Tallinna Ülikool

Digitehnoloogiate instituut

Informaatika õppekava

Pythoni veebiraamistike Bottle ja Flask võrdlus

Bakalaureusetöö

Autor: Rauno Kaldmaa

Juhendaja: Inga Petuhhov

Autor: ..... „ ..... „2018

Juhendaja: ..... „ ..... „2018

Instituudi direktor: ..... „ ..... „2018

Tallinn 2018

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö 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

Mõisted

Mõistete loetelus on valdavalt kasutatud viitena e-Teatmiku sõnaraamatut. Mõne mõiste puhul on kasutatud teist allikat ning sellele viidatakse eraldi mõiste lõpus (e-Teatmik: IT ja sidetehnika seletav sõnaraamat, kuupäev puudub).

AJAX (*Asynchronous JavaScript and XML*) – lühend asünkroonsest JavaScriptist ja XML'ist, millega tähistatakse interaktiivsete veebirakenduste loomise meetodit, kus toimub kulissidetagune andmevahetus brauseri ja veebiserveri vahel nii, et iga kasutaja liigutuse peale pole vaja kogu veebilehte uuesti alla laadida.

API (*Application Programming Interface*) – rakendusliides, mis on arvuti operatsioonisüsteemiga või rakendusprogrammiga määratud reeglistik, mille alusel rakendusprogramm kasutab operatsioonisüsteemi või teise rakendusprogrammi teenuseid.

Dekoraator (*decorator*) – funktsioon, mis tagastab programmis teise funktsiooni ja mida tavaliselt kasutatakse funktsioonide teisendamisel (Hellkamp, 2018).

JSON (*JavaScript Object Notation*) – JavaScripti objektide notatsioon, mis on JavaScripti alamhulgal põhinev lihtne inimlugemiseks ja -kirjutuseks hõlbus andmevahetusvorming (Andmekaitse ja infoturbe leksikon, kuupäev puudub).

Localhost – kohalik host (võrguga ühendatud arvuti), mille taga istub kasutaja ise. Kõik paketid, mida kohalik host saadab aadressile (Ipv4 protokolli kohaselt kohaliku hosti IP aadress on tavaliselt 127.0.0.1 ja alamvõrgumask 255.0.0.0), liiguvad selle hosti enda sees ja võrku ei lähe.

PHP (*HyperText Processor*) – veebiprogrammeerimise jaoks mõeldud platvormist sõltumatu serveripoolne skriptimise keel.

RAD (*Rapid Application Development*) – arendussüsteem, mis annab programmeerijatele võimaluse kiiresti programme koostada.

HTML (*HyperText Markup Language*) – enimlevinud kodeerimissüsteem (tekstivorming) veebidokumentide loomiseks.

HTTP (*HyperText Transfer Protocol*) – hüperteksti edastusprotokoll. Teiste sõnadega TCP/IP klient-server protokoll HTML-dokumentide vahetamiseks veebis ehk andmevahetusprotokoll, mida kasutatakse Internetis dokumentide vahetamiseks.

USENET (*User Network*) - ülemaailmses võrgus asuvatel serveritel hoitav teatetahvlisüsteem ehk kirjade kollektsioon mitmesugustel teemadel.

URL (*Uniform Resource Locator*) - igale dokumendile või muule ressursile Internetis vastab oma unikaalne internetiaadress.

WSGI (*Web Server Gateway Interface*) – veebiserveri võrguvärava kasutajaliides, mis kirjeldab, kuidas veebiserver suhtleb veebirakendusega ja kuidas veebirakendusi saab omavahel siduda, et käsitleda ühte päringut (WSGI.org, kuupäev puudub).

XML (*Extensible Markup Language*) – laiendatav märgistuskeel suvaliste andmete struktueerimiseks.

Sissejuhatus

Pythoni programmeerimiskeele populaarsus tõuseb üha enam. TIOBE[1] indeksi põhjal oli Python 2003. aasta populaarseimate programmeerimiskeelte edetabelis 12. kohal. Seejärel tõusis 2008. aastal kuuendale kohale ning 2018. aasta senise seisuga asetseb Python neljandal kohal pärast Java, C ja C++ programmeerimiskeeli.

Võrreldes C ja C++ keeltega on Python lihtne ja kergesti õpitav, seega alustatakse paljudes ülikoolides programmeerimise õppimist Pythoniga. Kuigi Python on olemuselt lihtne keel, on see leidnud laialdast kasutust. Näiteks Javascript ja PHP on mõeldud peamiselt veebiprogrammeerimiseks, kuid Pythoniga saab programmeerida skripte, veebirakendusi ja muid. Pythoni kasutusega on välja arendatud järgmised rakendused nagu Dropbox, NetBeans, Blender 3D ja Instagram. Veebirakenduste seast on loodud näiteks Reddit, Youtube ja Quora. Python on jõudnud ka videomängude arendamiseni - Civilization IV, Battlefield 2 ja World of Tanks.

Kuid leidub palju valikuid, millega rakendusi tehakse. Pythonil on saadaval suur valik erinevaid raamistikke ja mooduleid, mille najal üles ehitada veebirakendusi, tarkvara või muud. Väga populaarsed neist on Django ning Pyramid, aga leidub ka Pyglet, Tornado, Dash, Falcon ja mitmed teised.

Käesoleva bakalaureusetöö eesmärgiks on uurida Pythoni programmeerimiskeele Bottle[2] ja Flaski[3] veebiraamistikke ning moodustada ülevaade mõlema kohta. Töö käigus uuritakse välja mõlema veebiraamistiku funktsioonid, nende eelised ja miinused ning töötamise põhimõtted.

Bottle ja Flask on Python Wiki [4] sõnul mikroraamistikud, mille kallal töötatakse järjepidevalt versiooniuuendustega. Üldiselt Tallinna Ülikooli õppetöö raames leiavad käsitlemist teiste programmeerimiskeeltega seotud veebiraamistikud, kuid käesoleva tööga soovib autor näidata, et Pythonit saab ka edukalt veebirakenduste vallas kasutada.

[1] Tiobe Programming Community Index: https://www.tiobe.com/tiobe-index/
[2] Bottle: http://bottlepy.org
[3] Flask: http://flask.pocoo.org
[4] Python Wiki: https://wiki.python.org/moin/WebFrameworks

Eesmärgi saavutamiseks autor annab esimeses peatükis ülevaate Pythonist ning selle tekke ajaloost. Teises peatükis autor selgitab lahti raamistiku mõiste ning seejärel tutvustab eraldi nii Bottle kui ka Flaski veebiraamistikku. Kolmas peatükk on raamistike võrdlus, milles autor selgitab põhjalikumalt, kuidas viib läbi võrdluse, mis on võrdlusmeetodiks ning mida täpsemalt võrdleb. Antud peatükis selgitatakse mõlema raamistiku töötamise põhimõtet, olulisi funktsioone ja võimalusi. Dokumentatsiooni läbi töötamise, koodinäidete testimise ning uuringu põhjal valmib ülevaatlik võrdlus ning raamistike võrdluse kokkuvõte.

1 Python

Pythoni oli loonud Guido van Rossum oma hobiprojekti raames 1989. aasta lõpus. Selle nimi Python ei tulene mitte samanimelisest maost, vaid briti komöödiast „Monty Python's Flying Circus." Pythoni eelkäijaks oli ABC keel, mis oli Hollandis asuva CWI uurimisinstituudis lõpetatud projekt (Wood, 2015).

1980-ndate lõpus oli Guido van Rossum seotud Amoeba hajusoperatsioonisüsteemi projektiga. Looja sõnul oli süsteemi haldamiseks vaja C-keeles programmeerimisest või Bourne'i kesta kaudu skriptimisest paremat viisi, sest Amoeba'l oli juba olemas oma süsteemikutse (ingl k „*system call*") kasutajaliides, millele polnud omakorda kerget ligipääsu Bourne'i kestast. Pythoni looja leidis, et tema vajadusi täidaks ABC keelele sarnaneva süntaksiga skriptimiskeel, millel oleks ligipääs Amoeba süsteemile. Guido van Rossum polnud rahul mitme aspektiga ABC keeles, kuid teisest küljest mitmed selle omadused olid tema arvates head. ABC keelt polnud võimalik laiendada, mis oligi peamine ajend Pythoni loomiseks (Python Software Foundation, kuupäev puudub-a).

Guido van Rossum jätkas Pythoni projekti kallal töötamist 1990. aastal. Python leidis tookord kasutust Amoeba projektis järjest rohkem ja kolleegide tagasiside pani teda loodud Pythoni keelt veel täiendama. Pärast aasta aega arendust Guido van Rossum avalikustas Pythoni ja laadis selle üles USENET'i keskkonda (Python Software Foundation, kuupäev puudub-a).

Python on interpreteeriv, objekt-orienteeritud ning kõrgtaseme programmeerimiskeel, millel on dünaaamiline semantika ja saadaval kõikidel platvormidel. Järgnevad omadused nagu andmestruktuuride kõrgtasemeline ülesehitus, dünaamiline tüüpimine ja sidumine võimaldavad Pythonil leida rakendust kiirprogrammeerimisel (ingl k „*RAD – Rapid Application Development*"), skriptimisel ning erinevate komponentide omavahel kokku sidumisel (Python Software Foundation, kuupäev puudub-b).

Pythoni süntaks on kergesti õpitav ning rõhutab loetavust, mis vähendab omakorda programmi ülalpidamise kulu. Lähtekoodis pole muutujate, parameetrite, funktsioonide

ega meetodite deklaratsioone, mis teeb koodi lühikeseks ja paindlikuks ning selle arvelt säästab kompileerimiseks kuluvat aega. Lisaks, Python toetab mooduleid ja pakette, mis omakorda julgustab looma modulaarset programmi ning koodi taaskasutust. Pythoni interpretaator ja sellega kaasasolev laiaulatuslik standardteek on saadaval kõikidel põhiplatvormidel ning neid võib vabalt levitada (Google Developers, 2016; Python Software Foundation, kuupäev puudub-b).

Pythoni kasutusega kasvab produktiivsuse tase, sest selles puudub kompileerimise etapp ja redigeeri-testi-silu (ingl k „*edit-test-debug*“) tsükkel on kiire. Vigade silumine on Pythoniga kirjutatud programmides kerge, kuna programmikoodis esinev viga ei põhjusta segmentatsioonivigu. Selle asemel interpretaator toob esile erindi (ingl k „*raise an exception*“) vea avastamisel. Juhul, kui erindit ei näidata, siis interpretaator prindib välja pinujärje (ingl k „*stack trace*“) (Python Software Foundation, kuupäev puudub-b).

Silur on kirjutatud Pythonis endas, mis võimaldab uurida lokaal- ja globaalmuutujaid, hinnata meelevaldseid avaldisi, seada üles katkestuspunkte ja jälgida programmi tööd teda lause haaval täites. Teisest küljest, kõige kiirem viis programmi silumiseks on hoopis lisada mõned printimise laused koodi sisse. Eelnevalt mainitud redigeeri-testi-silu tsükkel teeb selle lähenemise väga efektiivseks (Python Software Foundation, kuupäev puudub-b).

2 Raamistik

Enne Bottle ja Flaski raamistiku tutvustamist tuleb eelnevalt selgitada, mida tähendab raamistiku mõiste. Raamistik on taaskasutatav ja universaalne platvorm, mida kasutatakse tarkvarade, veebirakenduste ja digilahenduste arenduses. Raamistikke kasutatakse enamasti tarkvara või veebirakenduse arenduse hõlbustamiseks, kuna programmeerijatel jääb rohkem aega projekti nõuete täitmisele ning ei pea tähelepanu pöörama pisinõuetele (MoWeble, 2013).

Mis puudutab raamistiku arhitektuuri, siis see koosneb kahest osapoolest. Nendeks on kuum- ja külmkoht (ingl k vastavalt „*hot spot*“ ja „*frozen spot*“). Külmkohad defineerivad üleüldist süsteemi arhitektuuri, mis hõlmavad endas peamiseid komponente ning nendevahelisi suhteid. Kuumkohad on aga need süsteemi osad, kuhu arendajad lisavad omalt poolt koodi, et lisada projektile spetsiifilist funktsionaalsust (Christensen, 2011).

Raamistikud sisaldavad tavaliselt kompilaatorit, tugiprogramme, APIsid ja muid vahendeid koos komponentidega, mis mängivad olulist rolli arendusprotsessis. Raamistikud aitavad vähendada projektide arenduseks kuluvat aega. Kui arendaja jaoks muutub raamistiku kasutus selgeks, siis edasipidi suudab ta kiiremini ja kergemini lõpetada projekte. Üks olulisi põhjuseid, miks raamistikke kasutatakse, on rakenduste funktsionaalsuse laiendamine (MoWeble, 2013).

2.1 Raamistike tüübid

Veebilehtede ja rakenduste arenduseks kasutatakse erinevaid tüüpe raamistike, mis sõltub muidugi projektist ja nende spetsiifilistest nõuetest. Raamistike tüüpide kategoriseerimine aitab arendajal valida kergemini sobivat raamistikku projekti jaoks. Praeguseks on olemas AJAXi, veebirakenduse, rakenduse, sisuhalduse ning multimeedia raamistik ja muud. Näiteks veebirakenduse raamistik toetab dünaamiliste veebirakenduste ja -lehtede arendust. AJAXi raamistiku kategoorias kasutatakse veebirakenduse arenduses AJAXit. Seejuures multimeedia raamistik on tarkvara raamistik, millega käsitletakse arvutile talletatud meediat ja näiteks meediapleierid kasutavad seda (MoWeble, 2013).

Kuid üldplaanis saab raamistikke jaotada universaal- (ingl k „*full-stack*“) ja mikroraamistikuks. Universaalraamistik aitab arendajaid täieliku rakenduse arendusega alates kasutajaliidesest ja lõpetades andmehoidlaga. Kõike muud, mis ei kuulu universaalraamistiku alla, kutsutakse tehniliselt võttes mitte-universaalraamistikuks (ingl k „*nonfull-stack framework*“). Universaalraamistikud kasvasid aja jooksul järjest suuremaks, et need oleksid võimelised käsitlema suuri ning komplekksseid veebilehtesid. Suure kasvuga kaasnes negatiivne asjaolu, et neid on liiga raske kasutada lihtsamate projektide jaoks ja kiirprogrammeerimise puhul on see ülekoormav. Seetõttu arendati välja üleliigsetest teekidest ja komponentidest kärbitud mikroraamistikud, mida oli kergem kasutusele võtta väiksemates projektides, tagasid kiirema testimise ja toote väljastamise. Kui mitte-universaalraamistike seast nii raamistik kui ka teek kokku koosneb vähem kui 5000 koodireast, kutsutakse seda mikroraamistikuks (Habib, 2015).

Käesoleva bakalaureusetöö raames uuritavad raamistikud Bottle ja Flask kuuluvad mõlemad mikroraamistiku kategooriasse, kuid spetsiifilisemalt need on veebiraamistikud. Üldplaanis mikroraamistikud jätavad tavaliselt välja paljud komponendid. Nendeks on näiteks:

- Veebimallide mootor (ingl k „*web template engine*“)
- Sisestusandmete õigsuse kontroll (ingl k „*input validation*“)
- Andmebaaside abstraktsioon (ingl k „*database abstraction*“)
- Rollid, kasutajakontod ja autentimine

Kuid samuti tuleks silmas pidada, et mõnes mikroraamistikus võib olla mõni komponent esindatud, mida teises ei ole. Eelnevalt mainitud komponendid on tüüpiliselt universaalraamistikus esindatud (Habib, 2015).

Flask’i koduleheküljel on samuti kirjas järgnevalt: „mikro“ ei tähenda seda, et kogu veebirakendus peab mahtuma ühe Pythoni faili sisse (kuigi see on võimalik) ega seda, et Flaskis on puudu mitmed funktsionaalsused. Mikroraamistiku sõna „mikro“ all peetakse silmas, et Flaski eesmärk on hoida tuuma lihtsa, kuid laiendatavana (Ronacher, 2018).

2.2 Bottle

Ametlik koduleheküIg: https://bottlepy.org

Bottle-nimelise raamistiku (vt Joonis 1) on loonud Marcel Hellkamp, kes õppis Georg-Augusti ülikoolis Göttingenis informaatikat. Python oli tema lemmik programmeerimiskeel, kuigi ta programmeeris samuti Ruby's ja Javascriptis. Bottle oli tema jaoks esimene avatud lähtekoodiga projekt, mis algas väikese eksperimendiga, kuid tõi talle palju positiivset tagasisidet. Seetõttu otsustaski ta Bottle raamistiku välja arendada, mille avalikustas 2009. aastal (Hellkamp, kuupäev puudub).

Joonis 1. Bottle logo

Bottle on kiire, lihtne ja kergekaaluline WSGI mikro-veebiraamistik Pythoni jaoks. Seda levitatakse üheainsa failimoodulina ja ei sõltu ühestki teisest teegist peale Pythoni standardteegi (Hellkamp, 2018).

Bottle't soovitatakse kasutada peamiselt kolmel eesmärgil: uute ideede prototüüpimine, veebirakenduste ülesehituse tundma õppimine ja kergete veebirakenduste arendamine ning käitamine. Bottle raamistiku kasutusega on lihtsamate ideede prototüüpimine kergem kui näiteks Django-nimelise universaalraamistiku kaudu. Nagu eelnevalt sai mainitud, Bottle't levitatakse üheainsa failimoodulina. Mis sisuliselt tähendab seda, et on üks suur lähtefail nimega *bottle.py*, mis sisaldabki tervet Bottle raamistikku. Selles lähtefailis on olemas kõik vajalik, et veebirakenduse koodi ühendada Bottle raamistikuga. Kui veebirakenduse loomise protsess võib mõne vähem kogenud arendaja jaoks olla liiga keeruline, siis bottle.py sidumine projektiga aitab tulemust kergemini saavutada (Makai, 2014).

2.3 Flask

Ametlik koduleheküIg: http://flask.pocoo.org

Flaski looja Armin Ronacher kuulub rahvusvahelisse Pythoni entusiastide gruppi Pocoo Team[5], mis on osa Pythoni kogukonnast ja selle eesmärgiks on ilma kommertshuvita avatud lähtekoodiga Pythoni tarkvara kallal töötamine ning üksnes teiste heaolu nimel (Brandl & Ronacher, kuupäev puudub).

Erinevalt Bottle'st, mis algas eksperimendiga, siis Flaski sünnilugu on seotud aprillinaljaga. Armin Ronacher oli plaaninud mõnda aega teha Pythoni mikroraamistikuga seotud aprillinalja, mille nimel ta tegi 2010. aastal valmis HTML veebilehe, mis sisaldas Denied-nimelist mikroraamistikku (vt Lisa 1). Denied koosnes 160 koodireast, mis kasutas Werkzeugi ja Jinja2'l põhineval väga algelist WSGI rakendust. Kogu kood hõlmas endas ideesid, mis kõlasid tolleaegselt väga rumalalt, kuid kõik need ideed olidki praegu saadaval olevate mikroraamistike inspiratsiooniks. Denied mikroraamistikuga reklaamiti selle võimsat skaleerimisvõimet, võimsat jõudlust (6000 päringut sekundis), kaasa tulevat arendusserverit, sisseehitatud URL vastendamist, ühteainsat Pythoni faili ja muud. Paigaldamine ega seadistamine polnud vajalik ja sõltus ainult Pythoni standardteegist. Usutavuse lisamiseks sisaldas veebileht iroonilisi võltsitud kommentaare ja teiste Pythoni arendajate heakskiitusid (Ronacher, 2010).

Ronacherile endalegi suureks üllatuseks langesid paljud inimesed õnge. Temaga võttis ühendust mitu inimest, kes soovisid lähtekoodi kallal töötada ja seda edasi arendada. Veebileht saavutas suure populaarsuse ning mitmed inimesed nii Redditis kui ka emaili teel võtsid Denied projekti tõsiselt. Ronacher õppis sellest, et millegi reklaamimisel mängib julgus suurt rolli, sest inimesed läksid aprillinaljaga kergesti kaasa ning ei kahelnud selle autentsuses. Aprillinalja tõttu tundus talle, et inimestele ilmselt meeldivad mikroraamistikud ja üleüldine idee tarkvarast, mis koosneb ühest failist ja samal ajal ei sõltu teistest teekidest (Ronacher, 2010).

[5] Pocoo Team: http://www.pocoo.org/team/

Joonis 2. Flask logo

Werkzeugi [6] WSGI tööriistakomplektil ja Jinja2 [7] mallimootoril põhinev Flask on mikroraamistik (vt), mille eesmärk on hoida tuuma lihtsana, kuid laiendatavana. Flask ei otsusta näiteks kasutaja eest, millist andmebaasi kasutada. Kuid otsuseid, mida Flask teeb kasutaja eest, saab kergesti muuta, milleks võib olla näiteks mallide vahetamine. Flaskis puudub andmebaaside abstraktsiooni kiht, sisestusandmete õigsuse kontroll või muud funktsioonid, mille jaoks on olemas eraldi teegid. Selle asemel Flask toetab laiendusi, mis lisab rakendusele neid funktsioone justkui see oleks Flaskis olemas. Laiendused tagavad veebirakendusele erinevaid funktsioone nagu andmebaaside integratsioon, failide üleslaadimise haldamine, sisestusandmete ning sisestusväljade õigsuse kontroll ja muu. Otsuste tegemine jääb Flaski kodulehe sõnul arendajale ning Flask saab olla kõike, mida arendaja soovib ja samal ajal olla mitte midagi, mida arendaja ei soovi (Ronacher, 2018).

[6] Werkzeug: http://werkzeug.pocoo.org/
[7] Jinja2: http://jinja.pocoo.org/

3 Raamistike võrdlus

Kuna Bottle ja Flask on mikroraamistikud, siis need ei oma installeerimise järel paljusid funktsioone nagu universaalraamistikud (Ronacher, 2018). Seetõttu lähtub autor mõlema raamistiku puhul nende koduleheküljel saadaval olevast dokumentatsioonist, kuna autori meelest Armin Ronacher ja Marcel Hellkamp pakuvad raamistike loojatena kõige adekvaatsemat informatsiooni. Autor võrdleb neid võimalusi ja funktsioone, mis on Bottle's ja Flaskis vaikimisi olemas.

Võrdlemismeetodiks on mõlema dokumentatsiooni läbi töötamine, nendes sisalduvate näidete proovimine ja uurimine, mis viisil ning kuidas saavutatakse mingit eesmärki. Üheks oluliseks eesmärgiks on näiteks siluri käivitamine, mis on mõlemas raamistikus erinev, seega autor selgitab nende erinevust dokumentatsiooni ja näidete alusel. Kui ühe raamistiku teatud valdkonnas leidub rohkem võimalusi kui teisel, siis autor toob need erinevused välja. Raamistike võrdluses välja toodud koodinäited pärinevad dokumentatsioonist, mis on saadaval vastava raamistiku ametlikul koduleheküljel (Hellkamp, 2018; Ronacher, 2018).

Autor leiab isiklikult, et kriteeriumid nagu lihtsus, mugavus jm on liiga subjektiivne, et anda Bottle ja Flaski mikroraamistiku kohta objektiivset ülevaadet. Näiteks lihtsuse hindamiskriteerium on võib olla seotud lugeja oskusest hästi vallata Pythonit, HTML5, andmebaase ja muid. Mugavus võib sama hästi olla ühe lugeja jaoks ühe *.py* faili sisse mikroraamistiku mahutamine või teise lugeja jaoks kodeerida sama projekt valmis vähema arvu koodiridadega. Seetõttu autor tunneb, et võrdluse koostamisel tuleb lähtuda nendest funktsioonidest, mis on mikroraamistikega kaasa antud.

Kuna mõlema raamistiku puhul töötatakse järjepidevalt versiooniuuendustega, uurib autor kõigepealt, mis Pythoni versiooni kumbki raamistik toetab. Praeguse seisuga on üleüldiselt kasutuses nii Python 2 kui ka 3 versioonid, kuid Python 3 väljalaskega muutusid mõned asjaolud. Python 2.x arendatakse veel edasi, kuid uuendusi ei tule sama tihti kui Python 3'l. Kuna muudeti keelereegleid, siis Python 3's puudub tagasiühilduvus (ingl k „*backwards*

compatibility“) ja Python 2.x’ga kirjutatud programmid ei lähe Python 3’s tööle (Rossum, 2009).

Bottle 0.12 kasutamiseks peab olema Python 2.5 või sellest uuem versioon. Bottle toetab samuti kõiki Python 3 versioone. Bottle 0.13 on alles arenduses ning sellest alates hakkab olema tugi Pythoni versioonidel 2.7 ja 3. Samas Flask 1.0.2 jaoks soovitatakse kõige uuemaid Python 3 versioone. Flask toetab 3.4 ja sellest uuemaid ning Python 2.7 (Hellkamp, 2018; Ronacher, 2018).

3.1 „*Hello world*“ näide

Esimest korda programmeerimiskeelt õppima asudes tavaliselt harjutatakse läbi „*Hello world*“ printimine programmi kaudu. Programmeerimiskeelt selgitavad õpikud ja õpetused Internetis tutvustavad kõige esimesena „*Hello world*“ printimist, et anda väike ülevaade süntaksist ja sellest, kuidas kood töötab. Programmeerijate jaoks nende kahe sõna nägemine tähendab seda, et nende kirjutatud kood kompileerub, laeb, töötab ja väljundit on näha (Trikha, 2015).

Autor tunneb, et „*Hello world*“ näite selgituse kaasamine Bottle ja Flask’i võrdlusesse on oluline, kuna mõlema raamistiku puhul see seletab lahti olulise kontseptsiooni, kuidas luuakse ühendus Pythoni faili, sisseehitatud arendusserveri ning veebilehitseja vahel. Kuigi „*Hello world*“ on olemuselt väga algeline ja minimalistlik, on selle kaudu kerge selgitada erinevust Bottle ja Flaski vahel, kuidas need töötavad. Järgnevates koodinäidetes on näha kontseptuaalset erinevust nende raamistike vahel.

3.1.1 Bottle

Bottle koodinäite (vt **Error! Reference source not found.**) sees **route()** dekoraator seob koodi URL raja külge. Antud näites siis **/hello** rada on seotud **hello()** funktsiooniga. Seda kutsutakse marsruudiks, mis ongi Bottle mikroraamistiku juures kõige tähtsam kontseptsioon. Marsruutide arv pole piiratud ühega ja arendaja võib oma soovile vastavalt lisada nii palju marsruute kui tarvis. Iga kord, kui veebilehitseja esitab URL päringu, siis kutsutakse vastav funktsioon ja tagastatav väärtus saadetakse tagasi veebilehitsejasse. Tuleb samuti märkida, et esimesel korral **route()** kutsudes luuakse automaatselt **Bottle**

eksemplar (ingl k „*instance”*). See on teiste sõnadega marsruutide lisamine globaalsesse „vaikerakendusse“ (ingl k „*default application*“) (Hellkamp, 2018).

```
from bottle import Bottle, run
app = Bottle()

@app.route('/hello')
def hello():
    return "Hello World!"

run(app, host='localhost', port=8080)
```

Koodinäide 1. Bottle raamistikus „*Hello world*“ printimine

Koodinäite viimasel real **run()** kutse käivitab sisseehitatud arendusserveri, mis jookseb *localhost* pordil 8080 ja täidab päringuid. Arendajal on muidugi võimalus hiljem serveri tagasüsteemi (ingl k „*backend*“) vahetada, aga antud olukorras piisab arendusserverist, kuna see ei nõua paigaldamist ja on kerge meetod rakenduse tööle saamiseks ning testimiseks (Hellkamp, 2018).

3.1.2 Flask

Flaski koodinäide (vt **Error! Reference source not found.**) impordib kõigepealt Flaski klassi, mille eksemplarist saab olema WSGI rakendus. Teine koodirida loob klassi eksemplari ja selle argument on rakenduse mooduli või paketi nimi. Vastavalt sellele, kas seda käivitatakse kui rakendusena või imporditakse kui paketina, siis argumendis sisalduv nimi saab olema erinev. See on oluline, sest siis Flask teab, kust otsida malle, staatilisi faile ja muid (Ronacher, 2018).

Dekoraator **route()** ütleb Flaskile, millise URLiga käivitatakse arenendaja poolt kodeeritud funktsioon. Funktsioonile on antud nimi, mis samuti genereerib URLe just selleks funktsiooni eesmärgiks ja tagastab sõnumi, mida veebilehitsejas näidatakse (Ronacher, 2018).

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Koodinäide 2. Flask raamistikus *"Hello world"* printimine

Erinevalt Bottle'st, milles on *localhost* arendusserveri käivitamiseks **run()** kohe koodinäite viimasel real ja piisab kõigest Pythoni faili käivitamisest, siis Flaski raamistikus tuleb terminalis anda eraldi käsk programmide jooksutamises. Selleks võib olla nii **flask** käsk (vt Koodinäide 3) kui ka Pythoni **-m** lüliti (vt Koodinäide 4). Terminalis tuleb eksportida **FLASK_APP** keskkonnamuutuja, mille järel peab sisestama käsu (Ronacher, 2018).

```
$ export FLASK_APP=hello.py
$ flask run
 * Running on http://127.0.0.1:5000/
```

Koodinäide 3. Terminalis „flask run" käsuga arendusserveri käivitamine

```
$ export FLASK_APP=hello.py
$ python -m flask run
 * Running on http://127.0.0.1:5000/
```

Koodinäide 4. Terminalis Pythoni -m lülitiga arendusserveri käivitamine

Mõlema meetodi kasutusega käivitub Flaski sisseehitatud server, mis on testimiseks piisavalt hea ja seejärel kui veebilehitsejas minna aadressile http://127.0.0.1:5000/, kuvatakse seal „*Hello world*" tervitust (Ronacher, 2018).

3.2 Raamistike funktsioonid

Järgnevalt on toodud välja peamised funktsioonid ja eriärasused, mis on Bottle's ja Flaskis olemas ning kuidas need töötavad. Autor ei kaasa antud alapeatükis komponente ja funktsioone, mis on tavaliselt universaalraamistikus olemas. Autor kirjutab iga funktsionaalsuse kohta selgituse, mis see endast kujutab ja miks see on oluline ning seejärel toob välja nii erinevused kui ka sarnasused.

3.2.1 Laiendused

Mikroraamistike puhul on laiendused kõige olulisem tunnusjoon, kuna algselt on nendes paljud komponendid puudu ja vastavalt arendaja soovile saab neid laienduste kaudu kasutada. Näiteks andmebaasid ja objekt-relatsioonivastendus (ORM) on veebirakenduste puhul olulisel kohal.

Bottle koduleheküljel [8] on nimekiri kolmandate osapoolte pluginatest, mis laiendavad Bottle'i funktsionaalsust ja integreerivad teiste teekidega. Praeguse seisuga koosneb nimekiri 22 pluginast, millest näiteks andmebaaside ja ORMi funktsionaalse tagamiseks on olemas Bottle-Sqlalchemy (SQLAlchemy), Bottle-Sqlite (SQLite3 andmebaas) ning Macaron (objekt-relatsioonivastendaja SQLite'i jaoks) (Hellkamp, 2018).

Flaski koduleheküljel [9] on sarnaselt laienduste register, mida uuendatakse regulaarselt. Praeguse seisuga register sisaldab 68 laiendust, millest mõne autoriks on olnud Flaski meeskond või Armin Ronacher ise. Niivõrd suur laienduste olemasolu saab tagada veebirakendusele palju erinevaid funktsionaalsusi. Andmebaaside osas on samuti olemas Flask-SQLAlchemy (SQLAlchemy) laiendus (Ronacher, 2018).

3.2.2 Mallid

Mallimootor lubab arendajatel tekitada soovitud sisutüüpe. Märgistatud stringidest moodustatakse renderdatud stringid ja väljundis on märkide asemel väärtused. Malle kasutatakse tavaliselt vaheformaadina, et tekitada väljundina HTML, XML või PDF formaadid (Makai, 2015).

Bottle'l on olemas kiire ja võimas sisseehitatud mallimootor nimega SimpleTemplate Engine. Malli renderdamiseks kasutatakse **template()** funktsiooni (vt Koodinäide 5) või **view()** dekoraatorit (vt Koodinäide 6). Arendaja peab kõigest defineerima malli nime ja muutujad, mida saata mallile võtmesõnade muutujatena. Funktsiooniga **template()** laetakse fail **hello_template.tpl** ja renderdatakse vastavalt **name** muutujahulgaga. Samas **view()** dekoraator laseb kasutajal tagastada sõnastikku koos mallimuutujatega (Hellkamp, 2018).

[8] Bottle: List of Available Plugins: http://bottlepy.org/docs/dev/plugins/index.html
[9] Flask Extensions: http://flask.pocoo.org/extensions/

```
@route('/hello')
@route('/hello/<name>')
def hello(name='World'):
    return template('hello_template', name=name)
```

Koodinäide 5. Malli renderdamine template() funktsiooniga

```
@route('/hello')
@route('/hello/<name>')
@view('hello_template')
def hello(name='World'):
    return dict(name=name)
```

Koodinäide 6. view() dekoraatori kasutamine mallimootoris.

Juhul kui arendaja ei soovi kasutada sisseehitatud SimpleTemplate Engine-nimelist mallimootorit, siis Bottle'l on olemas tugi teistele mallidele nagu Mako[10], Jinja2 ja Cheetah[11] (Hellkamp, 2018).

Flaskis puudub sisseehitatud mallimootor ja kasutab automaatselt hoopis Jinja2't. Muidugi arendaja võib kasutada teist mallimootorit, aga Flaski käivitamiseks peab paigaldama nimelt Jinja2 mallimootori. Arendaja ei pea selle kasutamiseks midagi tegema, kuna Flask seadistab selle ise ära. Malli renderdamiseks Flaskis kasutatakse **render_template()** meetodit (vt Koodinäide 7). Sarnaselt Bottle'le tuleb anda malli nimi ja muutujad, mis seejärel edastatakse mallimootorile võtmesõnade muutujatena (Ronacher, 2018).

```
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Koodinäide 7. Malli renderdamine render_template() meetodiga Flaskis.

Meetodid **template()** Bottle's ja **render_template()** Flaskis toimivad sama põhimõttega. Mallide osas tuleb pigem taas rõhutada, et Bottle's on sisseehitatud mallimootor ja tugi Jinja2, Cheetah ja Mako mallidele. Flaskis on ainult Jinja2 mall. Kuigi Jinja2 on üks

[10] Mako: http://www.makotemplates.org/
[11] Cheetah: http://cheetahtemplate.org/

suurima kasutusega mallimootoreid Pythonil, siis pakub Bottle raamistik sellegipoolest rohkem valikuvõimalust (Ronacher, 2018).

3.2.3 Marsruutimine

Tänapäevastel veebirakendustel on ilusad URLid, mis aitab inimestel neid meelde jätta. See tuleb eriti kasuks siis, kui rakendust kasutatakse mobiiltelefonidel. Kasutaja seisukohalt on oluline, et saab minna otsekohe soovitud leheküljele nii, et pole vaja vajutada indeksilehele (Ronacher, 2018).

Marsruutide osas eksisteerib mitu erinevat lahendust ja see jääb puhtalt arendaja otsustada, kuidas ta soovib projekti teha. Nii Bottle'i kui ka Flaski puhul neist kõige põhilisem on **route()** dekoraator, mis on eelnevalt „*Hello world*" näites välja toodud. Kuid peale selle on teisi lahendusi, milleks on URLi dünaamilisemaks muutmine või mitmede reeglite sidumine sellega (Hellkamp, 2018).

Bottle mikroraamistikus on marsruutide lahendused järgmised: (Hellkamp, 2018)

- **Dünaamilised marsruudid –** marsruudid, mis sisaldavad metamärki ja võivad korraga samastuda rohkem kui ühe URLiga. Lihtne metamärk koosneb noolsulgude sees asuvast nimest (näiteks `<name>`) ja aktsepteerib viimase kaldkriipsuni veel teisi tähemärke. Näiteks marsruut `/hello/<name>` aktsepteerib selliseid päringuid nagu `/hello/alice` kui ka `hello/bob`, kuid mitte `/hello`,`/hello` või `/hello/mr/smith`.
- **HTTP päringute meetodid –** HTTP protokoll defineerib mitu päringu meetodit erinevate eesmärkide jaoks. Kõikide marsruutide jaoks on vaikimisi määratud GET meetod. Kuid leidub veel teisi meetodeid nagu POST, PUT, DELETE ja PATCH. Kõiki neid meetodeid saab siduda **route()** dekoraatoriga. Näiteks POST meetodit kasutatakse HTMLis väljade täitmisel ning saatmisel. Muuhulgas, saadaval on veel kaks spetsiaalset meetodit nagu HEAD ja ANY.
- **Staatilised failid** – staatiliste failide nagu piltide või CSS failide puhul peab lisama marsruudi ja tagasikutse, et mis faili võtta ning kust neid leida.
- **Veateate leht** – kui midagi läheb valesti, siis Bottle näitab vaikimisi informatiivset, kuid küllaltki lihtsat veateate lehte. Kui kasutada **error()** dekoraatorit, siis saab

veateadet ümber vahetada spetsiifilise HTTP staatuse koodi vastu ja kuvada kasutajale hoopis veateadet sisuga näiteks: „*Sorry, nothing to see here.*“

Flaski mikroraamistikus on marsruutide lahendused on järgmised: (Ronacher, 2018)

- **Muutujate reeglid** (ingl k „*variable rules*“) – URLi lõppu saab lisada võtmesõna argumendi. Tüüpiline näide sellest on näiteks kasutajanime näitamine URLi lõpus, kui kasutaja vaatab enda profiili. Lubatud on järgnevad teisendustüübid: *string, int, float, path, any* ja *uuid*.
- **Unikaalsed URLid / Käitumine ümbersuunamisel** (ingl k „*redirecting behaviour*“) – Flaski URLi reeglid põhinevad Werkzeugi marsruudimoodulil, mille eesmärk on tagada ilusaid ja unikaalseid nimesid. Sellega tagatakse näiteks kaldkriipu „/“ olemasolu URLi lõpus. Teatud juhtudel, kui koodilõigus defineeritakse URLi linki ilma kaldkriipsuta, siis tekib 404 veateade „*Not found*.“ Ümbersuunamise käitumine tagab selle, et URLid töötaksid kaldkriiputa ka edasi.
- **URLi konstrueerimine –** URLi konstrueerimine spetsiifilise funktsiooni otsa, mis aktsepteerib esimese argumendina funktsiooninime ja seejärel mingi arv võtmesõnade argumente, millest igaüks vastab URLis muutuja kohale.
- **HTTP päringute meetodid –** vaikimisi on marsruutide jaoks määratud GET meetod. Sarnaselt Bottle’i raamistikuga saab seda muuta teiste meetodite vastu nagu HEAD, POST, PUT, DELETE ja OPTIONS.
- **Staatilised failid** – dünaamilised veebirakendused nõuavad staatilisi faile ja siin tulevad mängu JavaScripti ja CSS failid. Selle meetodiga saab failinime lisada URLi lõppu.

3.2.4 Silur

Iga arendaja on programmeerimises pidanud koodilõigust otsima vigu. Vigade silumine on protsess, milles üritatakse üles leida viga ja võtta ette sammud selle parandamiseks, et lõpuks programmi veatult välja lasta. Vea leidmiseks on mitu viisi. Arendajal on võimalus ise kood läbi uurida lause haaval, kasutada veebilehtede puhul *Inspect* elementi või käivitada selleks silur (Akter, 2016).

Bottle's on olemas silumisrežiim, mis on varajases arenduses väga kasulik. Silumisrežiimi sisse lülitamisel (vt Koodinäide 8) annab Bottle vea silumiseks igal korral väga kasulikku informatsiooni. Silur lülitab välja mõned optimisatsioonid, mis võivad programmi tööd häirida ja hoiatab mõne halva konfiguratsiooni eest. Silumisrežiimis võivad olla järgmised muutused nagu pluginaid rakendatakse otsekohe, malle ei hoita vahemälus ja vaikimisi vealehel näidatakse tagasijälitust (ingl k „*traceback*“) (Hellkamp, 2018).

```
bottle.debug(True)
```

Koodinäide 8. Silumisrežiimi sisselülitamine Bottle's.

Bottle püüab kinni kõik erindid programmikoodis selle nimel, et WSGI server ei jookseks kokku. Kui aga tundub, et **debug()** režiimist ei piisa ja vahevara silumiseks on vaja midagi muud, saab selle välja lülitada. Selleks on Bottle's olemas käsud, millega Bottle sisseehitatud silur otsib ainult vigu enda seest ja ülejäänu eest hoolitseb vahevara (Hellkamp, 2018).

Flaskiga on samuti silur kaasas, kuid selle kontseptsioon on natuke teistsugune. Flaski skripti tuleb iga kord käsitsi uuesti laadida, mille eest hoolitseb Flask. Kui silumisrežiim sisse lülitada, siis server laeb ennast uuesti iga koodimuudatuse korral ja kui endiselt leidub viga, aitab silur välja. Silumisrežiimi sisse lülitamiseks tuleb keskkonnamuutuja `FLASK_DEBUG` eksportida enne serveri tööle panekut (vt Koodinäide 9) (Ronacher, 2018).

```
$ export FLASK_DEBUG=1
$ flask run
```

Koodinäide 9. Silumisrežiimi sisselülitamine Flaskis.

Windowsi versioonil tuleb `export` asemel kirjutada `set`. Kuid selle koodinäite sees juhtub kolm järgnevat asja: (Ronacher, 2018)

- Aktiveeritakse silur.
- Aktiveeritakse automaatne taaskäivitaja (ingl k „*reloader*“).
- Flaski rakenduses aktiveeritakse silumisrežiim.

3.2.5 Sisseehitatud arendusserver

Nii Bottle'is kui ka Flaskis on **run** käsk, mis käivitab kohaliku arendusserveri. Sisseehitatud arendusserver loob ühenduse serveri, veebilehitseja ja Pythoni koodi vahel (Hellkamp, 2018; Ronacher, 2018).

Bottle's on vaikimisi sisseehitatud wsgiref WSGIServer [12] , mis kujutab endast mittelõimelist (ingl k „*non-threading*") HTTP serverit ja see sobib väga hästi arenduseks. Aga kui serveri koormus muutub liiga suureks, võib esineda jõudluses kitsaskoht. Dokumentatsioonis soovitatakse alustada korraga mitu serveriprotsessi ja hajutada koormust. Kuid sellest lihtsam variant on paigaldada hoopis mitmelõimeline serveriteek. Seepärast Bottle's ongi olemas tugi järgmistele WSGI'd võimaldavatele HTTP serveritele nagu fapws3[14], bjoern[15] või cherrypy[16] jne (Hellkamp, 2018).

See-eest Flaski sisseehitatud arendusserver on kergekaaluline ja kasutamiseks lihtne, aga pole tootmiseks kõlblik, kuna see ei suuda hästi skaleerida ja teostab vaikimisi ainult ühe päringu korraga. Seetõttu veebiserveri püsti panemiseks on vaja kasutada järgmisi *host* teenuseid nagu Heroku[17], OpenShift[18], Webfaction[19] jm. Samas kui arendaja soovib ise hostida, saab ta valida näiteks mod_wsgi (Apache)[20], uWSGI[21], FastCGI[22] ja CGI[23] vahel (Ronacher, 2018).

3.2.6 Testimine

Flaski koduleheküljel on tsitaat, mille päritolu ja allikas on teadmata, kuid see kõlab nii: „Miski, mida pole testitud, on katkine." Selle väitega saavad ilmselt kõik arendajad nõustuda. Testimine on arendusprotsesis oluline etapp, sest kui rakendusi ei testita, pole võimalik ka olemasolevat koodi paremaks muuta (Ronacher, 2018).

[12] wsgiref WSGIServer: https://docs.python.org/3/library/wsgiref.html#module-wsgiref.simple_server
[14] Fapws3: https://github.com/william-os4y/fapws3
[15] Bjoern: https://github.com/jonashaag/bjoern
[16] Cherrypy: https://cherrypy.org/
[17] Heroku: https://devcenter.heroku.com/articles/getting-started-with-python#introduction
[18] OpenShift: https://help.openshift.com/
[19] WebFaction: http://flask.pocoo.org/snippets/65/
[20] mod_wsgihttp://flask.pocoo.org/docs/0.12/deploying/mod_wsgi/
[21] uWSGIhttp://flask.pocoo.org/docs/0.12/deploying/uwsgi/
[22] FastCGI: http://flask.pocoo.org/docs/0.12/deploying/fastcgi/
[23] CGI: http://flask.pocoo.org/docs/0.12/deploying/cgi/

Rakenduse testimisel pakub Flask välja lahenduse, milles kasutatakse Werkzeugi testi **Client**-nimelist klassi kohaliku konteksti haldamiseks. Muidugi jääb arendaja otsustada, millist testimise lahendust kasutada. Flaski puhul kasutatakse selleks tarbeks **unittest** paketti, mis tuleb vaikimisi Pythoni paigaldamisega kaasa (Ronacher, 2018).

Kuigi Bottle's on Pythoni paigaldamise kaudu samuti **unittest** pakett olemas, ei tööta sellega testimine nii nagu peaks. Üksuste testimist rakendatakse veebirakenduses defineeritud meetoditele, kuid WSGI keskkonda ei käivitata. Selleks, et WSGI rakendusi saaks testida reguleeritud keskkonnas, tagasijälitustega ja siluri abiga, soovitatakse Bottle'i koduleheküljе sõnul kasutada WSGI Testing Tools'i (Hellkamp, 2018).

3.2.7 Turvalisus

Edasiarengud veebirakenduste ja teiste tehnoloogiate valdkonnas on muutnud ligipääsu tagamist informatsioonile. Paljud ettevõtted on oma tegevusega üle läinud võrgu peale. Modernse Web 2.0 ja HTML5 veebirakenduste kaudu on muutunud isegi tarbijate nõudlus, kuna informatsioonile soovitakse igal ajal ligipääsu. Seetõttu ettevõtted (näiteks pangandussüsteemid, internetipoe veebilehed jm) on hakanud informatsiooni tagama läbi võrgu veebirakendustega. Edasiarengud veebirakenduses on tõmmanud ligi häkkereid ja pettureid, seetõttu on veebirakendustes hakanud suuremat rolli mängima turvalisus, kuna väiksemgi turvaauk võib tekitada ettevõttele suurt kahju (Netsparker Ltd, kuupäev puudub).

Veebirakendused peavad silmitsi seisma igasuguste turvaprobleemidega ja kõike on keeruline arvesse võtta. Flask lahendab arendaja jaoks murdskriptimise ja päringuvõltsingu, kuid ülejäänu eest peab arendaja ise hoolitsema. Murdskriptimine (ingl k „*cross-site scripting"* ja tähistatakse lühendina XXS) seisneb meelevaldselt HTMLi ja koos sellega JavaScripti süstimises veebilehe konteksti sisse. Selle vältimiseks arendajad peavad korralikult muutma koodi, et need ei sisaldaks teatud HTML märgendeid. Flask seadistab Jinja2 vaikimisi selliselt, et välistatakse murdskriptimisega seotud probleemid, aga tuleb olla ettevaatlik teiste asjade suhtes, milleks on näiteks HTMLi genereerimine Jinja2 abita või üles laaditud failidest tekitatud tekstifailide saatmine (Ronacher, 2018).

Teine suur probleem on päringuvõltsing (ingl k „*cross-site request forgery"* ja tähistatakse lühendina CSRF), mis on vägagi keeruline teema. See on olemuselt rünne, mis petab

laadima veebilehte, mille kaudu tehakse veebirakenduse kasutaja nimel ja õigustega mingi kahjulik toiming. Päringuvõltsingut tehakse väga palju vormide valideerimise raamistikes, seega Flaski rakendustes ei esine seda probleemi tihti (Andmekaitse ja infoturbe leksikon, kuupäev puudub; Ronacher, 2018).

Autor leiab Bottle koduleheküljelt uurimise kaudu, et Bottle ei võta turvalisust silmas pidades ette ühtegi sammu.

3.2.8 Tööriistad

Bottle'l on mugav ligipääs tööriistadele, millega saab veebirakendustes sooritada erinevaid tehinguid. Nendele pääseb ligi globaalse **request** objekti kaudu. Nendest kasutakse kõige enam järgmiseid tööriistu: (Hellkamp, 2018)

- **FormsDict** – Bottle kasutab vormiandmete ja küpsiste talletamiseks spetsiaalset tüüpi sõnastikku. FormsDict käitub nagu normaalne sõnastik, aga sellesse on lisatud funktsionaalsused, mis teevad arendaja elu lihtsamaks.
- **HTML <form> haldus**
- **HTTP päised** – kõiki klientide poolt saadetud HTTP päiseid talletatakse WSGIHeaderDict sees. See on põhimõtteliselt mittetõstetundlike võtmetega sõnastik.
- **Päringu muutujad** (ingl k „*query variables*“)– kasutatakse tavaliselt väikse arvu võtme/väärtuste paaride saatmiseks serverile (nagu näiteks `/forum?id=1&page=5`)
- **Failide üleslaadimine** – HTMLi `<form>` sees peab olema atribuut `enctype="multipart/form-data"`
- **Küpsised** – need on väiksed andmeblokid, mida talletatakse veebilehitsejas ja saadetakse iga päringuga tagasi serverile. Need on oleku säilitamiseks kasulikud rohkem kui ühe päringu puhul, kuid neid ei tasu kasutada turvalisuse mõttes. Kõik kliendi poolt saadetud küpsised on saadaval **BaseRequest.cookies** kaudu. All asuvas koodinäites (vt) on lihtne näide küpsiste-põhise vaate loendurist.
- **JSON sisu** – mõned JavaScripti või REST kliendid saadavad serverile `application/json` sisu. Sellisel juhul **BaseRequest.json** atribuut sisaldab saadavuse korral parsitud andmestruktuure.

```
from bottle import route, request, response
@route('/counter')
def counter():
    count = int( request.cookies.get('counter', '0') )
    count += 1
    response.set_cookie('counter', str(count))
    return 'You visited this page %d times' % count
```

Koodinäide 10. Küpsiste loendur

Veebirakenduste puhul on tähtis reageerida andmetele, mida klient saadab serverile. Selle informatsiooni tagab globaalne **request** objekt. Flaskis on samuti saadaval erinevaid vahendeid, millest mõned katuvad Bottle'i omadega, aga samas pakub teisi asju. Nendest tuleb välja tuua järgmised: (Ronacher, 2018)

- **Failide üleslaadimine** – failide üleslaadimine Flaskis on tehtud sama lihtsaks nagu Bottle'i ehk teisisõnu `<form>` sees peab olema järgnev atribuut `enctype="multipart/form-data"`
- **Küpsised** – küpsistele ligi pääsemiseks saab kasutada `cookies` atribuuti.
- **Veateated ja ümbersuunamine** – Flaskis on kaks funktsiooni – **redirect()** ja **abort()**. Esimene funktsiooniga saab kasutajat suunata teisele lehele ning teise funktsiooniga saab päringut katkestada, kuid see ei oma üldiselt praktilist otstarvet.
- **Sessioonid** – sessioone rakendatakse koos küpsistega ja märgistatakse neid krüptograafiliselt. See tähendab, et kasutaja saaks vaadata küpsiste sisu, kuid mitte seda modifitseerida. Vastasel juhul peab neil olema nende märgistamiseks kasutatud salajane võti. Muuhulgas sessioonide kasutamiseks peab olema samuti salajane võti.
- **Plaanid** (ingl k „*blueprint*“) – operatsioonide kogum, mida saab rakenduse sisse panna. Need on lühidalt öeldes plaanid, mis aitavad hõlbustada mõne modulaarrakenduse programmeerimist.

3.2.9 Unicode

Arvutisüsteemides tähemärke transformeeritakse ning talletatakse arvude ehk bittide jadana, mida haldab protsessor. Kuna on vaja teha märgikooditeisendust bittidest tähemärkideni ja vastupidi, arendati välja koodikomplektid. Enne Unicode'i kasutusele võtmist oli üle maailma sadu erinevaid kodeeringute skeeme, millest paljud sisaldasid kõigest 256

tähemärki. Kuigi see oli kompaktne, tekkisid probleemid näiteks ideograafiliste märkide kogumitega nagu hiina või jaapani keel ning paljude keelte tähemärkide kogumid ei olnud võimelised koos eksisteerima üksteisega. Unicode on katse neid kõiki erinevaid skeeme mahutada ühe universaalse tekstikodeeringu standardi sisse. UTF-8 on kodeeringuskeem, mida kasutatakse vaikimisi laialdaselt igal pool Internetis standardina. W3C sätestab samuti, et kõik XML protsessorid peavad lugema UTF-8 ja UTF-16 kodeeringuid (Bahjat, kuupäev puudub).

Bottle kasutab `Content-Type` päise märgistiku parameetrit, millega määratakse Unicode stringide kodeeringut. Päises on vaikimisi on määratud `text/html; charset=UTF8` ja seda saab vastavalt soovile muuta (Hellkamp, 2018).

Flask on täielikult Unicode-põhine. Jinja2 ja Werkzeugi teegid, mille najal on Flask üles ehitatud, ning enamik veebiga seotud Pythoni teegid tegelevad tekstiga. Flask annab elementaarset Unicode tuge ja pakub eelduse põhjal järgmisi asju:

- Veebilehel tekstilehe kodeering on UTF-8.
- Üksnes teksti jaoks kasutatakse seespidiselt alati Unicode'i. Erandiks on ASCII tähemärgipunkte sisalduvad stringiliteraalid.
- Kodeerimine ja dekodeerimine juhtub iga kord, kui protokoll nõuab baitide saatmist.

Teisisõnu Flask eeldab kõigest, et mis puudutab Unicode'i, siis arendaja soovib UTF-8 kodeeringut ja hoolitseb selle üle kasutaja eest (Ronacher, 2018).

3.3 Dokumentatsioon

Dokumentatsioon on raamistike puhul äärmiselt oluline osa, mis aitab arendajatel seda tundma õppida ja rakendada programmeerimises. Dokumentatsioon peab olema lihtne, täpne, kergesti loetav ja arusaadav. Üldiselt peaks dokumentatsioon algama juba raamistiku arenduse algusest saadik ja täienema järjepidevalt selle arenduse käigus. Igasugused näited, ülevaated, selgitused ja muu on oluline osa dokumentatsioonist (Chen & Xu, kuupäev puudub).

Bottle dokumentatsioon on jaotatud ühtlaselt kolmeks peatükiks (Hellkamp, 2018). Esimene peatükk on dokumentatsiooni põhiosa ehk kasutusjuhend, mille sees on õpetus, konfiguratsioon, marsruutimine, SimpleTemplate mallimootor, levitamine, API viited ja toetatud pluginate list. Teine peatükk on teadmistebaas, mis sisaldab spetsiiflisemat õpetust, näiteid konkreetsema rakendusetüübi kohta, korduma kippuvaid küsimusi ja selgitust, kuidas muuta veebirakendus asünkroonseks. Viimane peatükk on üleüldisem dokumentatsioon raamistiku olemuse kohta, mis sisaldab iga uue versiooniga välja tulnud muutustelogi, arendajate märkusi ja pluginate arenduse juhendit. Autor leiab, et Bottle dokumentatsioon on kompaktne, konkreetne ja üldisematele probleemidele leiab kiiresti lahenduse. Autori silmis on oluline rõhutada, et esimese peatükiga tutvumine loob selge arusaama, kuidas Bottle töötab.

Flaski dokumentatsioon on üles ehitatud teise põhimõttega (Ronacher, 2018). Sarnaselt Bottle'ga on see jaotatud kolmeks peatükiks, mis eristuvad üksteisest selgelt. Esimene peatükk on kasutusjuhend, mis on võrreldes viimase kahe peatükiga väga pikk. Selles on kõik Flaski raamistikku selgitavad õpetused sees nagu raamistiku paigaldamine, kasutusjuhend (silumisrežiim, marsruutimine, staatilised failid, sessioonid jm), õpetus (kaustade koostamine, andmebaasidega ühendamine jm), mallid, testimine, rakenduste veateated, siluri veateated, signaalid, plugina vahetuste vaade, päringute kontekst, plaanidega modulaarrakendused, Flaski laiendused, käsurea kasutajaliides, arendusserver, levitamise valikud ja mustrid erinevate ülesannete jaoks nagu AJAXi rakendamine koos Jquery'ga või konteksti voogedastamine. Teine peatükk koosneb API viidetest, milles on pikk nimekiri erinevate funktsioonide kohta. Nendeks on näiteks JSON tugi, klassidepõhine vaade, sessiooni kasutajaliides jne. Viimane peatükk sisaldab legaalset informatsiooni, disainiga seotud märkmeid ja muutustelogi.

Kuna Flask pakub Bottle'st hulgaliselt rohkem võimalusi, on arusaadav, miks selle kasutusjuhend on dokumentatsioonis nii mahukas. Bottle dokumentatsioon oli kompaktne ja konkreetne, aga mõnest alampeatükist oli autori jaoks raske aru saada. Näiteks autor pidi lugema mõne üldise arvutisüsteemis kasutatava termini kohta eraldi kuskilt mujalt. Autor leiab, et Flaski dokumentatsioon on lugemiseks kergem. Komponentide kasutuste ning nende selgituste kohta on arusaadavamalt ja laialdasemalt kirjutatud. Flaski dokumentatsioonis on lühidalt kirjas selgitused kõrvaliste terminite kohta, mis ei puuduta Flaski raamistikku, vaid pigem üleüldiselt arvutisüsteeme. Nende lisamine teeb

dokumentatsiooni mõnevõrra mahukamaks, kuid samas aitab kergemini aru saada ja ei pea eraldi otsima mujalt.

Kuid oluline on silmas pidada, et koduleheküIgedel asuv dokumentatsioon pole alati ainus koht, millele lähtudes saab raamistikku selgeks õppida. Internetis leidub tavaliselt hulgaliselt õpetusi, mis teatud juhtudel pakuvad rohkem selgitusi ja toovad näidetena midagi konkreetsemat, milleks võib olla näiteks spetsiifilisema veebirakenduse ülesehitus jne. Autor avastas Bottle ja Flaski mikroraamistike kohta uurimise käigus, et Flaski kohta leidub hulgaliselt rohkem õpetusi kui Bottle'le. Mitmes andmebaasis on olemas raamatud Flaski kohta, lisaks sellele leidub Google'i kaudu otsides hulgaliselt blogisid ja veebilehti. See ei tähenda muidugi, et Bottle'i kohta puuduvad sarnased õpetused, kuid õpetuste mahu ja saadavuse vahe on märgatav. Sellest saab järeldada, et arendajate seas on Flask leidnud rohkem kasutust.

3.4 Internatsionaliseeriumine ja lokaliseerumine

Autor selgitab i18n ja L10n mõisted lahti, sest lugeja jaoks võib tekkida nendel vahet tegemisega raskusi. Internatsionaliseerimine (kutsutakse mõnikord ka globaliseerimiseks) ehk **i18n** on tarkvaratoodete projekteerimine algusest peale selliselt, et neid oleks hõlbus kohandada eri keeltele ja piirkondadele ning et selleks poleks vajadust teha hiljem põhimõttelisi muudatusi (e-Teatmik: IT ja sidetehnika seletav sõnaraamat, kuupäev puudub; Ishida, Miller, Boeing, & W3C, 2015).

Lokaliseerimine, mida Internetis tähistatakse lühendina **L10n**, on tarkvara keeleline ja kultuuriline kohandamine kasutuskohale eemärgiga muuta see kasutajasõbralikumaks. Lokaliseerimise juurde kuulub näiteks kasutajaliidese tõlkimine, klaviatuurikasutuse, fontide, kuupäeva ja kellaaja vormingu, rahaühikute jms muutmine (e-Teatmik: IT ja sidetehnika seletav sõnaraamat, kuupäev puudub).

Internatsionaliseerimine mõjutab märgatavalt toote lokaliseerimise lihtsust, sest lingvistilise- ja kultuurikeskset tarkvara on globaalsele turule tagurpidi kohandada märksa raskem ja aeganõudvam protsess. Ideeliselt internatsionaliseerimine on juba disaini- ja arendusprotsessis fundamentaalne samm, mitte järelmõte pärast toote levitamist (Ishida et al., 2015).

Bottle mikroraamistikus pole ei internatsionaliseerimist ega ka lokaliseerimist esindatud. Esialgu need puudusid ka Flaskis, kuid Flaski looja Armin Ronacher arendas välja laienduse nimega Flask-Babel, mis lisab internatsionaliseerimise ja lokaliseerimise toe igale Flaski rakendusele. Flask-Babel sisaldab sisseehitatud tuge kuupäevavormingute ning ajatsoonide puhuks ja kerget kasutajaliidest **gettext** tõlgenduste jaoks (Bogoev, 2016; Ronacher, kuupäev puudub).

3.5 Asünkroonsus

Veebirakendustes kasutatakse AJAXit kasutajakogemuse täiustamise eesmärgil. Mõned blogid kasutavad näiteks lõputut hiirega kerimise tehnikat, mis muudab lehekülgedeks jaotuse hoopis nii, et veebilehte ei pea uuesti laadima ja AJAXi kutsed toovad esile järgmised andmed. Asünkroonse põhimõtte peale on üles ehitatud näiteks Twitter. Pikemas perspektiivis säästab see meetod ülekandekiirust, sest kasutaja ei pea tõmbama tervet HTML lehte, vaid kõigest toorandmeid läbi JavaScripti. Nüüd peetakse AJAXit arendajate hulgas standardseks töökorraks (Rocheleau, 2016).

Asünkroonsed disainimustrid ja sünkroonse loomuga WSGI ei sobi omavahel väga hästi kokku. Bottle on samuti WSGI raamistik, mis jagab WSGI sünkroonset loomust, kuid **gevent**[24]-nimeline projekt võimaldab Bottle raamistikul kirjutada asünkroonseid rakendusi sisuliselt kolmel olulisel põhjusel: (Hellkamp, 2018)

- **Greenlets** – Enamikes serverites nõuab samaaegsete harude (ingl k „*thread*“) loomine palju ressurssse, siis **gevent** moodul lisab sisse **greenlets**id. Need käituvad sisuliselt samamoodi nagu harud, aga neid on odav teha. Geventi-põhise serveriga saab korraga koostada tuhandeid greenlets’e (iga ühenduse jaoks üks), mis ei koorma uute päringute aktsepteerimisega serverit ja samaaegsete ühenduste arv saab olla virtuaalselt piiramatu.
- **Tagasikutse** (ingl k „*callback*“) – asünkroonsetes raamistikus on levinud disainimuster kasutada mitteblokeerivaid APIsid ja siduda tagasikutsed asünkroonsete sündmuste külge. Soklit (ingl k „*socket*“) hoitakse lahti nii kuni suletakse arendaja poolt eesmärgiga, et hiljem lubada uusi tagasikutseid kirjutada

[24] Gevent: http://www.gevent.org/contents.html

samasse soklisse. Päringute töötleja lõpetatakse varakult, et see saaks edasi liikuda ja aktsepteerida uusi päringuid samal ajal, kui tagasikutsed jätkavad eelmiste päringute sokkelite kirjutamisega. Kuid Geventi ja WSGI kombineerimise tulemusel muudetakse asjaolusid. Enam pole mõtet töötlejat varakult lõpetada, kuna nüüd on lõpmatu arv harusid, mis saavad aktsepteerida uusi ühendusi. Töötleja varajase lõpetamisega samuti suletakse sokkel ja itereeritava peab kohandama WSGI jaoks.

- **Websocket –** mõlemasuunaline ühendus veebilehitseja (klient) ja veebirakenduse (server) vahel. Geventi sees asuv **gevent-websocket** pakett teeb arendaja eest raske töö ära.

Gevent pakett teeb arendaja eest palju ära, kuid muidugi on võimalus teha veebilehte AJAXi ja JQuery abil (Tiram, 2015).

Flask pole samuti disainitud suurte rakenduste või asünkroonsete serverite jaoks, kuna selle eesmärgiks on arendada kiirel ja kergel moel traditsioonilisi veebirakendusi. Paraku pole Flaskis veebirakenduse asünkroonseks muutmisega vastavat paketti nagu gevent pakett Bottle's, kuid Flaskil on selleks puhuks teine meetod. Selleks saab kasutada kolmanda osapoole arendatud laiendust nimega Flask-Sijax. See on Pythoni ning JQuery teek, mis teeb AJAXi kasutuse kergeks. Asünkroonset veebirakendust on võimalik kirjutada AJAXi ning JQuery kombineerimisel ja Flaski koduleheküljel on selle jaoks olemas detailsem õpetus (Ronacher, 2018).

3.6 Raamistike võrdluse kokkuvõte

Autor leidis, et hindamiskriteeriumid nagu lihtsus, mugavus ja muu sarnane on liiga subjektiivne. Seetõttu keskendus autor rohkem sellele, mida mõlemad raamistikud pakuvad, mis nende sees on ja mis on olulisemad funktsioonid. Dokumentatsiooni läbi töötamise, koodinäidete testimise ning uurimise tulemusel tekkis autoril ülevaade mõlema raamistiku kohta.

Autori silmis on Bottle ja Flask üldplaanis peaaegu samaväärsed, kuna Bottle ja Flask jagavad palju ühiseid funktsioone. Teisest küljest pakub Flask teatud aspektides rohkem võimalusi, millest olulisena tasub välja tuua turvalisuse, suure koguse laiendusi,

internatsionaliseerimise ja lokaliseerimise. Ning nagu korduvalt mainitud, saab puuduvad komponendid kas pluginate või laienduste kaudu projekti kaasata. Kui autor peaks isikliku veebirakenduse projekti tarvis valima ühe raamistiku, siis ta valiks Flaski eelmainitud põhjusel, et pakub rohkem võimalusi. Lisaks sellele leidub rohkem Flaskiga seotud õpetusi, õpikuid ja materjale, mis aitab seda veelgi süvendatult tundma õppida ja abistada projektis. Autor leiab, et Flaski raamistikku saab edukalt veebirakenduse vallas kasutada, samas Bottle raamistik on sobilikum ideede prototüüpimisel.

Alljärgnev tabel (vt Tabel 1) iseloomustab Bottle ja Flaski raamistiku kokkuvõtvat võrdlust.

Tabel 1. Ülevaatlik tabel Bottle ja Flaskist

	Bottle	Flask
Pythoni versiooni nõuded	Versiooni 0.12 puhul: 2.5 või sellest uuem 3.x Versioon 0.13 arenduses (toetab alates 2.7 ja 3)	Versiooni 1.0.2 puhul: 2.7 3.4 või sellest uuem
Mallid	Sisseehitatud SimpleTemplate Engine, Cheetah, Jinja2, Mako	Jinja2
Marsruudid	• **route()** dekoraator • dünaamilised marsruudid • HTTP päringute meetodid • staatilised failid • veateate leht	• **route()** dekoraator • muutujate reeglid • HTTP päringute meetodid • staatilised failid • unikaalsed URLid / ümbersuunamise leht • URLi konstrueerimine
Arendusserver	Sisseehitatud wsgiref WSGIServer. Olemas tugi teistele serveritele nagu paste, fapws3, cherrypy	Nõrk arendusserver, soovitatav kasutada *host* teenuseid nagu Heroku, OpenShift, Webfaction või alternatiivselt mod_wsgi, uWSGI, FastCGI või CGI'd
Testimine	Unittest pakett	Unittest pakett olemas, kuid ebapraktiline. Soovitatav kasutada WSGI Testing Tools'i
Unicode	Vaikimisi määratud UTF-8	Täielikult Unicode-põhine
Andmebaas	Lisatav plugina kaudu nagu nt Bottle-Sqlalchemy (SQLAlchemy), Bottle-Sqlite (SQLite3)	Andmebaasi kiht puudub, lisatav laienduse kaudu nagu nt Flask-SQLAlchemy

	Bottle	Flask
Turvalisus	Ei lahenda ühtegi turvaprobleemi	Lahendab arendaja eest kahte turvaprobleemi – murdskriptimine ja päringuvõltsing
Pluginate / laienduste arv	22 pluginat	68 laiendust
i18n ja L10n	Puudub plugin	Lisatav laienduse Flask-Babel kaudu
Asünkroonsus	Vaikimisi ei ole, aga on võimalik plugina ja AJAX/JQuery kaudu	Vaikimisi ei ole, aga on võimalik laienduse ja AJAX/JQuery kaudu

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua Pythoni veebiraamistike Bottle ja Flask võrdlus.

Töö eesmärgi saavutamiseks püstitati uurimisküsimused, millele vastamise käigus esimeses peatükis andis autor ülevaate Pythoni programmeerimiskeelest ja selle ajaloost. See sisaldab Pythoni autorit, programmeerimiskeele tunnuseid, võimalusi, kasutusviise ja tekkelugu.

Teises peatükis tutvustas autor üleüldisemalt raamistiku kontseptsiooni ja mida need tavaliselt sisaldavad. Seejärel koostas ülevaate, millised raamistiku tüübid on olemas ja millisesse kategooriasse kuuluvad käesoleva bakalaureusetöö raames uuritavad Bottle ja Flask. Peatüki lõpus on eraldi alapeatükk mõlema mikroraamistiku Bottle ja Flaski kohta, mida need endast kujutavad, mis on nende tunnusjooned ja kuidas need tekkisid.

Viimases peatükis võrdles autor mõlema raamistiku kontseptuaalset erinevust, funktsioone, dokumentatsiooni ja muid iseärasusi. Võrdlemismeetodiks oli mõlema dokumentatsiooni läbi töötamine, nendes sisalduvate näidete proovimine ja nii erinevuste kui ka sarnasuste uurimine. Mõne funktsiooni puhul autor lisas koodinäite, mis hõlbustab lugejal arusaamist. Kogutud andmete põhjal autor koostas kokkuvõtva tabeli võrdlusest, milles on kirjas, mida kumbki raamistik pakub. Võrdluse käigus selgus, et Bottle ja Flask jagavad üldplaanis palju sarnasusi, ehkki Flaskis on mõnevõrra rohkem funktsioone ja võimalusi. Kui autor peaks tulevikus Pythoni veebiprojekti tarveks Bottle ja Flaski vahel valima, siis ta valiks Flaski. Flask sobib hästi veebirakenduse vallas kasutuseks, kuid Bottle on sobilikum ideede prototüüpimisel.

Kasutatud kirjandus

Akter, S. (2016). *Why Do We Debug Code?* Loetud 20.04.2018 aadressil https://sjinnovation.com/why-do-we-debug-code/

Andmekaitse ja infoturbe leksikon. (kuupäev puudub). Loetud 24.04.2018 aadressil https://akit.cyber.ee/

Bahjat, B. (kuupäev puudub). *Unicode 101: An Introduction to the Unicode Standard*. Loetud 22.04.2018 aadressil https://www.interproinc.com/blog/unicode-101-introduction-unicode-standard

Bogoev, D. (2016). *How to Localize your Flask Application*. Loetud 23.04.2018 aadressil https://damyanon.net/post/flask-series-internationalization/

Brandl, G., & Ronacher, A. (kuupäev puudub). *Pocoo Team*. Loetud 15.04.2018 aadressil http://www.pocoo.org/team/

Chen, Q., & Xu, F. (kuupäev puudub). *Framework Documentation*. Loetud 21.04.2018 aadressil https://www.cse.unr.edu/~chen_q/docu.html

Christensen, H. B. (2011). *Flexible, Reliable Software: Using Patterns and Agile Development*. Boca Raton: CRC Press. Loetud aadressil https://books.google.ee/books?id=VQaf_vOTDzMC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

e-Teatmik: IT ja sidetehnika seletav sõnaraamat. (kuupäev puudub). Loetud 23.04.2018 aadressil http://vallaste.ee/index.htm

Google Developers. (2016). *Python Introduction*. Loetud 20.04.2018 aadressil https://developers.google.com/edu/python/introduction

Habib, O. (2015). *PHP Microframework vs. Full Stack Framework*. Loetud 11.04.2018 aadressil https://blog.appdynamics.com/engineering/php-microframework-vs-full-stack-framework/

Hellkamp, M. (kuupäev puudub). *Contact*. Loetud 13.04.2018 aadressil https://bottlepy.org/docs/dev/contact.html

Hellkamp, M. (2018). *Bottle Documentation: Release 0.12*. Loetud aadressil https://bottlepy.org/

Ishida, R., Miller, S. K., Boeing, & W3C. (2015). *Localization vs. Internationalization*. Loetud 23.04.2018 aadressil https://www.w3.org/International/questions/qa-i18n

Makai, M. (2014). *Web Frameworks: Bottle*. Loetud 15.04.2018 aadressil https://www.fullstackpython.com/bottle.html

Makai, M. (2015). *Template Engines*. Loetud 28.04.2018 aadressil https://www.fullstackpython.com/template-engines.html

MoWeble. (2013). *What is Software Framework?: Types and Uses of Software Frameworks*. Loetud 11.04.2018 aadressil http://www.moweble.com/what-is-software-framework-its-uses-and-types.html

Netsparker Ltd. (kuupäev puudub). *Getting Started with Web Application Security* [ajaveebipostitus]. Loetud 18.04.2018 aadressil https://www.netsparker.com/blog/web-security/getting-started-web-application-security/

Python Software Foundation. (kuupäev puudub-a). *Python 3.6.5 documentation: General Python FAQ*. Loetud 02.04.2018 aadressil https://docs.python.org/3/faq/general.html

Python Software Foundation. (kuupäev puudub-b). *What is Python?: Executive Summary*. Loetud 15.03.2018 aadressil https://www.python.org/doc/essays/blurb/

Rocheleau, J. (2016). *What Is Ajax & How Is It Used In Modern Web Development?* Loetud 23.04.2018 aadressil http://www.vandelaydesign.com/what-is-ajax-webdev/

Ronacher, A. (kuupäev puudub). *Flask-Babel*. Loetud 23.04.2018 aadressil https://pythonhosted.org/Flask-Babel/

Ronacher, A. (2010, 3. aprill). *April 1st Post Mortem* [ajaveebipostitus]. Loetud 16.04.2018 aadressil http://lucumr.pocoo.org/2010/4/3/april-1st-post-mortem/

Ronacher, A. (2018). *Flask 1.0.2 documentation*. Loetud 17.04.2018 aadressil http://flask.pocoo.org/

Rossum, G. V. (2009). *Python v3.0.1 documentation*. Loetud 29.04.2018 aadressil https://docs.python.org/3.0/

Tiram, N. O. (2015, 17. august). *AJAX with bottle.py* [ajaveebipostitus]. Loetud 24.04.2018 aadressil http://oz123.github.io/writings/2015-08-17-AJAX-with-bottle.py/index.html

Trikha, R. (2015, 21. aprill). *The History of "Hello, World"* [ajaveebipostitus]. Loetud 18.04.2018 aadressil https://blog.hackerrank.com/the-history-of-hello-world/

Wood, S. (2015). *A Brief History of Python*. Loetud 02.04.2018 aadressil https://hub.packtpub.com/brief-history-python/

WSGI.org. (kuupäev puudub). *WSGI*. Loetud 24.04.2018 aadressil http://wsgi.readthedocs.io/

Summary

Comparison of Python's Frameworks Bottle and Flask

Bachelor's thesis

The main goal of this bachelor's thesis was to compare two Python's frameworks called Bottle and Flask.

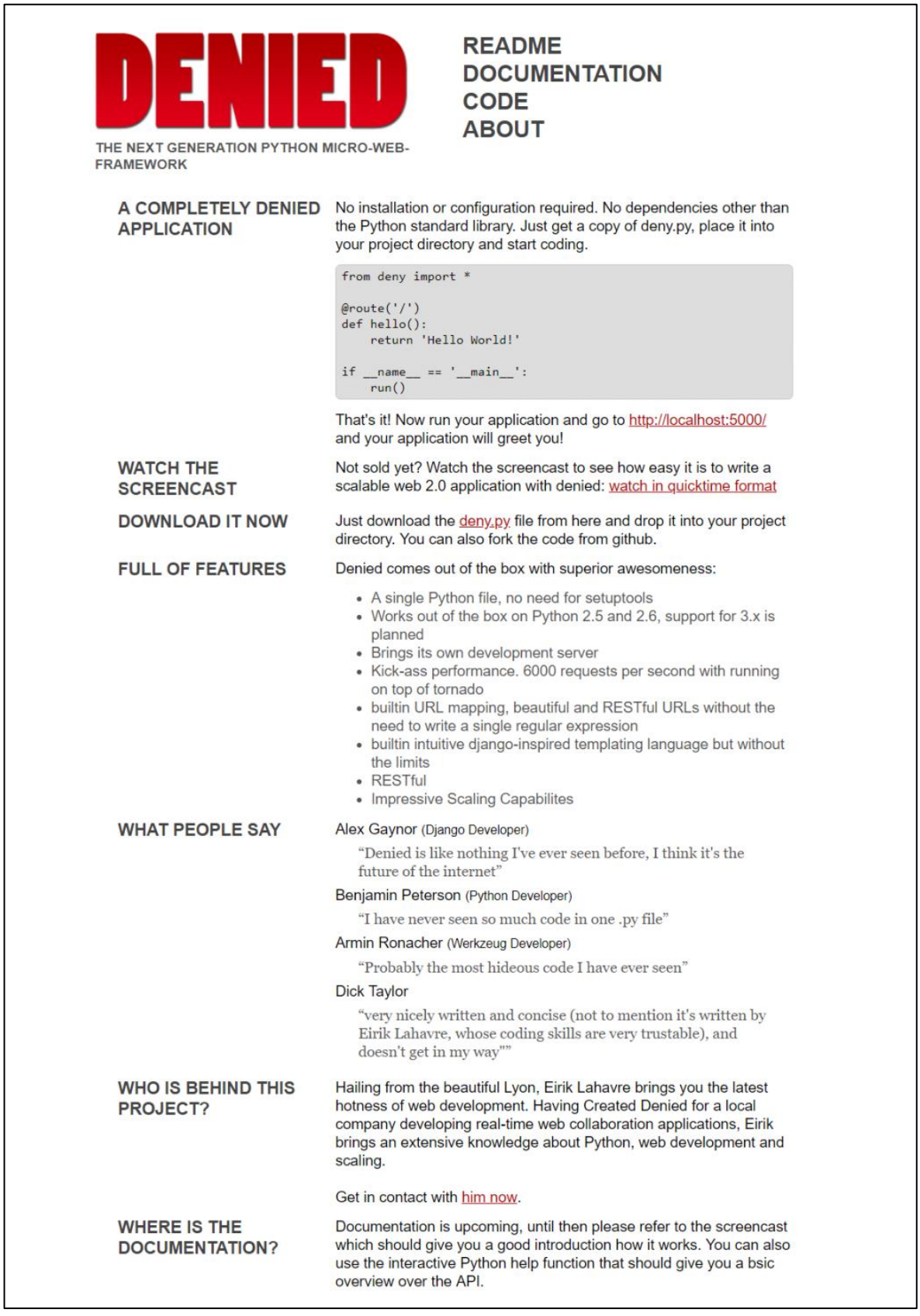
In order to achieve this purpose, research questions were put up for which provided an overview of Python as a programming language and its history in the first chapter. It contains various features, opportunities and common uses of Python programming language as well as the author behind Python and how was it developed.

The second chaper introduced an overall concept of a framework and what they include. An overview of various types of frameworks was created and it was also specified in which category do frameworks in this thesis such as Bottle and Flask belong to. In the end of this chapter there are separate subchapters for both Bottle and Flask which introduce them in general, how they were made and what are their features.

An author compared the conceptual difference, functions, documentation and other features of both frameworks in the last chapter. In order to create a comparison, an author studied and worked through a documentation of each framework, tested out its examples and researched about differences and similarities. Some functions feature a code example in order to ease understanding for readers. Based on collected information the author created a summarized table which is written in detail what does each framework has to offer. During the comparison it was revealed that Bottle and Flask share a lot of similarities in overall, however Flask has slightly more functions and features. If the author would have to choose between Bottle and Flask for a web application project in the future, then he would choose Flask. Flask is suitable in the field of developing a web application, while Bottle is more suitable for prototyping ideas.

LISAD

Lisa 1. Denied mikroraamistiku veebilehe ekraanipilt



DENIED

THE NEXT GENERATION PYTHON MICRO-WEB-FRAMEWORK

README
DOCUMENTATION
CODE
ABOUT

A COMPLETELY DENIED APPLICATION

No installation or configuration required. No dependencies other than the Python standard library. Just get a copy of deny.py, place it into your project directory and start coding.

```
from deny import *

@route('/')
def hello():
    return 'Hello World!'

if __name__ == '__main__':
    run()
```

That's it! Now run your application and go to http://localhost:5000/ and your application will greet you!

WATCH THE SCREENCAST

Not sold yet? Watch the screencast to see how easy it is to write a scalable web 2.0 application with denied: watch in quicktime format

DOWNLOAD IT NOW

Just download the deny.py file from here and drop it into your project directory. You can also fork the code from github.

FULL OF FEATURES

Denied comes out of the box with superior awesomeness:

- A single Python file, no need for setuptools
- Works out of the box on Python 2.5 and 2.6, support for 3.x is planned
- Brings its own development server
- Kick-ass performance. 6000 requests per second with running on top of tornado
- builtin URL mapping, beautiful and RESTful URLs without the need to write a single regular expression
- builtin intuitive django-inspired templating language but without the limits
- RESTful
- Impressive Scaling Capabilites

WHAT PEOPLE SAY

Alex Gaynor (Django Developer)

"Denied is like nothing I've ever seen before, I think it's the future of the internet"

Benjamin Peterson (Python Developer)

"I have never seen so much code in one .py file"

Armin Ronacher (Werkzeug Developer)

"Probably the most hideous code I have ever seen"

Dick Taylor

"very nicely written and concise (not to mention it's written by Eirik Lahavre, whose coding skills are very trustable), and doesn't get in my way""

WHO IS BEHIND THIS PROJECT?

Hailing from the beautiful Lyon, Eirik Lahavre brings you the latest hotness of web development. Having Created Denied for a local company developing real-time web collaboration applications, Eirik brings an extensive knowledge about Python, web development and scaling.

Get in contact with him now.

WHERE IS THE DOCUMENTATION?

Documentation is upcoming, until then please refer to the screencast which should give you a good introduction how it works. You can also use the interactive Python help function that should give you a bsic overview over the API.