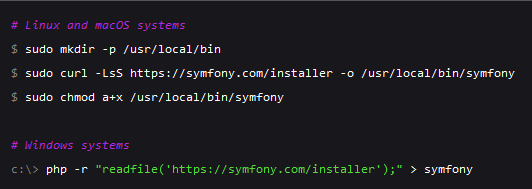
## Symfony karkaso įdiegimas ir konfigūravimas

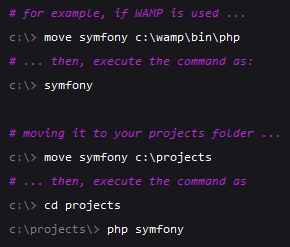
### Projekto sukūrimas

Čia bus aprašomas projekto sukūrimas nuo pat pradžių. Taigi iš pradžių reikia atsisiųsti ir įsidiegti Symfony Installer ir jį naudot savo projektų sukūrimui. Taip pat reikia turėti įsirašius kokį nors lokalų serverį (pavyzdžiui XAMPP). Toliau yra padaryta nuoroda į tai, kaip instaliuoti ir paruošti naudojimui Symfony. Pagrindiniai konfigūracijos punktai atsidarius komandinę eilutę:



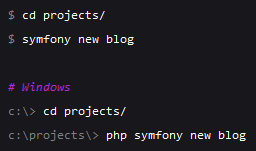
11pav. Pagrindiniai konfigūracijos punktai

Linux in macOS operacinėse sistemose yra sukuriama globali symfony komanda, o Windows reikia perkelti symfony failą į lokalaus serverio direktoriją, tam, kad būtų sukūrta globali komanda arba perkelti į bet kokią Jums tinkančią direktoriją:



12pav. Perkėlimas į direktoriją

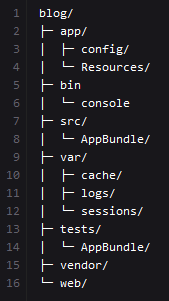
Po įdiegimo galima sukurti savo Symfony programą naudojant komandą ,,new‘‘:



13pav. Programos sukūrimas

Tiesa, tai yra ne visai pirmas dalykas, kurį reikėjo daryti atsisiunčiant Symfony Installer. Iš pradžių reikėjo instaliuoti Composer į komandinę eilutę įrašius „composer install“. Instaliavus jį, galima į Symfonį karkasą įtraukti norimas bibliotekas.

Sukūrus projektą ir įvedus ,,blog/“ direktoriją bus matomi automatiškai sukurti failai ir direktorijos:



14pav. Automatiškai sukurti failai ir direktorijos

Taip pat oficialiame tinklalapyje buvo aprašoma kiekvienos direktorijos rekomenduojamas pritaikymas:

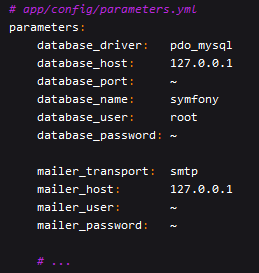
* app/config/ - visa nurodyta konfigūracija bet kokiai aplinkai
* app/Resources/ - visi šablonai ir vertimo failai
* src/AppBundle/ - Symfony valdikliai ir keliai, domeno kodas ir visa verslo logika
* var/cache/ – visi ,,cache“ failai
* var/logs/ - visi logo failai
* var/sessions/ - visi sesijos failai
* tests/AppBundle/ - automatiniai testai
* vendor/ - šioje direktorijoje ,,Composer“ instaliuoja programos priklausomybes ir vartotojas čia nieko neturėtų redaguoti
* web/ - stiliaus, JavaScript failai ir paveiksliukai

Taip pat reikėjo sukonfigūruoti lokalų serverį, kurį paleisdavau per programą „XAMPP“. O būtent reikėjo nustatyti portus, per kuriuos pasileistų lokalus serveris, tai iš esamo 8000 keičiau į 8080.

### Konfigūracija

Šioje skiltyje aprašau pagrindinę projekto konfigūraciją ir kaip geriau ką naudoti. Bendrai konfigūracija apima skirtingas projekto dalis ir aplinkas ir dėl to Symfony rekomenduoja padalinti projekto konfigūraciją į tris dalis:

* app/config/parameters.yml – apibrėžiami su infrastruktūra susiję nustatymai



15pav. Su infrastruktūra susiję nustatymai

* app/config/parameters.yml.dist – apibrėžiami visi projekto parametrai
* app/config/config.yml – apibrėžiamas projekto funkcionalumas

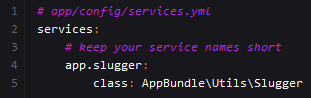
Taip pat parametrų konfigūracijos pavadinimai turėtų būti kuo trumpesni ir sudaromi su bendru priešdėliu visai programai:

16pav. Konfigūracijos pavadinima

### Verslo logikos organizavimas

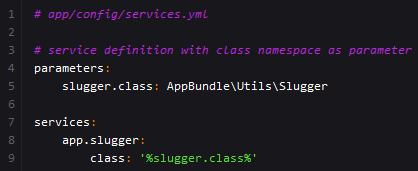
Programinėje įrangoje verslo logika arba domeno logika yra aprašoma kaip programos dalis, kuri nurodo verslo taisykles kaip turi būti sukurti, rodomi, laikomi ir redaguojami duomenys. Symfony programose verslo logika yra visas kodas, kurį rašo vartotojas, kuris nėra priskirtas prie jokio karkaso, pavyzdžiui keliai ir valdikliai. Domenų klasės, Doktrinų esybės ir paprastos PHP klasės, kurios naudojamos kaip servisai yra geri verslo logikos pavyzdžiai.

Programos servisų pavadinimai turėtų būti kuo trumpesni, bet pakankamai unikalūs, kad prireikus juos galima būtų surasti projekte:



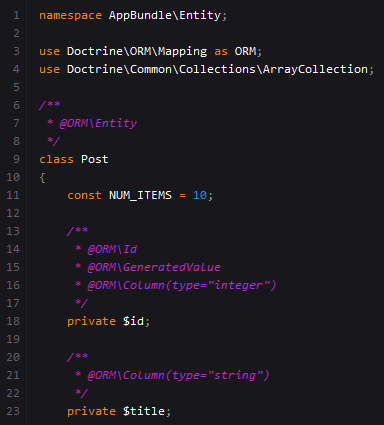
17pav. Programos servisų pavadinimai

Savo sukurtų servisų apibrėžimui reiktų naudoti YAML formatą, nes jis yra draugiškas pradedantiesiems ir glaustas. Taip pat nėra rekomenduojama turėti servisų klasių pavadinimus:



18pav. Servisų klasių pavadinimai

Doktrinos esybėms ir ,,mapping“ informacijai reikia naudoti anotacijas, nes jos yra labiausiai patikimas ir greitas būdas tam dalykui:

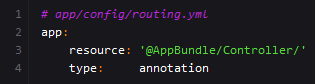


19pav. Anotacijų naudojimas doktrinų esybėms

### Valdikliai

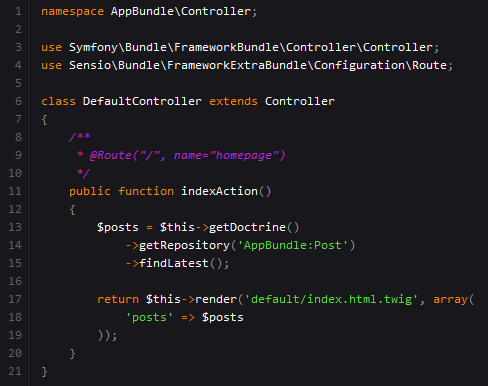
Symfony naudojasi filosofija ,,ploni valdikliai ir stori modeliai“, tai reiškia, kad valdikliai turėtų savyje turėti tik ploną sluoksnį kodo, reikalingo koordinuoti skirtingas programos dalis. Toliau aprašoma 5-10-20 taisyklė – valdikliai turi apibrėžti 5 arba mažiau kintamuosius, turėti 10 arba mažiau metodų, kuriuose turi būti 20 arba mažiau kodo eilučių.

Gera praktika yra sukurtam valdikliui paveldėti FrameworkBundle bazinį valdiklį ir kelių konfigūracijai, ,,cachinimui“ ir apsaugai kada tik įmanoma naudoti anotacijas:



20pav. Anotacijų naudojimas kelių konfigūracijai

Nepatariama naudoti @Template anotacijos valdiklio naudojamo šablono konfigūracijai. Taip turi atrodyti pagrindinio puslapio valdiklis:

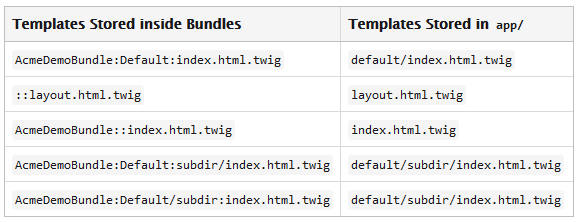


21pav. Pagrindinio puslapio valdiklis

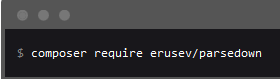
### Šablonai

Kai php buvo sukurtas 20 metų atgal, programuotojams patiko jo paprastumas ir kaip jame gerai susimaišė HTML ir dinaminis kodas. Bet laikas bėga ir atsirado šablonų kalbos, kurios padarė juos dar geresnius. Vienas iš tokių šablonų yra Twig, kuri ir siūlo pagal geriausią praktiką naudoti Symfony. Dėl to Twig yra nustatytas kaip Symfony šablonas pagal nutylėjimą. Taip pat Twig yra vienintelis šablonas, kuris garantuotai veiks su Symfony 3.0.

Symfony siūlo laikyti visus programos šablonus app/Resources/views/ direktorijoje. Prieš tai jie buvo laikomi tiesiog Resources/views/ direktorijoje, tačiau taip kartu prie projekto dirbantiems dizaineriams buvo nepatogu ieškoti tų šablonų išsimėčiusių po visą projektą skirtinguose aplankuose ir naudojant app/ aiškiai supaprastėja jų pavadinimai, pavyzdys (kairės lentelės pusėje kaip buvo anksčiau, o dešinėje, kaip yra dabar):

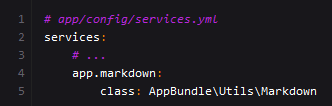


Mano projektas reikalauja specialaus md2html Twig filtro kad galima būtų paversti kiekvieno posto Markdown turinį į HTML. Tam, iš pradžių reikia instaliuoti Parsedown Markdown analizatorių kaip naują projekto priklausomybę:



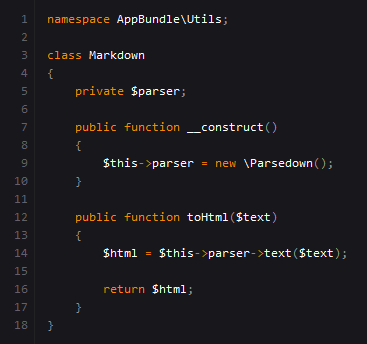
22pav. Analizatoriaus instaliavimas

Toliau reikia sukurti naują Markdown servisą, kuris bus toliau naudojamas Twig plėtiniui. Serviso apibrėžimas reikalauja tik kelio iki klasės:



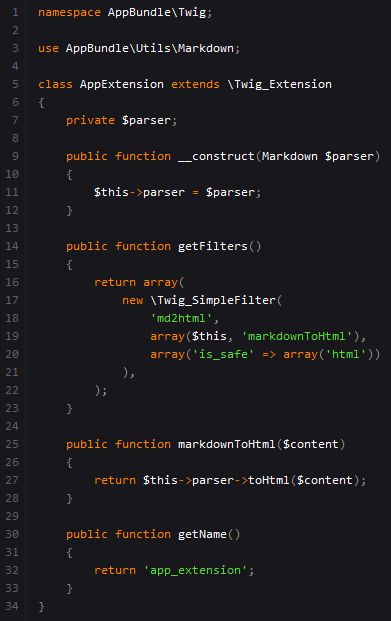
23pav. Serviso apibrėžimo kelias

Kaip ir Markdown klasei reikia apibrėžti tik vieną metodą, kad galima būtų paversti Markdown turinį į HTML:



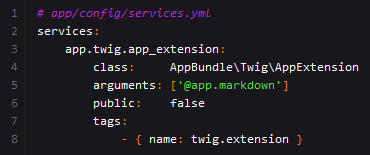
24pav. Markdown turinio pavertimas į HTML

Toliau, reikia sukurti naują Twig plėtinį ir apibrėžti naują filtrą ,,md2html“ naudojant ,,Twig\_SimpleFilter“ klasę ir įterpti naują apibrėžtą ,,markdown“ servisą Twig plėtinio konstruktoriuje:



25pav. Naujo twig plėtinio sukūrimas

Galų gale reikėjo apibrėžti naują servisą, kad galima būtų naudoti Twig plėtinį aplikacijoje:

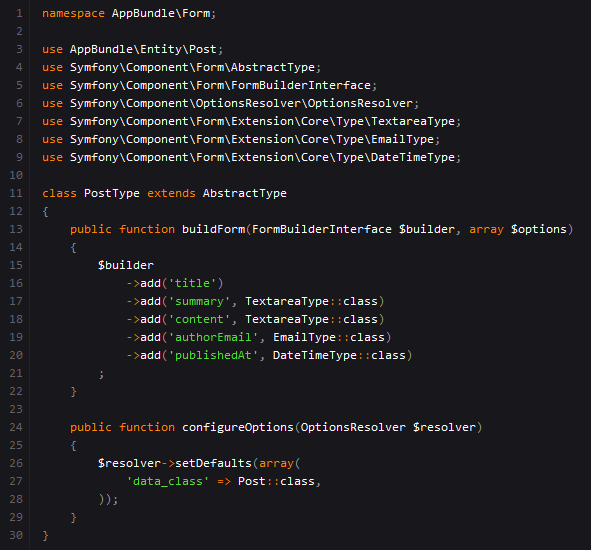


26pav. Naujo serviso apibrėžimas

### Formos

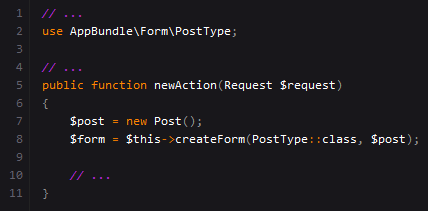
Formos yra vienas iš daugiausiai netinkamai naudojamų Symfony komponentų dėl jų plačios naudojimo srities ir begalinių privalumų. Čia bus aiškinamos geriausios formų naudojimo praktikos norint pagreitinti darbo atlikimą.

Taigi yra siūloma kiekvieną formą aprašinėti PHP klasėse, o tas klases talpinti AppBundle/Form vardo vietoje (namespace):



27pav. Formos aprašymas

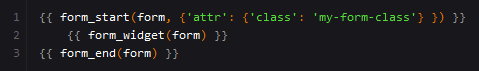
Norint naudoti sukurtą klasę, reikia panaudoti ,,createForm()“ ir perduoti tos klasės pavadinimą:



28pav. Klasės pavadinimo perdavimas

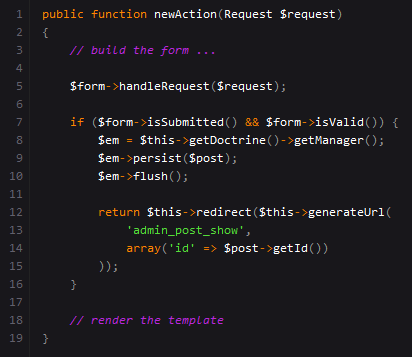
Reikia atidžiai stebėti kur bus naudojamos formų klasės, tai padeda ateityje paprasčiau jas naudoti. Patariama aprašyti mygtukus šablonuose, o ne formų klasėse arba valdikliuose. Taip yra todėl, nes talpinant juos pastaruosiuose bus sumažinta tos formos apimtis ir būtų problematiška naudoti mygtukus postų redagavimui. Taip pat tuo atveju būtų sumaišomi aprašų žymėjimai su grynu PHP kodu. Taigi atskyrimas yra geriausia praktika ir dėl to siūloma talpinti visus su atvaizdavimu susijusius dalykus į vaizdų sluoksnį.

Geriausia paleisti formą naudojant formos valdiklių funkciją:



29pav. Formos valdiklių funkcija

Formos patvirtinimui naudojamas panašus šablonas:



30pav. formos patvirtinimas

### Apsauga

Kalbant apie apsaugą Symfony siūlo nepriklausant nuo poreikių autentifikavimą konfigūruoti security.yml faile, naudojant ,,firewalls“ raktą. Ir yra rekomenduojama naudoti tik vieną ,,firewall“ įrašą su ,,anonymous“ aktyvuotu raktu. Slaptažodžių užkodavimui reikėtų naudoti ,,bcrypt“ kodatorių, o ne tradiciškai naudojamą ,,SHA-512“. Pagrindiniai ,,bcrypt“ privalumai yra ,,salt“ naudojimas apsisaugojimui nuo ,,rainbow table“ ir ,,brute-force“ atakų.

Autorizacijos geriausia praktika:

* security.yml faile naudoti ,,access\_control“
* kur tik įmanoma naudoti ,,@Security“ anotaciją
* tikrinti saugumą tiesiogiai ,,security.authorization\_checker“ servise, jeigu užklumpa sudėtingesnė situacija
* norint uždrausti prieigą prie bet kokio objekto kokiam nors naudotojui per administratoriaus sąsają naudoti Symfony ACL

Projekte norint sukurti naują įrašą reikia rolės ,,ROLE\_USER“.

### Interneto aktyvai

Interneto aktyvai yra tokie dalykai kaip CSS, JavaScript ir paveiksliukai, kurie padaro tinklalapį gražesnį. Dažniausiai jie būna talpinami Resources/public/ arba web/ direktorijoje. Tačiau reikia būti atsargiems talpinant failus web/, nes tuo atveju failai tampa viešai prieinami. Interneto aktyvų sujungimui ir sumažinimui reikia naudoti Assetic.

Assetic yra aktyvų valdytojas, kuris gali kompiliuoti aktyvus padarytus su įvairiomis technologijomis, tokiomis kaip LESS, Sass ir CoffeScript. Visus šituos aktyvus apgaubia viena Twig tagas.