# PHP Basic CRUD: TODO List

# Problem

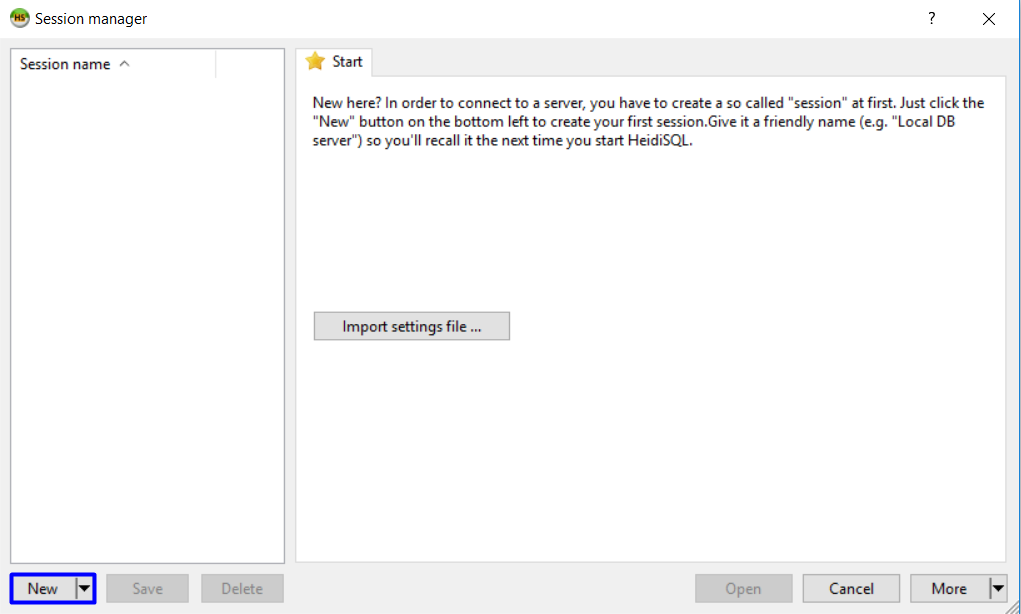
You have been tasked to create a simple **TODO List** application. The application should hold **tasks**, which are the main app **entities**.

The functionality of the application should support **creating**, **listing,** **editing** and **deleting** tasks.

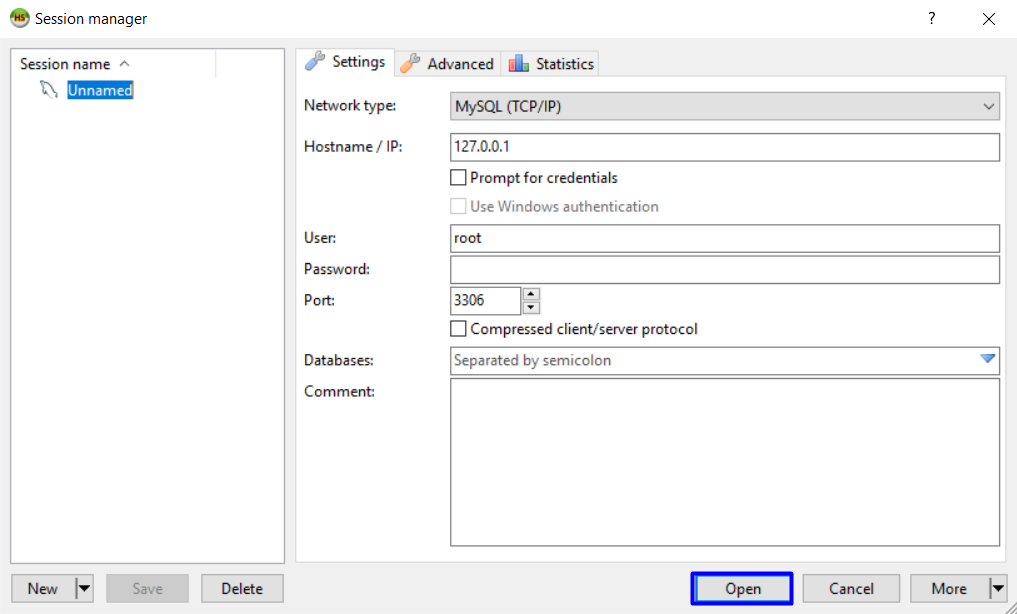
The application should **persist** the data into a **database**.

Before you start you can download [HeidiSQL](https://www.heidisql.com/) from [here](https://www.heidisql.com/installers/HeidiSQL_9.5.0.5196_Setup.exe). "Heidi" allows you to see and edit data and structures from databases. You can install the **tool** by clicking next, next, next…

When you open **Heidi** you are going to see this:



You can create a new session (click on "New") and open it. Then you can manage your databases:



# Overview

## Requirements

* **Symfony** framework
* **Twig** view engine
* **Doctrine** ORM
* **MySQL** database

## Data Model

The **Task** entity holds **3 properties**:

* **id – int**
* **title** **– non-empty text**
* **comments** **– non-empty text**

## Project Skeletons

You will be given the applications’ skeletons, which holds about **90%** of the logic. You’ll be given some **files** (**controllers**, **models**, **views**, etc.). The files will have **partially implemented logic**, so you’ll need to write some code for the application to **function properly**.

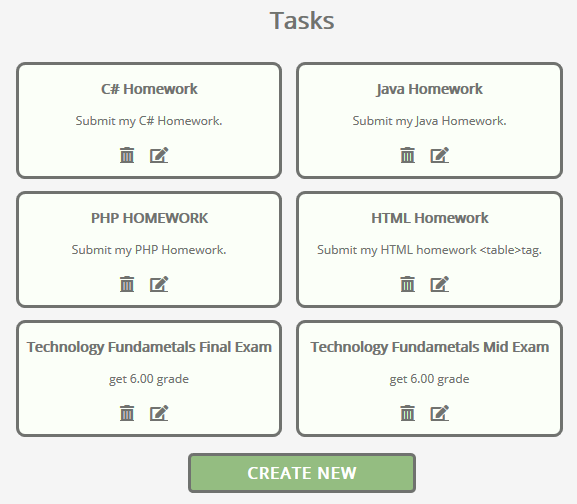
The application’s views will be given to you fully implemented. You only need to include them in your business logic.

Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

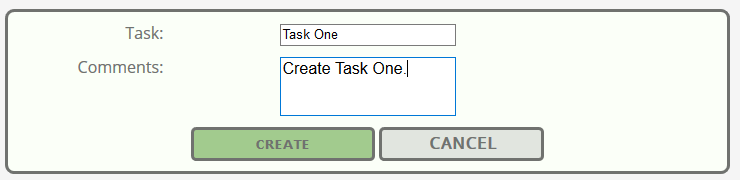
## User Interface

This is the user interface or how the application’s pages should look in their final form (fully implemented). You have several pages, described below:

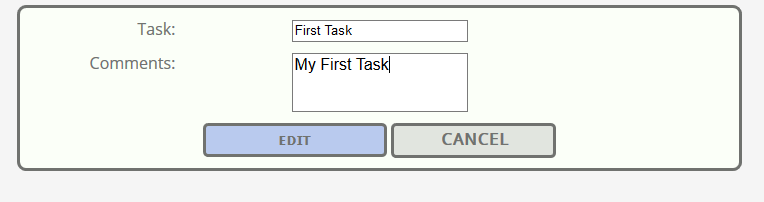
### Index Page



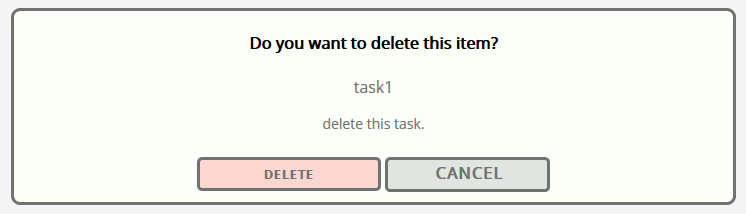
### Create Page



### Edit Page



### Delete Page



# Implementation

## Create Your Database

To create the database you can simply write in the terminal:



## Create Task Entity and Form

You need to create the Entity. Do this by writing in the terminal the following command:



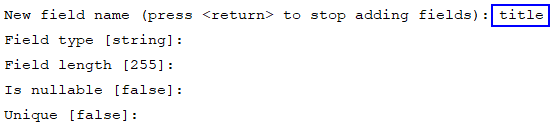
Then give it name. In our case: AppBundle:Task

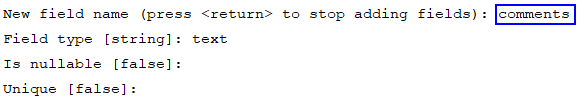


When you see this , just tap Enter:

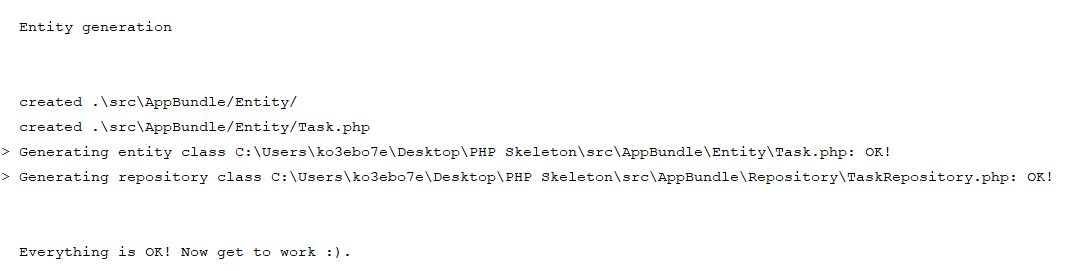


Then you need to create the fields. (2 fields: title and comments)





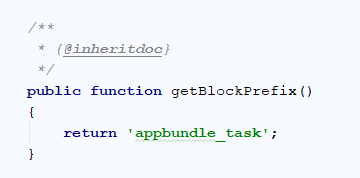
Finally, press '**Enter**' one more time and you should see this:



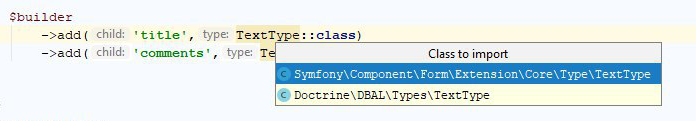
Now you need to create Form:



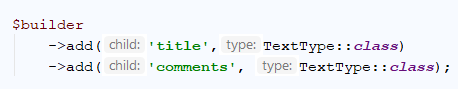
Then go to AppBundle\Form\Tasktype and delete the following:



Then go to buildForm and add types. Both will have TextType (Choose the following option):



It should look like this:



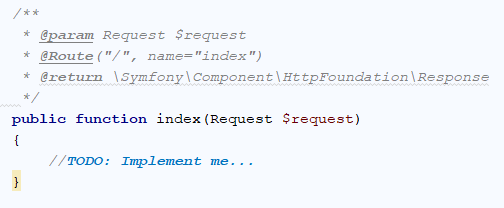
Update database:



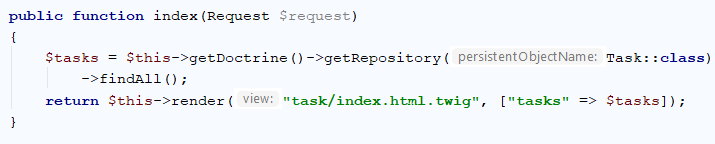


## Listing All Tasks

First, we need to list all tasks to the given route. Go to the **TaskController** and find **index** method

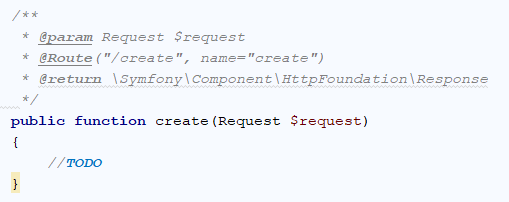


Thanks to the **task repository** we can get **all tasks** that are in the **database**:

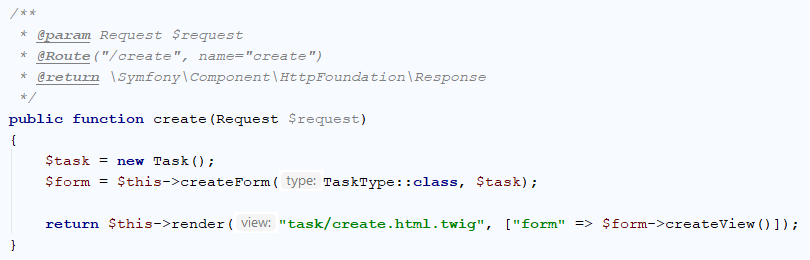


## Creating Tasks

First, we need to process the **get request**. As response we need to return the view to the user.



For a task to be created, we need to hydrate the request into the entity using the form type from above once the form is submitted. The populated entity has to be inserted into the database using the central access point to the ORM functionality - the Doctrine's Entity Manager.



What is this code doing? It’s simple – it **creates a new task**. Then it **creates** a **new form** from the template we’ve created earlier and tells the **form** that it **should** **fill our new task**. Finally, it **sends the form to a** **view** that we are going to **render** on the screen.

If we run the project, open **create** page and **fill the form and click** "Create" the **page gets refreshed**, but if we check the table in the **database**, **it is empty**. Let’s fix the problem. Get back to your function in the task controller. The problem is that we’ve never used the data from our form. Add to your function the following code:



This code **takes the data from request** and **fills** the **form**. After the form is filled, we check if there is **any data** in the form and if it **is valid**. If everything is okay, we get the **entity manager** from **doctrine** and using the "persist" function we **add** our **new task** in the **database**. Finally, we call the "flush" function, which sends the task to our database. **After** the task is **sent** to the database, we **redirect** the **view** to the **index page**.

## Editing Tasks

When we **click** on the **edit** **button** from the view we sent the **task id** to the **edit action**. We must find the **task** with given **id** from the **database**.

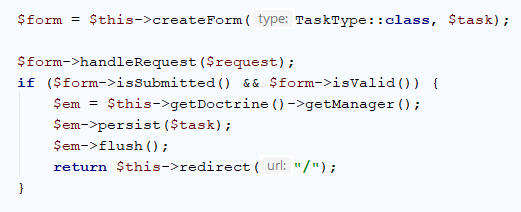
If the task is **null** (we **can't have task with that id**) **redirect** to index page. If **not null**, analogical we **creates** a **new form** from the template we’ve created earlier and tells the **form** that it **should** **fill our new task**.

Finally, it **sends the form to a** **view** that we are going to **render** on the screen:



Now we must implement **POST** **Request**.

First we **take** the data from the **request** and check whether the form is **submitted** and **valid**. And If it, we get the **entity** **manager** from **doctrine** and **add** the **new** **data** for the task. Finally **redirect** to index page.



## Deleting Tasks

Analogical like **edit** **action**, we receive **task** **id** from the view, **find** given task from the **database**, **check** whether task **is** **null** and **render** to the **delete** **view** (**GET Request**). For **POST** **Request** **take** the **data** from the **request**, **check** whether the **form** is **submitted**, **delete** **the** **task** in the **database** and **redirect** to **index** **page**:



**Now we are ready with our first CRUD Project.**

***We can't wait for the exercise, right?***