Tallinna Ülikool

Digitehnoloogiate instituut

SISSEJUHATUS PYTHONI VEEBIRAAMISTIKKU DJANGO LIHTSA NÄITE ABIL

Seminaritöö

Autor: Konstantin Tenman

Juhendaja: Inga Petuhhov

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Iga veebirakendus on erinev. Harva leidub lahendust, mis sobib täpselt kasutaja vajadustega. Sellepärast ongi loodud veebiraamistikud nagu Django või Rails[1]. Sellised raamistikud aitavad arendust kiirendada ning sisaldavad juba väljatöötatud lahendusi. Tänapäeval kasutatakse veebiraamistikke kõikjal ning sarnaselt Pythonile on kõikidel programmeerimiskeeltel vähemalt üks kõike vajalikku sisaldav raamistik (ingl *end-to-end framework*) (Ravindran, 2015).

2005. aastal välja antud Django raamistikul töötavad paljud veebilehed, sealhulgas Pinterest[2], Disqus[3] ja isegi The Onion[4] (Pinkham, 2015). Kuid ka suured organisatsioonid nagu näiteks Instagram[5], Mozilla.org[6] ja Openstack.org[7] kasutavad Djangot (Dauzon, Bendoraitis, & Ravindran, 2016).

Käesoleva töö teema valik tulenes vajadusest eestikeelse õppejuhendi järele, mille abiga oleks võimalik iseseisvalt õppida veebirakenduse loomist kasutades programmeerimise keelt Python. Töö valmimise hetkel oli võimalik Tallinna Ülikooli Digitehnoloogiate instituudis õppida ainet nimega Veebiraamistikud, kus oli võimalik antud õppejuhendit kasutada. Teema valikut mõjutas nii Django populaarsuse pidev kasv (Lisa 1) kuid ka autori isiklik huvi.

Käesoleva seminaritöö eesmärk on luua lühike õppejuhend, mille abil programmeerimiskogemusega huvilised, kellel on mõningased teadmised objektorienteerituse põhimõtete ja HTML-i kohta, saavad iseseisvalt luua lihtsa veebirakenduse, kasutades Django mudel-vaade-mall arhitektuurimustrit. Oodatav tulemus on õppejuhend, mis annaks ülevaate Python Django veebiraamistikust, arendamiseks kasutatavast tarkvarast ja tekitaks õppijates huvi veebirakenduste loomise vastu.

Eesmärkide saavutamiseks annab autor esimeses peatükis allikate põhjal ülevaate Python Djangost, selle populaarsusest, ja selle seadistamisest arendaja arvutis.

Teises peatükis tutvustatakse Python Django veebirakenduse esmase arendamise etappe. Käiakse üle mudelite loomine, mudelite salvestamine kontrollerite abil andmebaasi, ning lõpetuseks antakse ülevaade, kuidas saab luua erinevat tüüpi Django vaateid.

---

[1] http://rubyonrails.org

[2] https://www.pinterest.com

[3] https://disqus.com

[4] http://www.theonion.com

[5] https://www.instagram.com

[6] https://www.mozilla.org

[7] http://www.openstack.org

Kolmandas peatükis kirjeldatakse õppejuhendi testimist Tallinna Ülikooli Digitehnoloogiate Instituudi bakalaureuseõppe kolmanda kursuse tudengitega, antakse ülevaade testgrupilt saadud tagasisidest ja analüüsitakse tulemusi.

Antud seminaritöö raames valminud rakenduse lähtekood on saadaval aadressil https://github.com/ktenman/kogumik ja õppejuhend aadressil https://github.com/ktenman/kogumik/wiki.

1. Python Django

Django on veebiraamistik, mis kasutab programmeerimiskeelt Python[8] veebirakenduste loomiseks. Selle põhiline eesmärk on kirjutada kiiresti just dünaamilise sisuga veebilehti (Dauzon, Bendoraitis, & Ravindran, 2016).

Django on tasuta, avatud lähtekoodiga serveripoolne veebiraamistik (ingl *back-end web framework*), mis eemaldab veebilehtede loomisel keerulisema osa, pakkudes enamiku vajalikku funktsionaalsust. Django võtab üle valdava osa hüperteksti edastusprotokolli (ingl *Hypertext Transfer Protocol* ehk HTTP) päringu ja vastuse tsükli käsitlemisest. Arendajad saavad keskenduda ainult HTTP päringute saatmisele, mida Django sisseehitatud tööriistad võimaldavad lihtsasti teha. Järgnevates peatükkides tutvustakse neid vahendeid lähemalt.

1.1. Django veebiraamistiku olemus ja arhitektuur

Raamistik on tarkvara kogumik, mis organiseerib rakenduse arhitektuuri ja teeb arendaja töö palju lihtsamaks. Django üheks suurimaks plussiks on see, et sellega saab kohe kasutama hakata praktilisi tööriistu, mis kiirendavad arendajate tööd. Näiteks peale Django paigaldust luuakse administraatori õigustega kasutaja, kes pääseb ligi administraatori paneelile, kus saab mugavalt hallata andmebaasi ja kasutajaid.

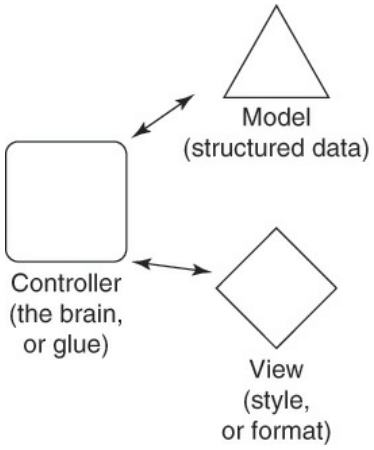
Django ametlik kirjeldus ütleb Django kohta, et see on kõrgtaseme Python veebiraamistik, mis võimaldab kiiret arendust ning puhtaid lahendusi (Django Software Foundation, 2016). Django järgib mudel-vaade-mall (ingl *Model-view-template* ehk MVT) arhitektuurimustrit (Ravindran, 2015).

Erialases kirjanduses on aga Django projekti struktuuri kõige tihedamalt kirjeldatud kui mudel-vaade-kontroller (ingl *Model-View-Controller* ehk MVC) arhitektuurimustrit, mis teeb raamistiku õppimise lihtsamaks (Pinkham, 2015). MVT mustril asendatakse vaated mallidega ning kontrollerid vaadetega (Dauzon, Bendoraitis, & Ravindran, 2016). Seega koosnevad maillid HTML-ist ning vaated ja mudelid Pythoni koodist.
Pinkhami (2015) järgi oli MVC algselt väga täpne arhitektuur, kuid nüüdseks kasutatakse seda terminit teekide kohta, mis täidavad järgnevaid punkte (Joonis 1).

- **Mudel** (ingl *model*) juhib andmete organisatsiooni ja ladustamist ning võib mõjuda konkreetsete andmete käitumist. Iga mudel defineerib tabeli andmebaasis ning seosed tema enda ja teiste mudelite vahel. Tänu mudelitele on iga osa andmetest hoiustatud andmebaasis.

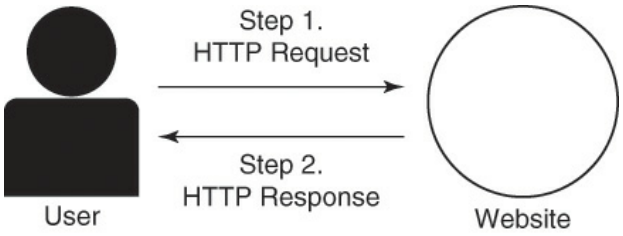
[8] https://www.python.org

- **Vaade** (ingl *view*) juhib kuidas andmeid näidatakse ning genereerib väljundi, mida näidatakse kasutajale. Seega võib vaatega seostada HTML koodi.
- **Kontroller** (ingl *controller*) on mudeli ja vaate vahelüli, mis teeb kindlaks, milliseid andmeid kasutaja soovib, ning tagastab need, kuid võib ka andmeid mudelist näidata või kasutada vaadet, et andmeid muuta. Põhimõtteliselt sisalduvad kontrolleris kõik tegevused, mida teeb server, ning mis ei ole kasutajale nähtavad.

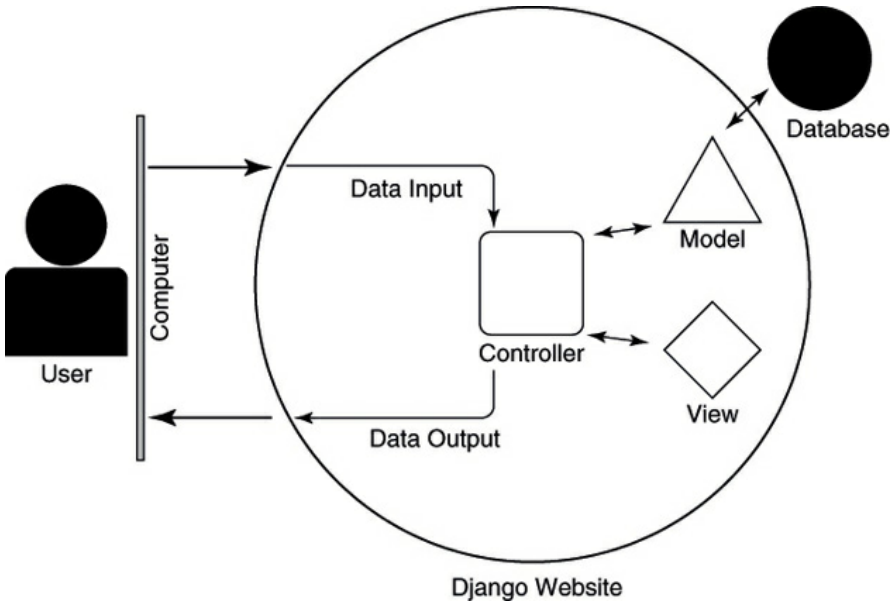


Joonis 1. MVC arhitektuurimuster (Pinkham, 2015)

Kombineerides HTTP päring-vastus tsükli diagrammi (Joonis 2) MVC arhitektuuri diagrammiga (Joonis 1), saab parema arusaama, kuidas Django töötab (Joonis 3).



Joonis 2. Kasutaja ja veebilehe päring-vastus tsükkel (Pinkham, 2015)



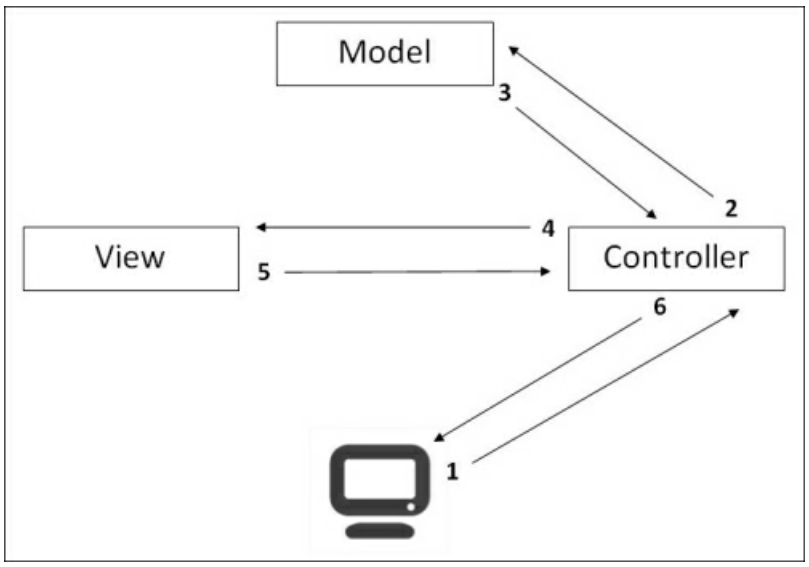
Joonis 3. MVC rakenduse arhitektuur (Pinkham, 2015)

Kontroller sümboliseerib Django tuuma ning on ainuke MVC arhitektuuri osa, mis on vajalik veebilehe genereerimiseks, ning vaade väljastab andmeid sellistes formaatides, nagu veebilehitsejad seda soovivad (nt. XML, HTML või HTML5). Kontrolleri jaoks sisu genereerimine vajab püsivaid andmeid ning mudelis on kõik vajalik, et neid struktureerida ja hoiustada.

Autori arvates võimaldab MVC muster luua iga arendaja ja projekti vahel sidususe. Näiteks veebiagentuuris, kus on veebidisainer ja arendajad, loob disainer vaateid. Vaated sisaldavad ainult HTML koodi. Nõnda tegutsedes ei sega disainerit arendajate kirjutatav kood, sest viimased muudavad mudeleid ja kontrollereid.

Mudeli sektsioon on ühendatud andmebaasiga, kuid mudel ise ei ladusta andmeid, vaid tagab vajalikud vahendid, et andmebaasiga suhelda. Django poolt pakutavad tööriistad võimaldavad kasutada järgnevaid andmebaasisüsteeme: SQLite, MySQL, PostgreSQL ning Oracle (Pinkham, 2015).

MVC arhitektuurimustrit kasutavad rakendused töötavad järgmisel põhimõttel (Joonis 4):



Joonis 4. Andmete liikumise skeem MVC rakenduses (Dauzon, Bendoraitis, & Ravindran, 2016)

1. Klient saadab serverile päringu, et talle lehte näidataks.
2. Kontroller kasutab andmebaasi läbi mudelite ning võib andmeid luua, lugeda, uuendada või kustutada.
3. Mudel edastab informatsiooni andmebaasist, näiteks saadab raamatukogu teavikute nimekirja kontrollerile.
4. Kontroller paigutab andmed nende põhjal genereeritud vaatesse.
5. Vaade tagastab sisu sõltuvalt kontrollerist saadud informatsioonile.
6. Kontroller tagastab kuvatava sisu, näiteks HTML dokumendi, kasutajale.

1.2. Django eelised ja puudused

George (2016) seisukohalt võimaldab Django kiiresti luua suuri ja dünaamilisi veebilehti. Django pakub kõrgetasemelisi veebiarendusmustrite abstraktsioone, otseteid sagedaste programmeerimisülesannete puhul, ning parimaid mustreid (ingl *patterns*), kuidas probleeme lahendada.

Python programmeerimiskeelel on väga palju teeke. 2. novembri 2016 seisuga oli PyPi (Python Package Index[9]) andmetel 15772 paketti, millest Django alla kuulus 7839.

[9] https://pypi.python.org/pypi

3. novembril 2016 oli Pythonil üle 63 aktiivse veebiraamistiku (Django Software Foundation, 2017). Nendest kõige populaarsemad on Django, TurboGears[10], Flask[11] ja Pyramid[12]. Pythoni raamistikud on ka väga erinevad. Pisike Bottle[13] veebiraamistik on sisuliselt kõigest üks Python fail millel ei ole ühtegi olemiseost ja on üllatavalt kompetentne lihtsa veebirakenduse loomisel.

Kuigi võimalusi on väga palju, siis Djangost on siiski saanud suur kasutajate lemmik. Djangosites.org[14] portaalis on nimetatud rohkem kui 5169 Django abil loodud veebilehte (Django Software Foundation, 2016). Siinkohal tuleb märkida, et antud statistikasse pääsemiseks on vajalik veebilehe registreerimine Djangosites.org keskkonnas.

Üks Django unikaalseid omadusi on **administraatori liides**, mis tuleb paigaldusega kaasa. See on tõhus vahend andmete sisestamisel ning testimisel. Tuleb ka ära mainida, et Django kasutusjuhend on avatud lähtekoodiga projekti kohta väga hästi kirjutatud.

Django abil loodud lehed on turvalised ning suudavad ennast kaitsta tuntud rünnakute eest nagu seda on murdskriptimine[15] (ingl *cross-site scripting* ehk XSS) ja päringvõltsing[16] (ingl *Cross-site request forgery* ehk CSRF).

Järgmises loetelus on välja toodud Django kasutamise eelised (Dauzon, Bendoraitis, & Ravindran, 2016):

- Django kuulub BSD litsentsi kaitse alla, mis tagab veebilehtede probleemidevaba tasuta kasutuse. BSD seab vähe piiranguid tarkvara ning selle lähtekoodi kasutamisele (Debian Project, kuupäev puudub).
- Django on täielikult kohandatav ning arendajad harjuvad sellega kergesti, luues mooduleid või üle kirjutades raamistiku meetodeid.
- Django modulaarsuse tõttu on olemasolevale projektile võimalik paljus teisi teeke lisada. Tänu sellele on võimalik abiks võtta teiste inimeste tehtud kvaliteetsed ja vajalikud moodulid.

[10] http://www.turbogears.org

[11] http://flask.pocoo.org

[12] http://www.pylonsproject.org

[13] http://bottlepy.org

[14] https://www.djangosites.org

[15] Murdskriptimine võimaldab ründajatel enda koodijuppe veebilehte süstida.

[16] Päringvõltsing on tuntud kui ühe kliki (ingl *one-click*) rünne või sessiooni ärandamise rünne, mille käigus kasutatakse veebilehe usaldust kasutaja vastu. See annab võimaluse ründajal tegutseda kasutaja nimel ja tema õigustega ning teha muutusi vastavas süsteemis (Ristic, 2005).

- Pytho Django raamistikku kasutades võib lisaks liidestada standard- või kolmanda osapoole Python teeke (Pinkham, 2015).
- Django raamistiku peamine eesmärk on täiuslikkus ning see on loodud inimestele, kes soovivad oma rakendustele puhast koodi ning head arhitektuuri. Django järgib *Don't Repeat Yourself*[17] filosoofiat, mis rõhub koodi taaskasutatavusele, mille põhieesmärgiks on vältida igasugust informatsiooni kordust (Ravindran, 2015).
- Django omab head kogukonda ning tänu sellele on oma koodi vigadele võimalik saada kiireid lahendusi. Lisaks on võimalik leida häid koodinäiteid, mis tutvustavad parimaid praktikaid.

Kuigi teoreetiliselt võib Django abil luua ükskõik millise veebilehe, ei pruugi see alati kõige parem lahendus olla. Näiteks reaalaja jututoa loomisel tasub selleks kasutada Tornadot[18] ning ülejäänud rakendus võib olla arendatud Djangoga (Ravindran, 2015). Seega iga töö jaoks tuleb kasutada õiget tööriista.

Dauzoni, Bendoraitise ja Ravindrani (2016) arvates peab iga arendaja, kes alustab raamistiku kasutamist, algselt läbima õpifaasi, mille pikkus oleneb raamistikust ning arendajast. Django õpifaasi pikkus on võrdlemisi lühike kui arendaja on tuttav Pythoniga ja objekt-orienteeritud programmeerimisega.

Lisaks on alati võimalik olukord, et Django raamistiku uuema versiooniga tuleb süntaksis muutusi. Vaatamata sellele tuleb iga Django uuendusega kaasa dokumentatsioon, mis seletab lahti uuenduses toimunud muudatused.

1.3. Django rakenduse arendus

Käesolevas peatükis kirjeldakse seda, kuidas näeb välja Django rakenduse struktuur, milliseid tarkvarasid võiks kasutada arendamiseks ning millises keskkonnas oleks kõige mõistlikum arendada.

Django Arenduskeskkond

Alguses toetas Django ainult Python 2-te, kuid alates 26. veebruarist 2013, mil ilmus Django versioon 1.5, hakati toetama ka Python 3-e (Bennett, 2013). Antud seminaritöö tegemisel kasutati Pythoni 3.5.2 ja Django 1.10.2 versioone. Andmebaasina kasutatakse SQLite'i, mis tuleb kaasa Django raamistikuga.

[17] *Don't Repeat Yourself* on tarkvaraarenduse põhimõte, mille põhieesmärgiks on vältida igasugust informatsiooni kordust.

[18] http://www.tornadoweb.org

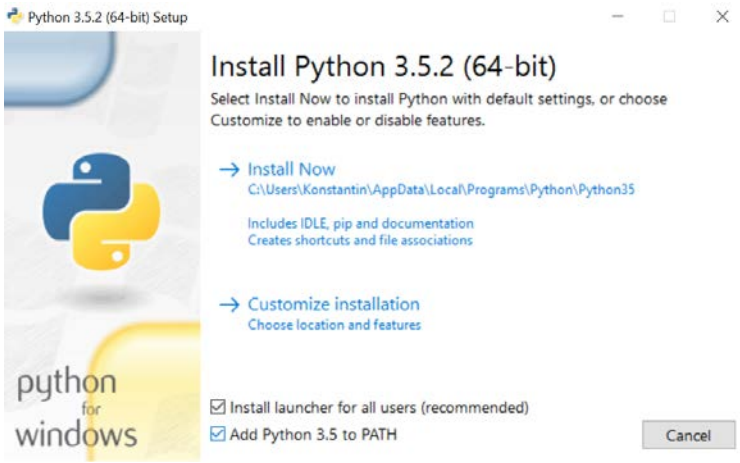
Selle seminaritöö raames loodud juhendit on võimalik läbi teha nii Windows'i kui ka UNIX baasil operatsioonisüsteemidel. Kuid UNIX masinates Python 3-e käivitamine käsureal toimub käsuga `python3`.

Lisaks eelnevale on tungivalt soovitatav kasutada mõnda integreeritud programeerimiskeskkonda (ingl *Integrated Development Environment ehk IDE*), näiteks PyCharm Professional Edition'it (Lisa 2), mida üliõpilased saavad kasutada tasuta litsentsi alusel. Suurimateks plussideks on see, et PyCharm annab enne kompileerimist märku, kas taane on õigesti tehtud või mitte. Lisaks pakub IDE järgmiseid eeliseid (Tõnisson, kuupäev puudub):

- levinud vigade ja probleemide jooksvalt välja toomine (ingle *code inspections*);
- muutujate ja funktsioonide nimede automaatne lõpetamine (ingle *code completion*);
- kiire ja mugav koodis navigeerimine (leiab kõik meetodi kasutuse kohad, kasutamise kohast definitsiooni juurde hüppamine jne).

Arenduskeskkonna seadistamine

Windows'i operatsioonisüsteemiga arvutis peab olema keskkonnamuutujas PATH määratud kataloginimi, kus paiknevad programmid `python` ja `pip`[19]. Seega peaks olema paigaldatud PATH versioon (Joonis 5) või lisatud PATH käsitsi, et saaks mugavalt kasutada `python` ja `pip` käske. Linuxis töötades peab konsooli aknasse kirjutama `python` käsu asemel `python3` ja `pip` asemel `pip3`. Kuid mugavuse mõttes soovitab autor Linuxi arvutis luua käsurealt sünonüümi järgneva käsuga: `alias python=python3`.



Joonis 5. Tuleb kindlasti valida PATH

[19] https://pypi.python.org/pypi/pip

2. Näidis veebirakenduse loomise juhend

Djangoga on kaasas tööriistad, mis teevad veebilehtede ehitamise lihtsaks. Näiteks Djangosse on sisseehitatud autentimise süsteem ning peetakse meeles andmebaaside struktuuri muudatuste ajalugu. Seega juhendis tutvustakse autori arvates kõige olulisemad Djangoga seotud funktsionaalsused. Töös on näidetena kasutatud autori enda tehtud koodinäiteid ja internetis vabalt levivaid õpetusi.

Juhendis allkirjastatakse vaid pikemad koodinäited, aga koodi üksikud väljavõtted, mida kasutatakse koodi parema loetavuse huvides, jäetakse allkirjastamata. Pärast igat suuremat muudatust on peatüki lõpus viidatud Githubis valmiva koodi versioon hetkeseisuga.

2.1. Rakenduse idee

Käesoleva lühijuhendi eesmärgiks on teha lihtne veebirakendus, mis kasutaks relatsioonilist andmebaasi. Luuakse kaks andmebaasitabelit (raamatud ja autorid), mida on administraatori vaates võimalik hallata. Lisaks antakse ülevaade sellest, kuidas mudel-vaade-mall on üles ehitatud.

Seega rakenduse loomise käigus tutvustatakse järgnevaid aspekte:

- mudelid ja mitu-mitmele seose loomine;
- mudelite andmebaasi salvestus ja lihtsamate filtrite rakendamine;
- mudelite haldamine administraatori vaates.

2.2. Uue projekti tegemine

Django moodulite paigaldamine tehakse käsurealt administraatori õiguste alt järgneva käsuga:

```
pip install django
```

Tee valmis projekt nimega “kogumik” käsurealt järgneva käsuga:

```
django-admin startproject kogumik
```

Seejärel navigeeri “kogumik” kataloogi käsuga `cd kogumik`. Tulemuseks peaks olema järgnev struktuur (Joonis 6):

```
manage.py
kogumik
    settings.py
    urls.py
    wsgi.py
    __init__.py
```

Joonis 6. Näidisrakendus failistruktuur peale loomist

- `manage.py`[20] on Django käsureautiliit, mille kaudu pannakse käima Django serveripoolne rakendus ja teostatakse muid haldusega seotuid ülesandeid (Django Software Foundation, 2016). Järgnevas peatükis tutvustatakse pikemalt selle kasutust.
- `settings.py`[21] on Djangoga loodud projekti konfiguratsiooni fail.
- `urls.py` on deklareeritud linkide kogumik või teisisõnu Djangoga loodud projekti "sisukord" (Django Software Foundation, 2016).

Ülejäänud failid (`wsgi.py` ja `__init__.py`) ei vaja antud töö kontekstis lahtiseletamiset, sest nende sisu ei muudeta.

Olles "kogumik" juurkataloogis geneeri käsurealt SQLite[22] andmebaasi failid, kasutades `manage.py` utiliiti:

```
python manage.py migrate
```

Peale seda on juurkataloogi tekkinud `db.sqlite3`. Selles failis hoitakse veebirakenduse kasutamise käigus tekkinud sisu.

Koodi seisukord: https://github.com/ktenman/django/tree/01-uue-projekti-tegemine

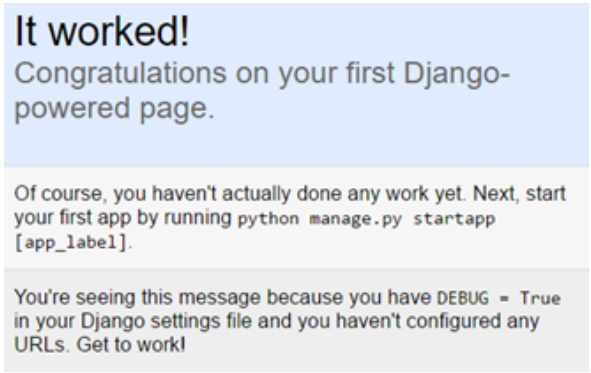
2.3. Esimene käivitamine

Djangoga loodud projekti käivitamiseks sisesta käsureale järgnev käsk:

```
python manage.py runserver
```

Nüüd peaks server lokaalselt töötama pordil 8000: http://127.0.0.1:8000 või http://localhost:8000

Kui kõik läks edukalt, siis brauseri aknas peaks kuvatama midagi sarnast (Joonis 7):



Joonis 7. Djangoga korrektselt loodud projekt

[20] https://docs.djangoproject.com/en/1.10/ref/django-admin

[21] https://docs.djangoproject.com/en/1.10/topics/settings

[22] https://sqlite.org

2.4. Veebirakenduse paigaldus ja integreerimine projekti

Katkesta serveri töö:

```
ctrl + c
```

Järgneva käsuga genereeri valmis rakenduse alamosa “books”, mida hakatakse kasutama administraatori lehel:

```
django-admin startapp books
```

Administreerimise eesmärgil loodud rakenduse alamosa võimaldab administraatori paneeli alt hallata andmebaasis olevaid kirjeid. Nüüd on loodud eraldi `books` kaust, kuid ka `settings.py` failis tuleb see ära määrata. Seega lisa eraldi reana sõna `books`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'books',
]
```

Kood 1. Books alamprogrammi integreerimine Django projekti, kasutades settings.py faili

Koodi seisukord: https://github.com/ktenman/kogumik/tree/02-books-veebirakenduse-paigaldus

2.5. Mudelite arendus

Iga mudel viitab ühele tabelile andmebaasis ja koosneb järgnevatest aspektidest (Django Software Foundation, 2016):

- Iga mudel on Pythoni klass, mis on `django.db.models.Model` alamklass.
- Iga mudeli atribuut vastab ühele andmebaasi väljale.
- Kõige eelnevaga koos pakub Django automaatselt genereeritud andmebaasi ligipääsu API-liidest. See API võimaldab näiteks objekte luua, otsida, uuendada ja kustutada.

Mudeli loomiseks tuleb defineerida `books` kaustas `models.py` failis uus Pythoni klass `Book` (Kood 2):

```
from django.db import models

# Create your models here.
```

```
class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    comment = models.TextField(blank=True, null=True)
    date_commented = models.DateField(blank=True, null=True)
    is_bookmarked = models.BooleanField(default=False)

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

Kood 2. Näide mudeli defineerimisest Django projektis

`models.CharField(max_length=200)` – atribuudi tüübiks on string ja selle maksimaalseks pikkuseks on 200 sümbolit.

`models.TextField(blank=True, null=True)` – antud välja hoitakse andmebaasis teksti kujul, kusjuures on lubatud jätta tekstiväli tühjaks ning tühi väli võib-olla kas `NULL` väärtus või tühistring.

`models.BooleanField(default=False)` – andmeid hoiustatakse tõeväärtusmuutuja abil, mille vaikeväärtuseks on `False`.

`__str__(self):` - tänu sellele meetodile väljastakse raamatu pealkiri mitte objekti aadress. Ehk siis on tegemist näiteks Javast tuntud `toString()` meetodiga.

Lisa `models.py` faili ülal välja toodud koodiread (Kood 2), salvesta ning käivita järgnev käsk:

```
python manage.py makemigrations
```

Nüüd peaks `migrations` kausta tekkima uus fail nimega `0001_initial.py`.

Migratsioonid võimaldavad igal meeskonnaliikmel andmebaasi nullist genereerida kaotamata vahepealseid muudatusi. Seetõttu on võimalik jooksvalt muuta välju andmetabelites. Näiteks kui mingi hetk otsutatakse, et mõni väli on üleliigne või puudu. Sünkroniseerimine ja migreerimine vii läbi käsuga:

```
python manage.py migrate
```

Andmebaasi luuakse järgnev SQLite süntaksis tabel (Kood 3).

```
CREATE TABLE "books_book" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "title" varchar(200) NOT NULL,
    "author" varchar(100) NOT NULL,
    "comment" text NULL,
    "date_commented" date NULL,
    "is_bookmarked" bool NOT NULL
);
```

Kood 3. Djangoga loodud rakenduse “Book” mudel SQLite andmebaasis

Koodi seisukord: https://github.com/ktenman/kogumik/tree/03-mudeli-loomine

2.6. Andmete salvestamine andmebaasi

Selles peatükis tutvustakse Django shell tööriista, mille abil lisatakse rakendusse näidisandmeid ja filtreeritakse neid.

Django shell tööriista väljakutsumine toimub käsurealt:

```
python manage.py shell
```

Näiteks salvesta andmebaasi raamat nimega “Minu Raamat", autoriks “Mina” ja kommentaariks “Hea raamat!” Selleks kasuta järgmist algoritmi (Kood 4):

1. Impordi `Book` mudel.
2. Loo raamatu objekt `book1`.
3. Salvesta loodud objekt andmebaasi.

```
from books.models import Book

book1 = Book(title="Minu Raamat", author="Mina", comment="Hea raamat!")

book1.save()
```

Kood 4. Objekti salvestamine andmebaasi, kasutades Django shell tööriista

`Book.objects.all()` käsuga on võimalik kontrollida, millised raamatud parasjagu andmebaasis asuvad.

Lisaks eelnevale saab Djangoga kasutada erinevaid **filtreid**:

- `Book.objects.filter(author="Mina")` - otsib üles kõik raamatud, mille autoriks on “Mina”.
- `Book.objects.filter(title__contains="Raamat")` – otsib üles kõik raamatud, mille pealkiri sisaldab sõna “Raamat”.
- `Book.objects.filter(author__contains="ina").filter(comment__contains="lugemine")` – saab kombineerida mitut filtrit korraga.

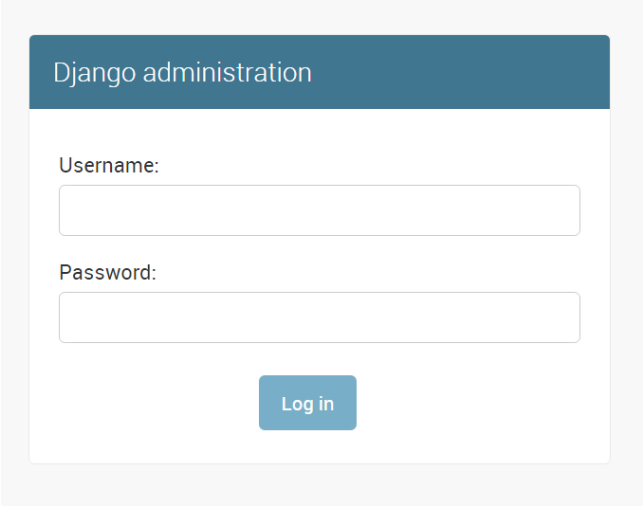
Koodi seisukord: https://github.com/ktenman/kogumik/tree/04-andmete-salvestamine-andmebaasi

2.7. Administraatori õigustega kasutaja loomine

Administraatori paneeli sisselogimiseks loo käsurealt uus kasutaja: `python manage.py createsuperuser`. Näiteks võib kasutajanimeks valida “kasutaja” ja parooliks panna “123test123”.

Seejärel täida ära kõik väljad ja kui kasutaja on edukalt loodud, siis käivita Django server: `python manage.py runserver`. Edaspidi kui koodis tehakse mingi muudatus, siis tuleb server taaskäivitada.

Nüüd peaks adminstraatori liides olema saadaval siin: http://127.0.0.1:8000/admin/ (Joonis 8).



Joonis 8. Django administraatori liidese sisselogimise vaade

2.8. Mudelite haldamine

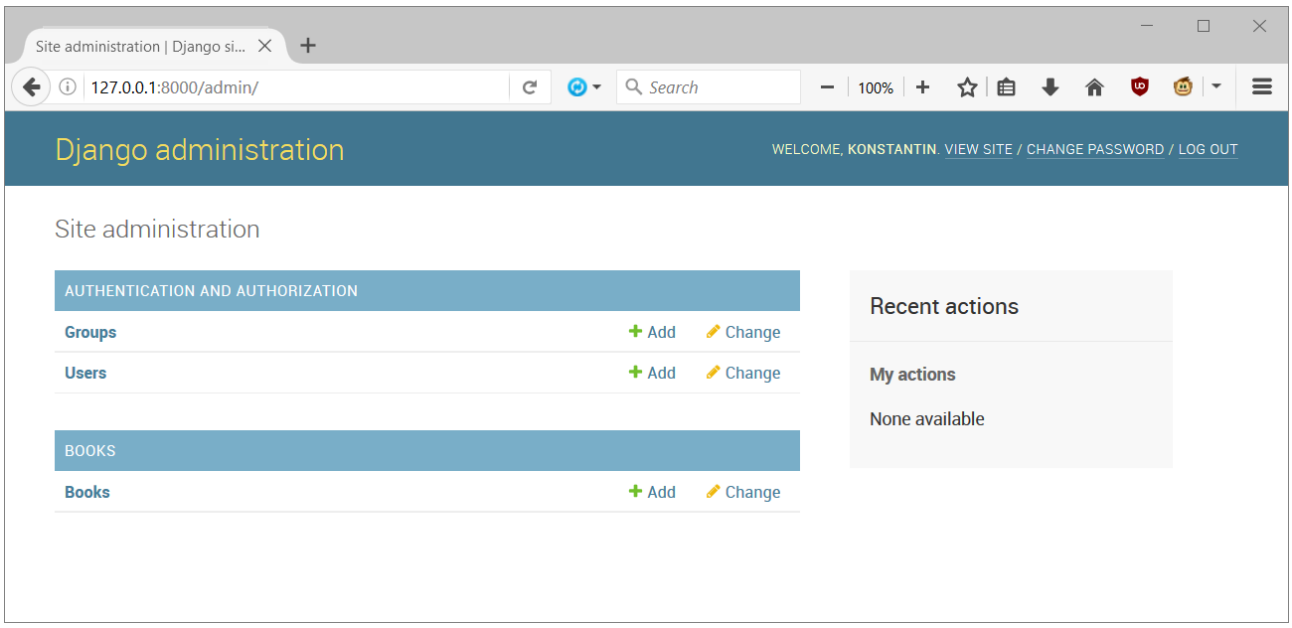
Raamatute haldamiseks administraatori paneeli alt on vajalik nende mudeli “Book” registreerimine `admin.py` failis (Kood 5).

```
from django.contrib import admin
from .models import Book

# Register your models here.
admin.site.register(Book)
```

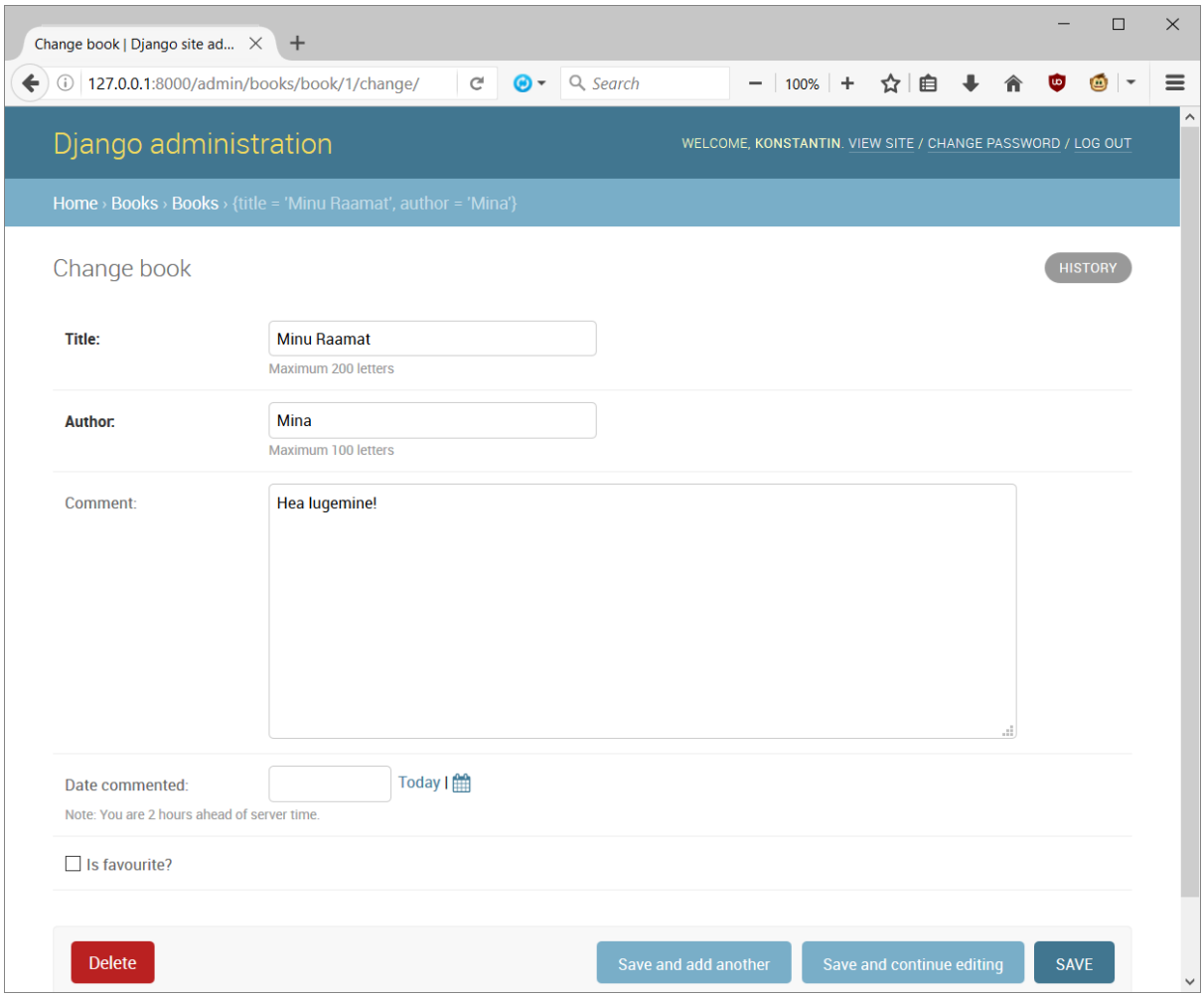
Kood 5. "Book" mudeli registreerimine admin.py failis

Taaskäivita rakendus ja märka, et administraatori liideses tekib uus sektsioon nimega “Books” (Joonis 9).



Joonis 9. Sisselogitud administraatori vaade

Lisaks eelnevale, et välju oleks mugavam hallata, võib lisada mõned kirjeldavad ja abistavad tekstid (Joonis 10).



Joonis 10. Ühe mudeli andmete muutmise kuva

Täienda `models.py` faili "Book" klassi järgnevate atribuutidega (Kood 6).

```
from django.db import models


# Create your models here.
class Book(models.Model):
    title = models.CharField(max_length=200, help_text="Maximum 200 letters")
    author = models.CharField(max_length=100, help_text="Maximum 100 letters")
    comment = models.TextField(blank=True, null=True)
    date_commented = models.DateField(blank=True, null=True)
    is_bookmarked = models.BooleanField(verbose_name="Is favourite?", default=False)

    def __str__(self):  # represents object with title and author
        return "title = '%s', author = '%s'" % (self.title, self.author)
```

Kood 6. Mudeli atribuudid koos abistavate tekstidega

`help_text="Maximum 200 letters"` – kuvab sisestuskasti alla antud lause (Joonis 9).

`verbose_name="Is favourite?"` – "Is bookmarked" asemel kuvatakse "Is favourite".

Kuna mudelid on muudetud, siis peab muudatused migreerima ka andmebaasi (Käsurida 1). Neid kolme käsku tuleb kindlasti käivitada peale iga mudeli muudatust, sest vastasel korral andmebaasi ja rakenduse struktuur ei muutu.

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

Käsurida 1. Migratsiooni tegemine ja serveri käivitamine (vaata selgitusi lk 16)

Kui kõik õnnestub, siis `migrations` kausta luuakse uus fail nimega `0002_auto_20******_****.py` (Käsurea logi 1).

```
Operations to perform:
  Apply all migrations: admin, auth, books, contenttypes, sessions
Running migrations:
  Applying books.0002_auto_20161009_1522... OK
```

Käsurea logi 1. Migratsioonide tegemine andmebaasi oli edukas

Koodi seisukord: https://github.com/ktenman/kogumik/tree/05-mudelite-haldamine

2.9. Mitu-mitmele seose loomine

Ühel autoril võib olla mitu raamatut, seetõttu on mõttekas autori jaoks teha eraldi tabel andmebaasis. Selleks lisa autori klass mudelitesse `models.py` failis (Kood 7). Kuid suuremate rakenduste puhul ei ole soovituslik erinevad mudelid ühte faili kokku kirjutada.

```
class Author(models.Model):
    name = models.CharField(max_length=100, help_text="Maximum 100 letters", unique=True)

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

Kood 7. Klass autori mudelist

`unique=True` – antud väli on unikaalne, mis tähendab, et sama nimega autori sisestamine on keelatud.

Viita autoritele "Book" klassis asendades `author` selle reaga:

```
    authors = models.ManyToManyField("Author", related_name="books")
```

Nüüd on defineeritud mitu-mitmele seos. Kuna `author` atribuuti enam pole olemas, siis tuleks korda teha `__str__(self)` meetod "Book" klassis (Kood 8).

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

Kood 8. Meetod, mis objekti poole pöördumisel, väljastab raamatu pealkirja

Administraatori vaates autorite lisamiseks on vaja "Author" klass registreerida `admin.py` failis (Kood 9).

```
from django.contrib import admin
from .models import Book, Author

# Register your models here.
admin.site.register(Author)
admin.site.register(Book)
```

Kood 9. admin.py koos registreeritud mudelite klassidega

Migreeri uuendused andmebaasi (Käsurida 1). Andmebaasis tekib abitabel `books_book_authors`, mis määrab mitu-mitmele seose (Lisa 2).

Nüüd tuleb autorid uuesti luua, sest andmebaasi seosed on üle kirjutatud. Autori tabel on tühi ja sinna peab lisama mõned autorid. Seda on võimalik teha administraatori paneeli alt: http://localhost:8000/admin/books/author/add/. Lisa see autor ka esimesele raamatule: http://localhost:8000/admin/books/book/1/change/.

Raamatute sisestamise lihtsustamiseks võib kuupäeva lahtri ära täita ning kuvada kommenteerimise kuupäeva ja kellaaja. Selleks impordi päises kuupäeva ja kellaaja kuvamise teek:

```
from django.utils.datetime_safe import datetime
```

Muuda `date_commented` välja nii, et see kuvaks nii kuupäeva kui ka kellaaega:

```
date_commented = models.DateTimeField(default=datetime.now, blank=True)
```

Administraatori paneeli alt on võimalik igale raamatule määrata mitu arutorit, siis ka sellist olukoda võiks kuvada, seega lisa uus meetod `list_authors(self)` ja modifitseeri väljatrüki meetod `__str__(self)`:

```
    def __str__(self):
        return "%s by %s"% (self.title, self.list_authors())

    def list_authors(self):
        return ", ".join([author.name for author in self.authors.all()])
```

`","join(array)` – Stringi meetod, mis liidab kokku kõik elemendid massiivist ja eraldab need omavahel komaga (Python Software Foundation, 2017).

Uuenda andmebaas (Käsurida 1).

Koodi seisukord: https://github.com/ktenman/kogumik/tree/06-mitu-mitmele-seose-loomine

2.10. Raamatute administreerimine (Fieldsets)

Põhjalikemate vaadete kuvamiseks kasutatakse `fieldsets` tempiiti, millega on väljad võimalik jagada gruppidesse ning muuta kuvamise järjekorda täpselt nii nagu meeldib. `fieldsets` paremini mõistmiseks, lisa antud read `books` kaustas olevasse `admin.py` faili (Kood 10).

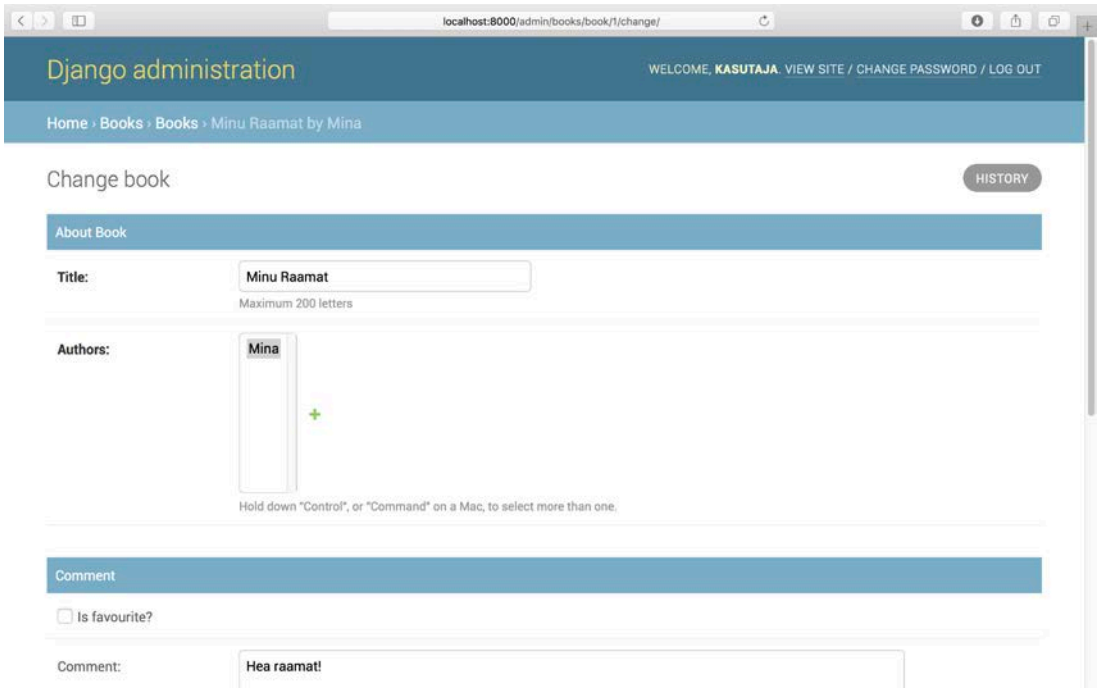
```
from django.contrib import admin
from .models import Author, Book

class BookAdmin(admin.ModelAdmin):
    fieldsets = [
        ('About Book', {'fields': ['title', 'authors']}),
        ('Comment', {'fields': ['is_bookmarked', 'comment', 'date_commented']}),
    ]
```

```
admin.site.register(Book, BookAdmin)
```

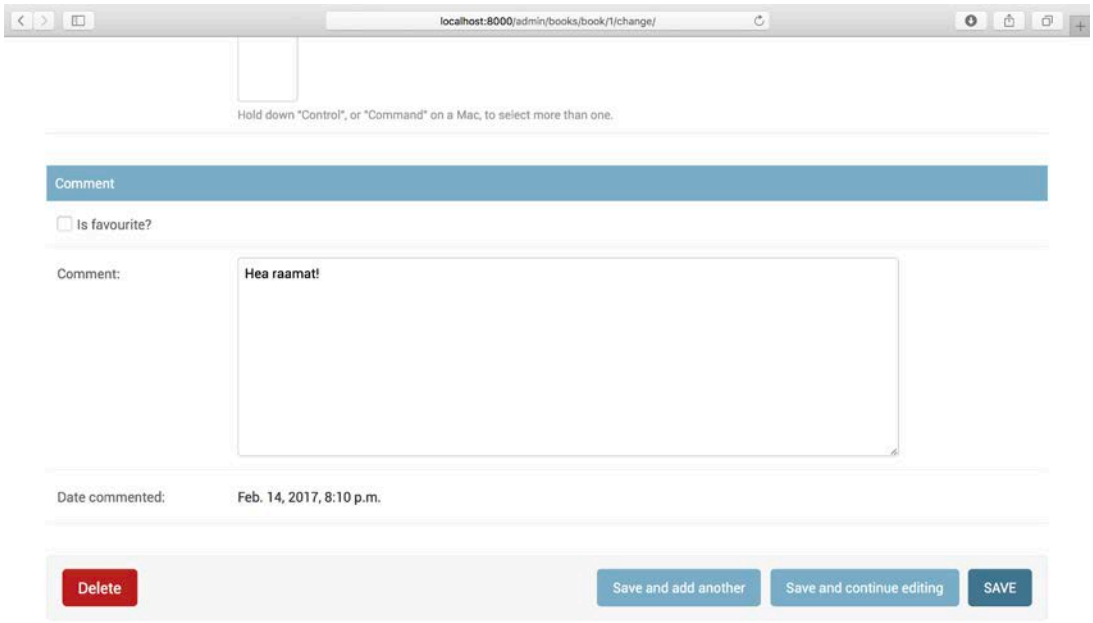
Kood 10. *admin.py* fail koos lihtsa *fieldsets* näitega

Tulemust saab näha siin: http://localhost:8000/admin/books/book/1/change/ (Joonis 11).



Joonis 11. Raamatu muutmise leht koos *fieldsets* templiidiga

Lisaks kui sa ei soovi, et mõnda välja saaks muuta, siis võid kasutada `readonly_fields` muutujat: `readonly_fields = ('date_commented',)` (Joonis 12).



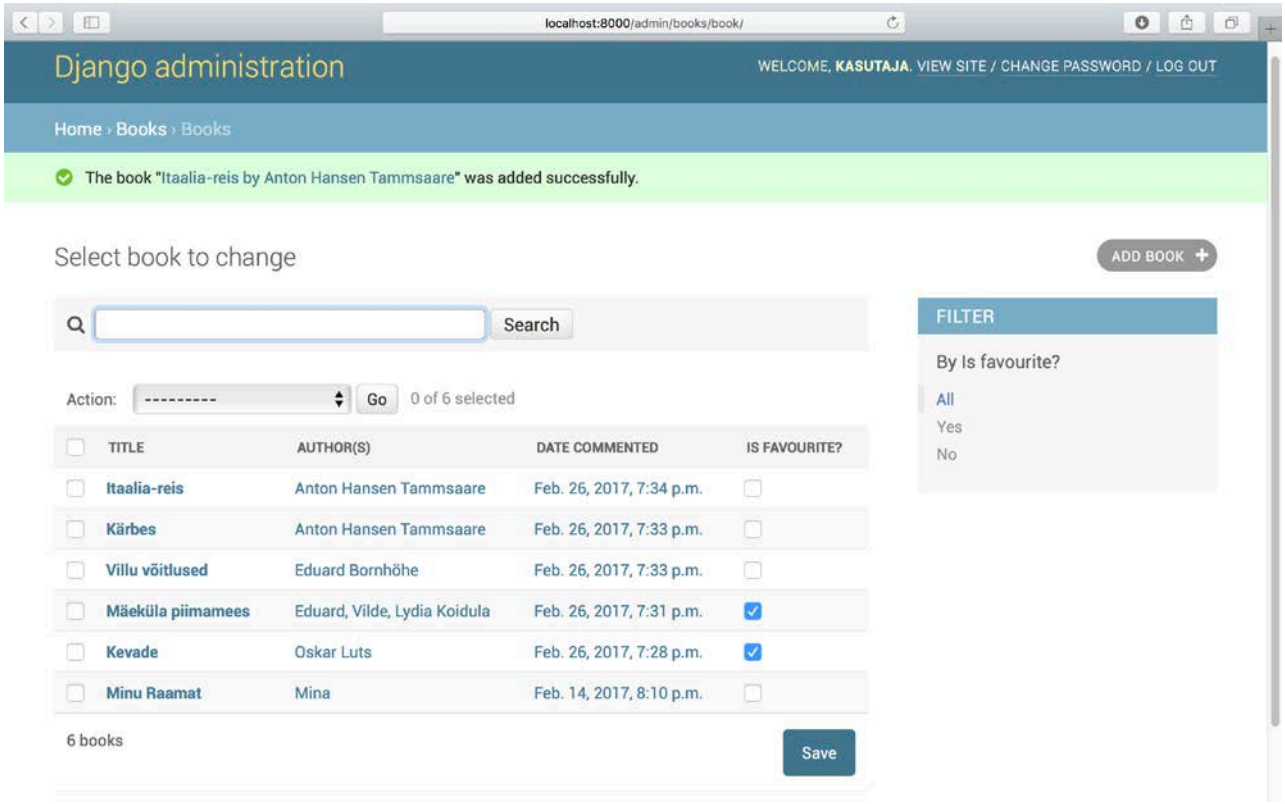
Joonis 12. "Date commented" välja ei saa muuta, sest on kasutatud "readonly_fields" muutujat

Kuna igal raamatul saab olla mitu autorit ja viidatakse teisele tabelile, on vajalik juurde lisada veel üks meetod, mis seda võimaldab (Kood 11).

```
    def book_authors(self, obj):
        return obj.list_authors()
    book_authors.short_description = 'Author(s)'
```

Kood 11. Mitme autori (objekti) kuvamise meetod

Raamatute mugavamaks haldamiseks tasub kasutusele võtta palju põhjalikum ülevaade, mida kuvatakse listina (Joonis 13).



Joonis 13. Mudelite listi vaade

Selleks lisa admin.py faili erinevad muutujad, mille abil kuvatakse rohkem informatsiooni raamatute kohta (Kood 12).

```
    list_display = ('title', 'book_authors', 'date_commented', 'is_bookmarked',)
    list_editable = ('is_bookmarked',)
    list_display_links = ('title', 'book_authors', 'date_commented',)
    list_filter = ('is_bookmarked',)
    search_fields = ['title', 'authors__name', ]
```

Kood 12. Valik erinevatest muutajatest, mille abil kuvatakse rohkem informatsiooni raamatute kohta

`list_display` – määrab väljad, mida kuvatakse;

`list_editable` – võimaldab välju muuta;

`list_display_links` – need väljad kuvatakse linkidena, peale klikkides avaneb detailsem vaade ühe raamatu muutmise jaoks;

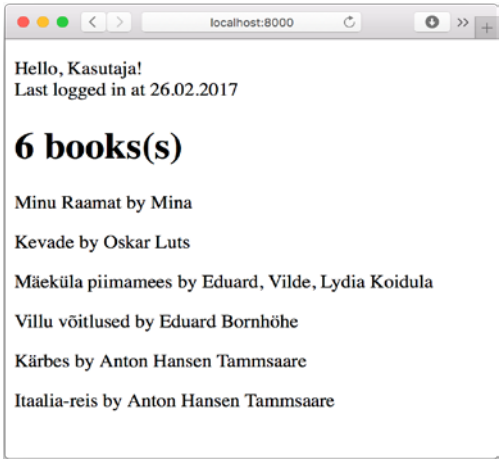
`list_filter = ('is_bookmarked',)` – tõeväärtusmuutuja väljale võib lisada filtri, mis pakub kolme võimalust: "All", "Yes" ja "No" (kõik, valitud ja mitte valitud raamatud) (Joonis 12);

`search_fields = ['title', 'authors__name', ]` – lisab otsingu raamatute pealkirjade ja autorite hulgast.

Koodi seisukord: https://github.com/ktenman/kogumik/tree/07-raamatute-administreerimine

2.11. Funktsioonipõhine vaade

Antud peatüki eesmärgiks on kuvada rakenduse pealehel (http://localhost:8000) kõik raamatud, mis andmebaasis on (Joonis 14).



Joonis 14. Pealeht raamatute nimekirjaga

Täienda `views.py` fail `books` objektiga. Siin tehakse päring andmebaasi ning saadakse kõikide raamatute info. `list_books` meetod töötab põhimõttel päring-vastus (ingl *request-response*) (Kood 13).

```
from django.shortcuts import render
from books.models import Book

def list_books(request):
    books = Book.objects.all()

    context = {
        'books': books,
    }

    return render(request, "list.html", context)
```

Kood 13. Funktsioonipõhine vaade

`context` – viitab rakenduse poolt kuvatavale sisule;

`render` – meetod, mis vastutab informatsiooni visualiseerimise eest.

Loo `books` kataloogi `templates` kaust ja lisa sinna `list.html` fail (Kood 14).

Loodud faili asukoht: `~/kogumik/books/templates/list.html`.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>kogumik</title>
</head>
<body>
{% if request.user.is_authenticated %}
    <p>Hello, {{ request.user.username | title }}! <br/> Last logged in
        at {{ request.user.last_login | date:"d.m.Y" }}</p>
{% else %}
    <p>Hello, guest!</p>
{% endif %}
<h1>{{ books.count }} books(s)</h1>
{% for book in books %}
    <p>
        {{ book.title }} by {{ book.list_authors }}
    </p>
{% endfor %}
</body>
</html>
```

Kood 14. `list.html`, kus kasutatakse *if*-lauset ja *for*-tsüklit

Kusjuures pane tähele, et ülalolevas koodis saadakse kätte `request` ja tagastakse samuti `request`. See tähendab, et `list.html` failis saad informatsiooni kuvamisel kasutada `request` ja `books` sisu.

`{%  %}` – loogeliste sulgude ja %-märkide vahele paigutub Django raamistiku kood.

`{{  }}` – topelt loogeliste sulgude vahele paigutuvad samuti Django rakenduse poolt tehtud objektid.

`{% if request.user.is_authenticated %}` – kontroll, kas kasutaja on sisse logitud.

`{{ request.user.username | title }}` – kuvatakse kasutajanime suure algustähega.

`{{ request.user.last_login | date:"d.m.Y" }}` – kuvatakse viimati sisselogitud kuupäeva vastava vorminguga (Näiteks: 26.02.2017).

Iga veebileht Internetis omab mingisugust aadressi. Tänu sellele teab rakendus, mida on vaja kasutajale kuvada, kui ta teatud lehekülge külastab. `urls.py` failis hoitaksegi reegleid või

mustrite kogumik, mis kirjeldavad, millist vaadet päringule vastuseks kuvada (DjangoGirls, 2017).

Täienda `urls.py`, selleks lisa `list_books` funktsioon, kus hoitakse vaadet, mis kuvab raamatute nimekirja (Kood 13).

```
from django.conf.urls import url
from django.contrib import admin

from books.views import list_books

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', list_books, name='books'),
]
```

Kood 15. URL-ide reeglid, mis panevad vastavusse rakenduse vaated ja aadressid

`r'^$'` – viitab regulaaravaldisele (ingl *regex*), kus `^` on regulaaravalise algus ja `$` on regulaaravaldise lõpp, mis kokku annavad tühja stringi. See šabloon kirjeldab: http://localhost:8000/ aadressi.

`r'^admin/`'- viitab aadressile: http://localhost:8000/admin/

`list_books` - vaade, mida antud URL-ile ette antakse.

`name='books'` – igale URL-ile võib anda unikaalse nime, mida saab tulevikus rakenduses kasutada. Selle kohta tuleb järgnevates peatükkides paar näidet.

Koodi seisukord: https://github.com/ktenman/kogumik/tree/08-funktsiooni-vaade

2.12. Klassipõhine vaade

Eesmärgiks on teha leht, kus kuvatakse selliseid autoreid, kes on kirjutanud vähemalt ühe raamatu. Selleks lisa järgmised *import*-käsud `views.py` faili:

```
from django.views.generic import View

from django.db.models import Count
```

Ja loo autorite kuvamise klass:

```
class AuthorList(View):
    @staticmethod
    def get(request):
        authors = Author.objects.annotate(
            published_books=Count('books')
        ).filter(
```

```
            published_books__gt=0
        )

        context = {
            'authors': authors
        }

        return render(request, "authors.html", context)
```

`urls.py` failis impordi vajalik klass:

```
from books.views import list_books, AuthorList
```

Ja lisa uus URL-i šablooni reegel:

```
    url(r'^authors/$', AuthorList.as_view(), name='authors'),
```

`templates` kausta tuleb luua `authors.html` mall, et saaks kuvada `AuthorList` (Kood 16).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    {% for author in authors %}
    <h3>{{ author.name }}</h3>
    {% endfor %}
</body>
</html>
```

Kood 16. `authors.html` mall

Koodi seisukord: https://github.com/ktenman/kogumik/tree/09-klassi-vaade

2.13. Genereeritud klassipõhine vaade

Nüüd loo dünaamiline veebilehestik, kus iga raamatu või autori kohta kuvatakse informatsiooni eraldi aadressil. Selleks impordi `views.py` faili:

```
from django.views.generic import View, DetailView
```

Ja lisa uued klassid:

```
class BookDetail(DetailView):
    model = Book
    template_name = "book.html"
```

```
class AuthorDetail(DetailView):
    model = Author
    template_name = "author.html"
```

Raamatute või autorite unikaalsete võtmete abil võib üles ehiteda URL-i šabloonid raamatute ühekaupa kuvamiseks `urls.py` failis (Kood 17).

```
from django.conf.urls import url
from django.contrib import admin

from books.views import list_books, AuthorList, BookDetail, AuthorDetail

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', list_books, name='books'),
    url(r'^authors/$', AuthorList.as_view(), name='authors'),
    url(r'^books/(?P<pk>\d+)/$', BookDetail.as_view(), name='book-detail'),
    url(r'^authors/(?P<pk>\d+)/$', AuthorDetail.as_view(), name='author-detail'),
]
```

Kood 17. Valik täiendatud URL-ide šabloonidest (Coding For Entrepreneurs, 2016)

`?P<pk>\d+` - aadressilt saadav parameeter on "pk" ja see tohib sisaldada ainult numbreid.

Nüüd tuleb `templates` kausta luua ka vajalikud HTML mallid: `author.html` (Kood 18) ja `book.html` (Kood 19).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Author</title>
</head>
<body>
    <h1>{{ author.name }}</h1>
    {% for book in author.books.all %}
        <h3>{{ book.title }}</h3>
        <p>{{ book.comment }}</p>
    {% endfor %}
</body>
</html>
```

Kood 18. Mall autori informatsiooni kohta

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Book</title>
```

```
</head>
<body>
    <h3>{{ book.title }}</h3>
    <p>By {% for author in book.authors.all %}
        {{ author.name }}{% endfor %}</p>
    <p>{{ book.comment|linebreaks }}</p>
</body>
</html>
```

Kood 19. Mall raamatu informatsiooni kohta

`{{ book.comment | linebreaks }}` - andmebaasi salvestatud reavahetused asendatakse HTML-i `<br />` elementidega (Django Software Foundation, 2016).

Nüüd saab tulemust näha aadressil: http://localhost:8000/books/1/ ja http://localhost:8000/authors/1/.

Koodi seisukord: https://github.com/ktenman/kogumik/tree/10-genereeritud-vaade

2.14. Korduste vältimine

Juhendi abil loodud rakenduses on väga palju koodikordusi, mis ei ole mõistlik. Koodi tasub taaskasutada ja vältida igasugust informatsiooni kordust (Ravindran, 2015)

Tee `kogumik` kausta uus `templates` kataloog ja loo sinna sisse uus `base.html` fail, mida hakatakse kasutama kõikides vaadetes (Kood 20). Edaspidi `base.html` mall hakkab olema kõikide teiste HTML failide aluseks.

```
<!doctype html>
<html lang="et">
<head>
    <meta charset="UTF-8">
    <title>Rakendus</title>
</head>
<body>
    {% if request.user.is_authenticated %}
        <p>Hello, {{ request.user.username | title }}! <br/> Last logged in
            at {{ request.user.last_login | date:"d.m.Y" }}</p>
    {% else %}
        <p>Hello, guest!</p>
    {% endif %}
    <nav>
        <a href="{% url 'books' %}">Books</a>
        <a href="{% url 'authors' %}">Authors</a>
    </nav>
    {% block content %}{% endblock %}
</body>
</html>
```

Kood 20. Baas html fail

Nüüd on olemas navigatsioon autorite ja raamatute vahel. Lisaks `{% block content %}{% endblock %}` – siin kuvatakse dünaamilist sisu vastavalt veebilehe aadressile.

Django rakendus suudab üles leida loodud faili `base.html`, kui täiendada `settings.py` fail ühe reaga, mis näitab sellle faili asukohta (Kood 21).

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'kogumik', 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Kood 21. Koodilõik `settings.py` failist

Faili `base.html` kasutamiseks tuleb muuta `book.html` fail, kus kasutakse `extends` ja `include` Django käske:

```
{% extends "base.html" %}
{% block content %}
{% include "book-detail.html" %}
{% endblock %}
```

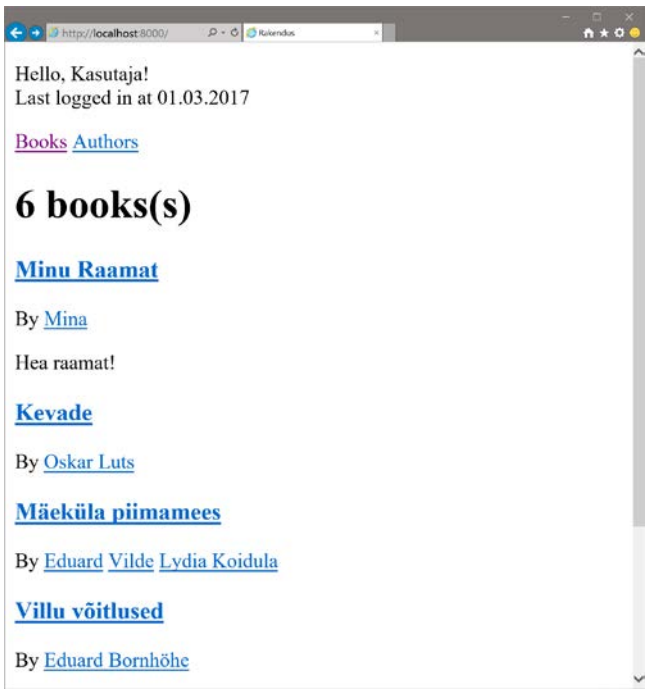
Mainida, et pannakse base.html vahele.

Loo `book-detail.html` fail samasse kataaloogi, kus on `book.html` (Kood 22).

```
<h3><a href="{% url 'book-detail' book.pk %}">{{ book.title }}</a></h3>
<p>By {% for author in book.authors.all %}
    <a href="{% url 'author-detail' author.pk %}">{{ author.name }}</a>
{% endfor %}</p>
<p>{{ book.comment|linebreaks }}</p>
```

Kood 22. Näide dünaamilisest mallist

Igal raamatul on viide raamatu detailidele. `book.pk` on antud raamatu unikaalne võti. Sellest koodilõigust võib aru saada, miks varem šabloonide defineerimisel kasutati nime `'book-detail'` . Samamoodi jätkates võib kõik HTML failid ümber kirjutada. Lõplik ilma kordusteta kood on saadaval siin: https://github.com/ktenman/kogumik/tree/11-ara-korda-ennast (Joonis 15).



Joonis 15. Lõpliku rakenduse pealehekülg

3. Juhendi testimine

Antud juhendit tehti läbi "Veebiraamistikud" aine raames. Kokku kulus Pythoni ja PyCharmi paigaldamiseks umbes pool tundi õppeajast. Kui lõpuks said kõik üliõpilase endale Python Django rakenduse vundamendi loodud, siis sujus edasine töö kiirelt. Testmise käigus leiti mõned kirjavead, ühes kohas olid vales järjekorras näidatud koodilõigud. Lisaks oli üks süntaktiline viga. Kogu juhendi läbi tegemiseks kulus umbes 3 akadeemilist tundi.

Autor andis seda juhendit läbi teha veel viiele õppurile iseseisvalt ja tagasisidena küsis järgnevaid küsimusi:

- Kas juhendit oli keeruline järgi teha?
- Kas tekkisid mingid tõrked?
- Mis oli hästi ja halvasti?
- Mida tulevikus võiks paremini teha?

Testimise käigus tuli välja mitmeid juhendi positiivseid ja negatiivseid külgi (Lisa 3). Nendest olulisemad võib sõnastada järgmiselt:

- juhend täitis oma eesmärgi;
- juhend on loogiliselt üles ehitatud;
- koodinäited on hästi lahti seletatud ja illustreeritud;
- Githubis on viidatud valmiva koodi versioon hetkeseisuga;
- pandi rõhku kirjutatud koodi abstraheerimisele ning korduste vältimisele;
- loodud näidisrakendus võiks suuremaid väljakutseid pakkuda;
- tuleb rohkem lahti seletada, mida iga muutuja tähendab;
- võiks olla mingi üldisem skeem, millised failid lõpuks tekkisid ja kus nad on;
- veidi kohmakas sõnastus.

Tulemuste ja ettepanekute põhjal võib järeldada, et juhend on loogiliselt üles ehitatud, kuid vajab rohkem sisulise seletusi ja täpsustusi. Positiivne oli ka see, et lõpuks said kõik testijad rakenduse täiesti valmis. Antud tulemusi autor arvastab tulevikus sarnase lühijuhendi loomisel.

Kokkuvõte

Käesolev seminaritöö lähtus probleemist, et Tallinna Ülikooli Digitehnoloogiate instituudi "Veebiraamistikud" aine raames puudus Python Django veebiraamistikku tutvustav lühike õppejuhend. Töö eesmärgiks oli tutvustada võimalusi, mida pakub Python Django veebiraamistik veebirakenduste loomisel ja luua lühike õpetus sellel teemal.

Eesmärgi saavutamiseks kirjutas autor teoreetilisest taustast, sealhulgas lühidalt Python Django olemusest ja selle töötamise põhimõttest. Loodi lühike juhend, kus autor tutvustas järgnevaid aspekte:

- mudelid ja mitu-mitmele seose loomine;
- mudelite andmebaasi salvestus ja lihtsamate filtrite rakendamine;
- mudelite haldamine administraatori vaates.

Juhendis selgitati, kuidas luua lihtne dünaamilise sisuga navigeeritav veebileht, kuhu on võimalik sisestada erinevaid andmeid kahte erinevasse andmebaasi tabelisse.

Tagasiside saamiseks jagas autor juhendi üliõpilastele, kes proovisid seda läbi teha. Tagasiside ja kommentaarid anti autorile vabas vormis. Saadud tagasiside põhjal muudeti töö struktuuri, parandati kirjastiili ning kirjavigu. Juhendi kohta negatiivseid kommentaare ei olnud ja süsteemi seadistusega tõrkeid ei tekkinud.

Seminaritöö kirjutamine andis autorile juurde uusi teadmisi Python Django raamistikust ja selle võimalustest. Lisaks ka teadmise, et head juhendit, mis kõikidele sobiks, on väga raske valmis teha. Sellegipoolest autori hinnangul on seminaritööks püstitatud eesmärgid saavutatud.

Täiendavalt võib autori arvates uurida, kuidas saab luua Django abil serveripoolne REST liidestus. Lisaks võib käsitleda andmete muutmist POST päringute kaudu ilma administraatori paneeli kasutamata.

Kasutatud kirjandus

Bennett, J. (2013, 26. veebruar). *Django 1.5 released*. Loetud aadressil https://www.djangoproject.com/weblog/2013/feb/26/15/

Coding For Entrepreneurs. (2016, oktoober 27). *Common Regular Expressions for Django URLs*. Loetud aadressil https://github.com/codingforentrepreneurs/Guides/blob/master/all/common_url_regex.md

Dauzon, S., Bendoraitis, A., & Ravindran, A. (2016). *Django: Web Development with Python.* Birmingham: Packt Publishing Ltd.

Debian Project. (kuupäev puudub). *Debian*. Loetud aadressil http://www.debian.org/legal/licenses/.

Django Software Foundation. (2016, 1. november). *Django documentation*. Retrieved from Django documentation: https://docs.djangoproject.com/en/1.10/

Django Software Foundation. (2017, 21. veebruar). *Python Wiki*. Loetud aadressil https://wiki.python.org/moin/WebFrameworks

DjangoGirls. (2017). *Django Girls Tutorial.* Loetud aadressil https://www.gitbook.com/book/djangogirls/djangogirls-tutorial/details

George, N. (2016). *Mastering Django: Core: The Complete Guide to Django 1.8 LTS.* Hamilton: GNW Independent Publishing.

Google. (2017). *Google Trends*. Loetud aadressil https://trends.google.com/trends/explore?date=all&q=python%20Django

Pinkham, A. (2015). *Django Unleashed.* Indianapolis: Sams Publishing.

Python Software Foundation. (2017). *Python 3.5.3 documentation.*

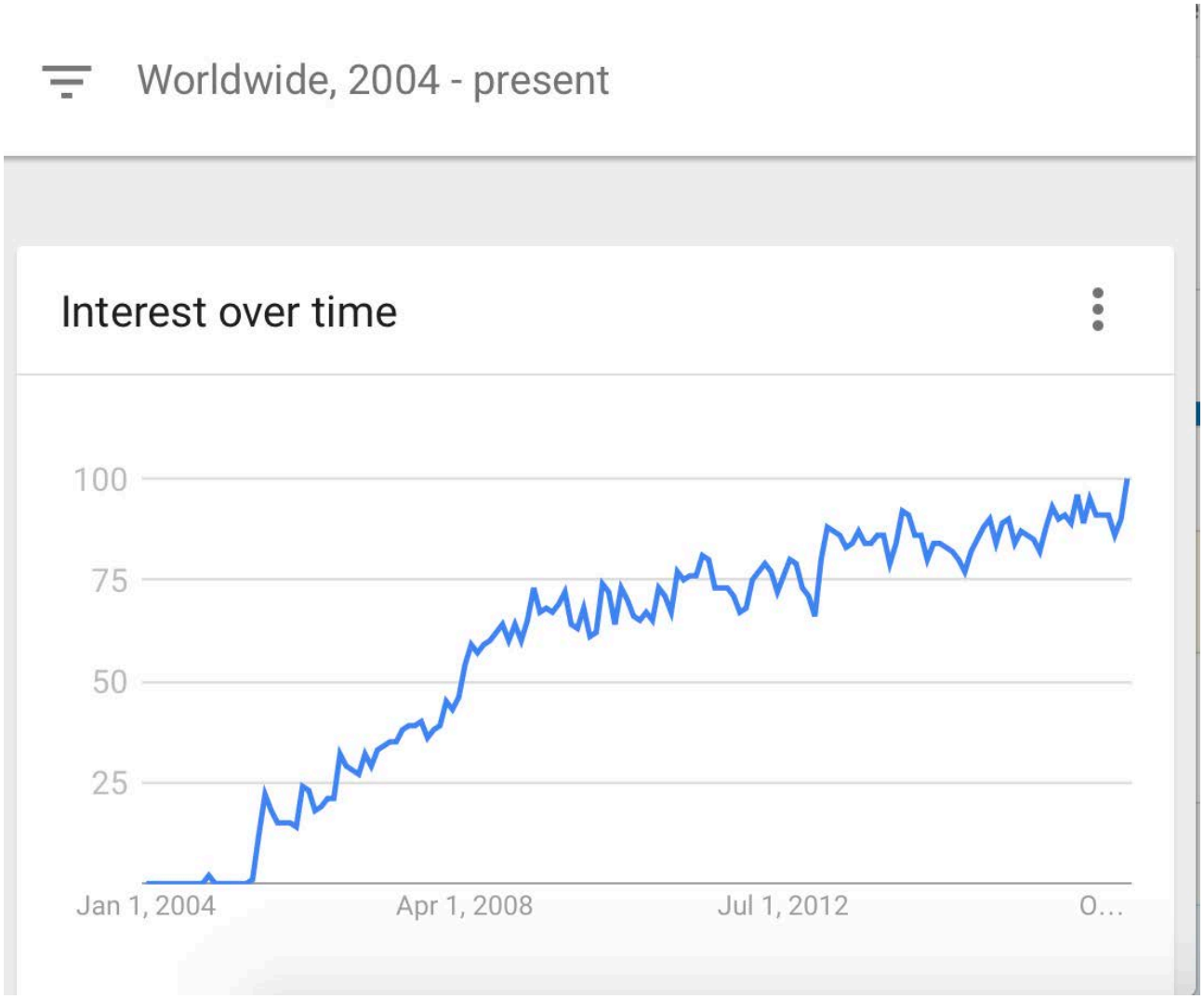
Ravindran, A. (2015). *Django Design Patterns and Best Practices.* Birmingham: Packt Publishing Ltd.

Ristic, I. (2005). *Apache Security.* Sebastopol: O'Reilly Media.

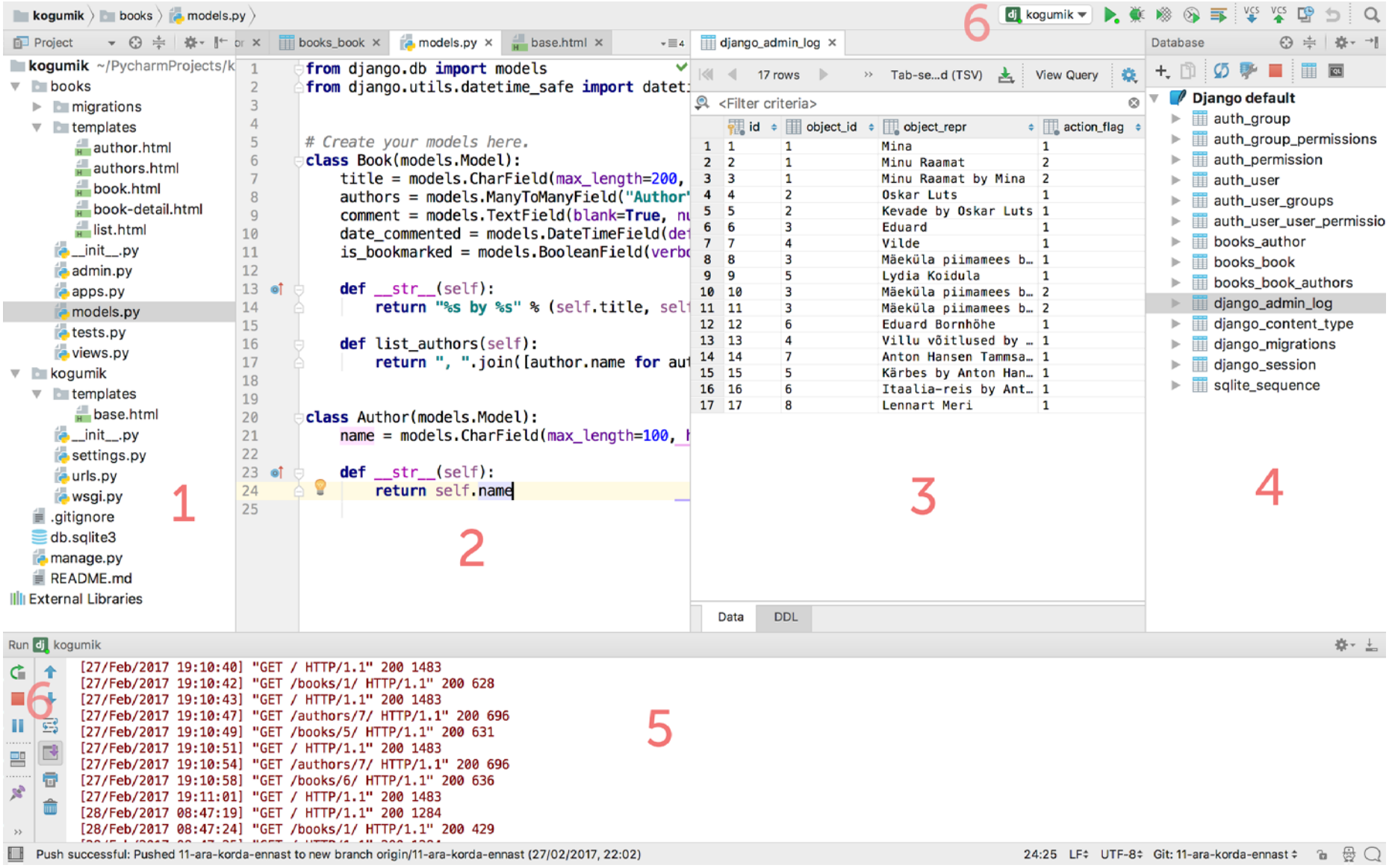
Tõnisson, E. (kuupäev puudub). *Objektorienteeritud programmeerimine*. Loetud aadressil https://courses.cs.ut.ee/2017/OOP/spring/Main/IDEGuides

LISAD

Lisa 1. Python Django populaarsus



Joonis 16. Python Django populaarsus (Google, 2017)


kogumik
books
models.py
Project
books_book
models.py
base.html
django_admin_log
Database
kogumik ~/PycharmProjects/k
books
migrations
templates
author.html
authors.html
book.html
book-detail.html
list.html
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
kogumik
templates
base.html
__init__.py
settings.py
urls.py
wsgi.py
.gitignore
db.sqlite3
manage.py
README.md
External Libraries
1
from django.db import models
from django.utils.datetime_safe import datet
# Create your models here.
class Book(models.Model):
title = models.CharField(max_length=200,
authors = models.ManyToManyField("Author"
comment = models.TextField(blank=True, nu
date_commented = models.DateTimeField(de
is_bookmarked = models.BooleanField(verbo
def __str__(self):
return "%s by %s" % (self.title, self
def list_authors(self):
return ", ".join([author.name for aut
class Author(models.Model):
name = models.CharField(max_length=100,
def __str__(self):
return self.name
2
6
kogumik
17 rows
Tab-se...d (TSV)
View Query
<Filter criteria>
id
object_id
object_repr
action_flag
1 1 1 Mina 1
2 2 1 Minu Raamat 2
3 3 1 Minu Raamat by Mina 2
4 4 2 Oskar Luts 1
5 5 2 Kevade by Oskar Luts 1
6 6 3 Eduard 1
7 7 4 Vilde 1
8 8 3 Mäeküla piimamees b… 1
9 9 5 Lydia Koidula 1
10 10 3 Mäeküla piimamees b… 2
11 11 3 Mäeküla piimamees b… 2
12 12 6 Eduard Bornhöhe 1
13 13 4 Villu võitlused by … 1
14 14 7 Anton Hansen Tammsa… 1
15 15 5 Kärbes by Anton Han… 1
16 16 6 Itaalia-reis by Ant… 1
17 17 8 Lennart Meri 1
3
Data
DDL
Django default
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissio
books_author
books_book
books_book_authors
django_admin_log
django_content_type
django_migrations
django_session
sqlite_sequence
4
Run kogumik
6
[27/Feb/2017 19:10:40] "GET / HTTP/1.1" 200 1483
[27/Feb/2017 19:10:42] "GET /books/1/ HTTP/1.1" 200 628
[27/Feb/2017 19:10:43] "GET / HTTP/1.1" 200 1483
[27/Feb/2017 19:10:47] "GET /authors/7/ HTTP/1.1" 200 696
[27/Feb/2017 19:10:49] "GET /books/5/ HTTP/1.1" 200 631
[27/Feb/2017 19:10:51] "GET / HTTP/1.1" 200 1483
[27/Feb/2017 19:10:54] "GET /authors/7/ HTTP/1.1" 200 696
[27/Feb/2017 19:10:58] "GET /books/6/ HTTP/1.1" 200 636
[27/Feb/2017 19:11:01] "GET / HTTP/1.1" 200 1483
[28/Feb/2017 08:47:19] "GET / HTTP/1.1" 200 1284
[28/Feb/2017 08:47:24] "GET /books/1/ HTTP/1.1" 200 429
5
Push successful: Pushed 11-ara-korda-ennast to new branch origin/11-ara-korda-ennast (27/02/2017, 22:02)
24:25 LF UTF-8 Git: 11-ara-korda-ennast

1 - Loodud rakenduse failide struktuur
2 - Tekstiredaktor
3 - Konkreetses andmebaasi tabelis olevad andmed
4 - Andmebaasi tabelid
5 - Päringulogi
6 - Rakenduse käivitamine, taaskäivitamine, peatamine

Lisa 3. Testimise tagasiside

Tagasiside #1

Antud teema kohta puudus varasem kogemus. Pärast juhendi läbitegemist sain hea sissejuhatuse, kuidas arendada veebirakendust, kasutades Python Django raamistikku. Juhend oli illustratiivne ja põhjalik ning piisavalt kergelt arusaadav inimesele, kellel puudub varasem kogemus. Iga juhendi alamosa alguses oli selgitus, mis on alamosa eesmärgiks ning samuti koodilõikudel olid juures selgitused, mida ja miks tehakse. Funktsioonid olid lahti kirjutatud.

Puuduseks võib pidada veidi kohmakat sõnastust, aga üldjoontes täitis juhend oma eesmärgi ning alguses lubatud eesmärgid olid ka täidetud. Järgmises osas sooviksin näha informatsiooni rohkemate võimaluste kohta.

Kõige rohkem meeldis viimane peatükk, kus pandi rõhku kirjutatud koodi abstraheerimisele ning korduste vältimisele. Antud lähenemine muudab koodi lisandumisel selle kergemini hallatavaks. Kogu juhendi läbitegemiseks kulus 3 tundi koos tarkvara paigaldamise ning seadistamisega.

Tagasiside #2

Veebiraamistike aine raames oli mul au Konstantini meeskonna liikmena käesolevas seminaritöös valminud juhendit klassi ees samm-sammult ette kanda. Juhendit õnnestus hoolimata esinemispingest edukalt jälgida. Ühes kohas tekkis väike segadus, kus punktid olid vales järjekorras, kuid see probleem sai hiljem lahendatud. Üldiselt jäi mulle mulje hästi ja piisavalt detailselt paika pandud juhendist.

Tagasiside #3

Konstantin Tenman-i lühiõpetus jättis positiivse mulje. Õpetust on lihtne jälgida – kogu õppeprotsess on jaotatud sammudesse, mis on loogiliselt hästi järjestatud. Koodinäited on hästi lahti seletatud ja illustreeritud. Kuigi mul puudus varasem kogemus Django raamistikuga, siis õpetuse järgimisega mul tehnilisi tõrkeid ei tekkinud ja kõik oli arusaadav. Sain kasulikke teadmisi antud raamistikust ja mul tekis huvi selle edasi õppimiseks. Tulevikus, kui autoril tekkib soov seda teemat edasi arendada, soovitaksin kasutada suuremaid väljakutseid pakkuvaid näiteid.

Tagasiside #4

Minu arvates oli juhend piisavalt detailne ning seda oli mõnus järgi teha. Kui asja teha mõistusega, siis juhend on kindlasti jälgitav. Oli väga palju abi sellest, et minu valmiva rakenduse

koodi sai võrrelda Githubis oleva koodiga. Meeldis seminaritöö teoreetiline osa, see aitas mõista, mis Django sees toimub.

Lisaks täheldasin, et Djangos on üsna palju maagiat. Näiteks peale mitu-mitmele seose tegemist tekib andmebaasi veel üks tabel, kus on kirjeldatud suhted autorite ja raamatute tabelite vahel.

Tagasiside #5

Kindlasti on positiivne, et viidatud on poolikutele rakenduse variantidele. Et kui ikka järg käest läheb ja vigu enam üles ei leia, on võimalik edasi liikumiseks mingi puhas ja töötav versioon endale saada.

See, kuhu faili midagi lisada, on valdavalt jälgitav. Ehk oleks abiks ka mingi üldisem skeem, millised failid lõpuks tekkisid ja kus nad on (sarnaselt algse kataloogi sisu pildile).

Minu muljeks juhendist on, et liiga vähe on sisulisi selgitusi, mis kasutatud laused, käsud jms natuke laiemalt lahti seletaks.