TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Genet Schneider 142678IAPB

POPULAARSEMATE PHP MIKRORAAMISTIKE VÕRDLUS JA NENDE KASUTAMINE TUDENGI ÕPPETÖÖS

Bakalaureusetöö

Juhendaja: Tarvo Treier
Magister MSc

Tallinn 2017

Autorideklaratsioon

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

Autor: Genet Schneider

22.05.2017

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on hinnata PHP mikroraamistiku potentsiaalset kasutamist, õpetamist ja võimalikkust TTÜ programmeerimise algõppes.

Autor analüüsib kolme mikroraamistikku: Silex, Slim ja Lumen, ning arendab sarnase veebirakenduse igas raamistikus. Mikroraamistikud valis autor nii enda kogemuse kui ka internetist leitavate raamistike võrdluste hinnangute põhjal. Arendamisest saadud kogemust kasutab autor raamistike hindamiseks, et selgitada välja, milline raamistik pakub tudengitele kõige rohkem väärtust.

Oodatavaks tulemuseks analüüsitakse, kas TTÜ informaatika bakalaureuse õpingute vältel peaks tudengitele õpetama baasprogrammeerimise kõrval ka mõnda populaarset raamistikku ning kui jah, siis milline raamistik oleks algõppes parim valik.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 50 leheküljel, 5 peatükki, 41 joonist.

Abstract

The comparison of the most popular PHP microframeworks and their usage in students schoolwork

The objective of this bachelor thesis is to evaluate the potential usage, teaching and possibility of PHP microframeworks in TUT programming basic studies.

The author analyses three PHP microframeworks: Silex, Slim and Lumen, as well as develops a simple application in all of those frameworks. The microframeworks were selected on the basis of the author's programming experience as well as the popularity and the comparison of the microframeworks found from several technical articles. In order to resolve which framework is the best to teach students, the assessment of them is conducted by using the author's experience from the development in each of those frameworks.

To reach the expected result the author analyses whether TUT students of Bachelor degree in Informatics, should be taught a popular and widely used programming framework, along with teaching the basics of a programming language. Based on the research results the author recommends the best PHP microframework.

The thesis is written in Estonian and contains 50 pages of text, 5 chapters, 41 figures.

Lühendite ja mõistete sõnastik

Apache	HTTP Veebiserver [1]
API	*Application Programming Interface*, rakendusliides [2]
Composer	*Dependency Management tool for PHP*, sõltuvuste haldamise tööriist PHP keeles [3]
Doctrine DBAL	*Database Abstraction* Layer, andmebaasi abstraktsioonikiht [4]
HTML	*Hyper Text Markup Language*, hüpertekstimärgistuskeel
HTTP	Hüperteksti edastusprotokoll
Laravel	PHP tarkvara raamistik
Lumen	PHP tarkvara mikroraamistik
MAMP	Ingl. *Web development solution stack for MacOs, Apache, MySql and PHP, Perl or Python* [5]
MySql	Andmebaasi haldamise süsteem
ORM	*Object–Relational Mapping*, objekt–relatsiooniline kaardistamine [6]
PHP	*HyperText Preprocessor*, skriptimiskeel
PHP–renderer	Vaadete formuleerimise raamistik
Pimple	*PHP Dependency Injection Container*, sõltuvuse sisestamise konteiner [7]
PSR-7	HTTP sõnumite liides [8]
Silex	PHP tarkvara mikroraamistik
Sinatra	*Domain-specific language*, domeeni–spetsiifiline keel [9]
Slim	PHP tarkvara mikroraamistik
Symfony	PHP tarkvara raamistik
Twig	Vaadate formuleerimise raamistik

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tarkvaraarendus on ala, milles parimate tehnoloogiate, programmeerimiskeelte ning raamistike kasutamine on hädavajalik, et arendada konkurentidest paremaid rakendusi, mis on kindlad, turvalised ning jõulised. Eirates populaarseid ning standardikohaseid tehnoloogiaid ning arendusmeetodeid, võib inimesteni jõudev rakendus olla liiga aeglane, halvasti hallatav ning üldiselt toores. Rakenduste paremini, efektiivsemalt ning standardikohasemalt arendamise eelduseks on tarkvararaamistike kasutamine, mis on läbini testitud ning laialt kasutatud mitmete tarkvaraarendusettevõtete poolt. Kuna tarkvara raamistik on laialt levinud tööriist arendamises ning selle kasutamine on tänapäeval tugevalt soovitatud, siis tarkvara raamistiku tundmine on nii noorem– kui ka vanemarendaja jaoks vajalik [10].

Autor asus erialasele tööle olles samal ajal ka TTÜ bakalaureuseõppe informaatika õppekava tudeng. Autoril oli enne tööle asumist baaskogemus PHP programmeerimiskeeles, kuid mitte üheski tarkvararaamistikus, kuna TTÜ bakalaureuseõppe informaatika õppekava esimestel kursustel tarkvara raamistikke veel ei käsitleta. Tarkara raamistiku kogemuse puudumise tõttu, jäid algselt mitmed disainimustrite, tehnoloogiate, sealjuures ka tarkvara raamistiku kasutamise põhimõtted arusaamatuks. TTÜ bakalaureuseõppe informaatika õppekaval ei panda õpingute alguses piisavalt rõhku kindla tarkvara raamistiku õpetamisele. Tarkvara raamisiku tundmine aitaks ülikoolist värskelt tulnud noorarendajatel integreeruda tarkvarafirmasse kiiremini, mis teeb teda väärtuslikumaks tööandjate jaoks. Tarkvaraettevõttesse tööle asudes peab noorarendaja oma praeguseid teadmisi kohandama, et vastu võtta ettevõtte poolt kehtestatud arendamismeetodid ning tavad. Olles kogenenum juba ülikooli lõpetades, aitaks see noorarendajal kaasa töö leidmisel, mis omakorda aitaks kaasa Eesti IT–alase tööjõu puuduse vähendamisele. IT–alaseid inimesi on Eestis juba liiga vähe – olles liiga vähese kogemusega, tõmbab seda numbrit veel rohkem alla [11].

Antud bakalaureusetöö eesmärgiks on hinnata PHP mikroraamistiku potentsiaalset kasutamist, õpetamist ja võimalikkust TTÜ programmeerimise algõppes.

Esimeseks ülesandeks eesmärgi saavutamiseks autor uurib ja selgitab, mis on tarkvara raamistik, mis on selle kasutamise eelised ning puudused. Autor uurib eelised, kuidas tarkvara raamistiku kasutamine tuleks tudengile kasuks nii koolitöödes kui ka tulevasel tööturul, samuti puuduseid, mis võivad tarkvara raamistike õpetamist ning kasutamist raskendada.

Teiseks ülesandeks eesmärgi saavutamiseks on saada veebirakenduse arendamise kogemus valitud raamistikes ning kasutada saadud teadmisi nende hindamiseks, võrdlemiseks ning järelduste tegemiseks.

Kolmandaks ülesandeks on uurida tarkvara raamistiku eeliseid, kasu kui ka negatiivseid külgi raamistikes arendamisest saadud kogemuse põhjal.

Töö hüpoteesideks on autor tõstatanud järgmised punktid:

- Tarkvara raamistiku õppimine ning kasutamine koolitöödes on tudengi jaoks kasulik ning oskuskohane
- Töö käigus välja valitud raamistik on sobilik tudengitele õpetamiseks

Bakalaureusetöö käigus arendab autor kõigis valitud raamistikus lihtsa blogitaolise rakenduse, milles kasutaja saab sisestada postitusi, neid salvestada ning pärast neid kuvada. Rakenduses sisestatud andmed salvestatakse MySql andmebaasi. Autor analüüsib tarkvara raamistiku õpetamise eeliseid kui ka puuduseid ning pakub välja oma kogemuse põhjal arvamuse, millist ja kuidas raamistikku õpetada.

Antud bakalaureusetöö erinevate osade ning komponentide koostamisel osaleb autor tarkvaraarendaja rollis. Töö koosneb viiest peatükist, millest kaks on sissejuhatus ning kokkuvõte. Töö kolm sisulist peatükki jagunevad järgmiselt:

- Teises peatükis teeb autor tarkvara raamistiku olemusest ülevaate ning tutvustab raamistiku definitsiooni, levinumaid kasutusalasid, kirjeldab raamistiku kasutamise eeliseid ning milline mõju on raamistike kasutamisel tänapäevasel tööturul kui ka tudengi koolitöödes.
- Kolmandas peatükis kirjeldatakse rakenduste ehitamise protsessi erinevates tarkvara raamistikes. Autor annab detailse ülevaate samm–sammult, kuidas rakenduse ehitamine igas raamistikus kulges.

- Neljandas peatükis kirjeldab ning analüüsib autor tarkvara raamistiku kasutamise mõju ning eeliseid tudengi õppetöös. Samuti pakub autor enda arvamusi ning lahendusi, kuidas tarkvara raamistiku õpetamisele läheneda.

2 Tarkvara raamistik

Antud peatükis autor uurib ja selgitab, mis on tarkvara raamistik, mis on selle kasutamise eelised ning puudused. Et võrrelda tarkvara raamistike, peab kõigepealt teadma, mis see on ja miks seda kasutatakse. Autor kirjeldab eelised, kuidas tarkvara raamistiku kasutamine tuleks tudengile kasuks nii tulevasel tööturul kui ka koolitöödes. Samuti autor analüüsib tarkvara raamistiku kasutamise puuduseid.

2.1 Mis on tarkvara raamistik?

Tarkvara raamistiku eesmärk on muuta tarkvaraarendus efektiivsemaks. Erinevad raamistikud tõstavad tarkvaraarendaja produktiivsust ja ka kirjutatud koodi kvaliteeti, töökindlust ning jõulisust [12]. Tarkvara raamistik tõstab arendaja produktiivsust lubades tal keskenduda oma rakenduse unikaalsete nõuetele, mitte kulutada enamik oma ajast rakenduse struktuuri või algoritmide implementeerimisele, mille on juba määranud tarkvara raamistik [13].

Paljude veebirakenduste kontsept on sarnane, mis tähendab, et nad sisaldavad sarnast struktuuri, algoritme, koodi, kasutavad sarnaseid andmebaasisüsteeme ja kolmanda osapoole raamatukogusid (*third–party libraries*). Tarkvara raamistik sisaldab endas erinevaid algoritme, valmis kirjutatud meetodeid ning tuge erinevatele disainimustritele, mida iga arendaja saab taaskasutada oma rakendustes.

Raamistiku taaskasutatavust saab iseloomustada näite abil. Nimelt lõigata paberitükk, mõõtudega 5m x 5m on suhteliselt kerge. Samas, kui lõigata 1000 paberitükki samade mõõtudega, siis igat lõigatavat paberitükki mõõta eraldi oleks suur ajaraiskamine. Efektiivsem oleks valmistada 5m x 5m raam, mille abil saaks lõigata paberitükke palju kiiremini ja efektiivsemalt. Selle raami kasutamine on idee poolest sama, nagu tarkvara arendades tarkvara raamistikku kasutada. Selle asemel, et sama ülesannet korduvalt korrata, me loome raami, mis teeb seda meie eest [14].

Tihti aetakse omavahel segamini kaks terminit: tarkvara raamistik ja objekt–orienteeritud tarkvara raamatukogu (*Software libray*). Objektid, meetodid ja algoritmid implementeeritud raamatukogu poolt, on kasutatud ja initsialiseeritud arendaja enda rakenduse poolt [15]. Arendaja peab teadma, milliseid meetodeid kutsuda ning millised objekte kasutada, et oma eesmärk saavutada. Tarkvara raamistiku korral implementeerib arendaja objekte ja meetodeid, mida kutsub ja initsialiseerib raamistik ise. Tarkvara raamistik defineerib kontrolli voo arendaja rakenduses [13].

Õppides ära programmeerimiskeele, õpitakse seda tehes ka algoritme ja andmestruktuure. Noorem– või vanemarendaja poolt kirjutatud algoritmid ja andmestruktuurid võivad oma eesmärki täita, kuid suure tõenäosusega jäävad need tarkvara raamistikus sisalduvate algoritmidele, andmestruktuuridele efektiivsuse poole pealt alla. Raamistiku algoritmid ja andmestruktuurid on läbini testitud. Need on hoolega programmeeritud, testitud ja parandatud, et nende toimimise efektiivsus ja kasutatavus oleks parim. "Raamistike algoritmid ja andmestruktuurid esindavad meie kollektiivset investeeringut tarkvara infrastruktuuri" [15].

2.2 Tarkvara arendamine raamistikus

Tarkvara loomine hõlmab endas erinevate spetsifikatsioonide ja nõuete täitmist. Arendades lihtsat hobiprojekti, mida arendaja teeb oma lõbuks, ei pea tingimata kasutama raamistikku. Samuti lihtsad ühelehelised veebirakendused, milles ei toimu dünaamilist andmete laadimist, registreerimist või muid komplektseid operatsioone, ei pea samuti olema ehitatud kasutades tarkvara raamistikku. Saame järeldada, et raamistiku kasutamine ei ole absoluutne kohustus [12]. Siiski raamistik on tööriist, mis aitab arendada kiiremini ja paremini. Raamistiku kasutamine annab kindluse, et arendatav rakendus on täielikult vastavuses ärireeglitega. Raamistik defineerib rakenduse koodibaasi struktuuri, mis on hooldatav, uuendatav ja skaleeritav [12].

Tarkvarafirmades on tarkvara arendamisel raamistiku kasutamine peaaegu alati kohustuslik. See kindlustab koodi pikaealisuse. Kui ettevõttes üks meeskond arendab koodi nii, nagu neile meeldib, ilma kindla raamistikuta, siis tulevikus saab seda koodibaasi hallata, parandada ja hooldada ainult see sama meeskond [12]. Kui veebirakendus on arendatud kasutades kindlat raamistiku, siis iga arendajal, olenemata sellest, kas nad osalesid arendamisel, on eeldus adopteerida see rakendus ja hallata seda

ka tulevikus. Tarkvara arendamine kasutades raamistikku on koostalitlusvõimeline turu standarditega [12].

2.3 Tarkvara raamistiku eelised

Tarkvara raamistiku peamine eelis on koodi taaskasutatavus. Selle asemel, et arendajad kulutaksid oma aega madala väärtusega ülesannete peale (näiteks geneeriliste koodi komponentide loomine), saaksid programmeerijad keskenduda äriloogika implementeerimisele. Tarkvara raamistik säästab arendajal kaks kuni kolm päeva tööd, selle asemel, et arendada täiesti algusest registreerimisvorm [12]. Üldjuhul raamistik hõlmab endas autentimismooduleid ning ka erinevaid vormi mudeleid, et selliseid ülesandeid kiiremini, ilma suurema vaevata teha. Raamistik kehtestab parimad programmeerimispraktikad, sobivaima disainimustri kasutuse ning moodsamad programmeerimistööriistad. Nendest tulenevalt tõstab raamistik programmi jõudlust ning tagab koodibaasi skaleeritavuse.

Tarkvara raamistike mõju on programmeerimismaailmale olnud märkimisväärselt suur. Tänapäeva programmid on enamasti paljude API'de[1] omavahel sidumine. Loomulikult peidab iga rakendus endas ka äriloogika ning andmete töötlemise koodi. Et mitte leiutada ratast, on loodud erineva funktsionaalsusega API'd, mida programmeerijad saavad kasutada enda rakendustes [15]. Enam ei pea mõtlema välja uusi algoritme, kuidas krüpteerida andmeid või implementeerida e–kirja süsteeme. Kõik see funktsionaalsus on juba valmis ja kasutatav. Iga moderne ja kaasaegne veebirakendus kasutab neid liideseid. Tarkvara raamistik defineerib, kuidas neid liideseid oma rakenduses edukalt ja efektiivselt siduda. Üha enam kaldub rõhk mitte programmeerimiskeele õppimisele, vaid just liideste õppimisele, kuidas nad toimivad, kuidas neid skaleerida. Tarkvara raamistik aitab liideste õppimisele rohkesti kaasa [15].

2.4 Tarkvara raamistiku puudused

Tarkvara raamistik teeb reeglina arenduse efektiivsemaks ning aitab parandada koodikvaliteeti, kuid siiski võib hoida endas ka mitmeid puudusi. Suurimaks ohuks

[1] API – *Application Programming Interface*, rakendusliides

tarkvara raamistiku kasutamisel, kujuneb arendaja oskamatus arendada rakendusi loomulikus programeerimiskeeles ilma temale tuntud raamistikus. See tähendab, et arendaja oskab rakendusi ehitada ja hallata ainult õpitud raamistikus. Kui arendaja oskab tarkvara raamistikku kasutada, ei tähenda see, et ta oskab seda programmeerimiskeelt [16]. Tehnoloogiad, disainimustrid ning koodikirjutamistavad muutuvad ajas pidevalt ning arendaja peab nendega kaasas käima, et omada kaasaegset teadmist oma professioonist.

Otsustades kasutada teatud tarkvara raamistiku, tähendab see ka teatud piiranguid. Kõik raamistikud ei toeta samu liideste implementatsioone või teeke, ega sisalda sama palju algoritme. Valides tarkvara raamistiku, peaks arendaja teadma, millist funktsionaalsust valitud raamistik pakub. Halvasti valitud raamistiku korral võib tulevikus tekkida probleem, mille korral peab arendaja raamistiku enda koodi muutma, et enda rakenduse funktsionaalsust parendada. Peab arvestama iga raamistiku piirangutega ja funktsionaalsusega, mida see raamistik pakub. Koos raamistiku funktsionaalsuse piirangutega kaasnevad ka raamistikus peituvad turvaaugud. Kuna tuntumad tarkvara raamistikud on üldiselt väga laialt kasutatud ja kui nendel raamistikel esineb mõni tõsine turvapuudus või viga, siis on haavatavad kõik seda kasutavad veebirakendused [13]. Tarkvara raamistiku kood on üldjuhul avalikult kätte saadav, mis teeb rakenduse ründajal vea või turvaaugu otsimise eriti kergeks ja kättesaadavaks.

3 Rakenduse arendamine mikroraamistikes

Käesolevas peatükis selgitab autor samm–sammult rakenduse arendamise protsessi kõigis valitud mikroraamistikes. Autor põhjendab valitud raamistike valiku ning tulemuseks analüüsib kõiki raamistike sarnaste omaduste järgi.

3.1 Mikroraamistike valik

Bakalaureusetöö näidisrakenduse arendamiseks valis autor kolm PHP mikroraamistikku. Rakenduse funktsionaalsus on lihtne: kasutaja saab sisestada, kuvada ja salvestada blogi postitusi. Sisestades enda nime, postituse tiitli ja sisu ning seejärel vajutades *salvesta* nuppu, salvestatakse postitus MySql andmebaasi. Kõiki postitusi saab hiljem vaadata, sisestatud aja järjekorras. Veebilehte tehes, jälgis autor eelkõige, kui kerge on raamistikku kasutada, kuidas implementeerida kontrollereid (*controllers*), vaateid (*views*) ning salvestada raamistikuga teenuseid (*services*), mis suhtlevad MySql andmebaasiga. Raamistike hindamine toimub tudengi vaatepunktist, et hiljem analüüsida, kas tudengid peaksid ja oleksid suuteliselt kasutama ja õppima bakalaureuse õpingute ajal ka tööturul kasutatavaid tarkvara raamistike.

Valitud mikroraamistikeks osutusid Silex, Slim ja Lumen. Mikroraamistikud valis autor nii enda kogemuse kui ka internetist leitavate raamistike võrdluste hinnangute põhjal. Valitud mikroraamistikud on programmeerijate kommuunis väga tuntud ja laialt kasutatud.

Autor võrdleb igat valitud mikroraamistikku tudengi vaatepunktist ning kommenteerib detailselt, kuidas veebilehe arendamine edenes. Kõik arendatud rakendused jooksevad lokaalses serveris (MAMP server[1]) ning kasutavad lokaalset MySql andmebaasi. Rakenduste arendamiskeel on PHP 7.0.17.

[1] https://www.mamp.info/en/ – Lokaalne arendus keskkond operatsioonisüsteemi MacOs'le

3.2 Silex mikroraamistik

3.2.1 Silex

Silex on PHP mikroraamistik, mis on ehitatud põhinedes Symfony[1] raamistikule, Pimple[2] (*PHP Dependency Injection Container*) konteinerile ja on inspireeritud Sinatra'st[3] (*Domain–specific language* Ruby programmeerimiskeeles). Nii Silex'i kui ka Symfony üks peamistest autoritest on Fabien Potencier [7]. Seetõttu sarnaneb Silex rohkesti Symfony PHP raamistikuga. Silex mikroraamistik ilmus 26.09.2014 [17].

3.2.2 Installimine ja kasutamine

Silex raamistikku saab kahel viisil installeerida [7]:

- kasutades Silex *skeleton*'i (Joonis 1) [7]:

```
composer create-project fabpot/silex-skeleton
path/to/install "~2.0"
```

Joonis 1. Silex raamistiku installeerimise käsk

- kui on soov ehitada rakendus üles paindlikumalt, võib installerida raamistiku käsuga (Joonis 2) [7]:

```
composer require silex/silex:~2.0
```

Joonis 2. Alternatiivne Silex raamistiku installeerimise käsk

Autor installeeris raamistiku viimasega. Pärast faili *.htaccess* konfigureerimist, et rakendus töötaks Apache serveris korrektselt, hakkab järgneva koodijupiga (Joonis 3) esialgne veebirakendus Silex raamistikus töötama [7].

[1] symfony.com – Tarkvara raamistik PHP keelele

[2] https://pimple.sensiolabs.org/ – Sõltuvuse sisestamise konteiner PHP keeles

[3] http://www.sinatrarb.com/ – Raamistik veebirakenduste ehitamiseks Ruby keeles

```
require_once __DIR__.
      '/../vendor/autoload.php';

$app = new Silex\Application();

$app->get('/hello/{name}',
      function($name) use($app) {
         return 'Hello '
            . $app->escape($name);
      });

$app->run();
```

Joonis 3. Näidiskood rakenduse töötamiseks Silex raamistikus

Koodibaasi kataloog *app* sisaldab kahte faili *App.php* ning *AppFactory.php*, mille eesmärkideks on registreerida vajalikud teenused (*services*), kontrollerid (*controller*), vaated (*views*), marsruudid (*routes*) ja muud rakenduse toimise jaoks vajalike mooduleid ning seejärel initsialiseerida rakendus. Faili *AppFactory.php* eesmärgiks on luua meie rakendus, ning registreerida vajalikud moodulid (Joonis 4) ning teenusepakkujad.

```php
<?php

namespace Fin\App;

use Fin\App\Controllers\BaseController;

final class AppFactory
{
	public static function createApp(): App
	{
		$app = new App();

		$app
		->registerServices()
		->registerGlobalServiceProviders()
		->registerController(
			'base_controller',
			BaseController::class
		)
		->registerRoutes();

		return $app;
	}
}
```

Joonis 4. Silex rakenduses teenuste registreerimine (*AppFactory.php*)

Sedaviisi rakenduse ülesehitamine on intuitiivne ning tagab arendajale paindlikkuse. Rakenduse jaoks vajalikud moodulid registreeritakse ühes failis. Kuna tegemist on siiski mikroraamistikuga, siis viise, kuidas rakenduse jaoks kõik vajalik registreerida, on mitmeid. Autor valis sellise meetodi, kuna tulevikus, kui rakendusele lisanduks funktsionaalsust, uue koodibaasi lisamine oleks kergem ja rakendus paremini skaleeritav.

3.2.3 Marsruudid

Autor alustas marsruutide (*routes*) registreerimisest, kuna just need defineerivad, kuidas ja milliseid kontrollereid kasutatakse. Silex'i marsruudid koosnevad [7]:

- Mustrist – muster defineerib tee ressursile ning võib sisaldada muutujaid ning ka regulaaravaldisi [7].
- HTTP meetodist (GET, POST, PUT, DELETE, PATCH, või OPTIONS) – defineerib ressursiga koostoime [7].

Failis *App.php* on marsruudid seotud rakendusega (Joonis 5):

```
public function registerRoutes(): App
{
      $this->mount('/silex',
            new BaseRoutes());

      return $this;
}
```

Joonis 5. Silex raamistikus marsruutide sidumine

Klass *BaseRoutes* defineerib ja registreerib meie rakenduse marsruudid ja nende seotuse kontrolleritega. Marsruut ’/’ seotakse kontrolleriga *BaseController (*Joonis 6*).* See tähendab, et lokaalses võrgus navigeerides aadressile *localhost:8888/silex*, kutsustakse kontrolleriklass *BaseController* ja tema meetod *index().*

```
$controllers->get('/',
      'base_controller:index');
```

Joonis 6. Silex raamistikus *GET* marsruudi registreerimine

POST marsruut defineeritakse erinevalt (Joonis 7). See defineerib, et antud marsruut on mõeldud *POST* päringute vastuvõtmiseks ja muud HTTP meetodiga päringud ei ole sobilikud antud marsruudi jaoks. Kutsustakse välja kontrolleriklass *BaseController* ja tema meetod *addPost().*

```
$controllers->post('/add-post',
      'base_controller:addPost');
```

Joonis 7. Silex raamistikus *POST* marsruudi registreerimine

3.2.4 Kontrollerid

Kontrollerid defineerivad käitumise, kui erinevatele rakenduse marsruutidele saadetakse päringuid [7]. Autor kasutab enda rakenduses kontrollerid kui klasse. See küll nõuab lisakoodi kirjutamist, kuid kokkuvõttes on koodist kergem aru saada ning rakenduse funktsionaalsus on jaotatud eraldiseisvateks funktsionaalsetes kihtideks. Autor registreeris kontrollereid *App.php* klassis. Antud meetod (Joonis 8) võtab sisendiks kontrolleri nime, kontrolleriklassi ning seob selle rakendusega.

```
public function registerController(
string $controllerKey,
string $controllerClass
): App {
      $this[$controllerKey] = function($app)
      use ($controllerClass) {
            return new $controllerClass($app);
      };

      return $this;
}
```

Joonis 8. Silex raamistikus kontrollerite registreerimine

Rakenduse *BaseController* klass kasutab *BlogUserService* ning *BlogPostService* teenuseklasse, et andmebaasist päritud andmeid saada kätte ning hiljem kuvada neid kasutajale läbi vaadete. Kontrolleriklassi meetod, mida kutsutakse juhul, kui navigeeritakse rakenduse pealehele näeb välja järgnevalt. Formuleeritakse (Joonis 9) vaate nimega *main.twig*, mis kasutab Twig[1] vaadete formuleerimismootorit.

```
return $this->render('main', $this->twig);
```

Joonis 9. Silex raamistikus vaate *renderdamine*

3.2.5 Teenused

Silex raamistik hõlmab endas teenuste konteinerit, mida saab kasutada erinevate teenuste, nagu näiteks vaadete formuleerimise mootori Twig, andmebaasiga suhtlemise, või muude teenuste registreerimiseks [7]. Sõltuvuse sisestamine (*Dependency Injection*) on disainimuster, mille korral sisestatakse sõltuvusi teenustele konstruktori kaudu, selle asemel, et initsialiseerida neid teenuste sees, või tugineda globaalsete muutujate peale. Selline disainimuster tagab koodi taaskasutamise, paindlikuse ja testitavuse [7]. Silex raamistikus hoitakse teenuseid teenuse konteineris (Joonis 10) ning teenuseid luuakse ainult juhul, kui neid kutsutakse [7].

[1] https://twig.sensiolabs.org/ – Vaadete formuleerimise mootor PHP keele jaoks

```
$app['some.service'] = function ($app) {
      return new Service(
            $app['some.other.service'],
            $app['some.service.config']
      );
};
```

Joonis 10. Silex raamistikus teenuste registreerimine

Autori rakenduses eksisteerib kahte tüüpi objekte: kasutaja (*User*) ning postitus (*Post*). Autor defineeris kaks eraldi teenust iga rakenduses kasutatava objekti jaoks: *BlogUserService* ning *BlogPostService*, mille ülesanneteks on pärida andmebaasist vastava objekti kohta andmeid ning tagastada need.

3.2.6 Andmebaasiga ühildumine

Silex raamistikus on implementeeritud *DoctrineServiceProvider*, mis on integreeritud Doctrine DBAL[1] andmebaasi abstraktsiooni kihiga, et koodi ühildumine andmebaasiga oleks kerge. Et andmebaasi integratsiooni kasutada, tuleb vastav teenus installeerida käsuga (Joonis 11) [7].

```
composer require "doctrine/dbal:~2.2".
```

Joonis 11. *Doctrine DBAL*'i installeerimise käsk

Rakenduse ühildamiseks MySql andmebaasiga, tuleb Silex rakendusega registreerida *DoctrineServiceProvider* (Joonis 12).

```
$this->register(
new SilexProvider\DoctrineServiceProvider(), [
      'db.options' =>
            $this['silex.db.config'],
]);
```

Joonis 12. *DoctrineService* registreerimine Silex raamistikus

[1] http://www.doctrine-project.org/projects/dbal.html – *Database Abstraction* Layer, andmebaasi abstraktsioonikiht

Andmebaasi korrektseks toimimiseks on vajalik registreerida andmebaasi asukoht ning autentimisinformatsioon (Joonis 13).

```
$this['silex.db.config'] = function($app) {
      return [
            'driver' => 'pdo_mysql',
            'host' => 'localhost',
            'port' => 8889,
            'dbname' => 'silex',
            'user' => 'root',
            'password' => 'root',
      ];
};
```

Joonis 13. Silex raamistikus andmebaasi konfigureerimine

Rakendus on mõeldud toimimiseks ainult lokaalses võrgus ning seda ei kasutata avalikus võrgus. Sellest tulenevalt kirjutas autor andmebaasi autentimisinformatsiooni koodi sisse, et kood oleks mõistetavam.

Koodinäide, kuidas pärida kõiki postitusi seotud MySql andmebaasist (Joonis 14).

```
public function getAll(): array
{
      $sql = 'SELECT * FROM ' . static::TABLE.
            ' p INNER JOIN users u ON u.id =
            '. 'p.written_by' .
            ' ORDER BY p.id DESC';

      $posts = $this->db->fetchAll($sql);

      return $posts;
}
```

Joonis 14. Kõikide postituste pärimise kood

3.2.7 Vaated

Silex raamistikus on reeglina vaadete genereerimine teostatud *TwigServiceProvider*'i abil, mis on integreeritud Twig vaadete genereerimise mootoriga. Twig kompileerib spetsiaalses süntaksis kirjutatud faile optimiseeritud PHP koodi kujule. Vaade võib

sisalda nii muutujaid kui ka operaatoreid. Twig on Silex'i jaoks vajalik installeerida käsuga (Joonis 15) [7].

```
composer require twig/twig.
```

Joonis 15. Twig'i installeerimise käsk

Twig'i kasutamiseks, tuleb see kõigepealt registreerida rakenduse teenuseks (Joonis 16) [7].

```
$this->register(
new SilexProvider\TwigServiceProvider(),
      array(
      'twig.path' => __DIR__ .
            '/../views')
)
```

Joonis 16. Twig teenuse registreerimine Silex raamistikus

Rakenduse kõik vaated on genereeritud kasutatud Twig tööriista. Kõikide postituste kasutajale kuvamise Twig kood (Joonis 17).

```
<h3 id="allPostsHeader">All posts</h3>
  {% if posts is defined %}
    {% for post in posts %}
      <div class="onePost">
        <h4 class="postTitle text-info">
          {{ post.title }}
        </h4>
        <blockquote class="blockquote">
          <p class="mb-0">
                  {{ post.content }}
            </p>
          <footer class="blockquote-footer">
            <cite title="Source Title">
              <a
                  href="
                  /silex/user/{{post.name}}">
                  {{ post.name }}
              </a>
            </cite>
          </footer>
       </blockquote>
      </div>
    {% endfor %}
  {% endif %}
```

Joonis 17. Kõikide postituste kuvamise kood Twig'is

Twig on väga võimas tööriist, millel on palju erinevaid võimalusi. See on kiire, laialt kasutatud ning väga efektiivne. Teiselt poolt on tegu eraldi tööriistaga, mida peab õigesti kasutama õppima, ning mis võib raamistiku õppimise kõrval muutuda suhteliselt raskeks ülesandeks.

3.2.8 Dokumentatsioon

Uue raamistiku õppimisel on selle dokumentatsioon hädavajalik. Dokumentatsioon peaks olema arusaadav, selge ning pakkuma piisavalt näiteid. Silex'i dokumentatsioon on laialdane ning sisaldab piisavalt informatsiooni, et juunior arendaja oma esimese rakenduse raamistikus edukalt tööle saaks. Lisaks abivalmi dokumentatsioonile, leiab raamistiku küsimuste kohta vastuseid ka mitmest internetiportaalist. Kuna Silex on Symfony lihtsustatud versioon ning koosneb mitmest Symfony komponendist, siis isegi Symfony dokumentatsioonist võib leida kasulikku informatsiooni [7].

3.3 Slim mikroraamistik

3.3.1 Slim

Slim on PHP mikroraamistik, mis on arendatud lihtsate ning võimsate veebirakenduste ja liideste arendamise jaoks. Raamistik sobib väga hästi ka kiireks prototüüpimiseks. Slim on loodud Rob Allen, Josh Lockhart, John Porter, Glenn Eggleton ning Andrew Smith'i poolt [8]. Esimene stabiilne Slim'i versioon ilmus 02.10.2010 [17].

3.3.2 Installimine ja kasutamine

Raamistiku loojad soovitavad Slim'ga alustada installeerides ning seejärel genereerides endale raamistiku malli käsuga (Joonis 18) [8].

```
php composer.phar create-project
      slim/slim-skeleton [my-app-name]
```

Joonis 18. Slim raamistiku installeerimise käsk

Pärast raamistiku struktuuri ülesseadmist ning *.htaccess* faili konfigureerimist, on vaja lühikest koodijuppi, et rakendus hakkaks tööle (Joonis 19) [8].

```
<?php
use \Psr\Http\Message\ServerRequestInterface
      as Request;
use \Psr\Http\Message\ResponseInterface
      as Response;

require 'vendor/autoload.php';

$app = new \Slim\App;
$app->get('/hello/{name}', function (
      Request $request, Response $response
) {
   $name = $request->getAttribute('name');
   $response->getBody()->write("Hello,
            $name");

   return $response;
});

$app->run();
```

Joonis 19. Slim rakenduse töötamiseks vajalik näitekood

Rakenduse struktuur Slim mikroraamistikus

Slim rakenduse arendas autor sarnaselt Silex raamistikus ehitatud rakendusega. Kataloogis olevate klasside *AppFactory.php* ning *SlimApp.php* ülesanneteks on initsialiseerida rakendus ning registreerida nii kohandatud teenuseid kui ka kontrollereid.

3.3.3 Marsruudid

Slim raamistikus on marsruutide registreerimine sarnase struktuuri ja koodiga. Erinevalt Silex raamistikust, on Slim raamistikus ettenähtud PHP fail, mille eesmärgiks on ainult marsruutide nimetamine ja defineerimine. Faili sisu on seega algaja arendaja jaoks rohkem intuitiivne ning mõistetavam. Marsruut '*/all-posts*' seotakse kontrollerklassi *PostController* meetodiga *getAll()* (Joonis 20).

```
$app->get('/all-posts',
      'PostController:getAll');
```

Joonis 20. *GET* marsruudi registreerimine Slim raamistikus

Marsruut koosneb mustrist ning HTTP meetodist. Marsruutide sidumine rakendusega toimub *index.php* failis tavapärase faili kaasamisega (Joonis 21).

```
// Register routes
require __DIR__ . '/src/routes.php';
```

Joonis 21. Marsruutide faili registreerimine Slim raamistikus

3.3.4 Kontrollerid

Sarnaselt Silex raamistikuga eksisteerib ka Slim raamistikus sõltuvuste konteiner, mis baseerub Pimple sõltuvuse sisestamise konteinerile, kus on võimalik hoida nii teenuseid, kui ka kontrollereid [8]. Klassis *SlimApp.php* registreeritakse rakenduse kontrolleriklassid. Ka Slim raamistikus defineeris autor kaks põhiolemit: kasutaja (User) ning postitus (Post). Mõlema olemi jaoks lõi autor kaks eraldi seisvat kontrolleriklassi (*UserController* ja *PostController*) ning ühe baaskontrolleriklassi (*AppController*). Olemite jaoks mõeldud kontrolleriklassid (Joonis 22) võtavad vastu olemitega seotud päringuid.

```
$container = $this->getContainer();

$container['UserController'] = function(
      Container $container) {
            $blogUserService =
                  $container->get(
                        'BlogUserService'
                  );
            $renderer =
                  $container->get('renderer');
      return
      new UserController(
            $blogUserService,
            $renderer
      );
};
```

Joonis 22. Slim raamistikus kontrolleri registreerimine

3.3.5 Teenused

Et kasutada sõltuvuse sisestamise konteinerit, peab selle kaasa andma rakenduse konstruktorile (Joonis 23).

```
$container = new \Slim\Container;
$app = new \Slim\App($container);
```

Joonis 23. Sõltuvuse konteineri registreerimine Slim raamistikus

Sarnaselt kontrolleritele toimub ka rakenduse iseloodud teenuste (*BlogUserService* ja *BlogPostService*) registreerimine. Teenuste eesmärgiks on olla abstraktne vahekiht andmebaasi ja kontrolleriklasside vahel. Mõlemale teenusele lisatakse konstruktori kaudu andmebaasi tabeli objekt. Meetod *BlogUserService* klassis, mis pärib andmebaasi kasutajanime järgi ning tagastab selle kasutaja ID (Joonis 24).

```
public function insertAndReturnId(
string $name
): int {
      $userId =
      $this->getByNameAndReturnId($name);

      if ($userId == -1) {
            $user = new User($name);
            $userId = $this
                  ->table
                  ->insertGetId(
                  $user->getInsertArray()
                  );
      }

      return $userId;
}
```

Joonis 24. Kasutaja lisamine andmebaasi Slim raamistikus

3.3.6 Andmebaasiga ühildumine

Slim raamistikus kasutas autor Eloquent ORM andmebaasi implementatsiooni. Objekt–relatsiooniline kaardistamine (ORM) on programmeerimismeetod andmete konverteerimiseks sobimatute tüüpi süsteemide vahel objekt orienteeritud programmeerimiskeeltes. ORM loob *virtuaalse objekti andmebaasi*, mida saab pärida kasutades programmeerimiskeelt [18]. Eloquent ORM pärineb Laravel PHP raamistikust. Igal andmebaasi tabelil on vastav mudel (*Model*), mida kasutatakse selle tabeliga suhtlemiseks. Mudel võimaldab pärida andmeid tabelist, sealjuures ka uusi andmeid lisada [4]. Eloquent andmebaasi peab Slim raamistikus installeerima käsuga (Joonis 25) [8].

```
composer require
      illuminate/database "~5.1"
```

Joonis 25. Eloquent ORM'i installeerimise käsk

Andmebaasi konfiguratsioon tuleb seada raamistiku sätete failis (Joonis 26).

```
return [
      'settings' => [
            'displayErrorDetails' => true,
            'addContentLengthHeader' => true,

            'db' => [
                  'driver' => 'mysql',
                  'host' => 'localhost',
                  'database' => 'slim',
                  'username' => 'root',
                  'password' => 'root',
                  'charset'   => 'utf8',
                  'collation' =>
                        'utf8_unicode_ci',
                  'prefix'    => '',
            ],
];
```

Joonis 26. Andmebaasi konfigureerimine Slim raamistikus

Samuti peab andmebaasi registreerima sõltuvuse sisestamise konteineris (Joonis 27) [8].

```
$container['db'] = function($container) {
      $capsule = new Manager;
      $capsule
            ->addConnection(
                  $container['settings']['db']
            );
      $capsule->setAsGlobal();
      $capsule->bootEloquent();

      return $capsule;
};
```

Joonis 27. Andmebaasi registreerimine Slim raamistikus

Andmebaasist postituse pärimine ID järgi (Joonis 28).

```
$postArray = $this->table
      ->get()->where('id', $id)->first();
```

Joonis 28. Postituse pärimine andmebaasist ID järgi

3.3.7 Vaated

Ka Slim toetab Twig vaadete genereerimise tööriista. Lisaks Twig'le on raamistiku dokumentatsioonis kirjeldatud ka PHP–renderer[1] kasutamine. PHP–renderer töötleb PHP vaadete skriptid PSR–7 (HTTP sõnumite liides) vastuse objektideks [19]. PHP–renderer tuleb sarnaselt Twig-ga installeerida (Joonis 29) [8].

```
composer require slim/php-view
```

Joonis 29. PHP–renderer'i installeerimise käsk

Vaadete töötlemise tööriista registreerimine rakendusega (Joonis 30) [8]:

```
$container['renderer'] = function($c) {
      $settings =
            $c->get('settings')['renderer'];
      return new Slim\Views\PhpRenderer(
            $settings['template_path']
      );
};
```

Joonis 30. PHP–renderer'i registreerimine Slim raamistikus

Rakenduse sätetes on defineeritud, millises kataloogis asuvad rakenduse vaadete failid. PHP–renderer tööriista kasutamine on kergem kui Twig kasutamine, kuna andmete töötlemine HTML failidega on tavapärasem ning noorarendaja jaoks tuttavam. Vaadete failid koosnevad tavapärasest HTML ning PHP koodist. Mainitud koodistiili õpitakse ka informaatika bakalaureuseõppes teisel kursusel, mistõttu ei tohiks PHP–renderer informaatika tudengitele raskusi tekitada. Kõikide postituste kuvamise kood failis *allPosts.phtml* (Joonis 31).

[1] https://github.com/slimphp/PHP-View – Vaadete formuleerimise tööriist PHP keele jaoks

```
<?php foreach($posts as $post): ?>
  <div class="onePost">
    <h4 class="postTitle text-info">
      <?php echo $post['title']?>
    </h4>
      <blockquote class="blockquote">
        <p class="mb-0">
          <?php echo $post['content']?>
        </p>
        <footer class="blockquote-footer">
        <cite title="Source Title">
            <a href="/slim/user/<?php echo
                $post['id']?>">
              <?php echo $post['name']?>
            </a>
          </cite>
       </footer>
     </blockquote>
  </div>
<?php endforeach; ?>
```

Joonis 31. Kõikide postituste kuvamise kood Slim raamistikus

3.3.8 Dokumentatsioon

Autori arvates sisaldab Slim'i dokumentatsioon piisavalt informatsiooni ning vajalike näiteid, et raamistikust aru saada. Mõnest küljest võib puudust tunda internetiportaalide abist. Võrreldes Silex'i raamistikuga, leiab internetist uuritava raamistiku kohta vähem informatsiooni. Teisest küljest raamistiku autorite poolt loodud dokumentatsioon on piisav, et raamistikuga mitte hätta jääda.

3.4 Lumen mikroraamistik

3.4.1 Lumen

Lumen on PHP mikroraamistik, mida kasutatakse eelkõige mikroteenuste ning kiirete API'de arendamiseks. Raamistik on arendatud lähtudes Laravel[1] raamistikust, ning seetõttu on kaks raamistikku väga sarnased. Lumen on praeguse seisuga üks kiiremaid PHP mikroraamistikke. Lumen suudab realiseerida ning vastu võtta 1900 päringut

[1] https://laravel.com/ – PHP tarkvara raamistik

sekundis. Kuna Lumen on arendatud lähtudes tema eelkäijast, siis paljud raamistiku komponendid tulenevad just Laravel'ist ning raamistikku saab uuendada väga lihtsalt Laravel'i raamistikuks, ilma et funktsionaalsus rakenduses muutuks [20]. Lumen'i esimene stabiilne versioon ilmus 14.05.2015 [17].

3.4.2 Installimine ja kasutamine

Dokumentatsioon pakub välja kaks võimalust, kuidas raamistik endale installeerida. Autor kasutas Composer'i[1] *create-project* käsku (Joonis 32) [20].

```
composer create-project --prefer-dist
laravel/lumen blog
```

Joonis 32. Lumen'i installeerimise käsk

Et rakendus töötaks Apache'i[2] serveris, pidi autor *.htaccess* faili konfigureerima, et kõik päringud suunatakse *index.php* faili. Kuna terve projekti struktuuri genereeris autor kasutades Composer'i käsku *create-project*, siis rakenduse toimimiseks ei ole suurt koodimuudatust vaja teha. Kataloogis *routes* olev fail *web.php* sisaldab rakenduse marsruute. Et rakendus kuvaks veebibrauseris tulemust, piisab lühikesest koodijupist (Joonis 33) [20].

```
$app->get('/', function() {
      return ['version' => '5.3']
});
```

Joonis 33. Koodinäide kuvamaks raamistikus versiooni Lumen'is

Lumen'i korral ei loonud autor eraldi rakenduse klasse, et registreerida rakenduse jaoks vaja minevad kontrollerid ning teenused. Raamistikus paiknevad teenused, kontrollerid kui ka marsruudid kindlatest asukohtades ning failides, ning raamistik ise oskab neid leida ning kutsuda. Seega pole vaja luua eraldi rakenduse initsialiseerimise faile (*App.php* ja *AppFactory.php*), erinevalt kahe eelmise raamistikuga.

[1] https://getcomposer.org/ – Sõltuvuste haldamise tööriist PHP jaoks

[2] https://httpd.apache.org/ – HTTP server

3.4.3 Marsruudid

Sarnaselt Slim raamistikuga, paiknevad ka Lumen'is marsruudid kindlas failis (*routes/web.php*). Marsruutide defineerimises ei ole midagi uut võrreldes eelnevate raamistikega. Koodijupp defineerib, et kõik päringud aadressile *localhost:8888/all-posts* suunatakse kontrolleriklassi *PostController* meetodile *getAll()*, mis tagastab kõik andmebaasis olevad postitused (Joonis 34).

```
$app->get('/all-posts',
      'PostController@getAll');
```

Joonis 34. *GET* kontrolleri registreerimine Lumen raamistikus

3.4.4 Kontrollerid

Sarnaselt Silex ning Slim raamistikule, luuakse Lumen'is kontrollereid kontrolleriklassidena, et mitte kõik päringute loogika marsruutide failis hoida. Rakenduse mõlema objekti (*User* ja *Post*) jaoks lõi autor kaks eraldi kontrolleriklassi *UserController* ja *PostController*, mille eesmärgiks on implementeerida nende objektidega seotud päringute funktsionaalsust.

Vaadeldavas raamistikuks arendatud rakenduse kontrolleriklasside ülesanneteks on, erinevalt eelnevast kahest raamistikust, pärida ka andmebaasi ning saadud andmeid vaadete abil kasutajale kuvada. See tähendab, et Lumen raamistikus täidavad kontrolleriklassid ka eelnevalt mainitud *BlogPostService* ning *BlogUserService* ülesandeid, milleks oli andmete pärimine andmebaasist. Põhjuseks on Lumen'i dokumentatsiooni põhine Eloquent ORM'i andmebaasi implementatsiooni kasutamine, mida autor kirjeldab hiljem detailsemalt.

Kontrolleriklassi *PostController* meetod *getAll()*, mille eesmärgiks on andmebaasist pärida kõik postitused, ning kuvada need vaadete abil (Joonis 35).

```
public function getAll()
{
	$posts =
		Post::query()->join(
			'users',
			'users.id',
			'=',
			'written_by'
		)->orderBy(
			'posts.id',
			'desc'
		)->get();

	if (!$posts) {
		return view('allPosts', [
			'posts' => [],
			'result' => 'No results.'
		]);
	}

	return view('allPosts', [
		'posts' => $posts
	]);
}
```

Joonis 35. Kõikide postituste kuvamise kood Lumen raamistikus

3.4.5 Teenused

Sarnaselt kahe eelneva raamistikuga, eksisteerib ka Lumen raamistikus sõltuvuse sisestamise konteiner. Mikroraamistik kasutab Laravel'i raamistikuga sama konteinerit, et arendajal oleks võimalikult palju tööriistu ning võimalusi [20].

Erinevalt Silex ning Slim raamistikust, jättis autor *BlogUserService* ja *BlogPostService* teenuste implementeerimise Lumen'i raamistikus vahele, kuna Lumen'i poolt kasutatava Eloquent ORM tõttu on nende teenuste defineerimine ebavajalik. Rakenduse mudelid *User* ja *Post* käituvad teatud mõttes teenustena, mille kaudu saab andmebaasi tabelitest pärida andmeid. Rakenduse andmebaasi igal tabelil on vastav mudel (näiteks *User* objektil on andmebaasis tabel *user*). Laravel pakub võimsat tööriista: Eloquent ORM, mis lubab arendajal otse mudeli käest küsida andmeid oma tabelist. Selleks on vaja rakenduses defineerida kaks mudelid, mis laiendavad Eloquent mudeli tüüpi (Joonis 36) [20].

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{

}
```

Joonis 36. Postituse objekti klassi Lumen raamistikus

Selline disainimuster teeb andmete pärimise arendaja jaoks väga lihtsaks. Andmete pärimine *User* tüüpi mudeli käest ID järgi (Joonis 37).

```
$user = User::query()->get()
      ->where('id', $id)->toArray();
```

Joonis 37. Andmebaasist kasutaja pärimine ID järgi Lumen raamistikus

3.4.6 Andmebaasiga ühildumine

Andmebaasi konfigureerimine on Lumen raamistikus väga lihtne. Failis *.env* tuleb määrata oma andmebaasi autentimisinformatsioon ning asukoht (Joonis 38).

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=8889
DB_DATABASE=lumen
DB_USERNAME=root
DB_PASSWORD=root
```

Joonis 38. Andmebaasi konfigureerimine Lumen raamistikus

Et kasutada Lumen'i andmebaasi fassaadi (*DB facade*), tuleb kirjutada faili *bootstrap/app.php* kaks koodirida (Joonis 39) [20].

```
$app->withFacades();
$app->withEloquent();
```

Joonis 39. Lumen raamistikus andmebaasi fassaadi võimaldamise kood

3.4.7 Vaated

HTML (*Hypertext Markup Language*) koodi sisaldavad vaated, aitavad eraldada kontrolleriklasside ning rakenduse loogikat üksteisest. Vaadete eesmärk on kasutajale kuvada asjakohaseid andmeid, loetaval kujul [20]. Sarnaselt Slim raamistikule kasutab Lumen tavapärast HTML ning PHP koodi, et vaadetes andmeid dünaamiliselt kuvada ning töödelda. Tavapäraselt ei kasuta Lumen Twig vaadete formuleerimise tööriista. Lihtne vaade Lumen'i raamistikus (Joonis 40) [20].

```
<html>
   <body>
       <h1>Hello, <?php echo $name; ?></h1>
   </body>
</html>
```

Joonis 40. Vaate kood Lumen raamistikus

Võrreldes vaadete kuvamist Slim raamistikus, siis Lumen'i vaadete tehnoloogia toimub väga sarnaselt. Raamistikus on vaadete jaoks määratud kindel asukoht (*resources/views*). Vaate genereerimine, koos kõigi postitustega (Joonis 41).

```
return view('allPosts', [
      'posts' => $posts
]);
```

Joonis 41. Vaate genereerimine Lumen raamistikus

3.4.8 Dokumentatsioon

Kuna vaadeldav raamistik lähtub suurelt osalt Laravel'st, siis paljudel juhtudel suunab mikroraamistiku dokumentatsioon lugema Laravel'i dokumentatsiooni, pakkumata ise piisavalt palju vajalike näiteid. Autori arvates peaks mikroraamistiku dokumentatsioon sisaldama rohkem näiteid ning informatsiooni, kuidas koodi struktureerida ning implementeerida just mikroraamistikus, mitte suunama lugejat pidevalt muud dokumentatsiooni lugema. Informatsioon, mis oli leitav ka mikroraamistiku dokumentatsioonist, oli enamasti abivalmis. Dokumentatsioon on abivalmis ning heade näidetega, kuid sisu jäi vajaka.

3.5 Mikroraamistike võrdlus

Kui võrreldakse raamistikke, siis tihtipeale arutatakse ning analüüsitakse raamistiku jõudlust ning kiirust, kui mitu päringut suudavad nad sekundis töödelda [17]. Raamistiku jõudluse uurimine ning analüüs on oluline, kuid praeguses kontekstis keskendub autor muudele raamistiku atribuutidele, nagu näiteks kasutamise lihtsusele, dokumentatsioonikvaliteedile ja raamistiku intuitiivsusele.

3.5.1 Silex

Silex teeb lihtsa veebirakenduse arendamise kergeks. Raamistiku dokumentatsioonist leiab kõik, mis vaja, et rakendus edukalt tööle saada. Silex'i sõltuvuse konteinerit (*Dependency Injection container*) on väga mugav kasutada ning teeb raamistikust arusaamise lihtsamaks ning arendamise intuitiivsemaks. Andmebaasi kasutamisega rakenduses (nii teenuse registreerimine kui ka andmete pärimine) ei esinenud ühtegi probleemi. Probleemseks kohaks võib kujuneda Silex'i poolt kasutatav vaadete formuleerimise teenus Twig. Juunior arendajale, võib uue raamistiku õppimine kujuneda keeruliseks ning kui peab lisaks raamistikule õppima ka eraldiseisvat suuremahulist tööriista. Et täies kasutada Silex raamistikku täies ulatuses, peaks õppima osaliselt ka Twig'i. Järgides dokumentatsiooni ning internetiportaale on antud raamistikus veebirakenduse arendamine väga lihtne ning kiire.

3.5.2 Slim

Slim on mikroraamistik, mille suurim eelis on lihtsus. Raamistiku tööle saamiseks lokaalses võrgus ei ole palju teha. Slim'i võib kasutada nii tõsiste veebirakenduste, kui ka lihtsate kodutööde arendamiseks kasutada ilma suurema vaevata. Kergesti hallatav ning arusaadav dokumentatsioon pakub tuge nii vanem– kui ka nooremarendajale. Raamistiku struktuur ning failide sisu on koheselt mõistetav, ning iga faili eesmärk on vägagi intuitiivne. Slim pakub minimaalne arv tööriistu, et implementeerida vajalik funktsionaalsus. Negatiivseks küljeks tuleb mainida raamistiku populaarsust. Võrreldes Silex'ga, on antud raamistik vähem kasutatud. Sellest tulenevalt võib küsimuste korral internetiportaalidest informatsiooni leidmisel esineda raskusi.

3.5.3 Lumen

Lumen teeb arendaja jaoks rakenduse ehitamise väga lihtsaks. Andmebaasiga ühildumine on tehtud mugavaks ning arusaadavaks ka nooremarendajate jaoks. Arendades rakenduse

mikroraamistikus on tulevikus võimalus väga kergesti uuendada oma rakendus Laravel'i raamistikule. Dokumentatsiooni kohalt leidis autor mõningaid puudujääke. Kasulikku informatsiooni ning näiteid küll leiab, kuid tihtipeale ei pakkunud mikroraamistiku dokumentatsioon õpetusi, vaid suunas Laravel'i dokumentatsiooni lugema. Laravel on aga palju võimsam ja suurem raamistik ning seega hõlmab ka palju rohkem funktsionaalsust ning komplektseid operatsioone, mistõttu on ka dokumentatsioon mahukam. Nooremarendajal võib see aga tekitada segadust ning uue raamistiku õppimine võib osutuda liiga keerulisemaks.

3.5.4 Järeldus

Kõik uuritud mikroraamistikud on oma valdkonna parimad tööriistad. Arendades rakendust kõigis kolmes raamistikus kujunes lõplik valik pigem raskeks. Ei ole raamistikku, mis domineerib teist täielikult, ega ole ka raamistikku, mis jääks teistest palju alla. Rakenduse arendamine igas valitud raamistikus aitas rohkesti kaasa parima valikul. Lumen on võimas ja kiire, kuid samas kompleksne, Slim on väga intuitiivne ning lihtne mõista, kuid mitte piisavalt populaarne ning Silex pakub kvaliteetset ning hästi dokumenteeritud koodibaasi. Autor jõudis järelduseni, et tudengi õppetöö jaoks sobib valitud raamistike seast kõige paremini Silex. Valitud raamistik on hästi dokumenteeritud ning väga populaarne, mis tähendab, et lisaks dokumentatsioonile leiab raamistiku kohta küsimustele vastuseid ka muudest tehnoloogia blogidest.

Võib järeldada, et parima raamistiku valimisel tuleb eelkõige hinnata selle dokumentatsioonikvaliteeti, koodi ning raamistiku struktuuri arusaadavust ning intuitiivsust. Kuna noorarendajal ei ole esialgu kuigi suurt kogemust tarkvara raamistikus, siis raamistiku dokumentatsioon aitab arendajat selle kasutamisel nii esimeste sammudega kui ka tehnilistemate lahenduste implementeerimisel. Samuti tuleb silmas pidada raamistiku populaarsust ning kasutatavust. Kui õpetada raamistikku, mis ei ole populaarne ega mille kohta ei leia palju abi internetifoorumitest, siis tudengid saavad küll tarkvara raamistiku kasutuskogemuse, kuid tõenäoliselt raamistiku vähese populaarsuse tõttu ei hakata seda raamistikku professionaalselt kasutama. Õpetades tudengitele algõppes baasprogrammeerimise kõrvalt ka populaarset tarkvara raamistikku, aitab tudengil mõista tänapäeval laialt kasutusel olevaid tehnoloogiaid, raamistikke ning nendes kasutatavaid disainimustreid. Oluline on, et tarkvara raamistikku õpetatakse baasprogrammeerimise ainete kõrval, et tudengid omandaksid teadmised ka

programmeerimiskeeltest, mitte ainult tarkvara raamistikest. See aitab ennetada olukorda, kus tudengid oskavad tarkvara raamistikku küll kasutada, kuid baasprogrammeerimise kogemus jääb vajaka.

4 Tarkvara raamistike kasutamine õppetöös

Antud peatükis autor analüüsib ja mõtestab tarkvara raamistiku kasutamise eeliseid ning kasu TTÜ bakalaureuseõppe informaatika õppekava tudengitele, kasutades rakenduste arendamisest saadud kogemust. Samuti uurib autor tarkvara raamistiku kasutamise negatiivseid pooli.

4.1 Kasu tudengitele

Tänapäeval on arendajal väga suur valikuvõimalus, kuidas tarkvara arendada. Üldiselt ei ole vahet, millist programmeerimiskeelt või raamistikku kasutada. Lõpptulemus sõltub ikkagi arendaja pädevusest ning tema kirjutatud koodi kvaliteedist. Tarkvara raamistikud koguvad iga aastaga jõudlust ning funktsionaalsust juurde. Need muutuvad järjest rohkem populaarsemaks ning standardikohasemaks. Tarkvara raamistikud on saamas uueks arendamiskeeleks [15]. Iga arendaja, kes soovib tulevikus tööturul edukalt hakkama saama, peab selle ideoloogiaga kaasas käima. Tarkvara raamistiku õppimine loob head eeldused. Tarkvara raamistiku kasutusoskus ja tundmine on saamas sama oluliseks nagu programmeerimiskeele oskus [15].

TTÜ Informaatika õppekava tudeng kuulub samuti arendajate ühiskonda. Paljud tudengid, sealhulgas autor, alustavad oma professionaalset töökarjääri juba ülikooli õpingute ajal. Asudes tööle tarkvara firmasse, kus kasutatakse arendamiseks populaarseid tarkvara raamistike, arendamismetoodikaid ning modernseid tehnoloogiaid, ilma et noorarendajal nendes metoodikates ning raamistikes kogemus oleks, võib muuta noorarendaja kohanemise liialt raskeks. Arendaja integreerimine ettevõttesse ei pea olema nii raske ning aeganõudev. Programmeerimiskeelte alused, kuidas defineerida muutujaid ning milline on süntaks, on väga oluline ning vajalik, kuid ülikoolid ei tohiks õpetamisel tõmmata piiri siin. Vajalik on ka parimaid tarkvara raamistikke õpetada. Analüüsida, millised raamistikud on kvaliteetsed ning miks neid kasutada. Ei pea õppima ega uurima kõiki raamistike, kuna enamik raamistikud täidavad samu ülesandeid ning jagavad osaliselt koodibaasi. Piisab ühe populaarse, laialt kasutatud tarkvara raamistiku õpetamisest. Pühendada koguni üks kuni mitu semestrit sama raamistiku kasutamise ning õppimise peale õpingute algusest, võib muuta noorarendaja suhtumist enda tulevasse ametisse drastiliselt ning valmistab teda ette astumaks tööturule. Tarkvara raamistiku

läbini õpetamine informaatika erialal aitab tudengit kasvatada edukaks ning professionaalseks arendajaks.

4.2 Raskused tarkvara raamistiku õpetamisel

Õpetada tudengitele populaarseid tehnoloogiaid ning raamistikke ei ole kerge töö. Veel raskem on õpetada neid nii, et enamik tudengeid oleksid motiveeritud ning võtaksid selle teadmise vastu. Kõige raskemaks osutub programmeerimiskeele ning sellele vastava raamistiku valik, mida õpetada. See, mis on populaarne täna, ei pruugi olla laialt kasutatud enam aasta pärast, seega tuleb valiku otsus korralikult läbi mõelda. PHP programmeerimiskeel on kasutusel 82.6% tänavu eksisteerivatest veebilehtedest [21]. Võib järeldada, et PHP ei jää ilma tähelepanuta nii pea. Seega oleks PHP õpetamine Silex raamistikus tudengitele kasulik pikemas perspektiivis. Selline teadmine aitaks neil mõista paljusid teisi raamistikke, ning raamistiku tundma õppimisel tõstaks tudengi enesekindlust hakkama saada tööturul. Analüüsides tänavu kiiresti populaarsust koguvat Node.js, võib arvata, et tulevikus võib just Node.js võtta domineeriva positsiooni veebiarhitektuuris. „Node.js on JavaScripti käitussüsteem, mis kasutab JavaScripti interpreteerimiseks Google V8 JavaScript mootorit“ [22]. JavaScript’i kasutamise tõttu seob Node.js nii *back–end* kui ka *front–end* arendajate keskkonnad, tehes keelt veel laiemalt kasutatuks ning populaarsemaks [22]. Võib väita, et Node.js on samuti väga tugev kandidaat, mida õpetada tudengitele, tarkvara raamistiku kontekstis.

Tarkvara raamistiku kasutamine on kujunemas eraldiseisvaks keeleks. Paljud fundamentaalsed meetodid, algoritmid ja tavad on tarkvara raamistikus implementeeritud hoopis teisiti, kui nooremarendaja harjunud on. Kui tarkvara raamistiku kasutamise oskuse vajadus aina kasvab, siis arendajad võivad tulevikus kaotada oskuse arendada rakendusi ilma endale tuntud raamistikuta. Tarkvara raamistikud aga ajaga muutuvad, kaovad ning uued populaarsemad tulevad asemele. Sellega võib kaasneda arendaja programmeerimiskeele baasoskuste kadumine. Et seda vältida tuleks kindlasti noortele arendajatele õpetada esmalt programmeerimiskeelte baasteadmisi ning alles hiljem õpetada tarkvara raamistike.

Autor jääb arvamusele, et valides tarkvara raamistiku, mis on mõeldud PHP keele jaoks (Silex), saab tudeng kõige kiiremini ja efektiivsemalt endale vajalikud teadmised ning eeldused õppida ka ise erinevaid teisi raamistikke.

5 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli hinnata PHP mikroraamistiku potentsiaalset kasutamist, õpetamist ja võimalikkust TTÜ programmeerimise algõppes.

Hüpoteesidena tõstatas autor järgmised punktid:

- Tarkvara raamistiku õppimine ning kasutamine on tudengi jaoks kasulik ning oskuskohane
- Töö käigus välja valitud raamistik on sobilik tudengitele õpetamiseks

Bakalaureusetöö käigus igas valitud raamistikus arendati edukalt valmis lihtne blogirakendus. Sellest tulenevalt parima PHP mikroraamistiku välja selgitamisest, võib töö autor väita, et seatud hüpoteesid leidsid kinnitust.

Veebirakenduse arendamine valitud raamistikes oli informatiivne ning kasulik. Arendamise käigus kogus autor palju uusi teadmisi, arvamusi ning see aitas kujundada seisukohta tarkvara raamistiku õpetamise ettepanekust. Uuritavad mikroraamistikud on kontseptilt sarnased, neid kasutatakse sarnaste eesmärkide täitmiseks ning nad täidavad ühiseid ülesandeid. Raamistikud erinesid üksteisest nii kasutuslihtsuselt, dokumentatsiooni kvaliteedist kui ka koodi intuitiivsuse poole pealt. Töö käigus võrreldi ja hinnati raamistikke eelkõige tudengi perspektiivist, mistõttu ei võrreldud raamistike jõudluse kiirusi. Mikroraamistike võrdlustest järeldades, on tudengitele sobiv mikroraamistikul arusaadav ja abivalmis dokumentatsioon, Mikroraamistik peaks olema piisavalt populaarne, et selle kohta leiaks informatsiooni ka internetifoorumitest.

Tarkvara raamistiku tundmine on tänapäevasel tööturul tähtis. Kui noorarendaja on tarkvaraarendusettevõtete poolt kasutatud tehnoloogiatega kursis ning oskab neid algtasemel kasutada, siis ta on väärtuslikum ka tööandjale. Arendaja kiire integreerumine uute ettevõttesse hoiab kokku nii ettevõtja raha kui ka kiirendab arendaja professionaalset kasvu. Harides tudengeid tarkvara raamistike kohta juba bakalaureuseõpingute alguses, aitab see tudengitel mõista tänapäeval laialt kasutatud tehnoloogiaid, raamistikke ja

disainimustreid paremini. Tudeng saaks varajaselt aimu, kuidas arendatakse tarkvara tööturul.

Sissejuhatuses püstitatud eesmärgid said töö käigus valitud raamistikke kasutades lihtsa blogi arendamisega ning sellest tulenevate järelduste ning otsuste tegemisega täidetud. Lisaks leidis kinnitust hüpotees, et tarkvara raamistiku õppimine ning kasutamine on tudengi jaoks väärtuslik kogemus ning eeldus tulevikus paremini tööturul konkureerida. Töö autor sai blogi arendamisega hakkama, kasutades valituid raamistikke. Võib järeldada, et autori poolt soovitatud mikroraamistikul on potentsiaal olla tulevikus informaatika tudengite õppimisaine. Lisaks seatud eesmärkidele, andis bakalaureuse töö käigus valitud raamistikes arendamine töö autorile väärtuslikku erialast kogemust.

Kasutatud kirjandus

[1] M. Robert, „https://httpd.apache.org/,“ [Võrgumaterjal]. Available: https://httpd.apache.org/. [Kasutatud 17 mai 2017].

[2] Wikipedia, „Wikipedia – API,“ 11 märts 2013. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Rakendusliides. [Kasutatud 17 mai 2017].

[3] A. Nils ja B. Jordi, „https://getcomposer.org/,“ [Võrgumaterjal]. Available: https://getcomposer.org/. [Kasutatud 10 mai 2017].

[4] Laravel, „laravel.com – documentation,“ [Võrgumaterjal]. Available: https://laravel.com/docs/5.4. [Kasutatud 8 mai 2017].

[5] Appsolute, „https://www.mamp.info/en/,“ appsolute, [Võrgumaterjal]. Available: https://www.mamp.info/en/. [Kasutatud 17 mai 2017].

[6] Hibernate, „http://hibernate.org,“ [Võrgumaterjal]. Available: http://hibernate.org/orm/what-is-an-orm/. [Kasutatud 17 mai 2017].

[7] SensioLabs, „silex.sensiolabs.org – A PHP micro-framework standing on the shoulder of giants,“ [Võrgumaterjal]. Available: https://silex.sensiolabs.org/. [Kasutatud 12 aprill 2017].

[8] J. Lockhart, A. Smith ja R. Allen, „slimframework.com – documentation,“ [Võrgumaterjal]. Available: https://www.slimframework.com/. [Kasutatud 12 aprill 2017].

[9] H. Andy, „https://www.sitepoint.com,“ 11 september 2013. [Võrgumaterjal]. Available: https://www.sitepoint.com/php-to-sinatra/. [Kasutatud 18 mai 2017].

[10] D. Lloyd, „bibliotecauniversitaria.ge.it,“ [Võrgumaterjal]. Available: http://www.bibliotecauniversitaria.ge.it/tms/expresso/doc/edg/edg_WhyUseFramework.html. [Kasutatud 18 mai 2017].

[11] A. Pau, „tehnika.postimees – Eesti pole IT–talentidele tõotatud maa,“ 22. november 2016. [Võrgumaterjal]. Available: http://tehnika.postimees.ee/3918779/ekspert-eesti-pole-it-talentidele-tootatud-maa. [Kasutatud 7 mai 2017].

[12] SensioLabs, „silex.sensiolabs.org – Why should I use a framework?,“ [Võrgumaterjal]. Available: http://symfony.com/why-use-a-framework. [Kasutatud 10 mai 2017].

[13] M. Baker, „Cimetrix,“ 02 Oktoober 2009. [Võrgumaterjal]. Available: http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em. [Kasutatud 8 mai 2017].

[14] N. Choudhary, „Stackoverflow.com – Answer to "What is a software framework?",“ 4 Oktoober 2012. [Võrgumaterjal]. Available: http://stackoverflow.com/questions/2964140/what-is-a-software-framework. [Kasutatud 12 mai 2017].

[15] P. Wayner, „infoworld.com – 7 reasons why frameworks are the new programming languages,“ 30 märts 2015. [Võrgumaterjal]. Available: http://www.infoworld.com/article/2902242/application-development/7-reasons-why-frameworks-are-the-new-programming-languages.html. [Kasutatud 6 mai 2017].

[16] 1stwebdesigner.com, „1stwebdesigner.com,“ 28 Mai 2015. [Võrgumaterjal]. Available: https://1stwebdesigner.com/web-frameworks/. [Kasutatud 17 mai 2017].

[17] N. Modess, „PHP micro framework for your REST API,“ 7 jaanuar 2016. [Võrgumaterjal]. Available: https://modess.io/php-micro-framework-for-a-restful-api-part-1/. [Kasutatud 6 mai 2017].

[18] Wikipedia, „Wikipedia – Object-relational mapping,“ 9 märts 2017. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Object-relational_mapping. [Kasutatud 8 mai 2017].

[19] G. Eggleton, „Github.com,“ 29 september 2015. [Võrgumaterjal]. Available: https://github.com/slimphp/PHP-View. [Kasutatud 12 mai 2017].

[20] Laravel, „lumen.laravel.com – documentation,“ [Võrgumaterjal]. Available: https://lumen.laravel.com/docs/5.4. [Kasutatud 12 aprill 2017].

[21] „W3Techs,“ 12 mai 2017. [Võrgumaterjal]. Available: https://w3techs.com/technologies/overview/programming_language/all. [Kasutatud 12 mai 2017].

[22] Wikipedia, „Wikipedia – Node.js,“ 10 september 2016. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Node.js. [Kasutatud 8 mai 2017].

Lisa 1 – Rakenduse lähtekoodi aadress Silex raamistikus

https://bitbucket.org/genetschneider/silex-public

Lisa 2 – Rakenduse lähtekoodi aadress Slim raamistikus

https://bitbucket.org/genetschneider/slim-public

Lisa 3 – Rakenduse lähtekoodi aadress Lumen raamistikus

https://bitbucket.org/genetschneider/lumen-public