Verkkokaupan alennuskuponkijärjestelmä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tietojenkäsittelyn koulutusohjelma

Kevät 2018

Sami Jussinniemi

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Visamäki

Tekijä Sami Jussinniemi **Vuosi** 2018

Työn nimi Verkkokaupan alennuskuponkijärjestelmä

Työn ohjaaja Tommi Saksa

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli tutkia, miten toteuttaa verkkokaupan alennuskuponkijärjestelmä. Lisäksi työssä pohdittiin alennuskuponkien ominaisuuksia ja niiden ominaisuuksien tuomia haasteita suunnittelussa ja toteutuksessa.

Tutkimustyö tehtiin toimeksiantajan asiakkaan Levykauppa Äxän verkkokaupan kehitystyötä varten. Levykauppa Äxän verkkokauppa on toteutettu Python- ja Django-ohjelmointikielillä, jotka olivat opinnäytetyön tekijälle uusia ohjelmointikieliä.

Teoriaosuudessa käsiteltiin yleisesti verkkokauppaa, alennuskuponkeja, työssä käytettyjä työkaluja ja ohjelmistoja ja pohdittiin alennuskuponkien ominaisuuksia. Käytännön osuudessa pystytettiin virtuaalinen kehitysympäristö ja rakennettiin alennuskuponkijärjestelmän prototyyppi tehtävälistan pohjalta.

Lopputuloksena on yksinkertainen prototyyppijärjestelmä, joka havainnollistaa alennuskuponkien toimintaa ja haasteita ostoprosessin eri vaiheissa. Opinnäytetyön jälkeen tavoitteena on toteuttaa alennuskuponkijärjestelmä Levykauppa Äxän verkkokauppaan käyttäen apuna opinnäytetyössä hankittuja tietoja.

Avainsanat Alennuskupongit, Django, Python, Verkkokauppa

Sivut 28 sivua, joista liitteitä 2 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki

Author Sami Jussinniemi **Year** 2018

Subject E-commerce Discount Coupon System

Supervisor Tommi Saksa

ABSTRACT

The purpose of this thesis was to examine how to implement a discount coupon system to an e-commerce site. Additionally, the design of the discount coupons and challenges posed by these features were considered in design and implementation.

The research was executed for commissioner's client Levykauppa Äx and its online store. Levykauppa Äx's online store was built with the Python and Django programming languages, which were new programming languages for the thesis author.

This thesis is divided in two sections: theory and practical. The theory section of the thesis discusses e-commerce, discount coupons work tools and software. Consideration was given to the features of discount coupons. In the practical part of the thesis, a virtual development environment was built, and a basic prototype of the discount coupon system was constructed.

The outcome of the research is a simple prototype system that demonstrates functionality of discount coupons and challenges through different phases of the purchase process. After thesis, the goal is to implement a discount coupon system at the Levykauppa Äx's online store using the information obtained in the thesis.

Keywords Discount Coupon, Python, Django, E-commerce

Pages 28 pages including appendices 2 pages

SISÄLLYS

1 JOHDANTO

Työn toimeksiantaja on Sovelluskontti Oy, jonka asiakas on levykauppa Äx. Levykauppa Äx on vuodesta 1997 lähtien toiminut levyihin, elokuviin, kirjoihin, paitoihin ja muihin oheistuotteisiin erikoistunut yritys. Äxällä on kivijalkakauppoja kymmenen kappaletta ympäri Suomen, muun muassa Helsingissä, Tampereella, Kuopiossa, Oulussa ja Turussa. Asiakkaalla on myös verkkokauppa.

Opinnäytetyön aiheena on suunnitella ja toteuttaa alennuskuponkijärjestelmän prototyyppi. Prototyypin tarkoitus on toimia oppimistyökaluna uusille ohjelmointikielille ja esikuvana opinnäytetyön jälkeen aloitettavalle alennuskuponkijärjestelmälle Levykauppa Äxän verkkokaupassa. Verkkokaupassa on valmiina kanta-asiakkaan leimat-ominaisuus ja nyt tarkoituksena on toteuttaa perinteisempi alennuskuponkiominaisuus, jolla ostaja saa alennuksia tai muita etuja. Verkkokauppa on aiemmin rakennettu käyttäen Python-ohjelmointikieltä ja tämän sovelluskehyksellä, Djangolla. Työtä varten pystytettiin virtuaalinen kehitysympäristö. Opinnäytetyön tuloksena syntyvät suunnitelmat ja ominaisuudet on tarkoitus ottaa käyttöön verkkokaupassa.

Valitsin aiheen, koska ammattikorkeakoulun opetussuunnitelmaan ei kuulu verkkokaupat ja tässä oli mahdollisuus tutustua uuteen ja mielenkiintoiseen aiheeseen sekä kehittää omaa osaamistani. Samoin Python ja Django olivat itselleni tuntemattomia ohjelmointikieliä. Aihe on laaja, mutta tarkoituksena on jatkaa työtä opinnäytetyön valmistuttua. Työssä esitellään toimeksiantajan tarpeita ja pohditaan ongelmia, joita nämä tarpeet tuovat esiin. Tämän pohjalta suunnitellaan toimintoja, jotka vastaavat toimeksiantajan näkemystä.

Opinnäytetyössä on tarkoituksena vastata seuraaviin tutkimuskysymyksiin: Miten suunnitella ja toteuttaa alennuskuponkijärjestelmä verkkokauppasivustoon Pythonia ja Djangoa käyttäen? Minkälaisia ominaisuuksia alennuskupongeilla on ja miten ne tulee ottaa huomioon suunnittelussa ja toteutuksessa?

2 VERKKOKAUPPA YLEISESTI

Verkkokauppa tarkoittaa internetissä tapahtuvaa kaupankäyntiä sivuston ja ihmisen välillä. Kaupankäynti on tyypillisesti kuluttajille suunnattua (B2C), yritysten välistä (B2B) tai kuluttajien välistä kaupantekoa (C2C). Verkkokaupalla pyritään automatisoimaan myyntiä ja pienentämään kauppiaalle kohdistuvia kustannuksia. Kun osa kauppiaan työstä siirtyy asiakkaalle, kauppiaalta vaaditaan ymmärrystä niin asiakaskuntansa ostokäyttäytymisestä kuin verkkopalveluista, jotta asiakkaat tekisivät kauppiaan kannalta myönteisiä ostopäätöksiä. (Hallavo 2013, 1.1.)

2.1 Verkkokaupan hyödyt

Verkkokauppoja käytetään yhä enemmän tuotetietojen hakemiseen, saatavuuteen ja vertailuun. Vuorokauden ympäri saatavilla olevat tiedot onkin tehnyt tästä toiminnasta helppoa ja järkevää asiakkaan kannalta. Asiakkaan näkökulmasta etuja ovat myös maksu- ja toimitustapojen monipuolisuus yhdistettynä ostohistorian saatavuuteen ja mahdollisuus palauttaa ostokset. Lisäksi verkkokauppojen valikoimat ovat tyypillisesti laajempia kuin kivijalkakauppojen. Edellä mainittujen ominaisuuksien tarjoaminen nostattaa asiakastyytyväisyyttä. (Hallavo 2013, 3.1.)

Kauppiaan näkökulmasta tavoitteena verkkokaupassa on myydä mahdollisimman paljon minimoiden kustannuksia. Verkkokaupan hyödyt ovat kuitenkin monimuotoiset. Erikoistuotteita myyvät verkkokaupat tavoittavat pienet asiakasryhmänsä laajemmin niin kotimaassa kuin ulkomailla kuin perinteiset kivijalkakaupat. Verkkokaupan avulla kyetään myymään asiakkaalle laajempaa valikoimaa kuin mitä kauppiaan todellinen fyysinen varastotila on. Verkkokauppa tukee myös kivijalkakauppojen myyntiä. Esittämällä ja ylläpitämällä tuotteiden saatavuustietoja myymälöissä vähennetään asiakasyhteydenottoja ja ohjataan kuluttajat suoraan myymälään. (Hallavo 2013, 3.1.)

Asiakashankinta verkkokauppaan ei välttämättä ole halvempaa kuin kivijalkakauppaan. Mutta markkinointikanavilla ja työkaluilla asiakkaiden tavoittaminen on halvempaa. Esimerkkeinä: alennuskupongit, mainosbannerit, sähköpostiviestit, hakusana- ja hakukonemarkkinointi. Tilaushistorioiden pohjalta kauppias voi luoda persoonallisempaa markkinointia massamainonnan sijaan. (Hallavo 2013, 3.1.)

2.2 Alennuskupongit verkkokaupassa

Verkkokaupan alennuskupongit tarjoavat yleisesti ilmaista toimitusta, prosenttialennusta tai jotain muuta etua tai alennusta asiakkaalle. Verkko-

kauppiaat käyttävät alennuskupongeista tyypillisesti termiä ”koodi” tai jotain sen synonyymiä. Koodi on merkkijono jonka asiakas syöttää tilauksen yhteydessä sivustolle. (Pathath 2015.)

Koodit voidaan jakaa karkeasti kahteen eri luokaan: yleisiin ja uniikkeihin. Yleiset kuponkikoodit ovat helposti muistatettavia sanoja eivätkä välttämättä ole käyttäjäkohtaisia. Tällä tavoin koodeja voidaan jakaa kuluttajille potentiaalisesti luoden lisää liikennettä verkkokauppaan. Uniikit koodit ovat merkkijonoltaan yleensä satunnaisia. Uniikit koodit ovat hyödyllisiä silloin, kun halutaan luoda kertakäyttöisiä koodeja ja rajata koodien määrää. (Thomas 2014.)

Kauppiaalle alennuskupongit ovat oikein käytettynä tehokas työkalu verkkokauppiaalle. Hyviä käyttöesimerkkejä alennuskupongeille on esimerkiksi liittää kuponki tuotelahjaan varaston tyhjentämiseksi ja uusien tuotteiden tai palveluiden markkinoinnissa. Äskeisessä kappaleessa mainittiin liikenteen kasvun potentiaalisuus, mutta kupongeilla voidaan houkutella asiakas tekemään uusi tilaus myöhempänä ajankohtana. (Wong 2016.)

2.3 **Alennuskupongit verkossa**

Alennuskupongit ovat kuluttajille haluttuja hyödykkeitä, ja tämän ovat huomanneet muutkin toimijat kuin verkkokauppiaat. Käyttämällä hakukonepalveluita kuluttajat löytävät hetkessä useita verkkosivustoja, jotka tarjoavat alennuskuponkeja. Lisäksi sosiaalinen media on täynnä alennus- ja kuponkitarjouksia. Tämä ei ole epätavallista nykypäivän verkkomarkkinoinnissa, mutta valitettavasti kuponkien kysyntä houkuttelee paikalle epärehellisiä henkilötietojenkeräilijöitä sekä huijareita. (Lehmann 2017; Mysecurityawareness n.d.)

Tyypillisesti huijaussivustot pyytävät käyttäjän sähköpostiosoitetta tai puhelinnumeroa luvaten että alennuskuponki lähetetään käyttäjälle. Tiedot annettuaan käyttäjä ohjataan toiselle sivustolle, jossa tarjotaan lisää alennuksia ja etuja. Huijarit myyvät yhteystietoja ja käyttäjien puhelimet sekä sähköposti alkavat vastaanottamaan roskapostia. Huijaussivustot saattavat asentaa käyttäjän käyttöjärjestelmään haittaohjelmia, joilla huijarit voivat kerätä esimerkiksi salasanatietoja. (Lehmann 2017; Mysecurityawareness n.d.)

3 TYÖKALUT JA OHJELMISTOT

Tässä kappaleessa käsitellään keskeisimpiä Levykauppa Äxän verkkokaupassa ja ohjelmointityössä tarvittavan kehitysympäristön pystytyksessä käytettyjä työkaluja ja ohjelmistoja.

3.1 Python

Python on tulkattava, alustariippumaton olio-ohjelmointikieli (Python Software Foundation 2018). Tulkattavassa ohjelmointikielessä lähdekoodi ajetaan konekielelle rivi kerrallaan ohjelman etenemisen mukaan, siinä missä käännettävän ohjelmointikielen lähdekoodille tehdään kokonainen käännös konekielelle ennen ajoa. (Kasurinen 2009, 6.)

Python on yleiskäyttöinen ohjelmointikieli. Se sisältää useita kirjastoja, joita voidaan soveltaa erilaisten ongelmien ratkaisemiseksi. Pythonia pystyy laajentamaan sekä sisällyttämään myös C ja C++ -ohjelmointikieliin. Python on syntaksiltaan eli kirjoitusasultaan siistiä ja helppolukuista. Python on laajennettavissa useilla sovelluskehyksillä esimerkiksi Djangolla. (Python Software Foundation 2018)

Python syntyi Guido Van Rossumin halusta tuottaa ABC-ohjelmointikielestä paranneltu versio, jolla kyettäisiin virheidenkäsittelyyn ja kommunikointiin käyttöjärjestelmän kanssa. Nykyään Pythonin kehityksestä vastaa järjestö Python Software Foundation. Pythonin vuonna 2000 julkaistu versio 2.0 sisältää tekstin ja muistinkäsittelyä helpottavia ominaisuuksia. Vuonna 2008 julkaistu Python versio 3.0 poistaa vanhoja toimintoja ja parantaa syntaksia. Python on ladattavissa osoitteesta https://www.python.org/ (Kasurinen 2009, 7). Versiot Python 3.0 ja 2.0 eivät ole keskenään taaksepäin yhteensopivia ilman aputyökaluja (van Rossum 2006).

3.2 Django

Django on avoimen lähdekoodiin perustuva Pythonin sovelluskehys verkkosivustojen kehittämistä varten. Sovelluskehyksen tarkoituksena on helpottaa web-kehittäjän työtä tarjoamalla valmiita komponentteja web-sivustolla yleisesti tarvittuihin ominaisuuksiin, automatisointia, tietoturvaa ja käyttäjähallintaa. Sovelluskehyksen osaaminen lisää web-kehittäjän tuottavuutta. (Dauzon 2014, 27.)

Django perustuu model-view-controller -malliin (MVC), jossa sivuston rakenne jaetaan kolmeen erilliseen tasoon. Model eli malli pitää sisällään sivuston datarakenteen tietokantataulut. Views eli näkymä sisältää sivuston rakenteeseen liittyvät HTML-tiedostot, jotka palvelin esittää sivuston käyt-

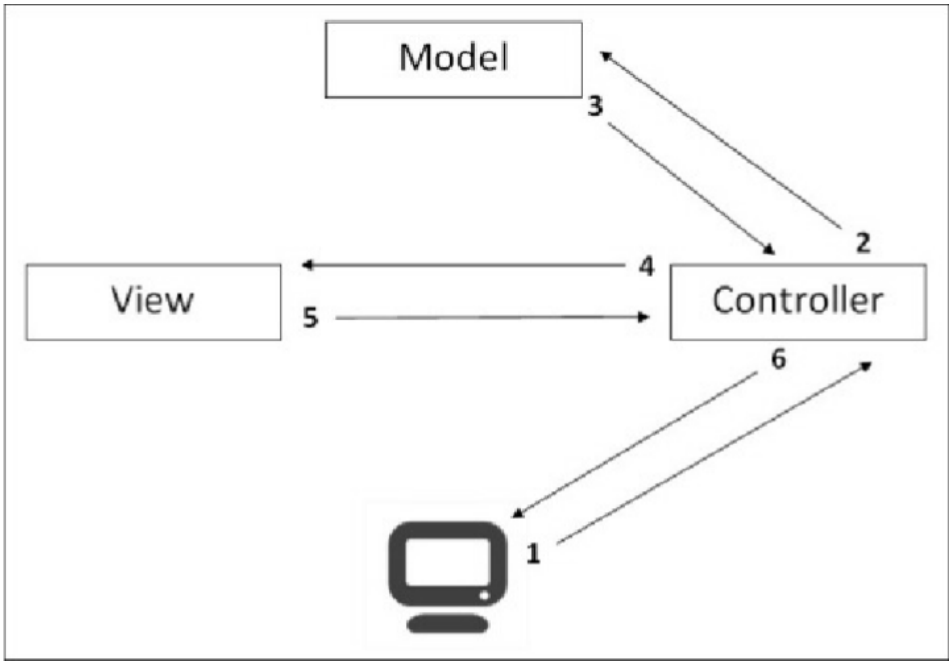
täjälle. Controller eli ohjain sisältää palvelinpuolen toimintoja joita sivuston käyttäjä ei näe, toimii välikappaleena mallin ja näkymän kanssa. (Dauzon 2014, 28)

Django kutsuu itseään MVC-malliin perustuvaa model-view-template -malliksi (MVT). Tämän mallin erona on ainoastaan nimitykset, view korvataan templatella ja controller korvataan viewsillä. (Dauzon 2014, 29)

MVC-mallia noudattava sivusto noudattaa seuraavia vaiheita (kuva 1):

1. Käyttäjä pyytää sivustoa palvelimelta.
2. Ohjain pyytää tietoja tietokannasta.
3. Malli palauttaa tiedot ohjaimelle.
4. Ohjain vie tiedot näkymän ulkoasu tiedostoon.
5. Näkymä palauttaa ohjaimelle tämän tiedoston.
6. Ohjain palauttaa ulkoasun ja tiedot käyttäjälle.

(Dauzon 2014, 28)



Kuva 1. Havainnekuva MVC -mallista (Dauzon 2014, 28).

Django tukee muun muassa MySQL, SQLite ja PostreSQL tietokantajärjestelmiä (Django 2018). Djangon käyttää Pythonin Object relational mapperia (ORM) tekemään linkityksen olioiden ja tietokannan välillä. Tästä syystä SQL-syntaksia ei tarvita vaan Djangon mallit kirjoitetaan luokaksi, joka muutetaan tietokantatauluksi. Luokan kentät määrittävät tietokantataulun sarakkeet ja taulun suhteen muiden taulujen kanssa (kuva 2). (Dauzon 2014. 73)

```python
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField()
    authors = models.ManyToManyField(Author)
    n_comments = models.IntegerField()
    n_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __str__(self):
        return self.headline
```

Kuva 2. Esimerkki tietokantataulujen luonnista (Django 2018a).

Djangossa on sisäänrakennettuna muutamia tietoturvaan liittyviä ominaisuuksia. Cross site scripting (XSS) hyökkäyksissä haittaohjelmat tallennetaan tietokantaan, josta sivusto hakee ja näyttää käyttäjille tai käyttäjää houkutellaan klikkaamaan linkkiä, jolloin hyökkääjän skripti suoritetaan käyttäjän selaimessa. Djangon templatet suojaavat yleisimpiä XSS-hyökkäyksiä vastaan, mutta suojaus ei ole täydellinen. Cross site request forgery (CSRF) hyökkääjä käyttää toisten käyttäjien tietoja näiden tietämättä. Djangon CSRF-suojauksessa tarkistetaan jokaisessa POST pyynnössä salainen avain. Hyökkääjän tulisi tietää tämä avain, jotta voisi hyödyntää toisen käyttäjän tunnuksia. Näiden esimerkkien lisäksi on SQL-injektiosuojaus, HTTPS-tuki ja muita toimintoja. (Django 2018b)

3.3 **Vagrant**

Web-pohjaisten projektien kehitystyö voi olla monimutkaista. Jokainen projekti tarvitsee omat kehitystyökalut ja kehitysympäristön. Projektien lisääntyessä kasvavat myös kehitystyökalujen määrä ja versioerot. Yhdellä projektilla voi olla useita työntekijöitä ja yhdellä työntekijällä voi olla useita

projekteja. Mikäli työntekijällä on omalla työasemallaan useita kehitysympäristöjä, muodostuu ongelmaksi ylläpitää johdonmukaista työskentelykokoonpanoa. Nämä ongelmat voidaan ratkaista virtualisoiduilla kehitysympäristöillä. (Peacock 2013, 21.)

Virtualisointi on laaja käsite, mutta ohjelmistokehityksessä virtualisoinnilla tarkoitetaan fyysistä tietokonetta tai palvelinta, jolla ajetaan yhtä tai useampaa virtuaalista tietokonetta (Pääkkö, 2015). Virtualisoinnilla voidaan siis jokainen projekti palvelimineen ja työkaluineen pitää omalla virtuaalikoneella. Virtualisoinnin avulla kyetään matkimaan oikeaa tuotantoympäristöä ja näin ollen uudet muutokset voidaan testata ensin virtuaalisesti ennen tuotantoon ajamista. (Peacock 2013, 21.)

Vagrant on virtuaalisten ympäristöjen rakentamiseen ja hallintaan tarkoitettu työkalu. Vagrant pyrkii tarjoamaan työkalun helposti määritettäviin ja helposti uudelleen tuotettaviin virtuaaliympäristöihin. Vagrantissa ei ole ulkoista käyttöliittymää, vaan ohjelmaa ajetaan komentokehoitteen kautta. (Peacock 2013, 22.)

Vagrant asentaa ja käynnistää käyttäjän puolesta kolmannen osapuolen virtuaalikoneen, esimerkiksi Oraclen VirtualBoxin. Tämän jälkeen kokoonpanoasetukset luetaan yhdestä tekstitiedostosta, Vagrantfilesta. Koska kokoonpanotiedot ovat Vagrantfilessa, voidaan tätä tiedostoa jakaa projektin työntekijöiden kesken. Ideaalitapauksessa uusi projektijäsen hankkii tiedoston ja ympäristö käynnistyy yhdellä komennolla. (Peacock 2013, 22.)

3.4 **Ansible & Sphinx**

Ansible on avoimen lähdekoodin IT-automatisointialusta. Ansible on tarkoitettu esimerkiksi pilvipalvelujen, kokoonpanohallinnan ja organisaatioiden sisäisten palveluiden käyttöönottoon ja konfigurointiin. (Ansible n.d.) Playbook on Ansiblen kokoonpano, käyttöönotto ja orkestrointikieli. Sen sijaan että käyttäjä antaisi useita käskyjä komentokehotteella, voidaan playbook-tiedostoon asettaa useita toimintoja ja ajaa ne kaikki kerralla yhteen tai useampaan järjestelmään. (Ellingwood 2014.) Opinnäytetyössä kehitysympäristön riippuvuudet ja ohjelmistot asennettiin Ansiblen playbookin avulla.

Full-text hakukoneet käsittelevät suuret määrät tekstiä nopeasti ja tehokkaasti sen perusteella, kuinka hyvin ne vastaavat käyttäjän hakukriteeriä. Parhaimmat hakukonejärjestelmät voivat luokitella hakutuloksia tiettyjen kenttien arvojen perusteella sekä omaavat tuen +/- ja Boolean operaattoreiden syntaksiin. (Krellenstein 2009.)

Sphinx on avoimen lähdekoodin full-text hakupalvelin, joka on suunniteltu tuottamaan suorituskykyisiä ja laadukkaita hakuja. Sphinx on kirjoitettu C++ ohjelmointikielellä ja toimii muun muassa Linux-, Windows- ja MacOS-käyttöjärjestelmissä. Sphinxin avulla voidaan tallennettua dataa lukea SQL-

ja NoSQL-tietokannasta sekä suoraan tiedostoista. Sphinx voidaan ohjata yleisimmillä skriptauskielillä, esimerkiksi Pythonilla. (Sphinxsearch, 2016.)

4 ALENNUSKUPONKIEN SUUNNITTELU

Tässä kappaleessa tarkastellaan projektin alkutilannetta ja pohditaan alennuskuponkeihin liittyviä vaatimuksia, ominaisuuksia ja logiikkaa.

4.1 Vaatimukset

Projektin alussa oli Levykauppa Äxän ja Sovelluskontti Oy:n välillä keskusteltu alustavasti tavoitteista ja toiminnollisuuksista kuponkeihin liittyen (Taulukko 1). Vaatimuksia tarkennettiin ja käyttötarkoituksista keskusteltiin asiakkaan ja toimeksiantajan kanssa opinnäytetyön aikana.

Taulukko 1. Asiakkaan haluamia ominaisuuksia

Alennuskupongin syöttö
Alennuskupongilla voimassaoloaika
Alennuksen jälkeinen loppusumma määrää kanta-asiakasleimojen määrän.
Postikuluttomuus
Kaksinkertaiset kanta-asiakasleimat
Käyttäjätunnukseen sidonnainen alennuskuponki
Alennus tietylle maksutavalle
Alennuskoodi tilauksen jälkeen
Prosenttiale tietyistä tuotteista

4.2 Alennuskuponkien ominaisuudet

Ominaisuuksia tarkastellessa huomattiin, että alennuskupongeilla haluttiin muokata useita ostoskorin ominaisuuksia. Jokainen kuponkityyppi tarvitsisi oman logiikan ostoskorin käsittelyyn. Tästä syystä kaikkia haluttuja ominaisuuksia ei siis kyettäisi toteuttamaan yhdentyyppisellä kupongilla. Ominaisuuksista kirjoitettiin yli kaksikymmentä käyttäjätarinaa. Käyttäjätarinat priorisoimalla pystyttiin toteamaan, että alennuskuponkien vaatimukset voidaan jakaa etuihin ja ehtoihin (Taulukko 2).

Taulukko 2. Ominaisuuksien edut ja ehdot

Edut	Ehdot
Alennus eurot	Käyttäjätunnus
Alennus prosentti	Käyttömäärä
Postikuluttomuus	Maksutapa
Kanta-asiakasleima määrä	Ostoskorin vähittäisarvo
Kanta-asiakasleima kerroin	Tuotekohtainen
	Tuoteryhmä kohtainen
	Ajanjakso

Tarkastelemalla taulukkoa 2 kyetään tekemään havaintoja, jotka tulee ottaa huomioon alennuskuponkien kehitystyössä. Yleisellä tasolla edut edustavat ohjelmallista logiikkaa, jonka avulla ostoskoria muokataan. Tyypillisesti yhdellä alennuskupongilla on vain yksi etutyyppi, mutta teoriassa mikään ei pois sulje useamman edun tarjoamista yhdellä kupongilla. Ehdot ovat tietoja, joille tulee tehdä käsittelyä sekä tarkistuksia. Yhdellä kupongilla voi olla useita ehtoja ja jotkin ehdoista, kuten esimerkiksi ajanjakso ja ostoskorin vähittäisarvo, ovat lähes pakollisia määreitä.

Alennuskuponkeja luodaan ja hallinnoidaan tietokantaratkaisun avulla. Aluksi kuponkeja voidaan luoda komennoilla ja skripteillä. Myöhemmässä vaiheessa on tarkoitus kehittää kauppiaan näkökulmasta helppokäyttöisempi ratkaisu. Tietokannan ja tietokantataulujen suunnittelussa ja toteutuksessa täytyy myös ottaa huomioon taulukossa 2 esitetyt ehdot. Esimerkkinä merkintä alennuskupongin käytöstä käyttäjätunnuksen kanssa tarvitsee oman tietokantataulunsa ja on hyvä paikka liittää mukaan tilauksen tunniste. Vaatimuksia ja määritelmiä on siis kaiken kaikkiaan useita ja jokaisessa käyttötapauksen mahdollisessa toteutuksessa on lukuisia variaatiota. Lisäksi erityistä huomiota tulee kiinnittää alennuskuponkien väärinkäytösten estämiseksi, sillä väärinkäytökset tarkoittaisivat verkkokauppiaalle potentiaalisesti mittavaa taloudellista tappiota. Tämän perusteella onkin järkevämpää keskittyä toteuttamaan yksi kuponkiominaisuus kerrallaan ja lisätä uusia kuponkiominaisuuksia jatkossa, sen sijaan että yritettäisiin ottaa huomioon kaikki vaatimukset alusta lähtien.

Tarkastellaan alennuskupongin toimintaa yleisellä tasolla. Käyttäjä siirtää verkkokaupassa tuotteita ostoskoriin ja siirtyy ostoskorin sivulle. Käyttäjä syöttää alennuskuponkikoodin tekstikenttään ostoskori sivulla ja lähettää tiedon syötetystä tiedosta palvelimelle. Tämän jälkeen tarkastetaan, onko järjestelmässä syötettä vastaavaa alennuskoodia ja onko tämän alennuskoodin voimassaoloaika erääntynyt. Mikäli alennuskoodia ei löydy tai voimassaoloaika on umpeutunut, saa käyttäjä ilmoituksen virheellisestä syötteestä. Jos taas tiedot löytyvät, tallennetaan tieto käyttäjän sessioon ja ostoskorin tietoja muokataan alennuskupongin tyypin mukaan. Kun käyttäjä on asettanut tilauksen, tiedon alennuskupongin käytöstä tulisi ilmetä sekä kauppiaalle että asiakkaalle.

Levykauppa Äxän verkkokauppa on ollut jatkuvassa kehityksessä ja toiminnassa useita vuosia. Pitkälle kehitetyn järjestelmän sisäistäminen on aikaa vievä prosessi kokeneellekin ammattikehittäjälle Koska opinnäytetyön tekijällä ei ollut kokemusta Python-ohjelmointikielestä ja sen sovelluskehyksestä Djangosta, tultiin yhdessä toimeksiantajan kanssa siihen johtopäätökseen, että olisi viisaampaa aloittaa kehitystyö rakentamalla yksinkertainen prototyyppimäinen harjoitustyö. Näin voitaisiin tehdä oppimishavaintoja prototyypistä sekä verkkokaupasta rinta rinnan.

5 KEHITYSYMPÄRISTÖN ASENNUS

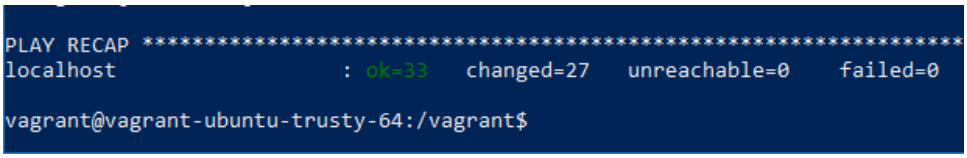
Tässä kappaleessa kuvataan virtuaalisen kehitysympäristön pystyttämisessä tarvittuja vaiheita. Verkkokaupan kehitysympäristön pystytykseen tarvittavat tiedot saatiin toimeksiantaja Sovelluskontti Oy:ltä. Verkkokaupan lähdekoodi ladattiin Subversion-versiohallintaohjelmistolla etävarastosta omalle koneelle. Vagrant-ympäristöön tarvittavat tiedostot ladattiin Git-versiohallinta ohjelmiston kautta etävarastosta.

5.1 Virtuaalikone ja riippuvuuksien asennus

Virtuaalisen kehitysympäristön asennuksessa tarvitaan Vagrantin lisäksi VirtualBoxia ja Ansiblea. Ansible tuotti ensimmäisen ongelman. Opinnäytetyön tekijä käytti Windows-käyttöjärjestelmää, jolle Ansible ei tarjoa käyttötukea. Sovelluskontti Oy on luonut ympäristön käyttäen Mac-käyttöjärjestelmää. Tätä varten asennettiin Ubuntun komentokehoite, jolla kyettiin käyttämään Linux-pohjaisia komentoja Windows-alustalla. Jotta komentokehoite toimisi täytyi Windows Subsystems for Linux sallia. Tämä onnistui antamalla PowerShellissä järjestelmänvalvojana komento sekä valitsemalla Windowsin asetuksista Developer mode.

Ubuntun kometokehoitteelle tulee antaa komento
export VAGRANT_WSL_ENABLE_WINDOWS_ACCESS="1” (1)
Jotta Vagrant suostuu toimimaan Linuxin alijärjestelmässä. Komento täytyi antaa joka kerta kun bash-pohjainen terminaali käynnistettiin. Tämän jälkeen virtuaalikone voitiin käynnistää. Käynnistyksen lisäksi asennettiin Ansiblen playbookin avulla tarvittavat riippuvuudet kuten Python, PostreSQL ja Spinxh. Komento palautti virheen tiedoston liian avoimista lukuoikeuksista. Kun Linuxin chmod-komento ei suostunut muokkaamaan oikeuksia, avuksi otettiin Cygwin-terminaali, jolla oikeuksia kyettiin muuttamaan, mutta Ansible ei tästä huolimatta suostunut yhteistyöhön.

Ratkaisuna päädyttiin asentamaan Ansible suoraan virtuaaliympäristöön ja ajamaan asennuskomento virtuaalikoneelta käsin. Muutaman epäonnistuneen ajon ja tekstitiedostojen muokkaamisen jälkeen saatiin riippuvuudet asennettua onnistuneesti (kuva 3). Tämän jälkeen Djangon riippuvuudet asennettiin erillisellä komennolla lähdekoodin juuressa olevasta tekstitiedostosta.



Kuva 3. Riippuvuuksien onnistunut asennus Ansiblen avulla.

5.2 PostgreSQL-tietokannan asennus

Tietokantakannasta ladattiin kopio, joka vastaa tuotannossa käytössä olevaa tietokantaa. Koska virtuaalikoneen PostgreSQL-asennus estää muista kuin localhost-osoitteesta tulevat yhteydet, asetustiedostoja täytyi muokata, jotta kehitysympäristöön voitiin yhdistää myös virtuaalikonetta isännöivältä tietokoneelta.

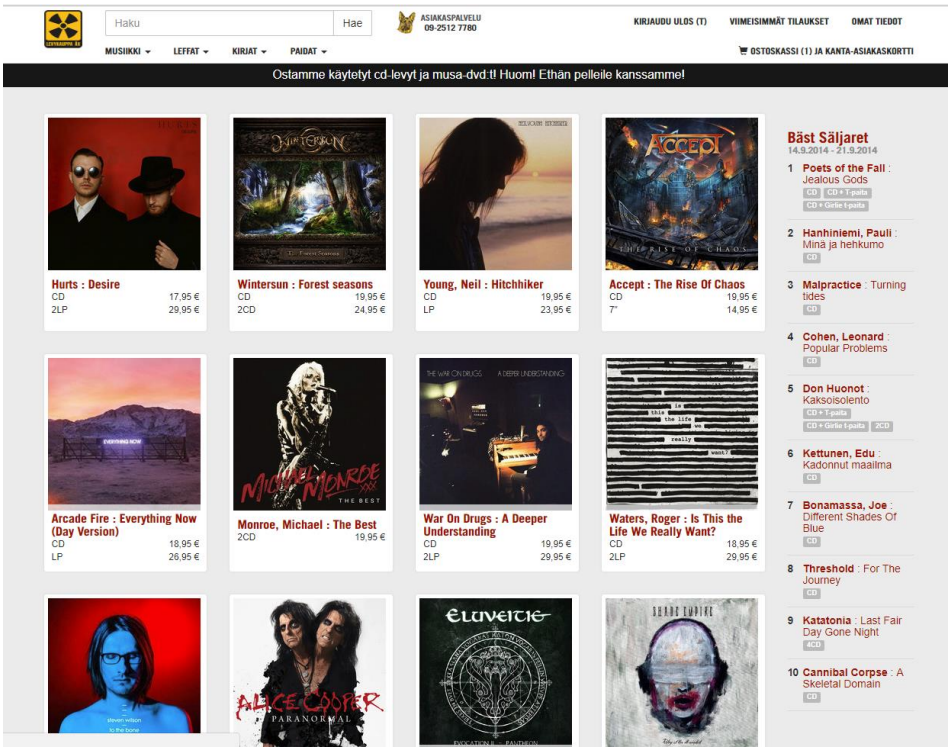
Seuraavaksi luotiin PostgreSQL:n terminaalin kautta tyhjä tietokanta käyttäjineen ja asetettiin käyttäjien oikeudet. Jonka jälkeen aloitettiin Levykauppa Äxän tietokannan kopiointi. Tietokannan kopion palauttaminen tietokantaan aiheutti nopeasti useita virheitä. Ratkaisuna muokattiin luotujen käyttäjien oikeuksia sekä siirrettiin kopion luonti virtuaalikoneelle: näin vältyttiin Windows-alustasta johtuvilta virheiltä, virtuaalikoneen ollessa Linux-alustaan pohjautuva. Onnistuneen palautuksen jälkeen sivuston palvelin käynnistettiin ja sivusto toimi. Ilman tyylitiedostoja sivusto oli kuitenkin vielä varsin alkeellisen näköinen (kuva 4).

Kuva 4. Levykaupan sivusto ilman tyylitiedostoja.

5.3 Nginx- ja Sphinx-konfigurointi

Nginx-web-palvelin ja Sphinx-hakupalvelu asennettiin Ansiblen avulla aikaisemmin. Jotta sivuston tyylitiedostot saataisiin käyttöön, tarvitsi Nginx-asetukset kopioida ensin virtuaalikoneelle. Tämän jälkeen Nginx käynnistettiin uudelleen, jotta asetuksiin saataisiin luotua oikeat linkitykset. Django-palvelin käynnistettiin samaan tapaan kuin ennenkin, mutta staattiset tiedostot kuten CSS-tyylitiedostot välitettiin Nginxin kautta. Sivusto näytti ulkoasullisesti oikeankaltaiselta, mutta ei vielä toiminut kategorioiden ja tuotteiden osalta.

Tuotteiden saamiseksi sivustolle tarvitaan Nginx. Samaan tapaan kuin Nginxin kanssa, Sphinx asetukset kopioitiin ensin virtuaalikoneelle, minkä jälkeen palvelu käynnistettiin uudelleen. Tämän jälkeen yhdistettiin palvelu tietokannan kanssa. Onnistuneen yhdistämisen merkiksi sivuston kategoriat eivät enää aiheuttaneet virheitä, mutta olivat tyhjiä. Seuraavaksi ajettiin Python skripti, jolla saatiin tuotteet indeksoitua hakupalveluun. Laajasta tuotevalikoimasta johtuen tämä kesti usean minuutin. Lopputuloksena on tuotantoa vastaava sivusto, jonne voidaan turvallisesti tehdä ohjelmallisia muutoksia (kuva 5).



Kuva 5. Levykaupan sivusto tyylien kanssa.

6 ALENNUSKUPONKIEN TOTEUTUS

Tässä kappaleessa tarkastellaan prototyypin toteutuksen vaiheita. Prototyypillä oli kaksi tarkoitusta. Työskentelyn kautta uuden kielen ja sovelluskehyksen sisäistäminen sekä tutustua ja luoda simppeleitä ratkaisuja joita tarvittaisiin alennuskuponkijärjestelmässä. Prototyypissä luodut ratkaisut eivät välttämättä suoraan soveltuisi käytettäväksi Levykauppa Äxän verkkokaupassa, mutta prototyypin avulla voitaisiin demonstroida haasteita ja keksiä vaihtoehtoisia ratkaisuja käytettäväksi verkkokaupassa. Prototyypissä keskityttiin seuraaviin tehtäviin:

- mallit Djangossa
- keskittyä toimintoihin, ei ulkoasuun
- alennuskupongin syöttölomake
- alennuskupongille tehtävät tarkistukset
- yksinkertaisen ostoskorin ja session luonti
- tuotteiden lisäys ostoskoriin
- ostoskorin logiikan tutustuminen
- ostoskorin arvojen käsittely
- aluksi yksi alennustoiminto
- mukaan käyttäjä ja lisätarkistukset.

6.1 Alennuskuponkimalli

Prototyyppi aloitettiin luomalla Alennuskupongille oma app coupons. Seuraavaksi luotiin alennuskuponkimalli Djangon models.py tiedostoon (ohjelmakoodi 1). Mallin kentät ovat code, valid_from, valid_to ja code_discount. Code on alennuskupongin nimi, jota verrataan käyttäjän syötteeseen. Valid_from ja valid_to kentillä määritetään alennuskupongin voimassaoloaika. Valid_from on tarpeellinen, mikäli kauppias haluaisi luoda kupongit etukäteen ennen käyttöönottoa. Code_discount edustaa kupongin tyyppiä, eli toiminnallisuutta, jota kupongin halutaan toteuttavan. Code_discountille määritettiin valmiiksi vaihtoehdot, jotta jokaiselle kuponkityypille voitaisiin luoda logiikka joka käsittelisi ostoskoria omilla säännöillään. Discount_value edusti haluttua alennuksen määrää. Esimerkkinä arvolla 10 voitaisiin halutessa antaa joko kymmenen euron tai kymmenen prosentin alennus. Tyypillisesti alennuskupongit sisältävät ehdon, jossa alennuksen saadakseen asiakkaan tulisi ostaa tuotteita tietyllä summalla. Cart_req_value edusti tätä arvoa. Django luo tietokannoista esiintyvän ID tiedon automaattisesti, joten tätä ei tarvitse erikseen kentäksi määrittää. Tietokantaan lisättiin muutamia alennuskuponkeja, jotta lomaketta pystyttäisiin testaamaan.

```
class Coupon(models.Model):

    euro_ale = 'euroale'
    prosentti_ale = 'prosenttiale'
    posti_ale = 'postiale'

    CODE_DISCOUNT_CHOICES = (
        (euro_ale, 'euroale'),
        (prosentti_ale, 'prosenttiale'),
        (posti_ale, 'postiale'),
    )

    code = models.CharField(max_length=20)
    valid_from = models.DateTimeField()
    valid_to = models.DateTimeField()
    code_discount = models.CharField(max_length=20,
                                     choices=CODE_DISCOUNT_CHOICES,
                                     default=euro_ale)
    discount_value = models.IntegerField()
    cart_req_value = models.IntegerField(default=50)

    def __str__(self):
        return self.code
```

Ohjelmakoodi 1. Alennuskupongin tiedosto models.py.

6.2 **Alennuskuponkilomake**

Seuraavaksi tehtiin yksinkertainen lomake, jolla voitiin tarkastaa alennuskupongin olemassaolo ja voimassaoloaika tietokannasta. Koska lähes kaikki osiot jaetaan omiin tiedostoihin Djangossa, myös lomakkeille tehdään oma tiedosto nimeltä forms.py (ohjelmakoodi 2). Forms.Modelform on näppärä esimerkki siitä, kuinka Djangolla pyritään helpottamaan kehittäjän työtä. Meta luokalla kerrottiin käytettäväksi Coupon-mallia. Fields muuttujalla valittiin, mitkä kentät otettiin mukaan esitettäväksi lomakkeen sivulla. Tässä tapauksessa tarvittiin vain yksi eli alennuskupongin nimi.

```
class CouponApplyForm(forms.ModelForm):

    class Meta:
        model = Coupon
        fields = ['code']
```

Ohjelmakoodi 2. Alennuskupongin tiedosto forms.py.

Seuraavaksi kirjoitettiin lomakkeen ohjain tiedosto views.py (ohjelmakoodi 3). Ohjelmakoodin 2 lomakemalli liitettiin näkymään form_class muuttujalla. Post metodissa käsitellään tietoja, kun lomake lähetetään. Coupon.objects.get metodi haki käyttäjän syötettä vastaavan alennuskupongin. Tämän jälkeen kupongille tehtiin tarkastuksia olemassaoloon ja voimassaoloaikaan. Ostoskorin valmistuttua myöhemmin lisättiin tarkistus ostoskorin olemassaololle. Lisäksi verrattiin, oliko ostoskorin kokonaissumma suurempi kuin kupongissa määritetty ostoskorin vähittäismäärä. Virheilmoitukset välitettiin käyttäjälle Djangon messages.error metodin kautta.

```
class CouponView(View):
    form_class = CouponApplyForm

    def post(self, request):
        form = self.form_class(request.POST)

        if form.is_valid():
            try:
                code = form.cleaned_data['code']
                coupon = Coupon.objects.get(code=code)
                tz = timezone.get_current_timezone()
                now = datetime.datetime.now(tz)
                cart = Cart(request)

                if not cart:
                    messages.error(request, "Ostoskori on tyhja")

                if coupon.cart_req_value >= cart.get_total_price():
                    messages.error(request, 'Ostoskori ei tayta kriteereja')

                if now > coupon.valid_from and now < coupon.valid_to:
                    # kuponki voimassa, asetetaan sessioon.
                    request.session['coupon_id'] = coupon.id
                else:
                    messages.error(request, 'Kuponki ei ole voimassa')
            except Coupon.DoesNotExist:
                messages.error(request, 'Kuponkia ei loydy')
        return redirect('cart:cart_detail')
```

Ohjelmakoodi 3. Alennuskupongin views.py tiedosto.

Virheilmoitusten esittäminen tapahtui ostoskorin HTML-tiedostossa, jonne lomake liitettiin (ohjelmakoodi 4). Mikäli lomakkeen tarkistuksessa tapahtuisi yksi tai useampi virhe siitä ilmoitettiin käyttäjälle. Tiedosto sisälsi myös Djangon CSRF-suojausmerkin. Lomakkeen action kentän URL viitteellä viitataan ohjelmakoodi 5 coupon appsin urls.py-tiedostoon, jonka välityksellä yhdistetään views-tiedoston logiikka lomakkeeseen. Urls.py-tiedostossa (ohjelmakoodi 5) asetettiin näkymälle haluttu verkko-osoite. Tähän tarkoitukseen /coupons/ riitti mainiosti.

```
<h2>Kuponkiproto</h2>

<form action="{% url "coupons:form" %}" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Käytä koodi" />
    {% if messages %}
        {% for message in messages %}
            {{ message }}
        {% endfor %}
    {% endif %}
</form>
```

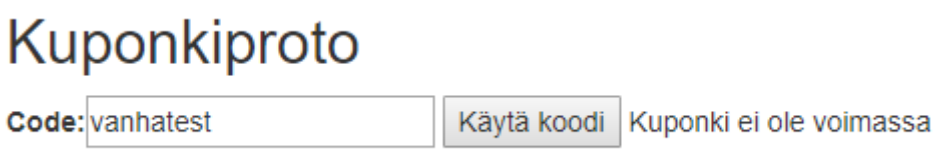
Ohjelmakoodi 4. Lomakkeen ohjelmakoodi HTML-tiedostossa.

```
app_name = 'coupons'

urlpatterns = [
    # /coupons/
    url(r'^$', views.CouponView.as_view(), name='form'),
]
```

Ohjelmakoodi 5. Alennuskupongin tiedosto urls.py.

Lopputuloksena onnistuttiin luomaan yksinkertainen lomake. Lomake vertasi syötettä alennuskuponkien tietokantaan ja teki tarkistuksia. Kuvan 6 esimerkissä syötettiin alennuskuponkikoodi, jonka voimassaoloaika oli umpeutunut.



Kuva 6. Esimerkkisyöte erääntyneelle alennuskupongille.

6.3 Tuotteet

Jotta ostoskoria voitaisiin käsitellä, tarvittiin tuotteita. Tavoitteena oli luoda tuotteita, luoda näkymä jossa listata tuotteet sekä luoda yksittäisen tuotteen näkymä. Yksittäisen tuotteen sivulta käyttäjä voisi lisätä tuotteen ostoskoriin. Seuraavaksi luotiin shop niminen app, jonka models tiedostoon luotiin tuotteiden malli (ohjelmakoodi 6). Mallin kaikkia kenttiä ei käytetty prototyypissä, mutta tässä oli hyvä mahdollisuus tutustua tarkemmin mallien kenttätyyppeihin. Oleellisimmat kentät olivat name, price ja

slug. Slug tarkoittaa verkko-osoitteissa käytettävää merkkijonoa, jolla voidaan luoda tyylikkäämpiä verkko-osoitetta sekä edistää hakukonenäkyvyyttä.

```
class Product(models.Model):
    name = models.CharField(max_length=200, db_index=True)
    slug = models.SlugField(max_length=200, db_index=True)
    description = models.TextField(blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField()
    available = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ('name',)
        index_together = (('id', 'slug'),)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('shop:product_detail', args=[self.id, self.slug])
```

Ohjelmakoodi 6. Tuotteiden models-tiedosto.

Tämän jälkeen luotiin views tiedosto tuotteiden esittämistä varten (Ohjelmakoodi 7). Djangossa appsin mallin datat oletuksena asetetaan muuttujaan object_list. Muuttujalla context_object_name voitiin uudelleennimetä object_list sisältöä paremmin kuvaavammaksi eli tässä tapauksessa all_products.

```
class ListProducts(generic.ListView):
    template_name = 'shop/product/list.html'
    context_object_name = 'all_products'

    def get_queryset(self):
        return Product.objects.all()
```

Ohjelmakoodi 7. Tuotteiden views-tiedosto.

Ohjelmakoodissa 8 iteroitiin tietokannan tuotteet HTML-tiedostoon käyttäen apuna aikaisemmin määritettyä all_products muuttujaa. Tuotteiden linkit johdattivat käyttäjän yksittäisen tuotteen näkymään.

```
<h1>Tuotteita</h1>

{% if all_products %}
    {% for product in all_products %}
        <p>
            <a href="/shop/{{ product.id }}/{{ product.slug }}">
                Tuote: {{ product.name }} Hinta {{ product.price }}€
            </a>
        </p>
    {% endfor %}
{% endif %}
```

Ohjelmakoodi 8. Tuotteiden listaus HTML-tiedostossa.

Seuraavaksi views tiedostoon luotiin näkymä yksittäiselle tuotteelle. Mikäli tuotetta ei löydy käyttäjälle esitetään 404 virhe eli sivua ei löydy. Näkymään yhdistettiin ostoskorin yhteydessä luotu tuotteen lisäyslomake (ohjelmakoodi 9). Lisäys lomakkeen luomisessa käytettiin samoja periaatteita kuin alennuskuponki lomakkeen kanssa. Yksittäisen tuotteen HTML tiedosto ei sisältänyt mitään uutta, jota tässä opinnäytetyössä olisi jo esitetty.

```
def product_detail(request, id, slug):
    product = get_object_or_404(Product, id=id, slug=slug, available=True)
    cart_product_form = CartAddProductForm()
    return render(request, 'shop/product/detail.html',
                  {'product':product, 'cart_product_form': cart_product_form})
```

Ohjelmakoodi 9. Yksittäisen tuotteen ohjelmakoodi views-tiedostossa.

Urls tiedostossa tuotteiden listaussivu oli samankaltainen kuin aikaisemmin esitelty alennuskuponki. Erotuksena aikaisemmassa yksittäisen tuotteen sivun verkko-osoitteessa käytettiin hyväksi tuotteen ID ja slug tietoja (ohjelmakoodi 10)

```
urlpatterns = [
    #/shop/
    url(r'^$', views.ListProducts.as_view(), name='product_list'),
    #/shop/ID/Slug/
    url(r'^(?P<id>\d+)/(?P<slug>[-\w]+)/$', views.product_detail, name='product_detail'),
]
```

Ohjelmakoodi 10. Tuotteiden urls-tiedosto.

Lopputulokset sekä tuotelistauksesta että yksittäisestä tuotteesta on esitelty kuvassa 7.

Kuva 7. Tuotelistaus (yllä) ja yksittäinen tuote verkkosivustolla (alla).

6.4 **Ostoskori**

Tuotteiden valmistuttua oli seuraavana vuorossa ostoskori. Ostoskorin tehtävänä on säilöä tuotteet, jotka verkkokaupan asiakas haluaisi ostaa. Tyypillisesti verkkokauppojen ostoskorit säilyttävät tiedon ostoskorin sisällöstä vaikka asiakas poistuisi sivustolta ja palaisi asioimaan myöhemmin. Tämä saavutetaan käyttämällä niin sanottuja sessioita. Djangon sessiokehys otettiin käyttöön ympäristön settings tiedoston kautta. Ostoskorin aloitettiin luomalla cart-ohjelmaluokka. Liite 1 sisältää cart-luokan ohjelmakoodin.

Cart-luokan init-konstruktorissa asetettiin sessio ostoskoriin, mikäli ostoskoria ei olisi olemassa luotiin tyhjä ostoskori sessioon. Lisäksi alennuskuponki liitettiin ostoskoriin id-tunnisteella. add-metodi tallensi parametrina saadun tuotteen ostoskoriin ja otti huomioon tuotteiden lukumäärän, vaikka tuotteita lisättäisiin myöhemmin lisää ostoskoriin. Lopuksi kutsuttiin save-metodia. Save-metodi päivittää session ostoskorin ja merkitsee session muokatuksi. Modified-avainsana kertoo Djangolle session muuttuneen ja että session tila tarvitsee uudelleentallennuksen. Iter-metodissa ostoskorin tuotteiden hinnat muutettiin desimaali muotoon ja asetettiin total_price-määre jokaiselle tuotteelle. Tällä tavoin kyettiin laskemaan ja esittämään yhden tuotetyypin kokonaishinta yhdelle riville, vaikka tuotteita olisi useampi kappale ostoskorissa. len-metodi palautti tuotteiden kokonaismäärän ja get_total_price-metodissa palautettiin ostoskorin kokonaisarvo. Clear-metodissa tyhjennettiin koko ostoskori tuotteista ja samalla poistettiin alennuskuponki sessiosta. Tämä ei olisi hyvä tapa oikeassa

ympäristössä poistaa tuotteita ostoskorista, mutta kelpaa prototyyppitarkoituksessa. Coupon-metodissa palautetaan coupon-objekti, mikäli ostoskorissa oli id tunnisteen mukainen alennuskuponki. Metodi get_discount käsittelee alennuksia ja palauttaa alennukseen oikeuttavan desimaaliluvun alennuskuponkityypin perusteella. Metodi get_total_price_after_discount palautti ostoskorin kokonaisarvon ja alennuksen välisen erotuksen.

Seuraavaksi tehtiin ostoskorin views-tiedosto (ohjelmakoodi 11). cart_add käsitteli tuotteen ostokorin lisäyksessä tarvittuja toimintoja. Cart_detailissä palautetaan ostoskori ja alennuskuponkilomake käyttäjälle. Cart_clear tyhjensi ostoskorin ja palautti ostoskorisivun käyttäjälle.

```
@require_POST
def cart_add(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    form = CartAddProductForm(request.POST)

    if form.is_valid():
        data = form.cleaned_data
        cart.add(product=product, quantity=data['quantity'], update_quantity=data['update'])
    else:
        print("not valid form")

    return redirect('cart:cart_detail')

def cart_detail(request):
    cart = Cart(request)
    form = CouponApplyForm()
    return render(request, 'cart/detail.html', {'cart': cart, 'form': form})

def cart_clear(request):
    cart = Cart(request)
    cart.clear()
    return HttpResponseRedirect("/cart")
```

Ohjelmakoodi 11. Ostoskorin views-tiedosto.

Ostoskorin HTML-tiedostossa (kuva 8) päätavoitteena oli tulostaa ostoskorin tuotetiedot ja kokonaishinta ilman alennusta. Mikäli ostoskorissa on alennus myös alennuksen arvo ja kokonaishinta alennuksen kanssa tulostettiin. Kuvassa ostoskoriin on liitetty kuponki jolla asiakas saa kymmenen prosentin alennuksen ostoskorin kokonaissummasta ostoskorin arvon ollessa yli viisikymmentä euroa.

Kuva 8. Ostoskorin näkymä käyttäjälle.

6.5 **Käyttäjäsidonnaisuus**

Prototyypin tehtävälistan viimeinen tehtävä oli ottaa huomioon sivuston käyttäjä alennuskupongin yhteydessä. Tällä tarkoitettiin merkintää siitä, onko sivuston käyttäjä lunastanut alennuskupongin. Tavallisesti tällainen merkintä alennuskupongin käytöstä lisättäisiin tietokantaan onnistuneen maksutapahtuman jälkeen. Prototyypissä ei kuitenkaan maksuja käsitelty, joten aikaisemmin luotu lomake ja ostoskorisivu toimi sijaisena maksutapahtumalle. Käytetyille alennuskupongeille luotiin oma tietokantataulu. Ohjelmakoodissa 12 tietokantataulun kentissä viiteavaimina käytettiin käyttäjätunnustaulun ja alennuskuponkitaulun ID-tunnusta ja lisättiin aikaleimalle oma kenttä. Mikäli tilauksia käsiteltäisiin myös tilauksen ID-tunnus olisi luonnollinen lisäys tähän tauluun.

```
class RedeemedCoupon(models.Model):
    username = models.ForeignKey(User)
    coupon = models.ForeignKey(Coupon)
    used_coupon = models.BooleanField()
    used_at = models.DateTimeField()
```

Ohjelmakoodi 12. Käytettyjen alennuskuponkien malli.

Alennuskuponkilomakkeen views-tiedostoon lisättiin tarkistus, jossa katsottiin sisältääkö RedeemedCoupon taulu riviä sivustolle kirjautuneen käyttäjän ja syötetyn alennuskuponkikoodin tiedoista. Mikäli rivi löytyi, alennuskuponkia ei lisätty ostoskoriin. Alennuskupongin käytöstä lisättiin merkintä tietokantatauluun samalla kun alennuskuponki liitettiin ostoskoriin. Kuvassa 9 on esitetty RedeemedCoupon tietokantataulun sisältöä.

id	used_coupon	coupon_id	username_id	used_at
Filter	Filter	Filter	Filter	Filter
7	1	11	1	2018-02-23 0...
8	1	7	1	2018-02-23 0...
9	1	10	1	2018-02-23 0...

Kuva 9. RedeemedCoupon taulu tietokannassa.

7 YHTEENVETO

Opinnäytetyön tavoitteena oli tutkia alennuskuponkeja ja niiden ominaisuuksia sekä toteuttaa alennuskuponkijärjestelmä. Yksinkertaisesti ajateltuna alennuskupongit vain muokkaavat ostoskorin hintatietoja. Todellisuudessa alennuskupongeilla on useita ehtoja, määritteitä ja käyttötapauksia, joilla kaikilla on omat tekniset vaatimukset ja haasteensa ostosprosessin eri vaiheissa. Lähtökohtaisesti oli selvää, ettei kokonaista alennuskuponkijärjestelmää saada opinnäytetyölle varatussa ajanjaksossa valmiiksi. Tästä huolimatta opinnäytetyö oli mielenkiintoinen projekti. Uusien asioiden oppiminen olikin yksi henkilökohtaisesta tavoitteesta.

Projektissa päästiin tutustumaan uusiin ohjelmointikieliin, ja kokemusta on kertynyt ohjelmistokehityksen eri tekniikoista ja työskentelytavoista, asiakasrajapintaa unohtamatta. Vaikkei opinnäytetyön käytännön osuudessa käsitelty asiakkaan verkkokauppaympäristöä, oli silti mielestäni erityisen antoisaa päästä näkemään oikean verkkokaupan ohjelmakoodillinen skaala ja tutkia sen toiminnallisia rakenteita ja ratkaisuja. Prototyyppi oli hyvä toteutusvalinta projektille, sillä näin kyettiin oppimaan prosessin logiikkaa uusien ohjelmointikielen kanssa sekä havainnollistaa mahdollisia haasteita.

Opinnäytetyön jälkeen tarkoituksena on aloittaa alennuskuponkien toteuttaminen Levykauppa Äxän ympäristöön ja ottaa alennuskupongit käyttöön verkkokaupassa.

LÄHTEET

Ansible (n.d.). How Ansible works. Haettu 1.2.2018 osoitteesta https://www.ansible.com/overview/how-ansible-works

Dauzon, S. (2014). *Django Essentials*. Packt Publishing

Django (2018a). Making queries Haettu 1.2 osoitteesta https://docs.djangoproject.com/en/2.0/topics/db/queries/

Django (2018b) Security in Django Haettu 1.2 osoitteesta https://docs.djangoproject.com/en/2.0/topics/security/

Ellingwood, J. (2014). How To Create Ansible Playbooks to Automate System Configuration on Ubuntu Haettu 1.2.2018 osoitteesta https://www.digitalocean.com/community/tutorials/how-to-create-ansible-playbooks-to-automate-system-configuration-on-ubuntu

Hallavo, J. (2013). *Verkkokaupan rautaisannos.* Talentum

Kasurinen, J. (2009). *Python 3 ohjelmointi*. 1.painos. Jyväskylä: Docento.

Krellenstein, M. (2009). Full Text Search Engines vs. DBMS. Haettu 25.1.2018 osoitteesta https://lucidworks.com/2009/09/02/full-text-search-engines-vs-dbms/

Lehmann, R (2017). How you can spot fake coupons on Facebook -- and why it matters. Haettu 1.2.2018 osoitteesta https://www.bradsdeals.com/blog/is-this-coupon-fake

Melé, A. (2015) *Django By Example.* Packt Publishing

Mysecurityawareness (n.d.). Coupon Scams Promise Savings but Deliver Spam Haettu 1.2.2018 osoitteesta http://mysecurityawareness.com/article.php?article=377&title=-coupon-scams-promise-savings-deliver-spam#.WnLa6aiWaUk

Pathath, R (2015). Coupon Redemption System. Opinnäytetyö. Tietojenkäsittelytiede. University of Missouri. Haettu 25.1.2018 osoitteesta https://mospace.umsystem.edu/xmlui/bitstream/handle/10355/48351/PathathCouRedSys.pdf?sequence=1&isAllowed=y

Peacock, M (2013). Creating Development Environments with Vagrant. Packt Publishing

Python Software Foundation. Python Frequently Asked Questions. Haettu 15.01.2018 osoitteesta

https://docs.python.org/2.7/faq/

Pääkkö, J. (2015). Työkalupakkisi onnistuneeseen DevOpsiin Blogijulkaisu 10.3.2015. Haettu 19.1.2018 osoitteesta https://gofore.com/tyokalupakkisi-onnistuneeseen-devopsiin/

van Rossum, G. (2006). PEP 3000 -- Python 3000 Haettu 15.01.2018 osoitteesta https://www.python.org/dev/peps/pep-3000/

Sphinxsearch (2016). Sphinx 2.2.11-release reference manual. Haettu 25.1.2018 osoitteesta http://sphinxsearch.com/docs/current.html#about

Thomas, G. (2014). Ecommerce Tips: Proper Use of Generic and Unique Coupon Codes. Blogijulkaisu 19.11.2014. Haettu 25.1.2018 osoitteesta https://blog.justuno.com/ecommerce-tips-generic-and-unique-coupon-codes

Wong, D. (2016). Smart Strategies for Issuing Ecommerce Coupons. Blogi-julkaisu 25.7.2016 Haettu 25.1.2018 osoitteesta https://www.huffington-post.com/danny-wong/smart-strategies-for-issu_b_8196662.html

Cart-luokan ohjelmakoodi

```
class Cart(object):

    def __init__(self, request):
        self.session = request.session
        cart = self.session.get(settings.CART_SESSION_ID)

        if not cart:
            cart = self.session[settings.CART_SESSION_ID] = {}
        self.cart = cart

        self.coupon_id = self.session.get('coupon_id')

    def add(self, product, quantity=1, update_quantity=False):
        product_id = str(product.id)
        if product_id not in self.cart:
            self.cart[product_id] = {'quantity': 0, 'price': str(product.price)}

        if update_quantity:
            self.cart[product_id]['quantity'] = quantity
        else:
            self.cart[product_id]['quantity'] += quantity

        self.save()

    def save(self):
        self.session[settings.CART_SESSION_ID] = self.cart
        self.session.modified = True

    def __iter__(self):
        product_ids = self.cart.keys()
        products = Product.objects.filter(id__in=product_ids)

        for product in products:
            self.cart[str(product.id)]['product'] = product

        for item in self.cart.values():
            item['price'] = Decimal(item['price'])
            item['total_price'] = item['price'] * item['quantity']
            yield item

    def __len__(self):
        return sum(item['quantity'] for item in self.cart.values())
```

```
def get_total_price(self):
    return sum(Decimal(item['price']) * item['quantity'] for item in self.cart.values())

def clear(self):
    del self.session[settings.CART_SESSION_ID]
    if self.coupon:
        del self.session['coupon_id']
    self.session.modified = True

@property
def coupon(self):
    if self.coupon_id:
        return Coupon.objects.get(id=self.coupon_id)
    return None

def get_discount(self):
    if self.coupon:
        if self.coupon.code_discount == 'euroale':
            return Decimal(self.coupon.discount_value)
        if self.coupon.code_discount == "prosenttiale":
            return (self.coupon.discount_value / Decimal(100)) * self.get_total_price()

def get_total_price_after_discount(self):
    return self.get_total_price() - self.get_discount()
```