Tallinna Ülikool

Digitehnoloogiate Instituut

Ruby on Rails raamistik
Veebilahenduse või veebiteenuse koostamine selle abil

Seminaritöö

Autor: Kaspar Tint
Juhendaja: Jaagup Kippar

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud.

Kuupäev: 30. oktoober 2015

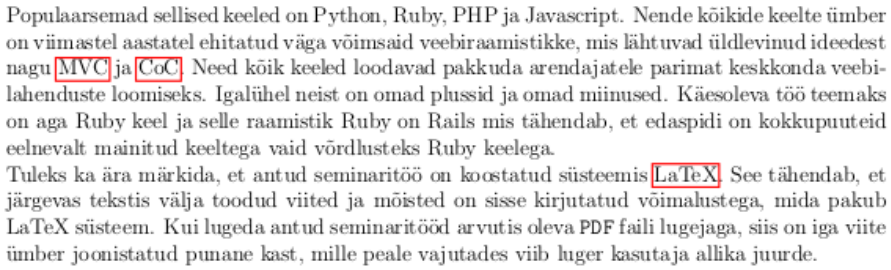
Ees- ja perekonnanimi: Kaspar Tint

Allkiri:

Sisukord

Sissejuhatus

Viimastel aastatel on järjest jõudu kogunud niiöelda kõrgemate keelte peale loodud veebiraamistikud. Selle põhjuseks võiks lugeda IT alaste idufirmade rohkust, kus peamiseks sooviks on võimalikult kiiresti üks töötav prototüüp valmis saada. Kuid nagu öeldakse - iga töö jaoks on oma tööriist. Selline ütlus kehtib ka tehnoloogia maailmas. Kiiresti areneval turul kus alles virguva firma eksistents oleneb puhtalt sellest kui kiiresti suudetakse prototüüp välja lasta, ei ole võimalik aega raisata valede tööriistadega.
Startupi maailma parimaks tööriistaks on keel, mille juures ei ole vaja muretseda mäluhalduse üle ja mis ei ole staatiliselt kirjutatud. Lisaks võiks ta ka olla võimalikult objekti põhine, et operatsioonid andmetega oleks võimalikult lihtsad.
Sellise keelega ei ole vaja muretseda antud kontekstis ebatähtsate probleemide üle, saab keskenduda ainult sellele, mis on kõige tähtsam - projekti valmimine.
Populaarsemad sellised keeled on Python, Ruby, PHP ja Javascript. Nende kõikide keelte ümber on viimastel aastatel ehitatud väga võimsaid veebiraamistikke, mis lähtuvad üldlevinud ideedest nagu MVC ja CoC. Need kõik keeled loodavad pakkuda arendajatele parimat keskkonda veebilahenduste loomiseks. Igalühel neist on omad plussid ja omad miinused. Käesoleva töö teemaks on aga Ruby keel ja selle raamistik Ruby on Rails mis tähendab, et edaspidi on kokkupuuteid eelnevalt mainitud keeltega vaid võrdlusteks Ruby keelega.
Tuleks ka ära märkida, et antud seminaritöö on koostatud süsteemis LaTeX. See tähendab, et järgevas tekstis välja toodud viited ja mõisted on sisse kirjutatud võimalustega, mida pakub LaTeX süsteem. Kui lugeda antud seminaritööd arvutis oleva `PDF` faili lugejaga, siis on iga viite ümber joonistatud punane kast, mille peale vajutades viib luger kasutaja allika juurde. Näidet selle kohta saab näha pildil 1.


Populaarsemad sellised keeled on Python, Ruby, PHP ja Javascript. Nende kõikide keelte ümber on viimastel aastatel ehitatud väga võimsaid veebiraamistikke, mis lähtuvad üldlevinud ideedest nagu MVC ja CoC. Need kõik keeled loodavad pakkuda arendajatele parimat keskkonda veebilahenduste loomiseks. Igalühel neist on omad plussid ja omad miinused. Käesoleva töö teemaks on aga Ruby keel ja selle raamistik Ruby on Rails mis tähendab, et edaspidi on kokkupuuteid eelnevalt mainitud keeltega vaid võrdlusteks Ruby keelega.
Tuleks ka ära märkida, et antud seminaritöö on koostatud süsteemis LaTeX. See tähendab, et järgevas tekstis välja toodud viited ja mõisted on sisse kirjutatud võimalustega, mida pakub LaTeX süsteem. Kui lugeda antud seminaritööd arvutis oleva PDF faili lugejaga, siis on iga viite ümber joonistatud punane kast, mille peale vajutades viib luger kasutaja allika juurde.

Pilt. 1: Viide LaTeX dokumendis

1. Ruby on Rails raamistik

Enne Ruby on Rails käsile võtmist tuleks ka eelnevalt saada tuttavaks keelega, mille peale see veebiraamistik on ehitatud.

1.1 Ruby programmeerimiskeel

Ruby on loodud Jaapani päritolu Yukihiro Matsumoto poolt, 20 aastat tagasi, 1995 aastal. See keel on saanud inspiratsiooni pisut vanematest keeltest nagu Perl, Ada ja Lisp. Ruby keel toetab mitmeid programmeerimise paradigmasid nagu funktsionaalne- ja objekt-orienteeritud programmeerimine. Samuti tasub ära märkida, et Ruby keel on nö. dünaamilisi tüüpe kasutav keel ning omab automaatset mäluhaldus süsteemi.
Need omadused ei tee sellest keelest õiget tööriista riistvara lähedaseks programmeerimiseks. Siiski, on Ruby keelel omad eelised - nimelt on sellise keelega võimalik väga kiirelt valmis kirjutada tarkvara, mille juures ei ole emsatähtis mäluhaldusest viimase välja pigistamine.
Sellepärast on ka viimasel ajal tekkinud järjest rohkem tarkvara projekte, mis kasutavad just nimelt Ruby keelt. Mõned näited:

1. **Vagrant** - See programm tegeleb virtuaal masinate ülesseadistamise automatiseerimisega. Kuna selline lahendus ei nõua arvutilt tohutul ressursse, siis oli Ruby keele valik antud juhul väga õige otsus.
2. **Teadustööd** - Ruby keelt kasutatakse laialdlaselt teaduslikes töödes. Sarnaselt Python keelele, on sellega võimalik kiirelt üles kavandada teaduslikke projekte.

Samas on olemas üks veelgi kiiremini arenev ala milleks on veebiarendus. Nutiseadete suure kättesaadavuse pärast on väga tähtis tänapäeval igal endast lugupidaval ettevõttel omada korralikku kodulehte. Selline olukord aga tekitab tohutult suure nõudluse. Kuna tööd on veebiarendajatel palju, siis on paljud neist otsustanud kiirema tee kasuks. Nimelt on loodud võimalus kirjutada veebirakendusi Ruby keeles.
2005 aasta detsembris loodi hakkaja grupi poolt veebi raamistik Ruby keele ümber millel nimeks Ruby on Rails. Sellest ajast saadik on toimunud tegus arendamine selle raamistiku ümber kuna kasutajate hulk on suur ja nõuded aina keerulisemad. Ruby on Rails on olnud aluseks teistele veebiraamistikele, mis on ehitatud dünaamiliselt kirjutatud keeltele. Samas on selle raamistiku populaarsus siaamaani väga kõrge. Hotframeworks leheküljel olevaid tulemusi on mõõdetud Stackoverflow saidi kasutuse järgi. Nagu tulemustest näha, siis Microsofti poolt loodud ASP.NET on küll esimesel kohal populaarsuselt aga Ruby on Rails on kohe napilt järel teine.

1.2 Ruby on Rails raamistiku olemus

Ruby on Rails on veebiraamistik mis sai arendatud David Heinemeier Hanssoni poolt, 2005. aastal. Mees juhtus töökorras arendama seda raamistikku ja mingil hetkel otsustas, et jagab Ruby on Rails lähtekoodi `MIT` litsentsi alusel ka üleanud maailmale.
Ruby on Rails sai arendatud MVC ja Convention over Configuration mustrit järgides. See tähendab, et tegemist on veebiraamistikuga, mis üritab selle pruukijaid standardsete probleemide puhul n.ö sundida teatud toiminguid üldlevinud tavade järgi tegema panna.
See kokku eeldab, et arendaja paigutab oma tekitatud koodi heade tavade järgi loogilistesse osadesse - Controller osa MVC'st on see mis tegeleb otsese kliendi, ehk veebisirvijaga suhtlemisega, Model osa tegeleb andmete käitlusega, mis eeldatavasti on SQL andmebaasi

salvestatud ja View osa tegeleb Ruby keelega segatud HTML koodi käitlusega ja selle serveerimisega kliendile.
Convetion over Configuration omakorda teeb arendaja eest teatud otsused ülejäänud raamistiku erinevate valikute kohta. Sellepärast on ka igal standardsel Ruby on Rails projektil standardne projekti üles ehitus ülejäänud komponentide kohta.

1.3 Ruby on Rails projekti struktuur

Ühe standardse projekti struktuur näeb välja järgnev:

1. app
2. bin
3. config
4. db
5. lib
6. log
7. public
8. test

App kausta alla paigutatakse üldine projekti loogika. Sinna lähevad kõik controllerid, modelid ja viewd. View'de puhul tasub tähele panna, et app kausta paigutatakse harilikult ainult .html.erb failid. Selle põhjuseks on see, et enne kui veebirakendus need lehed kliendile jagab, peab ta need tavaliseks .html failiks ümber töötlema ja arvesse võtma selle sees olevaid .erb laiendiga kaasnevaid Ruby koodi lõike. Samuti paigutatakse selle kausta sisse ka stiili failid mille laiendiks harilikult on .css ning samuti ka .js ehk Javascript failid.

Bin kaust sisaldab endas levinumaid tööriistu mis tulevad kaasa Ruby on Rails raamistikuga. Sinna alla kuuluvad programmid nagu bundler ja rake. Bundler on programm mis tegeleb lisa teekide haldamisega ja Rake on programm mis tegeleb andmebaasi haldamisega. Nimelt on Rake võimeline looma andmebaasi tabeleid arendaja poolt loodud Ruby koodi järgi.

Config kausta paigutatakse üldiselt Rails raamistiku sätted. Seal annab muuta valikuid näiteks andmebaasi ühenduse kohta ja veebirakenduse marsuutide kohta, mis hoolitsevad selle eest, et lõppkasutajale oleksid saadav ainult teatud marsuudid arendataval veebirakendusel.

Db kaust hoiab endas alati ühte Ruby faili(.rb) mis kirjeldab kogu andmebaasi hetke skeemi. Kui arendaja peaks jagama enda Ruby on Rails projekti mõnele teisele arendajale, siis saab see arendaja teatud Rake käsu abil luua enda arvutisse täpselt samasuguse andmebaasi seisu, nagu see fail kirjeldab. Samas sisaldab see kaust ka versioneeritud kujul andmebaasi muudatusi. See on hea selleks, et saaks viia andmebaasi seisu mõnele vanemale versioonile, kui peaks tekkima vigu rakenduses ja on vaja tagasi võtta mõni andmebaasi muudatus mis selle vea põhjustas, siis Rake oskab nende versiooni failide järgi toimetada ja andmebaasi tagasi vanemale kujule viia.

Lib kausta paigutatakse tihti teeke, mis võivad olla nii Ruby teegid kui ka näiteks Javascript teegid. Rails projekt oskab automaatselt sisse lugeda selles kaustas paiknevaid teeke ja neid

arendajale kasutamiseks anda rakenduse koodis.

Log kausta eesmärk on hoida veebirakenduse logi faile. Kõik serverist logisse prinditud tekstijupid paigutatakse ühte logifaili, mis loodi jooksva sessiooni jaoks. Kui rakendus uuesti käivitada, siis logi fail arhiveeritakse, ja luuakse uus logi fail uue sessiooni jaoks.

Public kaust sisaldab endas staatilisi .html faile, mida saab näiteks kasutada siis kui kasutaja soovib avada mõnda mitte eksisteerivat marsuuti. Sellisel juhul saab kasutajale esitada .html lehe, mis näitab, et sellist marsuuti ei eksisteeri, selle asemel, et näitata mitte-kasutajasõbralikku veateadet. Samuti paiknevad siin kaustas kõik kasutajate poolt üles laetud pildid, millele siis kasutajatel on võimalik ligi pääseda ilma, et nad peaks veebirakenduse controlleritega suhtlema enne selle kuvamist.

Test kausta paigutatakse testid, millega saab läbi testida kogu rakenduse koodi. Hea arendus tava on see, et kirjutatakse teste kriitiliste rakenduse komponentide kohta. Serveri käivitamisel saab alati ennem käivitada kõik testid ja veenduda, et rakendus töötab, nii nagu ta peaks testide järgi töötama.

Lisaks paikneb iga Rails projekti juurkaustas alati ka fail mida nimetatakse Gemfileks. Gemfile on fail, kuhu arendaja kirjeldab, milliseid teeke ta soovib oma rakendusse kaasata. Nende teekide veebist alla laadimise ja rakendusse integreerimisega oskab tegeleda Bundler. Arendaja saab Bundler programmile käsu anda, et oleks vaja alla tõmmata äsja Gemfile sisse lisatud teek.

1.4 Ruby ja Ruby on Rails raamistiku paigaldamine

Kuna Ruby arendus toimub põhiliselt arvutitel, mille peale on paigaldatud UNIX baasil operatsiooni süsteem, siis lähtub ka antud õppematerjal sellest. Windows masinatel ei pruugi töötada suur hulk Ruby teeke, kuna nende arendajad ei ole näinud vajadust aega kulutada Windows operatsioonisüsteemile arenduseks.
Esiteks on vaja paigaldada Ruby keel ise ja selle seminaritöö raames saab kasutatud 2.2.3 versiooni Ruby keelest. Selleks oleks vaja esiteks paigaldada mõned vajalikud teegid oma UNIX süsteemile. Terminali emulaatorisse tuleks sisestada järgmised teegid paigalduseks:

```
sudo apt-get update
sudo apt-get install git-core curl zlib1g-dev build-essential
   libssl-dev libreadline-dev libyaml-dev libsqlite3-dev sqlite3
   libxml2-dev libxslt1-dev libcurl4-openssl-dev
   python-software-properties libffi-dev
```

Järgnevalt oleks vaja paigaldada Ruby versioneerimis tarkvara, mis on võimeline ühes arvutis hoidma mitut versiooni Ruby keelest. Kasutajal on selle programmi abil võimalik ise valida, millist versiooni ta hetkel kasutada tahab. Kasutame selle töö raames `rbenv` Ruby haldus tarkvara.

```
cd
git clone git://github.com/sstephenson/rbenv.git .rbenv
echo 'export PATH="\$HOME/.rbenv/bin:\$PATH"' >> ~/.bashrc
```

```
echo 'eval "\$(rbenv init -)"' >> ~/.bashrc
exec $SHELL
git clone git://github.com/sstephenson/ruby-build.git
    ~/.rbenv/plugins/ruby-build
echo 'export PATH="\$HOME/.rbenv/plugins/ruby-build/bin:\$PATH"' >>
    ~/.bashrc
exec $SHELL
git clone https://github.com/sstephenson/rbenv-gem-rehash.git
    ~/.rbenv/plugins/rbenv-gem-rehash
rbenv install 2.2.3
rbenv global 2.2.3
ruby -v
```

Nende käskude sisestamisega sai paigaldatud Rbenv ja Ruby keele `2.2.3` versioon. Järgnevalt oleks vaja paigaldada ka Ruby programm bundler ja veenduda, et iga tõmmatud gem ei paigaldaks oma dokumentatsiooni arvutisse lokaalselt.
`echo "gem: --no-ri --no-rdoc" > /.gemrc`
`gem install bundler`
Nüüd oleks vaja paigaldada nodejs, mis on tarvilik, et Ruby on Rails suudaks tegeleda veebi ressursside manageerimisega:
`sudo apt-get install nodejs`
Peale kõikide eelnevate eelduste täitmist on võimalik paigaldada Ruby on Rails raamistik arvutisse. Ruby keel kannab endaga kaasas teegihaldus programmi nimega `gem`.
Järgnev paigaldab arvutisse Ruby on Rails gemi, versiooniga `4.2.4`:
`gem install rails -v 4.2.4`.
Rails gem paigaldus võib võtta omajagu aega. Kui paigaldus toiming on valmis, tuleks `rbenv` programmile üelda, et `Rails` gemi käivitaja oleks kätte saadav: `rbenv rehash`.
Kui nüüd kirjutada konsooli käsk: `ruby -v`, siis peaks terminal peale seda välja kuvama teksti `Rails 4.2.4`. Sellest võime järeldada, et Ruby on Rails raamistik on edukalt arvutisse paigaldatud.

1.5 Andmebaasi paigaldamine

Oleks vaja paigutada PostgreSQL andmebaasi programm, mida on vaja hiljem rakenduse loomisel. Paigutada käsuga
`sudo apt-get install postgresql-common postgresql-9.3 libpq-dev`.
Kui toiming on valmis, siis oleks vaja luua ka kasutaja millega andmebaasi tabeleid luua.
`sudo -u postgres createuser minuKasutaja -s`
Nüüd oleks vajalik ka just loodud kasutajale valida parool. `sudo -u postgres psql`. See käsk viib kasutaja `psql` konsooli. Siin saab kirjutada `\password minuKasutaja`, mis annab valiku sisestada parool, millega edaspidi saab andmebaasile ligi "minuKasutaja" nime alt.
Lugejad võivad paigaldada ka mõne enda poolt välja valitud andmebaasi rakenduse. Selleks aga tuleks uurida viidet Ruby on Rails andmebaasi teegid, kus on välja toodud erinevaid andmebaase ja nende kasutamise õpetust Ruby on Rails raamistikus.

2. Veebirakenduse loomine Ruby on Rails raamistikus

Nüüd kus Ruby, Ruby on Rails ja andmebaas on paigaldatud, saab hakata looma reaalset rakendust. Loome lihtsa tüüp rakenduse, mis kasutab Ruby on Rails raamistiku põhi komponente. Peale selle seminaritöö lugemist, peaks inimesel olema põhi arusaam sellest, kuidas alustada ühe Ruby on Rails rakenduse loomist.

2.1 Rakenduse idee

Rakendus hakkab kasutama ühte SQL tabelit PostgreSQL andmebaasis, kuhu saab salvestada raamatuid. Raamatutel on nimi(string), väljastus aasta(integer) ja autor(string). Kuna kasutame rake programmi andmebaasi tabeli manageerimisel, siis ei ole vaja eraldi välja tuua tabeli indeks veergu, rake tegeleb selliste asjadega ise automaatselt.
Raamatuid on võimalik vaadata nö. `index` lehel ehk aadressil `localhost:3000`, kus kuvatakse kõik raamatud andmebaasis. Raamatuid saab lisada aadressil
`localhost:3000/books/new/` vaadata aadressil `localhost:3000/books/x`, kus x on võrdne raamatu indeksi numbriga andmebaasis ja raamatuid saab ka kustutada, mis toimub samal aadressil kasutaja jaoks, kus saab kindlat raamatut vaadata.

2.2 Rakenduse vundament

Tuleks valida sobib koht kuhu rakendus paigutada oma süsteemis. Näiteks teeme kausta `ruby-on-rails` kasutaja kodukataloogi.

```
cd
mkdir ruby-on-rails
cd ruby-on-rails.
```

Eeldades, et Ruby on Rails on edukalt paigaldatud, saame kasutada käsku `ruby new minu-rakendus -d=postgresql`. See loob kausta `minu-rakendus` meie preaguse kausta sisse. Tasub tähele panna, et programmi argument `-d=postgresql` ütleb Rails rakenduse loojale, et soov on luua rakendus, kasutades PostgreSQL andmebaasi. Rails programm sätib ise paika kõik vajalikud sätted, et rakendus suudaks kohe seda tüüpi andmebaasiga ühenduda.
Liigume konsoolis selle kausta sisse ja kirjutame `ls` käsu. Tulemus peaks olema selline:

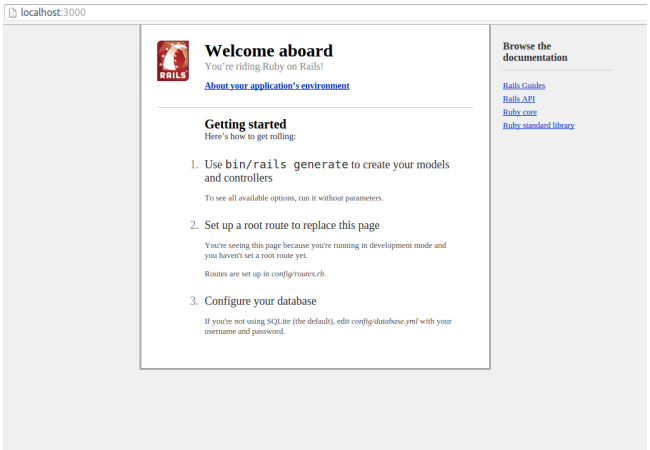
```
→ minu-rakendus ls
app  config  db       lib  public    README.rdoc  tmp
bin  config.ru  Gemfile  log  Rakefile  test       vendor
```

Pilt. 2: Värske Ruby on Rails rakendus

Enne, kui rakendus esimest korda käima panna, tasuks veenduda, et rakenduse andmebaasi seaded on korrektsed. Tuleks avada `database.yml` fail, mis paikneb `config` kaustas. See fail oleks vaja avada soovipärase tekstiredaktoriga ja otsida sealt rida `development:`. Selle rea alla on rails programm juba ette kirjutanud andmebaasi skeemi nime rakenduse nime järgi. Oleks vaja veel ära määrata andmebaasi kasutaja, parool ja port.

```
username:  minuKasutaja
password:sinuValitudParool
port:5432
```

Kui andmebaasi paigaldusel mõni samm valesti pole läinud siis peaks olema võimalik nüüdsest edukalt käivitada Ruby on Rails rakendus. Oleks vaja kirjutada rakenduse juur kaustas `rails s -p 3000`, mis käivitab rakenduse pordil 3000.



Pilt. 3: Ruby on Rails rakenduse esimene käivitus

2.3 Vaikimise root lehe muutmine

Nüüdseks on olemas Ruby on Rails rakendus, mis on käivitatav ja andmebaasiga ühenduv. Oleks vaja muuta vaikimisi säte rakenduse juur lehe kohta. Hetkel kuvatakse vaikimisi Ruby on Rails arendajate poolt loodud tervitus leht kui minna aadressile `localhost:3000`. Muudame selle nii, et lehe külastajale pakutakse leht, mis tervitab teda ja pakub navigeerimist raamatute "index" lehele. Selleks oleks vaja avada esiteks `application_controller` fail `app/controllers` kaustas. See on ruby klass, mis annab edaspidi loodavatele controller tüüpi klassidele kõik vajalikud meetodid, et oleks võimalik neid klasse veebilehitsejasse serveeritud failidega suhtlema panna. Oleks vaja lisada uus meetod selle klassi sisse nimega `index`. Lõpp tulemust on nähe faili näites 1

Fail. 1: ./app/contollers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  def index

  end
end
```

Kui Ruby on Rails controller tüüpi klassi (Edasipidi lihtsalt controller) lisada meetod mis ei ole eraldi märgitud privaat meetodiks, siis tagastab see meetod alati endale controlleri nimega `views` kasutas oleva `html.erb` faili, mille nimi vastab selle meetodi nimele. Kui nüüd luua `app/views/` kausta uus kaust nimega `application` ja selle sisse luua fail nimega `index.html.erb`, siis on võimalik seda meetodit kasutades kuvada antud `html` fail veebilehitsejas. Selle `html` faili sisse võiks ka luua väikese tervituse, et oleks aru saada, kui leht on reaalselt laetud.

```
<h1>Tervitus pealehelt aadressil localhost:3000</h1>
```

Nüüd oleks vaja veel veenduda, et loodud rakendus kasutab `index` meetodit, mis sai äsja loodud. Tuleks avada `config/` kasutas olev fail `routes.rb` ja lisada sinna järgmine käsk: `root 'application#index'`. Lõpp tulemus on näha faili näites 2, kus on eemaldatud failis olevad kommentaarid.

Fail. 2: ./config/routes.rb

```
Rails.application.routes.draw do
  root 'application#index'
end
```

Nüüd kui teha Rails serveri taaskäivitus(`rails s`), siis peaks veebilehtijas aadressil `localhost:3000` tervitama pildil 4 olev tekst.

Pilt. 4: Isetehtud root leht

2.4 Raamatute tabeli loomine andmebaasi

Ruby on Rails raamistik laseb andmebaase hallata ruby klasside abil. See tähendab, et arendajal ei ole enamasti vaja otseselt andmebaasi käskudega kokku puutuda. See võib aga muutuda, kui on tarvis hakata teatud andmebaasi osi optimeerima. Kõik lihtsamad tööd saab aga Ruby on Rails raamistiku poolt loodud tööriistadega ära teha. Oleks vaja luua uus tabel nimega `books`, mis hakkaks sisaldama raamatu pealkirja, autorit ja väljalaske aastat.
Sellejaoks oleks vaja luua uus andmebaasi migratsiooni fail. Selline fail on mõeldud selleks, et oleks mugav viia käiku uued muudatused ja samas need pärast poole tagasi võtta, kui peaks tekkima selline vajadus. Andmebaasi hetke seis kuvatakse alati `db` kaustas faili `schema.rb` sees. Kui migratsiooni fail lisada rakenduse sisse, siis lisatakse selle sees olevad muudatused `schema.rb` faili. Loome uue migratsiooni faili järgmise käsuga: `rails generate migration CreateBook name:string author:string published:integer`. See käsk loob uue migratsiooni faili kausta `db/migrate/` alla. Kui käsk lõpetas oma tegevuse ilma vigadeta, siis peaks olema selles kaustas olema uus fail mille sisu on nähtav faili näites 3.

Fail. 3: ./db/migrate/x_create_books.rb

```
class CreateBooks < ActiveRecord::Migration
  def change
    create_table :books do |t|
      t.string :name
      t.integer :published
      t.string :author
    end
  end
```

```
    end
```

Fail kirjeldab ära, et vaja on luua uus tabel nimega books ja selle sees peavad olema name, published ja author veerud. Kui nüüd jookustada käsku `rake db:migrate` konsoolis, luuakse andmebaasi eelnevalt kirjeldatud tabel koos vajalikke veergudega. Lisaks muudetakse `db/` kaustas olev `schema.rb` fail ära, mille sisu peale migratsiooni peaks olema samasugune nagu faili näites 4.

Fail. 4: ./db/schema.rb

```
# encoding: UTF-8
# This file is auto-generated from the current state of the database.
    Instead
# of editing this file, please use the migrations feature of Active
    Record to
# incrementally modify your database, and then regenerate this schema
    definition.
#
# Note that this schema.rb definition is the authoritative source for your
# database schema. If you need to create the application database on
    another
# system, you should be using db:schema:load, not running all the
    migrations
# from scratch. The latter is a flawed and unsustainable approach (the
    more migrations
# you'll amass, the slower it'll run and the greater likelihood for
    issues).
#
# It's strongly recommended that you check this file into your version
    control system.

ActiveRecord::Schema.define(version: 20151012192441) do

  # These are extensions that must be enabled in order to support this
     database
  enable_extension "plpgsql"

  create_table "books", force: :cascade do |t|
    t.string "name"
    t.integer "published"
    t.string "author"
  end
end
```

Sellega on edukalt rakenduse loomiseks vajalik raamatute tabel loodud ja nüüd on võimalik Ruby on Railsil seda tabelit kasutama hakata.

2.5 Raamatute tabeli sisu kuvamine

Selleks, et ära kasutada Ruby on Rails raamistiku sees olevaid mugandusi andmebaasi tabelite kasutamisel, oleks vaja luua igale tabelile ka vastav model tüüpi fail. Model fail oleks vaja luua kausta `app/models/`, nimega `book.rb`. Faili sees peaks ära defineerima

Ruby klassi nimega `Book` ja see klass omakorda peaks sõltuma `ActiveRecord::Base` klassist, mis annab sellele Ruby klassile Ruby on Rails poolsed võimed ja teeb sellest nö. Model klassi. Fail sisu peaks olema samasugune kui on näha faili näites 5.

Fail. 5: ./app/models/book.rb

```
class Book < ActiveRecord::Base

end
```

Peale seda oleks veel vaja luua raamatute infot liigutav controller ja ka vastav view. Lisaks oleks vaja ka `routes.rb` failis ära defineerida raamatu obektile vastavad marsuudid. Alustuseks luua controller. Fail nimega `books_controller.rb` oleks vaja luua `app/controllers/` kausta. Selles failis peaks ära defineeritud olema klass nimega `BooksController` ja see klass peaks sõltuma `ApplicationController` ist, mis omakorda sõltub `ActionController::Base` klassist. `ActionController::Base` on klass, mis teeb ühest Ruby klassist Ruby on Rails controlleri. Kõik klassid mis sellest klassist sõltuvad, on võimelised kuvama view objekt veebilehtisejale ja küsima model objektidelt andmeid ja neid ka sinna sisestema.
Book controller klassi sees oleks veel vaja ära defineerida meetod `index` mille sees oleks vaja pärida andmebaasist kõik raamatud: `@books = Book.all`. Märgis @ muutuja nime ees teeb sellest muutujast globaalse muutuja, mida annab kasutada ka veebilehtisejale serveeritud view objektide sees. Lõpplikku controller faili sisu on näha faili näites 6. Lõpplik controller fail peaks välja nägema järgmine:

Fail. 6: ./app/controllers/books_controller.rb

```
class BooksController < ApplicationController
  def index
    @books = Book.all
  end
end
```

Et veebilehitsejast saada ligi äsjaloodud `index` meetodile, oleks vaja välja kirjutada ka marsuudid, mis on võimelised seda tegema. Oleks vaja `routes.rb` faili lisada järgnev rida: `resources :books`. Selle ühe reaga luuakse kõik `RESTful` marsuudid books objektile. Kui kirjutada käsureal `rake routes`, siis on võimalik näha kõiki rakenduse poolt loodud marsuute, mida kasutaja saab tarbida. Näide siiani loodud rakenduse marsuutidest:

```
Prefix Verb    URI Pattern              Controller#Action
root     GET   /                        application#index
books    GET   /books(.:format)         books#index
POST           /books(.:format)         books#create
new_book GET   /books/new(.:format)     books#new
edit_book GET  /books/:id/edit(.:format) books#edit
book GET       /books/:id(.:format)     books#show
PATCH          /books/:id(.:format)     books#update
PUT            /books/:id(.:format)     books#update
DELETE         /books/:id(.:format)     books#destroy
```

Nagu näha, on olemas nüüdseks marsuut `/books`, mis käivitav GET päringu peale `index` meetodi äsja loodud controller klassis. Lisaks lõi `resources :books` ka kõik teised REST-

ful marsuudid, mille abil saab igat raamatut eraldi vaadata, raamatut kustutada ja ka muuta.
Nüüd oleks vaja luua veel html fail, mida raamatute `index` meetod kuvaks. Kausta `app/views/` oleks vaja luua uus kaust nimega `books`, mille sisse uus fail `index.html.erb`. Selles view objektis oleks mõistlik kuvada kuvada kõik raamatud, mis andmebaasi salvestatud on. Selleks, et `.erb` prefiksiga failides Ruby koodi pruukida, on vaja see koord mähkida `<% %>` või `<%= %>` märkide sisse. Esimene neist käivitab Ruby koodi aga ei kuva seda inimese silmale, teine neist kuvab selle sees oleva koodi tulemi veebilehitsejas välja. Kui näiteks kirjutada `<%= 2 + 2 %>`, siis kuvatakse lehel arv 4. Kui võtta võrdusmärk ära ja kirjutada lihtsalt `<% 2 + 2 %>`, siis ei kuvata midagi lehele.
Et raamatuid kuvada, oleks hea kuvada see info tabelina. Lõpuks võiks see fail näha välja sarnane faili näites 7 välja toodud sisule.

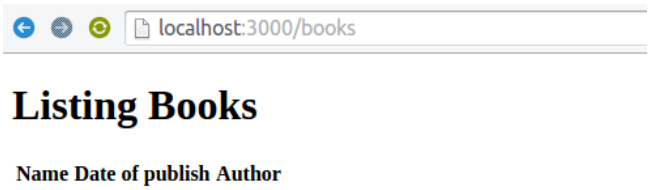
Fail. 7: ./app/views/books/index.html.erb

```
<h1>Raamatud</h1>

<table>
  <tr>
    <th>Pealkiri</th>
    <th>Ilmumise aasta</th>
    <th>Autor</th>
  </tr>

  <% @books.each do |book| %>
    <tr>
      <td><%= book.name %></td>
      <td><%= book.published %></td>
      <td><%= book.author %></td>
    </tr>
  <% end %>
</table>
```

Tasuks tähele panna rida `<% @books.each do |book| %>`. Siin luuakse tsükkel, mille tsükli muutujaks on `book` muutuja. Seda muutujat pruukides, saab välja kuvada lehitsejasse kõik raamatu parameetrid.
Kui nüüd panna rakendus käima ja avada lehitsejas marsuut `localhost:3000/books/`, siis peaks avanema vaatepilt mida on näha pildil 5.



Pilt. 5: Raamatute pealeht

Kuna raamatute andmebaasi tabelisse pole veel andmeid lisatud, siis ei ole ka html tabeli

päise all ühtegi väärtust näha. Järgnevalt oleks vaja luua võimalus rakenduses uusi raamatuid andmebaasi sisestada, et oleks ka reaalseid väärtusi võimalik lehel näha.

2.6 Andmebaasi raamatute lisamine

Selleks, et raamatuid lisada oleks võimalik, oleks vaja esiteks luua kaks uut meetodit raamatute controller klassi. Meetodite nimed peaks olema `new` ja `create`. New meetodi abil saame kuvada uue view, millel on võimalik sisestada raamatu andmed ja vajutad nupule, et andmed rakenduse controllerile edastada. Create meetod tegeleb new meetodi poolt tekitatud view nupu vajutuse kuulamisega. Kui nupp on vajutatud ja andmed saadetud, kontrollib create meetod üle, et kõik andmed on olemas ja üritab nende abil luua raamatute tabelisse uue kirje andmebaasis. Samas ei tohiks create meetod omada eraldi endale vastavat html lehte. Tuleks `if` lausega aru saada kui andmed on salvestatud ja siis kasutaja suunata tagasi index meetodi juurde.
New meetodi sisu peaks tegema temale vastava `new.html.erb` failile kättesaadavaks muutuja, mille külge raamatu andmeid lisada. Selleks oleks vaja new meetodi sisse kirjutada järgmine rida: `@book = Book.new`. See loob uue `Book` tüüpi klassi, mis on hetkel veel tühi. Samal ajal oleks vaja ka luua new meetodile vastav html fail. Books kausta tuleb luua selleks `new.html.erb` fail, mille sees peaks olema andmete sisestus vorm. Andmete sisestus vormi saab luua käsuga `form_for` ja see peaks olema sisestatud `<%= %>` märkide sisse, kuna soov on terve vorm välja kuvada. Tasub ka tähele panna, et vormi loomisel on vaja luua tsükkel. Luuakse vorm `@book` muutuja kohta, ja tsükli sees on vaja luua selle muutujaga seotud andmete sisestus väljad. Andmesisestuseks vajaminevaid lahtreid saab välja kutsuda käskudega `text_field` ja `number_field`, nagu on näha faili näited 8.

Fail. 8: ./app/views/books/new.html.erb

```
<h1>Uue raamatu lisamine</h1>

<%= form_for @book do |b| %>
   <label>Raamatu pealkiri: </label><%= b.text_field :name %><br>
   <label>Raamatu ilmumise aasta: </label><%= b.number_field :published
      %><br>
   <label>Raamatu autor: </label><%= b.text_field :author %><br>
   <%= b.submit 'Sisesta!' %>
<% end %>
```

Nüüd kui rakendus uuesti käivitada, peaks olema nähtav leht marsuudil `localhost:3000/books/new/`, kus on võimalik sisestada raamatu parameetreid ja vajutada nupule `Sisesta!`. Nupule vajutus saadab küll andmed books controllerisse, create meetodile. Kuna create meetod aga endas veel mingit loogikat ei sisalda, siis uut raamatut siiski veel ei looda. Nüüd oleks vaja luua vastav loogika.
Create meetodi sisse peaks kirjutama järgmist: `@book = Book.create(book_params)`. Selle reaga loome uue muutuja nimega `@book`, mis sisaldab endas uut Book tüüpi objekti. Enne selle objekti loomist käivitatakse ka meetod `create(book_params)`. Sellega luuakse Book tüüpi klass, andes talle kaasa ka parameetrid. Hetkel on parameetriks küll meetodi nimi, mida veel ei ole defineeritud. Books controller klassi lõppu oleks vaja kirjutada selle defineerimiseks `private`. Kõik selle rea alla kirjutatud kood on kasutatav ainult konkreetse klassi siseselt. View ja Model objektid sellistele meetoditele ligi ei pääse. Äsja loodud rea alla oleks vaja ära defineerida `book_params` meetod. See meetod hakkab kon-

trollima raamatute controlleri sisse saadetud andmeid ja veenduma järgmistes tingimusted:

1. Andmed on saadetud õiges vormingus.
2. Andmeid ei ole saadetud liiga palju
3. Andmetega ei proovita saata raamatu objektiga mitte seotud objekte.

Oleks siis vaja kirjutada `book_params` meetodi sisse
`params.require(:book).permit(:name, :published, :author)`.
Nüüd suudab `create` meetodis olev `Book.create(book_params)` käsk kätte saada kõik rakendusele saadetud parameetrid, tänu funktsiooni parameetrina antud meetodile. Peale seda oleks veel vaja tegeleda kasutaja ümber suunamisega korrektsele marsuudile.

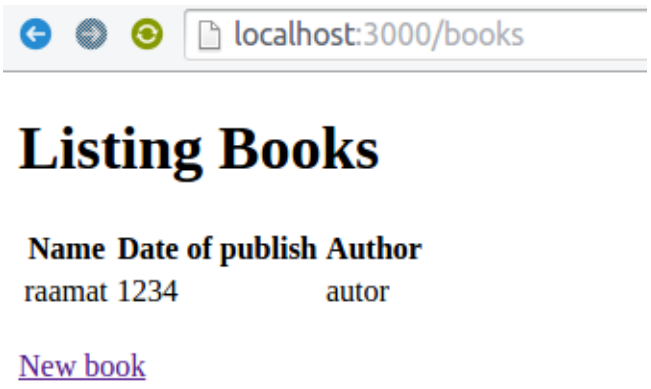
Fail. 9: ./app/controllers/books_controller.rb

```
...
def create
   @book = Book.create(book_params)
   if @book.save(book_params)
      redirect_to @book
   else
      render :index
   end
end

...

private
   def book_params
      params.require(:book).permit(:name, :published, :author)
   end
```

Kui selline rida kirjutada peale uue raamatu loomise koodi, saab programm aru kas raamat suudeti ka reaalselt ära salvestada. Kui raamat salvestati ära, siis viiakse kasutaja seda raamatut kirjeldavale marsuudile, vastupidisel juhul aga viiakse kasutaja marsuudile, mis kuvab kõiki raamatuid andmebaasis (`localhost:3000/books`).



Pilt. 6: Books index marsuut peale uue raamatu lisamist

2.7 Raamatu kustutamine andmebaasist

Raamatute eemaldamiseks oleks vaja raamatu controllerisse lisada uus meetod, mille nimi võiks olla `destroy`, mis on korrektne nimetus ühele RESTful marsuudile. Kasutaja peaks view poolelt sellele meetodile saatma raamatu, mida on soov andmebaasist kustutada. Selleks oleks vaja `destroy` meetodi sees esiteks andmebaasist leida vastav kirje, kasutades sisend parameetrina saadud raamatu indeksi numbrit. Kui raamat on kustutatud, oleks vaja kasutaja ümber suunata raamatute pealehele.
Raamatu kustutamiseks vajalik kood hõlmab ennast raamatu leidmist ja seejärel leitud raamatu peal `destroy()` meetodi kasutamist.

Fail. 10: ./app/controllers/books_controller.rb

```
def destroy
  Book.find(params[:id]).destroy
  redirect_to books_path
end
```

Selleks, et saata raamat serveri poolele kustutamisele, oleks vaja luua ka vastav kood html poolel. Raamatuid võiks saada kustutada raamatute pealehelt, kus iga välja toodud raamatu järel oleks link kirjaga "Kustuta". Lingi vajutamisel kuvatakse kasutajale ka lisadialoog, mis kontrollib kas soov on tõesti raamat eemaldada.

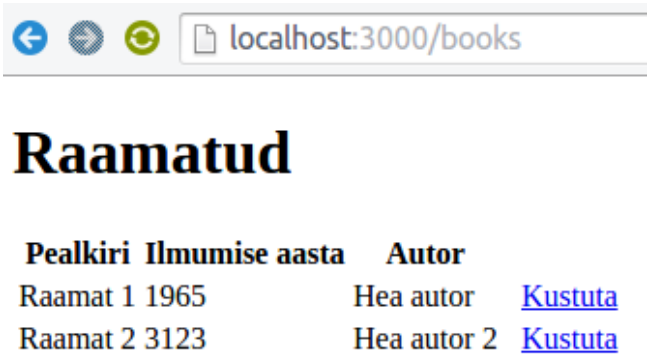
Fail. 11: ./app/views/books/index.html.erb

```
<h1>Raamatud</h1>

<table>
  <tr>
    <th>Pealkiri</th>
    <th>Ilmumise aasta</th>
    <th>Autor</th>
  </tr>

  <% @books.each do |book| %>
    <tr>
      <td><%= book.name %></td>
      <td><%= book.published %></td>
      <td><%= book.author %></td>
      <td><%= link_to 'Remove', book_path(book), method: :delete,
          data: { confirm: 'Are you sure?' } %></td>
    </tr>
  <% end %>
</table>
```

Viimase reaga faili näite 11 tsükli sees luuakse html link, mille pealkirjaks on "Kustuta". See link viitab raamatu marsuudile HTTP meetodiga DELETE. Viimase parameetrina `link_to` funktsioonis on määratud ära üleküsimise tekst. See kuvatakse kasutajale lingi vajutusel.
Nüüd kui minna raamatute pealehele, peaks olema kuvatud kustutamise lingid peale iga raamatu kirjet.



Pilt. 7: Raamatute pealeht võimalusega raamatut kustutada

2.8 Olemasoleva raamatu kuvamine ja muutmine

Viimaseks funktsionaalsuseks loodava rakenduse juures oleks iga raamatu muutmine ja vaatamine. Selle jaoks oleks vaja raamatute controllerisse luua kolm uut meetodit. Kaks neist oleks mõeldud kasutajale info kuvamiseks (`show ja edit`) ja viimane neist tegeleks andmete uuendamisega (`update`).
`Show` ja `edit` meetodites oleks vaja lihtsalt otsida üles raamat andmebaasist, parameetrina saadud indeksi numbri järgi. Seejärel saab mõlemale meetodile vastavas views kasutada antud raamatut kas siis lihtsalt näitamiseks või muutuste pakkumiseks. `Update` meetodis oleks samuti vaja leida raamat indeksi järgi aga peale seda kutsuda sama raamatu küljest `update(book_params)` ja kontrollida kas raamat sai ka reaalselt uuendatud andmebaasis ja vastavalt sellele kasutaja viidata kas raamatu vaate marsuudile või raamatute pealehele. Lõpp tulemus on näha faili näites 12.

Fail. 12: ./app/controllers/books_controller.rb

```
def show
   @book = Book.find(params[:id])
end

def edit
   @book = Book.find(params[:id])
end

def update
   @book = Book.find(params[:id])
   if @book.update(book_params)
      redirect_to @book
   else
      render :index
   end
end
```

Lisades need meetodid raamatute controllerisse, on serveri poolega lõpule jõutud. Lisada oleks veel vaja uued html lehed `show (13)` ja `edit(14)` meetodite jaoks. Esimene neist peaks lihtsalt kuvama kindla raamatu infot ja teine peaks kuvama raamatu muutmise vormi, mille lahtrid on eelnevalt juba täidetud raamatu hetke väärtustega.

Fail. 13: ./app/view/books/show.html.erb

```
<h1>Raamatu vaade</h1>

<p>Pealkiri: <%= @book.name %></p>
<p>Valjalaske aasta: <%= @book.published %></p>
<p>Autor: <%= @book.author %></p>

<%= link_to 'Muuda raamatut', edit_book_path(@book) %><br>
<%= link_to 'Tagasi raamatute pealehele', books_path %>
```

Fail. 14: ./app/view/books/edit.html.erb

```
<h1>Raamatu: <%= @book.name %> muutmise leht</h1>

<%= form_for @book do |b| %>
  <%= b.text_field :name %>
  <%= b.number_field :published %>
  <%= b.text_field :author %>
  <%= b.submit 'Muuda!' %></br>
<% end %>

<%= link_to 'Tagasi raamatute pealehele', books_path %>
```

Tasub tähele panna, et kuna raamatu muutmise lehel on `@book` muutuja andmebaasist võetud andmetega eelnevalt täidetud, siis kuvab leht iga vormi lahtrisse raamatu algsed väärtused. Peale seda oleks mõistlik ka lisada viited äsja loodud uutele lehtedele raamatute pealehel. Oleks vaja lisada sealse tsükli sisse kaks uut rida:
`<td><%= link_to 'Näita', book %></td>` ja
`<td><%= link_to 'Muuda', edit_book_path(book) %></td>`
Nüüd on võimalik edukalt rakenduse erinevate vaadete vahel mugavalt liigelda ja uusi raamatuid luua, vaadata, muuta ja kustutada. Sellega on ka 2.1 sektsioonis välja toodud rakenduse idee edukalt valmis saadud.

2.9 Valminud rakenduse lähtekood

Valminud rakenduse lähtekood on saadaval aadressil https://github.com/Veske/Seminar. Koodi võib alla laadida antud lehelt kasutades `DOWNLOAD ZIP` nuppu või kasutas `git` programmi käsku `git clone https://github.com/Veske/Seminar.git`.
Peale rakenduse alla laadimist oleks vaja paigaldada kõik vajalikud teegid käsuga `bundle install`, luua andmebaas käskudega `rake db:create` ja `rake db:schema:load`. Kui kõik õnnestus, saab rakenduse käima panna käsuga `rails s` ja rakendust uurida marsuudil `localhost:3000`.

3. Rakenduse loomine Veebiraamistikud aine raames

Jaagup Kippari läbiviidud aines sai proovitud antud seminariöö järgi luua Ruby on Rails raamistik. Pooleteise tunni jooksul jõudsid õpilased luua rakenduse, milles sai lisada uusi raamatuid andmebaasi ja kõiki andmebaasis olevaid raamatuid üle vaadata. Tervet rakendust, koos raamatute muutmise ja kustutamisega ei jõudnud valmis saada kuna tekkisid komplikatsioonid.
Rakendust loodi Tallinna ülikooli, Informaatika instituudi `greeny.cs.tlu.ee` serveris. Kuna kõik üliõpilased hakkasid korraga endale rakendust looma, siis kippus see esimene etapp väga pikaks. Ruby on Rails rakenduse loomisel luuakse esiteks projekti struktuur ette nähtud kausta ja kohe peale seda tõmmatakse alla kõik vajalikud teegid. Kuna üliõpilastel ei olnud administraatori õigusi antud serveris, siis oli vaja paigaldada kõik teegid lokaalselt. Selline tegevus aga koormas üle Tallinna Ülikooli interneti ühenduse ja arvatavasti ka greeny serveri.
Kokku kulus lihtsalt rakenduse vundamendi loomiseks kuskil pool tundi õppeajast. Kui lõpuks said kõik üliõpilased ikkagi endale Ruby on Rails rakenduse vundamendi loodud, sujus edasine töö kiirelt. Siiski tekkis mõnel õpilasel ka vigu. Kaks õpilast olid oma raamatute skeemi andmebaasi loonud valede nimedega. `Books` tabel oli nimetatud hoopis `Book` nimega. Selline nimetus aga lõhub ära kõik Ruby on Rails poolt pakutud mugandused ja sellisel juhul tuleb andmebaasi päringuid hakata manuaalselt välja kirjutama. Kui nimetada kõik rakenduse komponendid heade tavade järgi, siis selliseid probleeme ei teki. Lisaks oli mõnel õpilasel raske tempoga järgi jõuda. Sai nendele õpilastele täiendavalt seletatud asjad üle, et nemad ka järgi jõuaks ja rakenduse lõpuks valmis saaks.
Väikesed takerdused õppetööd aga oluliselt ei pidurdanud ja õppetöö lõpuks said kõik õpilased valmis rakenduse, kus sai raamatuid edukalt lisada ja neid ka uurida. Puudu jäi pisut ajast, mida kahjuks oli kõigest poolteist tundi. Kuskil poole tunni lisa ajaga, oleks arvatavasti olnud võimalik terve ettenähtud rakendus valmis luua.

Kokkuvõtteks

Nagu sissejuhatuses mainitud, on Ruby on Rails raamistik hea tööriist neile, kes kiiresti valmis arendada mõne veebirakenduse. Peatükis 3 tuli välja, et keskeltläbi poole äsja loodud rakenduse valmistamiseks läheb poolteist tundi. Kui veel arvestada, et iseseisvalt sellise rakenduse loomisel ei teki probleeme, kus rakenduse vundamendi loomine võtab 30 minutit aega, siis võiks väita, et Ruby on Rails raamistikuga saab tõesti väga kiiresti luua lihtsa rakenduse prototüübi, mis suudab ka andmebaasiga ühenduda.
Seda protsessi annab veelgi kiirendada, kasutades Rails Composer rakendust. Selle rakenduse abil on võimalik luua Ruby on Rails veebirakenduse vundament, mis sisaldab endas veelgi erinevaid valmis lahendusi. Saab näiteks selle abil lisada rakenduse külge kasutajad ja nende autentimise. Tegemist on muidu keerulise osaga ühes rakenduses, millele tuleks pöörata suurt tähelepanu turvakaalutlustel. `Rails Composer` aga ühendab automaatselt sellise funktsionaalsuse arendatava rakenduse külge kasutades mõistlikke vaikimisi sätteid. Muidugi saab iga arendaja neid hiljem muuta vastavalt soovile. Lisaks suudab `Rails Composer` lisada loodava rakenduse külge mõne levinuma Ruby on Rails veebiserveri nagu `Nginx` või `Puma`.
Üks populaarsemaid rakendusi mis on Ruby on Rails raamistiku abil loodud on Twitter. Twitteril tekkis küll hilisemas staadiumis probleeme kuna kasutajaid tekkis liiga palju ja arendajad ei suutnud enam olemasolevat rakendust optimeerida suurele kasutajate hulgale. Sellepärast kirjutas Twitteri arendus meeskond osa rakendusest uuesti kasutades Java keelt. Siiski on praegusel hetkel veel küllaga Ruby on Rails rakendusi liikvel ja mõnega neist saab tutvuda järgmisel veebilehel: `http://skillcrush.com/2015/02/02/37-rails-sites/`.
Ruby on Rails kogukond on palju vaeva näinud, et teha arendajate töö võimalikult lihtsaks seda raamistikku kasutades. Kogu Ruby on Rails ja selle erinevate teekide arendus toimub põhimõttega, mis tahab arendajate tööd lihtsustada ja kiirendada. Sellepärast võib öelda, et Ruby on Rails raamistik on väga sobilik tööriist arendajatele, kes soovivad luua mõnele idufirmale kiiresti veebilahendust.

Kasutatud kirjandus

Wikipedia. Ruby programming language. -
https://en.wikipedia.org/wiki/Ruby_(programming_language)

Hotframeworks -
http://hotframeworks.com/

Stackoverflow -
http://stackoverflow.com/

Hashicorp (2010, märts 8). Vagrant -
https://www.vagrantup.com/

Ruby-Lang. Ruby Success Stories. -
https://www.ruby-lang.org/en/documentation/success-stories

Wikipedia. Ruby on Rails -
https://en.wikipedia.org/wiki/Ruby_on_Rails

Daniel Kehoe, Github (2012, juuli 24). Rails Composer -
https://github.com/RailsApps/rails-composer

Ruby on Rails guides. Ruby on Rails database gems. -
http://guides.rubyonrails.org/v2.3.11/getting_started.html#configuring-a-database

Latex-Porject. LaTeX -
https://www.latex-project.org/

Twitter. Twitter blog about swapping out Ruby on Rails in their frontend -
https://blog.twitter.com/2011/twitter-search-is-now-3x-faster

Mõisted

MVC(Models View Controllers) - Enamasti veebiarenduses kasutatav mudel, kus teatud komponendid eraldatakse 3 loogilisse osasse - Model View Controller.

CoC(Convention over Configuration) - Arendusmeetod, kus eelistatakse järgida kokkuleppeid enimlevinud asjade tegemiseks, selleasemel, et ise ratast leiutada.

Controller - Ruby klass mis tegeleb andmete liigutamisega View ja Model komponentide vahel. Controller klassi nimed kirjutatakse alati mitmuses ja _controller prefiksiga. Näiteks: `books_controller.rb` ja `users_controller.rb`.

Model - Ruby klass mis defineerib Ruby on Rails rakenduse jaoks ära rakenduse loogikas vaja olevaid andmebaasi objekte. Model klassi nimed kirjutatakse alati ainsuses. Näiteks: `book.rb` ja `user.rb`.

View - Osa Ruby on Rails rakendusest, mis tegeleb andmete kuvamisega kasutaja silmadele. View objektid on defineeritud kui `.html.erb` failina. Näiteks: `index.html.erb` ja `new.html.erb`.