
Web-sovellus Django-sovelluskehysellä



Ammattikorkeakoulun opinnäytetyö

Tietotekniikka

Riihimäki, kevät 2015

Karo Raita

Karo Raita



RIIHIMÄKI
Tietotekniikka
Ohjelmistotekniikka

Tekijä	Karo Raita	Vuosi 2015
Työn nimi	Web-sovellus Django-sovelluskehysellä	

TIIVISTELMÄ

Opinnäytetyön aihe saatiin aikaisemmin toteutetusta asiakastyöstä. Sovellus tuotettiin asiakkaan luomien spesifikaatioiden mukaiseksi. Työn tavoite oli luoda toimiva, palvelimelle asennettava web-sovellus, jolla olisi mahdollista visuaalisesti demonstroida liike-idea mahdollisille sijoittajille.

Sovelluksen luomiseen käytettiin ilmaisia, avoimeen lähdekoodiin perustuvia ohjelmia ja palveluita kuten Linux, MySQL sekä sovelluskehysenä Djangoa, joka perustuu Python-ohjelmointikieleen. Alkuperäinen sovellus toteutettiin virtuaalipalvelimella käyttäen MySQL-tietokantaa ja Djangoa omaa palvelinohjelmaa. Myöhemmin sovellus siirrettiin julkiselle palvelimelle esittelyä varten käyttämällä Nginx-, Gunicorn- ja PostgreSQL-ohjelmistoja aikaisempien sijasta.

Suurimmat haasteet työn toteutuksessa olivat aikaisemmin tuntematon Django sekä Pythonin opettelu ja sovelluksen suunnitteluun liittyvät ennalta tuntemattomat ongelmat. Lisähaasteena projektissa oli asiakaspohjainen lähestymistapa ja sen tuomat muutokset suunnitelmaan ja toteutukseen.

Opinnäytetyö tulee käsittelemään sovelluksen toimintaa, ulkoasua ja siinä ohessa Djangoa toimintaa esimerkkien kautta. Liikkeelle lähdetään alustasta ja sovelluksen vaatimusmäärittelystä, taustalogiikkaan ja ulkoasuun.

Avainsanat Sovelluskehys, tietokannat, verkko-ohjelmointi, WWW-sivustot, Django

Sivut 30 s. + liitteet 9 s.

RIIHIMÄKI

Degree Programme in Information Technology

Author

Karo Raita

Year 2015

Subject of Bachelor's thesis

Web application with Django framework

ABSTRACT

The topic of this thesis stems from a previously conducted customer project, a web application developed according to specifications of the client. The objective was to create a working web application for a web server environment that would allow showcasing the business idea to potential investors.

The web application was built on freely available open source programs, services, and systems such as Linux, MySQL, and Django. Django is a web framework based on the Python programming language. The original web application ran on a virtual server with a MySQL database and the setup of Django's own HTTP server. Later the application was transferred to a public server which also then switched to using Nginx, Gunicorn and PostgreSQL replacing the earlier choices.

The greatest challenges during the project were the previously unfamiliar concepts of the Django framework and the Python language, and in addition unexpected or unknown issues related to application development. An additional challenge was the workflow and planning process required by a client-driven approach.

This thesis covers the functionality and visual design of the web application, and illustrates Django as a framework through examples. First are covered the platform and the requirements of the web application, followed by the functional logic and visual design.

Keywords Framework, database, web programming, World Wide Web, Django

Pages 30 p. + appendices 9 p.

TERMIT JA LYHENTEET

Suomi	English	Selitys
Taulu	Table	Tauluun tallennetaan tietokannassa määritelmän mukaista tietoa eli dataa.
Ylläpito	Administrator / Admin	Pääkäyttäjä, jolla on suuremmat valtuudet tehdä muutoksia.
Malli	Model	Tietokantakuvaus tietuetyypin käsittelystä.
Näkymä	View	Djangon yhteydessä, näkymä määrittelee mitä datalla tehdään sekä mikä sivupohja palautetaan käyttäjälle
Käsittelijä	Controller	Vastaanottaa käyttäjän käskyt jakaen tiedot mallille sekä näkymäl- le.
Sivupohja	Template	Djangon vastine näkymälle, määritellään näytettävä ulkoasu
Määrite	Attribute	Toimintoa tarkentava määrite
Monta yhteen	Many-To-One	Relaatiotietokannan yhteystyyppi, monella taululla voi olla yhteys yhteen
Pudotusvalikko	Drop-down	Valikkotyyppi, joka on esitäytetty lista valinnoista.
Murupolku	Breadcrumbs	Web-sivuissa käytetty hierarkinen navigaatioelementti
HTML	HTML	HyperText Markup Language
Lomake	Form	Sivurakenne, jolla voidaan käsitellä käyttäjän antamaa dataa
Eväste / Keksi	Cookie	Tietoa, jonka palvelin tallentaa käyttäjän päätelaitteelle
Tagi / Merkintä	Tag	HTML-elementtien merkintätyyli, esim. <html>

SISÄLLYS

1	JOHDANTO.....	1
2	ALUSTA	1
2.1	Nginx.....	1
2.2	PostgreSQL	2
2.3	Gunicorn.....	2
3	SOVELLUS.....	3
3.1	Sovelluksen tarkoitus	3
3.2	Vaatusmäärittely	3
3.3	Settings.py – Asetukset	4
3.4	Models.py – Sovelluksen mallit	5
3.4.1	Perusmalli	6
3.4.2	Erityispiirteitä malleissa	7
3.5	Ylläpitopaneeli	8
3.5.1	Ylläpitopaneeli on tietueiden lisäämistä varten.....	9
3.6	Tags.py – Erikoisuuksia varten	11
3.7	Urls.py – Kaksin kaunihimpi	12
3.8	Applikaation urls.py-tiedosto	13
3.9	Forms.py.....	13
3.10	Sivupohjat ja niiden rakenne	15
3.10.1	base.html – Sivupohjien alku ja juuri	15
3.10.2	base_logged_in.html.....	16
4	ULKONÄKÖ JA NÄKYMÄT.....	17
4.1	Tunnuksen luonti ja sisäänkirjautuminen.....	18
4.2	Sisäänkirjautuminen	20
4.3	Sovelluksen sisäinen maailma.....	21
4.4	Henkilötiedot.....	22
4.5	Tarjotut lainat. otetut lainat	24
4.6	Luo uusi laina – lainojen perusta.....	25
4.7	Etsi lainaa	27
5	YHTEENVETO	28
5.1	Lähtötilanne.....	28
5.2	Kohdatut ja ratkotut ongelmat.....	28
5.3	Jatkokehitys.....	29
	LÄHTEET	30

1 JOHDANTO

Tein ensimmäiset sivut internetiin vuosituuhannen taitteessa. Opiskeltuani hetken toisten sivujen lähdekoodia ja tutkailtua muutamia neuvoa antavia sivustoja, sain sivut julkaistua. Sivuston koodi oli tehty tekstieditorilla käyttäen puhdasta HTML-koodia. Opinnäytetyön sovellus on kaukana siitä staattisesta sivusta.

Varsinainen tuotos tehdään asiakastyönä vaatimusmäärittelyiden perusteella esittelemään liikeidea. Määrittelyt ovat pinnallisia ja vaatimustasoltaan sovellus sopi erinomaisesti ensimmäiseksi projektiksi uudella tekniikalla.

Sovelluksen suunnitteluvaiheessa törmättiin moniin erilaisiin tekniikoihin, jotka kaikki soveltuivat sovelluksen tuottamiseen. Yksi suurimmista kriteereistä oli tietenkin ilmaisuus. Toinen huomattava seikka oli tukiverkosto sekä ohjeiden saatavuus. Kolmanneksi, vaatimusmäärittelyn ja asiakastapaamisen jälkeen, tarvittiin helppokäyttöinen tietokanta-rajapinta. Vaihtoehdot rajautuivat kahteen, Ruby on Rails sekä Django. Python oli meille tutumpi vaihtoehto, joten päädyimme Djangoon.

Opinnäytetyön kirjallinen osuus keskittyy selvittämään kuinka aiheena oleva sovellus toimii ja miten se käyttäytyy. Sovelluksen kehitysvaiheessa ymmärrys ei aina seurannut toimivaa funktiota, joten asioiden selvittäminen kirjalliseen muotoon syventää tietouttani Djangon käytöstä sovelluskehiksenä. Tärkeimpänä lähteenä on yksinkertaisesti Djangon oma ohjekirja, joka yhtäläisesti esimerkkien kautta käy läpi jokaisen funktion ja käskyn.

2 ALUSTA

Alkuperäinen sovelluskehitysalusta oli koululta saatu virtuaaliserveri, jossa ajoimme Ubuntu. Kehitysvaiheessa käytimme MySQL-tietokantaa sekä Djangon omaa kehitykseen tarkoitettua http-palvelinta. Sovellus siirrettiin sellaisenaan asiakkaan omalle palvelimelle ja samalla vaihdoimme sovellusalan ohjelmistoja. Ubuntu pysyi kuitenkin valittuna käyttöjärjestelmänä.

2.1 Nginx

Nginx on vapaaseen lähdekoodiin perustuva välitys- ja verkkopalvelin, joka tukee useita verkkoprotokollia. Ohjelmisto on noin 10 vuotta vanha, ensimmäinen virallinen julkaisu tapahtui 2004. Sen väitetään pystyvän käsittelemään yli 10 000 samanaikaista asiakasyhteyttä käyttämällä vain vähän muistia. Se on myös IPv6-yhteensopiva ja tukee monia uudempiä protokollia, esimerkiksi FLV- ja MP4-strimausta. Se on myös erityisen hyvä tasapainottamaan kuormaa. (Nginx 2015.)

Nginx eroaa kilpailijastaan Apachesta pääosin monisäikeisyyden käytössä. Nginx käyttää yleisesti vähemmän säikeitä, mutta pystyy yhdellä säikeellä palvelemaan suuren määrän asiakasyhteyksiä. Se toteuttaa palvelupyynnöt asynkronisesti sekä käyttämällä välimuistia lataamalla sinne valmiiksi sisältöä lähetettäväksi. Sovelluksessa käytän sitä palvelemaan staattisia tiedostoja, kuten kuvia ja CSS-tiedostoja. (Anturis Blog 2015.)

```
server {
    server_name 178.62. . . ;

    access_log off;

    location /static/ {
        alias /home/dev/ls/lendsome/static;
    }

    location / {
        proxy_pass http://127.0.0.1:8001;
        proxy_set_header X-Forwarded-Host $server_name;
        proxy_set_header X-Real-IP $remote_addr;
        add_header P3P 'CP="ALL DSP COR PSAa PSDa OUR NOR ONL UNI COM NAV"';
    }
}
```

Kuva 1. Ote Nginx:n konfigurointi-tiedostosta

Nginx:n konfigurointi on suhteellisen helppoa. Lisäämällä *location /static/ {}*-komento, kerromme Nginx:lle että kaikki kyselyt /static/nimiseen kansioon tulee hakea määritellystä absoluuttisesta tiedostopolusta. Lisäksi määrittelemme sovelluksessa *STATIC_ROOT*-komennon osoittamaan samaan hakemistoon. (Kuva 1.)

2.2 PostgreSQL

PostgreSQL on avoimeen lähdekoodiin perustuva relaatiotietokantajärjestelmä, jonka toiminnan pääpainona on standardinen yhteensopivuus ja laajennettavuus. PostgreSQL on hyvin monipuolinen järjestelmä, joka skaalautuu hyvin isoille sovelluksille. Tämän projektin yhteydessä en juurikaan käytä PostgreSQL:n suomia mahdollisuuksia, Django-sovelluskehitys hoitaa kaiken kommunikaation tietokannan ja sovelluksen välillä. Mitään erityisempää konfiguraatiota ei myöskään tarvittu, koska asiakkaan virtuaalipalvelimella oli tuki Pythonille ja PostgreSQL:lle valmiina. (PostgreSQL 2015.)

2.3 Gunicorn

Gunicorn on Pythonilla toteutettu WSGI (Web Server Gateway Interface) http-palvelin, jonka funktio tässä projektissa on toimia pelkkänä http-palvelimena, ottaen vastaan http-kyselyt. Gunicornin etu on sen suora integraatio Djangoon, automaattisuus sekä keveys. (Wikipedia 2015.)

```
setuid django
setgid django
chdir /home/django

exec gunicorn \
  --name=ls \
  --pythonpath=ls \
  --bind=0.0.0.0:9000 \
  --config /etc/gunicorn.d/gunicorn.py \
  ls.wsgi:application
```

Kuva 2. Ote Gunicorn-sovelluksen konfiguraatitiedostosta. Tiedostopolut annetaan *name* ja *pythonpath* argumenteille.

3 SOVELLUS

3.1 Sovelluksen tarkoitus

Asiakas halusi web-sivuston, jolla käyttäjät voisivat lainata ja antaa lainoja toisille käyttäjille. Käyttäjillä tarkoitetaan joko yrityksiä tai yksityisiä lainanantajia/-ottajia. Yritysten tilanteessa sovelluksen on tarkoitus toimia välittäjänä, joka huolehtii laskujen lähettämisestä, maksujen välittämisestä sekä mahdollisten ongelmatilanteiden, kuten laskun perimisen hoitamisesta.

3.2 Vaatimusmäärittely

Alkuperäinen määrittely antoi lyhyet ohjeet sovelluksen halutusta toiminnasta ja toimintatyylisestä. Perustoiminnallisuuksien lista on lyhyt, mutta työläs:

- käyttäjän tai yrityksen rekisteröityminen, käyttäjän kirjautuminen sekä omien tietojen muuttaminen
- uuden lainan luonti
- lainan ottaminen eli lainan antajan ja lainan ottajan parittaminen keskenään.

Vaatimusmäärittelyyn lisättiin myös mahdollisesti myöhemmin toteutettavia toimintoja, jotka täytyi ottaa huomioon sovellusta suunniteltaessa:

- maksuohjelmien muutokset
- lainalaskuri
- käyttäjäryhmät
- laskutus
- yhteydet pankkiin
- käyttäjän tunnistaminen ja luottotietojen tarkistaminen
- perintä.

Osa näistä määrittelyistä täytyi heti sulkea pois suunnitelmasta niiden laajuuden ja aikarajoitusten vuoksi. Pankkiyhteyksien luominen vaatii omat rajapinnat ja turvallisuusmääritteet. Samat ongelmat toistuvat käyttäjän

henkilötietojen ja perinnän kanssa. Maksuohjelmien muutokset, lainalaskuri, käyttäjäryhmät ja laskutus onnistuttiin lisäämään suunnitelmaan, koska ne vaativat taustatoiminnoista käyttäjätaulun ja erilaisia lainatauluja, joita käytimme muihinkin osioihin sovelluksessa. Näiden toimintojen toteutus oli täten mahdollista toteuttaa annetussa aikataulussa.

Lisäksi vaatimusmäärittelyssä haluttiin kiinnittää huomioita ei-toiminnallisiin vaatimuksiin:

- käytettävyys
- tietoturva
- toimintavarmuus
- ylläpidettävyys ja huollettavuus
- siirrettävyys, laajennettavuus ja uudelleenkäytettävyys
- monikielisyys.

Käytettävyyden osalta haluttiin yksinkertaista ja selkeätä. Tietoturva oli alkuvaiheessa toissijaista, koska käyttäjä tunnistetaan vain sähköposti-osoitteen perusteella, eikä sovellusta alun perin tarkoitettu julkaistavaksi. Toisaalta, Django tarjoaa sisäänrakennettuja suojauksia monille tietoturvaongelmille. (Django Documentation 2015.)

Yksi syy Djangon valintaan oli myös toimintavarmuus. Linux-palvelin yhdistettyä Pythoniin sekä Djangoon ja Djangon kyky toimia monella eri ohjelmistoyhdistelmällä ja tietokannoilla loivat yhdessä pohjan, joka toimintavarmuudellaan että siirrettävyydellään täytti esitetyt vaatimukset.

Ylläpidettävyys ja huollettavuus luetellaan yleisesti myös Djangon eduiksi. Siinä on sisäänrakennettu ylläpitopalvelu, joka on myös hyvin muokattavissa. Tässä työssä emme kuitenkaan muokanneet ylläpitopaneelia. Käytimme ainoastaan Djangon omia, automaattisesti luomia toimintoja-

Viimeinen kohta ei-toiminnallisista vaatimuksista sisälsi monikielisyyden, jolle myös löytyy kattava tuki Djangoista. (Django Documentation 2015)

3.3 Settings.py – Asetukset

Sovelluksessa käytetään pääosin Djangon automaattisesti luomaa *settings.py*-tiedostoa. Muutamia lisäyksiä sinne täytyi tehdä, sekä tietenkin täyttää tarvittavat tiedot sovelluksen alustamiseksi. Kehitysvaiheessa kannattaa pitää päällä *DEBUG = TRUE*, jotta Django kertoo millaisia virheitä tapahtuu. Julkaisussa tämä tietenkin muutetaan arvoon *FALSE*.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': '',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

Yksi tärkeimmistä on tietokanta-asetukset. Sovelluksen tekovaiheessa käytimme MySQL-tietokantaa, mutta tuotannon jälkeen siirryimme käyttämään PostgreSQL-tietokantaohjelmistoa. Muuttamalla `settings.py`-tiedostosta `ENGINE`-määritteen ylläolevaan muotoon, otetaan PostgreSQL käyttöön. Tietenkin täytyy lisätä tietokannan nimi, käyttäjätunnus sekä salasana. Sovelluksessa ei tarvitse erikseen määritellä `HOST` tai `PORT`-attribuuttia.

Toinen tärkeä määrittely on staattisten tiedostojen polku sekä URL-etuliite. Nämä määritellään käyttäen `STATIC_ROOT`- ja `STATIC_URL`-komentoja. `STATIC_ROOT` määrittelee absoluuttisen osoitteen tiedostoille ja `STATIC_URL` määrittelee etuliitteen, jolla tiedostojen sijainnit määritellään HTML-tiedostoissa. `STATICFILES_DIRS`-komentoon määritellään absoluuttinen tiedostopolku, jossa sijaitsevat loput sivuilla käytettävät kuvat ja tiedostot.

3.4 Models.py – Sovelluksen mallit

Sovellus sisältää tässä tuotannon vaiheessa yhteensä yhdeksän eri luokkaa.

- Currency
- FeeOther
- LoanProposal
- Loan
- LoanPayments
- UserManager
- User
- Country
- UserProfile.

Kaksi näistä, `UserManager(BaseUserManager)` ja `User(AbstracBaseUser, PermissionMixin)` ovat erikoisluokkia, jotka täydentävät Django käyttäjänhallintaa sovelluksen mukaiseksi ottamalla käyttöön vain tarvittavat määreet demosovellusta varten sekä ohittamalla osan käyttäjän turvatarkistuksista. Muut luokat liittyvät sovelluksen toimintoihin ja niiden hallintaan käyttäen `models.Model`-alaluokkaa. (Liite 6, Liite 7, Liite 8, Liite 9)

3.4.1 Perusmalli

Djangon mallit määritellään pythonin luokkien avulla. Alla olevassa esimerkissä tehdään luokka *Currency Model*-argumentilla. Määritellään kolme määrettä (*attribute*), joille annetaan saraketyyppi tietokantaa varten. Tässä pääavain määritellään *models.AutoField(primary_key=True)*-lausekkeella. Tämä lauseke ei ole Djangoissa pakollinen, sovelluskehys tuottaa pääavaimen automaattisesti. Tässä tapauksessa halusin määritellä sen itse, jotta nimi olisi ennalta määritelty ja selkeä lukea tiedostosta. *Autofield* ohjaa Djangoa täyttämään sarakkeen tiedot automaattisesti jokaisen uuden tietueen kohdalla juoksevin numeroin. Muissa määreissä on käytetty *models.CharField*-tyyppiä, joka on tekstille tarkoitettu. Näille kannattaa aina määritellä *max_length*-argumentti, jotta sarakkeen koko pysyy sopivana tukien tehokkuutta. Erikoisuutena määreille on annettu selkokielinen vastine komennolla *verbose_name=_(u'Nimi')*. Monikielisyystuki lisätään ympäröimällä haluttu teksti *_u('')*-merkinnällä sekä lisäämällä tiedoston alkuun *import*:

```
from django.utils.translation import ugettext as _
```

Yleisesti mallia suunniteltaessa halutaan luoda monta samantyyppistä objektia. Tällöin niihin viitataan tekstimuodossa käyttäen monikkoa. Django lisää automaattisesti englannikielisen *'s* sanan perään ilmaisemaan monikkoa. Tämä ei kuitenkaan toimi joka sanan kohdalla eikä tietenkään toisilla kielillä. Tällöin määritellään luokan *Meta*-tietoihin määre *verbose_name_plural* ja annetaan sille haluttu tekstiarvo. Monikielisyystuki liitetään jälleen merkinnällä *_u('')*.

```
class Currency(models.Model):
    currencyKey = models.AutoField(primary_key=True)
    currencyName = models.CharField(max_length=127, verbose_name=_(u'Nimi'))
    currencyAbbreviation = models.CharField(max_length=5, verbose_name=_(u'Lyhenne'))

    def __unicode__(self):
        return u'%s' % self.currencyAbbreviation

    class Meta:
        verbose_name_plural =_(u'Rahayksiköt')
```

Jokaiselle luokalle tulee myös määrittää *__unicode__*-metodi, jota kutsumalla palautetaan unicode-standardin mukainen *string*-tieto. Pythonin mukaisesti käytetään *return*-komennossa paikanpitäjä *%s*-tyyppiä ja annetaan haluttu määre. Tässä tapauksessa on käytetty *self.currencyAbbreviation*-termiä, jolloin ylläpitopaneeliin ilmestyy tietueen tunnisteksi sille annettu lyhenne. (Kuva 3.)

Valitse muokattava currency

Kuva 3. Ylläpitopaneelin Currencies-osio

3.4.2 Erityispiirteitä malleissa

Seuraavassa esimerkissä on käytetty hyväksi vakioita luomaan tietueelle tiloja, joita voidaan käyttää esimerkiksi lainan seurannassa.

```
class Loan(models.Model):
    ENDED = 0
    ACTIVE = 1
    DECLINED = 2
    PENDING = 3
    APPROVED = 4
    STATUS = (
        (ENDED, 'Laina maksettu'),
        (ACTIVE, 'Aktiivinen'),
        (DECLINED, 'Evätty'),
        (PENDING, 'Hausa'),
        (APPROVED, 'Hyväksytty')
    )
    loanKey = models.AutoField(primary_key=True)
    proposal = models.ForeignKey(LoanProposal)
    user = models.ForeignKey("User")
    startTime = models.DateTimeField(null=True, blank=True, verbose_name=_('Lainan alkamispäivämäärä'))
    loanStatus = models.SmallIntegerField(choices=STATUS, default=None)

    def __unicode__(self):
        return u'%s' % (str(self.loanKey) + ', ' + str(self.user) + ', ' + str(self.loanStatus))

    class Meta:
        verbose_name_plural = 'Loans'
```

Luokan *Loan* sisään määritellään neljä vakiota ja annetaan niille kokonaislukuarvot. *STATUS*-lista määrittelee näille vakioille selkokieliset vastineet. Nyt voidaan lisätä luokalle *SmallIntegerField*-tyyppinen sarake, jolle annetaan argumentiksi *choices*, joka on Django'n sisäänrakennettu argumentti. Django osaa tällöin luoda ylläpitopaneeliin vaihtoehdotentän saraketta varten täyttäen sen selkokielisellä tekstillä. Tietokannassa on kuitenkin pientä kokonaislukua käyttävä sarake, jolloin sen käyttämä muistimäärä jää pieneksi. Suurten tietokantojen ja hakumäärien yhteydessä tästä on selkeä etu. (Kuva 4.)

Esimerkissä on myös kaksi *Many-to-One*-yhteyttä, *user* sekä *proposal* *models.ForeignKey*(-)funktiolla. Toisessa määritellään ilman lainausmerkkejä luokka-objekti *LoanProposal* ja toisessa lainausmerkeillä *User*. Erona on se, että *User*-luokka määritellään tiedostossa myöhemmin, joten tiedostoa ajettaessa sisään sitä ei ole vielä määritelty.

Luokassa määritellään myös *DateTimeField*, joka saa olla tyhjä tai null-arvoinen. Lisäksi annetaan tietueelle *__unicode__*-funktiolla kuvaava nimike muuttaen Pythonin *str*(-)-funktiolla objektien arvot teksti-tyyppisiksi sekä lisäämällä pilkut ja välit.

Muokkaa loan

Proposal:	3000.00, 10.00%, 6 terms ▼ +
User:	s@s.com ▼ +
Lainan alkamispäivämäärä:	Pvm: 2015-02-13 <small>Tänään</small> Klo: 14:18:42 <small>Nyt</small>
LoanStatus:	Haussa ▼

Kuva 4. Ylläpitopaneeli – Lainanmuokkaus, huomaa *dropdown*-kentät

3.5 Ylläpitopaneeli

Ylläpitopaneeli (*admin-paneeli*) kannattaa Djangossa luoda käyttäen Django omaa, pitkälle automatisoitua toimintoa. Se lukee metadatan sovelluksesta, luomalla toimintoja niiden pohjalta automaattisesti `/admin/`-päätteiselle osoitteelle. Kaikki sivulla on muokattavissa, mutta perusasetuksilla pääsee hyvin pitkälle. Alla oleva komento lisää `settings.py`-tiedostoon `INSTALLED_APPS`-asetuksen alle

```
'django.contrib.admin',
```

ja varmistetaan että `INSTALLED_APPS` sisältää komennot:

```
'django.middleware.common.CommonMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware'
```

Seuraavan kerran käytettäessä `syncdb`-komentoa Django luo ylläpitopaneelin tarvitsemat taulut tietokantaan ja kysyy pääkäyttäjätilin tietoja. Ylläpitopaneelin sisältöön täytyy myös kiinnittää huomiota. Jokainen malli joka halutaan näkymään, täytyy lisätä `admin.py`-tiedoston alkuun:

```
from models import Country, LoanProposal
admin.site.register(Country)
admin.site.register(LoanProposal)
```

Tämä rivi kertoo Djangolle sen, että etsi mallit `Country` ja `LoanProposal` `models.py`-tiedostosta ja luo niiden `models.py`-tiedostossa määriteltyjen tyyppien mukaiset käyttöliittymät. Tuotantovaiheen tässä kohdassa nämä ovat riittävät toiminnot ylläpitopaneeliin, jotta voidaan tarkastella sovelluksen toimintojen toimivuutta sivustolta, ilman että tarvitsee koskea tietokantaan palvelimella. Näin voidaan myös lisätä helposti täytedataa esitelyä varten. (Kuva 5.)

Paneelia voi muokata haluamukseen admin.py-tiedostossa. Jokainen osa-alue on muokattavissa, ulkonäöstä toimintoihin, mutta katsoin sen olevan toissijaista tässä projektissa.



Kuva 5. Ylläpitopaneeli sovelluksessa – huomaa kielierot.

Erot eri osioiden kielessä johtuvat kahdesta seikasta. Ylläpitopaneeli huomaa automaattisesti selaimen käyttämän kielen. Tämä asetus tulee päälle lisäämällä `MIDDLEWARE_CLASSES`-asetukseen

```
'django.middleware.locale.LocaleMiddleware',
```

`SessionMiddleware`-asetuksen jälkeen `settings.py`-tiedostossa. Toiseksi, englanninkieliset tekstit tulevat suoraan `models.py`-tiedoston määrittelemistä malleista. Mallien nimet voidaan kääntää

3.5.1 Ylläpitopaneeli on tietueiden lisäämistä varten



Kuva 6. Ylläpitopaneeli – `LoanProposal`-taulun kaikki tietueet

Sovelluksessa ylläpitopaneelin virka on toimia tietueiden lisäämistä ja tarkistamista varten. Perusasetuksillaan paneeli luo jokaisesta mallista oman kansion, johon voidaan lisätä, poistaa tai muokata tietueita manuaalisesti. Tämä on kuitenkin huomattavasti kätevämpi metodi datan lisäämiseen kuin suorat SQL-käskyt. Tulevaisuutta ajatellen, olisi syytä lisätä sivulle

julkaistavaa tekstiä ja uutisia käsittelevä luokka, jota voidaan hallita paneelin kautta. (Kuva 6.)

Country-mallilla on neljä tietosaraketta, mutta jokaista ei ole pakko täyttää datalla uutta maata luotaessa. (Kuva 7)

Etusivu > Lendsome > Countries > Suomi

Muokkaa country

Muokkaushistoria

Nimi:

Kuvaus:

Continent:

Region:

[✖ Poista](#) [Tallenna ja lisää toinen](#) [Tallenna välillä ja jatka muokkaamista](#) [Tallenna ja poistu](#)

Kuva 7. Ylläpitopaneeli – Suomi-tietueen muokkausikkuna

Käyttäjän henkilötietoja tallentavan *UserProfile*-mallin muokkausnäkymä on huomattavasti mielenkiintoisempi. Lihavoidut nimikkeet indikoivat pakollisia tietoja, joita ilman tietuetta ei voi tallentaa. Datan täytyy myös täyttää tietokannan kriteerit kyseiselle sarakkeelle. Käyttäjäprofiilissa on monesta-yhteen-relaatioita, jotka näkyvät *dropdown*-valikkoina, tässä esimerkissä näkyvät *Maa* ja *Valuutta*.

Etusivu > Lendsome > User profiles > User profile: oppari@hamk.fi

Muokkaa user profile

Muokkaushistoria

User:

Etunimi:

Sukunimi:

Puhelinnumero:

Syntymäaika: Tänään |

IBAN:

Osoite:

Osoite 2:

Maa:

Valuutta:

Y-tunnus:

[✖ Poista](#) [Tallenna ja lisää toinen](#) [Tallenna välillä ja jatka muokkaamista](#) [Tallenna ja poistu](#)

Kuva 8. Ylläpitopaneeli – Käyttäjäprofiilin muokkaus

Lainaehdotuksia on myös mahdollista luoda ylläpitopaneelistä, jopa toisille käyttäjille. Tämä on selkeä tietoturvariski, mutta demovaiheessa tämä ei ole kriittinen asia. Esimerkissä näkyy myös ero nimikkeissä, ne ovat konaan englanniksi ja tulostetaan suoraan mallien nimissä. Tämä on seurausta kehitysvaiheen hajanaisuudesta ja aikataulutuksesta. Toiminnol-

lisuus oli ulkonäköä tärkeämpi seikka. Toiseksi, ulkonäön viilaaminen on työmäärältään kevyempää. Erikoisuutena tässä on *FeeOther*-valinta, jossa voidaan tehdä monta-moneen-relaatio. (Kuva 9.)

Etusivu > Lendsome > Loan Proposals > Lisää loan proposal

Lisää loan proposal

Continuous:	<input type="checkbox"/>
Active:	Luonnos (näkyv vain itselle) ▼
User:	oppiari@hamk.fi ▼ +
TermType:	Vuosittain (1 maksu vuodessa) ▼
TermLength:	120
Currency:	EUR ▼ +
Amount:	150500
Description:	<div>Asuntolaina</div>
FeeStartup:	100
FeeBilling:	5
FeeOther:	Myöhästymismaksu ▲ + <small>Pida "Ctrl"-näppäin (tai Macin "Command") pohjassa valitaksesi useita vaihtoehtoja.</small>
InterestType:	Normaali ▼
Interest:	5

Tallenna ja lisää toinen Tallenna välillä ja jatka muokkaamista Tallenna ja poistu

Kuva 9. Ylläpitopaneeli – Lainaehdotus, huomaa *FeeOther*-valinta

3.6 Tags.py – Erikoisuuksia varten

Otsikon tiedosto on Djangossa tarkoitettu omien sivupohjamerkintöjen luomiseen. Sovelluksessa sitä käytetään vain yhdessä kohtaa, erittelemään aktiivinen sivu rekisteröidyn käyttäjän sivunavigaatiassa.


```

from django import template
import re

register = template.Library()

@register.simple_tag
def active(request, pattern):
    if re.search(pattern, request.path):
        return 'current_subpage_strong'
    return ''

@register.simple_tag
def active_sub(request, pattern):
    if re.search(pattern, request.path):
        return ' class="current_subpage"'
    return ''

```

Tiedostossa sisällytetään pythonista regex-metodi *re*, jolla etsitään kyselyn lähettäneestä URL-osoitteesta tiettyä merkkijonoa. Jos sellainen löytyy, palautetaan CSS-komento. Kummatkin funktiot toteuttavat saman tehtävän eri paikassa palauttaen eri komennon.

```

<li{% active_sub request profile %}>
<a href="{% url "lendsome:profile" %}"><span>Henkilötiedot</span></a></li>

```

Yllä olevassa esimerkissä ``-tagin lisätään CSS-komento *active_sub*-funktioilla. Toiminnon käytettäväksi ottaminen tapahtuu lisäämällä html-tiedostoon `{% load tags %}` tiedoston alkuun.

3.7 Urls.py – Kaksin kaunihimpi

Sovelluksessa käytetään kahta eri *urls.py*-tiedostoa. Toisessa määritellään koko projektin URL-osoitteet, jotka liittyvät käyttäjän login/logout-proseduuriin sekä salasanan vaihtoon ja palauttamiseen. Tiedosto on hakemistopuussa sijoitettu sivuston kansioon `/ls/ls/`. Tiedossa on määritelty applikaation URL-osoitteet sekä ylläpitopaneelin osoite. Erityismaininnan ansaitsee luokka *admin* joka sisällytetään *django.contrib*-moduulista.

```

urlpatterns += patterns('',
    url(r'^$', include('lendsome.urls', namespace="lendsome")),
    url(r'^admin/', include(admin.site.urls)),
)

```

Normaalisti muuttuja *urlpatterns* määriteltäisiin käytämällä `=`-operaattoria, mutta tässä tapauksessa käytetään `+=`-operaattoria, koska *urlpatterns* on määritelty aikaisemmin tiedostossa. Toisella rivillä määritellään applikaation *lendsome.urls.py*-tiedosto *lendsome*-nimiavaruuteen. Tällöin voidaan sovelluksessa viitata kyseiseen nimiavaruuteen jos halutaan luoda URL-osoitteita applikaatiolle. Kolmas rivi määrittelee ylläpitopaneelin osoitteen ja lisää Djangon sisäiset osoitteet `/admin/`-osoitteen alle.


```
class LoanForm(ModelForm):
    class Meta:
        model = Loan

class FindForm(forms.Form):
    search = forms.CharField(max_length=16, label="Kirjoita lainan tunniste")
```

Ylempi luokka *LoanForm* käyttää *ModelForm*-argumenttia, jolloin Django luo lomakkeen automaattisesti valitun mallin mukaisesti ottaen huomioon siellä käytetyt saraketyypit. *Meta*-luokkaan määritellään *model*-muuttuja ja annetaan halutun mallin nimi.

Alempi *FindForm*-luokka on tyyppiä *Form eli* perinteinen lomake, jolle täytyy itse määritellä kentät ja niiden tyypit sekä mahdolliset argumentit. Tässä luokassa on luotu *search*-muuttuja joka on *Charfield*-tyyppiä. Sille on määritelty maksimipituus 16 merkkiä ja annettu tunniste *label*-argumentilla. *Max-length*-argumentti tekee kaksi asiaa. Django luodessa lomaketta, se lisää `<input>`-tagin attribuutiksi `maxlength="16"`, jotta selain ei hyväksy pidempiä kirjainyhdistelmiä. Toiseksi, Django osaa tällöin automaattisesti validoida palautetun datan. (Django Documentation 2015)

```
class LoanProposalForm(ModelForm):
    feeOther = forms.ModelMultipleChoiceField(queryset=FeeOther.objects.all(),
        required=False, widget=forms.CheckboxSelectMultiple)
    class Meta:
        model = LoanProposal
        widgets = {'user': forms.HiddenInput(),
        }
    def save(self, commit=True):
        return super(LoanProposalForm, self).save(commit=commit)
```

LoanProposalForm-luokka on hieman erikoisempi. Sen lisäksi, että siinä määritellään *ModelForm*-tyyppinen lomake, siihen lisätään ylimääräinen kenttä toisesta taulusta *feeOther*-muuttujaan. *ModelMultipleChoiceField*-funktioon annetaan argumentiksi `queryset=FeeOther.objects.all()`, jolloin kenttää haetaan *FeeOther*-taulusta kaikki olemassa olevat objektit. *Required=false* disabloidaan käyttäjän pakottaminen valitsemaan jokin, jolloin voidaan tallentaa *null* tai *blank*. Lisäksi määritellään *widget*, jolloin voidaan tehdä monivalintakysely *querysetin* tuottamista valinnoista. *Meta*-luokkaan lisää *widget*-muuttujaan lomakkeen *user*-kenttään funktio *Hiddeninput()*, jolloin Django ei luo *user*-kenttää automaattisesti. *User* määritellään näkymässä lomaketta käytettäessä. Lopuksi määritellään uudelleen automaattinen *save()*-funktio käyttämällä *super()*-funktioita, jolloin käytetään *save()*-funktioita *ModelForm*-luokasta.

```

class UserCreationForm(ModelForm):

    def clean_username(self):
        email = self.cleaned_data["email"]
        try:
            get_user_model().objects.get(email=email)
        except get_user_model().DoesNotExist:
            return email
        raise forms.ValidationError(self.error_messages['duplicate_username'])

    password1 = forms.CharField(label='Salasana', widget=forms.PasswordInput)
    password2 = forms.CharField(label='Toista salasana', widget=forms.PasswordInput)

    class Meta:
        model = get_user_model()
        fields = ('email', 'password1')

    def clean_password2(self):
        password1 = self.cleaned_data.get("password1")
        password2 = self.cleaned_data.get("password2")
        if password1 and password2 and password1 != password2:
            raise forms.ValidationError("Passwords don't match")
        return password2

    def save(self, commit=True):
        user = super(UserCreationForm, self).save(commit=False)
        user.set_password(self.cleaned_data["password1"])
        if commit:
            user.save()
        return user

```

Yllä oleva lomakeluokka on selkeästi monimutkaisin, tosin sekin käyttää jo aiemmin käytettyjä tapoja lomakkeen tuottamiseen. Lomakkeelle on määritelty *Meta*-luokkaan *model*-muuttujaan funktio *get_user_model()*, jolla referoidaan suoraan Django:n käyttämää *User*-mallia. Tämä on tarpeen, koska sovelluksessa käytetään omaa *User*-mallia. Salasanakentät määritellään myös erikseen, jotta voidaan luoda *clean_password()*-funktio, joka tarkistaa syöttövaiheessa täsmäävätkö salasanat.

3.10 Sivupohjat ja niiden rakenne

Djangon sivupohjien tarkoitus on esittää tieto käyttäjälle miellyttävässä muodossa. Sivupohjissa käytetään yleisesti Django:n omaa syntaksia ja XHTML/HTML-merkintäkieltä sekä muita funktionaalisempia skriptikieliä kuten PHP tai JavaScript. Tarkoituksena on erotella työlogiikka esityslogiikasta. (The Django Book 2009.)

Nykyiset internet-sivut ovat laajoja monimutkaisia kokonaisuuksia joissa on usein tuhansia lähes identtisiä HTML-sivuja joiden asiasisältö muuttuu. Tässä sovelluksessa sivuja ei ole aivan niin paljon, mutta siinä käytetään Django:n tuomaa sivupohjan periytymistä hyväksi. (The Django Book 2009.)

3.10.1 base.html – Sivupohjien alku ja juuri

Sivupohjana *base.html* on uniikki yksilö. Sen perimmäinen tarkoitus on sisällyttää ne ulkoasun elementit, jotka toistuvat sovelluksessa lähes kaikkialla. Tässä sovelluksessa on kuitenkin näennäisesti kaksi erilaista pohjaa, yksi rekisteröityneille käyttäjille sekä toinen ulkopuolisille käyttäjille.

Toimivan käyttäjätunnuksen haltijan sivusto näyttää erilaiselta ja sisältää toimintoja, joihin ulkopuoliset eivät pääse käsiksi. Djangossa sivupohjasyntaksissa käytetään hakasulkeita prosenttimerkeillä etu- ja takaliitteinä erottamaan ne HTML-komennoista.

Ensimmäisillä riveillä otetaan käyttöön käännösmoduuli sekä staattiset tiedostot komennolla `{% load i18n %}` ja `{% load staticfiles %}`. Tämän jälkeen esitellään normaalit HTML-sivujen esitykseen käytettyjä komentoja kuten `<title>` ja `<meta>`. Tässä vaiheessa tuodaan myös CSS-tiedosto käytettäväksi käyttämällä sivupohjasyntaksin komentoa `{% static 'lendsome/css/_style.css' %}`. Static-komento kertoo renderöintikoneistolle mistä CSS-tiedosto löytyy ja liittää sen tilalle tarvittavan tiedostopolun, jolloin käyttäjälle näkyy normaali HTML-syntaksi. (Liite 1.)

Mukana on myös hyvä esimerkki miten Djangossa voidaan linkittää eri HTML-tiedostoja samaan näkymään.:

```
{% include "lendsome/top_toolbar.html" %}
{% include "lendsome/header_menu.html" %}
{% block content %}
{% endblock %}
```

Include-komennolla sisällytetään pätkiä HTML-skriptiä sivupohjaan. Näin voidaan eritellä elementtejä sivustolla. Lisäksi ylläolevassa koodissa esitellään lohko nimeltä *content*. Sovelluksessa tähän sijoitetaan vaihtuva sisältö etusivulla. Viimeisenä *include*ä käyttäen sisällytetään vielä *footer.html* sekä normaalia HTML-kieltä käyttäen etusivun käyttämät skriptit.

Sovelluksen kurentissa versiossa *index.html* on lähinnä paikanpitäjän roolissa johtuen projektin demo-statuksesta ja ainoastaan *index.html* sekä käyttäjän perustoiminnot sisäänkirjautumisesta salasanan palautukseen käyttävät *base.html*-tiedostoa pohjanaan.

3.10.2 base_logged_in.html

Sisäänkirjautuneille käyttäjille näytetään sovelluksessa hieman erilainen näkymä. Heille tarkoitetut toiminnot ja navigaatiot tulevat esiin kirjautumisen jälkeen. Kuitenkin, koska halusimme yhtenäisen ilmeen ja helposti muokattavissa olevan sivuston, käytimme hyväksi Djangon tukemaa sivupohjien periytymistä.

```
{% extends "base.html" %}
{% load i18n %}
{% load staticfiles %}
{% block content %}
{% include "lendsome/small_banner.html" %}
<section id="content-container" class="clearfix">
  <div id="main-wrap" class="clearfix">

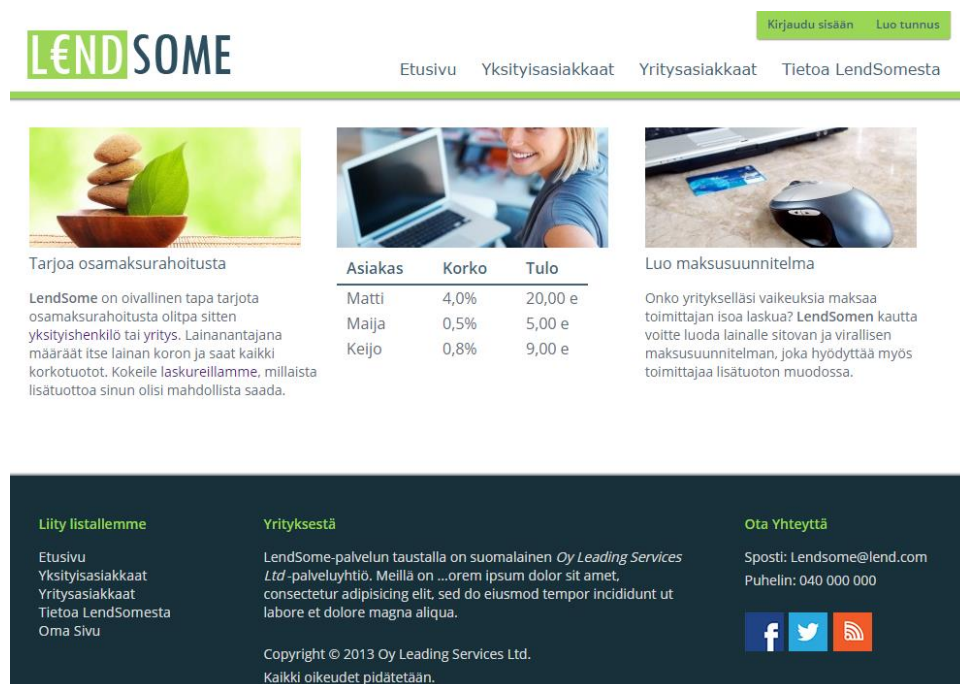
    {% include "lendsome/subnav.html" %}

  </div>
  <!-- END main-wrap -->
</section>
<!-- END content-container -->
{% endblock %}
```

Extends-komennolla otamme käyttöön aiemman *base.html*-sivupohjan ja lisäämme siihen sivunavigaation *subnav.html* sekä *small_banner.html*-pohjan, joka lisää näkymään *breadcrumbs*-tyylisen navigaation.

4 ULKONÄKÖ JA NÄKYMÄT

Alkuperäinen asiakkaan toivoma ulkonäkö oli luotu valmiin WordPress-sommitelman pohjalta. Sovelluksen ensimmäinen versio noudatti sommitelmaa tarkasti, mutta toisessa versiossa CSS-tyylitiedostoa muokattiin yksinkertaisempaan muotoon. Ulkonäkö on mielestäni yksinkertainen ja selkeä, soveltuen hyvin demokäyttöön. Asiakkaan toimittamat sisältömateriaalit olivat lyhyitä ja muodollisia, joten sivustomme kokonaissisältö on myös niukka. (Kuva 10.)



Kuva 10. Index.html eli etusivun näkymä

Kurentissa versiossa suurin osa hyperlinkeistä tällä sivulla ei sisällä toiminnallisuutta. Ainoat toimivat linkit ovat oikean yläkulman *Kirjaudu sisään* ja *Luo tunnus*.

4.1 Tunnuksen luonti ja sisäänkirjautuminen

Klikkaamalla oikean yläkulman *Luo tunnus*-linkkiä päädytään seuraavalaiselle *content*-lohkon sivulle.

Oma sivu

Tunnuksen luonti

Email address:

Salasana:

Toista salasana:

Kuva 11. Tunnuksen luonti. Selkeyden vuoksi kuvissa ei näy koko näkymää.

```
{% extends "base.html" %}
{% load i18n %}
{% load staticfiles %}
{% block content %}
{% include "lendsome/small_banner.html" %}
<section class="content">
  <div id="main-wrap">

    <div class="page_content_right sub-content">
      <h1>Tunnuksen luonti</h1>
      <p>{{ message }}</p>

      <form action="{% url "lendsome:register" %}" method="post">
        {% csrf_token %}
        <input type="hidden" name="next" value="{{ next }}">
        <table class="register">
          {{ form.as_table }}
          <tr><th>
            <input class="button" type="submit" value="Luo tunnus">
          </th></tr>
        </table>
        <br />
      </form>
    </div>
  <!-- END main-wrap -->
</section>
<!-- END content-container -->
{% endblock %}
```

Käyttäjä syöttää käyttäjätunnuksena toimivan sähköpostiosoitteen sekä salasanan kahteen kertaa. Lomake luodaan automaattisesti käyttäen sivupohjassa `{{ form.as_table }}`-muuttujaa, joka luo sen automaattisesti käyttäen `<table>`-tyylistä asettelua. Lomakkeen `action`-attribuutiksi annetaan Django syntaksin mukainen kysely `lendsome`-nimiavaruudesta nimeltään `register` käyttäen metodia POST. `<Form>`-tagien sisään laitetaan myös komento `{% csrf_token %}`. Edellistä komentoa käytetään jokaisessa lomakkeessa sovelluksessa. (Kuva 11.)

CSRF (Cross Site Request Forgery) on tietoturvahyökkäystyyppi, jossa käytetään hyväksy sivuston luottoa rekisteröityneeseen käyttäjään. Käyttämällä `RequestContext`-komentoa ja ylläolevaa `csrf_token`-komentoa, Django automaattisesti huolehtii, ettei tällaista hyökkäystä ole mahdollista suorittaa. (Wikipedia 2015)

```
def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            new_user = form.save()
            return HttpResponseRedirect(reverse("lendsome:my"))
        else:
            form = UserCreationForm()
    return render(request, "lendsome/registration_form.html", {
        'form': form
    })
```

Sivupohjan kysely `register` on määritelty `views.py`-tiedostossa, jossa sijaitsevat kaikki sovelluksen näkymät. Django palauttaa sivun `registration_form.html` johon on lisätty `UserCreationForm()` `forms.py`-tiedostosta mikäli kyselyn metodi EI ole POST. Tapauksessa, jossa se on POST-tyyppinen, lomakkeen tieto validoidaan ja mikäli validointi onnistuu, tallennetaan lomakkeen tiedot tietokantaan. Tallentamisen jälkeen käyttäjä ohjataan automaattisesti `my`-nimisen näkymän sijaintiin. Tarkistukset hoidetaan `forms.py`-tiedostossa. (Liite 2.)

Huomaa, että demo-versiossa tarkistukset ovat hyvin pinnallisia. Sivusto on hyvin altis erilaisille hyökkäyksille eikä tällaisenaan sovellu julkiseen käyttöön.

4.2 Sisäänkirjautuminen

Oma sivu

Kirjaudu sisään

Email:	<input type="text" value="oppiari@hamk.fi"/>
Salasana:	<input type="password" value="....."/>
<input type="submit" value="Kirjaudu sisään"/>	Unohtuiko salasana?

Kuva 12. Yksinkertainen sisäänkirjautuminen

```
<p>{{ message }}</p>
{% if form.errors %}
<p>{% trans "Käyttäjätunnus tai salasana ei kelpaa." %}</p>
{% endif %}
<form action="{% url "lendsome:login" %}" method="post">
{% csrf_token %}
<input type="hidden" name="next" value="{{ next }}">
<table class="logintbl">
<tr>
<td>Username:</td><td><input type="text" name="username"></td>
</tr><tr>
<td>Password:</td><td><input type="password" name="password"></td>
</tr><tr>
<td colspan="2">
<input type="submit" value="Sign in">
<input type="hidden" name="next" value="{{ next|escape }}">
</td></tr>
</table>
</form>
```

Rekisteröidytyään käyttäjä voi kirjautua sisään *Kirjaudu sisään*-linkistä. Rekisteröitymislomakkeesta poiketen, lomake luodaan sivupohjassa manuaalisesti antaen attribuutit ja arvot kentille. Lisäksi listauksessa näkyy kuinka Django palauttama virheilmoitus tuodaan käyttäjälle näkyviin *form.errors*-listan kautta. *{{ message }}*-muuttujaan lisätään näkymästä virheilmoituksia. Näkymässä autentikoidaan käyttäjätunnus Django omalla systeemillä käyttäen *authenticate()*-metodia. Metodi palauttaa boolean arvoja, esimerkiksi *is_active*, joiden avulla tarkistetaan eri virheilmoitukset ja toimitaan niiden mukaisesti if/else-lauserakenteilla. (Kuva 12.)

```
def login(request):
    if request.POST:
        u = authenticate(email=request.POST["username"],
                        password=request.POST["password"])
        if u is not None:
            if u.is_active:
                auth_login(request, u)
                return HttpResponseRedirect(reverse("lendsome:my"))
            else:
                message = _("Tunnuksesi on deaktivoitu")
        else:
            message = _("Väärä käyttäjätunnus tai salasana")
        return render(request, "lendsome/login_form.html",
                    {"message": message})
    else:
        if request.user.is_authenticated():
            return HttpResponseRedirect(reverse("lendsome:my"))
        return render(request, "lendsome/login_form.html")
```

4.3 Sovelluksen sisäinen maailma

Sovelluksessa varsinaiset toiminnot ovat ainoastaan näkyvillä rekisteröityneille käyttäjille. Kirjaututtuaan sisään käyttäjälle avautuu näkymä, jossa uusina komponentteina ovat *content*-lohkon sisälle sijoittuva *subnav*-navigaatio ja *breadcrumbs*.

The screenshot shows the LendSome user interface. At the top, there is a navigation bar with the LendSome logo and links for 'Etusivu', 'Yksityisasiakkaat', 'Yrityisasiakkaat', 'Tietoa LendSomesta', and 'Omat sivut'. The user is logged in as 'Hei, oppari@hamk.fi' and can 'Kirjautu ulos'. The main content area is titled 'Oma sivu' and features a sidebar with navigation options: 'Yhteenveto', 'Henkilötiedot', 'Viestit', 'Tarjotut lainat', 'Otetut lainat', 'Lainaehdotukset', 'Luo uusi lainaehdotus', 'Etsi lainaa', and 'Ohjeita'. The 'Yhteenveto' section contains a pie chart showing the distribution of loans by date: 10.11.2014 (14.7%), 05.02.2014 (80.9%), and 09.03.2015 (4.4%). Below the chart are two tables: 'Omat avoimet lainat' and 'Annetut lainat'. The 'Omat avoimet lainat' table lists three active loans with their respective amounts and statuses. The 'Annetut lainat' table lists one loan that has been repaid.

Myönnetty / erien määrä	Summa	Tila
10.11.2014 / (24)	10000.00 €	Aktiivinen
05.02.2014 / (6)	3000.00 €	Haussa
09.03.2015 / (36)	55000.00 €	Aktiivinen

Myönnetty / erien määrä	Summa	Tila
08.03.2015 (36 kk)	100000.00 €	

Kuva 13. Sisäänkirjautumisenäkymä – Käyttäjällä on lainoja lainattavaksi sekä lainattu.

Piirakkakaavio on toteutettu sivupohjassa Javascriptillä ja Django syntaksilla yhteisesti. Tietueita tuodaan Django Javasciptin käytettäväksi. Javascriptissä käytetään Googlen APIa kaavion luomisessa *corechart*-moduulilla ja lisätään sarakkeet Tilalle ja Määrälle ja for-loopilla lisätään rivejä tietokannasta. (Liite 3)

Lainalistaukset on toteutettu pelkästään sivupohjasyntaksilla for-looppien sisällä. Data haetaan suoraan näkymässä luotujen muuttujien arvoista lainakohtaisesti sarakkeiden nimillä. (Liite 4)

Jotta data saadaan sivupohjan ja Javascriptin käytettäväksi, haetaan se tietokannasta näkymässä. Näkymään on myös määritelty *@login_required*-komento ennen näkymän määrittelyä, joka estää sivulle siirtymisen ilman sisäänkirjautumista. Määritellään kaksi muuttujaa, *loansTaken* ja *loansGiven*, filteröidään lainan statuksen mukaan *Q()*-kyselyillä. *Q()*-komentoa käyttämällä voidaan määritellä operaattoreita ja eritellä tarkalleen, mitkä tietueet halutaan etsiä tietokannasta. *LoansTaken*in määreinä ovat kurentti käyttäjätunnus ja *loanStatus = 1*, jotka liittyvät käyttäjän tekemiin lainaehdotuksiin. *LoansGiven* haetaan suoraan lainoista joilla on *loanStatus* arvolla 1, jotka liittyvät käyttäjän *proposal* tietoon.

```
@login_required
def mypage(request):
    loansTaken = Loan.objects
        .filter(Q(user=request.user, LoanStatus=1) | Q(user=request.user, LoanStatus=3))
        .select_related('proposal')

    loansGiven = Loan.objects
        .filter(LoanStatus=1).prefetch_related('proposal')
        .filter(proposal__user_id=request.user)

    return render(request, "lendsome/mypage.html",
        { 'loansTaken': loansTaken, 'loansGiven': loansGiven })
```

4.4 Henkilötiedot

Sovelluksessa uuden käyttäjän profiilitiedot luodaan vasta ensimmäisellä käynnillä *Henkilötiedot*-sivulla tietokantaan. Tämän jälkeen käyttäjä voi muokata omia tietojaan. Varsinaisia tarkistuksia ei sovelluksen demoversiossa tehdä eikä tietoja käytetä missään toiminnoissa. Lomake on toteutettu suoraan Django *ModelForm*-luokan mukaan. Ainoastaan käyttäjätunnus eli sähköposti on piilotettu kenttä jota käyttäjä ei voi muokata. Kenttien lisääminen ja poistaminen tapahtuu muuttamalla tietokannasta informaatiota, mutta sovelluksen tässä vaiheessa henkilötiedot ovat toissijaisia.

Henkilötiedot

Etunimi:	<input type="text" value="Oppari"/>
Sukunimi:	<input type="text" value="Täysvitone"/>
Puhelinnumero:	<input type="text" value="020102015"/>
Syntymäaika:	<input type="text" value="1.9.2010"/>
IBAN:	<input type="text" value="FI02 0123 4567 8901"/>
Osoite:	<input type="text" value="Kaartokatu 2 11100"/>
Osoite 2:	<input type="text"/>
Maa:	<input type="text" value="Suomi"/>
Valuutta:	<input type="text" value="EUR"/>
Y-tunnus:	<input type="text" value="XYZ"/>
<input type="button" value="Muuta tietojasi"/>	

Kuva 14. Henkilötiedot – suora lista taulusta ilman käyttäjätunnusta

Koodina henkilötietojen ulosotto on hyvin yksinkertaista. *Forms.py*-tiedostossa määritellään *ModelForm*-tyyppinen luokka ja piilotetaan *user*-rivi. (Kuva 14)

```
class ProfileForm(ModelForm):
    class Meta:
        model = UserProfile
        widgets = {'user': forms.HiddenInput()}
```

Views.py-näkömätiedostossa määritellään miten toimitaan tietojen päivityksen yhteydessä. Ensiksi yritetään avata käyttäjän profiili, mikäli sellaista ei ole, luodaan tyhjä profiili. Jos lomake on validi, tallennetaan tiedot tietokantaan.

```

@login_required
def profile(request):
    user = request.user
    try:
        profile = UserProfile.objects.get(user=user)
    except UserProfile.DoesNotExist:
        profile = UserProfile.objects.create(user=user)
    form = ProfileForm(request.POST or None, instance=profile)
    if form.is_valid():
        if profile.user == request.user:
            profile = form.save(commit=False)
            profile.save()
        return HttpResponseRedirect(reverse('lendsome:my'))
    return render(request, 'lendsome/profile.html', {'form' : form})

```

4.5 Tarjotut lainat. otetut lainat

Kummatkin, tarjotut sekä otetut lainat-osiot ovat hyvin samanlaisia. Tarkoituksena oli erotella lainat jotka käyttäjä ottaa ja jotka käyttäjä antaa sekä lisäksi erotella missä vaiheessa kukin laina on *loanStatus*-tiedon mukaisesti. Sivut ovat ulkoasultaan identtiset, ainoastaan tekstit ja data vaihtuvat. Liukuvat listat tulevat esiin klikkaamalla otsikkoa CSS:n avulla toteutettuna. (Kuva 15.)

Tunniste	Laina-aika	Lainamäärä	%
#6	36 erää	100000,00 €	10,00
#5	12 erää	5000,00 €	8,50

Kuva 15. Tarjotut lainat – lista lainaluonnoksista

Otetut lainat

+ Haussa olevat lainat

- Aktiiviset lainat

Tunniste	Laina-aika	Lainamäärä	%
#6	24 erää	10000,00	6,50
#8	36 erää	55000,00	7,00

+ Päättyneet lainat

+ Hylätyt lainahakemukset

Kuva 16. Otetut lainat – lista aktiivista lainoista

Sivupohjat ovat yksinkertaisia for-silmukoita, varsinainen datankeruu tahtuu näkymän logiikassa. Jokainen erityyppinen laina haetaan omilla tietokantakyselyillään, filteröidään ja järjestellään haluttuun muotoon. Edellisten toimintojen jälkeen ne esitetään taulukkomuodossa sivupohjassa. Listaus näyttää hyvin pitkältä ja monimutkaiselta, mutta sama kaava toistuu niissä jokaisessa. (Liite 5.) (Kuva 16.)

4.6 Luo uusi laina – lainojen perusta

Sivuston tarkoitus on luoda linkki lainanantajien ja -ottajien välille sekä toimia operaattorina verkkokauppiiaan ja kuluttajan välissä toimimalla osamaksusopimuksen hoitajana. Tätä varten täytyy ensin luoda lainaehdotus, joka on uudelleenkäytettävä. Käyttäjä määrittelee lainan summan, lainaajan, takaisinmaksusopimuksen keston, korkoprosentin sekä mahdolliset lisämaksut. Sivulla on yksinkertainen liukupalkki koron merkitsemiseen sekä kokonaiskorkolaskuri. Ajan puutteen vuoksi osa sivun laskelmista ovat staattisia. Lomake muodostetaan jälleen kerran käyttämällä *ModelForm*-tyyppistä lomaketta, piilottaen käyttäjäkentän. Ainoa erikoisuus on lisämaksujen tuominen lomakkeeseen komennolla:

```
class LoanProposalForm(ModelForm):
    feeOther = forms.ModelMultipleChoiceField(queryset=FeeOther.objects.all(),
                                              required=False, widget=forms.CheckboxSelectMultiple)
```

Lainan perustiedot

Laina määrä:

Laina-aika (kk):

Takaisinmaksu:

Lainan tila:

Korko: 5.5%

Kokonaiskorko:

Perusmaksut

Aloituskorko: Laskutuslisä:

Lisämaksut (laskutetaan lainan ottajalta)

Myöhästymismaksu

Yhteenveto

Lainan tuotto lainan antajalle: 70 €

Lainan todellinen vuosikorko: 10 %

Tallenna uusi laina

Kuva 17. Lainaehdotuksen luominen

Näkymässä on muutama muista poikkeava komento. Syy tähän on lainaehdotuksen tietueen relaatiot (*feeOther*) muiden taulujen kanssa, tällöin tallentamiseen käytetään *save_m2m()*-metodia. Toiseksi, data tallennetaan käyttäjään sidottuna, juoksevin numeroin. Tällöin voidaan hakea lainaa sen *loanProposalKeyn* avulla. Jos luotu lainaehdotus tallennetaan, siirrytään *Lainaehdotukset*-sivulle, jossa näkyy VAIN uusin lainaehdotus. Tämä hoidetaan filteröimällä datasta ainoastaan uusin lisäämällä *order_by()*-komennon perään *[:1]*. *Message*-attribuutille annetaan arvo *True*, jotta näkymään ei tule muita sillä sivulla käytettäviä toimintoja. (Kuva 17.)

```

@login_required
def new_loan_proposal(request):
    user = request.user
    form = LoanProposalForm(request.POST or None, initial={'user': user})
    if request.method == 'POST':
        if form.is_valid():
            proposal = form.save(commit=False)
            proposal.save()
            form.save_m2m()
            data = LoanProposal.objects.all().filter(user=user)
            .order_by('-loanProposalKey')[:1]
            return render(request, "lendsome/proposals.html",
                          {'data': data, 'message': True})
        else:
            return render(request, 'lendsome/new_loan_proposal.html',
                          {'form' : form})
    else:
        return render(request, 'lendsome/new_loan_proposal.html',
                      {'form' : form})

```

4.7 Etsi lainaa

Lainanottaja voi hakea lainaa uniikin numeron mukaan tietokannasta. Sovelluksessa käytetään lainan hakemiseen *loanProposalKey*-numeroa, joka on riittävä demoversion vaatimukseen. Mikäli lainan numerolla ei löydy lainaa, printataan virheilmoitus. Lainan löytyessä, tulostetaan lainan tiedot ja ”Hae tätä lainaa” -painike. Painamalla painiketta, lainan status muuttuu ja siihen liitetään hakijan käyttäjätunnus, jolloin lainanantajalle ilmestyy *Yhteenveto*-sivulle alareunaan ilmoitus haetusta lainasta. Lainanantaja voi tällöin hyväksyä tai hylätä lainahaun. (Kuva 18.)

Etsi laina

Voit etsiä itsellesi sopivia lainoja tunnisteen tai hakukriteerien perusteella. Teksiä, tekstiä...

Kirjoita lainan tunniste:

Etsi lainaa

Hakuasi vastaavat lainat:

Lainan tunniste on 3. Voit käyttää tunnistetta jakaessasi lainaa yksityisesti.

Lainamäärä: 3000,00 €

Lainaerien määrä: 6 erää

Laskutustyyppi: Kuukausittain (12 maksua vuodessa)

Korkoprosentti: 10,00

Lainan tila: Luonnos (näkyv vain itselle)

Avausmaksu: 10,0000 €

Laskutuslisä: 5,0000 €

Lainan todellinen vuosikorko:

Lainan kokonaiskustannukset lainan ottajalle: €

Lainan tuotto lainan antajalle: €

Hae tätä lainaa

Kuva 18. Lainahaku – tunnisteella 3 on löytynyt haettavissa oleva laina

Näkymässä on jälleen hakulomake, joka muodostetaan tällä kertaa *forms.Form*-tyyppisenä ja siinä on vain yksi kenttä, *CharField*. Hakuksen tieto palautetaan POST-metodilla sivulle ja sen tietoa verrataan *LoanProposal*-mallin tietueiden *pk* eli *primary key*-sarakeeseen erotellen kaikki käyttäjän omat lainat. Painettaessa ”Hae tätä lainaa”, suoritetaan GET-metodi, jolloin lisätään *Loan*-tauluun tieto, johon viitataan *proposalId*:llä sekä annetaan *loanStatus* arvo 3. Sivupohjassa ei ole mitään erikoista muihin sivuihin verrattaessa.

```
@login_required
def find_loans(request):
    user = request.user
    form = FindForm(request.POST or None)
    error_message = ''
    if request.method == 'POST':
        if form.is_valid():
            search = form.cleaned_data['search']
            try:
                data = LoanProposal.objects.filter(pk=search).exclude(user=user)[0]
            except IndexError:
                data = ''
                error_message = "Ei löytynyt sopivaa lainaa"
            form = FindForm()
            return render(request, "lendsome/find_loans.html",
                { 'data': data, 'search': form, 'error_message': error_message })
        elif request.method == 'GET':
            if request.GET.get('proposalId'):
                loan = Loan(proposal_id=request.GET.get('proposalId')
                    , user_id=user.id, loanStatus=3)
                loan.save()
                return HttpResponseRedirect(reverse("lendsome:my_loans"))
            else:
                form = FindForm()
                return render(request, 'lendsome/find_loans.html', {'search': form})
```

5 YHTEENVETO

5.1 Lähtötilanne

Sovelluksen kehitys tehtiin pääasiassa ryhmätyönä, jossa oma osuuteni kokonaisuudessaan oli Django:n kanssa työskentelyä. Näkymien logiikka, mallien suunnittelu ja sivupohjien koodi on lähes kokonaisuudessaan omaa työtä. Apua ryhmältä sain ulkoasuun ja sovelluksen alun työstämiseen sekä suunnitteluun. Ensimmäisessä työstövaiheessa iso osa ajasta meni Django:n tutoriaalien tekemiseen sekä ohjekirjaan tutustumiseen. Jokainen työvaihe oli minulle täysin uutta. Tämä huomioon ottaen sovelluksen lopputulos on hyvä.

5.2 Kohdatut ja ratkotut ongelmat

Yksi suurimmista alun hidastajista oli puutteellinen tietokantasuunnitelma. Saimme asiakkaalta valmiin suunnitelman, jonka toteuttaminen ja logiikka olivat monimutkaista. Projektin toisessa työstövaiheessa suunnitimme tietokannan kokonaisuudessaan uusiksi. Tämä selkeytti logiikkaa huomattavasti ja opetti perusteita tietokannan suunnittelusta. Django:n kanssa suurimmaksi ongelmaksi muodostui yksinkertaisesti Django:n sisäisen logii-

kan hahmottaminen, joka alkuun tuntui monimutkaiselta. Sivupohjien ja näkymien suhteet sekä if-lauserakenteet täytyi miettiä hartaasti ja huolella. Onneksi kehitysvaiheessa koodin muokkaaminen ja käsin testaus oli nopeaa.

5.3 Jatkokehitys

Tämän työn kirjoittamisen aikana kävin läpi jokaisen tärkeämmän tiedoston ja selvitin itselleni tarkalleen mitä sovellus tekee. Vaikka työ on ensimmäinen laajempi web-sovellus ja ainoa Djangolla tekemäni, on työ hyvin tehty. Laajamittaiseen julkiseen käyttöön se ei kuitenkaan sovellu tällaisenaan. Työ kokonaisuudessaan antoi hyvää esimerkkiä millaista sovelluskehitys on ja kuinka työlästä tyhjistä aloittaminen on. Loppujen lopuksi olen tyytyväinen tulokseen, vaikka täydellinen se ei olekaan.

LÄHTEET

- Anturis. 2015. Anturis Blog. Viitattu 15.3.2015.
<https://anturis.com/blog/nginx-vs-apache/>
- The Django Book n.d. The Django Admin Site Viitattu 5.2.2015.
<http://www.djangobook.com/en/2.0/chapter20.html>
- The Django Book. 2009. Templates. Viitattu 10.3.2015.
<http://www.djangobook.com/en/2.0/chapter04.html>
- Django Documentation. 2015. Building a form in Django. Viitattu 24.2.2015
<https://docs.djangoproject.com/en/1.7/topics/forms/#building-a-form-in-django>
- Django Documentation. 2015. Security. Viitattu 23.3.2015.
<https://docs.djangoproject.com/en/1.7/topics/security/>
- Django Documentation. 2015. Translation. Viitattu 23.3.2015.
<https://docs.djangoproject.com/en/1.7/topics/i18n/translation/>
- Django Documentation. 2015. Translation. Viitattu 24.2.2015.
<https://docs.djangoproject.com/en/1.7/topics/i18n/translation/#miscellaneous>
- PostgreSQL. 2015. PostgreSQL. Viitattu 16.3.2015.
<http://www.postgresql.org/>
- Nginx. 2015. Nginx. Viitattu 23.3.2015. <http://wiki.nginx.org/>
- Wikipedia. 2015. Unicorn. Viitattu 18.3.2015.
http://en.wikipedia.org/wiki/Unicorn_%28HTTP_server%29
- Wikipedia. 2015. Cross-site request forgery. Viitattu 10.3.2015.
http://en.wikipedia.org/wiki/Cross-site_request_forgery

BASE.HTML

```
{% load i18n %}
{% load staticfiles %}

<!DOCTYPE HTML>
<html dir="ltr" lang="en-US">

  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, minimum-scale=1.0, maximum-scale=1.0" />
    <title>LendSome</title>
    <link rel="stylesheet" type="text/css" href="{% static 'lendsome/css/_style.css' %}" />
    <link href="http://fonts.googleapis.com/css?family=Open+Sans:400,600" rel="stylesheet" type="text/css">
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script type="text/javascript" src="{% static 'lendsome/js/jquery-1.7.2.min.js' %}"></script>
  </head>
  <body>
    <div class="wrapper">

      {% include "lendsome/top_toolbar.html" %}
      {% include "lendsome/header_menu.html" %}
      {% block content %}
      {% endblock %}
    </div>
    <div class="push"></div>
  </div>

  {% include "lendsome/footer.html" %}

  <script type="text/javascript" src="{% static 'lendsome/js/custom-main.js' %}"></script>
  <script type="text/javascript" src="{% static 'lendsome/js/jquery.prettyPhoto.js' %}"></script>
  <script type="text/javascript" src="{% static 'lendsome/js/slides.min.jquery.js' %}"></script>
  <script type="text/javascript" src="{% static 'lendsome/js/jquery.cycle.all.min.js' %}"></script>
  <script type="text/javascript" src="{% static 'lendsome/js/jquery.easing.1.3.js' %}"></script>
  <script type="text/javascript">
    jQuery(document).ready(function(){
      jQuery('#slides').slides({
        preload: false,
        //preloadImage: 'preloader url here',
        autoHeight: true,
        effect: 'slide',
        slideSpeed: 500,
        play: 0,
        randomize: false,
        hoverPause: false,
      });
    });
  </script>

</body>
</html>
```

USERCREATIONFORM FORMS.PY

```
class UserCreationForm(ModelForm):

    def clean_username(self):
        email = self.cleaned_data["email"]
        try:
            get_user_model().objects.get(email=email)
        except get_user_model().DoesNotExist:
            return email
        raise forms.ValidationError(self.error_messages['duplicate_username'])

    password1 = forms.CharField(label='Salasana', widget=forms.PasswordInput)
    password2 = forms.CharField(label='Toista salasana', widget=forms.PasswordInput)

    class Meta:
        model = get_user_model()
        fields = ('email', 'password1')

    def clean_password2(self):
        password1 = self.cleaned_data.get("password1")
        password2 = self.cleaned_data.get("password2")
        if password1 and password2 and password1 != password2:
            raise forms.ValidationError("Passwords don't match")
        return password2

    def save(self, commit=True):
        user = super(UserCreationForm, self).save(commit=False)
        user.set_password(self.cleaned_data["password1"])
        if commit:
            user.save()
        return user
```

MYPAGE.HTML – PIIRAKKAKAAVIO

```
{% extends "base.html" %}
{% load i18n %}
{% load staticfiles %}
{% block content %}
{% include "lendsome/small_banner.html" %}
<section class="content">
  <div id="main-wrap">
    {% include "lendsome/subnav.html" %}
    <!--<div class="page_content_right sub-content">-->
    <h1>Yhteenveto</h1>

    <script type="text/javascript">
      google.load('visualization', '1.0', {'packages':['corechart']});
      google.setOnLoadCallback(drawTakenChart);

      function drawTakenChart() {
        var data = new google.visualization.DataTable();
        data.addColumn('string', 'Tila');
        data.addColumn('number', 'Määrä');

        {% for loan in loansTaken %}
          data.addRow(['{{ loan.startTime|date:"d.m.Y" }}', {{ loan.proposal.amount }}]);
        {% endfor %}

        var options = {
          width: 500,
          height: 300,
          colors: ['#335f74', 'rgb(93, 111, 117)', 'rgb(23, 48, 57)']
        }

        var chart = new google.visualization.PieChart(document.getElementById('chart_div'));
        chart.draw(data, options);
      }
    </script>
```

MYPAGE.HTML - LAINALISTAUKSET

```
<div id="chart_div" style="margin: 0 auto;width: 500px"></div>
<h3>Omat avoimet lainat</h3>

<table style="width:100%; margin-top:25px;">
  <tr style="font-weight:bold">
    <th style="width: 45%">Myönnetty / erien määrä</th>
    <th style="width: 40%">Summa</th>
    <th style="width: 15%">Tila</th>
  </tr>

  {% for loan in loansTaken %}
    <tr>
      <td>{{ loan.startTime|date:"d.m.Y" }} / ({{ loan.proposal.termLength }})</td>
      <td>{{ loan.proposal.amount }} €</td>
      <td>{{ loan.get_loanStatus_display }}</td>
    </tr>
  {% endfor %}
</table>
<br>
<h3>Annetut lainat</h3>
<table style="width:100%; margin-top:25px;">
  <tr style="font-weight:bold">
    <th style="width: 45%">Myönnetty / erien määrä</th>
    <th style="width: 40%">Summa</th>
    <th style="width: 15%">Tila</th>
  </tr>
  {% for loan in loansGiven %}
    <tr>
      <td>{{ loan.startTime|date:"d.m.Y" }} ({{ loan.proposal.termLength }} kk)</td><td>{{
loan.proposal.amount }} €</td><td>{% ifequal loan.proposal.active 1 %}Avoin{%
endifequal %}</td>
    </tr>
  {% endfor %}
</table>
</div>
</div>
<!-- END main-wrap -->
</section>
<!-- END content-container -->
{% endblock %}
```

TARJOTUT JA OTETUT LAINA NÄKYMÄ

```
@login_required
def offered_loans(request):
    loansOpen = LoanProposal.objects.filter(Q(user=request.user, active=1) | Q(user=request.user, active=2))
    .order_by('-loanProposalKey')
    loansBilling = Loan.objects.filter(proposal__user=request.user, LoanStatus=1)
    loansDraft = LoanProposal.objects.filter(user=request.user, active=0).order_by('-loanProposalKey')
    loansPending = Loan.objects.filter(proposal__user=request.user, LoanStatus=3)
    user = request.user
    if request.method == 'GET':
        if request.GET.get('loankey') and request.GET.get('decline'):
            loan = Loan.objects.filter(LoanKey=request.GET.get('loankey')).update(LoanStatus=2)
            return HttpResponseRedirect(reverse("lendsome:offered_loans"))
        elif request.GET.get('loankey'):
            loan = Loan.objects.filter(LoanKey=request.GET.get('loankey')).update(LoanStatus=1)
            return HttpResponseRedirect(reverse("lendsome:offered_loans"))
    return render(request, 'lendsome/offered_loans.html',
        {'loansBilling': loansBilling,
         'loansOpen': loansOpen,
         'loansDraft': loansDraft,
         'loansPending': loansPending})

@login_required
def my_loans(request):
    loansActive = Loan.objects.filter(user=request.user).select_related('proposal').filter(LoanStatus=1)
    loansEnded = Loan.objects.filter(user=request.user).select_related('proposal').filter(LoanStatus=0)
    loansDeclined = Loan.objects.filter(user=request.user).select_related('proposal').filter(LoanStatus=2)
    loansPending = Loan.objects.filter(user=request.user).select_related('proposal').filter(LoanStatus=3)
    return render(request, 'lendsome/my_loans.html',
        { 'loansPending': loansPending, 'loansActive': loansActive,
          'loansEnded': loansEnded, 'loansDeclined': loansDeclined })
```


MODELS.PY – OSA 1

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

from django.db import models
from django.utils import timezone
from django.conf import settings
from django.contrib.auth.models import AbstractBaseUser, BaseUserManager
from django.contrib.auth.models import AbstractUser, UserManager, PermissionsMixin
from django.utils.translation import ugettext as _

class Currency(models.Model):
    currencyKey = models.AutoField(primary_key=True)
    currencyName = models.CharField(max_length=127, verbose_name=_(u'Nimi'))
    currencyAbbreviation = models.CharField(max_length=5, verbose_name=_(u'Lyhenne'))

    def __unicode__(self):
        return u'%s' % self.currencyAbbreviation

    class Meta:
        verbose_name_plural = 'Currencies'

class FeeOther(models.Model):
    feeOtherKey = models.AutoField(primary_key=True)
    description = models.CharField(max_length=300)
    amount = models.DecimalField(max_digits=10, decimal_places=4)

    def __unicode__(self):
        return u'%s' % self.description

    class Meta:
        verbose_name_plural = 'Other Fees'

class LoanProposal(models.Model):
    LUMP = 0
    ANNUAL = 1
    BIENNIAL = 2
    QUARTERLY = 3
    BIMONTHLY = 4
    MONTHLY = 5
    BIWEEKLY = 6
    WEEKLY = 7
    TERM_TYPES = (
        (LUMP, 'Kertamaksu'),
        (ANNUAL, 'Vuosittain (1 maksu vuodessa)'),
        (BIENNIAL, 'Puolivuosittain (2 maksua vuodessa)'),
        (QUARTERLY, 'Kvartaaleittain (4 maksua vuodessa)'),
        (BIMONTHLY, 'Joka toinen kuukausi (6 maksua vuodessa)'),
        (MONTHLY, 'Kuukausittain (12 maksua vuodessa)'),
        (BIWEEKLY, 'Joka toinen viikko (26 maksua vuodessa)'),
        (WEEKLY, 'Viikoittain (52 maksua vuodessa)'),
    )

    DRAFT = 0
    PUBLIC = 1
    HIDDEN = 2
    ACTIVE_STATE = (
        (DRAFT, 'Luonnos (näkyvää vain itselle)'),
        (PUBLIC, 'Julkinen (kaikki näkevät)'),
        (HIDDEN, 'Piilotettu (näkyvää linkillä)')
    )
)
```

MODELS.PY – OSA 2

```
NORMAL = 0
INTEREST_TYPE = (
    (NORMAL, 'Normaali'),
)
loanProposalKey = models.AutoField(primary_key=True)
continuous = models.SmallIntegerField(null=True, blank=True)
active = models.SmallIntegerField(choices=ACTIVE_STATE, default=DRAFT)
user = models.ForeignKey("User")
termType = models.SmallIntegerField(choices=TERM_TYPES, default=MONTHLY)
termLength = models.PositiveIntegerField()
currency = models.ForeignKey(Currency)
amount = models.DecimalField(max_digits=10, decimal_places=2)
description = models.TextField(max_length=65535, blank=True)
feeStartup = models.DecimalField(max_digits=10, decimal_places=4)
feeBilling = models.DecimalField(max_digits=10, decimal_places=4)
feeOther = models.ManyToManyField(FeeOther)
interestType = models.SmallIntegerField(choices=INTEREST_TYPE, default=NORMAL)
interest = models.DecimalField(max_digits=10, decimal_places=2, null=True, blank=True)

def __unicode__(self):
    return u'%s' % (str(self.amount) + ', '
        + str(self.interest) + '%, ' + str(self.termLength) + ' terms')

class Meta:
    verbose_name_plural = 'Loan Proposals'

class Loan(models.Model):
    ENDED = 0
    ACTIVE = 1
    DECLINED = 2
    PENDING = 3
    APPROVED = 4
    STATUS = (
        (ENDED, 'Laina maksettu'),
        (ACTIVE, 'Aktiivinen'),
        (DECLINED, 'Evätty'),
        (PENDING, 'Haussa'),
        (APPROVED, 'Hyväksytty')
    )
    loanKey = models.AutoField(primary_key=True)
    proposal = models.ForeignKey(LoanProposal)
    user = models.ForeignKey("User")
    startTime = models.DateTimeField(null=True, blank=True, verbose_name='Lainan alkamispäivämäärä')
    loanStatus = models.SmallIntegerField(choices=STATUS, default=None)

    def __unicode__(self):
        return u'%s' % (str(self.loanKey) + ', ' + str(self.user) + ', ' + str(self.loanStatus))

    class Meta:
        verbose_name_plural = 'Loans'

class LoanPayments(models.Model):
    loanPaymentsKey = models.AutoField(primary_key=True)
    loan = models.ForeignKey(Loan)
    dueDate = models.DateField()
    paymentAmount = models.DecimalField(max_digits=10, decimal_places=4)
    paymentStatus = models.SmallIntegerField()

    def __unicode__(self):
        return u'%s' % self.loanPaymentsKey
```

MODELS.PY – OSA 3

```
class Meta:
    verbose_name_plural = 'Loan Payments'

class UserManager(BaseUserManager):
    def create_user(self, email, password=None):
        if not email:
            raise ValueError("Users must have an email address")

        user = self.model(email=self.normalize_email(email),
                           last_login=timezone.now(),
                           )
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password):
        u = self.create_user(email, password=password)
        u.is_admin = True
        u.is_superuser = True
        u.save(using=self._db)
        return u

class User(AbstractBaseUser, PermissionsMixin):
    id = models.AutoField(primary_key=True)
    email = models.EmailField(verbose_name='email address'
                              , max_length=255, unique=True, db_index=True)
    is_company = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_admin = models.BooleanField(default=False)

    objects = UserManager()

    USERNAME_FIELD = 'email'

    class Meta:
        app_label = 'lendsome'

    def get_group_permissions(self, obj=None):
        return set()

    def get_all_permissions(self, obj=None):
        return set()

    def get_full_name(self):
        return self.email

    def get_short_name(self):
        return self.email

    def __unicode__(self):
        return self.email

    def is_active(self):
        return self.is_active

    def is_company(self):
        return self.is_company
```

MODELS.PY – OSA 4

```
@property
def is_staff(self):
    return self.is_admin

class Country(models.Model):
    countryKey = models.AutoField(primary_key=True)
    countryName = models.CharField(max_length=127, blank=True, verbose_name=_('Nimi'))
    countryDescription = models.CharField(max_length=255, blank=True, verbose_name=_('Kuvaus'))
    continent = models.CharField(max_length=127, blank=True)
    region = models.CharField(max_length=127, blank=True)

    def __unicode__(self):
        return u'%s' % self.countryName

    class Meta:
        verbose_name_plural = 'Countries'

class UserProfile(models.Model):
    user = models.ForeignKey(User, unique=True)
    first_name = models.CharField(max_length=127, verbose_name=_('Etunimi'))
    last_name = models.CharField(max_length=127, verbose_name=_('Sukunimi'))
    phone_number = models.CharField(max_length=127, blank=True, verbose_name=_('Puhelinnumero'))
    date_of_birth = models.DateField(null=True, blank=True, verbose_name=_('Syntymäaika'))
    primary_IBAN_account_number = models.CharField(max_length=127, blank=True, verbose_name=_('IBAN'))
    address1 = models.CharField(max_length=255, blank=True, verbose_name=_('Osoite'))
    address2 = models.CharField(max_length=255, blank=True, verbose_name=_('Osoite 2'))
    country = models.ForeignKey(Country, default=1, verbose_name=_('Maa'))
    currency = models.ForeignKey(Currency, default=1, verbose_name=_('Valuutta'))
    business_id = models.CharField(max_length=127, blank=True, verbose_name=_('Y-tunnus'))

    def __unicode__(self):
        return u'User profile: %s' % self.user.email
```