Opinnäytetyö (AMK)

Tietojenkäsittely

Yrityksen tietojärjestelmät

2016

Tapio Pensasmaa

ODOON LOMAKKEIDEN JA RAPORTTIEN KUSTOMOINTI

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Yrityksen tietojärjestelmät

2016 | 49 sivua

Ohjaaja: Tuomo Helo

Tapio Pensasmaa

ODOON LOMAKKEIDEN JA RAPORTTIEN KUSTOMOINTI

Opinnäytetyön tavoitteena oli selvittää, miten avoimeen lähdekoodiin perustuvan Odoo-yritysohjelmiston lomakkeita sekä tarjouksissa ja laskuissa käytettäviä raporttipohjia voidaan kustomoida vastaamaan työn toimeksiantajan EM Systems Oy:n asiakkaiden tarpeita. Työssä perehdyttiin Odoon rakenteeseen ja moduulien toimintaan sekä selvitettiin Odoon kehittäjätilan ominaisuuksia.

Odoon periytymismekanismit antavat mahdollisuuden tehdä moduulien avulla laajennuksia tai muutoksia muihin moduuleihin ilman, että muokataan olemassa olevien moduulien koodia. Periytymistä voidaan käyttää kaikilla tasoilla: malleissa, sovelluslogiikassa ja näkymissä. Odoon kehittäjätilassa muutoksia voidaan tehdä suoraan ohjelmiston käytön aikana.

Kustomointi moduulien avulla on huomattavasti selkeämpää ja kontrolloidumpaa verrattuna Odoon kehittäjätilaan. Halutut muutokset voidaan myös toteuttaa useissa tietokannoissa asentamalla moduuli niihin. Kehittäjätilan muutokset tallentuvat ainoastaan tietokantaan, johon ne tehdään. Jos samoja muutoksia halutaan käyttää toisessa tietokannassa, ne on tehtävä aina uudestaan.

Työn tuloksena valmistui kaksi moduulia Odoon asiakaslomakkeen ja raporttipohjan kustomointiin. Moduulien rakennetta voidaan käyttää pohjana uusissa moduuleissa, jotka on tarkoitettu saman tyyppisten muutosten tekemiseen.

ASIASANAT:

Odoo, moduuli, toiminnanohjausjärjestelmä, ERP-järjestelmä, MVC, kustomointi

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Business Information Systems

2016 | 49 pages

Instructor: Tuomo Helo

Tapio Pensasmaa

CUSTOMIZING FORMS AND REPORTS OF ODOO

Odoo is an open-source business management application framework. The objective of this thesis was to study how to customize Odoo's forms and built-in invoice and quotation reports to meet the needs of customers using Odoo. The basics of the Odoo architecture, the functions of modules, and features of the Odoo developer mode were studied in this work. The thesis was commissioned by EM Systems Oy.

Odoo provides inheritance mechanisms that allow modules to add features and extensions to existing modules without altering their original code. The inheritance and changes can happen at all levels: models, business logic, and views. In the developer mode, changes can be made without having to restart the Odoo server or reinstall the module.

Customization through modules gives better ability to manage modifications than the Odoo developer mode. The desired changes can be implemented in any other database by installing the module. All the developer mode changes are isolated within the modified database and can not be copied or moved to other instances of Odoo.

As a result of the study, two modules were built to customize Odoo's contact form and default report templates. The basic structure of these modules can be used to develop the new custom modules of similar type to extend existing modules.

KEYWORDS:

Odoo, customization, module, enterprise resource planning, ERP system, model, view, MVC

SISÄLTÖ

SANASTO **6**

1 JOHDANTO **7**

2 ERP-JÄRJESTELMÄ JA SEN MUKAUTTAMINEN **9**
2.1 Järjestelmän konfigurointi 10
2.2 Järjestelmän kustomointi 11

3 ODOON RAKENNE **16**
3.1 Model-View-Controller (MVC) 16
3.2 Odoon moduulit 18

4 DEVELOPER MODE – KEHITTÄJÄTILA **26**

5 ASIAKASLOMAKKEEN KUSTOMOINTI **30**
5.1 Asiakastietojen tuominen 31
5.2 Periytyminen Odoossa 32
5.3 Moduuli asiakaslomakkeen kustomointiin 33

6 RAPORTTIPOHJAN KUSTOMOINTI **39**
6.1 Raporttien rakenne 39
6.2 Moduuli raporttipohjan kustomointiin 41

7 POHDINTA **46**

LÄHTEET **48**

LIITTEET

Liite 1. Asiakaslomakkeen näkymän XML-tiedosto.
Liite 2. Asiakaslomakkeen mallin Python-tiedosto.
Liite 3. Muokattu laskupohja.
Liite 4. Odoon oletuslaskupohja.

KUVAT

Kuva 1. Odoon arkkitehtuuri (Dony 2014, 14). 16
Kuva 2. MVC-mallin vastuunjako verkkopalvelussa (mukaillen Arkkitehtuuri ja MVC 2014). 18
Kuva 3. Esimerkki moduulin rakenteesta (Odoo S.A, 2015b). 20
Kuva 4. Esimerkki __openerp__.py-tiedostosta. 21
Kuva 5. Esimerkki mallin määrittelystä. 22
Kuva 6. Relaatiokenttiä (Odoo S.A. 2015i). 23
Kuva 7. Compute ja onchange (Odoo S.A. 2015a). 23
Kuva 8. Näkymän perusrakenne. 24
Kuva 9. Kehittäjätilan pudotusvalikko. 26
Kuva 10. Manage Views -lomake. 26
Kuva 11. Näkymäeditori – View Editor. 27
Kuva 12. Näkymäeditorin Add-valikko. 27
Kuva 13. Uuden kentän attribuuttivalikko. 28
Kuva 14. Lomakenäkymäeditori – Edit FormView. 29
Kuva 15. Import-toiminto käyttöliittymässä. 30
Kuva 16. Odoon asiakastietolomake – henkilötiedot. 34
Kuva 17. Osa näkymän XML-tiedostosta. 35
Kuva 18. Osa mallin Python-tiedostosta. 36
Kuva 19. Esimerkki mallin XML-tietueista. 37
Kuva 20. Toimialaluokitusmallin tietueita. 37
Kuva 21. Asiakaslomakkeen uusia kenttiä. 38
Kuva 22. Esimerkki report-elementin määrittelystä. 40
Kuva 23. QWeb-templaatin perusrakenne. 41
Kuva 24. Oletuslaskun yläosa. 42
Kuva 25. Oletuslaskun alatunniste. 42
Kuva 26. external_layout-templaatti. 43
Kuva 27. layout_emsys.xml-tiedoston minimal_layout-templaatti. 44
Kuva 28. Raporttitiedoston perusrakenne. 44

KUVIOT

Kuvio 1. Kustomoinnin taso vertailluissa yrityksissä (Panorama Consulting Solutions 2015, 8). 15

TAULUKOT

Taulukko 1. Kustomoinnin ajurit (Aslam ym. 2014). 12

SANASTO

API	Application Programming Interface, ohjelmointirajapinta, tarjoaa sovellusten laatimiseen matalamman abstraktiotason välineen, jossa käytettävän ohjelma- tai luokkakirjaston rakenteet ovat suoraan ohjelmoijan käytettävissä (Laine 2006).
CSS	Cascading Style Sheets, tekniikka, joka mahdollistaa tyylien ja ulkoasun määrittelyn www-sivuja luotaessa (MOT 2016).
CSV	Comma-Separated Values, tiedostomuoto, jolla tallennetaan yksinkertaista taulukkomuotoista tietoa tekstitiedostoon (Shafranovich 2005).
HTML	Hypertext Markup Language, sivunkuvaus- ja ohjauskieli, jolla luodaan eri ympäristöihin sopivia hypertekstiasiakirjoja (MOT 2016).
HTTP	Hypertext Transfer Protocol, protokolla, jolla www-asiakkaat ja -palvelimet viestivät keskenään (MOT 2016).
JavaScript	Selaimissa yleisesti käytetty komentosarjakieli vuorovaikutteisten, dynaamisten www-sivujen laatimiseen (MOT 2016).
Kustomoida	Muokata tai muuntaa asiakkaan tai käyttäjän toiveiden mukaiseksi, räätälöidä (MOT 2016).
Moduuli	Toiminnallinen osa, jolla on täsmällisesti määritelty liittymä ympäristöönsä ja josta yhdessä muiden osien kanssa voidaan muodostaa laajuudeltaan tai toiminnoiltaan erilaisia kokonaisuuksia (MOT 2016).
MVC	Model-View-Controller, arkkitehtuurimalli, jossa sovellus on jaettu malleihin, näkymiin ja kontrollereihin (Kasampalis 2015, 93).
ORM	Object-Relational Mapping, tekniikka, jolla olio-ohjelmoinnin oliot ja niiden suhteet voidaan tallentaa automaattisesti relaatiotietokantaan (Hoffer ym. 2013, 14-7.)
SQL	Structured Query Language, relaatiotietokantojen käsittelyssä käytetty kyselykieli (MOT 2016).
Template	Pohja, kaavain, malline, malli (MOT 2016).
XML	Extensible Markup Language, kehittynyt dokumenttien ja rakenteisen tiedon kuvauskieli, jolla voidaan lisätä myös esitystapaa kuvaavia määritelmiä dataan (MOT 2016).
XPath	XML Path Language, kieli, jonka lausekkeiden sijaintipolkuja käytetään osoittamaan XML-dokumentin puumallin solmuihin (Silberschatz 2011, 999).

1 JOHDANTO

Toiminnanohjausjärjestelmät perustuvat niiden valmistajien näkemykseen ja kokemukseen erilaisten liiketoimintaprosessien parhaista käytännöistä. Koska yritykset ovat erilaisia, on järjestelmissä runsaasti erilaisia konfigurointivaihtoehtoja, joilla yritys voi mukauttaa ohjelmistoa vastaamaan omia toimintatapojaan. Suurin osa yrityksistä tekee kuitenkin myös muita muutoksia järjestelmään.

Yritys voi joutua sopeuttamaan hyvinkin paljon liiketoimintaprosessejaan, kun se ottaa ERP-järjestelmän käyttöön. Jos yritys ei halua tai voi muuttaa toimintatapojaan eikä järjestelmästä löydy riittävästi konfigurointimahdollisuuksia, vaihtoehtona on ohjelmiston muokkaaminen tai mukauttaminen vastaamaan yrityksen prosesseja.

Kustomoinnissa ERP-sovellusten koodiin tehdään lisäyksiä, laajennuksia tai poistoja, joilla ei kuitenkaan muuteta suoraan ydinohjelmiston koodia tai rakenteita. Kustomointi on aina riski, koska siinä muutetaan ohjelman toimintaa. Muutokset voivat myös aiheuttaa ongelmia ohjelmiston ylläpidossa ja tukeen liittyvissä asioissa. Lisäksi ohjelmistopäivitykset saattavat muuttaa järjestelmää niin, että kustomoinneissa käytetty koodi täytyy kirjoittaa ainakin osittain uudestaan. Viimeistään silloin, kun muokataan ohjelmiston lähdekoodia tai ydinrakenteita, kehittäjien tulisi koodauksen lisäksi ymmärtää myös ERP-järjestelmä ja siihen liittyvä liiketoiminnan hallinta kokonaisuutena.

Odoo on modulaarinen yritysjärjestelmä, johon on liitetty perinteisten ERP-toimintojen lisäksi myös sisällönhallintaan liittyviä ominaisuuksia. Tässä työssä on käytetty Odoon versiota 8, joka on julkaistu AGPLv3-lisenssillä, joten ohjelmiston lähdekoodi on vapaasti saatavilla ja muokattavissa, kunhan lisenssiä ei poisteta. Avoin lähdekoodi tekee kustomoinnin ja lisäosien kehittämisen helpommaksi verrattuna omisteisten ohjelmistojen suljettuun lähdekoodiin.

Odoo antaa hyvät mahdollisuudet erilaisten ominaisuuksien kustomointiin ilman, että muokataan ohjelmiston lähdekoodia. Periytymismekanismien avulla voidaan ydinmoduulien malleihin ja näkymiin tehdä muutoksia muilla moduuleilla. Odoon kehittäjätilassa voidaan muokata näkymiä ja toimintoja suoraan käyttöliittymästä.

Opinnäytetyön tavoitteena on selvittää, miten Odoon lomakkeita sekä tarjouksissa ja laskuissa käytettäviä raporttipohjia voidaan kustomoida vastaamaan Odoon käyttäjien

tarpeita. Erilaisissa lomakkeissa olevia kenttiä käytetään tietojen syöttämiseen ja muokkaamiseen. Puuttuvien kenttien lisääminen ja tarpeettomien poistaminen ovat tyypillisiä toiveita, joita asiakkailla on. Yritykset myös haluavat, että heidän lähettämänsä tarjoukset ja laskut vastaavat sommittelultaan ja ulkoasultaan yrityksen visuaalista ilmettä. Odoon oletuspohjat sekä tarjouksille että laskuille ovat persoonattomia, eikä tavallinen käyttäjä pysty niitä juurikaan muokkaamaan. Odoon laskupohjasta puuttuu myös tilisiirtolomake.

Työn toisena tavoitteena on, että työn pohjalta voidaan laatia toimeksiantajalle käytännönläheiset ohjeet siitä, miten moduuli tehdään ja miten sillä voidaan tehdä muutoksia Odoon ominaisuuksiin. Odoon omat kehittäjille tarkoitetut ohjeet ovat melko teoreettisia ja vaativat perehtymistä Odoon moduulien rakenteeseen ja toimintaan. Erilaisilta foorumeilta ja muilta nettisivuilta löytyvät neuvot ja ohjeet ovat tasoltaan kovin kirjavia, ja niiden etsiminen on työlästä ja aikaa vievää. Yleensä niissä pyydetään neuvoa jonkin yksittäisen ohjelmointiongelman ratkaisuun. Kirjallisuudestakaan ei juuri apua löydy: on olemassa vain yksi kirja, joka käsittelee Odoon kehitystyötä, mutta sekin on tarkoitettu kokeneemmille sovelluskehittäjille.

Opinnäytetyö on luonteeltaan toiminnallinen kehittämistyö, ja se on tehty toimeksiantona EM Systems Oy:lle, joka on pk-yritysten tiedonhallintaan keskittynyt yritys.

2 ERP-JÄRJESTELMÄ JA SEN MUKAUTTAMINEN

ERP-järjestelmä (Enterprise Resource Planning) eli toiminnanohjausjärjestelmä on integroitu, modulaarinen tietojärjestelmä, joka yhdistää yrityksen tai organisaation kaikkiin toimintoihin liittyvät datavirrat yhteen kokonaisvaltaiseen tietokantaan. ERP-järjestelmä auttaa hallitsemaan ja koordinoimaan yrityksen liiketoimintaprosessien eri osa-alueita. Tietojärjestelmä koostuu esimerkiksi talouteen, kirjanpitoon, tuotantoon, myyntiin, markkinointiin ja henkilöstöhallintoon liittyvistä ohjelmisto-osista, joista kerätty informaatio on yrityksen jokaisen osaston saatavilla yhtenäisenä ja reaaliaikaisena. (Hooshang ym. 2014, 358; Monk & Wagner 2013, 1.)

ERP-järjestelmät on suunniteltu parantamaan päätöstentekoa lisäämällä yrityksen kykyä tuottaa koko toiminnan kattavaa ajankohtaista ja paikkansapitävää informaatiota. Järjestelmä yhdistää yrityksen eri osastoilta saatavan datan ja synkronoi sen poistamalla päällekkäisyydet. Se varmistaa datan täsmällisyyden ja tarjoaa monipuolisen kokonaisnäkymän kaikesta tarvittavasta informaatiosta muodossa, josta on käyttäjille hyötyä. Koska useat työntekijät voivat käsitellä dataa samanaikaisesti päivittäisissä tehtävissään, myös työn tehokkuus lisääntyy. (Hooshang ym. 2014, 368.)

Internetin myötä on ollut mahdollista laajentaa ERP-sovellusten toiminta myös yrityksen ulkopuolelle. Esimerkiksi sisäiset liiketoimintaprosessit voidaan yhdistää asiakkaiden ja toimittajien liiketoimintaprosessien kanssa. ERP-järjestelmää voidaan käyttää myös alustana erilaisille sovelluksille, joiden avulla pystytään esimerkiksi alentamaan inventaariokustannuksia ja hallitsemaan paremmin toimitusketjuja sekä asiakassuhteita. Tuottajat, toimittajat ja jälleenmyyjät voivat koordinoida toimintojaan ja jäljittää tuotteita. Viivakoodien ja RFID-tunnisteiden avulla yritykset tietävät tuotteiden täsmällisen sijainnin ja pystyvät määrittämään tarkan toimitusajan asiakkaalle. Myös varkauksien ehkäisy on helpompaa. (Hooshang ym. 2014, 358.)

Nykyisiin yritysjärjestelmiin sisältyy web-komponentteja verkkokauppaa ja kansainvälistä viestintää varten. ERP-järjestelmillä pystytään parantamaan informaation kulkua eri paikkojen ja maiden välillä, mikä on tärkeää etenkin tuotanto- ja palvelusektoreilla. Järjestelmät pystyvät kommunikoimaan yli kielirajojen ja kulttuurien, ottamaan huomioon eri valuutat, ja kauppaa voidaan käydä reaaliaikaisten valuuttakurssien mukaan. Ohjelmistoissa voidaan ottaa huomioon eri maiden tai alueiden lait ja säädökset. (Hooshang ym. 2014, 358; Monk & Wagner 2013, 36.)

Päätöksenteko helpottuu, kun yritysjohto voi seurata liiketoimintaa riippumatta siitä, missä se tapahtuu. ERP-järjestelmän avulla liiketoimia on mahdollista hallita, eikä tarvitse tyytyä pelkkään seuraamiseen. Kaikki data on johdon saatavilla, ja se voi keskittyä liiketoimintaprosessien parantamiseen. Tehokas informaation kulku lisää tietoa myös tuotteista ja asiakkaista. Asiakaspalvelua voidaan parantaa, ja asiakkaille voidaan tarjota monipuolisempia palveluita. Markkinoiden muutoksiin voidaan reagoida nopeammin ja tuotteiden kehittäminen sekä markkinoille saaminen nopeutuu. (Bradford 2015, 6; Monk & Wagner 2013, 36.)

ERP-järjestelmät myydään moduuleina tai toisiinsa liittyvien ohjelmien ryhminä. Yritys voi ostaa ja ottaa käyttöön järjestelmän moduuleita tarpeen mukaan. Esimerkiksi tuotteiden jakeluun erikoistunut yritys voi jättää hankkimatta tuotantoon liittyvät moduulit. Toiminnan laajentuessa voidaan asentaa järjestelmään tarvittavat uudet sovellukset. Moduulien määrä on verrannollinen hintaan. Mitä enemmän moduuleita otetaan käyttöön, sitä kalliimmaksi järjestelmä tulee. Yleensä järjestelmien valmistajat antavat kuitenkin alennusta, jos asiakas ostaa lisämoduuleita. (Bradford 2014, 2–3.)

ERP-järjestelmä on mahdollista rakentaa kokoamalla eri valmistajien sovelluksista paketti, joka vastaa parhaiten yrityksen tarpeita (*best of breed*). Näin voidaan säästää hankintakustannuksissa ja saavuttaa kilpailuetuakin. Ohjelmistojen yhdistäminen voi aiheuttaa ongelmia, etenkin jos joudutaan hankkimaan lisää sovelluksia yrityksen toiminnan laajentuessa. Saman valmistajan integroidun ERP-järjestelmän yhtenä etuna on, että kaikkien moduulien ulkonäkö on samanlainen. Yhtenäinen käyttöliittymä ja navigointi helpottavat eri moduuleiden käyttöä. Myös ohjelmiston uusien osien opettelu on sujuvampaa verrattuna monista erilaisista ja erinäköisistä sovelluksista koostuvaan järjestelmään. (Aldrich 2015; Bradford 2014, 7.)

2.1 Järjestelmän konfigurointi

ERP-järjestelmien suunnittelu perustuu niiden valmistajien näkemykseen eri liiketoimintaprosessien parhaista käytännöistä (engl. best practices). Koska on erilaisia ja eri aloilla toimivia yrityksiä, on järjestelmissä valmiina runsaasti asetuksia, joita voidaan säätää, ja ominaisuuksia, jotka voidaan ottaa käyttöön tai pois käytöstä. Suurimmilla järjestelmien valmistajilla on tuhansia eri aloille suunniteltuja parhaita käytäntöjä ohjelmoituna ohjelmistoihinsa. (Bradford 2015, 6–7, 105.) Ennen kuin järjestelmä voidaan ottaa käyttöön, on myös tehtävä monenlaisia pakollisia perusmäärityksiä, joita ovat esimerkiksi maat,

alueet, valuutat ja kieli. Yleisten asetusten lisäksi jokaiseen moduuliin on tehtävä omat määrityksensä.

Konfiguroinnissa ei tehdä muutoksia ydinohjelmiston koodiin, vaan järjestelmän piirteitä ja ominaisuuksia mukautetaan vastaamaan yrityksen toimintatapoja käyttämällä ohjelmiston valmistajan antamia vaihtoehtoja ja mahdollisuuksia. ERP-järjestelmän konfigurointi on prosessi, johon yritykset saattavat käyttää hyvin paljon aikaa järjestelmän käyttöönoton yhteydessä. Erilaisia konfigurointipäätöksiä saatetaan joutua tekemään tuhansia. Esimerkiksi Dell Computers käytti yli vuoden pelkkään konfigurointiin, kun yritys otti käyttöön SAP ERP -järjestelmän. (Bradford 2015, 105.)

ERP-järjestelmien tarjoamat konfigurointivaihtoehdot riittävät harvojen yritysten tarpeisiin. Vaikka ohjelmiston asennus sellaisenaan (ns. vanilja-asennus) antaisi järjestelmän käyttöönotolle paremmat mahdollisuudet pysyä aikataulussa ja budjetissa, helpottaisi tulevaisuuden järjestelmäpäivityksiä ja vähentäisi ylläpitokustannuksia, suurin osa yrityksistä tekee myös muita muutoksia järjestelmään. (Aslam ym. 2014, 1.)

2.2 Järjestelmän kustomointi

Kun puhutaan ERP-ohjelmistojen muokkaamisesta ja mukauttamisesta, eri termien käyttö on kirjavaa ja vaihtelevaa. Yhtenäistä tapaa ei yrityksistä huolimatta tunnu löytyvän. Kustomointi (engl. customization) on yleisimmin käytetty termi, joka lähteestä riippuen voi tarkoittaa konfigurointia, tietokannan taulujen muokkaamista, lisäosien käyttämistä, ydinohjelmiston koodin muokkaamista tai koodilisäysten ja -laajennusten tekemistä koskematta ydinkoodiin.

ERP-ohjelmistojen kehityksen myötä niihin on lisätty säätömahdollisuuksia ja ominaisuuksia, joita aikaisemmin ei voitu saada käyttöön koskematta ydinkoodiin tai tietokannan tauluihin. Yhä edelleen kustomoinniksi nimitetään kuitenkin selkeästi konfigurointiin liittyvää ohjelmiston mukauttamista.

Käytän tässä työssä termiä kustomointi sen lyhyyden vuoksi ja tarkoitan sillä sellaisia ohjelmiston koodiin tehtäviä lisäyksiä, laajennuksia tai poistoja, joilla ei muuteta ydinohjelmiston koodia tai rakennetta. Ohjelmiston muokkaamisella (engl. modification) tarkoitan suoraan ydinohjelmiston koodiin tai rakenteisiin tehtyjä muutoksia.

Ohjelmiston kustomointi

ERP-järjestelmän käyttöönotto on monimutkainen prosessi ja vie paljon aikaa. Data ja informaatio täytyy siirtää vanhoista järjestelmistä uuteen, toimintoja täytyy testata ja vastaan tulevia ongelmia ratkoa. Ohjelmistojen lisäksi myös tietokonelaitteistot saatetaan uusia kokonaan.

Teknisten ongelmien lisäksi työntekijöiden motivaatio ja osallistuminen voi aiheuttaa murheita. Henkilökuntaa ei kuunnella ennen kuin päätökset järjestelmästä tehdään, eikä heille anneta riittävää koulutusta. Työntekijät saattavat olla hyvin tyytyväisiä vanhoihin tapoihin ja järjestelmiin, joilla asioita on hoidettu vuosia ja vuosikymmeniä. Myös työpaikkojen puolesta pelätään. (Bradford 2015, 8.)

Yritykset saattavat joutua muuttamaan hyvinkin paljon liiketoimintaprosessejaan ja toimintatapojaan vastaamaan uutta ERP-järjestelmää. Samalla voidaan menettää omat, ainutlaatuiset piirteet, joilla bisnestä on tehty ja pärjätty kilpailussa. Jos yrityksessä koetaan, että kaikki ERP-järjestelmiin ohjelmoidut parhaat käytännöt eivät sovi yrityksen toimintatapoihin, vaihtoehtona on muokata ERP-ohjelmistoa niin, että se mukautuu yrityksen toimintaan. (Bradford 2015, 9.)

Kustomoinnissa ERP-sovellusten koodiin tehdään lisäyksiä, laajennuksia tai poistoja, joilla ei kuitenkaan muuteta suoraan ydinohjelmiston koodia tai rakenteita. Eri tyyppisiä kustomoinnin tarpeeseen vaikuttavia tekijöitä on lyhyesti kuvattuna taulukossa 1.

Taulukko 1. Kustomoinnin ajurit (Aslam ym. 2014).

Tyyppi	Lyhyt selitys
Erottautuminen	ERP-paketit ovat yleisiä tuotteita. Organisaatiot tekevät muutoksia saadakseen kilpailuetua tai erottuakseen valtavirrasta.
Vaatimukset	ERP-paketissa ei ole vaadittuja toimintoja. Yrityksellä saattaa olla esimerkiksi erityinen hintarakenne tuotteessa tai alan toimintatavat poikkeavat tavanomaisesta. ERP-järjestelmän toimintoja laajennetaan lisäämällä puuttuvat ominaisuudet.
Lait ja säädökset	Organisaation täytyy kustomoida toimintoja, jotta erilaisten lakien, säädösten ja standardien vaatimukset täyttyvät.
Vastarinta	Käyttäjät eivät halua muuttaa entisiä toimintatapoja. Myös pelätään, että henkilökuntaa supistetaan. Vastarinta voi pakottaa tekemään muutoksia ohjelmistoon.
Perinteet	Organisaatio haluaa säilyttää joitain perinteisiä työskentelytapojaan.
Hyödyt	Järjestelmää halutaan kustomoida, koska siitä on hyötyä. Esimerkiksi käyttöliittymiä voidaan muokata toiminnan tehostamiseksi.

Tavallisimpia muokkauskohteita ovat erilaiset raportit, näkymät, lomakkeet ja työnkulut. Esimerkiksi lasku- ja tarjouspohjien ulkoasu muutetaan vastaamaan yrityksen visuaalista tyyliä tai raportit muokataan johdon haluamaan muotoon lisäämällä niihin tarvittavaa dataa. Lomakkeisiin voidaan lisätä uusia kenttiä tiedon syöttämistä varten tai niitä voidaan yksinkertaistaa poistamalla turhia kenttiä. Samaan tapaan voidaan muokata käyttöliittymiä toiminnan tehostamiseksi. (Bradford 2015, 31–32,106.) Työkulkujen mukauttaminen on nykyisissä ERP-järjestelmissä liitetty jo pääosin konfigurointiin, työnkulkutasoihin tai työnkulkumoottoreihin (Uppström ym. 2015).

Jos vanhoista järjestelmistä halutaan säilyttää osia tai uudessa järjestelmässä on eri ERP-valmistajien moduuleita, saatetaan joutua kehittämään uusia rajapintoja tai ohjelmia, joiden avulla dataa voidaan poimia, siirtää ja ladata tietokantojen välillä. Dataa voidaan joutua myös muokkaamaan muodosta toiseen, kun sitä siirretään. (Bradford 2015, 106.)

ERP-järjestelmien ydinkoodissa voi olla määritelty tilanteita (*customer exits*), joissa käyttäjä voi lisätä toiminnallisuutta joihinkin ohjelmiston toimintoihin muuttamatta varsinaista koodia. Esimerkiksi pudotusvalikoihin voidaan lisätä kohteita, joihin voidaan liittää tiettyjä toimintoja, ja sovellusikkunoihin voidaan lisätä kenttiä tai ali-ikkunoita. (Souza 2013; Bradford 2015, 107.)

Kustomointi on aina riski, vaikka lähdekoodiin ei kosketakaan. Siinä kuitenkin muutetaan ohjelman toimintaa. Sen vuoksi testaaminen olisi hyvin tärkeää, mutta usein kiireen vuoksi se jää puolitiehen. Kustomointi voi olla myös kallista, koska teknisten ja toiminnallisten määrittelyjen sekä koodin kirjoittaminen ja bugien testaaminen vie aikaa. Harvoilla yrityksillä on koodareita valmiina palkkalistoillaan. (Bradford 2015, 107.)

Kustomointi saattaa aiheuttaa ongelmia ohjelmiston ylläpidossa ja tukeen liittyvissä asioissa. Ohjelmistopäivitykset saattavat muuttaa järjestelmää niin, että kustomoinneissa käytetty koodi täytyy kirjoittaa ainakin osittain uudestaan. Uuden ERP-järjestelmän käyttöönotossa ohjelmiston virittely ei myöskään saisi viedä huomiota muista, jopa tärkeämmistä asioista, kuten muutosjohtamisesta ja henkilökunnan koulutuksesta (Bradford 2015, 107.)

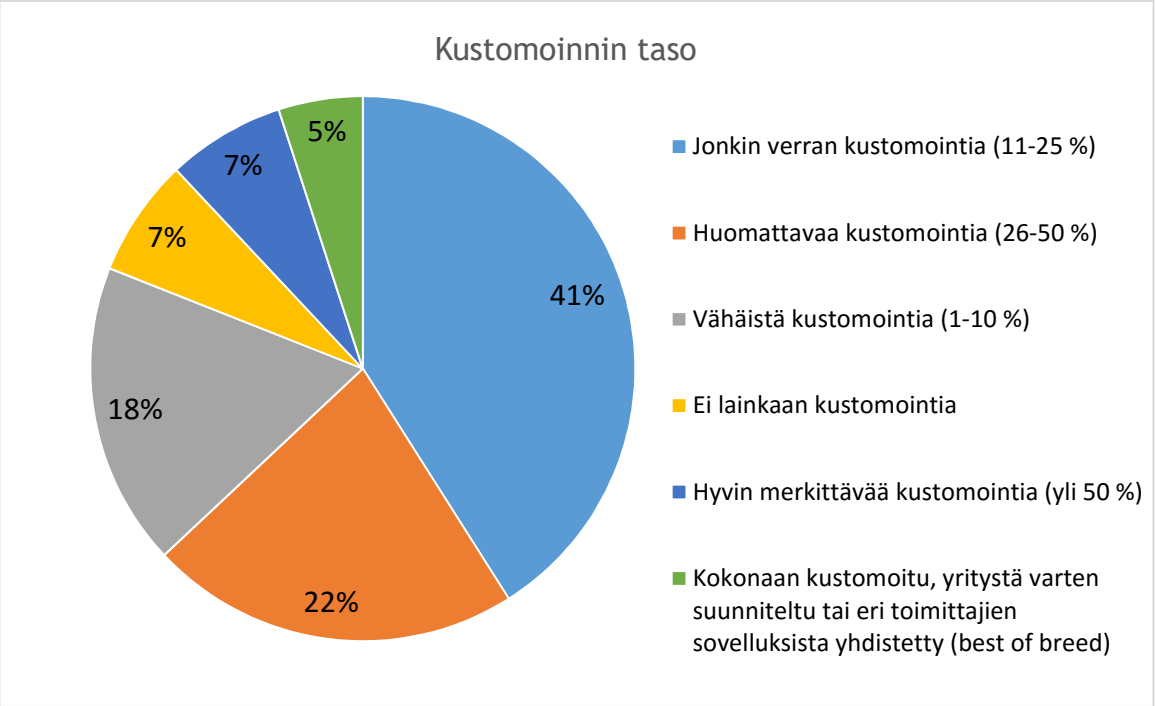
Joskus kustomointia on pakko tehdä. Ohjelmistosta ei välttämättä löydy valmiina kaikkien maiden lakeja ja asetuksia tai kansainvälisiä säädöksiä vastaavia ominaisuuksia. Yrityksen toiminta saattaa myös olla niin erikoistunutta, että järjestelmää joudutaan muokkaamaan ja räätälöimään varta vasten yritystä varten.

Viimeistään silloin, kun muokataan ohjelmiston lähdekoodia tai ydinrakenteita, kuten tietokannan relaatiotauluja, kehittäjien tulisi koodauksen lisäksi ymmärtää myös ERP-järjestelmä ja siihen liittyvä liiketoiminnan hallinta kokonaisuutena. Muutokset yhdessä moduulissa voivat vaikuttaa myös muihin moduuleihin, jolloin järjestelmän koko perustoiminta voi muuttua. Etukäteen huolellisesti tehty ja kalliskin suunnittelu voi tulla halvemmaksi kuin yritykset muokata järjestelmää sen jälkeen, kun se on jo toiminnassa ja data on syötetty tietokantaan.

Kustomoinnin taso organisaatioissa

Panorama Consulting Solutions julkaisee vuosittain raportin, jossa tutkitaan erilaisia ERP-järjestelmän käyttöönottoon liittyviä asioita. Tällaisia ovat esimerkiksi järjestelmän valinta, kustannukset, tyytyväisyys projektin lopputulokseen, projektin kesto ja kustomointi. Vuoden 2015 raportin tiedot kerättiin vuoden ajalta, ja tutkimukseen osallistui 562 organisaatiota ympäri maailmaa. Organisaatioiden koot ja toimialat vaihtelivat.

Kuviosta 1 näkyy ERP-ohjelmiston kustomoinnin taso vertailussa mukana olleilla yrityksillä. Vain 7 % ei ole muokannut mitään ohjelmistossa. Suurin osa yrityksistä, eli 63 %, on tehnyt jonkin verran tai huomattavia muutoksia järjestelmään. Vähäistä kustomointia on tehnyt noin viidennes yrityksistä. Kustomoinnilla tarkoitetaan tutkimuksessa ohjelmistokoodiin tehtyjä muutoksia tai lisäyksiä, ei vanilja-asennukseen liittyvää konfigurointia tai personointia.



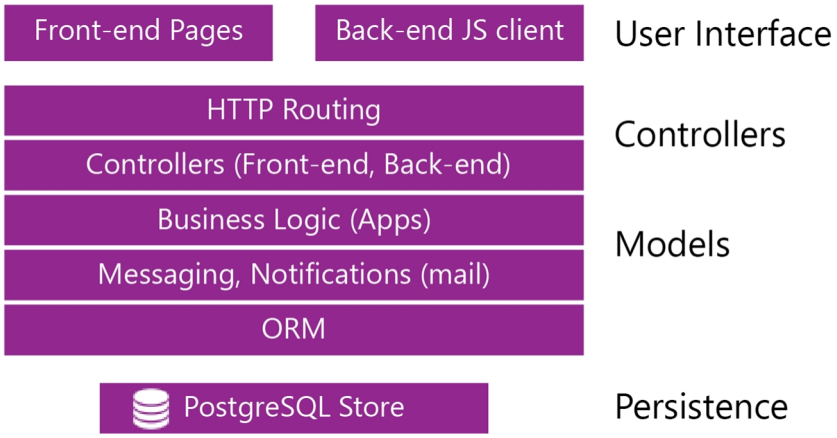
Kuvio 1. Kustomoinnin taso vertailluissa yrityksissä (Panorama Consulting Solutions 2015, 8).

Ohjelmiston kustomoinnista johtuvat odottamattomat kustannukset voivat johtaa ERP-projektin budjetin ylittymiseen. Hyvin tehdyllä vaatimusmäärittelyllä ja mallinnuksella voidaan varmistaa, että valittava ERP-järjestelmä on yhdenmukainen yrityksen liiketoimintaprosessien kanssa. Huolellinen suunnittelu ennen järjestelmän valintaa vähentää tarvetta kustomointiin. (Panorama Consulting Solutions 2015, 8.)

Samalla kun ERP-järjestelmän muokkaamisen tarve on kasvanut, myös tyytymättömyys projektien onnistumiseen on lisääntynyt. Vaikka suurin osa organisaatioista aloittaa ERP-projektinsa olettaen käyttävänsä ohjelmistoa sellaisenaan, ne joutuvat kuitenkin tekemään muutoksia järjestelmän toimintoihin. Tämä saattaa osittain selittää, miksi ERP-projektit koetaan yhä useammin myös epäonnistuneiksi. (Kimberling 2015.)

3 ODOON RAKENNE

Odoon rakenne perustuu kerrosarkkitehtuuriin ja *Model-View-Controller*-arkkitehtuurimalliin, jossa ohjelmisto koostuu eri käsitetasoilla olevista loogisesti yhtenäisistä kerroksista (Kuva 1).



Kuva 1. Odoon arkkitehtuuri (Dony 2014, 14).

PostgreSQL-tietokantapalvelin sisältää kaikki Odoon tietokannat, joihin tallennetaan sovellusdata ja suurin osa järjestelmän konfigurointielementeistä. Odoo-palvelin muodostaa järjestelmän ydinkerroksen, joka sisältää kaiken sovellus- ja liiketoimintalogiikan ja johon moduulit asennetaan. Sovellusten Python-mallien oliot tallennetaan relaatitietokantaan käyttämällä O*bject-Relational Mapping* -ohjelmistokehystä (ORM). Käyttöliittymäkerroksen asiakas on selaimessa toimiva JavaScript-sovellus, joka kommunikoi Odoo-palvelimen web-kerroksen kanssa. (Odoo S.A. 2015j.)

3.1 Model-View-Controller (MVC)

Model-View-Controller-mallissa ohjelmiston tai verkkopalvelun arkkitehtuuri on jaettu kolmentyyppisiin osiin: malleihin, näkymiin ja kontrollereihin. MVC-mallista on olemassa erilaisia versioita, mutta perusajatuksena on erottaa käyttöliittymä sovelluslogiikasta ja tietomallia käsittelevästä logiikasta. MVC-mallin toiminta sisältää useampia suunnittelu-

malleja (engl. design patterns). Observer-, Composite- ja Strategy-suunnittelumallit toteuttavat pääasiassa MVC-luokkien väliset suhteet, mutta myös Factory Method- ja Decorator-suunnittelumalleja käytetään (Gamma ym. 1998, 16).

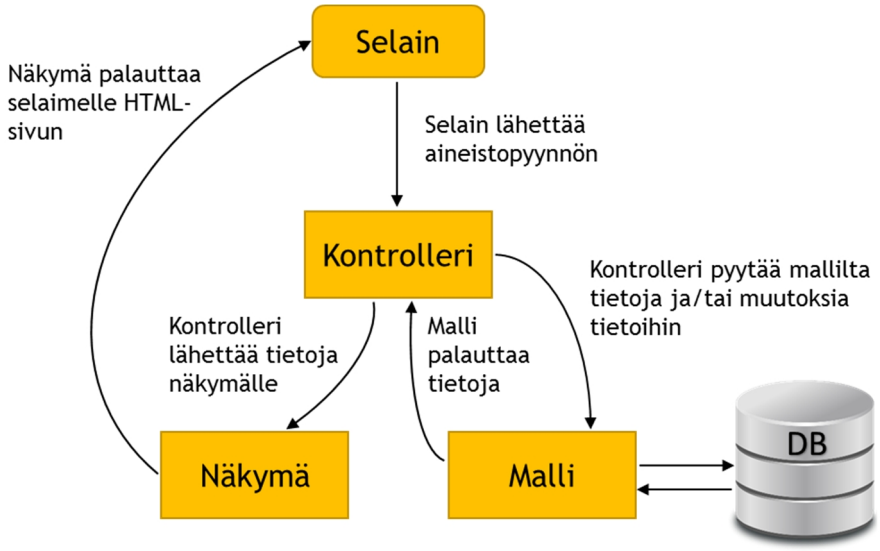
Malli on ydinkomponentti ja edustaa tietosisältöä. Se hallitsee sovelluksen logiikkaa, dataa, tilaa ja sovellusoperaatioita. Näkymä on visuaalinen esitys mallista. Näkymiä voivat olla esimerkiksi tietokoneen tai kännykän graafinen käyttöliittymä, pdf-dokumentti tai esitysdiagrammi. Näkymä vain esittää dataa eikä käsittele sitä. Kontrolleri toimii linkkinä tai sovittimena mallin ja näkymän välillä, ja näiden kommunikointi tapahtuu kontrollerin välityksellä. (Kasampalis 2015, 93.)

Tyypillinen vuorovaikutus alkaa, kun käyttäjä aktivoi näytön esimerkiksi napsauttamalla nappia, kirjoittamalla näppäimistöllä tai koskettamalla näyttöruutua. Näytön vastuulla ei ole tietää mitä seuraavaksi tehdään, vaan se välittää informaation kontrollerille. Kontrolleri käsittelee käyttäjän syötteen ja on vuorovaikutuksessa mallin kanssa. Malli toteuttaa kaikki tarvittavat toiminnot, tilan muutokset ja datan validoinnin sekä ilmoittaa kontrollerille mitä tulee tehdä. Kontrolleri ohjaa näkymän päivittymään muuttuneeseen tilaan sopivaksi. (Kasampalis 2015, 94.)

MVC erottaa näkymät ja mallit muodostamalla niiden välille rekisteröitymiseen ja ilmoittamiseen perustuvan protokollan. Näkymät rekisteröityvät tietyn mallin tarkkailijoiksi. Mallin tilan muuttuessa kontrolleri ilmoittaa muutoksesta mallin tarkkailijoille, jotka päivittävät itsensä vastaamaan mallin tilaa. (Gamma ym. 1998, 14–16.)

Koska näkymä on erotettu mallista, eikä mallilla ole suoraa riippuvuutta näkymään, samasta mallista voidaan esittää useita näkymiä samaan aikaan. Näkymät voivat sisältyä myös toisiin näkymiin. Samaa mallikoodia voidaan käyttää myöskin erilaisissa käyttöliittymissä. Jopa yksittäisessä web-käyttöliittymässä voi olla erilaisia asiakassivuja eri kohdissa sovellusta. Sovelluslogiikan ja esitystason erottaminen antaa mahdollisuuden muuttaa näkymiä koskematta malliin ja päinvastoin. Myös sovelluksen testaaminen ja ylläpitäminen tulee helpommaksi. (Fowler ym. 2002, 300–301.)

Verkkopalveluissa sisältö on tallennettu yleensä tietokantaan tai XML-tiedostoihin, joiden käsittelyyn liittyvä koodi sisältyy malleihin. Malliluokat abstrahoivat tietokannan ohjelmointirajapinnaksi, jota muu koodi käyttää. Tietokohteisiin liittyvät toiminnot ovat mallien luokkien metodeissa tai funktioissa. Malleissa ei koskaan käsitellä HTML-koodia tai selaimen lähettämiä pyyntöjä. (Arkkitehtuuri ja MVC 2014.)



Kuva 2. MVC-mallin vastuunjako verkkopalvelussa (mukaillen Arkkitehtuuri ja MVC 2014).

HTML-koodi on näkymissä, joissa yleensä käytetään *template*- eli sivupohjatiedostoja. Tietokantaa ei käsitellä näkymissä suoraan, vaan tiedot haetaan ja näytetään käyttämällä sovitun nimisiä muuttujia, joiden koodi on upotettu templaattien HTML-koodin sekaan. Näkymät eivät tiedä mistä sisältö muuttujiin tulee, vaan kontrollerit huolehtivat datan sijoittamisesta oikeisiin muuttujiin. (Arkkitehtuuri ja MVC 2014.)

Perinteisissä verkkopalveluissa käytetään HTTP-pyyntöjä, kun kommunikoidaan mallin kanssa. Kontrollerit ottavat vastaan selaimen lähettämät GET- ja POST-parametrit ja käyttävät mallia tiedon käsittelyyn. Kontrollerit eivät käsittele koskaan tietokantaa suoraan, eivätkä ne lähetä selaimelle HTML:ää tai muuta sisältöä. (Arkkitehtuuri ja MVC 2014.)

3.2 Odoon moduulit

Odoon järjestelmästä puhuttaessa voidaan erottaa sovellukset ja moduulit. Odoon sovellukset eivät poikkea rakenteeltaan tavallisista moduuleista, mutta toiminnallisesti ne poikkeavat siten, että niillä toteutetaan järjestelmän keskeiset toiminnot ja elementit, kuten esimerkiksi henkilöstön- ja myynninhallinta. Näihin ydintoimintoihin voidaan muilla moduuleilla lisätä erilaisia ominaisuuksia ja toimintoja. Moduulit ovat siis Odoon sovellusten rakenneosia, joilla voidaan lisätä, muokata tai muuntaa Odoon ominaisuuksia. (Reis 2015, 18.)

Odoon periytymismekanismit antavat mahdollisuuden tehdä laajennuksia tai muutoksia olemassa oleviin moduuleihin ilman, että muokataan suoraan niiden koodia. Periytymistä voidaan käyttää kaikilla tasoilla: malleissa, sovelluslogiikassa ja näkymissä. (Reis 2015, 19.)

Moduulit koostuvat erilaisista elementeistä, joita voivat olla

- liiketoimintaoliot, jotka määritellään Python-luokissa
- XML- tai CSV-tiedostot, joissa määritellään muun muassa näkymät, työnkulut ja konfigurointidata
- web-kontrollerit, jotka käsittelevät selainten pyyntöjä
- staattinen data, kuten kuvat sekä CSS- ja JavaScript-tiedostot, joita käytetään web-käyttöliittymässä tai www-sivuilla (Odoo S.A. 2015a).

Moduulien rakenne

Moduulin nimeä käytetään Python-tunnisteena. Jos nimen haluaa vaihtaa, tunniste täytyy muuttaa myös kaikkiin moduulin sisältämiin tiedostoihin. Nimen tulisi alkaa kirjaimella, ja se voi sisältää vain kirjaimia, numeroita ja alaviivan.

Odoo-moduulin sisältö on tavallisesti järjestetty selkeyden vuoksi muutamaan hakemistoon (Kuva 3):

- *controllers* sisältää ohjaimet, jotka tulkitsevat selaimen pyyntöjä ja palauttavat dataa.
- *data* sisältää demo- ja data-XML-tiedostot.
- *models* sisältää tiedostot mallien määrittelyyn.
- *static* sisältää web-käyttöliittymän tai -sivujen sisältöön liittyviä hakemistoja ja tiedostoja, kuten JavaScript-, CSS- ja kuvatiedostoja.
- *security* sisältää käyttöoikeuksien määrittelytiedostot.
- *views* sisältää näkymät ja XML-templaatit. (Odoo S.A. 2015b.)

```
addons/<my_module_name>/
|-- __init__.py
|-- __openerp__.py
|-- controllers/
|   |-- __init__.py
|   `-- main.py
|-- data/
|   |-- <main_model>_data.xml
|   `-- <inherited_main_model>_demo.xml
|-- models/
|   |-- __init__.py
|   |-- <main_model>.py
|   `-- <inherited_main_model>.py
|-- security/
|   |-- ir.model.access.csv
|   `-- <main_model>_security.xml
|-- static/
|   |-- img/
|   |   |-- my_little_kitten.png
|   |   `-- troll.jpg
|   |-- lib/
|   |   `-- external_lib/
|   `-- src/
|       |-- js/
|       |   `-- <my_module_name>.js
|       |-- css/
|       |   `-- <my_module_name>.css
|       |-- less/
|       |   `-- <my_module_name>.less
|       `-- xml/
|           `-- <my_module_name>.xml
`-- views/
    |-- <main_model>_templates.xml
    |-- <main_model>_views.xml
    |-- <inherited_main_model>_templates.xml
    `-- <inherited_main_model>_views.xml
```

Kuva 3. Esimerkki moduulin rakenteesta (Odoo S.A, 2015b).

Manifest-tiedosto *__openerp__.py* on Python-tietohakemisto (*dictionary*), jossa olevat avain-arvo-parit määrittävät Python-paketin Odoon moduuliksi ja sisältävät metadatan. Ainoa pakollinen attribuutti on moduulin nimi *name*, mutta moduulin toiminnan kannalta olennaisia ovat myös *depends* ja *data*. Kuvassa 4 on esimerkki tiedostosta.

Kuten nimi kertoo, *depends*-avaimen arvoilla määritellään muut moduulit, jotka on ladattava ennen kyseistä moduulia, koska joko käytetään niiden luomia ominaisuuksia tai muutetaan niiden määrittelemiä resursseja. Jos riippuvuuksia ei ole asetettu oikein, moduulin asentaminen voi epäonnistua tai aiheuttaa virheitä. *Data*-attribuutissa luetellaan datatiedostot, jotka täytyy aina ladata tai päivittää moduulin kanssa. (Odoo S.A. 2015c; Reis 2015, 20)

```
# -*- coding: utf-8 -*-
{
    'name': "Emsys Invoice",
    'summary': """
        Custom Finnish invoice and quotation template""",
    'author': "Tapio P.",
    'category': 'Uncategorized',
    'depends': [
        'base',
        'sale',
    ],
    'data': [
        'security/ir.model.access.csv',
        'views/report_emsys.xml',
        'views/layout_emsys.xml',
        'views/saleorder_emsys.xml',
    ],
    'installable':True,
    'application':True,
}
```

Kuva 4. Esimerkki __openerp__.py-tiedostosta.

Alustustiedosto *__init__.py* sisältää import-komentoja, joilla halutut Python-tiedostot, hakemistot ja alihakemistot sisällytetään Odoo-moduulin Python-paketteihin. Jokaisessa hakemistossa olevassa Python-paketissa tulee olla oma *__init__.py* tai tiedostoja ei haeta käyttöön. Import-komennot voivat Pythonissa olla melko monimutkaisia: esimerkiksi *from package import item* -komennossa *item* voi olla paketin alipaketti, Python-moduuli, funktio, luokka tai muuttuja (Odoo S.A. 2015a; Python Sofware Foundation 2015.)

Hakemistojen käyttö ei ole pakollista, eikä niitä yksinkertaisissa Odoo-moduuleissa edes tarvita. Odoossa on alikomento *odoo.py scaffold*, jolla voidaan automaattisesti luoda tyhjä moduuli, joka sisältää tarvittavat perustiedostot. Tiedostoissa on kommenteiksi muutettua esimerkkikoodia. Ainoa hakemisto, jonka *scaffold* tekee, on *security*.

Mallit

Odoon mallit (*models*) määritellään Python-luokissa, jotka kuvaavat liiketoiminnan entiteettejä ja niihin liittyviä operaatioita. Mallit tallennetaan päärekisterin dictionary-tiedostoon, jossa säilytetään viittaukset kaikkiin malliluokkiin. Mallien nimet toimivat avaimina ja tunnisteina rekisterissä ja ovat siksi tärkeitä. Mallien nimien tulee olla ainutkertaisia, ja yleisen tavan mukaan niissä käytetään pistenotaatiota ja pienaakkosia. Jos on esimerkiksi Odoo-moduuli *academy*, niin selkeyden vuoksi luokka *Teachers* tulisi nimetä *academy.teachers*. (Reis 2015, 72–73.)

Python-luokat liittyvät paikallisesti Python-tiedostoon, jossa ne määritellään. Niissä käytetyt tunnisteet ovat merkityksellisiä vain tiedostossa käytetylle koodille. Tämän vuoksi luokkatunnisteissa ei tarvitse käyttää niihin liittyvän pääsovelluksen nimeä. Samanniminen *Teachers*-luokka voi olla myös muussa Odoo-moduulissa, eikä siinä ole ristiriitaa. (Reis 2015, 73.)

Suurin osa Odoon mallien luokista periytyy *models.Model*-yliluokasta. Näiden mallien data tallennetaan pysyvästi tietokannan tietueisiin. Odoossa voidaan käyttää myös kahta muuta mallityyppiä: tilapäisiä ja abstrakteja malleja. Ohjatuissa toiminnoissa käytettävät mallit perustuvat *models.TransientModel*-luokkaan ja niiden data tallennetaan vain tilapäisesti tietokantaan. Yliluokkaan *models.AbstractModel* pohjautuvista abstrakteista luokista ei luoda tallennettavia olioita, vaan luokkia käytetään muiden luokkien yliluokkina. Perivät luokat toteuttavat abstraktien luokkien toiminnot. (Reis 2015, 73.)

Mallit konfiguroidaan niiden määrittelyssä annetuilla attribuuteilla. Tärkein ja pakollinen attribuutti on *_name*, jolla määritetään mallin nimi Odoon järjestelmässä. Itse Python-luokan nimellä ei ole merkitystä muille moduuleille. Mallin kentät (*fields*) ovat luokan attribuutteja ja niissä määritellään mitä malli voi varastoida ja minne. Kenttiä voidaan konfiguroida antamalla niille määrittelyssä parametreja. Esimerkki mallin määrittelystä ja erilaisia attribuutteja on kuvassa 5. Kenttää *name* käytetään tietueen otsikkona ja sitä käytetään esimerkiksi tietyissä näyttö- ja hakutoiminnoissa. (Odoo S.A. 2015a; Reis 2015, 24.)

```
from openerp import models, fields

class MyModel(models.Model):
    _name = 'mymodule.mymodel'
    name = fields.Char('Title', required=True)
    char = fields.Char('Char', 64) # name, maximum size
    text = fields.Text('Text')
    intg = fields.Integer('Integer')
    flot = fields.Float('Float', (16,4)) # name, digits
    time = fields.Datetime('Date and Time')
```

Kuva 5. Esimerkki mallin määrittelystä.

Relaatiokentillä voidaan linkittää saman mallin tai eri mallien välisiä tietueita. Odoossa käytettävät kentät ovat monen suhde yhteen (*Many2one*), yhden suhde moneen (*One2many*) ja monen suhde moneen (*Many2many*). Kuvassa 6 esitetyissä suhteissa on ajateltu, että jokaisella kurssilla on vain yksi opettaja, joka voi puolestaan opettaa monella kurssilla. Jokaiselle kurssille voi osallistua monta opiskelijaa, ja opiskelija voi

osallistua monelle kurssille, mikä voitaisiin puolestaan kuvata Students-luokassa. (Odoo S.A. 2015i.)

```
class Teachers(models.Model):
    _name = 'academy.teachers'
    name = fields.Char()
    course_ids = fields.One2many(comodel_name='academy.courses', \
                                 inverse_name='teacher_id', string="Courses")

class Courses(models.Model):
    _name = 'academy.courses'
    name = fields.Char()
    teacher_id = fields.Many2one(comodel_name='academy.teachers', string="Teacher")
    student_ids = fields.Many2many('academy.students', string="Students")
```

Kuva 6. Relaatiokenttiä (Odoo S.A. 2015i).

Pakollinen parametri *comodel_name* on kohdemallin nimi, ja One2many-kentässä vaadittava *inverse_name* on kentän nimi tietueessa, johon mallissa viitataan. Lisäksi voidaan antaa valinnaisia parametreja, kuten *ondelete* ja *auto_join*.

Peruskenttien arvot tallennetaan suoraan tietokantaan ja luetaan sieltä. Kenttien arvoja voidaan myös esimerkiksi laskea lennossa mallin metodin avulla. Laskettavan kentän *compute*-attribuutin arvona annetaan metodin nimi. Laskettavien kenttien riippuvuudet määritellään *depends()*-dekoraattorin parametreina (Kuva 7). Laskettua kenttää ei tallenneta tietokantaan, ellei kentän määrittelyssä käytetä attribuuttia *store=True*. (Odoo S.A. 2015a; Reis 2015, 84.)

```
amount = fields.Float(compute='_compute_amount', string='Amount', store=True)
@api.depends('product_id.price', 'quantity')
def _compute_amount(self):
    for record in self:
        record.amount = record.product_id.price * record.quantity

@api.onchange('amount', 'unit_price') # if these fields are changed, call method
def _onchange_price(self):
    self.price = self.amount * self.unit_price
    return {
        'warning': {
            'title': "Something bad happened",
            'message': "It was very bad indeed",
        }
    }
```

Kuva 7. Compute ja onchange (Odoo S.A. 2015a).

Laskennassa käytettävät kentät voivat sijaita samassa tai eri mallissa, mutta niillä täytyy olla jonkinlainen riippuvuus toisistaan. Kenttien arvojen ei tarvitse olla lukuja vaan niiden tietotyyppinä voi olla esimerkiksi C*har*, ja metodi voi palauttaa *boolean*-tyyppisen arvon.

Toisena esimerkkinä olioon lisäominaisuuksia dynaamisesti lisäävistä dekoraattoreista voidaan mainita *onhange*. Onchange-dekoraattorin metodia kutsutaan, jos määriteltyjen kenttien arvot muuttuvat. Kenttien arvojen muutoksia ei tallenneta tietokantaan, vaan kaikki työ tehdään välimuistissa, ellei lomaketta tallenneta. Kentän arvoa ei myöskään tallenneta, jos sille on annettu *readonly*-attribuutin parametrina *True*. Toisin kuin lasketta-vissa kentissä, käytettävien onchange-kenttien tulee olla samassa mallissa. On-change-metodeissa voidaan myös määritellä varoituksia käyttäjälle, jos esimerkiksi ti-lauslomaketta täytettäessä tuotetta ei ole riittävästi varastossa. (Odoo S.A. 2015a; Reis 2015, 128–129.)

Näkymät

Näkymissä määritellään, miten mallin tietueet esitetään. Odoon kolme olennaisinta nä-kymätyyppiä ovat *list*, *form* ja *search*. Muita tyyppejä ovat esimerkiksi *QWeb*, *calendar*, *graph* ja *kanban*. Jokainen näkymätyyppi edustaa erilaista visuaalisen esityksen muotoa, joita voivat olla esimerkiksi asiakaslista, tilauslomake ja diagrammi. (Odoo S.A. 2015a.)

Näkymä määritellään XML-tiedoston <record>-elementissä, joka ladataan tietokantaan, kun moduuli asennetaan (Kuva 8). Tärkein attribuutti on *arch*, joka sisältää näkymän määrittelyn. Näkymän sisältönä on XML, joten arch-kentän tyypiksi on annettava *type="xml"* tai sisältö ei rakennu oikein. Kaikki näkymät tallennetaan tietokantaan *ir.mo-del.view*-malliin. (Reis 2015, 27–28.)

```
<record id="view_partner_form_inherit_emsys_contacts" model="ir.ui.view">
  <field name="name">res.partner.form.inherit.emsys</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.view_partner_form"/>
  <field name="arch" type="xml">
    <!-- view content: <form>, <tree>, <graph>, ... -->
  </field>
</record>
```

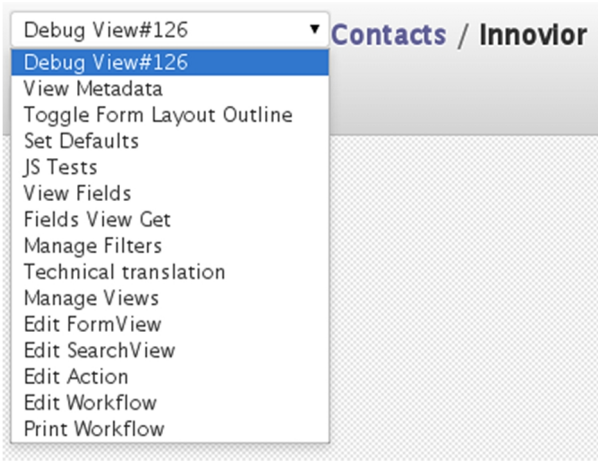
Kuva 8. Näkymän perusrakenne.

Lomakkeita käytetään yhden tietueen datan esittämiseen. Niiden juurielementtinä on <form> ja sisältö muodostetaan HTML:llä. Lisäksi käytetään rakenteellisia komponentteja, kuten *header*, *sheet*, *notebook* ja *group*, sekä semanttisia komponetteja, kuten *button* ja *field*. Etenkin painike- ja kenttäkomponettien määrittelyssä voidaan käyttää monenlaisia attribuutteja. Lomakkeiden elementtien näkyvyyttä voidaan muuttaa dynaamisesti esimerkiksi käyttäjäryhmän tai prosessin tilan mukaan. (Odoo S.A. 2015k.)

Listanäkymät ovat lomakkeita yksinkertaisempia, ja ne muodostetaan <tree>-elementin sisälle. Ne voivat sisältää kenttiä ja painikkeita, ja suurin osa niiden attribuuteista on samoja kuin lomakenäkymässä. Hakunäkymät esittävät sisältönään muista näkymistä suodatettua sisältöä. Niiden juurielementti on <search>. (Reis 2015,103–104.)

4 DEVELOPER MODE – KEHITTÄJÄTILA

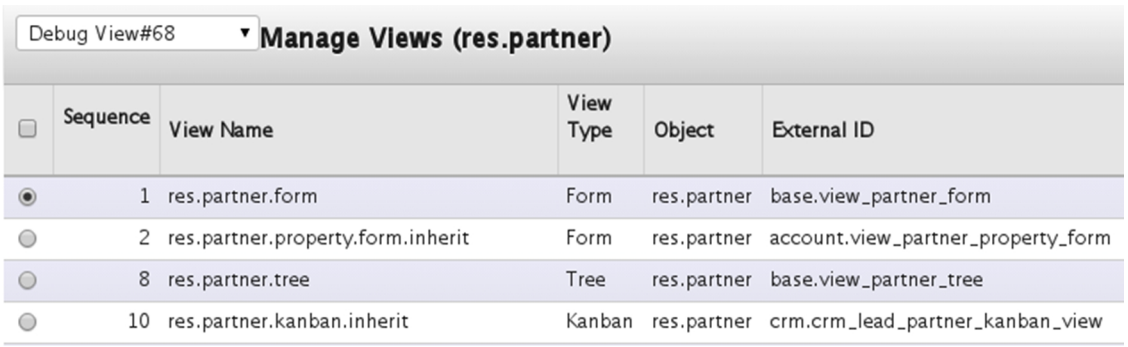
Odoon kehittäjätilassa voidaan tarkastella monia näkymiin liittyviä asioita, kuten metadataa, kenttiä, asettelua ja tehtyjä käännöksiä. Kehittäjätilassa voi tehdä JavaScript-testejä, asettaa kenttiin oletusarvoja ja lisätä erilaisia suodattimia. Lisäksi on mahdollista muokata toimintoja ja näkymiä suoraan käytön aikana. Kuvassa 9 on asiakaslomakkeeseen liittyvät vaihtoehdot.



Kuva 9. Kehittäjätilan pudotusvalikko.

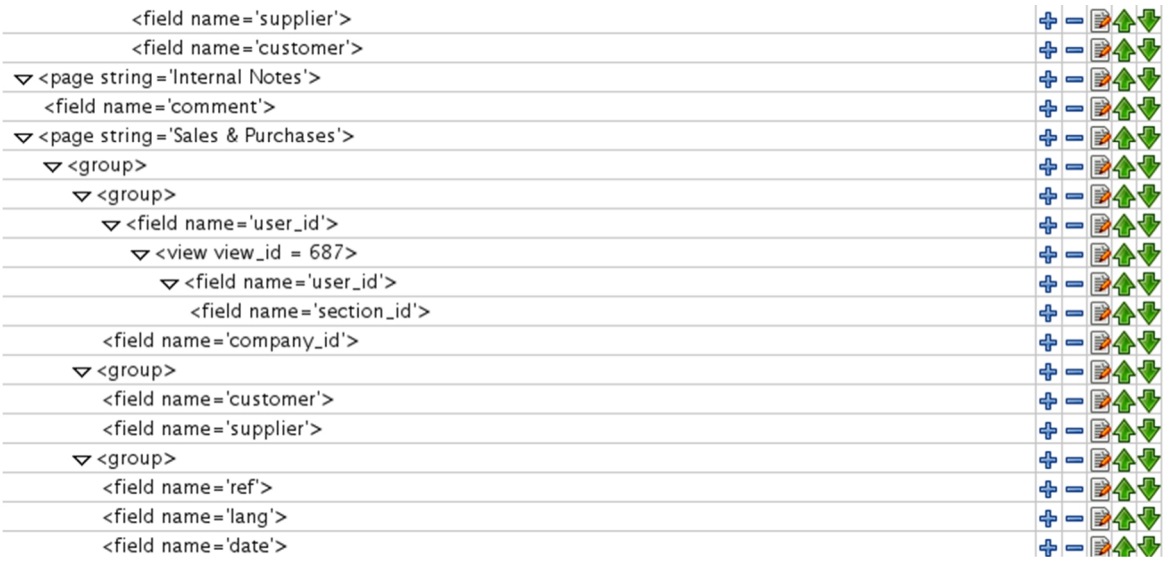
On houkuttelevaa muokata näkymiä kehittäjätilassa, johon *Manage Views* ja *Edit FormView* -vaihtoehdot antavat mahdollisuuden, mutta muutoksia tehdessä täytyy tietää mitä tekee ja ymmärtää hyvin näkymien rakenne. Monilta murheilta voi säästyä, jos pieniltäkin tuntuvat muutokset tehdään ensin kehitysversioon.

Manage Views -lomakkeelta voidaan valita näkymä, jota halutaan muokata näkymäeditorissa (Kuva 10).



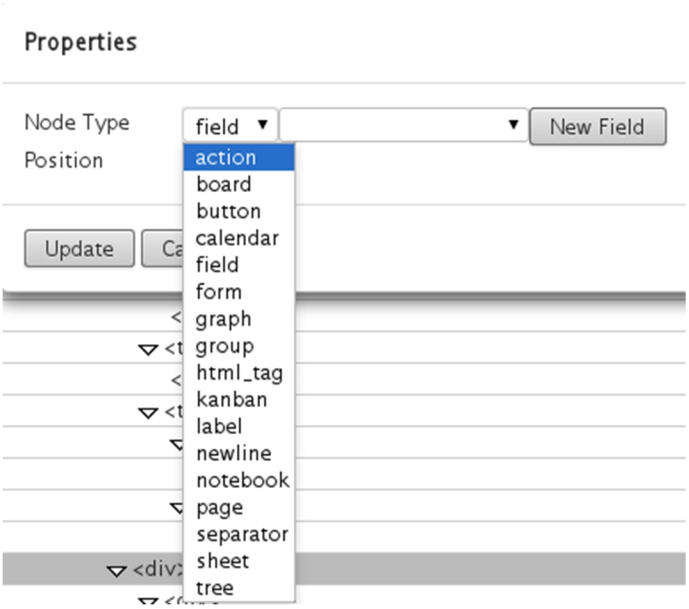
Kuva 10. Manage Views -lomake.

Näkymäeditorissa (Kuva 11) voidaan plus-nappia painamalla lisätä lomakenäkymään monenlaisia osia, kuten toimintoja, nappeja, välilehtiä, ryhmiä ja kenttiä. Osia voidaan poistaa miinus-napilla. Nuolien avulla osia voidaan järjestellä.



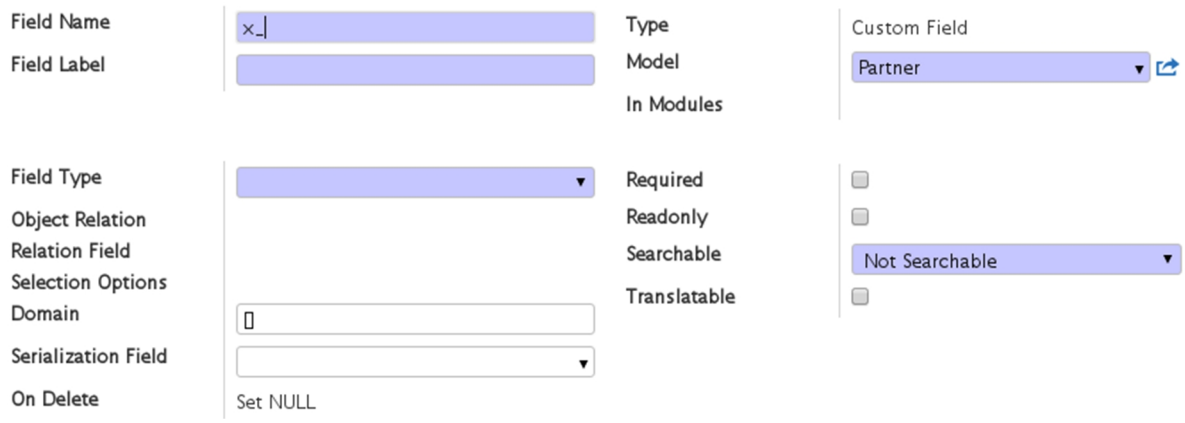
Kuva 11. Näkymäeditori – View Editor.

Uuden osan lisääminen avaa Add-valikon, josta voidaan valita näkymään lisättävä osa. Jos halutaan lisätä esimerkiksi kenttä, voidaan valita pudotusvalikosta jokin olemassa oleva tai luoda kokonaan uusi (Kuva 12).



Kuva 12. Näkymäeditorin Add-valikko.

Uusille näkymän osille voidaan määrittää attribuutteja ja niiden arvoja avautuvista valikoista. Uuden kentän lisääminen avaa kuvassa 13 näkyvän attribuuttivalikon.

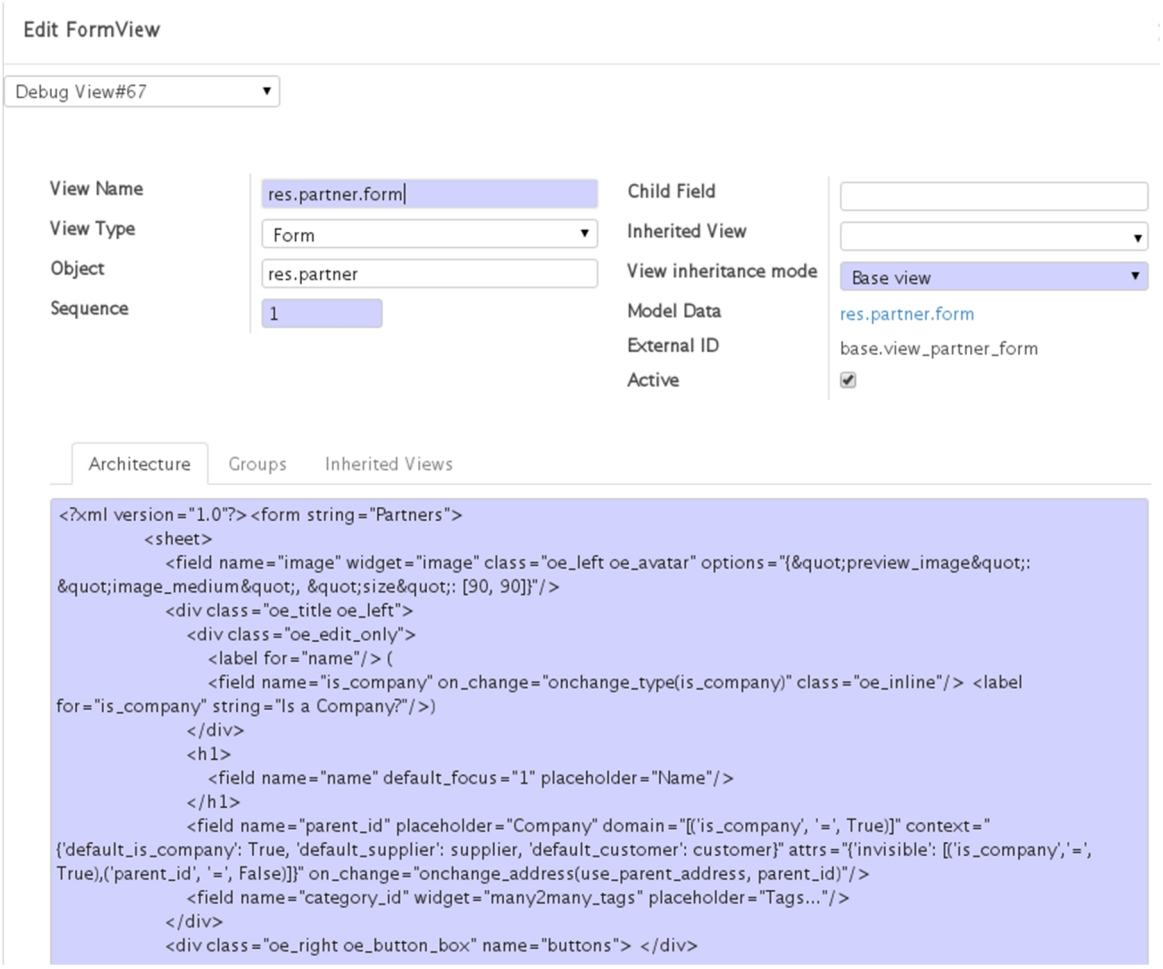


Kuva 13. Uuden kentän attribuuttivalikko.

Erityisesti näkymäeditorin käyttäminen vaatii ymmärrystä siitä, miten lomakkeen näkymä muodostuu. Kun avaa esimerkiksi *res.partner.form*-näkymän, editorissa näytetään myös kaikki perityt näkymät, joita lomakkeella saattaa olla useita. Käyttäjä saa virheellisen vaikutelman, että hän pystyy kätevästi muokkaamaan samalta editorin sivulta kaikkia näkymiä, myös perittyjä. Tämä ei pidä paikkaansa, eikä esimerkiksi lisätty kenttä tule aina toivottuun paikkaan, jos näkyy ylipäätään.

Näkymäeditori ei anna juurikaan varoituksia, joten sen avulla on mahdollista tehdä muutoksia, jotka sekoittavat lomakkeen rakenteen. Oman kokemukseni perusteella etenkin perittyjen näkymien muokkaamisella voi saada lomakkeen niin sekaisin, että virheiden korjaaminen on hyvin vaikeaa. Kehittäjätilan näkymäeditorin käyttö on mielestäni myös hankalaa ja hidasta, ja tehtyjen muutosten muistaminen voi olla vaikeaa.

Lomakenäkymäeditorilla muutosten hahmottaminen on helpompaa, koska siinä muokataan suoraan XML-koodia (Kuva 14). Itse editori toimii huonosti. Esimerkiksi rivinvaihdossa kursori menee aina rivin alkuun, eikä editori-ikkunan kokoa voi muuttaa, joten koodista saa helposti sekavan näköistä. Editori ei myöskään osaa varoittaa virheistä koodia kirjoitettaessa. Onneksi Odoo osaa tunnistaa joitain virheitä eikä anna tallentaa lomaketta, jos esimerkiksi tageja ei ole kirjoitettu oikein.



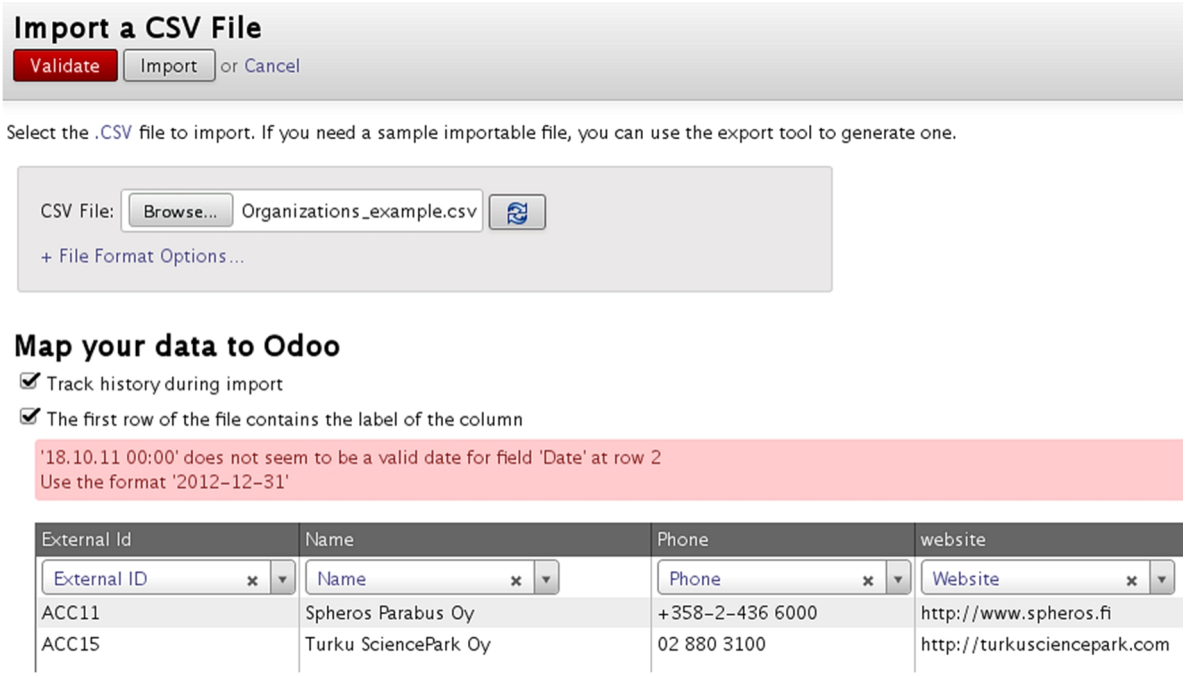
Kuva 14. Lomakenäkymäeditori – Edit FormView.

Kehittäjätilassa muokkaukset tehdään aina ainoastaan kyseiseen tietokantaan. Jos moduuli joudutaan asentamaan uudestaan, tehdyt muutokset menetetään. Muutoksia ei voi myöskään siirtää uuteen tai toiseen tietokantaan, vaan ne on tehtävä aina uudestaan. Kehittäjätila on erinomainen työkalu, kun halutaan tarkastella näkymien rakennetta, analysoida lomakkeita ja niiden osia sekä ymmärtää Odoon sovelluskehystä. Sen avulla voi nähdä helposti esimerkiksi kenttien luokat ja riippuvuudet. (Moss 2015, 364–365.)

Pientenkin muutosten tekeminen kehittäjätilassa on hankalaa ja virheiden mahdollisuus on suuri. Jos haluaa muokata näkymiä, muutoksia varten on järkevämpää tehdä moduuli.

5 ASIAKASLOMAKKEEN KUSTOMOINTI

Odoon tietokantaan voidaan tuoda CSV-tiedostossa olevaa dataa selaimen käyttöliittymän *Import a CSV File* -toiminnon avulla (Kuva 15). Ennen kuin data tallennetaan, Odoo tarkastaa jokaisen rivin ja tietueen, että tiedot ovat oikeita ja tietokannan eheys säilyy. Datan oikeellisuuden voi myös tarkastaa etukäteen painamalla Validate-nappia. En ole saanut selvitettyä, validoiko Odoo tiedot vielä uudestaan, kun Import-nappia painetaan. Datan tarkastamiseen kuluu aikaa, mutta vioittuneen tietokannan korjaaminen on varmasti ikävämpää kuin useamman tarkastuksen odottelu.



Kuva 15. Import-toiminto käyttöliittymässä.

Tuotavien kenttien nimien ei tarvitse vastata tietokannassa olevia nimiä. Pudotusvalikosta voi valita kentälle oikean nimen. Ongelmana on, että Odoo ei näytä pudotusvalikossa kenttien todellisia nimiä, vaan niille annetut kuvailevat nimet (*field labels*), joten esimerkiksi *comment*-kenttä löytyy valikosta nimellä *Notes*. Kehittäjätilassa ja XML-tiedostoissa näkyvistä kenttien nimistä ei ole siten välttämättä hyötyä, vaan valikossa näkyvistä nimistä voi joutua arvailemaan sopivaa.

Tuotavan CSV-tiedoston kenttien nimien muuttaminen todellisia nimiä vastaaviksi helpottaa datan tuomista tietokantaan. Kuvassa 15 on varoitus, että 'Date' voi sisältää vain

päivämäärän. Tässä tapauksessa täytyy CSV-tiedostoa muokata, ja muokkaamisen jälkeen päivitysnappia painamalla voidaan tiedosto ladata uudelleen. Jos kenttien nimet eivät vastaa todellisia nimiä, joudutaan pudotusvalikoista hakemaan aina uudestaan sopivat nimet, kunnes virheet on korjattu.

Ohjatun toiminnon käyttäminen datan tuomisessa helpottaa tiedon oikeellisuuden tarkistamista, mutta sitäkin käytettäessä täytyy tietokannan taulujen, kenttien ja tietueiden rakenne sekä relaatiot tuntea. Import-toimintoa ei ole tarkoitettu uusien tietokantojen tai kenttien luomiseen, eikä se osaa esimerkiksi yhdistää tauluja, ellei sille kerrota, miten se tehdään.

Odoon *Export Data* -toiminto on perusominaisuus, joka löytyy jokaisesta Odoon listanäkymästä. Tiedostojen vientimuotona voi olla CSV tai Excel. Kun listanäkymästä on valittu vietävät tietueet, avautuvasta vientilomakkeesta voidaan valita halutut kentät. Viennin tyypiksi voidaan valita *Export all Data* tai *Import-Compatible Export*, joista jälkimmäisellä voidaan varmistaa, että kentät ja niissä oleva data voidaan viedä takaisin tietokantaan. Vietyä CSV-muotoista tiedostoa voidaan myös käyttää mallina tai pohjana datan tuomisessa, jolloin yksinkertaisesti kirjoitetaan tai kopioidaan data oikeisiin kenttiin. (Reis 2015, 56–57.)

Suurien datatiedostojen tuominen Odoon Import-toiminnolla kestää kauan. Isot tiedostot kannattaakin jakaa osiin. Tietysti Odoon PostgreSQL-tietokantaan voi halutessaan siirtää dataa suoraan, mikä on huomattavasti nopeampaa. On olemassa mallikomentosarjoja, joiden avulla voi tuoda dataa esimerkiksi CSV-tiedostoista.

5.1 Asiakastietojen tuominen

Tehtävänä oli tuoda organisaatio- ja henkilötietoja Odoon asiakastietoihin. Organisaatioita oli 754 ja henkilöitä 859. Organisaatiotiedostossa oli 24 tuotavaa kenttää, joista puolet oli sellaisia, joita ei ollut tietokannassa eikä käyttöliittymän lomakkeissa. Henkilötiedoissa oli 25 kenttää, joista 8 oli uutta. Tuotavat tiedostot olivat Vtiger CRM -asiakkuudenhallintaohjelmistosta ladattuja CSV-tiedostoja.

Eri ohjelmistojen ja tietokantojen tavat ja kyvyt käsitellä merkistökoodauksia vaihtelevat, mikä aiheuttaa ongelmia datan siirtämisessä. Vtiger-tiedostoista löytyi esimerkiksi kolmenlaista koodausta ääkkösille. Tämänkaltaiset virheet on helpointa korjata ennen tietojen tuomista, joten korjasin ne ja muokkasin myös esimerkiksi päivämäärät Odoon

tietokannassa käytettävään muotoon. Olisi houkuttelevaa käyttää Exceliä ääkkösten korjaamiseen, mutta kannattaa muistaa, että Excel ei osaa edelleenkään käyttää utf-8-koodausta CSV-tiedostoissa.

Jostain syystä Odoossa ei ole erikseen kenttiä suku- ja etunimelle, vaan nimelle on oletuksena yksi kenttä. Erillisten nimikenttien luomiseen on olemassa valmis moduuli, joka myös jakaa yhdessä nimikentässä olevan merkkijonon kahteen kenttään. Oletuksena on tosin, että nimi on muodossa sukunimi–etunimi.

Tämä aiheutti hieman ongelmia, koska tietokannassa oli valmiina nimiä muodossa etunimi–sukunimi, jolloin nimet menevät vääriin kenttiin. Muokkasin moduulin Python-koodia niin, että ohjelma erotti nimet oikeisiin kenttiin. Jos tuotavissa CSV-tiedostoissa olevia nimiä täytyy yhdistää tai erottaa etukäteen, sen voi sujuvasti tehdä esimerkiksi Excelissä.

5.2 Periytyminen Odoossa

Mallien periytyminen

Odoossa on kolme mekanismia, joilla voidaan laajentaa malleja moduulien avulla:

- Muissa moduuleissa määriteltyjä malleja laajennetaan luomatta uutta mallia.
- Olemassa olevasta mallista luodaan uusi malli, johon lisätään informaatiota muuttamatta alkuperäistä moduulia.
- Joitain mallin kenttiä voidaan välittää sen sisältämille tietueille. (Odoo S.A. 2015e.)

Kun laajennetaan olemassa olevaa mallia, käytetään Python-luokassa attribuuttia *_inherit*. Uusi luokka perii perittävän mallin kaikki piirteet, eikä luokkaan tarvitse määritellä kuin muutokset, jotka halutaan tehdä. (Reis 2015, 39.)

Käyttämällä pelkkää *_inherit*-attribuuttia ei luoda uutta mallia, vaan laajennetaan olemassa olevaa mallia esimerkiksi muuttamalla sen määrityksiä tai lisäämällä siihen uusia kenttiä tai metodeja. Kaikki data tallennetaan samaan tauluun tietokannassa. Malli on valmiiksi liitetty näkymään tai näkymiin. (Odoo S.A. 2015e.)

Kun käytetään luokassa sekä *_inherit*- että *_name*-attribuutteja, Odoo luo kokonaan uuden mallin, jossa on kopioituna kaikki pohjana olevan mallin piirteet. Uuden mallin data tallennetaan omaan tauluunsa. Näin luotu uusi malli-olio ei ole liitettynä mihinkään näkymä-olioon, joten olemassa olevat näkymät jättävät luokan huomioimatta. (Odoo S.A. 2015e.)

Delegointiperinnässä attribuutteina ovat *_inherits* (monikko) ja *_name*. Tässä mekanismissa moniperintä on mahdollista, mutta periä voidaan vain kenttiä, ei metodeja. Jokainen kenttä, jota uudessa mallissa ei ole, delegoidaan perityistä malleista. Data tallennetaan uuteen tauluun, eikä mallilla ole vaikutusta näkymiin. (Odoo S.A. 2015e.)

Näkymien periytyminen

Sen sijaan että muokattaisiin olemassa olevia näkymiä, Odoossa voidaan käyttää näkymien periytymistä, jossa lapsinäkymien avulla lisätään, poistetaan tai piilotetaan juuri- tai isänäkymien sisältöä. Lapsinäkymässä käytetään *inherit_id*-kenttää ja sen *ref*-attribuuttia viittauksena isänäkymään. Perittävän näkymän sisältöä voidaan samalla kertaa muuttaa usealla <*xpath*>-elementillä. Liitteessä 1 on esimerkki useamman <xpath>-elementin kenttien lisäämisestä isänäkymään. (Odoo S.A. 2015a.)

Isänäkymän elementti valitaan XPath-lausekkeella, jonka *position*-attribuutin arvon perusteella sijoitetaan lapsinäkymän sisältö. Attribuutin arvoina voidaan antaa *inside*, *replace*, *before*, *after* ja *attributes*, joista viimeksi mainitulla voidaan muokata vastaavan elementin attribuutteja. (Odoo S.A. 2015a.)

XML Path Language, XPath, on kieli, jonka lausekkeiden sijaintipolkuja käytetään osoittamaan XML-dokumentin puumallin solmuihin. XPath-lausekkeen vinoviivalla erotettujen peräkkäisten askelten avulla päästään haluttuihin elementteihin. (Silberschatz 2011, 999.)

5.3 Moduuli asiakaslomakkeen kustomointiin

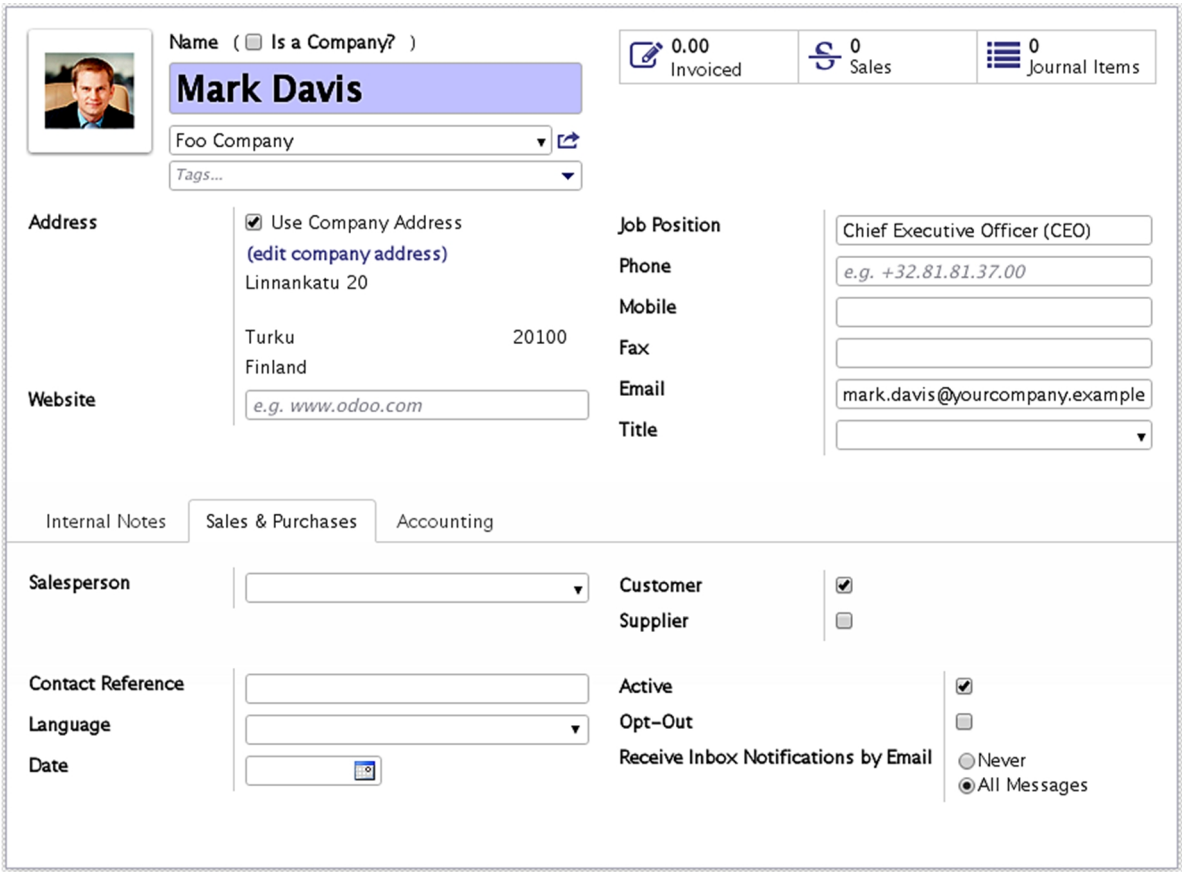
Uusia välilehtiä ja kenttiä voidaan lisätä kehittäjätilassa. Kehittäjätilassa tehdyt muokkaukset ja muutokset ovat kuitenkin käytettävissä vain siinä tietokannassa, johon ne tehdään. Muutoksia ei voi siirtää uuteen tai toiseen tietokantaan, vaan ne on tehtävä aina

uudestaan. Tämän vuoksi muutoksia varten kannattaa tehdä moduuli, jolloin myös lopputuloksen kontrollointi on helpompaa. (Moss 2015, 364.)

Kenttien lisääminen näkymään

Odoossa käytetään samaa *partner.form*-lomaketta organisaatio- ja henkilötiedoille. Lomakkeessa näkyvät tiedot ovat osittain erilaisia sen mukaan, onko lomakkeesta rastitettu kohta *Is a Company?*.

Asiakastietolomakkeelle (Kuva 16) lisättäviä kenttiä varten tein moduulin, jossa sijoitin suurimman osan uusista kentistä omalle välilehdelleen. Lomakkeen rakennetta voi tarkastella kehittäjätilan *Edit FormView* -näkymässä, mutta on selkeämpää katsoa rakenne suoraan *base*-moduulin *res.partner.view.xml*-tiedostosta.



Kuva 16. Odoon asiakastietolomake – henkilötiedot.

Kuvassa 17 on esimerkki tekemäni näkymän XML-tiedostosta, jossa näkyy vain osa lisätyistä kentistä. Kuvassa kaikki kentät on ryhmitelty uuteen välilehteen. Lisäsin kenttiä myös olemassa olevaan välilehteen. Koko tiedosto on liitteessä 1.

```
<record id="view_partner_form_inherit_emsys_contacts" model="ir.ui.view">
    <field name="name">res.partner.form.inherit.emsys</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <xpath expr="//page[last()]" position="after">
            <page name="More Information" string="More Information">
                <group attrs="{'invisible': [('is_company', '=', False)]}" string="Company Information">
                    <field name="tol_2008"/>
                    <field name="comp_category"/>
                    <field name="comp_type" class="oe_inline"/>
                    <field name="turnover" class="oe_inline"/>
                </group>
                <group name="group top">
                    <group attrs="{'invisible': [('is_company', '=', True)]}" string="Additional Information">
                        <field name="birthday"/>
                        <field name="responsibility"/>
                    </group>
                    <group attrs="{'invisible': [('is_company', '=', True)]}" string="SoMe Addresses">
                        <field name="linkedin" widget="url"/>
                        <field name="twitter" widget="url"/>
                    </group>
                </group>
            </page>
        </xpath>
    </field>
</record>
```

Kuva 17. Osa näkymän XML-tiedostosta.

Olennaisimpia näkymän määrittelyjä ovat seuraavat:

- Näkymä on mallille *res.partner*.
- *inherit_id*-kenttä identifioi näkymän laajennetuksi, ja *ref*-attribuutin arvona on perityn näkymän external id *base.view.partner.form*.
- *arch*-kentässä määritellään näkymän rakenne.
- XPath-sijaintipolkuna on viimeinen välilehti *page[last()]*, ja uusi välilehti sijoitetaan sen jälkeen.
- Kenttäryhmien attribuuttina on annettu näkymättömyys sen mukaan, onko kyseessä yritys vai ei.

Kuvan 17 esimerkissä XPath-sijaintipolkuna voitaisiin käyttää esimerkiksi *page[@name='sales_purchases']*, jolloin uusi välilehti tulisi *sales_purchases*-nimisen välilehden jälkeen.

Kenttien lisääminen malliin

Python-luokat liittyvät paikallisesti tiedostoon, jossa ne määritellään, eikä luokkien nimillä ole merkitystä muille moduuleille. Luokka-attribuutti *_inherit* kertoo Odoolle, että tämä luokka periytyy *res.partner*-mallista. Attribuuttia *_name* ei tarvita, koska tarkoituksena on laajentaa mallia, eikä luoda uutta. (Reis 2015, 40.)

Kuvan 18 esimerkkikoodissa on Odoossa käytettäviä tietotyyppejä. Koko mallitiedosto on liitteessä 2. Merkkijonon (*Char*), totuusarvon (*Boolean*) ja päivämäärän (*Date*) lisäksi on monen suhde yhteen -tietotyyppi *Many2one*.

Tein asiakastietolomakkeeseen 20 uutta kenttää, joista kahdeksan näkyi lomakkeessa pudotusvalikkona, jolloin kentän tietotyyppinä on *Many2one*. Odoossa voidaan käyttää myös *Many2many*- ja *One2many*-tyyppisiä kenttiä.

```
from openerp import models, fields, api

class EmsysContacts(models.Model):
    _inherit = 'res.partner'
    tol_2008 = fields.Many2one('tol_main', 'TOL2008 Main Category')
    comp_category = fields.Char(string='Comp. Category/Products')
    contacted = fields.Boolean(string='Contacted')
    turnover = fields.Many2one('year_turnover', 'Turnover')
    comp_type = fields.Many2one('company_type', 'Company Type')
    responsibility = fields.Many2one('duty', 'Responsibility')
    birthday = fields.Date('Date of Birth')
    linkedin = fields.Char('LinkedIn')
    twitter = fields.Char('Twitter')

class Tol2008Main(models.Model):
    _name = "tol_main"
    name = fields.Char('TOL2008 Main Category')

class Turnover(models.Model):
    _name = "year_turnover"
    name = fields.Char('Turnover')

class CompanyType(models.Model):
    _name = "company_type"
    name = fields.Char('Company Type')
```

Kuva 18. Osa mallin Python-tiedostosta.

Jos halutaan tehdä pudotusvalikko, jossa on valittavia arvoja valmiina, monen suhde yhteen -relaatiokentille täytyy tehdä malli ja valmiit tietueet. Kuvan 18 Python-koodissa on näkyvissä kolme tekemääni mallia. Mallille *year_turnover* tein kuvassa 19 näkyvät

tietueet. Kirjoitin samaan XML-tiedostoon myös muiden mallien tietueet, poikkeuksena oli *tol_main*-malli.

```
<record id="turnover_1" model="year_turnover">
    <field name="name">0-0.2 MEur</field>
</record>
<record id="turnover_2" model="year_turnover">
    <field name="name">0.2-0.5 MEur</field>
</record>
<record id="turnover_3" model="year_turnover">
    <field name="name">0.5-1 MEur</field>
</record>
<record id="turnover_4" model="year_turnover">
    <field name="name">1-2 MEur</field>
</record>
<record id="turnover_5" model="year_turnover">
    <field name="name">2-5 MEur</field>
</record>
<record id="turnover_6" model="year_turnover">
    <field name="name">5-20 MEur</field>
</record>
<record id="turnover_7" model="year_turnover">
    <field name="name">20-100 MEur</field>
</record>
<record id="turnover_8" model="year_turnover">
    <field name="name">yli 100 MEur</field>
</record>
```

Kuva 19. Esimerkki mallin XML-tietueista.

Halutut tietueet voidaan kirjoittaa XML-tiedostoon, mutta mallien datatiedostoina voidaan käyttää myös CSV-tiedostoja. Jos saman mallin tietueita on paljon, niiden kirjoittaminen XML-tiedostoon on melko puuduttavaa. Myös virheiden mahdollisuus kasvaa, koska tietueita kuitenkin vain kopioidaan ja liitetään jonon jatkoksi.

Kirjoitin ja kopioin *tol_main*-mallin tietueet CSV-tiedostoon. Osa tietueista näkyy kuvassa 20. Kyseessä ovat Tilastokeskuksen Toimialaluokitus 2008 -pääluokat, joita on 22 kappaletta. Tiedoston nimen on oltava *mallin_nimi.csv*, joka tässä tapauksessa on siis *tol_main.csv*.

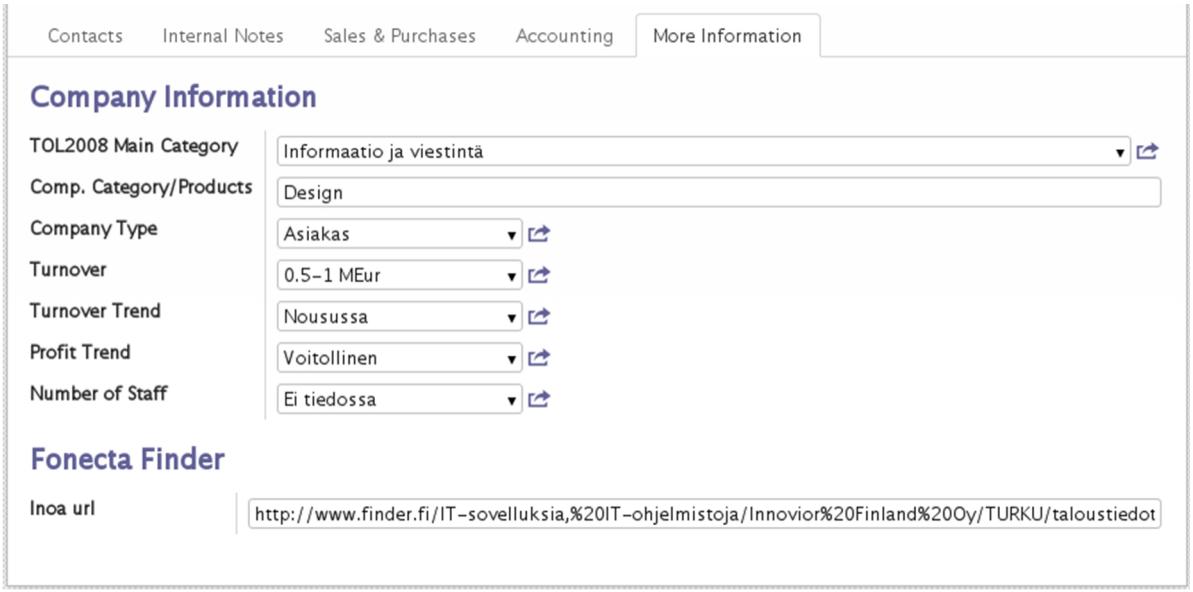
```
"id","name"
"tol_main_1","Ammatillinen, tieteellinen ja tekninen toiminta"
"tol_main_2","Hallinto- ja tukipalvelutoiminta"
"tol_main_3","Informaatio ja viestintä"
"tol_main_4","Julkinen hallinto ja maanpuolustus, pakollinen sosiaalivakuutus"
"tol_main_5","Kaivostoiminta ja louhinta"
"tol_main_6","Kansainvälisten organisaatioiden ja toimielinten toiminta"
"tol_main_7","Kiinteistöalan toiminta"
```

Kuva 20. Toimialaluokitusmallin tietueita.

Ongelmana on, että tuotavien organisaatio- ja henkilötietotiedostojen kentissä ei saa esiintyä muita arvoja kuin mallin *Many2one*-kenttiin kirjoitetut arvot. Olemassa olevien arvojen täytyy myös vastata toisiaan. Tämä tarkoittaa, että niiden täytyy olla vastaavia myös kirjoitusasultaan.

Odoon lomakkeelle pudotusvalikoiksi halutut kentät olivat sellaisia jo Vtiger-lomakkeissa, joten minun täytyi vain luoda tuotavien CSV-tiedostojen arvoja vastaavat kentät. Poikkeuksena olivat toimialapääluokat, jotka eivät olleet pudotusvalikkona Vtiger-ohjelmassa. Näiden kenttien arvot jouduin tarkistamaan tuotavasta organisaatiotiedostosta ja korjaamaan ne mallin kenttiä vastaaviksi.

Kun pudotusvalikot on tehty, Odoossa voi lomakkeelta lisätä arvoja valikoihin tai muuttaa olemassa olevia. Tietueiden poistaminen lomakkeen kautta ei ole mahdollista. Osa asiakaslomakkeelle luoduista uusista kentistä on näkyvissä kuvassa 21.



Kuva 21. Asiakaslomakkeen uusia kenttiä.

Kaikki datatiedostot, jotka moduulin halutaan lataavan, täytyy muistaa kirjoittaa *__openerp__.py*-tiedoston data-attribuutin arvoiksi. Uusille malleille täytyy myös määritellä niiden käyttöoikeudet moduulin *security*-hakemiston *ir.model.access.csv*-tiedostoon.

Odoon käännöstiedostot sijaitsevat moduulin *i18n*-hakemistossa olevissa *language.po*-tiedostoissa. Malliin ja näkymään tekemieni kenttien ja välilehden nimet olivat englanninkielisiä. Lisäsin moduuliin hakemiston, johon tein käännökset suomeksi fi.po-tiedostoon.

6 RAPORTTIPOHJAN KUSTOMOINTI

Toimeksiantaja halusi selvittää, miten Odoon tarjous- ja laskuraporttien pohjia voi muokata asiakkaiden toiveiden mukaisiksi. Odoon oletuslaskupohja on persoonaton, eikä siinä voi esimerkiksi muuttaa yrityksen logoa suuremmaksi. Ajatuksena oli myös lisätä pohjaan tilisiirtolomake.

6.1 Raporttien rakenne

QWeb

QWeb on Odoossa käytettävä XML-pohjainen templaattimoottori (*template engine*), jota käytetään HTML-sivujen ja -osien dynaamiseen muodostamiseen. Se otettiin alun perin käyttöön ohjelmiston versiossa 7, jotta OpenERP:ssä käytettävistä kanban-näkymistä saataisiin monipuolisempia. Odoossa sitä käytetään myös raporttien tuottamiseen. (Reis 2015, 133.)

QWeb-näkymien elementit sijoitetaan <template>-tagin sisälle. Koska jotkin Odoon rakenteellisista malleista ovat monimutkaisia, niiden määrittelyissä voidaan käyttää oikopolkuja usein käytettyihin malleihin. Tagi <template> toimii oikopolkuna *ir.ui.view*-malliin. (Reis 2015, 65.)

QWeb-parseri hakee templaateista erityisiä XML-attribuutteja, direktiivejä, ja korvaa ne dynaamisesti muodostetuilla HTML-osilla. Direktiivien etuliitteenä on *t-*. Esimerkiksi ehtolauseiden direktiivi on *t-if*, iteraatioiden *t-foreach* ja datan renderöintiin *t-esc*, joka muuttaa sisältönsä automaattisesti HTML escape -koodiksi. Merkistökoodauksen siistimisellä (escaping) voidaan rajoittaa cross-site scripting -haavoittuvuuksia. Mikäli voidaan olla varmoja, että lähdedata on turvallista, voidaan käyttää direktiiviä *t-raw*, joka ei muuta sisältöä. (Odoo S.A. 2015f.)

QWeb-templaatteja voidaan käyttää myös muiden templaattien sisältä käyttämällä *t-call*-direktiiviä. Myös muissa moduuleissa määriteltyjä templaatteja voidaan kutsua, jolloin tunnisteena käytetään moduulin nimeä, kuten muissakin näkymissä. Alitason templaatin koko sisältö saadaan *t-call*-elementin sisälle käyttämällä muuttujaa *0* (nolla), esimerkiksi <t t-call="module_name.template_id"><t t-raw="0"/></t>. (Reis 2015, 146–147.)

QWeb-direktiivejä on monia muitakin. Dynaamisesti sisältöä voidaan muuttaa käyttämällä etuliitteitä *t-att-* tai *t-attf-*, ja muuttujia voidaan luoda *t-set*-direktiivin ja sen *t-value*-attribuutin avulla. On olemassa myös erityisiä Python- ja JavaScript-direktiivejä. (Odoo S.A. 2015f.)

Raportit

OpenERP:ssä raportit tehtiin RML-kielellä (Report Markup Language), mutta Odoossa sitä ei suositella, vaan käytössä on HTML/QWeb. Raportit ovat näkymiä, jotka renderöidään HTML-muotoon. HTML-sivu muunnetaan PDF-dokumentiksi Wkhtmltopdf-ohjelmalla.

Periaatteessa raportti on toiminta (*action*), joka yhdistää datan ja näkymät. Jokaisessa raportissa täytyy määritellä *report action*, joka käynnistää tulostuksen. Samoin kuten <template>-tagi, myös <report>-tagi toimii oikopolkuna, joka puolestaan luo *ir.actions.report.xml*-mallin mukaisen tietueen datan kirjoittamista varten. (Odoo S.A. 2015g.)

Report-elementillä määritellään raportin perusparametrit, kuten käytettävä malli, tyyppi ja tallennetaanko raportti tietokantaan muodostamisen jälkeen. Attribuutit *file* ja *name* kertovat käytettävän tiedoston ja templaatin nimen. Tallennettavan raportin nimi määritellään *attachment*-attribuutin Python-lausekkeella (Kuva 22). (Odoo S.A. 2015h.)

```
<report
  id="invoice_emsys"
  string="Emsys Invoice"
  model="account.invoice"
  report_type="qweb-pdf"
  file="emsys.report_emsys"
  name="emsys.report_emsys"
  attachment_use="True"
  attachment="(object.state in ('open','paid')) and
    ('INV'+(object.number or '').replace('/','')+'.pdf')"
/>
```

Kuva 22. Esimerkki report-elementin määrittelystä.

Yleensä raporteissa noudatetaan tiettyä perusrakennetta. Esimerkki yksinkertaisesta QWeb-templaatista on kuvassa 23.

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
 <data>
  <template id="report_emsys_document">
    <t t-call="report.html_container">
      <t t-foreach="docs" t-as="o">
        <t t-call="emsys.external_layout">
          <div class="page">
            <div class="row">
              <div class="col-xs-5">
                <h2>Reportin otsikko</h2>
                <p>Tämän olion nimi on <span t-field="o.name"/></p>
                <p>ja puhelinnumero on <span t-field="o.phone"/></p>
              </div>
            </div>
          </div>
        </t>
      </t>
    </t>
  </template>
 </data>
</openerp>
```

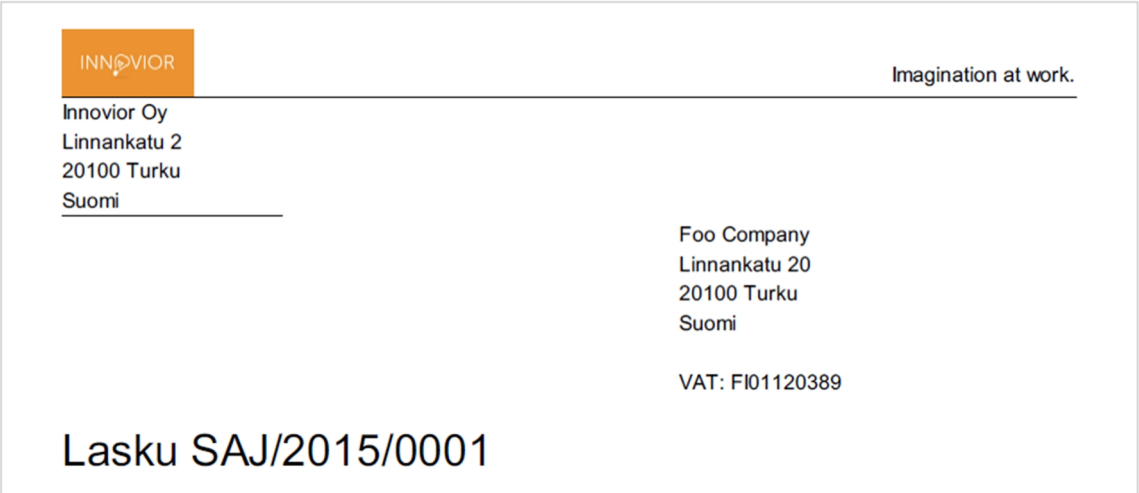
Kuva 23. QWeb-templaatin perusrakenne.

Olennaisimmat osat ovat *t-call*-direktiivit: templaatti *report.html_container* antaa HTML-pohjan ja *report.external_layout* lisää raporttiin ylä- ja alatunnisteet. Raportin sisältö tulee -elementin sisään. Muuttuja *docs* on iteroitava kokoelma tulostettavia tietueita, ja *t-as*-attribuutilla määritellään nimi, jolla viitataan jokaiseen iteraation kohteeseen. (Reis 2015, 153.)

6.2 Moduuli raporttipohjan kustomointiin

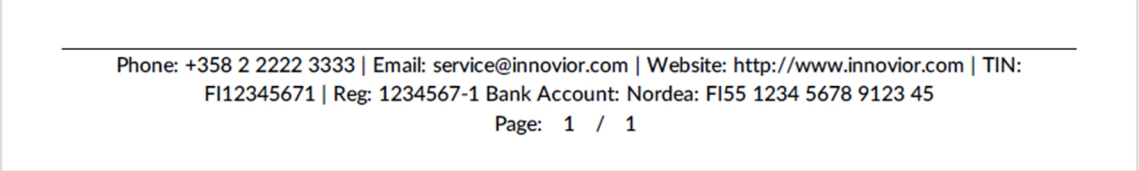
Tekemäni moduulin tarkoituksena ei ollut muuttaa Odoon oletuslaskupohjaa vaan tehdä sen rinnalle vaihtoehtoinen, persoonallisempi pohja, jonka avulla toimeksiantaja voisi esimerkiksi näyttää asiakkailleen, että tarjouksia ja laskuja voi muokata sellaisiksi kuin yritys haluaa.

Oletuksena tulostuvissa laskuissa ylätunnisteessa olevan yrityksen logon kokoa ei voi muuttaa, vaan täytyy tyytyä kuvaan, jonka maksimikorkeus on 40 pikseliä. Kuvassa 24 ylätunniste on yritystietojen yläpuolella oleva osa, jossa on logon lisäksi yrityksen slogan.



Kuva 24. Oletuslaskun yläosa.

Alatunnisteeseen tulevien yritystietojen asettelua ei pysty muuttamaan, joten tiedot voivat jakautua riveille sekavan näköisesti (Kuva 25).



Kuva 25. Oletuslaskun alatunniste.

Raporttien layout kuvataan HTML-kielellä. Odoossa on käytössä Bootstrap-tyylikehys, jossa on valmiiksi määriteltyjä HTML-komponentteja, CSS-luokkia ja JavaScript-funktioita. Bootstrapin tyylimäärittelyjen pohjana on 12 samanlevyisen sarakkeen ruudukko, joka mukautuu päätelaitteiden erikokoisten näyttöjen resoluutioon. Uusi rivi voidaan lisätä -elementin sisään. Ruudukon rivi voidaan jakaa erikokoisiin osiin esimerkiksi <div class="col-xs-N>-määrittelyllä, jossa N on sarakkeiden määrä, jonka ruudukon solu kattaa. (Reis 2015, 155.)

Tarjous- ja laskuraporttien asettelun pohjana on Odoon report-moduulin views-hakemistossa oleva *layouts.xml*-tiedosto. Tiedoston templaateissa määritellään raportin ylä- ja alatunnisteiden asettelu ja sisältö sekä web- ja pdf-raporttien ulkoasun attribuutteja ja tyylitiedostoja.

Raportin varsinainen sisältö ja asettelu määritellään erillisessä XML-tiedostossa. Kutsumalla raportissa layouts-tiedoston *external_layout*-templaattia (Kuva 26) saadaan raporttiin oletuksena olevat ylä-ja alatunnisteet sekä CSS-tiedostoissa määritetyt tyyliohjeet.

```
<template id="external_layout">
    <!-- Multicompany -->
    <t t-if="o and 'company_id' in o">
        <t t-set="company" t-value="o.company_id"></t>
    </t>
    <t t-if="not o or not 'company_id' in o">
        <t t-set="company" t-value="res_company"></t>
    </t>
    <t t-call="report.external_layout_header" />
    <t t-raw="0" />
    <t t-call="report.external_layout_footer" />
</template>
```

Kuva 26. external_layout-templaatti.

Koska halusin muuttaa Odoossa valmiina olevien ylä- ja alatunnisteiden sisällön ja asettelun, tein moduulin oman *layout_emsys.xml*-tiedoston, jossa on muokatut *external_layout_header*- ja *external_layout_footer*-templaatit. Sijoitin nämä samaan tiedostoon, mutta yhtä hyvin voidaan käyttää erillisiä XML-tiedostoja.

Raporttien CSS-tyylimäärittelyjen ohjeet ovat Odoon omilla ohjesivuilla hyvin puutteelliset ja osittain vääriä. Samoja neuvoja myös toistetaan eri foorumeilla, vaikka ne eivät toimi. Tämä aiheutti melkoisesti päänvaivaa.

CSS-tyylit voidaan määritellä paikallisesti suoraan raporttien layout-tiedoston *style*-templaatissa. Jos omia CSS-tyylejä halutaan käyttää yleisesti, Odoon ohjeissa neuvotaan käyttämään tyyli-templaattien perintää. Tämä on tietysti melko hankalaa verrattuna erillisten CSS-tiedostojen käyttöön.

Tarjous- ja laskuraporttien oletusmuoto on PDF, mutta ne voidaan esittää myös HTML-muodossa, mikä täytyy ottaa huomioon CSS-määrityksiä tehdessä. Raporttien layout-tiedostossa peritään Odoon web-moduulin *web.layout*-templaatti, johon tehdään laajennuksia lisäämällä sivun esittämiseen liittyviä attribuutteja ja CSS-tiedostojen määrittelyjä. Jos haluaa tehdä omia CSS-tyylisääntöjä HTML-dokumentin esittämiseen, voi CSS-tiedoston sijaintipolun lisätä raportin *layout*-templaatin <xpath>-elementtiin.

PDF-raporttien tyylitiedostojen linkitys tapahtuu report-moduulin layouts.xml-tiedoston *minimal_layout*-templaatissa. Määrittelin moduulissani CSS-tiedoston sijainnin kuvassa 27 esitetyllä tavalla, jossa tiedosto sijoitetaan muiden templaatin tyylitiedostojen jälkeen.

```
<template id="minimal_layout" inherit_id="report.minimal_layout">
  <xpath expr="//head/link[last()]" position="after">
        <link href="/emsys/static/src/css/emsys.css" rel="stylesheet"/>
    </xpath>
</template>
```

Kuva 27. layout_emsys.xml-tiedoston minimal_layout-templaatti.

Laskun raporttipohjan rakenne ilman varsinaista sisältöä näkyy kuvassa 28. Raportin rakenne ja sisältö, joka tulee -elementin sisälle, on HTML-koodia ja sen joukossa olevia erilaisia *t*-direktiivejä. Myös tarjouspohjan perusrakenne on samanlainen.

```
<openerp>
<data>
  <report
    id="invoice_emsys"
    string="Emsys Invoice"
    model="account.invoice"
    report_type="qweb-pdf"
    file="emsys.report_emsys"
    name="emsys.report_emsys"
    attachment="(object.state in ('open','paid')) and
      ('INV'+(object.number or '').replace('/','')+'.pdf')"
  />

<template id="report_emsys_document">
<t t-call="emsys.external_layout">
  <div class="page">

    <!-- Content here -->

  </div>
</t>
</template>

<template id="report_emsys">
  <t t-call="emsys.html_container">
    <t t-foreach="doc_ids" t-as="doc_id">
      <t t-raw="translate_doc(doc_id, doc_model, 'partner_id.lang', \
        'emsys.report_emsys_document')"/>
    </t>
  </t>
</template>
</data>
</openerp>
```

Kuva 28. Raporttitiedoston perusrakenne.

Raportit tai niiden osia voidaan kääntää dynaamisesti eri kielille kutsumalla metodia *translate_doc*. Metodissa annetaan käännettävän templaatin nimi ja kentän nimi, josta käytettävä kieli löydetään. Kuvan 28 esimerkissä käännös valitaan asiakkaan kielen mukaan, mutta yhtä hyvin se voisi olla käyttäjän kieli *user_id.lang*.

Käytin raporteissa englantia, joten lisäsin moduuliin i18n-hakemiston, jonne tein fi.po-käännöstiedoston suomenkielistä käännöstä varten. Tätä GNU gettext -tiedostoa voi muokata lähes millä tahansa tekstieditorilla, mutta helpoimmin se käy esimerkiksi Poedit-käännöseditorilla. i18n-hakemistoon voidaan lisätä minkä kielinen käännös tahansa.

Edellä kuvaamiani näkymiä ja CSS-määrittelyjä lukuun ottamatta kaikki muu tekemässäni moduulissa peritään muista moduuleista. Tarjous- ja raporttipohjien asettelun ja sisällön muokkaaminen ei itsessään ole kovin vaikeaa. Hankalinta on ymmärtää, miten moduulit rakentuvat ja toimivat. Tekemäni kaksisivuinen muokattu raporttipohja on liitteessä 3, ja kuva Odoon oletuslaskupohjan PDF-tulosteesta on liitteessä 4.

7 POHDINTA

Alun perin opinnäytetyön tavoitteena oli selvittää, miten Odoon kehittäjätilassa voidaan tehdä muutoksia lomakkeisiin ja raporttipohjiin, mutta työn edetessä kävi selväksi, että Odoon kustomointi on järkevämpää tehdä moduulien avulla.

Moduulien rakenteen ja toiminnan selvittäminen lisäsi huomattavasti työn määrää ja haasteellisuutta. Aloittelevalle moduulien kehittäjälle sopivien ohjeiden löytäminen aiheutti ongelmia. Ainoan käytettävissä olevan kirjan ohjeet ja Odoon omat, kehittäjille tarkoitetut ohjesivut tuntuivat aluksi kovin teoreettisilta ja vaikeilta. Neuvojen löytäminen erilaisilta foorumeilta ja muilta nettisivuilta oli työlästä, etenkin kun suuri osa ohjeista liittyi Odoon aikaisempiin versioihin.

Työn edetessä omat tiedot tietenkin karttuivat ja taidot kehittyivät. Omien, toimivien moduulien myötä vahvistui käsitys siitä, että kustomointi moduulien avulla on huomattavasti selkeämpää ja kontrolloidumpaa verrattuna Odoon kehittäjätilaan. Tehtävät muutokset kirjoitetaan suoraan koodiin, ja kokonaisuudesta muodostuu parempi käsitys. Samat muutokset voidaan myös toteuttaa tarvittaessa useissa tietokannoissa asentamalla moduuli niihin. Vastaavasti muutokset voidaan peruuttaa poistamalla moduulin asennus.

Odoossa erilaisten lomakkeiden näkymien rakenne on samankaltainen. Myöskään moduuli, jossa esimerkiksi lisätään asiakastietolomakkeeseen kenttiä, ei poikkea rakenteeltaan moduulista, jossa lisätään kenttiä tilauslomakkeeseen. Kysymys on vain eri malleista ja näkymistä. Valmiin moduulin rakennetta voidaan käyttää pohjana uudessa moduulissa, joka on tarkoitettu saman tyyppiseen kustomointiin.

Tekemäni moduulit olivat melko yksinkertaisia ja perustuivat lähinnä mallien ja näkymien perintään. Niissä ei lisätty toiminnallisuutta uusilla funktioilla tai edes uusilla painikkeilla. Odoon moduulien kehittämisessä en tämän opinnäytetyön puitteissa päässyt siis juuri alkua pitemmälle. Tavoitteisiin nähden työ onnistui kuitenkin mielestäni hyvin, ja siitä on hyötyä myös toimeksiantajalle.

Käytin työssäni Odoon versiota 8, vaikka versio 9 julkaistiin syksyllä 2015. Uudemman version käyttöön ollaan siirtymässä vasta vähitellen, eivätkä muutokset näiden versioiden välillä ole niin merkittäviä kuin ne olivat versioiden 7 ja 8 välillä, jolloin esimerkiksi

ORM-ohjelmointirajapinta muuttui täysin. Oman kokemukseni mukaan Odoo 8:aan tehdyt yksinkertaiset moduulit eivät vaadi monimutkaista muokkaamista toimiakseen uudessa versiossa.

LÄHTEET

Aldrich, J. 2015. Is a Best-of-Breed ERP System the Right Choice for Your Organization? Viitattu 5.12.2015 http://panorama-consulting.com/is-a-best-of-breed-erp-system-the-right-choice-for-your-organization.

Arkkitehtuuri ja MVC 2014. Kurssimateriaali. Helsingin yliopisto. Viitattu 2.1.2016 http://advancedkittenry.github.io/koodaaminen/arkkitehtuuri/index.html.

Aslam, U.; Coombs, C. & Doherty, N. 2014. RE-CONCEPTUALIZATION OF TAILORING TYPES AND THEIR IMPACTS: A PERSPECTIVE FOR CONTEMPORARY ERP SYSTEMS. MCIS 2014 Proceedings. Paper 30. Viitattu 3.12.2015 http://www.researchgate.net/profile/Neil_Doherty2/publications.

Bradford, M. 2015. Modern ERP: Select, Implement, and Use Today's Advanced Business Systems. Third Edition. Raleigh, NC: Lulu Press, Inc.

Dony, O. 2014. Improve the performance of Odoo deployment. Viitattu 1.1.2016 http://www.slideshare.net/openobject/performance2014-35689113.

Fowler, M.; Rice, D.; Foemmel, M.; Hieatt, E.; Mee, R. & Stafford, R. 2002. Patterns of Enterprise Application Architecture. Boston, MA: Addison-Wesley.

Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. 1998. Design Patterns CD: Elements of Reusable Object-Oriented Software. Boston, MA: Addison-Wesley.

Hoffer, J.; Ramesh, V. & Topi, H. 2013. Modern Database Management. Eleventh Edition. Online Chapter 14. Viitattu 12.1.2016 http://wps.prenhall.com/wps/media/objects/14735/15089538/M14_HOFF2253_11_SE_C14WEB.pdf.

Hooschang, M.; Beleshti, H.; Blaylock, B.; Henderson, D. & Lollar, J. 2014. Selection and critical success factors in successful ERP implementation. Competitiveness Review, Vol. 24, Iss 4, 357–375. Viitattu 4.12.2015 http://www.emeraldinsight.com.ezproxy.turkuamk.fi/doi/full/10.1108/CR-10-2013-0082.

Kasampalis, S. 2015. Mastering Python Design Patterns. Birmingham: Packt Publishing Ltd.

Kimberling, E. 2015. Key Findings From the 2015 ERP Report. Viitattu 2.12.2015 http://panorama-consulting.com/key-findings-from-the-2015-erp-report.

Laine, H. 2006. Tietokantaohjelmointi ja Java-tietokantaliittymä. Viitattu 31.1.2016 https://www.cs.helsinki.fi/u/laine/tikas/material/ohjelmointi.html.

Monk, E. & Wagner, B. 2013. Concepts in Enterprise Resource Planning. Fourth Edition. Boston, MA: Course Technology.

Moss, G. 2015. Working with Odoo. Birmingham: Packt Publishing Ltd.

MOT 2016. MOT Dictionaries. Viitattu 31.1.2016 https://mot-kielikone-fi.ezproxy.turkuamk.fi/mot/TURKUAMK/netmot.exe.

Odoo S.A. 2015a. Building a Module. Viitattu 22.11.2015 https://www.odoo.com/documentation/8.0/howtos/backend.html.

Odoo S.A. 2015b. Module Structure. Viitattu 22.11.2015 https://www.odoo.com/documentation/8.0/reference/guidelines.html.

Odoo S.A. 2015c. Manifest. Viitattu 24.11.2015 https://www.odoo.com/documentation/8.0/reference/module.html.

Odoo S.A. 2015d. Installing Odoo. Viitattu 25.11.2015 https://www.odoo.com/documentation/8.0/setup/install.html.

Odoo S.A 2015e. ORM API. Viitattu 25.11.2015 https://www.odoo.com/documentation/8.0/reference/orm.html.

Odoo S.A. 2015f. Qweb. Viitattu 28.11.2015 https://www.odoo.com/documentation/8.0/reference/qweb.html.

Odoo S.A. 2015g. Data Files. Viitattu 28.11.2015 https://www.odoo.com/documentation/8.0/reference/data.html.

Odoo S.A. 2015h. QWeb Reports. Viitattu 10.12.2015 https://www.odoo.com/documentation/8.0/reference/reports.html.

Odoo S.A. 2015i. Building a Website. Viitattu 15.12.2015 https://www.odoo.com/documentation/8.0/howtos/website.html.

Odoo S.A. 2015j. Architecture. Viitattu 2.2.2016 https://doc.odoo.com/trunk/server/02_architecture.

Odoo S.A. 2015k. Views. Viitattu 2.2.2016 https://www.odoo.com/documentation/8.0/reference/views.html.

Panorama Consulting Solutions 2015. 2015 ERP Report. Viitattu 2.12.2015 http://go.panorama-consulting.com/2015-ERP-Report_Download.html.

Python Sofware Foundation 2015. Packages. Viitattu 22.11.2015 https://docs.python.org/2/tutorial/modules.html#packages.

Reis, D. 2015. Odoo Development Essentials. Birmingham: Packt Publishing Ltd.

Shafranovich, Y. 2005. Common Format and MIME Type for Comma-Separated Values (CSV) Files. Viitattu 31.1.2016 https://tools.ietf.org/html/rfc4180.

Silberschatz, A.; Korth, H. & Sudarshan, S. 2011. Database System Consepts. Sixth Edition. New York, NY: McGraw-Hill.

Souza, D. 2013. What is User Exits and Customer Exits? Viitattu 20.1.2016 http://scn.sap.com/docs/DOC-29212.

Uppström, E.; Lönn, C.-M.; Hoffsten, M. & Thorström, J. 2015. New Implications for Customization of ERP Systems. 2015 48th Hawaii International Conference on System Sciences (HICSS), pp. 4220–4229. Viitattu 21.1.2016 http://ieeexplore.ieee.org.ezproxy.turkuamk.fi/xpls/icp.jsp?arnumber=7070324.

Asiakaslomakkeen näkymän XML-tiedosto

```
<?xml version="1.0" encoding="utf-8"?>

<openerp>
    <data>
        <record id="view_partner_form_inherit_emsys_contacts" model="ir.ui.view">
            <field name="name">res.partner.form.inherit.emsys</field>
            <field name="model">res.partner</field>
            <field name="inherit_id" ref="base.view_partner_form"/>
            <field name="arch" type="xml">
                <xpath expr="//field[@name='comment']" position="after">
                    <group attrs="{'invisible': [('is_company', '=', False)]}" string="De-
tails">
                        <field name="desc_customers"/>
                        <field name="principals"/>
                        <field name="competitors"/>
                        <field name="core_competency"/>
                    </group>
                </xpath>
                <xpath expr="//page[last()]" position="after">
                    <page name="More Information" string="More Information">
                        <group attrs="{'invisible': [('is_company', '=', False)]}"
string="Company Information">
                            <field name="tol_2008"/>
                            <field name="comp_category"/>
                            <field name="comp_type" class="oe_inline"/>
                            <field name="turnover" class="oe_inline"/>
                            <field name="trend_turnover" class="oe_inline"/>
                            <field name="trend_profit" class="oe_inline"/>
                            <field name="number_staff" class="oe_inline"/>
                        </group>
                        <group attrs="{'invisible': [('is_company', '=', False)]}"
string="Fonecta Finder">
                            <field name="inoa_url" widget="url"/>
                        </group>
                        <group name="group top">
                            <group attrs="{'invisible': [('is_company', '=', True)]}"
string="Additional Information">
                                <field name="birthday"/>
                                <field name="responsibility"/>
                                <field name="education"/>
                                <field name="lead_source"/>
                                <field name="contact_method"/>
                            </group>
                            <group attrs="{'invisible': [('is_company', '=', True)]}"
string="SoMe Addresses">
                                <field name="linkedin" widget="url"/>
                                <field name="twitter" widget="url"/>
                                <field name="skype" widget="url"/>
                            </group>
                        </group>
                    </page>
                </xpath>
                <xpath expr="//field[@name='title']" position="after">
                    <field name="contacted"/>
                </xpath>
            </field>
        </record>
    </data>
</openerp>
```

Asiakaslomakkeen mallin Python-tiedosto

```
# -*- coding: utf-8 -*-

from openerp import api, fields, models

class EmsysContacts(models.Model):
    _inherit = 'res.partner'
    tol_2008 = fields.Many2one('tol_main', 'TOL2008 Main Category')
    comp_category = fields.Char(string='Comp. Category/Products')
    desc_customers = fields.Char(string='Customers')
    principals = fields.Char(string='Principals')
    core_competency = fields.Char(string='Core Competency')
    competitors = fields.Char(string='Competitors')
    inoa_url = fields.Char(string='Inoa url')
    contacted = fields.Boolean(string='Contacted')
    turnover = fields.Many2one('year_turnover', 'Turnover')
    trend_turnover = fields.Many2one('trend_turno', 'Turnover Trend')
    trend_profit = fields.Many2one('trend_prof', 'Profit Trend')
    number_staff = fields.Many2one('staffno', 'Number of Staff')
    comp_type = fields.Many2one('company_type', 'Company Type')
    lead_source = fields.Many2one('lead_src', 'Lead Source')
    responsibility = fields.Many2one('duty', 'Responsibility')
    education = fields.Char('Education')
    birthday = fields.Date('Date of Birth')
    contact_method = fields.Char('Method of Contact')
    linkedin = fields.Char('LinkedIn')
    twitter = fields.Char('Twitter')
    skype = fields.Char('Skype')


class Turnover(models.Model):
    _name = "year_turnover"
    name = fields.Char('Turnover')

class NumberStaff(models.Model):
    _name = "staffno"
    name = fields.Char('Number of Staff')

class CompanyType(models.Model):
    _name = "company_type"
    name = fields.Char('Company Type')

class LeadSource(models.Model):
    _name = "lead_src"
    name = fields.Char('Lead Source')

class Responsibility(models.Model):
    _name = "duty"
    name = fields.Char('Responsibility')

class TrendTurnover(models.Model):
    _name = "trend_turno"
    name = fields.Char('Turnover Trend')

class TrendProfit(models.Model):
    _name = "trend_prof"
    name = fields.Char('Profit Trend')

class Tol2008Main(models.Model):
    _name = "tol_main"
    name = fields.Char('TOL2008 Main Category')
```

Muokattu laskupohja, sivu 1

INNOVIOR

Innovior
Linnankatu 2
20100 Turku
Suomi

+358 2 4555 6666
info@innovior.com
http://www.innovior.com
Y-tunnus: 1234567-1

Asiakas:

Foo Company
Eerikinkatu 4
20100 Turku
Suomi

ALV: FI25782977

Lasku SAJ/2016/0007

Laskun yhteenveto (erittely seuraavalla sivulla):

Laskun päiväys:	Kuvaus:	Viitteemme:	Laskun numero:	Viitenumero:
03.02.2016	Projekti	SO010	20160007	201600076

Veroton kokonaishinta	Verot	Yhteensä
12 256,00 €	2 907,84 €	15 163,84 €

Saajan tilinumero Mottagarens kontonummer	IBAN Nordea FI55 1234 5678 9123 45 OP FI16 2131 4654 9787 55	BIC NDEAFIHH OKOYFIHH
Saaja Mottagare	Innovior	
Maksajan nimi ja osoite Betalarens namn och adress	Foo Company Eerikinkatu 4 20100 Turku Suomi	
Allekirjoitus Underskrift		Viitenumero Ref. nr: 201600076
Tililtä nro Från konto nr		Eräpäivä Förfallodag: 18.02.2016 — Euro: 15 163,84 €

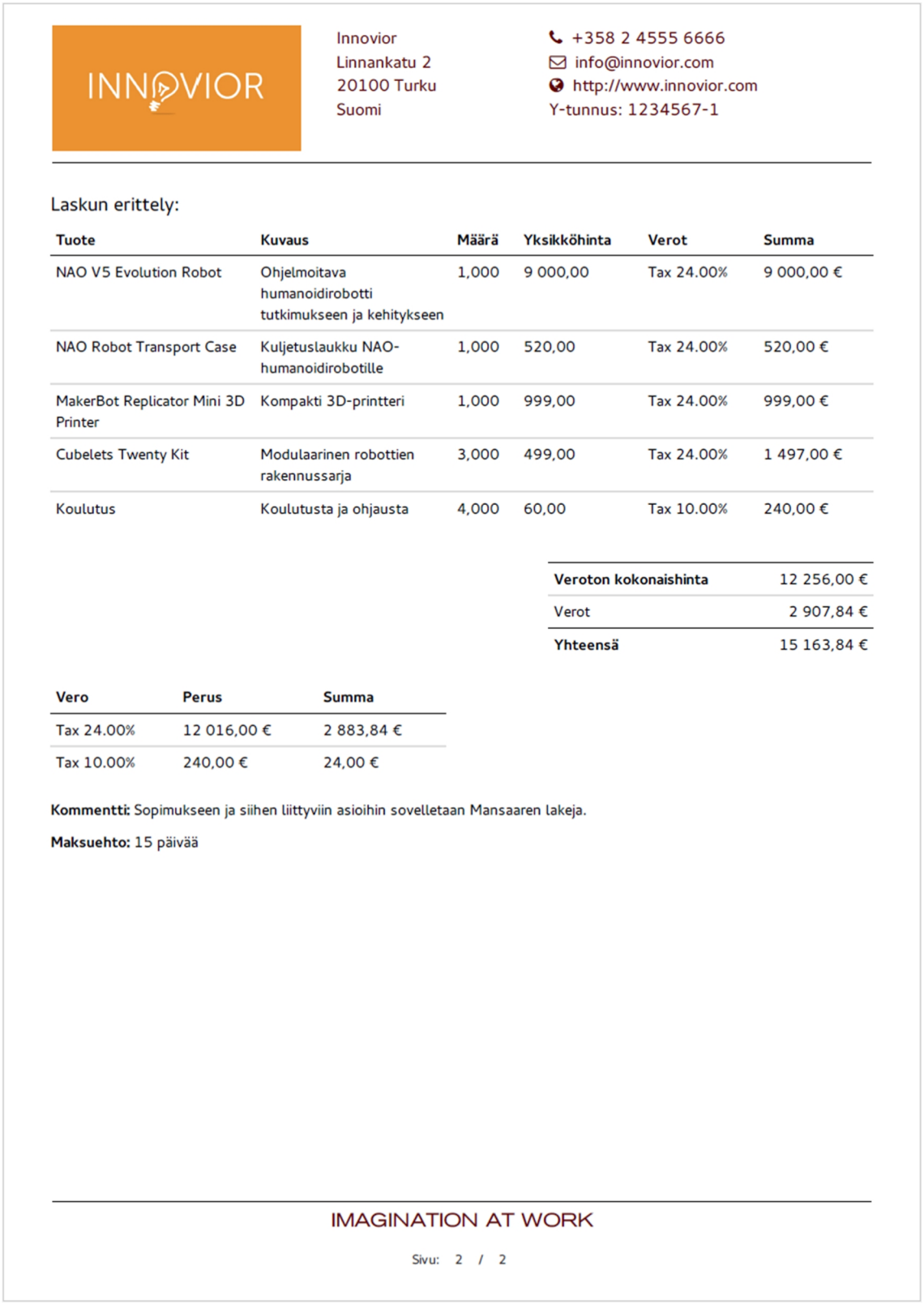
4551234567891234501516384000000000000020160007616021 8

Maksu välitetään saajalle maksujenvälityksen ehtojen mukaisesti ja vain maksajan ilmoittaman tilinumeron perusteella.
Betalningen förmedlas till mottagaren enligt villkoren för betalningsförmedling och endast till det kontonummer som betalaren angivit.

IMAGINATION AT WORK

Sivu: 1 / 2

Muokattu laskupohja, sivu 2



Innovior
Linnankatu 2
20100 Turku
Suomi

+358 2 4555 6666
info@innovior.com
http://www.innovior.com
Y-tunnus: 1234567-1

Laskun erittely:

Tuote	Kuvaus	Määrä	Yksikköhinta	Verot	Summa
NAO V5 Evolution Robot	Ohjelmoitava humanoidirobotti tutkimukseen ja kehitykseen	1,000	9 000,00	Tax 24.00%	9 000,00 €
NAO Robot Transport Case	Kuljetuslaukku NAO-humanoidirobotille	1,000	520,00	Tax 24.00%	520,00 €
MakerBot Replicator Mini 3D Printer	Kompakti 3D-printteri	1,000	999,00	Tax 24.00%	999,00 €
Cubelets Twenty Kit	Modulaarinen robottien rakennussarja	3,000	499,00	Tax 24.00%	1 497,00 €
Koulutus	Koulutusta ja ohjausta	4,000	60,00	Tax 10.00%	240,00 €

Veroton kokonaishinta	12 256,00 €
Verot	2 907,84 €
Yhteensä	15 163,84 €

Vero	Perus	Summa
Tax 24.00%	12 016,00 €	2 883,84 €
Tax 10.00%	240,00 €	24,00 €

Kommentti: Sopimukseen ja siihen liittyviin asioihin sovelletaan Mansaaren lakeja.

Maksuehto: 15 päivää

IMAGINATION AT WORK

Sivu: 2 / 2

Odoon oletuslaskupohja, sivu 1

INNOVIOR

IMAGINATION AT WORK

Innovior
Linnankatu 2
20100 Turku
Suomi

Foo Company
Eerikinkatu 4
20100 Turku
Suomi

TIN: FI25782977

Lasku SAJ/2016/0007

Kuvaus:	Laskun päiväys:	Lähde:	Viittaus:
Projekti	03.02.2016	SO010	Projekti

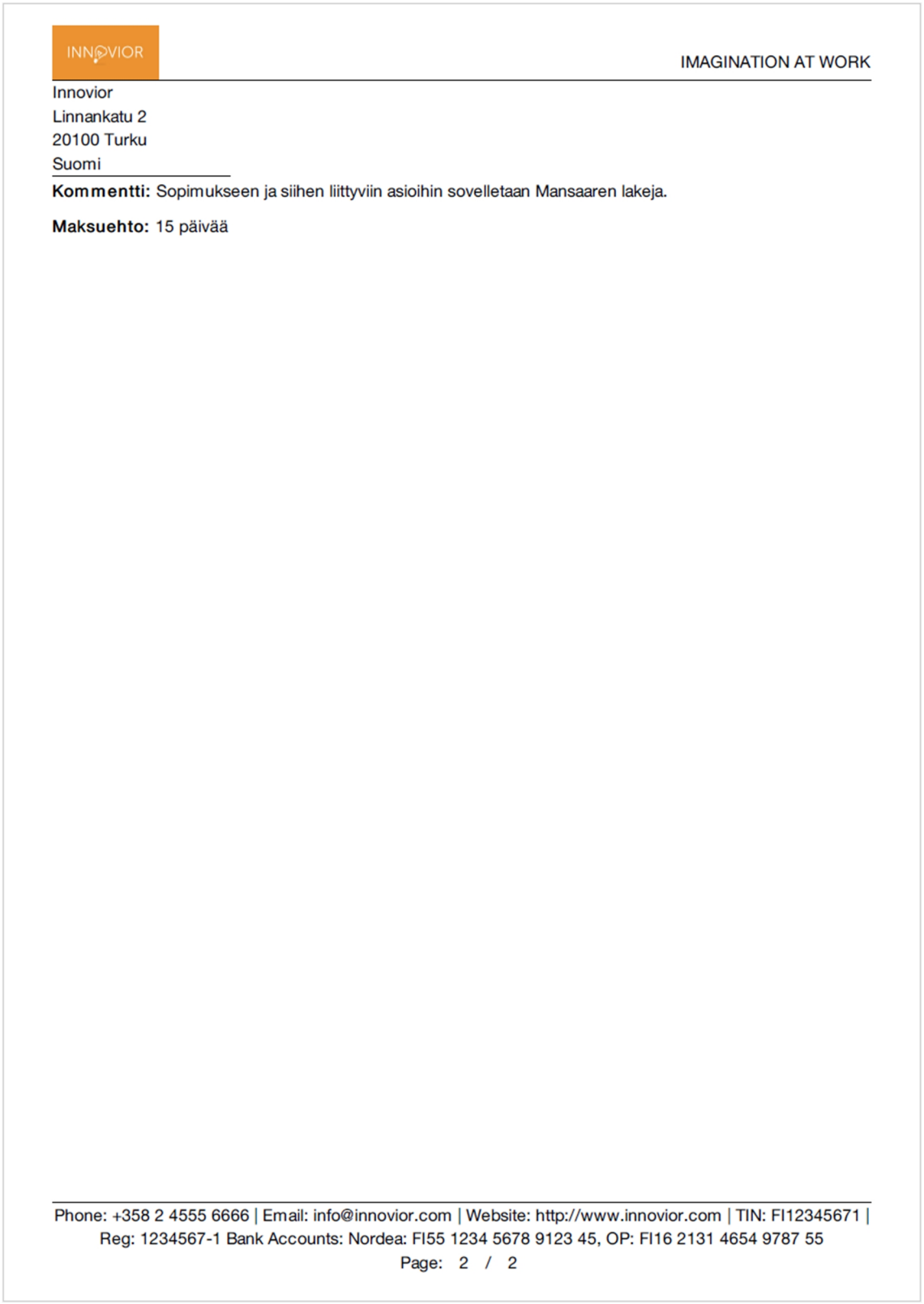
Kuvaus	Määrä	Yksikköhinta	Verot	Summa
Ohjelmoitava humanoidirobotti tutkimukseen ja kehitykseen	1,000	9 000,00	Tax 24.00%	9 000,00 €
Kuljetuslaukku NAO-humanoidirobotille	1,000	520,00	Tax 24.00%	520,00 €
Kompakti 3D-printteri	1,000	999,00	Tax 24.00%	999,00 €
Modulaarinen robottien rakennussarja	3,000	499,00	Tax 24.00%	1 497,00 €
Koulutusta ja ohjausta	4,000	60,00	Tax 10.00%	240,00 €

Veroton kokonaishinta	12 256,00 €
Verot	2 907,84 €
Yhteensä	15 163,84 €

Vero	Perus	Summa
Tax 24.00%	12 016,00 €	2 883,84 €
Tax 10.00%	240,00 €	24,00 €

Phone: +358 2 4555 6666 | Email: info@innovior.com | Website: http://www.innovior.com | TIN: FI12345671 |
Reg: 1234567-1 Bank Accounts: Nordea: FI55 1234 5678 9123 45, OP: FI16 2131 4654 9787 55
Page: 1 / 2

Odoon oletuslaskupohja, sivu 2



INNOVIOR IMAGINATION AT WORK

Innovior
Linnankatu 2
20100 Turku
Suomi

Kommentti: Sopimukseen ja siihen liittyviin asioihin sovelletaan Mansaaren lakeja.

Maksuehto: 15 päivää

Phone: +358 2 4555 6666 | Email: info@innovior.com | Website: http://www.innovior.com | TIN: FI12345671 |
Reg: 1234567-1 Bank Accounts: Nordea: FI55 1234 5678 9123 45, OP: FI16 2131 4654 9787 55
Page: 2 / 2