Eero-Pekka Halinen

Selainkäyttöisen ostojen hallintajärjestelmän toteutus Symfony2-sovelluskehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

14.4.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Eero-Pekka Halinen Selainkäyttöisen ostojen hallintajärjestelmän toteutus Symfony2-sovelluskehyksellä 40 sivua + 1 liite 14.4.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Teknologiajohtaja Tuukka Antikainen Lehtori Simo Silander

Insinöörityössä toteutettiin Pointteri Oy:n asiakkaalle selaimella käytettävä työkalu heidän tuotteidensa ostoprosessin helpottamiseksi. Järjestelmän tarkoitus oli mahdollistaa eri maissa toimivien ostajien helppo kommunikointi ostopäätöksen tekevien ostopäälliköiden kanssa erilaisilla päätelaitteilla.

Työn raportti sisältää järjestelmän määrittelyn, erittelee sen toteutukseen käytetyt työvälineet ja tutkii, kuinka Symfony2-sovelluskehys soveltuu olennaisimpine osineen tämän kaltaisen sovelluksen toteuttamiseen. Raportti tutkii sovelluskehyksessä käytettyjä hyviä käytäntöjä ja suunnittelumalleja, joiden pohjalle sen arkkitehtuuri on rakennettu.

Symfony2 on PHP-kielellä toteutettu laaja ja laajalti käytetty oliopohjainen sovelluskehys, joka avoimen lähdekoodinsa ansiosta on ollut pitkään suosittu ja jota kehitetään edelleen. Se käyttää MVC-mallia ja riippuvuusinjektiota olennaisimpina arkkitehtuurisina elementteinään. Siinä on myös monipuolisesti toteutettu, web-sovelluksille tärkeä lomakkeiden käsittely.

MVC-mallin todettiin yhä olevan hyvä ratkaisu sovelluksen rakenteen eri kerrosten erottelemiseksi. Palvelukeskeinen arkkitehtuuri osoittautui myös toimivaksi ratkaisuksi modulaarisen ja ylläpidettävän sovelluksen toteutukseen.

Avainsanat	Symfony2, riippuvuusinjektio, PHP, sovelluskehys, ORM

Author Title Number of Pages Date	Eero-Pekka Halinen Implementing Online Purchase Workflow System Using Symfony2 Framework 40 pages + 1 appendix 14 April 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Tuukka Antikainen, Chief Technology Officer Simo Silander, Senior Lecturer

The aim of this thesis was to implement a web-based tool for a client of Pointteri Oy to facilitate their product purchase process. The purpose of the system was to enable buyers in different countries to communicate efficiently using different devices with the purchase managers making the final purchase decision.

This paper includes the system specification, looks at the tools used in its implementation and studies how the Symfony2 framework and its most essential parts can be used in implementing a system such as the one produced. The paper studies the best practices and design patterns the architecture of the framework is based on.

Symfony2 is an extensive and widely used object-oriented PHP framework which has long been popular due to it being open source and still being developed further. It uses the MVC-pattern and dependency injection as its most integral architectural elements. It also has a versatile form handling component crucial for web-applications.

The MVC-pattern was established as a good practice for separating the different architectural layers of an application. Service-oriented architecture was also proven to be a useful solution for implementing a modular and maintainable application.

Keywords	Symfony2, dependency injection, PHP, framework, ORM

Sisällys

Lyhenteet ja määritelmät

Active Record	Rakenteellinen suunnittelumalli, jossa oliot osaavat tallentaa itsensä tietokantaan.
Ad hoc -kysely	Tiettyä tarkoitusta varten tehty, dynaamisesti muuttuva tietokantakysely.
AJAX	Asynchronous JavaScript and XML. Teknologioita, joilla websovelluksista tehdään vuorovaikutteisempia.
Annotaatio	Syntaktista metadataa, joka vaikuttaa ohjelman ajonaikaiseen käytökseen.
API	Application Programming Interface, ohjelmointirajapinta.
Boilerplate	Koodia, joka esiintyy useassa paikassa liki tai täysin samanlaisena.
Bundle	Symfonyn moduulia tai pluginia vastaava käsite, ohjelman rakenteellinen, irrotettava osa.
Byte Code Cache	Valmiiksi käännetyn koodin varastoiva välimuisti.
CLI	Command-line interface. Tekstuaalinen komentopohjainen käyttöliittymä.
cron	Ajastuspalvelu Unix-pohjaisille käyttöjärjestelmille.
CRUD	Create, Read, Update, Delete. Pysyvän tallennuksen neljä perusfunktiota ja niitä toteuttava sovellus.
CSRF	Cross-site Request Forgery. Tietoturvaongelma, jossa hyökkääjä naamioituu luotetuksi käyttäjäksi.
Data Mapper	Rakenteellinen suunnittelumalli, jossa olioita tallennetaan muistista tietokantaan.

DOM	Document Object Model, dokumenttioliomalli. Puumainen tapa kuvata rakenteisen dokumentin rakenne.
Dynaaminen sidonta	Myöhäinen sidonta, muuttujan tyypin päätteleminen suoritusvaiheessa.
Fikstuuri	Tunnettu tila, sen osa tai prosessi tilaan pääsemiseksi, jossa järjestelmää voidaan testata.
IoC	Inversion of Control. Suunnittelutapa, jossa tarkoitusta varten tehdyt ohjelman osat suoritetaan yleiseen tarkoitukseen tehtyjen osien kutsumana.
jQuery	Suosittu avoimen lähdekoodin JavaScript-kirjasto front end -perustoimintoihin.
MVC	Model-View-Controller, Malli-Näkymä-Käsittelijä. Ohjelmistoarkkitehtuurityyli, jossa käyttöliittymä erotetaan sovellusaluetiedosta.
ORM	Object-relational mapping. Oliomallin mukaisen esityksen kuvaus relaatiomallin mukaiseksi esitykseksi.
PHPUnit	PHP:n yksikkötestaussovelluskehys.
Responsiivisuus	Sovelluksen käyttäjän päätelaitteen ominaisuuksiin mukautuvuus.
Skeema	Tietokannan rakennekuvaus sen tuntemalla formaalilla kielellä.
SOA	Service Oriented Architecture, palvelukeskeinen arkkitehtuuri. Suunnittelutapa, jossa järjestelmän eri osat toimivat itsenäisesti.
SQL	Structured Query Language. Kyselykieli, jolla relaatiotietokantaan tehdään hakuja, muutoksia ja lisäyksiä.

Token	Esimerkiksi autentikoinnissa käytetty uniikki merkkijono.
Twig	Avoimen lähdekoodin näkymämoottori PHP:lle.
XML	Extensible Markup Language, merkintäkieli, jolla tiedon merkitys kuvataan tiedon kanssa.
YAML	YAML Ain’t Another Markup Language, alkujaan Yet Another Markup Language. Merkintäkieli.

1 Johdanto

Tämä insinöörityö tehtiin ohjelmistoyhtiö Pointteri Oy:lle. Pointterin toiminta keskittyy verkkopalveluina toteutettuihin yritysten palvelunohjausjärjestelmiin, verkkokauppoihin ja online-tilausjärjestelmiin. Tämä työ raportoi, kuinka Symfony2-sovelluskehyksellä toteutettiin Pointterin eräälle asiakkaalle sisäiseen käyttöön tarkoitettu tuotteiden oston hallintajärjestelmä.

Tilaajalle syntyi tarve saada keskitetty tapa hallinnoida eri maissa toimivien ostajiensa potentiaalisia tuoteostoja. Toteutettu järjestelmä helpottaa heidän työnkulkuaan antamalla mahdollisuuden syöttää selaimen kautta järjestelmään dataa, kuvia ja dokumentteja tuotteista, joita ostopäälliköt voivat käydä läpi antaen hintakaton ja ostoluvan sekä päättämällä, missä maassa tuote lopulta myydään. Järjestelmän vaatimukset syntyivät asiakastapaamisissa, joissa luotiin käyttötapaukset ja järjestelmän spesifikaatio.

Järjestelmän toteutukseen käytetään Pointterissa hyväksi havaittujen työvälineiden uusia versioita. Raportin pääpaino on Symfony2-sovelluskehyksen ominaisuuksissa ja siinä, kuinka sitä voidaan käyttää tämän kaltaisen sovelluksen toteuttamiseen. Työssä tarkastellaan tuloksena olleen järjestelmän kautta siinä käytetyn sovelluskehyksen taustalla olevia tärkeimpiä ohjelmointiparadigmoja ja suunnittelumalleja ja sitä, kuinka ne on kyseisessä ohjelmistokehyksessä toteutettu. Aineistona käytetään pääasiassa sovelluskehyksen dokumentaatiota.

2 Määrittely

2.1 Järjestelmän tarpeellisuus

Työn tilaajan prosessi toimii siten, että Euroopassa toimivat ostajat etsivät sopivia tuotteita ja ilmoittavat niistä ostopäälliköille. Ostopäälliköt tietävät tuotteiden tarpeen ja osaavat määrittää niille kattohinnan, jonka ostaja voi tuotteesta myyjälle maksaa. Päätöksentekoon vaikuttavat kunkin oston kohdalla myös tuotekohtainen vero ja maittain vaihteleva arvonlisävero. Tuotteen ostamisen kannattavuus vaihtelee sen lopullisen myyntimaan mukaan, joten uuteen järjestelmään tulee voida syöttää tuotteeseen liittyviä erilaisia hintoja ja kuluja ja esittää ne mahdollisimman selkeästi, jotta niitä voidaan hyödyntää päätöksentekoprosessissa.

Toinen tapa, jolla uusi tuote voi järjestelmään tulla, on myyjäroolissa toimivan käyttäjän kautta. Myyjä syöttää useimmissa tapauksissa asiakkaan toivomuksesta lähteneen tuotteen perustiedot järjestelmään, jolloin ostajat voivat etsiä vastaavaa tuotetta omilta markkinoiltaan. Ostopäälliköt voivat seurata tuotteen tilaa uudesta tai toivotusta myyntikuntoiseksi asti koko ostoprosessin läpi.

Kullakin tuotteella on ostoprosessin alun perin käynnistänyt ostaja tai myyjä vastuuhenkilönään. Tämän lisäksi tuote saa vastuuhenkilökseen siitä kiinnostuneen ostopäällikön. Tuotteeseen liittyvät vastuuhenkilöt voivat viestiä siihen liittyen paitsi epäsuorasti antamalla hintoja, mutta myös suorille viesteille on erilaisia käyttötilanteita. Ostopäällikkö voi haluta ostajalta tuotteesta lisätietoja, mutta myös ostopäälliköt voivat keskenään viestiä siitä, mihin maahan tuotteen olisi järkevintä päätyä. Tähän päätökseen vaikuttavat eri maiden erilaiset arvonlisäverot, joita on syytä pystyä helposti vertailemaan. Samasta syystä järjestelmän on pystyttävä tekemään valuuttamuunnokset yksinkertaisiksi.

2.2 Tärkeimmät käyttötapaukset

Sovellusta käytetään selaimen välityksellä, jotta päivitykset voidaan tehdä keskitetysti, sovellusta ei ole tarpeen asentaa ja siihen on helppo päästä käsiksi. Se sisältää erilaisia käyttäjärooleja, joista tärkeimmät ovat ostaja ja ostopäällikkö. Sisäänkirjautumisen jälkeen kukin käyttäjä näkee listan tuotteista, jotka hänelle on tiettyjen kriteerien, kuten päivitysajankohdan ja tuotteen tilan mukaan suodatettu. Tuotteista näytetään niiden merkki,

malli, tila, vuosimalli ja tärkeimmät hinnat. Samalla käyttäjä näkee kohteita koskevat uusimmat viestit, joihin on suoraan etusivulta mahdollisuus vastata. Kaikki vastaanotetut viestit ovat nähtävissä erillisessä viestiosiossa.

Tärkeä ominaisuus on myös tuotteiden haku. Haun päätteeksi käyttäjä saa samanlaisen listauksen tuotteista, kuin hän etusivulla näkee. Listauksessa ovat linkit varsinaiselle tuotesivulle, jossa niitä voi muokata. Uusi kohde voidaan lisätä valitsemalla valikosta ”Uusi tuote”, mikäli käyttäjän rooliin kuuluu tuotteiden lisääminen.

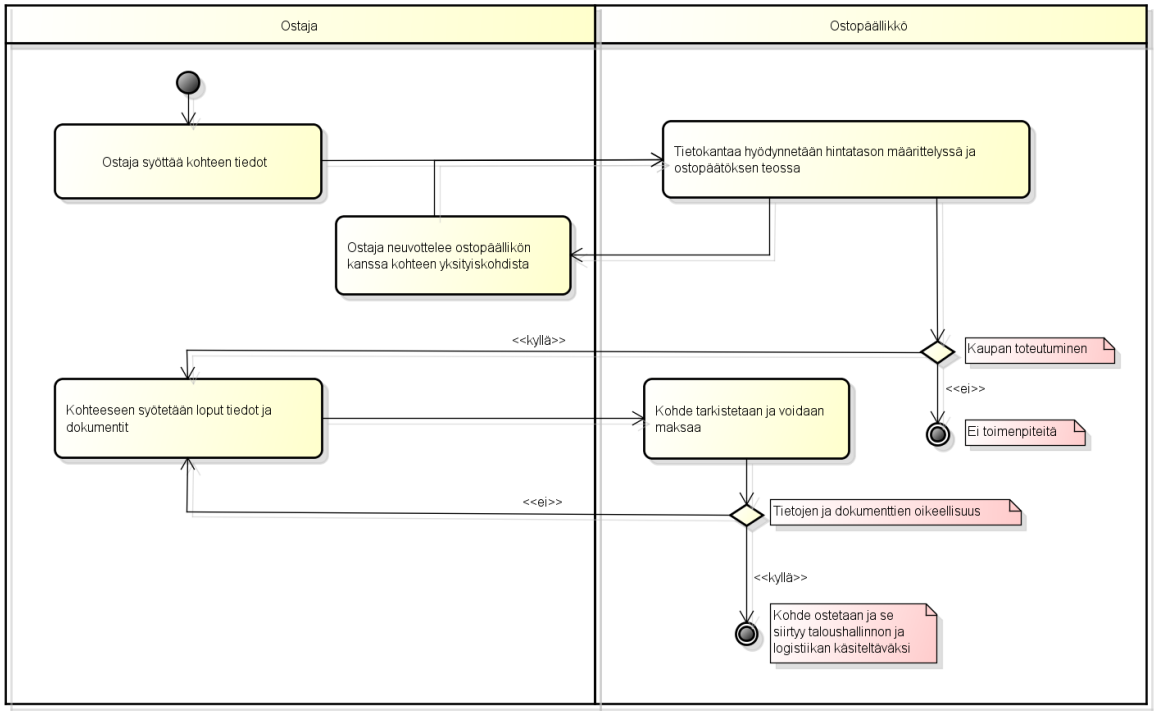
Käyttäjä ohjataan tuotesivulla hänen roolinsa mukaiselle välilehdelle. Näitä ovat varsinainen tuotevälilehti, taloushallinnon välilehti, logistiikan välilehti ja dokumenttivälilehti. Taloushallinnon sivulla tuotteelle voi syöttää muita kuluja ja tarkastella hintahistoriaa. Logistiikan sivulla ovat tuotteen lähtöpaikka ja lopullinen sijainti. Dokumentit-välilehdellä voi syöttää tuotteeseen liittyviä dokumentteja. Ennen kuin tuotteita syötetään, tarvitsee järjestelmään lisätä tuotemerkkejä ja -malleja sekä myyjäliikkeitä.

Tuotesivulla ovat kaikki tuotteen tiedot syötettävissä lomakkeelle sekä tuotteeseen liittyvät viestit. Samoin sivulla voi lisätä kuvia tuotteesta. Hintoja on neljä tyyppiä arvonlisäveroineen ja valuuttoineen: pyyntihinta, kattohinta, tarjottu hinta ja lopullinen hinta. Hintoja voi katsella eri valuutoissa ja arvonlisäveroissa valitsemalla sivulla olevasta muuntajasta niille eri arvoja. Muunnetut arvot näkyvät kyseisen hinnan alapuolella. Arvonlisäverot ovat staattisia, ja ne päivitetään tarpeen mukaan. Valuuttakurssit päivitetään kerran päivässä tietokantaan.

Käyttäjä voi myös muokata omia tietojaan ja vaihtaa salasanansa. Pääkäyttäjä pystyy lisäksi lisäämään uusia käyttäjiä, hakemaan käyttäjiä ja vaihtamaan muiden käyttäjien salasanoja. Kun uusi käyttäjä luodaan, lähtee hänelle määriteltyyn sähköpostiosoitteeseen linkki, josta hän pääsee asettamaan salasanansa. Samanlainen sähköposti lähertetään käyttäjän unohtaessa salasanansa. Täydempi lista edellä mainituista käyttötapauksista on taulukkomuodossa liitteessä 1.

2.3 Aktiviteettikaavio

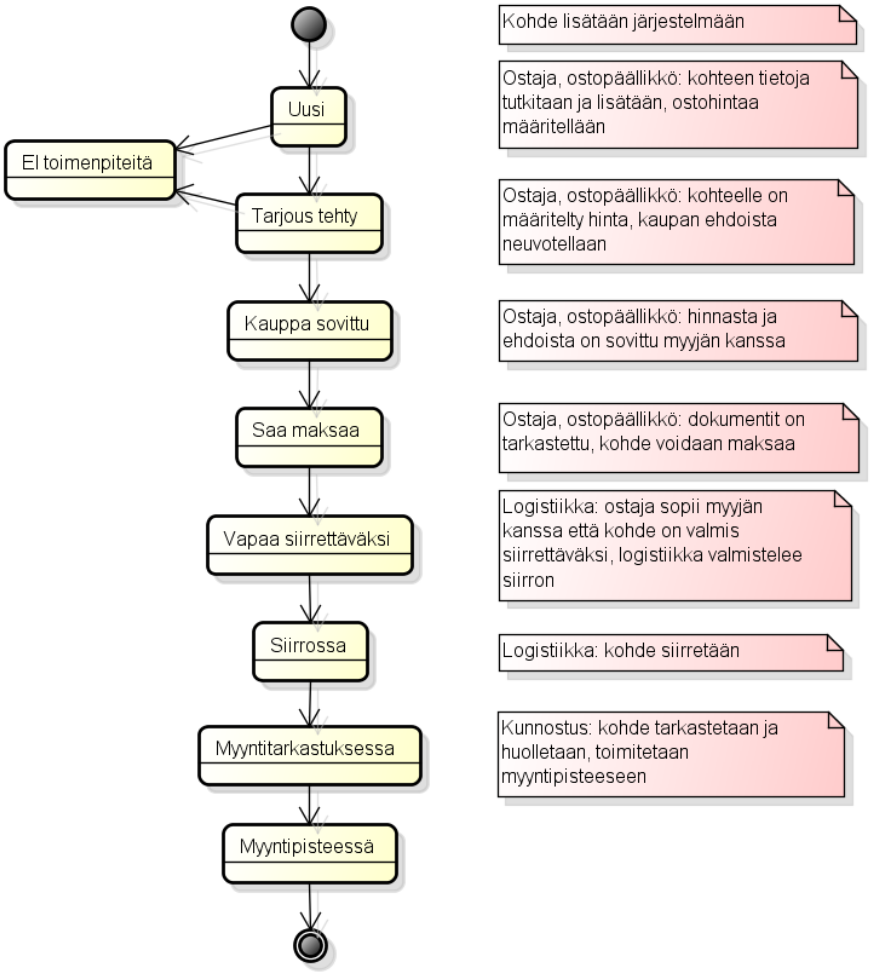
Kuvassa 1 näkyy tuotteen normaali elinkaari järjestelmässä sen syöttämisestä ostopäätöksen tekemiseen asti. Useimmiten tuotteen syöttää ostaja. Ostopäällikkö on aina mukana tuotteen ostopäätöksen tekemisessä.



Kuva 1. Aktiviteettikaavio

2.4 Tilakaavio

Kuvassa 2 näkyvät järjestyksessä tilat, joita tuotteella sen elinkaaren aikana järjestelmässä on. Tuote alkaa uutena yleensä ostajan tai myyjän syöttämänä, jälkimmäisessä tapauksessa silloin, kun tuotteelle on vain tarve ja sitä etsitään. Tällöin tuote on wanted-tilassa. Kun myönteinen ostopäätös on tehty tai wanted-tuote löydetty, siirtyy tuote logistiikalle ja kunnostukselle tarkoitettuihin tiloihin ja lopulta päätyy myyntipisteeseen.



Kuva 2. Tilakaavio

2.5 Käytettävyysvaatimukset

Järjestelmään tulee olla helppo pääsy useilta päätelaitteilta mobiililaitteet mukaan lukien. Tästä syystä tuote toteutettiin pilvipalveluna ja responsiivisuuden periaatetta käyttäen. Käyttäjän on myös päästävä mahdollisimman helposti käsiksi hänelle olennaiseen informaatioon mahdollisimman vähillä klikkauksilla. Järjestelmän etusivu palvelee tätä tarkoitusta.

Tuotelomake ei saa olla liian hankala täyttää, eli pakollisten kenttien määrä on oltava pieni ja kenttien määrän rajattu. Mahdollisia lisätietoja voi syöttää dokumenttien muodossa. Eri liiketoimintaroolien toimintojen on oltava eroteltuja, mikä toteutettiin tuotesivun välilehtien avulla.

3 Työvälineiden valinta

3.1 Back end -sovelluskehys

PHP

PHP oli jo valmiiksi käytetty kieli Pointterin projekteissa, joten tämä ja nopea prototyypitys puhuivat sen valinnan puolesta tähänkin projektiin. Se on dynaamisten web-sivujen luontiin suunniteltu komentosarjakieli, joka tarjoaa laajat luokkakirjastot. Se sai alkunsa vuonna 1994 [1, s. 7] ja on kasvanut yhdeksi suosituimmista ohjelmointikielistä web-ohjelmointikäytössä. PHP:n kehitys on jatkuvaa, ja se on kehityskaarensa aikana perinyt yhä enemmän ominaisuuksia muista kielistä, kuten modernissa ohjelmistokehityksessä hyvin olennaisen oliopohjaisuuden versiossa 5 [1, s.7].

PHP:tä kohtaan on esitetty osittain perusteltuakin kritiikkiä esimerkiksi sen heikosta tyypityksestä, mutta kielessä on myös paljon hyviä puolia. Kehittäminen sillä on ketterää, ja prototyyppejä voidaan saada nopeasti aikaiseksi. Kritisoitua tyypityksen puutetta on korjattu ja kielellä on isoja kehittäjiä ja tukijoita, kuten Facebook [2]. PHP skaalautuu hyvin, eli sopii niin pieniin kuin isompiinkin projekteihin, jälkimmäisiin varsinkin käytettäessä jotakin monista sovelluskehyksistä.

Sovelluskehys

Keskisuuressa tai isommassa web-sovellusprojektissa sovelluskehyksen käyttö on aiheellista. Se korottaa projektin laatua tuomalla siihen rakennetta hyväksi havaittujen suunnittelumallien kautta ja tekee projektista helpommin ylläpidettävän ja päivitettävän. Myös kehittämisen nopeus lisääntyy sovelluskehyksen mukanaan tuomien uudelleenkäytettävien moduuleiden kautta, jolloin kehityksessä voidaan keskittyä olennaiseen, eli liiketoiminnan vaatimaan logiikkaan ilman, että kuitenkaan sitoudutaan lopullisesti tiettyyn sovelluskehykseen. [3.]

Sovelluskehyksen avulla päästään eroon yksinkertaisista, mutta aikaa vievistä ja toistuvista tehtävistä, jotka liki jokaisessa sovelluksessa tulee jollakin tavalla toteuttaa, eli niin sanotusta boilerplate-koodista. Hyvä sovelluskehys tarjoaa valmiit tai melkein valmiit ratkaisut yleisimpiin sovelluskehityksen ongelmiin. Hyväksi havaittujen mallien ja valmiiden moduuleiden käyttö varmistaa myös sen, että muutkin kuin ohjelman alkuperäiset tekijät voivat omaksua sen helposti sekä kehittää ja ylläpitää sitä tulevaisuudessa.

Sovelluskehykseksi valittiin Symfony2. Paitsi että Symfony2-sovelluskehystä ja myös sen vanhempia versioita oli käytetty jo aiemmissa projekteissa, on se myös yksi tunnetuimmista PHP-sovelluskehyksistä. Se syntyi vuonna 2005 ja on kehittynyt vakaaksi kehykseksi, jonka ympärillä on aktiivinen käyttäjäkunta. Lähdekoodin avoimuus ja kehittäjien aktiivisuus takaavat, että sovelluskehyksen varaan voi rakentaa ohjelmistoja tulevaisuudessakin, ohjelmistokehys saa päivityksiä ja että sille löytyy tukea ja dokumentaatiota. [4.]

Kehittäjät lupaavat jokaiselle Symfonyn pääversiolle kolmen vuoden tuen ja pienemmillekin versioille rajapintojen yhteensopivuuden [5]. Symfony2 käyttää osittain jo PHP-standardeiksi muodostuneita välineitä, kuten yksikkötestaus-sovelluskehystä PHPUnit, ja on muutenkin modulaarisesti rakennettu. Symfony kutsuu moduuleita bundleiksi, ja kehittäjä voi käyttää itselleen hyödyllisiä kolmansien osapuolten bundleja tai kirjoittaa omiansa. Kehittämisen avuksi sovelluskehys tarjoaa debug-välineitä ja valmiita komentorivikomentoja. Symfony2-sovelluskehyksen kehittäjät kiinnittävät huomiota suorituskykyyn ja ovat parantaneet sitä joka versiossa. Sovelluskehys on jo valmiiksi varsin nopea, mutta on edelleen optimoitavissa esimerkiksi käyttämällä byte code cachea [6].

3.2 Front end -sovelluskehys

HTML

Pilvipalveluna toteutettu järjestelmä tarjotaan käyttäjän selaimelle luonnollisesti HTML-kielellä (Hypertext Markup Language, hypertekstin merkintäkieli). PHP pohjimmiltaan määrittää sovelluksen logiikan, jonka lopputuloksena on palvelimelta käyttäjälle lähetettyä HTML-merkintää, jonka selain tulkitsee ja esittää visuaalisesti. HTML esittää ideaalitapauksessa vain sivun rakenteen ja sisällön tägimuodossa, jolloin sivun varsinainen visuaalinen ilme on jätetty CSS-määrittelyjen (Cascading Style Sheets, porrastetut tyyliarkit) varaan. World Wide Web Consortium pitää yllä sekä HTML- että CSS-standardeja.

CSS

CSS (Cascading Style Sheets) on erityisesti WWW-sivuille kehitetty tapa määrittää niiden visuaalista ilmettä eriytettynä sivun rakenteesta. CSS-määrittelyjen käyttö tuo koodiin selkeyttä, johdonmukaisuutta, ylläpidettävyyttä ja ajansäästöä. Sana "cascading" (porrastettu) tarkoittaa sitä, että tyyliohjeilla on hierarkia, jossa tarkemmat ohjeet voivat korvata ylemmän tason ohjeita.

CSS-tiedosto, joka lähetetään palvelimelta käyttäjän koneelle siinä missä HTML-koodikin, koostuu tyyliohjeista. Tyyliohjeissa määritellään valitsin, johon kukin tyyliohje kohdistetaan. Valitsin voi muodostua esimerkiksi HTML-elementin nimestä, luokasta tai id-attribuutista. Se voi myös olla abstraktimpi pseudoelementti tai pseudoluokka, joka ei ole suoraan kytköksissä sivun rakenteeseen, DOM:iin (Document Object Model) [7]. Pseudoelementti voi esimerkiksi viitata tekstin ensimmäiseen kirjaimeen, ja pseudoluokka voi muuttua käyttäjän toiminnan seurauksena dynaamisesti, esimerkiksi klikkauksen seurauksena.

JavaScript ja jQuery

Moderni sovellus tarvitsee täyden käyttökokemuksen aikaansaamiseksi JavaScriptiä ja AJAX-toimintoja (Asynchronous JavaScript and XML). Näillä saadaan aikaan sekä näyttävämpi ja käytettävämpi visuaalinen ilme, mutta myös voidaan vastata responsiivisuuden vaatimuksiin helpommin. JavaScript-teknologioiden tarkoituksena on toteutetussa

järjestelmässä ja muissa web-pohjaisissa sovelluksissa parantaa käyttäjän käyttökokemusta. JavaScript on alun perin Netscape Communications Corporationin kehittämä komentosarjakieli, jonka yleinen käyttötarkoitus selaimessa on lisätä käyttöliittymän interaktiivisuutta ja dynaamisuutta. Sillä voidaan esimerkiksi antaa selaimelle komentoja, kommunikoida palvelimen kanssa asynkronisesti ja muokata DOM:ia. JavaScriptiä voidaan kirjoittaa omiin tiedostoihinsa, jotka käyttäjän selain lataa, tai script-tägin sisään HTML:n sekaan.

JavaScriptin käyttöä on syytä helpottaa ja nopeuttaa käyttämällä jotakin kirjastoa, johon on sisällytetty vähintäänkin yleisimmät web-ohjelmoinnin vaatimat perusominaisuudet. jQuery on avoimeen lähdekoodiin perustuva yleisimmin käytössä oleva JavaScript-kirjasto, joka helpottaa yleisimmin vaadittujen toimintojen toteuttamista web-sovelluskehityksessä [8]. Sen etuina ovat yksinkertaisuus, selainten välinen yhteensopivuus, laajennettavuus ja mahdollisuus erottaa HTML- ja JavaScript-koodi toisistaan. [9.]

Bootstrap ja Inspinia

Kehitystyön nopeuttamiseksi järjestelmään päätettiin valita valmis visuaalinen teema. Tällöin työläitä tyylimäärittelyjä ei tarvitse kirjoittaa ja toteuttaa alusta lähtien. Valmiin teeman käyttäminen julkisesti saatavilla olevalla sivustolla tarkoittaisi, että se olisi mahdollisesti tyylillisesti hyvin samankaltainen monen muun jo olemassa olevan sivuston kanssa. Tämä voi asiakasvaatimuksista riippuen olla ongelma, mutta sisäiseen käyttöön tarkoitetun järjestelmän tapauksessa näin ei ole. Toteutettua järjestelmää on tarkoitus voida käyttää useilla päätelaitteilla, joten valmiiksi toteutettu responsiivisuus voi nopeuttaa kehitystyötä frontendin osalta.

Bootstrap on Twitterin sisällä alun perin kehitetty avoin HTML5- ja CSS3-standardeja hyödyntävä frontend-sovelluskehys, joka sisältää valmiita HTML- ja CSS-määrittelyitä esimerkiksi fonteille, painikkeille, navigoinnille ja muille käyttöliittymäkomponenteille. Siinä on mukana valinnaisia JavaScript-kirjastoja (esimerkiksi jQueryn lisäosia), jotka lisäävät siihen toiminnallisuutta. Bootstrappia on versiosta 3 alkaen kehitetty mobiilikäyttö edellä painottaen responsiivisuutta [10], eli käyttöliittymän reagoimista käyttäjän päätelaitteen näytön kokoon ja navigointityyliin. Tarkoituksena on minimoida käyttäjältä vaaditun panoroinnin, suurentamisen, pienentämisen ja vierityksen määrä.

Bootstrapille on saatavissa valmiita teemoja ja sapluunoita, jotka lisäävät siihen käyttöliittymätoimintoja ja -elementtejä. Järjestelmään valittiin kehityksen nopeuttamiseksi ja järjestelmän ei-julkisen käyttötarkoituksen vuoksi valmis Inspinia-niminen teema. Teema sisälsi suurimman osan vaadittavista ominaisuuksista, kuten lomake-elementit, gallerian, valikon, taulukoita ja eri sivupohjia, kuten sisäänkirjautumislomakkeen. Teemalla saatiin helposti aikaan yhtenäinen visuaalinen tyyli ja responsiivisuus.

3.3 Muut välineet

Composer

Sovellusta kehittäessä tarvitaan useimmiten monenlaisia kirjastoja ja paketteja, jotka myös usein riippuvat toisistaan. Pakettien ja niiden riippuvuuksien hallinnasta voi projektin edetessä tulla hyvin hankalaa, joten paketinhallintajärjestelmän käyttäminen on suotavaa. Paketinhallintajärjestelmällä tarpeellisten kirjastojen päivittäminen voidaan automatisoida. Composer on PHP:lle kehitetty sovellustason paketinhallintajärjestelmä, joka ajetaan komentoriviltä, ja joka hallitsee myös asennettujen luokkien lataamisen. Composerin asetuksiin voidaan tarkasti määritellä, mihin versioon asti kutakin pakettia halutaan päivittää. Pakettia poistettaessa Composer poistaa myös siihen liittyvät riippuvuudet, ellei jokin toinen paketti niitä tarvitse. Symfony-projekti on myös helppo luoda Composerin create-project-komennolla.

Debugging

Kehitystyössä on tärkeää, että käytetään kehitystyökaluja, jotka näyttävät mahdollisimman kattavasti ja helposti mahdollisten virheiden lähteen. Symfony2 tarjoaa kehitysympäristössä kehittäjän avuksi Web Debug Toolbarin, joka näyttää yhteenvedon datasta, jota sovelluskehys käsittelee. Toolbar näyttää tietoa muun muassa HTTP-pyynnöstä, kulloisestakin ajoympäristöstä, kontrollerista, reitistä, AJAX-pyynnöistä, käyttäjästä, muistinkäytöstä ja tietokantakyselyistä. Tarkempiin tietoihin (Web Profiler) pääsee myös käsiksi toolbarin kautta. Profiler tarjoaa tarkempaa tietoa esimerkiksi sivulla olevista lomakkeista, asetuksista, poikkeuksista, tapahtumista ja logeista.

Tärkeää websovelluksen kannalta on myös tarkkailla sen toimintaa selaimen päässä, eli käyttäjän päätelaitteen ympäristössä. Sovelluksen visuaalinen ilme tulee testata eri selaimilla, mutta toiminnallisuuteen riittävät pitkälti jonkin selaimen tarjoamat kehittäjätyökalut. Google Chromen kehittäjätyökalut mahdollistavat sivun asettelun tarkkailun, JavaScriptin tulosteiden seuraamisen ja koodin optimoinnin suorituskyvyn tarkkailun kautta. DOM-näkymästä voi seurata elementtien attribuutteja ja konsolinäkymästä voi asettaa JavaScript-pysäytyspisteitä. Network-näkymä mahdollistaa HTTP-pyyntöjen ja -vastausten seuraamisen. [11.]

Tietokanta

Relaatiotietokannan käyttäminen on tämän järjestelmän tapauksessa aiheellista johtuen sen keskitason kompleksisuudesta, ad hoc -kyselyiden tarpeellisuudesta ja datan rakenteesta [12]. PostgreSQL on avoimeen lähdekoodin perustuva, SQL-standardia seuraava tietokannan hallintajärjestelmä. Se on monipuolinen, luotettava ja eheä. Toteutettu järjestelmä ei erityisesti tarvitse PostgreSQL:n uniikeimpia ominaisuuksia, mutta on ollut aiemmin käytössä ja on hyvä vaihtoehto esimerkiksi MySQL-järjestelmälle. [13]

Propel ORM (Object-relational Mapping) on kirjasto, jolla voidaan abstrahoida sovelluksen tietokantakerros. ORM:n käyttäminen mahdollistaa tietokantajärjestelmän vaihtamisen tarpeen vaatiessa. Propel oli Symfonyn oletusarvoinen ORM ainakin sen versioon 1.2 asti [14] ja on vieläkin aktiivinen projekti, joka on helppo sovittaa Symfonyn kanssa yhteen. ORM:ää valitessa se vertautuu lähinnä Doctrineen, joka on samankaltainen ja myös Symfonyn kanssa käytetty järjestelmä.

Doctrine on Data Mapper -pohjainen, joka on hieman monimutkaisempi tapa toteuttaa ORM kuin Propelin Active Record -tyyli. Active Recordissa yksi tietokannan rivi on sidottu yhteen PHP-olioon. Tämä on intuitiivinen ja riittävä tapa tavallisen CRUD-sovelluksen (create, read, update and delete) toteuttamiseen. Data Mapperin tarkoitus on erottaa olion esitys muistissa sen esityksestä tietokannassa. Propelissa mallia hallitaan tietokannan rakenneskeemalla, Doctrinessa taas PHP-luokkiin liitettävällä metadatalla. Data Mapper -pohjaisessa ORM:issa käytetään jotakin erillistä mapper-luokkaa olioiden tallennukseen, kun taas Active Record -olio osaa tallentaa itse itsensä.

4 Symfony2-sovelluskehys

4.1 Arkkitehtuuri

Symfony-sovelluksen hakemistorakenne on yleensä seuraavanlainen:

- app/ sisältää sovelluksen pääasetustiedostot, välimuistin, logit ja ytimen.
- src/ sisältää sovelluksen lähdekoodin.
- vendor/ sisältää kirjastot.
- web/ sisältää front controllerin ja resurssit. [15.]

Symfony käyttää front controller -suunnittelumallia käsittelemään jokaisen HTTP-pyynnön. Front controller sijaitsee web-kansiossa, joka on Symfony-sovelluksen ainoa suoraan ulospäin näkyvä kansio. Front controller tarjoaa keskitetyn aloituspisteen sovellukselle [16] ja voi näin hallita kaikkia toimintoja, jotka ovat yhteisiä jokaiselle pyynnölle. Navigointia eri sivuille on myös helpompi hallita keskitetysti. Symfonyn front controller sijaitsee sovelluksen web-kansiossa ja on tuotantoympäristössä oletusarvoisesti nimeltään *app.php* ja muissa ympäristöissä *app_ympäristö.php.* Front controllerissa voidaan Symfonyssä asettaa, missä ympäristössä sovellus ajetaan ja ovatko debug-toiminnot päällä. Reititinkomponentti tutkii pyynnön URI:n ja muuta tietoa, etsii siihen sopivan reitin ja päättää, mihin käsittelijään pyyntö näin ohjataan. [17.]

Front controller alustaa sovelluksen käyttämällä AppKernel-ydinluokkaa, luo pyyntöolion *Request* ja antaa sen ytimelle. Viimeiseksi se lähettää ytimeltä saamansa vastausolion käyttäjälle. *AppKernel* rekisteröi sovelluksen kaikki bundlet ja lataa asetukset. Composerin generoima autoloader lataa kaikki PHP-luokat automaattisesti. [18.]

Kaikki Symfonyn koodi on organisoitu bundleihin. Myös kehittäjän kirjoittama sovelluskohtainen koodi järjestellään bundleihin aihealueen mukaan. Bundlet sisältävät lähdekoodinsa ja siihen liityvät asetukset. Hyvin jaotellut bundlet tuovat uudelleenkäytettävyyttä ja niitä voi ottaa käyttöön jopa eri sovelluksissa tarpeen mukaan. Asetukset voivat vaihtua Symfonyn ajoympäristön (environment) mukaan. Sovelluksessa voi olla eri asetukset kehitykselle, testaukselle ja käytölle, jotka määräytyvät *front controllerissa* määritellyn ajoympäristön mukaan. Esimerkiksi kehitysympäristössä halutaan useimmiten kaikkien virheilmoitusten olevan päällä, mikä ei tuotantoympäristössä ole toivottavaa.

Symfonyä käytetään yleensä MVC-suunnittelumallin pohjalta (Model-View-Controller, Malli-Näkymä-Käsittelijä), vaikkei se mallikerroksen puuttumisesta johtuen olekaan täysiverinen MVC-sovelluskehys. MVC-mallissa tietomalli ja tiedon esitys eli näkymä on eriytetty riippuvuuksien vähentämiseksi toisistaan ja niiden välissä on käsittelijä, joka hoitaa niiden välisen koordinoinnin. CRUD-sovelluksessa käsittelijän tavoite on ottaa vastaan asiakkaan pyyntö ja käsitellä se käyttäen vaadittavia toimenpiteitä, joita ovat esimerkiksi tietokantatoiminnot ja etenkin varsinkin palvelukeskeisessä arkkitehtuurissa eri palveluiden käyttäminen. Lopuksi käsittelijä palauttaa asiakkaalle HTTP-vastauksen.

4.2 Malli

Symfony ei nykyään tarjoa mukanaan oletus-ORM:ia, mutta sen – kuten muutkin kirjastot – voi asentaa helposti PHP:n riippuvuuksienhallintajärjestelmä Composerilla [19]. Suurin syy ORM:n käytölle on, että se helpottaa domain-suunnittelumallin käyttöönottoa. Domain-malli on käsitteellinen suunnittelumalli, joka kuvaa sovelluksen erilaisia entiteettejä, niiden attribuutteja, rooleja ja suhteita [20]. Malli luo sovelluksen kohdealueelle käsitteistön ja esittää sen rakenteen. Näin entiteetit perustuvat liiketoiminnan sanelemaan malliin. Tällöin ei tarvitse erikseen implementoida CRUD-ominaisuuksia, vaan voidaan keskittyä liiketoimintalogiikkaan. [21.]

Propel ORM:issa malli määritellään yhteen skeematiedostoon, josta voidaan komentorivillä annettavien Symfony-komentojen avulla generoida mallin luokat ja tietokannan rakentamiseen tarvittavat SQL-lauseet. Propelissa on myös komento SQL:n ajamiseen kantaan. Mikäli malliin halutaan tehdä muutoksia, voidaan vain muuttaa skeematiedostoa joutumatta tekemään erillisiä SQL-komentoja. CRUD-toiminnot tehdään käyttämällä Propelin oliopohjaista kysely-API:a. Kyselykieli käännetään vastaamaan kulloinkin käytetyn tietokannan syntaksia. Näin kehittäjän ei tarvitse aina huolehtia tietyn tietokannan SQL-syntaksista tehdessään kyselyitä. Tarvittaessa erittäin optimoituja ad-hoc-kyselyitä voidaan ORM toki ohittaa. Seuraavassa on esimerkki yksinkertaisen tietokantaoperaation toteuttamisesta Propelilla:

```
$price = new Price();
$price->setAmount(10);

$product = ProductQuery::create()
          ->findPk(1);

$product->setPrice($price);
$product->save();
```

Koodiesimerkki 1. Haku, muutos ja tallennus Propelilla.

ORM:ia käyttäessä kyselyn tuloksena on useimmiten valmis kohdealuetta mallintava olio. ORM mallintaa oliot niin, että niiden suhteet ovat helposti nähtävissä esimerkiksi valmiiksi generoitujen funktioiden muodossa. Esimerkiksi tuoteoliota voidaan pyytää noutamaan siihen liittyvät hintaoliot joko tietokantakyselyn yhteydessä tai myöhemmin. Tässä yhteydessä on hyötyä myös tehokkaasta välimuistin käytöstä, jossa lapsioliot noudetaan kannasta vain, jos niitä ei ole jo noudettu muistiin. Propelissa on myös valmiit toiminnot yhtäaikaisille käyttäjille ja transaktioille.

Propel otetaan käyttöön konfiguroimalla tietokannan asetukset, jolloin sen mukana tulevia Symfony-komentoja voidaan käyttää. Edelle mainittujen komentojen lisäksi hyödyllinen on migraatio-komento, jolla tietokannan jo olemassa olevaa rakennetta voidaan muuttaa hävittämättä sen sisältämää dataa.

4.3 Näkymä

Näkymäkerros saa mallikerroksesta käsittelijän kautta tietoa, jonka esittämisestä käyttäjälle se huolehtii. Käsittelijä renderöi annotaation (syntaktista metadataa, joka vaikuttaa osaltaan ohjelman käytökseen) tai funktiokutsun avulla tietyn näkymän (template), jolle se tarjoaa muuttujia dynaamisen sisällön aikaansaamiseksi. Ilman sovelluskehystä tai muuta työkalua näkymän esittäminen tapahtuu sekoittamalla keskenään PHP-koodia ja HTML-koodia. Tällä tavalla toteutetuista näkymäpohjista tulee helposti runsassanaisia, monimutkaisia ja sekavia. Tämä johtuu esimerkiksi periytymisen ja suodatustoimintojen (filtering) puutteesta. [22.]

Twig

Symfony2-sovelluskehyksen oletusarvoisena näkymänmallinnustyökaluna toimii myöskin SensioLabsin kehittämä Twig. Se on hyvin tunnettu ja tuettu sekä tulee pysymään oletuksena Symfonyn näkymänmallinnuksessa myös seuraavassa pääversiossa. Twig käännetään optimoiduksi PHP-koodiksi, joten sen käytöstä ei synny merkittävää hidastumista. Kehittäjä pystyy helposti luomaan omia funktioita, suodattimia, tägejä ja operaattoreita. Twig on hyvin dokumentoitu, testattu ja sisältää tärkeitä ominaisuuksia kuten debug-funktioita ja automaattisen koodinvaihdosmerkkien lisäämisen. Se myös eriyttää ohjelmalogiikan näkymästä niin, että näkymän suunnittelijan ei tarvitse välttämättä osata lukea eikä kirjoittaa PHP-koodia, kunhan hän osaa yksinkertaisen Twig-syntaksin. [23.]

Twig-malli on tiedosto, joka voi generoida mitä tahansa web-sisältöä. Twig-käskyt erotetaan sivun varsinaisesta sisällöstä käyttämällä tulostus-, kontrolli- tai kommenttierottimia. Mallille asetetaan käsittelijässä muuttujia, jotka voivat olla esimerkiksi merkkijonoja, taulukoita tai olioita. Jälkimmäisessä tapauksessa mallista pääsee käsiksi myös olion attribuutteihin ja metodeihin pistenotaatiota käyttämällä. Symfony asettaa myös näkymään automaattisesti tarpeellisia globaaleja muuttujia kuten istunnon, pyynnön ja sovelluksen käyttäjän.

Jokainen Twigin malli käännetään automaattisesti PHP-luokaksi, joka renderöidään ajonaikaisesti. Debug-tilan ollessa päällä esimerkiksi ajettaessa koodia kehitysympäristössä Twig-mallit käännetään automaattisesti muutoksia tehdessä. Kehittäjän ei siis tarvitse manuaalisesti tyhjentää Twigin-välimuistia. [24.]

Web-sovelluksille on tavallista, että niissä on uudelleenkäytettäviä ja toistuvia elementtejä. Näitä voivat olla esimerkiksi valikot, ala- ja ylätunnisteet sekä erilaiset listaukset. Kullekin toistuvalle elementille on tavallista luoda oma näkymänsä, joka sisällytetään emonäkymään korvaamalla siinä määritelty lohko samannimisellä lohkolla lapsinäkymässä. Lapsinäkymälle täytyy vain kertoa, mitä hierarkiassa ylempänä olevaa näkymää se laajentaa. Mikäli lapsinäkymässä ei määritellä lohkolle korvaavaa sisältöä, käytetään emonäkymän lohkoa.

Koodiesimerkissä 1 on tiedostossa *base.html.twig* määritelty lohkosivun perusrakenne, joka sisältää body-lohkon. Se puolestaan sisältää lohkot valikoille, ylä- ja alatunnisteelle sekä varsinaiselle sivun sisällölle (content-lohko). Kun kyseistä näkymää laajennetaan

toisessa näkymässä, tulostuvat HTML-tägit sellaisenaan ja lohkojen sisältö siten, kun ne on muissa näkymissä määritelty.

```
{% block body %}
  <div id="wrapper">
  {% block mainmenu %}
  {% endblock mainmenu %}
    <div id="page-wrapper" class="gray-bg">
    {% block topnavigation %}
    {% endblock topnavigation %}
    {% block pageheading %}
    {% endblock pageheading %}
    <div class="wrapper wrapper-content">
      {% block content %}
      {% endblock content %}
    </div>
  {% block footer %}
  {% endblock footer %}
    </div>
  </div>
{% endblock body %}
```

Koodiesimerkki 2. Lohkojen käyttö.

Hierarkkisen mallin ohella on myös mahdollista laittaa pohjia sisäkkäin include-funktiolla, mikä on hyödyllinen ominaisuus tapauksessa, jossa samaa näkymää käytetään useammassa paikassa. Emonäkymää ei tarvitse myöskään kokonaan korvata, vaan siihen voidaan myös vain lisätä sisältöä parent-funktiolla. Tapauksessa, jossa sisällytettävä näkymä tarvitsee laajempaa sovelluslogiikkaa kuten tietokantakyselyitä, voidaan sille tehdä oma käsittelijänsä, jossa logiikka sijaitsee. Tällöin näkymässä, johon tämä toinen näkymä halutaan sisällyttää renderöidäänkin tämän uuden käsittelijän tuottama sisältö render- ja controller-funktioilla.

Sovellukseen määriteltyihin reitteihin viitataan esimerkiksi linkkejä luodessa niille annetuilla tunnisteilla, joita käytetään yhdessä path-funktion kanssa varsinaisen reitin osoitteen tulostamiseen HTML-sivulla. Tällöin voidaan vain muokata reittikonfiguraatiota, kun halutaan vaihtaa jonkin sivun URL. Näkymä huolehtii URL:in lopullisesta tulostamisesta. Reittimäärityksiin voidaan määritellä pakollisia tai vapaaehtoisia parametreja, jotka annetaan näin ollen myös path-funktiolle sitä kutsuttaessa.

Näkymäpohjia voi olla niin monessa tasossa kuin kehittäjä haluaa, mutta useimpiin sovelluksiin riittää yksinkertainen kahden tai kolmen tason malli. Sovellukseen luodaan näkymätiedosto nimeltä *base.html.twig*. Tämä sisältää HTML-tägit, sivun headerit, resurssit ja sovelluksen näkymän rungon. Seuraavaksi voidaan luoda sovelluksen jokaiselle osiolle oma base-näkymää laajentava näkymänsä. Nämä näkymät sisältävät kullekin osiolle ominaiset lohkot, otsikot ja muut tarpeelliset elementit. Viimeisenä luodaan kullekin sivulle oma yksilöllinen, kuvaavasti nimetty näkymänsä, joka laajentaa kyseisen osion emonäkymää. Kahden tason malli on useimmiten riittävä, mutta kolmen tason näkymähierarkia mahdollistaa sen, että uudelleenkäytettävien sovelluksen osien emonäkymän voi helposti korvata laajentamaan koko sovelluksen base-näkymää.

Resurssit

Resurssit (assets) tarkoittavat näkymäkerroksessa käytettäviä, sovelluksen tarvitsemia HTML-koodin ulkopuolisia tiedostoja, kuten kuvia sekä tyyli- ja JavaScript-tiedostoja. Symfonyssä resurssien osoitetta ei ole tarpeen kovakoodata, vaan niiden sisällyttämisessä voidaan käyttää Twigin asset-funktiota. Funktio tekee sovelluksesta siirrettävän. Mikäli sovellus esimerkiksi siirretään toiseen osoitepolkuun, asset-funktio huolehtii muutoksesta luomalla oikeat resurssipolut ilman, että resurssien polku olisi vaihdettava jokaisessa näkymässä erikseen. Funktio huolehtii myös tilanteesta, jossa käyttäjän on saatava päivitetyt eikä välimuistiin säilötyt resurssit. Tämä tapahtuu lisäämällä resurssin polkuun kyselymerkkijono (query string), jolloin selain noutaa resurssit uudelleen käyttäjän koneelle.

Tyyli- ja JavaScript-tiedostojen tuontia varten Symfonyn mukaan on laitettu Assetic-kirjasto. Sitä käytetään lisäämällä päänäkymämalliin lohkot *stylesheets* ja *javascripts*, joihin sijoitetaan koko sovelluksessa käytettävät resurssit. Muissa näkymämalleissa lisätään tarvittavat resurssit kyseisiin lohkoihin parent-funktiokutsun jälkeen. Assetic sisältää suodattimia, joiden avulla resursseja käsitellään ennen kuin ne tarjotaan käyttäjälle. Tämä mahdollistaa erottelun sovelluksen käyttämien resurssitiedostojen ja varsinaisten käyttäjälle tarjottujen resurssitiedostojen välillä. [25.]

Ilman Assetic-kirjastoa resurssit tarjoiltaisiin tavalliseen tapaan suoraan asset-funktion parametrina. Assetic-filttereiden avulla voidaan tiedostoja kuitenkin manipuloida etukäteen esimerkiksi minifoimalla ja yhdistelemällä CSS- ja JavaScript-tiedostot, ajamalla tie-

dostot esimerkiksi LESS-parserin läpi tai optimoimalla kuvatiedostoja. Minifointi on prosessi, jossa lähdekoodista poistetaan kaikki sen suorittamisen kannalta tarpeettomat merkit, kuten tulostumattomat merkit (välilyönnit, sarkainmerkit, rivinvaihdot, kappaleenvaihdot). Nämä merkit ovat tarpeellisia vain selkeyden vuoksi ihmisen lukiessa ja kirjoittaessa koodia. Minifointi on erityisen hyödyllistä tulkatuille kielille, joiden lähdekoodi lähetetään internetin välityksellä, sillä se pienentää lähetettävän tiedon määrää. Minifointia perustellaan usein myös tietoturvalla sen ollessa eräänlaista koodin obfuskointia, eli sen lukemisen hankaloittamista. Kyseessä on kuitenkin ”security through obscurity” -periaate eikä oikea kryptografia, sillä minifointi on helppo kumota eikä ole mitään syytä, miksei päättäväinen murtautuja sitä tekisi.

Assetic mahdollistaa sen, että tiedostot voidaan tallettaa eri paikkaan, kuin mistä ne tarjotaan käyttäjälle. Tarjottavat resurssit voidaan myös yhdistää eri paikoista. JavaScript-tiedostot sijoitetaan javascript-lohkon sisään käyttäen javascript-tägiä ja @AppBundle-syntaksia, joka määrittää tiedostojen oikeat polut.

```
{% javascripts
"@PointteriPWSUICommonsBundle/Resources/public/Inspinia/js/jquery-2.1.1.js"
"@PointteriPWSUICommonsBundle/Resources/public/Inspinia/js/bootstrap.min.js"
%}
<script type="text/javascript" src="{{ asset_url }}"></script>
{% endjavascripts %}
```

Koodiesimerkki 3. JavaScript-tiedostojen sisällytys.

CSS-tiedostot sisällytetään stylesheets-tägillä. Koska tiedosto tarjotaan eri paikasta kuin missä ne varsinaisesti säilötään, mahdolliset suhteelliset viittaukset kuvatiedostoihin CSS-tiedostoissa täytyy antaa myös Asseticin hoidettavaksi. Tämä onnistuu käyttämällä Asseticin cssrewrite-filtteriä, joka parsii CSS-tiedostot ja korjaa viittaukset. Tällöin CSS-tiedostoihin ei voida viitata JavaScript-esimerkin mukaisella @AppBundle-syntaksilla, vaan on käytettävä suoraa viittausta tiedostojen julkiseen osoitteeseen. CssImport-suodatin yhdistää CSS-tiedostot.

```
{%stylesheets
'bundles/pointteripwsuicommons/Inspinia/css/bootstrap.min.css'
filter="cssimport, cssrewrite"
%}
  <link rel="stylesheet" href="{{ asset_url }}" />
{% endstylesheets %}
```

Koodiesimerkki 4. CSS-tiedostojen sisällytys.

Resurssien yhdistäminen lisää myös front endin suoritusnopeutta vähentämällä HTTP-pyyntöjen määrää [25]. Yhdistämisen automatisointi mahdollistaa front end -lähdekoodin jakamisen eri tiedostoihin ilman, että suorituskyvystä tarvitsee tältä osin huolehtia, mikä tuo ohjelmistokehitykseen tarvittavaa modulaarisuutta ja uudelleenkäytettävyyttä. Yhdistäminen toimii toki myös muiden kuin sovelluskohtaisten resurssien kuten jQuery-kirjaston kanssa. Symfonyn kehitysympäristössä tiedostot tarjoillaan kuitenkin erikseen, jotta kehittäjän olisi helpompi käyttää selainten debug-ominaisuuksia.

Assetic ajaa suodattimet automaattisesti, mikä helpottaa kehitysprosessia. Kehitysympäristössä tämä voi entisestään lisätä sovelluksen testauksen hitautta selaimessa, sillä Assetic generoi resursseihin osoitteet, joissa ei fyysisesti sijaitse tiedostoja. Symfony käsittelee tiedostot, ajaa suodattimet ja tarjoaa tiedoston käyttäjälle, mikä mahdollistaa resursseihin tehtyjen muutoksen reaaliaikaisen näkymisen selaimessa. Tuotantoympäristössä halutaan käyttää nopeuden vuoksi valmiiksi generoituja tiedostoja, jotka luodaan Symfonyn assetic:dump -komennolla.

4.4 Käsittelijä

Sovelluskehyksen käyttämisen suurimpia etuja on, että pyyntö ja vastaus ovat helposti saatavilla erinäisine funktioineen ja kehittäjä voi keskittyä sovelluslogiikan kirjoittamiseen. Varsinainen pyynnön ja vastauksen käsittelevä koodi on MVC-mallissa jäsennetty käsittelijään. Käsittelijä (Controller) saa parametrikseen HTTP-pyynnön (Symfonyn Request-tyyppi) ja palauttaa sen käsittelyn lopuksi HTTP-vastauksen (Symfonyn Response-tyyppi). Vastaus on useimmiten HTML-sivu, ja se voi olla esimerkiksi myös esimerkiksi AJAX-pyynnön tapauksessa Symfonyn JsonResponse-tyyppinen.

Normaali käyttötapa sivujen esittämiselle on luoda kullekin oma käsittelijänsä, joka asettaa näkymän tarvitsemat muuttujat, renderöi sen ja lähettää HTTP-vastauksen heade-

reineen asiakkaalle. Käsittelijät järjestetään Symfonyssä eri bundlejen alle käsittelijäluokkiin, yleensä kohdealueen mukaan. Reittimääritykset on mahdollista tehdä annotaatioilla käsittelijän yhteyteen. Määrityksen yhteyteen voidaan myös määrittää reitin vaatimat tai vapaaehtoiset parametrit ja niiden oletusarvot, jotka käsittelijäfunktio saa automaattisesti parametreinaan.

Kun omat käsittelijäluokkansa laittaa laajentamaan Symfonyn Controller-luokkaa, pääsee helposti käsiksi sekä palvelusäilöön get-funktiolla että muihin hyödyllisiin funktioihin. Näillä funktioilla voi hoitaa esimerkiksi pyynnön uudelleenohjauksen tai edelleenvälityksen. Render-funktiolla voi renderöidä näkymän, mutta tämän voi tehdä myös annotaatiolla. Symfonyssä on myös Session-olio, joka säilöö attribuuttinsa oletuksena evästeisiin. Funktiolla *addFlash* voidaan helposti tallentaa yhden pyynnön ajaksi esimerkiksi virheilmoituksia.

4.5 Palvelut

Riippuvuusinjektio

Kaikissa paitsi yksinkertaisimmissa sovelluksissa sovellusarkkitehtuuri käsittää ison joukon eri olioita, mistä syntyy helposti ongelma sovelluksen arkkitehtuurin suunnittelussa: kuinka näitä olioita luodaan, kuinka ne organisoidaan ja kuinka niitä haetaan. Palvelu tarkoittaa yleiskäyttöistä, globaalia oliota, joka suorittaa rajattua tehtävää.

Palvelukeskeinen arkkitehtuuri (SOA, Service Oriented Architecture) on yleisesti käytetty suunnittelutapa, jolla sovelluksen eri toiminnallisuudet erotellaan toisistaan itsenäisiksi palveluiksi erillisten luokkien muodossa [26, s. 1]. Jokaisen palvelun on tarkoitus suoriutua yhdestä tehtäväalueesta, joten palvelun voi ottaa käyttöön aina tarvittaessa. Tehtävien erottelu mahdollistaa myös jokaisen palvelun helpon testaamisen ja konfiguroinnin. Ideaalitapauksessa palveluita käytetään rajapintojen kautta, ja eri palvelujen väliset sidokset on toteutettu löyhästi [26, s. 7], eli eri palveluilla on mahdollisimman vähän tietoa järjestelmän muiden komponenttien toteutustavasta.

Perinteisessä proserudaalisessa (aliohjelmiin jaotellussa) mallissa kehittäjän sovelluskohtainen koodi kutsuu käytettävien kirjastojen funktioita suorittaakseen geneerisiä tehtäviä. Symfony-sovelluskehys käyttää Inversion of Control -suunnittelumallia (IoC), jossa

nimensä mukaisesti ohjelman kontrolli tuleekin käytettävistä kirjastoista, jotka kutsuvat sovelluskohtaista koodia. IoC lisää sovelluksen modulaarisuutta ja tekee siitä helpommin laajennettavan.

IoC:hen liittyvä käsite on *riippuvuusinjektio,* jossa IoC-periaatetta toteutetaan määrittelemällä olioiden väliset sidokset abstraktioiden kautta ja määrittelemällä muuttujien varsinaiset tyypit vasta suoritusvaiheessa käyttäen dynaamista sidontaa. Symfony-sovelluskehys hoitaa sovelluksen suoritusjärjestyksen, jossa se määrittää kussakin kohtaa tarvittavan palvelun varsinaisen luokan ja injektoi sen sinne, missä sitä tarvitaan. Injektio tässä tapauksessa tarkoittaa riippuvuuden, eli palvelun, sijoittamista siitä riippuvaiseen olioon. Näin riippuvaisen olion ei itse tarvitse luoda tai etsiä oliota, josta se on riippuvainen. Riippuvuusinjektiosta on useita eri tyyppejä, joista Symfony käyttää kolmea. Ne ovat:

- rakentajainjektio (constructor injection)
- setter-injektio (setter injection)
- jäsenmuuttujainjektio (property injection). [27.]

Kenties yleisin näistä on rakentajainjektio, jossa rakentajafunktion otsikkoon lisätään riippuvuutta vastaava argumentti, jonka tyyppinä on käytännössä usein rajapinta. Symfonyssä injektoitavan parametrin konkreettinen luokka määritellään konfiguraatiotiedostossa, mutta funktion argumentille voidaan antaa tyyppivinkki (type hint), jotta saadaan aikaiseksi virheilmoituksia tapauksessa, jossa virheellinen riippuvuus on injektoitu.

Kullakin injektiotyypillä on omat hyötynsä ja haittansa. Jos luokka tarvitsee välttämättä jotakin palvelua, on se luonnollista injektoida samalla hetkellä kun luokka luodaan, eli rakentajafunktiossa. Tällä tavalla on varmaa, että luokalla on sen tarvitsemat palvelut silloin, kun sitä käytetään, sillä rakentajaa kutsutaan vain palvelun luontivaiheessa, joten riippuvuus ei voi myöskään muuttua palvelun palveluolion elinkaaren aikana.

Mikäli riippuvuudet ovat välttämättömien sijaan vapaaehtoisia, voidaan käyttää setter-funktioita niiden injektoimiseksi. Setteriä voi myös kutsua useamman kerran, mistä on hyötyä tilanteessa, jossa riippuvuus lisätään johonkin palvelun sisäiseen kokoelmaan. Tästä seuraa myös ongelma suhteessa rakentajainjektioon, ellei setter-funktiossa varmisteta, onko sitä jo kutsuttu, voi riippuvuus korvautua toisella palveluolion elinkaaren aikana, eikä kehittäjällä ole tästä tällöin varmuutta. Varmaa ei ole myöskään se, että

setteriä ylipäänsä kutsutaan, ja riippuvuuden injektoinnista tuleekin setter-injektion tapauksessa huolehtia tarkemmin.

Kolmas injektiomahdollisuus on joidenkin luokan jäsenmuuttujien asettaminen julkisiksi, jolloin ne voidaan asettaa suoraan luokan ulkopuolelta. Tämä injektiotyyppi on hyvin samanlainen kuin setter-injektio, tosin se tuo mukanaan uusia ongelmia. Riippuvuuden injektoinnista ei ole varmuutta, ja riippuvuus voi korvautua toisella milloin tahansa olion elinkaaren aikana. Myöskään tyyppivihjeitä ei voida käyttää. Tämä injektiotyyli tulee kyseeseen lähinnä käytettäessä Symfonyn tyylistä sovelluskehystä, joka hoitaa injektoinnin erillisellä komponentilla (Symfonyn tapauksessa *Service Container*) konfiguraation avulla. Tällöinkin jäsenmuuttujainjektio tulee kyseeseen lähinnä käytettäessä kolmannen osapuolen kirjastoja, jotka käyttävät julkisia jäsenmuuttujia riippuvuuksiensa säilömiseen.

Service Container

Symfonyssä eri olioiden alustus, organisointi ja noutaminen on toteutettu palvelusäiliön (Service Container) avulla. Se on keskitetty ja yhdenmukainen tapa Symfonyllä tehdyssä sovelluksessa toteuttaa olioiden luominen. Sen hyötyinä ovat nopeus ja se, että kehittäjä ajattelee sitä käyttäessään automaattisesti palvelukeskeisesti. Symfonyn ydin käyttää säiliötä, ja sen voidaan sanoa olevan sovelluskehyksen tärkeimpiä osia, jonka käyttö lisää ohjelmakoodin uudelleenkäytettävyyttä, testattavuutta ja sidosten löyhyyttä. [28.]

Palvelusäiliö on toiselta nimeltään *Dependency Injection Container,* mikä kertoo sen toteuttavan edellä kuvailtua DI-suunnittelumallin ajattelutapaa Symfonyn viitekehyksessä. Ilman säiliötä oliot tulisi luoda new-syntaksilla manuaalisesti, mutta olion vaatiessa monimutkaisempaa alustusta ja konfigurointia luontinsa yhteydessä tulisi sen käyttämisestä useassa paikassa sovellusta itseään toistavaa. Muutostarpeen syntyessä olisi tällöin etsittävä jokainen paikka koodissa, missä kyseinen olio luodaan. Säiliö hoitaa olion luomisen ja lataa siihen tarvittavat tiedot yhdestä paikasta konfiguraatiotiedostossa.

Symfony luo säiliön käyttäen app/config-hakemistossa sijaitsevaa ajoympäristökohtaista konfiguraatiotiedostoa *config_ympäristö.yml.* Kun palvelu on konfiguroitu, on se helposti saatavilla säiliöltä käsittelijöissä käyttämällä get-funktiota. Konfiguraatiotiedostot ovat yleensä YAML-, XML- tai PHP-muotoisia tiedostoja, mutta Symfony mahdollistaa asetusten lataamisen myös tietokannasta tai jonkin web-rajapinnan yli.

Kun säiliöltä pyydetään palvelua, se rakentaa sen konfiguraation perusteella ja palauttaa palveluluokkaa ilmentävän olion. Tämä tapa luoda palveluolio vasta siinä vaiheessa, kun sitä pyydetään palvelusäiliöltä, säästää muistia ja lisää sovelluksen suoritusnopeutta. Säiliön suurimpia etuja onkin, että palveluita ei luoda ennen kuin niitä tarvitaan. Tällöin luokkia voidaan myös määritellä palveluiksi niin paljon kuin halutaan luodessa sovellusta palvelukeskeistä arkkitehtuuria käyttäen.

Palveluiden konfiguraatiossa voidaan määritellä myös parametreja, joiden avulla niihin voidaan syöttää tietoa. Parametrit löytyvät näin yhdestä paikasta ja niitä voidaan käyttää useassa palvelussa. Parametreillä voidaan näin helposti ainoastaan konfiguraatiota muuttamalla vaihtaa palvelun käyttäytymistä sovelluslogiikan tarpeiden mukaan.

Symfony käyttää yhtä kehittäjän määriteltävissä olevaa konfiguraatiotiedostoa palvelusäiliön luomiseen. Oletusarvoisesti tiedosto on *app/config/config.yml.* Muut asetustiedostot voivat olla sijoiteltuna erinäisten bundlejen alle, joten ne on tuotava tässä pääkonfiguraatiossa yleensä imports-direktiiviä käyttäen. Palveluun liittyvät konfiguraatiotiedostot on luonnollista sijoittaa sen bundlen alle, missä itse palvelukin sijaitsee. Käytäntönä on sijoittaa nämä tiedostot bundleen sijaintiin *Resources/config/services.xml.* Tiedoston tyyppi on kehittäjän preferensseistä riippuvainen. Tuotaessa bundlen konfiguraatiotiedosto jossakin ylempänä konfiguraatiohierarkiassa viitataan tuotavaan tiedostoon sen absoluuttisella sijainnilla, mutta @Bundle-syntaksilla voidaan viitata bundlen sijaintiin niin, että se voi muuttua myöhemmin kehityksen aikana ilman, että konfiguraatioon on sen takia tehtävä muutoksia.

Palvelukeskeisessä arkkitehtuurissa on väistämätöntä ja hyvä käytäntö, että palvelut riippuvat usein toisista palveluista. Esimerkiksi sähköpostin lähetyksen hoitava palvelu voidaan injektoida useamman muun palvelun sisään, jolloin postitusominaisuutta ei tarvitse toistaa useammassa paikassa, vaan saavutetaan toivottu tehtävien eriyttäminen ja modulaarisuus. Jos injektoitavasta palvelusta riippuvainen palvelu luotaisiin normaalilla tavalla, olisi sen argumenttien ja esimerkiksi nimen muuttaminen myöhemmin työläämpää. Palveluiden määrittelytiedostossa ne saavat kuitenkin yksilöllisen tunnisteen, jolla niihin voidaan ympäri sovellusta viitata. Määrittely huolehtii myös, että palveluun injektoidaan kaikki määritellyt palvelut, ellei niitä ole määritelty vapaaehtoisiksi. Vapaaehtoisten riippuvuuksien injektoiminen on paras tehdä setter-injektiolla. Konfiguraatiossa tämä tarkoittaa tavallisen argument-elementin ympäröimistä call-tageillä.

```
<service id="listener.login" class="…\MainUILoginListener" >
       <call method="setLogger">
         <argument type="service" id="logger" />
       </call>
</service>
```

Koodiesimerkki 5. Setter-injektion käyttö.

Symfony käyttää luonnollisesti myös omiin palveluihinsa palvelusäiliötä, samoin kuin muut vapaasti jaossa olevat bundlet. Koko palvelusäiliö injektoidaan automaattisesti käsittelijäluokkiin, joten käsittelijöiden määritteleminen palveluiksi ei ole aina tarpeellista. Symfonyn palvelut vastaavat esimerkiksi sähköpostien ja näkymien luonnista sekä HTTP-pyyntöjen käsittelystä. Näitä palveluita on luonnollisesti hyvä injektoida kehittäjän omiin palveluihin halutun toiminnallisuuden saavuttamiseksi.

Palveluiden määrittelytiedostossa niitä voidaan merkitä tägillä ilmaisemaan jotakin tiettyä käyttöaluetta. Tällainen tägi on esimerkiksi *twig.extension*, jota *TwigBundle* käyttää. Bundle löytää kaikki tällä tavalla merkityt palvelut ja rekisteröi ne laajennuksinaan.

4.6 Lomakkeet

Lomakkeiden käsittely on web-pohjaiselle CRUD-sovellukselle hyvin olennainen piirre. Koska käsittelijät halutaan pitää mahdollisimman ohuina, on hyvin rakennetussa sovelluksessa oltava erillinen lomakkeiden käsittely. Sillä voidaan saavuttaa helppo kenttäkohtainen validointi ja käyttäjälle virhetilanteesta viestiminen. Symfony tarjoaa Form-komponentin tähän tarkoitukseen. Komponenttia käyttämällä saavutetaan lomakkeiden laajennettavuutta ja uudelleenkäytettävyyttä sekä niiden käsittelyn automatisointi.

Lomakeluokka toteuttaa FormBuilderInterface-rajapinnan, jonka buildForm-funktiossa lomakkeen rakentajaan lisätään kenttiä. Symfonyssä on valmiita kenttätyyppejä optioineen yleisimpiin käyttötarkoituksiin, mutta kehittäjä voi luoda omia kenttätyyppejään, jotka ovat itsessään lomakkeita. Näitä lomakkeita voi tällöin sijoitella sisäkkäin. Kun lomakeluokka on luotu, lomaketta ei tarvitse rakentaa käsittelijässä, vaan sen createForm-funktiota voidaan käyttää antaen sille parametrina instanssi lomakeluokasta tai sen tunniste. Koodiesimerkissä 6 luodaan yksinkertaistettu käyttäjälomake omassa luokassaan text-tyyppisellä käyttäjänimikentällä ja email-tyyppisellä sähköpostikentällä. [29.]

```
public function buildForm(FormBuilderInterface $builder, array
$options)
  {

    $builder
    ->add('username', 'text', array(
        'label' => 'Username,
        'required' => true
    ))
    ->add('email', 'email', array(
        'label' => 'Email',
        'required' => true),
    ));
}
```

Koodiesimerkki 6. Lomakkeen luonti.

Lomake renderöidään eli näytetään sivulla antamalla näkymälle muuttujana lomakkeen näkymä kutsumalla sen createView-funktiota. Koodiesimerkissä 7 näytetään tuotelomakkeen käyttö käsittelijässä. Tuoteolio *$item* on noudettu aiemmin tietokannasta tai luotu uutena.

```
$item_form = $this->createForm('item_type', $item);
if($request->getMethod() === Request::METHOD_POST){
  $item_form->handleRequest($request);

  if($item_form->isSubmitted()){
    if ($item_form->isValid()) {
      $item->save();
    }
  }
}
```

Koodiesimerkki 7. Lomakkeen käyttö käsittelijässä.

Lomakkeisiin liittyviä Twig-funktioita käyttämällä voidaan näyttää jokainen kenttä niin tarkasti kuin halutaan kontrolloimalla sen virheilmoituksia, luokka-attribuutteja ja niin edelleen. Usein kuitenkin riittää kolmen eri funktion käyttäminen:

- form_start
- form_widget
- form_end.

Näistä *form_start* luo lomakkeen aloitustägin ja *form_widget* itse elementin otsikkoineen ja virheilmoituksineen. *Form_end* luo lomakkeen lopetustägin ja mahdolliset tähän asti renderöimättömät kentät. Useimmiten kehittäjä renderöi erikseen suurimman osan kentistä, koska haluaa kontrolloida lomakkeen ulkonäköä tarkemmin kuin mitä funktioiden automatiikka sallii. Tällöin *form_endin* hyötynä on tulostaa mahdolliset piilotetut kentät mukaan lukien Symfonyn automaattisen CSRF-tokenin (Cross-site Request Forgery).

Lomakkeet liittyvät yleensä läheisesti sovelluksen tietomalliin. Lomake voidaankin sitoa johonkin mallin luokkaan, jolloin lomakkeelta voidaan sen lähetyksen jälkeen pyytää muuttunut malliolio suoraan tallennettavaksi. Mikäli lomakkeita ei ole sisäkkäin, tunnistaa Symfony automaattisesti lomakkeelle annetun malliolion perusteella, mistä luokasta sen data on peräisin. Mikäli lomake sisältää jonkin toisen lomaketyypin, johon on sidottu jokin malliluokka, tallennetaan sekin automaattisesti.

Kun lomakkeen hoitavassa käsittelijässä kutsutaan handleRequest-funktiota, tutkii se, onko käyttäjä lähettänyt lomakkeen ja sitoo tällöin lomakkeen datan sille aiemmin annettuun malliolioon. isValid-funktiota kutsumalla voidaan lomake validoida esimerkiksi lomakkeeseen asetettujen kenttäkohtaisten tai malliolion attribuutteihin asetettujen rajoitteiden (constraint) pohjalta. Jos lomake on lähetetty ja validi, voidaan malliolio sovelluslogiikan salliessa tallentaa muutoksineen.

4.7 Autentikointi ja autorisointi

Autentikointi tarkoittaa mekanismia, jolla sovelluksen käyttäjät tunnistetaan turvallisesti. Sovelluskehystä käyttämällä tällainen tarpeellinen, mutta toteutukseltaan aikaavievä perusominaisuus saadaan pienellä vaivalla aikaiseksi. Symfonissa autentikoinnin määrittelyt sijoitetaan asetustiedostoon *security.yml,* jossa Symfonin palomuureiksi (firewalls) kutsumilla määrityksillä sovelluskehykselle kerrotaan autentikoinnin tyyppi, esimerkiksi HTTP-basic tai sisäänkirjautumislomake. [30.]

Tiedostossa määritellään myös access_control-parametrilla, mitkä osat sivustosta ovat suojattuja eli autentikoinnin piirissä. Lomakkeen tapauksessa asetuksissa määritellään myös, mikä lomakkeen reitti on. Reitti pitää määritellä myös sisäänkirjautumisen tarkistukselle, mutta Symfony hoitaa käyttäjien latausmäärittelyjen perusteella varsinaisen au-

tentikoinnin automaattisesti. Kun käyttäjä on autentikoitunut onnistuneesti, hänet ohjataan määrätylle reitille. Käyttäjien lataus hoidetaan rakentamalla UserProviderInterface-rajapinnan toteuttava palvelu, joka hoitaa käyttäjien noutamisen tietokannasta. Lataajapalvelu nimetään asetuksissa, samoin kuin käyttäjien salasanojen koodaustapa.

Autorisointi tarkoittaa mekanismia, jolla päätetään, mihin resursseihin käyttäjällä on sovelluksessa pääsy. Käyttäjille voidaan määritellä tiettyjä loogisia rooleja, jotka kertovat autorisoinnin tasosta. Kun tietokantaan tallennetaan rooleja ROLE-etuliitteellä, Symfony osaa käsitellä ne. Asetuksiin määritellään esimerkiksi URLeja, joiden yhteyteen laitetaan roolit, jotka voivat kyseistä resurssia käyttää. Symfony käy nämä määritykset järjestyksessä läpi ja katsoo, mikä vastaa senhetkistä pyyntöä ja päättää näin autorisoinnin tason käyttäjälle. Autorisointia voidaan tehdä myös esimerkiksi käsittelijöiden tasolla denyAccessUnlessGranted-funktiolla tai annotaatioilla, tai näkymien tasolla Twigin isGranted-funktiolla.

4.8 Testaus

Sovelluskehityksessä on olennaista sovelluksen kasvaessa pystyä hallitsemaan, etteivät uudet lisäykset lähdekoodiin aiheuta virheitä jo olemassa olevaan toiminnallisuuteen. Toiminnalliset testit ja yksikkötestit auttavat luotettavampien sovellusten luomisessa. Symfonyyn on integroitu yksikkötestaus-sovelluskehys PHPUnit, jolla kirjoitetut testit voidaan ajaa yhdellä komentorivikomennolla.

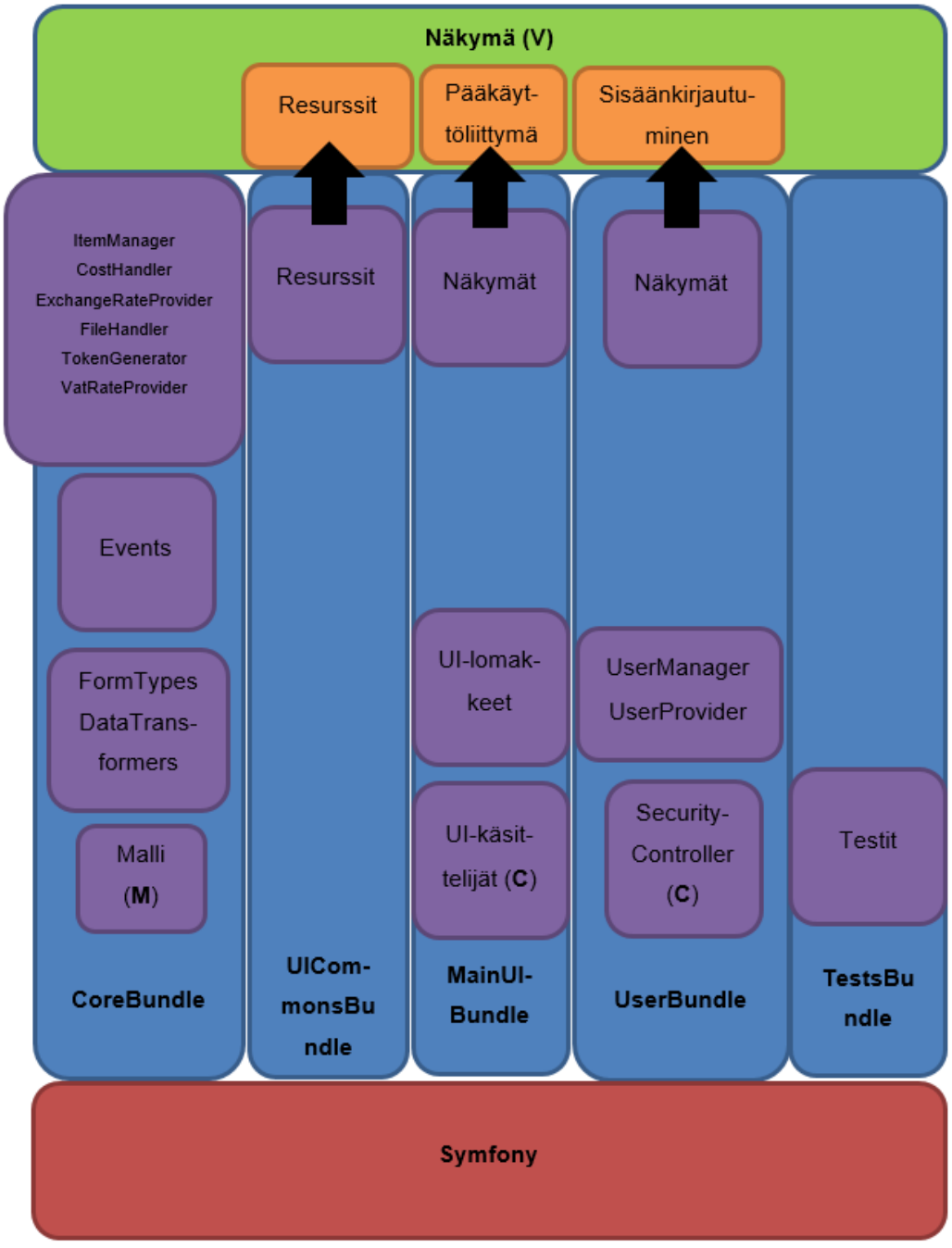
Yksikkötesti testaa nimensä mukaisesti jotakin pientä lähdekoodin yksikköä, kuten yksittäistä funktiota. Sen tarkoituksena on eritellä nämä yksiköt ja testata niiden oikea toimivuus. Testi määrittelee myös raamit, joiden sisällä ohjelman testattavan palasen katsotaan toimivan oikein. Yksikkötestit toteutetaan erillisiin testiluokkiin ja niiden metodeihin, jotka käyttävät PHPUnitin assert-funktioita toteamaan vertailuilla testin onnistumisen.

Toiminnalliset eli funktionaaliset testit testaavat sovelluksen eri palasten ja arkkitehtuurillisten kerrosten integraatiota, eli sovelluksen toiminnallisuutta. Niiden tarkoituksena voidaan sanoa olevan löytää eroavaisuuksia sovelluksen toteutuksen ja sen toiminnallisen määrittelyn välillä. Testit toteutetaan PHPUnitin kannalta samalla lailla kuin yksikkötestit, mutta niissä käydään läpi koko sovelluksen toiminnallinen ketju HTTP-pyynnöstä

vastaukseen saakka. Tähän tarkoitukseen on hyvä käyttää Symfonyn DomCrawler-komponenttia, jolla voi tutkia ja manipuloida HTML- ja XML-dokumenttien rakennetta. Rakenteellista dokumenttia voi tutkia käymällä sen solmuja läpi esimerkiksi CSS-valitsimien avulla. Crawler osaa myös esimerkiksi seurata linkkejä ja lomakkeita.

5 Valmiin järjestelmän kuvaus

Sovelluksessa käytettiin Symfonylle ominaista bundlerakennetta. Sovellukseen luotiin *CoreBundle*, joka sisältää sovelluksen yleiskäyttöiset luokat. Nämä luokat on jaoteltu kohdealueittain ja kansioihin. Kuvassa 3 on esitetty korkealla tasolla järjestelmän arkkitehtuuri.



Kuva 3. Toteutetun järjestelmän arkkitehtuuri.

5.1 Command

Symfony mahdollistaa komentoriviltä ajettavien komentojen luomisen usein toistuvia tehtäviä varten ajettavaksi manuaalisesti tai esimerkiksi cronin tapaisten ajastuspalveluiden kautta [31]. Mukana on valmiita komentoja liittyen sovelluskehykseen itseensä, kuten bundlen luontikomento, ja esimerkiksi Propelin luokkien generointikomento. Komennot luodaan laajentamalla Symfonyn ContainerAwareCommand-luokkaa. Järjestelmään tehtiin ajastettuja komentoja noutamaan valuuttakursseja ja tallentamaan ne tietokantaan sekä päivittämään uusien valuuttakurssien pohjalta tuotteiden hinnat pohjavaluutassa. Näin valuuttamuunnosta ei tarvitse tehdä jokainen kerta valuuttoja tarkastellessa erikseen. Muut luodut komennot liittyvät testifikstuurien lataamiseen tietokantaan.

5.2 EntityManager

ItemManager-luokka sisältää tuotteiden käsittelyyn liittyviä toimintoja. Toimintojen sijoittaminen yhteen manager-luokkaan pitää tuotteiden käsittelyn toistuvat toiminnot hallitusti yhdessä paikassa ja tuo näin paremman ylläpidettävyyden. Propel tarjoaa malliluokissa funktiot *preSave* ja *postSave*, joita voitaisiin periaatteessa käyttää tähän tarkoitukseen. Ongelma malliluokissa on kuitenkin palveluihin käsiksi pääseminen. Erillisen manager-luokan käytöstä on muitakin etuja, esimerkiksi jos sovellukseen tehdään myöhemmin muita käyttöliittymiä tai sen datalle rajapintoja, on hyvä eriyttää käsittelijäfunktioihin vain käyttöliittymään liittyvät asiat ja säilöä liiketoimintalogiikka jossakin muualla keskitetysti.

ItemManager-luokka on määritelty palveluksi, ja se käyttää Symfonyn TokenStorage-palvelua ja järjestelmän *CostHandleria*. Luokka sisältää erityyppisten hintojen alustusfunktioita ja saveItem-funktion, johon voidaan keskittää jokaisen tuotteen tallennuksen yhteydessä tehtävät toiminnot, kuten timestamp-tyyppisten kenttien päivittäminen. saveItem-funktio kutsuu myös handleCosts-funktiota, joka hoitaa tuotteeseen liittyvien eri tyyppisten hintojen ja kustannusten hallinnan CostHandler-palvelun avulla. Funktiossa on toteutettu esimerkiksi tuotteen tilasta riippuvainen hintojen lukitseminen ja annetuista hinnoista riippuvainen tuotteen statuksen vaihtaminen sekä lukittujen hintojen muokkauksen estäminen.

UserManager-luokka on tarkoitettu nimensä mukaisesti sovelluksen käyttäjien hallintaan. Se käyttää Symfonystä seuraavia palveluita:

- TokenGeneratoria
- EventDispatcheria
- UserPasswordEncoderia
- TokenStoragea
- AuthorizationCheckeriä

TokenGeneratorilla luodaan uudelle käyttäjälle käyttäjän tallennusfunktion käytön yhteydessä sähköposti-token, jota käyttäjä voi käyttää asettaakseen itselleen salasanan tai vaihtaakseen sen myöhemmin. Tokenin asetusfunktio käyttää *EventDispatcheriä* lähettämään SEND_SET_PASSWORD_EMAIL-tapahtuman, jolla on oma tapahtumakuuntelijansa, mikä toteuttaa sähköpostin lähetyksen. *UserPasswordEncoderia* käytetään salaamaan käyttäjän salasana ennen sen tallennusta Symfonyn asetustiedostoissa määritellyllä salausmetodilla.

TokenStoragelta voi pyytää sovelluksen senhetkisen käyttäjän, jolloin hänen liiketoimintalogiikan mukainen roolinsa voidaan tarkistaa. *AuthorizationChecker* puolestaan tarkistaa sovelluslogiikan mukaisen roolin, ja näitä käytetään yhdessä *UserManagerin* funktioissa tarkistamaan liiketoimintalogiikan mukaisia oikeuksia, kuten voiko käyttäjä muokata tuotteita ja mitä hän näkee käyttöliittymässä. Järjestelmän roolihierarkiassa päädyttiin ratkaisuun, jossa hierarkian ensimmäisellä tasolla ovat käyttäjien loogiset ryhmät (esimerkiksi ostaja, ostopäällikkö) ja toisella tasolla oikeudet (esimerkiksi oikeus lisätä tuotteita ja muokata yrityksiä). Näin saadaan liiketoimintalogiikka ja sovelluslogiikka eroteltua roolien osalta ja yksittäisen käyttäjän oikeuksia kontrolloitua tarkemmin, kuin mitä pelkät liiketoimintaroolit antavat myöten.

5.3 Event & EventListener

Symfonyn EventDispatcher-komponentti käyttää välittäjä-suunnittelumallia (mediator) helpottaakseen Symfonyllä rakennetun projektin laajentamista helposti sitomatta kehittäjän ainoastaan luokkaperinnän käyttämiseen [32]. Sovellusten ylläpidettävyys ja ymmärrettävyys heikkenevät luokkien määrän kasvamisesta seuraavan kompleksisuuden lisääntyessä. Muutokset jossakin luokassa voivat aiheuttaa muutoksia monessa muussa

luokassa. Välittäjä-mallilla saadaan aikaan se, että oliot eivät kommunikoi suoraan keskenään, vaan viestit menevät kapseloidun välittäjäolion kautta. Oliot eivät näin viittaa toisiinsa, vaan saadaan aikaan löyhempi sidos.

EventDispatcher toimii siten, että jokin kuuntelijaluokka kertoo sille tapahtumat, joita haluaa kuunnella. Kun tapahtuman lähetystä pyydetään, *dispatcher* ilmoittaa siitä kaikille kuuntelijoille apunaan Event-olio, joka voi sisältää tapahtumaan liittyvää tietoa, jotka ovat kuuntelijoille tarpeellisia. Järjestelmässä on käytössä *SendPasswordTokenMailEvent*, jota *UserManager* käyttää. Tapahtuma kuljettaa mukanaan tapahtumakuuntelijalle User-olion, jolle se muodostaa ja lähettää sähköpostin. Tämän jälkeen kuuntelija asettaa tapahtumaan tiedon postin lähetyksen onnistumisesta, joka voidaan tarkistaa *UserManagerissa*.

5.4 Form

Järjestelmään luotiin erilaisia *DataTransformereita*, joiden avulla saadaan automatisoitua käyttäjän lomakkeille syöttämän datan muunnos haluttuun muotoon, esimerkiksi id-arvosta olioksi. Näitä muuntajia käyttämällä saadaan muunnos aikaiseksi automaattisesti vain lisäämällä se lomakeluokkaan, joten muunnosta ei tarvitse tehdä jokaisessa käsittelijässä erikseen. Muuntajat toimivat myös toiseen suuntaan, eli käyttäjälle voidaan esimerkiksi näyttää hinnasta haluttu määrä desimaaleja, ja hintaa tallennettaessa vaihtaa desimaalien määrä taas tietokannan sarakkeen määrittelyä vastaavaksi. Toinen tärkeä muunnos järjestelmässä on Propelin array-tyypin (merkkijono) ja php:n array-tyypin (taulukko) välillä.

Sovellukseen luotiin myös erilaisia uudelleenkäytettäviä lomaketyyppejä. Käyttäjälomakkeet laajentavat BaseUserType-luokkaa, joka sisältää kaikissa käyttäjälomakkeissa käytettävät kentät, kuten etunimen ja sukunimen. Aliluokissa voidaan lisätä kenttiä tai kenttiin erilaisia vaatimuksia. UserAdminType-luokassa lisätään pääkäyttäjälle esimerkiksi käyttäjän roolien muokkausmahdollisuus ja merkataan nimikentät pakollisiksi.

Muut käyttäjälomakkeet ovat käyttäjien hakulomake ja käyttäjän tietojen muokkauslomake. CostType-luokkaan on määritelty hinta-, valuutta- ja arvonlisäverokentät. *DateRangeTypeä* käytetään jonkin aikajakson perusteella tehtävään hakuun ja sille voi op-

tiona määrittää, käytetäänkö datepicker- (päivämäärät) vai dropdown-tyyppistä (vuosiluvut) listaa. RangeType-luokkaa käytetään numerovälin syöttämiseen. *MessageType* on sidottu tietomallin Message-luokkaan ja sitä käytetään tuotekohtaisten viestien lähetyksessä. *ItemPropertyType* on sidottu ItemProperty-luokkaan ja sitä käytetään tuotelomakkeen yhteydessä lisäkenttien määrittämiseen.

5.5 Model

Tietomallissa keskeisin luokka on *Item*, joka kuvaa järjestelmään syötettäviä tuotteita. Jotta sovelluksesta saatiin mahdollisimman yleiskäyttöinen erilaisille tuotteille, päätettiin kohdealueen spesifisemmät tuotekohtaiset ominaisuudet eriyttää omaan tauluunsa. Niiden pitäminen Item-taulussa tekisi tietokantahauista yksinkertaisempia ja Propel generoisi getter-metodit automaattisesti. Malli olisi myös yksinkertainen, mutta varsin sidottu kohdealueeseensa ja tarkoittaisi, että Item-olioista tulee helposti isokokoisia.

Dynaaminen ratkaisu olisi sijoittaa tuotteen ominaisuuksien nimet yhteen tauluun ja niiden arvot toiseen tauluun. Tämä ratkaisu mahdollistaisi käyttäjän itse asettamat ominaisuudet, mutta tekisi mallista monimutkaisen sekä hankaloittaisi tietokantahakuja ja monimutkaistaisi sovelluslogiikkaa. Ominaisuudet päädyttiin näin ollen sijoittamaan ItemProperty-tauluun arvoineen. Ratkaisu on riittävän yksinkertainen, mahdollistaa helpot haut ja sarakkeiden lisäämisen sekä pitää Item-oliot keveinä.

Tuotteeseen liittyvät sen malli, merkki, myyjäliike, viestit, tiedostot ja hinnat sekä niiden vaihtokurssit. Nämä on sijoitettu omiin tauluihinsa. Kussakin taulussa käytetään Propelin timestampable-behavioria, joka luo automaattisesti luotu- ja päivitetty-kentät. Muut taulut säilövät käyttäjien tiedot, oikeudet, käyttäjäryhmät ja asetukset.

5.6 Utils

Messenger-luokka on kääre Symfonyn sähköpostin lähetystoiminoille ja tarjoaa yleiskäyttöisen sendEmail-funktion. VatRateProviderInterface-rajapinnan toteuttava FixedVatRateProvider-luokka tarjoaa staattisen listan maakohtaisista arvonlisäveroista. Rajapinta mahdollistaa veroarvojen lähteen muuttamisen myöhemmin. *ExchangeRateProvider* on toteutettu samalla tavoin ja hakee tietokantaan kerran päivässä päivitetyt valuuttakurssit. *TokenGenerator* luo salasanan asetussähköpostissa tarvittavan tokenin.

CostHandler-luokka hallitsee kuhunkin tuotteeseen liittyviä hintoja. Sillä voidaan asettaa hinnalle oletusarvonlisävero ja oletusvaluutta joko käyttäjän itse määrittelemistä asetuksista tai sovelluskohtaisista oletuksista. handleNew-funktiota kutsutaan ItemManager-luokasta uusien hintojen kohdalla, jolloin sille asetetaan tarvittavat lomakkeen ulkopuoliset perusarvot. handleOld-funktio tarkistaa hinnan tyyppikentän ja päättää, pitääkö siitä pitää yllä historiatietoja. Jos näin on, se luo hinnasta kopion.

5.7 Tiedostot

FileHandler-luokka hoitaa sovelluksessa tiedostojen tallentamisen keskitetysti. File-luokka sisältää tietokantaan talletettavat tiedostojen metatiedot, kuten tyypin, sijainnin ja koon. Se myös pitää tietokantamallin ulkopuolisesssa, Symfonyn UploadedFile-tyyppisessä muuttujassa varsinaisen ladatun tiedoston, josta *FileHandler* ottaa tarvittavat tiedot tallettaakseen ne File-olioon ja sitten tietokantaan.

File-luokka käyttää Propelin preSave- (ajetaan ennen olion tallentamista tietokantaan) ja preDelete-funktioita (ajetaan ennen olion poistamista tietokannasta). Näissä funktioissa hoidetaan tiedoston varsinainen siirtäminen väliaikaisesta sijainnista lopulliseen tallennuspaikkaan ja sen poistaminen tiedostojärjestelmästä. Etuna tästä on se, että mikäli tällainen funktio palauttaa jotakin mutta kuin boolean-arvon tosi, keskeytyy tiedoston tallentaminen tietokantaan. Näin tiedoston käsittely tietokannassa ja tiedostojärjestelmässä ovat yksi atominen operaatio. Tiedostot tarjoillaan kontrollerin kautta FileController-luokassa. Kuvia ja HTTP-vastausta voidaan näin haluttaessa esikäsitellä.

5.8 Tuotteet

MainUIBundle pitää sisällään sovelluksen pääkäyttöliittymän kontrollerit, näkymäpohjat ja lomakkeet. Bundle on nimetty näin siltä varalta, että sovellus tarvitsee toisenkin käyttöliittymän. Tämä on kuitenkin responsiivisesti toteutetun front endin vuoksi epätodennäköistä. Käsittelijät on jaoteltu yksittäisen luokan paisumisen estämiseksi kohdealueen mukaan. Sovelluksen kohdealueita kuvaavat kontrolleriluokat Bundlessa ovat:

- FileController tiedostojen tarjoamiseen
- FrontpageController etusivulle
- ItemController tuotesivulle
- MessageController viestitystoimininnoille
- ProfileController käyttäjän asetusten hallintaan
- SettingsController sovelluksen yleisille asetuksille.

Näistä tärkein on *ItemController*, joka ohjaa tuotesivua. Tuotesivulla on useita lomakkeita liittyen esimerkiksi tuotteen dokumenttitiedostoihin, mutta tärkein näistä on ItemType-luokka. Se käyttää aiemmin esiteltyä *CostTypeä* hintojen syöttämiseen ja *ItemProperty-Typeä* ItemProperty-taulussa nimettyjen ominaisuuksien syöttämiseen omien kenttiensä lisäksi. Luokkaan on tehty erillinen validateForm-funktio, jota kutsutaan aina, kun lomake lähetetään. Tämä on toteutettu korvaamalla yliluokka *AbstractTypen* setDefaultOptions-funktio ja asettamalla siellä lomakkeen käsittelyn yhteyteen rajoite (constraint) nimeltä *callback*, jossa määritellään kutsuttavan funktion nimi.

```
public function validateForm($data, ExecutionContextInterface
$context){
    if($context->getViolations()->count() > 0){
      $context->buildViolation('There were errors in the form');
    }
}

public function setDefaultOptions(OptionsResolverInterface
$resolver){
    $resolver->setDefaults(
        array('constraints'=>array(
new
Callback(array('methods'=>array(array($this,'validateForm')))),
            ),
        )
    );
}
```

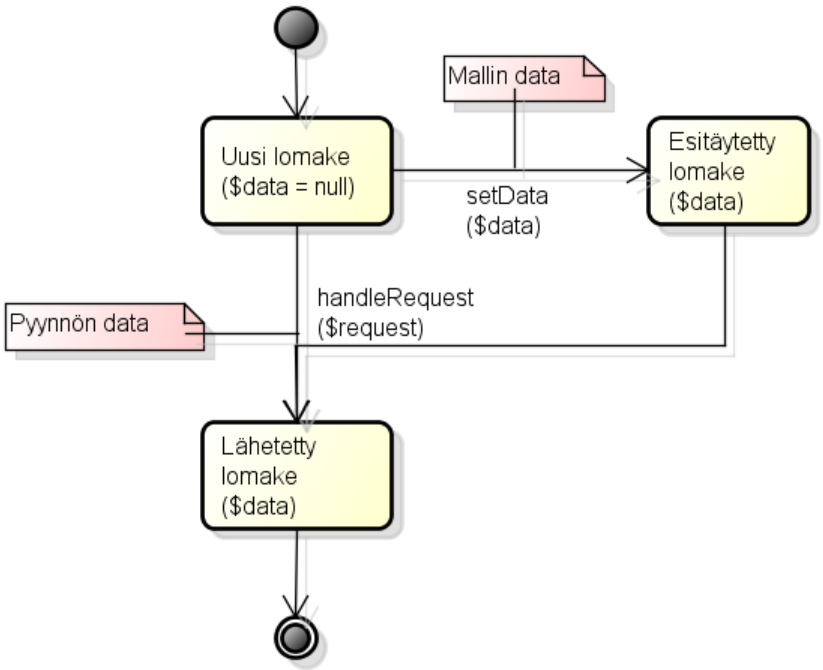
Koodiesimerkki 8. Lomakkeen validointi.

Edellä mainittua tapaa on käytetty muuallakin järjestelmässä ja sillä saadaan helposti aikaan lomakekohtainen validointi silloin, kun Symfonyn valmiit rajoitetyypit eivät riitä tai sellaisia ei erikseen haluta tehdä lisää. Validointi olisi mahdollista toteuttaa myös Prope-

lin tasolla, mutta lomakkeet eivät aina ole yksi-yhteen tietokantamallin kanssa. Jos lomakkeet noudattaisivat aina tietokannan sanelemaa mallia, rajoittaisi se käyttöliittymän suunnittelua huomattavasti. Tästä syystä validointi on sijoitettu lomakeluokkiin.

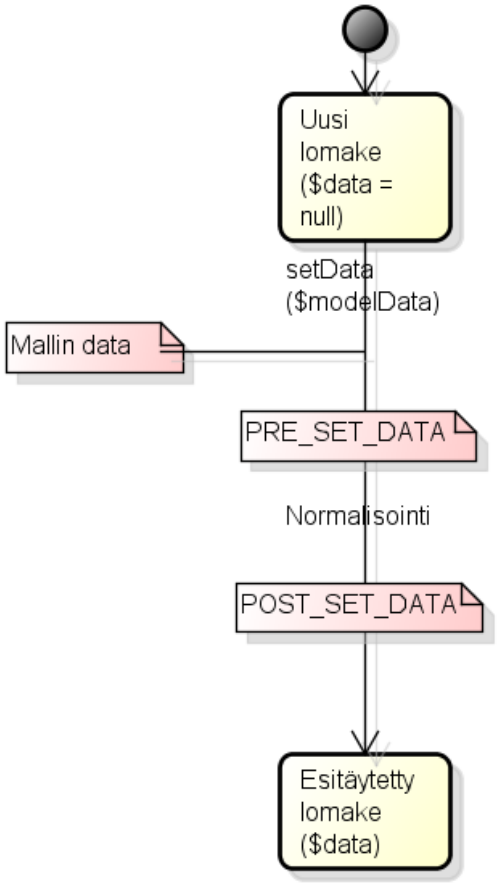
Lomakkeet olisi käsittelijöiden tapaan hyvä pitää mahdollisimman ohuena, eli liiketoimintalogiikka olisi hyvä sijoittaa omiin luokkiinsa, ettei sitä joudu kopioimaan lomakkeesta toiseen. Tällöin noudatetaan DRY-periaatetta (Don't Repeat Yourself) [33]. Tällöin toteutuu myös *Single responsibility principle* [34], eli jos liiketoimintalogiikkaan tulee muutos, tarvitsee se tehdä vain yhteen paikkaan, kun jokainen lomake ei erikseen toteuta jotakin logiikan osaa vaan huolehtii vain itse lomakkeen rakentamisesta ja validoinnista. Ohjelmakoodin koheesio lisääntyy, eli moduulin toiminnan kohde on tarkkaan määritelty ja keskittynyt [35]. Lopputuloksena korkeassa koheesiossa ovat koodin uudelleenkäytettävyys, luotettavuus, ymmärrettävyys ja testattavuus.

Järjestelmä käyttää rakennetun lomakkeen muokkaamiseen Symfonyn lomaketapahtumia. Lomake lähettää elinkaarensa aikana eri kohdissa tapahtumia, joihin voidaan lomakkeen luonnin yhteydessä sitoa kuuntelijoita, joiden avulla lomaketta tai siihen kiinnitettyä dataa voidaan muokata. Kuvassa 4 on esitetty lomakkeen käsittelyn kulku.



Kuva 4. Lomakkeen lähetyksen työnkulku [36].

Järjestelmässä käytetään tapahtumaa PRE_SET_DATA, joka lähetetään ennen kuin lomake täytetään tuoteolion tiedoilla. Tapahtuman lähetys tapahtuu, kun käsittelijässä kutsutaan lomakkeen setData-funktiota. Kuvassa esitettynä lomakkeen esitäytön aikana lähetetyt tapahtumat.



Kuva 5. Lomakkeen esitäytön työnkulku [36].

Tuotteen kenttien mahdollisia valintoja muokataan vastaamaan kunkin käyttäjän liiketoimintaroolin mukaista käyttötapaa. Esimerkiksi myyjä voi luoda tuotteen vain tietyllä statuksella ja ostaja- sekä -ostopäällikkökentät voidaan täyttää valmiiksi kirjautuneen käyttäjän tietojen perusteella. Samoin jo annetusta hinnasta voidaan kopioida oletusarvoja muille hinnoille. Koodiesimerkissä 9 on esimerkki lomaketapahtuman asetuksesta.

```
$builder->addEventListener(FormEvents::PRE_SET_DATA,     function
(FormEvent $event) {
          $form = $event->getForm();

          if($this->token_storage->getToken()->getUser()
          ->getBusinessRole() === User::BUSINESS_ROLE_SELLER){
                    $choices = array(
                              Item::STATUS_WANTED => 'Wanted'
                              );
          }
          else{
                    $choices = ItemQuery::getStatusSelectList();
          }
});
```

Koodiesimerkki 9. Lomaketapahtuman asetus.

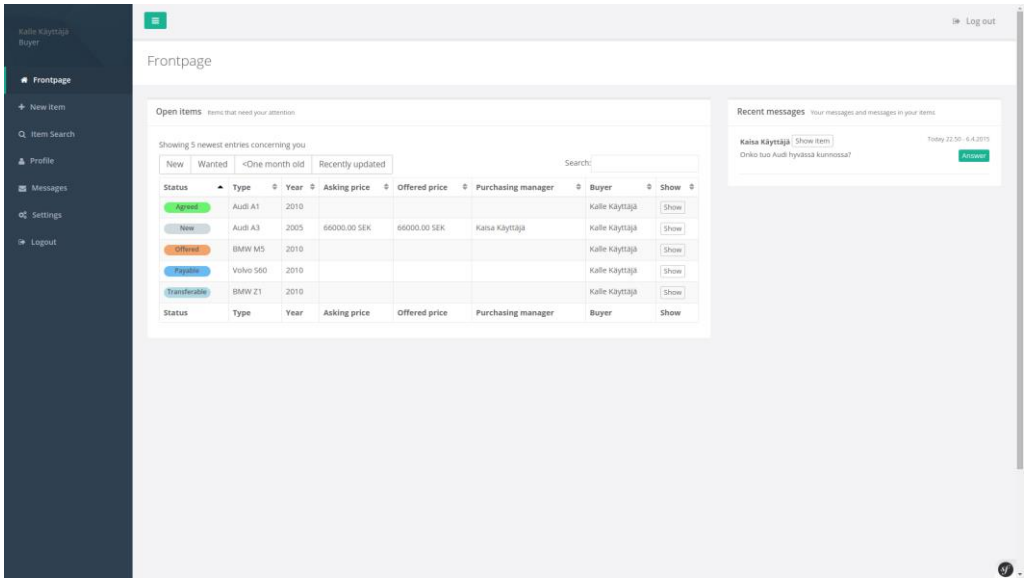
5.9 Muut Bundlet

Sovelluksen turvallisuusasetuksissa käyttäjien lähteeksi on määritelty Symfonyn UserProviderInterface-rajapinnan toteuttava UserProvider-luokka, joka sisältää metodit käyttäjien noutamiselle. Samassa asetustiedostossa määritellään reitit sisäänkirjautumiselle ja sen tarkistukselle. *SecurityController* sisältää käsittelyfunktiot sisäänkirjautumissivulle sekä uuden tai unohtuneen salasanan asetukselle. *UserProvider* on osa *UserBundlea*.

TestsBundle sisältää sovelluksen *PHPUnittia* käyttävät yksikkötestit ja funktionaaliset testit. Jälkimmäisessä käytetään Symfonyn crawler-komponenttia, jonka avulla sovelluksen käyttöliittymässä voidaan navigoida. *UICommonsBundle* sisältää kaikille mahdollisille käyttöliittymille yhteiset käyttöliittymän osat, kuten resurssit, ja lomaketyyppien määritykset.

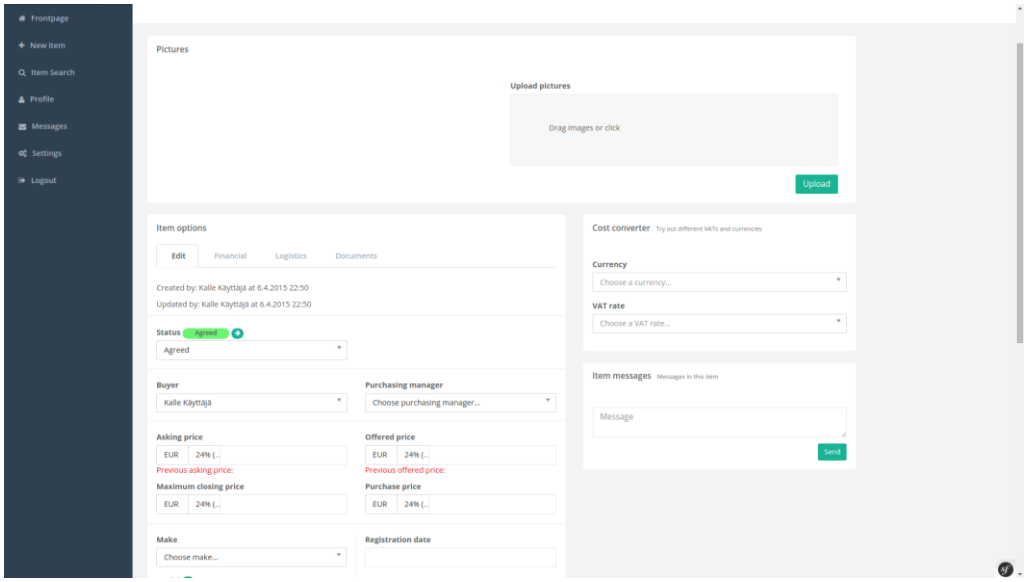
5.10 Käyttöliittymä

Järjestelmän etusivulla käyttäjä näkee hänelle ajankohtaiset ja tärkeät tuotteet sekä niihin liittyvät viestit. Navigointi tapahtuu sivun oikealla laidalla sijaitsevasta valikosta. Kuvassa 6 on järjestelmän etusivu.



Kuva 6. Järjestelmän etusivu.

Tuotesivulla käyttäjä voi syöttää tietoja eri välilehdille, viestiä tuotteesta muille käyttäjille ja vertailla hintoja eri valuutoissa. Kuvassa 7 on esimerkki tuotesivusta.



Kuva 7. Järjestelmän tuotesivu.

Käyttöliittymä toimii responsiivisesti eli selainikkunaa pienennettäessä tai pieniruutuisemmalla päätelaitteella selatessa valikko muuttuu pienemmäksi, ja sivun eri elementit siirtyvät allekkain elementtien luokka-attribuutteihin määriteltyjen leveyksien mukaisesti.

6 Yhteenveto

Tämän insinöörityön tarkoituksena oli toteuttaa helppokäyttöinen asiakkaan tuotteiden hankkimisen prosessia helpottava järjestelmä. Järjestelmän luonti alkoi asiakkaan prosessin selvittämisestä ja vaatimuksista, joita se sovellukselle asettaa. Työ aloitettiin suunnittelemalla tietomalli, joka olisi niin geneerinen kuin mahdollista, jotta se sopisi hyvin monenlaisille tuotteille.

Kirjoitushetkellä asiakaskokemukset sovelluksesta ovat vähäisiä, mutta sen toteuttaminen hyvillä käytännöillä modulaarisesti ja sovelluskehystä käyttäen takaa tulevaisuudessa myös helpon muokattavuuden. Käytettävyysvaatimuksiin responsiivisuudesta lähtien pystyttiin vastaamaan hyvin käyttämällä ominaisuuden valmiiksi sisältävää front end -sovelluskehystä.

Järjestelmä toteutettiin Symfony-sovelluskehyksen kirjoitushetkellä uusimmalla versiolla. Uusimpien ominaisuuksien lisäksi sovelluskehyksestä ja järjestelmän kehityksestä opittiin web-sovellusten kehitykseen liittyviä hyviä käytäntöjä. Symfony2 liitännäisineen osoittautui erinomaiseksi sovelluskehykseksi web-ohjelmointiin. Se kattaa laajasti kaikki ominaisuudet, joita web-kehityksessä tarvitaan. Se on jatkuvasti kehittyvä, avoin ja helposti laajennettavissa tarpeellisilla moduuleilla.

Symfonyn kaltaisen sovelluskehyksen käyttäminen on aina kaikkein pienimpiä sovelluksia lukuun ottamatta suositeltavaa. Ne saavat kehittäjän kirjoittamaan laadukasta, ylläpidettävää koodia kuin itsestään ja tekevät tarpeettomaksi kaikille sovelluksille yhteisten perusominaisuuksien uudelleenkirjoittamisen. Symfony helpottaa päivittäistä kehitystyötä esimerkiksi hyvien debug-työkalujensa ja komentojensa ansiosta.

Java-maailmasta tuttu riippuvuusinjektio osoittautui kehityksen edetessä erinomaiseksi tavaksi hallita sovelluksen eri palvelut ja niiden väliset riippuvuudet. Se on arkkitehtuurisesti selkeä ja vankka tapa kirjoittaa testattavaa, ylläpidettävää ja helposti luettavaa koodia ja kannustaa Single Responsibility -periaatteen mukaisesti luomaan modulaarisesti ja rajapintojen kautta toimivan sovelluksen, jonka kehittämiseen on myös uusien kehittäjien helppo lähteä mukaan. Tällaisella modulaarisella arkkitehtuurilla toteutettua sovellusta on helppo laajentaa ilman, että se vaatii laajoja, raskaita muutoksia lähdekoodiin.

Lähteet

1 Doyle, Matt. 2010. Beginning PHP 5.3. Indianapolis: Wiley Publishing, Inc.

2 Schechter, Greg. Visualizing Facebook's PHP codebase. 2011. Verkkodokumentti. https://www.facebook.com/note.php?note_id=10150187460703920. Päivitetty 31.5.2011. Luettu 3.2.2015.

3 Why should I use a framework? Verkkodokumentti. SensioLabs. <http://symfony.com/why-use-a-framework>. Luettu 3.2.2015.

4 6 good reasons to use Symfony. Verkkodokumentti. SensioLabs. <http://symfony.com/six-good-reasons>. Luettu 10.12.2014.

5 The Release Process. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/contributing/community/releases.html>. Luettu 3.2.2015.

6 Performance. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/book/performance.html>. Luettu 8.4.2015.

7 Pseudo-elements. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements>. Päivitetty 3.3.2015. Luettu 3.2.2015.

8 Usage of JavaScript libraries for websites. Verkkodokumentti. World Wide Web Technology Surveys. <http://w3techs.com/technologies/overview/javascript_library/all>. Päivitetty 18.3.2015. Luettu 18.3.2015.

9 Browser Support. Verkkodokumentti. The jQuery Foundation. <https://jquery.com/browser-support/>. Luettu 8.4.2015.

10 Bootstrap 3 released. Verkkodokumentti. Otto ym. The Official Bootstrap Blog. <http://blog.getbootstrap.com/2013/08/19/bootstrap-3-released/>. Päivitetty 19.8.2013. Luettu 8.4.2015.

11 Chrome DevTools Overview. Verkkodokumentti. Google. <https://developer.chrome.com/devtools>. Luettu 18.3.2015.

12 When it's still best to use a relational DBMS. Verkkodokumentti. DBMS2. <http://www.dbms2.com/2011/05/29/when-to-use-relational-database-management-system/>. Päivitetty 29.5.2011. Luettu 18.3.2015.

13 Tezer, O.S. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems. Verkkodokumentti. <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>. Luettu 18.3.2015.

14 Bowler, T., Bancer, W. 2009. Symfony 1.3 Web Application Development. Birmingham: Packt Publishing, s. 9.

15 The Architecture. Verkkodukumentti. SensioLabs. <http://symfony.com/doc/current/quick_tour/the_architecture.html>. Luettu 12.4.2015.

16 Alur, D. Crup, J., Malks, D. 2003. Core J2EE Patterns, Best Practices and Design Strategies, 2nd Ed. Westford, Massachusetss: Sun Microsystems Press, s. 166.

17 Controller. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/book/controller.html>. Luettu 9.4.2015.

18 Understanding how the Front Controller, Kernel and Environments Work together. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/cookbook/configuration/front_controllers_and_kernel.html>. Luettu 8.4.2015.

19 Databases and Propel. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/book/propel.html>. Luettu 10.4.2015.

20 Fowler, Martin. 2002. Patterns of Enterprise Application Architecture. Crawfordsville, Indiana: Addison-Wesley Professional, s. 116.

21 Ten advantages of an ORM (Object Relational Mapper). Verkkodokumentti. MSDN Blogs. Glenn Block. <http://blogs.msdn.com/b/gblock/archive/2006/10/26/ten-advantages-of-an-orm.aspx>. Luettu 10.4.2015.

22 Templates. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/best_practices/templates.html>. Luettu 18.3.2015.

23 Twig. Verkkodokumentti. SensioLabs. <http://twig.sensiolabs.org/>. Luettu 18.3.2015.

24 Creating and Using Templates. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/book/templating.html>. Luettu 18.3.2015.

25 How to Use Assetic for Asset Management. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/cookbook/assetic/asset_management.html>. Luettu 24.3.2015.

26 Davis, Jeff. 2009. Open Source SOA. Greenwich, CT: Manning Publications Co.

27 Types of Injection. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/components/dependency_injection/types.html#property-injection>. Luettu 26.3.2015.

28 Service Container. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/book/service_container.html>. Luettu 26.3.2015.

29 Forms. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/quick_tour/the_architecture.html>. Luettu 12.4.2015.

30 Security. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/book/security.html>. Luettu 9.4.2015.

31 How to Create a Console Command. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/cookbook/console/console_command.html>. Luettu 8.4.2015.

32 The EventDispatcher Component. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/components/event_dispatcher/introduction.html>. Luettu 8.4.2015.

33 Venners, Bill. Orthogonality and the DRY principle. 2003. Verkkodokumentti. <http://www.artima.com/intv/dry.html>. Päivitetty 10.3.2003. Luettu 7.4.2015.

34 Martin, Robert C. 2002. Agile Software Development, Principles, Patterns and Practices. New Jersey: Prentice Hall.

35 Yourdon, E., Constantine, Larry L. 1979. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. New Jersey: Yourdon Press.

36 Form events. Verkkodokumentti. SensioLabs. <http://symfony.com/doc/current/components/form/form_events.html>. Luettu 7.4.2015.

Käyttötapaukset

Taulukko 1. Avoimien kohteiden näyttäminen

Yleiskuvaus	Etusivulla näytetään lista kohteista, jotka ovat sellaisessa tilassa, että ne odottavat kirjautuneelta käyttäjältä toimenpiteitä.
Käyttäjäroolit	Ostaja, ostopäällikkö, sihteeri, logistiikka, kunnossapito
Esiehdot	Käyttäjä on kirjautunut järjestelmään.
Tarkempi kuvaus	Järjestelmän etusivulla näytetään lista kohteista, joissa kirjautunut käyttäjä on vastuuhenkilönä ja jotka ovat sellaisessa tilassa, että ne odottavat käyttäjältä toimenpiteitä. Näytettävät kohteet riippuvat käyttäjän roolista: - Ostajalle näytetään tilat uusi, tarjous tehty, kauppa sovittu, saa maksaa, vapaa siirrettäväksi - Ostopäällikölle näytetään tilat uusi, tarjous tehty, kauppa sovittu, saa maksaa, vapaa siirrettäväksi - Logistiikalle näytetään tilat vapaa siirrettäväksi, siirrossa - Kunnostukselle näytetään tilat siirrossa, myyntitarkastuksessa Etusivulla näytetään kohteista tila, merkki, malli, vuosimalli ja linkki tarkempiin tietoihin.
Poikkeukset	-
Lopputulos	Käyttäjä näkee listan itseään koskevista kohteista.
Muuta	Kohdetaulukossa näytetään painikkeita, joilla kohteiden listausta voidaan edelleen rajata.

Taulukko 2. Uusimpien kommenttien näyttäminen

Yleiskuvaus	Etusivulla näytetään lista uusimmista kommenteista, jotka on lisätty kohteisiin, joissa käyttäjä on vastuuhenkilönä.
Käyttäjäroolit	Ostaja, ostopäällikkö, sihteeri, logistiikka, kunnossapito
Esiehdot	Käyttäjä on kirjautunut järjestelmään
Tarkempi kuvaus	Etusivulla näytetään lista uusimmista kommenteista, jotka liittyvät kohteisiin, joissa käyttäjä on vastuuhenkilönä. Kommenttien yhteydessä on linkki kommenttiin liittyvään kohteeseen.
Poikkeukset	-
Lopputulos	Käyttäjä näkee listan itseään ajankohtaisista kommenteista.
Muuta	-

Taulukko 3. Kohteiden hakeminen

Yleiskuvaus	Käyttäjä hakee järjestelmässä olevia kohteita eri hakuehdoilla.
Käyttäjäroolit	Ostaja, ostopäällikkö, sihteeri, logistiikka, kunnossapito
Esiehdot	Käyttäjä on kirjautunut järjestelmään ja hänellä on oikeus käyttää kohdehakua.
Tarkempi kuvaus	Kohteita voidaan hakea seuraavilla kriteereillä: myyjäliike, merkki, malli, vuosimalli (aikajakso), lisäyspäivä (aikajakso), viimeisin muokkauspäivä, vastuullinen ostaja, vastuullinen ostopäällikkö ja kohteen tila. Hakutuloksissa näytetään suppeat tiedot kohteesta: tila, merkki, malli, vuosimalli, lisäyspäivämäärä, linkki tarkempiin tietoihin.
Poikkeukset	Mikäli yhtään hakukriteereitä vastaavaa kohdetta ei löydy, näytetään käyttäjälle tyhjä lista.
Lopputulos	Käyttäjä näkee listan hakuehtoihin sopivista kohteista ja pääsee linkin kautta tarkastelemaan ja muokkaamaan kohdetta.

Muuta	-

Taulukko 4. Uuden kohteen lisääminen

Yleiskuvaus	Käyttäjä lisää järjestelmään uuden kohteen.
Käyttäjäroolit	Ostaja, ostopäällikkö, myyjä
Esiehdot	Käyttäjä on kirjautunut järjestelmään ja hänellä on oikeus lisätä kohteita.
Tarkempi kuvaus	Käyttäjä syöttää kohteesta haluamansa tiedot lomakkeelle ja painaa Tallenna-painiketta. Kohde tallentuu järjestelmään myöhemmin tarkasteltavaksi ja muokattavaksi.
Poikkeukset	Mikäli käyttäjällä ei ole oikeitta lisätä kohteita, päävalikon Lisää kohde -valinta ei ole näkyvissä. Mikäli kohteesta puuttuu pakollisia tietoja, näytetään käyttäjälle virheilmoitus ja kohdetta ei tallenneta.
Lopputulos	Kohde on lisätty järjestelmään.
Muuta	-

Taulukko 5. Kohteen tietojen tarkasteleminen

Yleiskuvaus	Käyttäjä tarkastelee kohteen tietoja
Käyttäjäroolit	Kaikki
Esiehdot	Käyttäjä on kirjautunut järjestelmään ja löytänyt haluamansa kohteen kohdehaulla tai muuten.
Tarkempi kuvaus	Käyttäjä painaa kohdelinkkiä, jolloin hänet ohjataan kohteen tarkkoihin tietoihin. Tiedot on jaoteltu eri välilehdille (perustiedot, taloustiedot, logistiikan tiedot, dokumentit). Käyttäjä ohjataan oman roolinsa mukaiselle välilehdelle. Käyttöoikeuksista riippuen käyttäjä voi katsella myös muilla välilehdillä olevia tietoja.
Poikkeukset	-
Lopputulos	Käyttäjä näkee oikeuksiensa mukaiset kohteen tarkat tiedot.

Muuta	-

Taulukko 6. Kohteen tietojen muokkaaminen

Yleiskuvaus	Käyttäjä muokkaa kohteen tietoja
Käyttäjäroolit	Ostaja, ostopäällikkö, sihteeri
Esiehdot	Käyttäjä on kirjautunut järjestelmään, löytänyt haluamansa kohteen kohdehaulla tai muuten ja hänellä on oikeus muokata kohdetta.
Tarkempi kuvaus	Käyttäjä painaa kohdelinkkiä, jolloin hänet ohjataan kohteen tarkkoihin tietoihin. Käyttäjä lisää kohteeseen tietoja tai muokkaa niitä ja painaa Tallenna-painiketta.
Poikkeukset	Mikäli käyttäjällä ei ole oikeutta muokata kohdetta, Tallenna-painike ei ole näkyvissä. Mikäli kohteesta puuttuu pakollisia tietoja, näytetään käyttäjälle virheilmoitus ja kohdetta ei tallenneta.
Lopputulos	Kohteen tiedot päivittyvät järjestelmään.
Muuta	Eri välilehdillä olevat tiedot tallentuvat erikseen.

Taulukko 7. Omien tietojen muokkaaminen

Yleiskuvaus	Käyttäjä muokkaa omia tietojaan
Käyttäjäroolit	Kaikki
Esiehdot	Käyttäjä on kirjautunut järjestelmään.
Tarkempi kuvaus	Käyttäjä painaa Omat tiedot -painiketta, jolloin hänelle näytetään lomake hänen omista tiedoistaan. Käyttäjä muokkaa tietoja ja painaa Tallenna-painiketta.
Poikkeukset	-
Lopputulos	Käyttäjän tiedot päivittyvät järjestelmään.
Muuta	-

Taulukko 8. Uuden käyttäjän lisääminen

Yleiskuvaus	Lisätään uusi käyttäjä järjestelmään.
Käyttäjäroolit	Pääkäyttäjä
Esiehdot	Käyttäjä on kirjautunut järjestelmään. Käyttäjällä on tiedossa uuden käyttäjän sähköpostiosoite.
Tarkempi kuvaus	Käyttäjä painaa Uusi käyttäjä -painiketta, jolloin hänellä näytetään käyttäjälomake. Käyttäjä painaa tallenna-painiketta, jolloin käyttäjä tallennetaan järjestelmään ja käyttäjälle lähetetään sähköposti, jossa on uuden käyttäjän käyttäjätunnus ja linkki, josta käyttäjä pääsee asettamaan itsellensä salasanan. Kun salasana on asetettu, käyttäjä voi kirjautua järjestelmään.
Poikkeukset	-
Lopputulos	Uusi käyttäjä on lisätty järjestelmään.
Muuta	-

Taulukko 9. Käyttäjätietojen muokkaaminen

Yleiskuvaus	Pääkäyttäjä muokkaa käyttäjien tietoja tai käyttöoikeuksia
Käyttäjäroolit	Pääkäyttäjä
Esiehdot	Käyttäjä on kirjautunut järjestelmään.
Tarkempi kuvaus	Käyttäjä etsii haluamansa käyttäjän käyttäjähaulla, lainaa listauksessa olevaa linkkiä päästäkseen käyttäjälomakkeelle ja muokattuaan tietoja painaa Tallenna-painiketta.
Poikkeukset	Mikäli käyttäjällä ei ole oikeutta muokata kohdetta, Tallenna-painike ei ole näkyvissä. Mikäli kohteesta puuttuu pakollisia tietoja, näytetään käyttäjälle virheilmoitus ja kohdetta ei tallenneta.
Lopputulos	Pääkäyttäjä on muuttanut käyttäjän tietoja.
Muuta	Käyttäjän käyttöoikeudet määräytyvät käyttäjäroolin mukaan.

Taulukko 10. Myyjäliikkeen lisäys ja muokkaus

Yleiskuvaus	Käyttäjä lisää uuden myyjäliikkeen järjestelmään tai muokkaa vanhaa myyjäliikettä.
Käyttäjäroolit	Ostaja, ostopäällikkö
Esiehdot	Käyttäjä on kirjautunut järjestelmään.
Tarkempi kuvaus	Käyttäjä painaa Lisää myyjäliike -painiketta lisätäkseen liikkeen tai myyjäliikelistauksen yhteydessä olevaa linkkiä päästäkseen lomakkeelle, jossa hän voi syöttää ja muokata liikkeen tietoja. Lopuksi käyttäjä painaa Tallenna-painiketta.
Poikkeukset	-
Lopputulos	Myyjäliike on tallennettu järjestelmään ja se näkyy valittavana kohteen tiedoissa.
Muuta	-

Taulukko 11. Merkin lisäys ja muokkaus

Yleiskuvaus	Käyttäjä lisää uuden merkin järjestelmään tai muokkaa vanhaa merkkiä.
Käyttäjäroolit	Ostaja, ostopäällikkö, myyjä
Esiehdot	Käyttäjä on kirjautunut järjestelmään.
Tarkempi kuvaus	Käyttäjä painaa Lisää merkki -painiketta lisätäkseen merkin tai merkkilistauksen yhteydessä olevaa linkkiä päästäkseen lomakkeelle, jossa hän voi syöttää ja muokata merkin tietoja. Lopuksi käyttäjä painaa Tallenna-painiketta.
Poikkeukset	Merkkejä ei voi poistaa, jos järjestelmässä on kyseisen merkkisiä kohteita.
Lopputulos	Merkki on tallennettu järjestelmään ja se näkyy valittavana kohteen tiedoissa.
Muuta	-

Taulukko 12. Mallin lisäys ja poisto

Yleiskuvaus	Käyttäjä lisää uuden mallin järjestelmään tai muokkaa vanhaa mallia.
Käyttäjäroolit	Ostaja, ostopäällikkö, myyjä
Esiehdot	Käyttäjä on kirjautunut järjestelmään ja hänellä on oikeus käsitellä malleja.
Tarkempi kuvaus	Käyttäjä valitsee merkin, jolle haluaa lisätä mallin, syöttää mallin nimen ja painaa Tallenna-painiketta. Poistaminen tapahtuu mallilistauksen yhteydessä olevasta painikkeesta.
Poikkeukset	Malleja ei voi poistaa, jos järjestelmässä on kohteita, joissa on kyseinen malli valittuna.
Lopputulos	Malli on lisätty/poistettu järjestelmästä.
Muuta	-

Taulukko 13. Hinnat

Yleiskuvaus	Kohteiden hintatiedot tulee pystyä syöttämään usealla eri valuutalla ja arvonlisäverolla.
Käyttäjäroolit	-
Esiehdot	-
Tarkempi kuvaus	Tuotteita ostetaan useasta eri maasta usealla valuutalla. Käyttäjän tulee pystyä syöttämään tuotteen hinta haluamallaan valuutalla ja arvonlisäverolla ja järjestelmän tulee näyttää tuotteen hinta alkuperäisessä valuutassa sekä euroissa valuuttakurssin mukaan.
Poikkeukset	-
Lopputulos	-
Muuta	Kohteen maksupäivän jälkeen tuotteen hinta sekä alkuperäisessä valuutassa että euroissa tulee säilyä muuttumattomana.