

**Ministerul Educației, Culturii și Cercetării a Republicii Moldova**

**Universitatea de Stat a Moldovei**

**Facultatea Matematică și Informatică**

Raport

**Lucrare de laborator Nr.4  
la disciplina Framework-uri pentru Dezvoltarea de Aplicații Web  
Tema:”Formulare”**

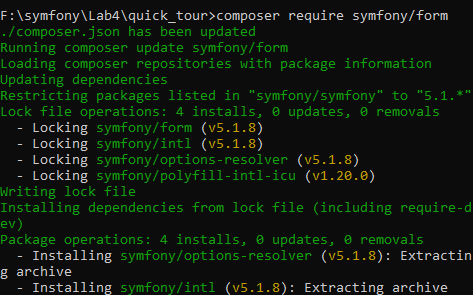
**A efectuat : Batîr Ana**

**A verificat : Bodrug Svetlana**

**Chișinău 2020**

# **Instalare**

În aplicațiile care utilizează Symfony Flex, rulez această comandă pentru a instala caracteristica formularului înainte de a o utiliza:



# **Utilizare**

Fluxul de lucru recomandat atunci când lucrez cu formularele Symfony este următorul:

* Construesc formularul într-un controler Symfony sau utilizând o clasă de formular dedicată;
* Redau formularul într-un șablon, astfel încât să îl pot edita și trimite;
* Procesez formularul pentru a valida datele trimise, transformându-le în date PHP și fac ceva cu acesta (de exemplu, le persist într-o bază de date).

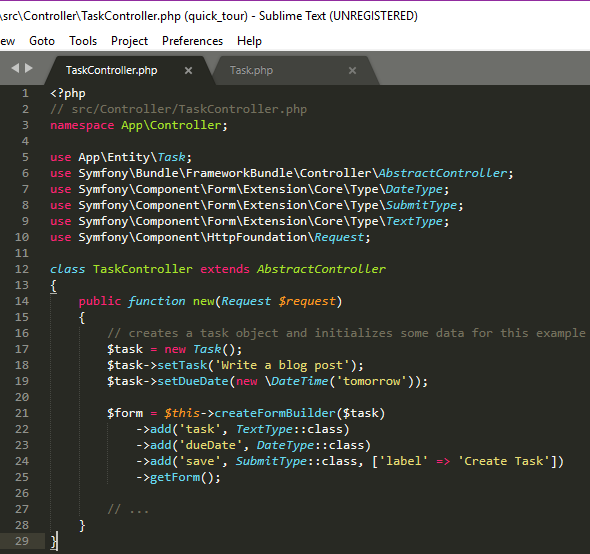
Fiecare dintre acești pași este explicat în detaliu în secțiunile următoare. Pentru a face exemplele mai ușor de urmărit, toți presupun că construiți o mică aplicație de listă Todo care afișează „sarcini”.

Creez și editez sarcini folosind formularele Symfony. Fiecare activitate este o instanță a următoarei clase de activități:



# **Crearea de formulare în controlere**

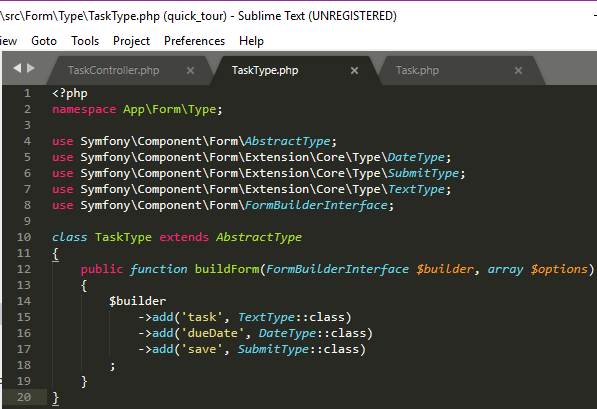
Dacă controlerul se extinde de la AbstractController, utilizez ajutorul de tip createFormBuilder ():



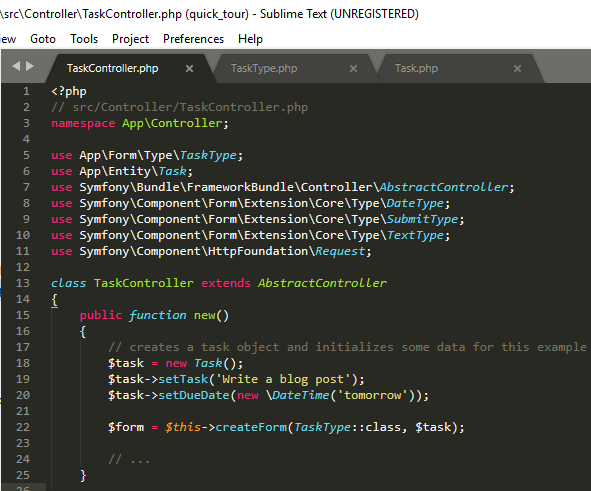
# **Crearea claselor de formular**

Symfony recomandă să pun cât mai puțină logică în controlere. De aceea, este mai bine să mut forme complexe în clase dedicate, în loc să le definesc în acțiunile controlerului. În plus, formularele definite în clase pot fi refolosite în mai multe acțiuni și servicii.

Clasele de formulare sunt tipuri de formulare care implementează Symfony \ Component \ Form \ FormTypeInterface. Cu toate acestea, este mai bine să extind din Symfony \ Component \ Form \ AbstractType, care implementează deja interfața și oferă câteva utilitare:

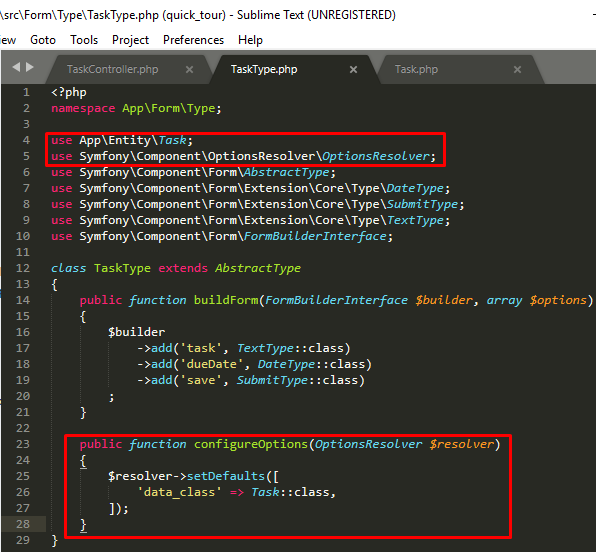


Clasa formular conține toate direcțiile necesare pentru a crea formularul sarcinii. În controlerele care se extind de la AbstractController, utilizez ajutorul helpFreeForm () (în caz contrar, utilizez metoda create () a serviciului form.factory):



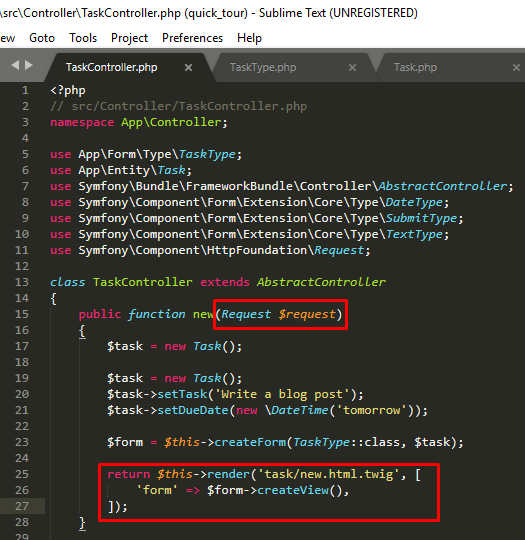
Fiecare formular trebuie să știe numele clasei care deține datele subiacente (de exemplu, App \ Entity \ Task). De obicei, acest lucru este doar ghicit pe baza obiectului trecut la al doilea argument pentru createForm () (adică $ task). Mai târziu, când încep să încorporez formulare, acest lucru nu va mai fi suficient.

Deci, deși nu este întotdeauna necesar, este, în general, o idee bună să specific în mod explicit opțiunea data\_class adăugând următoarele la clasa de tip formular:

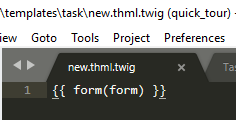


# **Formulare de redare**

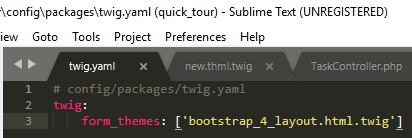
Acum că formularul a fost creat, următorul pas este redarea acestuia. În loc să trec întregul obiect formular la șablon, utilizez metoda createView () pentru a construi un alt obiect cu reprezentarea vizuală a formularului:



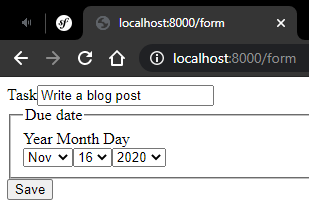
Apoi, utilizez câteva funcții de ajutor de formular pentru a reda conținutul formularului:



Pe cât de scurtă este această redare, nu este foarte flexibilă. De obicei, voi avea nevoie de mai mult control asupra aspectului întregului formular sau a câtorva dintre câmpurile sale. De exemplu, datorită integrării Bootstrap 4 cu formulare Symfony pot seta această opțiune pentru a genera formulare compatibile cu cadrul Bootstrap 4 CSS:



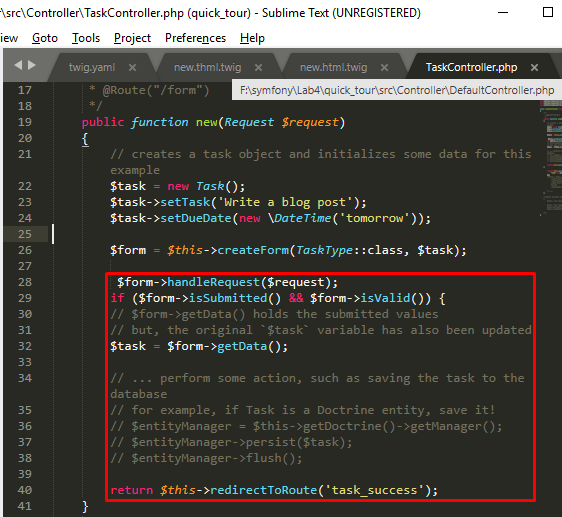
## **REZULTAT:**



# **Formulare de procesare**

Modul recomandat de procesare a formularelor este acela de a utiliza o singură acțiune atât pentru redarea formularului, cât și pentru gestionarea formularului de trimitere. Pot utiliza acțiuni separate, dar folosind o acțiune simplifică totul, păstrând în același timp codul concis și mentenabil.

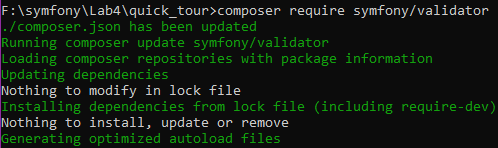
Prelucrarea unui formular înseamnă traducerea datelor trimise de utilizator înapoi la proprietățile unui obiect. Pentru a face acest lucru, datele trimise de la utilizator trebuie să fie scrise în obiectul formular:



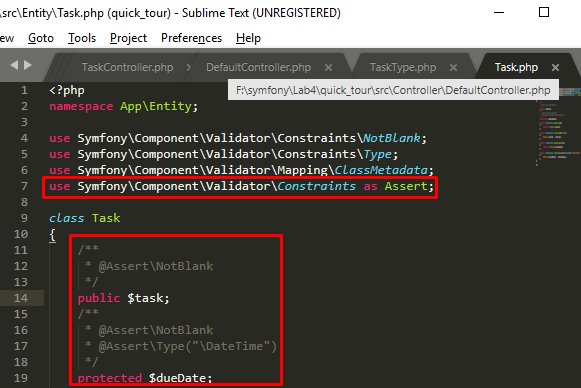
# **Formulare de validare**

În secțiunea anterioară, am aflat cum poate fi trimis un formular cu date valide sau nevalide. În Symfony, întrebarea nu este dacă „formularul” este valid, ci dacă obiectul subiacent ($ task în acest exemplu) este valid sau nu după ce formularul i-a aplicat datele trimise. Apelarea $ form-> isValid () este o comandă rapidă care întreabă obiectul $ task dacă are sau nu date valide.

Înainte de a utiliza validarea, adaug asistență pentru aceasta în aplicație:



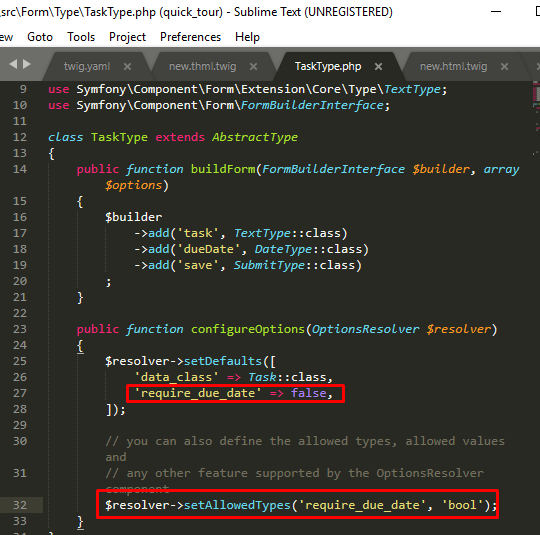
Pentru a vedea prima abordare - adăugarea de constrângeri la entitate - în acțiune, adăugați constrângerile de validare, astfel încât câmpul de activitate să nu poată fi gol, iar câmpul dueDate să nu poată fi gol și trebuie să fie un obiect DateTime valid.



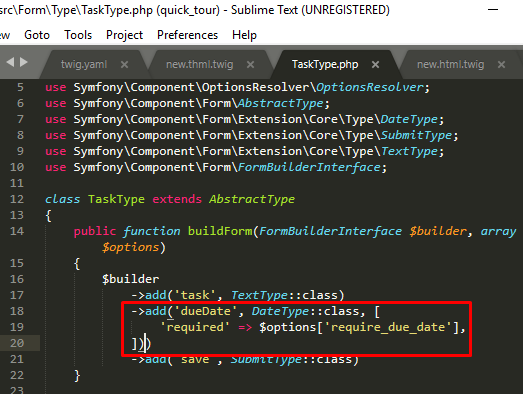
# **Alte caracteristici comune ale formularului**

# **Trecerea opțiunilor la formulare**

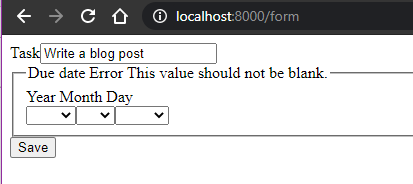
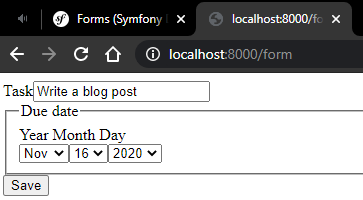
Dacă încerc să utilizez formularul acum, voi vedea un mesaj de eroare: opțiunea „require\_due\_date” nu există. Acest lucru se datorează faptului că formularele trebuie să declare toate opțiunile pe care le acceptă folosind metoda configureOptions ():

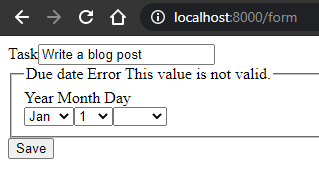


Acum pot utiliza această nouă opțiune de formular în cadrul metodei buildForm ():



**REZULTAT:**



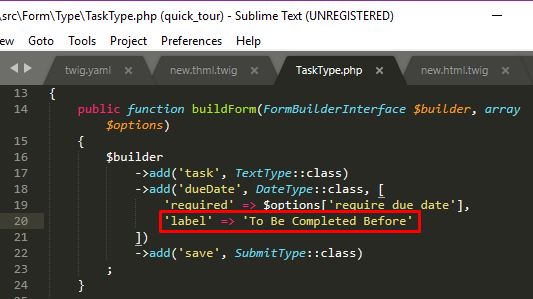


# **Opțiuni tip formular**

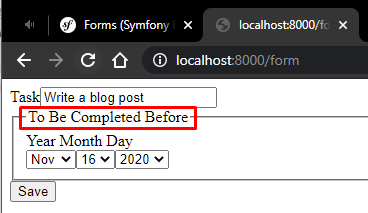
Fiecare tip de formular are un număr de opțiuni pentru a-l configura, așa cum se explică în referința tipurilor de formular Symfony. Sunt necesare două opțiuni utilizate în mod obișnuit și etichetă.

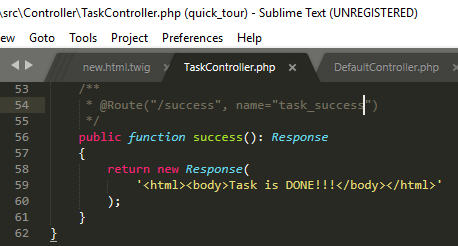
# **Eticheta Opțiune**

În mod implicit, eticheta câmpurilor formularului este versiunea umanizată a numelui proprietății (utilizator -> Utilizator; Adresă poștală -> Adresă poștală). Setez opțiunea etichetă pe câmpuri pentru a le defini în mod explicit etichetele:



**REZULTAT:**





**REZULTAT:**

