Filipp Lepalaan

Hyvä yritys: yhden huoltojärjestelmän tarina

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

24.02.2022

Tiivistelmä

Tekijä:	Filipp Lepalaan
Otsikko:	Hyvä yritys: yhden huoltojärjestelmän tarina
Sivumäärä:	42 sivua
Aika:	24.02.2022

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmistotuotanto

Tässä insinöörityössä tarkastellaan yksinyrittäjyyden haasteita ohjelmistotuotannossa. Esimerkkinä käytetään Apple-huolloille tarkoitettua kokonaisvaltaista Django-pohjaista huoltojärjestelmää. Työssä keskitytään kyseisen järjestelmän rakenteen lisäksi myös verkkosovellusten yleisarkkitehtuuriin.

Uudet ketterät kehitysalustat mahdollistavat yhä kattavampien järjestelmien kehityksen yhä pienemmissä tiimeissä. Työssä analysoidaan eri alustojen ja kehitystrendien muutoksia viimeisen 10 vuoden aikana. Mitkä tekniikat ja työkalut ovat vielä relevantteja, jos projektissa on vain yksi kehittäjä?

Miten tuotteistetaan yhden firman tarpeisiin suunniteltu toiminnanohjausjärjestelmä? Voiko yksi henkilö vastata sekä tuotteen kehityksestä ja ylläpidosta että myynnistä ja asiakaspalvelusta?

Avainsanat: Yksinyrittäjyys, web-ohjelmointi, huoltojärjestelmä, Python, PHP, Django, Apple

Abstract

Author:	Filipp Lepalaan
Title:	A nice try: the story of a service management system
Number of Pages:	42 pages
Date:	24 February 2022
Degree:	Bachelor of Engineering
Degree Programme:	Software engineering
Professional Major:	Software production

This thesis observes and documents the challenges of self-employment in the software engineering business. The central example for the work is a Django-based system called Servo – a service management system designed specifically for Apple service providers. In addition to the technical architecture of the system, the thesis also discusses the general architectural considerations of modern process management software.

Modern agile development platforms enable smaller and smaller teams to build increasingly complicated software systems. Which techniques and tools are still relevant if the project team only has one member?

How does one commercialise a software system that was primarily designed for the needs of one company? Can a single person handle both development and system administration as well as sales and customer service?

Keywords: Self-employment, wen-development, service management system, Python, PHP, Django, Apple

Sisällys

Lyhenteet

AASP: *Authorised Apple Service Provider*. Applen valtuuttama huoltoyritys.

GSX: *Global Service Exchange*. Applen maailmanlaajuinen huoltojärjestelmä vuodesta 2003.

HSX: *Humac Service Exchange*. Humac Oy:n huoltojärjestelmä.

SOAP: *Simple Object Access Protocol*. XML-pohjainen sovellusprotokolla.

WSDL: *Web Service Description Language*. SOAP-rajapintaa kuvaava XML-kieli.

VPS: *Virtual Private Server*. Virtuaalipalvelin, jossa asiakkaalla on myös pääkäyttäjäoikeudet virtuaalikoneen käyttöjärjestelmää.

ORM: *Object Relational Mapping*. Relaatiotietokannan ja ohjelmaolioiden välinen rajapinta.

1 Johdanto

Toiminnanohjausjärjestelmät ovat olleet tärkeässä roolissa palvelusektorin digitalisaatiossa 2000-luvulla. Näitten työkalujen avulla organisaatiot ovat pystyneet hallitsemaan kasvavia työmääriä jatkuvasti monimutkaistuvassa teknisessä työssä. Voidaan jopa sanoa, että nämä järjestelmät ovat tietotyölle sama mitä liukuhihnat ovat teolliselle tuotannolle.

Tämän insinöörityön aiheena on tunnetusti maailman ensimmäinen SaaS-mallisesti toimiva, Apple-huolloille tarkoitettu toiminnanohjausjärjestelmä nimeltä Servo (1). Järjestelmä kehitettiin alun perin mcare Oy:lle ja siitä luotiin itsenäinen yritys vuonna 2013.

Päättötyön tavoite on kuvata sekä järjestelmää ja sen arkkitehtuuria, että liiketoimintaa sen ympärillä. Erityisesti halutaan tutkia ohjelmointia yksinyrittäjyyden näkökulmasta. Työn ensimmäisessä osassa keskitytään tuotteen tekniseen toteutukseen ja toisessa puoliskossa yritystoimintaan. Koska järjestelmän tekninen toteutus seurasi pitkälti kehitysalustan asettamia standardeja niin työssä ei listata jokaista teknistä yksityiskohtaa vaan yritetään keskittyä yleispäteviin aiheisiin.

Projektin laajuuden ja pitkän historian (2008-2019) takia sen tutkiminen tuo myös näkemystä ohjelmistokehityksen trendeihin ja kestävän liiketoiminnan haasteisiin. Tämän aikajanan aikana syntyi yhteensä:

- Neljä versiota kahdesta huoltojärjestelmästä
- Kolme osakeyhtiötä
- Neljä avoimen lähdekoodin projektia

2 Projektin lyhyt historia

2000-luvun loppu oli hyvin tapahtumarikasta aikaa Suomen Apple-kaupassa:

- 2008 - HSX korvaa noin kolmen kuukauden kehityksen jälkeen Humacin huollossa vanhan FileMaker-pohjaisen (17) järjestelmän.
- 2009 - MacPeople Oy ostaa Humac Oy:n ja yrityksessä pidetään YT-neuvottelut. Humacin huolto muuttaa uusiin toimitiloihin Fredrikinkadulla.
- 2009 - Mekanisti Oy perustetaan, HSX:n immateriaalioikeudet siirtyy Mekanisti Oy:lle ja järjestelmän nimeksi muutetaan mechDesk. PeopleGroupin kanssa solmitaan sopimus, joka antaa yritykselle ikuisen järjestelmän käyttöoikeuden ilman lisenssimaksua.
- 2009 - mcare Oy perustetaan ja lähes koko Humacin huollon henkilökunta siirtyy mcare Oy:n palvelukseen. Firmassa otetaan käyttöön mechDesk. Servon suunnittelutyö alkaa.
- 2011 - gsxlib-kirjasto julkaistaan Github-palvelussa.
- 2013 – mcaren kehityspäällikkö ja perustaja poistuu firmasta, First Party Software Oy perustetaan ja Servon immateriaalioikeudet siirretään FPS:n omistukseen.
- 2013 - Servon GSX-toiminnot kerätään Python-kirjastoon ja julkaistaan Github ja PyPi-palveluissa.
- 2016 - gsx-mockserver julkaistaan Github-palvelussa.
- 2019 - Servon aktiivinen kehitys jää tauolle.

2.1 Järjestelmän ominaisuudet

Koska mcare Oy oli uusi firma, jossa käytettiin huoltojärjestelmän lisäksi vain kirjanpitojärjestelmää, piti ohjelmiston ominaisuuksien olla erittäin kattavat. Järjestelmän piti myös korvata Applen GSX web-liittymä. Perusidea oli, että henkilökunnan piti päivittäisessä käytössä kirjautua vain yhteen järjestelmään. Esimerkkejä avainominaisuuksista olivat mm:

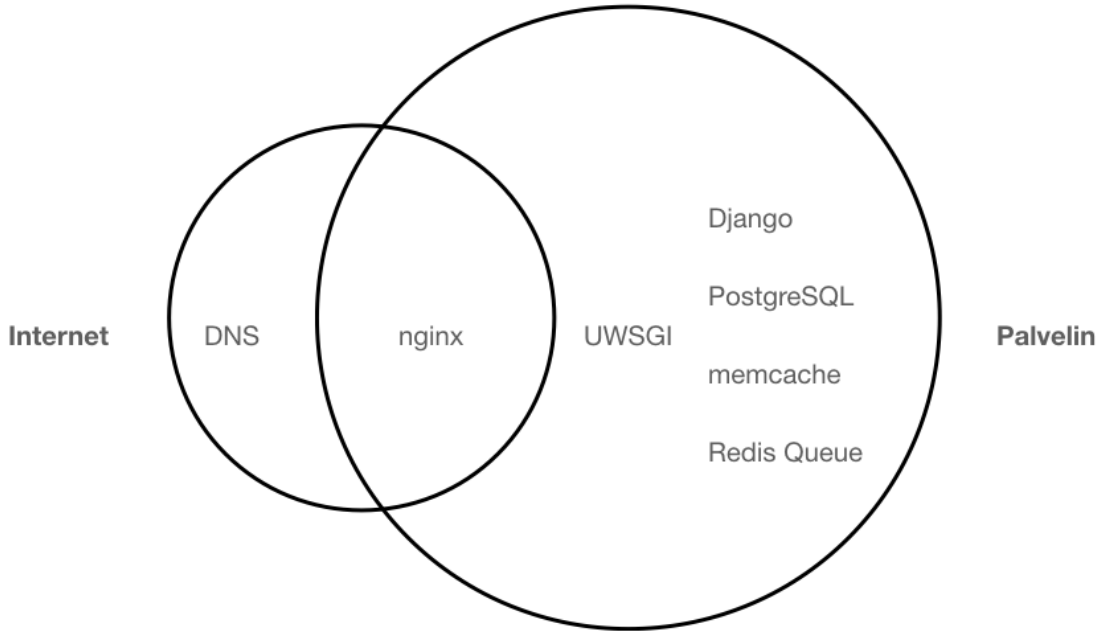
- Tilausrekisteri huoltotöistä sis. kaikkien huoltoon liittyvien asiakirjojen hallinta (tilausvahvistukset, kuitit, kustannusarviot)
- Asiakasrekisteri
- Laiterekisteri korjattavista laitteista
- Varaosahallinta ja logistiikka

- Tuotevarasto
- Asiakasviestintä
- Kattava integraatio Applen globaaliin huoltojärjestelmään

Kehittäjä joutui lähtemään mcaresta (43) juuri ensimmäisen testiversion valmistuttua. Järjestelmä oli vielä hyvin keskeneräinen, kun sitä piti alkaa jo myymään alkupääoman puutteen takia. Hankaluuksista huolimatta, Servo otettiin ensimmäisen kahden vuoden aikana käyttöön kolmessa eri maassa.

3 Järjestelmän rakenne

Modernit verkkosovellukset koostuvat yleensä useammasta liikkuvasta osasta. Keskitetyssä taloushallintajärjestelmässä nämä ovat tyypillisesti selaimessa toimiva asiakaskoodi ja palvelimen puolella toimiva sovelluspalvelin. Sovelluspalvelin koostuu todennäköisesti verkkopalvelimesta, jollain ohjelmointikielellä toimivasta ajoympäristöstä, tietokantapalvelimesta ja välimuistista.



Kuva 1. Järjestelmän komponentit.

3.1 Palvelinarkkitehtuuri

3.1.1 Verkkopalvelin

Verkkopalvelin tai tarkemmin ottaen HTTP-palvelin on ensimmäinen palvelin, joka vastaa asiakkaan pyyntöön. Sen tehtävänä on tyypillisesti ohjata pyyntö eteenpäin sovelluspalvelimelle jonkun välityspalvelimen kautta sekä jakaa staattisia resursseja kuten kuvatiedostoja, CSS-tiedostoja ja JavaScript-tiedostoja.

3.1.2 Ohjelmointikieli ja kehitysalusta

HSX oli kirjoitettu PHP-kielellä ja perustui täysin omaan, vielä vuonna 2007 kehitettyyn koodikirjastoon (35). Servon kanssa haluttiin minimoida ylläpidettävän koodin määrää, joten oman PHP-alustan käyttö lopetettiin ja projektin alkumetreillä testattiin lukuisia muita kehitysalustoja ja ohjelmointikieliä.

Servon ensimmäinen prototyyppi kirjoitettiin vielä PHP:lla Zend Frameworkin päälle mutta ZF osoittautui lopulta huonoksi vaihtoehdoksi lähinnä sen dokumentaation surkean tilan takia vuonna 2010. Järjestelmästä tehtiin rajalliset prototyypit Ruby on Rails ja Node.js alustoilla. Node.js oli tässä vaiheessa vielä hyvin uusi (versiossa 0.9) eikä toiminut vakaasti FreeBSD:ssä (12).

Palvelinosuus päätettiin lopulta toteuttaa Django-alustalla Python-ohjelmointikielellä. Django:n valintaan vaikutti monta tekijää:

- Laadukas, kattava ja hyvin käytännönläheinen dokumentaatio
- Sisältää ratkaisut kaikkiin yleisimpiin web-kehityksen tarpeisiin
- Edistynyt ORM-rajapinta
- Hyvä integraatio oliomallien ja lomakkeiden välillä
- Järkevät oletusarvot tietoturvan osalta
- Hyvät käännöstyökalut
- Hyvät vianhakutyökalut

- Laaja valikoima kolmannen osapuolen lisäosia

Servo on perusluonteeltaan hyvin perinteinen asiakastietojärjestelmä, jonka toiminta perustuu pitkälti lomakkeiden täyttämiseen, taulukoiden selailuun ja tekstihakuun. Djangon 2000-luvun alkuun ulottuvat ja julkaisualasta kumpuavat juuret tekevät siitä erityisen sopivan juuri tämänkaltaisten järjestelmien nopeaan kehittämiseen.

3.1.3 Tietokanta

2010-luku oli turbulenttia aikaa myös tietokantojen maailmassa. MySQL oli ylivoimaisesti suosituin tietokantapalvelin vielä 2000-luvun alussa ja relaatiokantoja pidettiin standardina alustana tiedon tallennukseen ja mallinnukseen. MySQL:n omistuksen siirtyminen Sun Microsystemsille vuonna 2008 ja erityisesti Oraclelle vuonna 2010 kyseenalaisti kuitenkin sen avoimen lähdekoodin tulevaisuuden ja NoSQL-tietokantojen suosion kasvu kyseenalaisti omalta osaltaan taas koko relaatiokantojen roolin moderneissa verkkosovelluksissa.

HSX käytti MySQL tietokantaa, mutta Servolle alettiin jo heti alussa etsimään uutta tietokantaa. MongoDB:tä testattiin vuonna 2010, mutta se todettiin olevan vielä liian uusi ja epävarma teknologia. MongoDB:n hierarkkinen oliopohjaisempi arkkitehtuuri olisi todennäköisesti sopinut paremmin Servon tarpeisiin, mutta vuonna 2010 ei ollut vielä tarpeeksi dokumentaatiota parhaista toimintatapoista tietomallinnuksen osalta.

Tietokannaksi valittiin lopulta PostgreSQL, joka oli myös yksi Djangon oletusarvoista. Valtaosa Servon toiminnoista käyttää Djangon ORM-rajapintaa ja räätälöityjä SQL-kyselyitä tarvittiin vain tilastomoduulissa (36). Nopea tekstihaku sai erityisesti kiitosta käyttäjiltä, joka oli täysin tietokantapalvelimen tehokkaan täysteksti-indeksoinnin ansiosta.

3.1.4 Välimuisti

Järjestelmässä hyödynnetään paljon memcache-välimuistia (25) koska Servo suorittaa usein hitaita ulkopuolisia API kutsuja, joiden tulokset eivät muutu kovin usein. Sitä käytetään myös käyttäjäistuntojen toteutuksessa - käyttäjän istuntotiedot ja profiili ladattiin memcache-välimuistiin onnistuneen autentikoinnin yhteydessä. Järjestelmä sisälsi myös paljon näkymiä, joita luettiin paljon useammin, kun niitä muutettiin, joten memcache-välimuistia käytettiin myös niissä.

Servon lisäksi myös GSX-kirjastolla (gsxws) (33) oli oma välimuisti, johon tallennettiin GSX-yhteyksien istuntotunnisteet. Istunto-ID piti esittää jokaisessa API-kutsussa, joten sen tallentaminen jonkinlaiseen välimuistiin oli välttämätöntä sujuvan käytön takaamiseksi. Koska kirjastoa saatettiin käyttää muissakin projekteissa, toteutettiin sen välimuisti mahdollisimman geneerisella tavalla tallentamalla se mahdollisimman turvallisesti väliaikatiedostoon.

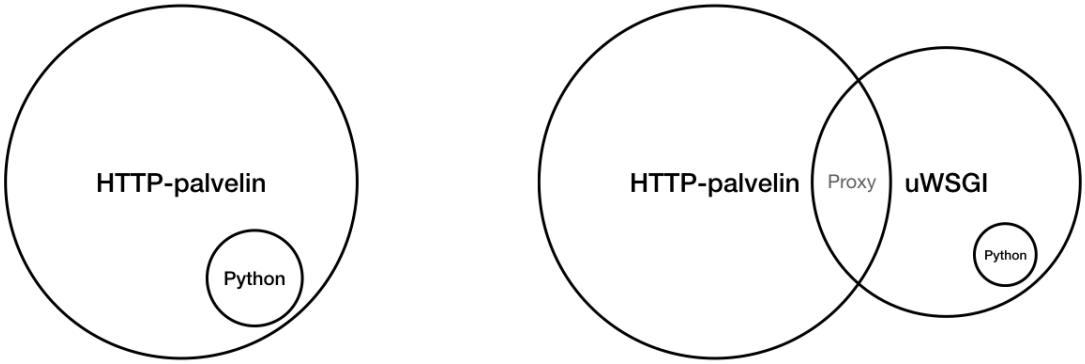
3.1.5 Viestijono

Servo sisältää synkronisten komentojen lisäksi myös asynkronisia toimintoja kuten esimerkiksi ominaisuuden, jonka avulla ylläpitäjä voi määritellä järjestelmässä tiettyjä automaattisia toimintoja (esim. viestin lähetys asiakkaalle, kun tilauksen status muuttuu). Nämä toiminnot oli toteutettu Redis Queue-kirjastolla (26) ja ratkaisu mahdollisti käytännössä ajaa palvelimella useita prosessisäikeitä.

Tietyt toiminnot saattoivat olla niin hitaita, että niitä voitiin ajaa vain työajan ulkopuolella. Esim. varaosien hintojen tai GSX-tilausten statusten päivitys batch-toimintona. Nämä toiminnot toteutettiin yksinkertaisesti käyttöjärjestelmän omilla työkaluilla (cron-skriptit).

3.1.6 Ajoympäristö

Asiakkaita palvelevan http-palvelimen ja varsinaisen verkkosovelluksen väliin tarvitaan joku rajapinta. Tämä voi olla esimerkiksi suoraan http-palvelimeen ladattava laajennus tai ulkopuolinen prosessi. 90-luvulla yleisin rajapinta oli Common Gateway Interface (CGI), (34) 2000-luvun alussa käytettiin enemmän laajennuksia (esim. *mod_php* Apache-palvelimelle). 2010-luvulla ollaan taas siirrytty takaisin enemmän CGI-tyyppisiin rajapintoihin ja välityspalvelimiin.



Kuva 2. Ajoympäristöjen toimintamallit.

Yleisesti voidaan sanoa, että tämä arkkitehtuuri on ylivoimainen verrattuna suoraan http-palvelimeen ladattaviin laajennuksiin sekä suorituskyvyn että turvallisuuden kannalta. Pyyntöjä voidaan uudelleenohjata useammalle sovelluspalvelimelle ja sovelluksen kaatuminen ei voi koskaan kaataa koko verkkopalvelinta.

Pythonin ja Djangon protokolla HTTP-rajapinnoille on Web Server Gateway Interface (WSGI) (21) jota HTTP-palvelimet eivät tue natiivisti, joten WSGI-sovelluksen hallinnasta vastasi uWSGI, joka toimi käytännössä välityspalvelimena http-palvelimelle (nginx). Nginx hoiti myös kaikkien staattisten resurssien (JavaScript, CSS, kuvatiedostot) jakamisen.

3.2 Selaimessa

Selainteknologiat ovat kehittyneet huimasti viimeisen 20 vuoden aikana. 2000-luvun alun JavaScript kirjastot keskittyivät lähinnä selainten välisten JavaScript-tulkkien erojen ja puutteiden paikkaamiseen. Tilanne parani huomattavasti 2010-luvulla web-standardien kehittyessä samalla kun modernit, standardeja paremmin tukevat selaimet yleistyivät. Tämä johti myös kehitysalustojen yltäkylläisyyteen ja myös palvelinteknologioissa havaittuun jatkuvaan ”hopealuodin etsintään”. Servon konseptointivaiheessa kuluikin valtava määrä aikaa eri kielten, työkalujen ja alustojen (mm. Dojo Toolkit, Backbone JS, Angular, Ember JS, SproutCore, Cappuccino) testaamiseen.

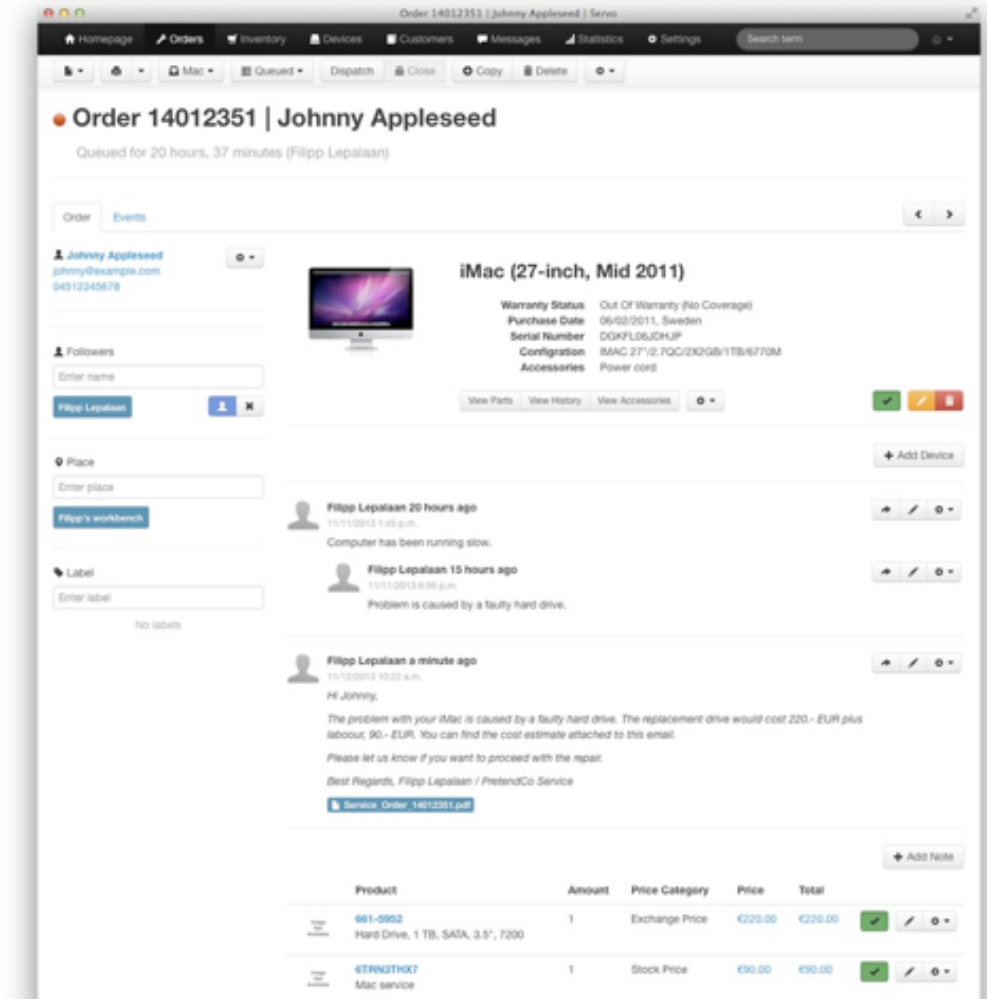
Vaikka tietty teknologia lupaa esittelyissä uusia mahdollisuuksia ja tuottavuuden parannuksia, sen soveltuvuus tiettyyn projektiin on aina tapauskohtaista. Pitkän harkinnan jälkeen tultiin johtopäätökseen, että vaikka raskas JavaScript-sovellus voisi teoriassa tuoda mukanaan jotain etuja, niin perinteinen palvelinpohjainen arkkitehtuuri, jossa käyttöliittymä kasataan jo palvelimella, toimisi teknisesti ihan yhtä hyvin. Tämä vähensi myös koodin duplikointia koska palvelin hoitaa kaiken raskaan työn ja selainkoodi sisälsi lähinnä käytettävyyttä parantavia toimintoja kuten esimerkiksi lomakkeiden kenttien validointia ja erilaisia käyttöliittymäelementtejä (ns ”widgettejä”).

Päätökseen vaikutti myös skaalautuvuus. Palvelimen taakkaa kannattaa ehdottomasti minimoida, jos palvelulla on miljoonia yhtäaikaisia käyttäjiä, mutta maailman suurimmissakin huoltofirmoissa henkilöstömäärä lasketaan sadoissa, ehkä maksimissaan tuhansissa työntekijöissä.

3.2.1 Käyttöliittymän toteutus

Koko käyttöliittymän ulkoasu toteutettiin Bootstrap-kirjastolla (5), joka ei vaatinut minkään tietyn JavaScript-kirjaston käyttöä ja sisälsi hyvän valikoiman järjestelmän tarvitsemia käyttöliittymäkomponentteja. Se säästi paljon

kehitysaikaa ja antoi koko käyttöliittymälle yhtenäisen ulkoasun. Servon käyttöliittymän asettelu perustuu aika pitkälti Bootstrap 2:n aloituspohjaan.



Kuva 3. Huoltotilauksen muokkausnäkymä.

Bootstrapin käytössä ilmeni myöhemmin ongelmia, kun uusi versio (versio 3) rikkoi täysin taaksepäin yhteensopivuuden, mutta yleisesti voidaan sanoa, että valinta oli hyvä ja täytti hyvin projektin tarpeet. Kirjasto on edelleen hyvin suosittu ja sitä näkee käytettävän hyvin monessa verkko-ohjelmistossa. Voidaan melkein sanoa, että sen käytöstä on tullut oletusarvoista projekteissa, jossa ei ole oma graafista suunnittelijaa.

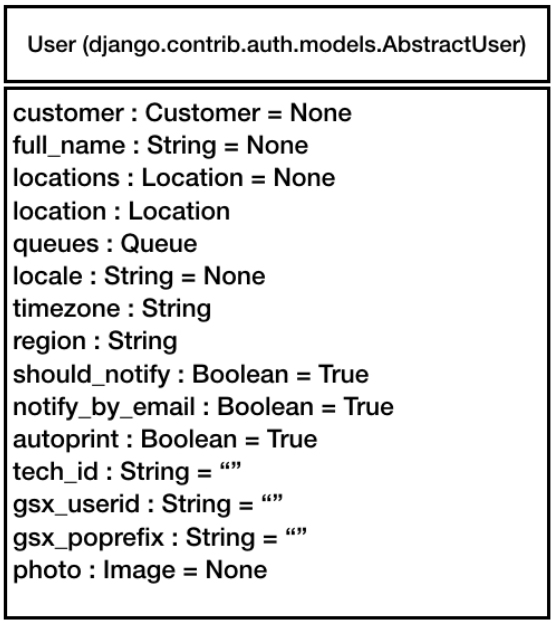
3.3 Tietomalli

Järjestelmän tietomalli on usein paras tapa tutustua sen toiminnallisuuteen ja rakenteeseen sekä hahmottaa järjestelmän laajuutta. Servon tietokannassa on yhteensä 73 taulukkoa, jotka muodostavat yhdessä kymmeniä eri malleja. Seuraavissa kappaleissa keskitytään muutamaan keskeisempään esimerkkiin - käyttäjiin, asiakkaisiin ja merkintöihin.

3.3.1 Käyttäjä

Jokainen toiminto Servossa on aina sidottu johonkin käyttäjätiliin, jonka takia lähes kaikki muut mallit viittaavat siihen. Malli on myös monipuolinen esimerkki Djangon tietokantarajapinnan ominaisuuksien hyödyntämisestä:

- Luokka periytyy *django.contrib.auth.models.AbstractUser*-luokasta, joka sisältää käyttäjätilin perusattribuutteja kuten käyttäjätunnuksen, salatun salasanan ja rekisteröintipäivämäärän.
- *customer*-kenttä viittaa Asiakas-malliin, jonka arvon vaihtoehdot ovat rajattu yrityksiin (Asiakas-mallin *is_company = True*). Tämän avulla asiakkaalle voidaan luoda oma tunnus ja rajata kaikki hakutulokset ja näkymät kyseiseen asiakastiliin.
- *locations* ja *queues*-kentät käyttävät Djangon *ManyToManyField*-tyyppiä, joka hoitaa automaattisesti M/N-tyyppisiä relaatioita (käyttäjä voi toimia monessa toimipisteessä ja käsitellä useampaa työjonoa).

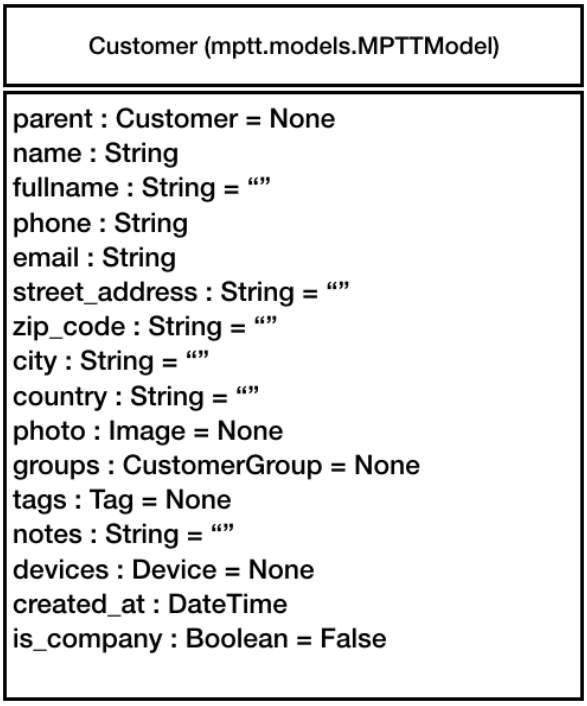


Kuva 4. Käyttäjä-mallin kentät.

3.3.2 Asiakas

Asiakas-malli on todennäköisesti yleisin käsite yritysohjelmistoissa. Niitten toteutuksessa on historiallisesti havaittu kuitenkin puutteita, jotka Servossa yritettiin huomioida. Esimerkiksi organisaatiorakenteita voidaan kuvata tarkasti hierarkkisen rakenteen avulla ja sama asiakas voi liittyä eri tilauksissa joko yritysasiakkaana tai yksityishenkilönä.

Kenttien määrä on jätetty mahdollisimman pieneksi koska käytännössä mikään asiakaskanta ei voi koskaan ennakoida kaikkia mahdollisia kenttiä, joita kukin yritys tarvitsee. Tätä varten Servossa on erikseen Property-malli, jolla huoltoyritys voi itse määritellä lisäkenttiä asiakkaan tietoihin.

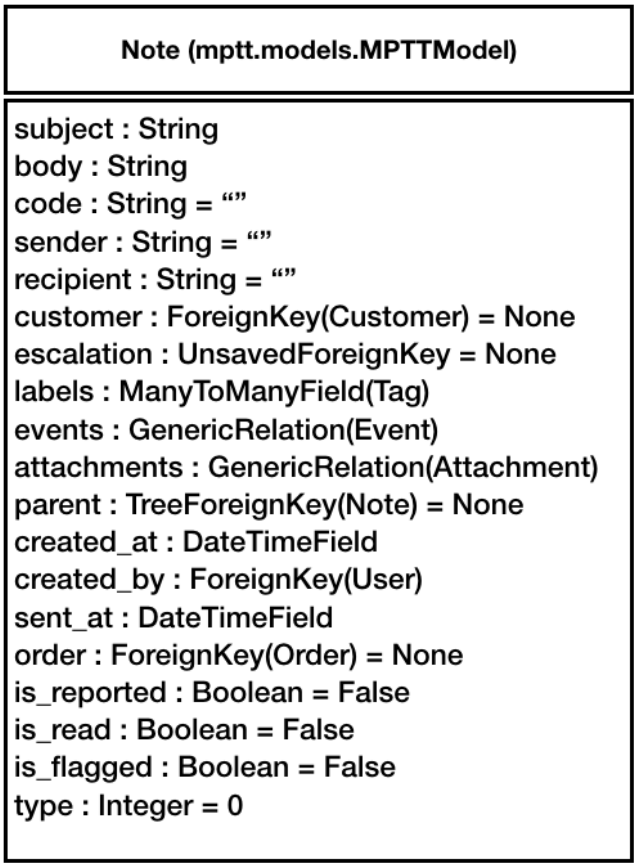


Kuva 5. Asiakas-mallin kentät.

3.3.3 Merkintä

Merkintä-malli suunniteltiin mahdollisimman yleispäteväksi koska järjestelmässä käytetään tätä käsitettä useassa eri kontekstissa. Merkintä voi olla esimerkiksi asiakkaan kirjottama vikakuvaus, teknikon diagnoosi tai Applelle lähetetty huoltotapauksen eskalaatio. Tekstiviesti on käytännössä vain merkintä, johon on lisätty saajan puhelinnumero. Viestiketjut voivat olla myös hierarkkisia.

SQL-relaatiokannat kuten PostgreSQL eivät ole alun perin tarkoitettu hierarkkisen tiedon tallentamiseen ja käsittelyyn. Vuosien varrella onkin kehitetty erilaisia algoritmeja tämän ongelman ratkaisemiseksi (mm. Adjacency List (28), Nested Set (29) ja Modified Preorder Tree Traversal). Djangon ORM-rajapinnalle löytyi tähän tarkoitukseen paketti (38), joka sisältää hierarkkisten tietomallien lisäksi myös HTML-pohjissa käytettäviä tägejä, joiden avulla voidaan hyvin helposti kuvata hierarkkisia rakenteita (esim. valikoita tai keskusteluketjuja).



Kuva 6. Merkintä-mallin kentät.

3.3.4 Geneeriset relaatiot

Tietyt oliot kuten liitetiedostot, tapahtumat ja ominaisuudet (tägi, jolla on sekä nimi että arvo) voivat liittyä useampaan muuhun malliin. Esimerkiksi liitetiedostoja esiintyy sekä merkinnöissä, laitteissa että asiakkaissa ja tägeillä voidaan merkitä melkein mitä vaan. Servossa tällaiset relaatiot toteutettiin Djangon contenttypes-framework:in (18) avulla. Käytännössä tämä tarkoittaa, että viittaavan malliin merkitään viitattavan mallin primääriavaimen lisäksi myös sen ”tyyppi” (eli taulukon nimi). Konkreettisena esimerkkinä tästä ovat edellä mainitun merkintä-mallin *events* ja *attachments*-kentät.

Geneeriset relaatiot ovat tehokas ja joustava tapa mallintaa relaatioita, jota voi soveltaa hyvin monessa käyttökohteessa. Täytyy kuitenkin muistaa, että niitten käyttö estää tietokannan natiivien viite-eheys sääntöjen käytön.

3.4 REST-rajapinta

Servoon lisättiin yksinkertainen REST-rajapinta (7), jonka avulla huoltoyritykset pystyivät integroimaan sen omiin tietojärjestelmiin ja prosesseihin. Ennen kaikkea tätä tarvittiin järjestelmän integroinnissa yrityksen kotisivuihin, esimerkiksi “tarkista huollon tila”-tyyppiselle toiminnolle. Rajapinnan lisääminen jo tuotantokäytössä olevaan järjestelmään voi olla haasteellista, mutta Djangolle löytyi juuri tähän tarkoitukseen lisäosa (14). Ratkaisu olikin lopulta aika suoraviivainen ja Servoon lisättiin erillinen “api”-niminen sovellus, joka sisälsi kaikki rajapinnan päätepisteet.

3.4.1 Todennus

Kaikki API-kutsut sidottiin aina johonkin pääkäyttäjän määrittelemään käyttäjätiliin ja todennus toteutettiin yksinkertaisesti API-avaimilla, joita voitiin generoida useampi per käyttäjä. Tämän järjestelyn avulla huoltoyrityksen pääkäyttäjä pystyi luomaan yhden yleiskäyttäjän mahdollisimman suppeilla oikeuksilla ja lisäämään jokaiselle ulkoiselle integraatiolle oman API-avaimen. Jos avain vuoti, se voitiin vaan poistaa ja luoda uusi rikkomatta muiden asiakkaiden integraatioita.

Käyttäjätunnuksen pystyy myös kytkemään tiettyyn asiakastiliin, jolloin kaikki rajapinnan kautta tulevat uudet tilaukset kirjattiin automaattisesti kyseisen asiakkaan nimiin. Ominaisuus rajasi myös kaikki hakutulokset kyseisen asiakkaan tilauksiin. Tämä ominaisuus oli erityisen hyödyllinen huolloille, joilla oli isoja yritysasiakkaita tai alihankintasopimuksia koska asiakas pystyi luomaan täysin oman lomakkeen huoltolähetteille.

3.5 Yhteys Applen huoltojärjestelmään

Kaikki Applen valtuuttamat huollot kommunikoivat Applen huoltokanavan kanssa Global Service Exchange (GSX)-järjestelmän kautta. Selainpohjainen järjestelmä otettiin käyttöön vuonna 2003 jolloin se korvasi Apple Order Global-

nimisen varaosien tilaamiseen tarkoitetun, vielä puhelinverkkoyhteyttä käyttävän työpöytäsovelluksen. Tässä ympäristössä käsitellään kaikki takuuhuollot sekä takuun ulkopuoliset korjaukset, jotka käyttävät Applen virallisia varaosia.

Valtaosa huolloista käyttääkin Applen luomaa web-käyttöliittymää joko oman huoltojärjestelmän tai pelkän kirjanpitojärjestelmän ohella. Tämä tarkoittaa, että kaikki huoltotilauksen tiedot täytyy kirjata vähintään kaksi kertaa - ensin huollon omaan ja sitten Applen järjestelmään. Yksi Servon myyntiargumenteista oli tämän tuplakirjauksen poistuminen.

GSX:n web-liittymän takana on verkkopalvelu, jonka kanssa voidaan kommunikoida SOAP protokollalla ja vuoden 2014 päivityksen jälkeen myös REST-rajapinnan kautta. Järjestelmällä on melko korkeat tietoturvavaatimukset. Asiakkaan täytyy olla valtuutettu huoltoyritys, jonka sovelluspalvelimen julkinen IP-osoite on lisätty Applen palomuuriin ja käyttäjätunnuksen ja salasanan lisäksi todennuksessa käytetään myös asiakassertifikaatteja.

HSX tuki GSX:n yleisempiä hakutoimintoja kuten takuun tarkistusta ja varaosien hakua jo vuonna 2008. Nämä ja lukuisat muut toiminnot kerättiin omaan gsxlib-nimiseen projektiin (32), jota käytettiin aktiivisesti siihen asti, kunnes GSX muuttui REST-rajapinnaksi.

Kirjaston toteuttaminen PHP:llä oli suhteellisen helppoa koska PHP tukee natiivisti SOAP-protokollaa (20). Pythonissa tilanne oli täysin toinen – Pythonin standardikirjasto ei sisällä lainkaan SOAP-toimintoja ja vuonna 2012 ei löytynyt vielä yhtäkään kolmannen osapuolen kirjastoa, joka olisi toiminut edes jotenkin hyväksyttävällä suorituskyvyllä. Applen WSDL-tiedostot olivat niin raskaita, että yhteys verkkopalveluun aikakatkaistiin usein ennen, kun ohjelma oli edes ehtinyt parsia rajapinnan kuvauksen. Pythonille löytyi onneksi laadukas XML-kirjasto (27), jonka avulla kirjoitettiin täysin oma GSX-kirjasto. Se ei tukenut lennosta kaikkia rajapinnan toimintoja koska kyseessä ei ollut yleispätevä SOAP-implementaatio, mutta järjestely toimi muuten hyvin ja kirjastoa integroitiin vuosien aikana myös muihin Appleen liittyviin ylläpitotyökaluihin.

GSX-rajapinnan dokumentaatio koostui pitkälti XML-esimerkeistä, jossa kuvattiin ensin asiakkaan lähettämä pyyntö ja sitten palvelun palauttama vastaus. Nämä esimerkit osoittautuivat hyvin arvokkaiksi Python-kirjaston luomisessa ja esimerkit koottiin myöhemmin erilliseen työkaluun nimeltään gsx-mockserver (31) joka simuloi GSX-rajapintaa ja helpotti tällä tavalla rajapintatoimintojen testaamista.

3.5.1 Välikäden riskit

Integrointi GSX:ään oli yksi Servon suurista myyntivalteista, mutta se oli myös ylivoimaisesti suurin huolenaihe. Koska uusi järjestelmä korvasi käytännössä Applen järjestelmän, myös kaikista Applen rajapinnassa ilmenevistä ongelmista tuli uuden järjestelmän ongelmia. Jos aikaisemmin teknikko oli näissä tilanteissa yhteydessä Applen asiakastukeen, niin nyt ongelmasta ilmoitettiin ensin kehittäjälle, jonka piti selvittää asiaa yhdessä asiakkaan ja Applen tuen kanssa.

Toinen tärkeä liiketoimintaa haittaava tekijä olivat rajapinnan ja huoltoprosessien päivitykset. Kehittäjän piti olla jatkuvasti ajan tasalla molempien muutoksista, joka oli hyvin vaikeaa, jos kehittäjällä ei ollut omaa huoltotoimintaa. Huoltoprosessien muutokset tulivat kehittäjän tietoon vasta kun järjestelmästä löydettiin puute, eli toisin sanojen silloin kun oli jo liian myöhäistä.

Välikäden roolia hankaloitti myös se, että Applen huoltokanava kommunikoi vain valtuutettujen huoltojen kanssa, joten kaikki yhteydenotot piti aina suorittaa jonkun huoltoyrityksen nimissä.

3.6 Tietoturvamalli

Käyttöoikeuksien hallinta perustui hyvin pitkälti “implicit deny/explicit allow”-periaatteeseen - jossa kaikki toiminnot olivat oletusarvoisesti estettyjä ja ylläpitäjän piti antaa käyttäjälle eksplisiittisesti oikeudet tiettyyn toimintoon. Järjestelmässä ei ole sisäänrakennettua “järjestelmäkäyttäjä”-tiliä, jolla

suoritetaan esimerkiksi automatisoituja toimintoja. Tämä lisää järjestelmään läpinäkyvyyttä ja helpottaa myös auditointia.

Oikeuksien hallinta suoritettiin ryhmätasolla ja sen toteutukseen käytettiin kattavasti Djangon tarjoamia työkaluja (13). Järjestelmä oli välillä jopa liian hienojakoinen ja asiakkailla oli joskus vaikeuksia tämän ominaisuuden hahmottamisen kanssa.

Kaikki toiminnot suoritettiin aina jollain käyttäjänä ja käyttäjän ID kirjattiin aina jokaiseen tapahtumaan ja tilamuutokseen, myös toiminnot, jotka toimivat päältäpäin katsottuna vieraskäyttäjinä. Esimerkiksi jos huolto käytti Servon itsepalveluliittymää, jonka kautta asiakkailla oli mahdollisuus kirjata itse laitteensa huoltoon niin pääkäyttäjän piti ensin luoda käyttäjä ja määritellä se itsepalvelunäkymän omistajaksi. Myös REST-rajapinta käytti tätä samaa logiikkaa.

3.7 Monivuokraus (multi-tenancy)

Servo oli alun perin suunniteltu yhden yrityksen käyttöön, joten mahdollisuutta ajaa useampaa huoltoyritystä samassa asennuksessa ei koskaan edes harkittu. Tämä loi haasteen myyntiprosessissa koska jokaiselle asiakkaalle piti luoda kokonaan oma asennus. Jokaisella asiakkaalla oli siis oman datan lisäksi myös oma kopio koko koodipohjasta.

Järjestelmää yritettiin päivittää sellaiseksi, että yksi koodipohja ja tietokanta voisi palvella useampaa asiakasta, mutta tämä osoittautui jälkeenpäin liian vaikeaksi ja nosti myös tietoturvakysymyksiä, joita ei oltu huomioitu alkuperäisessä määrittelyssä. Vaikka järjestely vaikeutti ylläpitoa, se toi myös yhden ison hyödyn - mahdollisuuden ajaa eri koodia eri asiakkailla. Esimerkiksi, jos yksi asiakas törmäsi ongelmaan, joka ei esiintynyt muilla, voitiin hänelle asentaa täysin oma versio, jossa pystyttiin testaamaan tuliko ongelma korjatuksi tai mahdollisesti lisätä jopa monitorointikoukkuja ongelman selvittämiseksi. Tämä osoittautui erittäin hyödylliseksi työkaluksi erityisesti, kun piti selvittää

ongelmia Applen rajapintojen kanssa koska eri asiakkailla saattoi olla Applen järjestelmässä erityyppisiä tilejä ja käyttötapoja.

3.8 Sähköpostit ja SMS

Asiakasviestintä on hyvin tärkeä osa huoltoprosessia, joten järjestelmän piti pystyä lähettää tekstiviestejä sekä lähettää että lukea sähköposteja. Sähköpostituen lisääminen oli helppoa - SMTP ja IMAP ovat hyvin tuettuja protokollia, jolle löytyy myös vakaita koodikirjastoja mille tahansa suositulle ohjelmointikielelle.

Vaikka itse tekstiviestit ovat avoin GSM-standardi, HTTP-rajapinnat yhdyskäytäväoperaattoreiden palveluihin ovat valitettavasti aina operaattorikohtaisia. Tämän takia jouduttiin lisäämään useampia operaattoreita. Yksi niistä oli Kannel (9) - ainoa SMS yhdyskäytävä, jonka asiakas pystyi asentamaan myös itse. Pääasiallinen operaattori oli SMSApi (10) ja asiakas pystyi ostamaan kehittäjältä tietyn määrän tekstiviestejä kuukaudessa.

4 Kehitysprosessi

4.1 Seuraavan version kirous

Servo oli käytännössä “HSX 2.0” ja kuten niin moni muu “2.0 projekti”, se yritti korjata kaikki edeltäjänsä puutteet. Sen tuloksena Servon suunnitteluvaihe kesti pidempään, kun koko HSX:n kehitys ja käyttöönotto ja johti ilmiöön, jota Frederick P. Brooks kuvailee osuvasti teoksessaan “The Mythical Man-month”:

> “An architect’s first work is apt to be spare and clean. He knows he doesn’t know what he’s doing, so he does it carefully and with great restraint.
>
> As he designs the first work, frill after frill and embellishment after embellishment occur to him. These get stored away to be used “next time”. Sooner or later the first system is finished, and the architect, with firm confidence and a demonstrated mastery of that class of systems, is ready to build a second system.

> The second system is the most dangerous system a man ever designs.” [Brooks, 1995, s. 55]

Kyseessä on psykologinen ilmiö, jota on hyvin vaikea välttää. Ohjelmistokehitys on varsinkin arkkitehdin näkökulmasta innovatiivisten ratkaisujen ja ideoiden rakennustyömaa ja juuri näiden ideoiden toteuttaminen auttaa meitä pysymään motivoituneena joskus hyvinkin pitkien ja haastavien projektien aikana. Ensimmäisessä projektissa meidän tietämättömyys suojaa meitä ideoiden tulvalta. Juuri kuten Brooks toteaa, seuraavassa versiossa kehittäjän kohonnut itsevarmuus nostaa tulvaportit ja tuloksena ei ole vain pitkittynyt suunnitteluvaihe vaan todennäköisesti myös epämääräinen spesifikaatio, joka taas johtaa joko täysin turhiin ominaisuuksiin tai toimintoihin, jotka eivät toimi kunnolla.

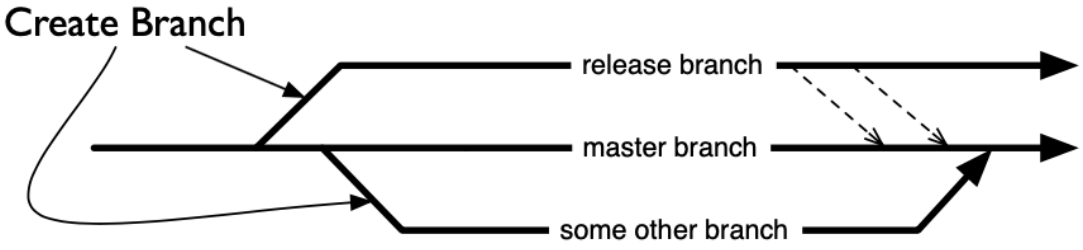
Tämä ilmiö on erityisen vaarallinen yhden ihmisen kehitystiimeissä koska resurssit määritelmän yksinkertaistamiseen ovat niin rajalliset. Näissä tiimeissä puuttuu myös ulkopuolinen kriittinen silmä, joka auttaisi karsimaan turhia lisäyksiä ja muutoksia. Servon kehitys kärsi pahasti tästä ilmiöstä myös senkin takia, että kehitystyötä tehtiin usein hyvin läheisesti asiakkaan kanssa ja asiakkailla oli jatkuvasti lisätoiveita ja ideoita. Asiakas tietää harvoin mitä hän haluaa ja vielä harvemmin mitä hän ei halua. Tarkemmin sanottuna, asiakas ymmärtää tarpeitaan vasta kun kyseinen osio on jo valmis.

4.2 Versiohallinta

Versiohallintatyökalut ovat välttämättömiä, jos tiimissä on useampi kehittäjä, mutta niistä voi olla myös hyötyä yhden kehittäjän projekteissa. Ne auttavat pitämään kirjaa muutoksista ja toimivat myös hyvänä lähdekoodin varmuustallennuksena.

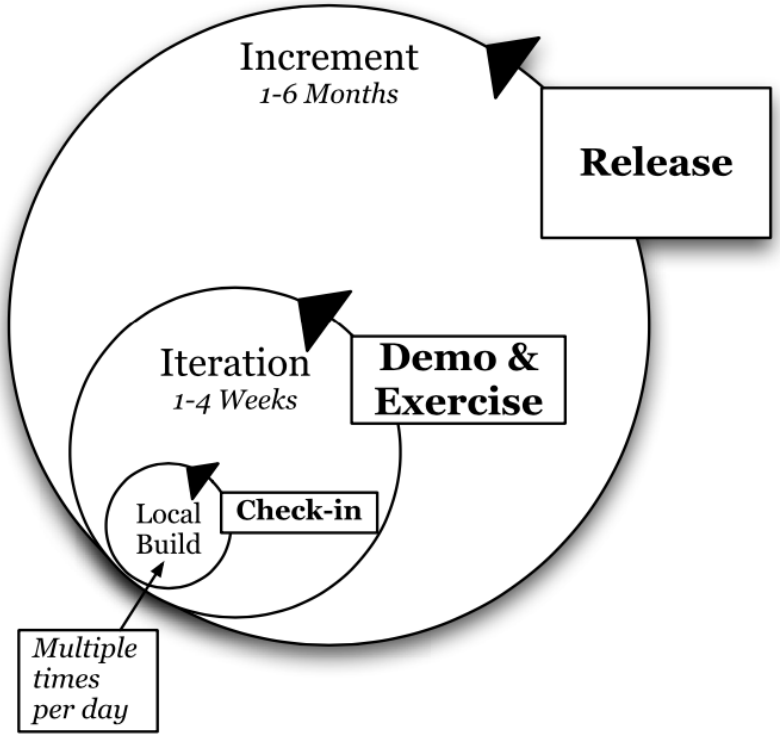
HSX:n versiohallinta oli vielä vuonna 2008 toteutettu Subversion työkalulla, mutta sen käyttö edellytti keskitetyn palvelimen olemassaoloa. Servon kehitysprosessin aikana Gitistä oli tulossa hyvin suosittu ja kun siitä julkaistiin myös käytännönläheinen kirja (6), päätettiin viedä versiohallinta yli uuteen

järjestelmään. Gitin hajautettu toimintamalli sopii hyvin sooloprojekteille koska sen käyttö ei vaadi mitään lisäinfraa.



Kuva 7. Haarojen toiminta Git:issä (6).

Tuotannossa pyörivä koodi synkronoitiin lähes koko yritystoiminnan aikana päähaaran kanssa (master branch) kanssa. Jos päähaarassa oli paljon keskeneräisiä muutoksia ja asiakkaalla ilmeni kriittinen vika, käytettiin git:in stash-komentoa joka ”piilotti” keskeneräiset muutokset väliaikaisesti, kunnes kriittinen korjaus oli viety päähaaraan ja asennettu sieltä tuotantopalvelimelle.



Kuva 8. Ketterän kehitysprojektin julkaisusyklit (15)

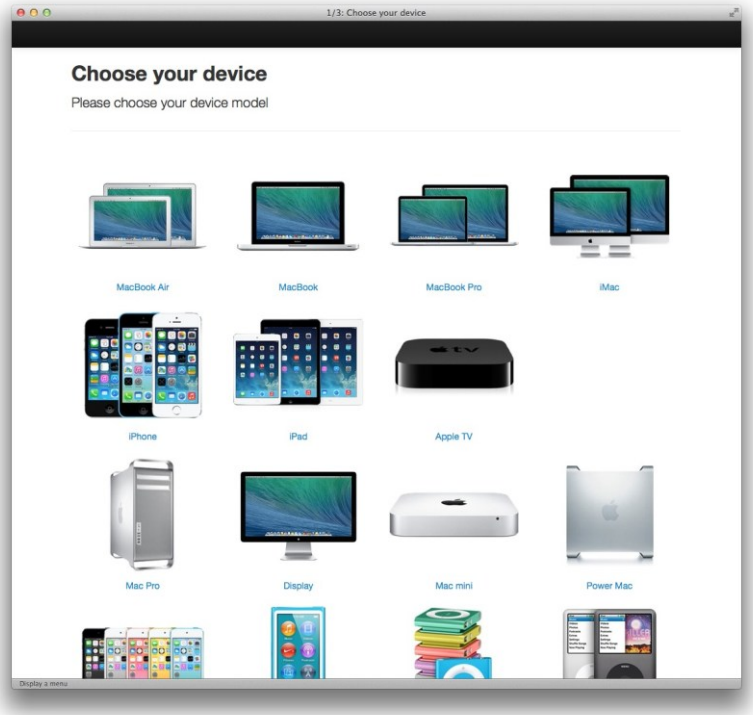
Uusia versioita asennettiin projektin alkuvaiheessa päivittäin ja loppuvaiheessa melkein joka viikko ja muutokset ja parannukset listattiin muutoslokiin (16). Päivitykset suoritettiin asiakkaan työajan ulkopuolella hallintokomennolla joka:

- Otti varmuuskopiot päivitettävän asennuksen tietokannasta
- Latasi Github-palvelusta uusimman version
- Suoritti tietokantamigraatiot
- Tyhjensi aktiiviset istunnot
- Uudelleenkäynnisti WSGI-sovellukset

4.3 Testaus

Huolellisesti suunniteltu testausautomaatio voi parhaimmillaan parantaa ohjelmiston laatua sekä estää teknisen velan kertymistä. Niistä saama hyöty voi olla erityisen iso juuri hyvin rajallisten resurssien projekteissa. Huonosti suunnitellut testit voivat toisaalta luoda väärän turvallisuudentunteen. Aiheesta on viime vuosina julkaistu paljon kirjallisuutta ja alalle on tullut kokonaisia uusi termejä kuten jatkuva integraatio (continuous integration) ja uusia rooleja kuten DevOps.

Yksi projektin tavoitteista oli hyödyntää testausautomaatioita mahdollisimman paljon, mutta ainoaksi helposti testattavaksi osaksi Servossa osoittautui lopulta sen REST API:n rajapinta (7). Yleensä ominaisuuksia testattiin käsin kehitysvaiheessa, mutta esimerkiksi uuden itsepalvelukäyttöliittymän kanssa tämä osoittautui niin hitaaksi prosessiksi, että ominaisuudelle kehitettiin oma Selenium-pohjainen funktionaalinen testi (39), joka liitettiin osaksi Servo:n testipakettia.



Kuva 9. Itsepalveluliittymä, jonka testaus automatisoitiin funktionaalisilla testeillä.

Sekä gsxlib (PHP) (40) että gsxws (Python) (41) sisälsi melko kattavat testit koska kirjastojen selkeät rajapinnat ja graafisen käyttöliittymän puute tekivät niistä suhteellisen helposti testattavia. Molemmissa projekteissa käytettiin pitkään TDD (Test Driven Development) - menetelmää, jossa uudelle toiminnolle kirjoitettiin ensin testi ja sitten vasta toteutus.

GSX-yhteys lisäsi omat haasteensa testausprosessiin koska rajapinnan testiversiot olivat hyvin hitaat ja epävarmat. Kaikkien rajapintapyyntöjen piti myös täyttää Applen tietoturvavaatimukset, joka ei ollut edes mahdollista, jos kehittäjällä ei esimerkiksi ollut voimassaolevaa GSX-tiliä. Tämän helpottamiseksi kehitettiin erillinen *gsx-mockserver*-välikerros (31), joka simuloi GSX-rajapinnan toimintoja. Tämänkaltainen testaus on tavallaan kyseenalaista koska testauskohde ei vastaa todellisuutta, mutta se auttoi esimerkiksi vähentämään regressioita.

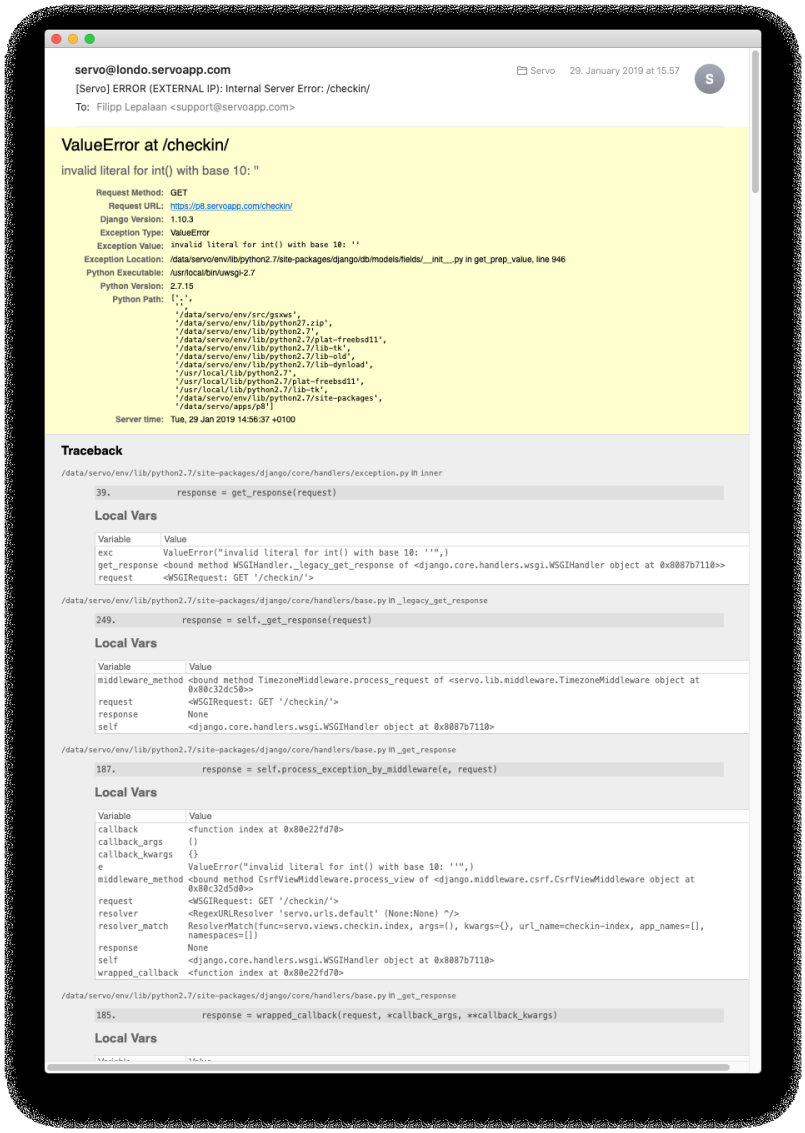
Servon testauksen automatisointi olisi ollut helpompaa, jos järjestelmän arkkitehtuuri olisi alun perin toteutettu web-rajapintana.

4.4 Monitorointi ja vianhaku

Järjestelmien toiminnan seuraaminen tuotannossa on tärkeä osa minkä tahansa ohjelmistotuotteen kehitystä. Siellä voidaan havaita ohjelmointivirheitä, koodipolkuja ja tiloja, joihin ei vielä törmätä toimintojen kokeiluvaiheessa.

Palvelinten toimintaa ja kuormaa seurattiin käyttöjärjestelmän perustyökaluilla kuten *top*, *lsof* ja *uptime* sekä seuraamalla yksittäisten palveluiden lokitiedostoja. Tärkein monitorointityökalu olivat Djangon virheraportit, jonka järjestelmä generoi joka kerta kun Python-tulkki törmäsi käsittelemättömään poikkeukseen. Nämä sisälsivät poikkeusviestin lisäksi myös täyden pinojäljityksen kaatumisen hetkellä. Käyttäjille kuvattiin näissä tilanteissa yleinen virheilmoitus, jossa oli myös tekstikenttä, johon käyttäjä voi lisätä vapaasti merkintöjä ongelman kontekstista. Nämä viestit yhdistettiin piilotetun tunnisteen avulla virheraporttiin myöhempää analysointia varten.

Virheraporteille perustettiin oman sähköpostiosoite ja tuotantojärjestelmät säädettiin lähettämään raportit tähän osoitteeseen. Virheviestejä kerääntyi vuosien aikana kymmeniä tuhansia.



Kuva 10. Virheraportti sähköpostiviestinä.

Ongelmien toistaminen kehitysympäristössä oli joskus täysin mahdotonta, joten korjauspäivityksiä tehtiin välillä “sokkona” näiden raporttien perusteella. Hyvin usein kyseessä oli GSX-yhteyteen liittyvä ongelma, joka sisälsi tuotantoversiossa tiloja, joita ei yksinkertaisesti pystytty toistamaan kehitysversiossa. Tuotannossa löydettiin myös lukuisia virheitä Applen SOAP-rajapinnassa ja sen dokumentaatiossa ja kaikki löydetyt virheet raportoitiin myös Applelle.

Samaa menetelmää käytettiin myös palvelimen tietoturvan monitorointiin. Kaikki tuotantopalvelimet säädettiin lähettämään päivittäiset turvaraportit kehittäjän sähköpostiin. Nämä raportit sisälsivät mm. tietoa ohjelmistopakettien haavoittuvuuksista ja epäonnistuneista sisäänkirjautumisyrityksistä. Muitakin monitorointityökaluja kokeiltiin, mutta niistä oli vähän hyötyä koska tuotantopalvelimia oli vain muutama.

4.5 Kansainvälistäminen ja lokalisointi

Järjestelmä kehitettiin englanniksi ja käännettiin sen lisäksi viidelle eri kielelle - Suomi, Viro, Ruotsi, Tanska ja Puola. Koska kuittien ja kustannusarvioiden piti täyttää paikalliset lakisääteiset vaatimukset, käännöksissä otettiin huomioon myös päivämäärien ja valuuttojen oikea formatointi.

Django sisälsi kaikki tarvittavat ominaisuudet järjestelmän kansainvälistämiseen:

- Kaikki HTML-pohjissa ja koodissa käytettävät tekstit korvattiin alusta lähtien gettext-merkeillä.
- Aikaleimoissa tallennettiin myös aikavyöhyke

Käännökset toteutettiin *gettext*-formaatissa Poedit-työkalulla. Englannin, Suomen ja Viron tekstit kirjoitettiin itse, Ruotsi, Tanska ja Puola toteutettiin Google Translate:in ja asiakkaiden avulla.

Djangon mukana tuli myös iso määrä käännöksiä yleisimmille termeille useammalle kielelle. Yleisesti voidaan todeta, että pienikin ohjelmistoyritys voi nykytyökalujen avulla palvella aika hyvin globaaleja markkinoita.

4.6 Dokumentaatio

Laadukas ja kattava dokumentaatio voi auttaa asiakasta ymmärtämään tuotetta paremmin jo ennen ostopäätöksen tekemistä ja keventää samalla myös kehittäjän ylläpidollista taakkaa ostopäätöksen jälkeen. Se ei kuitenkaan ole

ehdoton edellytys menestyksekkäälle yritystoiminnalle – paljon tärkeämpää on suunnitella helppokäyttöinen tuote jonka käyttö ei vaadi ohjekirjan lukemista. Poikkeus tässä ovat kehitysalustat ja erityisesti ohjelmointikielet, jossa hyvä dokumentaatio on sekä kilpailuetu että elinehto.

Servon dokumentaatiolle luotiin oma Django-sovellus (4), johon kerättiin vastauksia kaikkiin yleisimpiin kysymyksiin sekä jokaisen versiojulkaisun yhteydessä päivitettävä muutosloki (16). Uusia ohjeita lisättiin aina kun huomattiin joku toistuva kysymys, mutta asiakkaat eivät valittaneet kertaakaan puutteellisesta dokumentaatiosta.

5 Ylläpito

Servon ylläpitomalli oli hyvin perinteinen - ohjelmisto asennettiin omille virtuaalipalvelimille ja kaikki osat - paitsi itse laitteisto olivat kehittäjän hallinnassa. Tämä päätös oli osittain historiallinen koska haluttiin hyödyntää ja ylläpitää omaa tietotaitoa myös järjestelmäylläpitäjänä. Palvelimien ylläpito toi myös välillä paljon kaivattua vaihtelua ohjelmoinnille ja se vei loppujen lopuksi hyvin vähän aikaa suhteessa ohjelmointiin.

5.1 Palvelininfra

Palvelinalustan valinta on tärkeä päätös, jonka ohjelmistokehittäjän on tehtävä ennen markkinoille menoa. Hänen täytyy harkita huolellisesti alustan teknistä soveltuvuutta, toimintavarmuutta sekä alustan elinkaarikustannuksia. Tälläkin osa-alueella on tapahtunut paljon muutoksia viimeisen 20 vuoden aikana, mutta onneksi ei yhtä paljon, kun kehityspuolella ja perinteiset toimintatavat ovat edelleen hyvin yleisiä ja erittäin toimivia.

5.1.1 VPS-palvelimet

Django-sovelluksille on saatavilla lukuisia integroituja pilvialustoja kuten Amazon Web Services, Heroku ja Azure App Services, mutta niiden käyttöä

haluttiin välttää sekä kustannussyistä että toimittajan toimesta tapahtuvan lukituksen riskin takia (ns. ”vendor lock-in”). Järjestelmävaatimuksissa haluttiin välttää erikoisratkaisujen käytön myös sen takia, että yksi tuotteen mahdollisista jakelumalleista oli sen asentaminen asiakkaan omille palvelimille.

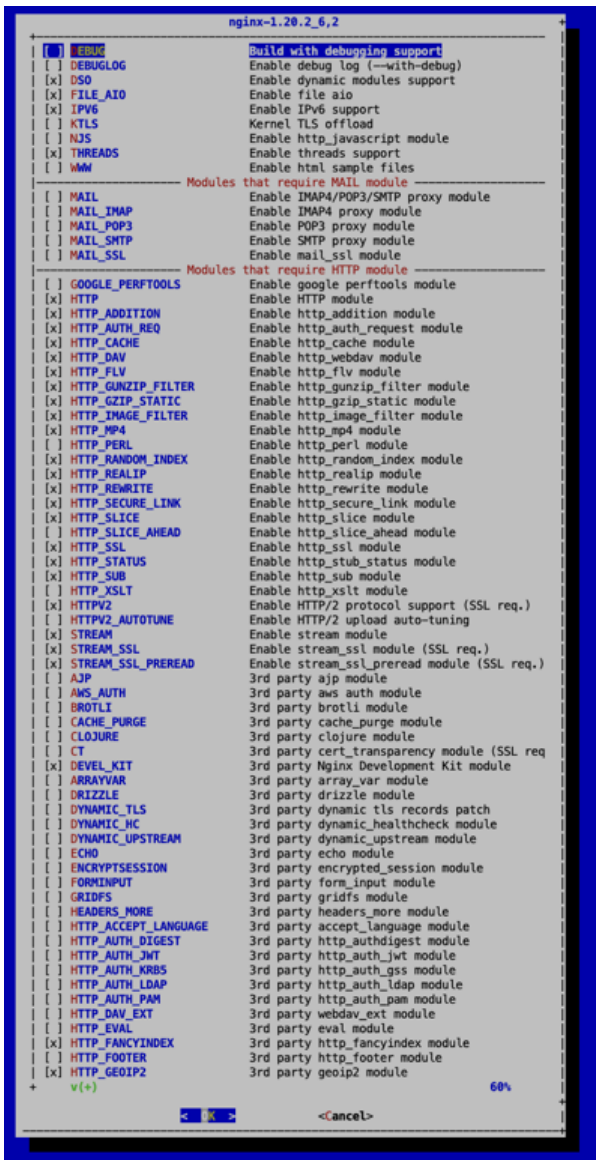
Virtuaalikoneiden konfiguraatiot vaihtelivat jonkun verran asiakastilien tarpeiden mukaan, mutta tyypillinen palvelin sisälsi 8Gt keskusmuistia, kaksi 2,4Ghz suoritinta ja noin 80Gt tallennustilaa, josta käyttöjärjestelmä ja Servo veivät vajaat 2Gt. Levyjärjestelmä päivitettiin SSD-pohjaiseksi heti kun se oli saatavilla koska SSD-levyjen pienempi hakuaika vaikutti paljon tietokannan suorituskykyyn. Tällainen kokoonpano pystyi hyvin palvelemaan asiakasta, jolla oli useita satoja samanaikaisia käyttäjiä ja yli sata tuhatta tallennettua huoltotilausta.

Käyttöjärjestelmän räätälöintimahdollisuus antaa kehittäjälle täysin vapaat kädet optimoida suoritusympäristö omiin tarpeisiin. Vaikka ratkaisu vaatii enemmän tietotaitoa, oli mahdollisuus siirtää koko palvelu vaikka omalle kotikoneelle todella arvokas.

5.1.2 FreeBSD

FreeBSD on kokonainen UNIX-pohjainen käyttöjärjestelmä, jonka ensimmäinen versio julkaistiin jo vuonna 1993. Pääasiallinen syy FreeBSD:n käytölle oli kehittäjän aikaisempi vuosien kokemus sen käytöstä palvelimissa ja palomuureissa ja se osoittautui erinomaiseksi alustaksi myös verkkopalvelun pyörittämiseen.

Käyttöjärjestelmä tukee sekä binaari että -lähdekoodimuotoisia ohjelmistopaketteja, joka mahdollistaa ohjelmistojen räätälöimisen ja optimoinnin moduulitasolla jo asennusvaiheessa. Ylläpitäjä voi siis itse lisätä verkkopalvelimelle tarvittavia laajennuksia tai jättää kokonaan pois ominaisuuksia, joita hän ei tarvitse parantaen samalla järjestelmän suorituskykyä ja pienentäen sen hyökkäyspinta-alaa.



Kuva 11. nginx-verkkopalvelimen moduulivaihtoehdot FreeBSD:ssä.

Oletusarvoinen asennus on hyvin minimalistinen ja ylläpitäjän täytyy itse lisätä ja määritellä jokainen lisäosa. SSH (Secure Shell) on ainoa palvelu, joka voidaan aktivoida jo käyttöjärjestelmän asennusvaiheessa eikä asentaja tarjoa esimääriteltyjä kokonaisuuksia tiettyihin käyttötarkoituksiin (työpöytäkone, web-palvelin, tiedostopalvelin jne) kuten esimerkiksi monet suositut GNU/Linux-jakelut.

Tuotteen elinkaaren aikana suoritettiin neljä suurta käyttöjärjestelmän päivitystä (9.1 -> 13.0) sekä useita kymmeniä tietoturvapäivityksiä ja prosessi oli

suhteellisen vaivaton käyttöjärjestelmän mukana tulevien työkalujen ansiosta (42).

5.2 DNS

DNS on kriittinen osa jokaista verkkopalvelua koska ilman toimivaa nimipalvelua asiakkaan verkkoselaimet eivät yksinkertaisesti löydä kyseistä palvelua. Siksi haluttiin välttää kokonaan julkisen DNS infran itsenäistä ylläpitoa ja palvelu ulkoistettiin. Tärkeimmiksi kriteereiksi palvelutarjoajan valinnassa olivat toimintavarmuus, mahdollisimman lyhyt vasteaika muutoksille ja hinta. Jos esimerkiksi asiakkaan asennus jouduttiin siirtämään uudelle palvelimelle kapasiteettipulan takia, oli tärkeätä pystyä luottaa siihen, että DNS-muutokset tulevat voimaan nopeasti ja tasaisesti.

Sovelluspalvelimille asennettiin myös paikalliset DNS palvelimet (24), jotka toimivat palvelinten sisäisinä DNS välimuisteina ja jotka vastasivat vain palvelimien omiin kyselyihin. Tämä lyhensi kyselyiden vasteaikoja sekä paransi vikasietoisuutta.

5.2.1 Nimeämiskäytäntö

Asiakastilejä oli kahdentyyppisiä - demo (d) ja tuotanto (p) ja jokaiselle asiakastilille annettiin oma numero ja luotiin oma A (IPv4) ja AAAA (IPv6) -tietue. Esimerkiksi p1.servoapp.com vastasi asiakas ID 1 tuotantojärjestelmää. Jotkut asiakkaat halusivat käyttää osoitteissa firman nimeä, jolloin sille luotiin oma CNAME-tietue, joka ohjattiin asennuksen kanoniseen osoitteeseen. Jotkut asiakkaat taas halusivat, että osoite on heidän oman verkkotunnisteen alla (esim. servo.mcare.fi) jolloin asiakkaan piti itse huolehtia DNS-tietueiden ylläpidosta.

Demoympäristöt jätettiin usein aktiivisiksi myös kokeiluvaiheen jälkeen. Tämä mahdollisti mm. uusien ominaisuuksien testaamisen ennen tuotantoympäristöön viemistä. Asiakkaat käyttivät demoympäristöä usein myös kokeilu ja

koulutusalustana uusille käyttäjille koska niissä ei ollut vaaraa tuotantodatan korruptoitumisesta.

5.3 SSL-varmenteet

Salattujen HTTP-yhteyksien käyttö oli 2000-luvun alussa vielä suhteellisen harvinaista, erityisesti yritysten verkkopalveluissa, joiden käyttö oli rajattu firman lähiverkkoon. Suurimmat esteet salauksen käytön yleistymiselle olivat varmenteiden suhteellisen korkea hinta ja hankintaprosessin hankaluus. Varmenteiden hinnat alkoivat laskemaan kilpailun kasvaessa 2010-luvulla ja suurin muutos tapahtui vuonna 2014 kun markkinoille tuli ensimmäinen täysin ilmainen sertifikaattiauktoriteetti Let’s Encrypt (LE).

LE:n ainoa miinuspuoli oli varmenteiden suhteellisen lyhyt voimassaoloaika (90 päivää) (23). Onneksi LE:llä on hyvä työkalu varmenteiden uusimiseen (certbot) joten prosessi kiteytyy käytännössä yhden komennon ajamiseen. Varmenteiden uusimisen todennus automatisoitiin dns-01-haasteen avulla (8).

6 Liiketoiminta

Jopa paras tekninen ratkaisu on valitettavasti vasta puolet kestävän yritystoiminnan perustamisessa. Sen pitää myös myydä. Seuraavaksi haluaisin tutkia projektin liiketoiminnallisia ja taloudellisia puolia.

Asiakasyhteydenottoja tuli eri puolelta maailmaa: Suomi, Viro, Ruotsi, Tanska, Intia, Saudi-Arabia, Sveitsi, Uusi Seelanti ja Etelä-Afrikka. 2013 - 2018 liiketulos oli yhteensä vajaat 100 000 EUR, eli noin 1600 EUR kuussa, tai noin 10 EUR tunnissa, jos työviikoksi lasketaan 40 tuntia. Todellisuudessa tehtiin usein 100 tunnin työviikkoja, joten keskimääräinen tuntipalkka oli lähemmäs 3 euroa. Tarkkaa työaikakirjanpitoa ei pidetty ja työ oli pitkiä jaksoa lähes ympärivuorokautista – päivällä koodattiin, illalla päivitettiin ja aamulla seurattiin päivityksen vaikutuksia.

6.1 Hinnoittelu

Hinnoittelu on yksi vaikeimmista kysymyksistä ohjelmistoyrityksen pyörittämisessä. Pääsyy tähän on viitearvojen puuttuminen - jos kyseessä on uusi tuote, jolle ei löydy kilpailijaa niin sille on myös täysin mahdotonta antaa “kilpailukykyistä” hintaa. Tuotteen arvon päätteleminen sen tuomasta taloudellisesta hyödystä on myös mahdotonta koska se riippuu ennen kaikkea siitä miten hyvin asiakas hyödyntää kyseistä työkalua sekä asiakkaan yritystoiminnan mittakaavasta.

Liiketoiminnan alkuvaiheessa harkittiin useita vaihtoehtoja sopivan hinnoittelumallin löytämiseksi. Vaihtoehtoina olivat:

- Maksu per käyttäjä per kuukausi
- Maksu per huoltotilaus per kuukausi
- Provisio kuukaudessa tehdystä laskutetusta työstä
- Kiinteä hinta ilman kuukausimaksua (asiakas vastaa itse järjestelmän ylläpidosta)
- Kiinteät paketoidut kuukausihinnat (rajoitettu ominaisuuksilla, asiakastuen kattavuudella tai käyttäjämäärällä)

Jokaisella vaihtoehdolla oli omat hyvät ja huonot puolet. Tietokonehuollot ovat hyvin epäyhtenäinen asiakaskunta, joiden koot ja taloudelliset resurssit vaihtelevat parin hengen perheyrityksistä isoihin kansainvälisiin firmoihin. Pienet asiakkaat tarvitsivat kuitenkin usein paljon enemmän tukea koska heillä ei ollut esimerkiksi henkilökuntaa, joka olisi hoitanut tekniikoiden ja myyjien perehdyttämisen tai integraatiot yrityksen omiin järjestelmiin.

Hinnoittelussa haluttiin ennen kaikkea välttää asiakkaiden eriarvoistumista, joten lopulta päädyttiin yksinkertaiseen kiinteään kuukausimaksuun (650 EUR, ALV 0%) ilman rajoituksia. Kaikki järjestelmän ominaisuudet olivat käytössä kaikille asiakkaille. Joillekin asiakkaille hinta oli liian korkea, joillekin taas liian matala (epäsuhteessa muihin yritysohjelmistojen hintoihin).

6.2 Järjestelmän räätälöinti

Vaikka Servo oli alun perin suunniteltu hyvin helposti muokattavaksi kattavilla asetuksilla niin kaikki asiakkaat tarvitsivat silti jonkin asteen apua järjestelmän räätälöinnin kanssa. Yleisin räätälöintipyyntö oli aina tulostuspohjien muokkaaminen. Tämä olisi ollut potentiaalinen lisätulon lähde, mutta koska uusia asiakkaita oli hyvin vaikea löytää ja vielä vaikeampi saada, luvattiin käyttöönotto ja räätälöinti aina kaupan päälle. Tähän kuului usein myös asiakkaan tilauskannan tuominen vanhasta järjestelmästä uuteen.

Tämä oli suoraan sanottuna strateginen virhe koska räätälöintityötä tuli hyvin paljon ja siihen kului lopulta myös paljon aikaa. Jälkeenpäin katsottuna räätälöinnin myyminen, vaikka esimerkiksi tuntilaskutuksella olisi ollut parempi ratkaisu ja antanut toiminnalle kestävämmän taloudellisen pohjan.

6.3 Projektihallinta

Prioriteetit voivat muuttua hyvinkin nopeasti, jos projektin ainoa kehittäjä vastaa myös asiakaspalvelusta, joten tärkein kriteeri projektihallintatyökalulle on juuri sen ketteryys ja mahdollisimman lyhyt etäisyys itse ohjelmointityökaluista. Lukuisten eri työkalujen kokeilun jälkeen todettiin, että yksinkertainen tekstitiedosto projektin juuressa toimii tähän tarkoitukseen parhaiten.

Asiakaspalvelu hoidettiin sähköpostilla ja tärkeät sähköpostit liputettiin, jotta niihin olisi helpompi palata. Tämä järjestely osoittautui täysin riittäväksi näin pienessä kehitystiimissä.

Valtaosa projektihallinnan haasteista poistuu kokonaan, jos tuotteesta vastaa vain yksi henkilö. Tämä on yksi yksinyrittäjyyden suurista vahvuuksista – asiakkaalla on aina yksi auktoritatiivinen kontakti toimittajaan ja toimittajan ei tarvitse käyttää lainkaan aikaa ja resursseja yhteisten toimintatapojen suunnitteluun ja ylläpitoon.

2010-luvulla julkaistiin lukuisia uusia projektihallintatyökaluja mutta niitten testauksessa todettiin, että ne eivät luo juuri lainkaan lisäarvoa, jos projektissa on vain yksi osallistuja.

6.4 Nimi, brändi ja tavaramerkit

Tuotteen nimeäminen voi olla yksi hankalammista päätöksistä ja hyvä idea voi lopulta tulla hyvinkin yllättävästä lähteestä. Servon nimi tuli Martti Servo ja Napander-yhtyeen nimestä. Nimi osoittautui sopivaksi myös sanan alkuperäisessä tarkoituksessa (2) ja koska se muistuttaa englannin kielistä sanaa “service”.

Koska kyseessä on yleinen termi, sille ei voitu hakea tavaramerkkiä, joten eurooppalainen tavaramerkki saatiin nimelle ServoApp (eli “Servo-niminen sovellus”). Vasta käyttöönoton jälkeen huomattiin, että sillä nimellä löytyy myös Windows-sovellus ServoApp.exe jota on myös joskus luokiteltu haittaohjelmaksi. Tämä aiheutti jonkun verran hämmennystä asiakkaiden keskuudessa, jotka viittasivat tuotteeseen juuri nimellä ServoApp.

Tuotteen logo tilattiin graafiselta suunnittelijalta, joka oli mm. suunnitellut Twitterin Bootstrap-kirjaston alkuperäiset kuvakkeet.

Kuva 12. Servon tuotegrafiikkaa.

Tuotteen kotisivut (1) ohjelmoitiin itse ja niitten pohjana käytettiin kaupallista HTML pohjaa. Sivut toteutettiin mahdollisimman yksinkertaisesti staattisina HTML-sivuina.

6.5 Markkinointi

Markkinoinnille jäi kehitystyön ohella aivan liian vähän aikaa ja resursseja. Asiakastiedotusta varten perustettiin postituslista, johon kerääntyi vuosien varrella tuotteen kotisivun kautta parisen sataa jäsentä, mutta listalle ei ehditty lähettää yhtäkään viestiä koska virhekorjaukset ja asiakastuki veivät niin paljon aikaa.

Valtaosa uusista asiakaskontakteista tuli kotisivun kautta (asiakas sattui etsimään tämänkaltaista järjestelmää), mutta kaikki ostopäätökseen johtavat kontaktit tulivat aina jonkun henkilökohtaisen kontaktin (joko kehittäjän tai asiakkaan suosituksen) kautta.

Pääasiallinen tapa lähestyä uusia asiakkaita oli sähköposti ja prosessi oli äärimmäisen turhauttava. Jopa huoltopäälliköt, joita tunnettiin henkilökohtaisesti, jättivät usein kokonaan vastaamatta yhteydenottopyyntöihin tai vastasivat muutaman kuukauden viiveellä. Tämä jatkuva epävarmuus ja näköalattomuuden tunne oli hyvin rasittavaa ja jopa henkisesti raastavaa.

Google-mainontaan käytettiin vuonna 2015 sata euroa. Se summa kului alle viikossa eikä tuottanut minkäänlaista merkittävää tulosta.

6.5.1 Tuote-esittelyt

Tuotedemoja järjestettiin sekä asiakkaan tiloissa että etänä, sekä Suomessa että ulkomailla. Asiakkaat tuntuivat aina olevan innostuneita järjestelmän mahdollisuuksista, mutta innostus ei valitettavasti johtanut kovinkaan usein ostopäätökseen. Siihen voi olla monta syytä:

- Huoltotoiminta oli usein myynnin sivubisnes eikä saanut sen takia tarpeeksi resursseja.
- Henkilöt, jotka tekivät lopulta ostopäätökset eivät olleet mukana demotapahtumassa.
- Huoltotoiminta oli niin heikolla taloudellisella pohjalla, että lisäinvestoinneille ei ollut varaa, vaikka ne olisikin pitkällä tähtäimellä parantanut kannattavuutta.
- Firmalla oli jo talon sisällä kehitetty järjestelmä, josta ei haluttu kuitenkaan päästää irti (esim. jos sen kehittäjä oli vielä firman työntekijä).

Olla samanaikaisesti järjestelmän myyjä ja kehittäjä oli psyykkisesti rasittavaa koska roolit vaativat täysin eri lähestymistapaa ja painotusta. Kehittäjän vetämät demot jumiutuvat liian helposti hänelle tärkeisiin teknisiin yksityiskohtiin, jotka eivät välttämättä ole niin tärkeitä asiakkaalle. Asiakkaan kysymyksiin vastatessa on aivan liian helppoa mennä liikaa yksityiskohtiin tai paljastaa liikaa järjestelmän puutteista.

6.6 Laskutus ja kirjanpito

Asiakkaita laskutettiin kerran vuodessa ja laskutus hoidettiin lähinnä freelance-yrittäjille tarkoitetulla laskutustyökalulla. Prosessi vei aika vähän aikaa ja sen ehti hoitaa hyvin ohjelmointityön ohella.

Varsinainen kirjanpito ulkoistettiin kirjanpitoyritykselle, joka hoiti myös vuosittaisen tilipäätöksen laatimisen.

6.7 Rahoituksen ja liikekumppanien hakeminen

Vuonna 2013 osallistuttiin Slush startup-tapahtumaan, jonka kautta toivottiin löytää rahoittajia ja sopivia yhteistyökumppaneita. Tämä ei valitettavasti tuottanut tulosta koska koko vastuu kontaktien löytämiselle oli jätetty yrittäjälle. Tapahtuma oli myös aivan liian iso satunnaisille tapaamisille. Kokemus oli suuri pettymys ja tapahtuma on selvästi suunnattu enemmän yrityksille, jolla on takana jo isompi tiimi.

Koska kohdeyleisö oli kuitenkin hyvin selvä - valtuutetut Apple-huollot - etsittiin jatkuvasti kanavia, joiden kautta heitä voitaisiin tavoittaa. Huollot ja varsinkin Apple-huollot ovat melko suljettu piiri. Markkinat ovat pienet, katteet suhteellisen matalat, ja huoltojen välinen kommunikointi hyvin vähäistä. Alalla ei ole omia merkittäviä messutapahtumia tai lehtiä, joita voisi käyttää jonkinlaisena foorumina. Apple oli yhtenäinen nimittäjä, mutta yrityksen sisäinen politiikka ei valitettavasti sallinut kolmannen osapuolen tuotteiden mainostamista huoltoverkostolleen.

Yhdeksi poikkeukseksi osoittautui koulutustoiminta. Kaikkien valtuutettujen huoltojen on koulutettava tietty määrä sertifioituja tekniikoita ja tästä vastaa pohjoismaissa yksi ainoa yritys - Hollannissa sijaitseva LAI (3). Vuonna 2016 solmittiin LAI:n kanssa yhteistyösopimus, joka antoi heille käytännössä virallisen jälleenmyyjän aseman. LAI suosittelisi järjestelmää omille asiakkaille ja saisi provision jokaisesta myydystä asennuksesta.

Käytännössä tämä järjestely ei kuitenkaan edistänyt lainkaan myyntiä. Sopimuksen jälkeen vaihdettiin muutamia sähköpostiviestejä, mutta mitään demoja ei koskaan järjestetty eikä järjestely johtanut uusiin asiakkaisuuksiin.

6.8 Lähdekoodin julkaiseminen

Servon lähdekoodi päätettiin julkaista loppuvuodesta 2013. Asiakkaille tätä perusteltiin ennen kaikkea riskien minimoimisella koska lähdekoodin

hallussapito varmistaa kriittisen järjestelmän toimivuuden myös siinä tapauksessa, että kehittäjä lopettaa toimintansa. Tämä avasi mahdollisuuden myös lähdekoodin auditoinnille itsenäisesti asiakkaan toimesta. Lähdekoodin julkaiseminen voi myös tuoda esille haavoittuvaisuuksia tuotteen tietoturvassa mutta tämä vaara ei käytännössä koskaan toteutunut.

Päätöksestä oli lopulta valitettavan vähän käytännön hyötyä. Git-arkisto (30) keräsi kymmeniä kopioita ja seuraajia, mutta aktiivista kehittäjäyhteisöä ei muodostunut kahdeksan vuoden aikana ollenkaan. Koodin julkaiseminen ei myöskään tuonut uusia asiakkaita. Tähän mennessä on tiedossa myös yksi tapaus, jossa konsulttifirma oli ottanut Servon lähdekoodin ja myynyt sen eteenpäin huoltofirmalle, joka oli täysin sallittua käytetyn BSD-lisenssin (19) puitteessa.

GSX-kirjaston (gsxlib) lähdekoodi (32) oli julkaistu jo vuonna 2011 ja kokemus sen kanssa on ollut samankaltainen. 10 vuoden tulos oli kymmeniä seuraajia ja kloonattuja Git-arkistoja ja vain muutamia ulkopuolisia parannuksia. Kirjaston Python-version (33) kanssa kävi samalla tavalla. Molemmista saatiin satunnaisia virheraportteja, joista valtaosa oli ongelmia käyttäjän GSX-tilin kanssa.

Vaikka lähdekoodin julkaiseminen ei tuottanut mitään taloudellista hyötyä, se tuotti kuitenkin mielihyvää. Jakaminen on yksi ihmisen perustarpeista, joka korostuu erityisesti avoimen lähdekoodin yhteisössä. Mikään tässä työssä mainituista projekteista ei olisi toteutunut ilman vapaita ohjelmistoja ja puhtaasti inhimillisellä tasolla tuntuu hyvältä antaa jotain takaisin, kun on itse saanut niin paljon näin pitkään.

Lähdekoodin julkaisemisesta ei myöskään ollut mitään haittaa ja yksikään asiakas ei lopettanut tuotteesta maksamista, vaikka he olisivat voineet käyttää sitä myös ilmaiseksi. Tämä vahvistaa havainnon, että asiakasta ei loppukädessä kiinnosta niinkään lähdekoodi vaan ratkaisu hänen ongelmaan.

7 Yhteenveto

Oma menestyvä yritys on unelma, josta suuri osa meistä haaveilee jossain vaiheessa työelämäämme. Ohjelmointi on ehkä paras väylä tämän unelman toteuttamiseen koska työn tulokset voivat parhaimmillaan koskettaa hyvin suuria asiakasmääriä ja luoda heille valtavasti arvoa. Ihmiskunnan historiassa ei todennäköisesti ole koskaan ollut alaa, jossa yksi henkilö voi vaikuttaa niin monen ihmisen elämään yhtä lyhyessä ajassa.

Ohjelmistoala on hyvä markkina yksinyrittäjälle koska toiminnan aloittamiseen tarvitaan hyvin vähän investointeja ja tärkein resurssi on yrittäjän oma työpanos ja ammattitaito. Yritystoimintaa voi myös harjoittaa päivätyön ohella, ainakin ihan alkuvaiheessa.

Servon kaltaiset yritysjärjestelmät ovat yksinyrittäjälle sekä suuri mahdollisuus että uhka. On luonnollista ja jopa hyvin todennäköistä että tietyllä alalla pitkään toiminut ammattilainen näkee mahdollisuuden paremmille työkaluille, mutta paremman työkalun keksiminen ei valitettavasti riitä kestävään yritystoimintaan. Tuotteelle pitää löytää myös oikeat ostajat ja sitä täytyy tukea ja ylläpitää vuosikausia, mahdollisesti jopa vuosikymmeniä, jos tuote on tärkeä. Jos kehittäjän pääasiallinen motiivi on keksiä parempia työkaluja, voi kaikki tämä muu toiminta alkaa tuntua hyvinkin raskaalta ja viedä häneltä työkyvyn. Tämän takia olisi hyvin tärkeää löytää parempia rahoitus - ja yhteistyömalleja, jotka auttaisivat yksinyrittäjää kasvattamaan liiketoimintaa orgaanisesti. Yksi taitava ohjelmistosuunnittelija pystyy toteuttamaan hyvän tuotteen, mutta hän tulee hyvin todennäköisesti tarvitsemaan apua yritystoiminnan jatkamiseen.

Myös verotuksella on tässä tärkeä rooli. Jos yksinyrittäjä on onnistunut käynnistämään liiketoiminnan ilman lisävelkaa ja pystyy juuri ja juuri elättämään sillä itsensä niin onko kohtuullista viedä noin puolet hänen tuloista? Todennäköisesti ei, varsinkin jos se johtaa siihen, että yritys ei pysty jatkamaan toimintaa.

Ohjelmointi on kognitiivisesti erittäin kuormittavaa ja vaatii valtavasti keskittymiskykyä. Työ voi olla myös poikkeuksellisen yksinäistä. Tämän takia on myös erityisen tärkeätä huolehtia yksinyrittäjän mielenterveydestä. Ongelmien kanssa yksin jäänyt yrittäjä ei jossain vaiheessa enää osaa pyytää apua.

Muutos on ja tulee vielä pitkään olemaan IT-alan ainoa vakio. Teknologioita ja trendejä tulee ja menee, mutta se ei tarkoita sitä, että insinöörin täytyy jatkuvasti seurata niitä. Tarkemmin sanoen, trendejä seuratessa kannattaa aina muistaa tietty turvaetäisyys. Paljon tärkeämpää on keskittyä perusasioihin – hyvään suunnitteluun ja työn laatuun. Kaikista tärkein tekijä tuotteen suunnittelussa on ja tulee aina olemaan ihminen ja paras työkalu järjestelmän toteutukseen on se, jonka insinööri tuntee parhaiten, riippumatta siitä kuinka trendikäs se mahtaa olla juuri sillä hetkellä.

Lähteet

(1) Welcome | Servo - Service Management for Authorized Apple Service Providers. Verkkodokumentti. <https://servoapp.com/>. Luettu 14.12.2021.

(2) Servo – Wikipedia. Verkkodokumentti. <https://fi.wikipedia.org/wiki/Servo>. Luettu 14.12.2021.

(3) Apple certifications – LAI. Verkkodokumentti. <https://www.lai.nl/certifications/apple-certifications/?lang=en>. Luettu 19.12.2021.

(4) ServoApp Help. Verkkodokumentti. <https://docs.servoapp.com/>. Luettu 20.12.2021.

(5) Bootstrap, from Twitter. Verkkodokumentti. <https://getbootstrap.com/2.3.2/examples/starter-template.html>. Luettu 20.12.2021.

(6) Swicegood Travis, Pragmatic Version Control Using Git, 2008

(7) Using the API. Verkkodokumentti. <https://docs.servoapp.com/using-the-api/>. Luettu 04.01.2022.

(8) Welcome to certbot-dns-dnsmadeeasy’s documentation! — certbot-dns-dnsmadeeasy 0 documentation. Verkkodokumentti. <https://certbot-dns-dnsmadeeasy.readthedocs.io/en/stable/>. Luettu 02.02.2022.

(9) Kannel User's Guide. Verkkodokumentti. <https://kannel.org/download/kannel-userguide-snapshot/userguide.html>. Luettu 19.01.2022.

(10) API Docs – SMSAPI. Verkkodokumentti. <https://www.smsapi.com/docs/#sms-api-documentation-1-introduction>. Luettu 10.01.2022.

(11) The uWSGI project — uWSGI 2.0 documentation. Verkkodokumentti. <https://uwsgi-docs.readthedocs.io/en/latest/>. Luettu 12.01.2022.

(12) Segmentation fault under FreeBSD 8.0 · Issue #21 · nodejs/node-v0.x-archive · GitHub. Verkkodokumentti. <https://github.com/nodejs/node-v0.x-archive/issues/21>. Luettu 14.01.2022.

(13) Using the Django authentication system | Django documentation | Django . Verkkodokumentti. <https://docs.djangoproject.com/en/1.10/topics/auth/default/#permissions-and-authorization>. Luettu 17.01.2022.

(14) Home - Django REST framework. Verkkodokumentti. <https://www.django-rest-framework.org/>. Luettu 17.01.2022.

(15) Subramanian, Hunt, Practices of an Agile Developer, 2008.

(16) Changelog. Verkkodokumentti. <https://docs.servoapp.com/changelog/>. Luettu 19.01.2022.

(17) FileMaker – Wikipedia. Verkkodokumentti. <https://en.wikipedia.org/wiki/FileMaker>. Luettu 23.01.2022.

(18) The contenttypes framework | Django documentation | Django. Verkkodokumentti. <https://docs.djangoproject.com/en/1.10/ref/contrib/contenttypes/#generic-relations>. Luettu 25.01.2022.

(19) The 2-Clause BSD License | Open Source Initiative. Verkkodokumentti. <https://opensource.org/licenses/BSD-2-Clause>. Luettu 26.01.2022.

(20) PHP: SOAP – Manual. Verkkodokumentti. <https://www.php.net/manual/en/book.soap>. Luettu 14.12.2021.

(21) PEP 3333 -- Python Web Server Gateway Interface v1.0.1 | Python.org. Verkkodokumentti. <https://www.python.org/dev/peps/pep-3333/>. Luettu 26.01.2022.

(22) The Selenium Browser Automation Project | Selenium. Verkkodokumentti. <https://www.selenium.dev/documentation/>. Luettu 27.01.2022.

(23) Why ninety-day lifetimes for certificates? - Let's Encrypt. Verkkodokumentti. <https://letsencrypt.org/2015/11/09/why-90-days.html>. Luettu 18.01.2022.

(24) Unbound DNS Server Tutorial @ Calomel.org. Verkkodokumentti. <https://calomel.org/unbound_dns.html>. Luettu 28.01.2022.

(25) Django’s cache framework | Django documentation | Django. Verkkodokumentti. < https://docs.djangoproject.com/en/1.8/topics/cache/#memcached>. Luettu 17.01.2022.

(26) RQ: Documentation Overview. Verkkodokumentti. <https://python-rq.org/docs/>. Luettu 08.02.2022.

(27) Parsing XML and HTML with lxml. Verkkodokumentti. <https://lxml.de/parsing.html>. Luettu 07.02.2022.

(28) Adjacency list – Wikipedia. Verkkodokumentti. <https://en.wikipedia.org/wiki/Adjacency_list>. Luettu 15.02.2022.

(29) Nested set model – Wikipedia. Verkkodokumentti. <https://en.wikipedia.org/wiki/Nested_set_model>. Luettu 15.02.2022.

(30) git.fps.ee Git - Servo/.git/tree. Verkkodokumentti. <https://git.fps.ee/?p=Servo/.git;a=tree>. Luettu 20.02.2022.

(31) git.fps.ee Git - gsx-mockserver/.git/tree. Verkkodokumentti. <https://git.fps.ee/?p=gsx-mockserver/.git;a=tree>. Luettu 20.02.2022.

(32) git.fps.ee Git - gsxlib/.git/tree. Verkkodokumentti. <https://git.fps.ee/?p=gsxlib/.git;a=tree>. Luettu 20.02.2022.

(33) git.fps.ee Git - py-gsxws/.git/tree. Verkkodokumentti. <https://git.fps.ee/?p=py-gsxws/.git;a=tree>. Luettu 20.02.2022.

(34) RFC 3875 - The Common Gateway Interface (CGI) Version 1.1. Verkkodokumentti. <https://datatracker.ietf.org/doc/html/rfc3875>. Luettu 18.02.2022.

(35) git.fps.ee Git - main /.git/tree. Verkkodokumentti. <https://git.fps.ee/?p=main/.git;a=summary>. Luettu 20.02.2022.

(36) git.fps.ee Git - Servo/.git/blob - servo/stats/queries.py. Verkkodokumentti. <https://git.fps.ee/?p=Servo/.git;a=blob;f=servo/stats/queries.py;h=df93146e3b7e436bac034a53203c507791aa7d45;hb=HEAD>. Luettu 20.02.2022.

(37) git.fps.ee Git - Servo/.git/blob - servo/models/common.py. Verkkodokumentti.

<https://git.fps.ee/?p=Servo/.git;a=blob;f=servo/models/common.py;h=2c7fa082eca9b6bc807bf3f1bdb36c797134adfc;hb=HEAD#l658>. Luettu 20.02.2022.

(38) Django MPTT documentation — django-mptt 0.13.4 documentation. Verkkodokumentti. <https://django-mptt.readthedocs.io/en/latest>/. Luettu 20.02.2022.

(39) git.fps.ee Git - Servo/.git/blob - servo/tests/test_functional.py. Verkkodokumentti. <https://git.fps.ee/?p=Servo/.git;a=blob;f=servo/tests/test_functional.py;h=f62a26e6492b589dfc2b782e53fd64ae2baf1a69;hb=HEAD>. Luettu 20.02.2022.

(40) git.fps.ee Git - gsxlib/.git/blob - runtests.php. Verkkodokumentti. <https://git.fps.ee/?p=gsxlib/.git;a=blob;f=runtests.php;h=90523e26a69d8de50cd27cc72b7db5cdc5d0a697;hb=HEAD>. Luettu 20.02.2022.

(41) git.fps.ee Git - py-gsxws/.git/blob - tests/test_gsxws.py. Verkkodokumentti. <https://git.fps.ee/?p=py-gsxws/.git;a=blob;f=tests/test_gsxws.py;h=201a066bc8c0f2f9bbaaab9bcfc2e431ffbba6a4;hb=HEAD>. Luettu 20.02.2022.

(42) freebsd-update. Verkkodokumentti. <https://www.freebsd.org/cgi/man.cgi?freebsd-update>. Luettu 20.02.2022.

(43) Remembering mcare - unflyingobject.com. Verkkodokumentti. <https://unflyingobject.com/posts/remembering-mcare/>. Luettu 23.02.2022.