

**Project “JOBIT - Innovative teaching methodologies and courseware for software development VET to reduce skills gap in IT“**

**Kuidas luua lihtne veebirakendus kasutades PHP raamistikku Silex?**

**Õppematerjal**

Author: Kadri Vahtramäe, I-Sepp OÜ

BCS Koolitus (Estonia)

Contributors:

European University Cyprus (Cyprus)

Centro Internazionale per la Promozione dell’ Educazione e lo Sviluppo (Italy)

Mai 2017

Table of Contents

[Sissejuhatus 4](#_30j0zll)

[Mis edasi? 4](#_1fob9te)

[PHP põhiteemad 5](#_3znysh7)

[Keele süntaks 5](#_2et92p0)

[Muutujad 5](#_tyjcwt)

[Muutuja skoop 5](#_3dy6vkm)

[Eeldefineeritud muutujad 5](#_1t3h5sf)

[Konstandid 5](#_4d34og8)

[Avaldised ja tehted 5](#_2s8eyo1)

[Aritmeetikatehted 6](#_17dp8vu)

[Omistamine 6](#_3rdcrjn)

[Võrdlustehted 6](#_26in1rg)

[Loogikatehted 6](#_lnxbz9)

[Stringitehted 6](#_35nkun2)

[Massiivitehted 6](#_1ksv4uv)

[Juhtstruktuurid 6](#_44sinio)

[Valikud 6](#_2jxsxqh)

[Kordused (Loops) 6](#_z337ya)

[Funktsioonid 7](#_3j2qqm3)

[Vigade käsitlemine – vead ja erindid 7](#_1y810tw)

[Objekt-orienteeritus 7](#_4i7ojhp)

[Klassid ja objektid 7](#_2xcytpi)

[Funktsioonid \_\_contstruct and \_\_invoke 7](#_1ci93xb)

[Nimeruumid 7](#_3whwml4)

[Klasside funktsionaalsuse laiendamine 7](#_2bn6wsx)

[Nähtavus (Visibility) 7](#_qsh70q)

[Silumine (Debugging) 8](#_3as4poj)

[Andmete saatmine veebiserverile - GET, POST, REQUEST 8](#_1pxezwc)

[PHP konfigureerimine 9](#_49x2ik5)

[PHP käsurealt kasutamise näide 9](#_2p2csry)

[PHP-serveri käivitamise võimalused 9](#_147n2zr)

[Dokumenteerimine 11](#_3o7alnk)

[Testimine 11](#_23ckvvd)

[Raamistikud 13](#_ihv636)

[Tuntumad PHP raamistikud 13](#_32hioqz)

[Symfony/Silex 13](#_1hmsyys)

[Git 13](#_41mghml)

[Veebirakenduse loomine 14](#_2grqrue)

[GitLab 14](#_vx1227)

[SSH-võtme lisamine GitLabis 14](#_3fwokq0)

[Git 15](#_1v1yuxt)

[Uue repositooriumi loomine 15](#_4f1mdlm)

[Esimene commit (Git) 16](#_2u6wntf)

[Projektile Silexi lisamine 16](#_19c6y18)

[Composer’i kasutamine 16](#_3tbugp1)

[Gitignore 17](#_nmf14n)

[PHP sisseehitatud veebiserveri käivitamine ja serveris PHP-faili käivitamine 18](#_37m2jsg)

[Esimese HTML-malli loomine 19](#_1mrcu09)

[Twig kasutamine 19](#_46r0co2)

[Põhistiili lisamine ja HTML 20](#_2lwamvv)

[Esmakordne Twig kasutamine, jagades lehekülje komponentideks 21](#_111kx3o)

[Andmebaasi loomine kasutades Doctrine DBAL 21](#_3l18frh)

[Composer - PSR-4 Autoloading 23](#_206ipza)

[Veel meetodeid klassis Application.php 27](#_4k668n3)

[Vormi loomine 28](#_2zbgiuw)

[Kontrolleri koodi eraldamine eraldiseisvaks PHP klassiks 32](#_3ygebqi)

[Andmete salvestamine andmebaasi 34](#_2dlolyb)

[Andmebaasi sisestatud andmete vaatamine 36](#_sqyw64)

[Menüü loomine 39](#_3cqmetx)

[Sissejuhatus testimisse: PHPUnit 40](#_1rvwp1q)

Kuidas käivitada Gilab’ist alla laetud valmis rakendust? [43](#_4bvk7pj)

[Lõpetuseks 44](#_2r0uhxc)

# Sissejuhatus

Õppematerjali peamine eesmärk on õppida kasutama PHP programmeerimiskeelt. Kuid see pole ainus eesmärk. Lisaks tahame õppematerjali abil tõestada, et PHP keelel on tarkvaraarenduses oluline roll, ja näidata, kuidas seda keelt praktiliselt kasutatakse. Kuna ärimaailmas ei kasutada üldiselt PHP’d iseseisvalt, on õppematerjali eesmärk näidata seda, kuidas PHP on üks oluline ehitusblokk veebirakenduste arendamisel, kuid tegelikkuses seotakse teda tihti paljude teiste juba valmisehitatud süsteemidega. PHP populaarsuse põhjuseks on avatud lähtekood ja võimalus siduda kood HTML-iga. Õppematerjalis toodud näited ja soovitused peaksid andma algajale arendajale esimese praktilise kogemuse veebirakenduse loomisel.

Seda materjali lugedes paned ilmselt tähele, et siin on palju linke teistele materjalidele. Kindlasti on äärmiselt vajalik kõik need lingid avada ja põhjalikult läbi töötada. Selles õppematerjalis pole õppimise osa lugeja eest ära tehtud, nii et lugeja võiks vaid seda dokumenti lugedes kõigest aru saada. See õppematerjal sisaldab vaid praktilise rakenduse loomist koos selgitustega. Teoreetilisele osale viidatakse linkide abil. Enne rakenduse looma asumist, tuleks kõigepealt saavutada mingi tase PHP-keele põhifunktsionaalsuse tundmaõppimises. Et lugejal oleks lihtsam kuskilt peale hakata, lisasime siia materjali teemade kaupa materjali lingid, mida peaks PHP kohta teadma, enne kui hakkad ühte rakendust looma.

Lisaks, paned kindlasti tähele, et osad lingid on nime all *commit*, mis viitab Gitlabi lehele, kuhu on üleslaetud see sama rakendus juba valmis tehtult, ning õppematerjalis kirjeldatud sammude haaval on sealt võimalik kaasa jälgida kõiki koodimuudatusi.

## Mis ootab ees?

Lisaks uue keele (PHP) õppimisele, saad ka esimese kogemuse laialdaselt kasutatavatest töövahenditest, mida üldiselt tarkvaraarenduse meeskondades kasutatakse.

Teemad:

* PHP ja Silex raamistik
* Versioonihaldussüsteemid (Git, Gitlab)
* Pakihaldus (Dependency manager - tarkvara pakihaldus programm – Composer)
* Ühiktestimine (Unit testing - PHPUnit)
* Mallimootor (Templating languages - Twig)
* Andmebaasid (Databases - MySQL, SQLite)

# PHP põhiteemad

[PHP](http://php.net/manual/en/index.php) lühend tuleneb nimetusest „Personal Home Page Tools (PHPT), kuid nüüd tõlgendatakse *Hypertext Preprocessor* (hüperteksti eeltöötleja) ja on üks kõige enam kasutatud serveripoolne skriptikeel. [PHP](https://www.tutorialspoint.com/php/php_introduction.htm) on skriptimiskeel, mida kasutatakse peamiselt serveripoolsetes lahendustes dünaamiliste veebilehtede loomisel. Skriptimiskeel tähendab, et programmi koodi ei kompileerita enne kasutamist, vaid see tehakse täitmise ajal. PHP on serveripoolne keel, mis tähendab, et kasutaja saadab HTTP-päringu vastavale aadressile. Sellel aadressil asuv veebiserver võtab päringu vastu ja käivitab parseri. Parser on süntaksianalüsaator, mis jaotab koodi süntaksielementideks, antud juhul loob failist baitkoodi, seejärel baitkood interpreteeritakse ja veebiserverile tagastatakse HTML kood, mis omakorda saadetakse kasutaja brauserisse HTTP-vastusena.

## Keele süntaks

<http://php.net/manual/en/language.basic-syntax.php>

<https://www.tutorialspoint.com/php/php_syntax_overview.htm>

## Muutujad

<http://php.net/manual/en/language.types.php>

<https://www.tutorialspoint.com/php/php_variable_types.htm>

<https://www.tutorialspoint.com/php/php_arrays.htm>

<http://php.net/manual/en/language.variables.basics.php>

### Muutuja skoop

<http://php.net/manual/en/language.variables.scope.php>

### Eeldefineeritud muutujad

<http://php.net/manual/en/language.variables.predefined.php>

## Konstandid

<http://php.net/manual/en/language.constants.syntax.php>

<https://www.tutorialspoint.com/php/php_constants.htm>

## Avaldised ja tehted

<https://www.tutorialspoint.com/php/php_operator_types.htm>

<http://php.net/manual/en/language.operators.php>

<http://php.net/manual/en/language.expressions.php>

### Aritmeetikatehted

<http://php.net/manual/en/language.operators.arithmetic.php>

### Omistamine

<http://php.net/manual/en/language.operators.assignment.php>

### Võrdlustehted

<http://php.net/manual/en/language.operators.comparison.php>

### Loogikatehted

<http://php.net/manual/en/language.operators.logical.php>

### Stringitehted

<http://php.net/manual/en/language.operators.string.php>

### Massiivitehted

<http://php.net/manual/en/language.operators.array.php>

## Juhtstruktuurid

<http://php.net/manual/en/language.control-structures.php>

### Valikud

<http://php.net/manual/en/control-structures.if.php>

<http://php.net/manual/en/control-structures.else.php>

<http://php.net/manual/en/control-structures.elseif.php>

<https://www.tutorialspoint.com/php/php_decision_making.htm>

<http://php.net/manual/en/control-structures.switch.php>

### Kordused (Loops)

<https://www.tutorialspoint.com/php/php_loop_types.htm>

<http://php.net/manual/en/control-structures.while.php>

<http://php.net/manual/en/control-structures.do.while.php>

<http://php.net/manual/en/control-structures.for.php>

<http://php.net/manual/en/control-structures.foreach.php>

<http://php.net/manual/en/control-structures.break.php>

<http://php.net/manual/en/control-structures.continue.php>

## Funktsioonid

<https://www.tutorialspoint.com/php/php_functions.htm>

<http://php.net/manual/en/language.functions.php>

## Vigade käsitlemine – vead ja erindid

<https://www.tutorialspoint.com/php/php_error_handling.htm>

<http://php.net/manual/en/language.errors.php>

<http://php.net/manual/en/language.exceptions.php>

## Objekt-orienteeritus

<https://www.tutorialspoint.com/php/php_object_oriented.htm>

### Klassid ja objektid

<http://php.net/manual/en/language.oop5.php>

### Funktsioonid \_\_contstruct and \_\_invoke

Kasutame funktsioone \_\_construct ja \_\_invoke Silexui projektis!

<http://php.net/manual/en/language.oop5.magic.php>

### Nimeruumid

Nimeruumi kasutatakse Silexi projektis!

<http://php.net/manual/en/language.namespaces.rationale.php>

<http://php.net/manual/en/language.namespaces.importing.php>

<https://knpuniversity.com/screencast/php-namespaces-in-120-seconds/namespaces>

### Klasside funktsionaalsuse laiendamine

Kasutame laiendamist Silexi projektis!

<http://php.net/manual/en/reflection.extending.php>

### Nähtavus (Visibility)

Kasutame seda Silexi projektis!

<http://php.net/manual/en/language.oop5.visibility.php>

## Silumine (Debugging)

Väga oluline on õpetada uuele programmeerijale silumisoskust. Alates hetkest kui programmeerija hakkab programmeerima, siis peab ta ka oskama leida vigu, mida ta programmi kirjutamisel teeb.Silumisoskus aitab programmeerijat, kui ta jääb koodi kirjutamisel hätta ja aitab tal kiiresti probleeme tuvastada.

<http://php.net/manual/en/function.var-dump.php>

## Andmete saatmine veebiserverile - GET, POST, REQUEST

<https://www.tutorialspoint.com/php/php_get_post.htm>

<http://php.net/manual/en/reserved.variables.get.php>

<http://php.net/manual/en/reserved.variables.post.php>

<http://php.net/manual/en/reserved.variables.request.php>

## PHP konfigureerimine

PHP keelel on põhiliselt kolm põhilist kasutusala (loe lisaks artiklit: [What can PHP do?](http://php.net/manual/en/intro-whatcando.php)).

Neist kaks varianti on järgmised:.

1. PHP serveripoolne kasutamine. See on kõige traditsioonilisem ja peamine PHP kasutus. Selleks on tarvis kolme vahendit: PHP parser (CGI või serveri moodul), veebiserver ja veebibrauser. Sul peab olema töötav veebiserver,kuhu on installeeritud PHP. PHP programmi tulemused kuvatakse veebibrauseris, veebiserver saadab päringu PHP parserile ja parser täidab failis PHP osad ja saadab selle tagasi veebiserverile. PHP koodi tulemust võid näha oma veebibrauserist. Kui sa sisestad veebibrauserisse oma veebiserveris seadistatud aadressi, siis jõuab päring sinu veebiserverisse. Sealt edasi suunab server koodifaili PHP süntaksianalüsaatorile/parserile, kus see käivitatakse ning tagastatakse tulemus, mille veebiserver omakorda saadab tagasi veebibrauserile. Sa võid luua veebiserveri ka näiteks oma koduarvutisse, et oma PHP programme testida.

Inglise keelse juhendi leiad [siit](http://php.net/manual/en/install.php).

1. PHP kasutamine käsurealt. PHP koodi saab kirjutada ka ilma serveri ja brauserita. Läheb tarvis vaid PHP parserit, kasutusjuhendi parseri kasutamiseks leiad [siit](http://php.net/manual/en/features.commandline.php).

### PHP käsurealt kasutamise näide



1. Ava oma operatsioonisüsteemis käsurida.
2. [Installi](http://php.net/manual/en/install.unix.debian.php) PHP enda arvutisse.

Linux Ubuntu puhul kasuta järgmisi käske:

sudo apt-get install php7.0-cli

sudo apt-get install php7.0-xml

1. PHP info saad käsuga php -i (["PHP" korralduse kasutamise dokumentatsioon](http://php.net/manual/en/features.commandline.options.php))
2. Käivita PHP interaktiivses režiimis sisestades php -a
3. Testimiseks sisesta echo “Hello world!”; Kui tulemuseks on Hello world!, siis oled kasutanud PHP käsurealt ([PHP-failide käivitamine käsurealt](http://php.net/manual/en/features.commandline.usage.php))!

## PHP-serveri käivitamise võimalused

* LAMP (Linux, Apache, MySQL, PHP)
* WAMP (Windows, Apache, MySQL, PHP)
* PHP sisse-ehitatud veebiserver

Õppematerjalis kasutame viimast varianti st kasutame Linuxi operatsioonisüsteemi ja sisemist veebiserverit. Kaks esimest variant - LAMP või WAMP - konfigureerimine ja õppimine võtab rohkem aega.

# Dokumenteerimine

Mõned praktilised nõuanded rakenduste dokumenteerimiseks.

1. Rakenduse kood on kõige olulisem dokument. Kood peab olema selge ja lihtne, et teised arendajad sellest aru saaksid. Koodis peavad olema mõned vajalikud kommentaarid, aga kindlasti ei tohi üle kommenteerida. On oluline, et kood sisaldaks kasulikke kommentaare ja selgitusi keerukamatest kohtadest, samas on oluline teha vahet kommentaaridel, mis ei lisa selgust, vaid pigem lisavad koodi üleliigset kära. Tavaliselt on ebavajalikud kommentaarid need, mis koodile peale vaadates juba ilmselged on. Õppematerjalis on olemas ka mõned näited kommenteerimisest.
2. Versioonihaldusel on dokumenteerimises oluline roll. Kõikide koodi osade kohta tuleb teha sissekanne ehk inglise keeles *commit*. *Commit*’ile tuleb anda sisu kirjeldav nimetus (vajadusel täpsem). Nimetused on väga olulised, sest sellest moodustub ajalugu, mille poole saab silumise ajal tagasi pöörduda. Hea *commit’*i kirjeldus on äärmiselt oluline, sest sellest moodustub ajalugu. Näiteks hiljem, kui otsid koodist vigu või mõtled, kas mingit koodirida võiks ära kustutada, on need kirjeldused väga kasulikud. *Commit*’i kirjeldus peaks sisaldama endas üldiselt, et millist osa projektist see muudatus hõlmab ja mis oli nende muudatuste eesmärk. Kui on erimärkusi või probleeme, siis ka need tuleks *commit*’is märkida ([Hea artikkel sellest, kuidas kirjutada häid *commit*’i kirjeldusi](http://chris.beams.io/posts/git-commit/)).
3. Dokumenteerimist on rohkem projekti alguses, kindlasti tuleb dokumenteerida lisavõimalusi (-arendusi). Iga projekt peab sisaldama readme-faili, milles on kirjeldatud projekti kõige olulisemad osad. Õppematerjalis on README-faili näited ja on märgitud, millal tuleks faili uuendada.

# Testimine

Testimiseks on erinevaid mooduseid. Mõned moodused on näiteks järgmised:

1. **Testimine koodi kirjutamise ajal**. Kõige loomulikum testimismeetod iga programmi kirjutamisel. Koodi kirjutamisel tuleb see ka testida, et olla kindel, et iga kord kui käivitad ükskõik millise programmiosa, siis see töötab ootuspäraselt. Kirjutatud koodi ei tohi kunagi edastada või avalikustada ilma testimata. See võib juhtuda, kui oled kogenud programmeerija või tegid koodis väga lihtsaid muudatusi.
2. **Keegi teine testib.**  Väga kasulik on lasta oma koodi testida kellelgi teisel. Tavaliselt tuleks teha seda siis, kui oled mingi mooduli programmeerimise lõpetanud, suuremate projektide puhul tuleks teha etapi lõppemisel. Teine inimene võib märgata probleeme või loogikavigu, mida koodi kirjutaja ise pole märganud. Kõrvaltvaataja võib avastada uusi kasutamisvõimalusi, mis võivad põhjustada vigu, mis programmeerijal märkamata jäid.
3. **Koodi läbivaatamine (*code review*)**. Koodi peaks läbi vaatama samuti teine inimene, eeldatavasti teine arendaja. Teine inimene võib märgata probleeme ja loogikavigu, mida arendaja ise ei märganud. Läbi vaatamine aitab koodi lihtsustada või koodi tuleks refaktoreerida nii, et ka koodi hilisemal lugemisel on see mõistetav. Läbi vaatamise tulemusena muutub kood jätkusuutlikumaks.
4. **Automaattestid**. Automaatteste vajatakse keeruliste programmi osade testimisel, sest mõnikord on kõikide kasutuslugude käsitsi testimine keeruline või väga ajamahukas. Mida rohkem on automaatteste, seda parem. Hea on omada automaatteste kõigi projekti osade kohta. Väga hea tava on see, kui projektis on olemas testid iga erineva koha kohta, kus sa ühte ja sama keerukamat funktsionaalsust kasutad. Sellisel juhul, kui sa muudad oma taaskasutatavat üldist funktsionaalsust, mis on kasutusel mitmes kohas, saad olla kindel, et kõigis neis teistes kohtades ei tekkinud programmivigu. Sest testid katavad põhifunktsionaalsuse ja näitavad, kui kuskil mingi probleem tekkis. Õppematerjali praktilistes ülesannetes on näide ka PHP testimisvahendit PHPUnit kasutamisest.

# Raamistikud

[Raamistik](https://en.wikipedia.org/wiki/Software_framework) on kontseptuaalne struktuur, mille eesmärk on toetada ja juhendada veebirakenduse loomist. Raamistik annab ette struktuuri, millest lähtudes saab luua endale vajaliku eesmärgiga rakendus. Tarkvara raamistik on abstraktne tarkvara, mille üldiste funktsionaalsuste valikulisel kasutamisel ning enda kohandatud koodi lisamisel valmib tellimusrakendus. Sageli tuleb erinevates projektides kasutada sarnast funktsionaalsust erineva sisu ja andmetega. Raamistik väldib olukorda, kus iga uue projekti alustades tuleb samu probleeme uuesti lahendada ja koodi uuesti kirjutada. Programmeerijal on võimalik kasutada raamistiku üldist funktsionaalsust ning lisada omalt poolt vaid projekti eripärad. Raamistiku kasutamine aitab säästa aega ning aitab oma koodi paremini organiseerida ja on kergemini muudetav.

([Miks kasutada raamistikku?](https://www.lynda.com/CakePHP-tutorials/Why-use-framework/315196/378332-4.html), [Millal kasutada raamistikke?](https://www.templatemonster.com/blog/php-frameworks-when-and-why-to-use-them/))

## Tuntumad PHP raamistikud

1. Laravel
2. Symfony(Silex)
3. PhalconPhp
4. Zend
5. CakePhp
6. Slim

## Symfony/Silex

Symfony on komplekt PHP komponente, veebirakenduse raamistik, põhimõtete kogum ja kommuun —mis kõik harmooniliselt koos toimivad ([6 põhjust, miks kasutada Symfony raamistikku](https://symfony.com/six-good-reasons)).

# Git

Git on kõige enam kasutatud versioonihaldussüsteem. Versioonihalduse lihtne definitsioon ütleb, et see võimaldab hoida silma peal kogu kirjutatud koodil. Versioonihalduse kasutamine annab koodi kohta ajaloo, millal mingi osa kirjutati või uuendati. Võimaldab kirjutada koodi meeskonnas, nii et iga arendaja võib kirjutada oma koodi teisi segamata. Lisaks annab Git hea tava, kuidas dokumenteerida ja põhjendada koodis tehtud muudatusi. ([Lisalugemist](https://www.atlassian.com/git/tutorials/what-is-git/version-control-with-git) ja [video](https://git-scm.com/video/what-is-git))

Saamaks kiiret sissejuhatust, mis on Git, proovi järgmist linki [interaktiivne juhend](https://try.github.io/levels/1/challenges/1), lisaks veel [juhend](https://onlywei.github.io/explain-git-with-d3/) edasijõudnutele.

# Veebirakenduse loomine

Koodihoidla (*a source control application*), repositooriumi haldur – on rakendus, mida tuleb kasutada, kui arendus tehakse meeskonnas. Koodihoidla võimaldab koodi jagada, jälgida muudatusi ja automatiseerida mitmeid arendusprotsessi etappe. Üks kõige kasulikumaid töövahendeid ja siit saavad tavapäraselt projektid alguse.

Kõige tuntumad koodihoidlad:

1. Github
2. Bitbucket
3. GitLab

Põhimõtteliselt on kõik kolm olemuselt sarnased ja töötavad käsikäes versioonihaldussüsteemiga Git. Git-i kasutamist selgitatakse täpsemalt järgnevates peatükkides. Näidisrakenduse koostamisel on koodi hoidmiseks kasutatud GitLab-i ([Sissejuhatus GitLabi kasutamiseks](https://about.gitlab.com/2015/05/18/simple-words-for-a-gitlab-newbie/)).

Õppematerjalis kasutatud näidisrakenduse leiate aadressilt - <https://gitlab.com/i-sepp/bcs-koolitus/>

## GitLab

1. Loo endale kasutajakonto GitLab’i <https://gitlab.com/users/sign_in> ja logi sisse.
2. Loo uus projekt - <https://gitlab.com/projects/new>.
3. Ava oma projekt ja sealt leiad edasise juhtnöörid.

## SSH-võtme lisamine GitLabis

Vältimaks teiste kasutajate ligipääsu rakenduse koodile kasutame SSH-võtit. Võti võimaldab GitLabi sisse logida sisse käsurealt. See tähendab seda, et igal arvutil on oma unikaalne SSH-võti ja see lisatakse GitLab kasutajakontole. Seega iga kord, kui oled GitLabiga ühenduses, siis kasutatakse SSH-võtit kasutaja autentimiseks.

Esmalt sisene GitLabi ja otsi üles oma profiiliseaded (*profile settings*) ja seejärel vali [*SSH-keys*](https://gitlab.com/profile/keys)*.*

Järgmiseks pead lisama oma arvuti SSH-võtme. Eeldatavasti pead võtme genereerima.

Juhendid, kuidas teha:

<https://gitlab.com/help/ssh/README>

<https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>

Nüüd peaks olema kõik olemas, et alustada GitLabi kasutamist!

NB! Võimalik on ka mitte kasutada Gitlabi või serverit, vaid võid õppida kasutama Git kasutades seda lokaalselt oma arvutis. Selleks tuleb installeerida Git, konfigureerida globaalne muutuja (*global variable* (vaata järgmist peatükki)) ja initsialiseerida Git kaustas, kus soovid seda kasutada, korraldusega git init.

cd Projects/my\_project\_name

git init

Seejärel saad alustada kasutamist (*git add, git commit* jne.)

## Git

1. Installeeri Git oma arvutisse (eelistatud on operatsioonisüsteem Linux).

Näide: Ubuntu puhul sisesta käsk:

sudo apt-get install git

Juhend Git installeerimiseks erinevates keskkondades asub [siin](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git).

Linux juhendid leiad [siit](https://git-scm.com/download/linux).

1. Seejärel tuleb konfigureerida globaalsed muutujad, et Git oskaks neid tulevikus kasutada ja seostada kirjutatava koodiga.

Lõpuks tekib ajalugu, kus on näha, kes millise koodiosa kirjutas. Juhendi leiad oma uuest GitLabi projektist - Git global setup.

git config --global user.name “Firstname Lastname”

git config --global user.email “youremail@gmail.ee”

1. Git on nüüd kasutamiseks valmis!

## Uue repositooriumi loomine



Juhend on samuti olemas sinu GitLabi uues projektis.

Edasi saame luua ühenduse Sinu arvuti kausta ja GitLabi projekti vahel.

1. Alustuseks sisesta käsureale korraldus, millega liigud kausta, kuhu soovid luua oma projekti failid.

Meie näites on selleks eraldi kaust nimega Projects.

cd Projects/

1. Seejärel kopeeri GitLabist projekti kaust oma arvutisse.

NB! Nimi peab vastama projektile.

git clone [git@gitlab.com](mailto:git@gitlab.com):my\_project\_name.git

1. Loo fail README.md.

Readme-failil on oluline osa projekti dokumentatsioonis. Fail peab sisaldama kogu vajalikku informatsiooni projekti kohta. Loomulikult tuleks seda faili jooksvalt projekti edenedes täiendada, kuid esialgu tuleks fail lihtsalt luua.

Mõned readme-failide mallid leiad järgmistelt linkidelt:

<https://gist.github.com/PurpleBooth/109311bb0361f32d87a2>

<https://gist.github.com/jxson/1784669>

Readme-faili saad luua lihtsalt luues uue faili oma projekti kausta või kasutades käsurida:

cd my\_project\_name/

touch README.md

## Esimene *commit* (Git)

Nüüd saame muudatused GitLabi üles laadida ja see on meie projekti esimene ***commit*** ehk sissekanne!

git add README.md

git commit -m “You have to write here the message of your commit. Example: Added README.md”

git push -u origin master

Viimane korraldus saadab muudatused GitLabi serverisse.

Kui lähed GitLabi, siis peaks README-md fail juba seal olemas olema. Samuti näed Repository->Commits oma esimest *commit*’ti.

Lisaks saad vaadata meie projekti esimest sissekannet [siit](https://gitlab.com/i-sepp/bcs-koolitus/commit/5f7c35d234270291458ad5bb69846a4859edcd25).

Nüüd saame alustada koodi kirjutamisega!

## Projektile Silexi lisamine

### Composer’i kasutamine

Selleks, et [laadida alla Silex](http://silex.sensiolabs.org/doc/master/usage.html#installation) oma arvutisse, peame kasutama komponenti [Composer](https://getcomposer.org/doc/00-intro.md).

Composer on PHP pakihaldus või ka teegihaldur Saad seda kasutada laadides alla automaatselt kõik vajaliku kolmandate poolte tarkvara. See lihtsustab kolmandate osapoolte koodi kasutamist sinu tarkvaras.

Samuti saad muudatuste logi, milliseid muudatusi (*dependencies*) sinu projektis tarvis läheb. Ja igaüks, kes sinu koodi kasutab, saab täpselt identse teekide (või PHP pakkide) versiooni.

Tänu Composerile tekib sinu projekti fail nimekirjaga kõikidest kasutatavatest kolmanda osapoolte pakkidest. See tagab selle, et kui keegi sinu tarkvara kasutab, siis laetakse automaatselt ka temale sama versiooniga kolmanda osapoole PHP pakid.

Laadi alla järgneva juhendi põhjal - 

1. [Installeeri Composer](https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx) oma arvutisse, juhendi leiad [siit](https://getcomposer.org/download/).
2. Laadi alla Silex, juhendi leiad [siit](http://silex.sensiolabs.org/download).

Tulemuseks on fail composer.json , mis sisaldab kõiki projekti sõltuvusi.

Samuti loodi kaust vendor/. See kaust sisaldab kolmandate osapoolte koodi. Kui annad korralduse git status märkad faili composer.lock, mis on automaatselt genereeritud fail, mida uuendatakse automaatselt iga kord kui Composeri abil midagi installeerid.

### Gitignore

Kausta vendor/ laetakse alla automaatselt kõik kolmandate osapoolte PHP pakid ja [me ei tohi seal koodi muuta](http://www.lonecpluspluscoder.com/2014/12/29/im-strongly-opposed-modifying-third-party-library-code/), seetõttu ei lisa me seda ka Git poolt jälgitavate failide nimekirja. Me ei tohiks kolmandate osapoolte pakkides koodis mingeid muudatusi teha väga lihtsalt põhjusel. Nimelt, kui me seda teeksime, siis ei saaks me enam nende poolt pakutavaid uuemaid versioone kasutada ning me peaksime ka nende kirjutatud koodi haldama hakkama. Kuna me seal koodi muutma ei hakka, pole vaja ka seda Giti abil jälgida. Seega tuleb Git’ile teada anda, millised faile ja kaustasid ta ignoreerima hakkab ([gitignore](https://help.github.com/articles/ignoring-files/)).

Meil on tarvis veel uut faili, kus defineerida nende failide asukohad, mida me ei soovi GitLabi üles laadida.

1. Loo uus fail .gitignore.
2. Defineeri kolmandate osapoolte kausta tee. (vendor/)(vaata selle peatüki lõpus olevat *commit*’ti.)
3. Sisesta git status
4. Juhul kui sa ikka näed programmi töölaual kausta **vendor**, siis anna korraldus git rm -rf --cached vendor/. Täpsema kirjelduse leiad järgnevalt [lingilt](https://help.github.com/articles/ignoring-files/#create-a-local-gitignore).

Nüüd peaks olema programmi töölaual (*staging area*):

gitignore

composer.json

composer.lock

Loo oma projekti teine *commit*. *Commit*’i tegemine Gitiga:

1. git add . (Lisab kõik uued failid programmi tööpiirkonda, võid teha ka ükshaaval - git add .gitignore composer.json composer.lock)
2. git commit -m “Adding Silex to project by using composer”
3. git push origin master (valikuline - see käsk lükkab muudatused repositooriumisse, juhul kui sa seda hetkel teha ei soovi, siis seda käsku kasutama ei pea).



Märkus: Juhul kui keegi kloonib sinu GitLabi repositooriumi ja soovib koodi käivitada, siis peab ta installeerima Composer-i oma arvutisse, korraldus “composer install” projekti kaustas, sel juhul on ka vendor-kaust üles laetud ja töötav. Kui ta jooksutab korralduse “composer install” oma projekti kaustas, siis laetakse alla **vendor**i kausta sisu.

(Vaata, milline on vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/37adf88875f75af62605d59548f5407dd0d2e221) GitLabis)

Kuna võtsime kasutusele kolmandate osapoolte koodi ja see mõjutab meie rakenduse installeerimist, siis peame selle dokumenteerima failis README.md.

(Vaata, milline on vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/8a10c7754078fd52c82988493df8a2ae96e20423) GitLabis)

Kokkuvõtteks – peaks nüüd olema selge, kuidas *commit*e ehk sissekandeid teha ja neid GitLabi üles laadida.

## PHP sisseehitatud veebiserveri käivitamine ja serveris PHP-faili käivitamine

Veebiserverina kasutame PHP sisemist veebiserverit. Eelduseks on, et oled installeerinud PHP oma arvutisse.

Selleks on meil tarvis PHP-fail veebiserveris käivitada. 

1. Loo uus kaust nimega **web**. Sellesse kausta tuleb luua [front-controller](https://en.wikipedia.org/wiki/Front_controller).

mkdir web

1. Front-controller-i loomiseks sisesta korraldus touch index.php
2. Lisa mõni koodirida, et testida, kas veebiserver töötab. Lülita sisse silumine (debug) edasiseks arendustööks. [Käivita Silex](http://silex.sensiolabs.org/doc/2.0/usage.html#bootstrap) oma *front-controller*is.
3. Seejärel defineeri esimene marsruut ([route](http://silex.sensiolabs.org/doc/2.0/usage.html#routing)) testimiseks.

$app**->**get('/hello', function() {

**return** 'Hello world!';

});

1. Testi. Anna oma projekti kaustas korraldus ([Kuidas käivitada PHP sisse-ehitatud veebiserver käsurealt](http://php.net/manual/en/features.commandline.webserver.php)):

php -S localhost:8000 -t web

1. Mine brauserisse ja ava URL: <http://localhost:8000/hello>

Juhul kui näed lehekülge tekstiga “Hello world!”, siis server töötab!

(Vaata seniste muudatustega vastavat [*commit*i](https://gitlab.com/i-sepp/bcs-koolitus/commit/7c0ade69355ac1197362145b935169e72550e0ec) Gitlabis.)

Märkus: Nüüdsest alates, kui käsurealt käivitatudveebiserver on kasutusel, siis kõik PHP vead kuvatakse käsureal. Juhul kui kood ei tööta, siis käsurida on esimene koht, kuhu vaadata, et koodis viga üles leida.

## Esimese HTML-malli loomine

### Twig kasutamine

Hea tava on kasutada eraldi seisvate failidena HTML-faile, kus asub malliloogika, ja PHP-faile, kuhu koondatakse äriloogika ja andmebaasiga suhtlemine.

Seetõttu võtame kasutusele mallimootori [Twig](http://twig.sensiolabs.org/). Twig on rakendus, mis töötab käsikäes Silexiga (Symfony) ([Twig'i kasutamine Silex'iga](http://silex.sensiolabs.org/doc/master/providers/twig.html)).

Twigi kasutamisel eraldatakse HTML-dokument PHP-koodist. Twig on nagu vaherakendus PHP ja HTMLi vahel. Twig on väga kasulik tööriist, mis viib su HTMLi uuele tasemele. Põhitõdesid ära õppida on üpris lihtne. Kui otsustad kasutada oma projektis Twigi ning õpid ära Twigi pisut keerukamad featuurid, siis on neist palju kasu, ning need muudavad su elu programmeerimismaailmas palju lihtsamaks. Sul on võimalik luua Symfony/Silex-i raamistikus PHP funktsioone ning kasutada neid oma Twigi mallides ilma, et peaksid oma HTML faile üle ujutama mitmete pikkade PHP-meetoditega. Loodetavasti annab see õppematerjal ka esimese ülevaate Twigi kasutamisest. See on lihtne, sest see käitub nagu PHP, aga süntaks on veidi erinev.

1. Laadi alla Twig, ([juhendi leiad siit](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#registering)):

composer require twig/twig

1. Peame Twigi paki kasutamise defineerima *front-controller*-is:

$app**->**register(**new** Silex\Provider\TwigServiceProvider(), **[**

'twig.path' **=>** \_\_DIR\_\_**.**'/../views',

]);

Siin me ka defineerime, et kaustas nimega “views” hoitakse meie html.twig faile. „Views“ on suvaline nimi, mille võid ise vabalt valida.

1. Loo kaust mallide jaoks - “**views**”.
2. Sellesse kausta loo esimene HTML fail, nimeks “**base.html.twig**”.

Vajalik on twig-laiend, et saaksime faili kirjutada kasutades *Twig*-i keelt. See fail sisaldab meie rakenduse põhimalli ja stiili, mida kasutatakse rakenduse kõikidel lehekülgedel.

1. Sellese faili kirjutame oma HTML-faili põhja, mis sisaldab kogu meie lehe põhilisi HTML-märgendeid (vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/e9be656252ab10102fa1911a61427ff84ffc3a1b)).
2. Nüüd jääb üle veel [kasutada seda kontrolleris](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#usage) - index.php. Marsruudi (route) nimetus tuleks muuta arusaadavamaks, näiteks võiks nimi olla “**bookings/create**”.

Seega tuleks teha muudatus index.php-failis asendades /hello route järgnevalt:

$app**->**get('/bookings/create', **function** () **use** ($app) {

**return** $app['twig']**->**render('base.html.twig');

});

1. Testi tulemust veebibrauseris, uus URL: [http://localhost:8000/bookings/create](http://localhost:8080/bookings/create)
2. Dokumenteerimine. Kuna võtsime kasutusele uue programmi, siis tuleks see kirjutada ka faili README.md (vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/b9e70d04eb5f56a7065b52a7398b728e35f56055)).

## Põhistiili lisamine ja HTML

Et rakendus näeks kena välja, peame kasutama stiili, selleks lisame oma rakendusse stiilimiseks vajalikku CSS-i ja HTML-i.

Meil on vaja luua kaust, kus hakkame hoidma CSS-i faile. CSSi ja Javascripti faile tavaliselt minimeeritakse. Meie seda selles projektis tegema ei hakka, aga hea tavana seda suurtes projektides tehakse eraldi töövahenditega. Tavaliselt on projektis kaks kohta, kus hoitakse Javascripti ja CSSi faile. Esimene asukoht on kaust, kuhu sa faile lood ja oma koodi kirjutad. Teine kaust on asukoht, kus asuvad failid, mida veebilehel reaalselt lehe kuvamisel kasutatakse. Teine kaust genereeritakse automaatselt esimese kausta põhjal. Teise kausta sisu on minimeeritud ja failid on ka tihti üheks suureks failiks kokku pandud. Kõige selle põhjus on lihtne - kuna suured projektid kasutavad palju ressursse, siis minimeeritakse koodi, et lehe laadimine oleks kiirem. Esimese rakenduse kirjutamisel ei hakka me lisatööriistu ning lisakonfiguratsiooni kõige selle jaoks kasutama ning loome oma CSS’i kausta otse web/ folderi sisse.

Kuna selle projekti eelduseks on see, et õppijal on olemas juba põhiteadmised HTML-ist ja CSS-ist, siis konkreetselt CSS’i kirjutamisel me rohkem ei peatu. Võid tutvuda autori loodud stiiliga või luua omaenda stiili. (Näite leiad [siit](https://gitlab.com/i-sepp/bcs-koolitus/commit/00e5b1d0c5790ac9e873e156197c27dacf7c13ea)).

## Esmakordne Twig kasutamine, jagades lehekülje komponentideks

Twigi abil on võimalik eraldada lehe elemendid erinevateks komponentideks. Tulemusena hoitakse erinevad komponendid erinevates failides. Sellise hea tava järgimiseks on mitu põhjust:

1. Kood on korduvalt kasutatav.
2. Lihtne lugeda
3. Iga komponent on eraldi, neid on lihtne muuta ja refaktoreerida ilma teisi komponente rikkumata.

Selleks tuleks kasutada Twigi funktsionaalust nimega [include](http://twig.sensiolabs.org/doc/2.x/templates.html#including-other-templates).

Eralda HTML, CSS päised ja jalused (*header and the footer*) eraldi failidesse.

Tulemuseks peab olema 4 uut faili: \_header.html.twig, \_footer.html.twig, header.css, footer.css.

NB! Ära unusta komponente dokumenteerimast uute failide alguses

(Vaata seniste muudatustega vastavat[*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/6af096e920e1b0ffac4f50fc9e98365de2bd4747)’i Gitlabis.)

## Andmebaasi loomine kasutades Doctrine DBAL

[Doctrine DBAL](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/) on rakenduskiht sinu rakenduse ja andmebaasi vahel ([database abstraction layer](https://en.wikipedia.org/wiki/Database_abstraction_layer)), mis võimaldab andmebaasi tabeli struktuuri haldamist.

Me kasutame seda andmebaasi loomiseks ning rakenduse suhtlemiseks andmebaasiga.

Andmebaas on vajalik, et salvestada vormidesse sisestatud andmed. Andmebaasina kasutame [SQLite](https://www.sqlite.org/about.html).

Selleks tuleb teha järgnevat:

1. Esmalt tuleb alla laadida Doctrine/DBAL kasutades Composerit ([Doctrini kasutamine Silexis](https://silex.symfony.com/doc/2.0/providers/doctrine.html)).

composer require "doctrine/dbal:~2.2"

1. Installeeri SQLite oma arvutisse. Näites on kasutatud Ubuntu 16.04 versiooni, koos PHP7 versiooniga, sellisel juhul oleks kasutatav korraldus järgmine:

sudo apt-get install php7.0-sqlite3

Märkus: Korraldused sõltuvad kasutavast operatsioonisüsteemist ja PHP versioonist. Seega abi tuleks abi otsida internetist – kuidas installeerida SQLite oma arvutisse.

Pärast seda tuleb käivitada uuesti PHP sisse-ehitatud veebiserver.

1. Dokumenteerimine. Kindlasti tuleb uuendada faili README.md, kuna me kasutame kolmanda osapoole tarkvara nimega Doctrine DBAL. Samuti tuleb igale arvutile, mis rakendust kasutab, installeerida SQLite (vaata, kuidas seda kirjeldab vastav[*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/525df3931d0bad6bd8b5efe8398ba8cab60f6a79)).
2. Lõpuks tuleb uus teenus registreerida *front-controlleris*.

Esmalt tuleb luua uus kaust, kus hoitakse andmebaasi faile, kausta nimeks paneme ***database****.*

([Juhendi leiad siit.](http://silex.sensiolabs.org/doc/master/providers/doctrine.html#registering) [Rohkem informatsiooni Doctrine konfigureerimisest](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html).)

$app**->**register(**new** Silex\Provider\DoctrineServiceProvider(), **[**

'db.options' **=>** **[**

'driver' **=>** 'pdo\_sqlite',

'path' **=>** \_\_DIR\_\_**.**'/../database/app.db',

],

]);

db.options – defineerimine andmebaasi tüübiks (muutuja nimega [driver](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#driver)) 'pdo\_sqlite'.

Seejärel defineerime andmebaasi tee 'path'.

(SQLite konfigureerimisjuhendi kasutades SQLite leiad [siit](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#pdo-sqlite).)

Märkus: Juhul kui rakenduse kood tööle ei lähe, siis kui käsureal on veateade, et „*a driver is missing*“, siis see tähendab, et eelmine samm (SQLite installeerimine) ebaõnnestus.

1. Andmebaasi faile pole tarvis GitLabi üles laadida. See genereeritakse arenduskeskkonnas automaatselt. Seega peame lisama andmebaasi kausta asukoha ka gitignore-faili.
2. Koostada MySQL korraldus, et luua andmebaasi broneeringute tabel.

MySQL’s näeks see välja selline:

"CREATE TABLE bookings (

id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

firstName VARCHAR(40) NOT NULL,

lastName VARCHAR(40) NOT NULL,

phone VARCHAR(10) NOT NULL,

email VARCHAR(20) DEFAULT NULL,

birthday DATE NOT NULL,

startDate DATE NOT NULL,

endDate DATE NOT NULL,

arrivalTime TIME DEFAULT NULL,

additionalInformation TEXT,

nrOfPeople INT NOT NULL,

payingMethod VARCHAR(10) NOT NULL

);"

Kasutame DBAL Connection.php [executeQuery](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/data-retrieval-and-manipulation.html#executequery) funktsiooni, et täita eelnev MySQL-lause:

$app['db']**->**executeQuery("MySQL code here");

1. Kuna meil on tarvis luua andmebaasitabel, siis see luuakse vaid juhul, kui selline tabel puudub. Seega tuleb lisada IF-lause. Kasutame [SchemaManager](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/schema-manager.html), et saada teada, kas tabel eksisteerib või mitte.

**if** (**!**$app['db']**->**getSchemaManager()**->**tablesExist('bookings')) {

$app['db']**->**executeQuery("MYSQL code here");

}

1. Värskenda lehekülg [http://localhost:8000/bookings/create](http://localhost:8080/bookings/create).

Tulemuseks peaks loodama andmebaasi tabel nimega “**bookings**”.

Esmaseks testimiseks on võimalik alati kasutada PHP funktsiooni [var\_dump](http://php.net/manual/en/function.var-dump.php), et näha, kas IF-lause töötab. *Var\_dump’i* kasutamist pole tarvis *commit*’ida, sest seda kasutatakse vaid esmaseks visuaalseks testimiseks (loe lisaks PHP silumise peatükki).

Märkus: Saad alati andmebaasi tabelit uuesti luua, kui kustutad andmebaasi kaustas **database** faili **app.db**.

1. Tabeli loomist saad kontrollida, kas database-kaust sisaldab faili app.db.

(Vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/942eb77a32fb499c3cd487a3b43232b303433881)).

## Composer - PSR-4 Autoloading

Nagu näha, muutub *front-controller* aina suuremaks ja suuremaks. Me pole oma rakendusele isegi veel vormi lisanud, aga oleme defineerinud mitmeid uusi teenuseid ja funktsioone, ning neid lisandub veel edaspidi.

Kui kogu meie PHP kood asub *front-controller*-is, võib see osutada probleemiks. Kujutle, et sul on veebirakendus, mis koosneb 20st erinevast veebilehest. Võib-olla lisandub andmebaasi veel uusi tabeleid ja funktsionaalsus läheb keerulisemaks. Väga mittemõistlik oleks kirjutada kogu rakenduse kood ühte faili. Sellist faili on raske lugeda, edasi arendada ja automaattestide kasutamine on praktiliselt võimatu.

Seetõttu peame oma koodi jaotama loogilisteks osadeks ja ka struktureerima oma projekti kausta loogiliselt erinevate failide ja meetoditena. Seega alustame objektorienteeritud programmeerimisega.

*Front-controller* on alati see fail, mis veebirakenduse käivitamisel esimesena välja kutsutakse, mis omakorda käivitab koodi, mis asub teistes failides.

Seega peame osa oma koodi panema eraldi failidesse.

Selleks kasutame [PSR-4](https://getcomposer.org/doc/04-schema.md#psr-4) automaatset laadimist (*autoloading*), et laadida uued failid *front-controller*-isse. Composer kasutab täpselt sama tehnikat ka meie kolmanda osapoolte pakkide rakendusse sisse laadimiseks ([lisainfo PSR-4 automaatsest laadimisest](https://seld.be/notes/psr-4-autoloading-support-in-composer)).

1. Esmalt tuleb luua uus kaust, kus hakkame hoidma oma uusi PHP-faile, kausta nimi on **src** (lähtekood ehk *source*).
2. Seejärel peame defineerima selle kausta ka Composerile automaatseks laadimiseks. Ava fail composer.json ja lisa kausta nimi. Näites on nimeruum **BookingApp**. Soovitavalt tuleks tutvuda PHP nimeruumi kasutamisega (vaata lisaks [videot](https://knpuniversity.com/screencast/php-namespaces-in-120-seconds/namespaces)).

"autoload": {

"psr-4": { "BookingApp\\": "src" }

}

Seega nüüd võime kutsuda välja **src**-kausta faile kasutades nimeruumi **BookingApp**.

Composer tuleb käivitada, et kõik muudatused saaksid rakendatud.

composer install

1. Järgmisena loome kausta **src** faili nimega **Application.php**. Loodud fail sisaldab uut klassi ([loe rohkem PHP klasside kohta](http://php.net/manual/en/language.oop5.basic.php)) nimega **Application**.
2. Kuna tegemist on PHP-failiga, siis fail algab PHP märgendiga ([loe rohkem, kuidas alustada PHP faili](http://php.net/manual/en/language.basic-syntax.phptags.php)):

**<?php**

1. Järgmisena peame defineerima nimeruumi ([loe rohkem PHP nimeruumide kohta](http://php.net/manual/en/language.namespaces.rationale.php)), kus see fail asub.

namespace BookingApp;

1. Peame defineerima klassi nime, mis peab olema sama, mis faili nimi.

class Application

{

}

1. Kuna sellel klassil on Silex\Application funktsionaalsus, siis loodud klass on selle laiendus ([loe rohkem meetodi extend kohta](http://php.net/manual/en/reflection.extending.php)). Kuid esmalt peame seda klassi kasutama ([loe rohkem, kuidas kasutada teisi PHP klasse](http://php.net/manual/en/language.namespaces.importing.php)) ja andma talle uue nime meetodiga as **SilexApplication**, sest meil ei saa olla ühe klassi seest kahte erinevat klassi kahe nimega.

**use** Silex\Application **as** SilexApplication;

**class** **Application** **extends** SilexApplication

1. Toome koodi index.php-failist Application.php-faili.

Võtame index.php-failist esimese rea ja redefineerime selle funktsiooni Application.php-failis:

$app **=** **new** Silex\Application();

Selle lausega kutsutakse välja PHP-fail Silex\Application.php. Kui avad selle faili, siis leiad sealt meetodi [\_\_construct method](http://php.net/manual/en/language.oop5.decon.php), mis käivitub iga kord, kui Silex\Application.php fail välja kutsutakse. \_\_construct-meetodis sisalduv kood kutsub välja kõik rakenduse käivitamiseks vajalikud meetodid.

Kui laiendame seda faili, siis peame meetodi \_\_construct üle kirjutama. Failis index.php, kui kutsume välja faili Application.php, siis kutsutakse välja ka meetod \_\_construct method, millesse saame lisada uusi meetodeid.

Kuna kirjutame selle üle, siis parameetrid peavad olema samad, mis meetodil \_\_construct failis Silex\Application.php. Lisaks peame kutsuma välja ülakonstruktori ([loe rohkem, kuidas kutsuda välja laiendatava PHP-klassi meetodeid](http://php.net/manual/kr/keyword.parent.php)). Seega peame lisama esimeseks reaks \_construct-meetodisse järgneva koodi:

**public** **function** **\_\_construct**(**array** $values **=** **[]**) {

**parent::**\_\_construct($values);

}

Märkus: Paremaks arusaamiseks tuleks läbi lugeda selle peatüki lingid \_\_construct ja parent, milles on olemas lihtsad näited sama funktsionaalsusega.

1. Nüüd saame tuua index.php-failist koodi Application.php-faili. Kood üleviimiseks: $app konfigureerimissätted, uue teenuse registreerimine, andmebaasitabeli loomine ja asukoha defineerimine. [Klassis](http://php.net/manual/en/language.oop5.basic.php)  Application tuleb teha järgnevad muudatused:
   1. Muutuja *$app on nüüd $this*
   2. Faili algusesse tuleb märkida kõik kasutavad nimeruumid:

**use** Silex\Provider\DoctrineServiceProvider;

**use** Silex\Provider\TwigServiceProvider;

[ … ]

$this['debug'] **=** **true**;

$this**->**register(**new** TwigServiceProvider(), **array**(

'twig.path' **=>** \_\_DIR\_\_**.**'/../views',

]);

$this**->**register(**new** DoctrineServiceProvider(), [

'db.options' **=>** **[**

'driver' **=>** 'pdo\_sqlite',

'path' **=>** \_\_DIR\_\_**.**'/../database/app.db',

],

]);

*/\* Creating a table if it doesn't exist yet \*/*

**if** (**!**$this['db']**->**getSchemaManager()**->**tablesExist('bookings')) {

$this['db']**->**executeQuery("...MySQL..");

}

$this**->**get('/bookings/create', function() {

**return** $this['twig']**->**render('base.html.twig');

});

1. Failis index.php kutsume välja faili Application.php kasutades nimeruumi.

**<?php**

**use** BookingApp\Application;

**require\_once** \_\_DIR\_\_**.**'/../vendor/autoload.php';

$app **=** **new** Application();

$app**->**run();

1. Juhul kui kõik kenasti töötab, siis luuakse uus klass.

(Vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/77494941649bd5a49f380eb4434ae85589745f7b)).

## Veel meetodeid klassis Application.php

Nüüd võib tekkida küsimus, et milleks oli seda vaja, kui vaid laiendasime meetodit \_\_construct failis Application.php.

Tegelikult loome uued meetodid, et eraldada üksteisest loogilised programmi osad ja loome vastavalt uued klassid.

Meie koodis on kolm loogilist osa:

1. Teenuste registreerimine.
2. Andmebaasi tabeli loomine.
3. Asukoha defineerimine.

Vastavalt sellele, loome kolm privaatset (private) funktsiooni:

**private** **function** **configureServices**()

**private** **function** **createDBTables**()

**private** **function** **configureControllers**()

Private-määrangut kasutame seetõttu, et funktsiooni on kasutusel klassi sees ([Loe rohkem PHP meetodite nähtavussätetest (visibility)](http://php.net/manual/en/language.oop5.visibility.php)).

Märkus: Funktsioonide nimed on vabalt valitavad loogilised nimed. Nimed peaksid olema valitud selliselt, et need oleks mõistetavad ka teistele arendajatele. Funktsiooni nimi peab peegeldama tema otstarvet. (“Programs should be written for people to read, and only incidentally for machines to executes” – "Structure and Interpretation of Computer Programs" by Abelson and Sussman).



Nüüd jääb üle kirjutada veel funktsioonid ja kutsuda funktsioonid välja meetodiga \_\_construct.

**public** **function** **\_\_construct**(**array** $values **=** **[]**)) {

**parent::**\_\_construct($values);

$this**->**configureServices();

$this**->**createDBTables();

$this**->**configureControllers();

}

(Vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/70d38e52c45593bab8aa5410933e8e56831aacf0))

## Vormi loomine



Järgmisena tuleb luua rakendusele vorm. Selleks kasutame komponenti [FormServiceProvider](http://silex.sensiolabs.org/doc/2.0/providers/form.html).

1. Esmalt tuleb vormi komponendid [Composeri abil alla laadida](http://silex.sensiolabs.org/doc/2.0/providers/form.html#registering).

composer require symfony/form

composer require symfony/twig-bridge symfony/config symfony/translation

(Vastav[*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/1f083b5f07442344c86dcbace3de4475b510304b))

1. Registreerime uued teenused failis **Application.php**.

**use** Silex\Provider\FormServiceProvider;

**use** Silex\Provider\LocaleServiceProvider;

**use** Silex\Provider\TranslationServiceProvider;

[ ... ]

$this**->**register(**new** FormServiceProvider());

$this**->**register(**new** LocaleServiceProvider());

$this**->**register(**new** TranslationServiceProvider(), **[**

'translator.domains' **=>** **[]**,

]);

Märkus: Me küll ei kasuta hetkel tõlkimiseks mõeldud pakki tõlkimise eesmärgil, küll aga kasutame komponenti twig-bridge, mis vajab selleks, et ta töötaks, tõlkimise komponenti nimega [TranslationServiceProvider](http://silex.sensiolabs.org/doc/2.0/providers/translation.html).

1. Loo vormi objekt, milles defineerimine kõik väljad ([loe rohkem vormi objekti loomisest](https://symfony.com/doc/current/forms.html#building-the-form)):

$form **=** $this**->**formFactory**->**createBuilder(FormType**::**class)

**->**add('firstName', [TextType](https://symfony.com/doc/current/reference/forms/types/text.html)**::**class, **[**'required' **=>** **true**])

**->**add('lastName', TextType**::**class, **[**'required' **=>** **true**])

**->**add('phone', TextType**::**class, **[**'required' **=>** **true**])

**->**add('email', TextType**::**class,['required' **=>** **false**])

**->**add('birthday', [DateType](https://symfony.com/doc/current/reference/forms/types/date.html)**::**class, **[**

'required' **=>** **true**,

'widget' **=>** 'single\_text',

'format' **=>** 'dd.MM.yyyy',

])

**->**add('startDate', DateType**::**class, **[**

'required' **=>** **true**,

'widget' **=>** 'single\_text',

'format' **=>** 'dd.MM.yyyy',

])

**->**add('endDate', DateType**::**class, **[**

'required' **=>** **true**,

'widget' **=>** 'single\_text',

'format' **=>** 'dd.MM.yyyy',

])

**->**add('arrivalTime', [TimeType](https://symfony.com/doc/current/reference/forms/types/time.html)**::**class, **[**'required' **=>** **true**])

**->**add('nrOfPeople', [IntegerType](https://symfony.com/doc/current/reference/forms/types/integer.html)**::**class, **[**'required' **=>** **true**])

**->**add('payingMethod', [ChoiceType](https://symfony.com/doc/current/reference/forms/types/choice.html)**::**class, **[**

'choices' **=>** ['cash' **=>** 'cash', 'transfer' **=>** 'transfer'],

'required' **=>** **true**

])

**->**add('additionalInformation', [TextareaType](https://symfony.com/doc/current/reference/forms/types/textarea.html)**::**class, **[**

'required' **=>** **false**

])

**->**add('submit', [SubmitType](https://symfony.com/doc/current/reference/forms/types/submit.html)**::**class,['label' **=>** 'Send booking'])

**->**getForm()

;

Komponendi Form factory puhul on võimalik kasutada erinevaid vormide tüüpe ([siin on dokumentatsioon kõikide vormitüüpide kohta](https://symfony.com/doc/current/reference/forms/types.html)). Igal tüübil on oma sätted. Näiteks [DateType](https://symfony.com/doc/current/reference/forms/types/date.html) omab sätet “[format](https://symfony.com/doc/current/reference/forms/types/date.html#format)”, et oleks võimalik kasutada erinevaid kuupäeva vormindamise variante. Et paremini aru saada FormType võimalustest tuleks kindlasti täiendavalt uurida dokumentatsiooni.

1. Kõigi [kasutavate](http://php.net/manual/en/language.namespaces.importing.php)  vormi tüüpide nimeruumid tuleb defineerida faili alguses.

**use** Symfony\Component\Form\Extension\Core\Type\ChoiceType;

**use** Symfony\Component\Form\Extension\Core\Type\DateType;

**[ … etc …]**

1. Nüüd peame oma vormi kasutama mallil. Selleks peame looma uue malli nimega **form.html.twig**. Kasutame Twig pärimise loogikat ([loe rohkem malli pärimise kohta](http://twig.sensiolabs.org/doc/2.x/templates.html#template-inheritance)), et loodaval mallil oleksid sama HTML ja CSS, mille juba kirjutasime faili **base.html.twig**.

Selleks loome ploki, mida saame kasutada ka vormil form.html.twig.

Selleks vajame kolme plokki: **stylesheets, content** ja **javascripts**.

**{%** **block** stylesheets **%}**

<link rel="stylesheet" type="text/css" media="screen" href="/css/normalize.css">

<link rel="stylesheet" type="text/css" media="screen" href="/css/base.css">

<link rel="stylesheet" type="text/css" media="screen" href="/css/header.css">

<link rel="stylesheet" type="text/css" media="screen" href="/css/footer.css">

**{%** **endblock** **%}**

[ … ]

<div class="content">{% block content %}{% endblock %}</div>

[ … ]

{% block javascripts %}{% endblock %}

1. Vormil **form.html.twig**, defineerime, et uus mall on base.html.twig laiendus ([loe rohkem laiendamise kohta - meetod extends](http://twig.sensiolabs.org/doc/2.x/tags/extends.html)).

**{%** **extends** 'base.html.twig' **%}**

1. Võime content-plokki lisada ka veidi sisu. Kuna soovime vormi renderdada st andmed visualiseeritakse kuvamiseks sobivasse vormingusse ([loe rohkem vormi kuvamise kohta HTMLis](https://symfony.com/doc/current/forms.html#rendering-the-form)), siis selleks on võimalik kasutada Twigi valmis funktsionaalsust. Seda võib teha ka väljade kaupa ehk kirjutada koodi iga väli eraldi ([loe selle kohta siit](https://symfony.com/doc/current/form/rendering.html#rendering-each-field-by-hand)), aga meie seda varianti hetkel ei kasuta ning kasutame kolme koodirida kogu vormi korraga kuvamiseks.

**{%** **block** content **%}**

<h2>Some nice heading here!</h2>

**{{** form\_start(form) }}

**{{** form\_widget(form) **}}**

**{{** form\_end(form) **}}**

**{%** **endblock** **%}**

1. Nüüd kasutame malli kontrolleris (Vastav [*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/4340c044db42aa036be177e3f89dfab2aecc3ba7))

**return** $this['twig']**->**render('form.html.twig', **[**

'form' **=>** $form**->**createView(),

]);

1. Kui värskendad rakenduse lehekülge, siis peaksid nägema oma vormi. Vorm on üsna inetu, aga vähemalt töötab.

Kuidas saab olla nii, et ainult 3 HML-koodirida, suudavad renderdada nii palju HTML’i? Võib tekkida küsimus, et kuidas see nii töötab ehk kuidas kolme koodirea kirjutamine suudab genereerida kogu formi HTML-i?

Vaikimisi kasutatud Twig vormi teema asub kaustas - /vendor/symfony/twig-bridge/Resources/views/form/form\_div\_layout.html.twig

Seal on näha HTML ja Twigi segu, mis loob korduvalt kasutatava HTML-koodi loodud vormi elementidele.

Twig vormide teemade ([rohkem informatsiooni vormi teemade kohta](https://symfony.com/doc/current/form/form_customization.html#what-are-form-themes)) kasutamine teeb koodi kirjutamise kiiremaks. Sa ei pea iga uue vormi jaoks kirjutama suure hulga HTMl-i koodi, küll aga saad alati vajadusel vormi erinevaid elemente üle kirjutada. ([Vormi elementide üle kirjutamise kohta loe rohkem siit.](https://symfony.com/doc/current/form/form_customization.html)) Meie kasutame elementide ülekirjutamise võimalust hetkel vaid nii palju, et meie vorm oleks kenam.

1. Vormi stiilimiseks peame lisama tegema elemendile <form> oma klassi. >Seda on üpris lihtne teha, loe lisaks selle kohta järgnevalt [lingilt](https://symfony.com/doc/current/form/rendering.html#rendering-each-field-by-hand).

**{{** form\_start(form, {attr: {class: 'booking-form'**}}**) }}

1. Muudame pisut olemasolevat teemat ([loe oma teema kirjutamise kohta samasse faili](https://symfony.com/doc/current/form/form_customization.html#method-1-inside-the-same-template-as-the-form)). Igal vormi väli võiks olla oma HTML elemendi ja klassi sees, selleks kirjutame üle Twigi ploki form\_row ja lisame omad täiendused.

Vormi iga väli võiks olla eraldi klassina. Seega kirjutame üle rea form\_row Twig block ja lisame oma täiendused. Mõnikord on seda tarvis teha ainult teatud väljade puhul ([kuidas kirjutada teemat ühe konkreetse välja kohta](https://symfony.com/doc/current/form/form_customization.html#how-to-customize-an-individual-field)), aga praegu loome stiili **form\_row** kogu rakenduse tarvis.

**{%** **block** form\_row **%}**

<div class="form-row">

**{{** form\_label(form) **}}**

**{{** form\_widget(form) **}}**

</div>

**{%** **endblock** **%}**

1. Loodud teema kasutamiseks peame selle defineerima ([loe rohkem teema defineerimise kohta](https://symfony.com/doc/current/form/form_customization.html#method-1-inside-the-same-template-as-the-form)) faili alguses.

**{%** form\_theme form \_self **%}**

1. Kõikidele lisatud klassidele tuleb kirjutada CSS laused CSS-i faili ja lisada uus CSS-i fail malli külge.

**{%** **block** stylesheets **%}**

**{{** parent() **}}**

<link rel="stylesheet" type="text/css" media="screen" href="/css/form.css">

**{%** **endblock** **%}**

Kuna soovime kasutada stiili, mis pärineb failist base.html.twig, siis kasutame Twigi pärimisfunktsiooni [parent](http://twig.sensiolabs.org/doc/2.x/functions/parent.html)().

1. Nüüd peaks vorm juba märksa kenam olema ([*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/60a306e2051c0b89c88ca4cc2d970fc89ce01259)).

## Kontrolleri koodi eraldamine eraldiseisvaks PHP klassiks

Failis **Application.php** oleks hea eraldada kontrolleri kood ülejäänud koodist. Koodi eraldamise tulemusena muutub kood lihtsamaks ja loetavamaks, mis omakorda tõstab koodi jätkusuutlikust ning on lihtsamini testitav.

1. Peame looma src-kausta uue kausta **Controllers**.

cd src

mkdir Controllers

1. Seejärel loome uue klassi nimega **CreateBookingController.php**

cd Controllers

touch CreateBookingController.php

1. PHP-fail algab alati alustava märgendiga (opening tag):

**<?php**

1. Seejärel nimeruum ([namespace](http://php.net/manual/en/language.namespaces.rationale.php)):

**namespace** BookingApp\Controllers;

1. Defineerime klassi

**class** **CreateBookingController**

{}

1. Nüüd peame läbi mõtlema, milliseid muutujaid klassi sees vajame. Kui vaatame Application.php-failis olevat funktsiooni, siis märkame, et seal on kaks muutujat väljastpoolt funktsiooni. Need on:

$this[form.factory']

$this['twig']

Ei ole mõtet edastada kogu muutujat $this. Anname edasi vaid selle osa, mida loodud uutes klassides vajame. Sellisel juhul on paremini mõistetav, milliseid muutujaid kasutame uue klassi sees. Jällegi muutub kood paremini testitavaks ja loetavamaks. Järgmisena peame uue klassi välja kutsuma failis Application.php

$this**->**match('/get', **new** CreateBookingController(

$this['form.factory'],

$this['twig']

));

NB! Ära unusta importida uut nimeruumi.

**use** BookingApp\Controllers\CreateBookingController;

1. Seejärel peame muutujad kätte saama CreateBookingController’s.

Selleks kasutame meetodit [\_\_construct](http://php.net/manual/en/language.oop5.decon.php):.

**public** **function** **\_\_construct**(FormFactory $formFactory \Twig\_Environment $twigEnv)

{

$this**->**formFactory **=** $formFactory;

$this**->**twigEnv **=** $twigEnv;

}

Kasutame tüübi deklaratsioone ([loe selle kohta rohkem siit](http://php.net/manual/en/functions.arguments.php#functions.arguments.type-declaration)) või nn tüübi-vihjeid (type-hints), et näidata muutujate sisu. Tüübi-vihjed tähendavad, et sa võid olla kindel, et sobivat edastamismeetodit on kasutatud. Tüübi vihjed garanteerivad selle, et muutujad, mis on edasi antud on just selle sisuga nagu sa eeldad. Selle variandi puhul saad selge veateate, kui püüad edastada teist tüüpi muutujat.

Kasutame PHP klassi-siseseid muutujaid ([klassi siseste muutujate kohta loe siit - ing. k - non-static properties](http://php.net/manual/en/language.oop5.properties.php)), et imporditud muutujad oleks kasutavad uue klassi kõikides meetodites.

Nüüd saame kasutada selle klassi muutujaid igal pool, sisestades:

$this**->**formFactory;

\Twig\_Environment on globaalne nimeruum ning seda me ei pea importima, aga me peame importima FormFactory nimeruumi, sest see pole globaalne.

**use** Symfony\Component\Form\FormFactory;

1. Nüüd kirjutame klassi sisse oma kontrolleri funktsionaalsuse. Kasutame PHP funktsiooni \_\_invoke, et meie funktsionaalsus oleks alati kohe välja kutsutud, kui seda klassi kasutatakse ([Kindlasti loe selle PHP funktsiooni kohta lisainfot PHP kodulehelt.](http://php.net/manual/en/language.oop5.magic.php#object.invoke))

**public** **function** **\_\_invoke**() {}

1. Seejärel vajame juba eelnevalt kirjutatud koodi vormi loomiseks ja malli renderdamiseks:

$form **=** $this**->**formFactory**->**createBuilder(FormType**::**class)

**->**add('firstName', TextType**::**class, ['required' **=>** **true**])

[ … ]

**->**getForm()

;

**return** $this**->**twigEnv**->**render('form.html.twig', ['form' **=>** $form**->**createView()]);

1. Ära unusta kustutada enam mittevajalikke nimeruumide importimine failist Application.php ja kirjuta need laused faili CreateBookingController.php. ([*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/c9004eb7d7e2cf9549a266cd04d3a0fcbb810ddc))

## Andmete salvestamine andmebaasi

Meil on olemas nüüd vorm, kuid see ei salvesta veel andmeid andmebaasi. 

1. Andmebaasi salvestamiseks tuleb alustuseks luua andmebaasiga ühendus. Selleks peame failis Application.php edastama muutaja, mis sisaldab endas andmebaasiga ühendamiseks vajalikku infot, oma äsja loodud uude kontrollerisse.

$this

**->**match('/bookings/create', **new** CreateBookingController(

$this['form.factory'],

$this['twig'],

$this['db']

))

**->**method('GET|POST')

;

Siiani oleme kasutanud ainult meetodit [GET](http://silex.sensiolabs.org/doc/2.0/usage.html#dynamic-routing), aga kuna peame vormile ja väljastama, siis peame lubama ka meetodit [POST](http://silex.sensiolabs.org/doc/2.0/usage.html#example-post-route). Kuna vajame mõlemat meetodit, siis kasutame meetodit [match](http://silex.sensiolabs.org/doc/2.0/usage.html#other-methods) ja defineerime lubatud meetodid funktsioonis **method()**.

1. Nüüd võtame CreateBookingControlleris vastu andmebaasi muutuja ning importime ka PHP klassi Connection, mis on selle andmebaasi muutuja tüübiks. Defineerime muutuja globaalseks kasutamiseks CreateBookingControlleri klassis.

**use** Doctrine\DBAL\Connection;

[ … ]

**public** **function** **\_\_construct**(

FormFactory $formFactory,

\Twig\_Environment $twigEnv,

Connection $dbConnection

) {

$this**->**formFactory **=** $formFactory;

$this**->**twigEnv **=** $twigEnv;

$this**->**dbConnection **=** $dbConnection;

}

1. Peame lisama võimaluse vormi andmete andmebaasi salvestamiseks ([loe andmete salvestamise kohta siit](http://silex.sensiolabs.org/doc/2.0/providers/form.html#usage)).

Kuna [Silexi raamistik kasutab Symfony](http://silex.sensiolabs.org/doc/2.0/internals.html#application) valmisehitatud HttpKernelInterface’i (mille kohta saab lugeda [siit](http://api.symfony.com/master/Symfony/Component/HttpKernel/HttpKernelInterface.html)), mis tegeleb HTTP-päringute vastuvõtmisega, siis kui selle Kerneli dokumentatsiooni lugeda, saame teada, et HTTP vaikimis päringu argumendiks on alati [Request](http://api.symfony.com/master/Symfony/Component/HttpFoundation/Request.html). Seega saame selle defineerida oma meetodi esimeseks argumendiks. See meetod peab alati tagastama õiget tüüpi vastuse ehk [Response](http://api.symfony.com/master/Symfony/Component/HttpFoundation/Response.html), mida meie meetod juba teeb. (Loe rohkemat tüübi Response kohta [siit](http://api.symfony.com/master/Symfony/Component/HttpFoundation/Response.html).)

**use** Symfony\Component\HttpFoundation\Request;

**public** **function** **\_\_invoke**(Request $request)

1. Seejärel võtame objektilt $request kõik vajalikud andmed, meetodi POST parameetrid.

$form**->**handleRequest($request);

1. Järgmisena tuleb kontrollida, kas vormi väärtused on õiget tüüpi, selleks kasutame vormi valideerimismeetodit (lisaks tuleks lugeda [artiklit](http://symfony.com/doc/current/forms.html#handling-form-submissions) - form submission).

**if** ($form**->**isValid()) {}

1. Kui andmed on tõesed, siis peame kasutama MySql, et saata andmed andmebaasi.

$data **=** $form**->**getData();

$st **=** $this**->**dbConnection**->**executeQuery(

"INSERT INTO bookings (

firstName, lastName, phone, email, birthday, startDate,

endDate, arrivalTime, nrOfPeople, payingMethod,

additionalInformation

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",

**[**

$data['firstName'],

$data['lastName'],

$data['phone'],

$data['email'],

$data['birthday']**->**format('Y-m-d H:i:s'),

$data['startDate']**->**format('Y-m-d H:i:s'),

$data['endDate']**->**format('Y-m-d H:i:s'),

$data['arrivalTime']**->**format('H:i:s'),

$data['nrOfPeople'],

$data['payingMethod'],

$data['additionalInformation']

]);

Märkus: Peame andmed ka õigesti vormindama, vastavalt andmebaasi väljade vormingule st mõned Date-tüübi andmed stringiks vms.

MySQL päring tuleb teha nii, et ennetada SQL väärkasutust ([SQL injection](https://en.wikipedia.org/wiki/SQL_injection)), mis on üks lihtsamaid viise veebilehe ründamiseks. Ennetavaks meetodiks on kasutada päringu loomisel parameetreid, nii nagu kirjeldatud järgnevas artiklis ([artikkel teemal SQL injection](http://www.acunetix.com/websitesecurity/sql-injection/)).

1. Et aru saada, kas andmete saatmine oli edukas, siis kõige lihtsam on värskendada rakenduse lehekülg.

**use** Symfony\Component\HttpFoundation\RedirectResponse;

[ … ]

**return** **new** RedirectResponse($request**->**getUri());

1. Seejärel tuleks testida andmete edastamist, kui pärast Submit-nupule vajutamist leht on värskendatud, siis salvestati andmed ka andmebaasi.



NB! Kui pärast Submit-nupule vajutamist näed ekraanilt puhast valget lehte, siis pead olema veendunud, et täitsid väljad korrektselt (näiteks kuupäev peab olema 12.12.1990 jne). Meie vorm ei oska veel andmete sobivust ja korrektsust kontrollida. Igal juhul ei oska vorm veel väljastada veateadet, et sisestus on sobimatu. Täiendava ülesandena soovitame kasutada andmete kontrollimiseks vormi valideerimist ([form validation](http://symfony.com/doc/current/validation.html)).

([*Commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/fb80d0a965b553f1b96e8ae9a808e76b29c2d876)).

## Andmebaasi sisestatud andmete vaatamine

Tore oleks teada saada, milliseid broneeringuid on tehtud. Selleks loome uue lehekülje, millelt saame vaadata kõiki tehtud broneeringuid.

1. Esmalt vajame uut asukohta ja kontrollerit. Uue asukoha saame defineerida failis Application.php. Asukoht saab nime “bookings” ja uus kontroller nimega ListBookingsController. Kuna nüüd võtame kasutusel uue twig-malli, et renderdada broneeringuid, siis vajame kontrolleris Twig\_Environment muutujaid. Lisaks peame tegema andmebaasi MySQL päringu, selleks vajame Connection muutujat.

**use** BookingApp\Controllers\ListBookingsController;

[ … ]

$this

**->**get('/bookings', **new** ListBookingsController(

$this['db'],

$this['twig']

))

;

1. Nüüd peame tegema uue kontrolleri klassi kausta **Controllers**.

Samamoodi nagu CreateBookingsController’i puhul, defineerime nimeruumi ([namespace](http://php.net/manual/en/language.namespaces.php)), impordime parameetrite tüüpide nimeruumid ja salvestame edastatud parameetrid klassi muutujatesse ([non-static properties](http://php.net/manual/en/language.oop5.properties.php)).

**<?php**

**namespace** BookingApp\Controllers;

**use** Doctrine\DBAL\Connection;

**use** Symfony\Component\HttpFoundation\Request;

**class** **ListBookingsController**

{

**public** **function** **\_\_construct**(

Connection $dbConnection, \Twig\_Environment $twigEnv

) {

$this**->**dbConnection **=** $dbConnection;

$this**->**twigEnv **=** $twigEnv;

}

}

1. Defineerime meetodi [\_\_invoke](http://php.net/manual/en/language.oop5.magic.php#object.invoke), kus realiseeritakse päring ja tulemused edastatakse uuele mallile.

**public** **function** **\_\_invoke**(Request $request) {}

1. Päringu tarvis kasutame kahte Doctrine/DBAL/Connection.php-meetodit: [executeQuery()](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/data-retrieval-and-manipulation.html#executequery) ja [fetchAll()](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/data-retrieval-and-manipulation.html#fetchall).

$query **=** $this**->**dbConnection**->**executeQuery("SELECT \* FROM bookings");

$result **=** $query**->**fetchAll();

1. Seejärel anname päringu tulemuse Twigi mallile, mis andmed HML-iga seob ning anname päringule vastuse HTML-failina.

**return** $this**->**twigEnv**->**render('list.html.twig', **[**'bookings' **=>** $result]);

1. Nagu näha, defineerisime uue malli, mida veel ei eksisteeri. Peame selle looma, kirjutama HTML-koodi ja kasutama CSS’i. Tuletan meelde, et kõik andmed, mida on tarvis andmebaasist kätte saada, asuvad muutujas “bookings” massiivina. Saad pöörduda iga tabeli välja poole kasutades välja nime, mis on deklareeritud failis Application.php meetodiga createDBTables(). Kuna broneeringud paiknevad massiivis, siis pead kasutama kordust [Twig For loop](http://twig.sensiolabs.org/doc/2.x/tags/for.html). Kui tabelist pole ühtegi broneeringut, siis ei ole broneeringu näitamiseks vajalikku HTML vaja näidata, selleks kasutame if-lauset ning massiivi kontrollmeetodit, mis tagastab, kas massiiv on tühi või mitte ([loe Twigi meetodi empty kohta siit](http://twig.sensiolabs.org/doc/2.x/tests/empty.html)). Jällegi võid laiendusena ([extend](http://twig.sensiolabs.org/doc/2.x/tags/extends.html)) kasutada põhikujundust ja –stiili failist base.html.twig.

**{%** **extends** 'base.html.twig' **%}**

**{%** **block** content **%}**

<h2>The list of bookings</h2>

**{%** **if** bookings **is** empty **%}**

<p>You don't have any bookings!</p>

**{%** **else** **%}**

<table class="booking-table">

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>Birthday</th>

<th>Arrival Time</th>

[ … ]

</tr>

</thead>

<tbody>

**{%** **for** row **in** bookings **%}**

<tr>

<td>**{{** row.firstName **}}**</td>

<td>**{{** row.lastName **}}**</td>

<td>**{{** row.birthday**|date**('d.m.Y') **}}**</td>

<td>**{{** row.arrivalTime**|date**('H:i') **}}**</td>

</tr>

**{%** **endfor** **%}**

</tbody>

</table>

**{%** **endif** **%}**

**{%** **endblock** **%}**

**{%** **block** stylesheets **%}**

**{{** parent() **}}**

<link rel="stylesheet" type="text/css" media="screen" href="/css/list.css">

**{%** **endblock** **%}**

1. Mine aadressile: [http://localhost:8000/bookings](http://localhost:8000/hello), seal peaks näha olema kõik salvestatud broneeringud.

([*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/114293498767a32b28999293e84de063ae186acb)).

## Menüü loomine

Ebamugav on liikuda lehtede vahel kasutades nende aadresse, tore oleks teha seda menüü abil.

Me saame Twigi abil genereerida valmis pikad URLid, kui kasutame meetodid [path](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#symfony-components-integration). Igale URLile saame anda oma nime, kui kasutame marsruudi defineerimisel meetodit [bind](http://silex.sensiolabs.org/doc/2.0/usage.html#named-routes).

1. Marsruudile nime andmine:

$this

**[ … ]**

**->**method('GET|POST')

**->**bind('booking\_form')

;

$this

**->**get('/bookings', **new** ListBookingsController(

$this['db'],

$this['twig']

))

**->**bind('booking\_list')

;

1. Seejärel peame menüü jaoks kasutama natuke HTML’i. Jällegi peaks uus komponent olema eraldatud. Selleks loome uue faili nimega **\_menu.html.twig** ja CSS jaoks faili **menu.css**. Lisame need faili base.html.twig.

<link rel="stylesheet" type="text/css" media="screen" href="/css/menu.css">

[ … ]

**{{** include('\_menu.html.twig') **}}**

1. Menüüfaili \_menu.html.twig lisame täiendava funktsionaalsuse. Kasutaja peab aru saama, milline menüüvalik tal parajasti avatud on. Selleks kasutame CSS-is värve (*color property*) ja eraldi HTML klassi. See menüülink, mis on parajasti avatud saab eraldi HTMl klassi, mis annab talle teise värvi. Kõigepealt on koodis vaja loogikat, mis õigele menüüelemendile õige klassi juurde lisaks. Selleks on meil vaja võrrelda hetkel avatud aadressi menüülingi aadressiga. Selles aitab Twigi muutuja nimega global.request, mis annab meile ligipääsu meetodile getRequestUri(). Kui avad faili **Request.php**, siis näed, et meetod on olemas klassis Request ja seda on Twig’il võimalik välja kutsuda. Kuna mul on selle meetodi tulemust vaja kasutada mitmes kohas, siis salvestan selle muutujasse, kasutades Twigi set meetodit. ([Loe rohkem Twigi meetodi set kohta](http://twig.sensiolabs.org/doc/2.x/tags/set.html)).

**{%** **set** currentUrl = global.request.getRequestUri() **%}**

1. Võrdlemiseks kasutan  [Twigi meetodit IF](http://twig.sensiolabs.org/doc/2.x/tags/if.html), ning täispika URLi genereerimiseks kasutan Twigi meetodit path, et genereerida varem defineeritud marsruudi nimest täispikk URL. Pika IF-i kasutamise asemel kasutan lühemat versiooni, mida nimetatakse inglise keeles [ternary operator](http://twig.sensiolabs.org/doc/2.x/templates.html#other-operators).

<a class="menu-link **{{** path('booking\_form') **==** currentUrl ? 'active' **}}**"

href="**{{** path('booking\_form') **}}**">

Book

</a>

<a class="menu-link **{{** path('booking\_list') **==** currentUrl ? 'active' **}}**"

href="**{{** path('booking\_list') **}}**">

See bookings

</a>

1. Paigutan selle HTML-koodi div-lausesse ja kirjutan pisut CSSi, et menüü oleks ilusam ([*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/b05b21bfaae2a46109008870c3c948628f22b279)).

## Sissejuhatus testimisse: PHPUnit

Kõige viimaks tuleb rakendusele lisada testimisvõimalus. Edasises arendustöös saad kirjutada teste nii palju kui tarvis.

Testimiseks kasutame PHP testimise raamistikku [PHPUnit](https://phpunit.de/).

1. Lae alla PHPUnit:

composer require --dev "phpunit/phpunit=5.5.\*"

1. Lae alla Silex [WebTestCase testing dependencies](http://silex.sensiolabs.org/doc/2.0/testing.html#webtestcase)

composer require --dev symfony/browser-kit symfony/css-selector

Märkus: käsu jooksutamisel kasutame lippu --dev, sest neid pakke kasutame vaid arendusefaasis koodi testimisel, aga kliendi serverites seda koodi osa kunagi ei kasutata. See annab meile võimaluse luua oma rakendusest kaks erinevat koopiat, millest üks sisaldab testimist ja teine mitte.

1. Konfigureerime ([Configuring](http://silex.sensiolabs.org/doc/2.0/testing.html#configuration)) komponendi PHPUnit, et see töötaks koos meie rakendusega:

Selleks vajad eraldi faili nimega “**phpunit.xml.dist**”

Selles failis tuleb määrata kaust, kus testid asuvad.

**<?xml version="1.0" encoding="UTF-8"?>**

<phpunit bootstrap="./vendor/autoload.php"

backupGlobals="false"

backupStaticAttributes="false"

colors="true"

convertErrorsToExceptions="true"

convertNoticesToExceptions="true"

convertWarningsToExceptions="true"

processIsolation="false"

stopOnFailure="false"

syntaxCheck="false"

>

<testsuites>

<testsuite name="BookingApp Test Suite">

<directory>./tests/</directory>

</testsuite>

</testsuites>

</phpunit>

1. Loo uus kaust nimega “**tests**”.
2. Loome lihtsa testi, et teha kindlaks, et lehekülg veebibrauseris kuvatakse.

Testfaili nimeks paneme AppTest.php ja see test on WebTestCase laiendus ([extend the WebTestCase](http://silex.sensiolabs.org/doc/2.0/testing.html#webtestcase)).

**<?php**

**use** Silex\WebTestCase;

**use** BookingApp\Application;

**class** **AppTest** **extends** WebTestCase

{}

1. Järgmisena tuleb täita korraldused [juhendist](http://silex.sensiolabs.org/doc/2.0/testing.html#webtestcase). Alustuseks, rakendame meetodit createApplication. Kuna oleme ise loonud faili Application.php, siis see meetod peab tagastama muutuja $app.

**use** BookingApp\Application;

[ … ]

**public** **function** **createApplication**()

{

**return** **new** Application();

}

1. Seejärel peame kirjutama testi. Põhimõtteliselt testime, kas broneeringute lehekülg töötab ja et tulemuseks ei anta mingeid veateateid.

**public** **function** **testIfBookingsPageRenders**()

{

$client **=** $this**->**createClient();

$crawler **=** $client**->**request('GET', '/bookings');

$this**->**assertTrue($client**->**getResponse()**->**isOk());

}

1. Käivitame testi järgmise käsuga:

vendor/bin/phpunit tests/AppTest.php

Kui see kuvatakse rohelisega, siis oli test edukalt läbitud. Eksperimenteerimiseks, et näha kuidas test läbi kukub, näiteks kustutad ära mõne semikooloni failis ListBookingsController.php

1. Dokumenteerimine: uuenda README.md-fail, lisa informatsiooni testimise kohta.

([*commit*](https://gitlab.com/i-sepp/bcs-koolitus/commit/7ab5ba8a72a178a0736d346764dca2919a52d6b2)).

See on põhimõtteliselt automaattestimine. Loomulikult saab konfigureerida nii, et testid käivitatakse alati kui lõpetad mõne programmiosa koodi kirjutamise. Hetkel pead testi käivitama ise käsurealt. Automaatestimine nõuab testide kirjutamist programmeerimisega paralleelselt. Iga kord kui saad mõne programmi valmis, siis tuleks kirjutada ka vastav test. See on äri poolelt vaadates märksa ajamahukam, sest pead otsustama, millised programmid vajavad automaattestimist ja millised mitte. Tavaliselt vajavad testimist need koodi osad, mis on delikaatsed, keerukad. Tavaliselt vajavad pigem testimist rakenduse need osa, mis on keerukad ja võivad kergesti probleeme tekitada või, mis mõjutavad kriitilisel moel rakenduse kasutajaid. Seega on oluline, et kood oleks jaotatud väiksemateks osadeks – erinevatesse klassidesse, et oleks võimalik testida üht klassi või isegi üht meetodit korraga.

Kui soovid saada rohkem infot PHPUnit’st ja selle raamistiku abil testide kirjutamisest, siis on tuleks külastada ja lugeda veebist [PHPUnit website](https://phpunit.de/manual/current/en/writing-tests-for-phpunit.html).

## Kuidas käivitada GitLab’ist alla laetud valmis rakendust?

Valmis rakenduse leiad [siit](https://gitlab.com/i-sepp/bcs-koolitus/), võid selle kloonida ja kasutada oma arvutis (<https://gitlab.com/i-sepp/bcs-koolitus/>)

Järgnevalt on kokkuvõtlik juhend, mida pead tegema, et seda rakendust käivitada. Sama info on toodud ka rakenduse README.md-failis.

1. Alustuseks peab olema arvutisse installeeritud PHP ([installed PHP](http://php.net/manual/en/install.php)).

NB! Kui üritad tööle saada teise inimese loodud rakendust, mis on ehitatud PHP 7-nda versiooni peale, siis tee kindlaks, et sinu arvutisse on PHP7 versioon installeeritud.

Juhul kui sinu arvuti operatsioonisüsteemis PHP7 paketid puuduvad, siis saad need lisada järgneva [juhendi](https://www.digitalocean.com/community/tutorials/how-to-upgrade-to-php-7-on-ubuntu-14-04) alusel.

sudo apt-get install php7.0

sudo apt-get install php7.0-xml

1. Arvutisse peab olema installeeritud [SQLite](#_3q5sasy)

sudo apt-get install php7.0-sqlite3

1. Arvutis peab olema [Composer](https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx).

Komposeri installeerimiseks pead täitma korraldused [juhendist](https://getcomposer.org/download/)

Et komposter oleks globaalne, pead kasutama järgmisi [korraldusi](https://getcomposer.org/doc/00-intro.md#globally).

sudo mv composer.phar /usr/local/bin/composer

1. Giti seadistamine (valikuline)

Juhul kui kasutad Git

Laadi projekt alla koos Git’iga

Edaspidi toimub arendus Git’i abil

Git’i kasutamisel peab sul olema GitLab konto ja SSH-võti arvuti jaoks.

Git’i installeerimine:

sudo apt-get install git

git config --global user.name “Firstname Lastname”

git config --global user.email “youremail@gmail.ee”

Loe lähemalt [Git](#_25b2l0r), [GitLab](#_kgcv8k) ja [SSH-key](#_34g0dwd).

[SSH-võtme genereerimine](https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/) arvutile.

ssh-keygen -t rsa -b 4096 -C "[your\_email@example.com](mailto:your_email@example.com)"

cat ~/.ssh/id\_rsa.pub

Kopeeri tulemus SSH-key enda GitLab kasutaja seadetesse.

1. Laadi alla projekt aadressilt <https://gitlab.com/i-sepp/bcs-koolitus> .

git clone [git@gitlab.com](mailto:git@gitlab.com):i-sepp/bcs-koolitus.git

1. Käivita rakendus enda arvutis vastavas projekti kaustas:

composer install

Installeeritakse kolmandate osapoolte tarkvara.

Loo kaust andmebaasi hoidmiseks:

mkdir database

1. Käivita projekti kaustas veebiserver ([run the command-line web server](#_1jlao46)):

php -S localhost:8000 -t web

1. Mine brauserisse ja ava URL: [http://localhost:8000/bookings/create](http://localhost:8000/hello)
2. PHPUnit testing: vendor/bin/phpunit tests



Näidisrakenduse leiad aadressilt - <https://gitlab.com/i-sepp/bcs-koolitus>

## Lõpetuseks

See on siiski vaid üks näide, mis annab esmase ülevaate PHP’st ja Silex raamistikust. Rakendust ennast on võimalik luua mitmel erineval viisil ja erinevaid vahendeid kasutades, kuid siiski on see hea näide, kuidas PHP raamistikku kasutades luua lihtsat veebilahendust. Lisaks rakenduse koodi kirjutamisele, on lisatud nõuanded, kuidas kirjutada jätkusuutlikku ja loetavat koodi. Samuti on lisatud märkuseid ja nõuandeid, millele pöörata tähelepanud dokumenteerimisel ja testimisel. Neid ei tohiks koodi kirjutamise ajal jätta tähele panemata, sest nii dokumenteerimine kui testimine kuuluvad programmeerimise juurde, mitte ei ole eraldi etapid enne või pärast programmeerimist.



Kasulikud eesti keelsed õppematerjalid

<http://pld.ttu.ee/~priidu/courses/web/WebCourseV2/Slides/WEB-06-PHP-Syntax.pdf>

Programmeerimine keeles PHP, Andrei Porõvkin, Tartu Ülikool, 2009 <http://dspace.ut.ee/bitstream/handle/10062/14220/Programmeerimine_PHP.pdf;jsessionid=5F10184E591DE4A5B8E6F75D97AD93EA?sequence=1>