Haaga-Helia
ammattikorkeakoulu Oy

REST-rajapinnan kehittäminen Symfony2-sovelluskehyksellä

Ville Kumpula

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

16.05.2017

Tekijä(t)
Ville Kumpula

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
REST-rajapinnan kehittäminen Symfony2-sovelluskehyksellä

Sivu- ja liitesivumäärä
27+3

Opinnäytetyön otsikko englanniksi
Developing a RESTful API with Symfony2 framework

Opinnäytetyö käsittelee millaista oli rakentaa REST-rajapinta Symfony2 sovelluskehyksellä, ja kuinka nykyinen tuotannonohjausjärjestelmä integroitiin uuteen rajapintaan. Opinnäytetyö toteutettiin ajalla elokuu 2016 – toukokuu 2017.

Työn toimeksiantaja on turva-alan pk-yritys Vantaalta. Yrityksellä on oma tuotannonohjausjärjestelmä, joka sijaitsee korkean turvatason tuotantoalueella. Tästä syystä ulkomaailmasta ei saa olla suoria yhteyksiä järjestelmään. Tämä vaikeuttaa toimistolla työskentelevien työntekijöiden arkea, sillä he joutuvat menemään toimistolta tuotantoalueelle, jos haluavat saada reaaliaikaista tietoa, kuten varasto- tai laskutusraportteja tuotannonohjausjärjestelmästä.

Työn tavoitteena oli rakentaa tiedostopohjainen tietokannan replikaatiokirjasto, joka replikoi tuotannonohjausjärjestelmän tietokannasta sallitut tiedot toimistolle, sekä REST-rajapinta replikoitujen tietojen tueksi. Tämän rajapinnan päälle päälle voidaan rakentaa erilaisia käyttöliittymiä toimistotyöntekijöiden avuksi.

Raportin empiirisessä osiossa käydään läpi REST-arkkitehtuurin perusteita, sekä käydään läpi Symfony2-sovelluskehyksen tärkeimmät ominaisuudet, joita työssä käytettiin.

Työstä rajattiin pois käyttöliittymien kehitys, sillä REST-arkkitehtuurin avulla käyttöliittymien kehitys pystyttiin jättämään myöhempään vaiheeseen, ja opinnäytetyön aikana keskityttiin ainoastaan rajapinnan kehittämiseen.

Raportin lopuksi käydään läpi mahdollisia jatkokehitysideoita, sekä pohditaan mitä projektin läpivieminen opetti käytännössä.

Työn tuloksena syntyi toimiva tietokannan replikointi sekä REST-rajapinta, jotka ovat päivittäisessä käytössä niiden päälle kehitettyjen käyttöliittymien kautta.

Asiasanat
Symfony, MySQL, ORM, REST, ohjelmistokehys, ohjelmistoarkkitehtuuri

Sisällys

Käsitteitä

DMZ	(Demilitarized Zone). Aliverkko joka yhdistää yrityksen oman järjestelmän turvattomampaan alueeseen, esimerkiksi internetiin.
GIT	Suosittu Linus Torvaldsin kehittämä versionhallintajärjestelmä
JSON	(JavaScript Object Notation) Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
JWT	(JSON Web Token) Poletteihin perustuva autentikointimetodi
HSA	(High Security Area). Alue jossa on erittäin korkeat turvamääräykset.
LDAP	(Lightweight Directory Access Protocol) Hakemistopalvelujen käyttöön tarkoitettu verkkoprotokolla.
MySQL	Oraclen omistama suosittu relaatiotietokanta ojelmisto.
ORM	(Object-relational mapping). Tekniikka jolla tietokantaa kuvataan ja hallitaan oliopohjaisesti.
PCI	(Payment Card Industry Security Standards Council) Maailmanlaajuinen järjestö joka vastaa maksualan tietoturvastandardien kehittämisestä
PDO	(PHP Data Objects). PHP ohjelmointikielen kanssa käytetty tietokanta ohjelmointimalli.
PGP	(Pretty Good Privacy). Suosittu tiedon salausjärjestelmä.

PHP	(PHP: Hypertext Preprocessor). Suosittu web-palvelinympäristössä käytetty ohjelmointikieli.
REST	(Representational State Transfer). Arkkitehtuurimalli ohjelmistorajapintojen toteuttamiseen.
SYMFONY	Avoimen lähdekoodin PHP sovelluskehys.
TAR	(Tape Archiver). Paketointityökalu jolla yhdistetään monia tiedostoja yhden tiedoston sisälle.
XML	(Extensible Markup Language) Standardi, jonka avulla tiedot voidaan esittää rakenteisena dokumenttina.

1 Johdanto

Yritys jolle projekti toteutetaan, on turvallisuusalan yritys pääkaupunkiseudulta. Yrityksen pääasiallinen palvelu on valmistaa älykortteja pankeille.

Yrityksellä on käytössään sisäisesti kehittämä ja ylläpitämä tuotannonohjausjärjestelmä, joka prosessoi asiakkailta tulevat korttien tilausaineistot omaan tietokantaansa, ja tarjoaa monenlaisia palveluita tuotannon kortinvalmistuksen tarpeisiin, kuten tuotantokoneiden hallintaa, tilausten seurantaa ja varastoraportteja.

Yrityksellä on erillinen korkean turvallisuuden alue (HSA) jossa älykortteja valmistetaan. Alueen sisällä noudatetaan PCI Card Production Security Requirements manuaalin määrittelemiä loogisia ja fyysisiä turvavaatimuksia.

Tuotantoalueen tietoverkko on eriytetty ulkomaailmasta demilitarisoitun verkon (DMZ) avulla, tämän johdosta kaikki liikenne HSA-alueelta sisään tai ulos täytyy terminoida DMZ-verkkoon. Koska suorat yhteydet toimistolta HSA-alueelle ovat estetty, kaikki tuotantodata mitä tuotannonohjausjärjestelmä tallentaa tietokantaansa, on saatavilla vain tuotanto-alueella, jolloin toimiston työntekijöiden täytyy mennä tuotantoalueelle, jos haluavat saada reaaliaikaista tietoa tuotannosta.

Tämän opinnäytetyön aiheena on luoda tietokannan replikointipalvelu, joka siirtää kaiken sallittavan tuotantodatan tuotantoverkosta toimistoverkkoon tiedostopohjaisesti, ja Symfony2-sovellus joka prosessoi replikaatiotiedostot omaan tietokantaansa, sekä tarjoaa REST-rajapinnan, jonka päälle voidaan rakentaa erilaisia käyttöliittymiä.

Opinnäytetyössä käydään läpi sovelluksen kehittämisen vaiheet ja kompastuskivet. Samalla käydään läpi myös eri tekniikat, joita sovelluksen kehittämisessä käytetään.

Opinnäytetyö kiinnostaa todennäköisesti ihmisiä, joilla on jo jonkinverran kokemusta ohjelmistokehityksestä.

2 Ohjelmistorajapinta

Ohjelmistorajapinnat mahdollistavat eri ohjelmien välisen kommunikaation. Aikaisemmin yritysten ohjelmistorajapinnat olivat usein suljettuja, eikä ulkopuoliset päässeet niihin käsiksi. Viime vuosien aikana monet yritykset ovat kuitenkin huomanneet, että avoimista ohjelmarajapinnoista voi olla suurta hyötyä liiketoiminnan näkökulmasta. Niiden avulla yritys voi kaupallistaa heidän digitaalista omaisuuttaan sekä yhdistää eri kanavien ja laitteiden asiakkaat. (Patni 2017, 1)

Sisäisten ohjelmistorajapintojen hyötynä on että ne usein parantavat kehitystiimien tuottavuutta maksimoimalla uudenllenkäytettävyyttä ja parantamalla johdonmukaisuutta uusia ohjelmistoja kehittäessä.

Julkiset ohjelmistorajapinnat antavat lisäarvoa liiketoiminnalle, antamalla kolmansille osapuolille mahdollisuuden parantaa yrityksen palveluita ja mahdollisesti houkuttelemalla lisää asiakkaita palvelun käyttäjiksi.
(Patni 2017, 1)

2.1 REST-arkkitehtuuri

REST-arkkitehtuurin alkuperäinen määrittelijä on Roy Fielding, joka on yksi HTTP-protokollan luojista, sekä Apache palvelinsovelluksen kehittäjistä. REST-termin Fielding määritteli vuonna 2000 Kalifornian yliopistossa kirjoittamassaan väitöskirjassa.

Lyhyesti sanottuna REST on arkkitehtuurillinen tyyli, joka on kehitetty auttamaan hajautettujen järjestelmien luomista ja organisointia.
(Doglio 2015, 1)

REST-arkkitehtuurilla pyritään parempaan suorituskykyyn, skaalautuvuuteen, rajapinnan yksinkertaisuuteen, muokattavuuteen, siirrettävyyteen, luotettavuuteen sekä näkyvyyteen.

REST-arkkitehtuuri koostuu kuudesta eri rajoitteesta, joita järjestelmän täytyy noudattaa. Rajoitteet käydään läpi seuraavissa alaluvuissa.
(Patni 2017, 1)

2.1.1 Asiakas-palvelin

Asiakas-palvelin rajauksessa ydinteema on verkon tehtävien jakaminen.

Verkko on asiakas-palvelin järjestelmä, jossa asiakkaalla ja palvelimella on omat tehtävänsä suoritettavana. Ne voidaan toteuttaa ja ottaa käyttöön erillään, käyttäen mitä tahansa kieltä tai teknologiaa, kunhan ne mukautuvat verkon yhtenäisiin rajapintoihin. (Patni 2017, 1)

2.1.2 Yhtenäinen rajapinta

Pitämällä komponenttien välillä yhtenäisen rajapinnan, helpotat asiakkaan vuorovaikutusta rajapintasi kanssa. Yhtenäinen rajapinta myös helpottaa asiakas-sovellusten kehittämistä, kun kehittäjällä on käytössä selkeät säännöt jota noudattaa.

Yhtenäisen rajapinnan haittapuolena voidaan pitää sitä että se huonontaa tiedonsiirron tehokkuutta, varsinkin jos saatavilla olisi tehokkampia kommunikointitapoja.

Yhtenäinen rajapinta saavutetaan noudattamalla neljää rajausta, jotka ovat:

- Resurssien tunnistaminen
- Resurssien manipulointi esitysten kautta
- Itseselitteiset viestit
- Hypermedia sovelluksen tilakoneena

(Doglio 2015, 1)

2.1.3 Kerroksittainen järjestelmä

Kerroksittaista järjestelmää käytetään yleensä parantamaan verkkopohjaisen järjestelmän tietoturvaa, vastausten tallettamista välimuistiin sekä kuormituksen tasaamista.

Kerroksittaisen järjestelmän rajaukset mahdollistavat esimerkiksi välityspalvelimien ja yhdyskäytävien läpinäkyvän käyttöönoton asiakkaan ja palvelimen välillä.
(Patni 2017, 1)

2.1.4 Välimuisti

Välimuistin käyttö on internetin yksi tärkeimmistä rajauksista. Välimuistin rajaukset kertovat asiakaalle, voidaanko kutsuun tullut vastaus laittaa välimuistiin myöhempää käyttö varten.

Välimuistin käyttö auttaa pienentämään asiakkaan näkemää viivettä, parantamaan sovelluksen saatavuutta ja luotettavuutta sekä kontrolloimaan palvelimen kuormaa. (Patni 2017, 1)

2.1.5 Tilattomuus

Tilattomuudella tarkoitetaan, että palvelin ei ole pakotettu muistamaan asiakas-sovellusten tilaa muistissaan. Siitä syystä jokaisen asiakkaan on annettava tarvittava tieto tilastaan palvelimelle joka pyynnön yhteydessä.

Tilattomuudella saadaan kuormaa pois palvelimelta, jolloin palvelin voi palvella isompaa määrää asiakkaita hidastumatta.
(Patni 2017, 1)

2.1.6 Ladattava koodi

Ladattavalla koodilla tarkoitetaan, että palvelin voi lähettää asiakkaalle suoritettavan ohjelman, kuten JavaScript-koodia tai Java-sovelluksen. Ladattavan koodin haittana on että se usein pakottaa asiakkaan käyttämään samoja teknologioita palvelimen kanssa, koska asiakkaan täytyy osata suorittaa palvelimen lähettämä koodi. Tästä syystä ladattavan koodin rajaus on valinnainen.
(Patni 2017, 1)

2.1.7 HATEOAS

Viimeinen REST-arkkitehtuurin rajaus on HATEOAS eli Hypermedia As The Engine Of Application State. Kun kehitetään rajapintaa hyödyntäen HATEOAS-rajausta, palvelinpuolen logiikka voi muuttua milloin tahansa, eikä se vaikuta asiakas sovelluksiin.

Hypermedia on asiakaskeskeinen lähestymistapa, jossa dokumenttiin lisätään tuki lisätä linkkejä muihin palveluihin sekä lisäinformaatioon. Yhtenä esimerkkinä hypermediasta ja hyperlinkeistä on, kun kerätään tietoa yhdestä resurssista monesta eri lähteestä. Tämä tieto voi olla lähtöisin yrityksen omasta verkosta tai julkisesta verkosta.

Esimerkki:

```
<podcast id="111">
  <customer>http://customers.myintranet.com/customers/1</customers>
  <link>http://podcast.com/myfirstpodcast</link>
  <description> This is my first podcast </description>
</podcast>
```

(Patni 2017, 1)

3 Symfony

Symfony on avoimen lähdekoodin PHP-sovelluskehys, jolla tuotetaan web-applikaatioita. Alun perin se suunniteltiin ranskalaisen SensioLabs yrityksen omaan käyttöön, mutta vuonna 2005 se julkaistiin avoimen lähdekoodin versiona MIT lisenssin alle, ja se on nykyään yksi suosituimmista PHP-kielen sovelluskehyksistä.

Nykyään Symfonya kehittää laaja yhteisö, jota SensioLabs tukee. Symfony tarjoaa kehittäjille monia hyödyllisiä resursseja, kuten laajan dokumentaation, yhteisön tarjoaman tuen eri kanavien kautta, kuten irc, slack sekä sähköposti. SensioLabs tarjoaa myös konsultointia ja koulutusta yrityksille.

Tuhannet eri verkkopalvelut ovat kehitetty Symfony-sovelluskehyksen päälle, lisäksi Symfonyn komponentteja käytetään myös monissa muissa PHP-sovelluskehyksissä, kuten Drupal, phpBB, Laravel sekä ezPublish (Symfony 2017b).

Sovelluskehyksen käyttö antaa paremmat työkalut luoda sovellus, mikä noudattaa liiketoimintasääntöjä, on hyvin jaotelu, ylläpidettävä sekä päivitettävä.

Sovelluskehyksen käyttö nopeuttaa sovelluksen kehittämistä, koska monet tarvittavat peruskomponentit on jo valmiiksi kehitetty, ja näin ollen sovelluskehittäjä voi keskittyä muihin osa-alueisiin (Symfony 2017i).

3.1 Symfony-komponentit

Symfony Components kokoelma pitää sisällään monia eri komponentteja, joista kehittäjä voi valita haluamansa käyttöön omaan projektiinsa.

Suosittuja Symfony-komponentteja ovat muun muassa seuraavat:

- Console Component, joka helpottaa komentoriviltä suoritettavien toimintojen kehittämistä, siirtämällä perustoimintojen kehittämisen pois kehittäjän harteilta, jolloin kehittäjälle jää enemmän aikaa kehittää itse toiminnallisuutta.

- Debug Component, joka tarjoaa kehittäjälle työkalut jolla tulkita ohjelman suorittamista helpommin.

- Event Dispatcher Component, joka tarjoaa työkalut jolla sovelluksen eri komponentit voivat keskustella keskenään lähettämällä tapahtumia ja kuuntelemalla niitä.

- HttpKernel Component, joka tarjoaa jäsennellyn prosessin, jolla http-kutsu muunnetaan vastaukseksi käyttämällä hyväksi Event Dispatcher komponenttia.

- YAML Component, joka tulkitsee YAML muotoisia dokumentteja. Komponentti prosessoi YAML tiedoston ja muuntaa sen PHP-taulukkomuotoon.

  (Popular Symfony components used by CMSes. 2015)

Tämän opinnäytetyön kirjoittamisen aikana, Symfony-komponentteja on saatavilla 34 kappaletta.

3.2 Symfony-sovelluskehys

Symfony-sovelluskehys on PHP-kirjasto, joka pitää sisällään useita valmiiksi valittuja Symfony-komponentteja, sekä valittuja kolmannen osapuolen kirjastoja, kuten SwiftMailer sähköpostien lähettämistä varten, Monolog lokien kirjoittamista varten, PHPUnit yksikkötestauksia varten sekä Doctrine tietokannan hallintaa varten.

Symfony-sovelluskehys pitää myös sisällään järkevät konfiguraatiotiedostot, jotka kokoavat eri komponentit yhdeksi kokonaisuudeksi. (The Book, 13)

Symfony-sovelluskehyksestä on opinnäytetyön kirjoittamisen aikana saatavilla kaksi tuettua versiota, versio 2.8 jolle on luvattu tuki vuoden 2018 loppuun, sekä versio 3.2 jossa on uusimmat ominaisuudet, mutta tuki loppuu 2018 vuoden alussa.

Symfony käyttää versioinnissaan aikaan perustuvaa mallia. Uusi minor Symfony-versio julkaistaan kuuden kuukauden välein, toukokuussa ja marraskuussa. Kahden vuoden välein julkaistaan Long-Term-Support (LTS) versio, jolle luvataan kolmen vuoden tuki virhekorjauksille sekä neljän vuoden tuki turvapäivityksille (Symfony 2017g).

3.3 Symfony-sovelluksen toiminta

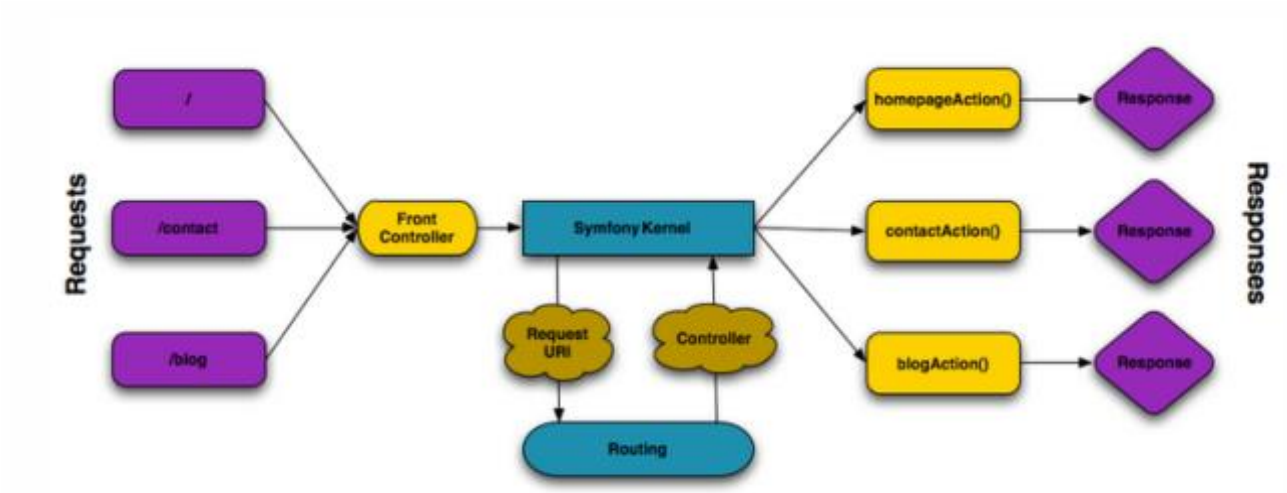
Symfony-sovelluksen tärkein ominaisuus on ottaa vastaan HTTP-kutsuja ja lähettää niille oikeanlainen vastaus. Tätä ominaisuutta varten Symfonyn HTTP-foundation komponentti tarjoaa kaksi eri objektia, joita käytetään HTTP-kutsujen ja vastauksien käsittelyyn.

Ensimmäinen on Request objekti, joka on oliopohjainen esitys HTTP-kutsusta. Request-objektin avulla pääset helposti käsiksi kutsun kaikkiin eri ominaisuuksiin, kuten kekseihin, otsakkeisiin, lähetystapaan sekä lomakkeen tietoihin.

Request objektilla on myös monia turvallisuuteen liittyviä metodeja, kuten isSecure() jolla tarkistetaan onko kutsu lähetetty salattua yhteyttä pitkin.
(Symfony 2017e).

Toinen objekti on Response, joka on oliopohjainen esitys HTTP-vastauksesta. Response-objektin avulla sovelluksessa voidaan muodostaa vastaus asiakkaalle käyttämällä oliopohjaisia menetelmiä.

Symfony-sovellus noudattaa samaa kaavaa kaikkien HTTP-kutsujen kanssa (kuva 1). Sisääntulevat HTTP-kutsut tulkataan routing komponentissa, joka välittää ne halutuille PHP-metodeille, joiden tehtävä on palauttaa Response objekti.



Kuva 1. Symfony-sovelluksen toiminta (The Book, 24)

Jokainen URL-osoite jolle halutaan jotakin toiminnallisuuta, tulee olla määriteltynä routing-komponentin konfiguraatiotiedostossa, joka kartoittaa kyseisen URL-osoitteen tiettyyn PHP-metodiin.

Nämä PHP-metodit sijaitsevat luokan sisällä, jota kutsutaan kontrolleriksi. Jokaisen näistä metodeista on tarkoitus käyttää apunaan Request-objektia, sekä muita tarvittavia Symfony-komponentteja, ja luoda niiden avulla Response-objekti. Tämän jälkeen Response-objekti palautetaan takaisin käyttäjälle. (The Book, 11)

3.4 Bundle-järjestelmä

Bundlet muodostavat Symfony sovelluksen. Kaikki Symfonyn perusominaisuudet ovat jaoteltu bundlejen sisään, ja toiminnallisuus jota kehittäjä luo, sijoitetaan bundlen sisään.

Bundle on yksinkertaisuudessaan kansio, joka pitää sisällään kaikki tarvittavat PHP-tiedostot, CSS-tyylitiedostot, JavaScript-tiedostot, mallinteet, yksikkötestit sekä muut tiedostot joilla toteutetaan sovelluksen tietty ominaisuus.

Bundlejen avulla voidaan jakaa omaa koodia helposti muille kehittäjille, tai voidaan itse ottaa helposti käyttöön kolmannen osapuolen kehittämä bundle. Bundlejen avulla voidaan myös rajoittaa mitä ominaisuuksia halutaan ottaa käyttöön omaan applikaatioon.

Bundlet otetaan käyttöön rekisteröimällä ne Symfony-sovelluksen AppKernel-luokassa (Symfony 2017f).

Symfony-sovelluskehyksen mukana tulee monia bundleja, kuten:

- FrameworkBundle, joka tarjoaa Symfony-sovelluskehyksen perustoiminnot yhdistämällä kaikki Symfony-komponentit saman bundlen alle. (Symfony 2017d).
- DoctrineBundle, joka integroi suositun Doctrine kirjaston käytettäväksi Symfonyn kanssa
- MonologBundle, joka integroi suositun Monolog kirjaston käytettäväksi Symfonyn kanssa

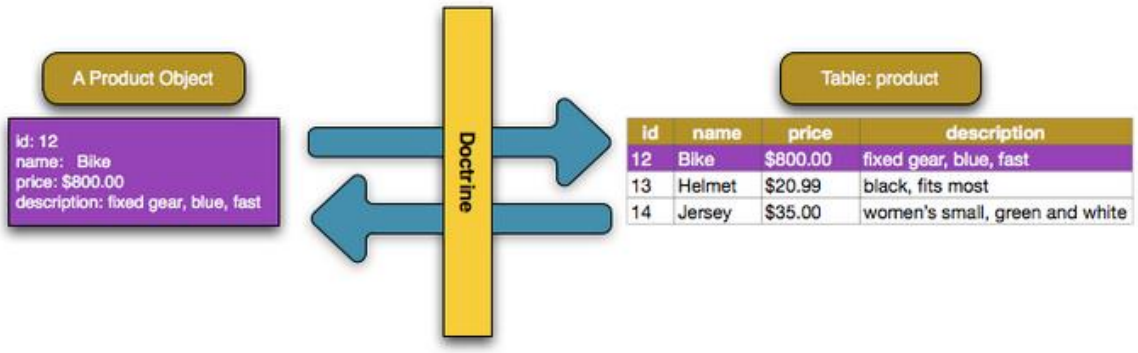
3.5 Doctrine

Yksi sovelluskehityksen haastavimmista tehtävistä on luoda toiminnalisuudet tietokantaa varten. Symfony-sovelluskehyksessä itsessään ei ole valmista komponenttia jolla toteuttaa kyseinen tehtävä, mutta se tarjoaa valmiin bundlen kolmannen osapuolen kehittämään Object-relational mapping (ORM) kirjastoon nimeltä Doctrine.

Doctrinen tehtävä on antaa sen käyttäjälle tehokkaat työkalut, jolla tehdä vuorovaikutus tietokannan kanssa helpoksi ja tehokkaaksi (Symfony 2017c).

Doctrine käsitelee PHP-luokkia ja objekteja, kuten ne olisivat tietokantatauluja ja tietueita (kuva 2). Tällä tavoin käyttäjän ei tarvitse käyttää ollenkaan SQL-kyselyitä. Doctrine

rakentaa kaikki kyselyt puolestasi ja tämä tekee kehittämisestä nopeampaa ja hauskempaa (Salehi 2016, 26).



Kuva 2. Doctrine ORM toiminta (Symfony 2017c).

Doctrine toimii monen eri relaatiotietokanta-alustan kanssa, käyttämällä sen Database Abstraction Layer (DBAL) kerrosta. DBALin nykyinen versio 2 tukee seuraavia tietokanta-alustoja:

- MySQL
- Oracle
- Microsoft SQL Server
- PostgreSQL
- SAP Sybase SQL Anywhere
- SQLite
- Drizzle

(Doctrine 2017b).

Doctrine tarjoaa käyttäjälle valmiiksi luonti, luku, poisto ja päivitys operaatiot, mutta jos käyttäjä haluaa tehdä monimutkaisia kyselyitä, se tapahtuu Doctrine Query Language (DQL) kielen avulla. DQL-kieli on samankaltainen kuin SQL-kieli, mutta erona on se, että tietokantakolumnien sijasta kysytään PHP-luokan ominaisuuksia (kuva 3).
(Symfony 2017c).

```php
$em = $this->getDoctrine()->getManager();
$query = $em->createQuery(
    'SELECT p
    FROM AppBundle:Product p
    WHERE p.price > :price
    ORDER BY p.price ASC'
)->setParameter('price', 19.99);

$products = $query->getResult();
```

Kuva 3. Esimerkki DQL-kyselystä

Vielä helpompaa kuin kirjoittaa DQL-kyselyitä, on käyttää Doctrinen tarjoamaa QueryBuilder rajapintaa, joka tarjoaa kehittäjälle monia luokkia ja metodeita, joiden avulla kyselyn voi muodostaa oliomaisesti (kuva 4).

```php
public function getCardsDoneBetweenTimes(\DateTime $from, \DateTime $to, $pStep = 702)
{
    return $this->createQueryBuilder('ml')
        ->select("sum(ml.cardsDone)")
        ->leftJoin(MachineConfiguration::class, 'mc', 'WITH', 'ml.machineId = mc.id')
        ->andWhere('ml.createTime >= :from')
        ->andWhere('ml.createTime <= :to')
        ->andWhere('mc.processStep = :pStep')
        ->setParameter(':from', $from->format('Y-m-d H:i:s'))
        ->setParameter(':to', $to->format('Y-m-d H:i:s'))
        ->setParameter(':pStep', $pStep)
        ->getQuery()
        ->getSingleScalarResult();
}
```

Kuva 4. Esimerkki QueryBuilder kyselystä

QueryBuilderin avulla voidaan palauttaa tieto joko suoraan olioina, tai PHP-taulukko-muodossa. (Doctrine 2017a).

3.6 Konfigurointi

Symfonyn yksi vahvuus on sen helppokäyttöiset konfiguraatiotoedostot. Konfiguraatiotiedostot sijaitsevat hakemistossa app/config ja ne voidaan konfiguroida YAML-formaatissa, XML-formaatissa tai PHP-muodossa.

Symfony tukee eri konfiguraatiotiedostoja kehitys ja tuotantokäytössä, kehitysympäristön konfiguraatiot löytyvät config_dev.yml tiedostosta, ja tuotantokonfiguraatiot config_prod.yml tiedostosta. Kummankin ympäristön yhteiset konfiguraatiot löytyvät config.yml tiedostosta. Oletuksena konfiguraatioihin käytetään YAML-formaattia (kuva 5).

```
framework:
    #esi:             ~
    #translator:      { fallbacks: ["%locale%"] }
    secret:          "%secret%"
    router:
        resource: "%kernel.root_dir%/config/routing.yml"
        strict_requirements: ~
    form:            ~
    csrf_protection: ~
    validation:      { enable_annotations: true }
```

Kuva 5. Config.yml esimerkki

3.7 Symfonyn vahvuudet ja heikkoudet

Symfonyn vahvuuksia ovat esimerkiksi:

Maine
Symfonyn on ottanut käyttöön alan ammattilaiset heti sen julkaisusta lähtien 2005 vuonna. Symfony on tänä päivänä tunnettu ja arvostettu kansainvälisesti. Symfonylla on myös erittäin aktiivinen kehittäjä-yhteisö, joka työskentelee projektin kanssa päivittäin (Symfony 2017a).

Pysyvyys
Symfonyny takana on yritys nimeltä SensioLabs. SensioLabs on perustettu yli 18 vuotta sitten, ja sillä on erittäin hyvä maine ja paljon suuria asiakkaita. Symfony on nykyään avoimen lähdekoodin ohjelmisto ja se on SensioLabsin päivittäisessä käytössä heidän asiakasprojekteissa.

SensioLabs myös ylläpitää Symfonyn Long Term Support ohjelmaa, jonka avulla kehittäjä voi olla varma ettei hänen käyttämä sovelluskehys menetä tukeansa yllättäen, tai muuta rajapintojaan siten, että vanhat sovellukset lakkaavat toimivasta (Symfony 2017a).

Viitteet

Monet tunnetut yritykset käyttävät Symfonya omissa projekteissaan, kuten Yahoo, DailyMotion tai Opensky.com.

Symfonya myös käytetään monissa tunnetuissa ohjelmistoissa kuten phpBB ja Drupal. (Symfony 2017a).

Innovaatio

Symfony pitää sisällään kaiken mitä sovelluskehykseltä voi odottaa, kehittämisen nopeuden, joustavuuden, uudelleenkäytettävät komponentit jne.

Symfonyn kehittäjät ovat myös ottaneet paljon vaikutteita muiden ohjelmointikielien parhaista käytännöistä ja tuoneet ne PHP-maailmaan (Symfony 2017a).

Resurssit

Symfony-sovelluksen kehittämisen kanssa ei jää helposti pulaan, sillä yhteisö tarjoaa erinomaiset tukipalstat. SensioLabs järjestää myös konsultointia ja koulutuksia yrtiyksille.

Symfonyn koodi on myös erittäin hyvin dokumentoitu ja sitä on helppo seurata. (Symfony 2017a).

Symfonyn heikkoutena voi pitää sitä, että sen oppimiseen menee yleensä pitkään.

Symfonyn dokumentaatio on yleisesti ottaen hyvää, mutta usein oikean dokumentin etsimiseen kuluu turhan paljon aikaa, sillä jotkut dokumenteista on sijoiteltu monen linkin taakse.

4 Sovelluksen suunnittelu

Sovelluksen suunnittelu aloitettiin käymällä eri sidosryhmien kanssa läpi, mitä sovellukselta odotetaan, ja mitkä ominaisuudet ovat prioriteettina korkeimmalla, ja mitkä matalimmalla.

Sen jälkeen tehtiin projektisuunnitelma, johon kaikki ominaisuudet kirjattiin. Tätä suunnitelmaa noudatettiin projektin toteutumisvaiheessa.

4.1 Vaatimukset ja toiminnot

Projektin tavoite oli luoda tiedostopohjainen tietokannan replikaatio tuotantoverkosta toimistoverkkoon, sekä Symfony-sovellus joka prosessoi replikaatio tiedostot omaan tietokantaansa, ja tarjoaa REST-rajapinnan kaikista tietokantatauluista.

Tämän REST-rajapinnan päälle tullaan myöhemmin rakentamaan erilaisia käyttöliittymiä toimistotyöntekijöiden käytettäväksi. Käyttöliittymät ei kuitenkaan kuulunut tähän opinnäytetyöhön, joten niitä ei käsitellä tässä raportissa.

Tietokantannan replikoinnin aikaväliksi määritettiin aluksi kerran päivässä, ja myöhemmin se voidaan nopeuttaa kerran tunnissa replikoitavaksi.

4.2 Tietokanta

Tietokannaksi projektille valittiin MySQL, lähinnä siitä syystä, että se on erittäin suosittu tietokantaratkaisu web-sovelluksissa, ja työpaikallamme on hyvät kokemukset sen käytöstä.

Myös nykyinen tuotannonohjausjärjestelmä käyttää MySQL tietokantaa, joten se oli luonnollinen valinta myös toimistoversiota varten.

4.3 Kehitysympäristö

Kehitysalustaksi valittiin Ubuntu 16.04, jota ajetaan Vmware-virtualisointisovelluksella. Tähän päädyttiin koska Ubuntulla on hyvä maine ja se soveltuu erinomaisesti PHP-sovelluksen ylläpitoon.

Kehitystyökaluksi valittiin JetBrains-nimisen yrityksen kehittämä PHPstorm, joka on räätälöity PHP-kehitystä ajatellen. Se tarjoaa koodin analysointia, virheiden tulkintaa, koodin automaattista formatointia ja monia muita hyödyllisiä ominaisuuksia. Se tukee myös kolmannen osapuolten kehittämiä liitännäisiä, ja esimerkiksi Symfonylle löytyi liitännäinen, joka toi monia hyödyllisiä ominaisuuksia kyseisen sovelluskehyksen kehitykseen.

4.4 Tuotantoympäristö

Tuotantoympäristö on miltei identtinen kehitysympäristön kanssa, ainoana erona on että Symfony-sovellus on asetettu siellä toimimaan tuotantotilassa, kun kehityksessä sitä ajetaan kehitystilassa, jolloin Symfony kirjoittaa paljon koodin analysointidataa levylle, joka helpottaa virheiden paikannusta niiden sattuessa. Tuotannossa tätä ei kuitenkaan haluta käyttöön, sillä se hidastaa sovellusta huomattavasti ja lokit vievät paljon levytilaa.

4.5 Projektinhallinta

Projektinhallintasovelluksena käytettiin Redmine sovellusta. Redmine on projektinhallintasovellus, jota käytetään selaimen kautta. Sovellukseen voi lisätä omia projekteja, jonka sisälle luodaan muutospyyntöjä (tickets), jotka ohjataan tietylle henkilölle tehtäväksi.

Muuntospyyntöihin voidaan määrittää tehtävä asia, aikamääreet, prioriteetti jne. Redmine tukee myös integraatiota GIT-versionhalinnan kanssa, jonka avulla versionhallintaan tehdyt muutokset (commitit) yhdistyvät Redminen muutospyyntöön, kunhan GIT-muutoksen viestiin merkitään Redmine muutospyynnön numero.

4.6 Versionhallinta

Versionhallintatyökaluksi valittiin Linus Torvaldsin kehittämä Git. Git oli jo ennestään tuttu itselleni, ja työpaikallamme sitä käytetään useissa projekteissa, joten se oli luonteva valinta myös tälle projektille.

Git komentoja voi ajaa joko komentoriviltä, tai käyttää kolmannen osapuolen tarjoamia käyttöliittymäsovelluksia.

Git tukee kehityshaarojen luomista, jolloin jokainen ominaisuus voidaan eriyttää omaksi kehityshaaraksi, joka helpottaa useamman ihmisen työskentelyä yhden projektin kanssa.

4.7 Aikataulu

Projektilla ei ollut mitään tiukkaa aikataulumäärettä, sillä se kehitettiin yrityksen omaan käyttöön, ja muut asiakasprojektit menivät tämän projektin edelle, jolloin tätä projektia kehitettiin aina kun asiakasprojekteja ei ollut.

Välillä kuitenkin asetimme joillekin ominaisuuksille toive päivämäärän, johon mennessä kyseisen ominaisuuden toivottiin olevan valmis.

5 Sovelluksen toteuttaminen

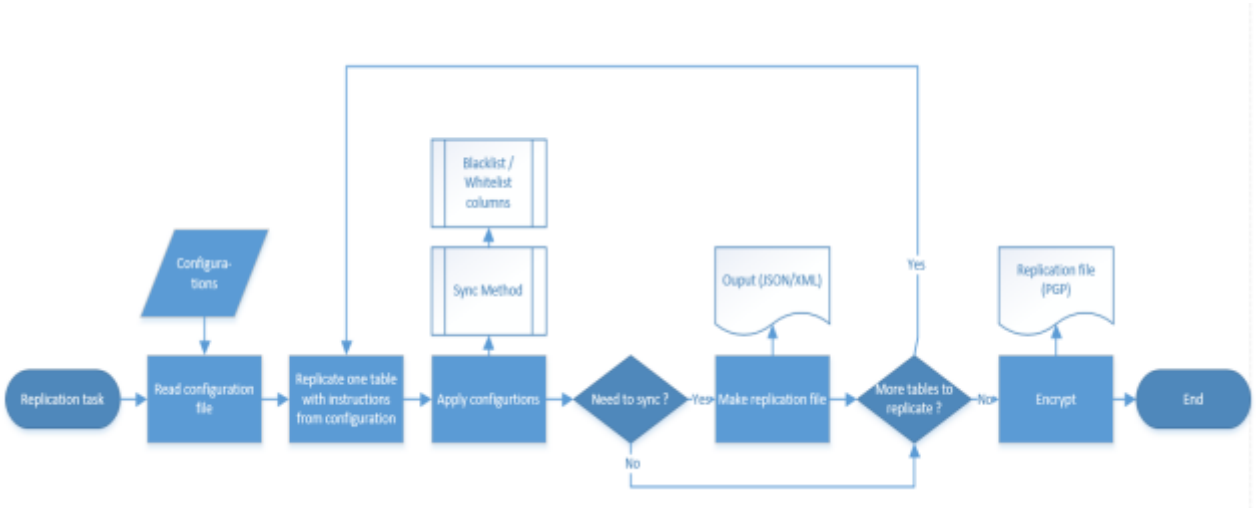
Projektin toteuttaminen aloitettiin kehittämällä tietokannan replikaatio ensiksi, jonka jälkeen siirryttiin kehittämään REST-rajapintaa replikoitujen tietojen päälle. Replikaatiota suunnitellessa käytiin ensiksi kaikki tuotannonohjausjärjestelmän tietokantataulut läpi, ja päätettiin mitä tietoa siirretään toimistolle, ja mitä ei.

Ennen kuin alettiin kehittämään replikointia, tutkittiin myös valmiita tietokannan replikointiohjelmia ja tutkittiin olisiko jokin niistä soveltuva tarkoitukseemme. Emme kuitenkaan löytäneet mitään sellaista ohjelmistoa, joka olisi tyydyttänyt kaikki tarpeemme.

5.1 Tietokannan replikaatio (HSA-alue)

Tietokannan replikaatio täytyi tehdä tietodostopohjaisena, sillä suoria yhteyksiä toimistoverkosta tuotantoverkkoon ei sallita. Tietokanta pitää sisällään myös dataa, joita ei PCI-vaatimuksien mukaan saa viedä pois HSA-alueelta.

Tästä syystä päädyimme tekemään täysin kustomoidun tietokannan replikointikirjaston (kuva 6).



Kuva 6. Replikointikirjaston toiminta

Kirjaston avulla voidaan määrittää, mitkä tietokantataulut replikoidaan ja miten. Kirjastolle kehitettiin neljä erilaista taulun replikointimenetelmää jotka ovat seuraavat:

Full table menentelmällä taulu kopioidaan joka kerta uudestaan, välittämättä siitä onko data sen sisällä muuttunut vai ei. Tätä menetelmää käytetään pienien taulujen replikointiinm kuten käyttäjätaulu.

Incremental id menetelmällä replikointikirjasto kirjoittaa joka iteraatiolla tietokantaan muistiin viimeisimmän primary key arvon, ja seuraavalla kerralla replikoidaan vain sitä arvoa uudemman tietueet. Tätä menetelmää käytetään lokitaulujen replikointiin, koska tiedetään että vanhat tiedot eivät tule enää muuttumaan.

Sync completed menetelmää käytetään tilaustaulujen replikointiin. Siinä joka replikoinnin iteraatiokierroksella kirjoitetaan tietokantaan sellaisten tilauksien id numerot jotka eivät tule enää ikinä muuttumaan. Seuraavalla kerralla ne id:t joille on merkitty sync completed, ohitetaan eikä niitä replikoida enää udestaan.

Hash check menetelmässä katsotaan tietokantataulun tiiviste, ja verrataan sitä edellisen replikointikerran tiivisteeseen. Tätä ominaisuutta ei käytetä mihinkään, mutta sitä voitaisiin käyttää sellaisiin tauluihin jotka ovat melko isoja, ja muuttuvat harvoin.

Kun replikaatiometodi on määritelty, voidaan määrittää mitä kolumneja replikoidaan. Oletusarvoisesti replikoidaan kaikki kentät, jotka eivät ole salattuja. Halutessa voidaan määrittää joko ne kentät, jotka halutaan replikoida käyttämällä Whitelist metodia. Toinen vaihtoehto on määrittää ne kentät joita ei halua replikoida, käyttämällä Blacklist metodia.

Kaikki asiakasdata on alkuperäisessä tietokannassa salattuna, ja niiden kolumnien tietotyyppi on BLOB. Kirjasto suodattaa nämä kolumnit automaattisesti pois replikoinnista, jotta kyseisiä tietoja ei siirretä pois tuotantoalueelta.

Lopuksi määritellään, halutaanko tiedot kirjoittaa levylle XML vai JSON-muodossa.

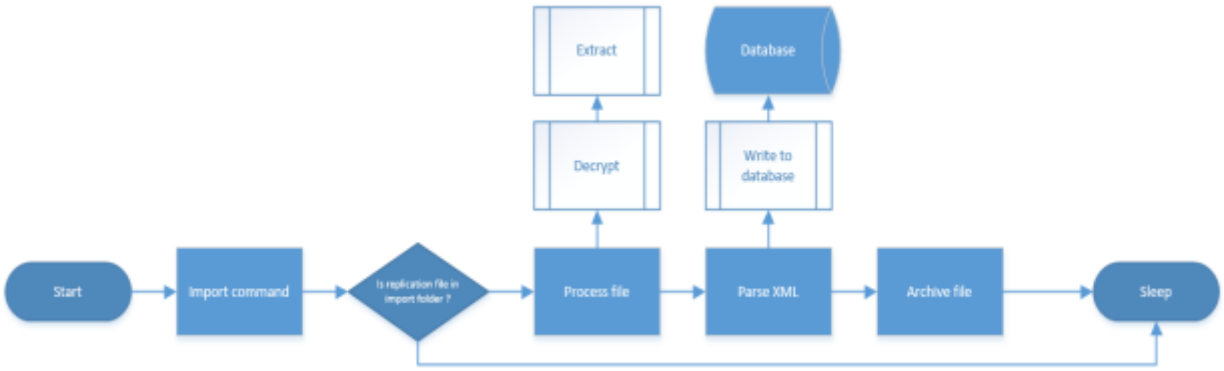
Jokaisesta taulusta joka replikoidaan, kirjoitetaan XML tai JSON-tiedosto väliaikaiseen kansioon, ja kun kaikki taulut on käyty läpi, yhdistetään tiedostot TAR-paketiksi, joka salataan PGP-avaimella.

Tämän jälkeen paketti siirretään automaattisesti tuotannosta toimistolle DMZ-verkon kautta.

5.2 Tietokannan replikaatio (Toimisto)

Toimiston puolella käytetään hyväksi Symfonyn-Console komponenttia, jolla voidaan ajaa PHP:n Command Line Interface (CLI) ympäristössä Symfony-sovellusta.

Replikaatiota varten kehitettiin uusi konsoli komento, joka huolehtii replikaatiotiedostojen prosessoinnista tietokantaan (kuva 7).



Kuva 7. Import komennon toiminta

Komento ajetaan kerran minuutissa käyttämällä Unixin CRON-ajastuspalvelua ja komennon tehtävä on tarkistaa, onko uusia replikaatio tiedostoja saatavilla määritetyssä hakemistossa. Jos tiedostoja löytyy, katsotaan niiden tiedostonimen perusteella, millaisesta tiedostosta on kyse. Kun tiedosto on identifioitu, puretaan sen salaus PGP-avaimella, ja aletaan tulkita sen sisältämiä XML-tiedostoja.

Jokainen XML-tiedosto pitää sisällään tietokantaan lisättäviä tai päivitettäviä rivejä. Aluksi ajatuksena oli muuntaa jokainen rivi tietokantataulua vastaavaan Doctrine-objektiin, mutta myöhemmin sen huomattiin tuovan suorituskykyongelmia, joten prosessoinnissa päätyttiin käyttämään SQL-lauseita ja PHP:n PDO objektia tietueiden päivittämiseksi tietokantaan.

Kun kaikki XML-tiedostot ovat prosessoitu, siirretään replikaatio tiedosto arkistohakemistoon, ja päivitetään tietokantaan tieto onnistuneesta replikaation prosessoinnista. Tätä tietoa hyväksikäyttämällä voidaan asiakas sovellukselle kertoa, kuinka tuoretta tietokannan data on (liite 2, liite 3).

5.3 REST-rajapinta

Jotta replikoituja tietoja voitaisiin käyttää turvallisesti ja helposti eri käyttöliittymäkerroksissa, kehitettiin Symfony-sovelluskehyksellä REST-rajapinta, jonka avulla saadaan tietokannan tietoja ulos JSON-muodossa.

Symfonylle on olemassa valmiita Bundleja jotka helpottavat REST-rajapinnan kehittämistä, mutta päädyimme käyttämään Symfonyn omia komponentteja rajapinnan rakentamiseen, koska uuden Bundlen opettelu olisi vienyt huomattavasti aikaa.

5.3.1 Reititys

Reititystä varten käytetään Symfonyn omaa routing-komponenttia.

REST-rajapinnassa reitityksellä on tärkeä rooli, sen avulla muut sovelluskehittäjät tietävät tarkalleen, miten he käyttävät haluamiaan resursseja (taulukko 1).

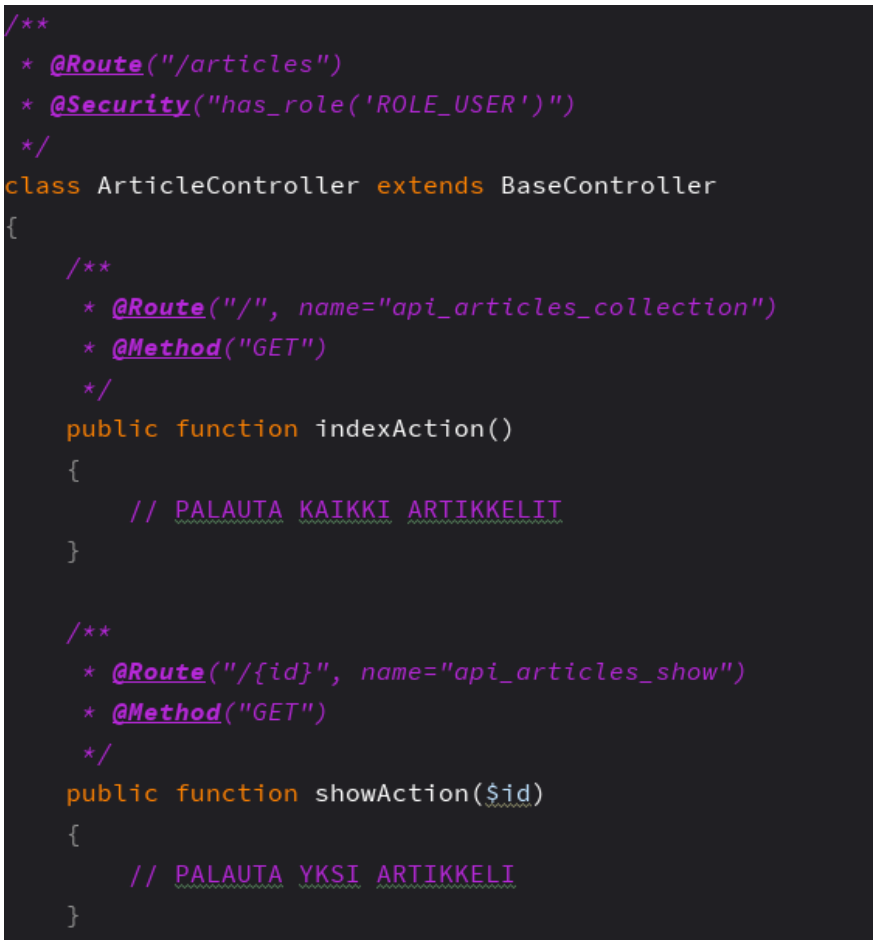
Taulukko 1. REST-rajapinnan reitit esimerkki

HTTP Verbi	URL	Kontrollerin Metodi	Käyttö
GET	/articles	indexAction	Listaa kaikki artikkelit
GET	/articles/:id	showAction	Listaa tietty artikkeli
POST	/articles	createAction	Lisää artikkeli
PUT	/articles/:id	updateAction	Päivitä artikkeli
DELETE	/articles /:id	deleteAction	Poista artikkeli

Symfonyssa reititys voidaan toteuttaa joko kommenttikenttinä (annotation), YAML-formaatissa, XML-formaatissa tai PHP-luokkana.

Itse koin kommenttikentät helpoimpana, sillä niitä käyttämällä kehittäjä näkee nopeasti mitä reittiä tietty funktio kontrollerissa vastaa.

Annotaatioita käytetään lisäämällä Symfony-kontrollerin metodille kommenttikenttä, johon merkataan @route kommentti, sekä tarvittaessa muita routea tukevia kommentteja, kuten @method tai @security (kuva 8).



```
/**
 * @Route("/articles")
 * @Security("has_role('ROLE_USER')")
 */
class ArticleController extends BaseController
{
    /**
     * @Route("/", name="api_articles_collection")
     * @Method("GET")
     */
    public function indexAction()
    {
        // PALAUTA KAIKKI ARTIKKELIT
    }

    /**
     * @Route("/{id}", name="api_articles_show")
     * @Method("GET")
     */
    public function showAction($id)
    {
        // PALAUTA YKSI ARTIKKELI
    }
```

Kuva 8. ArticleController reititys

Päädyimme tekemään jokaiselle tietokantataululle oman kontrollerin, jonka sisällä on kyseiselle taululle vastaavat reitit. Tämä oli mielestäni selkeämpi tapa, kuin tehdä yksi kontrolleri joka vastaisi koko rajapinnan reitityksestä.

5.3.2 Autentikointi

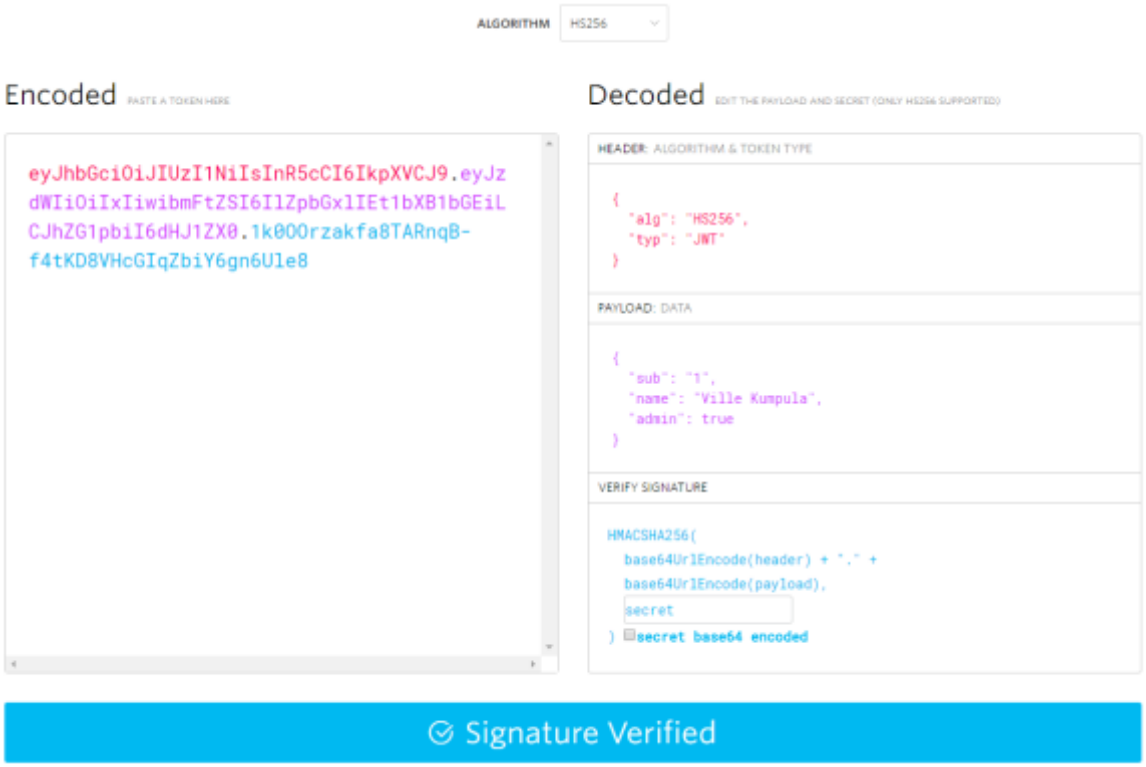
Rajapintaan haluttiin sisällyttää käyttäjien todennus, jotta vain rajatut henkilöt pääsevät käyttämään rajapinnan tarjoamia tietoja.

Todennukseen päädyttiin käyttämään JWT metodia. JWT todennuksessa ei käytetä keksejä (cookies) käyttäjän tunnistamiseen, vaan palvelin lähettää JSON-muotoisen poletin, jonka asiakas palauttaa jokaisen pyynnön yhteydessä.

Poletti pitää sisällään otsakkeen, tietosisällön sekä allekirjoituksen (kuva 9). Otsake pitää sisällään tiedon minkä tyyppisestä poletista on kyse, ja mitä algoritmia on käytetty sen salaamiseen.

Tietosisältö voi pitää sisällään mitä tahansa tietoa, mutta yleensä siellä pidetään ainakin käyttäjänumero, sekä muuta käyttäjään liittyvää sisältöä. Tässä rajapinnassa tietosisältönä on käyttäjänumero, käyttäjän etu- ja sukunimi, käyttäjän viimeinen kirjautumisaika, poletin voimassaoloaika sekä käyttäjän rooli. Käyttäjän rooli ja nimi laitettiin polettiin mukaan, jotta käyttöliittymässä voidaan näyttää perustiedot käyttäjästä.

Allekirjoitus muodostetaan salaamalla halutulla salausalgoritmilla otsake, tietosisältö sekä salaisuus (secret).



Kuva 9. JWT esimerkki

Jotta asiakas voi saada JWT-poletin, täytyy hänen tunnistautua yrityksen LDAP-palvelimen kautta.

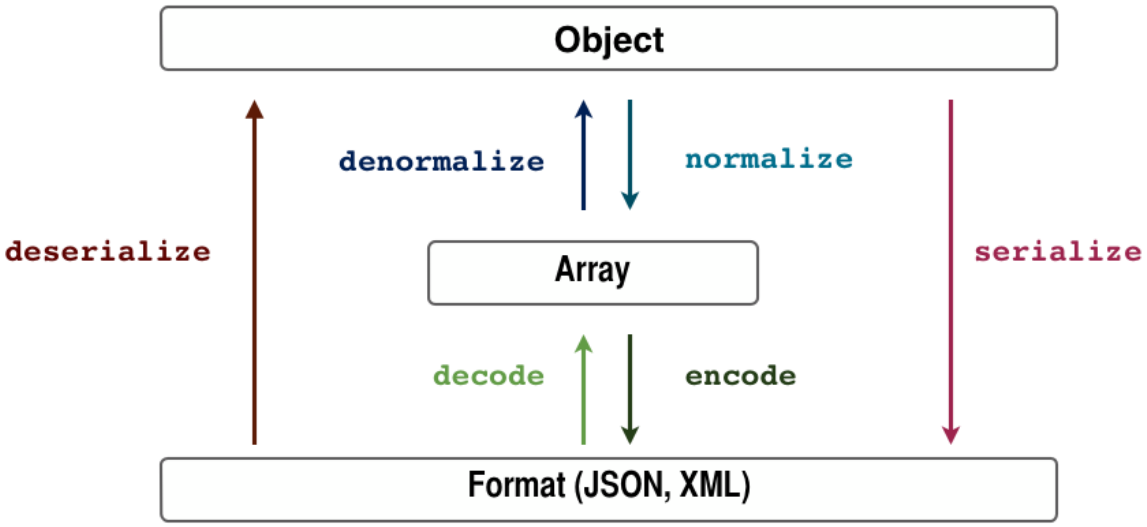
Autentikointia varten luotiin oma kontrolleri ja autentikointi reitti, johon asiakas lähettää Active Directory käyttäjänimensä sekä salasanan. Onnistuneen autentikoinnin jälkeen asiakkaalle palautetaan JWT-poletti, joka on voimassa tunnin ajan. Kun poletti vanhenee, on asiakkaan haettava uutta polettia tunnistautumalla uudestaan.

5.3.3 JsonResponse & Serializer

Symfony tarjoaa valmiina JsonResponse-luokan, jonka avulla voidaan palauttaa sovelluksesta JSON-muotoista dataa asiakkaalle.

JsonResponse ottaa parametrina palautettavan datan, sekä statuskoodin jonka oletusarvo on 200 eli OK. JsonResponse-luokka osaa kääntää PHP-taulukon tai tekstin JSON-muotoon, mutta objekteja varten joudutaan käyttämään Serializer komponenttia.

Serializer komponentin tehtävä on muuttaa objektit toiseen muotoon, kuten JSON, XML, YAML tai toisinpäin (kuva 10).



Kuva 10. Serializer komponentin toiminta (Symfony 2017h).

Yksinkertaisuuden vuoksi sovellukseen kehitettiin ApiResponse-metodi, joka muuntaa objektit PHP-taulukko muotoon Serializer komponentin avulla ja palauttaa käyttäjälle JSON-muotoisen palautteen (liite 1).

Virheen sattuessa ApiResponse-metodi ottaa virheen vastaan ja muokkaa sen JSON-muotoon joka palautetaan asiakas-sovellukselle.

5.3.4 Lokitus

Sovelluksen haluttiin kirjaavan tiettyjä asioita lokiin, kuten virhetilanteet, kirjautumiset sekä replikaation tilannetietoja.

Symfonylla ei ole omaa komponenttia lokitusta varten, mutta Symfony-sovelluskehyksen mukana tulee kolmannen osapuolen kehittämä kirjasto Monolog, sekä MonologBundle joka integroi kirjaston Symfony-sovelluskehykseen.

MonologBundlen avulla voidaan lokeja kirjoittaa helposti moneen eri paikkaan, kuten tiedostoihin, tietokantoihin, syslogiin, sähköpostiin tai web palveluihin.

Tässä projektissa päätimme kirjoittaa kaikki normaalit tapahtumat tietokantaan, ja virheiden sattuessa jäljitys-lokit kirjoitetaan tiedostoon palvelimelle. Jäljityslokeja ei haluttu kirjoittaa tietokantaan, sillä jos tietokantayhteys ei olisi saatavilla, ei jäljityslokeja voisi kirjoittaa ja se voisi vaikeuttaa vian etsimistä huomattavasti.

5.4 Tuotos ja jatkokehitys

Ensimmäinen toimiva versio sovelluksesta saatiin noin kuuden kuukauden päästä aloittamisesta. Sen jälkeen päästiin kehittämään ensimmäistä käyttöliittymäsovellusta rajapinnan päälle.

Tätä opinnäytetyötä kirjottaessa replikointi ja rajapinta ovat olleet käytössä noin 4 kuukautta, ja replikointi toiminut erittäin hyvin.

Tällä hetkellä replikointi kuitenkin tapahtuu vain kerran päivässä, sillä replikoinnin suorituskyvyssä on vielä parantamisen varaa. Myös replikoinnin virheidenhallintaa pitää kehittää enemmän. Jos yksi replikaatiotiedosto ei muodostu oikein, esimerkiksi virhetilanteen vuoksi, joudutaan usein replikointi aloittamaan täysin alusta.

Ensimmäisen replikoinnin muistinkäytön kanssa oli isoja ongelmia projektin alkuvaiheilla, sillä ensimmäisellä kerralla kirjasto joutuu replikoimaan miljoonia rivejä tietoa, jolloin koko tauluja ei voida lukea kerralla muistiin, vaan lukeminen ja kirjoittaminen on jouduttu pilkkomaan pienempiin osiin. Tämä hidastaa operaatiota, josta syystä ensimmäinen replikointi kestää yli tunnin.

Seuraavat replikointikerrat kestävät enää muutaman minuutin inkrementaalisen replikoinnin ansiosta, mutta tulevaisuudessa sen toivottaisiin tippuvan alle minuuttiin, parempien replikointialgoritmien avulla.

Erään rajapinnan päälle rakennetun käyttöliittymän avulla toimistotyöntekijät saavat nykyään reaaliaikaisia raportteja sekä tuotantostatistiikkaa omalle työpisteelleen

toimistolle. Tästä on ollut suurta hyötyä muun muassa varastonhallinnassa sekä laskutuksessa.

Toinen käyttöliittymä joka on kehitetty, on informaatiotelevisio toimiston aulassa. Siitä työntekijät näkevät esimerkiksi viimeisimmät saapuneet tilaukset, tämän sekä viime kuukauden tuotantomäärät sekä suosituimmat korttityypit viimeisen 30 päivän aikana.

Kaikki data jota nämä sovellukset käyttävät, haetaan tämän opinnäytetyön tuloksena syntyneen replikoinnin ja REST-rajapinnan kautta.

6 Pohdinta

Projekti oli erittäin mielenkiintoinen ja haastava. Lopputuloksen olen tyytyväinen, ja on mukavaa, että jatkokehitysideoita on tullut jatkuvasti.

Opinnäytetyön alussa meni todella paljon aikaa Symfony-sovelluskehystä opiskellessa, sillä se oli täysin uutta itselleni. Yrityksessämme ei myöskään ollut ketään joka olisi käyttänyt kyseistä sovelluskehystä aikaisemmin. Nyt kun projekti on valmis, olen kuitenkin tyytyväinen sovelluskehyksen valintaan, sillä se osoittautui olemaan maineensa veroinen ja sen kanssa on ilo työskennellä. Aion käyttää Symfonya myös tulevaisuuden projekteissa.

Projekti opetti myös monia uusia näkökulmia projektinhallinasta. Redminen GIT-integraatio osoittautui erittäin hyödylliseksi ominaisuudeksi, ja se helpotti huomattavasti projektin eri ominaisuuksien kehityksen hallintaa.

Myös tietokantareplikaation kehittäminen oli opettavainen kokemus, ja sen tuomat muistinkäyttö- sekä performanssiongelmat pakottivat paneutumaan syvemmin MySQL tietokannan rakenteeseen sekä PHP:n muistinkäytön hallintaan.

Aikataulun kanssa oli välillä ongelmia, sillä muut työpaikan projektit veivät paljon aikaa. Sovellus saatiin kuitenkin valmiiksi, ja olemme rakentaneet sen päälle jo kaksi eri käyttöliittymää jota toimiston työntekijät käyttävät päivittäin.

REST-rajapinta tuntui hyvältä ratkaisulta tähän projektiin, sillä se antaa hyvän mahdollisuuden kehittää käyttöliittymää sekä palvelinsovellusta eri aikaan.

Lähteet

Doctrine 2017a. The QueryBuilder. Luettavissa: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html. Luettu 3.5.2017.

Doctrine 2017b. Introduction. Luettavissa: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/introduction.html. Luettu 4.7.2017.

Doglio, F. 2015. Pro REST API Development with Node.js. USA: Apress

Welling, L & Thomson, L. 2016. PHP and MySQL Web Development. USA: Addison-Wesley Professional.

PCI Card Production Logical Security Requirements 2015. Luettavissa: https://www.pcisecuritystandards.org/documents/PCI_Card_Production_Logical_Security_Requirements_v1-1_March_2015_final.pdf. Luettu 4.5.2017.

Patni, S. 2017. Pro RESTful APIs. USA: Apress.

Salehi, S. 2016. Mastering Symfony. USA: Packt Publishing.

Symfony.fi. 2015. Popular Symfony components used by CMSes.Luettavissa: https://www.symfony.fi/entry/popular-symfony-components-used-by-cms. Luettu 4.5.2017

Symfony. 2016. The Book. Luettavissa: https://symfony.com/pdf/Symfony_book_2.8.pdf. Luettu 3.5.2017.

Symfony 2017a. 6 good reasons to use Symfony. Luettavissa: http://symfony.com/six-good-reasons. Luettu 3.5.2017.

Symfony 2017b. About. Luettavissa: https://symfony.com/about. Luettu 3.5.2017.

Symfony 2017c. Databases and the Doctrine ORM. Luettavissa: https://symfony.com/doc/current/doctrine.html#main. Luettu 3.5.2017.

Symfony 2017d. FrameworkBundle Configuration. Luettavissa: http://symfony.com/doc/current/reference/configuration/framework.html. Luettu 3.5.2017.

Symfony 2017e. HTTP-fundamentals. Luettavissa: http://symfony.com/doc/current/introduction/http_fundamentals.html Luettu 4.7.2017.

Symfony 2017f. The Bundle System. Luettavissa: http://symfony.com/doc/current/bundles.html. Luettu 3.5.2017.

Symfony 2017g. The Release Process. Luettavissa: http://symfony.com/doc/current/contributing/community/releases.html. Luettu 3.5.2017.

Symfony 2017h. The Serializer Component. Luettavissa: http://symfony.com/doc/current/components/serializer.html. Luettu 3.4.2017.

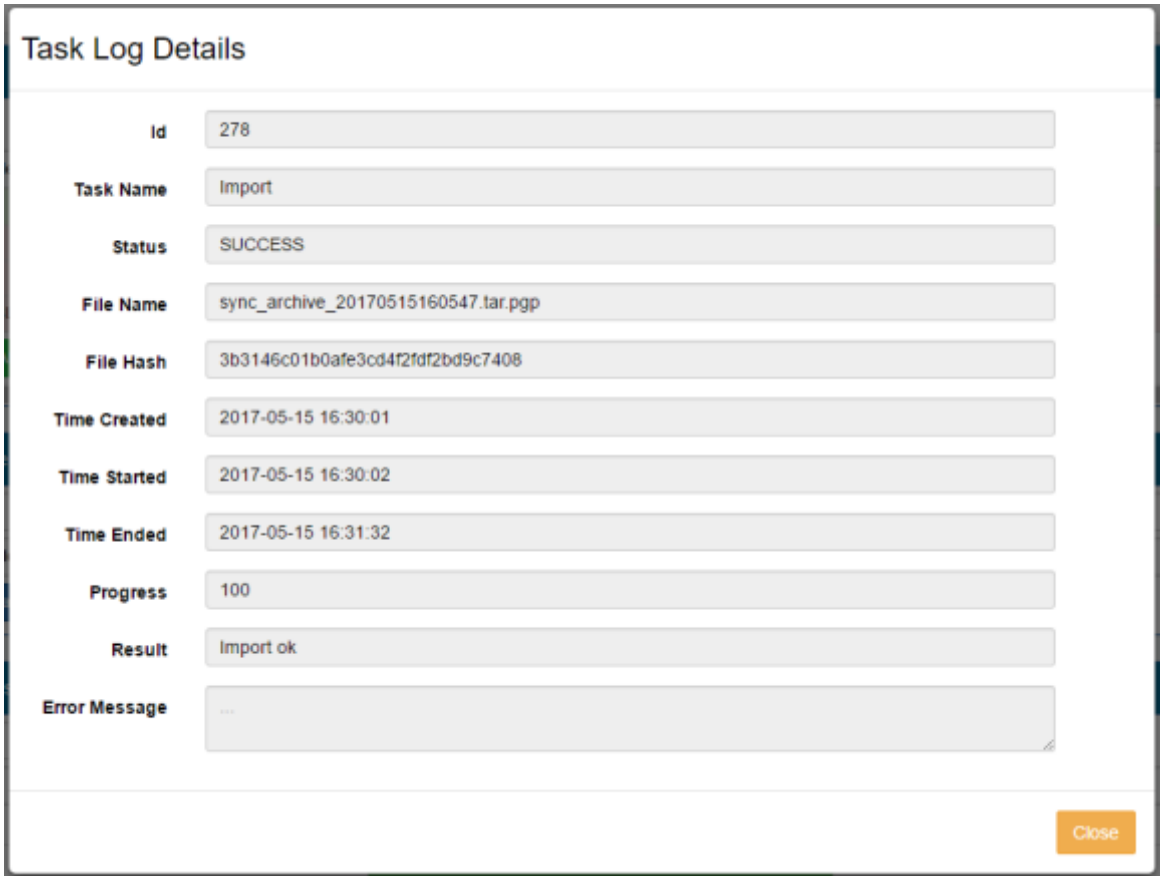
Symfony 2017i. Why should I use a framework? Luettavissa: http://symfony.com/why-use-a-framework. Luettu 3.5.2017.

Liitteet

Liite 1. REST-rajapinnan osoitteen /api/logs JSON-vastaus


```
    {
      "id": 2128,
      "channel": "taskbundle",
      "level": 200,
      "message": "Running command taskrunner:import",
      "time": 1494801131,
      "ip_address": "192.168.1.1",
      "user": "ville",
      "verboseLevel": "INFO"
    },
    {
      "id": 2127,
      "channel": "taskbundle",
      "level": 200,
      "message": "Stopped task with id: 1",
      "time": 1494801130,
      "ip_address": "192.168.1.1",
      "user": "ville",
      "verboseLevel": "INFO"
    },
    {
      "id": 2126,
      "channel": "officebundle",
      "level": 200,
      "message": "User [ville] logged in",
      "time": 1494801122,
      "ip_address": "192.168.1.1",
      "user": "---",
      "verboseLevel": "INFO"
    },
    {
      "id": 2125,
      "channel": "officebundle",
      "level": 200,
      "message": "User [ville] logged in",
      "time": 1494592779,
      "ip_address": "192.168.1.42",
      "user": "---",
      "verboseLevel": "INFO"
    },
    {
      "id": 2124,
      "channel": "officebundle",
      "level": 200,
      "message": "User [tero] logged in",
      "time": 1494592118,
      "ip_address": "192.168.1.21",
      "user": "---",
      "verboseLevel": "INFO"
    }
  ],
  "total": 2142,
  "count": 20,
  "_links": {
    "self": "/server/web/app.php/api/logs/?page=1",
    "first": "/server/web/app.php/api/logs/?page=1",
    "last": "/server/web/app.php/api/logs/?page=108",
    "currPage": 1,
    "maxPages": 108,
    "next": "/server/web/app.php/api/logs/?page=2"
  }
}
```

Liite 2. Replikoinnin statistiikkaa



Liite 3. Replikointilokien näkymä käyttöliittymässä