Jarkko Vilén

Web-pohjainen presentaatiotyökalu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

1. kesäkuuta 2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Jarkko Vilén Web-pohjainen presentaatiotyökalu 31 sivua 1. kesäkuuta 2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Ilpo Kuivanen, Lehtori Mikko Peltola, Projektipäällikkö

Insinöörityön tarkoituksena oli tehdä Sujuwa Oy:lle sovellus presentaatioiden eli esitelmien esittämistä ja jakamista varten. Sovelluksen esitelmät koostuivat videoista ja diaesityksistä, joita sovelluksessa oli tarkoitus yhtaikaa katsella. Sovellus sisälsi vaaditut ominaisuudet ja jätti mahdollisuuden sovelluksen jatkokehitykselle.

Sovellus päätettiin kehittää käyttäen palvelinpuolella PHP:ta ja Symfony-ohjelmistokehystä. Selainohjelman JavaScript-koodi kirjoitettiin käyttäen Backbone.js-kirjastoa ja se kirjoitettiin moduuleiksi, joiden käyttäminen mahdollistetaan Webpack-ohjelmistolla.

Insinöörityön raportti käsittelee insinöörityönä tehdyn sovelluksen käyttämiä teknologioita yleisesti. Tämän jälkeen kerrotaan sovelluksen toteutuksesta ja siitä, miten esiteltyjä teknologioita sovelluksessa käytetään.

Avainsanat	PHP, Symfony, Backbone.js, Webpack

Author	Jarkko Vilén
Title	Web-based Presentation Tool
Number of Pages	31 pages
Date	Thursday 1st June, 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Ilpo Kuivanen, Senior Lecturer Mikko Peltola, Senior Project Manager

The goal of this thesis was to develop a web-based application that allows sharing and viewing user created presentations. These presentations can contain video and slideshow files. The application allows the media files to be synchronized and viewed simultaneously.

It was decided to build the application with PHP using the Symfony framework. The JavaScript of the client application was written using Backbone.js library to make the code more structured. JavaScript components were written as modules that were combined using the Webpack module bundler.

This thesis report starts with the introduction and explaining the goals and major features of the application and after that, the major backend and fronted technologies used by the application are introduced. Then, it is explained how the technologies are used in the application.

Keywords	PHP, Symfony, Backbone.js, Webpack

Sisällys

Lyhenteet

AJAX Asynchronous JavaScript and XML. Tekniikka, jonka avulla voidaan lähettää HTTP-pyyntöjä lataamatta sivua uudestaan.

AMD Asynchronous Module Definition. JavaScript moduulien lataustekniikka selaimille.

API Application Programming Interface eli ohjelmointirajapinta on määritelmä, jonka avulla sovellukset voivat keskustella toistensa kanssa.

CSS Cascading Style Sheets. Selainten käyttämä tyylitiedostoformaatti.

CTI Class Table Inheritance. Tekniikka, jolla voidaan kartoittaa oliot perintäsuhteineen useaan relaatiotietokanta tauluun.

DBAL Database Access Layer. Sovelluksen kerros, joka on vastuussa tietokannan kanssa kommunikoinnista.

DOM Document Object Model. Ohjelmointirajapinta, joka mahdollistsaa muun muassa HTML-dokumentin käsittelyä puurakenteena.

DQL Doctrine Query Language. SQL:n kaltainen kieli, jonka avulla kirjoittaa kyselyitä DoctrineORM:ää varten.

HTML HyperText Markup Language. Merkintäkieli web-sivustojen luomiseen.

HTTP Hypertext Transfer Protocol. Protokolla, joka toimii perustana datan välittämiseen World Wide Webissä.

JSON JavaScript Object Notation. Etenkin datan välittämiseen käytetty tiedostoformaatti, jossa data koostuu avain-arvo pareista.

ORM Object Relational Mapping. Tekniikka oliorakenteen kartoittamiseen relaatiotietokantaan.

PHP PHP: Hypertext Preprocessor. Yleinen palvelinpuolen ohjelmointikieli web-kehitykseen.

REST Representational State Transfer. Yleensä HTTP:n avulla käytettävä web-ohjelmointi arkkitehtuurityyli, jonka ohjelmat voivat internetin välityksellä käyttää toistensa resursseja.

SQL Structured Query Language. Kieli, jolla suoritetaan tietokantakyselyitä.

STI Single Table Inheritance. Tekniikka, jolla voidaan kartoittaa olioiden perimissuhteet yhteen relaatiotietokanta tauluun.

URL Uniform Resource Locator. Internet-osoite.

1 Johdanto

Insinöörityönä Sujuwa Oy:lle tein selainpohjaisen sovelluksen, jonka avulla on tarkoitus pystyä katselemaan diaesityksistä ja videoista koostuvia esitelmiä.

Järjestelmän tarkoitus on mahdollistaa esimerkiksi koulutustilaisuuden tai jonkin muun esityksen tallennus kertaalleen, jonka jälkeen sen toistaminen useasti on mahdollista. Nykyään tämä on mahdollista esimerkiksi yhdellä videolla, mutta videon ja täten esityksenkin laaduntaso voi syystä tai toisesta olla erittäin vaihtelevaa.

Tässä insinöörityön raportissa aluksi kerrotaan sovelluksen ja työn tavoitteet sekä käsitellään sovelluksen hyödyntämiä teknologioita. Tämän jälkeen kerrotaan tehdyn sovelluksen toteutuksesta ja siitä, miten erilaisia teknologioita siinä hyödynnettiin. Lopussa on vielä yhteenveto insinöörityöstä.

2 Tavoitteet

Tavoitteena oli suunnitella ja toteuttaa selaimella käytettävä sovellus, jonka avulla on mahdollista esittää järjestelmään ladattuja esityksiä. Esitykset koostuvat videosta sekä diaesityksestä. Käyttäjillä tulisi olla keino ladata nämä video- ja diaesitystiedostot järjestelmään sekä järjestelmän tulisi muuntaa tiedostot selaimessa käytettävään muotoon.

Esitysten katselu on sovelluksen pääasiallinen tarkoitus. Esitysten tuli sisältää video, diaesitys sekä ainakin navigaatio diaesitystä varten. Esitystä tulisi olla mahdollista navigoida videon tai alla olevan navigaation avulla.

Esityksiä myös tulisi olla mahdollista selata ja etsiä. Sovelluksen tulisi myös sisältää joitakin esitysten haku- ja rajausmahdollisuuksia. Esityksiin pitää pystyä liittämään kategorioita, joita myös tulisi voida käyttää hakuja rajaamaan.

Käyttäjien pitää pystyä luomaan järjestelmään esityksiä. Tätä sekä muita esityksen tietoja varten tarvittiin keino syöttää dataa sekä tiedostoja järjestelmään. Videoista tulisi hyväksyä vain selaimessa toistettavissa olevat formaatit. Ladattavat diaesitykset ovat PowerPoint-esityksiä tai niiden OpenOffice-vastineita. Näiden lisäksi esitelmälle tuli antaa perustietoja kuten esityksen nimi sekä kategoriat.

Esityksiä katseltaessa videon ja diaesityksen tulisi olla synkronoituina eli toisin sanoen käyttäjän katsoessa videota diaesityksen tulisi edetä samaa tahtia videon mukana. Tätä varten tulisi sovelluksen myös sisältää ajastustoiminto. Ajastusta sekä esityksen muokkausta varten tulisi esitysten sisältää myös hallintasivu esitykselle.

3 Teknologiat

3.1 Symfony

Symfony on palvelinpuolen ohjelmistokehys PHP:lle. Symfony 1.0 julkaistiin 2007. Vuonna 2010 julkaistiin Symfonyn versio 2.0, jossa tuli myös suuria arkkitehtuurillisia muutoksia aiempiin versioihin nähden. Symfony on kirjoitettu useiksi uudelleenkäytettäviksi komponenteiksi. Näitä komponentteja käyttävät Symfony-ohjelmistokehyksen lisäksi monet muutkin projektit kuten Drupal 8 sekä Laravel-ohjelmistokehys. Symfony on 2.1 versioista lähtien käyttänyt erillistä Composer-paketinhallintaohjelmistoa sen riippuvuuksien hallintaan ja niiden lataamiseen. Vuonna 2015 julkaistiin Symfonyn versio 3. Erona Symfony2:een on muun muassa Symfony2:n arkkitehtuurissa olleiden ongelmien korjausta sekä Symfony2:n elinkaaren aikana vanhentuneiden toimintojen poisto. [1; 2.]

Symfonyn ytimen päälle on luotu useita jakeluita. Käytetyin näistä on Symfony Standard Edition, joka sisältää Symfonyn sekä integraatiot erillisille kirjastoille, joista koostuu käytettävissä oleva kokonaisuus.

3.1.1 Arkkitehtuuri

Ainakin web-kehityksessä on nykyään yleistä, että sovellukset käyttävät useita ulkopuolisia ohjelmistokirjastoja oman toiminnallisuutensa toteuttamiseen. Symfony-ohjelmistokehys on hajautettu useisiin pienempiin komponentteihin. Nämä komponentit ovat uudelleenkäytettäviä PHP-kirjastoja, jotka sijaitsevat jokainen omassa repositorioissaan. Symfony ottaa saman rajapinnan avulla käyttöön niin omat sovelluskehyksen osat kuin kolmannen osapuolen ohjelmistokirjastot. Kaikki nämä Symfony ottaa käyttöön sen bundle-järjestelmän avulla, jossa jokaista erillistä käyttöönotettua komponenttia kutsutaan bundleksi. [3.]

Bundlen on tarkoitus sisältää kaiken tarvittavan mitä se tarvitsee toteuttaakseen oman toiminnallisuutensa. Bundle koostuu useista PHP-luokista sekä konfiguraatiotiedostoista.

Bundle myös voi sisältää omia näkymiä ja sivulla käytettäviä tiedostoja, jotka voivat olla muun muassa javascript-, tyyli- tai kuvatiedostoja. Jokainen bundle sijaitsee omassa hakemistossaan. Jokainen bundle sisältää myös oman Bundle-luokan, jonka avulla bundle ja sen sisältö rekisteröidään ohjelmistokehyksen käyttöön. Tämä bundle-luokka voi minimissään olla tyhjä luokka, joka ainoastaan perii Symfonyn yleisen abstraktin Bundle-luokan. Symfony sisältää komentoriviohjelmassaan komennon, jonka avulla on mahdollista luoda uusi bundle. [4, s. 89-91.]

Bundlet voivat olla uudelleenkäytettäviä tai sovelluskohtaisia. Uudelleenkäytettäviä bundleja on mahdollista käyttää useissa eri sovellussa. Sovelluskohtaiseen bundleen sijoitetaan osa sovelluskohtaisesta koodista. Tämä bundle sisältää usein sovelluskohtaisia reitityksiä ja reittien käsittelijöitä sekä konfiguraatiot bundlen sisäiselle toiminnalle. [4; 5, s. 89-91.]

Symfony-ohjelman ytimenä toimii AppKernel-luokka. Tässä AppKernel-luokassa määritetään ohjelman ajonaikaiset bundlet sekä rekisteröidään sovelluksen konfiguraatiotiedosto tai -tiedostot. Jokainen sovelluksen käyttämä bundle on otettava käyttöön rekisteröimällä se AppKernelin registerBundles-metodissa.

Symfony käynnistetään sen etukontrollerissa. Etukontrolleri on PHP-tiedosto, joka käynnistetään Apachen, Nginxin tai jonkin muun web-palvelimen toimesta. Etukontrollerin pääasiallinen tarkoitus on hoitaa Symfonyn AppKernelin käynnistäminen sekä pyyntö-olion välittäminen Symfonylle. Tyypillisesti etukontrollereita on Symfony-sovelluksessa ainakin kaksi ja nämä ovat app.php ja app_dev.php. Näistä ensimmäinen on tarkoitettu käytettäväksi tuotantoympäristössä ja jälkimmäinen kehitystä varten. [4, s. 23.]

Symfony sisältää ja hyödyntää konfiguraatioita paljon. Jokainen bundle voi sisältää oman konfiguraationsa. Tämä koskee niin Symfonyn omia bundleja kuin ulkoisia, kolmannen osapuolen bundleja. Symfonyn konfiguraatiot voidaan kirjoittaa käyttämällä YAML:ia, XML:ää tai PHP:ta. Joidenkin komponenttien konfigurointi voidaan myös tehdä käyttämällä luokkiin kirjoitettavia annotaatioita. [4, s. 86-87.]

Symfony käyttää sille konfiguroituja joidenkin ohjelmaosien konfiguraatioiden määrittämiseen. Parametrit on tarkoitus asettaa erikseen jokaiselle sovelluksen eri instanssille. Parametreiksi voidaan määrittää sovelluksen käyttämää arkaluontoista dataa, jota ei haluta

päätyvän sovelluksen yhteiseen repositorioon. Parametreiksi myös asetetaan dataa, joka vaihtelee jokaisen sovelluksen eri instanssin välillä. Usein parametreiksi voidaan määrittää sovelluksen käyttämä tietokantayhteys sekä erilaiset tunnukset autentikointitarkoituksiin. [4, s. 92-93, 206.]

Symfony voidaan käynnistää erilaisissa ympäristöissä. Tällä tarkoitetaan sitä, että Symfony-ohjelmaa voidaan ajaa eri konfiguraatiolla sen tarkoituksesta riippuen. Tyypillisesti Symfony-sovelluksen ympäristöt ovat tuotanto-, kehitys- ja testiympäristöt. Käytännössä tämä tapahtuu siten, että Symfonylle annetaan sen AppKernelin luomisen yhteydessä ympäristön määrittävä merkkijono. Tätä merkkijonoa käytetään muuttujana ladattavien konfiguraatiotiedostojen määrittämiseen sekä siitä riippuen voidaan ottaa eri komponentteja tai niiden osia käyttöön. [4, s. 87-88.]

3.1.2 Reititys ja kontrollerit

Kontrollerien tarkoitus on vastaanottaa Symfonylle tulleista pyynnöistä luodut pyyntö-oliot sekä palauttaa niihin vastaukset. Kontrollerit sisältävät erilaisten pyyntöjen käsittelijämetodeita, joita Symfonyssä kutsutaan actioneiksi.

Symfony sisältää reititys komponentin, jonka tehtävänä on määrittää ja ohjata Symfonylle tulleet kutsut kontrollereiden action-metodeihin.

Symfonylle konfiguroidaan säännöt siitä, millaiset pyynnöt ohjataan minkäkin kontrollerin mihinkin actioniin. Reititystä varten luodaan konfiguraatioiden avulla reittejä. Nämä voidaan määritellä joko Symfonyn tukemilla konfigurointi-tiedostoilla tai kirjoittamalla annotaatioina action-funktioiden kohdalle sen reitti. Yleensä reitityksen määrittämiseen käytetään kutsuttua polkua sekä pyynnön HTTP metodia, joista yleisimpiä ovat GET, POST, DELETE ja PUT. [4, s. 5-6.]

Symfonyn reiteille voidaan myös annotaatioiden määrittää olioiden automaattinen hakeminen tietokannasta. Tämä tarkoittaa sitä, että action-funktiossa sivu osoitteen sisältävän muuttujan sijaan suoritetaan tietokanta kysely ja vaihdetaan action-funktion parametri tietokannasta peräisin olevaksi olioiksi. Tämä tapahtuu joko määrittelemällä reitin käsitteli-

jään annotaatioina entiteettiluokka ja mahdollisesti Doctrinen repositorio-metodi sekä tarkemmat datan kartoitusohjeet.

Symfonyn reiteille on mahdollista annotaatioiden avulla määrittää olioiden hakeminen automaattisesti tietokannasta. Tämä tarkoittaa sitä, että jonkin reitin muuttujan tai muuttujien perusteella etsitään tietokannasta oikea rivi ja luodaan sitä vastaava tietokanta entiteetti. Tämä entiteetti annetaan reitin käsittelijälle parametrina alkuperäisen vastaanotetun arvon sijaan. Mikäli dataa ei löydy, palautetaan käyttäjälle 404-vastaus. [6.]

3.1.3 Servicet

Servicet eli palvelut ovat Symfony-ohjelman yleisessä käytössä olevia luokkia, joiden tarkoitus on antaa ohjelman käyttöön jokin toiminto tai toiminnallisuus. Omia serviceitä on mahdollista määrittää konfiguraatioiden avulla. Serviceissä voidaan käyttää muita ohjelmiston käyttämiä serviceitä hyödyntämällä riippuvuusinjektiota. Tätä varten konfiguraatiossa määritetään servicen argumenteiksi sille välitettävät servicet. Symfony välittää serviceiden argumentit sille sen luontivaiheessa, joten luokalle on luotava konstruktori, jonka parametreina on sille välitettävät servicet. [4, s. 203.]

Servicet haetaan Symfonyn-ohjelman koodissa ServiceContainerilta. Kun serviceä kutsutaan ServiceContainerilta, se instantioidaan sekä samoin tarvittaessa kaikki muut sen vaatimat servicetkin. Servicet ovat vakiona globaalisti jaettuja eli niitä kutsuttaessa palautetaan joka kerta sama instanssi. Servicet voidaan myös haluttaessa konfiguroida palauttamaan joka kerta sitä tarvittaessa uuden instanssin. [4, s. 204.]

Kontrollereissa on mahdollista hakea serviceitä ServiceContainerin avulla. Kontrollereihin tämä on sisällytetty yleisen Controller-luokan, jonka tavallisesti ohjelman kontrollerit perii. ServiceContainer-olio mahdollistaa minkä tahansa servicen käyttämisen, joka löytyy jostain käyttöönotetusta bundlesta.

Symfonyn kontrollerien on tarkoitus olla kevyitä sekä sisältää vain minimaalinen koodi reittiä varten. Symfonyn kontrollerien ei ole lähtökohtaisesti tarkoitus olla uudelleenkäytettävissä eri puolilla sovellusta. Uudelleenkäytettävää koodia voi ja suositellaan kirjoitettavan serviceiksi. Näitä voi ladata ja käyttää kontrollereissa. Vaikka Symfonyn kontrollereihin

voisi kirjoittaa monimutkaisenkin datan käsittelyn, ei näin kannata tehdä, mikäli samaa logiikkaa tarvitaan toisella puolen sovellusta. Tämän logiikan voi kirjoittaa sen sijaan erilliseen PHP-luokkaan, jonka voi ottaa Symfonyssä käyttöön servicenä.

3.1.4 Security

Symfony myös sisältää Security-komponentin. Tämän komponentin tarkoitus on antaa työkalut kehittäjälle käyttäjien autentikointiin ja autorisointiin. Tätä varten Symfony sisältää security.yml konfiguraatiotiedoston. Tässä tiedostossa määritetään yleiset sovelluskohtaiset ohjeet http-pyyntöjen autentikointia sekä autorisointia varten sekä määritetään esimerkiksi sovelluksen käyttämä salasanojen salausalgoritmi. [4, s. 163-168.]

Security.yml:ssä määritetään käyttäjien hakeminen sovelluksen käyttöön. Tämä tapahtuu konfiguroimalla käyttäjien tarjoaja (provider). Tarjoaja on yleensä palvelu, ja sen tarkoitus on hakea sekä autentikoida käyttäjät. Symfony-sovelluksen reittejä eli pyyntöjen osoitteita voi suojata palomuurien (firewall) avulla. Palomuureille määritetään niille kuuluva reitti sekä mahdollisesti niiden käyttämät käyttäjätarjoajat ja kirjautumistavat. [4, s. 163-167.]

Käyttäjille voidaan Symfonyssä konfiguroida erilaisia rooleja, joita voi käyttää sovelluksessa autorisointia varten. Konfiguraatio sisältää myös access_control-osion, johon voi konfiguroida ohjelmiston sivut ja niiden vaatimat roolit. Rooleja voi käyttää eri puolilla sovellusta erilaisten toimenpiteiden autorisointiin ja käyttäjän oikeuksien tarkastamiseen. Kontrollereissa ja mallipohjissa voidaan usein tarvita oikeuksien tarkastamista. Twig-malleissa on mahdollista käyttää isGranted-funktiota, jota voidaan käyttää sivujen osien ehdolliseen renderöintiin. Kontrollereissa voidaan käyttää @Security-annotaatiota tai denyAccessUnlessGranted-funktiota, jotka kumpikin hylkäyksen yhteydessä lopettavat pyynnön käsittelyn ja palauttavat käyttäjälle kieltävän HTTP 403 -vastauksen. [4, s. 167-170.]

Security-komponentti sisältää autorisointipalvelun, joka sisältää isGranted-metodin, jota nämä aiemmin mainitut autorisointimenetelmät hyödyntävät. Tätä autorisointipalvelua on mahdollista käyttää suoraan, jos on tarvetta tarkastaa käyttäjän oikeudet ilman muita toimenpiteitä. [4, s. 169.]

Symfony sisältää roolien lisäksi myös muitakin autorisointi menetelmiämonimutkaisempia käyttötapauksia varten. Roolit ovat käyttäjän sisältämiä ominaisuuksia, joita voidaan käyttää toimenpiteiden autorisointiinkin. Monimutkaisempiin tarkoituksiin on kuitenkin mahdollista luoda käyttötapauskohtaisia autorisointi tarkastuksia äänestäjien (Voter) avulla. [4, s. 168.]

3.1.5 Twig

Symfonyn on luotava vastaus sen vastaanottamiin kutsuihin. Usein kutsuihin palautetaan vastaus HTML:nä, jonka selain esittää käyttäjälle. Tämä HTML voidaan palauttaa valmiina tekstinä, tai se voidaan generoida dynaamisesti PHP:n tai jonkin sivupohjakirjaston avulla. Symfonyn mukana tulee sivupohjakirjasto nimeltä Twig. Twigissä eri kutsujen välisesti samana pysyvät osat kirjoitetaan HTML:nä. Tämän HTML:n sekaan kirjoitetaan sivun dynaamiset osat erilaisten rakenteiden avulla. [4, s. 69.]

Twig käännetään ajon aikana PHP-koodiksi, joka varastoidaan myöhempää käyttöä varten. Tämä tarkoittaa sitä, että sivupohja ei käännetä uudestaan jokaisen pyynnön yhteydessä. Ensimmäisen pyynnön jälkeiset pyynnöt hyödyntävät natiivia PHP-koodia vastauksen generoimiseen. [4, s. 70.]

Twig-sivupohjat on tarkoitus kirjoittaa useiksi tiedostoiksi. Sovelluksen eri sivujen välillä on usein yhteisiä osia. Twigissä savipohjien on mahdollista periä muita ylemmän tason sivupohjia. Tämän avulla on mahdollista esimerkiksi sivun navigaation sekä ala- ja yläviitteiden jakaminen usealla sivulla. Ylemmän tason sivupohjaan määritetään block eli lohko, johon alemman tason sivupohjassa voidaan koodia kirjoittaa. Alemman tason sivupohjissa voidaan ylikirjoittaa ylemmän tason pohjan sisältämiä lohkoja. [4, s. 70-72.]

Sivun HTML:ää generoitaessa sivupohjalle yleensä välitetään muuttujia, joiden avulla määritetään sivulla esitettävä dynaaminen sisältö. Twigissä voidaan näiltä muuttujilta hakea renderöitävää dataa. Twig tukee myös esimerkiksi listojen iterointia ja konditionaalista renderöintiä. Twig-pohjissa voi myös määrittää Twigin sisäisiä makroja, joiden avulla voidaan koodia renderöidä parametrien perusteella. [4, s. 69.]

Twig-pohjille voi myös viedä uusia funktioita Twig-laajennuksien avulla. Nämä ovat tietynlaisia servicejä, joissa voidaan määritellä Twig-pohjille uusi funktio ja sitä vastaava PHP-koodina kirjoitettu logiikka. [4, s. 69.]

3.2 Doctrine

ORM eli Object Relation Mapping on yleinen olio-ohjelmoinnissa käytetty tekniikka tiedon muuttamiseen relaatiotietokantojen ja olio-rakenteen välillä. Doctrine on Symfonyn käytetyimmän jakelun käyttämä vakio-ORM. [4; 7, s. 92.]

Doctrine käyttää entiteeteiksi kutsuttuja olioita datan tallentamiseen tietokantaan sekä tietokannasta peräisin olevan datan säilyttämiseen ohjelman ajon aikana. Entiteettiluokat ovat luokkia, jotka sisältävät yksinkertaisimmillaan ominaisuudet sekä niille get- ja set-metodit. Entiteetit voivat kuitenkin sisältää muitakin kenttiä sekä metodeita, jotka Doctrine jättää huomioimatta. [8.]

Doctrinessa tietokantaa käytetään EntityManagerin avulla. Sen kautta hoituu tietokannasta datan hakeminen ja kirjoittaminen. EntityManager käyttää Unit of Work suunnittelumallin mukaista toiminnallisuutta pitääkseen yllä mitä tietokantaan kirjoitetaan. EntityManagerin persist-metodilla voidaan lisätä olio tietokantaan ja delete-metodin avulla poistaa olio sieltä. Näiden funktioiden kutsuminen lisää EntityManagerin tietoon tietokantaan vietävän datan muutokset. Flush-metodia kutsuttaessa generoidaan tarvittavat tietokanta kyselyt datan viemiseksi tietokantaan. Tämän jälkeen käynnistetään tietokanta transaktio ja suoritetaan kaikki kyselyt. [8; 9.]

Entiteettejä haetaan Doctrinessa repositorioiden avulla. Doctrine-repositoriot ovat ObjectRepository-rajapintaa toteuttavia olioita, jotka sisältävät metodeita datan hakemiseen tietokannasta. Vakiona Doctrine käyttää jokaista entiteettiä varten sen ennalta määrittelemää EntityRepository-luokkaa, joka sisältää perustoiminnallisuuden datan hakemiselle tietokannasta. On myös mahdollista luoda omia repositorioita. [9.]

Repositorioiden yleinen luokka myös sisältää valmiiksi toteutettuja metodeita, joiden avulla voidaan dataa hakea. Nämä valmiit metodit ovat findAll, findBy sekä findOneBy. Näistä ensimmäinen hakee tietokannasta kaikki repositoriota vastaavan luokan entiteetit. Kah-

delle seuraavista määritetään kriteereitä, joiden mukaan data haetaan. Nämä kriteerit ovat käytännössä rajoituksia entiteetin kenttien arvoille ja toimivat SQL:n WHERE-lauseen kaltaisesti. [9.]

Entiteettikohtaisesti voidaan luoda omia repositorioluokkia. Nämä yleensä perivät EntityRepository-luokan, jolloin omaan repositorioon ei tarvitse kirjoittaa muuta kuin metodit datan hakemislogiikkaa varten. Luomalla oman oma repositorio, voidaan siihen tallentaa yleisimpiä koodissa olevia tietokantahakuja sekä optimoimaan niitä. [9.]

Doctrine hyödyntää Proxy-luokkia mahdollistaakseen lazy loading -toiminnallisuuden. Yleisesti lazy loading on suunnittelumalli, jonka avulla pyritään lykkäämään raskasta olion luontia niin kauan kuin mahdollista ja luomalla sen vasta sitä käytettäessä. Doctrinen tapauksessa lazy loadingia varten luodaan jokaisesta entiteetistä proxy-luokka, joka perii varsinaisen entiteettiluokan. Tämä mahdollistaa sen, että tarvittaessa dataa vain joltain tietyltä entiteetiltä, vain haluttu entiteetti haetaan ilman sen sisältämien referenssien mukaisia olioita. Doctrine korvaa entiteetin sisältämät referenssit Proxyn instanssilla. Kun jotain proxyn funktiota kutsutaan, suoritetaan oikean entiteetin hakeva tietokantahaku ja proxy korvataan oikealla entiteetillä. [9; 10.]

Lazy loading on Doctrinen vakiostrategia entiteettien liitoksien käsittelyyn. Doctrinelle on kuitenkin mahdollista määrittää entiteettien välisten liitoksien haku strategia. Eager loadingin avulla voidaan proxy-luokan käyttämisen sijaan hakea saman tien oikea entiteetti. Monissa tapauksissa tämä ei ole kannattavaa ja kyselyt voivat johtaa tarpeettoman datan hakemiseen tietokannasta, joka taas hidastaa ohjelman toimintaa. Poikkeuksia kuitenkin löytyy. Dynaamisesti ladattavien entiteettien iteroiminen voi johtaa suureen määrään tietokantakutsuja ja olla täten hidasta. [9.]

Doctrinelle on kuitenkin mahdollista kyselykohtaisesti määrittää hakustrategia kullekin liitokselle. Tämän takia voidaan esimerkiksi iteroimista varten luoda kysely, joka hakee kaikki tarvittavat entiteetit kerrallaan. Liitoksen hakustrategiaa sekä omia hakuja varten on kuitenkin luotava oma repositorio luokka sekä tähän oma hakumetodi. [11.]

Entiteettien perimisrakenteita on myös mahdollista tallentaa tietokantaan. Single table inheritance on tietokanta ORM:ien käyttämä tapa kartoittaa tietokantaan abstrakteista luokista peräisin olevaa dataa. STI kartoittaa entiteetti rakenteen yhteen tietokantatauluun.

Tämä taulu sisältää jokaisen tietokantaan tallennetun kentän jokaiselta entiteetiltä. Tarkka entiteetti tallennetaan discriminator-kenttään, joka on yleensä luokan määrittävä merkkijono. Jokaiseen entiteetin sisältämättömään kenttään tallennetaan arvoksi null. [12.]

Toinen STI:n kaltainen menetelmä entiteettirakenteiden tallentamiseen tietokantaan on CTI eli Class Table Inheritance. CTI:tä käytettäessä luodaan yhden taulun sijaan jokaiselle eri entiteettiluokalle oma taulu. Näiden taulujen lisäksi luodaan yksi yhteinen taulu, johon tallennetaan kaikki yhteiset datat. Tämän avulla voidaan välttyä null-arvoilta tietokannassa. CTI-tauluille on myös tyypillistä käyttää yhteistä id-kenttää useiden taulujen välillä. [12.]

EntityManager voi käyttää tietokantaa DQL-kielen avulla. DQL on SQL:n kaltainen kieli, jossa on kuitenkin oma syntaksi. DQL:ää voi käyttää entiteettien hakemiseen, luomiseen sekä poistamiseen. INSERT-lausekkeitta ei kuitenkaan se tue, koska EntityManagerille on erikseen ilmoitettava muutettu ja tallennettava entiteetti. [11.]

Doctrinella voi entiteettihakuja luoda DQL:n kirjoittamisen sijaan . Doctrine myös tarjoaa QueryBuilderin, joka on useista funktioista koostuva rajapinta, jonka avulla voi tehdä suoritettavan kutsun. QueryBuilder toimii muuten DQL:n tapaan, mutta koostuu merkkijonon sijaan funktioista. Näitä funktioita kutsumalla kasataan itse kysely. [13.]

3.3 Javascript

3.3.1 Backbone.js

Backbone.js on avoimen lähdekoodin asiakaspuolella käytettävä javascript-ohjelmistokirjasto. Backbonen kehitys aloitettiin 2010 ja versio 1.0 julkaistiin 2013. Backbone.js:n tarkoitus on tuoda rakennetta javascript-koodiin sen malleilla (Model), kokoelmilla (Collection) ja näkymillä (View). Nämä komponentit muodostavat MV*-mallin kaltaisen rakenteen. Backbone.js myös sisältää reititys komponentin, joka mahdollistaa SPA:iden (Single Page Application) luomisen. Kaikki Backbone.js:n komponentit hyödyntävät Backbone.js:n tapahtumia (Event) ja käyttävät niitä kommunikointiin toistensa kanssa. [14; 15.]

Backbone.js on tiedostokooltaan pienempi kuin mikään tämän hetkisistä ja vähintään yhtä suosituista ohjelmistokehyksistä. Se sisältää myös vain yhden ohjelmisto-riippuvuuden, joka on underscore.js. Underscore.js on funktiokirjasto, joka sisältää useita funktioita erilaisten yleisten toimintojen toteuttamiseksi.

Backbone.js:ssä sovelluksen käyttämä data tallennetaan pääasiassa sen malleihin avain-rakenne pareina. Tyypillisesti mallit sisältävät sovelluksen palvelinpuolelta peräisin olevan datan. Data yleensä näitä varten ensimmäiseksi lähetetään HTML-dokumentin mukana, jossa data tallennetaan JavaScript-olioihin. Data voidaan myös hakea sivun latauksen jälkeen palvelimelta esimerkiksi AJAX-kutsuilla, joita varten Backbone.js sisältää toiminnallisuuden jo valmiiksi. Mallille voi asettaa REST-rajapinnan URL:in.Tämän jälkeen mallin dataan tehdyt muutokset voidaan lähettää rajapinnalle tai vaihtoehtoisesti rajapinnalta voidaan dataa sivulla olevan datan päivitystä varten. [15.]

Backbone.js-kokoelmiin voi tallentaa useita saman mallin instansseja. Kokoelmien tarkoitus on antaa työkalut useiden mallien yhtenäiseen käyttöön ja hallintaan. Backbone.js:n kokoelmat sisältävät rajapinnan, jonka avulla näitä toimintoja voidaan tehdä. Kokoelma-oliot sisältävät funktioita, joiden avulla niiden jäsenkokoelmaa voi käsitellä. Nämä funktioit sisältävät yleisiä funktioita kuten jäsenien lisääminen, etsiminen ja järjestäminen. Monet näistä funktioista on peräisin underscore.js:ltä. Kokoelmat myös automaattisesti kuuntelevat jäseniensä tapahtumia. [15.]

Backbone.js hyödyntää komponenttien väliseen kommunikointiin sen tapahtumia. Backbone.js automaattisesti käynnistää joissakin tietyissä tilanteissa tapahtuman tai tapahtumia. Tällaisia tilanteita ovat muun muassa mallin datan muuttuminen tai kokoelman muokkaus. Jokaiselle komponentille voi halutessaan asettaa kuuntelijan kuuntelemaan muiden komponenttien lähettämiä tapahtumia. [15.]

Backbone.js-tapahtumat toimivat normaalisti synkronisina, toisin kuin esimerkiksi JavaScriptin natiivit tapahtumat. Backbone.js:n tapahtuma-kuuntelijat käynnistetään komponenttien kutsumalla listenTo-funktiota, jolle annetaan parametreiksi kuunneltava olio, tapahtuman nimi ja callback-funktio. Trigger-funktion avulla voidaan käynnistää omia tapahtumia. Trigger-funktio ottaa vastaan tapahtuman nimen sekä valinnanvaraisen määrän argumentteja. Trigger-funktiota kutsuttaessa käydään vuorotellen läpi jokainen kyseiseen

tapahtumaan liitetty callback-funktio. Trigger() ja listenTo() ovat Backbone.js komponenttien sisältämiä funktioita. [15.]

Backbone.js-mallit ja kokoelmat toimivat pelkästään sovelluksen datakerroksena eli niiden ei kuulu tietää mitään sovelluksen HTML-koodista ja sen DOM-puusta (Document Object Model). Backbone.js näkymien on tarkoitus hoitaa HTML-koodin luominen ja sen liittäminen DOM-puuhun. [15.]

Jokainen näkymä on liitetty johonkin HTML-elementtiin. Näkymät voidaan liittää joko olemassa oleviin DOM-puun solmuihin tai näkymien HTML voidaan generoida renderöinti funktiossa. Renderöintiin voidaan käyttää esimerkiksi mallipohjia. Nämä luomisoperaatiot eivät ole toisiaan pois sulkevia vaan on myös mahdollista liittää näkymä DOM-puun solmuun ja jälkikäteen päivittää sitä esimerkiksi mallipohjien avulla. [15.]

3.3.2 Webpack

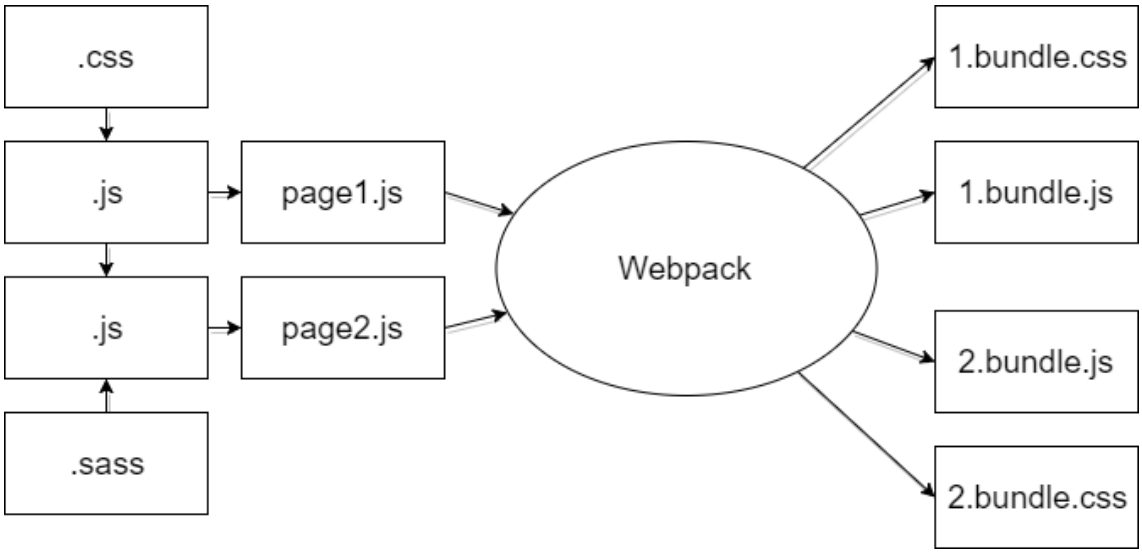
JavaScriptiä aiemmin käytettiin lähinnä pienen toiminnallisuuden toteuttamiseen websivustoilla. Nykyään JavaScript on erittäin yleinen kieli, jolla tehdää paljon muutakin kuin luodaan pienehköä interaktiivista toiminnallisuutta web-sivustoilla. JavaScriptiä käytetään muun muassa palvelimilla sekä sillä luodaan monimutkaisia web-sovelluksia. Sovellusten kasvaminen vaatii huomattavasti enemmän ohjelma koodia sekä tätä myötä tavan käyttää ohjelmistoja uudestaan. Moduulit ovat tapa jakaa ohjelmistoja useisiin uudelleen käytettäviin ja toisista riippumattomiin osiin. Tällaisia osia ovat esimerkiksi ohjelmistokirjastot tai sovellusten eri osat. Moduulien tarkoitus on luoda koodista ylläpidettävämpää kuin mitä se olisi verrattuna moduuleja hyödyntämättömään JavaScript-koodiin.

JavaScriptille on kehitetty monenlaisia työkaluja ja tapoja koodin sekä moduulien hallintaan. Modernit JavaScript moduulien hallinnan tekniikat ovat AMD (Asynchronous Module Definition), CommonJS-moduulit sekä EcmaScript 6:n moduulit. Jokainen näistä on kehitetty alun perin eri syistä ja jokaisella niillä on oma syntaksinsa. CommonJS kehitettiin selaimien ulkopuoliseen käyttöön ja sen moduuli syntaksi on tarkoitettu yhden moduulin lataamisen kerrallaan require-funktion avulla. AMD kehitettiin vastauksena CommonJS:n moduuleille, jotka eivät olleet vielä tuolloin valmiita selaimilla käytettäväksi. EcmaScript 6 (ES6) spesifikaatio sisältää tuen moduuleille ja toteuttaa sen import- ja export-

lausekkeiden avulla. Näistä kahta ensimmäistä varten löytyvät omat kirjastot selainympäristössä käyttämistä varten. Vaikka tuki ES6:n mukaiselle JavaScript koodille onkin jo melko kattava uusimmissa selaimissa, eivät nekään vielä moduuleita tue. Myös vanhempien selainversioiden käyttäminen on tällä hetkellä niin yleistä, että tukea niille ei haluta saman tien pudottaa. Esimerkiksi mikään Internet Explorerin versio ei tue ES6:sta millään tavalla. [16; 17.]

Webpack sisältää tuen kaikille edellä mainituille moduulien hallinta tekniikoille. Webpackin tarkoitus on muuttaa ja optimoida JavaScript- tai CSS -koodi kohdeympäristöä varten. Yleisiä käyttötarkoituksia Webpackille on juurikin modulaarisen koodin yhdistäminen selaimella ajettaviksi ohjelmiksi sekä koodin kääntäminen kieleltä tai standardilta toiselle.

Jokaista JavaScript ohjelmaa varten luodaan tulokohta eli käytännössä tiedosto, josta ohjelman suoritus alkaa ja missä määritetään osa sovelluksen käyttämistä moduuleista. Tulokohta tiedostoja voi yhdessäkin sovelluksessa olla useita. Esimerkiksi jokaista websivua varten voidaan luoda erillinen JavaScript-ohjelma, jossa ladataan vain sivulla tarvitut elementit. Webpack etsii sille määritetyistä tulokohdista sen sisältämät moduulimääritykset ja seuraa niitä etsien aina määriteltyjen moduulien uudest määritykset. Moduulimääritykset sisältävät yleensä merkkijonona moduulin lähteen. Webpack etsii tämän määrittelyn sekä sen konfiguraatioidensa perusteella tiedoston, joka sisältää tarvitun moduulin. Tulokohta-tiedosto ja sen käyttämät moduulit sitten kopioidaan bundle-tiedostoon, joka on valmis käytettäväksi ja se voidaan esimerkiksi ladata sivulle HTML:n <script>-tagin avulla. [18.]



Kuva 1: Kuvassa webpackin toimintaperiaate. Webpackille annetaan tulokohtatiedostot, joilla voi olla riippuvuuksia muihin tiedostoihin. Webpack luo sivuilla käytettävät bundle-tiedostot.

Webpackia varten luodaan konfiguraatiotiedosto tai -tiedostot. Konfiguraatiotiedostojen tarkoitus on luoda Webpackia varten konfiguraatio-olio, jonka perusteella Webpack-ohjelma suoritetaan. Webpack.config.js-tiedosto toimii Webpackin tulokohtana. Tämä tiedosto on moduuli, joka exportin avulla palauttaa itse konfiguraatio-olion. Tiedostossa voidaan ladata lisäosia sekä konfiguraatioiden osia muista moduuleista. [19.]

JavaScriptejä läpi käydessään Webpackille voidaan antaa käsiteltäväksi sen moduulirajapinnan avulla JavaScript-moduulien lisäksi tyylitiedostoja. Nämä tyylitiedostot on voitu kirjoittaa joko CSS:llä tai jollain muulla CSS:ksi kääntyvällä kielellä kuten LESS tai SASS. Tyylitiedoston määrittäminen JavaScript-moduuleissa mahdollistaa sivu- tai ohjelmakohtaisten tyylitiedostojen luomisen.

Webpack voidaan myös konfiguroida tekemään lisäosien avulla useita muitakin operaatiota. Lisäosien avulla on mahdollista hoitaa muidenkin kuin JavaScript tiedostojen lataaminen ja käsittely sekä kääntää koodia kieleltä toiselle. Näitä varten voidaan käyttää Webpackin lisäosia.

3.3.3 Video.js

Video.js on avoimenlähdekoodin javascriptiin perustuva videosoitin. Vaikka selaimet pystyvät omilla soittimillaan toistamaan videota ja audiota, mutta soittimet vaihtelevat ulkonäöltään ja toiminnallisuuksiltaan. Video.js:n soittimen ulkoasu on tehty HTML:n ja CSS:n avulla. Ulkoasu myös pysyy samanlaisena videontoisto tekniikasta riippumatta. Video.js tukee myös soittimen komponenttien HTML:n muokkaamista, joten soittimia voidaan muokata sellaiseksi kuin haluaa. Video.js tukee useita eri videontoisto tekniikoita, joista ilman lisäosia on käytettävissä HTML5:n <video>-tagi sekä Flash. Lisäosien avulla on mahdollista käyttää muitakin tekniikoita kuten YouTube sekä Vimeo. [20; 21.]

4 Toteutus

4.1 Backend

Insinöörityönä tehdyn sovelluksen palvelinosuutena toimii Symfony2-sovelluskehyksellä tehty sovellus. Tämä Symfony-ohjelma on vastuussa ohjelmiston reitityksestä, sivujen tarjoamisesta sekä muusta palvelimella tapahtuvasta sovelluksen toiminnallisuudesta. Sovelluksen web-sivut on kirjoitettu Twig-mallipohjien avulla. Nämä mallipohjat kääntyvät PHP:ksi ja lopulta HTML:ksi. Sivut palvelin tarjoilee valmiina selaimille HTML:nä.

4.1.1 Reititys

Kaikki käyttäjältä tulleet sivunlataus- sekä API-kutsut päätyvät Symfony-ohjelmalle. Symfony-ohjelma on vastuussa ohjelmiston sivu-kutsujen sekä käyttöliittymän käyttämien API-kutsujen reitittämisestä.

Symfonyn reittien käsittelijöille voidaan yleisiä joitakin toimintoja kirjoittaa PHP-koodin sijaan annotaatioina. Tällaisia toimintoja ovat esimerkiksi reittikohtaiset autorisointitarkistukset sekä tarkemmat ohjeet Doctrinelle entiteettien hakemiseen. Nämä ovat asioita, joita varten usein pitäisi kirjoittaa useaan kontrollerin käsittelijään samankaltaista koodia. Annotaatioita käyttämällä saadaan reitin koodi sisältämään vain datan käsittelyn ja sivun renderöinnin kannalta oleellisen koodin. Sovellus käyttää aina, kun mahdollista reittien yhteydessä annotaatioita parametrien vastaavan datan hakemiseen tietokannasta.

4.1.2 Lomakkeet

Lomakkeita sovelluksessa käytetään datan pyytämiseen käyttäjältä. Lomakkeita sovelluksessa on muutama. Presentaatioiden luomista ja editoimista varten on yksi pääloma-ke. Tähän lomakkeeseen sisältyy tiedostojen liittämistä varten luotu uudelleenkäytettävä

lomake, jonka tarkka entiteetti määritetään presentaatiolomakkeessa tiedostolomakkeen liittämisen yhteydessä.

Presentaation mediatiedostoluokat toteuttavat yhteistä rajapintaa, jonka avulla niitä voidaan käyttää lomakkeissa, yms.. Tätä mediatiedosto luokkaa varten on luotu tuo alalomake. Eri mediatiedostojen käsittely on lomakkeissa melko samanlaista ja ainoana erona on html, validointi lomakkeen jälkeinen käsittely. Lomakkeet luovat samankaltaisen HTML:n näille kahdelle kentälle ja tarkkaa luokkaa Symfony käyttää datan validointia varten.

Validointia varten on määritetty validation.yml-tiedosto. Tässä tiedostossa määritetään eri luokkien ja kenttien validointisäännöt. Säännöt määritetään käyttämällä Symfonyn validointiin tarkoitettuja rajoiteluokkia. Tiedostoon tai itse entiteettiluokkaan annotaatioina näitä määrittää jokaiselle kentälle yhden tai useamman. Esimerkiksi merkkijonoja varten löytyy validaattoreita pituutta sekä erilaisia merkkijonon syntakseja varten.

Validointiin voidaan myös pelkän konfiguraation lisäksi luoda serviceinä tuotuja validointiluokkia. Näitä tarvitaan monimutkaisempien validointistrategioiden kuin tyyppi-, luokka- tai syntaksitarkastuksien toteuttamiseen. Diaesitysten validointia varten luotiin erillinen validointiluokka. Tämä luokka on otettu Symfonyssa käyttöön servicenä ja määritetty entiteettiluokan validointikonfiguraatioon. Entiteettiä validoitaessa servicen validointimetodia kutsutaan. Metodi voi merkitä sille parametrina syötettyyn kontekstiolioon mahdolliset validointivirheet. Diakuvien validointiluokka käyttää sovelluksen tiedostonmuunto servicen toimintoja tarkistaakseen, voidaanko syötettyä tiedostoa ylipäätänsä käyttää.

4.1.3 Renderöinti

Ohjelmisto luo palvelimella HTML-koodin sivuja varten ja palauttaa sen vastauksena sivukutsuihin. HTML-koodi luodaan käyttäen Twig-mallipohjia. Jokaista sivua varten on luotu oma twig-pohja. Tämän lisäksi on luotu useita perusta-pohjia, joita käytetään sivun yhteisten osien renderöintiin. Yleiselle sivun rakenteelle, joka sisältää <body>- ja <head>-tagit sekä Twigin blockien määrittelyt. Toisessa pohjassa määritellään sivun yleiset navigointielementit ja muut sivuille yhteiset asiat. Esitys-sivuille on luotu yhteinen mallipohja, jonka esityksen esittämis- sekä muokkaamissivujen pohjat perivät.

Sivukohtaisia JavaScriptejä ja tyylitiedostoja varten on luotu yhteiseen mallipohjaan tyhjä block. Sivun omassa mallipohjassa tämä block joko ylikirjoitetaan tai sen loppuun lisätään tarvittavat <script>- tai <link>-tagit tiedostoja varten.

4.1.4 Tietokanta ja entiteetit

Web-sovellukset usein käyttävät jonkinlaista tietokantaa niiden käyttämän datan säilyttämiseen. Insinöörityönä tehty sovellus käyttää datan säilyttämiseen Doctrine ORM:n välityksellä MySQL-relaatiotietokantaa.

Sovelluksessa käytetään Doctrinen entiteettejä tietokantaan tallennettavan datan käsittelyyn. Jokainen entiteettiluokka sisältää tietokantaan tallennettavat kentät sekä relaatiot muihin sovelluksessa käytettäviin entiteetteihin. Esimerkiksi esityksiä varten on luotu oma entiteettiluokka, jonka tietokantaan tallennettavia kenttiä ovat muun muassa esityksen nimi ja kategoriat sekä esitykseen kuuluva video ja diaesitys. Presentaatioiden ja kategorioiden välillä on many to many -suhde eli kategorioita voidaan yhdelle presentaatiolle asettaa useita ja päinvastoin.

Video, diaesitys ja diakuva ovat esityksen tiedostoja sisältävät osat, joista jokaista varten on luotu oma luokka. Nämä luokat sisältävät tietokantaan tallennettavat itse mediaan liittyvän datan. Jokainen näistä entiteeteistä myös sisältää oman datansa lisäksi tiedostokentän, joka on viittaus median tiedostoentiteettiin.

Esitysten tiedostoja varten on luotu oma abstraktiluokka, joka sisältää kentät yleisiä medioiden tietoja varten. Tällaisia kenttiä ovat esimerkiksi tiedoston alkuperäisen nimi, sijainti kovalevyllä sekä tiedoston tarkistussumma. Itse video-, diaesitys- ja diakuva-tiedoistoille on kaikille luotu oma tiedostoluokka, joka perii yleisen esitystiedostojen yliluokan. Koska jokainen tiedostotyyppi sisältää jossain ohjelman osassa poikkeavaa käsittelyä muista, voidaan eri luokkien avulla määrittää luontevasti eri tiedostoille mahdollisia tyyppitarkastuksia ohjelman koodissa.

Useimpia entiteettiluokkia varten on luotu omat repositorioluokat, koska niissä voidaan määritellä tarkasti millaiset tietokantahaut halutaan suorittaa. Näitä repositorioluokkia voi

käyttää joko ohjelmakoodissa EntityManagerin kautta tai määritellä action-metodin annotaatioissa repositorion metodi.

Tämä oliorakenne tallennetaan hyödyntäen Doctrinea ja STI (Single Table Inheritance) -menetelmää. Tiedostojen tallentamiseen tietokantaan valittiin STI, koska tiedostot sisältävät lähes saman datan ja jokainen tiedostotyyppi voi periä sen tietokanta kentät yhteiseltä luokalta. Tietokantaan tallennettava data sisältää tiedon, jonka avulla voidaan löytää oikea tiedosto. Jokaiselle mediatyypille kuitenkin luotiin oma tiedostoluokka, jotta ohjelmassa voitaisiin hyötyä PHP:n tyyppi- ja luokkatarkastuksista.

Yhteistä tiedosto luokkaa käyttäessä olisi joutunut tallentamaan joko tiedoston datan itse medialuokalle tai tiedostoon olisi pitänyt tallentaa tyyppi-kenttä. Luokka- ja tyyppitarkastukset ovat yleensäkin siistimpi vaihtoehto kuin datan sisältävä kenttä, koska voidaan välttyä turhan koodin kirjoittamiselta hyödyntämällä luokkamäärityksiä metodien parametreissä. Tämän lisäksi kehitysympäristöt osaavat automaattitäydennyksiä varten hyödyntää luokkamäärityksiä paremmin kuin merkkijonokenttiä.

Jotta PHP:n olioita voitaisiin välittää JavaScriptille tai muualle sovelluksen ulkopuolelle, on ne ensin serialisoitava. Sovellus muuttaa muualle välitettävän datan JSON-muotoon. Tätä varten luotiin konfiguraatiot siitä, mitkä entiteetit serialisoidaan milläkin tavoin. Konfiguraatio on luotava, koska siinä voidaan määrittää tarkemmin käyttöliittymälle lähetettävät kentät ja niiden nimet. Tällä tavoin voidaan välttyä tarpeettoman datan välittämisesti käyttöliittymälle.

4.1.5 Tiedostot

Esitelmien luontia varten sovellukseen tehtiin lomake, jonka avulla voidaan syöttää esitelmän tiedot sekä sen käyttämät diaesitys- ja videotiedostot.

Tiedostosta tallennetaan tietokantaan myös sen sha256-muotoinen tarkistussumma. Tätä tarkistussummaa käytetään kovalevytilan säästämiseksi käyttämällä duplikaattitilanteessa vanhoja tiedostoja uudestaan. Kun tiedosto syötetään järjestelmään, lasketaan sen sha256sum ja etsitään, löytyykö tietokannasta vastaavaa. Jos löytyy, tarkastetaan tie-

dostoja tarkemmin. Jos tiedostot ovat samat, käytetään samaa kovalevyllä sijaitsevaa tiedostoa uudestaan.

Web-selaimessa ei voida näyttää powerpoint-tiedostoja vaan ne on ensin muutettava johonkin selaimen käyttämään muotoon. Selaimessa voidaan diakuvat esittää monella eri tavalla, mutta yksinkertaisinta on näyttää diat kuvatiedostoina tai svg-vektorigrafiikkana. Selaimet voivat myös varastoida lataamansa kuvatiedostot välimuistiinsa, josta ne voidaan ladata, mikäli käyttäjä lataa sivun hetken päästä uudestaan. Tämä nopeuttaa sivun lataamista huomattavasti. Kuvatiedostot eivät myöskään tarvitse erikoiskäsittelyä muunnosten jälkeen. PHP:lle ei ole ainakaan avoimen lähdekoodin kirjastoja, jotka kykenisivät muuttamaan PowerPoint-tiedostoja toisiin formaatteihin. Tästä syystä on käytettävä erillistä ohjelmaa. Sovellus käyttää itse diaesitystiedoston muuntamiseen unoconv-työkalua, joka taas ohjaa tiedostomuunnokset libreofficelle. Näiden ohjelmien avulla saadaan diaesityksestä pdf-tiedosto. PNG-tiedostoiksi tämän PDF-tiedoston voi voidaan muuntaa esimerkiksi ImageMagick-ohjelmalla.

Mediatiedostojen tarjoaminen tapahtuu Symfonyn kautta. Tämän avulla voidaan autorisoimattomilta käyttäjiltä kieltää resurssikutsut videoihin tai diaesityksiin. Kun kutsut kulkevat Symfonyn kautta, voidaan hyödyntää Symfonyssä tehtyjä tietoturvamäärittelyjä, joiden avulla voidaan tarkastaa tarkemmin se, kuka haluaa ja saa mitäkin tiedostoa käyttää.

4.2 Frontend

4.2.1 Yleisesti

Ohjelmistoa käytetään selaimella ja sen asiakaspuoli on tehty muun muassa HTML5-, CSS3- ja JavaScript-kieliä käyttäen. Ohjelman JavaScript on kirjoitettu EcmaScript 5:n käytäntöjen mukaisesti. Sivujen CSS-tyylikoodi generoidaan hyödyntäen LESS CSS -esiprosessoria. Sivujen HTML-koodi luodaan dynaamisesti palvelimella lähinnä PHP:n ja Twig-sivupohjamoottorin avulla.

Sovelluksen JavaScript-koodi on kirjoitettu useisiin moduuleihin, joista jokainen on omassa JavaScript-tiedostossaan. Moduulit ovat uudelleen käytettäviä sovelluksen osia. Javascript-moduulit ja koodin jakaminen useampaan tiedostoon mahdollistaa sen, että si-

vulla ladataan vain tarvittava JavaScript-koodi. Sivuilla vain tarvittavan koodin lataaminen mahdollistaa pienempien tiedostojen lähettämisen selaimelle ja tämä taas vaikuttaa positiivisesti muun muassa sivun latausaikoihin. Ohjelmisto käyttää Webpack-työkalua sivuille ladattavien JavaScript-tiedostojen rakentamiseen ja pakkaamiseen.

Jokainen JavaScriptiä tarvitseva sivu toimii omana JavaScript-ohjelmanaan. Tämä tarkoittaa sitä, että jokaiselle JavaScript-logiikkaa vaativaa sivua varten luodaan oma tiedosto. Tämä tiedosto toimii JavaScript-ohjelman tulokohtana. Tämä tulokohta tiedosto sisältää sivulla tarvittavien JavaScript-moduulien määrittämisen ja voi myös sisältää jotain omaa sivukohtaista JavaScript-koodia.

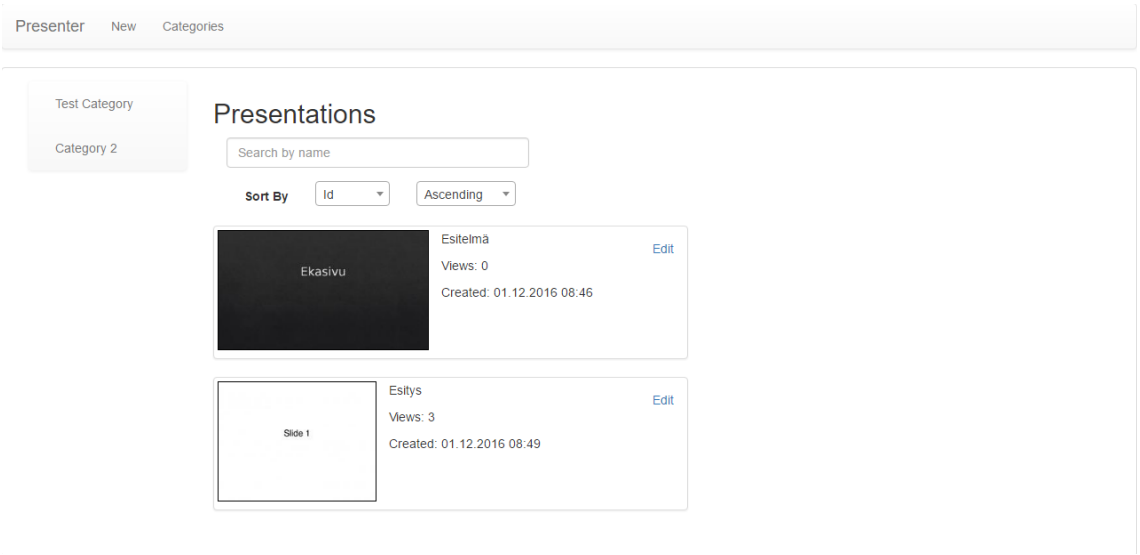
Tulokohtatiedostot määritetään webpackin webpack.config.json-konfiguraatiotiedostoon, jonka perusteella rakennetaan ja pakataan sivuille liitettävät tiedostot. Esimerkiksi etusivua varten kirjoitetaan oma JavaScript-tiedosto "homepage.js". Tämä tiedosto ja sen output-tiedosto määritetään sitä kanssa Webpackin konfiguraatiotiedoston entry-osioon. Tämän jälkeen määriteltyä JavaScript-tiedostoa käytetään lähtökohtana pakatun JavaScript-tiedoston luomista varten. Sama koskee kaikkia muitakin sivuja kuten esitelmän katselusivua. Output-tiedostoa voidaan tämän jälkeen käyttää sivun HTML:ssä <script>-tagin avulla.

```
//webpack.config.js
module.exports = {
    entry: {
        'homepage': 'js/homepage.js',
        'presentation_view': 'js/presentation_view.js'
    },
    output: {
        path: path.join(__dirname, 'dist'),
        publicPath: "/dist/",
        filename: '[name].bundle.js',
     }
}
```

Esimerkkikoodi 1: Esimerkki tulokohtien asettamisesta webpack.config.js-tiedostoon

Sovelluksen käyttämät tyyli-tiedostot on määritelty sen JavaScript-moduuleissa. Webpackin avulla voi pakata JavaScript-tiedostojen lisäksi CSS-tiedostoja, ja koska Webpackille on jo konfiguroitu sivukohtaiset JavaScript-tiedostot, voidaan näitä käyttää myös CSS-tiedostojen luomiseen.

Yhdistämällä ladattavat tiedostot voidaan pyrkiä minimoimaan sivulla käynnistyvien, staattisiin tiedostoihin kohdistuvia resurssipyyntöjä. Luomalla sivukohtaiset tiedostot voidaan pyrkiä minimoimaan ladattavien tiedostojen kokoa. Kun luodaan vielä erilliset tiedostot usealla sivulla käytettäville JavaScript- ja tyylitiedostoille, selain voi tallentaa nämä tiedostot sen välimuistiinsa ja käyttää joitakin osia uudestaan seuraavilla sivuilla.



Kuva 2: Kuva sovelluksen etusivusta

Suurin osa moduulien sisäisestä toiminnallisuudesta on kehitetty Backbone.js-ohjelmistokehystä hyödyntäen. Backbone-näkymiä ohjelma käyttää erilaisissa sivulla olevissa komponenteissa. Jos joku sivulla oleva DOM-puun elementti tarvitsee sovelluskohtaista JavaScript-toiminnallisuutta, luodaan elementtiä varten Backbone-näkymä, johon koodi kirjoitetaan. Tämä mahdollistaa Backbone-näkymässä kirjoitetun koodin kohdistamisen vain sen omaan elementtiin. Tämä näkymä voi kuitenkin sisältää ja usein sisältääkin riippuvuuksina muita kirjastoja ja moduuleja, joita käytetään toiminnallisuuden toteuttamiseen. Esimerkkejä tästä ovat etusivun (kuvassa 2) filtteröinti ja hakumahdollisuudet.

Sovelluksen käyttämät Backbone-komponentit käyttävät toistensa väliseen kommunikointiin Backbonen tapahtumia. Komponentit voivat kuunnella oman tai muiden komponenttien tapahtumia. Backbonen komponentit ilmoittavat joistakin valmiiksi määritellyistä tapahtumista, mutta tapahtuman laukaiseminen voidaan myös tehdä komponentissa manuaalisesti trigger-funktiota kutsumalla. Jokaiseen kuunneltavaan tapahtumaan liitetään

callback-funktio, jota kutsutaan tapahtuman yhteydessä. Näkymät voivat kuunnella oman elementtinsä tai sen alaisten elementtien laukaisemia JavaScript-tapahtumia.

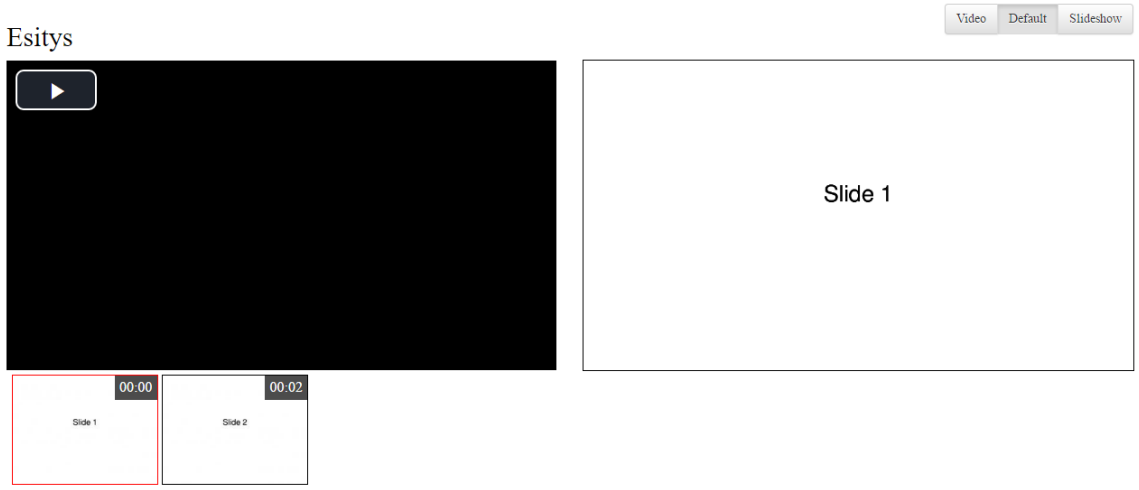
4.2.2 Esitykset

Kuten kaikkien sovelluksen sivujen HTML luodaan dynaamisesti PHP:n ja Twigin avulla. Sivulla oleva JavaScript suurimmaksi osaksi sisällytetään sivun HTML:ään <script>-tagin "src-attribuutin avulla. Pieni määrä JavaScript-koodia myös liitetään sellaisenaan <script>-tagin sisälle. Tämän koodin on tarkoitus asettaa esityksen luontiin tarvittava data JavaScriptin käyttöön jo sivun latauksen yhteydessä.

```
<script>
window.App = window.App || {};
window.App.presentationConfig = window.App.presentationConfig || {};
window.App.presentationConfig.presentation = {{ data.presentation }};
window.App.presentationConfig.slides = {{ data.slides }};
</script>
```

Esimerkkikoodi 2: Datan asettaminen esityksen alustusta varten. Kahdet kaarisulkeet ovat Twigin syntaksia ja niiden sisällä on PHP:ltä peräisin olevaa dataa.

Esityssivulla (kuvassa 3) on useita Backbone-näkymiä. Jokainen DOM-puussa oleva ja sivun JavaScript-toiminnallisuuteen vaikuttava elementti on jonkin Backbone-näkymän alla. Koska Backbone-näkymät voivat kuunnella tapahtumia kaikilta sen alla olevilta elementeiltä, jokainen nappula ei tarvitse omaa näkymää. Tästä esimerkkinä oikeassa yläkulmassa oleva videon ja diaesityksen kokoa muuttava valikko. Nämä nappulat sijaitsevat yhdessä Backbone-näkymässä, joka kuuntelee sen alla olevien elementtien klikkaustapahtumia.



Kuva 3: Esimerkki esityksestä

Sivulla olevista elementeistä olennaisimmat sen toiminnan kannalta ovat video, diaesitys ja esityksen alaosassa sijaitseva navigaatio. Nämä kaikki ovat yhden esitysnäkymän alla. Esitysnäkymä toimii muun muassa videon ja diaesityksen välisenä ohjaimena ja on vastuussa esityksen osien luonnista. Diaesitykseen ja siihen liittyvään navigaatioon on liitetty diakuvamalleista koostuva diakuvakokoelma. Backbone-näkymien sisällä myös käytetään erilaisia ulkopuolisia kirjastoja erilaisten toiminnallisuuksien tuottamiseen.

Videon toistamiseen sovellus käyttää video.js-videosoitinta. Videonäkymä alustaa video.js-soittimen sekä käynnistää kuuntelijan HTML5-videon omalle timeupdate-tapahtumalle. Tämän kuuntelijan tarkoitus on ottaa vastaan normaali JavaScript-tapahtuma ja välittää sen eteenpäin Backbone-tapahtumana. Esitysnäkymä ottaa vastaan tämän videonäkymän lähettämän tapahtuman ja kutsuu diaesitysolion sisältämää kuvanvaihtofunktiota, jolle annetaan parametrina videolta saatu aika. Diaesitys olio hakee kokoelmastaan saadun ajankohdan perusteella oikean diakuvan ja näyttää sen.

Diakuvamalleihin on jokaiseen tallennettu sen oma esitysaika eli se aika, jolloin kuva tulisi näyttää. Diakuvanäkymään on liitetty diakuvakokoelma, joka koostuu diakuvamalleista. Oikean diakuvan hakemista varten ajan perusteella on diaesityskokoelmaan kirjoitettu hakufunktio, jota kutsumalla saadaan haettua aina oikea dia.

Diakuvien esittämiseen ja vaihtamiseen sovellus käyttää diaesityksien esittämiseen tarkoitettua Reveal.js-ohjelmistokirjastoa. Diaesitys-näkymä alustaa Reveal.js-objektin ja käyttää sitä esityksen sivujen hallintaan. Jokaisesta diaesityksen sivusta luodaan pieni

thumbnail-kuva. Näitä thumbnail-kuvia käytetään sivun alaosassa olevaa navigoimista varten ja ne liitetään sivulle sen latauksen yhteydessä. Diaesitysten diat liitetään sivulle jokainen omana tiedostonaan. Jokaisen täysikokoisen diakuva tiedoston liittäminen sivulle sivunlatauksen yhteydessä olisi hidasta. Tämän takia sivulle liitetään aluksi vain muutama ensimmäinen dia ja loput ladataan dynaamisesti esityksen mennessä eteenpäin.

Diaesitystä voi navigoida esityksen alaosassa olevilla pienillä diakuvien thumbnail-kuvilla. Kun navigaatio kuvaa painetaan käynnistyy JavaScriptin click-event. Jokaista thumbnail-kuvaa vastaa erillinen Backbone.js-näkymäinstanssi. Jokainen näistä instansseista kuuntelee omaan kuvaelementtiinsä kohdistuvia click-eventteja. Click-eventin käsittelijä lähettää Backbone-eventtinä tiedon tapahtumasta eteenpäin.

```
Backbone.View.extend({
    //...
    events: {
        'click': function(e) {
            //0 equals left click
            if (e.button === 0) {
                this.trigger('click', this.model.get('id'));
            }
        }
    }
});
```

Esimerkkikoodi 3: Esimerkki Thumbnail-näkymän click-eventin käsittelystä

Esityksen yhteydessä luodaan myös Thumbnails-näkymä, johon jokainen thumbnail-näkymän instanssi liitetään. Tämä näkymä ottaa vastaan sen jäseniensä lähettämät tapahtumat ja ohjaa ne eteenpäin. Esimerkiksi kun navigointi kuvaa klikataan, käynnistyy JavaScriptin click-event. Tämän click-eventin ottaa vastaan se thumbnail-näkymä, joka sisältää klikatun kuvan. Näkymä käynnistää uuden Backbone.js-eventin, jonka taas ottaa vastaan thumbnails-näkymä. Thumbnails-näkymälle on asetettu kuuntelijat kuunteleman kaikkien sen jäsenien tapahtuvien lähettämiä thumbnailClick-tapahtumia.

```
//thumbnails.js
require('collectionView', function(CollectionView){
    CollectionView.extend({
        initialize: function (options) {
            //Kutsutaan CollectionViewin alustus funktiota.
            this.initParent(this, options);

            this.listenToChildEvents(this, 'click', this.handleClick);
    }
})
```

Esimerkkikoodi 4: Esimerkki Thumbnails-näkymän alustuksesta.

Kehityksen aikana myös huomasin, että sovelluksessa oli tarvetta näyttää useassa kohdassa erilaisia kokoelmiin kohdistuvia näkymiä, joiden tarkoitus on näyttää listana kokoelman jäsenet. Monissa ohjelmistokehyksissä on ratkaisu vastaavanlaiseen ongelmaan, mutta Backbone.js:stä sellaista ei löytynyt. Muun muassa thumbnails-näkymä hyödyntää tätä näkymää käyttämällä sen luomista varten CollectionView.extend-funktiota Backbonen View.extend-funktion sijaan, kuten esimerkistä 4 voi nähdä.

```
//collectionView.js
Backbone.View.extend({
    //...
    function(listener, event, callback) {
        _.each(this.childViews, function(el){
            listener.listenTo(el, event, callback);
        });
    }
})
```

Esimerkkikoodi 5: listenToChildEvents toteutus.

4.2.3 Esityksien muokkaaminen

Esityksien muokkaus sivu käyttää suurimmaksi osaksi saman toiminnallisuuden kuin itse esityksen katselu. Esitysten muokkaus sivua varten on luotu erillinen JavaScript-tulokohta. Muokkaussivu sisältää toiminnallisuuden esitysten ajastamiselle. Ajastusta varten sivulle on lisätty ajastusnappula, jota painamalla voidaan yhdistää videon aika sekä valittu dia.

Muokkausnäkymässä on mahdollista kytkeä pois päältä videon ja diaesityksen synkronointi. Normaalisti videon ajan vaihtuessa vaihdetaan myös diaesityksen ja diaesityksen diaa vaihdettaessa hyppää video oikeaan kohtaan. Synkronoinnin pois kytkemisen avulla voidaan klikata oikeaa diaa sekä valita haluttu videon kohta erikseen. Tämän jälkeen ajastusnappia painamalla lähetetään palvelinsovellukselle datan tallentava AJAX-pyyntö.

Esityssivu sisältää kaksi välilehteä. Toinen näistä on lisätoiminnot sisältävä esitysnäkymä ja toinen on lomake esityksen datan muokkaamista varten. Tällä sivulla oleva lomake on sama kuin se, jolla itse esitykset luodaan. Tältä sivulta tapahtuu myös esityksen poistaminen.

5 Yhteenveto ja pohdintaa

Insinöörityön lopputuloksena saatiin toteutettua moderneja web-teknologioita hyödyntävä selaimella käytettävä sovellus, joka sisälsi vaaditun toiminnallisuuden ja jota voi tulevaisuudessa kehittää eteenpäin. Sovellus myös on tehty noudattaen Symfonyn sekä muiden kirjastojen parhaita käytäntöjä. Sovelluksen avulla voi luoda, selata, esittää ja muokata järjestelmän olevia videon ja diaesityksen sisältäviä esityksiä. Sovellus sisältää myös pienimuotoisen käyttäjänhallinnan.

Ohjelmistoja yleisestikin kehitetään senhetkisten ja lähitulevaisuuden tarpeiden mukaisesti. Insinöörityönä tehty sovellus sisältää käyttöliittymän parantelun lisäksi useita jatkokehitysmahdollisuuksia uusien toimintojen luomisen sekä nykyisten toimintojen kehittämisen muodossa. Tiedostojen muuntaminen formaatista toiseen ei ole täydellistä ja esimerkiksi PowerPoint-tiedostoihin tulee tietyissä tapauksissa virheitä muunnoksen tekevistä ohjelmista johtuen. Miettiessä sovelluksen viemistä suuremmalle yleisölle jatkokehitettävää olisi myös käyttäjien ja käyttäjäryhmien käsittelyssä sekä niihin liittyvissä toiminnallisuuksissa. Myös esitysten luonnista voisi tehdä monipuolisempaa esimerkiksi tukemalla vaihtoehtoisia lähteitä esityskomponenteille nykyisen tiedostolatauksen lisäksi. Itse esityksiin voisi lisätä toiminnallisuutta kuten diaesitysten tekstiä etsivän hakukentän tai suurempia toimintoja kuten vaihtoehtoisia esityskomponentteja tai esimerkiksi esitysten suoratoiston, joiden avulla voisi myös itse sovelluksen käyttötarkoituksia laajentaa.

Sovelluksen koodi on pyritty kirjoittamaan joustavaksi. Symfonyn parhaiden käytäntöjen mukaan sovelluksen koodia on kirjoitettu mahdollisimman paljon Symfonyn palveluihin, joka ainakin teoriassa tekee siitä uudelleenkäytettävämpää. JavaScript koodi on kirjoitettu EcmaScript 5 -spesifikaatiota noudattaen, joten se tullaan luultavasti jossain vaiheessa muuttamaan noudattamaan uudempia EcmaScriptin spesifikaatioita. Sovelluksen JavaScript-koodi kuitenkin hyödyntää nyt moduuleja ja Webpackia, joten ainakaan niiden kannalta itse JavaScriptin arkkitehtuuria ei tarvitse muuttaa. JavaScript kehittyy kuitenkin kokoajan ja jossain vaiheessa uudempia tekniikoita voidaan käyttöliittymässäkin hyödyntää.

PHP-ohjelmistokehykseksi Symfony oli hyvä valinta. Se on yksi käytetyimmistä PHP-ohjelmistokehyksistä ja sen osia käytetään monissa muissa projekteissa, muun muassa tällä hetkellä suosituimmassa PHP-ohjelmistokehyksessä Laravelissa. Symfonysta voi myös halutessaan poistaa tarpeettomat ja räätälöidä se omiin tarpeisiin sopivaksi. Se sisältää myös erittäin kattavan dokumentaation.

Backbone.js ja Webpack teki JavaScript-koodista rakenteellisempaa verrattuna esimerkiksi pääasiassa jQueryyn pohjautuvaan JavaScript-koodiin. Opin myös Backbonen ja Webpackin avulla käyttämään JavaScript-moduuleita sekä tutustuin JavaScriptin AMD- ja CommonJS-moduulitekniikoihin. Backbone.js on jo melko vanha ohjelmistokirjasto. Kirjoitushetkellä sen viimeisimmän version 1.3.3 julkaisusta on kulunut jo yli vuosi. Backbone.js:ään perustuen on tehty kehittyneempiä ohjelmistokehyksiä, joista suurin on Marionette.js. Koska Backbone.js on tiedostokooltaan melko pieni, se sopii rakenteellista JavaScript-koodia tarvitseville sivuille, jotka eivät kuitenkaan ole liian monimutkaisia. Monimutkaisempia JavaScriptiin pohjautuvia sivuja varten on nykyään monia Backbone.js:ää kehittyneempiä työkaluja.

Lähteet

1 Symfony - Wikipedia. 2017. Verkkodokumentti. <https://en.wikipedia.org/wiki/Symfony>. Luettu 17.04.2017.

2 Symfony 3.0.0 released. 2015. Verkkodokumentti. <https://symfony.com/blog/symfony-3-0-0-released>. Luettu 5.5.2017.

3 Symfony - The Architecture (The Symfony Quick Tour). 2017. Verkkodokumentti. <https://symfony.com/doc/current/quick_tour/the_architecture.html>. Luettu 15.5.2017.

4 Symfony - The Book. 2016. Verkkodokumentti. symfony.com. <https://symfony.com/pdf/Symfony_book_3.1.pdf>. Luettu 5.5.2017.

5 Sébastien Armand. Extending Symfony 2 Web Application Framework. Packt Publishing. 2014.

6 Symfony - ParamConverter. 2015. Verkkodokumentti. <https://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/annotations/converters.html>. Luettu 16.5.2017.

7 Object-relational mapping - Wikipedia. 2017. Verkkodokumentti. <https://en.wikipedia.org/wiki/Object-relational_mapping>. Luettu 15.05.2017.

8 Doctrine ORM - 1. Architecture. 2015. Verkkodokumentti. <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/architecture.html>. Luettu 14.05.2017.

9 Doctrine ORM - 7. Working with Objects. 2017. Verkkodokumentti. <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/working-with-objects.html>. Luettu 14.05.2017.

10 Lazy loading - Wikipedia. 2017. Verkkodokumentti. <https://en.wikipedia.org/wiki/Lazy_loading>. Luettu 17.5.2017.

11 Doctrine ORM - 14. Doctrine Query Language. 2017. Verkkodokumentti. <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html>. Luettu 14.05.2017.

12 Doctrine ORM - 6. Inheritance Mapping. 2015. Verkkodokumentti. <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/inheritance-mapping.html>. Luettu 14.05.2017.

13 Doctrine ORM - 15. The QueryBuilder. 2017. Verkkodokumentti. <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>. Luettu 14.05.2017.

14 Swarnendu De. Backbone.js Patterns and Best Practices. Packt Publishing. 2014.

15 Backbone. 2016. Verkkodokumentti. backbone.js.org. <http://backbonejs.org>. Luettu 2.5.2017.

16 Webpack - CommonJS. 2017. Verkkodokumentti. <https://webpack.github.io/docs/commonjs.html>. Luettu 15.05.2017.

17 Webpack - AMD. 2017. Verkkodokumentti. <https://webpack.github.io/docs/amd.html>. Luettu 15.05.2017.

18 Webpack - Code Splitting. 2017. Verkkodokumentti. <https://webpack.js.org/guides/code-splitting>. Luettu 15.05.2017.

19 Webpack - Getting Started. 2016. Verkkodokumentti. <https://webpack.js.org/get-started>. Luettu 28.4.2017.

20 Video.js Github. 2017. Verkkodokumentti. <https://github.com/videojs/video.js/tree/v5.19.2>. Luettu 15.05.2017.

21 Getting Started with Video.js. 2016. Verkkodokumentti. <http://videojs.com/getting-started>. Luettu 15.04.2017.