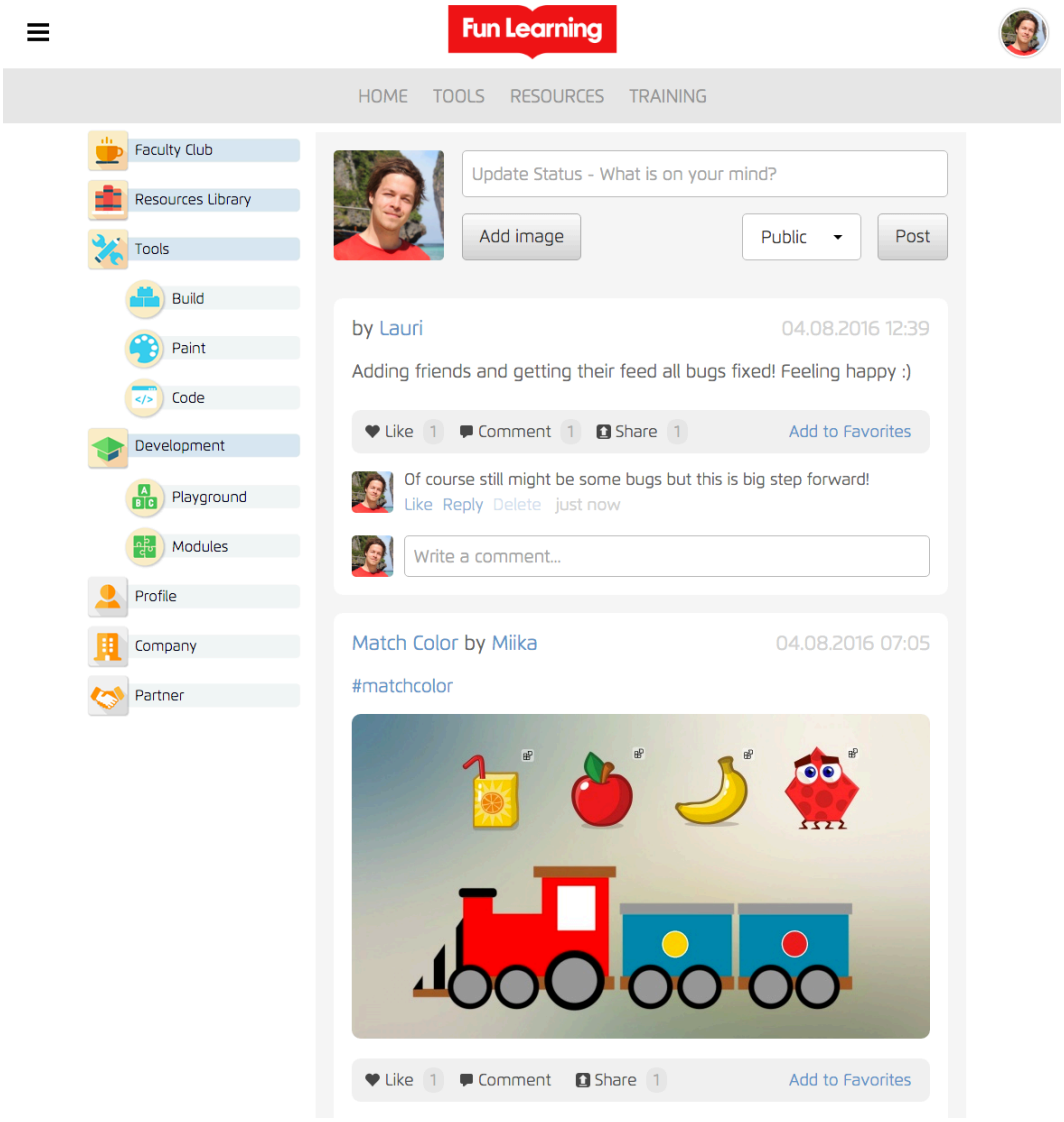
Verkko muuttuu valtavaa vauhtia, ja jo muutaman vuoden vanhat sivut voivat olla vanhentuneet. Verkon megatrendejä ovat muun muassa. mobiilikäyttäjien räjähdysmäinen kasvu, sosiaalinen median integroituminen lähes kaikkeen kuviteltavissa olevaan, käyttökokemuksen merkityksen korostuminen ja helppokäyttöisten julkaisujärjestelmien yleistyminen. Lisäksi verkossa pärjääminen vaatii yrityksiä muuttumaan medioiksi, mihin harva vielä kykenee. Migraatioprosessi voi olla ratkaisu tähän ongelmaan.

Migraatioprosessin tarkoitus oli uudistaa jo olemassa olevaa Fun Learning Community -yhteisöpalvelua (kuvio 1). Yritystoiminnan takana oli Fun Academy Oy. Yhteisöpalvelun tarkoituksena oli tarjota opettajille paikka, jossa he voivat jakaa toisilleen ideoita ja tehdä yhteistyötä. Palvelua käytetetään ympäri maailmaa, ja sillä oli prosessin aloittamisvaihessa tuhansia käyttäjiä. Yhteisöpalvelu on osa tuoteperhettä, joka on luotu parantamaan opetuksen laatua ympäri maailmaa. Tuoteperheeseen kuuluu vuonna 2017 yhteisöpalvelun lisäksi Builder-niminen palvelu, jossa ihmiset voivat nopeasti ja helposti luoda opetuskäyttöön laadukkaita pelejä ilman kokemusta ohjelmoinnista tai pelituotannosta.

Opinnäytetyössä käydään läpi back end migraatioprosessia ja tähän liittyvien ongelmien ratkontaa. Migraatioprosessi voidaan jakaa kahteen eri vaiheeseen, joista toinen oli front end ja toinen back end. Työssä käsitellään Laravel Lumen –mikro-ohjelmistokehyksen avulla rakennettua kokonaisuutta, jonka tavoite oli korvata vanha back end API. Lumen on uusi Laraveliin pohjautuva, laajasti käytetty kokonaisuus, jonka pääasiallisena ohjelmistokielenä toimii PHP. Ohjelmistokehys on pääasiallisesti tunnettu sen käytöstä API-rajapintojen tuotannossa. Vanha back end oli rakennettu puhtaasti PHP:n avulla ilman ohjelmistokehystä.

Samalla myös uudistettiin palvelun front end, mutta siihin ei tässä lopputyössä perehdytä. Uusi front end rakennettiin käyttäen kahden ohjelmistokehyksen sekoitusta. Nämä kaksi valittua ohjelmistokehystä

olivat Vue.js ja AngularJS 2.0.



KUVIO 1. Vanha yhteisöpalvelu

2 COMMUNITY

Kun Fun Learning Community -yhteisöpalvelun kasvu lähti nousuun, haluttiin taata tuotteen laatu ja turvallisuus. Tästä syystä haluttiin uudistaa yhteisöpalvelun teknologiat, jonka avulla kokonaisuus toimii. Web-ohjelmoinnin saralla oli lähiaikoina tapahtunut myös paljon muutosta. Päätelaitteet, joilla käyttäjät selasivat nettisivuja, olivat nopeutuneet ja käyttäjän päässä pystyi tekemään enemmän palvelimen pään työstä. Lähes jokaisella käyttäjällä löytyi älypuhelin, jolla sivua voitiin selata. API rajapinnan hyödyntäminen oli hyvin tunnettu tapa, jolla saatiin minimoitua palvelimen kuorma lähettämällä ainoastaan raakaa tietoa selaimelle. Käyttäjän laite käsittelee tämän tiedon ja rakentaa tiedon päälle käyttöliittymän.

Olemassa oli jo toimiva sivusto, joka oli toteutettu käyttäen pääasiallisesti PHP ja JavaScriptink.-ohjelmointikieltä ilman ohjelmistokehystä. Käytössä oli myös HTML-merkintäkieli. Lisäksi käytössä olevasta kokonaisuudesta puuttui monia haluttuja ominaisuuksia, kuten ryhmät ja mainostetut julkaisut, jotka haluttiin uuteen sivustoon. Vanhan toteutuksen tiedot oli tallennettuna MySQL–relaatiotietokantaan, jonka rakenne haluttiin pitää samana versioon 1.0. Salasanoissa käytettiin vanhaa MD5-algoritmia, joka haluttiin uudistaa turvallisempaan bcrypt-algoritmiin.

Vanhan toteutuksen palvelimet sijaitsivat AWS -pilvipalvelussa. Palvelimen käyttöjärjestelmänä toimi Linux Ubuntu ja Web-serverinä toimi Apache 2.

3 MENETELMÄT

3.1 Ohjelmistokehys

Ohjelmistokehys tarkoittaa tuotetta, joka muodostaa ohjelmiston rungon. Yleensä ohjelmistokehys on oliopohjainen, mutta poikkeuksiakin on, kuten Fusebox, joka on täysin proseduraalinen ohjelmistokehys. Ohjelmistokehyksen tarkoitus on tarjota kehittäjälle valmiita kokoelmia, joita hyväksi käyttäen ohjelmoija voi rakentaa oman ohjelmiston. Ohjelmistokehys nopeuttaa ohjelmistoprojektien työn osuutta, koska ohjelmoijan ei tarvitse rakentaa ohjelmiston kaikkia osia itse. Ohjelmistokehyksessä voi olla myös ohjelmistoarkkitehtuureja käytössä, kuten MVC. Arkkitehtuureilla saavutetaan modulaarisuus ja selkeys ohjelmistoprojektissa. (Koskimies & Mikkonen 2005,58-59 187-188)

Ohjelmistokehyksen koodi on monella ihmisellä käytössä ympäri maailmaa, joten kehyksen koodi on hyvin testattua ja täten laadukasta. Isosta käyttäjäkunnasta on myös apua ongelmatilanteissa. On tyypillistä, että ohjelmistokehyksen lähdekoodi on avointa. Avoimuudesta johtuen koodi on kaikkien katsottavissa ja mahdolliset tietoturvattomat aukot koodissa saadaan korjattua nopeasti.

3.2 Back end

Web-ohjelmistosuunnittelussa yleensä rajataan ohjelmisto kahteen eri kategoriaan, joista toinen on back end. Back end tarkoittaa ohjelmistokokonaisuutta, joka toimii palvelimen päässä. Suurin osa back end -puolen palvelimen työstä on käyttäjältä piilotettua, koska yleensä niissä käsitellään arkaluontoista tietoa tai tiedon käsittelytapa halutaan salata. Näistä yleisimpiä ovat kaikki tietokantoihin liittyvät prosessit, kuten käyttäjätietojen haku ja päivitys. Yleisiä ohjelmointikieliä back end -puolella on PHP, Python ja .Net. (Pluralsight 2015.)

Back end -ammattilaiselta vaaditaan myös tietoturvan tuntemusta, koska palvelimen puolella liikkuu paljon arkaluontoista tietoa, joka tulee pitää

ihmisiltä piilossa, kuten salasanoja. Tietoa voidaan salata monella tavalla, mutta yleisin on tiivistää tietoa yhdensuuntaisesti eri tiivisteratkaisuilla kuten MD5 tai bcrypt.

Vastausnopeus on myös tärkeää, joten tiedon haun ja käsittelyn prosessin optimointi on tärkeää. Usein kysyttyä tietoa voidaan siirtää välimuistiin, josta se on nopeampi hakea käyttäjän sitä kysyessä. Sosiaalisessa yhteisössä tämänkaltaisia asioita voi olla linkkien meta -merkinnöissä olevat kuvakaappaukset tai suosituimmat julkaisut verkkosivulla. (Pluralsight 2015.)

3.2.1 Laravel

Laravel on yksi suosituimmista web-ohjelmistokehyksistä, ja se on kirjoitettu PHP-ohjelmointikielellä. Laravelissa käytetty ohjelmistoarkkitehtuuri on MVC, joka on lyhenne sanoista Model, View ja Controller. Laravel on avoimen lähdekoodin ohjelmistokehys, jonka lähdekoodi on saatavilla GitHub -versionhallinnasta. Laravelin mukana myös tulee komentorivi työkalu nimeltään Artisan, joka avulla voidaan ajaa testejä, migraatioita ja ajastettuja tehtäviä. (Laravel 2017)

Taylor Otwell kehitti Laravelin vuonna 2011 yrityksenä tarjota edistyneemmän vaihtoehdon CodeIgniter -ohjelmistokehykselle, joka ei tarjonnut suoraan haluttuja palveluita, kuten autentikointia tai auktorisointia. Laravelin ensimmäinen betajulkaisu tuli testattavaksi 11. kesäkuuta 2011. Ensimmäisessä versiossa oli mukana autentikaatio, lokalisaatio, mallit, näkymät, ja sessiot. Versiosta kuitenkin puuttuivat ohjaimet, ja tämän puutteen takia Yläsävel ei kokonaan toteuttanut MVC-mallia. Laravel 1.0 julkistettiin samana kuukautena. (Maxoffsky 2013)

Laravel on julkaissut uuden version lähes vuosittain tästä vuoteen 2015 asti, jolloin Laravel 5 julkistettiin, jonka jälkeen kehitystahti on hidastunut. Versioiden mukana on tullut muun muassa mallintamismoottori Blade, jonka avulla voidaan luoda dynaamisia mallisivuja. Muita mainittavia

lisäyksiä Laravel -ohjelmistokehykseen ovat olleet Artisan, Scheduler ja ehkä suurimpana ohjaimet.

Laravel toteuttaa myös Eloquent Model –rajapinnan malleille, joten käyttäjän on helppo toteuttaa mallien logiikkaa rakentamatta SQL-lausekkeita manuaalisesti. Tämä mahdollistaa myös paremman tietoturvallisuuden, koska SQL-injektioita ei pysty tapahtumaan teknologiaa oikein käytettynä. SQL-injektio on yleisin tietoturvallisuusriski Web- ohjelmistoissa. (Maxoffsky 2013)

3.2.2 Lumen

Lumen on Laravel ohjelmistokehyksen kehittäjän Taylor Otwellin uusi projekti, joka on rakennettu Laravel ohjelmistokehyksen pohjalta, ja sen tarkoituksena on tarjota työkalut nopean API -rajapinnan rakentamiseen. Toimintatavoiltaan se muistuttaa Laravel ohjelmistokehystä. Se on riisutumpi versio Laravelista, joten monia toimintoja puuttuu puhtaassa asennuksessa. Puutteista johtuen se on kuitenkin huomattavasti Laravelia nopeampi. Lumen pystyy käsittelemään 1900 API kutsua sekunnissa, joka on huomattavasti muita samanlaisia sovelluskehyksiä enemmän. Verrattuna Silex -ohjelmistokehykseen, joka pystyy käsittelemän 1000 kutsua sekunnissa Lumen on nopeampi. Laravel pystyy käsittelemään noin 200 kutsua sekunnissa, joka on huomattavasti vähemmän. Laravel myös käyttää yli kaksinkertaisen määrän muistia kutsua kohden. Nämä arvot ovat tärkeitä sosiaalista yhteisöä rakentaessa, koska kutsuja tulee paljon, jos käyttäjämäärä kasvaa korkeaksi. (Lumen 2017e)

Koska Lumen on Laravel ohjelmistokehyksen pohjalta tehty, niin valmis Laravel rajapinta on helppo saada toimimaan Lumen ympäristössä. Lumen on tilaton ohjelmistokehys, joten sillä ei ole tietoa kutsua edeltävistä kutsuista, joten kaikki tarvittava tieto kutsun suorittamiseen pitää olla itse kutsussa. Koska Lumen on käytännössä typistetty versio Laravelista, monia ominaisuuksia joudutaan asentamisen jälkeen jälkiasentamaan. Näitä ominaisuuksia ovat esimerkiksi Facades ja Sessiot. (Lumen 2017e)

3.3 Front end

Front end on ohjelmistokehityksen toinen osa-alue, joka yleensä käsittää kaiken konkreettisen, jonka käyttäjä näkee omassa päätelaitteessaan. Front end käsitteenä on hyvinkin laaja, joten sen käsite elää muuttuvan maailman mukana. Varsinkin monessa ohjelmistoalan yrityksessä se voi tarkoittaa eri asiaa. Yleisiä Front end ammattitaitoja ovat HTML ja CSS ohjelmointi. Lisäksi front end –puolella on usein UX –suunnittelua. (Pluralsight 2015.)

Varsinkin lähiaikoina front end puolelle on tullut enemmän ohjelmistokehyksiä ja tästä johtuen enemmän logiikkaa. Tämä johtuu maailman tilanteesta, jossa käyttäjien laitteiden laskentateho on kasvanut. Internet yhteyksien nopeudet ovat myös kehittyneet nopeaa vauhtia. Nykytilanteen takia voidaan enemmän ja enemmän siirtää serverin laskentaa käyttäjän puolelle vähentäen serverin kuormaa. Tästä johtuen front end puolella tekniikoiden määrä on kasvussa ja osaamisen tarve kasvaa. Varsinkin reaktiivisten ohjelmistokehysten osaaminen on tärkeässä roolissa. Reaktiivisia ohjelmistokehyksiä ovat esimerkiksi React ja Vue.js. (Pluralsight 2015.)

3.4 Salaus

Salauksella tarkoitetaan arkaluontoisen tiedon muuntamista muotoon, jossa se on lukukelvotonta ihmiselle, joko väliaikaisesti tai lopullisesti. Salaus voi olla joko yhdensuuntaista tai kaksisuuntaista. Tapoja salata on monia, esimerkiksi kaksisuuntaisia salaustapoja ovat SSH, HTTPS ja GPG. Esimerkiksi saksalaisten toisessa maailmansodassa käyttämä Enigma oli kaksisuuntainen salaustapa. Kaksisuuntaisessa salauksessa periaatteena on, että tieto hajoitetaan lukemattomaan muotoon, mutta tämän jälkeen se voidaan vielä kasata uudestaan luettavaan muotoon. Kaksisuuntaisia salauskeinoja yleensä käytetään kahden päätelaitteen välillä viestintäsovelluksissa, jossa viestin sisältö on salattava, jottei ulkopuolinen pysty kuuntelemaan välitettävää tietoa. Jos ulkopuolinen

pystyy kaappaamaan viestin, tätä kutsutaan nimellä MITM eli Man In Theo Middle – hyökkäykseksi. (Surveillance Self-Defense. 2017)

Yksisuuntainen tiiviste eli hajautus on tapa luoda luettavasta merkkijonosta ihmiselle lukematon merkkijono pysyvästi. Yksisuuntaisissa tiivisteissä on tärkeää, ettei sitä voida enää muuttaa takaisin luettavaan muotoon. MD5, SHA-1 ja bcrypt ovat yksisuuntaisia tiivisteitä, näitä käytetään yleensä esimerkiksi käyttäjien salasanojen salaamiseen. (Schneider 2004)

Yleensä yksisuuntaisessa salauksessa myös käytetään suolaa, joka tarkoittaa käytäntöä, jossa salattavaan tietoon lisätään jokin satunnainen merkkijono, jonka tarkoitus on estää sanakirja ja rainbow table -hyökkäykset. Sanakirja hyökkäyksessä hyökkääjä käy läpi systemaattisesti tunnettuja sanoja sanakirjasta ja asettaa niitä salasanakenttään. Hyökkääjä jatkaa hyökkäystä kunnes oikea sana löytyy tai kaikki sanat on käyty läpi. rainbow table- hyökkäyksessä hyökkääjällä on valmis taulu sanoja ja niiden tiivisteitä. Hyökkääjä yrittää selvittää onko tiiviste jo listattuna taulussa. Jos tiiviste on listattuna taulussa voidaan olettaa, että tiiviste on muodostettu taulussa olleesta merkkijonosta. (ASPHeute 2004.)

3.4.1 MD5

MD5 on niin kutsuttu message-digest –algoritmi, jota käytetään muun muassa kryptografiassa. MD5 on Ronald Rivestin kehittämä tiivistealgoritmi ja sitä on edeltänyt MD4-algoritmi, joka osoitettiin turvattomaksi. MD5 tuottaa tuloksena 128-bittisen tiivisteen, joka yleensä osoitetaan 32- merkkisenä hexadesimaali muodossa (kuvio 2). Tavoitteena on luoda tiiviste josta ei päättelemällä pysty selvittämään mitään alkuperäisestä merkkijonosta. (MIT 1992)

Vuonna 2004 löydettiin MD5:llä koodatusta tiivisteestä heikkous, jossa kahden viestin MD5-tiivisteet olivat samat. Matemaattisesti merkkijonoja,

jotka tuottavat saman tiivisteen on rajattomasti, mutta niiden löytäminen on erittäin työlästä. (Arjen Lenstra. 2005)

Todellisuudessa ei ole mahdollisuutta tuottaa MD5-tiivisteestä alkuperäistä viestiä. Murtaminen onkin toinen asia, joka on mahdollista. Tämä toteutetaan niin, että eri merkkijonoista luodaan MD5-tiivisteitä ja verrataan niitä alkuperäiseen. Jos ne ovat samoja, voidaan olla lähes varmoja siitä, että tiiviste on tehty tietystä merkkijonosta. Normaalin käyttäjän päätelaitteen teho ei riitä yleensä tämäntapaiseen läpikäyntiin, mutta esimerkiksi isot supertietokoneet tai bottiverkostot, joilla on käytössä paljon laskentatehoa, on mahdollisuus tämän tapaiseen murtamiseen. (Peter Selinger 2006)

```
0a52730597fb4ffa01fc117d9e71e3a9
```

KUVIO 2. MD5 -tiiviste

3.4.2 bcrypt

bcrypt on Niels Provos:in ja David Mazières:in kehittämä yhdensuuntainen tiiviste (kuvio 3), joka perustuu Blowfish salauskirjoitusjärjestelmään. Se on kehitetty estämään rainbow-table ja bruteforce–hyökkäyksiä vastaan, koska murtajalta tarvitaan huomattavan paljon enemmän resursseja ja laskentatehoa saman lopputuloksen saavuttamiseksi. Koska bcrypt on suunniteltu olemaan hidas, niin se toimii todella hyvin bruteforce ja rainbow-hyökkäyksiä vastaan. bcrypt:in algoritmi on adaptiivinen, joten se on yhtä hidas, vaikka laskentateho kasvaa. bcrypt lisää salattavaan merkkijonoon automaattisesti suolan. (Niels Provos and David Mazieres 1999)

bcrypti:ä käytetään laajasti tietotekniikan toteutuksissa kuten monissa Linux distribuutiossa. Laajasta tarpeesta johtuen bcrypt on toteutettu monelle kielelle mukaan lukien C, C#, Java, JavaScript, PHP ja Python. Salausta ei ole murrettu vuoteen 2017 mennessä ja täten monet tietoturvallisuusammattilaiset suosittelevat bcryptiä.

```
$2a$04$OSn0X7.WnMOUMD5IRmfIROMqDfd4dHEpytbQySyJCRd1qRXEkzG0C
```

KUVIO 3. bcrypt -tiiviste

3.4.3 HTTPS

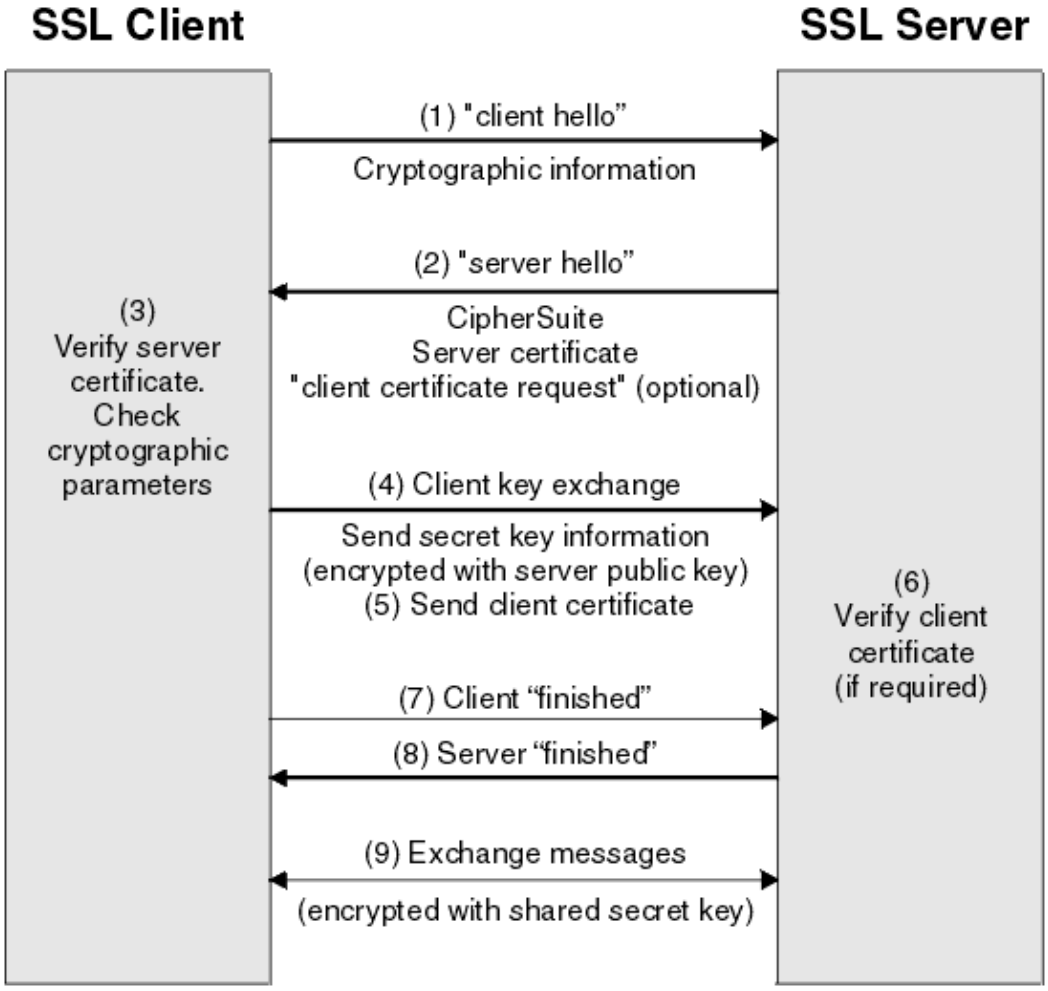
HTTPS eli Hypertext Transfer Protocol Secure on HTTP -protokollan ja TLS/SSL-protokollan yhdistelmä, jota käytetään salaisen tiedon siirtoon internetissä. Tiedot salataan ennen lähettämistä TLS-protokollan mukaisesti. TLS-salausta käytettäessä tarvitaan varmenne, jonka avulla voidaan selvittää paremmin kohde, joka on yhteydessä palvelimeen. Suomessa 15,04 % prosenttia nettisivuista käyttää HTTPS –protokollaa ja 45% sivulatauksista tapahtuu HTTPS-protokollan avulla. HTTPS-protokolla normaalisti käyttää porttia numeroa 443. (Dooli 2017)

HTTPS –protokollan tarkoitus on estää välitettävän tiedon lukeminen kahden kohteen välillä ja täten estää man-in-the-middle hyökkäykset. Yleisimmin HTTPS-protokollaa käytetään maksunvälityksiin, sähköpostien ja yritysten arkaluontoisen tiedon suojaamiseen internetissä. HTTPS on myös tärkeässä roolissa WIFI yhteyksien salaamisessa yleisissä verkoissa. (Eric Rescorla 2000)

HTTPS- protokollan askeleita on kolme kappaletta. Ensimmäinen askel on Hello, jossa kättely alkaa, kun asiakas lähettää ClientHello -viestin. Tämä viesti sisältää kaiken informaation, mitä palvelin tarvitsee asiakasyhteyden muodostamiseen SSL käyttäen (kuvio 4). Viestin mukana tulee monia salakirjoitus kokoelmia ja korkein SSL versio, jota asiakas tukee. Serveri lähettää takaisin ServerHello -viestin, joka sisältää asiakkaan tarvitsemat tiedot. Viestin mukana lähtee myös tieto, mitä salakirjoitusta ja SSL versiota käytetään ottaen huomioon asiakkaan toiveet. (Eric Rescorla 2000)

Toinen askel on sertifikaatin vaihto, jossa serveri todentaa itsensä asiakkaalle. Tämä toteutetaan käyttämällä SSL -sertifikaattia, joka

muistuttaa passia. SSL –sertifikaatti sisältää erinnäisiä palasia tietoa, kuten omistajan nimen, verkkotunnuksen, sertifikaatin julkisen avaimen, digitaalisen signeerauksen ja informaatiota sertifikaatin voimassaolo ajasta. Tämän jälkeen asiakas tutkii luottaako asiakas tai jonkin Certificate Authorities kohteista sertifikaattiin. (Eric Rescorla 2000)



KUVIO 4. SSL-handshake

Viimeinen eli kolmas askel on avaimien vaihto. Asiakkaan ja serverin välinen viestien salaus tapahtuu symmetrisellä algoritmillä, joka päätettiin Hello askeleessa. Symmetrinen algoritmi käyttää yksittäistä avainta salaukseen ja sen purkuun. Kummankin osapuolen on päätettävä yhteinen avain. Asiakas luo satunnaiset avaimen käytetäväksi algoritmissä. Avain salataan salauskirjoituksen avulla, jonka tekniikka sovittiin Hello askeleessa ja avaimena tähän toimii serverin julkinen avain. Asiakas lähettää salatun avaimen palvelimelle, missä salaus puretaan palvelimen

salaisella avaimella ja tämän jälkeen kättely prosessi on loppu. Sekä palvelin että asiakas ovat tyytyväisiä aloittamaan keskustelun. Viestit tästä hetkestä eteenpäin lähetetään salaamalla viesti yhteisesti sovitulla avaimella ja toinen osapuoli pystyy omalla avaimella salauksen purkamaan. (Eric Rescorla 2000)

3.5 Tietokannat

Tietokanta on kokoelma yhteenliittyviä tietoja. Tietokannat voidaan jakaa erityyppisiin tietokantoihin, joista yleisin on relaatiotietokanta. Relaatiotietokannat perustuvat predikaattilogiikkaan pohjautuvaan relaatiomalliin, jonka käsitteitä käyttäen tietokannan peruskäsitteet, kuten taulut määritellään. Relaatiotietokannoissa taulujen välille luodaan yhteyksiä, näitä yhteyksiä yleensä kutsutaan nimellä suhde. Taulut yhdistetään toisiinsa avaimien avulla. Relaatiotietokannassa on jokaisella rivillä yksilöivä avain, joka on yleensä surrogaattiavain. Surrogaattiavain on avain, jolla ei ole mitään tarkoitusta ja yleensä se on sattuinanen luku, kuten rivin numero. Vaihtoehtoisesti avain voi olla luonollinen avain eli bisnes avain. Bisnes avaimella on aina jonkin tarkoitus, kuten sosiaaliturvatunnus tai auton rekisterinumero. Tietokannan avain voi olla myös komposiitti avain, joka on kokolma kahdesta tai useammasta kentästä taulussa. Toisessa taulussa yhteyden luovaa saraketta kutsutaan vierasavaimeksi. (Breck Carter 1997)

Tietokantoja yleensä käytetään, kun tietoa halutaan säilyttää ja uudelleenkäyttää tulevaisuudessa. Tämäntapaisia tietoja ovat esimerkiksi käyttäjän tiedot, seuraajat, kuvat ja julkaisut sosiaalisessa yhteisössä.

3.5.1 MySQL

MySQL on avoimen lähdekoodin relaatiotietokannan ylläpitosovellus, jota kehittää Oracle Comporation. MySQL käyttää SQL kieltä, joka on lyhenne sanoista Structured Query Language. MySQL on keskeinen osa LAMP avoimen lähdekoodin kokoelmaa. LAMP koostuu sanoista Linux Apache

MySQL ja Perl/PHP/Python. MySQL käyttää maailmalla moni iso ohjelmisto kuten Symantec, Verizon, Nasa ja Neopets. (MySQL 2017a)

MySQL on kirjoitettu käyttäen C ja C++ ohjelmointikieliä. Sen SQL parseri on kirjoitettu käyttäen yacc kääntäjää. (MySQL. 2017b) MySQL pystyy käyttämään monilla käyttöjärjestelmillä, kuten Linux, Microsoft Windows, Oracle Solaris, Symbian ja SunOS. (MySQL 2017c)

3.6 Välimuistin tietokannat

Välimuistin tietokanta eli In-memory database (IMDB) tulee tarpeellisiksi, kun ohjelmiston käyttäjäkunta kasvaa ja halutaan minimoida palvelimen viiveet. Tallennettava tieto on yleisesti ei-relaationaalista ja pakattua, koska tallennustilaa on rajatusti. Välimuistin tietokantojen avulla voidaan vähentää kutsuja ulkopuolisiin SQL –tietokantoihin, jotka ovat hitaita verrattuna välimuistissa toimiviin. Lisäksi voidaan tallentaa monesti kysyttyjä laskentoja suoraan muistiin, jotta niitä ei jokaisella kyselyllä jouduta rakentamaan uudestaan. Välimuistiin tallennetaan väliaikaisesti useasti palvelimelta kysyttyjä tietoja, kuten suosituimmat kuvat ja linkeistä haettavat kuvankaappaukset sosiaalisen median viesteissä. (Mohit Kumar Gupta, Vishal Verma, Megha Singh Verma 2013)

Tietokantoja on monia, mutta yleisempiä on VoltDB, NuoDB ja Laravelin tukema Redis. Koska välimuisti on huomattavasta kiintolevyä nopeampi, saavutetaan tiedon haussa suuria eroja. Pääasiallisena tietojensäilytyspaikkana välimuisti on huono, koska tilaa on huomattavasti vähemmän ja hinta on korkeampi.

3.6.1 Redis

Redis on avoimen lähdekoodin In-memory database ratkaisu, jossa mahdollistetaan ohjelmiston tietojen tallennus palvelimen välimuistiin. Sen kehitti Salvatore Sanfilippo vuonna 2009 ja lyhenne koostuu sanoista REmote Dictionary Server. Redis liittää tallennettaviin tietoihin avaimen, jonka avulla tieto voidaan hakea välimuistista käyttöä varten. Tiedolle

asetetaan myös aika, jonka ajan se on muistissa ja tämän jälkeen se poistetaan. Redis on maailman käytetyin avain-tieto tietokanta. (Redis 2017)

Monet ohjelmointikielet tukevan Redis tietokantaa, kuten C, C#, PHP, Javascript ja Java. Laravel ohjelmistokehys, myös tukee Redistä vahvasti ja on suositeltu NoSQL –ratkaisu Laravelia käytettäessä. Suuri eroavaisuus Rediksen ja muiden NoSQL – tietokantojen välillä on se, että Redis tukee muitakin kun lukujono tyypisiä tietomuotoja kuten abstrakteja tietotyyppejä. Näitä ovat esimerkiksi lista, tai sarja lukujonot, sarja tiivisteet ja HyperLogLogit. (Redis 2017)

Redis pitää kaiken tiedon välimuistissa, mutta kirjottaa kaiken tiedon kiintolevylle 2 sekunnin välein siltä varalta, että palvelimelle tulee vikatila ja palvelin sulkeutuu nopeasti. Tämä estää arkaluontoisen tiedon häviämisen. Toimintanopeudentaan Redist pystyy suorittamaan keskimäärin 60 000 – 100 000 operaatiota sekunnissa, joka on noin 6 kertaa enemmän relaatiokantoihin kuten MySQL verrattuna. Redis toimii yhdessä prosessissa ja käyttää vain yhtä säiettä kerralla. Tästä johtuen Redistä ei voida käyttää töiden rinnakkaisajossa. (Rackspace 2017)

4 RAJAPINNAT

4.1 Rajapinta

Minimimuodossaan rajapintadokumentaatio kertoo, miten palvelu otetaan käyttöön, toisinsanoen. palvelun nimen, parametrit ja niiden tyypit sekä mahdollisen tuloksen tyypin. Tälle kokonaisuudelle käytetään nimitystä kutsumuoto (signature). Laajemmassa mielessä rajapinnan tulisi antaa kaikki se olennainen tieto palvelusta, mitä palvelun käyttäjän on syytä tietää. Rajapintoja voi olla monenlaisia kuten fyysinen, ohjelmisto tai GUI eli Graphical User Interface. (Koskimies & Mikkonen 2005, 187-188)

Fyysisen rajapinan esimerkkinä voidaan käyttää USB –liitintä kahden tietotekniikka laitteen välillä. Se mahdollistaa niiden välisen komminikoinnin fyysisen rajapinnan keinoin. Ohjelmallisen rajapinnan esimerkkinä voidaan käyttää kahden ohjelmistokielen rajapintaa Ajaxia, jossa kaksi ohjelmointikieltä kuten PHP ja Javascript pystyvät keskustelemaan keskenään. Ohjelmallista rajapintaa kutsutaan yleensä nimellä API (Application programming interface). GUI taas tulkkaa käyttäjän komentoja tietokoneelle ja tietokoneen vastauksia käyttäjälle joilloin käyttäjän ja tietokoneen välille syntyy rajapinta.

4.2 API

API eli Application Programming Interface, on ohjelmallinen rajapinta. Se määrittelee, miten suljettu ohjelmisto tarjoaa tietoja tai palveluita sovelluksille ja muille tietojärjestelmille. Rajapinta voi olla pelkkä tietorajapinta, jonka kautta saa luettua palvelun tietoa toisiin järjestelmiin, tai se voi olla toiminnallinen rajapinta, joka tarjoaa myös laskenta-algoritmeja tai mahdollisuuden muuttaa järjestelmän tietoja rajapinnan kautta. Jos järjestelmässä on useita erilaisia rajapintoja, on syytä täsmentää, mitkä niistä ovat avoimia. (Koskimies & Mikkonen 2005, 187-188)

Kuten GUI mahdollistaa ihmiselle helpomman tavan hallita ohjelmistoja, API tarjoaa kehittäjälle helpomman tavan käyttää valittuja teknologioita ohjelmistossaan. Piilottamalla tiedon prosessoinnin logiikan ja tuoden käyttäjälle dokumentaation vain sen käytöstä, API vähentää ohjelmoijan kognitiivista kuormaa. (Reddy Martin 2011, 1-2)

Web Service API on yleinen API muoto, joka on web sovelluksia varten rakennettu rajapinta back- ja front end välille. Kun esimerkiksi jaetaan uutta mediaa Facebookiin tai kysytään paikkatietoja Googleta, käytetään Web Service API :a. Web Service API kutsut läheteteään yleensä HTTP – kutsujen avulla, joita ovat GET, POST, PATCH, PUT ja DELETE. Näistä yleisin on GET, jota käytetään selaimessa tiedon hakuun. Takaisin tuleva vastaus API :lta on yleensä ennalta asetetun protokollan muodossa, yleensä JSON tai XML-muodossa, josta se voidaan käsitellä vastaanottajan puolella haluttuun muotoon.

API -rajapinnoilla on eri julkaisukäytäntöjä, jotka voidaan kategorisoida kolmeen eri tyyppiin: julkinen, yksityinen ja kumppani -API. Julkista API:ta voi käyttää kaikki ulkopuoleiset tahot. Esimerkiksi Apple on luonut julkisia rajapintoja. Näistä hyvinä esimerkkeinä ovat Carbon, joka tarjoaa tavan tuoda classic Mac OS -ohjelmistoja uuteen käyttöjärjestemäversioon nimeltään Cocoa. Myös Cocoa on itsessään rajapinta, jonka avulla voidaan rakentaa graaffisia rajapintoja Applen ohjelmistoihin. Yksityinen API on vain ja ainoastaan yrityksen omaan käyttöön. Kumppani API pystyy käyttämään vain tietyt yrityksen yhteistyökumppanit ja itse API:n omistama yritys. Esimerkiksi autoalan yritykset Uber ja Lyft sallivat muiden kolmannen osapuolien ohjelmien kutsua kyytejä omasta ohjelmastaan käyttäen yrityksen omaa API -rajapintaa. Tämän tyylinen API ratkaisu mahdollistaa API omistajan laadunvalvonnan rajapinnan käytöstä. Lisäksi tämä tuo yritykselle lisätuottoa, koska enemmän käyttäjiä käyttää heidän tuotettaan. (IBM 2017)

4.3 REST

REST eli Representational state transfer tai Restful Web services on vuonna 2000 Roy Fieldingin kehittämä tapa parantaa ohjelmistojen yhteentoimivuutta. REST perustuu toimintaan, jossa asiakas ja palvelin keskustelevat keskenään niin, että asiakaskone lähettää kutsuja joiden mukana on kaikki tarvittava tieto. Vastaanottavalla palvelimella ei ole tietoa edeltävistä tai tulevista kutsuista. (Roy Thomas Fielding 2000)

Yleisin tapa toteuttaa kommunikaatio palvelimen ja asiakkaan välillä REST protokollan mukaisesti, on tehdä se käyttäen HTTP -metodeja kuten GET, POST, PATCH, PUT, DELETE. Käyttämällä tilatonta protokollaa keskustelussa, voidaan mahdollistaa komponentien nopeus, modulaarisuus, uudelleenkäytettävyys ja vikasieto palvelimen puolella. Komponentteja on helppo ylläpitää ja korjata jopa ohjelmiston käynnissäolon aikana. (Roy Thomas Fielding 2000)

Vastauksena kutsun lähettäjälle REST –tuottaa kutsussa tai ennalta sovittua formaattia käyttäen. Yleisimpiä vastausformaatteja on JSON ja XML. Arkkitehtuurisia rajoitteita, joita REST asettaa rajapinnalle on monia. Rajoitteita ovat Asiakas-Palvelin -malli, tilaton protokolla, mahdollisuus tallentaa välimuistiin, kerros arkkitehtuuri ja yhtenäinen rajapinta. Asiakas-Palvelin -malli tarkoittaa toimintatapaa, jossa ohjelmiston rasite jaetaan asiakkaan ja palvelimen välille. Olotilattomuudella viitataan siihin, että palvelimella ei ole tietoa käyttäjän edeltävistä tai seuraavista kutsuista. Mahdollisuus tallentaa välimuistiin tarkoittaa sitä, että osa tiedosta voidaan sijoittaa välimuistiin, josta se on nopeasti haettavissa. Esimerkiksi aloitusivun siirtäminen välimuistiin parantaa sivun vastausnopeutta ja täten parantaa käyttäjäkokemusta. Kerrosarkkitehtuuri tarkoittaa ohjelmiston jakamista komponentti kerroksiin, jossa alemman tason komponentti kerrokset tarjoavat palveluita ylemmän kerroksen komponenteille. Yhtenäinen käyttöliittymä tarkoittaa, että rajapinnalla on mahdollisuus toimia omana osana, eikä se ole riippuvainen asiakkaasta. (Roy Thomas Fielding 2000)

Web service API REST palvelu rakentuu seuraavista komponenteista.

- runko URL, esimerkiksi http://api.example.com/resources/
- internet median tyypistä, joka määrittää olotilattoman tiedon välittämisen (esimerkiksi Atom, microformats, application/vnd.collection+json)
- standardin mukaiset HTTP -metodit kuten., OPTIONS, GET, PUT, POST ja DELETE.

4.4 JSON

JSON (lyhenne sanoista JavaScript Object Notation) on yksinkertainen avoimen standardin tiedon esitysmuoto tiedonvälitykseen (kuvio 5). Nimestään ja JavaScript-perustastaan huolimatta JSON on JavaScriptistä riippumaton. JSON protokollan Internet media type on application/json ja tiedostopääte on .json. JSON on maailman eniten käytetty tiedon formaatti asynkroonisessa Asiakas-Palvelin ympäristössä. JSON kehitettiin 2000 -luvun alkupuolella RESTful tyylin yleistyessä, sen tarpeiden mukaiseksi standardiksi. Alun perin JSON oli suunniteltu osaksi Javascript kieltä, mutta lopulta päädyttiin ratkaisuun, jossa JSON on kielivapaa tiedon formaatti. (Tim Bray 2014)

```
{
  "id": 4,
  "users_id": 457,
  "name": "second group with name",
  "membercount": 5,
  "images_id": 2,
  "cover": 0,
  "tags": "forth group with tags",
  "status": 2,
  "description": "third group with description",
  "material": "",
  "updated_at": "2017-03-09 12:26:07",
  "created_at": "2017-01-16 16:51:37",
  "postcount": 2,
  "role": 5,
  "image": {
    "id": 2,
    "image": "https://funlearning.fi/cloud/menu"
  },
  "coverimage": null
}
```


KUVIO 5. JSON esimerkki

JSON tukee monia eri tietotyyppejä kuten numeroita, merkkijonoja, boolean arvoja, taulukkoja, luokkarakenteita, ja tyhjiä arvoja. Syntaksissa JSON tukee tyhjiä merkkejä syntaksin keskellä ja sivuttaa ne elementtien sisällä ja ympärillä. Ainoastaan neljä kappaletta merkkejä luokitellaan JSON protokollassa tyhjiksi merkeiksi. Nämä neljä ovat välilyönti, vaakasuora sarkain, rivinvaihtomerkki ja vaununpalautus. JSON luontiin ja lukemiseen tarkoitetut kirjastot ovat laajasti käytössä monissa kielissä. (Tim Bray 2014)

4.5 HTTP Routing

Routing on Lumenissa tekniikka, jossa asiakkaan kutsu rajapinnassa yhdistetään ohjaimeen. Käytännössä routing toteutuu yhden route.php tiedoston avulla, jossa on listattuna kaikki mahdolliset kutsut, joita asiakas voi kutsua palvelusta. Yksittäistä yhdistystä kutsutaan nimellä route. Route

määrittää samalla oikean HTTP –metodin, kuten GET tai POST (kuvio 6). Jos metodi on väärä, Lumen palauttaa vataukseksi HTML –standardin mukaisen koodi 405 vastauksen, joka kertoo asiakkaalle, että käytössä on väärä kutsu. Routeen on mahdollista myös lisätä yksi tai useampi middleware (kuvio 7), joka asettaa sarjan koodirivejä suoritettavaksi ennen ohjaimeen siirtymistä. Hyvä esimerkki middlewaresta on authentikointi, jonka avulla tarkastetaan, onko pyynnön suorittavalla käyttäjällä oikeuksia suorittaa pyyntöä. (Lumen 2017f)

```
29
30      $group->post('logout', 'LoginController@Logout');
31
```

KUVIO 6. POST kutsu

Jos ohjelmistoprojektin koko kasvaa, routeihin pystyy määrittämään myös nimiavaruuden, jonka ohjaimia se käyttää. Lisäksi routeihin pystyy lisäämään myös etuliite, jonka tarkoitus on lisätä selvyyttä koodiin ja lisätä koodin modulaarisuutta. (Lumen 2017f)

```
17
18  $app->group(['middleware' => 'authToken'], function($group)
19  {
20      $group->put('user/update', 'UsersController@Update');
21  }
22
```

KUVIO 7. PUT kutsu, jolle on määritetty middleware ja ryhmä

4.6 Model

Model eli malli on luokka, jonka tehtävänä on muodostaa ohjelman bisnestaso. Bisnestason vastuulla on kaikki tiedon manipulointi, validointi, logiikka ja sen työnkulku toimialueen sisällä. Lumenissa malli on yleensä kuvaus tietokannan taulusta, mutta ne voivat kuvata mitä tahansa mikä muokkaa tietoa, kuten tiedostoa tai ulkopuolista verkkopalvelua. Laravel toteuttaa malli -rajapinnan nimeltään Eloquent ORM. Se perustuu yksinkertaisiin mallien välisiin kokonaisluku tyyppisiin suhteisiin. (Lumen 2017d)

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{

    //User that commented
    public function User()
    {
        return $this->hasOne('App\User', 'id', 'users_id')->select('id','images_id','
            username','firstname','lastname');
    }
    //Likes on the comment
    public function Likes()
    {
        return $this->hasMany('App\Like','target', 'id')->where('types_id',5)->with('
            user');
    }

}
```

KUVIO 8. Model esimerkki

Olio pohjaisessa ohjelmoinnissa malli voi olla esimerkiksi kuva, kommentti tai tykkäys (kuvio 8). Mallin nimi yleensä kuvaa tietokannan nimeä, esimerkiksi Lumenissa Users taulu mallin nimi olisi User. Mallin nimi voi olla kuitenkin erilainen tauluun verratessa, mutta silloin se tarvisee erikseen mainita mallissa. Mallien välillä on suhteita, kuten käyttäjällä voi olla monia kuvia, mutta kuvalla on vain yksi käyttäjä. Tämän tyylistä suhdetta kutsutaan nimellä yksi-moneen suhde. Suhteita on myös, yksi-yhteen ja moni-moneen tyylisiä. Suhteiden lisäksi voidaan malleissa suodattaa ja muokata suhteita. Esimerkiksi voidaan käyttäjä mallin kuvat hakea ja poistaa niistä kuvat, jotka ovat julkisia, jotta ainoastaan yksityiset kuvat palautetaan. Lisäksi kokoelmiin voidaan lisätä tietoja, kuten palvelimella oleva aika tai onko käyttäjä lisännyt sen suosikkeihin. (Lumen 2017d)

Koska malli luo rajapinnan tietokannan ja koodin välille, niin käyttäjän ei tarvitse käsin kirjoittaa Laravelissa SQL-lausekkeita (kuvio 9). Mahdollisuus tähän on, mutta se ei ole välttämätöntä. Rajapinnan ansiosta mahdollisten SQL-injektio tyylisten hyökkäyksien todennäköisyys laskee. Tietokannan tauluissa on myös yleensä aikaleima kentät, joista Eloquent malli huolehtii automaattisesti. updated_at kenttä kertoo viimeisimmän muutoksen riviin ja created_at rivin luontiajan. Aikaleima kentät ovat yleensä nimeltään updated_at ja created_at ja ne voidaan nimetä

uudestaan, mutta tämä muutos tarvitaan mainita mailleissa erikseen. (Lumen 2017d)

```
//SELECT * FROM `users` WHERE `role` = 5
$admins = User::where('role',5)->get();
```

KUVIO 9. Mallin SQL –lauseen rakennus verrattuna normaaliin SQL tapaan

Malli pitää myös huolen siitä, että arkaluontoista tietoa ei vuoda käyttäjän tai ulkopuolisen käsiin. Mallissa voidaan määrittää hidden kentät, joita ei palauteta mallin tietoja kysyttäessä (kuvio 10). Näitä kenttiä voi olla esimerksi käyttäjätaulun salasana tai maksutiedot. Malleihin voidaan myös sitoa tapahtumia, joiden tarkoitus on kuunnella tiettyjä tapahtumia ja toteuttaa rivejä koodia tämän tapahtuman tapahtuessa. Näitä tapahtumia voi olla esimerkiksi mallin päivitys tai poisto. (Lumen 2017d)

```
15
16          protected $hidden = [
17          'password','passwd','cookie','remember_token',
18          ];
19
```

KUVIO 10. Mallin hidden taulukkio

4.7 Controller

Controller eli ohjain-komponentti vastaanottaa käyttäjän syötteitä route palvelulta ja ne kutsuvat mallin palveluja (kuvio 11). Ohjaimen keskeinen tehtävä on sovelluksen logiikan ja käyttäytymisen muuntelu. Ohjain ei muuta sääntöjä, joiden mukaan malli toimii, vaan sovelluksen toiminnan muuntelu tapahtuu muokkaamalla käyttöliittymää tai muuttamalla tapaa, jolla käyttäjän syötteisiin reagoidaan. Tietyssä tilanteessa ohjain voidaan vaihtaa toiseen ohjaimeen, joka reagoi eri tavalla mallin tilan muutoksiin tai käyttäjän toimenpiteisiin. Sovelluksen ajonaikainen toiminnan muuntelu on erillisen ohjainkomponentin rakentava idea. (Lumen 2017a)

```
811
812     public function DeclineInvite(Request $request, $InviteID)
813     {
814         //Authenticate user
815         $user = Auth::User();
816
817         $invitation = Member::where('users_id',$user->id)->where('id',$InviteID)->
                where('role',3)->first();
818
819         //Check if invitation exists
820         if(!$invitation)
821             return Tools::error('invite not found',404);
822
823         //Delete invitation
824         $invitation->delete();
825
826         //Convert the response as json and return the value.
827         return response()->json(['declined' => true]);
828     }
829
```

KUVIO 11. Esimerkki ohjaimen funktiosta

Lumenissa ohjain odottaa routing kutsuja ja siirtää ne malleille suoritettavasksi. Suorituksen jälkeen ohjain palauttaa vastauksen takaisin kutsuvalle asiakkaalle, joka on yleensä json muodossa.

4.8 Middleware

Middleware on komponentti, joka suodattaa käyttäjän kutsuja ennen niiden siirtymistä ohjaimen käsiteltäväksi. Middlewarea käytetään esimerkiksi tarkastamaan, onko käyttäjällä tarvittavat oikeudet suorittaa kutsu (kuvio 11). Esimerkiksi vain kirjautuneet käyttäjät voivat suorittaa kuvien hakuihin liittyviä kutsuja. Tätä varten on luotu middleware, joka tarkastaa onko kutsun lähettäjä kirjautunut sisään. (Lumen 2017b)

```php
<?php
namespace App\Http\Middleware;
use Closure;
use Auth;
use App\User;
use App\Tools;
class AuthToken
{
    public function handle($request, Closure $next)
    {
        if(Auth::check())
        {
            return $next($request);
        }
        else
        {
            return Tools::error("Authentication failed",401);
        }
    }
}
```

KUVIO 11. Middleware

Middlewaren voi liittää yksittäiseen kutsuun tai kutsuryhmään (kuvio 7. s. 9). On mahdollista myös asettaa globaaleita middlewareja, jotka ajetaan ennen jokaista HTTP –kutsua. Authentikoinnin lisäksi esimerkiksi CORS middleware lisää ylätunnisteen vastauksiin, jotka palautetaan kutsun lähettäneelle asiakaalle. Lisäksi middlewarea voi käyttää hyväksi lokien kirjoittamisessa. (Lumen 2017c)

4.9 Migraatio

Migraatio ovat Laravelin osa, jonka tehtävä on toteuttaa versionhallinta tietokantamuutoksille. Migraatiot mahdollistavat tietokantamuutokset suoraan koodia muokkaamalla, jos sitä käytetään yhdessä schema builder toiminnallisuuden kanssa (kuvio 11). Jos kehittäjä on joutunut pyytämään toista kehittäjää lisäämään tietokantaan sarakkeen, on kehittäjä kohdannut ongelman, jota varten migraatiot on kehitetty. Rakentamalla tietokanta rakenteen koodin avulla schema builderia hyväksikäyttäen, luodaan yksittäinen ajettava migraatio. (Lumen 2017c)

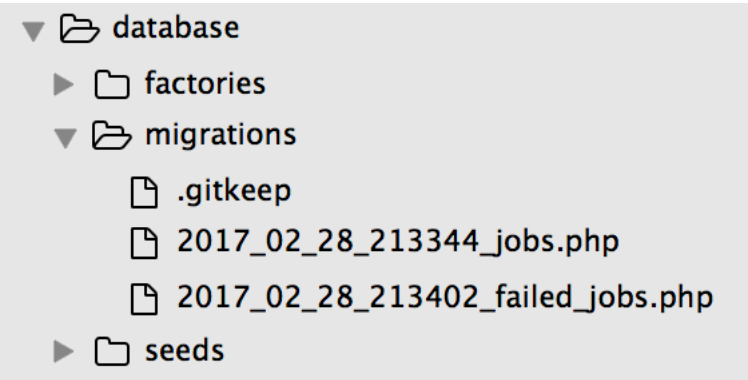
```php
class Jobs extends Migration
{

    public function up()
    {
        Schema::create('jobs', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('queue');
            $table->longText('payload');
            $table->tinyInteger('attempts')->unsigned();
            $table->tinyInteger('reserved')->unsigned();
            $table->unsignedInteger('reserved_at')->nullable();
            $table->unsignedInteger('available_at');
            $table->unsignedInteger('created_at');
            $table->index(['queue', 'reserved', 'reserved_at']);
        });
    }

    public function down()
    {
        Schema::drop('jobs');
    }
}
```

KUVIO 12. Migraatio schema

Käyttäjä ajaa migraation php artisan make:migration -komennolla, joka toteuttaa muutokset tietokantaan. Jokainen ajettu migraatio luo aikaleimalla varustetun tiedostoversion (kuvio 13), joita hyväksikäyttäen voidaan mennä tietokantaversioissa takaisinpäin käyttämällä php artisan migrate:rollback -komentoa. Lisäämällä migraatiot versionhallintaan voidaan pitää huolta, että kehittäjäympäristö on helppo rakentaa, kun tietokantarakenteen pystyy toteuttamaan yhden komennon avulla kohteeseen. (Lumen 2017c)



KUVIO 13. Aikaleimalla varustettu tietokantaversio

5 PILVIPALVELUT

Pilvi sanan tarkoituksesta tietotekniikasssa on paljon maailmalla kiistelyä. Monet ihmiset ajattelevat pilveä kokoelmana teknologioita. Se on totta, että pilvessä on kokoelma tunnettuja teknologioita, jotka yhdessä luovat pilviympäristön, mutta nämä eivät määrittele pilven olemusta. Pilvi on palvelu tai yhdistelmä palveluita. Tämän takia pilvi on vaikea määrittää. (Rountree Derrick, Castrillo Ileana 2014)

Alun perin pilvi oli kokoelma palveluita, teknologioita ja aktiviteettejä. Mitä pilven sisällä tapahtuu ei ole tiedossa ulkopuoliselle asiakkaalle. Tästä toimintatavasta pilvi on saannu nimensä. Nimen merkitys on muuttunut tästä ajan kuluessa. Pilvipalveluiden palveluntarjoajat ymmärsivät, että käyttäjät halusivat tietää, mitä pilvessä tapahtuu. Tästä johtuen monet palveluntarjoajat tarjoavat monitorointi työkaluja ja mahdollisuuksia muuttaa pilven toimintapoja asiakkaalle. (Rountree Derrick, Castrillo Ileana 2014

Pilvipalveluiden yleinen toimintatapa on tarjota *On-demand self-service* ratkaisua, jonka periaate on, että käyttäjä voi halutessaan käytää haluttua palvelua ilman pilvipalvelun ylläpitäjän erillistä hyväksyntää. Tämä vähentää ylläpitäjän taakkaa ja automatisoi pilvipalvelun prosessia. Esimerkki *On-demand self-service* tyylisestä ratkaisusta on AWS eli Amazon Web Services, joka on Amazonin tuoteperheen pilvipalvelu, jolla on 1,4 miljoonaa palvelinta 28 maassa. AWS tarjoaa monia palveluita, esimerkisi Amazon RDS, joka avulla voidaan rakentaa relaatio tietokantoja tietovarastointia varten pilveen.

6 OHJELMISTOTESTAUS

Ohjelmistotestaus on tapa tutkia ohjelmiston virheettymyyttä. Ohjelmistotestaaminen tarjoaa myös objektiivisen ja erillisen näkökulman ohjelmistoon, jonka avulla asiakas voi ymmärtää ohjelmiston implementoinnin riskejä. Testaustekniikka sisältää ohjelmiston suorittamista löytääkseen siitä ohjelmointivirheitä, mutta tekniikka ei rajoitu ainoastaan siihen.

Ohjelmistotestaus voidaan jakaa v-mallin mukaisiin lohkoihin jotka ovat: yksikkö-, integraatio-, järjestelmä- ja hyväksymistestaus. Testaamisen päätavoitteena on havaita ohjelmistossa ilmenevät häiriöt, jotta viat voidaan löytää ja korjata. Testaaminen ei voi vahvistaa, että ohjelmisto toimii oikein kaikissa tilanteissa tai olosuhteissa, vaan se osoittaa pelkästään, että se toimii oikein testeissä osoitetuissa ympäristöissä. (Jiantao Pan 1999)

6.1 Yksikkötestaus

Yksikkötestaus on yleinen tapa toteuttaa ohjelmistotestaus, se testaa pieniä ohjelmistokomponenteja tai –moduuleja pilkkoen ohjelmistokokonaisuuden osiin. Jokainen yksikkö testataan, jotta voidaan varmistaa, että yksikön yksityiskohtainen suunnittelu on toteutettu oikein. Objekti-orientoituneessa ympäristössä testaamista tehdään usein luokkatasolla ja minimaaliset testit sisältävät esimerkiksi rakentajien ja purkajien testaamisen. (Kshirasagar & Priyadarshi 2008,51-53)

Yleinen työkalu yksikkötestauksen toteuttamiseen on ohjelmisto nimeltään phpunit, joka on yksikkötestauksen rakennusta varten kehitetty ohjelmistokehys. Phpunitin avulla voidaan luoda ohjelmistolle yksikkötestejä, joiden avulla voidaan selvittää uuden ohjelmiston laatu testatussa ympäristössä. Yksikkötestien rakentaminen nopeuttaa, myös jatkokehitystä, koska uusien koodirivien jälkeen voidaan yhdellä komennolla tarkastaa, onko jonkin ohjelmiston osa rikki (kuvio 14). (Kshirasagar & Priyadarshi 2008, 51-53)

KUVIO 14. Epäonnistunut testi

7 HTTP-ILMOITUSKOODIT

HTTP-ilmoituskoodeilla palvelin viestii asiakkaalle hakupyynnön toteutumisesta. Yleisesti käytössä on vain muutamia koodeja, kuten maineikas 404 Not Found. Tässä kappaleessa esitellään HTTP/1.1-standardin mukaiset ilmoituskoodit, jotka jaetaan viiteen eri ryhmään. Nämä ryhmät ovat Informational, Succesful, Redirection, Client Error ja Server Error. Jokaisella ryhmällä on oma numero sarjansa, jonka sisällä ryhmään liityvät ilmoituskoodit on oltava. (IETF 2014)

Informational ryhmän koodi on 100-199 välillä ja palautetaan yleensä ennen varsinaista vastausta. Asiakkaan on oltava valmis vastaanottamaan yksi tai useampi koodi ennen varsinaista vastausta. Succesful ryhmän koodit ovat 200-299 välillä ja ne ilmoittavat asiakkaalle onnistuneesta pyynnöstä. Redirection ryhmän koodit ovat 300-399 välillä ja ne ilmoittavat asiakkaalle uudelleenohjauksesta. Asiakkaan täytyy hakea dokumentti jostain toisesta palvelimen antamasta osoitteesta. Client Error ryhmän koodit ovat 400-499 välillä ja ne ilmoittavat asiakkaan puolen virheistä. Server Error ryhmän koodit ovat 500-599 välillä ja ne ilmoittavat palvelimen puolen virheistä. (IETF 2014)

8 MIGRAATIO

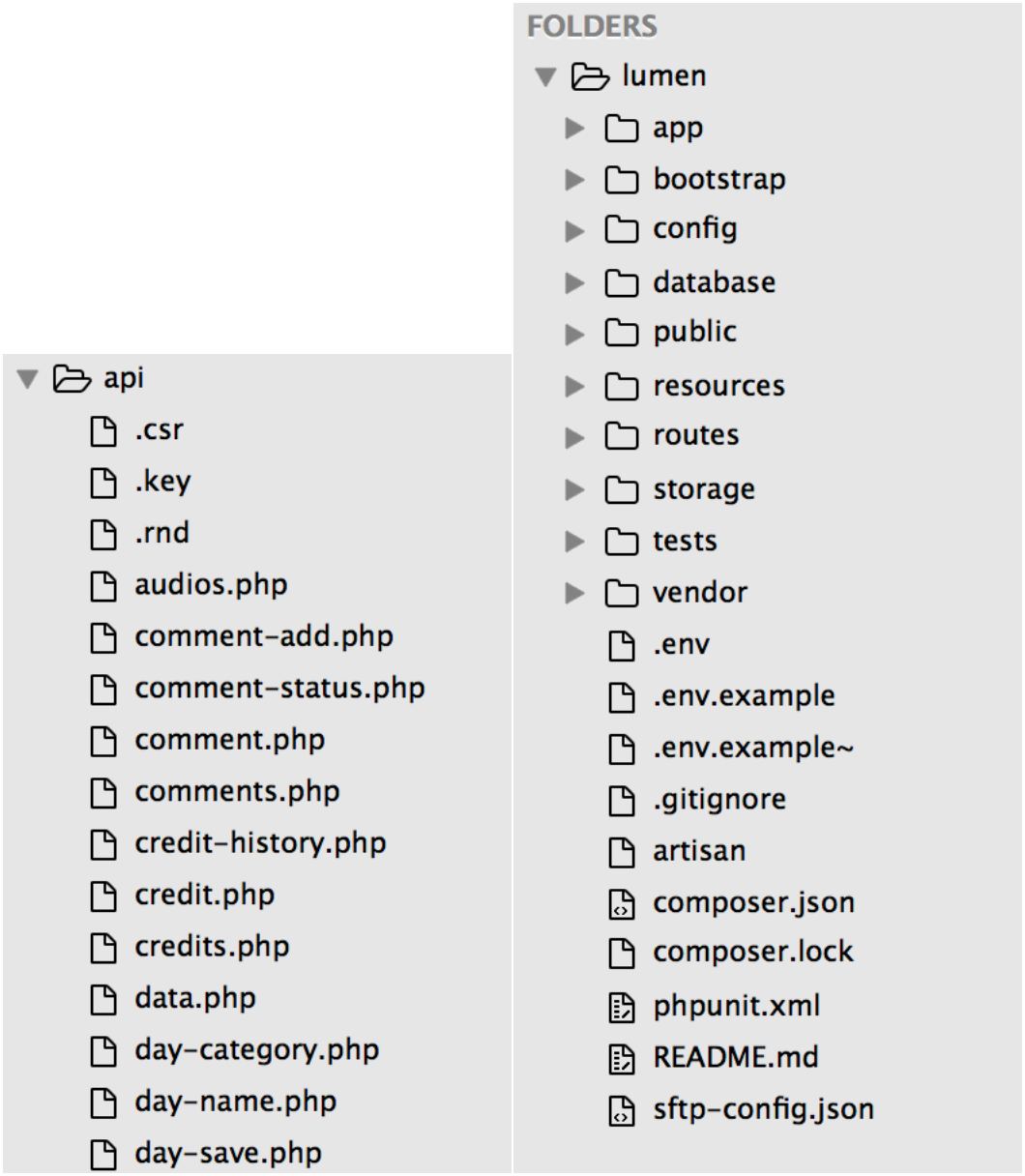
8.1 Funlearning community

Migraatioprosessin kohde palvelu on nimeltään Fun Learning Community, joka on sosiaalinen yhteisö. Se luo paikan opettajille kokoontua ja jakaa ajatuksia ja tehdä yhteistyötä. Fun Learning on osa Fun Academy Oy:n integroitua tuoteperhettä, johon kuuluu sosiaalisen yhteisön lisäksi myös Builder -ohjelmisto, jolla käyttäjä voi rakentaa opetuskäyttöön pelejä ilman ohjelmointia tai kokemusta pelin teosta. Builder ja Fun Learning jakoivat saman back end API -rajapinnan, joten muutokset API -rajapintaan vaikuttavat molempiin kokonaisuuksiin.

Builder palvelussa tehdyn pelin pystyi jakamaan napin painalluksella Fun Learning Community -palveluun. Lisäksi sivustolta löytyy kaikki sosiaalisen verkoston toiminnat kuten tykkäys, kommentointi, seuraus, jako ja suosikkien lisäys. Rekisteröityneitä käyttäjiä ennen migraatiota oli useita tuhansia ympäri maailmaa. Käyttäjistä suurin osa oli opettajia.

8.2 Vaatimusten määrittely

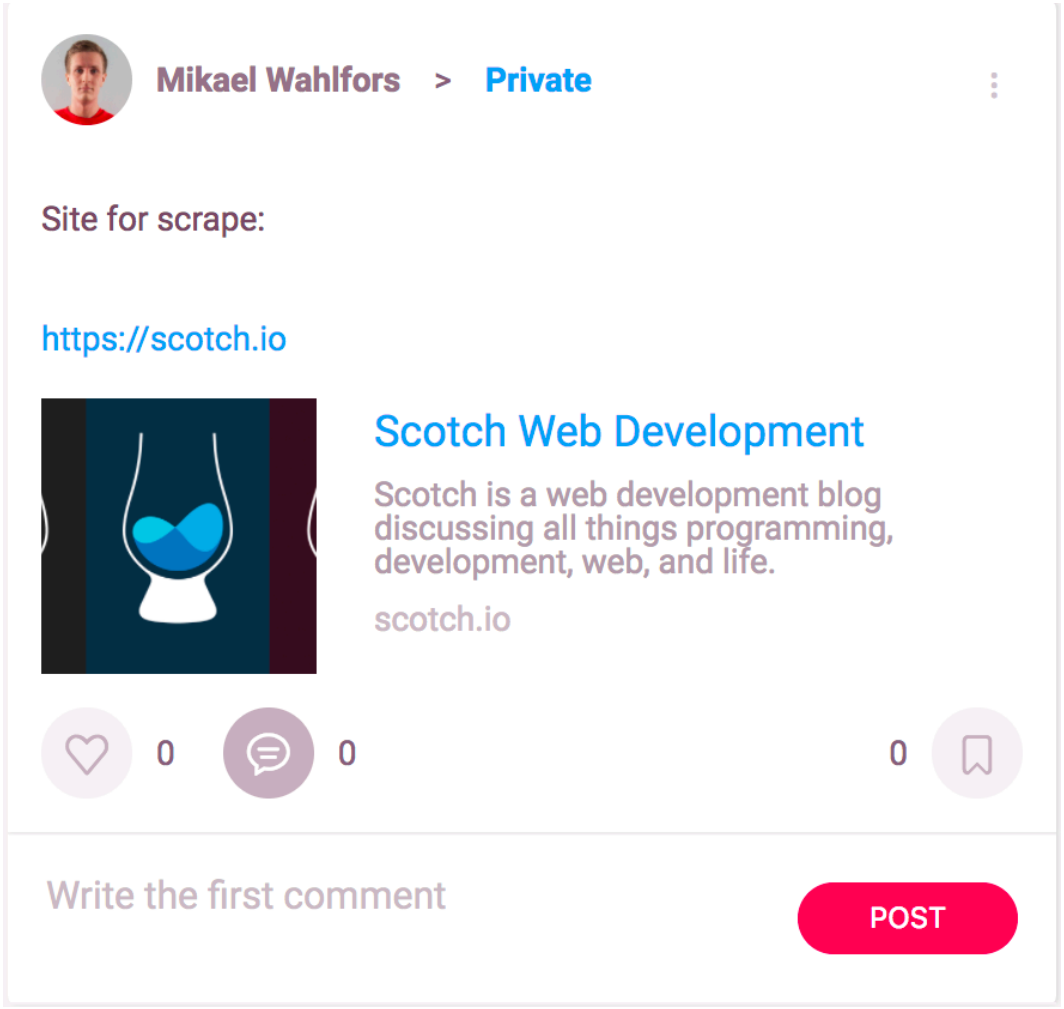
Vaatimuksia määritellessä oli tärkeää, että sivusto pysyi operatiivisena koko migraatioprosessin aikana. Sivustosta haluttiin tehdä tietoturvallisempi, laadukkaampi ja haluttiin käyttöön jokin ohjelmistokehys, koska haluttiin sivustoon modulaarisuutta. Lisäksi oli tärkeää, että sivuston back end tukisi monia yhdenaikaisia kutsuja. Koska Builder ja Community käyttävät samaa API rajapintaa ja täten samaa back end-palvelinta oli tärkeä suunnitella uusi sivusto niin, että molemman sivustot pystyivät käyttämään rajapintaa migraation aikana ja sen jälkeen. Edellinen API rajapinta oli tehty puhtaasti PHP -ohjemointikielellä ja siinä ei käytetty mitään ohjelmistokehystä. Vanhassa toteutuksessa kaikki PHP -tiedostot olivat samassa kansiossa (kuvio 15), kun taas Lumen -ohjelmistokehyksessä on tarkka ja selkeä tiedostorakenne (kuvio 15).



KUVIO 15. Vanha ja uusi tiedostorakenne

Uuteen ratkaisuun haluttiin kaikki vanhan API -rajapinnan toiminnallisuudet ja myös rajapintaan haluttiin lisätä uusia ominaisuuksia. Uusia ominaisuuksia ovat normaalit ja training –ryhmät, ryhmien toiminnot, käyttäjä moderointi, ilmoitukset tapahtumista, mahdollisuus mainostaa käyttäjien julkaisuja, ja sähköpostin vahvistus. Koska ohjelmiston käyttäjäkunta on kasvava, haluttiin palvelun skaalautuvuus korkeaksi, joten päädyttiin ratkaisuun, jossa web-sivuston palvelimet haluttiin pilvipalveluun. Tietokannan tiedot haluttiin säilyttää, joten tietokannan rakenne haluttiin pitää samana ensimmäiseen versioon asti. Tietokannat

sijaitsivat Amazon AWS -pilvipalvelussa jo vanhassa toteutuksessa, joten niiden käyttöä jatkettiin palvelussa. Tietokannan formaatti oli vanhassa toteutuksessa latin-1, mutta tämä haluttiin muuttaa, koska haluttiin tukea monia kieliä, joita tämä formaatti ei tukenut. Eniten haluttu kieli, jota haluttiin tukea, oli Kiina. Lisäksi haluttiin tuki emoji hymiöiden käyttöön Fun Academy -tuoteperheessä. Formatiksi valittiin utf8mb4, joka tukee 4-tavun emojien käyttöä sekä Kiinan kieltä. Tämä tuotti ongelmia, koska skandinaaviset kirjaimet rikkoutuvat, jos tietokannan formaatti vaihdetaan suoraan. Uuteen sivustoon haluttiin myös tuki scraper –teknologialle (kuvio 16). Jos käyttäjä lisäsi linkin sivustolle, scraper haki linkistä esikatselua varten kuvan, joka voitiin liittää käyttäjän julkaisuun. Tätä teknologiaa käyttää esimerksi Facebook ja Google+.



KUVIO 16. Scraper esimerkki

Koska uuteen toteutukseen haluttiin ohjelmistokehys, valittiin ohjelmistokehykseksi Lumen. Lumen 5.3 valittiin, koska se pystyi toteuttamaan kaikki määrittelyn vaatimukset, mutta samalla se oli kevyt kehys, jolla pystyi suorittamaan huomattavan määrän samanaikaisia kutsuja. Välimuistin hyötykäyttö on tärkeää monissa kyselyihin liittyvissä asioissa, jotta saadaan vastaus mahdollisimman nopeasti käyttäjälle. Välimuisti tietokannaksi valittiin Redis, koska Lumen ohjelmistokehyksessä oli vahva tuki sille suoraan. Palvelimelle haluttiin mahdollisuus sähköpostien lähettämiselle, koska käyttäjälle haluttiin lähettää sähköposteja monessa tilanteessa, esimerkiksi rekisteröitäessä. Jono teknologia haluttiin toteuttaa palvelimelle, jotta voitaisiin mahdollistaa

ajastetut sähköpostit. Jono on tärkeä, jotta käyttäjälle postiteta liikaa sähköposteja pienessä ajassa. Tämä olisi roskapostia, joka ärsyttäisi asiakasta ja huonontaisi asiakkaan käyttäjäkokemusta. Roskapostin määrää haluttiin vähentää mahdollisimman pieneksi, jotta käyttäjäkokemus parantuisi. Jonon käsittelijäksi valittiin myös Redis, koska se oli jo käytössä välimuisti tietokannan tarpeen takia. Jonon olisi voinnut myös rakentaa perinteiseen relaatiokantaan.

Tietoturvallisuuden parantamiseksi arkaluontoisen tiedon salausta haluttiin vahvistaa, koska vanha tieto oli tiivistetty käyttäen MD5 salausta. Uudeksi salaus tiivisteekksi valittiin bcrypt. Koska arkaluontoiseen tietoon kuului myös kaikki vanhassa tietokannassa sijaitsevat salasanat, niitä varten jouduttiin rakentamaan middleware, joka tutki oliko käyttäjällä jo uusi bcrypt salasana. Jos käyttäjältä ei löytynyt uutta bcrypt salasanaa back end automaattisesti loi käyttäjälle uuden bcrypt salasanan uuteen kenttään ja jätti vanhan MD5 salasanan voimaan, koska sitä vielä käytettiin vanhassa toteutuksessa.

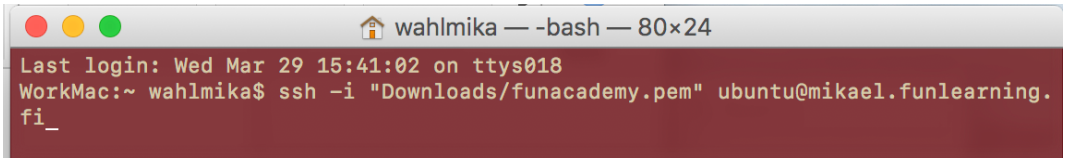
Autentikointi vanhassa sivustossa toteutettiin yksinkertaisella keksi ja token järjestelmällä, jossa käyttäjä lähetti oman käyttäjänimen ja salasanan palvelimelle. Jos käyttäjätiedot olivat oikein, vastauksessa palvelin asetti käyttäjälle token ja userid keksin. Lisäksi palvelin tallensi tämän keksin tietokantaan users taulun kenttään ilman tiivistettä. Kun käyttäjä lähetti API kutsun, palvelin tutki onko kutsun lähettäjän keksi validi. Jos keksi oli oikein, palvelin antoi käyttäjälle luvan suorittaa kutsun. Authentikaatiota haluttiin muuttaa uuteen toteutukseen paremmaksi. Suunnitelmissa oli muuttaa authentikaatio OAUTH 2.0 tyyliseksi ja erilliseksi palvelimeksi joka olisi yhteinen kaikille Fun Academy -tuoteperheen Back End -palvelimille. Aikarajoista johtuen tästä kuitenkin jouduttiin luopumaan. Autentikaatiota haluttiin kuitenki parantaa, joten niiden käyttötapaa muutettiin. Tokeneille haluttiin aikaraja ja ne haluttiin tiivistää tietokannassa, koska tietovuodon sattuessa haluttiin, että ne eivät olleet luettavissa.

Koska projektia haluttiin kehittää migraation jälkeenkin, vaatimuksena oli myös dokumentoinnin parannus. Vanhan rajapinnan dokumentaatio oli ainoastaan toteutettu koodin kommenteina ja sekin oli todella vähäistä. Uuden API -rajapinnan rajapintadokumentointi toteutettiin swagger –työkalulla. Swagger luo automaattisesti dokumentoinnin oikeinkirjoitetuista ohjain kommenteista. Koska uuden sivuston laatu haluttiin taata asiakkaille, luotiin phpunit ohjelmistolla yksikkötestit rajapinnalle.

Projektin aloitus tapahtui syyskuussa ja aikaraja projektin valmistumiselle asetettiin helmikuulle, joten aikaa projektin toteutukselle oli 6 kuukautta. Jonka jälkeen piti uusi front- ja back end olla valmiina ja täydessä toiminnassa asiakkaille.

8.3 Kehitysympäristö

Sivuston palvelimet perustettiin Amazon AWS –pilvipalveluun. Palvelimia pystytettiin kolme kappaletta, joista yksi oli live tuotannon palvelin. Toinen palvelin oli testipalvelin, jossa oli aina toimiva kokonaisuus rajapinnasta uusien ominaisuuksien testaamista varten. Kolmas palvelin oli kehityspalvelin, jota käytettiin uusien ominaisuuksien kehittämiseen ilman, että API olisi front end -puolen käytössä. Kaikki kolme palvelinta olivat Ubuntu 16.04.1 LTS -palvelimia ja sisälsivät Lumen 5.3 version. Kaikki palvelimet toimivat saman domainin (funlearning.com) alla, mutta jokaisella oli oma alidomain, jolla ne eroitettiin toisistaan. Versionhallinnassa käytettiin GitHub -palvelua, jossa oli repository sekä front end ja back end –osuudelle. Palvelimelle oli mahdollista yhdistää ssh yhteydellä käyttäen .pem tiedostoavainta (kuvio 17). Vanhan toteutuksen tietokantarakenne kopioitiin ja siitä tehtiin kaksi instanssia, joista toinen luotiin tuotantoon nimeltään funlearning ja toinen testiin nimeltään funtesting. Testikantaan kiinnitettiin testi- ja kehityspalvelin. Tuotannon tietokantaan liitettiin tuotannon palvelin.



KUVIO 17. ssh -yhteys .pem avainta käyttäen

8.4 Kehitysympäristön pystytys

Koko migraation ajan vanha sivusto toimii uuden rinnalla pääasiallisena portaalina sosiaaliseen verkostoon asiakkaille. Käyttäjällä oli mahdollisuus valita kumpaa sivustoa käyttää. Toteutuksen alkuvaiheesa oli tärkeää päästä mahdollisimman nopeasti tuotokseen, joka toteutti vanhan sivuston toiminnallisuuden. Toteutus aloitettiin palvelimien pystyksellä syyskussa. Se toteutettiin niin, että luotiin yksi palvelin instanssi ja asetettiin asetukset palvelimeen oikein. Tämän jälkeen palvelimesta otettiin image kopio, josta se kopiotiin kahteen uuteen instanssiin. Täten saatiin kolme täysin identtistä palvelinta kehitystä varten. Jokaiselle palvelimella asennettiin Lumen, Redis, Apache ja asetettiin oma .env tiedosto, jossa kaikki arkaluontoiset tiedot palvelimen käyttöä varten säilytettiin. Tiedossa oli esimerkiksi tietokannan salasanat ja osoitteet. Testi- ja kehityspalvelimella oli oma yhteinen tiedosto, mutta tuotannon palvelimella oli omansa, koska siellä oli käytössä eri tuotannon tietokanta. Se erosi muista palvelimista. Tätä tiedostoa ei oteta versionhallintaan mukaan, koska se sisältää arkaluontoista tietoa. Lisäksi kaikissa palvelimissa asetettiin käyttöön facades toiminto, joka on puhtaassa asennuksssa poistettu käytöstä. Facaces –toiminto mahdollistaa staattisten rajapinta luokkien käytön.

8.4.1 Kutsu rajapinnan luonti

Ensimmäinen ohjelmointia vaativa toteutuksen osa oli route rakenne kutsuille (kuvio 18). Tässä käytettiin hyväksi listaamalla vanhan sivuston kaikki kutsut niiden php tiedostoista ja jakamalla ne kahteen pääkategoriaan: todennusta tarvitseviin ja tarvitsemattomiin. Niihin jotka tarvisivat todennusta asetettiin middleware, joka todensi kutsun lähettäneen asiakkaan ennen kutsun siirtämistä ohjaimelle. Yhteensä kutsuosoitteita suunnitelmassa oli 71 kappaletta, mutta uusien ominaisuuksien taki määrä kasvoi 115 kappaleeseen, joista 101

kappaletta tarvitsi tunnistautumista. Loput 14 kappaletta ei tarvinnut tunnistaumista.

```
$app->group(['middleware' => 'authToken'], function($group)
{

    $group->post('logout', 'LoginController@Logout');

    $group->get('manuallink', 'LoginController@ManualLink');

    $group->post('resendverification', 'LoginController@ResendVerification');

    /*
        Users
    */

    $group->get('users', 'UsersController@Index');

    $group->get('users/{UsersID}/', 'UsersController@Show');

    $group->delete('users/{UsersID}/delete','UsersController@Delete');

    $group->put('users/{UsersID}/mute','UsersController@Mute');

    $group->put('users/{UsersID}/unmute','UsersController@UnMute');

    $group->get('users/{UsersID}/feed','UsersController@Feed');

    $group->get('users/{UsersID}/followers', 'UsersController@UserFollowers');

    $group->get('users/{UsersID}/following', 'UsersController@UserFollowing');

    $group->get('users/{UsersID}/favorites/images', 'UsersController@ImageFavorites')
        ;

    $group->post('users/{ID}/favorite', 'FavoritesController@Favorite');

    $group->put('users/{ID}/partner','UsersController@PartnerUser');

    $group->put('users/{ID}/role/update', 'UsersController@Role');
```

KUVIO 18. Route tiedosto

8.4.2 Mallit

Koska vaatimuksissa oli määritelmät vanhan tietokantarakenteen käytölle, voitiin suunnitella uuden toteutuksen mallit helposti. Mallien nimet tulivat suoraan vanhan kantarakenteen mukaan. Suurimmaksi osaksi voitiin luoda yksi malli, jokaista tietokanna taulua kohden. Malleja tuli suunnitelmaan yhteensä 19, mutta tämä kasvoi 24 kappaleeseen, koska uusia tauluja lisättiin uusien toiminnallisuuksien vuoksi. Lisäksi luotiin yksi malli Tools, joka toimi yleisten työkalujen säilytys luokkana. Tools luokkaan rakennettiin monia hyödyllisiä työkaluja, kuten domainin selvittäjä (kuvio 19), error viestin rakentaja, tiedostohakija ja scraper työkalu, jonka avulla voitiin linkin osoitteesta hakea esikatselua varten kuva. Näitä

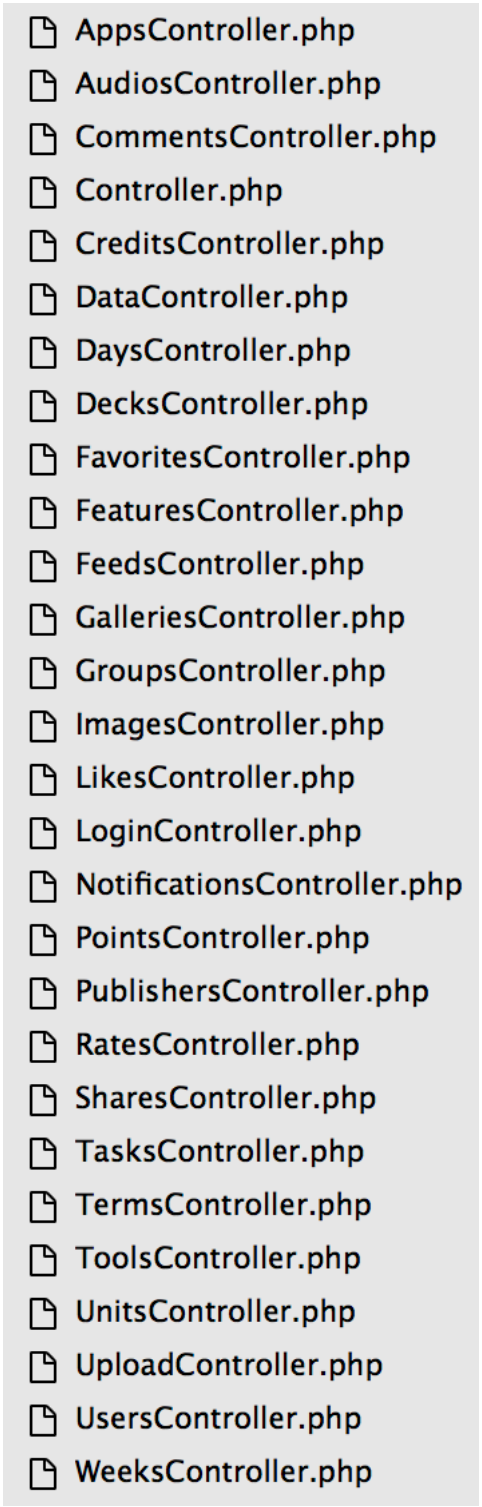
työkaluja käytettiin eri puolilla ohjelmistoa, joten niiden keskittäminen yhteen paikaan oli koodin ylläpidon kannalta järkevä ratkaisu. Mallien välille ohjelmointiin Eloquent ORM mukaiset suhteet, jossa niiden väliset suhteet kuvattiin. Mallien nimet pidettiin vakiona, joka tarkoittaa Lumenissa sitä, että mallin nimi on aina yksikkö muoto tietokannasta. Esimerkkinä *users* taulun model oli nimeltään *user.*

```
public static function extract_domain($domain)
{
    if(preg_match("/(?P<domain>[a-z0-9][a-z0-9\-]{1,63}\.[a-z\.]{2,6})$/i", $
        domain, $matches))
    {
        return $matches['domain'];
    } else {
        return $domain;
    }
}
```

KUVIO 19. Tools luokan funktio

8.4.3 Ohjaimet

Seuraavaksi lähdettiin suunnitelemaan ohjaimia, jotka voitiin suunnitella suoraan route kutsujen mukaan. Koska kaikki kutsut ovat listattuna route.php tiedostossa, voidaan ne ryhmitellä käytön mukaan ja jakaa ohjain ryhmiin. Ohjaimia tuli yhteensä 28 kappaletta (kuvio 20), joista 25 kappaletta oli käytössä heti ja loput 2 kappaletta otettiin käyttöön jälkikäteen. Lisäksi ohjelmoitiin yksi ohjain Controller.php, joka toimi pohjana kaikille ohjaimille. Tässä äitiohjaimessa asetettiin yhteiset muuttujat ja yhteinen rakentaja kaikille ohjaimille.



KUVIO 20. Uudet ohjain luokat

Kutsun tarkistus suoritettiin ohjaimissa käyttämällä Lumenin omaa validate komponenttia hyväksi käyttäen (kuvio 21). Validate komponentti tutki, oliko tarvittavat parametrit kutsun mukana ja oliko parametrit kelpaavia. Esimerkiksi oliko lisättävän kuvan nimi liian pitkä tai onko rekisteröitävän

käyttäjän käyttäjänimi uniikki. Jos tarkistus epäonnistui Lumen lähettää error viestin takaisin kutsuvalle asiakkaalle käyttäen http –ilmoituskoodia 400. API rajapinnan kaikki palautettavat arvot ovat json muodossa, joten kaikki ohjaimessa käsiteltävä tieto muunnettiin json muotoon ennen palautusta.

```
$this->validate($request, [
    'title' => 'required|unique:posts|max:255',
    'body' => 'required',
]);
```

KUVIO 21. Validate testi

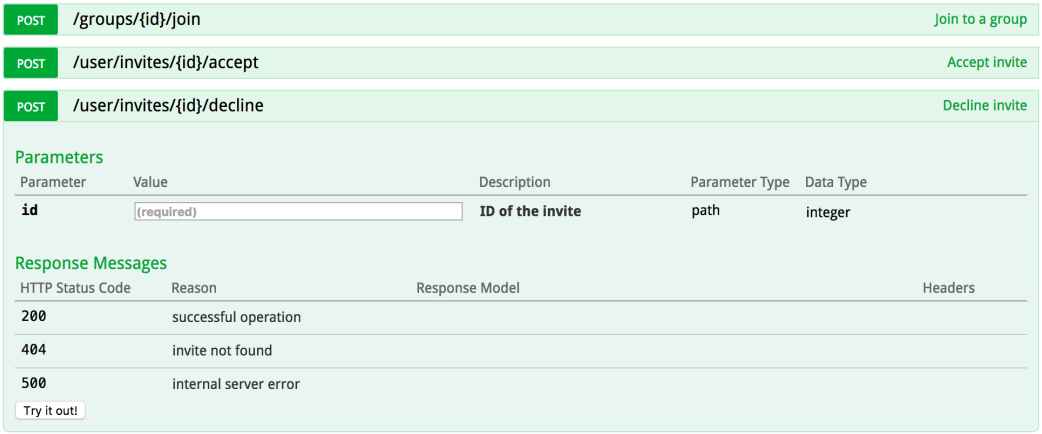
8.4.4 Rajapinnan dokumentointi

Ohjaimien ohjelmoinnin ohessa kirjoitettiin myös dokumentointia varten tarvittavat swagger standardin mukaiset kommentit, joiden avulla pystyttiin rakentamaan kattava kuvaus kaikista rajapinnan kutsuista (kuvio 22). Tätä varten jouduttiin asentamaan Lumeniin lisäosa komponentti SwaggerLume. Kommenteissa tulee ilmi kutsun url osoite, mitä pakollisia parametrejä se käyttää, mitä valinnaisia parametrejä sillä on, mitä virhekoodeja kutsu voi vastata, lyhyt kuvaus kutsun käytöstä, operaatio id ja minkä http –metodin avulla sitä kutsutaan. Lisäksi parametrien määrittelyssä voidaan antaa lisätietoa, missä parametri sijaitsee. Onko parametri jo url osoitteessa vai onko se esimerkiksi kutsun body –osioissa.

```
790
791     /**
792      * @SWG\Post(
793      *   path="/user/invites/{id}/decline",
794      *   summary="Decline invite",
795      *   operationId="postDeclineInvite",
796      *   @SWG\Parameter(
797      *     name="id",
798      *     in="path",
799      *     description="ID of the group",
800      *     required=true,
801      *     type="integer"
802      *   ),
803      *
804      *   @SWG\Response(response=200, description="successful operation"),
805      *   @SWG\Response(response=404, description="invite not found"),
806      *   @SWG\Response(response=500, description="internal server error")
807      * )
808      *
809      */
810     public function DeclineInvite(Request $request, $InviteID)
811     {
```

KUVIO 22. Swagger dokumentointi kommentti

Kommentoinnin jälkeen voitiin suorittaa linux terminalissa php artisan swagger-lume:publish –komento, jonka avulla lisäosa komponentti luo automaattisesti dokumentaatio sivuston aliosoitteeseen api/documentation (kuvio 23). Täältä kehitystiimin front end kehittäjän on helppo nähdä, mitä kutsuja palvelimen versio tukee jokaisena kehityksen hetkenä ja mitä niiden kuuluu tehdä. Dokumentaatiosivulta nähdään palautettavasta tiedosta esimerkkejä, joten palautettavan tiedon formaatti on luettavissa kehittäjälle.



KUVIO 23. Generoituva dokumentaatio

8.4.5 Autentikaation parannus

Seuraavaksi lähdettiin ratkomaan autentikaation ongelmia. Ensimmäisenä vaihdettiin arkaluontosen tiedon kuten salasanojen tiivistämistyyli. Vanhassa toteutuksessa tiivisteenä käytettiin MD5 –algoritmia, joka haluttiin vaihtaa vahvempaan bcrypt –algoritmiin. Ongelmana tuli vastaan käyttäjät, jotka käyttivät sivustoa ja joilla oli vanhan tyylinen salasana. Koska vanha toteutus käytti vielä md5 tiivistettä salasanan tarkastuksessa, MD5 tiivistettyä kenttää passwd ei voinnu vain korvata tietokannasta.

Päädyttiin ratkaisuun, jossa lisättiin users –tauluun uusi bcrypt kenttä password, jonka tarkotus oli toimia MD5 -tiivistetyn passwd kentän rinnalla, kunnes vanha toteutus poistettaisiin käytöstä. Uudistusta varten rakennettiin middleware, joka kirjautuessa tarkastaa onko käyttäjällä bcrypt salasanaa (kuvio 24). Jos käyttäjällä tätä ei löytynyt, niin middleware tarkasti onko käyttäjän antama salasana oikein MD5 tiivistettyyn kenttään verrattaessa. Jos salasana oli oikein, middleware automaattisesti loi käyttäjälle uuden bcrypt tiivistetyn salasanan password –kenttään. Jos käyttäjältä löyty bcrypt salasana middleware vertasi suoraan käyttäjän salasanaa siihin. Middleware liitettiin /login kutsuun, jotta se suoriutuu jokaisella kerralla, kun käyttäjä kirjautuisi sisään. Käyttämällä kyseistä ratkaisua saatiin käyttäjäkokemuksesta parempi, koska käyttäjän itse ei tarvinnut tehdä mitään paremman tietoturvallisuuden saavuttamiseksi. Käyttäjä antoi saman salasanan, kun ennenkin sivulle kirjautuessa.

```
//If user does not have bcrypt password make it.
if($user->password == NULL)
{
    //Checks if md5 password is valid
    if($user->passwd != md5($request->input('password'))){
        $userInfo = new User;
        $userInfo->username = $user->username;
        $userInfo->firstname = $user->firstname;
        $userInfo->lastname = $user->lastname;
        $userInfo->image = $user->Image;

       //Needed user data is sent to front
       return response()->json(['error' => 'Invalid password', 'user'
           =>$userInfo ],401);
    }
    //Generate user password from input
    $user->password=Hash::make($request->input('password'));
}

//If Bcrypt password is set checks for validity.
else if(!(Hash::check($request->input('password'),$user->password))){
        $userInfo = new User;
        $userInfo->username = $user->username;
        $userInfo->firstname = $user->firstname;
        $userInfo->lastname = $user->lastname;
        $userInfo->image = $user->Image;
    //Needed user data is sent to front
    return response()->json(['error' => 'Invalid password', 'user' =>$
        userInfo ],401);
}
```

KUVIO 24. Autentikaation middleware

Lisäksi jaettavien API -tokeneiden toimintatapaa muutettiin tietoturvallisemmaksi. Tokeneita varten luotiin oma taulu, koska haluttiin tokeineille muitakin käyttötarkoituksia, kuin kirjautumiskeksi. Taulu luotiin koska tokenin ja käyttäjän välinen suhde muuttui yksi-yhteen suhteesta, yksi-moneen suhteeseen. Tokeneille voitiin asettaa aika uudessa toteutuksessa ja tokenit salattiin tiivistämällä ne tietokanna puolella bcrypt –algoritmiä käyttäen. Käyttämällä bcrypt tiivistettä ennaltaehkäistään bruteforce tyylisiä hyökkäyksiä tietokantaa vastaan. Lisäksi, jos tietokantavuoto tapahtuu, api avaimet pysyvät salassa. Sivun keksien käytäntöä muutettiin paremman tuoteperhe-integraation vuoksi. Sivustolla kirjautuessa asetetaan päädomainin tasolla keksi, joka on käytettävissä kaikissa alidomaineissa. Jos käyttäjä kirjautuu community palvelussa ja

siirtyy tuoteperheeseen kuuluvalle sivustolle, hän on automaattisesti kirjautuneena kohteessa.

8.4.6 HTTP–virhekoodien standardisointi

HTTP –virhekoodien standardisointi tapahtui luomalla muokattuja vastausviestejä virheen tapahtuessa. Äitiohjaimeen luotiin lista joka sisälsi kokonaisluku ja validation parametri pareja. Kun ojaimessa tapahtui virhe, validate tutkii, onko virhekoodi listassa vastinetta kyseiselle tapahtumalle ja palautti alkion kokonaisluvun virhekoodin mukana. Standardisoituja kokonaislukuja haluttiin käyttää, koska sivustosta haluttiin tehdä monikielinen. Kun palautuva virhekoodin viesti on kokonaisluku, voidaan front end –puolella asettaa käyttäjän valitseman kielen mukaisesti sille oikean kielinen vastaus.

Uuden median latauksesta haluttiin nopeampi, joten niitä koskevia kutsuja muokattiin toimimaan osaksi asynkronisesti. Kun käyttäjä lähettää uuden kutsun, jonka mukana on ladattavaa mediaa, palvelin tunnistaa median ja suorittaa vain tarvittavat tallennukseen liittyvät prosessit kuten 640x640 pikselin esikatselukuvan luonnin ennen vastauksen palautusta. Asiat, joita ei käyttäjä tarvitse, heti suoritetaan asynkronisesti. Asynkronisesti suoritetettavia prosesseja olivat erikoisemmat esikatselukuvat ja median formaattikäännökset. Kuviossa 25 voidaan nähdä, kuinka osa työstä siirtyy suoritettavaksi asynkroonisesti, koska sitä ei tarvita kutsun vastauksessa.

```
297         if($request->type!=9) {
298
299             //Create job with delay
300             $job = (new UploadJob($code));
301             //Dispatch job
302             dispatch($job);
303
304             // create thumb (640px width jpg)
305             $mpl = 640/$w; if($mpl<1) { $pw = (int) $w*$mpl; $ph = (int) $h*$
                    mpl; } else { $pw = $w; $ph = $h; }
306             $thumb = imagecreatetruecolor($pw,$ph); $white =
                    imagecolorallocate($thumb,255,255,255);
307             imagefill($thumb,0,0,$white); imagecopyresampled($thumb,$src,0,0,0
                    ,0,$pw,$ph,$w,$h);
308             imagejpeg($thumb,"/var/www/cloud/".$code."-thumb.jpg",95);
309
310         }
311
```

KUVIO 25. Upload route asynkrooninen työ

8.4.7 Jonon luonti

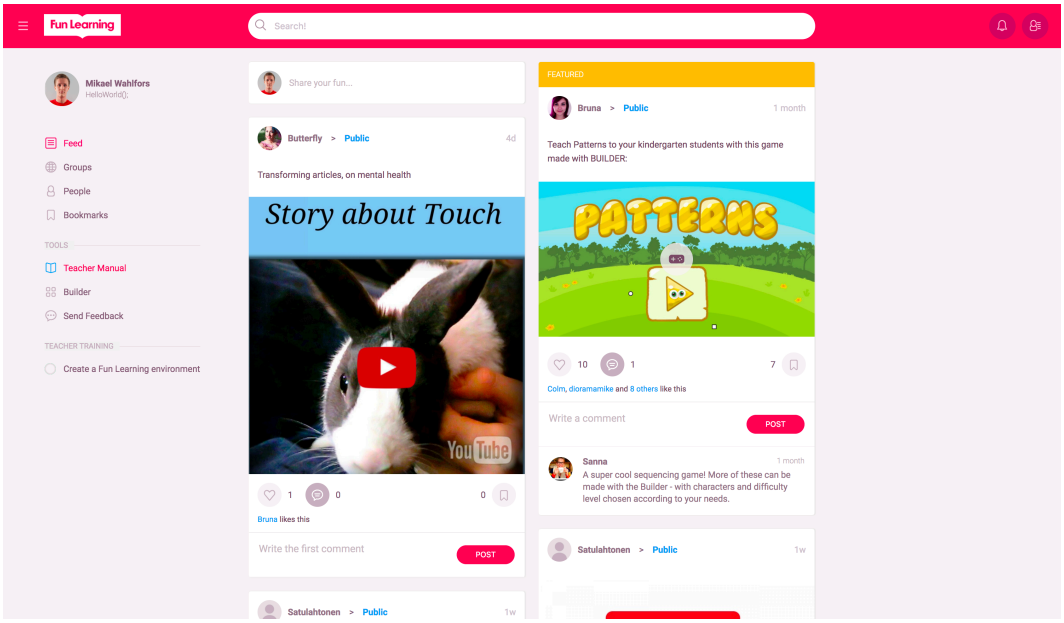
Seuraavaksi luotiin palvelimella mahdollisuus jonon käyttöön. Koska Lumen tukee jonoja suoraan asennuksesta lähtien, tarvittiin vain vaihtaa Lumenin asetuksia käyttämään Redis -välimuistia relaatiotietokannan sijaan. Kun asetukset oli vaihdettu, voitiin alkaa rakentamaan Job –luokkia eli työ luokkia, joiden tarkoitus oli toimia ajastettuina koodilohkoina, joita voitiin lisätä ja poistaa jonosta. Luokkiin pystyi määrittää asetuksia, kuten monta kertaa sitä yritetään suorittaa, kunnes työ siirretään epäonnistuneiden joukkoon. Job –luokkien ohjelmoinin jälkeen niitä pystyi käyttämään rajapinnassa lisäämällä niitä jonoon jonon push tai dispatch –funktioita käyttäen (kuvio 25). Kun työ asetetaan jonoon, voidaan työlle määrittää erillinen jono jos pääjonoa ei haluta käyttää tai lisätä viivettä. Uudessa API -rajapinnassa on käytössä vain yksi jono, koska kehittäjät eivät nähneet tarvetta usemmalle. Palvelimelle asetettiin jonon kuuntelija komentorivin php artisan queue:work & -komentoa käyttäen. Kuuntelija on pakollinen, jotta jono prosessoi töitä.

8.4.8 Versio 1.0

Sivustosta luotiin toimiva kokonaisuus testattavaksi neljässä kuukaudessa, jossa oli kaikki vanhan sivuston toiminnallisuudet (kuvio 26). Tämän jälkeen kuitenkin haluttiin uuteen sivustoon uudet toiminnallisuudet.

Vaatimuksissa määriteltyjä uusia toiminnallisuuksia olivat: notifikaatiot, ajastetut sähköpostit, normaalit ryhmät ja opettajien koulutukseen tarkoitetut training –ryhmät. Uusien toiminnallisuuksien lisääminen vaati 43 uutta kutsua, 5 uutta mallia, 4 uutta työ luokkaa ja 5 uutta ohjainta. Uusien toiminnallisuuksien lisäämisessä meni aikaa kaksi kuukautta. Uusien toiminallisuuksien toteutuksen aikana, myös testejä ja dokumentaatiota laajennettiin kattamaan uudet toiminnallisuudet.

Jono pääsi myös toimimaan tuotannossa uusien toiminnallisuuksien jälkeen, kun rajapinta alkoi käyttää ajastettuja sähköposteja. Ajastetuilla sähköposteilla estettiin sähköpostiviestien massalähetys. Esimerkkinä tästä oli, jos monta käyttäjää lähetti liittymiskutsun ryhmään samanaikaisesti, ei palvelin lähettänyt jokaisesta kutsusta erillistä sähköpostiviestiä. Ensimmäisestä kutsusta määritellyn ajan jälkeen palvelin kokosi kaikki pyynnöt kokoelmaksi ja lähetti sen ryhmän omistajalle. Uusien toiminnallisuuksien lisääminen oli modulaariseen ohjelmistokehykseen vaivatonta ja migraatioprosessi valmistuikin ajallaan määräaikaan mennessä.



KUVIO 26. Uusi sivusto

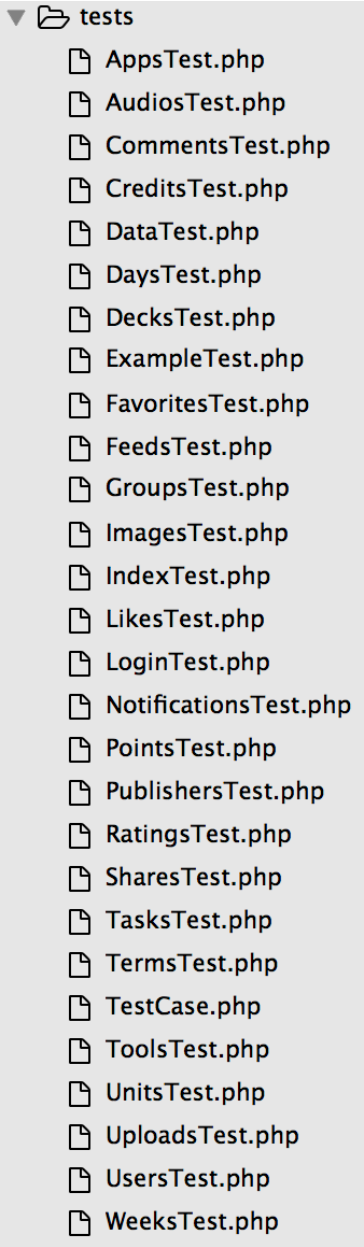
8.5 Yksikkötestit

Migraatioprosessin loppuvaiheessa rakennettiin kokoelma yksikkötestejä phpunit yksikkötesti ohjelmistokehystä käyttäen. Testaukset jaettiin kategorioihin, jotka määriteltiin ohjelmiston ohjaimien mukaan. Kategorioita oli yhteensä 26, joista 25 kappaletta oli ohjaintestejä ja yksi oli Tools luokan testiluokka. Jokainen ohjaimen metodi testattiin eri syötteitä ja kutsujia käyttäen (kuvio 27).

```
23      /**
24       * Test for Store function
25       * @return type
26       */
27      public function testStore()
28      {
29          //Generate temporary user
30          $user = factory('App\User')->create();
31          //Call user/credits/store route on API
32          $response = $this->actingAs($user)->call('POST', 'user/credits/store',['amount
                ' => 20]);
33          //Assert that correct amount is set for the user
34          $this->assertEquals(Credit::where('users_id',$user->id)->sum('amount'), 20);
35      }
36
```

KUVIO 27. user/credits/store kutsun testi

Koska ohjelmistossa oli käyttäjillä rooli, jossa roolin arvo määritti mitä kutsuja käyttäjä pystyi käyttäjä suorittamaan, piti tarkistaa, että validointi toimisi virhettömästi. Normaalia käyttäjää ei voida päästää suorittamaan ylläpidon kutsuja. Testejä oli yhteensä 94 ja ne koostuivat 3673 koodirivistä ja 2064 assertion todennuksesta (kuvio 28). Lumenissa on phpunitille suoraan tuki ja tästä johtuen testien luonnissa voidaan käyttää Database Transactions -rajapintaa, jonka avulla voitiin testit suorittaa ilman tietokannan commit komentoa. Tästä johtuen kaikki tietokanta muutokset, jotka testeissä tehdään eivät siirry sivuston tietokantaan. Testien avulla voitiin olla varmoja siitä, että rajapinta toimii testeillä osotetuissa ympäristössä uuden päivityksen julkaisuvaiheessa. Tämä nosti ohjelmiston laatua ja vähensi sivuston virhetiloja.



KUVIO 28. Luodut testit

9 YHTEENVETO

9.1 Projektista opittua

Projektin alussa huomasin, kuinka isosta projektista oikeasti oli kyse. Projektin aikana front end -ohjelmoija vaihtui suomalaisesta alihankkijasta Espanjassa sijaitsevaan työntekijään, joten sain kuvaa yhteistyöstä nykyaikaisessa yrityksessä internetin työkaluja hyväksikäyttäen. Kommunikaatio yrityksessä oli pääasiallisesti englanniksi, koska suurin osa työntekijöistä oli ulkomaalaisia. Samalla huomasin rajapintadokumentaation tärkeyden. Uusi työntekijä sisäisti rajapinnan toiminnallisuudet todella nopeasti, ja kysymyksiä oli todella vähän, kun dokumentaatio oli ajan tasalla.

Koska projektin aikamääreissä pysyttiin ja saavutettiin sille asetetut kriteerit, voidaan sanoa projektia onnistuneeksi. Projektista opittiin, että migraatioprosessi on mahdollista isollekin toteutukselle ja migraatioprosessi on kokonaisuutena suoraviivainen prosessi, jos ongelmat pilkkoo osiin. Lumen ohjelmistokehyksenä oli hyvä kohde, mutta se on uusi ja sen käyttäjäkunta on pieni. Tästä johtuen ongelmia kohdatessa on vaikea löytää vastauksia ongelmiin. Onneksi Lumen ongelmissa voi turvautua monessa asiassa Laravelin samankaltaisiin ongelman ratkaisuihin, koska arkkitehtuuri on lähes identtistä. Pienellä soveltamisella saadaan monimutkaisiin ongelmiin ratkaisuja.

Projektiin käytin yhteensä 420 tuntia, joka omasta mielestäni on paljon. Toisaalta projektin olisi voinnut valmistaa nopeammin, mutta laatu olisi kärsinyt. Projektin aikana koodia iteroitiin moneen kertaan, jotta modulaarisuus pysyisi korkeana. Lisäksi testien kirjoittamiseen käytettiin paljon aikaa, jotta niiden laatu pysyisi korkeana. Projektin aikana huomasin, kuinka helppoa on kirjoittaa testejä, jotka läpäisevät, joten jouduin pakottamaan itseni kirjoittamaan testejä tarkoituksella rikkoa ohjelma. Kommentoinnin tärkeys korostui projektin loppua kohden, kun koodia alkoi olemaan yli kymmenen tuhatta riviä. Jos oli muistanut

kommentoida rivit kunnollisesti, pääsi muokkaamaan koodia hyvinkin nopasti silmäilyn alottamisesta.

9.2 Jatkokehitys ja versio 2.0

Sivuston kehittämistä jatketaan migraatioprosessin jälkeen hyvin aktiivisesti. Tavoitteena on poistaa vanha toteutus mahdollisimman nopeasti, mutta tämä saattaa viedä aikaa, koska vanhalla sivustolla on vielä muutama asia, joita käyttäjät kaipaavat front end –puolella. API -rajapinnasta halutaan poistaa jatkossa login -kutsut kokonaan ja siirtyä keskitettyyn kirjautumispalvelimeen, joka toteuttaisi OAUTH 2.0 -protokollan mukaisen tunnistautumisen. Lisäksi Builder halutaan irrottaa käyttämään omaa API -rajapintaansa.

Tietokannan rakennetta halutaan muuttaa, kun vanha API-rajapinta saadaan pois käytöstä. Ylimääräisiä sarakkeita tauluissa on monia, ja taulukot olisi hyvä hajoittaa pienempiin kokonaisuuksiin, jotta rakenne olisi selkeämpi. Nimeämisessä olisi myös parantamista. Esimerkiksi images -taulu kattaa kaikki palvelun mediat, vaikkei se olisikaan kuva. Osa sarakkeista voitaisiin uudistaa, koska tietokannassa on paljon tyyppejä ja statuksia, jotka ovat kokonaislukumuodossa. Uuden kehittäjän on vaikea pysyä mukana, mitä kokonaisluku tarkoittaa kussakin tilanteessa.

Osa ohjaimista olisi hyvä siivota ja kirjoittaa uusiksi, kun Builder tuotteen tuki poistuu API-rajapinnasta. Ohjaimissa on koodia, joka on ainoastaan siellä, koska ohjaimen on tuettava Buider -tuotetta. Scraper -funktion toiminnallisuutta olisi hyvä edistää. 1.0 -versiossa scraper tukee ainoastaan sivustoja, joissa on meta -merkintä nimeltään og:image.

Kaiken kaikkiaan Fun Learning Communityn migraatioprosessi oli haastava ja mielenkiintoinen opinnäytetyön aihe, jonka aikana opin paljon www-tuotannosta ja API -rajapinnoista. Koska tuote oli lähes koko kehityksen aijan kansainvälisessä käytössä, opin, kuinka tärkeää hyvä testaus on. Monelta ongelmalta vältyttiin, kun testit olivat kattavat ja hyvin tehty. Jos palvelu yleistyy, kuten arviot osoittavat, on mukava ajatella, että

oli osana rakentamassa jotain, joka edistää monien ihmisten oppimista opetuksen laadun parantamisen vuoksi.

LÄHTEET

ASPHeute 2004. Storing Passwords - done right! [viitattu 29.3.2017]. Saatavissa: http://www.aspheute.com/english/20040105.asp

Arjen Lenstra and Xiaoyun Wang and Benne de Weger 2005. Colliding X.509 Certificates [viitattu 29.3.2017]. Saatavissa: http://eprint.iacr.org/2005/067/20050506:143625

Breck Carter 1997. Intelligent Versus Surrogate Keys. [viitattu 29.3.2017]. Saatavissa: http://www.bcarter.com/intsurr1.htm

Dooli 2017. Tilastoja suomen Internet-verkosta [viitattu 29.3.2017]. Saatavissa: https://www.dooli.fi/tilastoja-suomen-internet-verkosta

Eric Rescorla 2000. HTTP Over TLS. [viitattu 29.3.2017]. Saatavissa: https://tools.ietf.org/html/rfc2818

IBM 2017 Types of APIs. [viitattu 29.3.2017]. Saatavissa: https://developer.ibm.com/apiconnect/documentation/api-101/types-apis/

Jiantao Pan 1999. Software Testing. [viitattu 29.3.2017]. Saatavissa: https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/

Koskimies, K. & Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Valikko-sarja. Helsinki: Talentum.

Laravel 2017. Laravel Ohjelmistokehys [viitattu 29.3.2017]. Saatavissa: https://laravel.com/

Lumen 2017a. Lumen Controller [viitattu 29.3.2017]. Saatavissa: https://lumen.laravel.com/docs/5.4/controllers

Lumen 2017b. Lumen Middleware [viitattu 29.3.2017]. Saatavissa: https://lumen.laravel.com/docs/5.2/middleware

Lumen 2017c. Lumen Migrations [viitattu 29.3.2017]. Saatavissa: https://laravel.com/docs/5.4/migrations

Lumen 2017d. Lumen Model [viitattu 29.3.2017]. Saatavissa: https://laravel.com/docs/5.4/eloquent

Lumen 2017e. Lumen Ohjelmistokehys [viitattu 29.3.2017]. Saatavissa: https://lumen.laravel.com/

Lumen 2017f. Lumen Routing [viitattu 29.3.2017]. Saatavissa: https://lumen.laravel.com/docs/5.4/routing

MIT Laboratory for Computer Science and RSA Data Security, Inc 1992. The MD5 Message-Digest Algorithm [viitattu 29.3.2017]. Saatavissa: https://tools.ietf.org/html/rfc1321

Maxoffsky 2013. History of Laravel PHP framework, Eloquence emerging. [viitattu 29.3.2017]. Saatavissa: https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/

Mohit Kumar Gupta, Vishal Verma, Megha Singh Verma 2013. In-Memory Database Systems - A Paradigm Shift. [viitattu 29.3.2017]. Saatavissa: https://arxiv.org/abs/1402.1258

MySQL 2017a. Customers. [viitattu 29.3.2017]. Saatavissa: https://www.mysql.com/customers/

MySQL 2017b. The Main Features of MySQL. [viitattu 29.3.2017]. Saatavissa: https://dev.mysql.com/doc/refman/5.7/en/features.html

MySQL 2017c. What is MYSQL? [viitattu 29.3.2017]. Saatavissa: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

Naik, Kshirasagar, Tripathy, Priyadarshi 2008. Software Testing and Quality Assurance : Theory and Practice. Hoboken, N.J. : Wiley-Spektrum.

Niels Provos and David Mazieres 1999. MD5 Collision Demo [viitattu 29.3.2017]. Saatavissa: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html

Peter Selinger 2006. MD5 Collision Demo [viitattu 29.3.2017]. Saatavissa: http://www.mscs.dal.ca/~selinger/md5collision/

Pluralsight 2015. What's the Difference Between the Front-End and Back-End? [viitattu 29.3.2017]. Saatavissa: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end

Rackspace 2017. Redis Benchmark & Rackspace Performance VMs. [viitattu 29.3.2017]. Saatavissa: https://developer.rackspace.com/blog/redis-benchmark-and-rackspace-performance-vms/

Reddy, Martin, API Design for C++. eBook 2011. Boston : Morgan Kaufmann. 2011, 1-2

Redis 2017. Clients. [viitattu 29.3.2017]. Saatavissa: https://redis.io/clients

Redis 2017. Introduction to Redis. [viitattu 29.3.2017]. Saatavissa: https://redis.io/topics/introduction

Rountree Derrick, Castrillo Ileana 2014. The Basics of Cloud Computing : Understanding the Fundamentals of Cloud Computing in Theory and Practice. Amsterdam : Syngress.

Roy Thomas Fielding, 2000. Representational State Transfer (REST). [viitattu 29.3.2017]. Saatavissa: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Roy Thomas Fielding 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content [viitattu 29.3.2017]. Saatavissa: https://tools.ietf.org/html/rfc7231

Schneier 2004. Cryptanalysis of MD5 and SHA: Time for a New Standard [viitattu 29.3.2017]. Saatavissa: https://www.schneier.com/essays/archives/2004/08/cryptanalysis_of_md5.html

Surveillance Self-Defense 2017. What Is Encryption? [viitattu 29.3.2017]. Saatavissa: https://ssd.eff.org/en/module/what-encryption

Tim Bray 2014. The JavaScript Object Notation (JSON) Data Interchange Format. [viitattu 29.3.2017]. Saatavissa: http://www.rfc-editor.org/rfc/rfc7159.txt

w3techs 2017. Usage of web servers for websites [viitattu 29.3.2017]. Saatavissa: https://w3techs.com/technologies/overview/web_server/all

LIITTEET

LIITE 1. HTTP-ilmoituskoodit

100 Continue

Palvelin pyytää asiakata jatkamaan hakupyyntöä. Asiakas lähetti osan hakupyynnöstä ja palvelin jäi odottamaan loppua. (IETF. 2014)

101 Switching Protocols

Palvelin hyväksyy selaimen tekemän ehdotuksen protokollan vaihtamisesta. Asiakas voi pyytää protokollanvaihtoa yhteyden aikana Upgrade-kentällä esimerkiksi käyttääkseen ominaisuuksia uudemmasta versiosta HTTP-protokollasta. (IETF. 2014)

201 Created

Hakupyyntö toteutettiin ja uusi dokumentti luotiin. Tätä koodia voidaan käyttää esimerkiksi POST-metodin yhdeydessä. (IETF. 2014)

202 Accepted

Hakupyyntö hyväksyttiin, mutta sitä ei ole vielä toteutettu. Palvelin voi jatkaa hakupyynnön toteuttamista asynkroonisesti, vaikka yhteys selaimeen katkeaakin. Tällaista ilmoitusta voidaan käyttää, jos palvelin vastaanottaa webin kautta työläitä töitä. (IETF. 2014)

203 Non-Authoritative Information

Entity-kentässä lähetettyä metatietoa ei voida varmistaa. Kentässä lähetetty tieto ei ole peräisin alkuperäisestä dokumentista, vaan on palvelimen paras veikkaus. (IETF. 2014)

204 No Content

Dokumentti oli tyhjä. Palvelin toteutti hakupyynnön, mutta palautettava dokumentti on tyhjä. (IETF. 2014)

205 Reset Content

Palvelin toteutti lomakkeesta koostetun hakupyynnön ja selaimen tulisi tyhjentää lomake. Tätä ilmoitusta voidaan käyttää, kun samalla lomakkeella lähetetään enemmän tietoa palvelimelle heti lähetyksen perään. (IETF. 2014)

206 Partial Content

Asiakas lähetti hakupyynnön GET-metodilla, muttei Range-kenttää. Palvelin lähettää osan dokumentista ja lähetetyn datan sijainnista tietoa dokumentin Content-Range-kentässä. (IETF. 2014)

300 Multiple Choices

Pyydetystä dokumentista on useita versioita. Dokumentista voi olla esimerkiksi eri kieliversioita, joista asiakas voi valita mieleisensä. (IETF. 2014)

301 Moved Permanently

Dokumentti muuttanut osoitetta vakituisesti. Dokumentin osoite on vaihtunut, esimerkiksi sivuston migraatioprosessin seurauksena. (IETF. 2014)

302 Found

Dokumentti muuttanut osoitetta väliaikaisesti. Tunnetaan ehkä paremmin nimellä Moved Temporarily. Dokumentin osoite on vaihtunut väliaikaisesti toiseen osoitteeseen. (IETF. 2014)

303 See Other

Hae dokumentti toisaalta. Tätä koodia käyteteään tilanteessa, jossa kutsu on suoritettu ja käyttäjän halutaan, jatkavan toisessa osoitteessa. (IETF. 2014)

304 Not Modified

Asiakas voi lähettää ehdollisen hakupyynnön GET-metodilla, jossa pyydetään dokumenttia vain, jos sitä on muutettu annetun päivämäärän

jälkeen. Virhekoodi 304 lähetetään mikäli dokumenttia ei ole muutettu parametrinä annetun päivämäärän jälkeen. (IETF. 2014)

305 Use Proxy

Haettu dokumentti täytyy hakea annetun välityspalvelimen kautta. Palvelin ilmoittaa käytettävän välityspalvelimen Location-tietueessa. (IETF. 2014)

306 Switch Proxy

Poistettu käytöstä ja varattu myöhempää käyttöä varten. Tämä ilmoituskoodi oli käytössä aiemmassa HTTP:n versiossa, mutta on sittemmin poistettu käytöstä. Alkuperäinen koodi jolla pyydettiin seuraavia kutsuja käyttämään määriteltyä välityspalvelita. (IETF. 2014)

307 Temporary Redirect

Väliaikainen uudelleenohjaus. Pyydetty dokumentti sijaitsee väliaikaisesti eri osoiteessa. (IETF. 2014)

400 Bad Request

Asiakas lähetti palvelimelle pyynnön, jota ei voitu ymmärtää epäkorrektin syntaksin vuoksi. Pyyntö ei ollut HTTP-protokollan mukainen eikä palvelin käsittele sitä. (IETF. 2014)

401 Unauthorized

Pääsy evätty. Dokumentin hakemiseksi vaaditaan kirjautuminen. Palvelin lähettää tämän ilmoituskoodin aina kun salasanasuojattua dokumenttia haetaan niin kauan, kunnes käyttäjä lähettää hyväksyttävän tunnus/salasana-yhdistelmän. (IETF. 2014)

402 Payment Required

Maksu vaaditaan. Tätä ilmoituskoodia ei käytetä, se on varattu myöhempää käyttöä varten. (IETF. 2014)

403 Forbidden

Pääsy estetty. Palvelin ymmärtää selaimen tekemän pyynnön, mutta kieltäytyy toteuttamasta sitä. Tätä koodia käytetään esimerkiksi kun tiedoston oikeudet eivät ole kunnossa tai pääsy dokumenttiin on implisiittisesti estetty. (IETF. 2014)

404 Not Found

Haettua dokumenttia ei löytynyt palvelimelta. Siirretyille dokumenteille tulisi käyttää ilmoituskoodeja 301 ja 302, kokonaan poistetuille ilmoituskoodia 410. (IETF. 2014)

405 Method Not Allowed

Valittu HTTP-metodi ei vastaa kohde palvelimen kutsun tyyppiä. Valittua HTTP-metodia ei voida käyttää tähän dokumenttiin. (IETF. 2014)

406 Not Acceptable

Dokumentti on tiedostotyyppiä, jota asiakas ei hyväksy. Asiakas lähettää pyynnön mukana HTTP_ACCEPT-tietueen, joka kertoo, mitä tiedostotyyppejä asiakas haluaa vastaanottaa. Mikäli tarjottu dokumentti on jotain muuta tiedostotyyppiä, palvelin palauttaa ilmoituskoodin 406. (IETF. 2014)

407 Proxy Authentication Required

Selaimen tulee kirjautua ensin välityspalvelimeen. (IETF. 2014)

408 Request Timeout

Palvelimen odotusaika umpeutui. Palvelin odottaa selaimelta pyyntöä yhteydenoton jälkeen tietyn ajan ennen kuin katkaisee siihen yhteyden. Tämän ajan umpeuduttua palvelin lähettää ilmoituskoodin 408 ja katkaisee yhteyden. (IETF. 2014)

409 Conflict

Palvelin ei voi toteuttaa pyyntöä dokumenttiristiriidan takia. Tällainen tilanne voi syntyä RFC 2616:n mukaan esimerkiksi silloin, kun PUT-

metodia käyttäen lähetetään tiedosto palvelimelle ja tiedostosta on olemassa uudempi versio palvelimella. (IETF. 2014)

410 Gone

Haettua dokumenttia ei enää löydy palvelimelta. Koodia voidaan käyttää virhekoodin 404 asemesta sellaisen tilanteen ilmoittamiseen, että dokumentti on joskus ollut siinä osoitteessa, mutta on tarkoituksenmukaisesti poistettu ja että kaikki viittaukset (linkit) tähän dokumenttiin pyydetään poistamaan. (IETF. 2014)

411 Length Required

Palvelin ei hyväksy hakupyyntöä ilman määriteltyä pituutta. Selaimen tulee lähettää hakupyynnön pituus Content-Length-kentässä. (IETF. 2014)

412 Precondition Failed

Selaimen hakupyynnössä lähettämä ehto ei täyttynyt. Asiakas voi liittää hakupyyntöön ehtoja esimerkiksi tukemistaan merkkikoodauksista ja kielistä. Mikäli jokin näistä ehdoista ei täyty, palauttaa palvelin tämän ilmoituskoodin. (IETF. 2014)

413 Request Entity Too Large

*Ha*kupyyntö on liian suuri. Selaimen lähettämä hakupyyntö ylitti palvelimelle asetetun hakupyynnön koon ylärajan. Palvelin voi sulkea yhteyden lähetettyään ilmoituskoodin estääkseen asiakasta jatkamasta hakupyynnön lähettämistä. (IETF. 2014)

414 Request-URI Too Long

*O*soite on liian pitkä. Vaikka URI:n pituutta ei ole rajattu, voi palvelin asettaa rajoituksia esimerkiksi loputtomien silmukoiden tai ohjausten välttämiseksi. (IETF. 2014)

415 Unsupported Media Type

Asiakas lähetti hakupyynnön, joka oli tuntematonta mediatyyppiä. Palvelin ei ymmärrä käytettyä mediatyyppiä eikä näin ollen kykene suorittamaan pyyntöä. (IETF. 2014)

416 Requested Range Not Satisfiable

Selaimen hakupyynnössä lähettämä ehto dokumentin koosta ei täyttynyt. Asiakas voi lähettää hakupyynnössä Range request-header-kentällä, jolloin palvelin lähettää dokumentin vain, jos sen koko on määritellyn arvoalueen (range) sisällä. (IETF. 2014)

417 Expectation Failed

Selaimen hakupyynnössä lähettämä ehdoton ehto ei täyttynyt. Asiakas asetti ehdon ehdottomaksi Expect request-header-kentällä. (IETF. 2014)

500 Internal Server Error

501 Not Implemented

Hakupyyntöä ei voida suorittaa, koska tarvittavaa tekniikkaa ei ole toteutettu. Palvelin ei tue hakupyynnön edellyttämää toiminnallisuutta, esimerkiksi palvelin ei tue hakupyynnössä käytettyä HTTP-metodia. (IETF. 2014)

502 Bad Gateway

Palvelin sai väärän muotoisen vastauksen yrittäessään olla yhteydessä edelleen seuraavaan palvelimeen. Kyseinen palvelin toimi tällöin usein reitittimenä tai välimuistina eikä ymmärrä seuraavan palvelimen lähettämää dataa. (IETF. 2014)

503 Service Unavailable

Palvelimella ruuhkaa tai huoltotöitä. Tilanne, jossa palvelimen toiminnot ovat väliaikaisesti poissa käytöstä. Mikäli esimerkiksi huoltotyön pituus tiedetään, voidaan määritellä Retry-After-kenttä, jolloin asiakas tietää milloin kannattaa yrittää uudelleen. (IETF. 2014)

504 Gateway Timeout

Palvelin ei saanut ajoissa vastausta yrittäessään olla yhteydessä edelleen seuraavaan palvelimeen. Kyseinen palvelin toimi tällöin reitittimenä tai välimuistina ja odotti määrätyn ajan vastausta seuraavalta palvelimelta ja sulki sitten yhteyden. (IETF. 2014)

505 HTTP Version Not Supported

Palvelin ei tue käytettyä HTTP:n versiota. Asiakas lähetti hakupyynnön HTTP:n versiolla, jota palvelin ei tue tai halua tukea. (IETF. 2014)