TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Eerik Kodasma 185270IADB

Django baasprojekti koostamine ettevõttele Flowit Estonia OÜ

Bakalaureusetöö

Juhendaja: Einar Kivisalu
Magistrikraad

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Eerik Kodasma

15.05.2023

Annotatsioon

Lõputöö eesmärgiks oli luua Flowit Estonia OÜ-le Django baasprojekt, mida saab võtta tulevaste projektide arendamise aluseks. Lisaks oli veel eesmärk antud baasprojekti analüüsida teiste ettevõtte projektidega ning välja selgitada baasprojekti eeldatava kasulikkuse tulevastes projektides. Antud lõputöö lahendus annab ettevõtte tulevastele projektidele ühtse struktuuri, projektide kiiremat arendamist ning ajakulu kokkuhoiu.

Baasprojekt on kokkupandud lähtetingimustest tingitud tehnoloogiatest ja integratsioonidest, mis said välja valitud ettevõtte projektidest ning ka internetist. Baasprojektis kasutatud tehnoloogiad ja integratsioonid said arendatud ilma äriloogikata ja mis on tehtud projektis eraldi iseseisvateks Django rakendusteks.

Autor analüüsis valmis arendatud baasprojekti kasulikkust teiste ettevõtte projektide suhtes, arvutades välja ajakulu kokkuhoiu kasutades baasprojekti ning sellega saades eeldatava kasulikkuse tulevastes projektides. Analüüsimiseks sai võetud 6 projekti, mida valiti projektimahu ja uudsuse järgi. Autor sai tulemuseks, et baasprojekt kasulikkus nendes projektides oleks olnud kokku 26 tööpäeva ning keskmiselt ~4,3 tööpäeva, millest võib eeldada, et baasprojekti kasulikkus tulevastes projektides on samuti vähemalt 4,3 tööpäeva.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 6 peatükki, 8 joonist, 1 tabel.

Abstract

Creation of Django Base Project for Company Flowit Estonia OÜ

The purpose of this thesis was to create a Django base project for Flowit, which can be used as a basis for the development of future projects. In addition, the goal was to analyze this base project with other Flowit projects and to find out the expected effectiveness of the base project in future projects.

Thesis aims to solve Flowit future projects by giving them a unified structure, faster development of projects and saving time. This is achived by following Django good practises and using key features.

The base project is developed from the technologies and integrations described by the initial conditions. These were selected from Flowit projects and also from the internet. The technologies and integrations developed in the base project were developed without any business logic and made into independent Django applications in the project.

The author analyzed the effectiveness of the developed base project in relation to other Flowit projects, calculating the time savings using the base project and thereby obtaining the expected effectiveness in future projects. 6 project was taken for analysis, which was chosen according to the project volume and novelty. The author concluded that the usefulness of the base project in these projects would be a total of 26 working days and an average of ~4,3 working days. The author can assume that the effectiveness of the base project in future projects is at least 4,3 working days.

The thesis is in Estonian and contains 26 pages of text, 6 chapters, 8 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakenduseliides.
DRF	*Django REST Framework*, Django rakenduse moodul. Võimaldab Djangot kasutada tagarakendusenda.
DRY	*Don't Repeat Yourself*, ära korda ennast.
HTML	*HyperText Markup Language*, veebilehe struktuuri koostamise keel.
HTTP	*Hypertext Transfer Protocol*, võrguprotokoll.
LTS	*Long-Term Support*, pikkaajaline tugi.
MVT	*Model - View – Template*, rakenduse arhitektuurimuster mudel-vaade-mall.
ORM	*Object Relational Mapping*, Objekt-relatsiooniline kaardistamine.
PEP 8	Python koodi stiili ja parimate tavade juhis.
pre-commit hook	Enne failihaldussüsteemi üleslaadimist käivituv fail.
REST	*Representational State Transfer*, tarkvara arhitektuuriline stiil rakendusliidesele.
SQL	*Structured Query Language*, andmebaaside suhtlus keel.
URL	*Uniform Resource Locators*, internetiaadress.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Flowit Estonia OÜ (edasipidi Flowit) on tarkvaraettevõte, mille peamine tegevusala on infotehnoloogia süsteemide arendus. Lõputöö koostaja on aktiivne arendaja ning tegeleb erinevate arendusprojektidega ettevõttes. Flowiti projektid on enamus ülesehitatud Django raamistiku abil. Sellest lähtudes avastas lõputöö koostaja ettevõttes probleemi ning nii sündis lõputöö idee.

Lõputöö käsitletavaks probleemiks on, et iga Django projekti alustamine nõuab arendamist otsast peale. Selleks kulutavad igakord arendajad aega, et koostada projekti alusstruktuur, mis on kasutuses pea igas projektis.

Baasprojekti eesmärk on kiirendada ettevõttes põhitehnoloogiate arendust ja dokumenteerimist uutes projektides ning lisada uusi abistavaid tehnoloogiaid (lähtuvalt äriloogikast), mis annaksid projektidele kaasaegsemaid lahendusi. Eesmärgi saavutamiseks koostab autor Django baasprojekti, mis oleks iga uue projekti aluseks.

Lähtetingimusteks oli analüüsida olemasolevaid projekte. Selle raames tuvastati projektides ühisosa, mis on läbivalt sarnane erinevates projektides. Need ühisosad on lahutatud äriloogikast ning kirjutatud võimalikult universaalselt, et oleks võimalik kasutada uutes projektides. Lisaks sellele on uuritud häid tavasid, parendamaks turvalisust ja hallatavust.

Lõputöö eesmärgiks on välja arendada baasprojekt ning välja uurida selle eeldatavat kasulikkust Flowitis. Selleks analüüsib autor ettevõtte projekte kui ka interneti ning uurib, mis tehnoloogiad või integratsioone tuleb baasprojekti arendada. Antud tulemust uurib autor baasprojekti kasulikkusest lähtudes väljavalitud ettevõtte projektides ning selle abil saab eeldatava baasprojekti kasulikkuse tulevastes projektides.

Lõputöös antakse esmalt ülevaate Flowitist ja nende probleemi kirjeldamisest ning Django raamistikust. Teiseks, kirjeldab autor baasprojekti kuuluvaid tehnoloogiaid, integratsioone ning häid tavasid ning milleks neid kasutatakse. Lõputöö lõpus kirjeldatakse valmis saanud projekti tulemust ja edasisi arendamisplaane.

2 Projekti olemus

Enamasti iga arendusprojekti arendamise põhjus langeb mingi probleemi lahendamise peale ning sellega tekibki projekti eesmärk. Enne projekti arendamist tuleb selgeks teha projekti lahendatava/lahendatavad probleemi/probleemid, projekti lähtetingimused ja/või lahendatavad raskused ning ka projekti eesmärk ja lahendused.

Lõputöö autor kirjeldab antud peatükis lähemalt lahti lõputöö olemust, kus kirjeldab lähemalt millised ettevõtte probleemid baasprojekt lahendab, mis baasprojekti lahendatav tulemus on ning lisaks annab ülevaate baasprojekti lähtetingimustest.

2.1 Probleemipüstitus

Tänu klientide olemusele on Flowitil palju projekte, milles on märkimisväärne ühisosa. Enamus projektid Flowitil on Django raamistikus arendatud ning iga projekti arendamist on alustatud otsast peale. See on üpris tüütu ja korduv tegevus ning Flowit soovis lahendust.

Seniste ettevõtte Django projektide probleemideks on:

- ühtse projekti struktuuri puudumine;
- projekti alustamise korduvate tegevustele kuluv aega oli suur;
- kasutatavate tehnoloogiate ja integratsioonide lähteseadistustele kuluv aeg.

Flowiti mõte oli koostada alusprojekt (mall), mis oleks baasiks teistele projektidele. Baasprojekt sai alguse probleemide välja selgitamisega ja neile lahenduse variantide otsimisega, mis suudaks lahendada ülalpool nimetatud probleeme.

2.2 Baasprojekt kirjeldus

Tihti funktsionaalsuse kirjutamisel tuvastame olukordi, kus saaksime rakendada DRY (*Don't Repeat Yourself*) printsiipi läbi projektide, mis tugineksid ettevõte tehnoloogia valikutele (Django, Python, PostgresSQL). See tähendab integratsioonide ja laialt kasutavate funktsionaalsuste universaalseks muutmine, milles ei peituks äriloogikat.

Sellise lähenemisega saavad arendajad keskenduda äriloogika juurutamisele, mitte funktsionaalsuse ümberkirjutamisele. Kasutades baasprojekti muutuksid hinnangud kliendile täpsemaks ja on võimalik konkurentsivõimelisemaid hinnapakkumisi koostada.

2.3 Projekti eesmärk

Lõputöös käsitletud projekti eesmärgiks oli koostada Django baasprojekt, mida saab ettevõtte teistes projektides aluseks võtta.

Baasprojekt annab uutele projektidele:

- ühtse struktuuri;
- arendamiskulu(tundide) kokkuhoiu, võimaldades kiiremat arendamisprotsessi.

Lisaks projekti eesmärgi täitmisele, saavutatakse ka kaudseid tulemusi:

- projekti hinnangud muutavad kiiremaks, mis aitab ettevõttel kliendile täpsemaid hinnapakkumisi koostada;
- väiksema arendamiskulu tõttu muutuvad hinnangud konkurentsivõimelisemaks, seeläbi ka kliendile soodsamaks.

2.4 Projekti lähtetingimused

Enne baasprojekti arendamise alustamist oli vaja koostada baasprojektile lähtetingimused. Lähtetingimused annavad projektile piirangud, et baasprojekti kasutamine oleks kasulik ja efektiivne.

Lähtetingimused baasprojektis kasutatud tehnoloogiatele ja integratsioonidele:

- peavad olema universaalsed, et uutes projektis saaks äriloogikat neile lihtsasti lisada;
- peavad olema potentsiaalselt kasutatavad erinevates projektides;
- peavad andma märkimisväärse ajavõidu.

3 Django raamistik

Django on Pythonil põhinev, avatud lähtekoodiga veebirakenduse raamistik. Raamistik loodi aastal 2003, meeskonna poolt, kes tegelesid ajalehtede veebirakenduste loomisega ja hooldamisega ning hakkasid kasutama taaskasutatavaid koodijuppe, mille tulemusel pandi kokku raamistik nimega Django [1]. Django populaarsus on tagatud sisseehitatud funktsionaalsustele, millega lihtsustada programmeerijate igapäevast arendamist.

Djangot kasutavad paljud tuntud suurettevõtted nagu Instagram, Mozzila, Spotify, Pinterest, Bitbucket ja teised [2].

Django arendamine baasprojektis võttis autoril aega umbes üks tööpäev.

Lõputöö autor kirjeldab antud peatükis Django olemust ning veebilehe serveerimiseks vaja minevaid komponente. Lisaks on kirjeldatud Django omadusi lähtudes baasprojektist ning versioonidest.

3.1 Django omadused

Djangoga on võimalik ehitada nii väikseid kui ka suuri projekte ning erinevaid tüüpi rakendusi.

Django omadused:

- Django hoolitseb tavaarenduses kasutatava raskete korduvate ning aeganõudvate koodi arenduste eest ise ning jätab äriloogika arendamise arendajale, sellega kiirendades prototüüpide või projektide arendamist;
- Django rakendused on turvalised tänu sisseehitatud kaitsele, mis kaitseb rakendusi tuntud turvanõrkuste eest;
- Django printsiip on DRY, mille tõttu rakendused on lihtsasti skaleeritavad ja hooldatavad tänu koodi komponentide eraldusele ja iseseisvusele [1].

3.2 Django arhitektuur

Django on üles ehitatud MVT (M*odel* - *View* - *Template*) mustril, mis hoiab koodiloogikad eraldatud juppidena ning see võimaldab töötada projektis suurte tiimidena, ilma, et arendajad segaksid üksteise koodi/tööd. Veeblilehte serveerimiseks ja kokku panemiseks koosneb koostööst URL-st (*Uniform Resource Locators*), vaatest, mudelist ja mallist.

3.2.1 Päringu kaardistamine

URL Djangos on parameetritega ette defineeritud sõne, mille põhimõte on päringu kaardistamine õige vaatega. Kasutaja päringu ajal otsib Django ükshaaval päringule vastavat URL-i, mille tulemusel näidetakse vastavat vaadet või viga, mitte leidmisel. Djangos tavapäraselt hoitakse URL-e urls.py failis. Joonisel 1 on kujutatud üks näide baasprojekti URL-st.

```
urlpatterns = [
    path(
        "callbacks",
        csrf_exempt(views.EveryPayCallbackView.as_view()),
        name="everypay-callback",
    ),
]
```

Joonis 1. Baasprojekti URL näide (autori joonis).

3.2.2 Äriloogikaga sidumine

Djangos on vaade koht, kus asub äriloogikaga seotud kood. Vaates käsitletakse mudeleid ning pannakse andmed valmis, et saaks saata pärijale tagasi päringu vastuse koos malliga. Djangos tavapäraselt hoitakse vaateid views.py failis. Joonisel 2 on kujutatud üks näide baasprojekti vaatest.

```
class EveryPayCallbackView(View):
    template_name = "everypay-callback.html"

    def get(self, request):
        """Returns paginated page and all EverpayCallback objects."""
        everpay_cb = EverypayCallback.objects.all()
        paginator = Paginator(everpay_cb, 10)  # Show 25 contacts per
page.
        page_number = request.GET.get("page", 1)
        page_obj = paginator.page(page_number)

        return render(request, self.template_name, {"everpay_cb":
everpay_cb, "page_obj": page_obj})

    def post(self, request, *args, **kwargs):
        """Creates EverypayCallback object."""
        try:
            api_username =
self.request.query_params.get("api_username", None)
            payment_reference =
self.request.query_params.get("payment_reference", None)
            order_reference =
self.request.query_params.get("order_reference", None)
            event_name = self.request.query_params.get("event_name",
None)
        except AttributeError:
            return JsonResponse(data={"error": "incorrect query
params"}, status=406)

        try:
            callback = EverypayCallback.objects.create(
                invoice_reference=order_reference,
                payment_reference=payment_reference,
                status=event_name,
                request=request.data,
            )

        except:
            logger.error(traceback.format_exc())
            return Response(status=status.HTTP_404_NOT_FOUND)
        return Response(status=status.HTTP_200_OK)
```

Joonis 2. Baasprojekti vaate näide (autori joonis).

3.2.3 Andmebaasi mudel

Mudel on üksühene vaste andmebaasi tabelile ning vastutab kõigi andmetega seotud ülesannete käsitlemise eest. Andmebaasi tabeli koostamiseks tehakse uus mudeli klass ning klassi alla kirjeldatakse tabeli väljad ja nende tüübid koos kitsendustega ning mudelite vahelisi seoseid. Djangos tavapäraselt hoitakse mudeleid models.py failis.

Tavapäraselt kasutab Django andmebaasiga suhtlemiseks SQL (*Structured Query Language*) asemele ORM (*Object Relational Mapping*) lahendust. ORM kergendab andmebaasi suhtlust, ilma, et kasutaja peaks tundma hästi SQL keelt [3]. Joonisel 3 on kujutatud üks näide baasprojekti mudelist.

```
class EverypayCallback(BaseModel):
    """Everypay callback model."""

    # Value defined by merchant or entered by customer
    invoice_reference = models.CharField(max_length=255, null=False,
blank=False)
    # Payment reference ID
    payment_reference = models.CharField(max_length=255, null=False,
blank=False)
    # Current status of the payment. Possible scenarios:
    # 'settled' or 'authorised' (successful payments)
    # 'cancelled' or 'waiting_for_3ds_response' (cancelled/incomplete
payments)
    # 'failed' (failed payments)
    status = models.CharField(max_length=100, null=False, blank=False)
    # Request data in JSON
    request = models.JSONField(null=True, blank=True,
encoder=DjangoJSONEncoder)
```

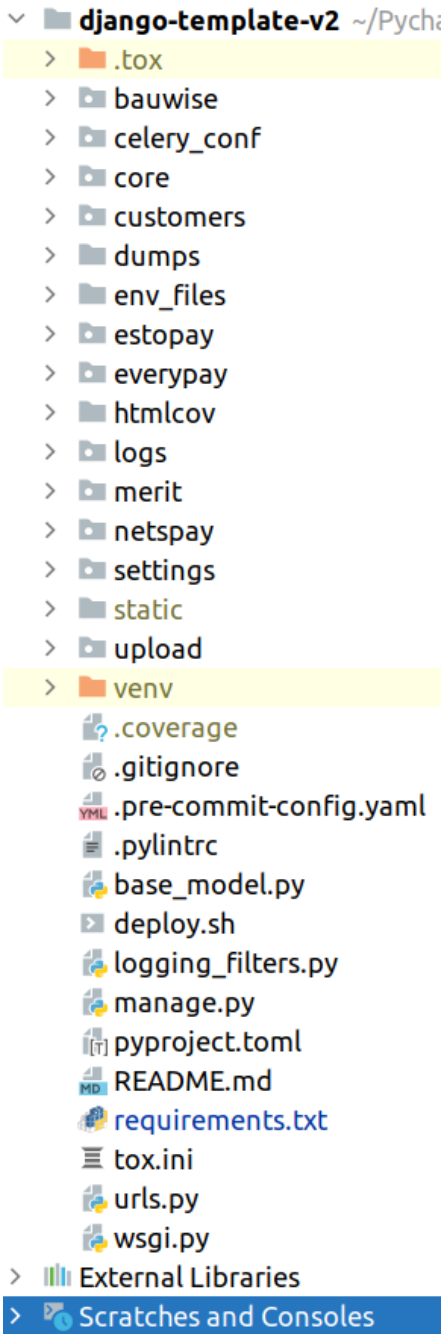
Joonis 3. Baasprojekti mudeli näide (autori joonis).

3.2.4 Veebilehe paigutuse mall

Mall on HTML-kood (*HyperText Markup Language*), mis kirjeldab veebilehe paigutust, kui ka vaatest kaasa tulnud andmete käsitlemist. Django kasutab mallis andmete käsitlemiseks eripärast süntaksit. Djangos tavapäraselt hoitakse malle eraldi kaustas ning on *.html* failitüüpi. Lisa 2 on kujutatud üks näide baasprojekti mallist.

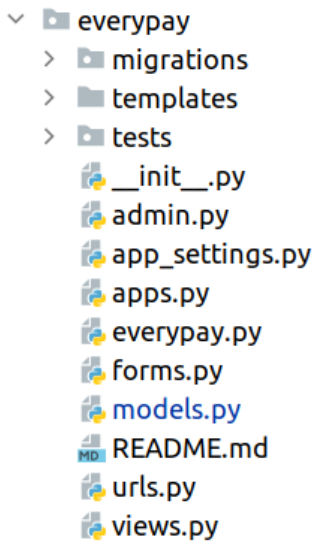
3.3 Django komponendid

Django projektid on arhitektuuriliselt samamoodi sarnaselt ülesehitatud. Projekti struktuur genereeritakse Djangos välja vastavate käskudega. Seda on kirjeldatud raamatus „Two scoops of Django 3.x“ [4]. Joonisel 4 on kujutatud baasprojekti ülesehitus.



Joonis 4. Baasprojekti projekti struktuur (autori joonis).

Django pooldab loogikalahusust, võimaldades loogilisi komponente hoida iseseisvates rakendustes (*Django app*). Seda võimalust kasutades on koodi haldamine lihtsam ja äriloogikat saab hoida rakenduse sees. Joonisel 5 on kujutatud baasprojekti ühe rakenduse ülesehitus.



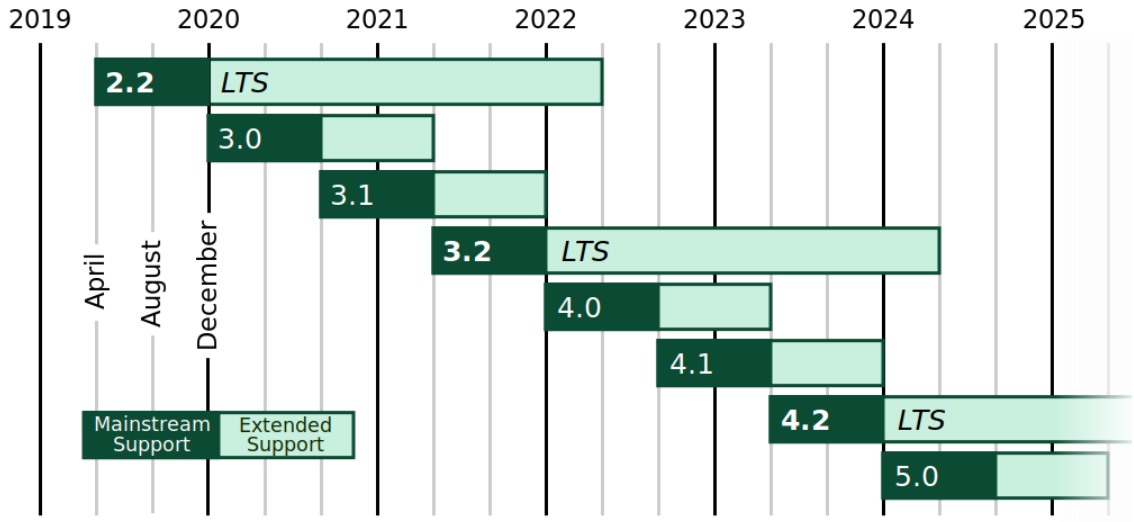
Joonis 5. Baasprojekti ühe rakenduse struktuur (autori joonis).

Olemasolevatele Django projektidele on võimalik kergelt luua, eemaldada või lisada juba valmis tehtud rakendusi, et ei lõhuks tervik projekti töötamist. Baasprojektis sai otsus tehtud sellel põhimõttel, et kõik tehnoloogiad ja integratsioonid oleks eraldi väiksemad rakendused. Selline lahendus võimaldab igas projektis lihtsasti erinevaid tehnoloogiaid ja integratsioone eemaldada, mida ei ole parajasti tarvis kasutada.

3.4 Versioonid ja LTS

Tihti valmis saanud arendusprojektid ei jää lõppversioonideks ning tulevikus tehakse lisaarendusi juurde. Nii ka Djangos on ajaliselt eraldi versioonid, et arendajad saaksid tekkinud vigu parandada, teha mingit aspekti paremaks või arendada uusi funktsionaalsust juurde.

Django versioonid enamasti ühilduvad eelmise/järgmise versiooniga. Uus versioon avaldatakse umbes iga 8 kuu tagant, millele kehtib 8 kuud aktiivne tugi ning selle järgneb 8 kuud turvatugi. Lisaks on Djangol ka LTS (*Long-Term Support*) versioonid, mis tulevad iga 2 aasta tagant, ning nende mõte on hoida Django ühte versiooni stabiilsemalt koos pikendatud toega, millel on tagatud turva- ja andmekaotugi kuni 3 aastat [5]. Joonisel 6 on kujutatud versioonide avaldamise aegu ning kasutajatoe pikkust.

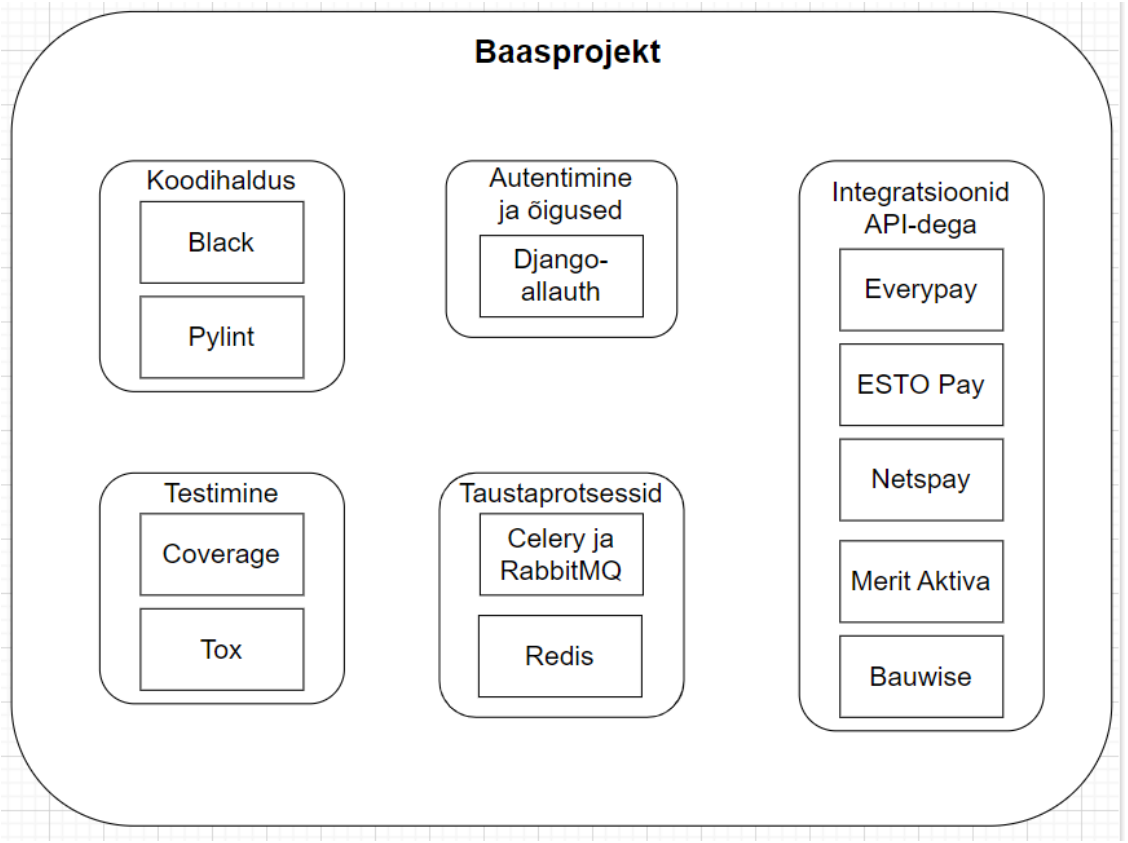


Joonis 6. Django versioonide toe eluiga [6].

4 Kasutatud tehnoloogia, integratsioonid ja head tavad

Baasprojekti tehnoloogia valik tuli vastavalt lähtetingimustest. Esmalt tuli uurida teistest Flowiti Django projektidest, mis tehnoloogiad on levinumad ehk universaalsed, et analüüsida, mis tehnoloogiad võiksid olla baasprojektis algselt juba olemas. Sellega kulutaksid arendajad vähem aega, ilma et neid igakord uuesti projektis teostama peaksid.

Teiseks, oli vaja uurida internetist tänapäeva Django projekti tehnoloogiate lahendusi või tavasid. Uute tehnoloogiate kasutamine baasprojektis muudab projekti aktuaalsemaks ja kaasaegsemaks ning säilitaks projekti relevantsuse. Lisaks muudavad uued lahendused või tavad koodi kontrollitavamaks ning aitavad arendajatel hoida koodi ühtlasena ja muuta koodi kirjutamist lihtsamaks. Joonis 7 annab ülevaate baasprojekti arendatud tehnoloogiatest ja integratsioonidest.



Joonis 7. Baasprojekti komponendid (autori joonis).

Lõputöö autor kirjeldab antud peatükis baasprojekti valitud tehnoloogiatest ja integratsioonidest, nende olemusest ning milleks Flowit seda kasutab ja kui suur ajavõit saavutatakse.

4.1 Koodihaldus

Enne koodi üleslaadimist koodivaramusse tuleks igal arendajal oma kood üle kontrollida, et kood oleks korrektne ja ei sisaldaks üleliigset, mida ei peaks koodis olema. Selleks tuleb arendajal manuaalselt kõik muudatused oma koodis üle käia ning kontrollida selle korrektsust, vastavust projekti stiili reeglitega, kõik see nõuab aega.

Tihti on projektides olemas koodi stiili vormindajad või tehnoloogiad, mida arendamisel kasutatakse. Selleks, et arendaja ei peaks käivitama need manuaalselt ning ei tekiks olukord, kus nende kasutamine ununeb, on võimalik need rakendada *pre-commit hook*'dena, mis käivituvad iga kord automaatselt koodi üles laadimisel koodivaramusse. Selleks tehti baasprojekti konfiguratsioon fail *pre-commit-config.yaml* (Joonis 8), kus peenhäälestatakse *hook*'id ja nende järjekord.

```
default_language_version:
  # default language version for each language
  python: python3.8
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.1.0
    hooks:
      # See https://pre-commit.com/hooks.html for more hooks
      - id: check-ast
      - id: check-byte-order-marker
      - id: check-case-conflict
      - id: check-docstring-first
      - id: check-executables-have-shebangs
      - id: check-json
      - id: check-yaml
      - id: debug-statements
      # - id: detect-aws-credentials
      - id: detect-private-key
        exclude: test_public.key, test_private.key
      - id: end-of-file-fixer
      - id: trailing-whitespace
      - id: mixed-line-ending
  - repo: https://github.com/psf/black
    rev: 22.1.0
    hooks:
      - id: black
  - repo: local
    hooks:
      - id: pylint
        name: pylint
        entry: pylint
        language: system
        types: [ python ]
```

Joonis 8. Baasprojekti pre-commit-config.yaml faili sisu (autori joonis).

4.1.1 Koodivormindaja Black

Suur arendajate hulk projektis oma arendustavadega, tekitab ühtse koodistiili hajumist, mis halvendab arendajatel koodi arusaamist ja haldamist. Selleks võttis autor antud projektis kasutusele Black. Black on avatud lähtekoodiga Pythoni PEP 8-ga ühilduv koodi stiili vormindaja, mille üldisemad reeglid on välja toodud raamatust „Effective Python“ [7] [8].

Kuna baasprojektis kasutatav Django on Pythoni raamistik, siis tuleb lähtuda Pythoni koodi stiili korraldustest. Tänu Black tehnoloogiale peavad arendajad vähem aega panustama stiili vorminduse jälgimisele ning selle asemel saavad keskenduda

funktsionaalsuse arendamisele. Baasprojektis on lisatud Black *pre-commit hook*'ina, et see käivituks automaatselt ning parandaks arendajate koodistiili iga kord. Black'i juurutamine baasprojekti kulus autoril aega umbes pool tööpäeva.

4.1.2 Stiilivormindaja Pylint

Nagu ka Black'i kasutuse vajadus on tingitud arendajate harjumustest, nii tekkis ka vajadus baasprojekti lisada Pylint. Pylint on Pythoni staatilise koodi analüüsimise rakendus [9].

Pylint aitab koodi analüüsida arendamisel ning selle põhimõtted on:

- tuvastada koodi kirjutamisel tekkinud süntaksivigasid või -rikkumisi;
- anda soovitusi, kui ka teostada koodimuudatusi vastavalt headele koodi kirjutamise tavadele [9].

Pylint on täielikult kohandatav rakendus, kus konfiguratsiooni failis saab määrata vastavatele muutajatele väärtused, mis muudavad analüüsimise käitumist. Konfiguratsiooni fail on nimetusega *pylintrc* ja asub projekti juurkaustas. Pylint'i arendamine baasprojekti kulus autoril aega umbes pool tööpäeva.

4.2 Taustaprotsessid

Ajamahukad protsessid mõjutavad kasutajamugavust, hoides kasutajat nii-öelda kinni vastust (*response*) oodates ja aeglustavad rakenduse kasutamise voogu. Sellistel olukordadel oleks mõistlik kasutada taustaprotsesse, mis võimaldavad kasutaja eraldada päringu vastuse ootamisest ning tõsta sellega kasutajamugavust. Taustaprotsessid kasutavad suhtlemiseks sõnumeid (*message*), mis asuvad sõnumite järjekorras (*message queue*).

Taustaprotsessi omadused:

- hajutab rakenduse koormust, jaotades tööd (*tasks*) erinevate järjekordades (*queue*);
- lubab kasutajal samal ajal rakenduse teisi funktsionaalsusi kasutada;
- võimaldab ebaõnnestunud töö ülesannet korrata.

4.2.1 Celery ja RabbitMQ tutvustus

Tänapäeva kasutajad soovivad saada kiiret ja kohest vastust rakendustelt, kuid mahukate päringute puhul tuleb kasutajal kauem oodata. Selle põhjuseks on, et rakenduse sisesed päringud toimuvad sünkroonselt ehk järjestikuselt. Üks päringu kiiruse tõhusamaks tegemise lahendus oleks kasutada Celery't.

Celery töötab ülesannete järjekorra (*task queue*) põhimõttel, mida kasutatakse töö asünkroonseks (*asynchronous*) täitmiseks väljaspool HTTP (*Hypertext Transfer Protocol*) päringu (*request*) ja vastuse (*response*) ajal [10]. Celery kasutab niinimetatud töötajaid (*worker*), mis ootavad ja täidavad etteantud ülesandeid. Suhtlus käib Celery'l läbi sõnumite (*message*), kasutades tavaliselt mingit sõnumisidevahendajat (*message broker*). Rakenduse sõnumid hoitakse sõnumite järjekorras (*message queue*). Celery on skaleeritav vastavalt projekti vajadusele, alustades töötajate ja sõnumisidevahendajate arvu suurendades [11].

Selleks, et rakenduselt tulnud ülesandeid saaks hallata, tuleb kasutada vastavat sõnumisidevahendajat. Üks populaarsemaid on RabbitMQ mida kasutatakse tihti Celery'ga. RabbitMQ on eraldi rakendatav tarkvara nii serveris kui ka pilves. RabbitMQ haldab sisse tulnud sõnumeid ning määrab need vastavalt sõnumite järjekorda, kus sellega edasi tegeletakse [12].

Baasprojekt kasutab Celery't, et hoida aega nõudvaid päringuid asünkroonselt, seeläbi parendades kasutajamugavust ning hallata rakenduse päringuid tõhusamini. RabbitMQ'd on enamasti Flowitis kasutatud Celery'ga koos nii on ka samamoodi arendatud mõlemad baasprojekti. Celery'i ja RabbitMQ arendamine baasprojekti kulus autoril aega umbes üks tööpäev.

4.2.2 Redis tutvustus

Rakenduse töökiirus on üks paljudest kvaliteetse tarkvara omadustest. Loogika optimeerimine on võimalik meetod töökiiruse parendamiseks, kuid ei pruugi säästa piisavalt andmebaasi pärimisest. Redis't kasutatakse selleks, et andmebaasi vähem koormata kulukate või tihedate päringute eest. Redis rakendus kasutab serveri vahemälu, et hoida etteantud päringuid mälus kiire HTTP tagastamise põhimõttel. Redis'e serveri vahemällu salvestatakse andmebaasi päringud võti-väärtus kujul.

Redist on võimalik kasutada ka sõnumisidevahendajana, kuid võivad tekkida selle kasutamisega ka probleemid, kuna Redis on limiteeritud. Et Redis muuta sõnumsidevahendajaks nõuab see rohkemat arendamist. Lisaks on suurte andmekogustel Redis'el takistusi sõnumeid järjestiku paigutada, mis võib suurenda andmekao võimalust ning Redis ei toeta automaatset uuesti proovimist [13].

Baasprojektis kasutatakse Redis't andmete vahemälus hoidmise ning kiire andmete pärimise tõttu. Autoril kulus Redis'e juurutamisele aega umbes üks tööpäev.

4.3 Integratsioonid API-dega

Tavaliselt mahukate projektide arendamisel tuleb projekti kasutusse võtta erinevaid lahendusi. Lahendused on teiste poolt arendatud projektid, mis lahendavad mingi valdkonna probleemi. Arendajatel on võimalik antud lahendusi juurutada enda projektidesse, sellega säästa aega olemasolevate lahenduste ise kirjutamisest ning laseb keskenduda projekti äriloogika kirjutamisele. Lahenduste juurutamine käib läbi liidestuste, mis suhtlevad läbi API (*Application Programming Interface*), ning pärivad ja edastavad andmeid omavahel.

Enamuse ettevõtete äri keskendub toodete või teenuste müümisele. Nii on ka tihti Flowiti projektides vaja kasutada maksetega seotud lahendusi. Ise selle arendamine võib olla keeruline, aeganõudev ja kulukas. Võetakse projektides kasutusse olemasolevad makselahendused, mis võtavad kõik maksetega seotud ülesanded enda peale. Makselahendusi on erinevaid ja iga projekt nõuab vastavalt ärimudelile erinevat makselahendust.

Flowiti enamus projekte on põhiliselt kas Eesti klienditele või naaberriikide omadele. Seetõttu kasutab Flowit peamisteks makselahendusteks Everypay-d, ESTO Pay-d, Netspay-d. Ettevõttest sõltuvalt kasutakse lisaks ka Merit Aktiva ja Bauwise lahendusi.

4.3.1 Everypay tutvustus

Everypay on pilvepõhine digitaalne makselahendus Balti regioonis, mis peale makseteenuse ka haldab ettevõtete kaardiandmeid ja pangamakseid. EveryPay-l on suured turvanõuded ja kaasaegne kasutajaliides ning toetab kaardimakseid,

pangaülekandeid ja PayPali makseid ning lisaks aitab teha digitaalseid makseid mugavamaks [14].

Baasprojekti sobib Everypay kuna pakub mitmeid maksevõimalusi ning opereerib Balti regioonis, kus enamus Flowiti projektide klientuur asub. Everypay liidestuse arendamine baasprojekti kulus autoril aega umbes üks tööpäev.

4.3.2 ESTO Pay tutvustus

ESTO Pay on Baltikumi ja Soome pankade makselahendus [15]. ESTO Pay võimaldab oma klientidele maksta pangalinkide kaudu või pangakaardiga, ning pakub klientidele võimalust jagada makseid osadeks, osta kohe, aga maksta hiljem või osamaksetena maksta [16].

ESTO Pay sobib baasprojekti, kuna ESTO Pay katab baltikumi riikide ning lisaks veel Soome makselahendusi. ESTO Pay liidestuse arendamine baasprojekti kulus autoril aega umbes üks tööpäev.

4.3.3 Netspay tutvustus

Netspay on maksekaardi maksmise lahendus. Netspay pakub turvalist pankade ja kliendi vahelist suhtlemist. Flowit kasutab Netspay-d krediitkaardi maksete vastuvõtmiseks. Netspay liidestuse arendamine baasprojekti kulus autoril aega umbes kaks tööpäeva.

4.3.4 Merit Aktiva tutvustus

Merit Aktiva on raamatupidamisprogramm, mis on Eesti üks populaarsemaid, see on varustatud paljude API otspunktidega, mis aitavad liidestada projekte raamatupidamise funktsionaalsustega [17].

Merit Aktiva liidestust komponent on väga vajalik paljudes uutes projektides. Selle arendamiseks baasprojekti kulus autoril aega umbes kaks tööpäeva.

4.3.5 Bauwise tutvustus

Bauwise on projektihaldustarkvara, mis aitab projektide seotud kulutusi hallata kiiresti ja kergelt. Bauwise’t annab hea ülevaate andmetest ning on kerge juurutada arendustesse [18].

Flowitis on kasutatud Bauwise’t eesmärgil ehitusprojektide staatuse ja kulude pärimiseks ning nende põhjal automaatsete raamatupidamiskannete genereerimiseks. Bauwise lisamiseks baasprojekti kulus umbes pool tundi, kuna Bauwise nõuab palju äriloogikat, mida baasprojektis kirjeldada ei saa.

4.4 Autentimine ja õigused

Kuigi Djangol on sisseehitatud autentimise süsteem, mis tegeleb kasutaja andmete autentimisega, õigustega ja gruppidega, jääb sellest tihtipeale ikkagi puudu. Tänapäeval ei piisa ainult rakenduse enda kasutaja loomisest ning nõuab erinevaid lahendusi, lisaks vajavad rakendused tugevat kaitset kasutaja loomisel ja sisse logimisel.

4.4.1 Django-allauth tutvustus

Django-allauth on Django autentimise süsteemi laiendus, mis teeb paremaks tavalist Django autentimise süsteemi. Django-allauth tegeleb kasutaja autentimisega, registreerimisega, haldamisega ning lisab erinevaid sotsiaalmeedia autentimisi võimalusi juurde [19].

Flowit kasutab django-allauth-i, et parendada Django tava autentimist ning lisada oma projektidele tänapäevaseid häid autentimise lahendusi. Selle arendamiseks baasprojekti kulus umbes üks tööpäev.

4.5 Testimine

Testimine on iga suures projektis vajalik korrasolu kontrollija. Testimine on jaotatud enamjaolt kahte gruppi: manuaaltestimine ja automaattestimine. Testimise mõte on kontrollida arendamise käigus tehtud tööde korrasolu, et arendusprojekt töötaks nii nagu peaks.

Baasprojekti pole võimalik teste ette kirjutada, kuna testid on erinevad igas projektis, küll aga baasprojekti arendati testimist abistavaid tehnoloogiaid. Nendeks on Coverage ja Tox.

4.5.1 Coverage tutvustus

Heade testide kirjutamine nõuab head oskust ning põhjalikku teadmist projektist, millele teste kirjutatakse. Testide eesmärk on testida projekti funktsionaalsust ning läbida kriitilise tähtsusega vood. Kirjutatud koodi testimist aitab kontrollida rakendus Coverage.

Coverage on testimist abistav tehnoloogia, mis analüüsib projekti koodi kasutuse tõhusust läbi testide. Coverage annab ülevaate, mis projekti kood on kasutuses olnud testides ja mis mitte, et selgeks teha arendajale, mis osa koodist veel testimata on jäetud.

Flowit kasutab Coverage, et hoida arendajate teadlikkust nende poolt kirjutatud testide tõhususest ning sellega ka hoida projekti loogika kontrollitavust. Selle arendamine baasprojekti kulus umbes pool tööpäeva.

4.5.2 Tox tutvustus

Arenduses käigus olevas projektis kasutatavad tehnoloogiad või integratsioonid on määratud kindlale versioonile, et saavutada kooskõla omavahel ning hoida projekti töökindlana. Arendatud projektid paiknevad tihtipeale erinevates keskkondades. Erinevates keskkondades projekti käima saamisel võivad kaasneda probleemid seoses projekti töötamisega. Probleem võib sõltub mitmest erinevast asjast, alustades projekti kokkupanemisest ning lõpetades lõppkasutaja Pythoni versioonidest või isegi arvuti operatsioonisüsteemist. Ilma, et ükshaaval kontrollida projekti toimimist erinevates keskkondades ning selle peale palju aega kulutada, on hea kasutada Tox tehnoloogiat.

Tox on Pythoni moodul, mille eesmärk on testida projekti erinevates virtuaalkeskkondades erinevate projekti sõltuvatest tehnoloogia versioonidega, jooksutades konfiguratsiooni faili, et kontrollida koodi töötamist, ning ka *unit test*. Konfiguratsiooni failis määratakse ära Tox'i käsud, kuidas ja mida hakkab Tox testima. Tox tagastab tulemuse ning annab ülevaate läbitud testidest [20].

Flowit tahab kasutada Tox'i eesmärgil, et kontrollida antud projekti toimimist erinevatest keskkondadest erinevate versioonidega, tagades projektile töökindluse ning parendada tuleviku haldust. Lisaks aitab Tox hallata koodi testimist. Tox'i arendus ajakulu on umbes üks tööpäev.

4.6 Head tavad

Aja jooksul on tekkinud Django kommuunina paljud ühised head tavad või viisid kuidas Django projekte võiks arendada. Et uued projektid põhineksid kohe headel tavadel, siis arendati baasprojekt neile vastavalt.

Autor uuris häid tavasid läbi veebi, et saavutada kõige värskemate ja populaarsemate Django heade tavade kasutuselevõtmine.

Väljavalitud head tavad:

- Django mudelitele tekitada ühtne baasmudeli klass, mis koosneb korduvatest väljadest ning mida saab laiendada teistele mudelitele;
- Django mudelites kasutada teistest väljadest koosnevaid loogikaväljasid (property) ja meetodeid, mis toetavad mudelite haldamist;
- meetoditele ja muutujatele lisada väärtuse ja tagastuse andmetüübid;
- eelnevalt mainitud alapeatükis 3.1 „Django omadused“ kohaselt tuleb hoida kood lahus eri rakendustes [21].

5 Tulemus

Lõputöö autor annab antud peatükis ülevaate baasprojekti tulemusest ning analüüsib projekti tulemust ning selle kasulikkust. Lisaks kirjeldab autor baasprojekti edasistest plaanidest tulevikus.

5.1 Projekti tulemus

Baasprojekti idee sai püstitatud eesmärgil, et tekiks ühtlustatud alusprojekt, mis koosneb tehnoloogiatest ja integratsioonidest ning uuetest headest tavadest, mida Flowit saab kasutada oma uute projektide aluseks. Tehnoloogiad ja integratsioonid, mis said valitud pidid katma võimalikult hästi lähtetingimusi.

Lõputöö autor sai baasprojekti edukalt valmis ning projekt on valmis Flowiti poolt kasutusse võtmiseks. Tehnoloogiad ja integratsioonid, mis said lisatud, täitsid lähtetingimusi ning said arendatud võimalikult abstraktselt. Nendest puudub äriloogika ning edasi arendamiseks on ainult vajalik kood loodud. Lisaks sai arendustele tehtud dokumentatsioon, mis kirjeldab vastava tehnoloogia või integratsiooni kasutamist.

Baasprojekti tegemist juhendas vanemarendaja, kes kontrollis, et baasprojekt vastaks ootustele ning oleks sobilik alusprojektiks teistele projektidele. Arenduse käigus ei tekkinud raskusi ega suuremaid probleeme.

5.2 Projekti analüüs

Projekti analüüsis tahab autor välja arvutada baasprojekti kasulikkuse. Kuigi lõputöö kirjutamise ajahetkel pole antud baasprojekti kasutusele veel üheski projektis võetud, saab autor arvutada eeldatava tõhususe, lähtudes nende tehnoloogiate ja integratsioonide arendamise ja dokumenteerimise ajakulu kokkuhoiu pealt. Selleks tuleb arvutada baasprojektis ja teistes projektides samade tehnoloogiate ja integratsioonidega ajakulu võidu, mis määrab kui kasulik oleks baasprojekt olnud nendes projektides ning saame teada umbkaudse ajakulu võidu tuleviku projektides.

Eelnevas peatükis iga tehnoloogia ja integratsiooni kirjelduses, mainis autor selle arendamiseks kulunud aja. Baasprojekti arendus ajakulu saame arvutada nende tehnoloogiate ja integratsioonid ajakulu kokku lüües. Iga projekti baasprojekti kasulikus on erinev, kuid maksimaalne ajavõit, mida baasprojekti kasutamine võib pakkuda, on kuni 11,5 tööpäeva, arvestades, et baasprojekti seadistamine võtab ka umbes 1 tööpäev aega.

Teiste projektide andmete saamiseks analüüsis autor teistes projektides baasprojektides kasutatavate tehnoloogiate arendamisele ja dokumenteerimisele kuluvat aega. Analüüsimine toimus läbi BitBucket'i, mis on Git baasil töötav visuaalne koodi haldamissüsteem ning kaasates Flowiti arendajaid, kes on arendanud varasemaid lahendusi. Autor võttis ettevõttest 6 projekti analüüsimiseks. Uuritavate projektide valik põhines projektimahu ja uudsuse järgi.

Tabelis 1 on kirjeldatud baasprojektis kasutatavate tehnoloogiate ja integratsioonid kasutust teistes projektides ning nende ajavõit kasutades baasprojekti. Ajavõidu arvutusse on sisse arvestatud Django seadistustega kuluv aeg.

Tabel 1. Baasprojekti kasulikkus teistes projektides.

Projekt	Tehnoloogia	Ajavõit
Projekt 1	Celery RabbitMQ Everypay Merit Aktiva Django-allauth Black Pylint	~7 tööpäeva
Projekt 2	Bauwise Merit Aktiva	~4 tööpäeva

Projekt	Tehnoloogia	Ajavõit
	Pylint Black	
Projekt 3	Netspay Redis ESTO pay Tox Pylint Black	~7 tööpäeva
Projekt 4	Django-allauth Redis ESTO pay	~4 tööpäeva
Projekt 5	Pylint Black Everypay	~3 tööpäeva
Projekt 6	-	~1 tööpäev

Analüüsides baasprojekti kasutamist antud projektides, saame arvutada baasprojekti kasulikkuse. Analüüsitavatest tulemustest ajakulu kokku lüües saame, et baasprojekt kasulikus kokku nendes projektides oleks 26 tööpäeva ning keskmine projekti kohta oleks ~4,3 tööpäeva. Arvutatud kasulikus oleks suurem projektides, kus puuduvad baasprojekti arendatud head tavad, mis võiksid olla iga projektis.

Tulevaste projektides baasprojektide kasutamise kasulikkus sõltub vastavalt projekti iseloomule, suurusele ning mis tehnoloogiaid ja integratsioone projekt vajab. Autor eeldab, et tulevased projektid nõuvad, eelnevalt analüüsitud tulemuste keskmise kasulikkuse põhjal, vähemalt 4,3 tööpäeva vähem arendamist tänu baasprojektile.

5.3 Edasised arendusplaanid

Antud lõputöö käigus sai lõputöö eesmärk täidetud, ning projekti tegemise ajal tekkisid autoril ja Flowitil edasised arenguplaanid. Valminud projekt võimaldab teha ainult Django veebirakendusi, kuid tänapäeva projektid on jagatud ees- ja tagarakenduseks. Djangot on võimalik ka kasutada ainult tagarakendusena. Selleks otsustas Flowit projekti edasiseks arendamiseks kasutada DRF (*Django REST Framework*), mida saab kasutada tagarakendusena. DRF on lisamoodul Djangole, mis võimaldab Djangos arendada REST (*Representational State Transfer*) API-si, ning sellega olla omaette tagarakendus.

Baasprojekt oli arendatud Django versioonil 4.0, mis töö kirjutamise ajal oli kõige viimane versioon. Selleks, et baasprojekt oleks tulevikus ka aktuaalne ning tulevikus ei piiraks vanaks jäänud versioon selle peale arendatud projekti töötamist ning uute funktsionaalsuste lisamist, tuleb baasprojekti versiooni ajakohasena hoida. Kõige parem on uuendada baasprojekti versioone Django LTS versioonideks, sest LTS versioonid saavad kõige kauem tuge ning on kõige töökindlamad.

Baasprojekt võimaldab projekti sees olevate rakenduste kerget eemaldamist. See võib olla tüütu tegevus ning nõuab baasprojekti tundmist, et leida mis Django rakendusi eemaldada. Selle probleemi lahendamiseks on võimalik kasutada Cookiecutter-it. Cookicutter on Pythoni moodul, mis võimaldab käsurea abil konstrueerida repositooriumeid mallist [22]. See võimaldab kiiresti ja kergelt genereerida uue arendusprojekti nende tehnoloogiate ja integratsioonidega nii nagu uus projekt seda nõuab, kõike läbi käsureal olevate valikute. Cookicutter võimaldab eemaldada vajaduse midagi kustutada midagi projektist, luua korrektse projekti repositooriumi ning anda võimaluse kohese arendamise alustamiseks.

6 Kokkuvõte

Lõputöö eesmärk oli teha Flowitile Django baasprojekt, mida saab aluseks võtta järgnevates projektides. Baasprojekti arendamiseks pidi autor alustama lähtetingimustest tingitud tehnoloogiate ja integratsioonide valimisega. Autor uuris nii Flowiti teistest projektidest, kui ka internetist, lähtetingimustest sõltuvad tehnoloogiaid, integratsioone ja head tavasid, mis arendusel abiks tuleksid.

Autor arendas baasprojekti valmis eelnevalt uuritud tehnoloogiatest, integratsioonidest ja headest tavadest. Baasprojekti arendus oli autorile jõukohane ning probleemide või arusaamatuste puhul oli tuge ka Flowiti töötajatelt saada. Arendamisel ülemääraseid raskusi ei tekkinud ning projekt sai edukalt tehtud.

Baasprojektis kasutatud tehnoloogiad ja integratsioonid said valitud tingimusel, et täidavad lähtetingimusi võimalikult hästi, milleks olid: arendatud tehnoloogiaid ja integratsioonid oleksid universaalsed, millel puuduks äriloogika, projektides oleks potentsiaalne kasutus ning annaks märkimisväärse ajavõidu. Baasprojektis arendatud tehnoloogiad ja integratsioonid arendas autor antud eraldi Django rakendusteks. See võimaldab igas projektis lihtsasti erinevaid tehnoloogiaid ja integratsioone eemaldada, mida ei ole tarvis kasutada.

Baasprojekti valmimisel, hakkas autor analüüsima umbkaudset projekti kasulikkuse ajalist võitu. Analüüsimiseks võeti Flowitist 6 projekti, mida saaks baasprojekti vastu analüüsida. Projektid valiti projektimahu ja uudsuse järgi. Analüüsi tulemusel jõudis Autor järeldusele, et baasprojekti eeldatav kasulikus tuleviku projektides on umbes 4,3 tööpäeva.

Baasprojekti arenduse käigus tekkisid autoril ideed projekti tuleviku edasised arendus plaanid. Baasprojekt hõlmab ainult Django veebirakenduste arendamist, kuid kui lisada projektile juurde DRF, siis hõlmaks projekt REST API-de tegemist ning projekti saaks kasutada tagarakenduse loomiseks. Samuti tuleks projekti kasutusse võtta Cookiecutter moodul, et konstrueerida kiirelt ja kergelt baasprojektil põhinevaid arendusi.

Kasutatud kirjandus

[1] MDN Web Docs, „Django introduction,“. [Võrgumaterjal]. Loetud aadressil: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_is_django. [Kasutatud 28.03.2023].

[2] Trio, „9 Examples of Companies Using Django in 2023,“. [Võrgumaterjal]. Loetud aadressil: https://www.trio.dev/blog/django-applications. [Kasutatud 28.03.2023].

[3] W3Schools, „ Django Introduction,“. [Võrgumaterjal]. Loetud aadressil: https://www.w3schools.com/django/django_intro.php. [Kasutatud 28.03.2023].

[4] D.Feldroy, A.Feldroy, Two Scoops of Django 3.x, Best Practices for the Django Web Framework, 2020.

[5] Django dokumentatsioon, „Django’s release process,“. [Võrgumaterjal]. Loetud aadressil: https://docs.djangoproject.com/en/dev/internals/release-process/. [Kasutatud 28.03.2023].

[6] Endoflife.date, „Django,“. [Võrgumaterjal]. Loetud aadressil: https://endoflife.date/django. [Kasutatud 06.04.2023].

[7] B.Slatkin, Effective Python: 59 Specific Ways to Write Better Python, 2015.

[8] Black documentation, „The Black Code Style,“. [Võrgumaterjal]. Loetud aadressil: https://black.readthedocs.io/en/stable/the_black_code_style/index.html. [Kasutatud 05.11.2022].

[9] R.Bhardwaj, „What is Pylint? Python Programming Tool,“, Ipwithease. [Võrgumaterjal]. Loetud aadressil: https://ipwithease.com/what-is-pylint-python-programming-tool/. [Kasutatud 05.11.2022].

[10] M.Makai, „Celery“, Full Stack Python,. [Võrgumaterjal]. Loetud aadressil: https://www.fullstackpython.com/celery.html. [Kasutatud 05.11.2022].

[11] Celery documentation, „Introduction to Celery,“. [Võrgumaterjal]. Loetud aadressil: https://docs.celeryq.dev/en/stable/getting-started/introduction.html. [Kasutatud 12.11.2022].

[12] Nick, „What is RabbitMQ? ,“, The Iron.io Blog, 12.11.2020. [Võrgumaterjal]. Loetud aadressil: https://blog.iron.io/what-is-rabbitmq/. [Kasutatud 22.04.2023].

[13] E.Etuk, „Why You Should use Celery with RabbitMQ, “, Section, 27.04.2021. [Võrgumaterjal]. Loetud aadressil: https://www.section.io/engineering-education/why-you-should-use-celery-with-rabbitmq/. [Kasutatud 22.04.2023].

[14] EveryPay. [Võrgumaterjal]. Loetud aadressil: https://every-pay.com/et/about-us/. [Kasutatud 03.12.2022].

[15] ShopRoller, „Esto makseviisid,“. [Võrgumaterjal]. Loetud aadressil: https://support.shoproller.ee/article/418-estopay-ja-jarelmakse. [Kasutatud 03.12.2022].

[16] Esto. [Võrgumaterjal]. Loetud aadressil: https://esto.eu/ee?lang=et. [Kasutatud 03.12.2022].

[17] Merit Aktiva. [Võrgumaterjal]. Loetud aadressil: https://www.merit.ee/merit-aktiva/raamatupidamisprogramm/. [Kasutatud 24.04.2023].

[18] Bauwise. [Võrgumaterjal]. Loetud aadressil: https://www.bauwise.com/. [Kasutatud 22.04.2023].

[19] F. Adhing´a, „How To Authenticate Django Apps using django-allauth,“, DigitalOcean, 14.06.2022. [Võrgumaterjal]. Loetud aadressil:

https://www.digitalocean.com/community/tutorials/how-to-authenticate-django-apps-using-django-allauth. [Kasutatud 31.03.2023].

[20] Rune, „Ultimate Python Tox Guide with Practical Examples with MyPy and PyTest,“, Learn Python With Rune, 20.08.2022. [Võrgumaterjal]. Loetud aadressil: https://www.learnpythonwithrune.org/ultimate-tox-guide-with-practical-examples-with-mypy-and-pytest/. [Kasutatud 30.03.2023].

[21] Django Styleguide, „ Django Styleguide,“ 30.11.2020. [Võrgumaterjal]. Loetud aadressil: https://github.com/HackSoftware/Django-Styleguide#how-to-ask-a-question-or-propose-something. [Kasutatud 06.05.2023].

[22] H.Canto, „Project templates and Cookiecutter,“, Medium, 18.07.2018. [Võrgumaterjal]. Loetud aadressil: https://medium.com/worldsensing-techblog/project-templates-and-cookiecutter-6d8f99a06374. [Kasutatud 01.04.2023].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Eerik Kodasma

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Django baasprojekti koostamine ettevõttele Flowit Estonia OÜ", mille juhendaja on Einar Kivisalu
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

15.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Baasprojekti malli näide (autori joonis)

```
{% extends "core/list.html" %}
{% load i18n %}
{% block content %}

<div class="container-fluid pt-4">
    <h4>EveryPay</h4>
    <div class="row card-row">
        <div class="col-sm-12">
            <div class="success-msg" style="display: none">
                  <i class="fa fa-check"></i>
                  Refreshed successfully
            </div>
            <table class="table">
                <thead>
                <tr>
                  <th scope="col"></th>
                  <th scope="col">Invoice reference</th>
                  <th scope="col">Payment reference</th>
                  <th scope="col">Status</th>
                  <th scope="col">Request</th>
                </tr>
              </thead>
                <tbody>
                    {% for i in page_obj %}
                    <tr>
                        <th data-model-name="id">{{i.id}}</th>
                        <td data-model-
name="invoice_reference">{{i.invoice_reference}}</td>
                        <td data-model-
name="payment_reference{{i.id}}">{{i.payment_reference}}</td>
                        <td data-model-
name="status{{i.id}}">{{i.status}}</td>
                        <td data-model-name="request">{{i.request}}</td>
                        <td><button id="{{i.id}}" name="payment"
type="button" class="btn btn-primary btn-sm">Check payment</button></td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
            {% if page_obj.has_other_pages %}
              <ul class="pagination">
                {% if page_obj.has_previous %}
                  <li><a href="?page={{ page_obj.previous_page_number
}}">&laquo;</a></li>
                {% else %}
```

```
                <li class="disabled"><span>&laquo;</span></li>
              {% endif %}
              {% for i in page_obj.paginator.page_range %}
                {% if i <= page_obj.number|add:5 and i >=
page_obj.number|add:-5 %}
                {% if page_obj.number == i %}
                  <li class="active"><span>{{ i }} <span class="sr-
only">(current)</span></span></li>
                {% else %}
                  <li><a href="?page={{ i }}">{{ i }}</a></li>
                {% endif %}
                {% endif %}
              {% endfor %}
              {% if page_obj.has_next %}
                <li><a href="?page={{ page_obj.next_page_number
}}">&raquo;</a></li>
              {% else %}
                <li class="disabled"><span>&raquo;</span></li>
              {% endif %}
            </ul>
          {% endif %}
        </div>
    </div>
</div>
{% endblock content %}
```