Opinnäytetyö (AMK)

Tietotekniikka

Sulautetut ohjelmistot

2018

Pasi Vuohijoki

REST-RAJAPINNAN SUUNNITTELU JA TOTEUTUS

-Case: FirstView MediaCloud

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Sulautetut ohjelmistot

2018 | 30 sivua

Pasi Vuohijoki

REST-RAJAPINNAN SUUNNITTELU JA TOTEUTUS

Case: FirstView MediaCloud

Tässä opinnäytetyössä tarkastellaan FirstView MediaCloud digital signage -pilvipalvelun REST-rajapinnan suunnittelua ja toteutusta. Tavoitteena on suunnitella ja luoda helppokäyttöinen ja tietoturvallinen rajapinta FirstView MediaCloud -palvelun ydintoiminnoille.

Projektissa käytettynä aineistona oli käytettyjen sovelluskehysten ja kirjastojen tekniset dokumentaatiot. Perehtyminen perustekniikoihin ja ohjelmointikieliin oli tässä tapauksessa tarpeetonta, koska projektiin vaadittava tietotaito oli jo olemassa. Opinnäytetyön metodi on toiminnallinen, First Technology Oy:n tilaama FirstView Manager -pilvipalveluun perustuva sovellus, joka on toiminut lähes sellaisenaan vuodesta 2009. Opinnäytetyössä toteutettu FirstView MediaCloud on FirstView Manageriin pohjautuva, uudelleenkirjoitettu ja nykypäivän standardeja vastaava pilvipalvelun taustajärjestelmä.

Opinnäytetyössä tutkittiin REST-rajapinnan hyötyjä ja perustellaan miksi nykypäivän www-sovelluksen taustajärjestelmä on hyvä ohjelmoida rajapinnaksi. Työssä käydään myös läpi ohjelmistokehityksen eri vaiheita suunnittelusta toteutukseen, sekä pohditaan projektinhallintametodeja ja niiden tärkeyttä onnistuneen projektin läpiviemiseksi.

Opinnäytetyönä tehty FirstView MediaCloud -pilvipalvelun taustajärjestelmä on käytettävissä oleva sovellus, joka voidaan jatkokehittää valmiiksi digital signage -pilvipalvelutuotteeksi ja myytäväksi palveluksi. Työ saavutti sille asetetut tavoitteet korkeatasoisesta nykypäivän www-sovelluksesta.

ASIASANAT:

REST-rajapinta, Laravel Framework, digital signage

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information technology | Embedded systems

2018 | 30 pages

Pasi Vuohijoki

DESIGNING AND BUILDING A REST API

Case: FirstView MediaCloud

This thesis studies the designing and building of a REST API for a FirstView MediaCloud digital signage cloud computing service. The goal was to design and build an easy-to-use and information secure interface for FirstView MediaCloud core functionalities.

The thesis is divided into three sections from which the first one studies the designing process. The first section justifies why the chosen frameworks and tools are selected and explains how the interface should be divided in smaller sections to help keeping the code simpler and easier to read.

The second section of thesis describes the building process and explains the logic of the chosen framework. The framework data flow from http-request to response is explained with the help of examples.

The third section studies the productization and release process of the API. It defines how the project is usable by product owners and customer needs of the product owner. This section also gives examples of usage of the API.

The thesis was commissioned by First Technology Ltd, whose core business is producing digital signage service. The FirstView Manager cloud service was built in 2009 and has worked nearly unchanged since then. The FirstView MediaCloud built in this thesis is based on FirstView Manager and is a rewritten cloud computing service that meets modern day needs.

KEYWORDS:

REST-API, Laravel Framework, digital signage

SISÄLTÖ

KUVAT

KÄYTETYT LYHENTEET JA SANASTO

digital signage	Digitaalisilla näytöillä tapahtuvaa markkinointi- ja informaatioviestintää.
http-protokolla	Standardisoitu tiedonsiirtoprotokolla www-ympäristössä.
ohjain	(engl.controller) luokka, joka käsittelee http-kutsussa tulleet parametrit ja palauttaa kutsuun vastauksen.
PHP	(Perl Hypertext Preprocessor) ohjelmointikieli dynaamisen www-sisällön ohjelmointiin.
päätepiste	(engl.endpoint) rajapinnasta kutsuttava tietue jolla on oma osoite, jota kutsutaan http-protokollan mukaisella kutsulla.
reitti	(engl.route) reitti ohjaa syötetyn URL osoitteen haluttuun ohjaimeen.
REST-rajapinta	(engl.representative state transfer) arkkitehtuurimalli, joka perustuu http-Protokollaan. (Fielding 2000, 76)
sovelluskehys	(engl.framework) joukko valmiita työkaluja, luokkia ja funktioita sovelluskehityksen helpottamista varten.

1 JOHDANTO

Työn tilaajana toiminut FirstView (First Technology Oy) on Turkulainen yritys, jonka ydintoimiala on digital signage-palveluiden tuottaminen, kehittäminen ja ylläpito. Nykyinen käytössä oleva järjestelmä on toiminut samalla alustalla lähes kymmenen vuotta ja palvelee yli 11000:tta näyttöä ympäri maailman. (FirstView Digital Signage) Opinnäytetyön aiheena oleva FirstView MediaCloud -projekti sai alkunsa asiakkaan yhteyshenkilön kehitysideasta FirstView Managerin käyttämisestä REST-rajapinnan yli. Asiakkaan toiveena oli saada edustamansa yrityksen sisällönhallintajärjestelmästä ohjattua myös digital signage -näyttöjä ilman, että hänen olisi pitänyt kirjautua FirstView Manageriin eri tunnuksilla. Tämä asiakastoive tuki myös tarvetta vanhan FirstView Manager'in uudelleenkirjoittamisesta.

Projektissa käytettäviä tekniikoita rajattiin niin, että vanhaa FirstView Manager -järjestelmään PHP5.6-kielellä kirjoitettua ohjelmakoodia saatiin käytettyä hyväksi. Näin ollen päädyttiin käyttämään PHP7-ohjelmointikieltä, joka on hyvin taaksepäin yhteensopivaa PHP5.6-kielen kanssa.

Hyvin varhaisessa suunnitteluvaiheessa päätettiin käyttää jotakin lukuisista sovelluskehyksistä, mikä helpottaa erittäin paljon koodin kirjoittamista. Sovelluskehyksen käyttö säästää aikaa varsinaisten toiminnallisuuksien kirjoittamiselle sen sijaan, että kaikki taustalla tapahtuva, esimerkiksi tietokantakyselyjen ja JSON-vastauksien muotoilu, pitäisi alusta asti kirjoittaa itse.

Lukuisia PHP:llä kirjoitettuja sovelluskehyksiä tutkittiin ja lopulta päädyttiin Laravel Framework -nimiseen sovelluskehykseen. Tässä vaihtoehdossa koettiin olevan kattavimmat työkalut kyselyjen sujuvaan reititykseen, tietokantakyselyihin sekä tietoturvan varmistamiseen. Laravel framework mahdollisti myös yksikkötestauksen helposti, joka tekee koodista entistä laadukkaampaa ja varmistaa, etteivät tehdyt muutokset riko aikaisemmin kirjoitettua koodia.

Lähteenä opinnäytetyössä käytettiin pääosin Laravel'n omaa dokumentaatiota, jossa hyvin esimerkein havainnollistetaan eri osa-alueiden ja komponenttien käyttöä. Laravel'n käyttöönottokynnys olikin yllättävän pieni juuri hyvän dokumentaation ja aiemman PHP-osaamisen vuoksi.

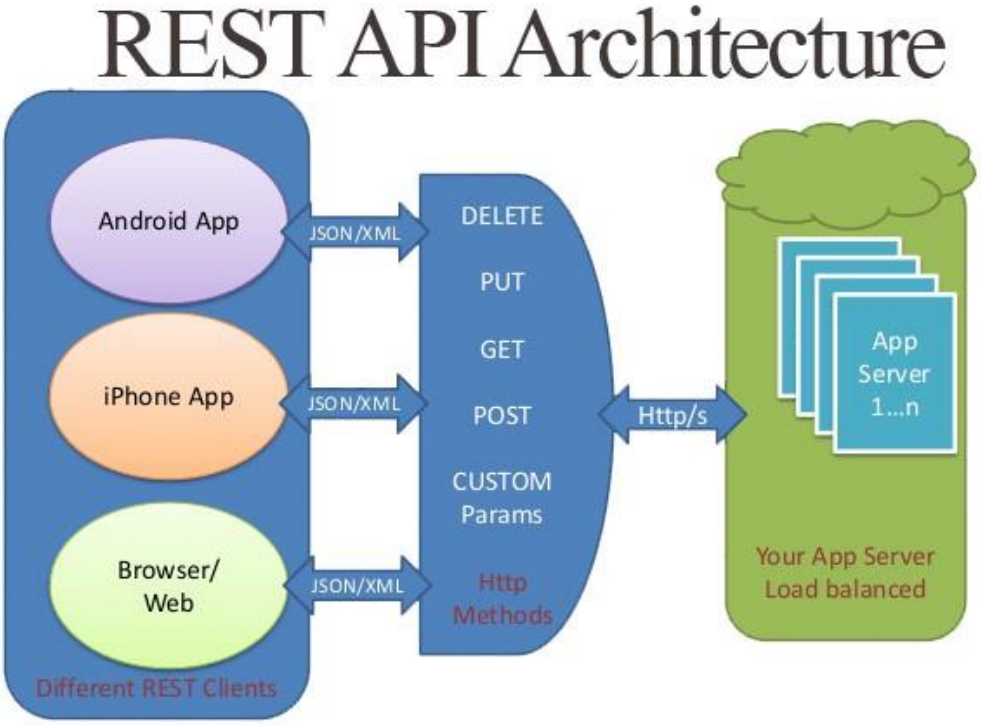
Opinnäytetyö jakautuu karkeasti kolmeen osaan, joista ensimmäinen osa käsittelee rajapinnan suunnittelutyötä. Ensimmäisessä osassa perustellaan esimerkiksi eri työkalujen ja ohjelmistokehysten valintoja, sekä pohditaan millaisiin osiin rajapinta kannattaa jakaa.

Toisessa osassa paneudutaan toteutusvaiheeseen ja ohjelmistokehysten logiikkaan ja käyttöön. Esimerkkien avulla käydään läpi ohjelmistokehyksen tiedonkulku http-kutsusta tiedon palauttamiseen saakka.

Kolmas osa käsittelee rajapinnan tuotteistamista ja julkaisua sekä tilaajan omaan, että tilaajan asiakkaiden käyttöön. Tässä osassa pureudutaan myös valmiin rajapinnan käyttöön käyttöliittymästä poimittujen esimerkkien avulla.

2 REST-RAJAPINNAN SUUNNITTELU

REST eli representative state transfer on arkkitehtuurimalli, joka perustuu HTTP-Protokollaan. (Fielding 2000, 76) REST-rajapinta mahdollistaa taustajärjestelmän ja käyttöliittymän kehittämisen eriyttämisen toisistaan, koska kaikki tiedonsiirto tietokannasta käyttäjän päätelaitteeseen kulkee REST-rajapinnan yli. (Kuva 1) REST-arkkitehtuurimalli on syrjäyttänyt monet SOAP ja WSDL -malliset www-palvelut ja muun muassa Google, Facebook ja Yahoo käyttävät rajapinnoissaan pelkästään REST-mallia. (Rodriguez 2015, 1)



Kuva 1. Esimerkki REST-rajapinnan arkkitehtuurista (James, G. 2017)

2.1 Sovelluskehyksen valinta

Koska käytössä ollut, alkuperäinen FirstView Manager oli kirjoitettu PHP5.6 -ohjelmointikielellä, päätettiin rajoittaa sovelluskehyksen valinnan PHP7-kielellä kirjoitettuihin sovelluskehyksiin. Haluttiin myös, että sovelluskehys olisi lisensoitu avoimen yhteisön lisenssillä kustannustehokkuuden ja riippumattomuuden vuoksi. Käytiin läpi tarjolla olevia vaihtoehtoja ja tehtiin lopulliset valinnat kolmesta hyväksi havaitusta vaihtoehdosta. Kolme tutkittua vaihtoehtoa olivat Slim 3, Symfony ja Laravel. Lopulta päädyttiin käyttämään Laravel'ia.

Slim 3 on erittäin kevyt REST-rajapintojen luomiseen käytettävä microframework, jonka etuna on erittäin pieni konfiguroinnin tarve ohjelmointityön aloittamista varten. Koettiin kuitenkin, ettei Slim 3 tarjonnut tarpeeksi valmiita työkaluja ja mahdollisuuksia näin mittavan projektin tarpeisiin.

Symfony on erittäin suosittu ja käytetty sovelluskehys PHP-pohjaisten www-sovellusten tuottamiseen. Symfonyn suurin etu on sen suuri suosio ja käyttöaste, joka takaa, että sovelluskehys pysyy ajantasaisena ja avoimen yhteisön tuki on erittäin laaja. Symfony ei kuitenkaan sellaisenaan tarjoa kovin paljon valmiita työkaluja varsinaisten toiminnallisuuksien ohjelmoimiseksi.

Valittu Laravel on ohjelmoitu Symfonyn päälle ja ydintoiminnot käyttäytyvät samalla lailla kuin Symfonyssä. Laravel'ssa etuna on erittäin laaja kirjasto työkaluja esimerkiksi tietokantakyselyjen tekemiseen ja tiedonkulun reitittämiseen. Laravel pitää sisällään myös valmiit autentikointi ja autorisointityökalut, joiden käyttöönotto sujui erittäin vaivattomasti. Symfony -pohja takaa myös tuen riittävän pitkälle tulevaisuuteen.

2.2 Päätepisteiden suunnittelu

Päätepisteiden suunnittelu tehtiin tarkasti REST-arkkitehtuurimallin mukaisiksi, jolloin ohjelmakoodi pysyy sopivan pieninä paloina ja jokainen päätepiste tekee vain yhteen resurssiin liittyviä toimenpiteitä.

2.2.1 REST arkkitehtuurimallin mukaiset päätepisteet

Päätepiste tarkoittaa yhtä mikropalvelua rajapinnassa. Esimerkiksi käyttäjän luomiseen tarvitaan oma päätepiste, jota kutsutaan tietyllä metodilla ja tarvittavilla parametreillä, jolloin Laravel'iin kirjoitettu ohjain hoitaa tiedon lisäämisen tietokantaan ja muotoilee vastauksen REST-mallin mukaiseen JSON-muotoon, joka ilmoittaa asiakasohjelmalle käyttäjän luomisen onnistumisesta tai epäonnistumisesta.

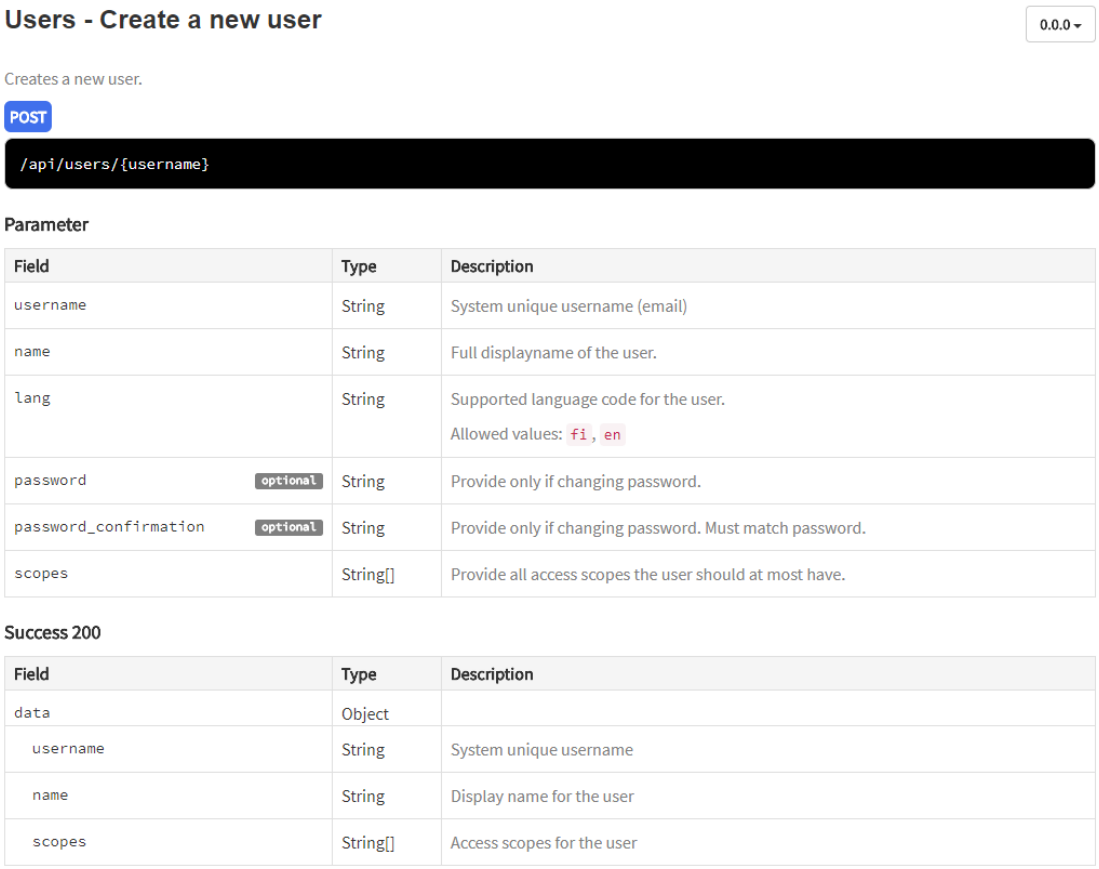
REST-arkkitehtuurimalli noudattaa tarkasti http-protokollaa, jolloin kutsussa on käytettävissä neljä eri metodia, GET, POST, PUT ja DELETE. Resurssin luontiin käytetään POST-metodia, noutamiseen GET-metodia, päivittämiseen PUT-metodia ja poistamiseen DELETE-metodia. (Rodriguez 2015, 2)

Http-protokollan mukaisesti palautukseen sisältyy aina protokollan mukainen statuskoodi, joka kertoo asiakasohjelmalle pyynnön tilan. Jos pyyntö onnistui, palautetaan *"200 OK"* tai jos pyyntö onnistui ja uusi resurssi luotiin, palautetaan "*201 Created*". (W3)

2.2.2 FirstView Managerin toiminnallisuuksien suunnittelu REST-päätepisteiksi

Päätepisteiden suunnittelu asetti haasteita, koska vanha järjestelmä on ohjelmoitu hyvin pitkälle käyttöliittymä ja taustajärjestelmä yhteen nivoutuneena. Tällöin taustajärjestelmän tekemät toimenpiteet piti vanhasta koodista tunnistaa ja sille tehdä oma päätepiste REST-rajapintaan. Tässä piti kuitenkin muistaa REST-arkkitehtuurin periaate, jossa yksi resurssi ja siihen liittyvät toimenpiteet ovat aina omassa päätepisteessään. Esimerkiksi käyttäjien hallinnalle tehtiin oma juuripäätepiste polkuun "*/api/users".*

Tämä päätepiste käyttäytyy siis eri lailla riippuen kutsun metodista. Esimerkiksi käyttäjän luomiseen päätepistettä kutsutaan POST-metodilla (Kuva 2). Kutsun otsakkeessa parametreinä menevät tarvittavat käyttäjätiedot ja autentikointiavain.



Users - Create a new user

0.0.0

Creates a new user.

POST

```
/api/users/{username}
```

Parameter

Field	Type	Description
username	String	System unique username (email)
name	String	Full displayname of the user.
lang	String	Supported language code for the user. Allowed values: `fi`, `en`
password optional	String	Provide only if changing password.
password_confirmation optional	String	Provide only if changing password. Must match password.
scopes	String[]	Provide all access scopes the user should at most have.

Success 200

Field	Type	Description
data	Object	
username	String	System unique username
name	String	Display name for the user
scopes	String[]	Access scopes for the user

Kuva 2. POST api/users päätepisteen apidoc -dokumentaatio

Mikäli käyttäjä halutaan poistaa, tulee REST-arkkitehtuurin mukaisesti päätepisteen perään poistettavan resurssin yksilöllinen id-numero ja kutsu tehdään DELETE-metodia käyttäen.

Samalla logiikalla suunniteltiin myös muut palvelun käyttämiseen ja ylläpitoon tarvittavat päätepisteet. Esimerkiksi asiakasyksikön lisäämiseeen tehtiin ”*/api/zones/*” -päätepiste.

2.2.3 Päätepisteiden dokumentointi

Alusta asti oli selvää, että koko rajapinta pitää kuvata erittäin tarkasti, koska tavoitteena on antaa rajapintakuvaus asiakkaiden käyttöön, jolloin he voivat ottaa FirstView’n ominaisuuksia ja toimintoja käyttöön omissa sovelluksissaan. Esimerkiksi yksi asiakkaista on digi- sekä printtipaino, jolla on oma sisällönhallintajärjestelmä, josta heidän loppuasiakas voi yhdestä paikasta tilata haluamansa kuvan printtitulosteena, ohjevihkona ja

lisätä kuvan omalle www-sivulleen. FirstView MediaCloud -REST rajapinnan avulla kuva voidaan samalla lisätä myös asiakkaan digital signage -näytölle.

Rajapinnan dokumentointiin päädyttiin käyttämään APIDOC-työkalua. Apidoc mahdollistaa rajapintakuvauksen kirjoittamisen suoraan ohjelmakoodiin kommenttina. Palvelimella ajettava skripti kääntää nämä kommentit halutun teeman ja ulkoasumäärittelyn mukaiseksi HTML5 -dokumentaatiosivuksi. Usein päätepisteiden suunnittelu alkoi apidoc-määrittelyllä, jolloin oli helppo hahmottaa, mitä ohjelman pitää tehdä dokumentin mukaisen lopputuloksen saavuttamiseksi. (Kuva 3)

```
/**
 * @api             {post} /api/users/{username} Create a new user
 * @apiDescription  Creates a new user.
 * @apiName         CreateUser
 * @apiGroup        Users
 *
 * @apiParam {String} username System unique username (email)
 * @apiParam {String} name Full displayname of the user.
 * @apiParam {String=fi,en} lang Supported language code for the user.
 * @apiParam {String} [password] Provide only if changing password.
 * @apiParam {String} [password_confirmation] Provide only if changing password. Must match password.
 * @apiParam {String[]} scopes Provide all access scopes the user should at most have.
 * @apiUse UserResponse
 *
 * @param NewUserRequest $Request
 *
 * @return JsonResponse
 */
public function newUser(NewUserRequest $Request) {
    //TODO: USER MANAGER - only allow if current user is allowed to create users

    $User = new User();
    $User->username = $Request->username;
    $User->name = $Request->name;
    $User->lang = $Request->lang;
    $User->password = $Request->password;
    $User->verified = 1;
```

Kuva 3. Esimerkki apidoc -määrittelystä ohjelmakoodin kommenteissa

Sovelluskehitysprosessin suunnittelu

Jotta FirstView MediaCloudin kehitys sujuisi mutkattomasti, oli tarpeen luoda etukäteen suunnitelma kehitysprosessin vaiheista ja versiohallinnan käytöstä. Laravel sovelluskehys ja FirstView MediaCloud -projekti alustettiin Ubuntu Linux 16.04 -käyttöjärjestelmällä toimivaan kehityspalvelinkoneeseen. Palvelimelle asennettiin nginx www-palvelinohjelma, joka mahdollistaa PHP7-ohjelmien ajamisen ja REST-apin osoiterakenteen purkamisen ja uudelleenohjaamisen. Kehityspalvelimella on jokaiselle kehittäjälle oma kansio, johon jokainen alusti oman asennuksena MediaCloudista GIT-versionhallinnasta projektin kloonaamalla.

2.2.4 Versiohallinnan käyttöönotto

Git'n avulla pystyttiin kehittämään ennalta sovitusti tiettyjä osa-alueita ja päätepisteitä omatoimisesti vaikuttamatta muiden tekemään työhön ja ilman pelkoa hallitsemattomista konflikteista, mikäli samaa tiedostoa muokataan. Git'iin alustetaan aina uusi haara uutta ominaisuutta tai korjausta varten ja lopuksi tämä haara yhdistetään pääkehityshaaraan, jolloin git pitää huolen, että mahdolliset konfliktit ratkotaan oikein ja päähaara pysyy aina ajan tasalla.

Aluksi luotiin tyhjä git-projekti erilliselle palvelimelle komennolla ”*git init --bare fview2-api.git”*. Tämä loi tyhjän projektin nimeltä fview2-api. Optiolla ”*--bare”* -luotiin bare-repositio, eli pelkän koodin ja kehityshaarat taltioiva repositio, mistä varsinainen ohjelma ei ole käytettävissä. Tämän jälkeen git repositio kloonattiin omaan kansioon tuotekehitys-palvelimelle.

2.2.5 Laravel'n asentaminen

Laravel'n asennus sujui helposti hyvien dokumentaatioiden avulla. Laravel sovelluskehykseen kuuluu artisan-komentotulkki, jonka avulla voidaan ajaa erilaisia komentoja esimerkiksi projektien tai projektin sisäisten komponenttien luomiseksi. Laravel sovelluskehys asennettiin PHP:n omalla paketinhallintaohjelmalla, Composerilla. Komennolla “*composer create-project --prefer-dist laravel/laravel fview2-api”* alustettiin uusi projekti nimeltään fview2-api. Komento pitää huolta kaikkien tarvittavien riippuvuuksien asentamisesta ja kansiorakenteen luomisesta. Kun Laravel projekti oli alustettu, siitä luotiin ensimmäinen commit git'n master-haaraan, jonka päälle itse kehitystyö oli helppo aloittaa. (Otwell, T.)

Laravel'in konfigurointi on myös erittäin yksinkertaista. Tarvittavat parametrit, kuten esimerkiksi tietokannan isäntäpalvelin, käyttäjänimi ja salasana tulevat .env -konfiguraatiotiedostoon nimi-arvo -pareina. (Otwell, T.) (Kuva 1)

```
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=https://xxx.firsttechnology.fi
APP_FVIEW1_URL=https://yyy.firsttechnology.fi
APP_FVIEW1_FONT_PATH=../../fview1assets/fonts
APP_FVIEW1_LAYOUT_PATH=../../fview1assets/layouts
APP_FVIEW1_LAYOUT=first
APP_STORAGE_BASE=/CloudFS/
APP_STORAGE_PATH=fview_xxx

#DEFAULT DB: FView1
DB_CONNECTION=default
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=
DB_USERNAME=root
DB_PASSWORD=

#AMAZON SDK
AMAZON_KEY=
AMAZON_SECRET=

#VTIGER API CREDENTIALS
V_TIGER_USERNAME=ftapi@firstview.fi
V_TIGER_ACCESSKEY=
V_TIGER_URL=https://firsttechnology1.od1.vtiger.com/webservice.php

#AMAZON VIDEO CONVERTER
VIDEO_CONVERSION_INPUT_BUCKET="ft-pi-input-video"
VIDEO_CONVERSION_OUTPUT_BUCKET="ft-pi-output-video"
VIDEO_CONVERSION_THUMBNAIL_BUCKET="ft-pi-output-video-thumbnail"
VIDEO_CONVERSION_PIPELINE="ft-pi-pipeline"

BROADCAST_DRIVER=log
CACHE_DRIVER=database
SESSION_DRIVER=file
QUEUE_DRIVER=database

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

PUSHER_APP_ID=
PUSHER_KEY=
PUSHER_SECRET=

```

Kuva 4. Esimerkki .env.example-tiedostosta

2.2.6 Scrum-mallin mukainen tuotekehitys

Tuotekehitysprojektia päätettiin lähteä työstämään hyväksi havaitun Scrum-mallin mukaisesti, jolloin tuotekehitys pysyy ketteränä ja saatiin reagoitua tehokkaasti ilmeneviin haasteisiin ja mahdollisuuksiin. Scrumin idea on tehdä lyhyitä, noin 4 viikkoa kestäviä sprinttejä, joiden jälkeen palaverissa puidaan menneen sprintin aikaansaannokset ja suunnitellaan seuraava sprint. (Opelt ym. 2013, 3) Projektinhallinnan tukena käytettiin Trello -projektinhallintasovellusta, jonka avulla on visuaalisesti helppo nähdä projektin tila kunkin sprintin aikana.

Trelloon laitettiin scrum-mallin mukaisesti lista tarvittavista ominaisuuksista (back log) ja listalta poimittiin aina jokaiseen sprinttiin sopiva määrä ominaisuuksia. Back logiin kirjattavat tehtävät jaettiin karkeasti tarvittavien rajapintapäätepisteiden mukaan.

3 REST-APIN TOTEUTUS

3.1 Tietokannan migraatio

Koska projektin tarkoitus on olla jossain määrin yhteensopiva vanhan järjestelmän kanssa, päätettiin käyttää vanhaa tietokantarakennetta ja suorittaa migraatio vanhoista asennuksista uuteen järjestelmään yhteensopivaksi. Tietokantamoottorina käytettiin MariaDB 5.5 -tietokantamoottoria. Tämä on täysin yhteensopiva vanhan järjestelmän tuotantotietokannoissa käytettyjen MySQL tietokantojen kanssa, jolloin migraatio on helpompi tehdä.

Tietokantarakenteen kuvaamiseen käytettiin Laravel'ssa mukana tulevaa Eloquent-kirjastoa, johon taulut kuvataan tietokantamalleina, modeleina. (Otwell, T.) Vanhan tietokannan tauluista mallinnetaan migraatiotiedosto, johon määritellään taulun kentät ja avaimet. Migraatiotiedoston saa luotua Artisan-komentotulkin avulla seuraavalla komennolla: *"php artisan make:migration create_<taulun_nimi>_table"*.

Kun kaikista tauluista on tehty migraatiotiedosto, voidaan migraatio ajaa komennolla *" php artisan migrate"*. Tämä luo tietokantarakenteen oikeanlaiseksi Laravel'iin konfiguroituun tietokantaan. (Otwell, T.)

3.2 Reititysten luonti ja tiedon kulku Laravel'ssa

Laravel sovelluskehys tarjoaa oletuksena kolmen erilaisen reittityypin (route) luomisen. Reittityypit ovat api, console ja web. (Otwell, T.) Koska laravel'ia käytetään tässä tapauksessa pelkästään REST-rajapinnan luomiseen, kaikki routet luotiin routes/api.php -tiedostoon. Palvelimella olevaan nginx -palvelinohjelmistoon on konfiguroitu kutsun url:n uudelleenreititys niin, että Laravel'n routet osaa kartoittaa kutsut oikein ja ohjata pyynnöt oikeille ohjaimille.

3.2.1 Reitin luonti

Reitin luonti aloitetaan luomalla reittiryhmä. Laravel'issa on reittien kirjoittamiseen valmis luokka, Route. Reittiryhmä luodaan kutsumalla Route luokan staattista funktiota *group()*.

Route::group() -funktion parametreiksi tulee joukko ryhmän asetuksia sekä funktio, joka kartoittaa kaikki ryhmän sisällä olevat uudelleenohjaukset ja käytettävät väliohjelmistot. Väliohjelma (engl.middleware) on reitille konfiguroitava lisätoimenpide tai funktio, joka injektoidaan toiminnallisuuden mukaan joko ennen tai jälkeen reitin ohjaimen suorittamista. Väliohjelma voi olla vastuussa esimerkiksi kutsun autorisoinnista tai käyttäjäoikeuksien tarkistuksesta. (Otwell, T.)

```
Route::group(
    [
        'namespace' => 'User',
        'prefix' => 'users',
        'middleware' => [
            'auth:api',
            'throttle:60,1',
            /*'scope:'.implode(',',[
                \App\AccessManager\UserScopeAccessManager::SCOPE_EDIT_USERS,
                \App\AccessManager\UserScopeAccessManager::SCOPE_CREATE_USERS,
                \App\AccessManager\UserScopeAccessManager::SCOPE_VIEW_USERS,
            ]),*/
        ],
        //'middleware' => 'auth:api'
        //'middleware' => 'requireAuthorization:media.add'
    ],
    function(){
        Route::get(
            '',
            'UserController@listUsers'
        )->middleware('scope:'.implode(',',[
                \App\AccessManager\UserScopeAccessManager::SCOPE_VIEW_USERS,
                // \App\AccessManager\UserScopeAccessManager::SCOPE_VIEW_MY_USER,
            ]));
        Route::get(
            '{username}',
            'UserController@getUser'
        )->middleware('scope:'.implode(',',[
                \App\AccessManager\UserScopeAccessManager::SCOPE_VIEW_USERS,
                \App\AccessManager\UserScopeAccessManager::SCOPE_VIEW_MY_USER,
            ]));
        Route::put(
            '{username}',
            'UserController@updateUser'
        )->middleware('scope:'.implode(',',[
                \App\AccessManager\UserScopeAccessManager::SCOPE_EDIT_USERS,
                \App\AccessManager\UserScopeAccessManager::SCOPE_EDIT_MY_USER,
            ]));
        Route::post(
            '',
            'UserController@newUser'
        )->middleware('scope:'.\App\AccessManager\UserScopeAccessManager::SCOPE_CREATE_USERS);
    }
);
```

Kuva 5. Esimerkki reittiryhmän luomisesta

Käyttäjähallinnan reittiin on ohjelmoitu reittiryhmän nimiavaruus *”User”* ja käytettävä url-etuliite *”users”*. Ryhmän yhteiset middlewaret on esitelty *”middleware”* joukossa. Käyttäjähallinnan kaikissa päätepisteissä käytetään *”auth:api”* ja ”*throttle:60,1”* middlewareja. Auth:api -middleware tarkistaa jokaisen kutsun yhteydessä, että käyttäjä on autentikoitunut palvelimelle. Throttle:60,1 -middleware rajoittaa kutsujen määrän 60 kutsuun 1 s:ssa. Kutsujen määrän rajoittamisella saavutetaan parempi suoja palvelunestohyökkäyksiä vastaan.

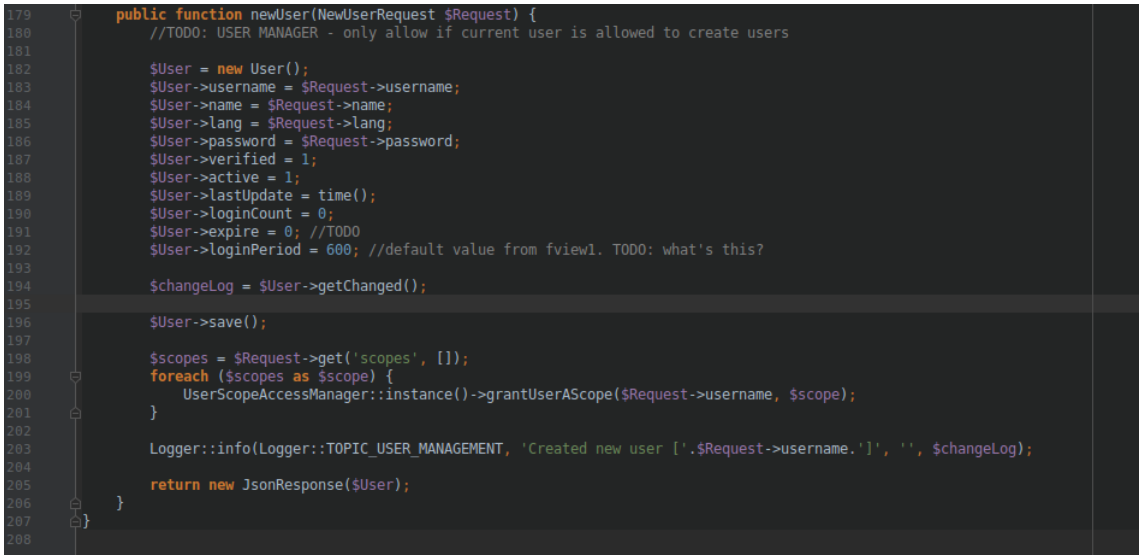
Ryhmän yhteisten asetusten jälkeen määritellään jokaisen päätepisteen uudelleenohjaus reitistä vastaavalle ohjainluokalle ja pyynnön käsittelevälle funktiolle. Jokaiselle http-metodille on olemassa oma staattinen funktio Route -luokassa, jonka avulla uudelleenkartoitus määritellään. Näiden avulla voidaan pyynnön metodia muuttamalla tehdä eri asioita http-protokollan asettaman logiikan mukaisesti. Esimerkiksi *”GET api/users/{id}”* palauttaa id:tä vastaavan käyttäjän tiedot ja *”DELETE api/users/{id}”* poistaa käyttäjän. Funktiossa määrittellään myös päätepistekohtaiset middlewaret. Esimerkiksi käyttäjähallinnan päätepisteissä voidaan tarkistaa, onko käyttäjällä oikeus lisätä uusia käyttäjiä, mikäli kutsutaan *”POST api/users”* päätepistettä.

3.2.2 Ohjaimen ja entiteetin luonti

Ohjain (engl.controller) vastaa reittiryhmän toiminnoista. Ohjain on aina reittiryhmäkohtainen ja sisältää funktiot jokaiselle ryhmän päätepisteelle. Esimerkiksi *”api/users”* -reittiryhmän toiminnallisuudesta vastaa Laravel’n Controller -luokasta periytetty UserController-luokka. Ohjaimia saa luotua kätevästi artisan -komentotulkin avulla komennolla *”php artisan make:controller <ohjainluokan nimi>”*. (Otwell, T.)

Reitti ohjaa esimerkiksi käyttäjän luonnin UserController-luokan newUser funktioon, jonka parametrina kulkee reitiltä tulevan pyynnön hyötykuorma. Periaatteessa ohjaimessa voisi jo luoda tietokantamallinnuksen uudesta käyttäjästä ja tallentaa se tietokantaan, mutta jokaisesta tietokantataulusta päätettiin luoda entiteettiluokat, jolloin ohjainluokat saatiin pidettyä siisteinä ja siellä tapahtuu vain tiedon kerääminen reitiltä ja palautuksen muotoilu JSON-vastaukseen.

Myös päätepistekohtainen dokumentointi on kommentoitu ohjainluokkaan vastaavan funktion yläpuolelle. Ohjainluokan kirjoittaminen on hyvä aloittaa dokumentaation kirjoittamisesta, jolloin kuva siitä, mitä kyseisessä ohjaimessa tehdään kussakin funktiossa, pysyy koko ajan selkeänä.



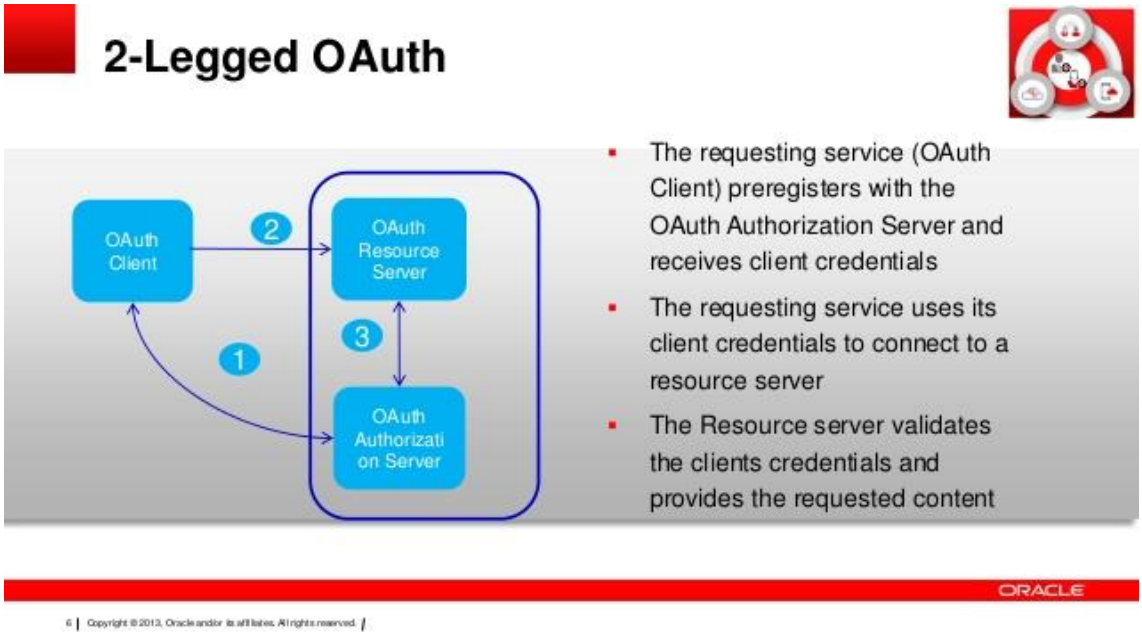
```
    public function newUser(NewUserRequest $Request) {
        //TODO: USER MANAGER - only allow if current user is allowed to create users

        $User = new User();
        $User->username = $Request->username;
        $User->name = $Request->name;
        $User->lang = $Request->lang;
        $User->password = $Request->password;
        $User->verified = 1;
        $User->active = 1;
        $User->lastUpdate = time();
        $User->loginCount = 0;
        $User->expire = 0; //TODO
        $User->loginPeriod = 600; //default value from fview1. TODO: what's this?

        $changeLog = $User->getChanged();

        $User->save();

        $scopes = $Request->get('scopes', []);
        foreach ($scopes as $scope) {
            UserScopeAccessManager::instance()->grantUserAScope($Request->username, $scope);
        }

        Logger::info(Logger::TOPIC_USER_MANAGEMENT, 'Created new user ['.$Request->username.']', '', $changeLog);

        return new JsonResponse($User);
    }
}
```

Kuva 6. Esimerkki UserController-luokasta ja newUser-funktiosta

Entiteettiluokat hoitavat tietokantamallinnosten noudon, luonnin, muokkaamisen ja poistamisen. Esimerkiksi uudesta käyttäjästä luodaan ensin entiteettiolio, johon täydennetään tiedot ohjaimeen tulleesta pyynnön hyötykuormasta. Entiteettiolio sisältää tietokantamallinnuksen, johon tiedot tallennetaan. Tietokantamallinnoksen save() -funktio tallentaa mallinnoksen datan tietokantaan. Entiteettiluokkaan voidaan kirjoittaa myös apufunktioita esimerkiksi salasanan enkryptaamiseksi tai entiteettidatan serialisoimiseksi datan palautusta varten.

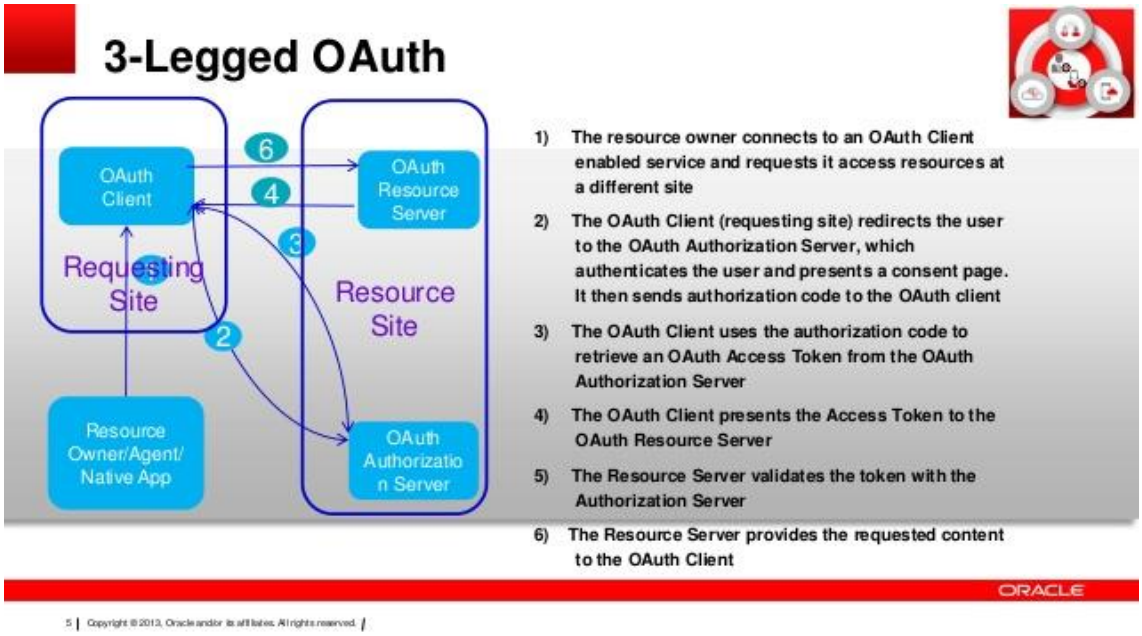
3.3 Autentikaatio ja autorisointi

Laravel tarjoaa useita valmiita autentikaatiotapoja. Rajapintakyselyjen autentikointiin valittiin Laravel’n Passport -kirjasto, joka on Laravel’n sisäinen oAuth2-palvelin. (Otwell, T.) OAuth2 on tällä hetkellä suosituin rajapintojen autentikaatiomenetelmä ja on käytössä muun muassa Facebookin, Instagramin ja Googlen käyttämissä rajapinnoissa. (Parecki, A.) OAuth2:en avulla kirjautuminen tapahtuu joko kaksi- tai kolmivaiheisesti. Kaksivaiheisessa kirjautumisessa käyttäjä yleensä käyttää palvelua suoraan palveluntarjoajan palvelimelta, jolloin autentikaatio tapahtuu esimerkiksi perinteisesti käyttäjänimen ja salasanan avulla. (Kuva 7) OAuth2-palvelin palauttaa oikeaa käyttäjänimi ja salasana -paria vasten avaimen (engl.access token), jonka avulla kyselyt autorisoidaan jokaisen kyselyn yhteydessä.



Kuva 7. Kaksiosainen OAuth-autentikointi (Oracle Corporation 2014)

Kolmiosaisessa autentikaatiossa käyttäjä antaa kolmannen osapuolen sovellukselle luvan käyttää rajapintaa puolestaan. Mikäli asiakas antaa luvan kolmannelle osapuolelle käyttää omia tietojaan kyselyjen tekemiseen, oAuth2-palvelin palauttaa asiakaskohtaisen avaimen kolmannelle osapuolelle. Tätä autentikaatiomenetelmää voidaan esimerkiksi käyttää, mikäli asiakas haluaa omasta sisällönhallintajärjestelmästä lisätä kuvia FirstView -näytöille. Tällöin asiakkaan palvelu tekee rajapintakyselyt omalla asiakastunnistenumerollaan ja avaimellaan loppukäyttäjäkohtaisen avaimen luvittamana. (Kuva 8)



Kuva 8. Kolmiosainen OAuth-autentikointi (Oracle Corporation 2014)

OAuth2-avaimiin voidaan liittää eri käyttäjäoikeuksia, jolloin autorisoinnin yhteydessä tarkistetaan aina, onko käyttäjällä oikeus käyttää haluttua päätepistettä. Käyttäjällä voi esimerkiksi olla lukuoikeus soittolistaan, mutta oikeus poistaa soittolista puuttu. Tällöin rajapinta palauttaa ”*403 FORBIDDEN”* -http-koodin, jolloin asiakasohjelma tietää, että tarvittava käyttöoikeus puuttuu. OAuth2-avain kulkee aina kyselyn otsakkeissa mukana.

3.4 Virheenkäsittely

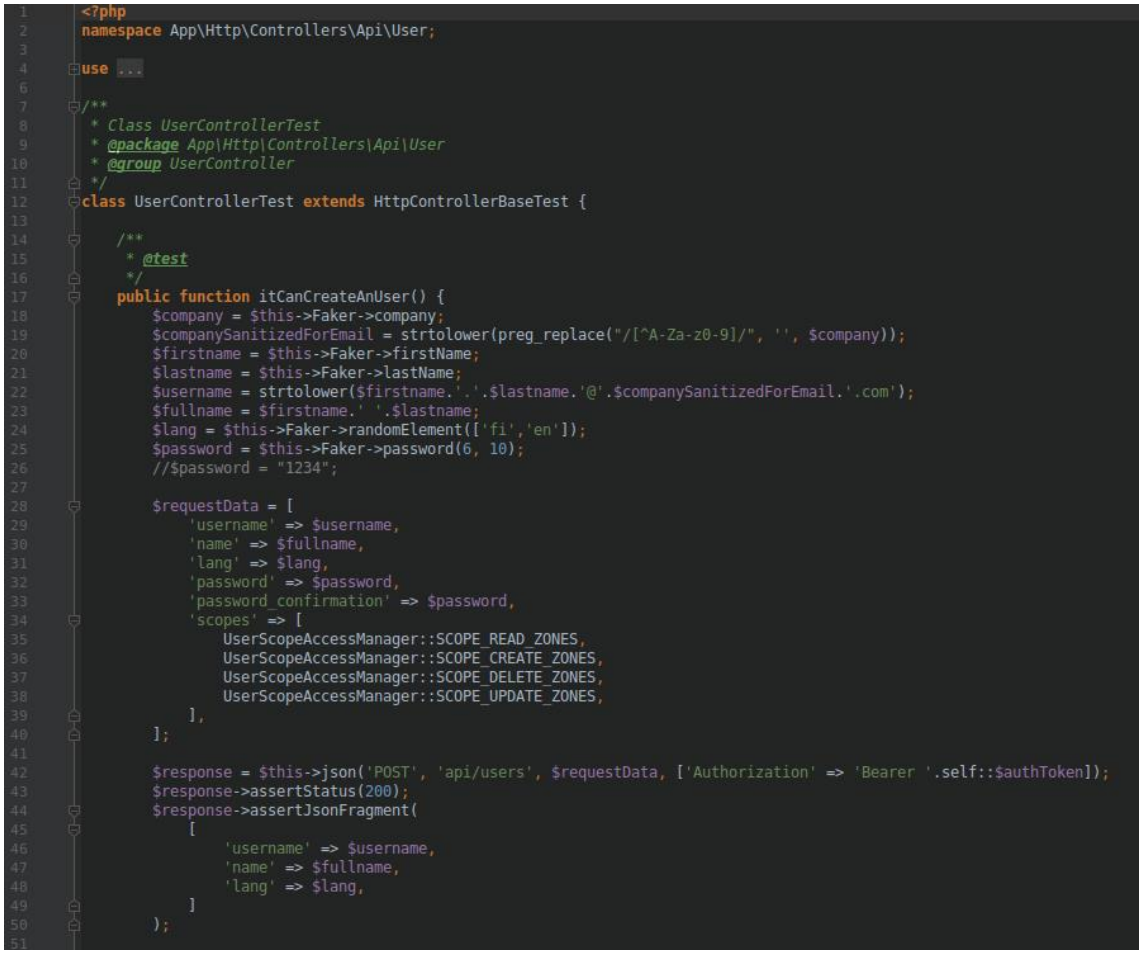
Laravel’n api-reittimäärittelyissä on sisäänrakennettuna virheviestien palautus JSON-muodossa asiakasohjelmalle. Mikäli esimerkiksi koodiin on jäänyt virhe, joka hajottaa ohjelman toiminnan, palautetaan automaattisesti http-statuskoodi ”*500 Internal Server Error”.* Mikäli virhe johtuu käyttäjän toimista, esimerkiksi yritetään kutsua päätepistettä, johon käyttäjällä ei ole oikeutta, palauttaa käyttöoikeudet tarkistava väliohjelma http-statuskoodin ”*403 Forbidden*”.

Omat poikkeusluokat kirjoitettiin yleisimmin tapahtuville virheille, jotta vältyttäisiin turhalta 500 http-statuskoodin palauttamiselta. Poikkeukset tarkistettiin niin, että itsekirjoitetut poikkeusluokat poimittiin ensin, mikäli virhe ei palauta mitään itsekirjoitettua poikkeusluokkaa, käytettiin juuripoikkeusluokkaa. Joka kerta kun poimitaan poikkeus, tämän poikkeuksen aiheuttanut toimenpide pyritään tallentamaan lokiin niin tarkasti, kuin se on

mahdollista. Turhan lokien keräämisen välttämiseksi lokin kirjoitus on huomattavasti vähäisempää, mikäli asennus on konfiguroitu tuotantotilaan. Kehitystilassa pyrittiin kirjaamaan lokiin virheet mahdollisimman kattavasti ohjelmakoodin korjauksen helpottamiseksi.

3.5 Yksikkötestien kirjoitus

Yksikkötestaus (engl.unit testing) on ohjelmallisesti suoritettavaa testausta, jolla pyritään varmistamaan ohjelman osan toimivuus odotetun kaltaisesti silloinkin, kun ympärillä oleva koodi muuttuu. (McFarlin, T. 2012) Laravel'ssa on mukana PHPUnit-kirjasto, joka mahdollistaa yksikkötestausten kirjoittamisen PHP-sovelluksiin. Yksikkötestejä kirjoittaessa testataan aina tiettyjä väitteitä. Esimerkiksi käyttäjänhallintaa testatessa voidaan luoda testifunktio *"itCanCreateUser"*, jossa luodaan ohjelmallisesti sovelluksen funktioita ja luokkia käyttäen uusi käyttäjä. Lopuksi tarkistetaan väite, että uusi id on syntynyt tietokantaan User-tauluun. Tämän lisäksi voidaan tarkistaa esimerkiksi, että käyttäjän salasana on salattu oikein ja käyttäjän sähköpostiosoite on oikean muotoinen. Esimerkki yksikkötestauksen ohjelmakoodista on kuvassa 9.



```php
<?php
namespace App\Http\Controllers\Api\User;

use ...

/**
 * Class UserControllerTest
 * @package App\Http\Controllers\Api\User
 * @group UserController
 */
class UserControllerTest extends HttpControllerBaseTest {

    /**
     * @test
     */
    public function itCanCreateAnUser() {
        $company = $this->Faker->company;
        $companySanitizedForEmail = strtolower(preg_replace("/[^A-Za-z0-9]/", '', $company));
        $firstname = $this->Faker->firstName;
        $lastname = $this->Faker->lastName;
        $username = strtolower($firstname.'.'.$lastname.'@'.$companySanitizedForEmail.'.com');
        $fullname = $firstname.' '.$lastname;
        $lang = $this->Faker->randomElement(['fi','en']);
        $password = $this->Faker->password(6, 10);
        //$password = "1234";

        $requestData = [
            'username' => $username,
            'name' => $fullname,
            'lang' => $lang,
            'password' => $password,
            'password_confirmation' => $password,
            'scopes' => [
                UserScopeAccessManager::SCOPE_READ_ZONES,
                UserScopeAccessManager::SCOPE_CREATE_ZONES,
                UserScopeAccessManager::SCOPE_DELETE_ZONES,
                UserScopeAccessManager::SCOPE_UPDATE_ZONES,
            ],
        ];

        $response = $this->json('POST', 'api/users', $requestData, ['Authorization' => 'Bearer '.self::$authToken]);
        $response->assertStatus(200);
        $response->assertJsonFragment(
            [
                'username' => $username,
                'name' => $fullname,
                'lang' => $lang,
            ]
        );
```

Kuva 9. Esimerkki UserController-testin ohjelmakoodista

Kun ohjelmakoodia on päivitetty, kannattaa testata aina kaikki osa-alueet, vaikka niihin ei suoranaisesti ole edes koskettu. Näin varmistetaan, ettei muutoksia tehdessä ole rikottu mitään aikaisemmin kirjoitettua koodia. Testien kirjoittajia on yleensä kahta koulukuntaa. Toiset kirjoittavat testit ennen varsinaista ohjelmakoodin kirjoittamista, jolloin he voivat todeta ohjelmakoodin olevan valmis, kun testi menee läpi. Toinen koulukunta kirjoittaa testit jälkeenpäin. Itse suosin testien jälkeenpäin kirjoittamista, koska koodin kirjoittaminen voi usein poikia uusia ideoita, miten asian voisi tehdä tehokkaammin ja alkuperäinen ajatus muuttuu.

4 REST-APIN TUOTTEISTUS

Toimiva sovellus ei koskaan ole vielä toimiva tuote. Jotta tuotteen voi julkaista asiakkaiden ja yhteistyökumppaneiden käyttöön, tulee tuotteen olla läpikotaisin testattu ja dokumentoitu. Koska dokumenttien kirjoittaminen on erittäin aikaa vievää, päätettiin yrittää automatisoida prosessi mahdollisimman pitkälle. Myös testaus päätettiin suorittaa vapaaehtoisilla asiakkailla alpha- ja betatestusvaiheissa.

4.1 Dokumentointi

Projektia suunnitellessa painotettiin, että dokumentointi on ensiarvoisen tärkeää näin laajassa projektissa. Varsinainen rajapintaesittely ja dokumentointi toteutettiin apidoc -työkalulla. Tämän lisäksi pidettiin muutoslokia ajan tasalla, josta asiakkaat näkevät kertasilmäyksellä viimeisimmät muutokset ja päivitykset järjestelmässä. Tämä muutosloki koostetaan versionhallinnan julkaisuviesteistä tunnisteiden avulla. Muutoslokiin tulevat asiat on merkattu tunnisteilla, jotta saadaan julkaisuviesteihin myös vain kehittäjien tarvitsemaa tietoa, joista ei ole loppuasiakkaille hyötyä.

Kolmannen osapuolen rajapintakäyttäjille päätetiin kirjoittaa oma dokumentointi, mikäli yhteistyökumppani haluaa omaan sovellukseensa integroida FirstView MediaCloud’n toiminnallisuuksia. Dokumentissa ohjeistetaan muun muassa OAuth2’n käyttöä ja kerrotaan tarvittaessa eri päätepisteiden rajoituksista ja toiminnoista pelkkää apidoc-rajapintakuvausta tarkemmin.

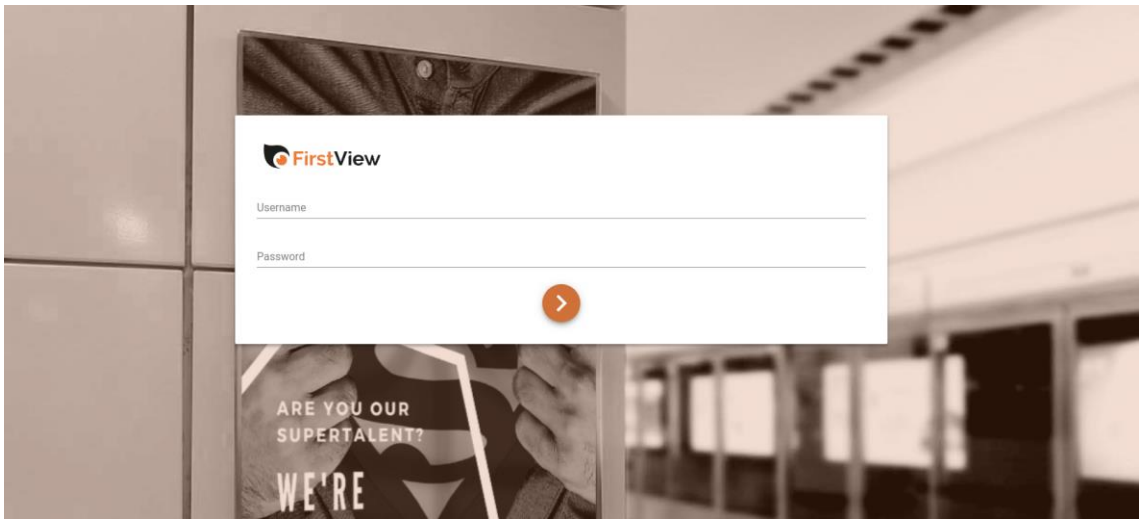
4.2 Esimerkki rajapinnan käytöstä asiakasohjelman kautta

FirstView MediaCloud rajapinnan lisäksi kehitettiin rinnalle täysin uutta käyttöliittymää, joka käyttää MediaCloud-rajapintaa. Käyttöliittymän kehittäminen jatkuvasti taustajärjestelmän rinnalla helpottaa rajapinnan ohjelmointia, koska tällöin saadaan ensikäden tietoa tarvittavista päätepisteistä ja realistista testitulosta taustajärjestelmän toimivuudesta.

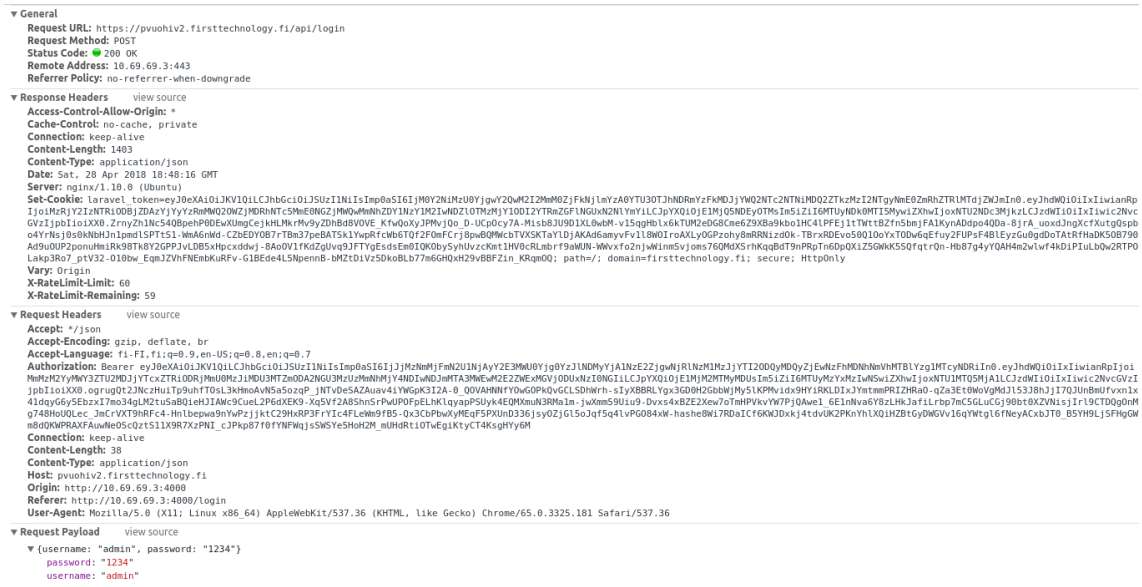
Käyttöliittymä on tehty react.js-sovelluskehystä hyödyntäen ja käyttöliittymäkomponentit saadaan material-ui -kirjastosta. Käyttöliittymästä löytyy react.js’n tarjoama kirjasto

asynkronisten http-kutsujen tekemiseen, minkä avulla dataa haetaan taustajärjestelmästä.

Esimerkiksi autentikaatio on toteutettu perinteisellä sisäänkirjautumislomakkeella. (Kuva 10) Käyttäjä syöttää kenttiin käyttäjänimensä ja salasanansa, jolloin käyttöliittymä kutsuu taustajärjestelmän *”POST api/auth/login”* päätepistettä, jossa parametrina menee käyttäjän syöttämä käyttäjänimi ja salasana. Kuvassa 11 on esimerkki kutsusta. Päätepiste palauttaa avaimen, mikäli käyttäjänimi ja salasana ovat oikein. Autentikoinnin jälkeen kaikki kyselyt taustajärjestelmään autorisoidaan kirjautumisessa saadulla avaimella. Rajapinnassa on myös ”logout” päätepiste, jonka avulla avaimesta tulee käyttökelvoton ja sessio päätetään.



Kuva 10. Login-sivu käyttöliittymässä



Kuva 11. Esimerkki POST /api/login -kutsusta

4.3 Asiakastestaus ja julkaisu

Projektin työstäminen aloitettiin syksyllä 2016 ja varsinainen kirjoitusprosessi aloitettiin keväällä 2017. Tavoitteeksi ensimmäisen testiversion, FirstView MediaCloud Alpha'n julkaisusta asiakastestaukseen asetettiin 2017—2018 vuodenvaihde. Tästä kuitenkin myöhästyttiin hieman, joten ensimmäinen testiversio julkaistiin maaliskuussa 2018. Asiakastestauksen hyöty saadaan palautteesta, jota testaajat voivat jättää suoraan käyttöliittymästä. Testausjakson aikana tutkittiin myös aktiivisesti lokeja joista ilmenee tapahtuneet poikkeukset ja virheet, sekä yleisimmin toistuvat kutsut. Tämä antaa arvokasta tietoa siitä, miten asiakkaat palvelua käyttävät.

Betatestausvaihe on ajoitettu kesä— heinäkuulle 2018, jolloin ensimmäisessä testauksessa esiinnousseet viat ja epäjohdonmukaisuudet on korjattu. Lopullinen julkaisu ja siirtymävaihe vanhasta järjestelmästä uuteen on asetettu syyskuulle 2018. Silloin tavoitteena on saada kaikki toiminnallisuudet vanhasta järjestelmästä toimimaan uudessa MediaCloudissa, jonka jälkeen voidaan alkaa keskittyä uusien ominaisuuksien luomiseen, mikä nyt uuden järjestelmän myötä on todella paljon helpompaa ja ketterämpää.

5 LOPUKSI

Opinnäytetyö koostui REST-rajapinnan suunnittelu- ja toteutustyöstä. Tämä käsitti käytettyjen sovelluskehysten valintaa, ohjelmistokokonaisuuden paloittelua projektinhallinnallisiksi osiksi ja näiden osien toteuttamista toimivaksi kokonaisuudeksi. Työn toteuttamiseksi käytettiin pääasiassa PHP7-ohjelmointikieltä ja Laravel-sovelluskehystä. Työn pohjana pidettiin jo olemassa ollutta FirstView Manager -järjestelmää, jonka ydintoiminnallisuudet kirjoitettiin täysin uusiksi.

Projektin alkuvaiheessa arvioitu työn määrä osoittautui nopeasti liian alhaiseksi. Suunnitteluvaiheessa havaittiin, että koko vanha järjestelmä täytyi kirjoittaa uusiksi, jotta tavoite modernista ja tämän päivän standardit täyttävästä rajapinnasta täyttyisi.

Ongelmana suunnitteluvaiheessa oli puutteellinen projektinhallinta. Mittavan kokonaisuuden hallinta osoittautui erittäin haastavaksi ilman kunnollista ja järjestelmällistä projektinhallintatyökalua. Tämän vuoksi osaksi opinnäytetyötä otettiin myös projektinhallintamenetelmien ja työkalujen tutkiminen. Suunnitteluvaiheessa toimivimmaksi osoittautunut projektinhallintamenetelmä ja työkalu päätettiin ottaa käyttöön varsinaiseen projektin toteutusvaiheeseen. Tämä edesauttoi projektin edistymistä järjestelmällisesti alkuperäisen suunnitelman mukaisesti. Valittujen ketterien ohjelmistokehitysmenetelmien käyttö mahdollisti esiin tulleiden muutosideoiden toteuttamisen vaivattomasti.

Opinnäytetyön edetessä osoittautui, että valmiiden työkalujen hyödyntäminen niin pitkälle kuin se järkevästi on mahdollista tehostaa huomattavasti tuotekehitysprojektia. Laravel-sovelluskehyksen käytön ansiosta säästettiin työtunteja. Nämä työtunnit voitiin tehokkaasti käyttää varsinaisten toiminnallisuuksien ohjelmointiin taustalla tapahtuvan tiedonsiirron ja raakatietokantakyselyjen sijaan.

Projektista saadun arvokkaan asiakaspalautteen avulla saatiin täydennettyä suunnitelmia lopullista julkaisua silmällä pitäen. Projektin myötä kirjoitetun ohjelmakoodin määrä on lähes puolittunut vanhaan järjestelmään verrattuna. Uusi taustajärjestelmä on myös todella paljon kevyempi suoritustehovaatimuksien suhteen. Myös testauksessa saatu käyttäjäpalaute on osoittanut, että loppukäyttäjälle on jäänyt kokemus palvelun nopeutumisesta ja käyttöliittymän ulkoasua on kiitelty kovasti. REST-rajapinta taustajärjestelmänä mahdollistaakin käyttöliittymäuudistusten tekemisen erittäin vaivattomasti, koska

taustajärjestelmän ja käyttöliittymän ohjelmakoodi ovat täysin erillisiä ja omia yksilöllisiä projektejaan.

LÄHTEET

Fielding, Roy Thomas 2000. Architectural Styles and the Design of Network-based Software Architectures. University Of California, Irvine. Saatavilla sähköisesti osoitteessa http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

FirstView Digital Signage. Yritys. Viitattu 24.4.2018 http://www.firstview.fi > Yritys

McFarlin, T. 2012 The Beginner's Guide to Unit Testing: What Is Unit Testing? Viitattu 26.4.2018 https://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-2Parecki, A. OAuth 2.0. Viitattu 25.4.2018 https://oauth.net/2/

Rodrigues, Alex 2015. RESTful Web services: The basics. IBM Corporation. Saatavilla sähköisesti osoitteessa https://www.ibm.com/developerworks/webservices/library/ws-restful/ws-restful-pdf.pdf

W3. 10 Status Code Definitions. Viitattu 25.4.2018 https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Opelt, A.; Gloger, B.; Pfarl, W. & Mittermayr, R. 2013. Agile Contracts: Creating and Managing Successful Projects with Scrum. John Wiley & Sons, Incorporated

Otwell, T. Installation. Viitattu 25.4.2018 https://laravel.com > documentation

Otwell, T. Eloquent: Getting Started. Viitattu 25.4.2018 https://laravel.com > documentation > Eloquent ORM > Getting Started

Otwell, T. Database: Migrations. Viitattu 26.4.2018 https://laravel.com > documentation > Database > Migrations

Otwell, T. Database: Routing. Viitattu 26.4.2018 https://laravel.com > documentation > The Basics > Routing

Otwell, T. Database: Middleware. Viitattu 26.4.2018 https://laravel.com > documentation > The Basics > Middleware

Otwell, T. Database: Controllers. Viitattu 27.4.2018 https://laravel.com > documentation > The Basics > Controllers

Otwell, T. Database: API Authentication (Passport). Viitattu 27.4.2018 https://laravel.com > documentation > Security > API Authentication

James, G. 2017. Creating a simple REST API in PHP. Viitattu 28.4.2018 https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php

Oracle Corporation 2014. Introduction to OAuth2.0. Viitattu 28.4.2018 https://www.slideshare.net/AliceLiu1980/introduction-to-oauth2