Tutorial

(Tuto/Tutorial)

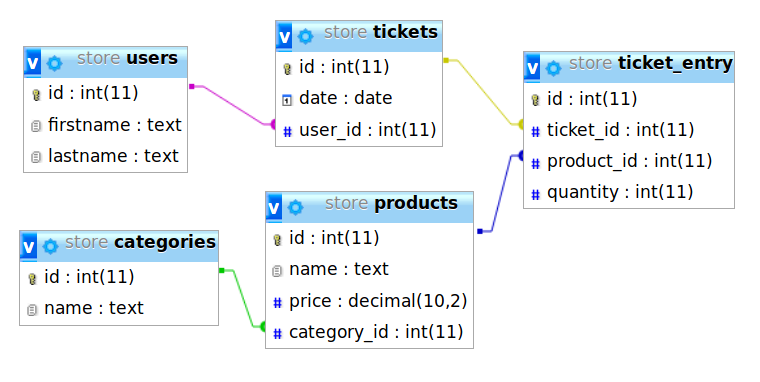


|  |  |
| --- | --- |
| Internship Bordeaux 2019 | Cel Pynenborg Ethan Van den Bleeken |

# Introduction and Set up

In order to tackle the assignment given to us by M. Felix, we were asked to familiarize ourselves with the Symfony Framework.

We made a little webapp based around this database structure:



We received some websites from M. Felix where there are some tutorials written in French.

* <https://gregwar.com/php/tds/td4.html>
* <http://info-timide.iut.u-bordeaux.fr/web/frameworks.html>

We chose to not follow these courses, since we might not understand every detail of the French text. Instead, we opted to follow the [Symfony documentation](https://symfony.com/doc/current/index.html#gsc.tab=0).

This documentation is very detailed and gives us a step-by-step guide in how to set up a project, how to connect to our database, how the templating or routing works in Symfony et cetera.

For our setup, we used [WAMPSERVER](http://www.wampserver.com/en/) as it has everything we needed: a web server (Apache), php (we use version 7.1.x, since 7.2 or higher is not supported by Symfony at the time of writing) and a MySQL database. We soon came to realize that Symfony doesn’t work out of the box with Apache, but there is a handy alternative: Composer.

[Composer](https://getcomposer.org) is a package manager that is used by modern php frameworks. You can use it to take care of dependencies, much in the same way NPM works for React or Angular for example.

Composer can install Symfony for you with just one command and also has a web server package that you can use instead of Apache.

Use these commands to get started with your first Symfony project:

$ composer create-project symfony/skeleton project-name

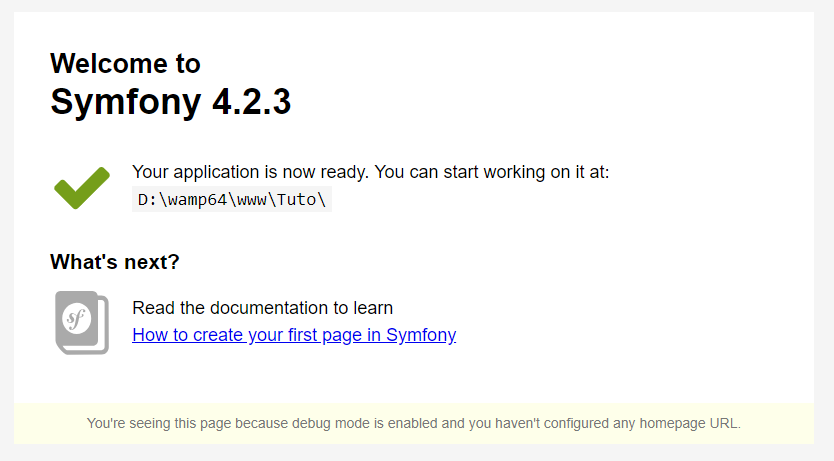
OPTIONAL: Install the Symfony web server (we recommend this):

$ cd project-name  
 $ composer require symfony/web-server-bundle --dev

In order to start the web server, use this command in the directory of the project:

$ php bin/console server:run

This web server will listen on port 8000 by default. For now, you can go to <http://localhost:8000> to see the default “you don’t have a homepage configured” page. If you see this:



You have successfully installed Symfony on your system!

Since we are also working on the same project at the same time, we opted to make a private repository on Github in order to do version control.

If you want to read the same information from the official Symfony documentation, you can do that [here](https://symfony.com/doc/current/setup.html).

# Creating your first page and Routing

Symfony uses a Model-View-Controller structure, so in order to present a view you must first make a controller, in which you make a method that serves a twig template (your view) when you navigate to it.

You can navigate to methods by something called “routes”. You configure your own routes so that you can make clean urls that make sense and are consistent.

There are multiple ways to assign routes to a method, all of which are explained in detail in these documentation pages:

This describes page creation and touches a little bit on twig and routing

<https://symfony.com/doc/current/page_creation.html>

This is a more in-depth explanation of how routing works and how you can configure routes

<https://symfony.com/doc/current/routing.html>

We chose to use annotation routes, since it keeps our assignments close to our programming logic, and it’s just plain cool too. (It’s also the recommended way to configure your routes)

**!! Don’t forget to install the annotation routing package by using composer !!**

You can do this with the following command:

$ composer require annotations

# Controllers and templating

Next up, you should learn the power of controllers. There’s not much of value we can add here, so you should just read the docs and experiment a little bit with the given examples.

<https://symfony.com/doc/current/controller.html>

What we can give you some extra information on, is Twig. Twig is a templating language developed by the people over at Symfony to replace php templates. Twig is awesome, it allows you to write base templates that can be implemented in child templates, is really fast to develop in, doesn’t require a lot of code to serve html, … it’s basically a web designer’s dream environment.

Check out templating here:

<https://symfony.com/doc/current/templating.html>

# Using a database with Symfony (Doctrine)

Since we’re creating a webapp, at some point we will want to query data from our database and present it to our users. There is no way to natively connect Symfony to a database, for that you’ll need something called “Doctrine”.

Remember composer, the handy dandy package manager? Let’s use it to install Doctrine in mere seconds!

$ composer require symfony/orm-pack  
 $ composer require symfony/maker-bundle --dev

Now that we have Doctrine, we need to set up our database connection string. This basically tells doctrine where it has to look for our data and which credentials it should use to connect to said database.

The connection string is located in a file named “.env”. You can change the string there like it says in the documentation, and everything will work flawlessly. But…

Remember when we talked about Github? Imagine the following scenario:

The credentials for my database are “user=root”, “password=root” and my database is hosted on port 3306.

Fairly standard, right?

Well what if Ethan’s database is hosted on port 3305? Or his user or password differs from mine? We’d have to change our credentials every time we pull the other’s changes. That in itself is pretty annoying, but here’s another question: Do you really want to publish your database connection string on Github?

Luckily, Symfony has a few handy .gitignore files preconfigured. What you can do now then, is copy the default contents of .env, paste them in a new file called .env.local, fill in the connection string, and you’ll never have to worry about conflicts again! (Or people messing with a public database for that matter)

You can now get started with Doctrine! Check out the documentation here:

<https://symfony.com/doc/current/doctrine.html>

Don’t forget to make Entities by using the command line! It makes making Entities a breeze and saves you a lot of repetitive programming work.

# Forms

In order to create or edit data via user input, we’ll need to add some way of receiving input from our users. We can achieve this by using forms.

Again, composer is used to install the forms package for this project, so let’s install it:

$ composer require symfony/form

Right now, there’s not a lot more that we can tell you about forms than what the documentation presents, so go check that out here:

<https://symfony.com/doc/current/forms.html>