# PHP Basic Web: Phonebook

## Problem

You have been tasked to create a simple **Phonebook** application. The application should hold **contacts**, which are the main app **entity**.

The functionality of the application should support:

* **Listing contacts**



* **Add Contact**

****

****

## Overview

### Requirements

* **Symfony** framework
* **Twig** view engine

### Data Model

The Contact entity holds **2 properties**:

* name – non-empty text
* number – 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**. 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.

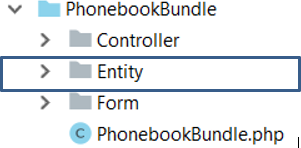
Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

## Preparation

Start PhpStorm and **open** the skeleton and install theproject **dependencies**. For help you can see **"Set\_up\_Symfony.docx"** document.

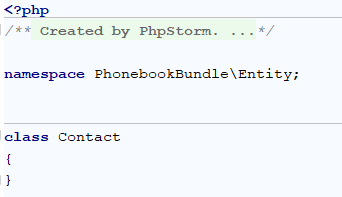
### Create Contact entity

Since we won’t be using a database in this exercise, we **won’t** be using **Doctrine** to generate our entities. Instead, we’ll **make them ourselves**. Head on over to the src/PhonebookBundle/Entity folder:

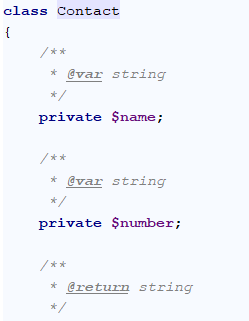


Create a new PHP class inside it, called Contact.php:

The file should look like this:

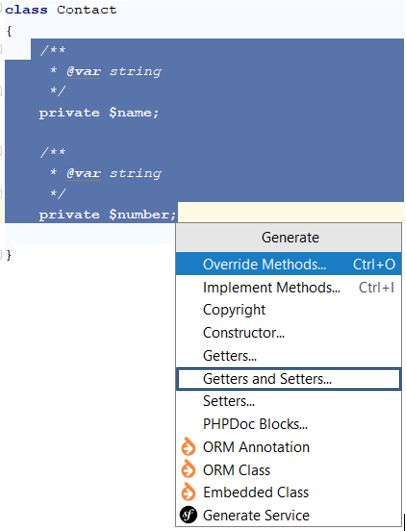


We need to define our contact entity. Create the following private fields:

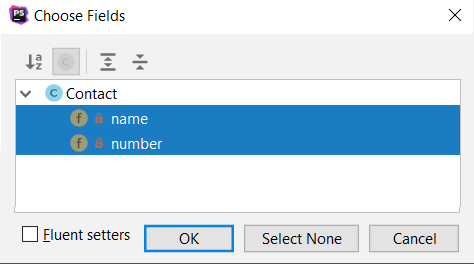


### Create Getters and Setters

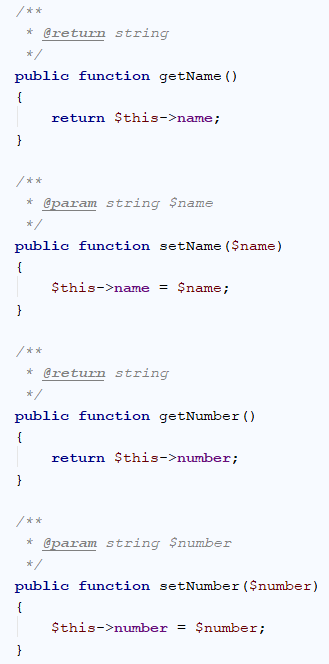
Let's create [getters and setters](http://thisinterestsme.com/php-getters-and-setters/) for our fields. You should already be familiar with them. There is a **simple way to create them** in **PhpStorm**. If you press "**Alt + Insert**", you should see that context menu:



Choosing the "**Getter and Setter**" option will **open new window**. You should select **all private fields** from there:



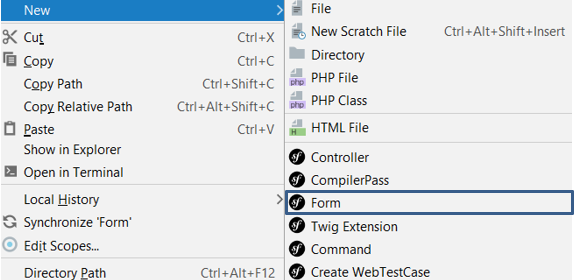
When you **click** "**OK**", you should **receive this code**:



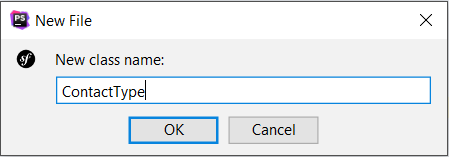
And this is pretty much everything. Our Contact entity is ready.

### **Creating the Contact Type Form**

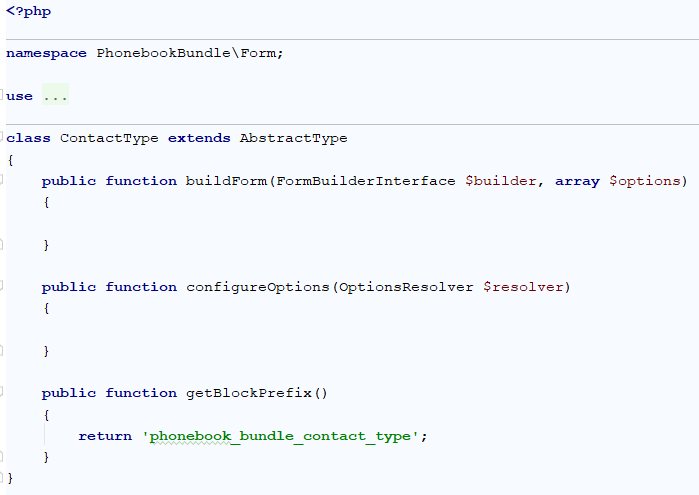
Our next step is to **create a form template** that we are going to fill, each time when we’re **creating** a contact. To create this form, just right-click on the **Form** folder and choose new **Form**:



Give it the name “ContactType”:

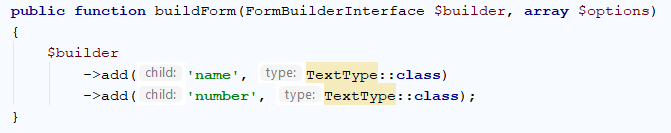


We should receive something like this:



Now you need to remove the last function **getBlockPrefix()**

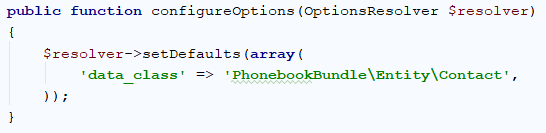
You may notice that we have 2 empty functions. “buildForm” will create our form and “configureOptions” will tell our form that it is for the Contact entity. Let’s start with the form creator:



It’s a pretty simple form. It should only contain **name** and **number** fields, both of type text. You should use specific using for the “TextType” to work. If you have another one **ending in \TextType** already imported – delete it and add:

|  |
| --- |
| **use** Symfony\Component\Form\Extension\Core\Type\TextType; |

Let’s create the logic for our “configureOptions” function:

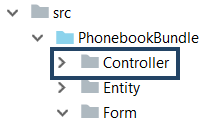


The default value for our resolver **tells controller that is going to use the form**, in what type of object it should save the date from our form. That’s it.

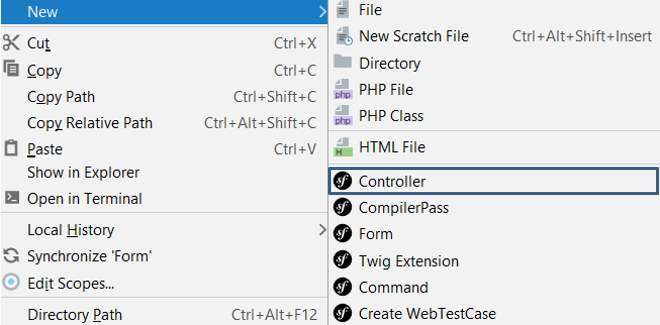


### Creating the Contact Controller

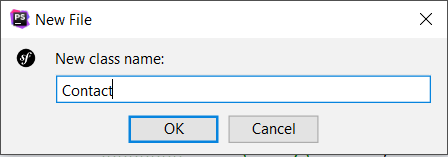
Now we should create a class that will control how the contacts are going to be viewed and created. In the "Controller" package create a new class called "ConctactController":



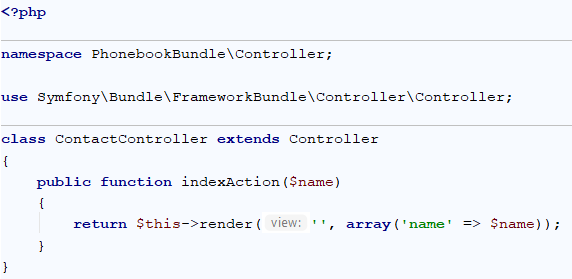
To create this controller, just right-click on the **Controller** folder and choose new **Controller**:

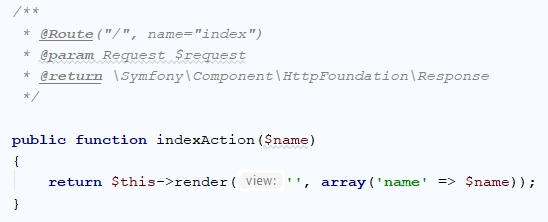


Give it the name ContactController:



We have just created the ContactController in the **Controller** folder, that looks like this:



Now we should create the annotations 

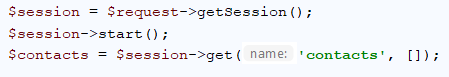
Let’s start from the first annotation. It tells our project that the function will receive **one parameter** of type [Request](http://api.symfony.com/3.1/Symfony/Component/HttpFoundation/Request.html). We will save what request is for some other time. The second annotation is more interesting. It defines a "[Route](https://symfony.com/doc/3.4/routing.html)". The **route represents** the **URL**, that the **current** **method will correspond to**. In this case the function will be called when we go to <http://localhost:8000/>. Each time we **use this URL**, the router will **call our function**. To **simplify** the **redirection** between our **pages**, we give a simpler name like “index”. The final parameter specifies that our **function** will **return** a **response**. We will talk about this later. In order for those annotations to work correctly, make sure you are using the right imports:



Now let’s write some real code. In the function, write the following:

### Session

Symfony provides a nice **session** object that you can use to store information about the user (be it a real person using a browser, a bot, or a web service) between requests. By default, Symfony stores the attributes in a cookie by using the native **PHP sessions**.



To retrieve the session, call getSession()method on the **Request** object and start it with the start()method.

After that we get **"contacts"** from the session with get() function and if does not contains key **"contacts"** create default value empty array ("[]")



What is this code doing? It’s simple – it **creates a new contact**. Then it **creates** a **new form** from the template we’ve created earlier and tells the **form** that it **should** **fill our new contact**. Finally, it sends the form to a view that we are going to render on the screen. Render means draw. Symfony uses **Twig**. Twig is a templating engine, which job is to display our data in an easier way, than creating the HTML by ourselves. Now we need some new imports:



Congratulations! If everything works okay, you’ve just created a very simple phonebook application where users can store their contacts.