

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

**Learning materials for software development**

**How to generate a simple application using**

**a PHP framework - Silex**

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)

March 2017



This project has been funded with support from the European Commission.

This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

[Introduction 5](#_Toc491327137)

[What is ahead? 5](#_Toc491327138)

[The basics of PHP 6](#_Toc491327139)

[Basic syntax 6](#_Toc491327140)

[Variables 6](#_Toc491327141)

[Types 6](#_Toc491327142)

[Basic of variables 6](#_Toc491327143)

[Variable scope 6](#_Toc491327144)

[Predefined variables 6](#_Toc491327145)

[Constants 6](#_Toc491327146)

[Expressions and operators 6](#_Toc491327147)

[Arithmetic operators 6](#_Toc491327148)

[Assignment operators 7](#_Toc491327149)

[Comparison operators 7](#_Toc491327150)

[Logical operators 7](#_Toc491327151)

[String operators 7](#_Toc491327152)

[Array operators 7](#_Toc491327153)

[Control structures 7](#_Toc491327154)

[Decision making 7](#_Toc491327155)

[Loops 7](#_Toc491327156)

[Functions 7](#_Toc491327157)

[Error handling - Errors and Exceptions 8](#_Toc491327158)

[Object Oriented Concepts 8](#_Toc491327159)

[Classes and objects 8](#_Toc491327160)

[Magic functions 8](#_Toc491327161)

[\_\_contstruct and \_\_invoke 8](#_Toc491327162)

[Defining namespaces and using them 8](#_Toc491327163)

[Extending other classes functionality 8](#_Toc491327164)

[Visibility 8](#_Toc491327165)

[Debugging 8](#_Toc491327166)

[Sending information to the web server 8](#_Toc491327167)

[GET, POST, REQUEST 8](#_Toc491327168)

[PHP configuration 10](#_Toc491327169)

[One example of using PHP from command line 10](#_Toc491327170)

[Possibilities to get your PHP-server up and running 10](#_Toc491327171)

[Documentation 10](#_Toc491327172)

[Testing 11](#_Toc491327173)

[Frameworks 12](#_Toc491327174)

[Some examples of Php frameworks 12](#_Toc491327175)

[Symfony/Silex 12](#_Toc491327176)

[Git 12](#_Toc491327177)

[Creating Web application 12](#_Toc491327178)

[GitLab 13](#_Toc491327179)

[Adding SSH-key to GitLab 13](#_Toc491327180)

[Git 14](#_Toc491327181)

[Creating a new repository 14](#_Toc491327182)

[First commit with Git 15](#_Toc491327183)

[Adding Silex to your project 15](#_Toc491327184)

[Using composer 15](#_Toc491327185)

[Gitignore 16](#_Toc491327186)

[Starting you PHP built-in web server and running the first PHP file inside of it 17](#_Toc491327187)

[Creating your first HTML template 18](#_Toc491327188)

[Using Twig for templating 18](#_Toc491327189)

[Add your base styling and HTML 19](#_Toc491327190)

[First usage of Twig for separating base page to components 19](#_Toc491327191)

[Creating database by using Doctrine DBAL 20](#_Toc491327192)

[Composer - PSR-4 Autoloading 22](#_Toc491327193)

[More methods to the class Application.php 25](#_Toc491327194)

[Adding a form 26](#_Toc491327195)

[Controller code to separate class 30](#_Toc491327196)

[Saving data to the database 32](#_Toc491327197)

[Retrieving submitted data from the database 34](#_Toc491327198)

[Creating menu 37](#_Toc491327199)

[Little insight into the testing: PHPUnit 39](#_Toc491327200)

[Running ready-made Booking application 41](#_Toc491327201)

[Concluding notes 42](#_Toc491327202)

# Introduction

The main purpose of this material is to teach you a new programming language - PHP. But that’s just one goal. Another very important part of the purpose is to understand how this programming language plays a huge role in software development industry and how this language is used in a practical way. As in practice, nothing is used on its own, we will use some examples to show, how PHP is one important block of a web-development and can be having connections with many other systems. Most importantly, this material will give you your first practical experience. This experience should give you a little taste and initial idea of developing software in a web.

While you read this material, you will see many links. It is highly recommended to open these links and read them thoroughly. In this material, learning part is not done for you by adding all the information only for you to read. It only includes the practical part with explanations, But theory is located in the links provided. Before starting to create this application here, you need to know the basics about PHP programming language. To make it easier for you, we have given you the main PHP themes to learn before starting to develop this application.

Also, you will see links that will lead to a page (Gitlab) usually named as commits, where this example application code has been already generated following all these steps that are described here. You can follow all the code changes all along from there.

## What is ahead?

Besides learning a new language you will get an experience in all stages of software development in a team.

You will get to know some widely used tools to develop a sustainable software as a team.

Most important subjects ahead are:

* PHP and frameworks (Silex)
* Version control systems and source control applications (Git, Gitlab)
* Dependency managers (Composer)
* Unit testing (PHPUnit)
* Templating languages (Twig)
* Databases (MySQL, SQLite)

# The basics of PHP

[PHP](http://php.net/manual/en/index.php) (recursive acronym for PHP: Hypertext Preprocessor) is one of the most commonly used server-side scripting language. [PHP](https://www.tutorialspoint.com/php/php_introduction.htm) is a scripting language, which means that the code does not need to be compiled before it is being used, it is processed during usage. PHP is a server-side language which means that the code is not executed in the browser or on your computer, but it is executed in the computer you requested it from. The results of the request are handed over to you and displayed in your browser.

## Basic syntax

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

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

## Variables

### Types

<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>

### Basic of variables

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

### Variable scope

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

### Predefined variables

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

## Constants

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

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

## Expressions and operators

<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>

### Arithmetic operators

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

### Assignment operators

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

### Comparison operators

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

### Logical operators

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

### String operators

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

### Array operators

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

## Control structures

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

### Decision making

<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>

### 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>

## Functions

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

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

## Error handling - Errors and Exceptions

<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>

## Object Oriented Concepts

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

### Classes and objects

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

### Magic functions

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

We will use \_\_construct and \_\_invoke in the Silex project!

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

### Defining namespaces and using them

We will use namespaces in the Silex project!

<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>

### Extending other classes functionality

We will use extending in the Silex project!

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

### Visibility

We will use this in the Silex project!

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

## Debugging

It is important to teach a new programmer a debugging skill. From the first moment a person starts to program, they need to know how to find the mistakes that he makes while he codes, so he could recover from failure.

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

## Sending information to the web server

### 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 configuration

There are mostly three different areas where PHP scripting is used (for more information see article: [What can PHP do?](http://php.net/manual/en/intro-whatcando.php)). Two of them are:

1. Server-side scripting. This is the most traditional and main target field for PHP. You need three things to make this work. The PHP parser (CGI or server module), a web server and a web browser. You need to run the web server, with a connected PHP installation. You can access the PHP program output with a web browser, web server forwards request to the PHP parser and the parser executes bits of PHP within the file and returns it back to the web server. You can run a web server on your home machine if you are just experimenting with PHP programming.

 Full instructions are [here](http://php.net/manual/en/install.php).

1. Command line scripting. You can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way (full instructions are [here](http://php.net/manual/en/features.commandline.php)).

### One example of using PHP from command line

1. Open the terminal in your operating system
2. [Install](http://php.net/manual/en/install.unix.debian.php) PHP on your computer.

For Linux Ubuntu following command would work:

sudo apt-get install php7.0-cli

sudo apt-get install php7.0-xml

1. You can see the PHP information by typing php -i ([command line options](http://php.net/manual/en/features.commandline.options.php))
2. Start the PHP interactive mode by typing php -a
3. That is basically it. To test it, type echo “Hello world!”; If Hello world! Is printed out, then you are using PHP on your [command line](http://php.net/manual/en/features.commandline.usage.php)!

## Possibilities to get your PHP-server up and running

* LAMP (Linux, Apache, MySQL, PHP)
* WAMP (Windows, Apache, MySQL, PHP)
* PHP built-in webserver

We will use the last one. Use a Linux operating system and the inner-built web server. Just to have more time on coding as configuring LAMP or WAMP takes time and learning.

# Documentation

Here are some practical tips about documenting while doing the practical work.

1. The code itself is the most important part. It has to be clear and simple in order for other developers to understand it. The code should have some needed comments, yet it is also important not to fill it up with comments in a way that it becomes useless and only adds some unnecessary “noise” to the code. We will add some examples of this later on.
2. Version control system also plays a huge part of the documentation. As you always “commit” a part of code. You will have to write a describing commit name (and longer description if needed). This is very important because it will create a history of coding where you can always turn back to for example, while you are debugging. The commit name should include the content of the commit, which means that you describe what code is committed here and what for. Also if you had any special notices or problems, it is very good to write it all up there ([Explaining article about commit good commit messages](http://chris.beams.io/posts/git-commit/)).
3. Any additional documents are needed if you start a new project or write a new specific feature. Every project should have a readme, where all the most important parts of the projects are written down. There are more examples of how a README should look like later in the practical task.

# Testing

There are different ways of testing. Some of them are the following::

1. **Testing while coding**. This is the basic way, which is the most natural part of the process. You need to test the code you write. You need to be sure that every time you run any part of your code, it works just as expected. You should never send your code anywhere without testing it first. It might happen if you become more confident and change very “easy” parts.
2. **A second person (other developer or tester) should test it**. It is very good for a second person to test it. It is usually in a part where you have finished coding, but in bigger tasks, it can be done in some checkpoints. Another person might see some problems or logical mistakes that the developer missed. In addition, another person might use another use case that would reveal some bugs that the developer might have missed.
3. **Code review**. Another person (usually developer) should read your code. He can see some problems a developer missed, any logic mistakes. Maybe he helps to make the code simpler or refactor it so that after some time has passed then everyone still would understand it and the code stays sustainable.
4. **Automatic tests**. They are mostly needed in the most complex parts because sometimes it is hard and time-consuming to test every use case manually. However, the more you have them, the better. It is good to have the basic tests on every page of your project. Because it will help you especially when you have reusable code, so you will not break something somewhere else in a code you did not change, but is still dependent of the code you did change. In the practical exercise, we slightly look into PHP testing framework called PHPUnit.

# Frameworks

In general, a [framework](https://en.wikipedia.org/wiki/Software_framework) is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful. A software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code. You often have similar code functionality in different places that your code needs to execute with different data. A framework helps you not to write all the code again from scratch, but you can use general code all over again by only specifying some details. It helps you to reduce time and effort and make your code organized in a better way and easier to modify if needed.

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

## Some examples of Php frameworks

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

## Symfony/Silex

Symfony is a set of PHP Components, a Web Application framework, a Philosophy and a Community — all working together in harmony ([6 good reasons to use Symfony](https://symfony.com/six-good-reasons)).

# Git

Git is the most widely used version control system. Simple definition of version control system is that it helps you to keep track on all the code you write. You will have a history of when something was written or updated. It also helps you to write code as a team. So that everyone could do their assignment without disturbing others. In addition, Git is a good way to document the reasons why you make any code changes. ([More information](https://www.atlassian.com/git/tutorials/what-is-git/version-control-with-git) and [a video](https://git-scm.com/video/what-is-git))

To get a fast introduction to the world of Git You should try this link [interactive tutorial](https://try.github.io/levels/1/challenges/1) and there is [another one](https://onlywei.github.io/explain-git-with-d3/) for more complicated Git interactions.

# Creating Web application

A source control application - repository manager - is something you most probably use if you develop a software in a team. It helps you to share your code, see the changes and automate many parts of the developing process. It is usually one of the most helpful tools. This is where the starting point of a new project usually lies.

The most well-known source control applications are:

1. Github
2. Bitbucket
3. GitLab

The basics are the same in all of them. They work hand-in-hand with version control system - Git. It is easy to use and as Git is also a part of the material, it would be perfect to use theory in a practical way. In this example, we will use GitLab ([Introduction to GitLab](https://about.gitlab.com/2015/05/18/simple-words-for-a-gitlab-newbie/)).

The application we are going to create with these instructions is located here - <https://gitlab.com/i-sepp/bcs-koolitus/>

## GitLab

1. Create yourself an account in <https://gitlab.com/users/sign_in> and log in.
2. Create yourself a new project - <https://gitlab.com/projects/new>.
3. Now if you open the project you should have some instructions there.

## Adding SSH-key to GitLab

To prevent other users from gaining access to our code, we use SSH-keys to log into GitLab from the command line. It means that every computer has its own unique SSH-key. And it is added to your GitLab user. This way every time you communicate with GitLab, it uses this key to authenticate the user.

First, go to GitLab and find your *profile settings* and then [*SSH-keys*](https://gitlab.com/profile/keys)*.*

There you have to add your computer SSH-key. Probably you need to generate it.

There are some instructions how to do it:

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

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

Now you are ready to start working with Git!

NB! You also have an option not to use Gitlab or any server and only learn Git by using it locally in your machine. For that you need to install git, config global variable (see next chapter) and initialize git in the folder that you want to use it, by running git init.

cd Projects/my\_project\_name

git init

Now you can start using git. (git add, git commit etc.)

## Git

1. Install Git on your local machine (preferably Linux).

 Example: I am using Ubuntu so I write to the command line:

 sudo apt-get install git

 All the instructions how to install Git in different environments are located [here](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git).

 Linux specific are [here](https://git-scm.com/download/linux).

1. Now we need to config the global variables for Git.

Git will use them in the future and link them with the code you will write.

That way we will create a history of who wrote what. You can also see the instructions in GitLab in your new project - Git global setup.

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

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

1. And you have configured your Git, now it’s time to use it!

## Creating a new repository

These instructions are also already in your new project in GitLab.

Now we can create a link between a folder in your computer to your new project in GitLab.

1. First, use your command line to go to the folder where you wish to create your new project file.

I have a special folder called Projects, so you must go there.

cd Projects/

1. Then we will copy the new project folder from GitLab to my computer.

 Note that the name would be according to your project.

 git clone git@gitlab.com:my\_project\_name.git

1. Create a README.md file.

Readme is a huge part of documenting your project. It should contain all needed information about your project. Of course you can add information gradually when developing your project, but at first, it would be good to create this file in the first place.

Some readme file templates to help you get started:

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

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

You can do it by opening files explorer and add a new file to your project folder.

Or you can use the command line:

cd my\_project\_name/

touch README.md

## First commit with Git

Now we can upload the changed to GitLab. This will be also the first commit of the project!

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

The last command pushed your changes to Gitlab server.

If you go to GitLab you should see your new README-md file there. Also in Repository->Commits you should see your first commit.

For more information see [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/5f7c35d234270291458ad5bb69846a4859edcd25) ready-made commit in GitLab.

Now we can finally start coding!

## Adding Silex to your project

### Using composer

To [download Silex](http://silex.sensiolabs.org/doc/master/usage.html#installation) to your local machine, we will use [Composer](https://getcomposer.org/doc/00-intro.md).

Composer is a dependency manager for PHP. You can use it to download all needed third-party software automatically. It simplifies using third-party code in your software project.

Also, you will have a file, where is a list of PHP packages that are needed for the project. And everybody who use your software will have exactly the same versions of those PHP packages.

Download it by using the instructions here -

1. [Install composer](https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx) on your machine by the instructions in [here](https://getcomposer.org/download/).
2. Download Silex by the instructions [here](http://silex.sensiolabs.org/download).

It will create for you a composer.json file that will hold all your project dependencies.

It also created a vendor/ folder. This folder will storage all your third-party code. If you run git status you can also see the file composer.lock It is auto-generated and updated automatically every time you install something with Composer.

### Gitignore

As vendor/ folder will be downloaded automatically with composer and [we can never ever change any code in there](http://www.lonecpluspluscoder.com/2014/12/29/im-strongly-opposed-modifying-third-party-library-code/), then we won’t add this to Git staging area. The reason why we shouldn’t change any code in third party libraries is that you will lose the ability to upgrade the library to newer versions and you will have much more code to manage. Thus, don’t need this folder to be uploaded to GitLab. We need to tell Git what kind of files need to be ignored. ([.gitignore](https://help.github.com/articles/ignoring-files/))

We need another new file, where to define the routes of the files you don’t want to upload to GitLab.

1. Create a new file to your project called .gitignore.
2. Define vendor folder route in that file. (vendor/)(See the commit linked at the end of this chapter.)
3. Type git status
4. If you still see the vendor folder in your staging area then run git rm -rf --cached vendor/. If you don’t see it, then it is done. This part is also described in the following [link](https://help.github.com/articles/ignoring-files/#create-a-local-gitignore).

Now there should be three files in your fit staging area:

.gitignore

composer.json

composer.lock

Commit them as the second commit of the project.

Committing:

1. git add . (This adds all the new files in project files to the commit staging area, alternatively you can do it separately - git add .gitignore composer.json composer.lock)
2. git commit -m “Adding Silex to project by using composer”
3. git push origin master(optional - you don’t have to push after every commit, you can do it whenever you need to get your code online)

Note: If anyone clones you GitLab repository and wants to run your code, they would need to install composer on their computer and run “composer install” in you project folder, then they would also have the vendor folder up and running.

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/37adf88875f75af62605d59548f5407dd0d2e221) ready-made commit in GitLab.)

As we took some new-third party software into use in our project for the first time and it affected the way our software can be installed, we should document it into the README.md.

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/8a10c7754078fd52c82988493df8a2ae96e20423) ready-made commit in GitLab.)

Also from now, we will assume you can already create new commits and upload them to the GitLab. So these instructions won’t have any more detailed instructions about creating commits.

## Starting you PHP built-in web server and running the first PHP file inside of it

For a web server, we will use PHP built-in web server. The presumption is that you already installed PHP on your local machine.

But before that, we will need a PHP file to run in the web-server.

1. Create a new folder called web. In there we will create a [front-controller](https://en.wikipedia.org/wiki/Front_controller).

 mkdir web

1. Then create front-controller called index.php. touch index.php
2. Add some initial code to use for testing in order to see if our web server is running. [Bootstrap Silex](http://silex.sensiolabs.org/doc/2.0/usage.html#bootstrap) into our front-controller and turn on the debug more for developing.
3. And then define the first [route](http://silex.sensiolabs.org/doc/2.0/usage.html#routing) for testing.

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

 **return** 'Hello world!';

});

1. Now, test it. Run your [command line web server](http://php.net/manual/en/features.commandline.webserver.php) by typing in your project folder:

 php -S localhost:8000 -t web

1. Go to your web browser and open URL: <http://localhost:8000/hello>

 If you see a white page with a single string saying: “Hello world!” then it is working!

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/7c0ade69355ac1197362145b935169e72550e0ec) ready-made commit for code changes until now.)

Note: From now on, while coding, the PHP errors would be displayed in your command line, where your command line web-server is running. If your code does not work, it would be the first place to look for an error to find mistakes in your code.

## Creating your first HTML template

### Using Twig for templating

It would be much preferable if we had an HTML template where all the template logic could be located and PHP-files, where all the back-end functionality will be managed.

For that, we are going to use [Twig](http://twig.sensiolabs.org/). Twig is something that goes hand-in-hand with Silex (Symfony) ([Twig in Silex](http://silex.sensiolabs.org/doc/master/providers/twig.html)).

By using Twig, we can separate HTML templates from PHP language, as Twig will have the role of the middleman between PHP and HTML. Twig is very helpful and brings your HTML to a new level. It is very easy to learn the basics. Yet, it has some very helpful, a little bit more complicated features that will make your life easier, as you choose to work with Twig and improve your skills in this templating language. You can create a custom function in Symfony/Silex and use it in your template with Twig without overflowing you templates with many long PHP methods. Probably during this course, we will touch the simplest surface of Twig. It will be easy as it behaves just as PHP just in a little bit of a different syntax.

1. Download this new Twig dependency. ([Instructions here](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#registering))

 composer require twig/twig

1. We also need to register it in the front-controller:

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

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

 ]);

Here we also define, that directory called “views” will become the storage folder of our html.twig files. This is just a name of the folder that you choose. I named it views.

1. Create a folder for templates - “**views**”.
2. In there we would create a first HTML file. I will name it “**base.html.twig**”.

It would need the Twig extension at the end of the file name so that we could write Twig language inside of this file. Also, this template would hold our apps base template and general style that would be used in all of the application pages.

1. In that file, I would add all basic HTML tags that a new page would need (See the code in [commit](https://gitlab.com/i-sepp/bcs-koolitus/commit/e9be656252ab10102fa1911a61427ff84ffc3a1b)).
2. Now, all that is left to do is [use it in a controller](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#usage) - index.php. I would also like to change the route name to something more logical. Like “bookings/create” for example.

So the changed code part instead of /hello route in index.php to:

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

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

});

1. Test it in your web browser with a new url: [http://localhost:8000/bookings/create](http://localhost:8080/bookings/create)
2. Documentation. Also, as we took some new dependencies into use, write it up into the README.md (see [commit](https://gitlab.com/i-sepp/bcs-koolitus/commit/b9e70d04eb5f56a7065b52a7398b728e35f56055)).

## Add your base styling and HTML

For everything to look good, we need a little bit of styling. So let us add some basic CSS and HTML.

We should have a new folder for CSS. As we won’t be getting into minifying CSS/Javascript then we will make this folder *into the web folder*. Usually, for CSS/Javascript, there are two locations in a project. First, a location where you write the code - your CSS and Javascript folders and then the same folder in the web folder that are actually used in a browser while displaying the page. The reason is simple. The duplicate folder is generated automatically and all the code is minified. As huge projects have a lot of resources then the code needs to be minified to cut the loading time. In this case for learning purposes, as it takes extra tools to use and extra configuration, we will just create the assets files directly to the web folder.

As everyone already knows HTML and CSS, you can only see my HTML and CSS as an example, but you should create your own (See my code from [here](https://gitlab.com/i-sepp/bcs-koolitus/commit/00e5b1d0c5790ac9e873e156197c27dacf7c13ea)).

## First usage of Twig for separating base page to components

We could separate page elements to different components by using Twig. This means that different parts of our pages will be stored in different files. It is a good practice for many reasons:

1. Code is more reusable
2. Easier to read
3. Every component is separately, so it is easy to change it, refactor, redesign it without breaking anything else.

To do that, use Twig functionality called [include](http://twig.sensiolabs.org/doc/2.x/templates.html#including-other-templates).

Separate the header and the footer HTML and CSS into separate files.

So there will be 4 new files \_header.html.twig, \_footer.html.twig, header.css, footer.css.

NB: Note the documentation of the new components in the new HTML files upper part

(see [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/6af096e920e1b0ffac4f50fc9e98365de2bd4747) ready-made commit for code changes until now).

## Creating database by using Doctrine DBAL

[Doctrine DBAL](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/) is a powerful [database abstraction layer](https://en.wikipedia.org/wiki/Database_abstraction_layer) with many features for database schema introspection, schema management and PDO abstraction.

We will use it to create a database and communicate between PHP and database.

We will need a database to save some data from a filled form. In addition, the database vendor we will use will be [SQLite](https://www.sqlite.org/about.html).

To do that, follow the next steps.

1. First, we need to download Doctrine/DBAL with Composer ([using Doctrine in Silex](https://silex.symfony.com/doc/2.0/providers/doctrine.html)).

 composer require "doctrine/dbal:~2.2"

1. Install SQLite on your computer. I use Ubuntu 16.04, so for me, one of the following commands will work depending on the version of PHP I use. Example:

 sudo apt-get install php7.0-sqlite3

Note: The instructions to install something might change over time and might be different according to the operating system you use. So google how to install SQLite on your machine depending on your operating system.

After this step, you need to restart your PHP inner-built web server.

1. Documentation. We should also update the README.md file, as we started to use a new third-party library called Doctrine DBAL. And also SQLite, as it needs to be installed on every machine that this software will run (see [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/525df3931d0bad6bd8b5efe8398ba8cab60f6a79) ready-made commit for changes until now).
2. Then we need to register the new service in our front-controller.

First, we need to create a new folder where the database file will be located. I will call the folder *database.*

 ([Instructions are here.](http://silex.sensiolabs.org/doc/master/providers/doctrine.html#registering) [More information about Doctrine general configuration](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',

 ],

]);

In the db.options, we are defining as a [driver](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#driver) the database vendor we will use. In our case, it would be 'pdo\_sqlite'.

Then we will define a path, which value is a string that defined the path to the folder location that will store our database.

(SQLite configuration instructions using SQLite are [here](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#pdo-sqlite).)

Note: If your application code is not working and error in the command line is saying that a driver is missing then it means that the previous step - installing SQLite did not succeed.

1. Database files shouldn’t be uploaded to Gitlab. It is something automatically generated in development environment. So we should add the database folder to the .gitignore file.
2. Create a MySQL query to create a table for bookings.

 The MySQL to use can be something like this:

 "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

);"

 And we will use DBAL Connection.php [executeQuery](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/data-retrieval-and-manipulation.html#executequery) function to run this MySql.

 So the code would be

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

1. As we only need to create this table once, only if it does not exist. We need to have this code inside an if-clause. We will use [SchemaManager](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/schema-manager.html) to check whether this table exists in our database or not.

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

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

}

1. Now if you reload your page ([http://localhost:8000/bookings/create](http://localhost:8080/bookings/create)) again,

 this code should be run and database with a table called “bookings” should be created.

For initial testing, you can always use PHP [var\_dump](http://php.net/manual/en/function.var-dump.php) to see if the code will enter the if-clause on the first time. The next times it should not do that. But never commit with Git the var\_dump parts as it is only for some initial visual testing purpose (see php part about debugging).

Note: You can alway recreate the generating database part if you delete app.db from the database folder.

1. You can check if the database file was created if you look into the database folder to see if there is a file called app.db.

 (see [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/942eb77a32fb499c3cd487a3b43232b303433881) ready-made commit for changes until now).

## Composer - PSR-4 Autoloading

As you can see, the front-controller is getting bigger and bigger. We even have not added a form to our page, but we have defined new services and new functionalities and we will continue to do so.

There is one problem. All our PHP code is located in the front-controller. Imagine now, that you will have a website that will have 20 different pages. Maybe many more tables in our database and much more complex functionality. It is very unreasonable to write all our code into one file. It makes it hard to read, develop and later automatic testing will also be quite impossible.

Thus, we need to separate out code into logical pieces and locate logically in our project folder, to different files and methods. Meaning - we will start to practice object oriented programming.

Front-controller is always the file that will be firstly called if you run your web page, but it will execute the code in other files.

So we will also separate some of our code into separate files.

To do that, we will use [PSR-4](https://getcomposer.org/doc/04-schema.md#psr-4) autoloading to load our new files to our front-controller. Just as the composer does it with the files located in vendor folder ([Good article about PSR-4 autoloading](https://seld.be/notes/psr-4-autoloading-support-in-composer)).

1. Firstly, we need to create a new folder, where we will store our new PHP files. I would call it src (meaning source).
2. Now we need to define this new folder for composer autoload. Open composer.json file and add the new path using PSR-4. I named the path with a namespace BookingApp. Be sure to learn about php namespaces to understand this part better. ([Good video about it.](https://knpuniversity.com/screencast/php-namespaces-in-120-seconds/namespaces))

"autoload": {

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

}

 Now we can call the files in src folder using a namespace called BookingApp.

 You also need to run composer install for the changes to take effect.

 composer install

1. Now we will create a file called Application.php into the src folder. This will hold a new [class](http://php.net/manual/en/language.oop5.basic.php) called Application.
2. As it is a PHP file, it has to start with [PHP opening tag](http://php.net/manual/en/language.basic-syntax.phptags.php):

 **<?php**

1. Next, we will define the [namespace](http://php.net/manual/en/language.namespaces.rationale.php), where this file is located.

namespace BookingApp;

1. We need to define the class name that has to be same as the file name.

class Application

{

}

1. As this class will have the same functionality as Silex\Application, then we will make this class to [extend](http://php.net/manual/en/reflection.extending.php) it. But first, we need to [use it](http://php.net/manual/en/language.namespaces.importing.php) in order to extend it. I will give it a custom name(SilexApplication) for better and clearer usage.

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

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

1. We will bring the code from index.php to Application.php

The first row we will take from index.php and redefine the functionality in Application.php is:

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

As you can see the PHP file Silex\Application.php is called. If you open that file then you will find the [\_\_construct method](http://php.net/manual/en/language.oop5.decon.php) that is triggered every time this file is called. The code in the \_\_construct method will trigger all other needed methods.

As we extend this file we need to overwrite the \_\_construct method. So in the index.php when we will call out custom Application.php, then also the new custom \_\_construct method will be triggered and we can add there some custom methods to call.

As we are overwriting it, it has to be with same parameters as the \_\_construct method in Silex\Application.php. And we still need to call call the [parent](http://php.net/manual/kr/keyword.parent.php) \_\_construct. So this is the first row we will add into the custom \_\_construct method.

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

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

}

Note: To understand this part better, please read the links about \_\_construct and parent in this chapter, there are simpler examples of the same functionality.

1. Now we can bring code from the index.php to Application.php. That code is $app config options, new service registering, database table creation and route definition. And as we bring it to the custom Application [class](http://php.net/manual/en/language.oop5.basic.php), then the code changes in existing code would be:
	1. *$app variable will be $this* now
	2. We need to use all the other file namespaces in the beginning of the file.

 **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. In index.php file we will call out our new custom Application.php by using its new defined namespace.

 **<?php**

**use** BookingApp\Application;

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

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

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

1. Now if everything works as before, then creating a new class succeeded.

(This [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/77494941649bd5a49f380eb4434ae85589745f7b) ready-made commit with code changes until now).

## More methods to the class Application.php

Well, now you might think that what was the point if now we will only enlarge the method \_\_construct in Application.php.

Actually, we will create some new methods to make more separate logical pieces and we will create new classes.

We have three logical pieces in our code.

1. Registering services
2. Creating tables to the database
3. Defining routes

According to that, create three private functions.

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

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

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

Private because we will only use them inside of this class ([Read more about visibility](http://php.net/manual/en/language.oop5.visibility.php)).

Note: Basically you can name them however you think is more logical. But remember that you have to use names that are understandable for other programmers. That the name of the functions would indicate the purpose of the function. (Programs should be written for people to read, and only incidentally for machines to execute. -- from "Structure and Interpretation of Computer Programs" by Abelson and Sussman)

Now all that is left to do is write the code inside of the functions and call the functions in the \_\_construct method.

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

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

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

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

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

 }

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/70d38e52c45593bab8aa5410933e8e56831aacf0) ready-made commit for the code changes until now.)

## Adding a form

Next step is to create a form on our page. For that, we will use [FormServiceProvider](http://silex.sensiolabs.org/doc/2.0/providers/form.html).

1. Firstly, we need to [download some form components](http://silex.sensiolabs.org/doc/2.0/providers/form.html#registering) with Composer.

composer require symfony/form

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

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/1f083b5f07442344c86dcbace3de4475b510304b) ready-made commit for the code changes until now.)

1. Then, we need to register new services in 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' **=>** **[]**,

]);

Note: We are not going to use any custom translation logic at the moment, but we will use twig-bridge component, that requires [TranslationServiceProvider](http://silex.sensiolabs.org/doc/2.0/providers/translation.html) to be registered.

1. Create a [form](https://symfony.com/doc/current/forms.html) object, where we will [define all the fields we wish to render](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()

;

Form factory provides many [form types](https://symfony.com/doc/current/reference/forms/types.html). Every type separately has different available options to use. For example, [DateType](https://symfony.com/doc/current/reference/forms/types/date.html) has an option “[format](https://symfony.com/doc/current/reference/forms/types/date.html#format)”, that you can use if you would like to have a different kind of format to the date string in form. To know more about FormType possibilities, I recommend exploring the documentation.

1. All the form types namespaces have to be defined, thus they need to be [used](http://php.net/manual/en/language.namespaces.importing.php) in the beginning of the file.

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

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

**[ … etc …]**

1. Now we need to show our created form in some template. For that, we will create a new template called form.html.twig. And we will use Twig [template inheritance](http://twig.sensiolabs.org/doc/2.x/templates.html#template-inheritance) to give this template the same basic HTML and CSS as we already have in our base.html.twig.

 Thus, at first, we will create some block that we can reuse in our form.html.twig.

 Those would be three blocks: stylesheets, content and 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. In form.html.twig, we will define that this new template [extends](http://twig.sensiolabs.org/doc/2.x/tags/extends.html) the base.html.twig.

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

1. We can also add some content to the content block. As we are trying to [render a form](https://symfony.com/doc/current/forms.html#rendering-the-form), we can use form rendering ready-made functionality. You could also [render every field separately](https://symfony.com/doc/current/form/rendering.html#rendering-each-field-by-hand), but at the moment we don’t need to.

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

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

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

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

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

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

1. Now we should use this template in the controller. (See code changes until now [here](https://gitlab.com/i-sepp/bcs-koolitus/commit/4340c044db42aa036be177e3f89dfab2aecc3ba7).)

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

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

]);

1. Now if you reload your page, you should see the form. Quite ugly, but it’s there.

 Why does the HTML can have only 3 rows of code to render so much HTML?

Default twig form theme was used. It is located - /vendor/symfony/twig-bridge/Resources/views/form/form\_div\_layout.html.twig

There you can see a mix of HTML and Twig that creates reusable HTML code for your form elements.

Twig [form theming](https://symfony.com/doc/current/form/form_customization.html#what-are-form-themes) is something that makes coding much faster. You can always [customize your form elements](https://symfony.com/doc/current/form/form_customization.html) but you don’t need to write a huge amount of HTML every time you create a new form. We will practice just a little bit of custom theming so we could make our form look nicer.

1. I want to have a custom class to my <form> element. [It’s quite simple to add it.](https://symfony.com/doc/current/form/rendering.html#rendering-each-field-by-hand)

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

1. We will also write a [little custom theme inside of the same file](https://symfony.com/doc/current/form/form_customization.html#method-1-inside-the-same-template-as-the-form).

I would like to have custom class around every field of a form. So we can overwrite the general form\_row Twig block and add what I want. In some cases, I only need to do this for a [specific custom field](https://symfony.com/doc/current/form/form_customization.html#how-to-customize-an-individual-field), but at the moment I am creating general form\_row styling in my whole application.

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

<div class="form-row">

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

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

 </div>

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

1. To use this custom theme I need to [define it](https://symfony.com/doc/current/form/form_customization.html#method-1-inside-the-same-template-as-the-form) in the beginning of the file.

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

1. Now for all the added classes I will write some CSS and add it to the template.

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

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

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

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

 As I wish to keep the styling files from base.html.twig, I will use Twig [parent](http://twig.sensiolabs.org/doc/2.x/functions/parent.html)() function.

1. Now your form page should look so much nicer (see [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/60a306e2051c0b89c88ca4cc2d970fc89ce01259) commit for code example).

## Controller code to separate class

It would be a good idea to separate controller code from our general Application.php code. Again, it is for more simple and readable code, which is sustainable and easier to test.

1. We should create a separate folder for Controllers in our src folder.

cd src

mkdir Controllers

1. Then, I would create a new class for CreateBookingController.php

 cd Controllers

touch CreateBookingController.php

1. Like always any new PHP file starts with PHP [opening tag](http://php.net/manual/en/language.basic-syntax.phptags.php):

 **<?php**

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

**namespace** BookingApp\Controllers;

1. And then, defining the class name

 **class** **CreateBookingController**

{}

1. Now we need to think what kind of variables we need to have inside of this class. If we look at the current anonymous function inside Application.php, then we can notice, that there are two different variables that are taken from outside of the function. They are:

 $this[form.factory']

 $this['twig']

There is no point in passing the whole $this variable. We will pass only the dependencies that we need to the newly created class. Then it is much clearer to control what kind of variables are used inside of the new class. Moreover, it will become easier to test and read. So now that is left to do is call the new class from Application.php

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

$this['form.factory'],

$this['twig']

 ));

Don’t forget to import the new namespace.

**use** BookingApp\Controllers\CreateBookingController;

1. Then we need to receive these variables inside of CreateBookingController.

 Again we will use the [\_\_construct](http://php.net/manual/en/language.oop5.decon.php) method for that.

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

{

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

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

}

We will use [type declaration](http://php.net/manual/en/functions.arguments.php#functions.arguments.type-declaration) or also called type-hints to show the content of the variables. Type hinting means that you can be sure that a suitable dependency has been passed. By type-hinting, you'll get a clear error immediately if an unsuitable dependency is passed

We are using PHP [non-static properties](http://php.net/manual/en/language.oop5.properties.php) to make the imported variables usable in every method if this new class.

Now we can call out the variables everywhere in this class by typing:

 $this**->**formFactory;

\Twig\_Environment is already a global namespace, but we need to import FormFactory.

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

1. Now, bring in the controller functionality. We will use PHP magic function \_\_invoke, so that the function will be called out immediately every time we use this class.

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

1. Now we only need the already written code about form creation and template rendering:

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

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

[ … ]

**->**getForm()

;

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

1. Do not forget to delete the previous namespace importing from Application.php and write them to CreateBookingController.php. (See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/c9004eb7d7e2cf9549a266cd04d3a0fcbb810ddc) ready-made commit for code changes.)

## Saving data to the database

We have a nice form, but it does not save anything to the database.

1. As it can be guessed to save something to the database we need database connection, thus, we need to pass the Connection variable from the Application.php file:

 $this

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

$this['form.factory'],

$this['twig'],

$this['db']

))

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

 ;

In addition, until now, we only used [GET](http://silex.sensiolabs.org/doc/2.0/usage.html#dynamic-routing) method, but as we are going to post a form, then we will need to allow [POST](http://silex.sensiolabs.org/doc/2.0/usage.html#example-post-route) method also. As we still need both of them, then we can use [match](http://silex.sensiolabs.org/doc/2.0/usage.html#other-methods) method and define the allowed methods in the method() function.

1. Now we need to receive the database variable from the CreateBookingController.

 It means importing the Connection namespace and defining a [non-static property](http://php.net/manual/en/language.oop5.properties.php).

 **use** Doctrine\DBAL\Connection;

 [ … ]

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

FormFactory $formFactory,

\Twig\_Environment $twigEnv,

Connection $dbConnection

) {

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

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

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

}

1. Then, we need to add the possibility to [submit some form data](http://silex.sensiolabs.org/doc/2.0/providers/form.html#usage).

As the method that handles the every route in [Silex framework implements](http://silex.sensiolabs.org/doc/2.0/internals.html#application) Symfony’s [HttpKernelInterface](http://api.symfony.com/master/Symfony/Component/HttpKernel/HttpKernelInterface.html), then by default the arguments that are always passed is [Request](http://api.symfony.com/master/Symfony/Component/HttpFoundation/Request.html) and you have to return an HTTP [Response](http://api.symfony.com/master/Symfony/Component/HttpFoundation/Response.html) as we are already doing. We just need to have the request in a variable for usage.

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

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

1. Then we will take all the needed information from the $request object, the POST parameters.

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

1. Then we check if the parameter values are valid (definitely read [this](http://symfony.com/doc/current/forms.html#handling-form-submissions) form submission article).

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

1. Moreover, if it is true, we need to use some MySql to submit this data to the database.

$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']

]);

Note that we need to format the data according to the format we wish to save it into the database field. It means formating some Date objects to string etc.

Also this MySQL query is done in a way to prevent [SQL injection](https://en.wikipedia.org/wiki/SQL_injection), which is one of the simplest way to attack a website. The preventing method is to use parametrized query as this example does (To know more about SQL injection, [read this arcticle](http://www.acunetix.com/websitesecurity/sql-injection/)).

1. To let the user know that the submission succeeded, the most common way is to reload the page.

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

 [ … ]

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

1. And now you can test the form submission, if the page reloads after pressing submit, then it succeeded.

NB! If you get blank page after submission, then be sure that you filled the form as it supposed to be filled. (Meaning, that if date was expected then you enter it as 12.12.1990 etc) Our form doesn’t have the form validation yet. Also it doesn’t have the part that it tells you if you entered any wrong field value. As an additional exercise you can add [form validation](http://symfony.com/doc/current/validation.html) to this app, to fix this problem.

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/fb80d0a965b553f1b96e8ae9a808e76b29c2d876) ready-made commit for code changes until now).

## Retrieving submitted data from the database

It would be nice to see what kind of booking have been submitted. So we can create a new page in our application where we can see all the submitted data placed in a table.

1. Firstly, we need a new route and controller. We can define a new route in Application.php. I will name the new route “bookings” and new Controller as ListBookingsController. As we are going to use another twig template to render the bookings, then we will need the Twig\_Environment variable in the controller. And also we will need to make a MySQL query to the database, thus, we will need the Connection variable.

**use** BookingApp\Controllers\ListBookingsController;

[ … ]

 $this

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

$this['db'],

$this['twig']

))

 ;

1. Now we need to create the new controller class to the Controllers folder.

Just similarly to the CreateBookingsController, we are defining the [namespace](http://php.net/manual/en/language.namespaces.php), importing the variables’ namespaces and defining the passed variables in [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. Now we need to define our [\_\_invoke](http://php.net/manual/en/language.oop5.magic.php#object.invoke) method, where we will perform a query and hand over the result to a new template.

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

1. For query we are using Doctrine/DBAL/Connection.php two methods: [executeQuery()](http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/data-retrieval-and-manipulation.html#executequery) and [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. Then, we need to return a response as a twig template and pass the result there.

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

1. As you can see, I defined a new template that does not exist yet. We need to create it, write some HTML and CSS. Now, remember that all that we retrieved from the database lies in the variable called “bookings” as an array. And you can access to every bookings table field just by using their table field names that are defined in the Application.php createDBTables() method. As bookings is an array you need to use [Twig For loop](http://twig.sensiolabs.org/doc/2.x/tags/for.html) first. Also, if there are no bookings I will not show some HTML, to do that, I am using [Twig empty method](http://twig.sensiolabs.org/doc/2.x/tests/empty.html). Again you can [extend](http://twig.sensiolabs.org/doc/2.x/tags/extends.html) the base.html.twig for the existing layout and style.

 **{%** **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. Now, if you got to URL: [http://localhost:8000/bookings](http://localhost:8000/hello), you should see your submitted data.

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/114293498767a32b28999293e84de063ae186acb) ready-made commit for the code changes until now).

## Creating menu

It is quite inconvenient to change between different URLs, so it would be nice to have a menu.

We can generate ready-made URLs in Twig by using Twig method called [path](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#symfony-components-integration). And we can name the routes, we have created with the [bind method](http://silex.sensiolabs.org/doc/2.0/usage.html#named-routes).

1. Naming the routes:

 $this

 **[ … ]**

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

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

 ;

 $this

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

$this['db'],

$this['twig']

))

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

 ;

1. Then, we need some HTML for our menu. Again, it is a good practice to separate this new component. So I will create a new file called \_menu.html.twig and CSS to the menu.css. And include the base.html.twig.

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

[ … ]

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

1. In the \_menu.html.twig I will have an extra functionality. I want the user to know on which kind of menu link URL is currently open. I will use CSS color property and extra HTML class to do that. But first I need to have a logic in the code to decide whether to add this extra class to the HTML element or not. So I need to know the current URL to compare it to the menu link URL. I can use Twig [global request variable](http://silex.sensiolabs.org/doc/2.0/providers/twig.html#global-variable) and call function getRequestUri(). If you open up Request.php then you can see, that this method is written into the Request class and it is callable from Twig. As I will need the current URL in both menu links, I will save this into a variable with a shorter name, to use it later. To do that I will use [Twig set method](http://twig.sensiolabs.org/doc/2.x/tags/set.html).

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

1. Now, you can use the [Twig If method](http://twig.sensiolabs.org/doc/2.x/tags/if.html) to do the comparison and path to create a path from route names. I will use the [ternary operator](http://twig.sensiolabs.org/doc/2.x/templates.html#other-operators) instead of full if.

 <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. I wrap this HTML up with a div and write some CSS to make it more beautiful and we have a nice menu (this [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/b05b21bfaae2a46109008870c3c948628f22b279) ready-made commit for code changes until now).

## Little insight into the testing: PHPUnit

The last thing, is adding a testing possibility to the application. Later, you can write yourself as many tests as you like while you develop your application further.

We are going to use PHP testing framework [PHPUnit](https://phpunit.de/).

1. Downloading PHPUnit:

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

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

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

Note that we use --dev property of composer as these dependencies are only for developing stage, but not for production.

1. [Configuring](http://silex.sensiolabs.org/doc/2.0/testing.html#configuration) PHPUnit to work with your application:

 You need a separate file for that called “phpunit.xml.dist”

 In there you will define the folder where your tests will be located.

 **<?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. Create that new directory called “tests”.
2. We are going to generate a simple test that will ensure, that the page will be safely rendered to the user.

So I will name this test file as AppTest.php and this test will [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. For the next steps, we need to follow the instructions [here](http://silex.sensiolabs.org/doc/2.0/testing.html#webtestcase). First, we are implementing the createApplication method. As we created our own custom Application.php, then this method needs to return our $app variable.

 **use** BookingApp\Application;

 [ … ]

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

{

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

}

1. Then, we need to write our test. So basically we will test if our bookings page is working generally. That the response does not return any errors etc.

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

 {

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

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

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

}

1. Now, we need to run the test. Type to your command line:

vendor/bin/phpunit tests/AppTest.php

If you see green, then the tests passed. As an experiment you can see that the test will fail, if you delete one semicolon from the ListBookingsController.php

1. Documentation: I would also update README.md with the information about testing.

(See [this](https://gitlab.com/i-sepp/bcs-koolitus/commit/7ab5ba8a72a178a0736d346764dca2919a52d6b2) ready-made commit for the code changed until now).

That is basically automatic testing. Of course, it can be configured that these tests will be run anytime you finish coding something. Now, you should run the tests manually from the command line. Automatic testing requires writing the tests while you code. Every time you write some new code, you can write a test to test this code. It is surely more time consuming, thus for business point of view, you need to decide what kind of code needs automatic testing and what kind of code does not. Usually, some code logic that is more delicate, complicated or user affecting needs to be tested. Then it is important that your code is separated into small pieces - different classes, so you can test one class or even one method at a time.

If you want to know more about PHPUnit and writing test using this framework, then I suggest reading about it more from the [PHPUnit website](https://phpunit.de/manual/current/en/writing-tests-for-phpunit.html).

## Running ready-made Booking application

This readymade application is located [here](https://gitlab.com/i-sepp/bcs-koolitus/) and you can clone it using git to your machine. <https://gitlab.com/i-sepp/bcs-koolitus/>

This chapter has some more detailed summarizing instructions how to run this application that has been developed as an example. This is also referenced in the README.md of the project.

1. Firstly, your machine has to have [installed PHP](http://php.net/manual/en/install.php).

NB You need php7 if you are using another person’s project that requires php7 for the php packages. If you don’t have php7 package in your operation system, then you can add it following the commands from [here](https://www.digitalocean.com/community/tutorials/how-to-upgrade-to-php-7-on-ubuntu-14-04).

 sudo apt-get install php7.0

sudo apt-get install php7.0-xml

1. Your machine needs to have installed [SQLite](#_1jlao46)

 sudo apt-get install php7.0-sqlite3

1. Your machine needs to have [Composer](https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx).

To install composer follow the commands listed [here](https://getcomposer.org/download/)

To make composer globally useable, then use this command listed [here](https://getcomposer.org/doc/00-intro.md#globally).

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

1. You need Git (optional)

 You need it if:

You download the project with Git

You want to develop it further by using Git)

And if you want to use Git you need a GitLab account and SSH-key in your computer.

Installing git:

sudo apt-get install git

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

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

See the chapter about [Git](#_43ky6rz), [GitLab](#_2iq8gzs) and [SSH-key](#_xvir7l).

[Generating SSH-key](https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/) in your computer.

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

cat ~/.ssh/id\_rsa.pub

Copy the output as a SSH-key in your gitlab user configuration.

1. You need to download the project from <https://gitlab.com/i-sepp/bcs-koolitus> .

 git clone git@gitlab.com:i-sepp/bcs-koolitus.git

1. In the folder where it is located on your computer, you need to run:

 composer install

It installs all the third-party code.

And you need to create a folder for the database file:

mkdir database

1. You can [run the command-line web server](#_3hv69ve) from the project folder:

 php -S localhost:8000 -t web

1. Go to your web browser and open URL: [http://localhost:8000/bookings/create](http://localhost:8000/hello)
2. PHPUnit testing: vendor/bin/phpunit tests

Example application locates here - <https://gitlab.com/i-sepp/bcs-koolitus>

## Concluding notes

Remember that this is only one example that should give the initial main encounter with PHP and PHP framework Silex. This task can be solved in many other ways using different tools and still be a good example how to use PHP framework to create a simple application. In addition to coding the application, there are instructions to write sustainable and easily readable code. In addition, every now and then there are hits to pay attention to documentation and testing while you are coding. These things should not be left unnoticed, as documentation and testing are rather a part of coding than a separate step after or before other actions.