Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Jaro Koski

Eläimen terveydenhuoltopaketti osana eläinklinikoiden hallintajärjestelmää

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

20.05.2019

Tekijä Otsikko Sivumäärä Aika	Jaro Koski Eläimen terveydenhuoltopaketti osana eläinklinikoiden hallintajärjestelmää 30 sivua 20.05.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Ilpo Kuivanen Projektipäällikkö Julius Leppälä

Insinöörityössä suunniteltiin ja toteutettiin laaja toiminnollisuus olemassa olevaan eläinklinikan toiminnanohjausjärjestelmään. Projekti tehtiin Finnish Net Solutions -yritykselle osana vakinaista Provet Cloud -järjestelmän kehitystyötä.

Tavoitteena oli järjestelmä, jolla pystyy luomaan, hallitsemaan ja myymään lemmikkieläimen terveydenhuoltopaketteja. Lisäksi oleellista oli toteuttaa järjestelmä, joka hoitaa kuukausittaisten laskujen generoimisen ja lähettämisen.

Teoriaosuudessa käydään läpi terveydenhuoltopakettijärjestelmän suunnittelemista ja kehittämistä, siihen liittyviä ongelmakohtia ja niiden ratkaisuja. Laskutusjärjestelmän kohdalla käydään läpi ajastettujen toimintojen toteutusta Django-ohjelmistokehyksessä.

Toiminnollisuus toteutettiin Python- ja JavaScript-ohjelmointikielillä Django-ohjelmistokehyksen päälle. Ajastettuihin toimintoihin käytettiin Celery-, Redis- ja Celerybeat tekniikoita.

Tuloksena saatiin järjestelmä, joka täyttää tavoitteessa asetetut vaatimukset, mutta jota tullaan jatkokehittämään tulevaisuudessa.

Avainsanat	Celery, Django, Python

Author Title Number of Pages Date	Jaro Koski Pet Health Plans as Part of Veterinary Practice Management Software 30 pages 20 May 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Software Engineering
Instructors	Ilpo Kuivanen, Senior lecturer Julius Leppälä, Project manager

The aim of this thesis was to design and implement a major feature for existing veterinary practice management software, Provet Cloud. This was done as part of regular development for Finnish Net Solutions.

The objective was to create a system that would allow creating, managing and selling pet health plans for customers. In addition, a system generating and sending invoices as email needed to be implemented to take care of the monthly billing.

The theory section covers health plan system planning as well as developing and overcoming surfaced obstacles. Furthermore, the paper looks into creating scheduled tasks on the Django software framework.

The feature was developed with Python and JavaScript languages on top of the Django software framework. The scheduled task managing was achieved with Celery, Redis and Celerybeat.

The study resulted in creating the pet health plan as a feature in Provet Cloud and it is going to be developed further in future.

Keywords	Celery, Django, Python

Sisällys

Lyhenteet

FNS	Finnish Net Solutions. Insinöörityön tilannut yritys.
JSON	JavaScript Object Notation. Yksinkertainen tiedonvälitys tiedostomuoto.
PMS	Practice management software. Klinikoiden toiminnanohjausjärjestelmä.
UI	User Interface. Sovelluksen käyttäjän näkymä.
URL	Uniform Resource Locator. Merkkijono, joka osoittaa www-sivuston lokaation.
UX	User Experience. Sovelluksen käyttäjäkokemus, siitä toimiiko sovellus inhimillisen logiikan mukaan.

1 Johdanto

Finnish Net Solutions on vuonna 2001 perustettu B2B-palveluja tarjoava ohjelmistoyritys. Yritys valmistaa pääasiallisesti kahta tuotetta, eläinklinikoiden toiminnanohjausjärjestelmää Provetia ja terapia-alan potilaskortistoa Diarumia. Kahden päätuotteen lisäksi FNS:ltä on mahdollista tilata omiin tarpeisiin sopivia räätälöityjä järjestelmiä. [1.]

Provet Cloud on FNS:n kehittämä kansainvälisille markkinoille suunnattu pilvipohjainen eläinklinikoiden toiminnanohjausjärjestelmä, jota yli 3000 eläinhoitoalan ammattilaista käyttää päivittäin. Provet Cloud on suunniteltu toimimaan minkä tahansa eläinklinikan tarpeisiin oli sitten kyse muutaman hengen klinikasta, suuresta sairaalasta tai yliopistosta. [1.]

Eläimen terveydenhuoltopaketti, englanniksi Health Plan, on kuukausittaiseen laskutukseen perustuva palvelu, jonka tarkoituksena on tarjota ennalta määritettyjä eläimen terveyttä edistäviä hoitoja. Terveydenhuoltopakettien sisältö riippuu palveluntarjoajasta, mutta sisältö on usein yksinkertaista ja mahdollisimman monelle lemmikkieläimelle sopivaa. Paketit sisältävät usein esimerkiksi mikrosirun, terveystarkastuksia, rokotteita ja muita tarvikkeita. [2.]

Terveydenhuoltopaketti toiminnallisuudesta on kysyntää Ison-Britannian ja Yhdysvaltojen markkinoilla. FNS pyrkii siihen, että Provet Cloud tarjoaa saumattoman terveydenhuoltopakettitoteutuksen Iso-Britannian markkinoille ensimmäisten joukossa, sillä kyseistä toiminnollisuutta on tiedusteltu useissa myyntineuvotteluissa.

Insinöörityö tehtiin osana Provet Cloudin kehitystyötä. Työn tavoitteena oli suunnitella ja kehittää sovellukseen integroitu järjestelmä, jolla eläinklinikan ylläpitäjä pystyvät luomaan ja hallitsemaan terveydenhuoltopaketteja sekä jolla eläinklinikan työntekijät kykenevät myymään tilauksia ja käyttämään tilaukseen sisältyviä tuotteita. Järjestelmän tuli sisältää myös kuukausittainen laskutusjärjestelmä sekä mahdollisuus hallita kyseisiä laskuja.

2 Provet Cloud

Provet on toinen Finnish Net Solutions Oy:n kehittämistä päätuotteista Diarium potilaskortistojärjestelmän ohella. Provet-ohjelmistot ovat eläinklinikan hallintajärjestelmiä, joita FNS on kehittänyt lähes kahden vuosikymmenen ajan. Suomessa FNS on markkinajohtaja alallaan Provet Net ja Provet Win tuotteillaan [3].

Provet Cloud on vuonna 2015 julkaistu kansainvälisille markkinoille suunnattu pilvipohjainen ohjelmisto, joka tarjoaa eläinlääkäriklinikoille päivittäiseen toimintaan tarvittavat työkalut ajanvarauksesta laskutukseen. Tavoitteena on tarjota asiakkaille mahdollisimman selkeä ja saumaton toteutus, jolla voidaan hoitaa kaikki eläinklinikan toiminnollisuudet. Järjestelmään on mahdollisuus integroida useita laboratorio- ja kuvantamislaitteita, printtereitä sekä maksupäätteitä. Provet Cloud tarjoaa asiakkailleen myös mahdollisuuden käyttää Cloudin dataa myös REST-apin kautta. [1.]

Provet Cloud:ia käyttää päivittäin yli 3000 eläinlääkinnän ammattilaista ympäri Eurooppaa. Ohjelmiston kehityksessä on panostettu siihen, että se vastaa kaikkien asiakkaitten tarpeita, oli kyseessä sitten muutaman hengen pieneläinklinikka, kymmeniä ihmisiä työllistävä eläinsairaala tai yliopisto. [1.]

Koska Provet Cloud on kansainvälisille markkinoille suunnattu ohjelmisto ja FNS:n tavoitteena on saavuttaa Euroopan markkinajohtajuus, Provet Cloudin on tärkeää pystyä mukautumaan alueellisiin lakisääteisiin vaatimuksiin sekä pyrkiä täyttämään markkinan tarpeet.

Varsinkin isobritannialaisten ja yhdysvaltalaisten klinikoiden kanssa käydyissä myyntineuvotteluissa on noussut useasti esille kiinnostus kuukausittaiseen maksuun perustuvista eläinten terveydenhuoltopaketeista, englanniksi Health- tai Wellness plan. Useat klinikat ja ketjut tarjoavat tätä palvelua, mutta ilmeisen harva PMS-järjestelmä tarjoaa tähän ohjelmistoon integroitua, automaattisesti palvelun käytöstä kirjaa pitävää ratkaisua.

3 Eläimen terveydenhuoltopaketti

Eläimen terveydenhuoltopaketti on palvelupaketti, joka koostuu eläimen perusterveyden kannalta oleellisista sekä ennaltaehkäisevistä toimenpiteistä ja tuotteista. Palvelupaketin konkreettinen sisältö on palveluntarjoajakohtainen, mutta usein paketin sisältö on hyvin samankaltainen. Mitä yksinkertaisempi ja globaalimpi sen sisältö on, sitä enemmän sillä on mahdollista tavoittaa asiakkaita. Tyypillisesti paketit sisältävät tuotteita ja palveluja, kuten [2.]

- rutiini- ja vuosittaiset käynnit
- rokotteet
- sydänmatolääkitykset
- mikrosiru
- kirppu- ja punkkilääkitykset
- kuohinta/kastraatio.

Eläimen terveydenhuoltopaketti tarjoaa asiakkaalle mahdollisuuden jakaa eläimen terveyden ylläpidosta koituvat kulut edullisiin maksueriin, sillä palvelupaketti perustuu kuukausittaiseen tilausmalliin. Palvelua siis maksetaan kuukausittain, jolloin sitä voidaan käyttää tarvittaessa. Koska palvelu kannustaa lemmikin säännöllisiin eläinklinikkakäynteihin, se helpottaa asiakkaan proaktiivista lähestymistä eläimen terveyteen. [2.]

Yksi syy terveydenhuoltopakettitoiminnollisuuden kysyntään Isossa-Britanniassa on tilauspalvelujen suosio. YouGov:n vuonna 2018 tuottaman raportin mukaan Ison-Britannian väestöstä 82 % väestöstä on tilaa ainakin yhtä palvelua [4].

Eläinklinikalle terveydenhuoltopaketit tarjoavat mahdollisuuden pysyvän asiakassuhteen luomiseen. Terveydenhuoltopaketit ovat usein toimipaikkakohtaisia, joten terveydenhuoltopaketin myymällä saadaan asiakas sitoutumaan käyttämään kyseistä toimipistettä.

Pysyvän asiakassuhteen luomisen kannalta palvelun tulee olla asiakkaalle mahdollisimman hyödyllinen ja helppokäyttöinen. Tämän vuoksi on tärkeää löytää asiakkaan tarpeisiin sopivin, eli lemmikkieläimelle hyödyllisin palvelupaketti. Eläimelle sopivaa pakettia valittaessa tulee ottaa huomioon eläimen laji, ikä ja paino. Vähemmän

sopivan tilauksen myyntiä ei kumminkaan tule estää ohjelmiston puolella, sillä rajatapaukset esimerkiksi painon suhteen täytyy ottaa huomioon.

Palvelupaketin sisällön lisäksi oleellista on, miten asiakas käyttää palvelua ja kuinka helppoa se hänelle on. Tilanteesta, jossa asiakkaalta jää käyttämättä osa hänen maksamastaan palvelusta, jää helposti negatiivinen kokemus palvelun kannattavuudesta. Automatisoidut muistutukset auttavat asiakasta paketin käytössä muistuttamalla käyttämättömästä sisällöstä. Asiakkaan positiivinen kokemus palvelusta mahdollistaa tilauksen jatkuvuuden, samalla luoden pysyvää asiakassuhdetta klinikkaan.

4 Suunnittelu ja prototyyppi

4.1 Suunnitteluprosessi

Suunnitteluprosessi lähtee lähes aina ideasta. Idea voi syntyä niin sanottuna empatiana, mikä helpottaisi asiakkaan toimintaa, mutta idea voi myös syntyä tarpeesta. Tarve voi olla pyyntö asiakasyritykseltä tai tarve täyttää markkinan asettamat vaatimukset. Yrityksiltä saadut kehitystarpeet ovat usein yrityksen vakiintuneita toimitapoja tai työnkulkua tukevia ja edistäviä uudistuksia, kun taas markkinoiden luomat tarpeet ovat usein lakisääteisiä toiminnallisuuksia, tai toiminnallisuuksia, joita markkinalla toimivat kilpailijat jo tarjoavat.

Idea Provet Cloud -järjestelmän terveydenhuoltopaketteihin syntyi Ison-Britannian markkinoille laajentumisesta, sillä terveydenhuoltopaketit ovat siellä yleisiä ja Provet Cloudin markkina-aseman vahvistamiseksi haluamme tarjota saumattoman ratkaisun kyseisen palvelun hallintaan.

Kun idea toiminnallisuudelle on selvä, aletaan suunnittelemaan, miten kyseinen toiminnallisuus on mahdollista kehittää järjestelmään. FNS on jo aiemmin kehittynyt vastaavanlaisen kuukausittaiseen laskutukseen perustuvan terveydenhuoltopakettijärjestelmän Suomen markkinoille suunnattuun Provet Net -ohjelmistoon. Kyseisen toiminnollisuuden kehittänyt, nykyään Provet Cloudin pääkehittäjänä toimiva Julius Leppälä, osallistui oleellisesti tekniseen suunnitteluun.

Terveydenhuoltopaketti toiminnollisuuden tietokantatoteutus suunniteltiin käyttämällä viitteenä Provet Net -toteutusta. Pakettien toiminta käyttöliittymässä suunniteltiin kehittämällä prototyyppi yhteistyössä FNS:n design-tiimin kanssa.

4.2 Prototyyppi

Prototyyppi tarkoittaa varhaista koekappaletta, mallia tai julkaisua, jolla pyritään todistamaan ja testaamaan konseptin toimivuutta. Ohjelmistokehityksessä prototyypillä suunnitellaan ja kehitetään käyttöliittymän toiminnollisuuksia visualisoimalla se testihenkilöille ja -ryhmille. [5.]

Ohjelmiston sekä suurten toiminnollisuuksien kehittäminen vie aikaa ja rahaa, eikä lopullinen tulos välttämättä vastaa asiakkaan asettamia odotuksia. Koska prototyypin luominen on itse ohjelmistokehitystä nopeampaa ja vaivattomampaa, prototyypillä visualisoituja toiminnollisuuksia voidaan esittää asiakkaalle ja saada arvokasta palautetta käyttöliittymäratkaisuista jo ennen niiden konkreettista toteuttamista. [5.]

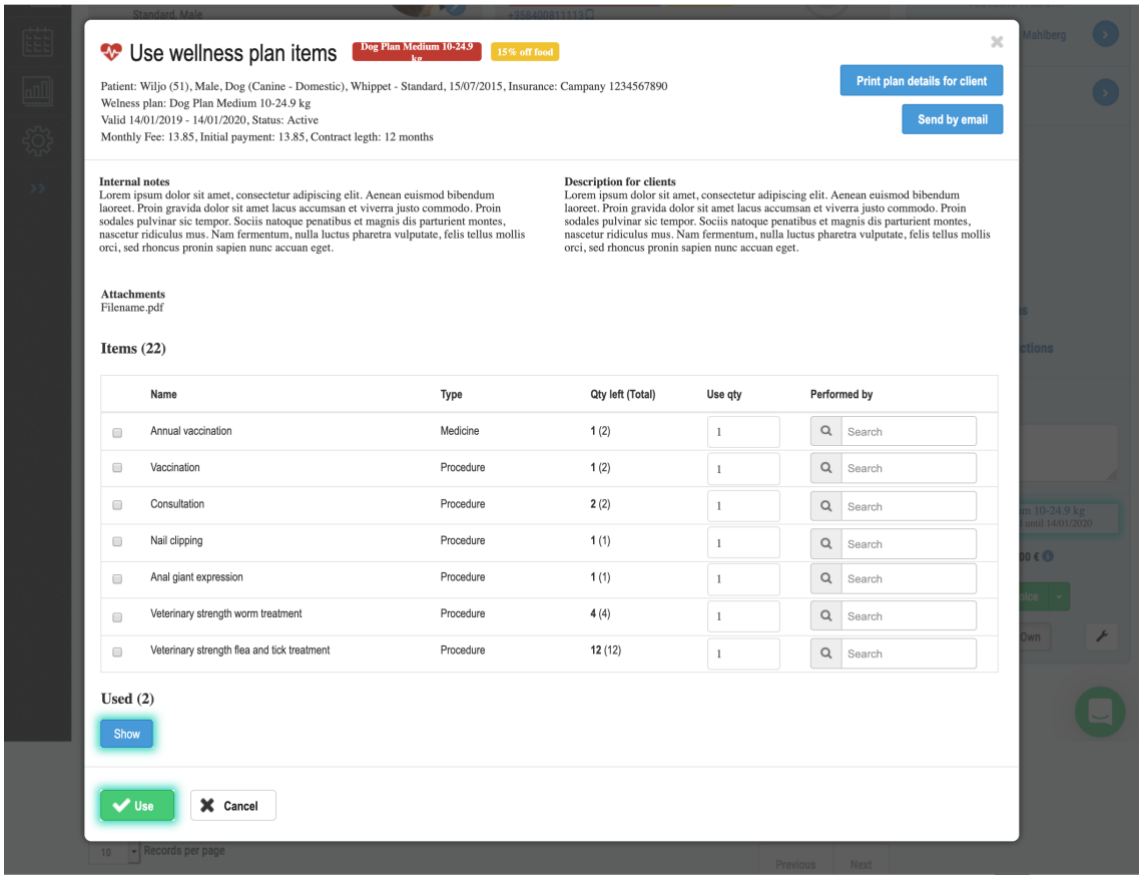
Eläinten terveydenhuoltopaketeista luotiin prototyyppi yhteistyössä FNS:n design-tiimin kanssa. Interaktiivinen prototyyppi luotiin käyttämällä pilvipohjaista Axure Share -ohjelmistoa. Prototyypin kehitti UX-designer, joka osallistui aktiivisesti terveydenhuoltopakettitoiminnallisuuteen liittyviin kokouksiin.

Suunnittelukokouksissa UX-designer esitteli prototyyppiä kokoukseen osallistujille, jonka jälkeen osallistujat antoivat palautetta toiminnallisuudesta. Prototyyppiä kehitettiin saadun palautteen perusteella ja päivitetty versio esiteltiin uudestaan seuraavassa kokouksessa, jotka järjestettiin muutaman viikon sykleissä.

Koska toiminnallisuutta kehitetään asiakkaiden tarpeisiin, on kannattavaa esitellä prototyyppiä myös asiakkaille sen edistyessä. Toiminnallisuudesta luotua prototyyppiä esiteltiin erinäisten asiakastapaamisten ja myyntineuvottelujen ohessa asiakkaille, joilta saatu palaute käytiin läpi kokouksissa yksityiskohtaisesti ja prototyyppiä muokattiin vastaamaan asiakkaan tarpeita.

Kuvassa 1 on kuvankaappaus prototyypistä, jossa kuvataan terveydenhuoltopakettituotteen käyttöä konsultaatiolla. Kuvaa 1 voidaan verrata sivulla

13 olevaan kuvaan 5, jossa esitetään kyseinen toiminnollisuus kehitettynä Provet Cloud -järjestelmään. Kuvia vertaamalla voidaan huomata, kuinka todenmukaista jälkeä Axure Share -prototyyppiohjelmalla voidaan kuvata.



Kuva 1. Kuvankaappaus Axure Share -ohjelmasta, terveydenhuoltopaketin tuotteen käyttäminen konsultaatiolla.

5 Terveydenhuoltopakettijärjestelmä

Eläimen terveydenhuoltopakettijärjestelmä koostuu kolmesta pääosiosta, joita ovat

- terveydenhuoltopakettisuunnitelma
- terveydenhuoltopaketin tilaus
- laskutus.

Terveydenhuoltopakettisuunnitelma kuvastaa palvelupaketteja, joita eläinklinikka tarjoaa asiakkaillensa. Suunnitelmaa kuvataan ohjelmistossa HealthPlan-mallina. Suunnitelmaan kuuluvat tuotteet kuvataan HealthPlanItem-mallina.

Terveydenhuoltopaketin tilaus luodaan asiakkaalle terveydenhuoltopakettisuunnitelman pohjalta. Ohjelmistossa tilaus kuvataan PatientHealthPlan-mallina. Tilauksen tuotteet perustuvat monimuotoiseen (Polymorphic) malliin PatientHealthPlanItem, jolla on neljä alaluokkaa PatientHealthPlanFood, PatientHealthPlaMedicine, PatientHealthPlanProcedure ja PatientHealthPlanSupply.

Kuukausittaista laskutusta varten luodaan järjestelmä, joka generoi ja lähettää laskuja asiakkaan sähköpostiin. Tilauksen luomisen yhteydessä luodaan laskutusta varten PatientHealthPlanPayment-objekteja, joiden perusteella luodaan laskuja ajastetusti. Laskutusjärjestelmään perehdytään tarkemmin luvussa 6: Ajastetut toiminnot.

5.1 Lähtökohdat

Provet Cloud on rakennettu Django-sovelluskehyksen päälle. Sovelluskehyksen pääohjelmointikieli on python, mutta koska kyseessä on webkehitysalusta, frontend-kehitys tapahtuu JavaScript-kielellä, AJAX-tekniikoita ja jQuery-kirjastoa hyödyntäen. Tämä takaa ohjelmiston käyttäjäpuolen dynaamisuuden, johon moderneiden sivustojen kanssa toimisessa on totuttu.

Palvelimen puolella Django tarjoaa monia ohjelmistokehitystä nopeuttavia ja helpottavia menetelmiä, kuten tietokantamallien luomisen ORM-tekniikalla. Tämä mahdollistaa tietokantamallien määrittelemisen python-koodilla, joista Djangon työkaluilla luodaan migraatio, josta malli luodaan tietokantaan. ORM mahdollistaa myös tietokanta-alkioiden kohtelemisen objekteina.

Jokaiselle laajalle Provet Cloud -toiminnalisuudelle luodaan oma Django- ”applikaatio”. Nimestä poiketen applikaatio ei tarkoita varsinaista sovellusta, vaan kokoelmaa python-tiedostoja, jotka tarjoavat toiminnallisuuden projektin kokonaisuuteen. Eläimen terveydenhuoltopaketin applikaatio nimettiin sen englanninkielisellä nimellä ”Health plan”. (Esimerkkikoodi 1.)

```
python manage.py startapp health_plan
```

Esimerkkikoodi 1. Terveydenhuoltopaketin applikaation luonti python komennolla.

Luotu kokoelma sisältää tiedostopohjat tarvittaville toiminnollisuuksille. Models.py-tiedostoon kirjoitetaan tietokantamallia kuvaavat luokat, joista Djangon tietokantamigraatiot luodaan.

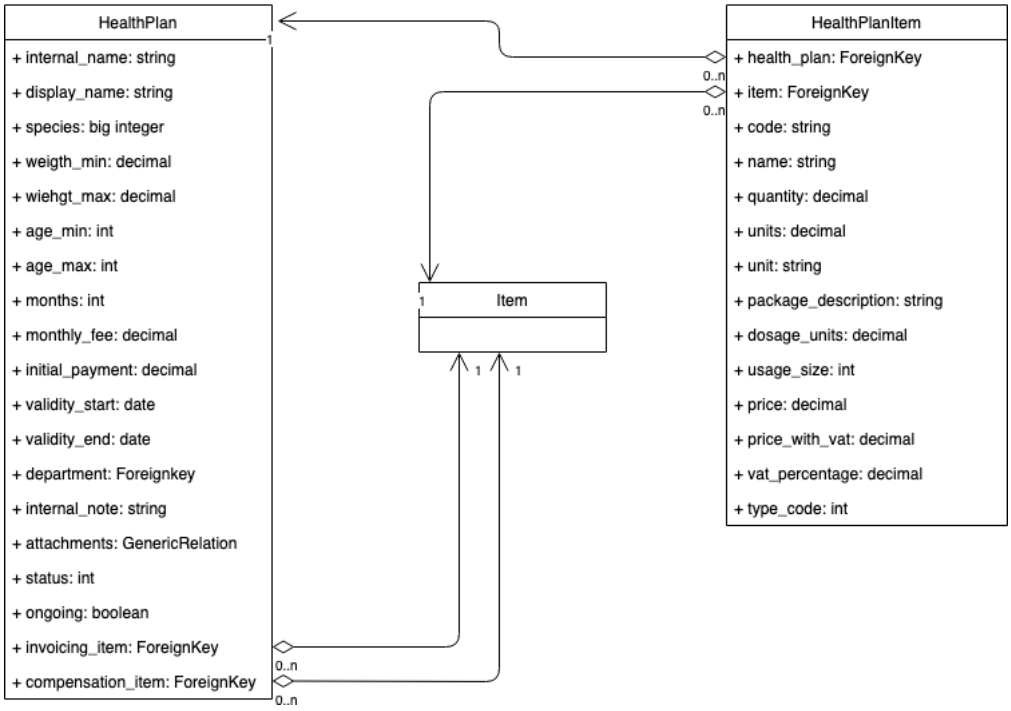
5.2 Terveydenhuoltopakettisuunnitelma

HealthPlan-tietokantamalli on viitekehys, joka sisältää paketin yleiset tiedot, myyntirajoitteet, alennukset sekä paketin laskutukseen liittyvät tiedot. HealthPlan-instanssia luodessa sille voidaan myös määrittää suunnitelmaan kuuluvia tuotteita, HealthPlanItem-mallista luotuja objekteja. Kuvankaappaus terveydenhuoltopaketin luomisnäkymästä on liitteenä 1.

Jokaiselle terveydenhuoltopakettisuunnitelmalle tulee määrittää eläinlaji, jolle paketti on soveltuva. Eläinlajin lisäksi sille määritettyjä ikä- ja painovälejä käytetään suositeltujen terveydenhuoltopakettien suodattamiseen tilausta luodessa.

Tehdessämme selvitystä jo Iso-Britannian markkinoilla olevista terveydenhuoltopalveluista huomasimme, että palvelu on lähes aina toimipistekohtaista, myös suurten eläinklinikkaketjujen tarjoama palvelu. Tämän vuoksi myös Provet Cloudin toteutus luotiin toimipistekohtaiseksi, käyttörajoitus yhteen toimipisteeseen toteutettiin viiteavaimella toimipistettä kuvaavaan Department-objektiin.

Kuvassa 2 on kuvattu HealthPlan- ja HealthPlanItem-mallin suhteita toisiinsa, sekä olemassa olevaan yleisesti tuotetta kuvaavaan Item-luokkaan.



Kuva 2. Kaavio HealthPlan- ja HealthPlanItem-tietokantamalleista

Jotta tuotteita voidaan lisätä terveydenhuoltopakettiin, on oleellista, että HealthPlan-mallista on ilmentymä. Tämän vuoksi uutta HealthPlan-ilmentymää luodessa palvelimelle lähetetään ensiksi Ajax POST -kutsu, jonka vastaanottava Django View'n tehtävänä on luoda uusi HealthPlan-ilmentymä ja palauttaa näkymälle luodun objektin muokkausnäkymän url JSON-vastauksena (esimerkkikoodi 2). Kun ilmentymä on luotu onnistuneesti, Ajax-kutsun "Success"-metodi uudelleenohjaa käyttäjän luonnin yhteydessä palautettuun url:iin.

```
def post(self, request, *args, **kwargs):
        health_plan = models.HealthPlan.objects.create(
            created_user=request.user,
            status=codes.PLAN_DRAFT
        )
        return JsonResponse({'redirect': reverse(
            'provet:settings:health_plans:edit_plan',
            kwargs={'provet_id': active_context.provet_id,
                    'plan_id': health_plan.id})}, safe=False)
```

Esimerkkikoodi 2. HealthPlan-objektin luonti ja uudelleenohjaus.

HealthPlanin muokkaus on jaettu kahteen lomakkeeseen HealthPlanInfoForm käsittelee yleiset tiedot sekä rajoitteet. HealthPlanPaymentForm käsittelee laskutukseen liittyvät tiedot, hinnan, keston, etukäteismaksun sekä laskutukseen ja hyvitykseen käytettävät toimenpideluokkaiset tuotteet. Kun HealthPlan on valmis

julkaistavaksi, lomaketiedot kootaan yhteen sarjallistamalla jonka jälkeen ne lähetetään palvelimelle, jossa lomakkeet käsitellään samassa näkymässä. Jako kahteen lomakkeeseen toteutettiin, sillä paketin tuotteiden lisäyksestä vastaava moduuli aiheutti sisäkkäisistä lomakkeista johtuvan tiedonkäsittelyongelman. Kahdella lomakkeella saatiin aikaan tilanne, jossa sisäkkäisiä lomakkeita ei synny.

HealthPlanItem kuvastaa pakettiin kuuluvia tuotteita. Provet Cloudissa tuotteet ovat toteutettuna monimuotoisena Item-mallina, jolla on neljä alaluokkaa: Medicine, Procedure, Supply ja Food. Paketin sisällön hallinnoimisen lisäksi HealthPlanItemin tarkoitus on säilöä tietoa jonka perusteella voidaan luoda PatientHealthPlanItem-ilmentymiä asiakkaalle myytyyn tilaukseen.

HealthPlanItem-tuotteet lisätään HealthPlaniin käyttämällä Provet Cloudissa yleisesti käytössä olevaa tuotteiden lisäysmoduulia. Moduuli hoitaa tuotteiden hakemisen AJAX-kutsuilla, sekä tuotteiden esittämisen UI:ssa. Kun tuote valitaan moduulista, sen tiedot lähetetään palvelimelle, jossa käsitellään tuotteen lisäys HealthPlaniin. Tuotteen lisäyksessä sille lasketaan, hinta, verot sekä käyttömäärät.

Terveydenhoitosuunnitelman toimipaikkakohtaisuuden kannalta on tärkeää tarkistaa, ettei pakettiin pysty lisäämään tuotteita, joita kyseisellä toimipaikalla ei ole varastossa. Tämän vuoksi lomakkeen validoinnissa tarkastetaan, onko pakettiin aiemmin lisätyt tuotteet samalta tuotelistalta kuin lisättävä tuote (esimerkkikoodi 3).

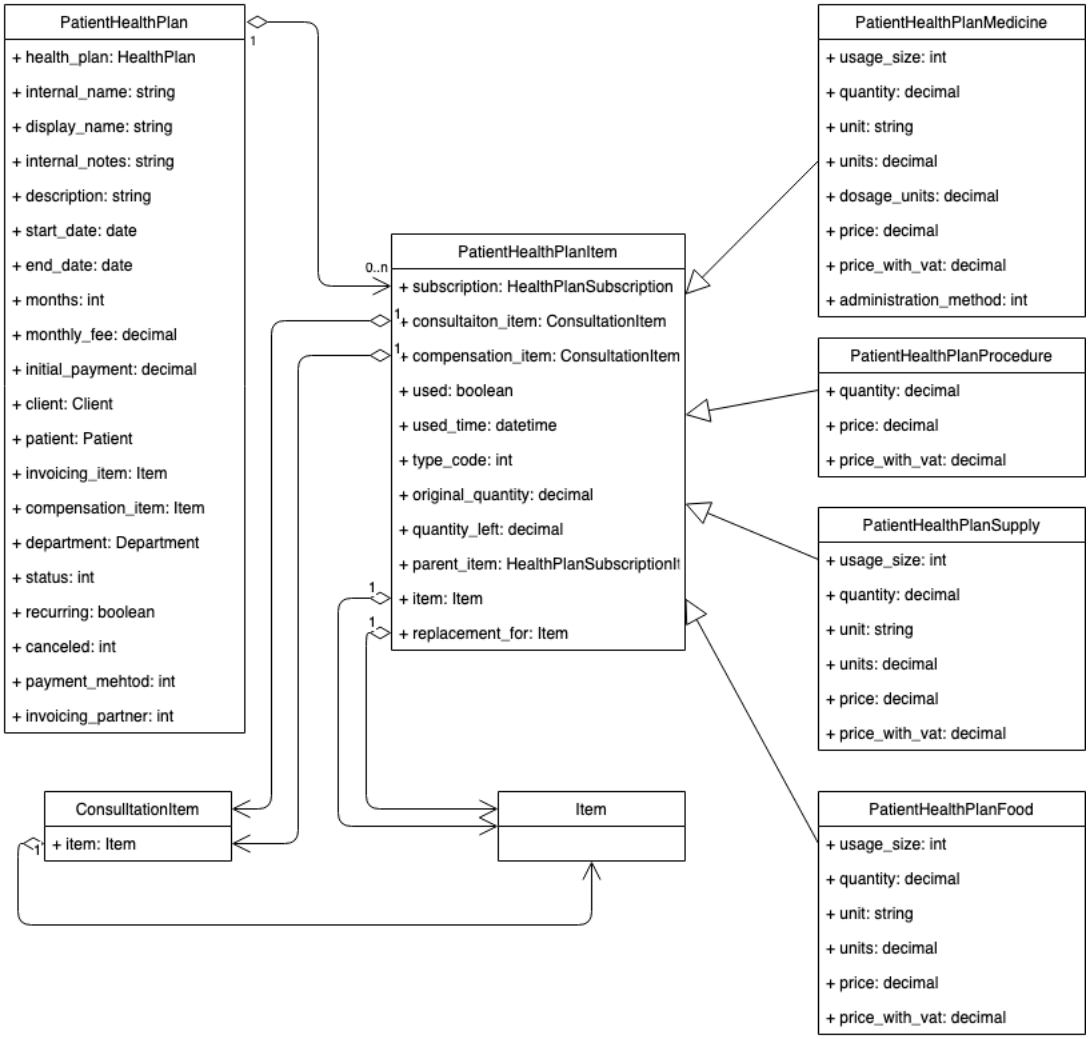
```
all_items = health_plan.healthplanitem_set.all()
        for plan_item in all_items:
            if plan_item.item.item_list_id != item.item_list.id:
                return JsonResponse({'item_list_error': True},
                                    status=400,
                                    safe=False)
```

Esimerkkikoodi 3. Toimipisteen tarkistus.

5.3 Terveydenhuoltopaketin tilaus

PatientHealthPlan on asiakkaalle luotu tilaus HealthPlan-mallista. PatientHealthPlan-objektiin kopioidaan sen hetkiset tiedot valitusta HealthPlan-mallista. Tämä sisältää yleistiedot, alennukset, liitteet sekä tuotteet. PatientHealthPlan-objektiin tallennetaan viiteavain HealthPlan-objektista. Kopioitujen tietojen lisäksi tilauksen luontilomakkeella välitetään informaatio tilaajasta, eläimestä sekä laskutustavasta.

Kuvassa 3 esitetään PatientHealthPlan- ja PatientHealthPlanItem-mallin sekä sen alaluokkien väliset tietokantasuhteet. Kaavion ConsultationItem kuvaa konsultaatiolla lisättyä Item-luokasta luotua tuotetta, joka luodaan PatientHealthPlanItemiin tallennetun Item-objektin pohjalta.



Kuva 3. Kaavio PatientHealthPlan- ja PatientHealthPlanItem-tietokantamalleista.

PatientHealthPlan-tilausinstanssia luodessa sille täytyy kopioida terveydenhuoltopaketin HealthPlanItem-tuotteet. Kuvassa 4 esitetään tuotteiden kopiointi tilaukseen. HealthPlanItem-taulusta haetaan pakettiin kuuluvat tuotteet, jotka muunnetaan PatientHealthPlanItem-alaluokiksi luomalla HealthPlanItemin kentistä avain-arvo-sanasto (dictionary). Mikäli kopioitava tuote on jotakin muuta luokkaa kuin toimenpide, sanastoon lisätään pakettikokoon liittyvät lisätiedot. PatientHealthPlanItemin alaluokka haetaan avain-arvo-sanastosta käyttämällä avaimena tuotteen tyyppikoodia. Kun oikea luokka on haettu, sille välitetään juuri luotu

sanasto, jonka Python-kieli kykenee purkamaan ja sen perusteella alustamaan objektin, joka tallennetaan tietokantaan.

```
def copy_plan_items(self, health_plan=None):
    class_mapping = {
        organization_codes.ITEM_TYPE_FOOD: PatientHealthPlanFood,
        organization_codes.ITEM_TYPE_SUPPLY: PatientHealthPlanSupply,
        organization_codes.ITEM_TYPE_MEDICINE: PatientHealthPlanMedicine,
        organization_codes.ITEM_TYPE_PROCEDURE: PatientHealthPlanProcedure,
    }
    plan = health_plan if health_plan else self.health_plan
    plan_items = plan.healthplanitem_set.all()
    for plan_item in plan_items:
        item_kwargs = {
            "health_plan": self,
            "item": plan_item.item,
            "usage_size": plan_item.usage_size,
            "quantity": plan_item.quantity,
            "price": plan_item.price,
            "price_with_vat": plan_item.price_with_vat,
            "type_code": plan_item.type_code,
            "original_quantity": plan_item.quantity,
            "quantity_left": plan_item.quantity
        }
        if plan_item.type_code != organization_codes.ITEM_TYPE_PROCEDURE:
            item_kwargs.update({
                "units": plan_item.units,
                "unit": plan_item.unit,
                "dosage_units": plan_item.dosage_units
            })
        class_mapping.get(plan_item.type_code).objects.create(**item_kwargs)
```

Kuva 4. HealthPlanItemin kopiointi tilaukseen.

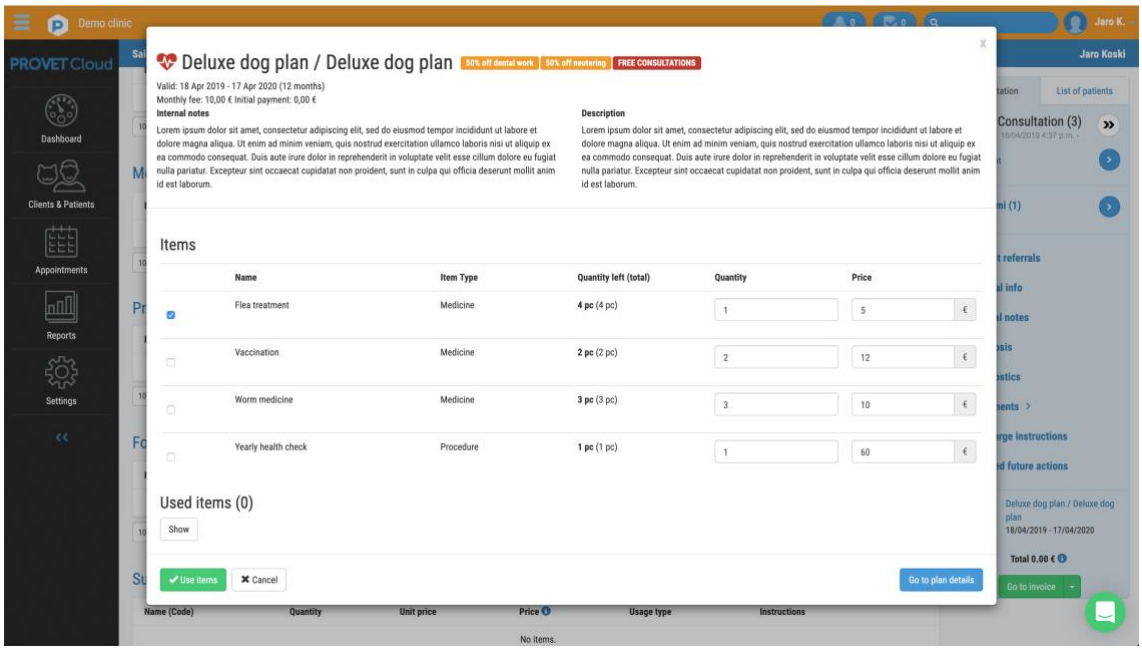
5.4 Tuotteiden käyttäminen konsultaatiolla

PatientHealthPlanItem säilyttää tiedon terveydenhuoltopakettitilaukseen kuuluvista tuotteista ja palveluista. PatientHealthPlanItem on monimuotoinen malli. Monimuotoisessa mallissa pääluokka toimii rajapintana sen alaluokille, joiden metoditoteutukset on luotu kyseiseen alaluokkaan ja malliin sopivaksi. PatientHealthPlanItemillä on neljä eri alaluokkaa, josta jokaisen toteutus perustuu Provet Cloudissa käytettävien tuotteiden (Item) malliin.

- PatientHealthPlanMedicine kuvastaa ohjelmistossa määritettyjä lääkkeitä. Lääkkeillä on enemmän kenttiä kuin muilla luokilla, sillä lääkkeiden käyttö vaatii tiettyä yksikkötarkkuutta.

- PatientHealthPlanProcedure kuvastaa ohjelmistossa määritettyjä toimenpiteitä. Toimenpiteet ovat muita luokkia yksinkertaisempia, koska toimenpiteet eivät ole konkreettisia esineitä.
- PatientHealthPlanSupply kuvastaa ohjelmistossa määritettyjä tarvikkeita.
- PatientHealthPlanFood kuvastaa ohjelmistossa määritettyjä ruokia.

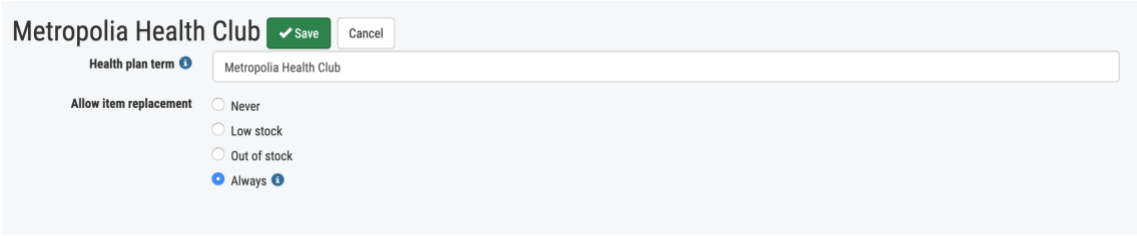
Prototyyppivetoisessa suunnittelussa keskityttiin laajasti käyttökokemukseen ja sen vuoksi teknisessä suunnittelussa tuotteiden lisäämiseen päädyttiin käyttämään Djangon Formset-toteutusta. Djangon Formset on abstraktiokerros (Abstraction layer) joka mahdollistaa usean lomakkeen luonnin ja tallennuksen tavalla, jolla lomakkeita voidaan käsitellä yhtenä suurempana kokonaisuutena. Kuvassa 5 on tuotteiden käyttöön tarkoitetun InlineFormsetin UI-toteutus, jonka jokainen rivi on yksittäinen lomake. Tuotteita käytetään syöttämällä haluttu määrä sekä muut valinnaiset tiedot ja rastittamalla rivin valintaruutu. Lomakkeet lähetetään palvelimelle, jossa jokainen validoidaan yksi kerrallaan.



Kuva 5. Kuvankaappaus tuotteiden käyttömodaalista.

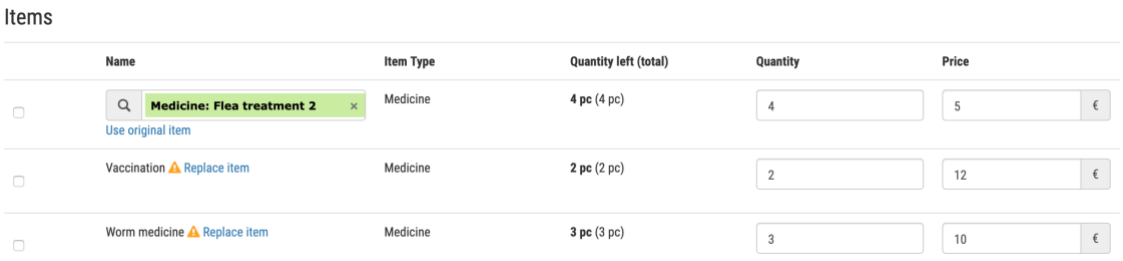
Terveydenhuoltopaketin yleisissä asetuksissa voidaan määrittää mahdolliset tilanteet, joissa tuotteet voidaan vaihtaa konsultaatiolla ennen niiden käyttöä. Tällöin voidaan välttyä tilanteelta, jossa ennalta määritetyt terveydenhuoltopaketin tuotteet ovat loppuneet tai loppumassa varastosta, tai tuote on vaihdettu kokonaan toiseen

tuotteeseen (ks. kuva 6). Korvaavat tuotteet ovat aina samaa tuoteluokkaa kuin korvattava tuote.



Kuva 6. Terveydenhuoltopaketin asetukset.

Kyseinen mahdollinen virhetilanne tuli esiin esitellessä toiminnollisuutta asiakastapahtumassa. Tapahtumasta saadun palautteen perusteella implementoitiin mahdollisuus tuotteiden vaihtoon. Kuvassa 7 ”Flea treatment” -tuote ollaan vaihtamassa ”Flea treatment 2” -tuotteeseen.



Kuva 7. Tuotteen vaihtaminen käyttötilanteessa.

Jokainen PatientHealthPlanItemin alaluokka toteuttaa item_to_consultation-metodin (ks. kuva 8), jossa valitun tuotteen tiedot välitetään tuotetyyppiä vastaavalle ConsultationItemin alaluokalle. Mikäli lomakedataan on määritetty replacement_item, eli tuote on vaihdettu, konsultaatiolla lisättävä tuote (item) vaihdetaan korvaavaksi tuotteeksi.

Jos PatientHealthPlanItemin tuotesaldoa (quantity_left) ei käytetä kokonaisuudessaan, kyseinen rivi kopioidaan copy_item-metodilla. Kopioituun tuotteeseen tulee tallentaa tieto alkuperäisestä tuoterivistä, jotta rivin käytetty osuus on mahdollista palauttaa alkuperäiseen riviin.

```
    def item_to_consultation(self, data, consultation, user):
        from pcloud.provet.health.utils import (calculate_instance_price,
                                                save_item_and_update_invoice)
        item = self.item
        replacement = data.get("replacement_item")
        if replacement:
            item = replacement

        split_item = None
        self.quantity_left -= data["quantity"]
        if self.quantity_left > 0:
            split_item = self.copy_item(replacement)

        kwargs = {
            'item': item,
            'name': item.name,
            'code': item.code,
            'quantity': data["quantity"],
            'price': get_formatted_price(self, data["price"]),
            'dosage_units': item.dosage_units,
            'unit': item.unit,
            'package_description': item.package_description,
            'vat_percentage': item.vat_group.percentage,
            'type_code': organization_codes.ITEM_TYPE_SUPPLY,
            'usage_type': organization_codes.USAGE_TYPE_ADMINISTERED,
            'usage_size': self.usage_size,
            'patient': self.health_plan.patient,
            'consultation': consultation,
            'created_user': user,
            'supervising_veterinarian': consultation.supervising_veterinarian,
            'performed_by': data.get("performed_by")}

        instance = ConsultationSupply(**kwargs)
        calculate_instance_price(instance)
        save_item_and_update_invoice(instance, user=user)
        self.save_health_plan_items(instance=instance, data=data, user=user,
                                    consultation=consultation, split_item=split_item,
                                    replacement=replacement)
```

Kuva 8. PatientHealthPlanSupplyn lisääminen konsultaatiolle.

Item_to_consultation-metodissa luotu konsultaatiotuote (ConsultationItem) tallennetaan ohjelmistossa yleisessä käytössä olevalla save_item_and_update_invoice-funktiolla, jossa tuotteen tallennuksen lisäksi tuotteesta luodaan laskutusrivi konsultaatiota vastaavalle laskulle.

PatientHealthPlanItem tallennetaan save_health_plan_items-funktiolla (ks. kuva 9), jolle välitetään käyttäjä, lomakedata, konsultaatiolle lisätty tuote (instance) sekä item_to_consutlation-funktiossa luodut split_item- ja replacement-muuttujat. Mikäli split_item on olemassa, eli tuote on ositettu, lasketaan jäljelle jäävä määrä alkuperäiselle tuotteelle sekä päivitetään ositetun tuotteen määrä ja asetetaan se

käytetyksi. Jos alkuperäinentuote on korvattu, riville tallennetaan tieto alkuperäisestä tuotteesta ja tuotteeksi annetaan vaihtotuote.

```
def save_health_plan_items(self, instance, data, user, consultation, split_item,
                           replacement):
    if split_item:
        self.quantity -= data['quantity']
        self.modified_user = user
        self.save()
        if replacement:
            split_item.replacement_for = self.item
            split_item.item = replacement
        split_item.quantity = data['quantity']
        split_item.original_quantity = data['quantity']
        split_item.quantity_left = 0
        split_item.consultation_item = instance
        split_item.used = True
        split_item.used_time = timezone.now()
        self.add_compensation_item_to_consultation(
            consultation=consultation,
            user=user,
            health_plan_item=split_item)
        split_item.save()
    else:
        if replacement:
            self.replacement_for = self.item
            self.item = replacement
        self.modified_user = user
        self.consultation_item = instance
        self.used = True
        self.used_time = timezone.now()
        self.add_compensation_item_to_consultation(
            consultation=consultation,
            user=user)
        self.save()
```

Kuva 9. PatientHealthPlanItemin tallennus

Käytetyistä tuotteista luodaan hyvitysrivit add_compensation_item_to_consultation-metodilla. Terveydenhuoltopakettisuunnitelmaa luodessa sille on määritetty Item-luokan objekti compensation_item eli hyvitystuote. Hyvitystuote on aina alaluokkaa toimenpide (Procedure), sillä hyvitystuote ei ole konkreettinen esine, joten siitä ei tarvitse pitää varastokirjanpitoa.

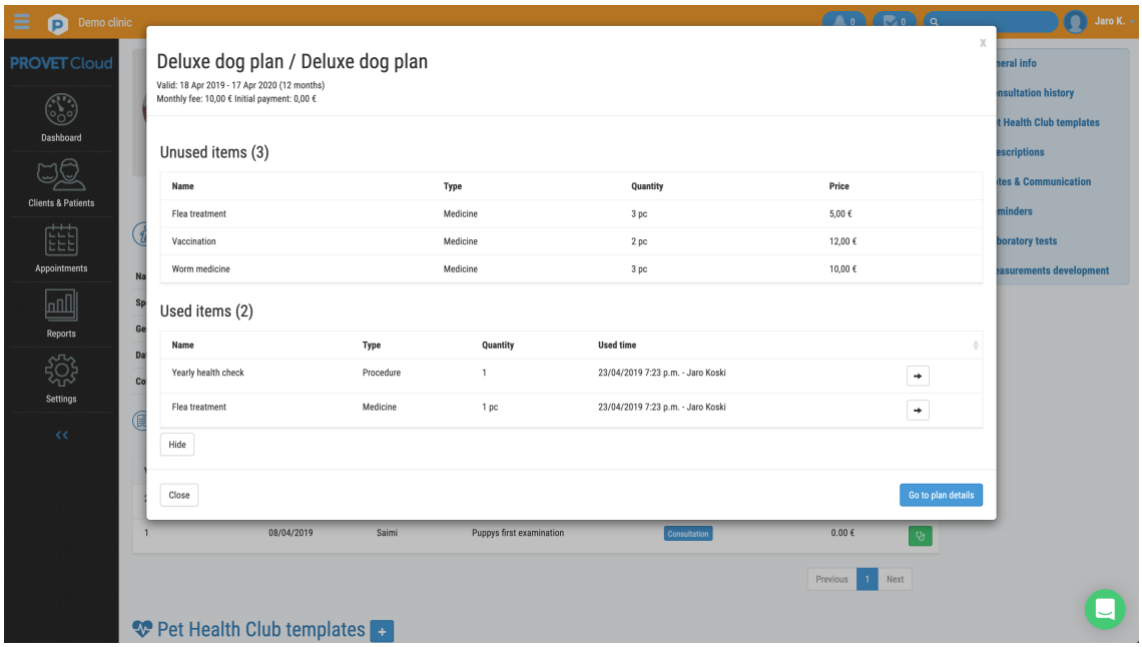
Hyvitystuotetta luodessa sille haetaan hinta terveydenhuoltopaketista konsultaatiolle lisätyn tuotteen laskuriviltä. Hinta muutetaan ennen tuotteen tallennusta negatiiviseksi, jotta se kumoaa paketista lisätyn tuotteen hinnan laskulla. Kuten terveydenhuoltopaketista konsultaatiolle lisätty tuote, myös hyvitystuote tallennetaan save_item_and_update_invoice-funktiolla (ks. kuva 10).

```
def add_compensation_item_to_consultation(self, consultation=None, user=None):
    from pcloud.provet.health.utils import save_item_and_update_invoice
    compensation_item = self.health_plan.compensation_item

    c_item = self.consultation_item

    invoice_row = c_item.get_invoice_row()

    kwargs = {
        'item': compensation_item,
        'name': compensation_item.name,
        'price': invoice_row.price * c_item.quantity * -1,
        'price_with_vat': invoice_row.price_with_vat * -1,
        'vat_percentage': self.item.vat_group.percentage,
        'consultation': consultation,
        'type_code': compensation_item.type_code,
        'no_department_rates': True,
        'no_commissions': True,
        'quantity': 1,
        'performed_by': user,
        'supervising_veterinarian': consultation.supervising_veterinarian,
        'patient': self.health_plan.patient}

    consultation_item = ConsultationProcedure(**kwargs)

    if consultation_item:
        save_item_and_update_invoice(consultation_item)
        self.compensation_item = consultation_item
```

Kuva 10. Hyvitystuotteen lisääminen konsultaatiolle.

5.5 Tuotteen palauttaminen terveydenhuoltopakettiin

Käytetyt tuotteet listataan terveydenhuoltopaketin hallintamodaaliin (ks. kuva 11), josta käyttäjille esitetään tieto, milloin tuotteet on käytetty sekä kuka on kirjannut tuotteet käytetyksi. Lisäksi jokaiselle käytetylle riville lisätään linkki konsultaatioon, jossa tuote on käytetty. Käytetyt tuotteet ovat mahdollista palauttaa terveydenhuoltopakettiin konsultaation laskun ollessa avoimessa tilassa.



Kuva 11. Terveydenhuoltopaketin tuotehallintamodaali.

Terveydenhuoltopaketista lisätyllä tuotteella, sekä siitä luodulla hyvitystuotteella, on kummallakin viiteavaimeen perustuva takaperoinen suhde (backwards relation) PatientHealthPlanItem-objektiin. Jos terveydenhuoltopaketit ovat järjestelmässä aktiivisia ja konsultaatiolta poistetaan tuote, tarkistetaan, onko kyseisellä konsultaatiotuotteella suhdetta terveydenhuoltopaketin tuotteeseen (PatientHealthPlanItem). Mikäli kyseinen suhde on olemassa, palautetaan tuote takaisin pakettiin PatientHealthPlanItemin restore_health_plan_item-metodilla (kuva 12).

```
if has_block_visible(
        'health_plan', active_context.provet_id, consultation.department_id):
    compensated_item = getattr(item, "compensated_item", None)
    if compensated_item:
        health_plan_item = item.compensated_item
        item_type_code = health_utils.item_type_code_to_string(
            health_plan_item.consultation_item.type_code)

        health_plan_item.restore_health_plan_item(
            delete_consultation_item=True,
            restore_qty=health_plan_item.consultation_item.quantity)

        if item_type_code not in refresh_tables:
            refresh_tables.append(item_type_code)

    health_plan_item = getattr(item, "health_plan_item", None)
    if health_plan_item:
        health_plan_item = item.health_plan_item
        item_type_code = health_utils.item_type_code_to_string(
            health_plan_item.compensation_item.type_code)

        health_plan_item.restore_health_plan_item(
            delete_compensation_item=True, restore_qty=item.quantity)

        if item_type_code not in refresh_tables:
            refresh_tables.append(item_type_code)
```

Kuva 12. PatientHealthPlanItem-suhteen tarkistaminen tuotteen konsultaatiolta poiston yhteydessä.

Kun terveydenhuoltopaketista lisättyjä tuotteita poistetaan konsultaatiolta, poistetaan myös siitä luotu hyvitystuote. Samoin, jos hyvitystuote poistetaan, poistetaan myös konsultaatiolle lisätty tuote. Tämä tehdään, jotta konsultaatiolle ei jää alennuksia, joihin asiakas ei ole oikeutettu. Tämän vuoksi restore_health_plan_item-metodille välitetään totuusarvomuuttujina tieto, poistetaanko konsultaatiotuote tai hyvitystuote.

Tuotetta palauttaessa restore_health_plan_item-metodilla sen tila muutetaan käyttämättömäksi ja käyttöaika nollataan. Jos palautettava tuote on ositettu toisesta terveydenhuoltopaketin tuotteesta, eli sillä on käyttämätön parent_item, palautettavan tuotteen käyttömäärät palautetaan alkuperäiseen terveydenhuoltopaketin tuoteriviin, jonka jälkeen ositettu tuote poistetaan. Mikäli tuotetta ei ole ositettu, mutta se on korvattu alkuperäisestä tuotteesta, terveydenhuoltopaketin tuotteelle palautetaan sen alkuperäinen tuote (Item), joka on tallennettu objektin replacement_for-kenttään (ks. kuva 13).

```
def restore_health_plan_item(self, delete_consultation_item=False,
                             delete_compensation_item=False, restore_qty=None):
    self.used = False
    self.used_time = None
    if delete_consultation_item and self.consultation_item:
        consultation_item = self.consultation_item
        self.consultation_item = None
        consultation_item.delete()
    if delete_compensation_item and self.compensation_item:
        compensation_item = self.compensation_item
        self.compensation_item = None
        compensation_item.delete()
    if self.parent_item and not self.parent_item.used:
        parent_item = self.parent_item
        parent_item.quantity_left += restore_qty
        parent_item.quantity += restore_qty
        parent_item.save()
        self.delete()
    else:
        self.quantity_left += restore_qty
        if self.parent_item:
            self.parent_item = None
        if self.replacement_for:
            self.item = self.replacement_for
            self.replacement_for = None
        self.save()
```

Kuva 13. PatientHealthPlanItemin palautusmetodi.

6 Ajastetut toiminnot

Provet Cloudin toimintojen tausta-ajot on toteutettu Celery-kirjastolla. Celery on asynkroninen tehtäväjono, joka jakaa suoritettavia tehtäviä useiden säikeiden tai koneistojen, kuten palvelimien välillä [6]. Celeryn ja tehtävänsuorittajien väliseen kommunikoitiin käytetään usein viestinvälittäjää, joka säilyttää tiedon Celeryn luomasta tehtäväjonosta. Provet Cloud käyttää viestinvälitykseen Redis-palvelinta.

Celerybeat on Celeryn ajastuksesta huolehtiva kirjasto, jonka avulla delegoidaan, mitä tausta-ajoja ajetaan ja milloin nämä tehtävät tulee suorittaa. [6.]

6.1 Laskutus

Oleellinen osa terveydenhuoltopaketin toimintaa eläinlääkäriklinikan kannalta on kuukausittainen laskutus, joka takaa klinikalle jatkuvan ansion, vaikkei asiakas käyttäisi palvelua. Jotta eläinklinikan laskutuskirjanpito pitää paikkansa,

terveydenhuoltopaketeista tulee generoida laskut kuukausittain sen sijaan, että ne luotaisiin kerralla tilausta luotaessa. Generoidut laskut tallennetaan Provet Cloud -järjestelmään ja asiakkaalle tulee lähettää lasku sähköpostiin pdf-muodossa.

6.1.1 Maksun luominen järjestelmään

Laskutuksen hallintaa varten luodaan PatientHealthPlanPayment-malli, jolla säilytetään tietoa kuukausittaisista maksuista, joista laskut luodaan. Malliin tallennetaan, mihin tilaukseen maksut kuuluvat, milloin lasku tulee luoda, viite objektista luotuun laskuun sekä maksun tila (ks. esimerkkikoodi 4).

```
class PatientHealthPlanPayment(models.Model):
    subscription = models.ForeignKey(PatientHealthPlan)
    invoice = models.ForeignKey(Invoice,
                                null=True,
                                blank=True)
    create_date = models.DateField()
    payment_status = models.IntegerField(choices=codes.PAYMENT_STATUS,
                                         default=codes.UPCOMING)
    initial_payment = models.BooleanField(default=False)
```

Esimerkkikoodi 4. PatientHealthPlanPayment-malli.

Kun asiakkaalle luodaan tilausta, luodaan samalla sarja PatientHealthPlanPayment-objekteja. Kuun viimeinen päivä haetaan Python calendar -kirjaston monthrange-funktiolla, joka palauttaa kuun ensimmäisen viikonpäivän sekä kuun viimeisen päivän numeerisina arvoina. Create_health_plan_payments-funktiolle on määritetty oletusarvoksi datetime-objektista haettavat numeroarvot tämänhetkiselle kuukaudelle sekä vuodelle. Näiden kolmen arvon pohjalta luodaan datetime-objekteja kuvaamaan laskujen luontipäiviä.

PatientHealthPlanPayment-objekteja luodaan ”for loop” -silmukassa niin monelle kuukaudelle, kuin kyseisessä tilauksessa on määritetty. Mikäli tilaukseen on määritetty initial_payment, eli tilauksen sopimuksen laatimisen yhteydessä on maksettu ensimmäinen erä, luodaan PatientHealthPlanPayment-objekti arvoilla, joilla sitä ei tulla laskuttamaan uudestaan (ks. kuva 14).

```
    def create_health_plan_payments(self, subscription=None, month=datetime.now().month,
                                    year=datetime.now().year):
        if not subscription:
            subscription = self
        for m in range(subscription.months):
            last_day = monthrange(year, month)[1]
            invoice_date = datetime(year, month, last_day)
            creation_kwargs = {
                "subscription": subscription,
                "create_date": invoice_date}
            if self.initial_payment and m == 0:
                creation_kwargs.update({
                    "initial_payment": True,
                    "payment_status": codes.CREATED,
                    "invoice_status": codes.INVOICE_CREATED})
            PatientHealthPlanPayment.objects.create(**creation_kwargs)
            if month < 12:
                month += 1
            else:
                month = 1
                year += 1
```

Kuva 14. PatientHealthPlanPayment-objektien luonti.

6.1.2 Celery-tehtäväjono

Ajoitettujen laskujen luonti toteutettiin kirjoittamalla sarja Djangon management-komentoja. Management-komennot ovat nimensä mukaisesti ylläpidollisia python-komentosarjoja, joita voidaan ajaa terminaalin kautta tai kutsumalla niitä koodissa. Manuaalisen työn välttämiseksi laskujen luonti hoidetaan Celerybeat-kirjastoa käyttämällä, sillä Celerybeat mahdollistaa Celery-tehtävien ajoittamisen.

Celeryn aikataulu luodaan Djangon settings.py-tiedostoon Celeryn konfiguraation yhteydessä. Celeryn crontab-objektilla määritetään, milloin kyseinen tehtävä ajetaan. Objektille voidaan antaa konkreettinen ajoaika tai ajoajat intervalleina. Esimerkkikoodissa 5 määritetty merkkijono ’4’ tarkoittaa, että Celery-tehtävä ajetaan päivittäin kello 04:00 aamuyöstä.

```
# Celery configuration
BROKER_URL = 'redis://localhost:6739'
CELERY_RESULT_BACKEND = 'redis://localhost:6739'
CELERY_SEND_TASK_ERROR_EMAILS = False
CELERY_IGNORE_RESULT = True
CELERY_DEFAULT_QUEUE = 'default'
CELERY_ACCEPT_CONTENT = ['json',]
CELERY_TASK_SERIALIZER = 'json'
CELERY_ENABLE_UTC = True

CELERYBEAT_SCHEDULE = {
    'scan_health_plan_payments': {
```

```
        'task': 'pcloud.provet.health_plan.tasks.scan_health_plan_payments',
        'schedule': crontab(hour='4', minute=0)
    },
}
```

Esimerkkikoodi 5. Celery-tehtävien aikatauluttaminen.

Celery-tehtäviä varten luodaan tasks.py-tiedosto health_plan-hakemistoon, joka sisältää funktiot management-komentojen ajamiseen. Celery-kirjaston shared_task decorator -objekti mahdollistaa funktioiden käyttämisen Celery-tehtävänä, toisin sanoen mahdollistaa funktioiden lisäämisen Celeryn tehtäväjonoon (ks. kuva 15).

```python
from celery.app import shared_task
from django.core.management import call_command

MINUTES_10 = 60 * 10


@shared_task(queue='slow', time_limit=MINUTES_10, soft_time_limit=MINUTES_10)
def scan_health_plan_payments():
    call_command('healthplanscanprovets')  # pragma: no cover


@shared_task(queue='slow', time_limit=MINUTES_10, soft_time_limit=MINUTES_10,
             rate_limit='1/s')
def task_payment_invoice_scan(provet_id):
    call_command('healthplanpaymentscan', provet_id=provet_id)   # pragma: no cover


@shared_task(queue='slow', rate_limit='1/s')
def task_create_invoices(provet_id, payment_id):
    call_command('healthplancreateinvoices',
                 provet_id=provet_id, payment_id=payment_id)  # pragma: no cover
```

Kuva 15. Tasks.py-tiedosto

Scan_health_plan_payments-tehtävä käy läpi koko tietokannan etsien Provet-applikaatioita, joilla on käytössä HealthPlan-toiminnallisuus. Mikäli Provet-applikaatiossa terveydenhoitopaketit on kytketty päälle, siitä luodaan task_payment_invoice_scan-tehtävä, joka asetetaan tehtäväjonoon Redis-palvelimelle.

Task_payment_invoice_scan tarkistaa sille options-parametrina annetusta Provet-applikaatiosta, löytyykö kyseisestä järjestelmästä PatientHealthPlanPayment-instansseja, joiden laskun luontipäivä on sama kuin tehtävän ajon päivä. Mikäli käsittelemättömiä laskuja löytyy, niistä luodaan task_create_invoices-tehtävä Redis-palvelimen jonoon (ks. kuva 16).

```
class Command(BaseCommand):

    option_list = BaseCommand.option_list + (
        make_option('--provet_id', help='Provet application id to scan.'),
    )

    def handle(self, *args, **options):
        self.provet_id = options.get('provet_id', None)
        provet_id = int(self.provet_id)
        with active_provet(provet_id):
            today = datetime.today()
            unhandled_payments = PatientHealthPlanPayment.objects.filter(
                create_date=today,
                payment_status=UPCOMING,
                initial_payment=False)
            for payment in unhandled_payments:
                task_create_invoices.delay(provet_id=provet_id, payment_id=payment.id)
```

Kuva 16. Task_create_invoices-tehtävän luonti käsittelemättömistä laskuista.

Celery-tehtäväketjun viimeinen pala on task_create_invoices-tehtävä. Task_create_invoices-funktiolle välitetään tieto Provet-applikaatiosta sekä käsiteltävästä PatientHealthPlanPayment-objektista. Tehtävän tarkoitus on luoda maksuinstanssin perusteella lasku, joka luonnin jälkeen lähetetään asiakkaan sähköpostiin.

Provet-järjestelmän asetuksiin voidaan tallentaan toimipaikkakohtaiset laskutustiedot, kuten IBAN-numero ja BIC-koodi. Laskutustiedot välitetään laskuobjektille sitä luodessa hakemalla niistä luotu avainsanasanakirja get_department_kwargs-metodilla. Maksuobjektiin tallennetun viitteen kautta haetaan tieto tilauksesta. Tilauksesta saadut asiakastiedot välitetään laskulle ja se tallennetaan tietokantaan (ks. kuva 17).

Luotu lasku on nollasummainen, joten sille tulee luoda laskutusrivi. Laskutusrivi luodaan tilauksesta hakemalla sille luomisen yhteydessä määritetty Item-luokan objekti ”invoicing_item”-laskutustuote. Kuten hyvitystuote, laskutustuote on järjestelmässä aina toimenpideluokkaa, koska kyseessä ei ole konkreettinen tuote, jolla on varastosaldo. Vaikka Item-tyyppiselle laskutustuotteeksi valitulle toimenpiteelle on määritetty hinta, tuotteen laskutusriviä luodessa laskutustuotteelle annetaan tilaukseen tallennettu kuukausihinta (ks. kuva 17).

```
    def handle(self, *args, **options):
        self.provet_id = options.get('provet_id', None)
        self.payment_id = options.get('payment_id', None)

        provet_id = int(self.provet_id)
        with active_provet(provet_id):
            user = User.objects.filter(is_superuser=True).first()
            payment = PatientHealthPlanPayment.objects.get(pk=int(self.payment_id))
            subscription = payment.subscription
            department = subscription.department
            client = subscription.client

            department_kwargs = self.get_department_kwargs(department, user)
            invoice = Invoice(**department_kwargs)

            self.client_to_invoice(invoice, client)
            invoice.save()

            payment.invoice = invoice
            self.create_payment_invoicerow(payment, invoice)
            payment.payment_status = INVOICE_CREATED

            request = HttpRequest()
            request.provet_id = provet_id

            invoice_kwargs = {"agreed_to_pay_later": True}
            finalize_invoice(invoice, request, user, **invoice_kwargs)
            payment.save()
            payment.send_invoice()
```

Kuva 17. Laskun luominen terveydenhuoltopakettitilauksesta.

Kun tilauksesta on luotu lasku, se tulee lähettää asiakkaan ennalta määritettyyn sähköpostiosoitteeseen. Asiakkaalle lähetettävän sähköpostiviestin sisältö luodaan tilauksen tiedoista html-tiedostoon ja juuri luotu lasku generoidaan pdf-liitteeksi sähköpostiin. Provet Cloud käyttää sähköpostien lähettämiseen Amazonin Simple Email Service -palvelua. Koska Provet Cloudista lähetetään päivittäin suuri määrä sähköposteja, myös sähköpostien generointi ja lähetys on toteutettu Celery-tehtävänä.

6.2 Tilauksen päättyminen ja uusiminen

Koska terveydenhuoltopaketin lopullinen tavoite on palvelun jatkuvuus, tilauksen uusiutuminen on oleellinen osa järjestelmää. Meneillään oleva tilaus voidaan uusia sellaisenaan uudelle tilauskaudelle sen loppuessa, tai tilauksen paketti voidaan vaihtaa toiseksi määrittämällä se tilausta luodessa.

Esimerkiksi jos asiakkaalle on myyty eläimen ensimmäiseksi elinvuodeksi paketti, joka sisältää kyseiselle ajalle tarvittavat rokotteet, joita ei tarvita ensimmäisen vuoden

jälkeen, on hyvä seuraavaksi tilaukseksi määrittää paketti, joka sopii tulevaan elämäntilanteeseen paremmin.

Tilauksien päättyminen sekä uusiutuminen tarkastetaan aamuyöllä ajettavana Celerybeat-tausta-ajona. Tilausten statuksen tarkastusfunktio, task_renew_plans, lisätään tasks.py-tiedostoon ja sitä kutsutaan scan_health_plan_payments-tehtävässä. Tehtävässä tarkistetaan, onko tilaus umpeutunut edellisenä päivänä. Mikäli tilaus päättyy, tilaus lopetetaan kutsumalla PatientHealthPlan-mallin end_plan-metodia. Jos tilaus on määritetty uudistuvaksi, se uusitaan mallin renew_plan-metodilla (ks. kuva 18).

```
class Command(BaseCommand):

    option_list = BaseCommand.option_list + (
        make_option('--provet_id', help='ID of the Provet'),
    )

    def handle(self, *args, **options):
        self.provet_id = options.get('provet_id', None)
        provet_id = int(self.provet_id)
        with active_provet(provet_id):
            yesterday = datetime.today() - timedelta(1)
            plans = PatientHealthPlan.objects.filter(end_date=yesterday)
            for plan in plans:
                if plan.renew:
                    plan.renew_plan()
                else:
                    plan.end_plan()
```

Kuva 18. Tilausten uusimisen ja lopetuksen suorittava management-komento.

Vaikka tilausta ei varsinaisesti päätettäisi, sen tuotteita ei voisi käyttää konsultaatiolla, sillä aktiiviset tilaukset suodatetaan konsultaatiolla tarkastamalla, onko tilauksen end_date jo mennyt. Tärkein toiminnallisuus tilauksen päättämisellä on alennuksiin oikeuttavien tag-merkkausten poisto. Koska lemmikkieläimelle on mahdollista lisätä samoja tag-merkintöjä kuin tilaukselle, tulee tarkistaa, onko eläimeen liitetty Tag-objekti lisätty sille tilausta luodessa.

Kuvassa 19 esitetään alennusmerkinnän (Tag) poistaminen eläimeltä. Tilaukseen liittyvät alennusmerkinnät haetaan ID:t sisältäväksi listaksi. Eläimelle määritetyt alennusmerkinnät käydään läpi silmukassa. Koska on tärkeää tarkistaa, onko alennusmerkintä luotu tilauksen pohjalta, sille luodaan totuusmuuttuja from_plan. Totuusmuuttujan arvo saadaan tarkistamalla, onko eläimelle luotu merkintä luotu viiden sekunnin sisällä tilauksen luomisesta. Viiden sekunnin intervallilla varmistetaan, ettei

merkintä jää poistamatta, vaikka merkintöjä luodessa järjestelmässä olisi ollut ongelmia. Mikäli merkintä on luotu tilauksen pohjalta, se poistetaan eläimeltä. Lopulta tilauksen status muutetaan loppuneeksi ja muuttuneet arvot tallennetaan.

```
def end_plan(self):
    plan_tags = self.get_tags_related().values_list("tag_text_id", flat=True)
    plan_created = self.created
    for tag in self.patient.get_tags_related():
        from_plan = plan_created <= tag.created < tag.created + timedelta(seconds=5)
        if tag.tag_text_id in plan_tags and from_plan:
            tag.delete()
    self.status = codes.SUBSCRIPTION_ENDED
    self.save()
```

Kuva 19. Alennukseen oikeuttavien Tag-objektin poisto eläimeltä.

Tilausta uusittaessa tarkistetaan, uusiutuuko tilaus sellaisenaan vai vaihdetaanko paketti next_plan-kenttään tallennettuun terveydenhuoltopakettiin. Terveydenhuoltopaketilla ja tilauksella on useita yhteisiä kenttiä, joten kopioitavat kentät on määritelty FIELDS_TO_COPY-listaan (ks. esimerkkikoodi 6).

```
FIELDS_TO_COPY = ["internal_name", "display_name", "internal_notes",
                  "months", "monthly_fee", "department", "invoicing item",
                  "compensation_item", "description"]
```

Esimerkkikoodi 6. Tilaukseen kopioitavan HealthPlan-mallin kentät

Kuvassa 20 esitetään tilauksen uusiminen. FIELDS_TO_COPY-lista käydään läpi ”for loop” -silmukassa, jossa kentistä luodaan avainsana-argumentit. Meneillään olevasta tilauksesta tallennetaan asiakas, eläin, sekä maksutapa että uusiutumistiedot. Tilaus-objektin luomisen jälkeen sille lisätään pakettiin kuuluvat tuotteet ja alennukset sekä siitä luodaan PatientHealthPlanPayment-objektit, jonka jälkeen tilaus on käytettävissä asiakkaalle.

```
    def renew_plan(self):
        if not self.renew:
            return
        plan = self.next_plan if self.next_plan else self.health_plan

        user = User.objects.filter(is_superuser=True).first()

        plan_kwargs = {'created_user': user}

        for plan_kwarg in HealthPlan.FIELDS_TO_COPY:
            plan_kwargs[plan_kwarg] = getattr(plan, plan_kwarg)

        today = datetime.now()
        start_date = datetime.now() + timedelta(days=1)
        end_date = today + timedelta(plan.months * 365 / 12)

        plan_kwargs.update({
            "health_plan": plan,
            "start_date": start_date,
            "end_date": end_date,
            "payment_method": self.payment_method,
            "invoicing_partner": self.invoicing_partner,
            "renew": self.renew,
            "client": self.client,
            "patient": self.patient})

        new_subscription = PatientHealthPlan.objects.create(**plan_kwargs)
        final_month = self.end_date.month + 1
        if final_month > 12:
            final_month = 1
        year = self.end_date.year
        self.create_health_plan_payments(subscription=new_subscription,
                                         month=final_month, year=year)
        self.copy_plan_items(subscription=new_subscription)
        self.transfer_tags(subscription=new_subscription)
```

Kuva 20. Tilauksen uusimisen metodi

7 Yhteenveto

Insinöörityön tavoitteena oli luoda toiminnollisuus, jolla eläinklinikat kykenevät luomaan, hallitsemaan, myymään ja laskuttamaan lemmikkieläimen terveydenhuoltopaketteja. Työssä kehitetty toiminnollisuus luotiin suunnittelupalavereissa toteutetun prototyypin pohjalta. Lisäksi työssä toteutettiin järjestelmä, jolla luodaan automaattisesti kuukausittaisia laskuja ja lähetetään ne asiakkaan sähköpostiin.

Raportin kirjoittamisen aikana tavoitteena kuvailtu paketin perustoiminnollisuus on kehitetty lukuun ottamatta automaattisia asiakkaalle lähetettäviä muistutuksia. Kehitystä tullaan jatkamaan välittömästi, sillä yrityksen kannalta insinöörityöni osuus oli niin sanotusti ”Phase one”, eli vaihe 1 suuremmasta kokonaisuudesta.

Seuraava vaiheessa toiminnollisuus otetaan pilottikäyttöön. Pilottikäytöstä kerätään palautetta, jonka pohjalta toiminnollisuutta kehitetään vastaamaan todellisen työtilanteen tarpeita. Pakettien käytöstä syntyvän datan pohjalta tullaan luomaan erinäisiä raportointityökaluja Provet Cloudin raporttiosioon. Lisäksi toiminnollisuuden eri osioista tullaan luomaan REST API -päätepisteet, jotta klinikat kykenevät integroimaan toiminnollisuutta esimerkiksi heidän asiakkaillensa tarjoamaan kännykkä- tai web-sovellukseen.

Yhtenä tärkeimpänä jatkokehityskohteena on luoda integraatio, yhteen tai useampaan pakettien laskuttamisen hoitavaan suoraveloitusjärjestelmään, kunhan mahdolliset yhteistyökumppanit on saatu valittua.

Toiminnollisuus on testattu kirjoittamalla sille yksikkötestit, mutta käyttäjätestausta ei ole vielä suoritettu.

Henkilökohtaisesti työ opetti suuresti isojen toiminnollisuuksien suunnittelusta ja määrittelemisestä. Lisäksi sain arvokkaan mahdollisuuden toimia yhteistyössä asiakkaiden kanssa, sillä pääsin esittelemään toiminnollisuutta useille asiakkaille kokouksissa sekä asiakastapahtumissa.

Terveydenhuoltopaketit pyritään ottamaan pilottikäyttöön yhdellä tai useammalla yhteistyöklinikalla alkukesästä 2019. Tavoitteena on julkaista toiminnollisuus yleiseen käyttöön kesän aikana.

Lähteet

1 Provet Cloud. Verkkoaineisto. <https://provet.info/>. Luettu 15.4.2019.

2 Pet insurance vs. pet wellness plans. Verkkoaineisto. Pet Insurance Review. <https://www.petinsurancereview.com/how-it-works/pet-insurance-pet-wellness>. Luettu 2.5.2019

3 Provet Net. Verkkoaineisto. <https://www.provet.fi/provet-net>. Luettu 20.4.2019.

4 A Nation Subscribed: 2018 state of the UK subscription economy. Verkkoaineisto. <https://www.zuora.com/resource/nation-subscribed-2018-state-uk-subscription-economy/>. Luettu 20.4.2019

5 Kriik, Greete. 2018. Prototyypit ohjelmistokehityksessä. Verkkoaineisto. <https://www.arter.fi/prototyypit-ohjelmistokehityksessa/> 20.2.2018. Luettu 15.3.2019

6 Celery. Verkkoaineisto <https://www.fullstackpython.com/celery.html/>. Luettu 05.05.2019

Terveydenhuoltopaketin luontinäkymä