LAPIN AMK
Lapland University of Applied Sciences

MEDIAN UPOTTAMINEN TAIDEART-HANKKEEN VERKKOSOVELLUKSEEN

TaideART-hanke

Tervonen Mika

Opinnäytetyö
Tietojenkäsittely ja tietoliikenne (ICT)
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

2017

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tietojenkäsittely ja tietoliikenne (ICT)
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Tekijä Mika Tervonen **Vuosi** 2017
Ohjaaja(t) Yrjö Koskenniemi
Toimeksiantaja TaideART-hanke
Työn nimi Median upottaminen TaideART-hankkeen verkkosovellukseen
Sivu- ja liitesivumäärä 34 + 3

Opinnäytetyön tavoitteena on selvittää ja tuottaa TaideART-hankkeen verkkosovellukselle soveltuva metodi median upottamiseen noudattaen avoimen lähteen periaatetta sekä oEmbed- ja Open Graph -protokollaa.

Opinnäytetyön tutkimusaineisto on hankittu pääosin avoimesta verkkomateriaalista ja käytännön kokeiden perusteella. Tutkimusaineisto käsitteli oEmbed- ja Open Graph -protokollien soveltamista käytäntöön.

Tutkimustyö toteutettiin ketteriä kehitysmenetelmiä hyväksi käyttäen ja testivetoista kehitysmallia soveltaen sekä noudattaen yleistä sovelluskehityksen kulkua.

Opinnäytetyön tuloksena syntyi TaideART-hankkeen käyttöön median upottamiselle metodi, jota voidaan soveltaa myös muihin sovellusprojekteihin.

Avainsanat oEmbed, Open Graph, embed, embed media

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

School of Business and Culture
Information and Communications Technology
Bachelor of Business Administration

Author Mika Tervonen **Year** 2017
Supervisor Yrjö Koskenniemi
Commissioned by TaideART-hanke
Subject of thesis Embedding media in TaideART-project web app
Number of pages 34 + 3

The aim of this thesis is to find out and produce a method for embedding media on the Taide ART project web application, following the open source principle and the oEmbed and Open Graph protocols.

The research data of this thesis was obtained mainly from open web materials and practical experiments. The material considered the oEmbed and Open Graph application protocols into practice.

The research was carried out for the benefit of agile development methods and using test-driven development model by applying and in keeping with the overall application development flow.

The result of this thesis was the use of the Taide ART project to embed media into a method that can be applied to other application projects as well.

Key words oEmbed, Open Graph, embed, embed media

Vastuuvapauslauseke

Kaikki tässä opinnäytetyössä esitellyt koodiesimerkit toimitetaan “sellaisenaan”, ilman minkäänlaisia takuita, mukaan lukien mutta siihen rajoittumatta, soveltuvuudesta tiettyyn tarkoitukseen. Missään tapauksessa tekijä ei ole vastuussa komentojen käytöstä aiheutuneesta välittömästä tai välillisestä aineettomasta tai aineellisesta vahingosta. Kaikkien komentojen käyttö on vapaata, mutta suoraan käyttäjän omalla vastuulla.

Mika Tervonen

SISÄLLYS

Lyhenteet

API	Application programming interface. Ohjelmointirajapinta.
AJAX	Asynchronous JavaScript and XML. Asynkroninen JavaScript ja XML on rakennusmenetelmä web-sovelluksille, jotka käsittelevät käyttäjän pyyntöjä välittömästi.
HTML	HyperText Markup Language. Verkon perusrakenne, joka kuvaa ja määrittelee verkkosivun sisällön.
HTTP	HyperText Transfer Protocol. Protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon.
JSON	JavaScript Object Notation. Standardimuoto jäsenneltyjen tietojen esittämiseksi JavaScript-objekteina, ja sitä käytetään yleisesti siirrettäessä tietoja palvelimelta asiakkaalle.
MVC	Model-View-Controller. Malli-Näkymä-Käsittelijä on ohjelmisto-arkkitehtuurityyli, jonka tarkoituksena on käyttöliittymän erottaminen sovellusalueen tiedosta.
PHP	Hypertext Preprocessor. Yleiskäyttöinen avoimen lähdekoodin palvelinpohjainen kirjoituskieli, joka soveltuu erityisesti dynaamiseen web-kehitykseen ja voidaan upottaa HTML-muotoon.
SEO	Search Engine Optimization. Hakukoneoptimointi on protokolla verkkosivun sisällön määrittelyyn metatietojen avulla näkyvyyttä hakukoneiden tuloksissa parantavalla tavalla.
URL	Uniform Resource Locator. URL-osoitetta käytetään yksilöimään resursseja webissä

1 JOHDANTO

Monet palveluntuottajat tarjoavat nykyään oman API[1]-rajapinnan lisäksi oEmbed-protokollan rajapinnan palveluunsa. oEmbed-protokolla tarjoaa mahdollisuuden noutaa palveluntarjoajan mediasisältöä palvelusta ja sen sisällyttämisen edelleen verkkosivustolle (oEmbed 2017). Osa palveluntarjoajista tosin edellyttää kehittäjätiliä palveluun ja API-avainta oEmbed-palvelun käyttämiseen (Facebook for Developers 2017a; 2017b), mutta iso osa tarjoaa palvelun vapaaseen käyttöön.

Palvelun tarjoajien lista on pitkä varsinkin käytettäessä maksullista palvelua mediasisällön upottamiseen (Embedly 2017), mutta myös avoimen lähdekoodin saralla määrä kasvaa koko ajan (oEmbed 2017). Esimerkiksi YouTube, Vimeo, DeviantArt ja Instagram ovat TaideART-hankkeen kannalta sovellukseen sopivia palveluntarjoajia.

Upottaminen eroaa normaalista verkkosivun linkityksestä teknisen toteutuksen osalta. Tuovan linkin upottamisella upotuskoodilla (embed code) sivuston ulkopuolista mediasisältöä tuodaan osaksi linkittäjän omia sivuja. Upottaminen siis tuo ulkopuolisen aineiston näkyviin linkittäjän sivulle, mutta aineiston todellinen tallennuspaikka säilyy edelleen linkittäjän sivun ulkopuolella. Visuaalisesti upotetun linkin tunnistaa yleensä aineiston ympärillä olevista aineiston tekoon käytetyn ohjelman kehyksistä kuten esimerkiksi iframe (Mozilla Developer Network 2017).

Tässä työssä on tarkoitus upottaa mediaa käyttämällä avoimeen aineistoon osoittavia sivustolinkkejä ja purkamalla sivun metatiedoista Open Graph -protokollan tiedot. Metatietojen avulla voidaan upottaa sivustolla oleva kuva, tai muu media, hankkeen web-sovellukseen. Upottamiseen käytetään javascript-kirjastoja ja PHP[2]-koodia, joiden avulla on tarkoitus vähentää ladattavan datan määrää käyttäjän selaimella ja kohdistaa haku tarkemmin sisältöön. Videomateriaali tarvitsee upotettuna edelleen iframe-ympäristöä

[1] *Application programming interface (API)*
[2] *Hypertext Preprocessor (PHP)*

(YouTube 2016), mutta purkamalla metadatasta videoon liittyvä esikatselukuva voidaan sovellukseen valita upotettavaksi ainoastaan se ja linkittää varsinainen video ulkopuoliseen palveluun katsottavaksi.

Hankkeen web-sovellus sallii käyttäjän liittää palveluun linkkejä ulkopuoliseen mediaan, joten on olemassa riski sisällön sopimattomuuteen koko perheelle (VAHTI 4/2010 2010). Purkamalla ja analysoimalla metadatan sisältöä upottamiseen käytetty jQuery-kirjasto voi estää sisällön näyttämisen, mikäli palveluntarjoaja on sisällyttänyt isfamilyfriendly-indexin metadataan. Jos indexi puuttuu tai on merkitty hyväksytyksi ([isfamilyfriendly]=>True), sisältö näytetään, muutoin se hylätään. Ongelmana on, että kaikki palveluntarjoajat eivät välttämättä sisällytä isfamilyfriendly-indexiä omaan materiaaliinsa.

Työn tarjoajana toimii TaideART-hanke, joka on kuuden ammattikorkeakoulun yhteinen hanke taiteilijan ansaintamallien, roolin ja toimeentulon (TaideART) edistämiseksi. Hankkeen aikana pyritään kehittämään taiteilijoille tuotteistamisen keinoja ja tuotemalleja yhteistyössä työelämän ja eri taidealojen kanssa. Hankkeen tulosten tarkoitus on edesauttaa taiteilijoita ansaitsemaan elantonsa omalla taiteellaan. (TaideART-hanke 2017.)

2 TYÖN TAVOITTEET JA ONGELMANASETTELU

2.1 Työn tavoitteet

Tämän opinnäytetyön tavoitteena on löytää TaideART-hankkeelle soveltuva metodi kolmannen osapuolen mediasisällön sulauttamiseen verkkosivulle, ja se toteutetaan oEmbed- ja Open Graph -protokollaa hyödyntämällä.

Työn varsinainen suunniteltu toteutus keskittyy käyttämään joko jo olemassa olevia metodeja tai luomaan niiden pohjalta hankkeen käyttöön soveltuvan mallin. Työn tarkoituksena ei ole luoda kokonaan uutta mallia vaan muokata jo olemassa olevia soveltumaan suuremmalle määrälle mediaa. Lähtökohtaisesti kaikki oEmbed-protokollaan liittyvä dokumentointi olettaa upotettavan sisällön olevan määrällisesti pienempi kuin tarve, joka hankkeen sovelluksella on. Toteutukseen päätyvä menetelmä on myös opinnäytetyön lopullinen tulos.

2.2 Työn rajaukset

Opinnäytetyössä ei käsitellä TaideART-hankkeen verkkosovelluksen toteuttamista muutoin kuin tilanteissa, joissa se on tarpeellista tuloksen ymmärrettävyyden ja todennettavuuden kannalta.

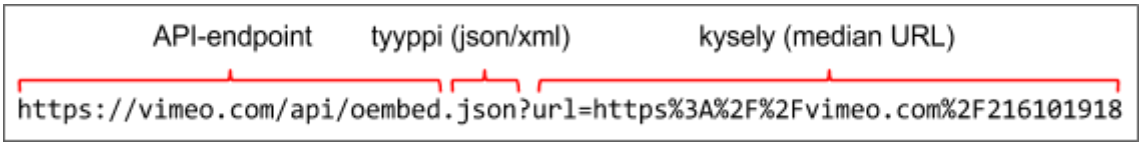
TaideART-hankkeen periaate avoimen lähdekoodin materiaalista asettaa myös rajoitteita varsinaiselle työn suoritukselle, koska sovelluksessa käytettävän materiaalin pitää olla joko kokonaan avointa tai ei kaupallisen, voittoa tuottamattoman järjestön/yhteisön käytössä maksutonta. Lisäksi on otettava huomioon hankkeen elinkaari sekä siihen liittyen sovelluksen toiminnallisuus ilman aktiivista ylläpitoa.

3 PROTOKOLLAT

3.1 OEMBED-protokolla

OEmbed-protokolla on median määrittelyn ohjeistus, joka tarjoaa palvelun tuottajalle metodin jakaa omaa materiaalia. Protokollan määritelmien mukaan URL[3]-osoitteen perusteella voidaan upottaa mediaa kolmannen osapuolen sivustolle. Yksinkertaisimmillaan tämä sallii verkkosivun näyttää upotettua sisältöä kuten kuvia, videota tai muuta mediaa. (oEmbed 2107.)

OEmbed-protokollan periaate toimii niin, että käyttäjä lähettää halutun median URL-osoitteen HTTP[4]-kyselyn parametrina palvelua tarjoavan toimijan palvelimelle kuten Kuvan 1 Vimeo-palvelun esimerkissä. Palvelin tulkitsee HTTP-kyselystä url-parametrin tiedot ja palauttaa siinä pyydetyn median tiedot JSON[5] -muodossa.



Kuva 1. Vimeo-palvelun API-endpoint esimerkki osiin purettuna

JSON-tiedot sisältävät yleensä pyydetystä mediasta ainakin median tyypin, joka Koodiesimerkissä 1 on video, palveluntarjoajan nimen (provider_name), upotukseen tarvittavan koodin (html tai code) ja median esikatselukuvan (thumbnail_url). Esikatselukuva on yleensä sisällytetty vain video- tai link-tyyppisen tiedon yhteyteen. Tosin kaikki videomateriaalia tarjoavat palvelut eivät välttämättä käytä sitä omien tietojensa joukossa, koska se ei ole pakollinen ja toisaalta se saattaa löytyä kuvia tarjoavasta palvelusta.

Kaikille oEmbed-palvelun tarjoajille pakollisia tietoja ovat mm. tyyppi, versio, palveluntarjoajan nimi ja palveluntarjoajan URL-osoite. Lisäksi tyyppikohtaisia pakollisia tietoja videolle ja kuvalle ovat html, width ja height, sekä jos parametri

---

[3] *Uniform Resource Locator (URL)*
[4] *HyperText Transfer Protocol (HTTP)*
[5] *JavaScript Object Notation (JSON)*

thumbnail_url on asetettuna, on myös thumbnail_width ja thumbnail_height oltava mukana. (oEmbed 2107.)

```
//https://vimeo.com/api/oembed.json?url=https://vimeo.com/216101918
//JSON-data
{
  "type": "video",
  "version": "1.0",
  "provider_name": "Vimeo",
  "provider_url": "https://vimeo.com/",
  "title": "The Carousel",
  "author_name": "Jonathan Napolitano",
  "author_url": "https://vimeo.com/nahtanoj",
  "is_plus": "1",
  "html": "<iframe src=\"https://player.vimeo.com/video/216101918\"
width=\"640\" height=\"360\" frameborder=\"0\" title=\"The Carousel\"
webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe>",
  "width": 640,
  "height": 360,
  "duration": 698,
  "description": "In the small town of Binghamton, New York, there spins a
1925 carousel that once inspired Rod Serling and has since become a portal
into...the Twilight Zone.",
  "thumbnail_url": "https://i.vimeocdn.com/video/632990364_640.webp",
  "thumbnail_width": 640,
  "thumbnail_height": 360,
  "thumbnail_url_with_play_button":
"https://i.vimeocdn.com/filter/overlay?src0=https%3A%2F%2Fi.vimeocdn.com%2F
video%2F632990364_640.webp&src1=http%3A%2F%2Ff.vimeocdn.com%2Fp%2Fimages%2F
crawler_play.png",
  "upload_date": "2017-05-04 21:43:58",
  "video_id": 216101918,
  "uri": "/videos/216101918"
}
```

Koodiesimerkki 1. Vimeo-palvelun oembed-palvelimen palauttamat median tiedot

JSON-tiedot saattavat sisältää paljon muutakin mediaa yksilöivää tietoa, ja tietojen määrä, järjestys tai merkkaustapa saattavat vaihdella suuresti eri palveluntarjoajien kesken kuten Koodiesimerkissä 2 YouTube-palvelun palauttamista tiedoista on nähtävissä. Youtube-palvelu palauttaa paljon vähemmän tietoja kuin edellisen esimerkin Vimeo.

```
//https://www.youtube.com/oembed?url=https://www.youtube.com/watch?v=_EJVfm
NjPUg
//JSON-data
{
  "provider_name": "YouTube",
  "thumbnail_width": 480,
  "type": "video",
  "version": "1.0",
  "provider_url": "https://www.youtube.com/",
```

```
  "title": "Demo Day 2017 - Hosted By Google",
  "html": "<iframe width=\"480\" height=\"270\"
src=\"https://www.youtube.com/embed/_EJVfmNjPUg?feature=oembed\"
frameborder=\"0\" allowfullscreen></iframe>",
  "height": 270,
  "author_name": "Google for Entrepreneurs",
  "thumbnail_url": "https://i.ytimg.com/vi/_EJVfmNjPUg/hqdefault.jpg",
  "width": 480,
  "author_url": "https://www.youtube.com/user/Google4Entrepreneurs",
  "thumbnail_height": 360
}
```

Koodiesimerkki 2. Median tiedot Youtube-palvelusta

3.2 OPEN GRAPH -protokolla

Open Graph -protokolla on verkkosivun sisällön määrittelyn ohjeistus ja osa Facebookin Social Graph -ydintä, johon se liitettiin laajennuksena vuonna 2010 (Fang 2011). Protokollan perusteella jokaiseen verkkosivuun voidaan sisällyttää tarkempaa tietoa itse verkkosivun sisällöstä ja näin muodostaa sisällöltään rikkaampi objekti mahdollista sulauttamista varten. Protokolla perustuu Resource Description Framework -konseptiin (RDFa), joka määrittelee HTML[6] -dokumentin metatietojen käsittelyn ja säännöt (World Wide Web Consortium, W3C 2014).

Protokolla on kehitetty hakukoneoptimointia (SEO)[7] varten vähentämään HTML <title>-elementin taakkaa ja parantamaan hakutuloksen osuvuutta. Hakukoneoptimointi yleisellä tasolla tarkoittaa verkkosivun sisällön määrittelyä metatietojen avulla näkymään mahdollisimman houkuttelevasti hakupalvelujen hakutuloksissa (Fishkin 2015).

Open Graph -protokollan meta-tiedot löytyvät HTML-dokumentin <head> elementistä, joka sisältää dokumentin muitakin yleisiä tietoja kuten dokumentin otsikon, linkit, skriptit ja tyylitiedostot. Open Graph -protokollan meta-tiedot noudattavat määrityksien mukaisesti og-etuliitteellistä <meta property=”og:{ parametri }” content=”{ parametrin sisältö }”> muotoa. Koodiesimerkissä 3 Vimeo-palvelun Open Graph -metatiedoista voidaan osoittaa mm. sisällön tyyppi (og:type), kuvaus (og:description) ja esikatselukuva (og:image), jotka ovat tätä opinnäytetyötä ajatellen tärkeimmät parametrit.

[6] *HyperText Markup Language (HTML)*
[7] *Search Engine Optimization (SEO)*

```
//https://vimeo.com/216101918
//HTML
<head>
...
<meta property="fb:app_id" content="19884028963">
<meta property="og:site_name" content="Vimeo">
<meta property="og:url" content="https://vimeo.com/216101918">
<meta property="og:type" content="video">
<meta property="og:title" content="The Carousel">
<meta property="og:description" content="In the small town of Binghamton,
New York, there spins a 1925 carousel that once inspired Rod Serling and
has since become a portal into...the Twilight Zone.">
<meta property="og:updated_time" content="2017-06-04T07:17:57-04:00">
<meta property="og:image"
content="https://i.vimeocdn.com/video/632990364_1280x720.webp">
<meta property="og:image:secure_url"
content="https://i.vimeocdn.com/video/632990364_1280x720.webp">
<meta property="og:image:type" content="image/jpg">
<meta property="og:image:width" content="1280">
<meta property="og:image:height" content="720">
<meta property="og:video:url"
content="https://player.vimeo.com/video/216101918?autoplay=1">
<meta property="og:video:secure_url"
content="https://player.vimeo.com/video/216101918?autoplay=1">
<meta property="og:video:type" content="text/html">
<meta property="og:video:width" content="1280">
<meta property="og:video:height" content="720">
<meta property="og:video:url"
content="https://vimeo.com/moogaloop.swf?clip_id=216101918&amp;autoplay=1">
<meta property="og:video:secure_url"
content="https://vimeo.com/moogaloop.swf?clip_id=216101918&amp;autoplay=1">
<meta property="og:video:type" content="application/x-shockwave-flash">
<meta property="og:video:width" content="1280">
<meta property="og:video:height" content="720">
...
</head>
```

Koodiesimerkki 3. Vimeo-palvelun html-dokumentin Open Graph -metatiedot

3.3 HTTP-protokolla

Internet pohjautuu valtaosaltaan perinteiseen asiakas-palvelin-arkkitehtuuriin, ja tämä toteutetaan tarkoitukseen kehitetyllä HTTP-protokollalla (The TCP/IP Guide 2005). Protokolla kuvaa ympäristön, jossa sovellus on jaettu kahteen osaan, asiakkaaseen ja palvelimeen, ja ne toimivat eri käyttöjärjestelmissä ja eri päätelaitteilla. Osat kuitenkin toimivat yhteen tarkoituksenaan luoda loppukäyttäjälle jokin palvelu. Asiakas on protokollan osa, jonka käyttäjä varsinaisesti näkee ja joka toimii loppukäyttäjän päätelaitteella. Palvelin-osa ei ole suoraan havaittavissa loppukäyttäjälle, mutta se koetaan osana palvelun

toimintaa (IBM Knowledge Center 2014).

Liitteessä 1 kuvataan yksinkertainen verkkoselaimen ja palvelimen välinen kommunikointi POST-metodilla. Käyttäjä avaa sivun, jolloin selain pyytää tiedostoa palvelimelta. Palvelin vastaa lähettämällä pyydetyn tiedoston selaimelle, joka edelleen piirtää näkymän käyttäjälle HTML-dokumentin ohjeiden mukaisesti.

Tämän esimerkin sivu voi olla vaikka jokin lomake. Käyttäjä syöttää lomakkeelle kysytyt tiedot ja lähettää lomakkeen palvelimelle HTTP-protokollan POST-metodilla. Palvelin käsittelee tiedot ja palauttaa selaimelle jälleen HTML-muotoisen vastauksen. Lopuksi selain piirtää koko sivun vastaanotetun vastauksen ohjeiden mukaisesti.

Toinen yleisesti käytetty kommunikointi toteutetaan AJAX[8]-metodilla. AJAX ei varsinaisesti ole ohjelmointikieli vaan selaimen puolella toimiva javascript-koodi, jonka avulla voidaan toteuttaa kommunikointi palvelimen ja selaimen välillä päivittämättä koko sivua uudelleen (Segue Technologies 2013). AJAX käyttää selaimen javascript-ympäristön XMLHttpRequest-objektia kommunikoinnin alustana.

Liitteessä 2 kuvataan verkkoselaimen ja palvelimen välinen kommunikointi AJAX-metodilla. Kommunikointi näin toteutettuna sallii selaimen päivittää vain muuttunut osa näkymää, eikä koko sivua tarvitse piirtää uudelleen. AJAX-funktio kutsuu palvelinta käyttäjän tai tapahtuman aktivoimalla herätteellä ja saa palvelimen vastauksen JSON-muotoisena. Tiedot voidaan liittää HTML-dokumenttiin javascriptillä, ja käyttäjä näkee sivulla muuttuneet tiedot.

[8] *Asynchronous JavaScript and XML (AJAX)*

4 KEHITYSYMPÄRISTÖ JA TYÖKALUT

4.1 Sovelluksen ohjelmointikieli

Sovelluksen kehityksessä käytettäväksi ohjelmointikieleksi määräytyi PHP, koska sovelluksen lopullista ajoympäristöä ei pystytty määrittelemään ennen kehityksen aloittamista. PHP on tuettuna valtaosalla palvelin-ympäristöistä (W3Techs 2017), joten se oli todennäköisin perustein paras vaihtoehto.

Sovelluksen kehitysympäristöä valittaessa ehdolla oli useita ilmaisia ja avoimen lähdekoodin PHP-pohjaisia vaihtoehtoja ympäristölle, kuten mm. Symphony, CakePHP ja Laravel. Vaihtoehtoja testattiin kevyesti, ja koekäytön myötä muut vaihtoehdot jäivät selkeästi Laravel-ympäristön varjoon monimutkaisuutensa vuoksi.

TaideART-hankkeen sovelluksen kehitysympäristöksi valikoitui siis Laravel-alusta, joka on avoimen lähdekoodin web-sovelluksille suunnattu PHP-kehitysympäristö. Laravel noudattaa rakenteessaan MVC[9]-arkkitehtuuria, ja modulaarisuutensa ansiosta se on helposti laajennettavissa erilaisilla PHP-kirjastoilla (Laravel 2017).

Hankkeen sovelluksen kannalta Laravelin merkittäviä ominaisuuksia on mm. sen mukana toimitettava Eloquent, tietokantamallinnuksen kirjasto, joka helpottaa työskentelyä tietokantojen kanssa riippumatta tietokannan tyypistä. Toinen sovelluksen kannalta olennainen ominaisuus Laravelissa on laajan dokumentoinnin selkeys sekä Laravelin ympärille kehittynyt laaja käyttäjäyhteisö, jonka tuottamia ohjeita on yksinkertaista seurata.

[9] Model-View-Controller (MVC)

4.2 Embed PHP-kirjasto

Tämän opinnäytetyön tavoitteena on löytää metodi mediasisällön upottamiseen, joten Oscar Oteron kirjoittama PHP-kirjasto soveltuu tähän tavoitteeseen erittäin mainiosti ja käytännössä lähes kokonaan ilman suurempia muutoksia. Oteron PHP-kirjasto perustuu oEmbed- ja Open graph -protokollien hyödyntämiseen noudettaessa meta-tietoja URL-osoitteen perusteella (Otero 2017).

Kirjasto on liitettävissä kaikkiin PHP-sovelluksiin, jotka käyttävät vähintään version 5.3 PHP:tä. Kirjaston voi helpoiten liittää sovellukseen käyttämällä Composer-riippuvuuksienhallintasovellusta, vaikka Embed-kirjasto ei muista kirjastoista olekaan riippuvainen. Composer on asennettavissa Linux- ja Windows-ympäristöihin.

Composer luo riippuvuuksia varten hakemiston /vendor ja PHP-skriptin, joka liittää kaikki tarvittavat kirjastot sovellukseen. Composer tarvitsee kehitettävää sovellusta varten asetustiedoston composer.json, jonka tietojen perusteella riippuvuudet lisätään projektiin. Riippuvuudet lisätään Koodiesimerkin 4 mukaisesti “composer require”-komennolla, ja ne ovat määriteltyinä Koodiesimerkin 5 mukaisesti “require”-avaimen alle composer.json -tiedostossa (Composer Basic Usage 2017).

```
$ composer require embed/embed
```

Koodiesimerkki 4. Composer komennosta php-kirjaston riippuvuuksien asentamiseksi sovellukseen

```
//JSON-data
...
{
    "require": {
        "php": ">=5.6.4",
        "embed/embed": "^3.0",
    },
    "autoload": {
        "classmap": [
            "database"
        ],
        "psr-4": {
            "App\\": "app/"
```

```
        }
    }
}
...
```

Koodiesimerkki 5. Composer asennustiedosto riippuvuuksien hallintaa varten

Tätä Embed-kirjastoa hyödynnetään sovelluksessa hieman alkuperäisestä tarkoituksesta poiketen pelkästään palvelin-tasolla. Sovellukselle tehtiin käyttäjän näkymään lomake, johon käyttäjä syöttää halutun median URL-osoiteen, ja se tarkistetaan palvelimella PHP-funktiolla. Näkymään palautetaan osoitteen takaa noudetut metatiedot. Metatietoja ei palauteta kokonaisuudessaan, vaan sieltä poimitaan vain sovellukselle olennaisimpia tietoja Koodiesimerkin 6 osoittamalla tavalla.

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Embed\Embed;

class ServiceController extends Controller
{
  /**
  *  Get oembed data
  *  @return \Illuminate\Http\JsonResponse
  */
  public function embed(Request $request)
    {
        $info = Embed::create($request->url);
        $data = [
                'provider' => $info->providerName,
                'providerIcon' => $info->providerIcon,
                'url' => $info->url, //alkuperäinen osoite
                'title' => $info->title,
                'description' => $info->description,
                'type' => $info->type,
                'image' => $info->image, // kuvan osoite
                'code' => $info->code,
                'tags' => $info->tags,
            ];
        return response()->json($data);
    }
}
```

Koodiesimerkki 6. Sovelluksen PHP-funktio URL-osoitteen metatietojen hakua varten

Koodiesimerkissä 6 sovelluksen funktio palauttaa näkymälle metatietoja, jotka sisältävät mm. kuvan todellisen osoitteen ($info->image). Tämä osoite

upotetaan näkymän <img>-tagiin Koodiesimerkin 7 mukaisesti javascriptiä käyttäen, jotta käyttäjällä on mahdollisuus tarkistaa tiedot ennen varsinaista tallennusta tietokantaan. Kuvan osoite tallennetaan sen jälkeen tietokantaan “esikatselukuvan” linkkinä yhdessä muiden referenssiin liittyvien tietojen kanssa.

```
//javascript
function testURL(){
    $("#spinner").show();
    $("#ref-img").empty();
    var a = $("#url").val();
    getdata(a);
}
function getdata(x){
    var qr = "{{ url('/service/embed')}}?url="+encodeURIComponent(x);
    $.getJSON(qr).done(function(data){
        if(data['type'] == 'Image'){
            $("#spinner").hide();
            $("#ref-img").append(
                '<img class="image fit" src="'+data['image']+'">'
                );
            $("#thumb").attr("value",data['image']);
            $("#ref_desc").val(data['description']);
            $("#type option[value=Image]").attr('selected','selected');
            $("#submitNew").removeClass('disabled');
        }else{
            $("#spinner").hide();
            $("#ref-img").append(data['code']);
            $('iframe').removeAttr( "width" ).removeAttr( "height" );
            $("#ref-img").append(
                '<img class="image fit" src="'+data['image']+'">'
                );
            $("#thumb").attr("value",data['image']);
            $("#ref_desc").val(data['description']);
            $("#type option[value=Video]").attr('selected','selected');
            $("#submitNew").removeClass('disabled');
        }
    }).fail(function( jqxhr, textStatus, error ) {
        var err = textStatus + ", " + error;
        console.log( "Request Failed: " + err );
        $("#spinner").hide();
        $("#ref-img").append('<img class="image fit" src="{{
url('/images/Broken_Link.jpg')}}">');
        $("#thumb").attr("value","{{ url('/images/Broken_Link.jpg')}}");
    });
}
```

Koodiesimerkki 7. Javascript-funktio, joka kutsuu sovelluksen ServiceController-funktiota ja upottaa vastauksen tiedot dokumenttiin

4.3 FancyBox javascript-kirjasto

FancyBox on javascript-kirjasto, joka toimii samoin periaattein kuin aikaisemmin mainittu Oteron Embed PHP-kirjasto mutta suoraan loppukäyttäjän selaimella, ja se pohjautuu alunperin Lokesh Dhakarin luomaan avoimen lähdekoodin Lightbox-kirjastoon (Dhakar 2016). FancyBox-kirjasto noudattaa samaa median esittämisen periaatetta kuin kirjasto, johon se pohjautuu, ja sitä käytetään kuvien, videoiden tai minkä tahansa html-sisällön esittämiseen verkkosivulla, mutta ulkoasultaan ja toiminnoiltaan alkuperäisestä poiketen (FancyBox 2017).

FancyBox-kirjasto tukee myös alkuperäisestä Lightbox-kirjastosta poiketen oEmbed-protokollan media-lähteitä suoraan (FancyBox 2017), joten se on hankkeen sovelluksen kannalta sopivampi vaihtoehto.

5 TOTEUTUS

5.1 Toteutuksen suunnittelu

Opinnäytetyön idea muodostui varsinaisen sovelluksen kehityskaaren ollessa jo toteutusvaiheessa. Sovelluksen runko oli jo valmiina, ja siihen tarvittiin parempi tapa liittää mediaa ilman ylimääräistä tallennustilaa palvelimelta.

Suunnittelu aloitettiin etsimällä ratkaisua, joka pystyisi syöttämään tietokannan tietoja PHP:llä muodostettuun käyttäjän näkymään niin, että varsinaista sivua ei olisi tarvetta ladata uudelleen. Javascript-kirjastolla tämä onnistuisi. Tarvittiin siis sopiva kirjasto noutamaan tietoa ulkopuolisen palveluntarjoajan palvelimelta, ja sopivaksi kirjastoksi määrittyi FancyBox-kirjasto. Muitakin vaihtoehtoja oli tarkastelun alla, mutta ne eivät täyttäneet tarvittavia ominaisuuksia. Tosin saatavilla ei edes ollut kuin muutama vaihtoehto, ja FancyBox oli näistä muokattavuudeltaan monipuolisin ja sen sisältämien palveluiden osalta myös kattavin (FancyBox 2017).

Testattaessa FancyBox-kirjaston soveltuvuutta törmättiin uuteen ongelmaan. Kirjasto upottaa yksitellen kaiken median HTML-dokumenttiin siitä huolimatta, onko upotettava media näkyvillä käyttäjälle vai ei, ja se aiheuttaa paljon turhaa liikennettä käyttäjän selaimelle. Tietojen lataus tapahtuu taustalla, eli käyttäjä ei sitä varsinaisesti huomaa, mutta jos otetaan huomioon latausten arvioitu määrä sovelluksen ollessa aktiivinen, saattaa ladattavan median määrä nousta liian korkeaksi. Tämä vaikuttaa huomattavasti käyttäjäkokemukseen näkyen esimerkiksi käytettävyyden selvänä hidastumisena.

Ratkaisuna tähän uuteen ongelmaan tarvittiin palvelimen puolelle toiminto, joka noutaa tarvittavan median, poimii siitä olennaiset osat ja palauttaa vain ne näkymään. Sovellukseen tarvittiin siis samanlaisilla ominaisuuksilla oleva kirjasto kuin aikaisemmin mainittu FancyBox, mutta PHP:llä kirjoitettuna, koska sovellus käyttää PHP:tä pääasiallisena ohjelmointikielenä.

Vastaavia kirjastoja PHP:lle oli myös verrattain vähän, ja niistä käyttökelpoisimmaksi valikoitui Oscar Oteron Embed-kirjasto. Kirjasto on lähes FancyBox-kirjaston kaltainen toiminnaltaan, mutta PHP-ympäristöön kirjoitettu. Tutkiessani Embed-kirjaston rakennetta totesin voivani poimia kyselyyn vastanneen embed-palvelimen tiedoista vain tarvittavat elementit.

Suunnitelmaksi muodostui käyttää Embed-kirjastoa noutamaan valittu tietojoukko ja FancyBox-kirjastoa sulauttamaan se näkymään käyttäjän selaimella. Näin saatiin aikaan joustava ja nopea ratkaisun median sulauttamiseen.

5.2 Ympäristön valmistelu

Lähdin toteuttamaan ratkaisua Linux-alustalla puhtaasti oman kokemuksen perusteella, ja oman kokemukseni mukaan Linux-ympäristö on helpommin hallittavissa verrattuna Windows-ympäristöön. Molemmissa ympäristöissä olen harjoitellut sovelluksen kehittämistä muutamia vuosia, ja linux on osoittautunut luotettavammaksi kumppaniksi. Lisäksi sovelluskehityksen kaipaamat työkalut istuvat paremmin linux-ympäristöön.

Sovelluskehitystä varten linux-ympäristöön asennettiin perinteinen LAMP-stack eli Apache2, MySQL ja PHP. Koodiesimerkissä 8 on esitettynä yhden rivin komento LAMP-stackin kaikkien tarvittavien sovellusten ja kirjastojen asennukseen.

```
$ sudo apt-get -y install mysql-server mysql-client apache2 php7.0
libapache2-mod-php7.0 php7.0-mysql php7.0-curl php7.0-gd php7.0-intl
php-pear php-imagick php7.0-imap php7.0-mcrypt php-memcache  php7.0-pspell
php7.0-recode php7.0-sqlite3 php7.0-tidy php7.0-xmlrpc php7.0-xsl
php7.0-mbstring php-gettext
```

Koodiesimerkki 8. Yhden rivin lamp-stack asennuskomento

Lisäksi ympäristöön asennettiin Composer, joka hallitsee PHP-sovellusten riippuvuuksia, ja Laravel asennettiin ympäristöön Composerin create-project komennolla, joka luo Laravel-sovelluksen hakemistorakenteen työhakemistoon ja asentaa kaikki tarvittavat riippuvuudet. Komento lataa ja asentaa työhakemistoon uusimman Laravel-julkaisuversion, joten komento asentaa version 5.4 tätä työtä kirjoitettaessa.

```
$ composer create-project --prefer-dist laravel/laravel artcanvas
```

Koodiesimerkki 9. Composer komento, jossa ”artcanvas” on sovelluksen työhakemisto

Laravelin käyttö ei varsinaisesti tarvitse node-ympäristöä, mutta sovelluksen kehityksen automatisointiin tarvittavat työkalut käyttävät pääasiassa sitä, joten se asennettiin myös kehitysympäristöön. Asennus suoritettiin komentokehotteella lataamalla node.js lähteestä cURL-komennolla ja asentamalla se globaalisti käytettäväksi.

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install -y nodejs
$ sudo apt-get install -y build-essential //
```

Koodiesimerkki 10. Node-ympäristön asennuskomennot

Laravel toimitetaan npm-pakettitietojen kanssa, joten javascript-kirjastojen liittäminen sovellukseen on helppoa. Asennus suoritetaan käyttämällä Koodiesimerkissä 11 esiteltyä npm-komentoa, joka asentaa uusimman jakelun mukaisen fancyBox-kirjaston /vendor-hakemistoon. Komennossa --save lipulla tallennetaan kirjastoriippuvuus npm-pakettitiedoston jatkeeksi myöhempää tarvetta varten.

```
$ npm install @fancyapps/fancybox --save
```

Koodiesimerkki 11. Npm-komennolla asennettava fancyBox-kirjasto

Sovellus käyttää myös Oteron PHP-pakettia embed\embed, joka asennetaan Composerin avustuksella. Komentona käytetään Koodiesimerkin 12 mukaista composer-komentoa. Komento lataa ja asentaa embed-kirjaston /vendor hakemistoon.

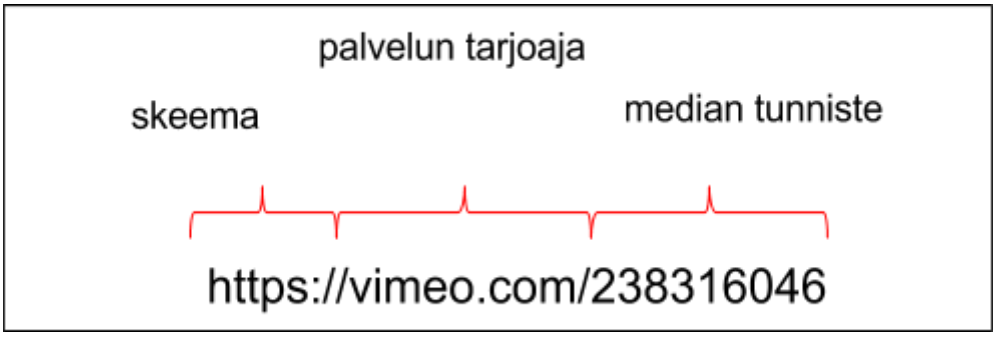
```
$ composer require embed\embed
```

Koodiesimerkki 12. composer-komennosta php-pakettien asennuksessa

5.3 Rakenteen toteutus

5.3.1 Median lisäyksen näkymä

Sovellukseen luotiin rekisteröityneelle käyttäjälle uusi näkymä media-referenssien lisäykseen. Näkymä on perustoteutukseltaan tavallinen lomake, johon käyttäjä syöttää halutun median URL-osoitteen. Osoitteen kelpoisuus tarkistetaan käyttäjän toimesta klikkaamalla “tarkista”-painiketta, joka lähettää URL-osoitteen AJAX-metodilla jQuery-kirjaston getJSON-funktiolla sovelluksen palvelimelle, jossa se ohjautuu edelleen ServiceController -funktiolle.



Kuva 2. Vimeo-palvelun URL-osoite medialle

Kuvassa 2 on esitettynä median URL-osoite Vimeo-palvelusta. Osoite on purettuna osiin, jotta voidaan erotella palvelun tarjoajan nimi ja median tunniste osoitteesta.

ServiceController-funktio ottaa kyselyn vastaan ja kutsuu edelleen embed-kirjastoa purkamaan osoitteesta palvelun tarjoajan nimen. Mikäli nimi löytyy kirjaston sisäisestä luettelosta, kutsuu funktio nimen mukaista palveluntarjoajan oembed-palvelinta ja lähettää edelleen kyselynä saapuneen alkuperäisen URL-osoitteen sinne.

Palveluntarjoajan oembed-palvelin vastaa ServiceControllerin funktiolle, joka purkaa palautuneet tiedot. Tiedoista poimitaan aikaisemmin määritellyt elementit ja ne palautetaan edelleen alkuperäisen kyselyn tehneelle AJAX-funktiolle, joka viimein upottaa ne käyttäjän näkymään esikatselukuvaksi. ServiceController-funktion tarkempi toiminta esitellään visuaalisesti Liitteessä 3.

Esikatselukuva, joka on staattinen kuva videosta tai mahdollisesti kooltaan pienempi kuva kuva-tyypin mediasta, upotetaan näkymään javascriptillä, jotta saadaan se heti käyttäjälle näkyviin. Esikatselukuvan alle liitetään alkuperäisen URL-osoitteen mukaisesti median esikatselu joko videona tai kuvana, tyypistä riippuen, kuten Koodiesimerkissä 7 on aikaisemmin esitetty.

Käyttäjä näkee suoraan esikatselusta, vastaako palautuneiden tietojen media aiottua, ja mikäli esikatselukuva täsmää, voi käyttäjä tallentaa median osoitteen tietokantaan muiden mediaan liittyvien tietojen kanssa. Jos URL-osoite on puutteellinen tai sitä ei voida jostain syystä noutaa, käyttäjä näkee rikkinäisestä linkistä ilmoittavan kuvan ja tekstinä saman huomautuksen.

Tähän näkymään liittyi jonkin verran käyttäjästä riippuvia ongelmia, joiden kiertäminen oli verrattain hankalaa. Ongelmat ilmenivät ensimmäisen opiskelijoilla toteutetun koekäytön yhteydessä. Ongelmia esiintyi tilanteissa, joissa käyttäjä yritti liittää esimerkiksi avoimesta lähteestä kokonaista kuvagalleriaa tai suljetun profiilin takana olevaa mediaa.

Kuvagalleriassa on liikaa materiaalia, jotta sitä voitaisiin upottaa järkevästi. Embed-kirjasto kykenee poimimaan lähteestä yhden kuvan, joka yleensä on ensimmäinen tai laadultaan paras lähteeseen kirjattu, joten gallerian esikatselukuvaksi muodostui juuri tämä nimenomainen kuva, vaikka se ei yleensä ollutkaan käyttäjän aikomaa materiaalia.

Ratkaisuna galleria-ongelmaan olisi ollut lisätä funktio tarkistamaan, löytyykö lähteestä liikaa materiaalia, mutta ajan puitteissa sitä ei ehditty luomaan.

Ongelma siis yksinkertaisesti kierrettiin ilmaisemalla käyttäjälle, että galleria ei ole soveltuva lähde upotettavaksi.

Toinen ongelma, jota ei pystytty suoranaisesti korjaamaan, oli suljetun lähteen takana oleva median lähde. Käyttäjä kyllä näkee esimerkiksi oman Instagram-profiilinsa kaikki kuvat, mutta niistä voidaan upottaa vain ne, jotka ovat avoimesti verkossa saatavilla. Yksityisiä tai ryhmälle julkisia kuvia ei voida noutaa ollenkaan. Tämä virhe kuitenkin näkyy käyttäjälle heti rikkinäisenä linkkinä esikatselukuvassa ja huomautuksena tekstissä.

Suljetun lähteen ongelmiin on kuitenkin olemassa ratkaisuja, vaikka niitä ei pystytty yrityksestä huolimatta toteuttamaan. Ratkaisu vaatisi esimerkiksi juuri Instagram-palvelun kohdalla sovelluksen rekisteröintiä Facebook-developer-tilille, ja sovellukselle pitäisi myös luoda Facebook-app-tunnus, jolle loppukäyttäjän tulisi antaa oikeudet Instagram-tilien hallintaan. Ratkaisu on mahdollinen mutta melko mutkikas ja vaatisi jatkuvaa ylläpitoa, joten sekin lopulta kierrettiin.

Lopullinen ratkaisu ongelmaan, joka kattaisi kaikki suljetut lähteet, oli yksinkertaisesti ilmoittaa käyttäjälle, että kaiken noudettavan median on oltava vapaasti verkossa saatavilla.

5.3.2 Julkinen näkymä

Varsinainen median upotus laajemmassa mittakaavassa tapahtuu sovelluksen avoimella puolella referenssi- ja profiili-näkymissä. Oletuksena on, että sovelluksella on jonkin verran käyttäjiä ja käyttäjillä profiiliin rekisteröitynä upotettavaa mediaa. Referenssi-näkymässä on kaikkien käyttäjien kaikki referenssi-media ja profiili-näkymässä yksittäisen käyttäjän kaikki referenssi-media.

Upotus käyttää sovelluksen tietokannan tietoja median esikatselukuvasta ja aiemmin mainittua FancyBox-kirjastoa. Näkymään liitetään tietokannasta kuvan

tiedot <img>-tagiin <a>-tagin sisälle ja micro-datana alkuperäinen URL-osoite. Tällä metodilla toteutettuna jokaisesta sovelluksen referenssistä on saatavilla pieni esikatselukuva heti näkymän latautuessa, ja myöhemmin kuvaa klikkaamalla aktivoituu koodi, joka upottaa dokumenttiin omana elementtinä kyseisen kuvan alkuperäisen URL-osoitteen perusteella suuremman kuvan tai vaihtoehtoisesti videon kuten Koodiesimerkissä 13.

```
// @blade markup
// php
/** view references.blade.php
* micro-data tietokannasta: $refs
* integer data-user -> käyttäjän tunniste
* integer data-likes -> referenssin tykkäykset
* integer data-refid -> referenssin tunniste
* data-fancybox -> fancyBox-kirjaston tunniste
* string data-caption -> referenssin kuvausteksti
*/
...

@if($ref->type == 'Video') //referenssin tyyppinä video

  <a class="icon" href="{{$ref->url}}"
      data-user="{{$ref->user_id}}"
      data-likes="{{$ref->likes}}"
      data-refid="{{$ref->id}}"
      data-fancybox
      data-caption="{{$ref->ref_desc}}">
   <img class="image" src="{{$ref->thumb }}" data-tooltip="{{$ref->name}}">
  </a>

@else //referenssin tyyppinä kuva

  <a class="icon" href="{{$ref->thumb }}"
      data-user="{{$ref->user_id}}"
      data-likes="{{$ref->likes}}"
      data-refid="{{$ref->id}}"
      data-fancybox
      data-caption="{{$ref->ref_desc}}">
   <img class="image" src="{{$ref->thumb }}" data-tooltip="{{$ref->name}}">
  </a>

@endif
...
```

Koodiesimerkki 13. Osa ReferenceController-funktion palauttamasta referenssi-näkymästä

Koodiesimerkissä 13 median tiedot on liitetty käyttäjälle palautettavaan näkymään Laravelissa käytetyllä blade-merkkauksella. Blade-merkkaus toimii PHP-ohjeina varsinaiselle sivun generointi-funktiolle, joka muodostaa käyttäjälle

palautettavan HTML-dokumentin. Esimerkissä FancyBox-kirjasto on kytketty <a>-tagiin niin, että median alkuperäinen osoite on href-määritteessä ja <a>-tagin sisällä on median esikatselukuva <img>-tagissä, joka latautuu käyttäjälle heti nähtäväksi. Klikkaamalla <a>-tagia, joka on merkitty “data-fancybox” määritteellä, aktivoituu upottamisen suorittava koodi.

```
//javascript

$( '[data-fancybox]' ).fancybox({
  caption : function( instance, item ) {

    var caption, link;

    caption = $(this).data('caption');

    link = '<a class="button" href="{{url('/profile')}}/' +
$(this).data('user') + '" style="text-decoration:none;">Show
Profile</a> <a class="button special" href="javascript:void(0);"
onclick="likebutton('+ $(this).data('refid') +')"
style="text-decoration:none;">Like me!</a><br />';

      return link + (caption ? caption + '<br />' : '');
  }
});
```

Koodiesimerkki 14. FancyBox-funktio medialle

Koodiesimerkissä 14 funktiolle määritellään jQueryllä elementtien tunniste data-fancybox, joihin koodi vaikuttaa, ja kutsutaan varsinaista FancyBox-kirjaston alustus-metodia fancybox. Fancybox-metodi rekisteröi sivun latautuessa jokaiselle valitulle elementille tapahtuman (onclick) käsittelijän ja jää odottamaan tapahtumaa. Käyttäjän klikatessa linkkiä koodi aktivoituu ja poimii klikatun elementin datamääritteistä tiedot.

Fancybox-metodin caption-muuttuja saa funktion, joka hyväksyy kaksi parametria. Funktion sisäiselle muuttujalle caption asetetaan arvoksi klikatun elementin data-caption määritteen arvo ja muuttujalle link rakennetaan painikkeet käyttäjän profiiliin ja median tykkäyksille. Funktio palauttaa fancybox-metodille caption-muuttujan, ja fancybox-metodi upottaa sen samalla kun aktivoi sisäisen upotusmetodin käyttäjän näkymässä.

Upottaminen tapahtuu siis yksi osoite kerrallaan ja vain käyttäjän toimien perusteella. Mediaa ei ladata etukäteen, vaan vasta kun käyttäjä sitä pyytää. Näin toimittaessa noudettavan tiedon määrä pysyy kohtuullisena. Tähän rakennetyyppiin päädyttiin, koska havaittiin pelkällä javascriptillä suoritetun upottamisen kuluttavan kohtuuttoman paljon selaimen resursseja, varsinkin näkymässä, jossa on paljon upotettavaa tietoa

5.4 Toteutuksen ongelmia

Toteutuksen aikana törmättiin moniin ongelmiin, joista osa oli ratkaistavissa vaihtoehtoisella metodilla ja osa taas jouduttiin kiertämään koodauksen kannalta hyvinkin epäsovinnaisella tavalla. Epäsovinnaisella tarkoitetaan tässä tapauksessa käytäntöä, jossa koodia ei varsinaisesti korjata tai muuteta mitenkään, vaan toiminto joko poistetaan kokonaan tai odotetaan käyttäjältä kykyä käyttää sitä oletuksen mukaisella tavalla.

Ongelmia, joita ei pystytty korjaamaan, ilmeni mm. median laadussa ja lähteissä, koska oletuksena oli, että käyttäjä voi vapaasti valita median lähteen. Nämä ongelmat jouduttiin kiertämään ilmoittamalla käyttäjälle median lähteen vapaasta saatavuudesta verkossa. Ongelmiin on olemassa ihan oikeita ratkaisuja, mutta resurssien puitteissa niitä ei pystytty toteuttamaan.

Korjatuista ongelmista isoimmaksi muodostui FancyBox-kirjaston tyylitiedoston ja sovelluksen oman tyylitiedoston yhteensopivuus moneltakin osin. FancyBox esimerkiksi upottaa kaiken median z-indeksin tasolle 99999, jota myös sovelluksessa käytettiin ja joka ehti aiheuttamaan runsaasti harmaita hiuksia ennen kuin ratkaisu löytyi. Ongelman huomasi, kun media oli jo upotettuna dokumenttiin ja käyttäjä klikkasi elementin ulkopuolelle tai painikkeita, eikä mitään toimintaa ollut havaittavissa.

Ratkaisuna z-indeksin aiheuttamaan ongelmaan oli korjata sovelluksen tyylitiedosto käyttämään eri tasoja (99990), ja siirtää FancyBox-kirjaston tyylitiedostossa upotettu media yhtä tasoa alemmaksi niin, että median kuvaus

ja painikkeet jäivät paikoilleen tasolle 99999. Näin median päällä olevat painikkeet olivat klikattavissa ja toiminnallisuus havaittavissa, mutta tämä aiheutti sen, että FancyBox-kirjasto piti manuaalisesti liittää sovellukseen eikä sitä voisi jatkossa päivittää ollenkaan. Muutos on rakenteellisesti pieni, mutta sovelluksen jatkokehityksen kannalta melko merkittävä.

6 POHDINTA

6.1 Työn yhteenveto

Median upottaminen ei enää ole varsinaisesti uutta tekniikkaa, mutta tässä opinnäytetyössä toteutetussa mittakaavassa sitä harvemmin käytetään. Normaali tapa olisi tallentaa media saataville sovelluksen omalle palvelimelle, vaikka se noudettaisiinkin ulkopuolisesta lähteestä. TaideART-hankkeen sovelluksen kohdalla menetelmää ei ollut mahdollista toteuttaa, koska palvelin-tilan piti pysyä mahdollisimman pienenä. Siksi päädyin dynaamisempaan ratkaisuun, jossa media noudetaan alkuperäisestä lähteestä ja upotetaan käyttäjän näkymään ilman median tallennusta palvelimelle.

Open Graph- ja oEmbed-protokollien käyttö sivustoilla ja palveluissa on omien havaintojeni perusteella hyvin kirjavaa. Palveluilla saattaa olla useita embed-palvelimen osoitteita, ja lisäksi ne voivat muuttua varsin nopeasti. Tämä vaatii aktiivista kirjaston ylläpitoa, jotta ne pysyisivät muutosten matkassa. Esimerkiksi tämän opinnäytetyön suorituksen aikana mm. Vimeo-palvelun embed-palvelimiin tuli muutos, joka esti salaamattomasta osoitteesta lähetetyn kyselyn käsittelyn. Muutos ei kokonaisuutena ollut ennalta arvaamaton, jos otetaan huomioon web-kehityksen nykysuuntaus parempaan turvallisuuteen, mutta sovelluksen kannalta Vimeo-palvelu oli hetken saavuttamattomissa.

6.2 Työn tulosten arviointi

Tämä opinnäytetyö oli osa suurempaa sovelluskehityksen projektia, ja työn suorituksen aikana törmäsin moniin ongelmiin. Monet ongelmista eivät suoranaisesti liittyneet opinnäytetyötä koskevaan osaan, vaan ne vaikuttivat pääasiassa koko sovelluksen kehitykseen. Keskeisimmäksi ongelmien kokonaisuudeksi muodostui jatkuva uusien muutosten tulva, jota resurssien puitteissa oli hankala hallita. Muutospyynnöt tulivat nopeammin kuin mitä niitä ehti edes kirjaamaan. Tästä syystä sovelluksen toteutus jäi lopulta hieman pirstaleiseksi ja epävakaaksi.

Itse opinnäytetyön osuus onnistui kohtuullisen hyvin, ja sovellukseen mediaa upottava elementti toimi lopulta odotetusti. Joitakin suunnittelu- ja rakennevirheitä kuitenkin jäi odottamaan sovelluksen seuraavaa kehittäjää, koska TaideART-hanke jatkuu vielä oman osuuteni jälkeen ja voidaan olettaa sovelluskehityksen jatkuvan.

6.3 Suunnitteluprosessin arviointi

Suunnitteluprosessin voidaan katsoa menneen kokonaisuutena puutteellisesti, koska suunnitelmassa ei pysytty. Projektin sovelluksen muut elementit nousivat tärkeämmiksi, ja suuri muutosten määrä vaikutti ratkaisevasti sovelluksen kehityskaaren kokonaisuuteen. Lisäksi monelle kehitysprojektille tyypillinen resurssien puute aiheutti suunnitteluun lievään epätarkkuutta ja siirtymistä ideasta suoraan toteutukseen.

Kuten sovelluksen toteutusvaiheessa kävi ilmi, ei kaikkien opinnäytetyössä läpikäytyjen suunnitelmien toteutus ole aivan helppoa, varsinkin kun parhaat käytännöt saattavat vaihdella palvelusta riippuen. Monet aiemmin epäselvyyttä aiheuttaneet kohdat ovat kuitenkin nyt selkeytyneet, ja dynaaminen upotusmetodin toteutusprosessi on kokonaisuudessaan selvillä.

LÄHTEET

Composer Basic Usage 2017. Composer dokumentaatio. Viitattu 30.10.2017 https://getcomposer.org/doc/01-basic-usage.md.

Dhakar, L. 2016. Lightbox dokumentaatio. Viitattu 30.10.2017 http://lokeshdhakar.com/projects/lightbox2/.

Embedly 2017. Dokumentaatio. Viitattu 30.10.2017 http://embed.ly/providers.

Facebook for Developers 2017a. Docs: Access Tokens. Viitattu 30.10.2017 https://developers.facebook.com/docs/facebook-login/access-tokens/.

Facebook for Developers 2017b. Docs: Open Graph Stories. Viitattu 30.10.2017 https://developers.facebook.com/docs/sharing/opengraph.

Fang, Z. 2010. How To: Get Started with the Open Graph. Viitattu 30.10.2017 https://developers.facebook.com/blog/post/564/.

FancyBox 2017. Dokumentaatio. Viitattu 30.10.2017 http://fancyapps.com/fancybox/3/docs/.

Fishkin, R.2015 Beginners Guide to SEO. Viitattu 30.10.2017 https://moz.com/beginners-guide-to-seo.

IBM Knowledge Center 2014. The Client/Server Model. Viitattu 30.10.2017 https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.ieak500/ieak511.htm.

Laravel 2017. Dokumentaatio. Viitattu 30.10.2017 https://laravel.com/docs/5.4.

Mozilla Developer Network 2017. Dokumentaatio. HTML element reference, iframe. Viitattu 30.10.2017 https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe.

oEmbed 2017. Dokumentaatio. Viitattu 30.10.2017 http://oembed.com/.

Otero, O. 2017. Embed-library. Viitattu 30.10.2017 https://github.com/oscarotero/Embed.

Segue Technologies 2013. What is Ajax and Where is it Used in Technology? Viitattu 30.10.2017 https://www.seguetech.com/ajax-technology/.

TaideART-hanke 2017.Hankkeen sivusto. Viitattu 30.10.2017 http://www.taidearthanke.fi.

The Open Graph protocol 2014. Dokumentaatio. Viitattu 30.10.2017 http://ogp.me/.

The TCP/IP Guide 2005. Verkkodokumentti. HTTP Operational Model and Client/Server Communication. Viitattu 30.10.2017 http://www.tcpipguide.com/free/t_HTTPOperationalModelandClientServerCommunication.htm.

VAHTI 4/2010 2010. Sosiaalisen median tietoturvaohje. Valtiovarainministeriö. Viitattu 30.10.2017 https://www.vahtiohje.fi/c/document_library/get_file?uuid=8b44c0bf-cff3-4e6c-a587-eea58a9e3ad7&groupId=10128.

World Wide Web Consortium, W3C 2014. RDF 1.1 Concepts and Abstract Syntax. Viitattu 30.10.2017 https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

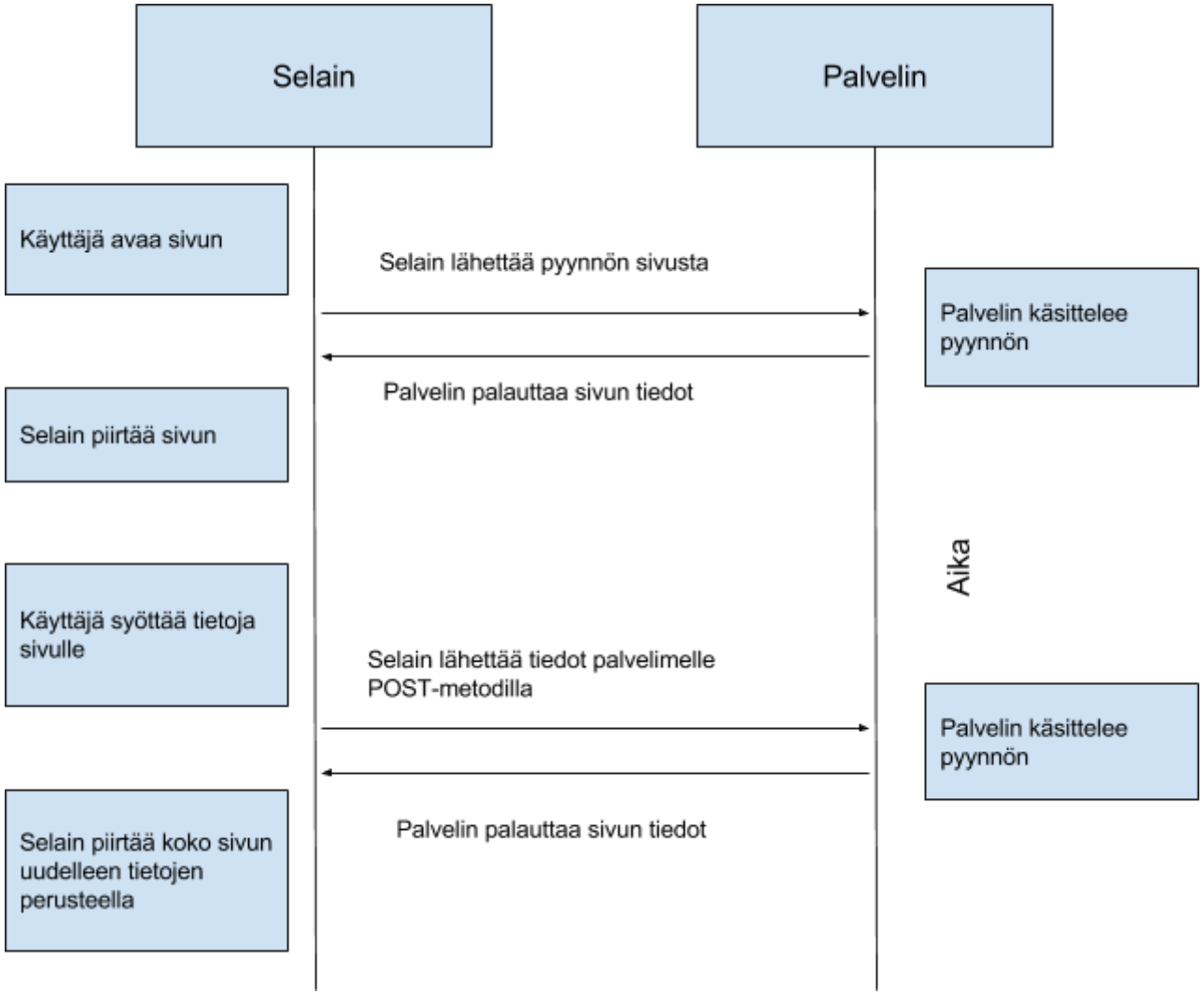
W3Techs. 2017. Usage of server-side programming languages for websites. Viitattu 30.10.2017 https://w3techs.com/technologies/overview/programming_language/all.

YouTube 2016. Dokumentaatio. IFrame Player API. https://developers.google.com/youtube/player_parameters.

LIITTEET

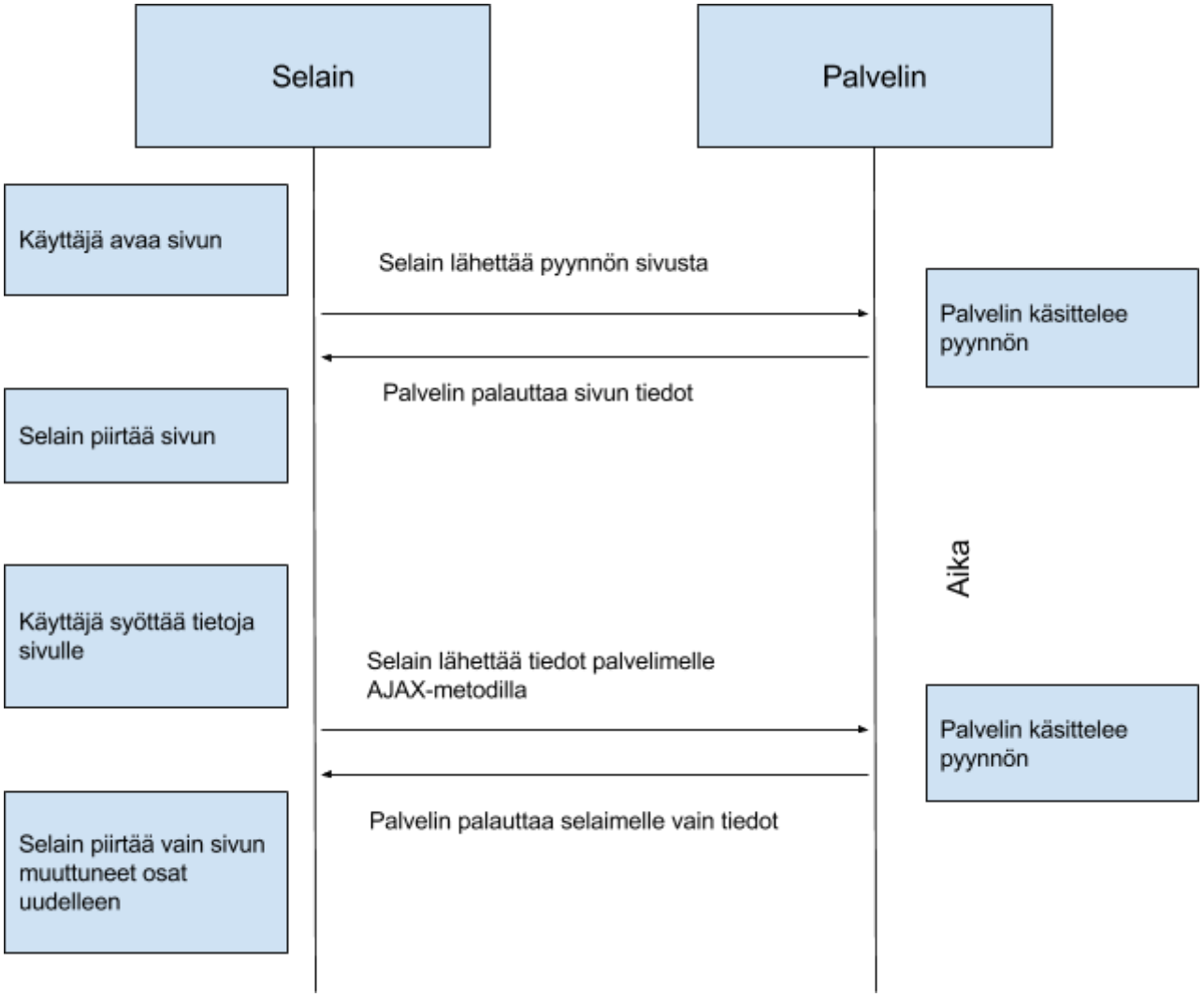
Liite 1

Selaimen ja palvelimen välinen kommunikointi POST-metodilla



Liite 2

Selaimen ja palvelimen välinen kommunikointi AJAX-metodilla



Liite 3

ServiceController-funktion kuvaus