Tuomas Kiiskinen

TOIMINNANOHJAUSJÄRJESTELMÄN TOTEUTTAMINEN SYMFONY-OHJELMISTOKEHYKSELLÄ

Opinnäytetyö

Tietojenkäsittelyn koulutus

2020

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Tuomas Kiiskinen	Tradenomi (AMK)	Toukokuu 2020
Opinnäytetyön nimi		
Toiminnanohjausjärjestelmän toteuttaminen Symfony-ohjelmistokehyksellä		31 sivua
Toimeksiantaja		
Advison Solutions Oy		
Ohjaaja		
Janne Turunen		

Tiivistelmä

Opinnäytetyössä käsitellään toiminnanohjausjärjestelmän kehitysprosessia Symfony-PHP-ohjelmistokehyksellä. Aluksi työssä esitellään työn kehityksessä käytettyjä menetelmiä ja tekniikoita, minkä jälkeen käydään läpi sovelluksen pääasiallisten toimintojen tuottamista vaiheittain. Tuotettuja ominaisuuksia tarkastellaan käyttöliittymään, koodiin sekä Symfony-komponentteihin ja jQuery- ja JavaScript-liitännäisiin tutustuen. Lopuksi arvioidaan työn onnistumista ja lopputuloksia.

Opinnäytetyön tavoitteena oli tuottaa toimeksiantajan asiakkaalle toimiva ja tarpeita vastaava sovellus. Sovellus on laajalti räätälöity yrityksen vaatimuksia ajatellen

Sovellus toteutettiin asiakastyönä Etelä-Suomessa toimivalle rakennusalan yritykselle. Tuotetun järjestelmän tarkoitus on pyrkiä helpottamaan yrityksen töiden ja resurssien jakoa, parantamaan viestintää sekä työtuntien ja tehtävien seurantaa, automatisoimalla ja korvaamalla vanhat, paperilla ja laskentataulukko-ohjelmistoilla toimivat, toimintatavat.

Lopputuloksena opinnäytetyön sovellus saatiin testaus- ja viimeistelyvaiheeseen. Sovelluksen päätoiminnot saatiin toteutettua, minkä jälkeen työstettäväksi jäi pienet korjaukset, kuten käyttöliittymän ratkaisut ja niissä sovelletut termit sekä testauksessa paljastuvat mahdolliset puutteet.

Toteutukseen valitut tekniikat soveltuivat toiminnanohjausjärjestelmän tuottamiseen mainiosti. Symfonyn komponenttikirjasto tarjosi kattavasti työkaluja sovelluksen ominaisuuksien toteuttamiseen ja automatisoi sovelluksen perustarpeita, kuten käyttöturvallisuutta.

Asiasanat

PHP, ohjelmointi, ohjelmistokehitys, toiminnanohjaus

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Tuomas Kiiskinen	Bachelor of Business Administration	May 2020

Thesis title

Development of an ERP system with the Symfony framework — 31 pages

Commissioned by

Advison Solutions Oy

Supervisor

Janne Turunen

Abstract

The thesis was about the development of an ERP system (enterprise resource planning) using Symfony, a PHP framework. The first chapters of the thesis outlined the methods and techniques used in the development process, after which it went through the developed features of the application. These features were examined through their user interface, code, as well as Symfony components and jQuery and JavaScript plugins. The final chapters were concerned with assessing the success and results of the project.

The objective of the thesis was to develop a functional application that addressed the needs of the customer. The system was highly customized to meet these requirements. The app was developed for a construction company from southern Finland. The purpose of the application was to ease the company's resource and task distribution, improve communication and the tracking of work tasks and hours by automating and replacing the old practices.

As the result of the project, the app reached a testing and finishing phase. The main features were completed, so that all was left was finetuning work, such as the UI improvements and other small features, including fixing any bugs found through testing. The methods and techniques used in the development of the system were very well suited for it. The tools provided by Symfony framework's component library made development easy, because of automated basic features, like security.

Keywords

PHP, programming, software development, enterprise resource planning

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on räätälöidyn toiminnanohjausjärjestelmän tuottaminen rakennusalan yritykselle. Toiminnanohjausjärjestelmä, eli ERP-järjestelmä (Enterprise Resource Planning), on liiketoimintaprosessien hallintaohjelmisto, jonka avulla voidaan hallita yrityksen taloushallintoa, toimitusketjua, toimintoja, raportointia, valmistusta ja henkilöstöhallinnon toimintoja ja integroida ne (Microsoft Dynamics s.a.). Päädyin aiheeseen, kun sain tilaisuuden liittyä mukaan järjestelmän kehitysprojektiin vuoden 2019 lopussa.

Opinnäytetyön tavoitteena on tuottaa toimeksiantajan asiakkaalle toimiva ja tarpeita vastaava sovellus. Työn toimeksiantaja on lahtelainen IT-konsultointiyritys Advison Solutions Oy ja ohjelmiston tilaaja on Etelä-Suomessa rakennusalalla toimiva yritys.

Tuotettavan järjestelmän tarkoitus on pyrkiä helpottamaan yrityksen töiden ja resurssien jakoa, parantamaan viestintää sekä työtuntien ja tehtävien seurantaa, automatisoimalla ja korvaamalla vanhat, paperilla ja laskentataulukko-ohjelmistoilla toimivat, toimintatavat. Sovellus toteutetaan Symfony-ohjelmistokehyksellä, jonka käyttöliittymässä hyödynnetään jQuery-ohjelmistokehystä sekä Bootstrap-CSS-ohjelmistokehyksen komponentteja.

Opinnäytetyöraportissa käydään läpi toiminnanohjausjärjestelmässä käytettyjä ohjelmistoja ja tekniikoita sekä toteutusta vaiheittain. Tuotettuja ominaisuuksia tarkastellaan käyttöliittymään, koodiin sekä Symfony-komponentteihin ja jQuery- ja JavaScript-liitännäisiin tutustuen. Ominaisuuksia illustroidaan myös kuvin. Raportin lopussa arvioidaan työn tuloksia ja tavoitteiden toteutumista.

Lukijalla olisi hyvä olla vähintään perustason tietämystä ohjelmistotuotannosta. Termien, kuten olio-ohjelmointi, tietokanta ja relaatiomalli, tuntemuksesta on myös apua tekstin ymmärtämiseen.

2 TOTEUTUKSESSA KÄYTETYT TEKNIIKAT

Tässä luvussa esitellään projektissa käytettyjä ohjelmistoja ja tekniikoita. Kerronnassa käydään läpi niiden historiaa, rakennetta sekä perustoimintoja. Ohjelmistoja esitellään kappaleessa vain päällisin puolin.

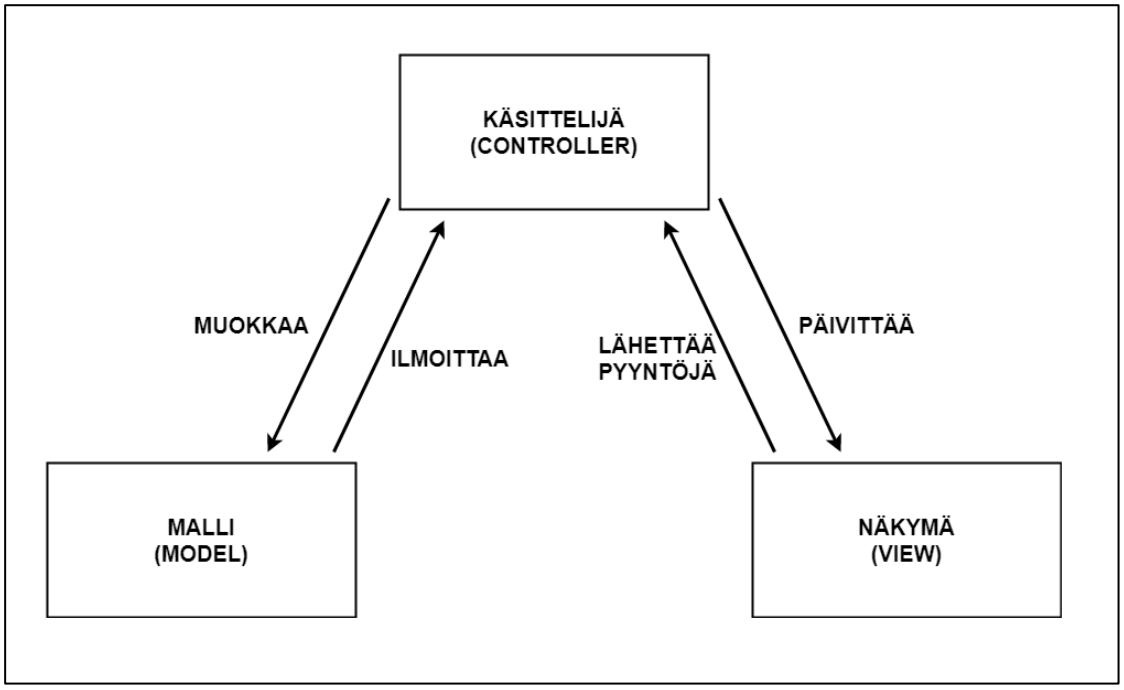
2.1 Symfony

Symfony on PHP-ohjelmointikieleen perustuva framework eli ohjelmistokehys. Ohjelmistokehykset ovat valmiiksi toteutettuja ohjelman osia, jotka toimivat sovelluksen ns. runkona. Sen tarkoituksena on nopeuttaa sovelluskehitystä pakkaamalla monimutkaisia toimenpiteitä yksinkertaisiin kutsuihin. Lisäksi kehys luo sovellukseen valmiin rakenteen auttaen kirjoittamaan parempaa ja luettavampaa koodia, jota on helpompaa hallita. (Potencier & Zaninotto 2007a.)

Symfonyn ensimmäinen versio julkaistiin vuonna 2005. Sen alkuperäinen luoja on Fabien Potencier, joka vuonna 2003 etsi tarpeisiinsa vastaavia web-kehitystyökaluja. Kun sopivaa ei löytynyt hän aloitti oman ohjelmistokehyksensä kehittämisen PHP 5:llä. Symfony on julkaistu avoimena lähdekoodina ja sen kehitystä johtaa Potencierin perustama yritys SensioLabs, joka sijaitsee Ranskassa. (Potencier & Zaninotto 2007a.)

Symfony perustuu MVC-ohjelmistoarkkitehtuurityyliin (Potencier & Zaninotto 2007b). MVC koostuu sanoista model-view-controller, suomennettuna malli-näkymä-käsittelijä. MVC-arkkitehtuurissa sovellus jaetaan näihin kolmeen komponenttiin (Google Chrome Developer s.a.).

Malli edustaa sovelluksen dataa, eli tietoja. Se vastaa pyyntöihin käsittelijästä hakea, tallentaa tai poistaa dataa. Näkymässä yhdistetään mallista tuleva data visuaaliseen muotoon käyttäjän nähtäville, muodostaen käyttöliittymän. Käsittelijässä validoidaan ja vastataan käyttäjän käyttöliittymässä tekemiin pyyntöihin ja tehdään muutoksia dataan. Kuvassa 1 havainnollistetaan MVC-tyylin toimintaa. (Google Chrome Developer s.a.)

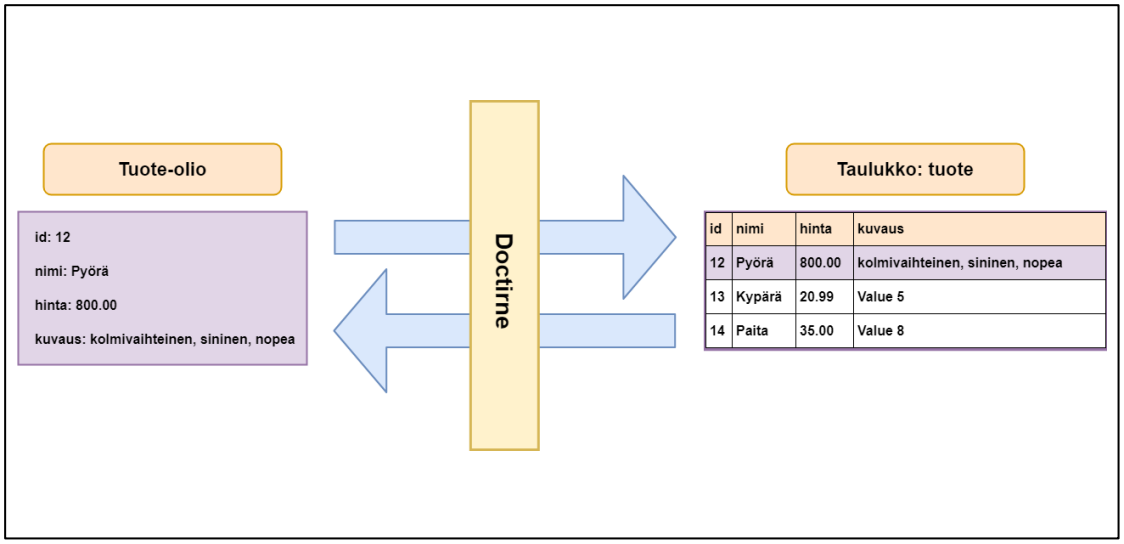


Kuva 1. MVC-arkkitehtuurityylin toiminta (Google Chrome s.a.)

Laajoissa projekteissa nousee usein tarve toistaa samaa koodia ja toimintoja, kuten esimerkiksi tietokannan hallinta, lomakkeiden käsittely ja sähköpostin lähettäminen. Tästä syystä Symfony on kehitetty nopeuttamaan kehitysprosessia tekemällä usein käytetyistä toiminnoista uudelleen kierrätettäviä itsenäisiä osia, eli PHP-kirjastoja, joita kutsutaan komponenteiksi. Näin kehittäjä voi keskittyä oman sovelluksensa, ei työkalujen, kehittämiseen. (SensioLabs 2015.)

Symfony-ohjelmistokehys itsessään on kokoelma näitä edellä mainittuja komponentteja, jossa se toimii eräänlaisena liimana komponenttien välillä. Komponentit voivat olla Symfony-komponentteja, jotka toimivat osana Symfonyn perustaa, tai kolmansien osapuolien kehittämiä. Komponenttien asentamiseen käytetään Symfonyssa Composer-paketinhallintatyökalua. (SensioLabs 2015.)

Keskeisimpiä Symfony-komponentteja ovat mm. HttpFoundation, joka pitää sisällään työkalut http-pyyntöjen ja -vastausten hallintaan, Translation sivujen kääntämiseen sekä Security, jolla hallitaan käyttäjien rooleja ja sivun käyttöoikeuksia (SensioLabs 2015). Olennaisiin komponentteihin luetaan myös Symfonyn ORM-packista löytyvä Doctrine. Sitä hyödynnetään sovelluksen ja tietokantojen välisessä vuorovaikutuksessa, jota havainnollistetaan kuvassa 2.



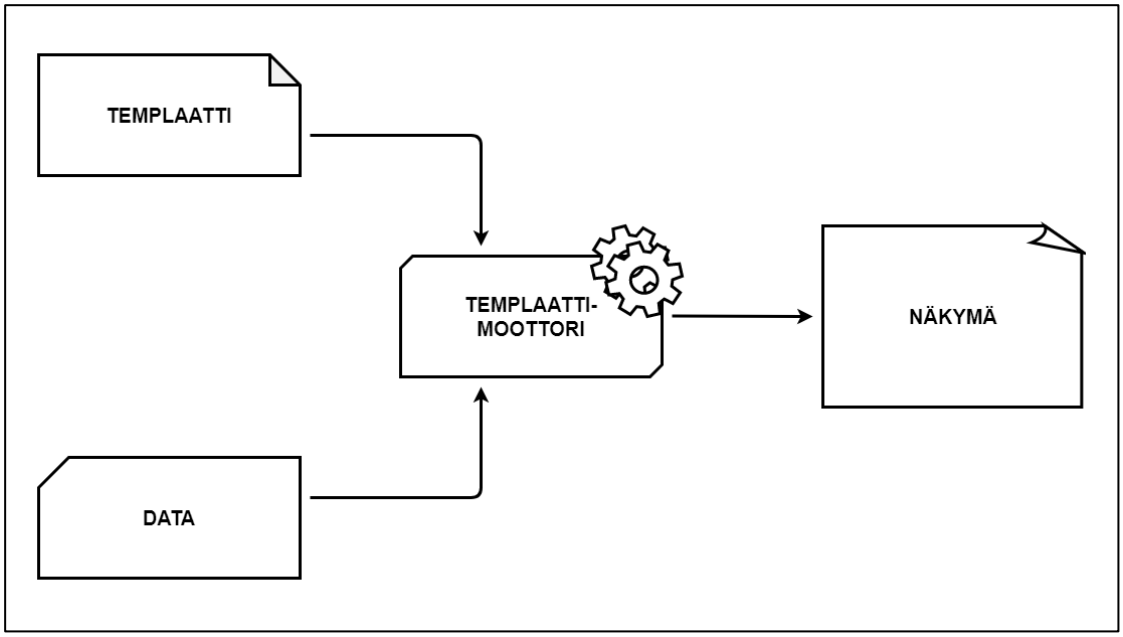
Kuva 2. Doctrinen ORM:n toiminta (SensioLabs Symfony 2020b)

Tietokannat pohjautuvat relaatiomalliin, PHP:n ja Symfonyn ollessa oliopohjaisia. Jotta sovelluksella päästään käsittelemään tietokannan tietoja, tarvitaan sovitin kääntämään oliopohjainen logiikka relaatiologiikaksi. Tästä käytetään nimitystä object-relational mapping (ORM), eli olio-relaatiomappaus. (Potencier & Zaninotto 2007a.)

2.2 Twig

Twig on template engine, eli ”templaattimoottori” PHP-ohjelmointikielelle. Sen on alun perin kehittänyt Armin Ronacher, jonka hän perusti Django-ohjelmistokehyksen templaattimoottoreihin. Hänen työnsä pohjalta projektia jatkoi eteenpäin Symfonyn luoja Fabien Potencier. Twigin kehitystä on vuodesta 2009 johtanut Potencierin SensioLabs-yritys. Tästä syystä Twig on oletusarvoinen templaattimoottori Symfony-ohjelmistokehyksessä. (Potencier 2009.)

Templaattimoottorin tehtävä on tiedon sijoittaminen ja muotoileminen WWW-sivuille (Potencier 2009). Moottorin toimintaa havainnollistetaan kuvassa 3. Siinä html-tyylinen templaatti yhdistetään mallista tulevaan dataan lopulta muodostaen käyttäjälle sivun näkymän ja käyttöliittymän.



Kuva 3. Templaattimoottorin toiminta

Twigissä käytetään kolmea eri merkintätapaa, joita esitetään kuvan 4 esimerkissä. {{ … }}-merkinnällä käytetään merkitsemään templaatissa kohtaa, johon tulostetaan dataa edustavia muuttujia. Sitä käytetään myös toisten templaattiblokkien sisällyttämiseen. {% … %}-merkintää käytetään ajamaan templaattimoottorissa erilaisia ohjausrakenteita, kuten for-silmukoita, if-lauseita tai muuttujien määrittelyä. {# ... #}-merkinnällä voidaan kommentoida sisältöä. (SensioLabs Twig s.a.)

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>My Webpage</title>
5      </head>
6      <body>
7          <ul id="navigation">
8          {% for item in navigation %}
9              <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
10         {% endfor %}
11         </ul>
12
13         <h1>My Webpage</h1>
14         {{ a_variable }}
15     </body>
16 </html>
```

Kuva 4. Twig-templaattimoottorin merkintätavat (SensioLabs Twig s.a.)

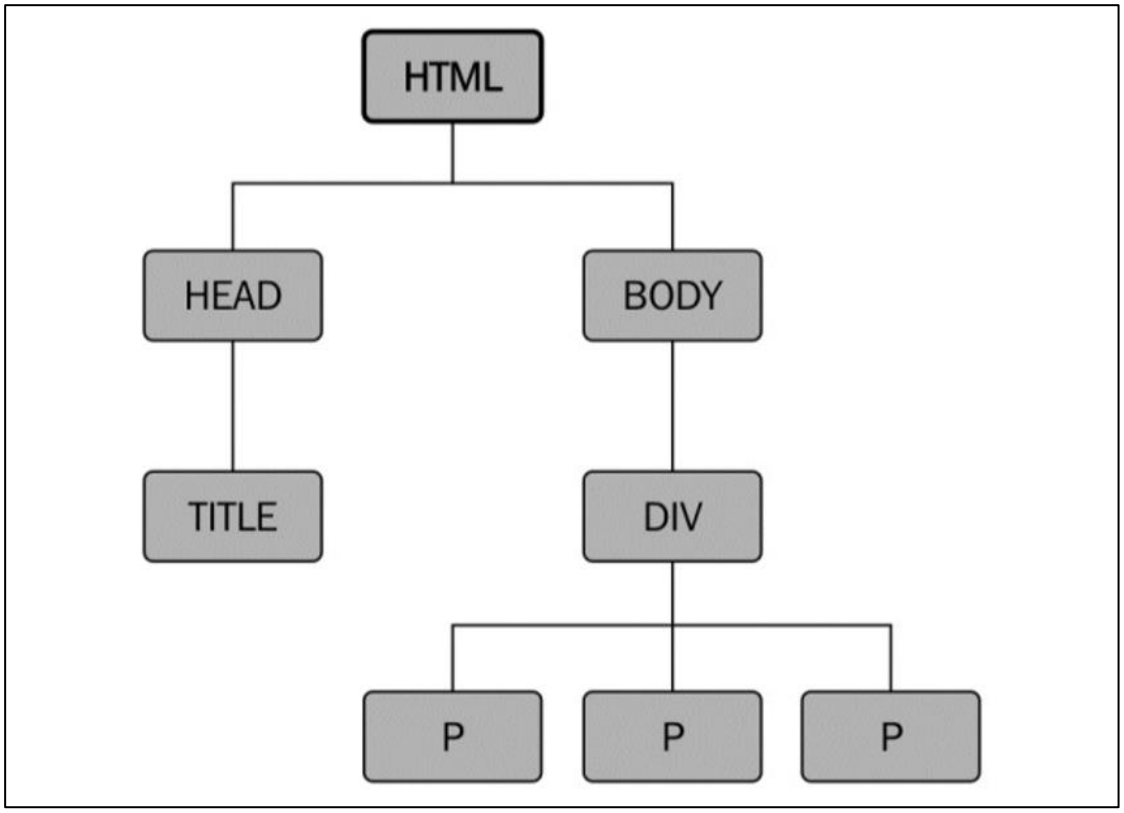
Twig on tehokas ja monipuolinen templaattimoottori. Se mahdollistaa omien mukautettujen laajennusten kehittämisen perusominaisuuksien jatkoksi. Twigin kattava dokumentaatio takaa sen, että sen käyttö on helppo opetella. (SensioLabs Twig s.a.)

2.3 JavaScript ja jQuery

JavaScript on pääasiassa front-endissä, eli käyttöliittymässä ajettava ohjelmointikieli. Sen kehityksen aloitti Brendan Eich tarpeesta luoda verkkosivuja, joiden käyttökokemus olisi laajempi sekä tarpeesta vähentää kutsujen lähettimistä back-endiin, eli palvelimeen. Sen ensimmäinen versio julkaistiin vuonna 1995 Netscape-selaimessa nimellä LiveScript. Lopullisen nimensä JavaScript sai Netscape selaimen 2.0-versiossa. (Stefanov & Sharma 2013.)

JavaScriptin suosion myötä muut selainkehittäjät aloittivat omaan web-selaimeensa sopivan version kehittämisen JavaScriptistä. Yksi esimerkki näistä oli Microsoftin JScript Internet Explorer-selaimelle. Tämä johti pyrkimykseen kehittäjien ja valmistajien toimesta luoda yhteinen standardi kielen käyttöön. ECMA (European Computer Manufacturers Association) loi standardin ECMA-262, jossa määritellään JavaScript-ohjelmointikielen pääosat ilman selain- tai verkkosivu-kohtaisia toimintoja. Standardoidusta JavaScript-ohjelmointikielestä käytetään nimitystä ECMAScript. (Stefanov & Sharma 2013.)

Nykyaikainen JavaScript voidaan jakaa kolmeen osaan: edellä mainittuun ECMAScriptiin, Document Object Modeliin (DOM) ja Browser Object Modeliin (BOM). DOM antaa JavaScriptille tapoja lukea ja käsitellä HTML- ja XML-tiedostoja luomalla niistä kuvan 5 mukaisia puukaavioita. BOM sisältää selainkohtaiset tiedot JavaScriptin soveltamiseen. Tänä päivänä JavaScriptiä hyödynnetään monipuolisesti mm. mobiilisovelluksissa ja verkkosovelluksissa. Käyttöliittymän lisäksi JavaScriptiä voidaan soveltaa myös palvelinpuolella esimerkiksi .NET- tai NodeJS-ympäristöissä. (Stefanov & Sharma 2013.)



Kuva 5. DOM-puu (Chaffer & Swedberg 2013)

JavaScriptiin perustuvan ohjelmointikehyksen, jQueryn, kehityksen aloitti ohjelmoija John Resig vuonna 2005. Hän kehitti sen helpottaakseen sivun elementtien löytämistä ja niihin vaikuttamista ohjelmallisesti. Vuonna 2006 julkistetun jQueryn ominaisuuksiin kuuluivat DOM-muokkaus sekä animaatiot, mutta vuosien saatossa sen toiminnot ovat laajentuneet ja tehokkuus parantunut. Tänä päivänä jQuery on julkaistu avoimena lähdekoodina ja sen kehitystä johtaa osaava joukko JavaScript-kehittäjiä. (Chaffer & Swedberg 2013.)

Kuvassa 6 havainnollistetaan JavaScriptin ja jQueryn syntaksia ja eroja. Kuvan esimerkissä on toteutettu sama toiminto, jossa näytetään ja piilotetaan elementti klikkaamalla toista elementtiä. jQuery yksinkertaistaa ja lyhentää tarvittavaa ohjelmointityötä huomattavasti.

```
// JavaScript
const login = document.getElementById("login");
const loginMenu = document.getElementById("loginMenu");
login.addEventListener("click", () => {
    if (loginMenu.style.display === "none"){
        loginMenu.style.display = "inline";
    } else {
        loginMenu.style.display = "none";
    }
});


// jQuery
$("#login").click(() => {
    $("#loginMenu").toggle()
});
```

Kuva 6. Esimerkki JavaScriptin ja jQueryn eroista (BitDegree 2020)

jQuery-ohjelmointikehyksen vahvuuksia ovat sen laaja ominaisuuksien kirjo, helposti opittava syntaksi ja soveltamismahdollisuudet eri alustoille. Lisäksi sillä on tuotettu satoja plugineja, eli liitännäisiä. Liitännäiset ovat valmiita toimintokirjastoja, jotka laajentavat ohjelmistokehyksen ominaisuuksia edelleen. (Chaffer & Swedberg 2013.)

3 TOTEUTUS

Luvussa käydään läpi projektin toteutuksen kulkua ja esitellään suurin osa sovelluksen ominaisuuksista. Tullessani mukaan projektiin, sovelluksen kehitys oli jo aloitettu. Ensimmäiseksi kävimme läpi työn toimeksiantajan kanssa sovelluksen valmiita toimintoja ja siinä käytettyjä käsitteitä sekä kehitettäviä ominaisuuksia ja muita vaatimuksia. Pyrimme järjestämään vastaavanlaisia kehityksen ohjauskeskusteluja viikoittain.

Sovellus on kehitetty Symfony-ohjelmistokehyksen versiolla 4. Sen front-end-puolella on hyödynnetty mm. jQuery- ja Bootstrap-kehyksiä ja niihin perustuvia liitännäisiä. Liitännäiset lisättiin projektiin käyttäen Yarn-paketinhallintajärjestelmää ja otettiin käyttöön sovelluksen app.js-tiedostossa. Sovelluksen kehitysympäristönä toimi Docker. Sovelluksen käyttö tulee valmistuessaan tapahtumaan kentällä, eli työmailla ja -kohteissa, joten kohdekäyttölaitteet ovat pääasiassa tablet-tietokoneet.

Varsinaisen sovelluksen työstämisen aloitin asentamalla kehitysympäristön vaatimat ohjelmistot ja tutustumalla sovelluksen koodiin, rakenteeseen ja toimintoihin. Peruskäytäntönä sovelluksen koodissa ja kommenteissa käytettiin englannin kieltä.

3.1 Tietojen syöttäminen

Sovelluksen käyttö vaatii käyttäjätunnuksen, joten sovelluksessa tulee olla ominaisuus käyttäjien luontiin. Koska sovellus on suunniteltu yrityksen ja sen työntekijöiden käyttöön, tämä ominaisuus on annettava pääkäyttäjän, eli adminin, kyvyksi eikä esimerkiksi käyttäjien itsenäiseksi rekisteröitymiseksi.

Sovelluksen yksi päätarkoituksista on helpottaa tehtyjen töiden ja kirjonpidon seurantaa, joten siinä myös oltava mahdollisuus syöttää tietoja. Sovellukseen toteutettiin ominaisuudet asiakkaiden, työmaiden, töiden, resurssien, tuotteiden, hintojen ja suoritettujen tehtävien lisäämiseksi. Niihin luotiin myös muokkaus- ja poistamismahdollisuudet.

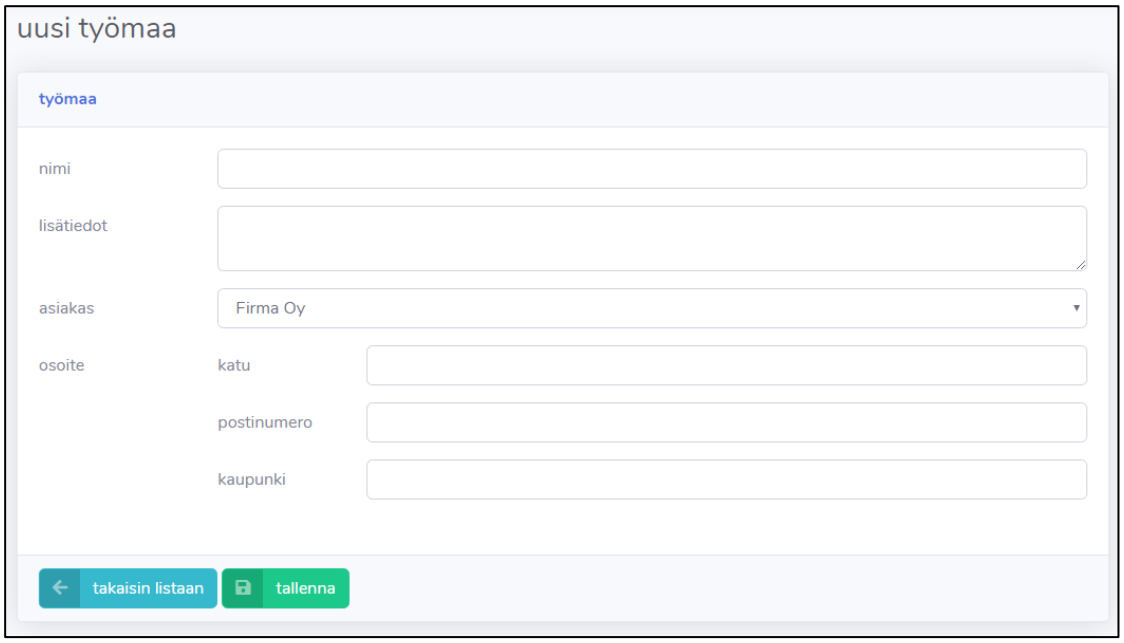
Jokaiselle ominaisuudelle luotiin oma sivu ja käsittelijä. Sivulle pääsy sijoitettiin sovelluksen sivupalkista löytyvään valikkoon. Seuraavaksi ominaisuudesta on tehtävä tietokannan entiteetti komentorivillä Doctrine-komennolla *php bin/console make:migration.* Komento avaa Doctrinen ohjatun entiteetin luonnin, jossa luokka nimetään ja siihen lisätään halutut ominaisuudet. Lopuksi entiteeteistä luodaan migraatiotiedosto ja muutokset ajetaan tietokantaan komennolla *php bin/console doctrine:migrations:migrate***.**

Symfony mahdollistaa lomakkeen luonnin entiteetin pohjalta hyödyntäen form builderia, eli lomakkeen rakentajaa (kuva 7). Rakentajassa kentän tyyppi määritetään valmiilla luokilla, kuten esimerkiksi merkkijonot TextType-luokkana, päivämäärät DateType-luokkana. Siinä voidaan lisätä kenttään myös attribuutteja, kuten tunnisteita, luokkia, otsikoita ja valintavaihtoehtoja.

```php
class SiteType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', TextType::class, [
              'label' => 'form.label.name'
            ])
            ->add('text', TextareaType::Class, [
              'label' => 'form.label.siteinfo',
            ])
            ->add('customer', EntityType::class, [
              'class' => Customer::class,
              'label' => 'form.label.customer',
            ])
            ->add('address', AddressType::Class, [
              'label' => 'form.label.address',
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Site::class,
        ]);
    }
}
```

Kuva 7. Työmaa-lomakkeen rakentaja

Lomakkeen rakentajan luoma lomake kuvassa 8. Siinä on tekstikenttä työmaan nimelle, tekstialuekenttä lisätiedoille, valintakenttä asiakkaista sekä tekstikentät osoitteen kadulle, postinumerolle ja kaupungille. Lomakkeen lähetys sidotaan lomakkeen ulkopuolella sijaitsevaan nappiin jQuerylla.



Kuva 8. Työmaa-lomake

Kuvassa 9 lomakkeen lähetyksen käsittely ja uuden työmaan tallennus käsittelijässä. Jos lomake on lähetetty ja todettu validiksi, lomakkeen tiedot tallennetaan ja näkymä siirretään työmaiden listaukseen. Muussa tapauksessa näkymään tulostetaan tyhjä lomake.

```php
/**
 * @Route("/new", name="site_new", methods={"GET","POST"})
 */
public function new(Request $request): Response
{
    $this->denyAccessUnlessGranted('ROLE_ADMIN', 'ROLE_SUPER_ADMIN');

    $site = new Site();
    $form = $this->createForm(SiteType::class, $site);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($site);
        $entityManager->flush();

        return $this->redirectToRoute('site_index');
    }

    return $this->render('site/new.html.twig', [
        'site' => $site,
        'form' => $form->createView(),
    ]);
}
```

Kuva 9. Uuden työmaan tallennus käsittelijässä

Kaikilla tietojen lisäämisen ominaisuudet ovat kaikki erilaisia, mutta ne kaikki luotiin edellä kuvatulla tavalla. Käyttäjän ja resurssin luonnissa, tarvittiin tiedoston liittämistä kyseisen ominaisuuden yhteyteen. Nämä tiedostot voivat resurssissa olla esimerkiksi yrityksen auton rekisteriote ja käyttäjässä työntekijän työ- tai paloturvallisuuskortti tai jokin muu tosite. Liitettävien tiedostojen oletetaan olevan pdf-, png- tai jpg-tiedostoja. Lomakkeen rakentajassa luodaan kenttä tiedoston liittämiseen käyttäen DocumentFileType-luokkaa.

Sovelluksen toiminnan kannalta tärkeä ominaisuus on tehtävien lisäys. Siinä työntekijä kirjaa päivän aikana suorittamansa työtehtävät syöttämällä päiväyksen, valitsemalla sovellukseen lisätyistä töistä ja tuotteista yhden sekä tehdyn työn määrän ja lisätiedot (kuva 10). Päiväyksen kentässä käytetään Bootstrap Datepicker-jQuery-liitännäistä, joka mahdollistaa päivän valinnan kalenterista.

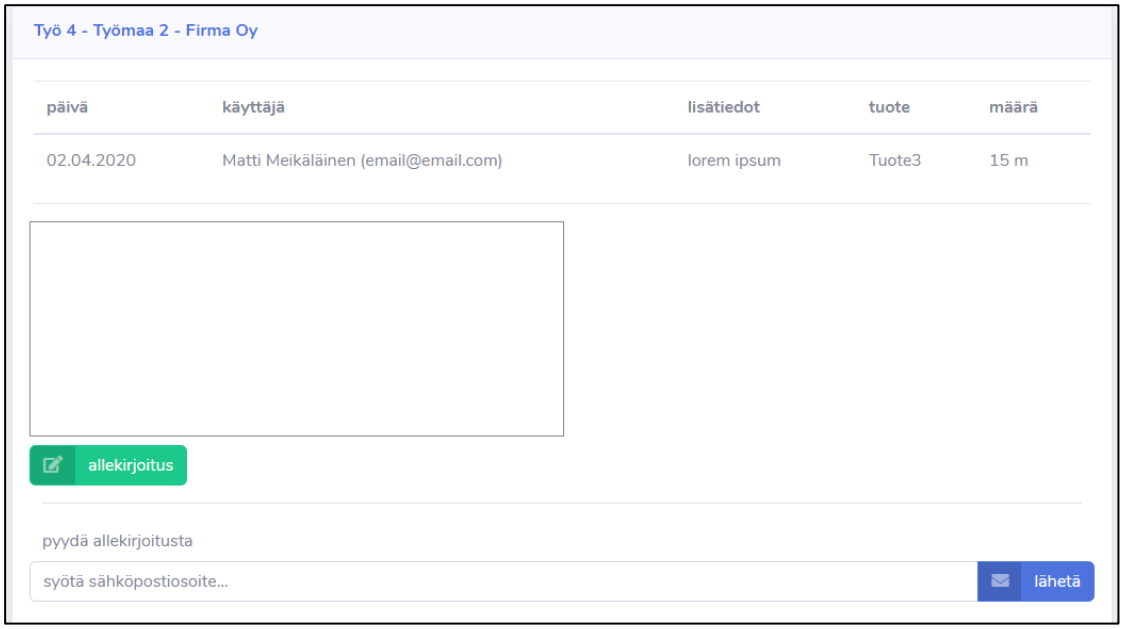
Kuva 10. Tehtävä-lomake

Tehtävästä tallennetaan lomakkeeseen syötettyjen tietojen lisäksi tiedot syöttänyt käyttäjä ja tehtävän luontipäiväys. Käyttäjätietoa tarvitaan mm. tehtävien koonnissa mittalappuihin. Tehtävän luontipäivää tarvitaan tehtävän esittämisen apuna käyttöliittymässä.

3.2 Tehtävien kuittaukset

Kun työntekijä on kirjannut ylös sovellukseen tekemänsä tehtävät, pääkäyttäjä kokoaa niistä kuitattavan kokoelman. Kokoelmalla on oma entiteettinsä, sivunsa ja käsittelijänsä. Entiteetin sisältönä on relaatio työ-entiteettiin, allekirjoitus-kuvan nimi sekä kokoelmalle uniikki hash-merkkijono.

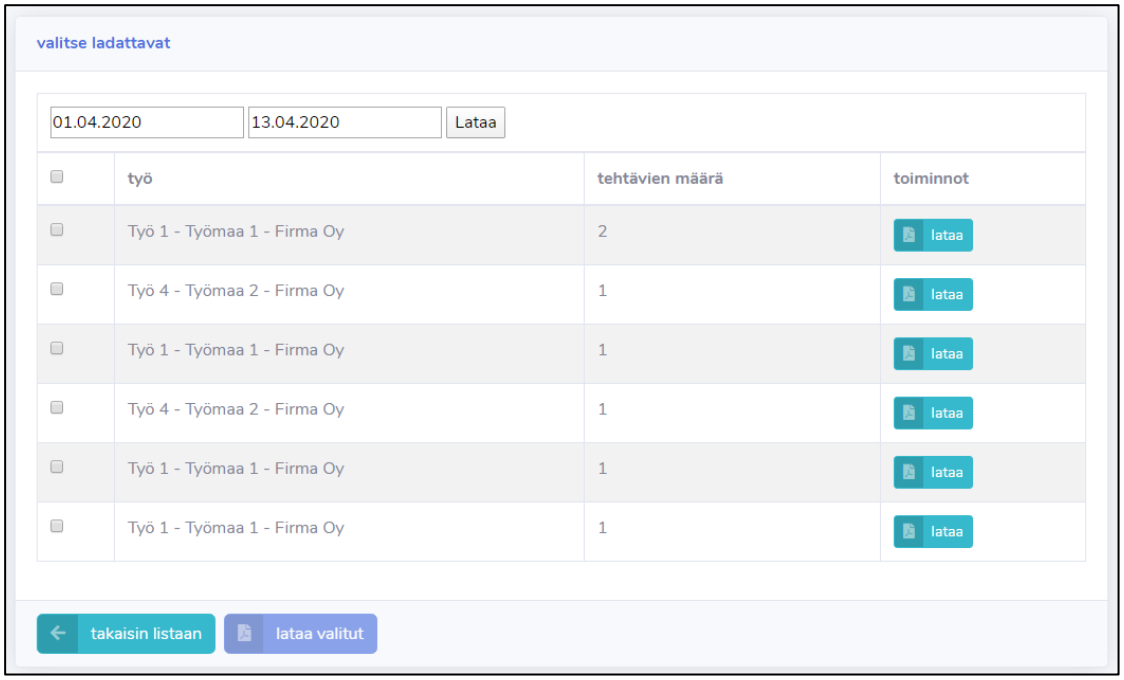
Sen kuittauksessa käytetään Signature Pad-JavaScript-liitännäistä (kuva 11), jossa hyödynnetään HTML:n canvas-elementtiä. Se mahdollistaa allekirjoituksen kirjoittamisen käyttäen hiiren osoitinta tai kosketusnäyttöä, joka voidaan tallentaa kuvana. Allekirjoituksen tallentamisessa käytetään Ajax-kutsua.



Kuva 11. Kokoelman kuittaussivu

Tehtäväkokoelmasivulla on myös ominaisuus, jolla voidaan pyytää allekirjoitusta sähköpostitse esimerkiksi yrityksen ulkopuoliselta henkilöltä. Sähköpostin lähetyksessä hyödynnetään Symfonyn SwiftMailer-komponenttia. Viestin sisältö määritetään Twig-templaatista, jossa on linkki allekirjoitussivulle. Linkin luonnissa käytetään entiteetin hash-merkkijonoa, jolle on luotava käsittelijään oma funktionsa ja templaatti, josta karsitaan linkit ja muut ulkopuoliselle kuulumattomat ominaisuudet. Lopuksi sivun polulle on luotava poikkeus Symfonyn Security-komponentin asetuksiin, jotta sivun lataus ei automaattisesti vaadi sisäänkirjautumista.

Allekirjoitetun kokoelman voi ladata pdf-tiedostona, jolloin siihen lasketaan summa, sen työmaata ja tuotetta vastaavaa hintaa käyttäen. Ladatusta pdf-tiedostosta käytetään nimitystä mittalappu. Mittalappuja voidaan ladata useampia kerralla, jolloin ne yhdistetään yhteen, monisivuiseen pdf-tiedostoon. Niiden valintaa voidaan rajoittaa sen sisältämien tehtävien tekopäiväyksen mukaan, jota esitellään kuvassa 12.



Kuva 12. Ladattavien mittalappujen valinta

Pdf-tiedoston luonnissa käytetään Dompdf-komponenttia, joka on HTML-PDF-muunnin. Se mahdollistaa pdf:n luonnin suoraan templaatista. Ladattuja mittalappuja hyödynnetään pääasiassa viestinnässä yrityksen asiakkaiden kanssa.

3.3 Tuntilista

Työntekijöiden tekemien työtuntien seuraamiseksi luotiin sovellukseen tuntilista-sivu. Ensiksi sivulla rajataan päivävalinta, jolta tunnit halutaan hakea. Kun aloitus- ja lopetuspäivä on valittu, sovellus näyttää työntekijöiden tunnit taulukossa (kuva 13). Päivävalinnoissa hyödynnettiin Datepicker-liitännäistä.

23.03.2020 03.04.2020 Lataa

PVM	Tuomas Kiiskinen (sahkoposti@email.fi)	Matti Meikäläinen (email@email.com)	Testi Testaaja (testi@testi.com)
Mon 23.03.2020	0	0	0
Tue 24.03.2020	0	0	0
Wed 25.03.2020	0	0	0
Thu 26.03.2020	8	0	0
Fri 27.03.2020	8	0	0
Sat 28.03.2020	0	0	0
Sun 29.03.2020	8	0	0
Mon 30.03.2020	8	8	0
Tue 31.03.2020	8	8	0
Wed 01.04.2020	0	8	0
Thu 02.04.2020	8	(8)	0
Fri 03.04.2020	8	8	0
Yhteensä	56	32 (40)	0

Kuva 13. Tuntilista

Koska sovelluksessa seurataan työntekijän tekemien tuntien sijaan työn määriä, käytetään päivän työtunteina oletuksena kahdeksaa tuntia, jos työntekijä suorittaa työtehtäviä päivän aikana. Tuntilistassa kuitattujen ja kuittaamattomien tehtävien tunnit on eroteltu esitystavassa. Kuittaamattomat tunnit esitetään käyttöliittymässä sulkumerkkien sisällä. Viimeisellä rivillä on tuntien yhteenlasku, jossa sulkujen sisällä myös kuittaamattomat tunnit laskettu mukaan.

3.4 Hintojen versiointi

Versioinnilla tarkoitetaan tietokantaan tehtyjen muutosten seurantaa ja loggingia, eli kirjaamista. Sovelluksessa versiointia käytetään vain hintojen, kuten tuotteiden ja työn laskutushinnan seuraamiseen. Versiointi toteutettiin DoctrineExtensions-komponentin Loggable-ominaisuudella. Ominaisuus luo automaattisesti tietokannan entiteetin, johon tallennetaan muutetun objektin lisäksi muutoksen päiväys, tehty muutos, versionumero sekä muutoksen tekijä.

Versiointi otetaan käyttöön seurattavassa entiteetissä ottamalla entiteetissä käyttöön Gedmo-annotaatiot ja lisäämällä entiteetin annotaatioon Loggable-ominaisuuden. Entiteetin seurattavat ominaisuudet merkataan myös versioitavaksi omissa annotaatioissaan. Käyttöönoton merkinnät nähtävissä kuvassa 14, jossa versioitava ominaisuus on value-muuttuja.

```php
/**
 * @ORM\Entity(repositoryClass="App\Repository\PriceRepository")
 * @UniqueEntity(
 *     fields={"site", "product"},
 *     errorPath="price",
 *     message="This port is already in use on that host."
 * )
 * @Gedmo\Loggable
 */
class Price
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="float")
     * @Gedmo\Versioned
     */
    private $value;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Site", inversedBy="prices")
```

Kuva 14. Versioinnin käyttöönotto Price-entiteetissä

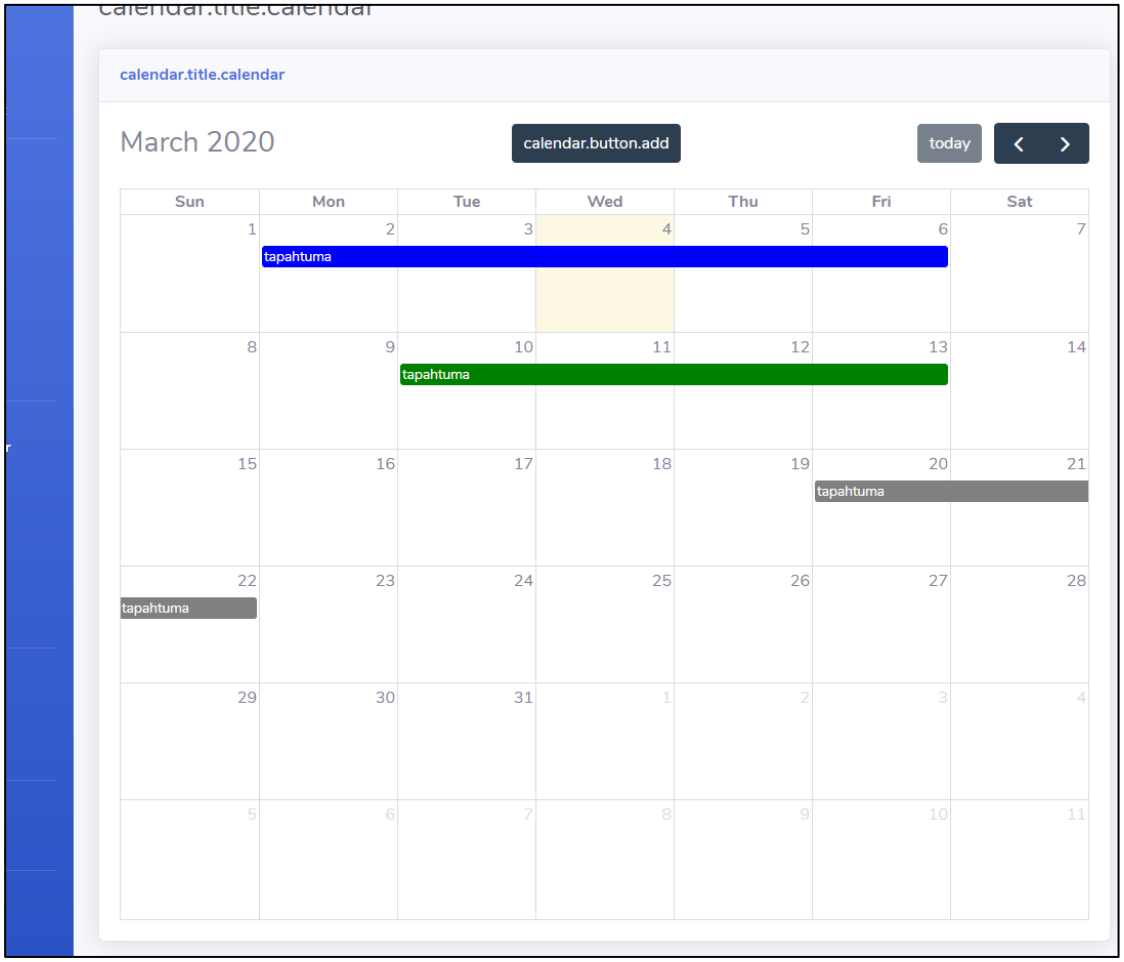
Version tarkastusta varten luotiin oma servicensä, jotta samaa koodia voitaisiin käyttää uudelleen helpommin. Serviceen luodussa funktiossa otetaan vastaan parametreinä tarkastettava entiteetti ja päiväys. Aluksi entiteetin perusteella haetaan sen muutosten kirjaukset. Sitten kirjattujen muutosten päiväyksiä verrataan parametrina saatuun päiväykseen ja palautetaan voimassa oleva versio hinnasta.

3.5 Kalenteri

Sovellukseen kehitettiin seuraavaksi kalenteri, johon voitaisiin lisätä ja näyttää tapahtumia käyttäjille. Lisävaatimuksina toteutukseen tuli tapahtuman ulkoasun värin valinta ja henkilöliitos. Kalenterin toteutuksessa käytettiin FullCalendar-, Datepicker- ja Select2-JavaScript-liitännäisiä. Kalenterin ulkoasu ja toiminnot pyrittiin pitämään selkeinä ja helppokäyttöisinä pääasiallisia käyttölaitteita, eli tablet-tietokoneita ajatellen.

Kalenterin teko aloitettiin tekemällä sivun templaatti ja lisäämällä käsittelijään osoite sivun kutsumiseksi ja vastaus kyseiseen kutsuun. Aluksi templaattiin

tehtiin muiden sivujen kanssa yhtenevä ulkoasu ja FullCalendar-kalenteri perusasetuksilla (kuva 15). Seuraavaksi luotiin kalenterin tietokannan entiteetin.



Kuva 15. Kalenterin käyttöliittymä

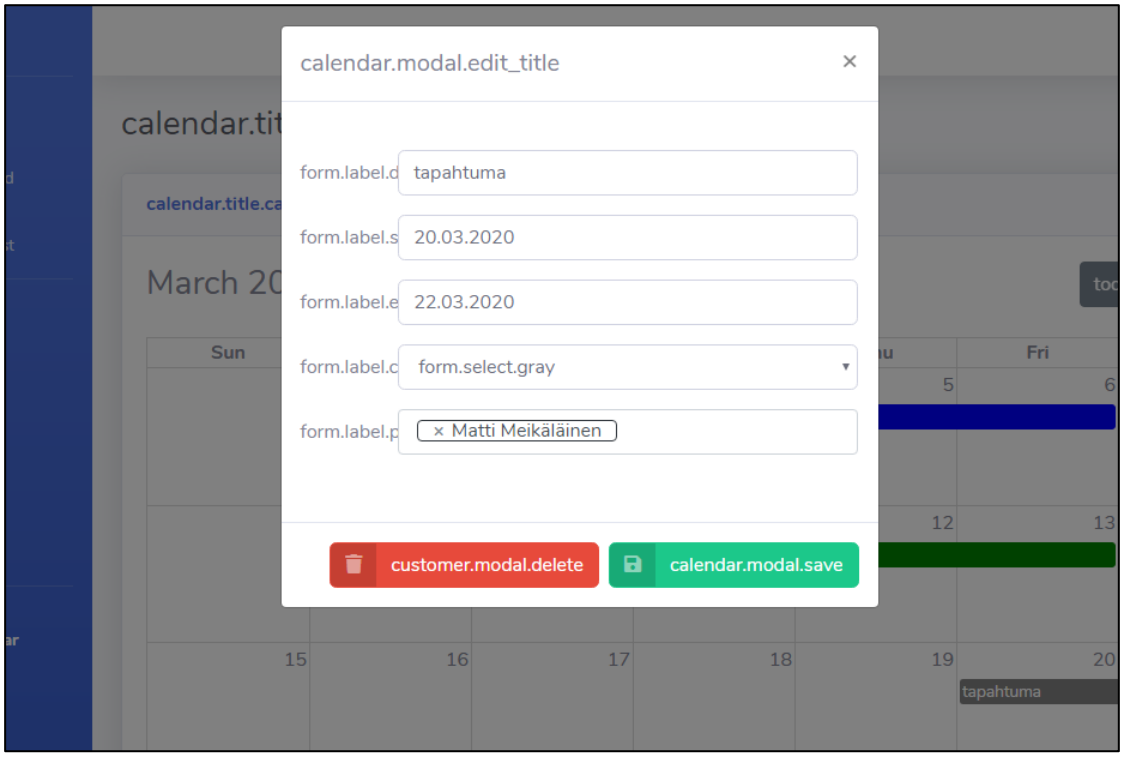
Ominaisuuksista määritellään mm. nimi, tyyppi sekä mahdollisuus jäädä tyhjäksi. Kalenterin tapahtuman entiteetti vaatii sisällöksi nimityksen, aloitus- ja lopetuspäiväyksen, värivalinnan, joka tässä tapauksessa tallennetaan merkkijonona sekä liitetyt käyttäjät. Koska tapahtumaan voidaan liittää osallistujaksi useampi käyttäjä, on luotava assosiaatiotaulu, jossa liitetyn käyttäjän tunniste yhdistetään tapahtuman tunnisteeseen. Tämä onnistui antamalla ominaisuuden tyypiksi relaation ja itse relaation tyypiksi ManyToMany.

FullCalendar-liitännäisessä ei ole valmista logiikkaa tapahtumien lisäämiseen, muokkaukseen ja poistoon, mutta tarjoaa valmiin tapahtuma-, eli event-kokoelman. Tapahtumia ovat esimerkiksi elementin klikkaus tai elementin päällä leijutus kursorilla. Varsinainen lomake luotiin entiteetin pohjalta hyödyntäen Symfonyn lomakkeen rakentajaa (kuva 16).

```
])
->add('startdate', DateType::class, [
  'label' => 'form.label.startdate',
  'widget' => 'single_text',
  'format' => 'dd.MM.yyyy',
  'html5' => false,
  'attr' => [
    'class' => 'js-datepicker modal_startdate',
    'autocomplete' => "off"
  ]
])
->add('enddate', DateType::class, [
  'label' => 'form.label.enddate',
  'widget' => 'single_text',
  'format' => 'dd.MM.yyyy',
  'html5' => false,
  'attr' => [
    'class' => 'js-datepicker modal_enddate',
    'autocomplete' => "off"
  ]
])
->add('color', ChoiceType::class, [
  'label' => 'form.label.color',
  'attr' => [
    'class' => 'modal_color'
  ],
  'choices' => [
    'form.select.blue' => "#0000FF",
    'form.select.red' => "#FF0000",
    'form.select.green' => "#008000",
    'form.select.yellow' => "#FFC300",
    'form.select.gray' => "#808080",
    'form.select.black' => "#000000",
  ],
])
```

Kuva 16. Tapahtuman alku- ja loppupäiväyksen sekä värivalinta lomakkeen rakentajassa

Tapahtuman luonti- ja muokkauslomakkeet sijoitettiin Bootstrap modaaliin (kuva 17). Modaali on näkymän keskelle ilmestyvä ikkuna, jonka voi avata ja sulkea ilman sivun uudelleenlatausta. Luontilomake tulee näkyviin kalenterin yläosaan sijoitetusta painikkeesta, kun taas muokkauslomake klikatessa tapahtumaa kalenterissa.



Kuva 17. Modaali, jossa tapahtuman muokkauslomake ja poistopainike

Muokkauslomakkeen populointi sille kuuluvilla tiedoilla onnistui hyödyntäen FullCalendarin eventClick-metodia, jossa on parametrina tapahtumamerkinnän tiedot. Osallistujien lisäämisessä hyödynnettävä Select2-liitännäinen luo lomakkeeseen kentän, joka mahdollistaa usean käyttäjän valinnan pudotusvalikosta. Muokkauslomakkeen modaali pitää sisällään myös napin tapahtuman poistoa varten.

3.6 Käännökset

Termillä lokalisointi tai kansainvälistäminen viitataan ohjelmointikielessä prosessiin, jossa merkkijonoja eristetään sovelluksesta erilliselle tasolle. Siellä ne käännetään ja korvataan käännetyllä merkkijonolla käyttäjän kielivalinnan mukaan (SensioLabs Symfony 2020a). Projektissa tämä prosessi toteutettiin Translation-komponentin avulla.

Komponentti mahdollistaa merkkijonojen merkitsemisen käännettäväksi käyttämällä templaatissa trans-filtteriä (kuva 18). Se tekee merkkijonosta eräänlaisen paikkamerkin, jonka tilalle käännetty teksti ilmestyy. Käännöksiä voidaan käyttää myös lomakkeen rakentajassa. Koska käännettäviä merkkijonoja löytyy sovelluksesta satoja, ne pyrittiin nimeämään tavalla, joka selkeyttää sen

olinpaikkaa sovelluksessa. Käytetty nimi on yleensä kolmiosainen, pisteellä eritelty merkkijono, joista ensimmäinen osa on sen templaatin nimi, toinen osa elementti templaatissa ja kolmas varsinaiseen käännökseen viittaava, esimerkiksi ”base.menu.job_card”.

```
<!-- Nav Item - Job Card -->
<li class="nav-item {{ route starts with 'job_card' ? 'active' }}">
  <a class="nav-link" href="{{ path('job_card') }}">
    <i class="fas fa-th"></i>
    <span>{{ 'base.menu.job_card'|trans }}</span></a>
</li>

<!-- Nav Item - Hour List -->
<li class="nav-item {{ route starts with 'hour_list' ? 'active' }}">
  <a class="nav-link" href="{{ path('hour_list') }}">
    <i class="fas fa-th"></i>
    <span>{{ 'base.menu.hour_list'|trans }}</span></a>
</li>
```

Kuva 18. Käännettävät merkkijonot templaatissa trans-filtterillä merkittynä

Kuvassa 19 käännettävistä merkkijonoista kootaan automaattisesti tiedosto, jossa varsinainen käännös määritetään, komennolla *php bin/console translation:update –force*, jonka loppuun määritetään haluttu kieli lyhennyksenä, esim. *fi* suomeksi. Sovelluksesta tuotetaan alustavasti käännökset suomen- ja englannin kielellä.

```
I have no name!@78b0bc8d9b41:/usr/src/app$  php bin/console translation:update --force en

Translation Messages Extractor and Dumper
=========================================

 // Generating "en" translation files for "default
 // directory"

 // Parsing templates...

 // Loading translation files...

 // Writing files...

 [OK] Translation files were successfully updated.
```

Kuva 19. Käännöstiedoston tuottaminen komentorivillä

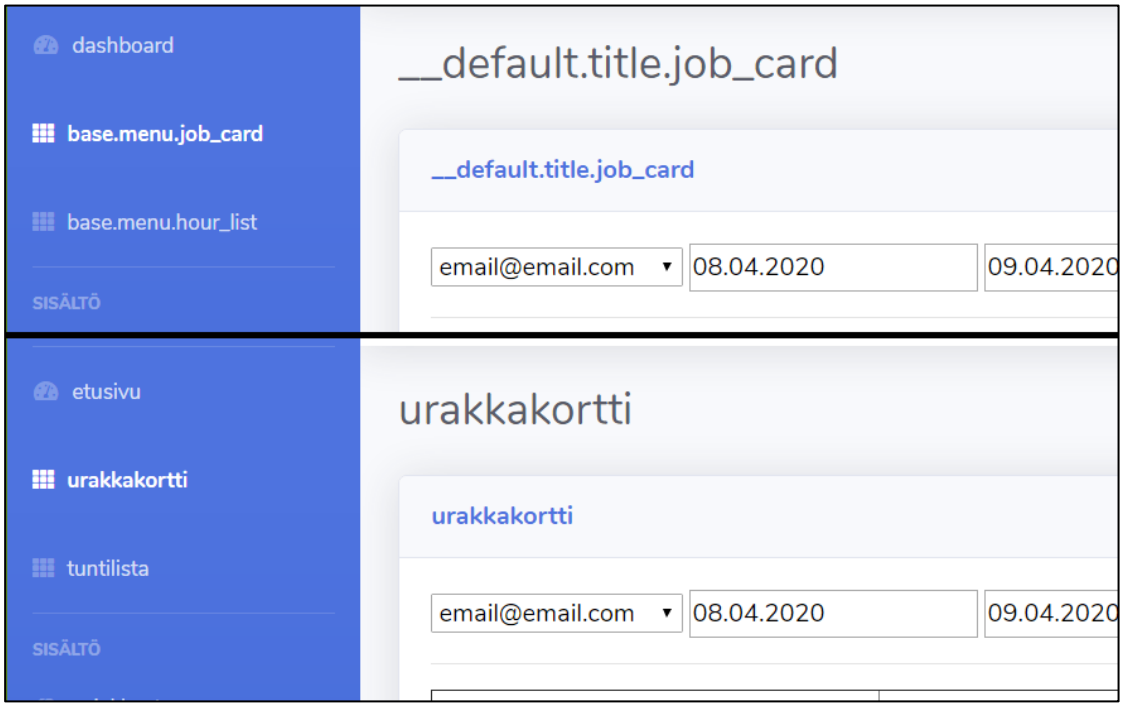
Komennon tuottamassa XLIFF-tiedostossa käytetään XML-formaattia. Siinä jokaista käännettävää merkkijonoa kohti on luotu oma yksikkönsä, jolla on oma uniikki tunnisteensa (kuva 20). Yksikön sisällä on korvattava väliaikainen merkkijono source-tagin sisällä sekä käännös target-tagin sisällä. Lisäksi

käännökseen voidaan jättää muistiinpanoja käännöstyön helpottamiseksi note-tagilla.

```
    <target>tehtävät</target>
    <note priority="1">src/Form/ConfirmationType.php:33</note>
    <note priority="1">new</note>
  </trans-unit>
  <trans-unit id="KWwMKUQ" resname="base.menu.job_card">
    <source>base.menu.job_card</source>
    <target>urakkakortti</target>
  </trans-unit>
  <trans-unit id="pwMx6AP" resname="base.menu.hour_list">
    <source>base.menu.hour_list</source>
    <target>tuntilista</target>
  </trans-unit>
  <trans-unit id="sUY8RX6" resname="base.menu.calendar">
    <source>base.menu.calendar</source>
    <target>kalenteri</target>
  </trans-unit>
  <trans-unit id="6YZdQjs" resname="calendar.title.calendar">
    <source>calendar.title.calendar</source>
```

Kuva 20. Käännökset XML-formaatissa

Esimerkki käännöksen lopputuloksesta on nähtävissä kuvassa 21. Yllä kääntämättömät väliaikaiset merkkijonot sivupalkissa sekä sivun otsikossa. Kuvan alapuoliskossa merkkijonot käännettynä suomeksi.



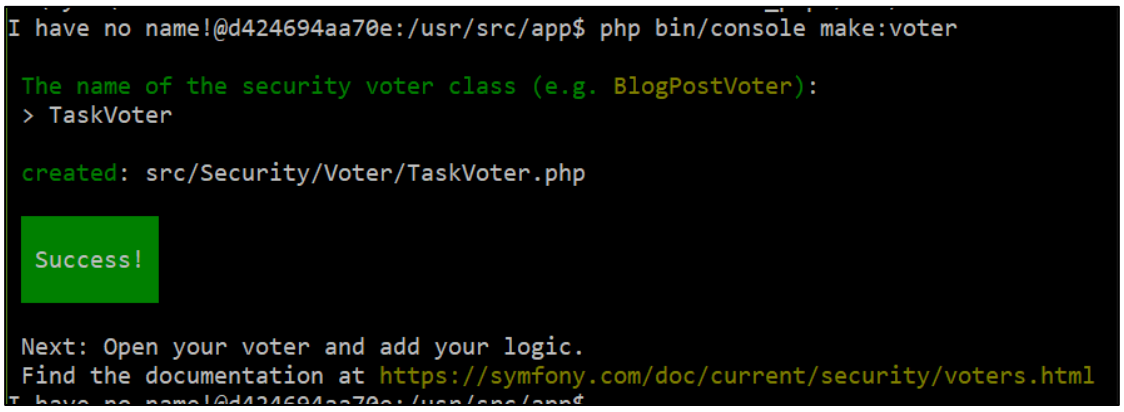
Kuva 21. Käännös sivupalkin valikossa ja sivun otsikossa

Translation-komponentti helpottaa ja automatisoi sivun kääntämistä hyvinkin pitkälle. Varsinainen käännöstyö vaatii silti paljon kielitaitoa ja manuaalista tietojen syöttämistä. Myös käännettävien merkkijonojen seurannassa on oltava tarkkana, että käännökset ovat oikein ja oikeassa kohdassa sovellusta.

3.7 Käyttöoikeuksien käsittely

Lopuksi sovellukseen luotiin oikeuskäsittelijät, jossa määritetään käyttäjien oikeudet käyttää ominaisuuksia. Käyttäjät on määritetty kolmeen eri rooliin: user, eli peruskäyttäjä, admin ja super-admin, eli ylläpitäjä ja ylempi ylläpitäjä. Tässä tapauksessa peruskäyttäjä edustaa työntekijöitä, ylläpitäjä työnjohtajia ja ylempi ylläpitäjä yrityksen ylintä johtoa. Käyttäjien roolit määritetään niitä luotaessa.

Symfonyssa oikeuskäsittely toteutetaan sivun käsittelijässä tai mukautetusti siihen liitetyssä servicessä, jota kutsutaan Voteriksi. Voterin luonti onnistuu komentokehotteessa komennolla *php bin/console make:voter*. Komento avaa Voterin luonnin, jossa Voter nimetään (kuva 22). Jokaiselle käsittelijälle luotiin oma Voterinsa, joka nimettiin oman käsittelijänsä mukaan.



Kuva 22. TaskVoter-nimisen Voter-servicen luonti

Kuvassa 23 havainnoidaan oikeuksien rajaamista käsittelijässä. Kuvan yläosassa käyttöoikeus sovelluksen ominaisuuteen rajataan käyttäjän roolin mukaan. Alaosassa kutsutaan Voterissa määriteltyä oikeuskäsittelylogiikkaa. Jokaista käsittelijää kohti luotiin oma Voter-servicensä.

```
$this->denyAccessUnlessGranted('ROLE_SUPER_ADMIN');
```

```
$user = $this->getUser();

$this->denyAccessUnlessGranted(UserVoter::USER_PROFILE_EDIT, $user);
```

Kuva 23. Oikeuden rajoitus ja Voter-servicen kutsu käsittelijässä

Voter-servicessä määritellään ensiksi muuttujiin sen sisällä käytettävät komennot merkkijonoina. Tämän jälkeen Voterin varsinaisessa äänestysmetodissa, nimeltään voteOnAttribute, hyödynnetään switch-lausettta, jossa jokaisesta määritellystä komennosta luodaan oma case, eli tapaus. Näin tapauksen sisällä oleva logiikka toteutetaan käsittelijästä tulevan kutsun mukaan. Jokaisen tapauksen on palautettava totuusarvomuuttuja *true* tai *false*, *true*n myöntäessä ja *false*n kieltäessä oikeuden ominaisuuteen.

Sovelluksessa käyttäjien muokkaus on jaettu kahteen: profiiliin, jota käyttäjät pääsevät itse muokkaamaan ja varsinaiseen käyttäjän muokkaukseen, jossa on mukana mm. roolin hallinta ja on tarkoitettu ylläpidolle. Tämän vuoksi kuvassa 24 havainnollistetussa käyttäjien Voter-servicessä tarvittiin neljää eri komentoa. Ylimmälle ylläpidolle, eli super adminille, myönnetään oikeus jokaiseen ominaisuuteen. Tavallisen käyttäjän pääsy profiilin muokkaukseen myönnetään vertaamalla käyttäjän tunnistetta avattavan profiilin tunnisteeseen.

```php
protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
{
    $user = $token->getUser();

    // if the user is anonymous, do not grant access
    if (!$user instanceof UserInterface) {
        return false;
    }

    switch ($attribute) {
        case self::USER_SHOW:
            return $user->hasRole('ROLE_SUPER_ADMIN');
            break;
        case self::USER_EDIT:
            return $user->hasRole('ROLE_SUPER_ADMIN');
            break;
        case self::USER_PROFILE_SHOW:
            if ($user->hasRole('ROLE_SUPER_ADMIN')) {
                return true;
            } else {
                /** @var User $profile_user */
                $profile_user = $subject;

                // if user is not super_admin, they can only view their own profile
                return $this->canViewProfile($profile_user, $user);
            }
            break;
        case self::USER_PROFILE_EDIT:
            if ($user->hasRole('ROLE_SUPER_ADMIN')) {
                return true;
            } else {
                /** @var User $profile_user */
                $profile_user = $subject;

                // if user is not super_admin, they can only edit their own profile
                return $this->canEditProfile($profile_user, $user);
            }
            break;
    }

    return false;
}
```

Kuva 24. Oikeuskäsittely Voterin sisällä

Jotta käyttäjä ei päädy ominaisuuksiin, joihin hänellä ei ole oikeuksia ja saa täten virheilmoituksia, on polut näihin ominaisuuksiin piilotettava käyttöliittymästä. Tämän vuoksi roolitarkistuksia voidaan ajaa myös Twig-templaateissa. Kuvassa 25 roolitarkistusta käytetään if-lauseen ehtona, jossa lauseen sisältö tulostetaan käyttöliittymään sillä ehdolla, että käyttäjällä on admin-rooli.

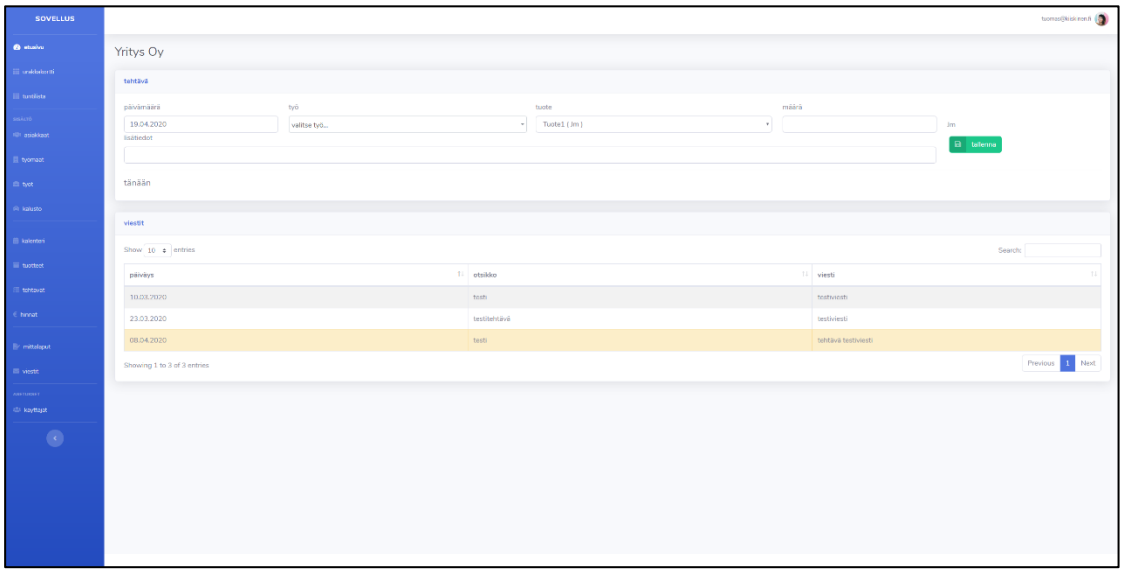
```twig
{% if is_granted('ROLE_ADMIN') %}

  <!-- Divider -->
  <hr class="sidebar-divider">

  <!-- Heading -->
  <div class="sidebar-heading">
    {{ 'base.menu.heading.settings'|trans }}
  </div>

  <!-- User Menu Item -->
  <li class="nav-item {{ route starts with 'user' ? 'active' }}">
    <a class="nav-link" href="{{ path('user_index') }}">
      <i class="fas fa-fw fa-users"></i>
      <span>{{ 'base.menu.users'|trans }}</span>
    </a>
  </li>

{% endif %}
```

Kuva 25. Roolitarkistus Twig-templaatissa

Käyttöoikeuksien käsittely on olennainen osa verkkosivun tuotantoa ja tietoturvan kannalta tärkeää. Symfonyn tarjoamat Voterit helpottavat tämän prosessin kehittämistä ja hallintaa huomattavasti. Ominaisuuden käyttöoikeuksien kokoaminen yhteen paikkaan selkeyttää koodia ja helpottaa näin kehitystyötä suuresti.

4 TULOKSET JA JOHTOPÄÄTÖKSET

Opinnäytetyön lopputuloksena saatiin toteutettua toimiva sovellus. Kehitystyö jatkuu kuitenkin käyttöliittymän hienosäätönä ja ominaisuuksien viilailuna. Sovellukseen saatetaan toteuttaa vielä myös uusia lisäominaisuuksia testien pohjalta. Kuvassa 26 esitellään sovelluksen etusivua.



Kuva 26. Sovelluksen näkymä etusivulla

Tällä hetkellä sovellus on asiakkaalla tutustumis- ja testikäytössä. Laadin sovelluksesta käyttöohjeet tätä tarkoitusta varten. Testaamalla kartoitamme mahdollisia puutteita ja muutoksia sovelluksen käyttöliittymään ja toimintoihin. Esimerkiksi yksi muutostoive, joka asiakkaalta jo tuli, oli muuttaa erään lomakkeen kenttien järjestystä.

Suurimpia haasteita työn toteutuksessa oli asiakkaan tarvitsemien ominaisuuksien kartoittaminen ja rajaus toimiviin ohjelman ominaisuuksiin. Toimintoja ja ominaisuuksia selvitettiin palaverein tietyin aikavälein. Sama käytäntö jatkunee testijakson jälkeen. Haastetta toi myös Symfonyn opettelu

ja käyttö ohjelman tuotannossa, vaikka aikaisempaa kokemusta ohjelmistokehyksestä minulla olikin työharjoittelujaksolta.

Toteutukseen valitut tekniikat soveltuivat toiminnanohjausjärjestelmän tuottamiseen mainiosti. Symfonyn komponenttikirjasto tarjosi kattavasti työkaluja sovelluksen ominaisuuksien toteuttamiseen ja automatisoi sovelluksen perustarpeita, kuten käyttöturvallisuutta ja tietojen hallintaa.

5 PÄÄTÄNTÖ

Opinnäytetyön tavoitteena oli tuottaa toimeksiantajan asiakkaalle toimiva ja tarpeita vastaava sovellus. Koska sovellus ei tätä kirjoittaessa ole vielä tuotantokäytössä, voitaisiin sanoa, että tavoitteeseen ei ylletty, mutta mielestäni tulos on varsin hyvä. Sovelluksen toteutuksessa ollaan loppuvaiheessa ja sen testiversio on toimitettu asiakkaalle.

Työn lopputuloksena saatuun sovellukseen saatiin toteutettua kaikki tärkeimmät toiminnallisuudet. Työstettäväksi jäi korjaukset, kuten käyttöliittymän ratkaisut ja sovelletut termit sekä käytettyjen jQuery- ja JavaScript-liitännäisten lokalisointi ym. Työn alle tulevat myös testauksessa paljastuvat puutteet.

Opinnäytetyön toteutusprosessi onnistui mielestäni hyvin. Jos projekti pitäisi toteuttaa uudelleen, pyrkisin parantamaan kommunikaatiota toimeksiantajan sekä sovelluksen tilaajan kanssa, jotta toteutettavista ominaisuuksista saataisiin selvempi kuva kaikille osapuolille. Tällä tavalla vältyttäisiin turhalta työltä, vaikka tätä ei projektin aikana tapahtunutkaan paljoa.

Jatkokehityksen mahdollisuuksia toiminnanohjausjärjestelmään voisivat olla esimerkiksi järjestelmän käytön opastus sovelluksen sisällä. Sovelluksen ominaisuuksiin voitaisiin sisällyttää jonkinlainen infolaatikko, jonka saisi klikatessa avattua. Myös sovelluksen viestimisominaisuutta voitaisiin laajentaa, jos siihen koetaan tarvetta. Kun järjestelmän kehitys saadaan vietyä kunnialla loppuun saakka, jää siitä itselleni käteen hyvää kokemusta full-stack sovelluskehityksestä sekä ohjelmistonkehityksen prosessista yleisellä tasolla.

LÄHTEET

BitDegree. 2020. jQuery vs JavaScript: What's your Choice?. WWW-dokumentti. Saatavissa: https://www.bitdegree.org/tutorials/jquery-vs-javascript/ [viitattu 30.3.2020].

Chaffer, J. & Swedberg, K. 2013. Learning jQuery. E-kirja. Birmingham: Packt Publishing Ltd. Saatavissa: https://kaakkuri.finna.fi/ [viitattu 28.3.2020].

Google Chrome Developer. s.a. MVC Architecture. WWW-dokumentti. Saatavissa: https://developer.chrome.com/apps/app_frameworks [viitattu 9.2.2020].

Microsoft Dynamics. s.a. Mikä ERP on ja miksi sitä tarvitaan. WWW-dokumentti. Saatavissa: https://dynamics.microsoft.com/fi-fi/erp/what-is-erp/ [viitattu 18.4.2020].

Potencier, F. & Zaninotto, F. 2007a. The Definitive Guide to symfony. E-kirja. Berkeley: Apress. Saatavissa: https://symfony.com/legacy/doc/book/1_0/en/01-Introducing-Symfony [viitattu 7.2.2020].

Potencier, F. & Zaninotto, F. 2007b. The Definitive Guide to symfony. E-kirja. Berkeley: Apress. Saatavissa: https://symfony.com/legacy/doc/book/1_0/en/02-Exploring-Symfony-s-Code [viitattu 7.2.2020].

Potencier, F. 2009. Templating Engines in PHP. Blogi. Päivitetty 7.10.2009. Saatavissa: http://fabien.potencier.org/templating-engines-in-php.html [viitattu 9.2.2020].

SensioLabs Symfony. 2020a. Translations. WWW-dokumentti. Saatavissa: https://symfony.com/doc/current/translation.html [viitattu 16.3.2020].

SensioLabs Symfony. 2020b. Databases and the Doctrine ORM. WWW-dokumentti. Saatavissa: https://symfony.com/doc/current/doctrine.html [viitattu 10.4.2020].

SensioLabs Twig. s.a. Twig for Template Designers. WWW-dokumentti. Saatavissa: https://twig.symfony.com/doc/3.x/templates.html [viitattu 28.2.2020].

SensioLabs. 2015. Symfony: The Book. E-kirja. Saatavissa: https://symfony.com/doc/2.4/book/index.html [viitattu 25.3.2020].

Stefanov, S. & Sharma, K. C. 2013. Object-Oriented JavaScript. E-kirja. Birmingham: Packt Publishing Ltd. Saatavissa: https://kaakkuri.finna.fi/ [viitattu 16.3.2020].