# Exercise: Blog - PHP and Symfony – Part I

This document defines a complete walkthrough of creating a **Blog** application with the [Symfony](https://symfony.com/) Framework, from setting up the framework through the [authentication](http://symfony.com/doc/current/security.html) module, to creating a **CRUD** around [Doctrine](http://www.doctrine-project.org/) entities.

Make sure you have installed [XAMPP](https://www.apachefriends.org/download.html), [HeidiSQL](http://www.heidisql.com/download.php) and added [PHP root folder to the path environment variable](http://php.net/manual/en/faq.installation.php#faq.installation.addtopath).

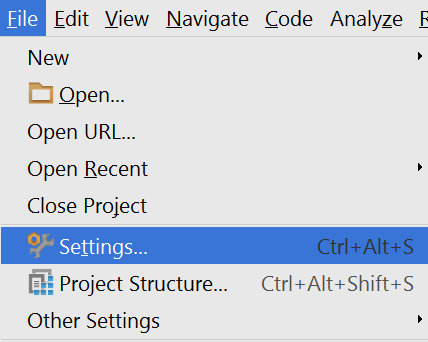
# Set Up Symfony Project

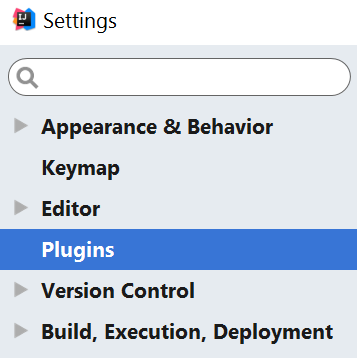
Symfony framework comes with various ways of creating a project, all of them involving the [presence of Symfony project](https://symfony.com/download). The most convenient way is to **create a project via your IDE**. Luckily there are several **plugins** for **PHPStorm** (and the other **IDEA**-based IDE’s) which help developing application with Symfony

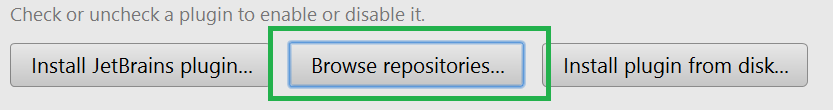
## Install Symfony-related Plugins

Before we start working on our project, we can make our life easier by **installing** a couple of related **plugins**:

* Go to **[File]** 🡺 **[Settings]** 🡺 **[Plugins] 🡺 [Browse repositories]**:

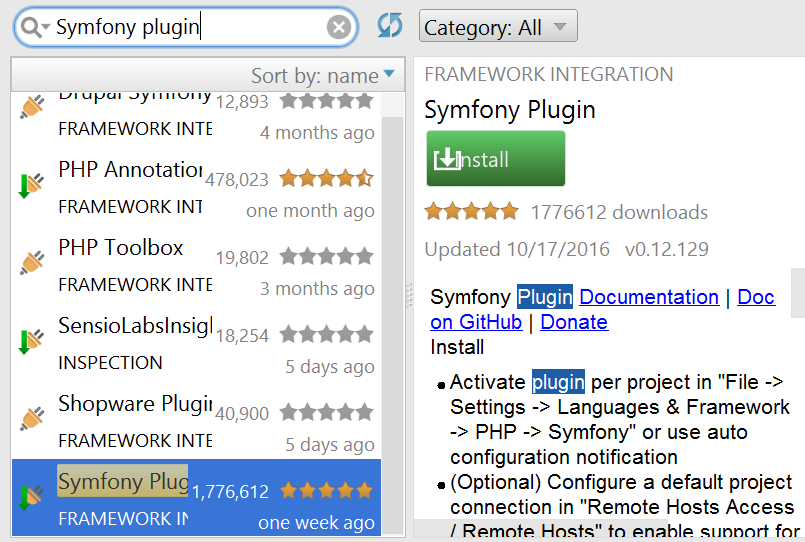




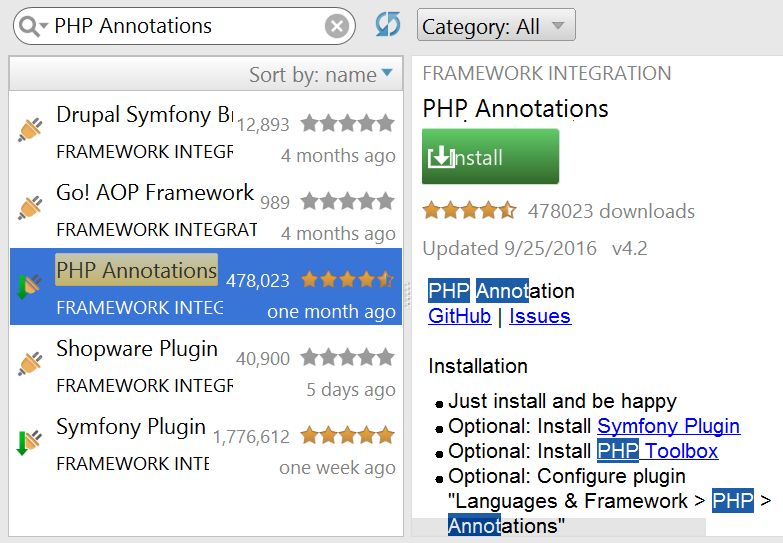


We need to install the following plugins:

1. Symfony Plugin



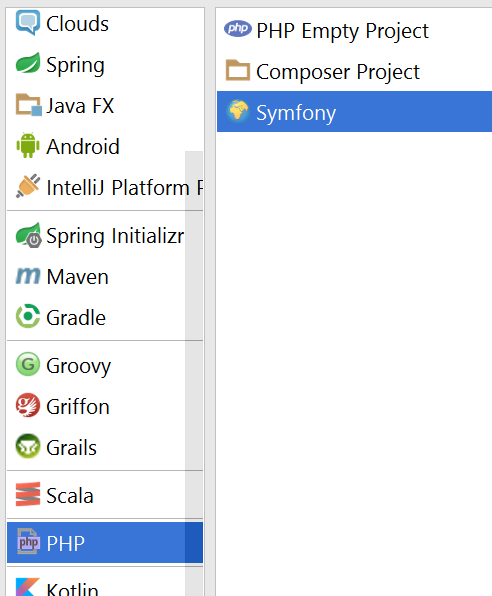
1. PHP Annotations

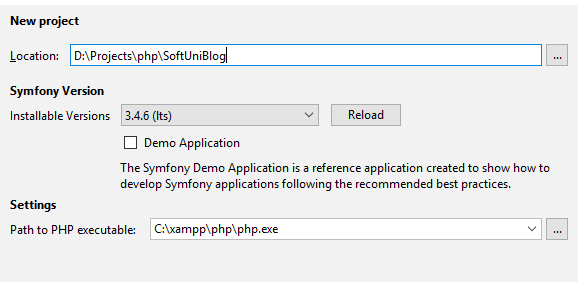


## Create Symfony Project from IDE

Once you have installed the plugins and restarted the **IDE**, you will have either a **PHP subcategory** (IntelliJ) or directly a **Symfony** one (PHPStorm) in the **Create Project** context menu:



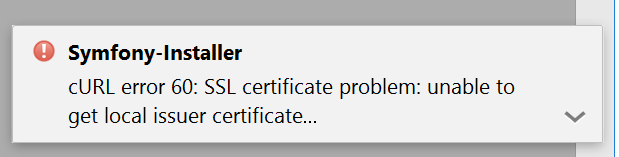




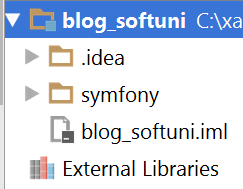
We need to specify the **php executable**, which most probably resides in **c:/xampp/php**

## Check Project Status

If you have received the following error:



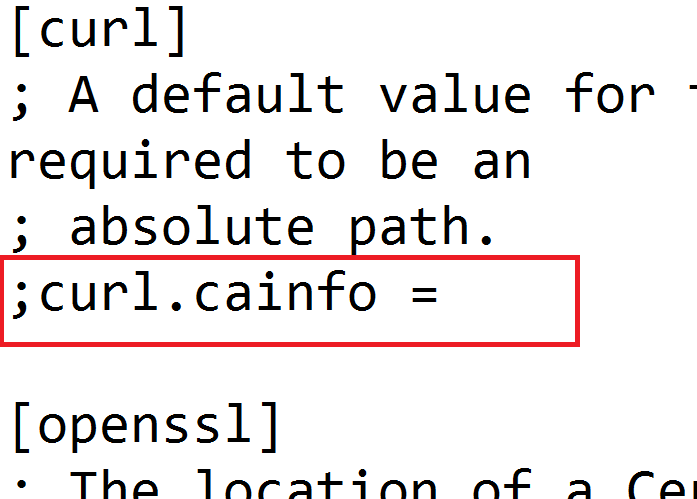
And your project looks like this:



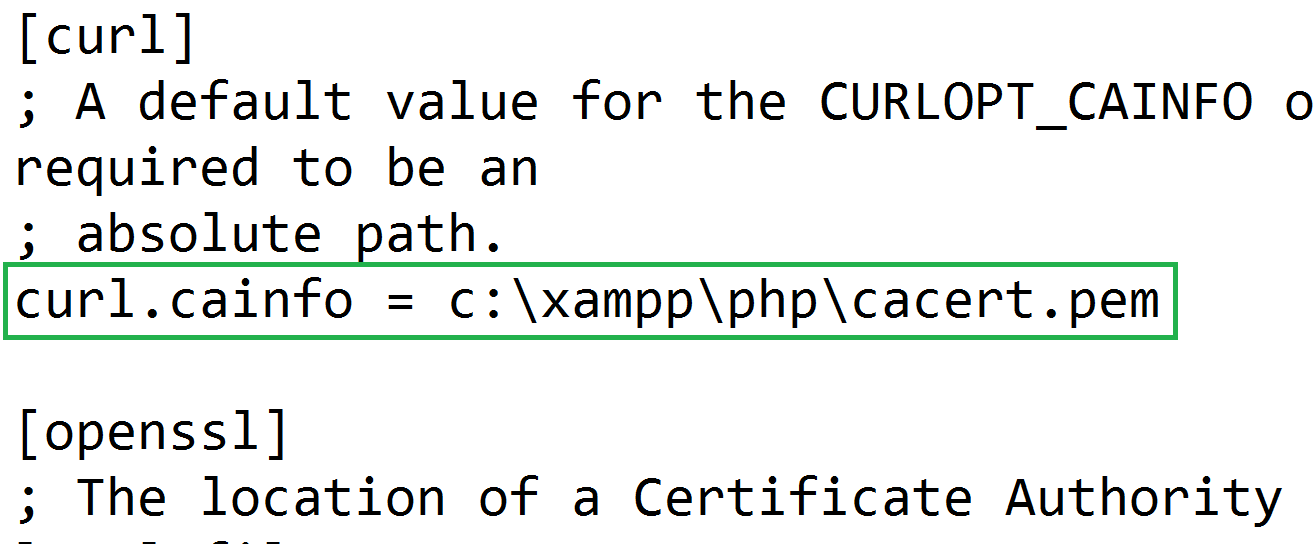
You most probably haven’t created the project properly. This could of possible missing curl.cainfo directive in **php.ini**.

Follow these [instructions](http://stackoverflow.com/questions/37997669/curl-error-60-ssl-certification-issue-when-attempting-to-use-symfony) **ONLY IF YOU HAVE RECEIVED THE ERROR ABOVE, OTHERWISE SKIP THIS STEP.**

1. Save this file: <https://curl.haxx.se/ca/cacert.pem> in **c:/xampp/php**
2. Edit the **c:/xampp/php/php.ini** file and find the following line



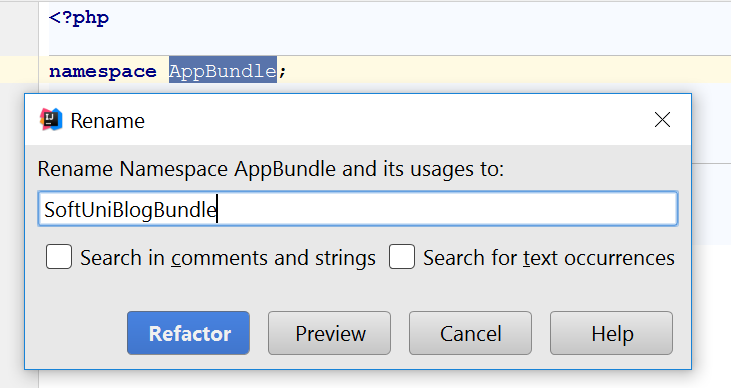
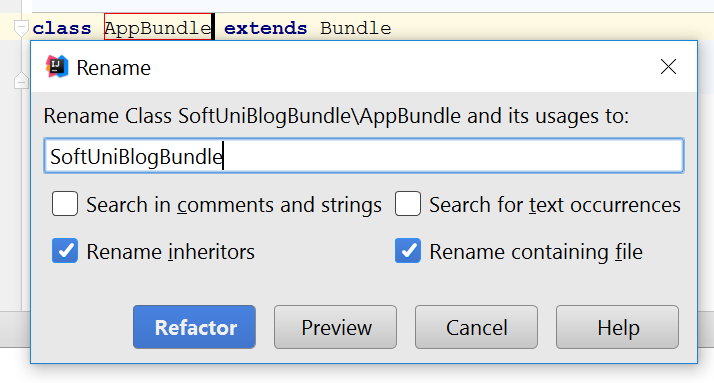
1. And make it: “curl.cainfo = c:\xampp\php\cacert.pem”



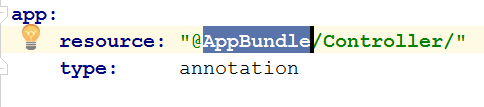
1. Create the project again

## Rename Default Bundle

The Default bundle located in src folder is called AppBundle. Rename with the following occurrences to SoftUniBlogBundle, using **[Shift+F6]**:

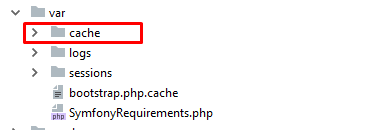
1. src/AppBundle folder
2. src/AppBundle/AppBundle.php
3. The namespace directive in src/AppBundle/AppBundle.php  
   
4. The classname in src/AppBundle/AppBundle.php  
   

Change the occurrence in app/config/routing.yml to SoftUniBlogBundle too:



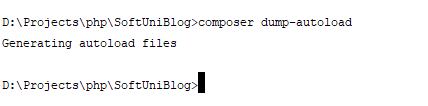
Change the occurrence in **composer.json** to SoftUniBlogBundle too.

Delete the cache in var folder:

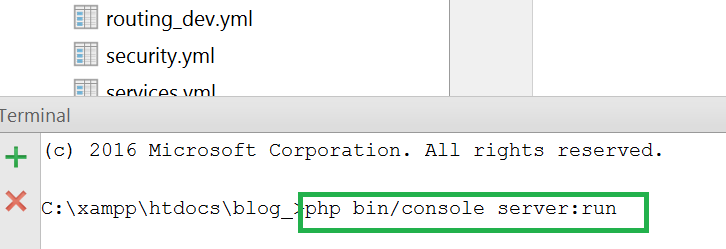


Run the following format in the console:

composer dump-autoload



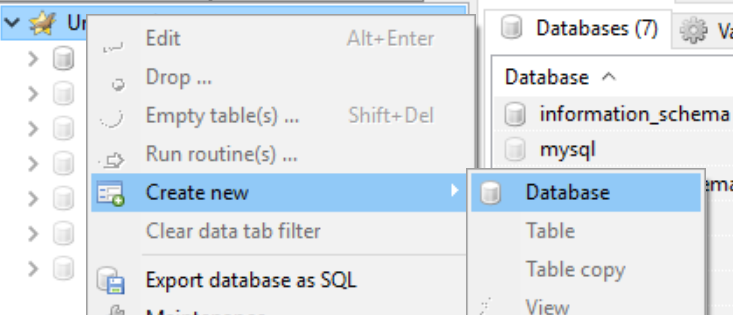
Start the server by running the following command in the project folder



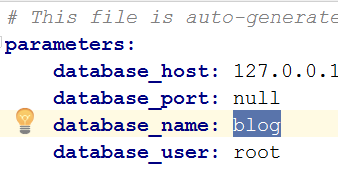
After that, you can see the result at <http://localhost:8000> ☺

## Create Database

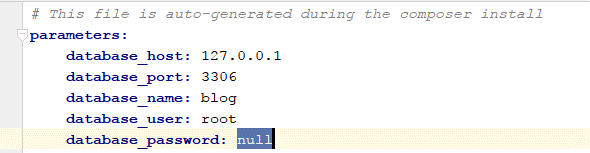
Open HeidiSQL, connect to the MySQL instance and create a database named “blog”



And change the database name in app/config/parameters.yml to “blog”



*Note: you also need to specify your* ***MySQL*** *database* ***root user password****:*



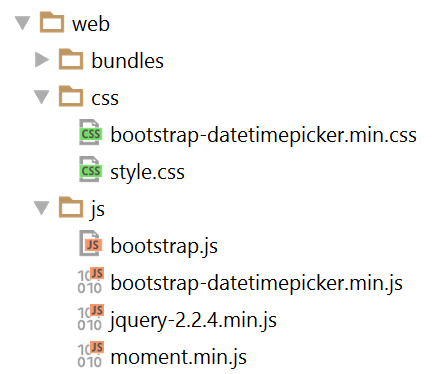
## Setup Layout

We will need a base layout for all our templates. As we are using **Bootstrap**, we will need its css included in all pages, and the related scripts too. In addition, we will need:

1. [Bootstrap Date Time picker](http://www.malot.fr/bootstrap-datetimepicker/) for choosing dates in our forms
2. [Moment JS](http://momentjs.com/) for validating dates

All our styles and scripts we need to include to our project. Create two folders in the “web” folder called “css” and “js” respectively. In the **blog design skeleton** in the folder scripts you can find the jquery and bootstrap files.

Place the needed scripts and styles there, ending up with the following structure:



Then we need to use this styles and script setting up a base layout in app/resources/views/base.html.twig.

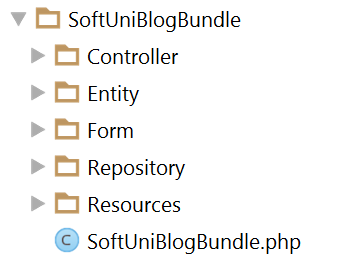
Setup a base layout as you wish or use the following one:

|  |
| --- |
| *{#  This is the base template used as the application layout which contains the  common elements and decorates all the other templates.  See http://symfony.com/doc/current/book/templating.html#template-inheritance-and-layouts #}* <!DOCTYPE **html**> <**html lang="en-US"**> <**head**>  <**meta charset="UTF-8"**/>  <**meta name="viewport" content="width=device-width, initial-scale=1"**/>  <**title**>{% **block** title %}SoftUni Blog{% **endblock** %}</**title**>  {% **block** stylesheets %}  <**link rel="stylesheet" href="**{{ asset(**'css/style.css'**) }}**"**>  <**link rel="stylesheet" href="**{{ asset(**'css/bootstrap-datetimepicker.min.css'**) }}**"**>  {% **endblock** %}  <**link rel="icon" type="image/x-icon" href="**{{ asset(**'favicon.ico'**) }}**"**/> </**head**>  <**body id="**{% **block** body\_id %}{% **endblock** %}**"**>  {% **block** header %}  <**header**>  <**div class="navbar navbar-default navbar-static-top" role="navigation"**>  <**div class="container"**>  <**div class="navbar-header"**>  *{#<a href="{{ path('blog\_index') }}" class="navbar-brand">SOFTUNI BLOG</a>#}* <**button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse"**>  <**span class="icon-bar"**></**span**>  <**span class="icon-bar"**></**span**>  <**span class="icon-bar"**></**span**>  </**button**>  </**div**>  <**div class="navbar-collapse collapse"**>  <**ul class="nav navbar-nav navbar-right"**>  *{#{% if app.user %}#}  {#<li>#}  {#<a href="{{ path('user\_profile') }}" class="navbar-brand">#}  {#My Profile#}  {#</a>#}  {#</li>#}  {#<li>#}  {#<a href="{{ path('article\_create') }}" class="navbar-brand">#}  {#Create Article#}  {#</a>#}  {#</li>#}  {#<li>#}  {#<a href="{{ path('security\_logout') }}" class="navbar-brand">#}  {#Logout#}  {#</a>#}  {#</li>#}  {#{% else %}#}  {#<li>#}  {#<a href="{{ path('user\_register') }}">#}  {#REGISTER#}  {#</a>#}  {#</li>#}  {#<li>#}  {#<a href="{{ path('security\_login') }}">#}  {#LOGIN#}  {#</a>#}  {#</li>#}  {#{% endif %}#}* </**ul**>  </**div**>  </**div**>  </**div**>  </**header**> {% **endblock** %}  <**div class="container body-container"**>  {% **block** body %}  <**div class="row"**>  <**div id="main" class="col-sm-9"**>  {% **block** main %}{% **endblock** %}  </**div**>  </**div**>  {% **endblock** %} </**div**>  {% **block** footer %}  <**footer**>  <**div class="container modal-footer"**>  <**p**>**&copy;** 2018 - Software University Foundation</**p**>  </**div**>  </**footer**> {% **endblock** %}  {% **block** javascripts %}  <**script src="**{{ asset(**'js/jquery-3.2.1.min.js'**) }}**"**></**script**>  <**script src="**{{ asset(**'js/moment.min.js'**) }}**"**></**script**>  <**script src="**{{ asset(**'js/bootstrap.js'**) }}**"**></**script**>  <**script src="**{{ asset(**'js/bootstrap-datetimepicker.min.js'**) }}**"**></**script**> {% **endblock** %}  </**body**> </**html**> |

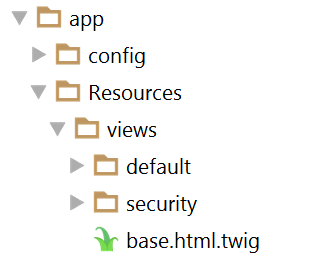
# Symfony Base Project Overview

Symfony is a modular enterprise web-framework, which comes with a solid vendor support, **bundle** system, **enterprise** mechanisms and is most-suited for **MVC** architecture.

Initially the project comes with a main [bundle](http://symfony.com/doc/current/bundles.html), which can be treated as a plugin later. A **bundle** often has **Controllers**, **Entities** and related components (e.g. **Repositories**, **Forms**, **Commands**…)



Standard templates (**views**) reside in the application folder (app) and are usually separated in a folder named after the **controller names**.



The de-facto standard **View Engine** in Symfony is [Twig](http://twig.sensiolabs.org/).

The base **configuration** of the project is placed in app/config, where configuration files for the [Doctrine](http://www.doctrine-project.org/) connection are defined among [Security](http://symfony.com/doc/current/security.html) management, [Routing](http://symfony.com/doc/current/routing.html) rules, registering [Services](http://symfony.com/doc/current/service_container.html) and so forth.



It's very important that the parameters.yml.dist file contains the **same** keys as the ones in parameters.yml, since installing a new bundle will **delete** **unused pairs**.

# User Authentication

Symfony has very powerful **security** management system, where the common work for checking user **permissions and dispatching the request** is well abstracted, yet the configuration could be confusing. In the walkthrough below, we will setup a **registration and login process** and accessing **secured** content.

## Creating User Entity

Our users should be stored in the database. This means we need a “users” table. Since tables are represented as objects in the **Object/Relation Mapping** paradigm, we need to create an **object, which represents that table**. The **classes** (**objects**) which represent tables are called **Models** and **Entities**.

In the de-facto, standard **ORM** in Symfony, called **Doctrine**, these objects are called **Entities.**

Let’s define our rules for a user:

* Should have a **unique** login name, let’s say email
* Should have a password
* Should have a full name, let’s say fullName

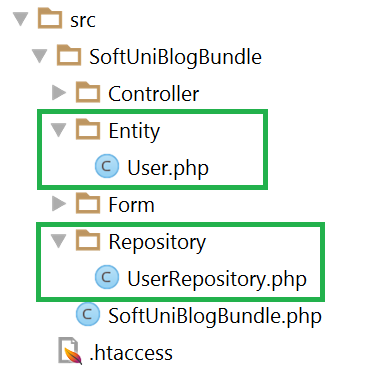
Doctrine comes with a [handy console tool](http://symfony.com/doc/current/doctrine/console.html) for managing the database and creating entities. Let’s use Doctrine to create an entity called User, using the entity generation wizard. To do this, we need to open a terminal window in the project root directory and type the following command:

|  |
| --- |
| **php bin/console doctrine:generate:entit**y |

This will prompt us to enter an entity name. Entities are prefixed with the bundle they should belong to. Our bundle is called SoftUniBlogBundle (the default name is AppBundle), so we’ll type in SoftUniBlogBundle:User (or AppBundle:User, if your bundle is called AppBundle).

Afterwards it will prompt us for the properties (fields) of the User object. As we have said above, it will have an email, password and a fullName, all of them are text fields (strings). The email should be unique, so when you are prompted for uniqueness there, type “true” instead of just clicking enter (which defaults to false)

When the last field (fullName) is created and you are prompted for another one, just click enter to exit the wizard. This will create the User entity and its corresponding UserRepository.



## Setting Up Security Configuration

As we have said, Symfony comes with a couple of configuration files, one of which is called security.yml. We need to specify a few things, such as:

* How the password will be **encrypted** and on **which entity**
* **Which** entity will be used for **users** and which of its **fields** will be the **username field** (e.g. **email**, **username**, etc.)
* **Where the login form will be located** (route name)
* Where this **login form will post to**

Below is a security.yml file, which has the following configuration:

* The bundle is called SoftUniBlogBundle
* The **user** entity is called User, and its username field is called email
* The login form will be accessed and posted to “security\_login”
* After a successful login, the user will be redirected to “blog\_index”

|  |
| --- |
| **security:  encoders:** *# Our user class and the algorithm we'll use to encode passwords  # http://symfony.com/doc/current/book/security.html#encoding-the-user-s-password* **SoftUniBlogBundle\Entity\User:** bcrypt   **providers:** *# in this example, users are stored via Doctrine in the database  # To see the users at src/AppBundle/DataFixtures/ORM/LoadFixtures.php  # To load users from somewhere else: http://symfony.com/doc/current/cookbook/security/custom\_provider.html* **database\_users:  entity:** { **class:** SoftUniBlogBundle:User, **property:** email }   *# http://symfony.com/doc/current/book/security.html#firewalls-authentication* **firewalls:  secured\_area:** *# this firewall applies to all URLs* **pattern:** ^/   *# but the firewall does not require login on every page  # denying access is done in access\_control or in your controllers* **anonymous:** true   *# This allows the user to login by submitting a username and password  # Reference: http://symfony.com/doc/current/cookbook/security/form\_login\_setup.html* **form\_login:** *# The route name that the login form submits to* **check\_path:** security\_login  *# The name of the route where the login form lives  # When the user tries to access a protected page, they are redirected here* **login\_path:** security\_login  *# Secure the login form against CSRF  # Reference: http://symfony.com/doc/current/cookbook/security/csrf\_in\_login\_form.html* **csrf\_token\_generator:** security.csrf.token\_manager   **logout:** *# The route name the user can go to in order to logout* **path:** security\_logout  *# The name of the route to redirect to after logging out* **target:** blog\_index    **access\_control:** *# this is a catch-all for the admin area  # additional security lives in the controllers # - { path: '^/(%locale%)/admin', roles: ROLE\_ADMIN }* |

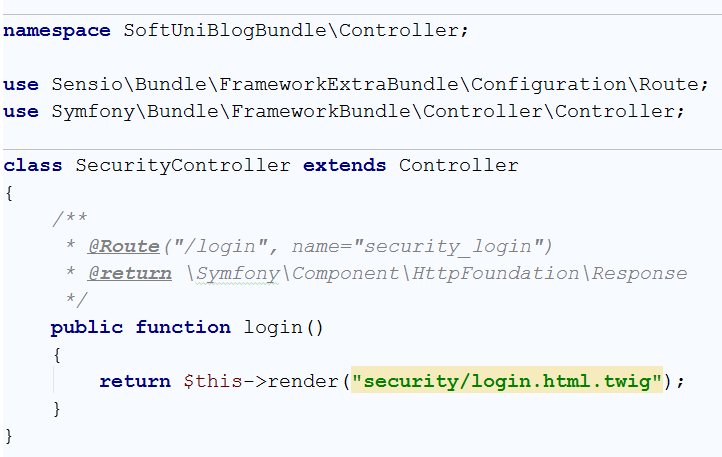
## Login Form

To create a login form, we need to create a so-called Controller which will **listen on** this **route** (which above we called “security\_login”) and render the View with the login form when someone goes to the /login route.

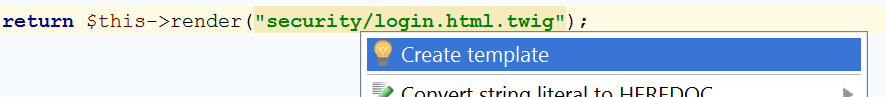
Let’s call our Controller “SecurityController“:



Then we need a method (which we will call “login()”), which listens on the “/login” **route** and renders a view (let’s point it to a login.html.twig file, which resides in the security **folder**)



The yellow background color in the view name tells us we don’t have that view yet. We could easily create it by clicking [Alt+Enter] ☺



Before messing with any layouts (which we have setup and will use in the next chapters) we will just create a simple login form with no styles.

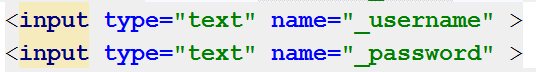
We need to define a <form> tag, which is posting to the security\_login route. **Twig**, fortunately, provides a function path() that uses route names and generates URLs from them



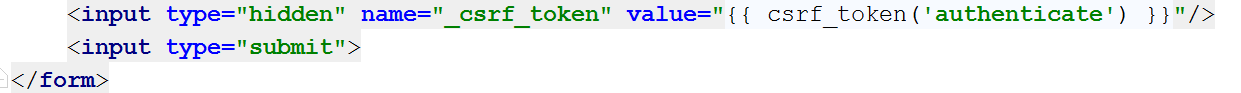
The form is named “**authenticate**” because we will use this name later to generate a [CSRF Token](https://en.wikipedia.org/wiki/Cross-site_request_forgery)

Symfony security requires the **username** (which is **email** in our case) and **password** fields to be named respectively \_username and \_password

We need to define these two text fields (or **password** field for the password type ☺)

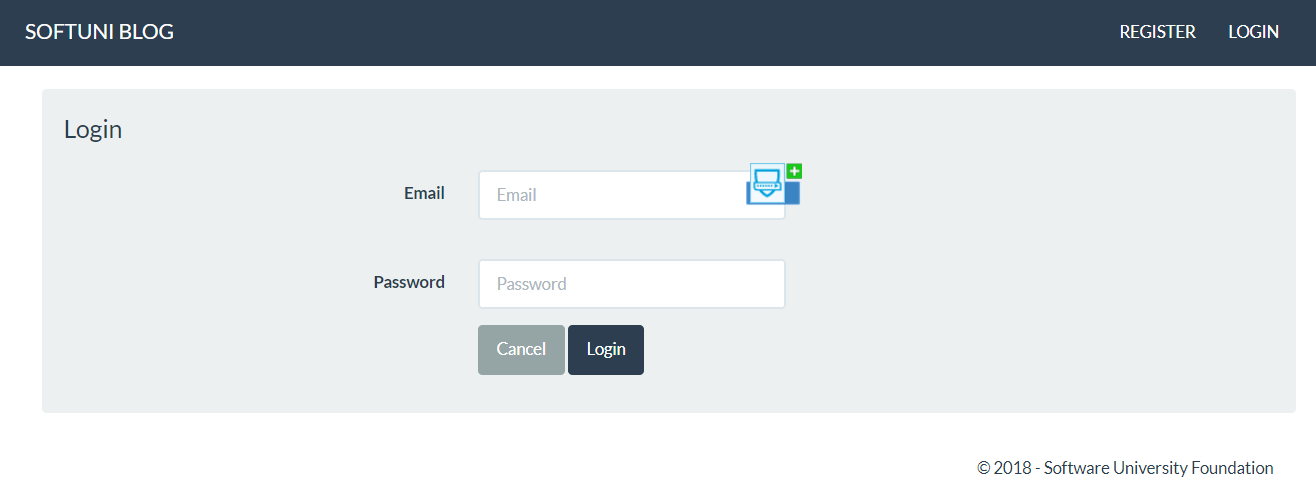


And a field for the CSRF Token using the Twig’s helper method csrf\_token() which accepts the form name.



|  |
| --- |
| {% **extends 'base.html.twig'** %}  {% **block** body\_id **'login'** %}  {% **block** main %}   <**div class="container body-content span=8 offset=2"**>  <**div class="well"**>  <**form name="authenticate" class="form-horizontal" action="**{{ path(**'security\_login'**) }}**" method="post"**>  <**fieldset**>  <**legend**>Login</**legend**>  <**div class="form-group"**>  <**label class="col-sm-4 control-label" for="user\_email"**>Email</**label**>  <**div class="col-sm-4 "**>  <**input type="email" class="form-control" id="user\_email" placeholder="Email" name="\_username"**>  </**div**>  </**div**>  <**div class="form-group"**>  <**label class="col-sm-4 control-label" for="password"**>Password</**label**>  <**div class="col-sm-4"**>  <**input type="password" class="form-control" id="password" placeholder="Password" name="\_password"**>  </**div**>  </**div**>  <**input type="hidden" name="\_csrf\_token" value="**{{ csrf\_token(**'authenticate'**) }}**"**/>  <**div class="form-group"**>  <**div class="col-sm-4 col-sm-offset-4"**>  *{#<a class="btn btn-default" href="{{ path('blog\_index') }}">Cancel</a>#}*  <**button type="submit" class="btn btn-primary"**>Login</**button**>  </**div**>  </**div**>   </**fieldset**>  </**form**>  </**div**>  </**div**> {% **endblock** %} |

Now opening <http://localhost:8000/login> should render this login form

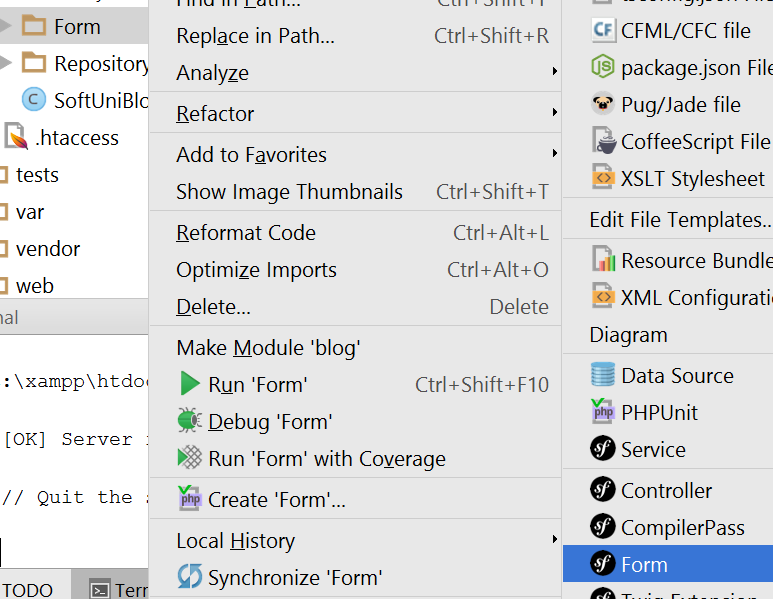


## Register Form

What is a login form without users – nothing. In order to have users, we need a registration form. By analogy, open the already generated DefaultController or create a new one (e.g. UsersController) and an action that listens on “register”.

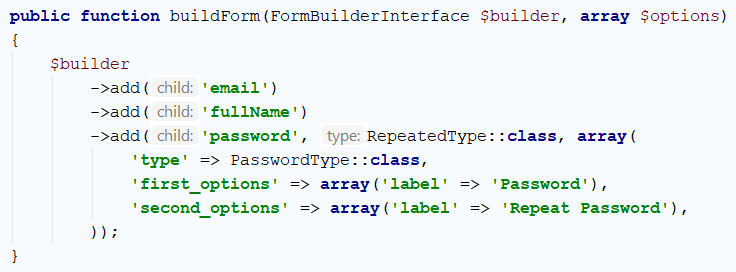
It will render the form the same way, but also needs to handle this form.

In order for a form to work with an entity, it needs a corresponding [FormType](http://symfony.com/doc/current/form/data_transformers.html). Before we can continue creating the register action, we need to create a Form Type. Create a folder “**Form**” in src/SoftUniBlogBundle. Then create a **Form Type** as follows:



Let’s call it UserType.

In the scaffold method “buildForm()” we need to the define pairs – the entity fields and their corresponding types in the form. Email and FullName are text types, so we will use a TextType from the Symfony\Component\Form\Extension\Core\Type\TextType namespace. Password is RepeatedType, so we will use Symfony\Component\Form\Extension\Core\Type\RepeatedType.



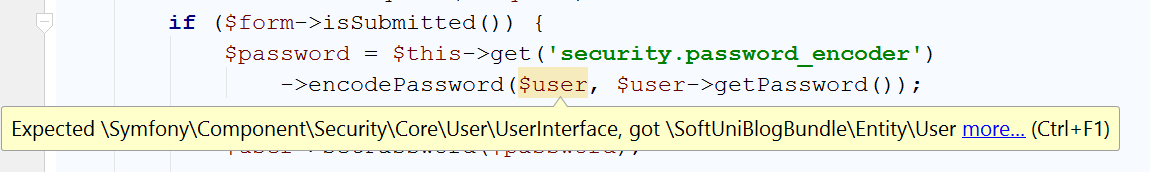
Going back to the controller’s registration method we can now create a form of UserType.



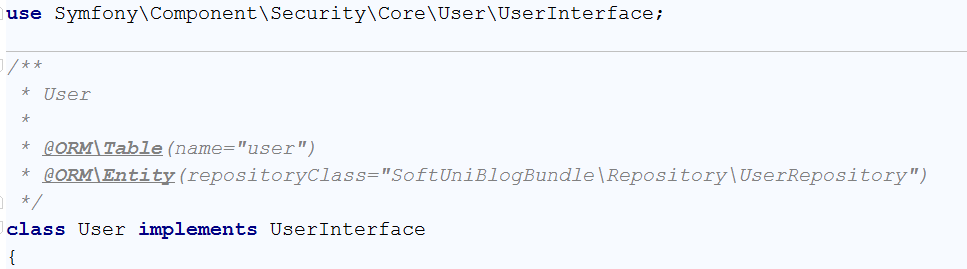
We have said here: Create a form of user type and after it’s submitted fill the $user object.

Then we need to tell the method – once the form is **submitted** and **all** the validations are **passed** (e.g. texts are filled), **save** the user entity in the **database**.

There’s one possible problem – the password will go **plain** into the DB. Luckily, in the security configuration we have registered an encryption provider, so we can use this provider to encode the password and then send it to the database



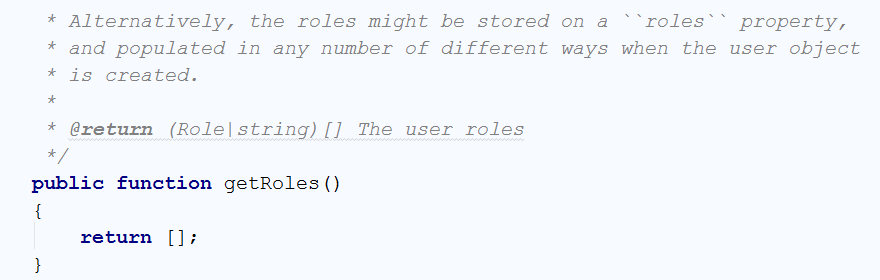
The encoder only works on **UserInterface** objects and our users is not one. What we need is to go to the User entity and make it implements the **UserInterface** interface.



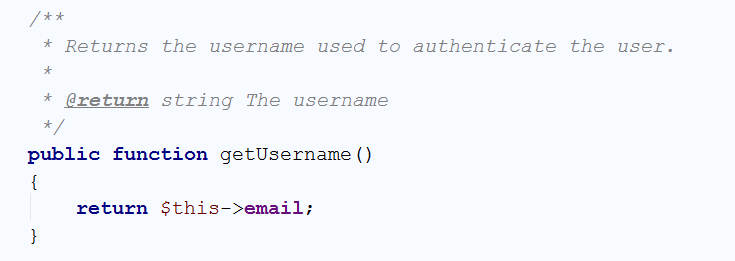
Then implement all of the missing methods with [ALT+ENTER].

You can leave most of the blank (auto-generated), but some of them should be filled.

The first method is getRoles(). It should return an array of roles (could be empty), but not null:



The other one is the getUsername() method, which will be used for authentication. We need to return our $email field in it, because that’s our username:



Now going back to the registration action, the error is gone. We can safely set the encoded password to the user object and persist it via [EntityManager](http://www.doctrine-project.org/api/orm/2.5/class-Doctrine.ORM.EntityManager.html) to the database



Here we have said that when everything is OK with the form, persist the user and redirect them to the login form. If the form is not submitted, then we need only to render the register form ☺

The form itself contains text fields with names corresponding to the object name and the properties as keys (like an associative array) e.g. the email field is called **user[email]:**

|  |
| --- |
| {% **extends 'base.html.twig'** %}  {% **block** body\_id **'register'** %}  {% **block** main %}  <**div class="container body-content span=8 offset=2"**>  <**div class="well"**>  <**form class="form-horizontal" action="**{{ path(**'user\_register'**) }}**" method="post"**>  <**fieldset**>  <**legend**>Register</**legend**>  <**div class="form-group"**>  <**label class="col-sm-4 control-label" for="user\_email"**>Email</**label**>  <**div class="col-sm-4 "**>  <**input class="form-control" id="user\_email" placeholder="Email" name="user[email]" required type="email"**>  </**div**>  </**div**>  <**div class="form-group"**>  <**label class="col-sm-4 control-label" for="user\_fullName"**>Full Name</**label**>  <**div class="col-sm-4 "**>  <**input type="text" class="form-control" id="user\_fullName" placeholder="Full Name" name="user[fullName]" required**>  </**div**>  </**div**>  <**div class="form-group"**>  <**label class="col-sm-4 control-label" for="user\_password\_first"**>Password</**label**>  <**div class="col-sm-4"**>  <**input type="password" class="form-control" id="user\_password\_first" placeholder="Password" name="user[password][first]" required**>  </**div**>  </**div**>  <**div class="form-group"**>  <**label class="col-sm-4 control-label" for="user\_password\_second"**>Confirm Password</**label**>  <**div class="col-sm-4"**>  <**input type="password" class="form-control" id="user\_password\_second" placeholder="Password" name="user[password][second]" required**>  </**div**>  </**div**>  <**div class="form-group"**>  <**div class="col-sm-4 col-sm-offset-4"**>  *{#<a class="btn btn-default" href="{{ path('blog\_index') }}">Cancel</a>#}*  <**button type="submit" class="btn btn-primary"**>Submit</**button**>  </**div**>  </**div**>   {{ form\_row(form.\_token) }}  </**fieldset**>  </**form**>  </**div**>  </**div**> {% **endblock** %} |

Open <http://localhost:8000/register> and test it:

