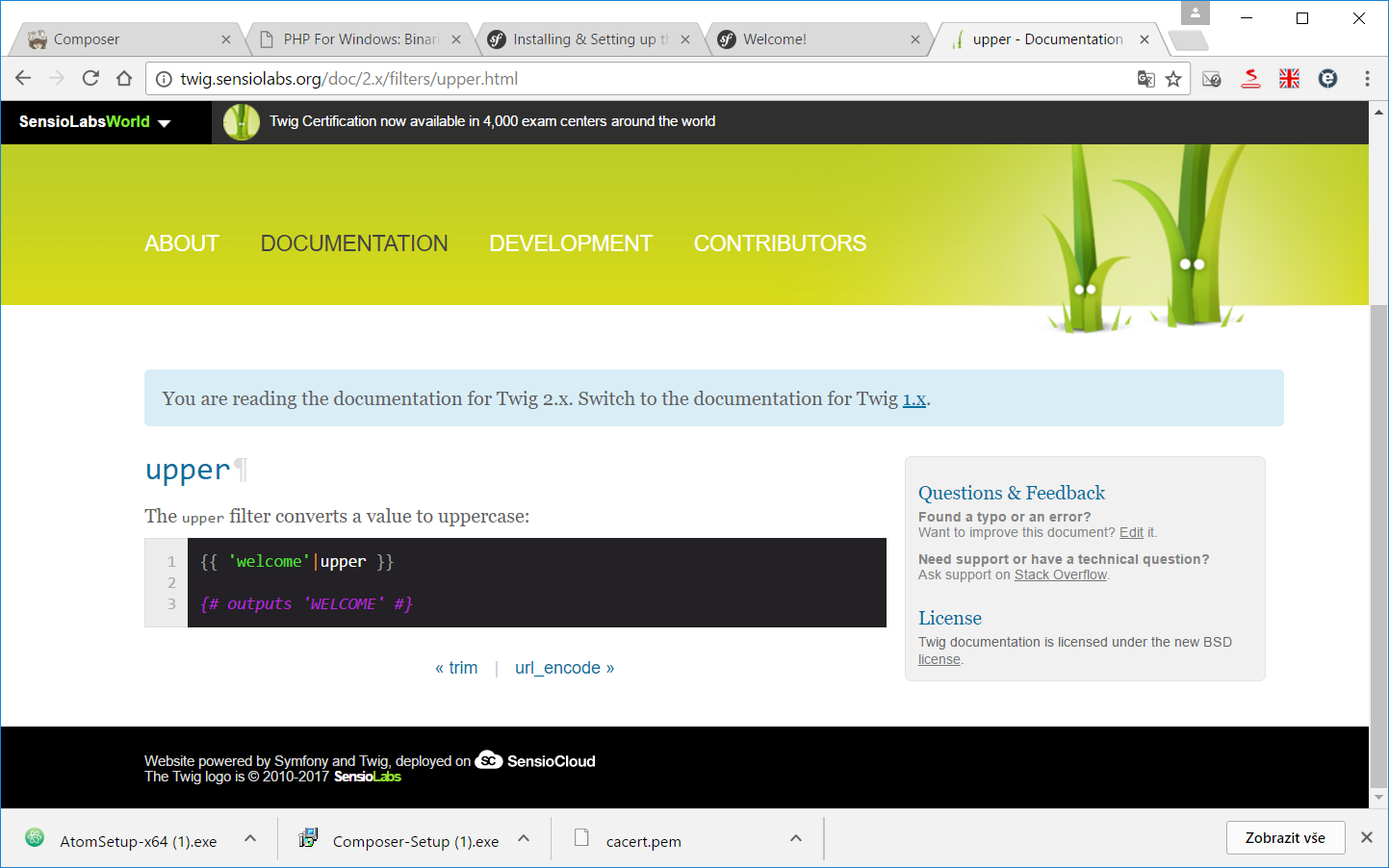
# Dokumentace ke twigu

<http://twig.sensiolabs.org/>

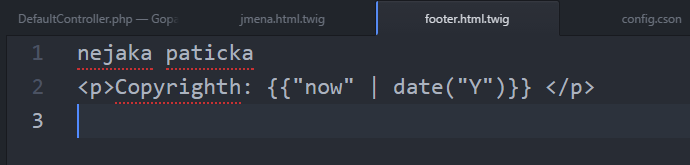
Filtr

Vezme například jmeno, může něco odfiltrovat atd

Upper



# Výpis aktuálního roku



Filtr se dá řetězit

{{jmeno | upper | ...}}

# CRUD

### **Cíl lekce**

Cílem této lekce je seznámení se s Doctrine a vytvoření vlastní entity a rozhraní CRUD pomocí Doctrine.

### **Teoretická část**

#### Doctrine

Doctrine je PHP knihovna pro práci s databází. Doctrine je tzv. ORM (object relational mapping) což znamená, že pomocí ní můžeme mapovat objekty na relační databázi (Doctrine umí i naopak). Toto znamená, že v podstatě nemusíme řešit, že máme nějakou databázi, ale pracujeme pouze s objekty. Doctrine má pro dotazy dokonce svůj vlastní jazyk DQL (Doctrine Query Language). Tento jazyk je velice podobný jazyku SQL (Select Query Language). Objekty se kterými Doctrine pracuje se nazývají entity a ty nesou data. Tato knihovna navíc použité dotazy optimalizuje a uchovává pro další použití. Tímto postupem může naši aplikaci i vcelku zrychlit.  
Více info o Doctrine [*zde*](https://www.doctrine-project.org/).

#### ORM

ORM je programovací technika, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Tato technika, v některých případech, úplně odstíní programátora od znalosti jazyka SQL. Použití této techniky usnadní zejména provádění běžných databázových operací jako mazání, editování či přidávání dat. Důležitou funkcí ORM je trvalé uchování dat tzn. aby data aplikace, která jsou v operační paměti, se během nečekaného chování aplikace neztratila. Nejdůležitější funkcí ORM je synchronizace mezi objekty, které jsou používané v aplikaci a databázovým systémem, kde jsou uložené.  
Více info o ORM [*zde*](https://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD).

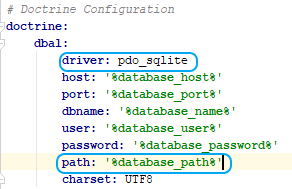
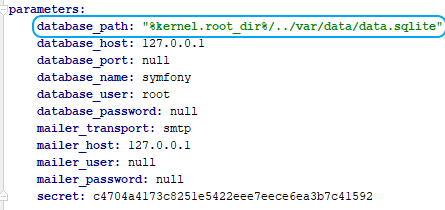
#### CRUD

CRUD je generované rozhraní pro práci s entitou. Jak již název prozrazuje (create, read, update, delete) zahrnuje základní operace nad entitou. Tyto operace jsou vytvoření, zobrazení, změna, a odstranění záznamu v entitě (tabulce). CRUD vygeneruje controller v námi zadaném bundlu nad námi zadanou entitou a také čtyři šablony (pro každou operaci jednu).

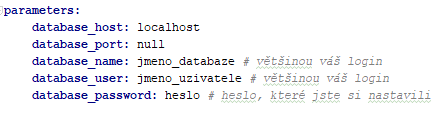
#### Úkoly pro cvičení

Úkoly pro cvičení budou vytvoření vlastní entity o minimálně třech sloupcích. Následně k této entitě vygenerovat rozhraní CRUD.

### **Praktické ukázky**

Jako první věc si ukážeme jak konfigurovat databázi a jak ji vytvořit. Nyní si ukážeme soubory ve kterých se databáze konfiguruje. Jsou to dva soubory a najdeme je ve složce app->config. Jsou to soubory *config.yml* a *parameters.yml*. Jestliže vyvíjíme projekt lokálně a používáme databázi **MySQL** není třeba měnit jakékoli nastavení (pokud nastane problém s ovladačem je nutné ho povolit v souboru php.ini, který jsme si vytvořili při instalaci). Jestliže chceme používat databázi **SQLite** (jednosouborová databáze) je nutné provést tyto změny. V souboru *config.yml* u konfigurace Doctrine změníme ovladač na pdo\_sqlite. Druhá změna v tomto souboru je ta, že přidáme parametr path.  
  
  
Jelikož jsme přidali nový parametr musíme upravit i soubor *parameters.yml*. Ještě než ho upravíme, vytvoříme si ve složce *var* složku *data* ve které bude databáze uložena. V souboru parameters.yml přidáme parametr path takto. Zde na pořadí parametrů nezáleží, pouze musejí být správně odsazené.  


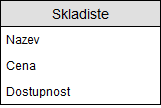
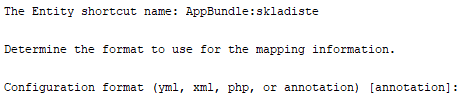
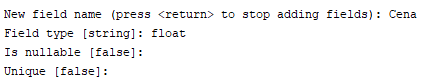
#### [**Ukázka kódu pro parametr path**](http://195.113.207.163:8001/crud#collapse1)

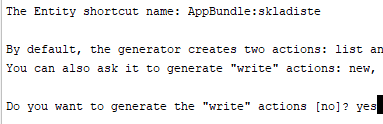
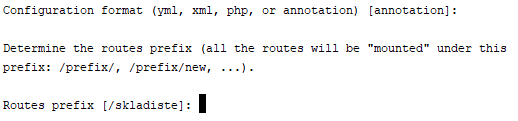
Pokud však budeme vyvíjet na nějakém serveru je třeba v souboru parameters.yml nastavit odpovídající hodnoty. Zde je ukázka připojení ke školní databázi, jestliže je projekt nahraný na školním serveru. Toto připojení se nastaví pouze v souboru parameters.yml.  
  


A teď se již vrhneme na tvorbu databáze a entit. Databáze se vytvoří jednoduchým příkazem.  
***php bin/console doctrine:database:create***  
create_database

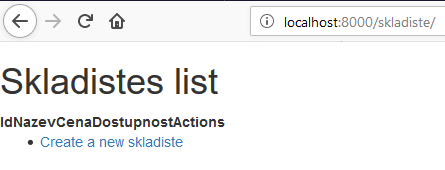
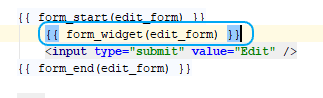
Jestliže máme databázi vytvořenou je čas vytvořit si naši entitu. My si vytvoříme entitu s názvem skladiste. Bude mít tři sloupce:

* Nazev - string (255)
* Cena - float
* Dostupnost - boolean

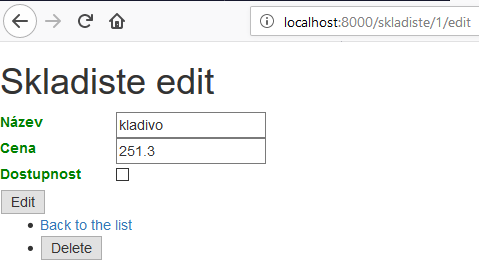
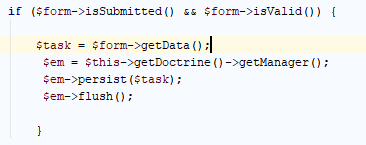
  
  
Na vytvoření entity opět použijeme Doctrinu, která naši práci s databází zjednodušší. Použijeme tento příkaz.  
***php bin/console doctrine:generate:entity***  
Nejprve musíme zadat, jméno entity a jméno bundlu ve které ji chceme vygenerovat. My zvolíme defaultní bundle (AppBundle) a název entity skladiste. Jako druhá věc se musí zadat konfigurační formát. Zde necháme annotation, což je defaultní možnost.  
  
  
  
Nyní již následuje samotná tvorba entity. Zde si definujeme jaké sloupce bude entita obsahovat. První je zde název sloupce. Následuje typ sloupce, další volba je zda-li může být nulový. Jako poslední je zde určení jedinečnosti v rámci entity. Sloupec *id* se do naší entity přidá defaultně sám.  
  


Po vytvoření naší entity je nutné zadat tento příkaz.  
***php bin/console doctrine:schema:update --force***  
Proto, aby se změna projevila v databázi. Po tomto kroku se v námi zadaném bundle (v našem případě AppBundle) ve složce Entity vytvořil .php soubor s názvem, který jsme zadali při vytváření (skladiste). Zde se nachází naše vygenerovaná entita. Nyní si ukážeme jak vygenerovat jednoduchou správu této entity. K tomu nám poslouží CRUD. Opět si toto vygenerujeme pomocí Doctrine příkazem.  
***php bin/console generate:doctrine:crud***  
Po potvrzení tohoto příkazu musíme zadat jméno a umístění entity pro kterou chceme CRUD generovat. V našem případě AppBundle:skladiste. Jako druhé musíme vybrat jestli chceme vygenerovat i "psané" akce, což je mazání, vytvoření, upravení. Zde je defaultní možnost "no" a tuto možnost musíme přepsat na "yes".  
  
  
  
Dále musíme opět vybrat konfigurační formát, kde vybereme annotation (defaultní možnost). Poté vybereme jaký prefix budeme chtít pro toto rozhraní používat. Doporučuji nechat defaultní, ale pokud chceme můžeme si zadat cokoli. Jako poslední je jenom potvrzení této generace.  
  


Po vygenerování máme kompletní správu naší entity, která ale není graficky nejlepší. Ještě musíme zadat příkaz pro update databáze.  
***php bin/console doctrine:schema:update --force***  
Tento vzhled můžeme samozřejmě upravit podle sebe. Můžeme ho měnit v šablonách, které se nám vygenerovali. Tyto šablony jsou ve složce *app->Resources->views->složka s naším jménem, které jsme zadali*. Zde se nacházejí čtyři šablony, každá pro jinou akci. Tyto šablony my můžeme měnit tak jak budeme potřebovat. Můžeme zde dělit do divu nebo přidat třídy Bootstrapu, jestliže je máme nalinkované.

Když naši CRUD aplikaci poprvé spustíme s routou (prefixem), kterou jsme si vybrali uvidíme toto.  
  
  
  
Tady si ukážeme menší ukázku jak lze editovat například formulář pro editaci dat (app->Resources->views->skladiste->edit.html.twig), aby vypadal trochu lépe. Uděláme to tak že pouze nahradíme tento řádek kódem, který je níže.  
  


#### [**Ukázka kódu pro úpravu šablony**](http://195.113.207.163:8001/crud#collapse2)

V prohlížeči vypadá tato úprava následovně.  
  
  
  
Nakonec si ještě ukážeme jak lze napojit formulář (lekce 6) na entitu. A jak do této entity následně uložit data zadaná ve formuláři. Aby vše fungovalo správně musí být entita vytvořena pomocí Doctrine. Při tvorbě formuláře budeme postupovat jako v šesté lekci. Změna bude v tom, že po kontrole jestli byla data odeslána tyto data zapíšeme do databáze. To se provede takto.  
  


Poznámky

# Upozornění mazání

<a href="{{ path('firma\_delete', {'firma': firma.id})}}" class="btn btn-danger" onclick="return confirm('Opravdu si přejete smazat vybranou firmu?')" title="Odstranit firmu">

Případně

<**input type="submit" value="Delete" onclick="return** *confirm*(**'Opravdu si přejete smazat vybranou firmu?'**)**" title="Odstranit firmu"**>

# Rozdělení řádků formuláře a české pojmenování. Nezapomenout v use TextType

**<?php  
  
namespace** AppBundle\Form;  
  
**use** Symfony\Component\Form\AbstractType;  
**use** Symfony\Component\Form\Extension\Core\Type\TextType;  
**use** Symfony\Component\Form\FormBuilderInterface;  
**use** Symfony\Component\OptionsResolver\OptionsResolver;  
  
**class** StrojType **extends** AbstractType  
{  
 */\*\*  
 \* {****@inheritdoc****}  
 \*/* **public function** buildForm(FormBuilderInterface $builder, **array** $options)  
 {  
 $builder->add(**'nazev'**,TextType::**class**, [  
 **'label'** => **'Název'**,  
 **'required'** => **true**,  
 ]);  
 $builder->add(**'cena'**,TextType::**class**, [  
 **'label'** => **'Cena'**,  
 **'required'** => **true**,  
 ]);  
  
  
  
  
  
 }*/\*\*  
 \* {****@inheritdoc****}  
 \*/* **public function** configureOptions(OptionsResolver $resolver)  
 {  
 $resolver->setDefaults(**array**(  
 **'data\_class'** => **'AppBundle\Entity\Stroj'** ));  
 }  
  
 */\*\*  
 \* {****@inheritdoc****}  
 \*/* **public function** getBlockPrefix()  
 {  
 **return 'appbundle\_stroj'**;  
 }  
  
  
}

# Vzhled formuláře

{% **extends 'base.html.twig'** %}  
{% **block** stylesheets %}  
<**link rel="stylesheet" href="**{{ asset(**'https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'**) }}**"**>  
 *{# {% form\_theme form 'bootstrap\_4\_layout.html.twig' %} #}* {% **form\_theme** form **'bootstrap\_4\_horizontal\_layout.html.twig'** %}  
{% **endblock** %}  
{% **block** body %}  
<**div class="container"**>  
 <**h1**>Stroj creation</**h1**>  
  
 {{ form\_start(form) }}  
 <**div style="width**: 800**px"**>  
 {{ form\_widget(form) }}  
 </**div**>  
 <**input type="submit" value="Create"** />  
 {{ form\_end(form) }}  
  
 <**ul**>  
 <**li**>  
 <**a href="**{{ path(**'stroj\_index'**) }}**"**>Back to the list</**a**>  
 </**li**>  
 </**ul**>  
</**div**>  
{% **endblock** %}

