Tallinna Ülikool
Digitehnoloogiate Instituut

SYMFONY2 RAAMISTIKU TUTVUSTUS

Seminaritöö

Autor: Mehis Muldma

Juhendaja: Inga Petuhhov

Autor: ....."....." 2015

Juhendaja:....."....." 2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Sisukord

Sissejuhatus 4
1 . Symfony2 6
1.1 . Ajalugu 6
1.2 . Komponendid ja kogumikud 9
1.3 . Võrdlus teiste raamistikega 11
1.3.1 . Yii2 vs Symfony2 11
1.4 . Raamistiku arhitektuur 13
2 . Paigaldamine ja esimesed mallid 14
2.1 . Raamistiku paigaldamine 14
2.2 . Esimese vaate loomine 17
2.3 . JSON vastusega vaade 19
2.4 . Dünaamilised URL'i mustrid 20
2.5 . Malli visualiseerimine kasutades teenusehoidlat 21
3 . Vormid 25
3.1 . Olemi loomine 25
3.2 . Vormi loomine 26
3.3 . Vormi visualiseerimine 28
3.4 . Vormi ära saatmine (*submit*) 29
3.5 . Vormi valideerimine 31
Kokkuvõte 34
Kasutatud kirjandus 35
LISAD 36
Lisa 1. Komponendid 37

Sissejuhatus

Nüüdisaja kiirelt arenevas infoajastus on oluline pidada sammu uudsete tehnoloogiatega. Nagu teisedki programmerimiskeeled, peab PHP olema pidevas arengus, et olla konkurentsivõimeline maailmas, kus erinevate programmerimiskeelte vahel on tihe konkurents. Kuna PHP on endiselt populaarne, siis tekib enamiku arendajate jaoks ühel hetkel küsimus – millist raamistikku PHP kirjutamiseks kasutada. Minu arvates on raamistike peamiseks eesmärgiks programmeerija töö lihtsamaks tegemine, seda eelkõige projektide eskaleerudes. Ilma raamistikuta on suurema projekti hooldamine ja veaparandused kordades suuremaks peavaluks. Erinevaid raamistikke, milles PHP'd kirjutada on märkimisväärne hulk, SitePoint'i selle aasta uuringu kohaselt on viie populaarseima raamistiku nimed järgmised – Laravel, Symfony2, Nette, CodeIgniter ning Yii2. (Skvorc, 2015) Selle töö käigus tutvustan lähemalt Symfony2'te ning põgusalt ka Yii2'te.

Teema valikul sai määravaks asjaolu, et tööalaselt olen üle kahe aasta erinevate PHP raamistikega lähemalt tuttavaks saanud. Peamiselt olen tegelenud Symfony2'e ning Yii2'ega, kuna Symfony2 on rohkem meelt mööda otsustasin ka sellele antud töö käigus keskenduda, lisaks toon välja ka suuremad erinevused võrreldes Yii2'ga. Lisaks eelnevale ajendas mind teema valikul eesti keelse materjali puudumine. Käesoleva töö sihtgrupina näeb autor eelkõige veebiarendusest huvi tundvaid inimesi.

Käesoleva töö peamiseks eesmärgiks on tutvustada Symfony2 nimelist raamistikku ning ühtlasi pakkuda ka juhendit, kuidas täpselt uut projekti Symfony2 raamistikule üles panna. Lisaks ka mõned lihtsamad õpetused (koos koodinäidetega), kuidas esimesi veebilehti luua. Juhendi kirjutamise hetkel oli viimaseks saadaolevaks versiooniks Symfony 2.7.5, seega on juhendis olevad koodinäited testitud ainult selle versiooniga ning vanemates versioonides võib olla mõningasi erisusi.

Töö esimeses osas keskendutakse põgusa ülevaate andmisele Symfony2'st. Käsitletakse üle kümne aasta pikkust ajalugu ning selle käigus toimunud muudatusi ning arengut. Tuuakse välja Symfony2 raamistiku üheks olulisimaks osaks olevad komponendid ning kogumikud ning nende lühikirjeldused. Võrreldakse teiste populaarsete raamistikega, tuuakse välja

tugevused ning nõrkused ja ka olulised punktid, mida raamistikku valides võiks silmas pidada. Lisaks antakse ülevaade raamistiku arhitektuurist, et oleks töö praktilises osas veidi lihtsam raamistiku struktuuri hoomata ning koodi kirjutada.

Töö teine osa algab detailse kirjelduse ning samm-sammult õpetusega, kuidas oma esimene projekt Symfony2 raamistikus Windows'i platvormil püsti panna. Sellele järgnevates peatükkides tehakse esimesed koodikatsetused kontroller tüüpi failides. Ning töö kolmas peatükk on pühendatud ainult vormidele, mis on enamiku projektide üheks raskuspunktiks veebiarenduse puhul.

1. Symfony2

Symfony2 on PHP'l põhinev veebirakenduste kirjutamiseks loodud raamistik, mis kasutab mudel-vaade-kontroller (*Model-View-Controller*), edaspidi "MVC", nimelist populaarset mustrit. Symfony2-e loojad said inspiratsiooni ning võtsid eeskuju ka teistest veebiraamistikest nagu Ruby on Rails, Django ning Spring. Symfony2 raamistik on hästi tuntud oma paindlikkuse poolest, mistõttu on seda võimalik kasutada nii väikse mahuliste kodulehtede kui suurettevõtetele mõeldud nõudlike rakenduste loomiseks. Samuti kasutavad mitmed projektid, sh Yii2, Joomla!, Laravel, Magento ning paljud teised, komponente Symfony2 raamistikust. Symfony2 on vabatarkvara (avaldatud MIT litsentsi alusel), mille loojaks, ning põhiarendajaks tänaseni on Fabien Potencier. (*Projects*, kuupäev teadmata)

1.1. Ajalugu

Symfony projekt sai alguse üle 10 aasta tagasi, 18. oktoobril 2005. a, mil loodi symfony-project.com nimeline veebileht. Esimene vabalt kasutatav Symfony versioon valmis 2007. aasta jaanuariks, mil avaldati Symfony1.0, mida oli võimalik kasutada alates PHP 5.0 versioonist. Järgneva 2 aasta ning 10 kuu jooksul avaldati versioonid 1.1 kuni 1.4, millest viimase versiooni tugi lõppes 2012. aasta novembris. Ametliku toe kadumisega kaovad ka potentsiaalsete süsteemivigade parandustega seonduvad uuendused. Mis tähendab, et tänasel päeval aegunud Symfony versiooni kasutamine on iga arendaja enda vastutusel ning kriitiliste vigade ilmnemisel raamistiku arendajad enam uusi parandusi ei tee, ega tuge ei paku.

2011. aasta juulist sai alguse pöördelise tähendusega Symfony2.0, mille minimaalseks PHP versiooniks oli juba 5.3.2. Uus Symfony tõi endaga kaasa massiliselt muudatusi, millega kohanemine ei olnud liialt keerukas, kuna endiselt oli kasutusel MVC arhitektuur, mistõttu eelmiste versioonidega tuttavaks saanud arendajad said suurema vaevata uuendustega kaasa minna. Suurematekst muudatusteks võrreldes eelnevatega oli see, et selle asemel, et mitut rakendust ühe projekti alla koondada oli nüüd vajalik iga uue rakenduse jaoks teha uus Symfony2 projekt. Lisaks sellele toimus ka väga suures osas koodi restruktureerimine, palju olulisi komponente kapseldati iseseisvateks üksusteks, analoogselt Symfony1.x versioonides kasutusel olevale pistikprogrammile (*plugin*) võeti kasutusele uus oluline struktuuri

muudatus, mis kandis nimetust „kogumik“ (*bundle*). Lühidalt öeldes andis see muudatus võimaluse ka teistel projektidel vastavalt vajadusele implementeerida kogumikke Symfony2'st. Näiteks, kui mõnes projektis, mis ei olnud Symfony2 raamistikul arendatud, oleks muidu olnud vaja kulutada märkimisväärne hulk arendustunde vajamineva funktsionaalsuse programmerimiseks, siis nüüd võis sagedasti leida lahenduse mõne olemasoleva Symfony2 kogumiku näol ning seda tasuta oma projektis ära kasutada. See omakorda andis Symfony2'le tohutu eelise, kuna hüppeliselt kasvanud kasutajaskonnaga kaasnes ka suurel hulgal uusi arendajaid, kes Symfony2 projekti meeleldi panustasid, nii omapoolsete ideede kui ka parandusettepanekutega. Üheks laialdasemalt kasutatavaks kogumikuks on FriendsOfSymfonyUserBundle. Nagu nimigi reedab ei ole autorite näol tegemist Symfony2 raamistiku ametlike arendajatega vaid raamistikust huvituvate aktiivsemate kommuuniliikmetega. Konkreetne kogumik võimaldab lihtsa vaevaga implementeerida turvalisusega seonduvaid funktsionaalsusi. (*Releases*, 2015)

Alates versioonist 2.2 võeti kasutusele uus versioneerimise praktika, mis tähendab, et igal Symfony versiooni uuendusel on kindel periood, mille jooksul vastavat versiooni ametlikult toetatakse ning uuendusi pakutakse. Laias laastus jagunevad versioneerimised kaheks - standard versioonid ning pikema toega versioonid (ingl. k. LTS version - Long Term Support version). Standard versioonidel on kaheksa kuu pikkune vigade parandamise periood ning 14 kuu pikkune turvavigade parandamise periood. Pikema toega versioonid avaldatakse iga 2 aasta järel ning neil on 3 aasta pikkune vigade parandamise periood ning 4 aasta pikkuna turvavigade parandamise periood. (*Releases,* 2015)

Hetkel on ametlikult toetatavateks versioonideks 2.3 ning 2.7, st ainult neil kahel versioonil on hetkel veel kehtiv hoolduse periood (*maintenance period*). Novembris 2015., tuleb taas märgilise tähtusega Symfony kommuuni jaoks, kuna planeeritud on kahe uue versiooni väljatulek, viimane 2.x harust versioon 2.8 ning kaua oodatud 3.0 versioon, mis tähistab taaskord suuremate muudatuste lävepakku Symfony jaoks. Seoses uue versiooni 3.0'ga kaasneb ka suurem hüpe PHP versiooni nõuetes, nimelt toetab Symfony3 versioone alates 5.5.9'st, mis teeb kindlasti vanemate PHP versiooni kasutavate organisatsioonide jaoks ülemineku mõnevõrra keerukamaks. Uuendused on aga paratamatud ning veebi kiire

arenguga peab sammu pidama ka Symfony projekti arendajate meeskond, eesotsas Fabien Potencier'iga. (*Releases*, 2015)

1.2 . Komponendid ja kogumikud

Alates Symfony2'st on põhiarendaja Fabien Potencier'i eestvedamisel järgitud põhimõtet, et iga suurem kogum loogiliselt kokku kuuluvat koodi peaks olema teistest võimalikult eraldatud, samas olles maksimaalselt dünaamiline ning taaskasutatav nii projekti, kui ka raamistike üleselt. Need loogilised üksused jagunevad omakorda kaheks – komponentideks (*Component*) ning kogumikeks (*Bundle)*. Järgnevalt toon ülevaate Symfony komponentidest, mida antud töö praktilises osas käsitletakse (täielik nimekiri vt lisa 1):

- ***Form*** – Vormi komponent võimaldab arendajal lihtsa vaevaga luua vastavalt vajadusele, kas lihtsaid või ka väga keerukaid vorme. Lisaks pakub vormi komponent ulatuslikke vahendeid vormide töötlemiseks ning muudab vormid lihtsasti taaskasutatavateks, et ei oleks sarnaste vaadete puhul tarvis iga kord uut vormi luua. Kuna vormid on üheks olulisemaks osaks veebiarenduses on neid töö praktilises osas käsitletud terve peatüki jagu.
- ***Templating*** **–** Ehk mallide haldamise komponent pakub tööriistu, mis võimaldavad ehitada arendaja soovile vastava mallimise süsteemi. Pakkudes infrastruktuuri, mille abil on võimalik laadida mallide faile ning valikuliselt monitoorida neid muudatuste suhtes. Lisaks on võimalik malle jagade plokkideks ning omakorda alammallideks, mida saab üksteise sees ära kasutada, et ei peaks ühte ja sama koodi mitmes kohas kirjutama. Mallimist tutvustatakse teise praktilise osa alampeatükkides.
- ***Routing*** **–** Ühendab HTTP päringu kindlalt määratletud konfiguratsiooni muutujate kogumikuks. Seda komponenti kasutatakse mõlemas praktilises peatükis, kuna seda läheb reeglina vaja igas kontrolleris.
- ***Validator*** – Võimaldab valideerimisreeglite defineerimist, kasutades kas XML'i, YAML'i, PHP'd või nn koodi kommentaare (*annotation*). Selle töö raames tutvustame viimast kasutusjuhtu, kus valideerimisreeglid on kirjeldatud phpdoc'is. Vormide valideerimisest on juttu peatükis 3.5.
  (*Components,* 2015)

Symfony standard versioonis esinevad kogumikud:

- ***SensioFrameworkExtraBundle*** - Lisab Symfony Standard Versioonile (*Symfony Standard Version*) reeglistiku ning võimaldab nn koodi kommentaaride abil

defineerida rajad, programmipuhvri, mallid ning turvalisusega seonduva. Selle tulemiks on tunduvalt lakoonilisemad kontrollerid (*concise controller*).

- ***SensioGeneratorBundle*** - Lisab mitmeid konsoolikäske genereerimaks koodi põhjasid (*skeleton*) nagu kogumikud, vormi klassid ning Doctrine'i olemitel põhinevad CRUD (*Creat Read Update Delete* - Loo Loe Uuenda Kustuta) kontrollerid.

  (*Bundles*, kuupäev teadmata)

Lisaks standard versioonis esinevatele kogumikele on Symfony'st rääkides vajalik välja tuua ka Doctrine'i projektiga seonduvad kogumikud. Lühidalt selgitades on Doctrine objekt-relatsiooniilne (*ORM - Object-Relational Mapping*) andmebaasi abstraktsiooni vahekiht (*DBAL - Database Abstraction Layer*), ehk vahend, mis kapseldab endas koodi, mille abil saab andmebaasis olevaid andmeid manipuleerida ilma ise puhast SQL'i kirjutamata. Kuigi Doctrine on Symfony'st täiesti eraldiseisev ning Symfony raamistikus projekte kirjutades ei ole tingimata vaja Doctrine'i kasutada on see siiski Symfony Standard Versiooni puhul vaikimisi integreeritud.

Järgnevalt mõned näited enimkasutatavatest Doctrine'i projekti kogumikest:

- ***DoctrineBundle*** - Integreerib Doctrine'i ORM'i ning DBAL'i projektid Symfony rakendusse.
- ***DoctrineFixturesBundle*** - Võimaldab Doctrine'i kaudu luua ning laadida fikstuure (*fixture* – andmete kogum andmebaasi initsialiseerimiseks). Andme fikstuure kasutatakse kindla andmekogumi laadimiseks andmebaasi. Neid andmeid võib kasutada nii testimiseks kui ka esmasteks rakenduse andmeteks.
- ***DoctrineMongoDBBundle*** - Integreerib MongoDB NoSQL'i andmebaasi kasutades ODM'i (*Object Document Mapper* – objekti dokumenti vastendaja), mis sarnaneb paljus Doctrine2 ORM'iga nii filosoofia kui ka toimimis metoodika poolest.
- ***DoctrineMigrationsBundle*** - Võimaldab programmiliselt paigaldada uute andmebaaside skeeme (*database schema*) turvalisel, lihtsal ning standardiseeritud viisil.

  (*Bundles*, kuupäev teadmata)

1.3 . Võrdlus teiste raamistikega

Töötades tarkvaraarendusega tegelevas firmas tuleb tihtilugu võtta vastu otsus, milline PHP raamistik oleks konkreetse projekti jaoks sobivaim. Tavaliselt on otsuse tegemisel kindlad mõjurid - peaarendaja eelistus, hankedokumendis esinevad nõuded ning projekti keerukus. Kahtlemata on ka erandeid, näiteks juhul, kui projektiks on mõne väiksema osa juurde kirjutamine olemasolevale süsteemile, mis kasutab juba konkreetset raamistikku. Alljärgnevalt toon välja mõned punktid, mida Symfony2 raamistikku võrreldes teistega valides silmas võiks pidada ning mis ehk valiku tegemist mõnevõrra lihtsustada võiksid.

Alustuseks mõned sarnasused, mida isikliku kogemuse põhjal võib paljude populaarsete raamistike - Symfony2, Yii2, CodeIgniter jpt - puhul tänapäeval välja tuua:

- Kasutatakse Mudel-Vaade-Kontroller mustrit, mis kirjeldab kolmetasemelist arhitektuuri.
- Konfiguratsiooni failide kasutamine, milledes kirjeldatakse kõige olulisemad metaandmed, nt andmebaasi asukoht ja põhi URL'id.
- Palju läbimõeldud ning testidega kaetud abiklasse, lihtsustamaks arendajate tööd.
- Võimaldavad raamistiku baasfunktsionaalsust vastavalt vajadusele muuta või spetsiifilistele vajadustele vastavaid muudatusi teha. Samuti võimalus oma abiteekide ning abifunktsioonide kirjutamiseks.
- Lisaks on sisse ehitatud programmipuhvri kasutus, mitmete andmebaaside kasutamine ühes rakenduses ning vormide tugi.

1.3.1 . Yii2 vs Symfony2

Järgnevalt toon välja olulisemad plussid ja miinused kahe populaarse raamistiku, Yii2'e ning Symfony2'e võrdluses.

Yii2 plussid:

- Lihtsalt paigaldatav ning uute funktsionaalsuste lisamine on mõnevõrra kiirem.
- Üheks suurimaks plussiks on erinevad turvalisuse probleeme vältida aitavad elemendid. Mõned näited: Loo Salasõna Räsi (*Generate Password Hash*), Genereeri Suvaline Võti(*Generate Random Key*) ja Valideeri Salasõna(*Validate Password*). Nimetatud funktsionaalsused muudavad rakenduse turvaliseks muutmise märkimisväärselt lihtsamaks.

- Märksa lühem õppimiskõver.

Yii2 miinused:

- Ajax'i (Asynchronous JavaScript and XML – asünkroonne JavaScript ja XML) funktsionaalsuse puhul jätab dokumentatsioon soovida, mistõttu võib algajal arendajal tekkida mõningaid raskuseid.
- Suurema rakenduse jaoks on oluliseks osaks erinevad sisseehitatud laiendused, mida vaikimisi Yii2'e installeerimisel kaasa ei tule.
- Test juhitud arenduse (*Test Driven Development*) praktiseerimine on Yii2 puhul tunduvalt keerulisem kui Symfony2 puhul.

  (Svatok, 2015)

Symfony2 plussid:

- Kodeerimisstandardi kasutus - Yii arendajate tiim kasutab oma kohandatud kodeerimisstiili, seevastu Symfony puhul on kasutusel laialt levinud standard PSR-2, mis teeb projektide vahel liikumise mugavamaks.
- Symfony2'e ORM vs Yii2'e AR (*Active Record*) - AR võib olla lihtne ning meeldiva funktsionaalsusega algajale arendajale, kuid on keerukamate olukordade jaoks liialt lihtsustatud. Samuti puudub piisav eraldatus päringute (*query*) ja olemite (*entity*) vahel. Piisavat eraldatust pakub aga Symfony2'ga vaikimisi kaasa tulev ORM, Doctrine2
- Symfony2 üheks peamiseks eeliseks on märkimisväärne arendajate kommuun, kes panustavad regulaarselt Symfony projekti, luues uusi komponente, mis on valdavalt kaetud ka PHPUnit'i testidega.
- Funktsionaalsete testide kirjutamine on küllalt lihtsa vaevaga tehtav, kasutades Selenium'i nimelist tööriista.

Symfony2 miinused:

- Peamiseks miinuseks on kindlasti tunduvalt pikem õppimiskõver, millega tuleks enne uue projekti alustamist kindlasti arvestada. Tänu põhjalikule dokumentatsioonile ei ole see siiski ületamatu raskus.
- Uute väikeste arenduste kirjutamine võtab raamistiku keerukuse tõttu rohkem aega.

  (Svatok, 2015)

Kahe raamistiku eelistest ja vajaka jäämistest joonistuvad välja mõned põhimõtted, mida uut projekti alustades võiks arvesse võtta. Kui projekti maht ei ületa 2-3 kuud võiks kaaluda Yii2 kasutamist, mida on võimalik küllalt väikse ajaga omandada ning väiksemaid muudatusi kiiremini paigaldada. Keerukamate ning pikemaajaliste projektide puhul tasuks kindlasti kaaluda Symfony2'e kasutusele võttu. Tugev arendajate kommuun, palju testidega kaetud komponente ning raamistiku üldine arhitektuur võiksid üles kaaluda mõne võrra pikema õppimisele kuluva aja.

1.4 . Raamistiku arhitektuur

Symfony kataloogide struktuur on võrdlemisi paindlik ning rangeid nõudeid, millist struktuuri uue projekti puhul järgima peab ei ole. Siiski on olemas soovituslik arhitektuur, mida järgivad valdav osa kommuuni poolt kirjutatud komponentidest, et oleks võimalikult lihtne erinevaid komponente omavahel kasutada. Symfony poolse soovituse kohaselt jaguneb projekti kood neljaks osaks:

1. *app/* - selle kataloogi alla kuuluvad konfiguratsiooni failid, mallid ning tõlked.
2. *src*/ - siia alla lisatakse kogu projekti PHP kood.
3. *vendor*/ - kolmandate osapoolte sõltuvused (*dependecies*).
4. *web*/ - avaliku poole juurkataloog. Siia alla lisatakse kõik avaliku poole staatilised failid, nagu pildid, laadilehed (*style sheets*) ja JavaScript'i failid. Siin paiknevad ka eessüsteemi kontrollerid (*front controllers*).
   (*The architecture*, 2015)

2 . Paigaldamine ja esimesed mallid

Et Symfony2 raamistikust paremat ülevaadet saada, on oluline lisaks põgusale teoreetilisele ülevaatele lisada ka praktilisi näiteid. Praktilise osa alustuseks on sobilik pakkuda samm-sammulist õpetust, kuidas Symfony2'te oma arvutisse paigaldada. Õpetuse paremaks jälgimiseks on lisatud ka ekraanitõmmised, mis loodetavasti potentsiaalselt segadust tekitavaid kohti vältida aitavad. Kui rakendus on edukalt paigaldatud, saab järgmistes alapeatükkides lihtsamate näidete abil esimesi koodi kirjutamise katsetusi teha ning väljundit enda veebilehitsejas ka visualiseerida.

2.1 . Raamistiku paigaldamine

Symfony paigaldamiseks on kaks erinevat moodust: installeerides Symfony installeerija, mis on ka Symfony enda poolt ainus soovituslik meetod või kasutades *Composer*'i nimelist komponenti.

Selle paigaldamise õpetuse käigus kasutan ma soovituslikku varianti, ehk Symfony installeerijat. Ainsaks eelduseks Symfony installeerija kasutamiseks on PHP 5.4 või uuema versiooni olemasolu. Vastavalt kasutatavale operatsioonisüsteemile on Symfony paigaldamine veidi erinev, selles paigaldusjuhendis kasutatav operatsiooni süsteem on Windows 10.

Esimese asjana on vajalik avada "Command Prompt" nimeline programm, edaspidi "käsuviip". Kontrollimaks, mis versioon PHP'st on arvutisse paigaldatud saab kasutada käsku "php -v" mis kuvab arvutisse paigaldatud PHP versiooni. (vt Joonis 1)

Joonis 1. PHP versiooni tuvastus

Nagu pildilt näha, on minul arvutisse paigaldatud PHP versiooniga 5.4.12, mis ongi minimaalseks nõudeks, et Symfony installeerijat kasutada.

Teiseks on vaja käivitada käsk, mis laeks alla Symfony installeerija, seda illustreerib Joonis 2.

Joonis 2. Symfony installeerija alla laadimine.

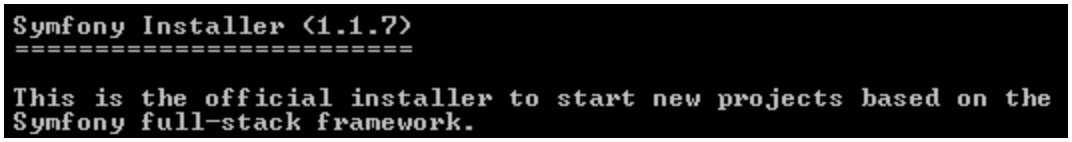
Nüüd peaks samas kataloogis "dir | FIND "symfony"" käsku käima lastes olema näha uut faili nimega symfony, käsuviibalt näeb see välja nagu joonisel 3.

Joonis 3. Faili leidmine kataloogist.

Seejärel võiks alla laaditud Symfony installeerija käivitada, kasutades käsklust "php symfony", joonis 4:

Joonis 4. Symfony installeerija käivitamine.

Kui kõik õnnestus, peaks käsuviip kuvama teksti, nagu on näha joonisel 5.



Joonis 5. Installeerija abitekst.

Nüüd on Symfony installeerija poolt pakutav funktsionaalsus töövalmis ning järgmise sammuna tuleks tekitada uus Symfony projekt. Kui kasutada kõige lihtsamat varianti, ja käskida installeerijal luua uus projekt ilma versiooni numbrit täpsustamata, siis luuakse uus projekt viimase stabiilse versiooniga. Kui on soov luua projekt kasutades viimast pigaaegse toega versiooni (*LTS*), siis tuleb käsu lõppu lisada argumendina lühend *lts*. Antud paigaldusjuhendis kasutama kõige lihtsamat varianti, ilma täpsustava versiooni numbri või lisa argumentideta, mis käsuviibalt näeb välja nagu joonisel 6., kus "demo_projekt" on projekti nimeks.

```
C:\Users\User\projektid>php symfony new demo_projekt
```

Joonis 6. Uue projekti loomise käsk.

Eelneva käsu käivitamine laadis alla Symfony projekti failid, ning tekitas "projektid" kausta alla uue projekti kausta nimega "demo_projekt". Lisaks trükiti käsuviibale veel mõned olulised punktid, alustades kõige tähtsamast: "Symfony 2.7.5 on edukalt installeeritud". Lisaks andis installeerija mõned soovitused, kuidas jätkata:

1. "Mine vastloodud kataloogi 'demo_projekt'"
2. Konfigureeri oma rakendus "parameters.yml" nimelises failis
3. Käivita oma rakendus:
    a. Kasutades käsku "php app/console server:run"
    b. Navigeerides veebilehitsejaga http://localhost:8000 URL'ile
4. Dokumentatsiooniga tutvumiseks on lehekülg "http://symfony.com/doc"

Ekraanitõmmis käsuviibast on näha joonisel 7.

```
Downloading Symfony...

Preparing project...

OK  Symfony 2.7.5 was successfully installed. Now you can:

    * Change your current directory to C:\Users\User\projektid\demo_projekt

    * Configure your application in app/config/parameters.yml file.

    * Run your application:
        1. Execute the php app/console server:run command.
        2. Browse to the http://localhost:8000 URL.

    * Read the documentation at http://symfony.com/doc
```

Joonis 7. Projekti loomise käsu väljund käsuviibal.

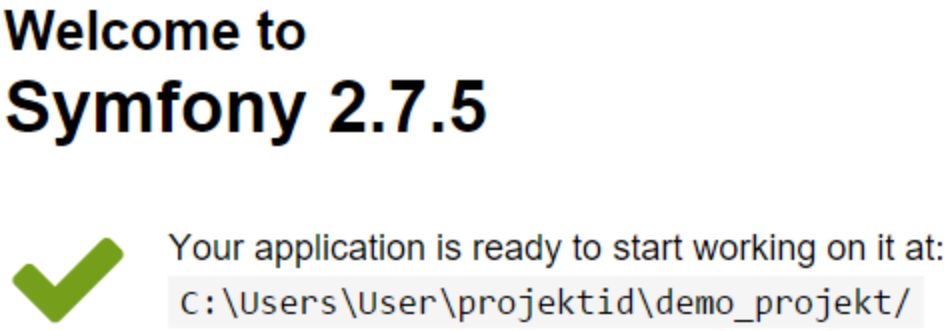
Järgmisena liigun kataloogi "demo_projekt" kasutades käsklust "cd demo_projekt" ning käivitan punktis 3.a. nimetatud käsu. (vt Joonis 8.)

```
C:\Users\User\projektid>cd demo_projekt

C:\Users\User\projektid\demo_projekt>php app/console server:run
Server running on http://127.0.0.1:8000
```

Joonis 8. Uude kataloogi navigeerimine ning serveri käimatõmbamine.

Käsuviip annab teada, et serveri käima tõmbamine õnnestus, ning nüüd on aeg kontrollida, kas punktis 3.b. viidatud veebileht avaneb. Selleks navigeerin "http://localhost:8000/" nimelisele lehele, kust avanevat pilti illustreerib Joonis 9.



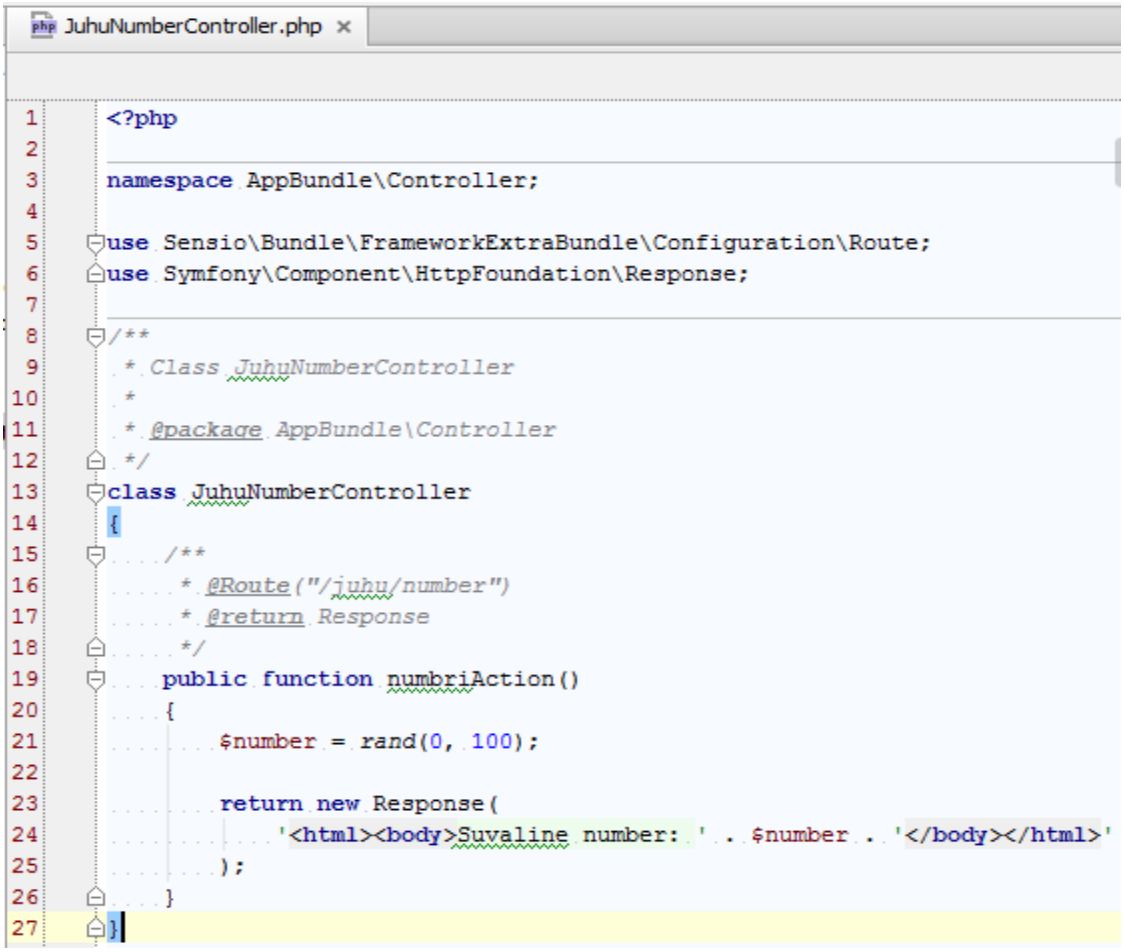
Joonis 9. Symfony tervitustekst veebilehitsejas.

Veebilehitsejast avanev pilt annab teada, et projekti püsti panemine õnnestus ning nüüd on aeg hakata arendama. (*Installation*, kuupäev teadmata)

2.2 . Esimese vaate loomine

Esimese vaate loomiseks on vaja läbi teha kaks etappi. Esiteks on vaja luua rada (*route*), mis viitaks kontrollerile ning teiseks on vaja luua kontroller. Kontroller on arendaja poolt kirjutatud funktsioon, mis ehitab veebilehe valmis. Vajalik on sissetulevast päringust (*request*) võtta vastu informatsioon ning seda kasutades luua uus Symfony *Response* objekt, mis sisaldab endas kas HTML sisu, JSON sõnet vms. (*Page creation*, kuupäev teadmata)

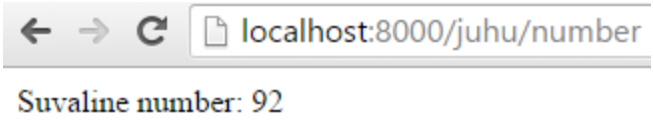
Nagu tihtipeale erinevates näidetes, teen ka mina esimesena veebilehe, mis kuvab suvalist arvu vahemikus nullist sajani. Selleks on vaja kõigepealt navigeerida kausta, kus asub projekti kood, mis minu näite puhul on järgmise rajaga "C:\Users\User\projektid\demo_projekt\src\AppBundle\Controller". Nüüd on vajalik luua uus PHP fail nimega "JuhuNumberController.php", selleks võib kasutada vabalt valitud tekstiredaktorit, käesoleva tööraames on kasutatud PhpStorm 8.0.3'e.
Loodud kontrollerit saab näha joonisel 10.



Joonis 10. Kontroller klass ning esimene meetod.

Joonis 10. rida 3 on kirjeldatud nimeruum, mis viitab kaustale, milles kontrolleri fail asub. Sellele järgnevatel ridadel 5-6 imporditakse *use* märksõna abil sisse kaks erinevat klassi. Esiteks *Route*, mida kasutatakse real 17 ning kus on näha, kuidas kasutatakse Symfony's rada, mis tähistab internetiaadressi lõpu osa ("/juhu/number"). Teiseks *Response*, mida kasutatakse veebilehitsejasse vastuse tagastamiseks. Oluline on ära märkida, et kontrollerid koosnevad reeglina tegevustest (*action*), eelmisel joonisel rida 19. Iga tegevus tagastab reeglina vastuse, näidisfailis rida 23, mida kuvatakse omakorda veebilehitsejas tavakasutajale.

Navigeerides vastloodud veebiaadressile "http://localhost:8000/juhu/number", on näha, et esimene katsetus õnnestus, ning lehel kuvatakse suvalist arvu nullist sajani. Minu näites juhtus selleks numbriks olema 92, mida illustreerib joonis 11.



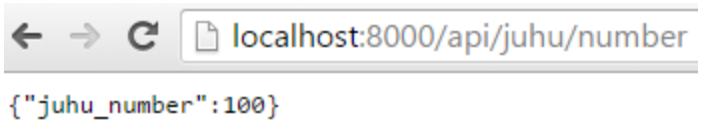
Joonis 11. Esimese meetodi väljund veebilehitsejas.

Selline on kõige lihtsam näide veebilehe loomisest Symfony2 raamistiku abil, mis ei tohiks raskusi valmistada ka alles esimesi katsetusi tegevale arendajale.
Kusjuures lisaks ühele kontrollerile, mis ma selles näites projekti koodi lisasin, genereeris Symfony ise 7359 faili, mis paiknevad kokku 1506 erinevas kaustas.

2.3 . JSON vastusega vaade

Vastuse objekt, mille kontrolleris paiknev *action* meetod tagastab võib endas sisaldada lisaks eelnevas näites kasutatud HTML'ile ka JSON'it. Järgnevalt teen näite, kuidas juhunumbri genereerimine ning veebilehele kuvamine võiks Symfony's välja näha.

Kuna JSON vastuseid kasutatakse tihtilugu API-liideste puhul, siis võiks sobilikuks meetodi nimeks olla "apiNumberAction(). Samuti tuleb lisada uus rada, mille abil oleks võimalik veebilehitsejast uue tegevuse tulemit visuaalselt näha. Rajaks sobiks antud juhul näiteks "api/juhu/number", mis veebilehitseja aadressiribale kirjutades näeb välja nagu joonisel 12.



Joonis 12. JSON vastuse kuvamine veebilehitsejas.

Lisan ka loodud *action*'i koodi, mis asetseb analoogselt numberAction() meetodile JuhuNumberController.php nimelises failis. (Joonis 13.)

```
29      /**
30       * @Route("/api/juhu/number")
31       * @return Response
32       */
33      public function apiNumberAction()
34      {
35          $data = [
36              'juhu_number' => rand(0, 100)
37          ];
38
39          return new Response(
40              json_encode($data),
41              200,
42              ['Content-Type' => 'application/json']
43          );
44      }
```

Joonis 13. JSON'it tagastava meetodi kood.

Taolist vastuse tagastamist võib JSON'i puhul kohata üsna sagedasti, mistõttu Symfony pakub lihtsustamiseks vastuse objekti nimega JsonResponse. Kasutades eelnimetatud sisseehitatud vastuse tüüpi, saab eelneval joonisel vastuse objekti (*Response*) koostada kolme argumendi asemel vaid ühega nagu on näha jooniselt nr 14.

```
40          return new JsonResponse($data);
```

Joonis 14. JsonResponse objekti tagastamine.

Lühidalt öeldes teeb JsonResponse kogu tülika töö arendaja eest ära, määrates vaikimisi sisu tüübi (*content-type*), vastuse staatuse koodi (*response status code*) ning viib etteantud *$data* massiivi JSON'i kujule.

2.4 . Dünaamilised URL'i mustrid

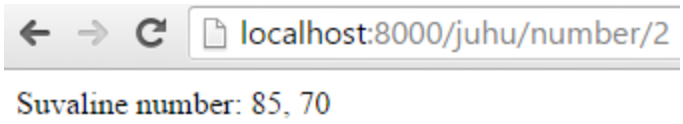
Eelnevates näidetes saime selgeks, kuidas mõnel erineval moel juhuslikku numbrit vahemikus 0 – 100, kontrollerit, *action*'it, rada ning *Response* objekti kasutades veebilehitsejas kuvada. Järgmise sammuna lisame rajale („/juhu/number") lisaks muutuja, mida Symfony's kutsutakse nimega *placeholder*, ning selle abil on võimali mõningasel määral juhtida kontrolleri tegevust. Järgmises näites on muutuja eesmärgiks määrata, mitut suvalist numbrit veebilehitsejas välja kuvatakse. Selleks tuleb muuta raja struktuuri järgmiselt - „/juhu/number/{kogus}". Uuenenud numbriAction() meetod vajab samuti sisendparameetrit,

et oleks koodi tasemel võimalik kasutaja poolset sisendit aadressiribalt kinni püüda. Uuel kujul action'i koodi illustreerib alljärgnev joonis 15.

```
16      /**
17       * @Route("/juhu/number/{kogus}")
18       * @return Response
19       */
20      public function numbriAction($kogus)
21      {
22          $numbrid = [];
23
24          for ($i = 0; $i < $kogus; $i++) {
25              $numbrid[] = rand(0, 100);
26          }
27
28          $numbriteNimekiri = implode(', ', $numbrid);
29
30          return new Response(
31              '<html><body>Nimekiri suvalistest numbritest: ' . $numbriteNimekiri . '</body></html>'
32          );
33      }
```

Joonis 15. Dünaamilise URL mustri kasutusnäide kontrolleri meetodis numbriAction()

Uuenenud rada on näha real 17 ning muutunud meetodi kirjeldust, koos parameetriga kujutab rida 20. Ridadel 24-26 koostatakse sisend parameetri põhjal massiiv, mille elementide arv on võrdne parameetri *$kogus* väärtusega. Real 28 moodustatakse massiivist komadega eraldatud sõne, et seda oleks võimalik hiljem veebilehitsejas kuvada. Näites kasutasin „kogusena" arvu 2, mis brauserist nägi välja nagu joonisel 16.



Joonis 16. Mitme juhusliku numbri väljund veebilehitsejas.

2.5 . Malli visualiseerimine kasutades teenusehoidlat

Eelmiste näidete põhjal võib väita, et üsna sageli on kontrolleri ülesandeks tagastada mingil kuhul HTML'i. Lihtsamate näidete puhul võib ka otse kontrollerisse HTML'i kirjutada, et kasutajal saaks vähesema vaevaga esimesi katsetusi teha, ilma et oleks vaja failide vahel projektis liikuda. Kuna Symfony jälgib lisaks paljudele teistele levinud mustritele ka MVC mustrit, siis selles näites saame tuttavaks selle mõiste tagumise poolega, ehk eraldame vaate või ka malli, kontrollerist.

Esimese asjana on vaja muuta klassi definitsiooni ning sellega seonduvalt importida Symfony baaskontrolleri klass. Koodipoolelt illustreerib neid muudatusi joonis 17.

```
8   use Symfony\Bundle\FrameworkBundle\Controller\Controller;
9
10  class JuhuNumberController extends Controller
```

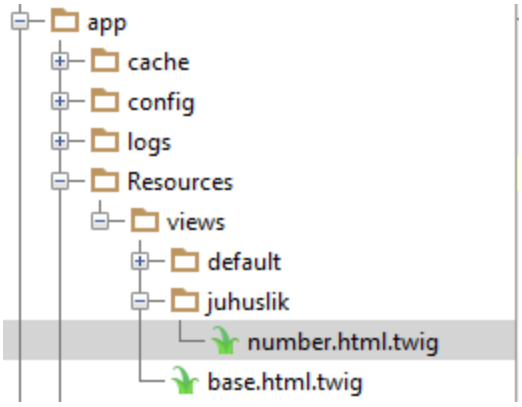
Joonis 17. Baaskontrolleri import ning kontrolleri definitsiooni muudatus.

Baaskontrollerit on vaja selleks, et saada ligipääs Symfony hoidlale (*container*), mis omakorda pakub ligipääsu teenustele (*service*). Et malli visualiseerida, on vaja kasutada teenust nimega mallide haldamine (*templating*). Teenuseid saab hoidla käest küsida *get()* meetodi abil, kus ainsaks parameetriks on teenuse nimi. Koodi näide, mis malli visualiseerimist demonstreerib on joonisel 18.

```
24          $numbriteNimekiri = implode(', ', $numbrid);
25
26          $html = $this->container->get('templating')->render(
27              'juhuslik/number.html.twig',
28              ['numbriteNimekiri' => $numbriteNimekiri]
29          );
30
31          return new Response($html);
```

Joonis 18. Hoidla kasutamine malli visualiseerimiseks.

Real 26 küsitakse hoidla käest mallimise teenust, ning mallimise teenusele öeldakse, et on vaja visualiseerida „juhuslik“ kaustas asuv mall nimega „number.html.twig“ ning lisaks antakse mallile kaasa muutuja „numbriteNimekiri“. Kuna hetkel ei ole sellist kausta, ega ka faili, siis veebilehitseja värskendamisel antakse veateade sisuga „Unable to find template ’juhuslik/number.html.twig’“ – tlk malli nimega „number.html.twig“ ei õnnestunud kaustast nimega „juhuslik“ leida. Et veateatest vabaneda on vaja tekitada uus kaust nimega „juhuslik“. Kuna kõik mallid ning konfiguratsiooni failid käivad *app/* kausta all, siis ongi vaja kataloogi puus natuke üles liikuda, kuni jõuad *app/ bin/* ning */source* kataloogideni. Edasi on vaja liikuda app->Resources->views nimelisse kausta ning seal tekitada uus kaust nimega „juhuslik“. Kui see on tehtud, siis järgmisena on vast loodud kausta vaja lisada mall nimega „number.html.twig“. Eelneva tulemusena peaks kataloogipuu struktuur olema selline nagu joonisel 19.



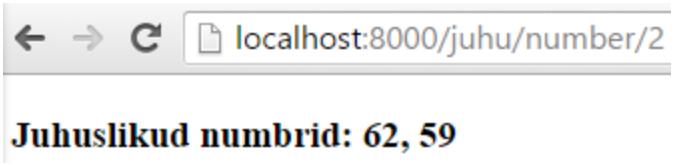
Joonis 19. Kataloogipuu näide.

Joonisel 20 on näide, milline võiks malli sisu välja näha, et kuvada kaasa antud muutuja abil juhuslikke numbreid „<h1>“ sildi sees.

```
1    {% extends 'base.html.twig' %}
2
3    {% block body %}
4        <h3>Juhuslikud numbrid: {{ numbriteNimekiri }}</h3>
5    {% endblock %}
```

Joonis 20. Esimene mall.

Kui uus mall koos sisuga on lisatud võib uuesti veebilehitseja lahti võtta ning lehte värskendada, seekord peaks veateade kadunud olema ning tulemus välja nägema nagu joonisel 21.



Joonis 21. Malli visualiseerimine veebis.

Nüüd, kui esimene malli visualiseerimine on õnnestunud, vaatame uuesti üle, kuidas saaks koodi lihtsamaks. Nimelt võimaldab baaskontrolleri kasutamine nii mitmegi meetodi väljakutse kapseldada üheks. Joonisel 22 on kujutatud koodinäide, mis teeb ära sama töö, mis joonisel 18. kuid mõnevõrra lühemalt ning mugavamalt.

```
26        $html = $this->render(
27            'juhuslik/number.html.twig',
28            ['numbriteNimekiri' => $numbriteNimekiri]
29        );
```

Joonis 22. Optimeeritud koodinäide malli visualiseerimisest.

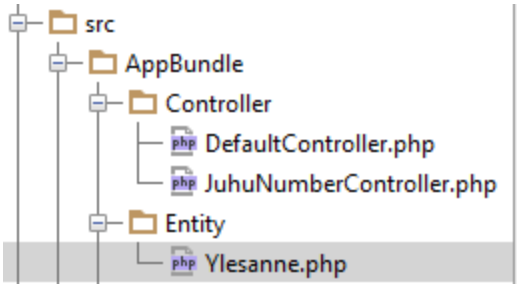
Tänu baaskontrolleri kasutamisele saab ära jätta nii hoidla väljakutse, kui ka mallimise teenuse väljakutse, kuna vaikimisi tehakse seda *$this->render()*'i abil.

3 . Vormid

Eelmises peatükis uurisime, kuidas ehitada lihtsamaid kontrollereid ning malle. Selles peatükis uurime, milline oleks järgmine samm, täielikult MVC mustrit kasutava näite puhul. See tähendab, et lisaks malli eraldamisele kontrollerist on vaja lisaks tekitada klass, mida kutsutakse mudeliks või ka olemiks (*entity*) Symfonys. Seda tehes on olemas kõik kolm MVC mustri komponenti - mudel ehk olem, vaade ehk mall ning kontroller. Järgnevates näidetes uurime, kuidas ühe veebi arendaja jaoks sagedasema ning ka keerulisema ülesandega Symfony2 näitel toime tulla. Selleks ülesandeks on HTML vormide kuvamine, salvestamine, valideerimine ja kõik muu, mis vormidega kaasneb. Järgnevates näidetes on aluseks võetud vajadus luua rakendus, mille eesmärgiks on ehitada ülesannete nimekiri. Alljärgnevates alapeatükkides välja toodud näidete koostamisel on kasutatud Symfony ametlikku dokumentatsiooni (*Forms*, kuupäev teadmata).

3.1 . Olemi loomine

Kuna kasutajatel on vaja ka uusi ülesandeid luua ning muuta, on meil vaja tekitada Ylesande nimeline php klass, mis hoiab endas ühe ülesande jaoks vajalikke andmeid. Selleks, et luua esimest olemi klassi on vaja *Controller* kaustaga samale tasemele luua kaust nimega *Entity*, kuna see on vaikimisi koht, kuhu olemite klassid lisatakse ning kust neid otsitakse. Olemi klassis ei ole iseenesest midagi erilist ning see näeb välja nagu tavaline PHP objekt. Kui *Entity* nimeline kaust on loodud, võib selle alla lisada esialgu tühja PHP faili nimega Ylesanne.php. Kui eelnev on õnnestunud, peaks kataloogi puu välja nägema nagu joonisel 23.



Joonis 23. Muutunud kataloogipuu koos olemi klassiga.

Järgmisena on vaja olemi klassi lisada kaks klassi muutujat, kaks set() meetodit ning kaks get() meetodit. Kuidas seda täpsemalt teha, on näha jooniselt 24.

Joonis 24. Olemi klass nimega Ylesanne.php.

Ridadel 6 ja 7 on näha kahe kaitstud (*protected*) muutuja deklaratsiooni, *$ylesanne* ja *$tahtaeg*. Et väljaspoolt klassi kahe muutuja väärtuseid välja saaks kutsuda, või üle kirjutada on olemas meetodid *getYlesanne(), setYlesanne()* ning *getTahtaeg, setTahtaeg()*. Get meetodid küsivad klassi käest kaitstud muutujate väärtuseid ning set meetodid omistavad kaitstud muutujatele uued väärtused. Hetkel ei ole olemi klassiga rohkem midagi vaja teha, hiljem lisame siia ka valideerimiseks vajaliku koodi.

3.2 . Vormi loomine

Pärast olemi loomist, on järgmisena vajalik luua ning visualiseerida HTML vorm. Lihtsuse mõttes ehitame vormi valmis kontrolleri *action*'i sees, hiljem tõstame ka vormi loomise osa

eraldi klassi. Vormi loomiseks kasutame projekti initsialiseerimise käigus loodud kontrollerit nimega „DefaultController.php“. Nagu malli visualiseerimise peatükis, on ka sel korral vajalik importida baas kontroller klassi, mis annab vahendid vormi visualiseerimiseks ning loomiseks. Samuti on vaja importide Ylesanne olem, kuna vormi visualiseerimiseks on esmalt vaja initsialiseerida olemi objekt ning anda talle sisse mingisugused esialgsed andmed. Lisaks on vaja importida *Route* objekt, loomaks vajalikku rada, et tulemust hiljem veebilehitsejast näha oleks võimalik. Joonisel 25. on näha vajalikud read importimaks eelnimetatud objekte.

```
5   use AppBundle\Entity\Ylesanne;
6   use Symfony\Bundle\FrameworkBundle\Controller\Controller;
7   use Symfony\Component\Routing\Annotation\Route;
```

Joonis 25. Vajalikud impordid.

Järgmisena on vaja luua uus meetod, mille sisse funktsionaalsust lisama hakata ning määrata meetodile ka rada. Enda näites andsin meetodile nimeks „uusYlesanneAction()“ ning määrasin raja selliselt: *@Route(„/uus/ylesanne“)*. Eelnevat illustreerib joonis 26.

```
11      /**
12       * @Route("/uus/ylesanne")
13       */
14      public function uusYlesanneAction()
```

Joonis 26. Meetod ja rada.

Järgmisena on vaja initsialiseerida objekt *Ylesanne* ning väärtustada objekti muutujad kasutades *set()* meetodeid. Eelnevat kirjeldab joonis 27.

```
16          $ylesanne = new Ylesanne();
17          $ylesanne->setYlesanne('Kirjuta blogi postitus');
18          $ylesanne->setTahtaeg(new \DateTime('tomorrow'));
```

Joonis 27. Objekti initsialiseerimine ning väärtustamine.

Real 16 luuakse uus objekt, järgneval real väärtustatakse muutuja *ylesanne* väärtusega „Kirjuta blogi postitus“. Veidi keerukam on rida 18, milles väärtustatakse klassi muutuja *tahtaeg*. Kuna tegemist on *DateTime()* tüüpi muutujaga on vaja kõigepealt initsialiseerida *DateTime()* objekt, andes talle väärtuseks „*tomorrow*“ – tlk „homme“, mis teisendatakse kuupäeva vormingusse sisemiselt ümber. Loodud Ylesanne() objekt on nüüdseks kahe *set()* meetodi abil väärtustatud ning muutujale *$ylesanne* omistatud. Eelmainitud muutujat saame kasutada vormi loomisel. Siin tulebki kasutusele eelnevalt imporditud baaskontroller, mille

funktsionaalsust on võimalik hõlpsasti vormi loomisel kasutada. Funktsionaalsus, mida vormi loomiseks vaja läheb on kapseldatud meetodisse nimega *createFormBuilder()*, mis võimaldab ükshaaval vormile välju lisada ning lõpuks tagastada objekti nimega „Form“. Koodi poolelt illustreerib seda joonis 28.

```
20        $vorm = $this->createFormBuilder($ylesanne)
21            ->add('ylesanne', 'text')
22            ->add('tahtaeg', 'date')
23            ->add('save', 'submit', ['label' => 'Loo Ülesanne'])
24            ->getForm();
```

Joonis 28. Vormi loomine kontrolleris.

Real 20 kasutame *createFormBuilder* meetodit, mis saab sisendiks eelnevalt koostatud muutuja *$ylesanne*, mille põhjal saavad järgmisena lisatavad väljad oma algväärtused. Real 21 lisame välja nimega „ylesanne“ ning määrame tüübiks „text“. Real 22 lisame välja „tahtaeg“ ning määrame tüübiks „date“ (kuupäev). Järgmisel real lisame salvestamiseks vajamineva „submit“ tüüpi nupu, ning lisame talle nimeks „Loo Ülesanne“. Ning viimaseks küsime vormi loojalt (Form Builder), vormi objekti tagasi (*getForm()*), et seda kasutades hiljem brauseris vormi visualiseerida. Sellega vormi loomine koodipoolelt ka lõppeb, järgmises peatükis käsitleme vormi visualiseerimist.

3.3 . Vormi visualiseerimine

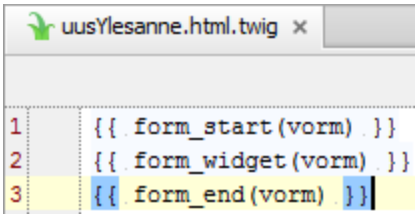
Vormi visualiseerimine selle kõige lihtsamal kujul on Symfony2’s väga lihtsaks tehtud, piisab vaid mõnest reast koodist ning ongi võimalik veebilehitsejas salvestatavat vormi kuvada. Järgmisena on vaja lisada veel mõned täiendused eelmises peatükis loodud meetodile *uusYlesanneAction()*. Vajalikud muudatused on välja toodud joonisel 29.

```
32        return $this->render('default/uusYlesanne.html.twig', [
33            'vorm' => $vorm->createView(),
34        ]);
```

Joonis 29. Vormi visualiseerimine kontrollerist.

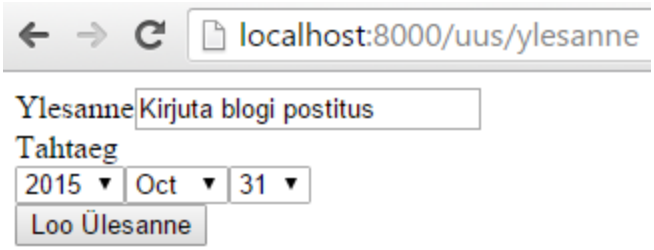
Real 32 on näha, et vormi visualiseerimiseks peab meetod tagastama uue malli nimega „uusYlesanne.html.twig“, mis asub „default“ nimelises kaustas. Lisaks sellele antakse mallile kaasa parameeter nimega „vorm“, mida läheb vormi väljade kuvamiseks vaja. Parameetri väärtustamisel on näha, et kasutatakse meetodit *Form::createView()*, mis teisendab sisemiselt

Form objekti veebis kuvamiseks sobivale kujule. Lisaks muudatustele kontrolleris on vaja lisada ka „deafult“ kausta uus mall ning lisaks ka mõned read koodi, mis ütleksid, kuidas vormi on vaja kuvada. Eelnevat illustreerib järgnev joonis 30.



Joonis 30. Vormi mall.

Esimesel real on näha, kuidas kasutatakse sisendparameetrit „vorm“, et defineerida vormi algus ning ühtlasi tekitatakse selle abil vormi alguse html silt (<form>). Järgmisel real tekitatakse kõik vormi väljad ning lisaks ka väljade pealkirjad ning vajadusel valideerimisest tulenevad veateated. Viimasena tekitatakse vormi lõpu silt (</form>). Kui kõik eelnev on tehtud peaks olema võimalik veebilehitsejast ka vormi näha. Selleks on vaja navigeerida aadressile http://localhost:8000/uus/ylesanne ning sealt avanev pilt võiks olla midagi taolist nagu joonisel 31.



Joonis 31. Vormi visualiseerimine veebis.

3.4 . Vormi ära saatmine (*submit*)

Järgmiseks ülesandeks on vormil vaja kasutaja poolt sisestatud andmed tõlgendada uuesti olemi objekti muutujateks. Selleks on vaja kontrollerisse lisaks importida päringu klass nimega *Request* (joonis 32) ning muuta ka meetodi kirjeldust (joonis 33).

Joonis 32. Päringu klassi import.

```
15    public function uusYlesanneAction(Request $request)
```

Joonis 33. Meetodi kirjelduse muudatus, lisandus sisendparameeter *Request*.

Seejärel on vaja muuta olemi initsialiseerimist, selliselt, et klassi muutujaid ei algväärtustata käsitsi, vaid väärtused tulevad kasutaja poolt. See tähendab, et joonis 27. tuleb eemaldada read 17 ja 18, ehk alles jääb ainult initsialiseerimine. Kui see on tehtud, siis on vaja vormi initsialiseerimise järgi lisada päringu töötlemiseks kasutatava meetodi väljakutse, mis saab sisendiks päringu. (Joonis 34.)

```
27        $vorm->handleRequest($request);
```

Joonis 34. Päringu töötlemine.

Päringu töötlemise tulemusena saame vormi käest küsida, kas sisestatud andmed on valiidsed ning vastavalt sellele edasi tegutseda. Antud näites suunatakse pärast andmete valideerimist kasutaja järgmisesse vaatesse, kus antakse teada, et „Ülesande lisamine õnnestu“. Antud töö raames reaalselt andmeid andmebaasi ei salvestata, küll aga lisame järgmises peatükis ka reaalsed valideerimisreeglid ning testime mõned olukorrad läbi. Vormi valideerimine ning uude vaatesse suunamine on kujutatud joonisel 35 ning toimub pärast päringu töötlemist, st pärast joonisel 34 kujutatud koodi.

```
29        if ($vorm->isValid()) {
30            // Siin tehakse tavaliselt ka andmete salvestamine vms..
31            return $this->redirectToRoute('ylesanne_onnestus');
32        }
```

Joonis 35 Vormi valideerimine ja ümber suunamine.

Hetkel ei juhtu vormi saatmisel midagi erilist, kuna valideerimist ei ole ning samuti on puudu eduteadet kuvav meetod ning vastav mall. Kui proovida vormi saata, siis kuvatakse veateadet, et vastavat rada (’ylesanne_onnestus’) ei leitud. Järgmisena lisame vajaliku meetodi, kuhu valiidse vormi saatmisel suunata ning samuti eduteadet kuvava malli. Uue meetodi võib lisada samasse kontrollerisse, eelmise alla. Joonisel 36 on näide vajalikust meetodist.

```
39      /**
40       * @Route("onnestus", name="ylesanne_onnestus")
41       */
42      public function ylesanneOnnestusAction()
43      {
44          return $this->render('default/eduTeade.html.twig');
45      }
46  }
47
```

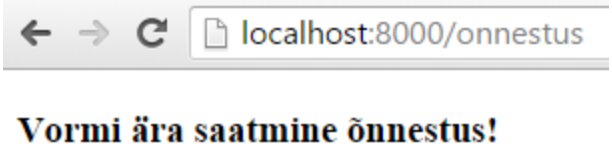
Joonis 36. Meetod eduteate kuvamiseks.

Real 40 on näha veidi teistsuguse kujuga raja süntaksit, nimelt on ära määratud ka „name“ atribuut. Mis tähendab, et koodisiseselt saab joonisel 36 kujutatud meetodile viidata nagu seda on tehtud joonisel 35. real 31. Veebilehitsejast vaadatauna vastab meetod aga järgnevale aadressile - http://localhost:8000/onnestus. Lisaks uuele meetodile on vaja lisada ka uus mall, nimega „eduTeade.html.twig“. Mis võiks välja näha selline nagu joonisel 37.

```
1   {% extends 'base.html.twig' %}
2
3   {% block body %}
4       <h3>Vormi ära saatmine õnnestus!</h3>
5   {% endblock %}
```

Joonis 37. Eduteate kuvamiseks vajalik mall.

Kui nüüd navigeerida veebilehitsejaga aadressile http://localhost:8000/uus/ylesanne , siis vormi ära saatmisel peaksime nägema taolist pilti nagu on kuvatud joonisel 38.



Joonis 38. Eduteade veebilehitsejas.

Hetkel keerulisemate näidete juurde vormi ära saatmisel ei lähe ning eduteade jääb seda peatükki lõpetama.

3.5 . Vormi valideerimine

Selles peatükis räägime HTML5 valideerimisest, mis on igas veebilehitsejas veidi erinev ning millel on vaikimisi olemas veateade, mida vigaste andmete korral vormi saatmisel kuvatakse.

Lisaks räägime kuidas ja milliseid valideerimisreegleid on võimalik olemi klassis defineerida, ning kuidas vaikimis tehtavat html5 valideerimist maha suruda.

Antud näites lisame valideerimisreeglid otse olemi klassi, muutujate dokumentatsioonidesse. Erinevaid valideerimisreegleid on tohutul hulgal, ning lisaks on võimalik kirjutada enda välja mõeldud reegleid, keerukamatel juhtudel kasutatakse näiteks regulaar avaldisi. Antud töö raames toon välja mõned sagedasemad reeglid, mis on raamistikku sisse ehitatud. Et valideerimisreegleid kirjeldada, on esimese asjana vaja Ylesanne.php's lisada rida, mis impordiks sisse valideerimise alla kuuluva piirangu klassi. (Joonis 39.)

```
5    use Symfony\Component\Validator\Constraints as Assert;
```

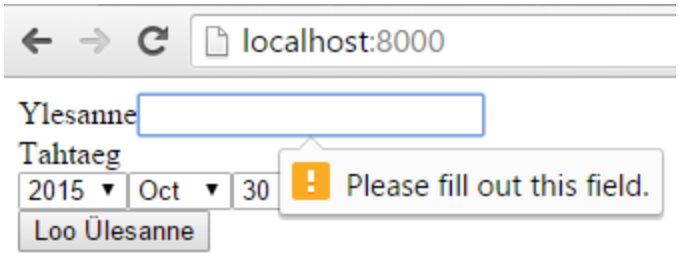
Joonis 39. Piirangu klassi importimine.

Pärast importi on võimalik igale klassi muutujale lisada piiranguid, joonis 40. illustreerib kahte näidet.

```
9        /**
10        * @Assert\NotBlank()
11        */
12       protected $ylesanne;
13
14       /**
15        * @Assert\NotBlank()
16        * @Assert\Type("\DateTime")
17        */
18       protected $tahtaeg;
```

Joonis 40. Klassi muutujatele defineeritud piirangud.

Ridadel 10 ja 15 on defineeritud piirang „mitte-tühi" (*not blank*), mis tähendab, et muutuja väärtus ei tohi olla tühi. Teiseks piiranguks on real 16 defineeritud tüübi piirang, millega öeldakse, et muutujale omistatav väärtus peab olema kuupäeva (*DateTime*) tüüpi. Kui need muudatused on tehtud, võime aadressile „http://localhost:8000/uus/ylesanne" navigeerides veenduda, et piirang töötab, vajutades nupule „Loo Ülesanne" ilma andmeid vormi väljadele sisestamata. Selle tulemusena peaks veebilehitseja kuvama html5 veateate, mida on näha joonisel 41.



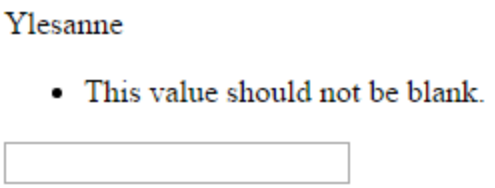
Joonis 41. HTML5 veateade.

Kui „Ylesanne“ välja midagi kirjutada, kaob veateade eest ning suunatakse jällegi eduteate lehele. Kui aga html5 veateatest tahta lahti saada, siis tuleb natuke vormi visualiseeriva malli koodi muuta. Täpsemalt on vaja muuta malli esimest rida, „form_start“ nimelist elementi, nagu seda on tehtud joonisel 42.

```
1    {{ form_start(vorm, {'attr': {'novalidate': 'novalidate'}}) }}
```

Joonis 42. Malli muudatus valideerimiseks.

Uuesti vormi ära saates, ehk nupule „Loo Ülesanne“ vajutades saame nüüd teisel kujul veateate, mida illustreerib joonis 43.



Joonis 43. Veateade HTML5 valideerimiseta.

Sellega valideerimise peatükk piirdub, ühtlasi on see viimane alapeatükk vormide kohta ning loodetavast annab ülevaate, kuidas Symfony2 abil on võimalik vorme väikse vaevaga luua ning vastavalt kasutajapoolsele sisendile töödelda.

Kokkuvõte

Käesoleva seminaritöö eesmärgiks oli anda ülevaade Symfony2 nimelisest PHP raamistikust ning ühtlasi pakkuda samm-sammulist eesti keelset paigaldusjuhendit. Töös tutvustati raamistikku Symfony2 ning anti põgus ülevaade raamistiku käekäigust viimase 10 aasta jooksul. Lisaks käsitleti raamistikus esinevaid olulisi komponente ja kogumikke, sarnasusi ja erinevusi teiste raamistikega ning raamistiku üldist ülesehitust.

Töö põhitulemuseks oli kolmest osast koosnev Symfony2'e tutvutus, mis kategoriseerusid teoreetiliseks ning praktiliseks osaks. Esimene, teoreetiline peatükk, andis ülevaate ja taustteadmised raamistikust ning teine ja kolmas peatükk kategoriseerusid praktiliseks osaks, milledes käsitleti raamistiku paigaldamist, projekti loomist ning mõningasi praktilisi näiteid ja õpetusi.

Seminaritöö andis autorile olulisel määral uusi teadmisi peamiselt kolmes valdkonnas – komponendid, raamistiku paigaldamine ning ka ajalugu. Kuna hetkel töökohas kasutatav Symfony versioon on juba mõned aastad vanad, siis ei olnud ma näiteks teadlik sellest, et selle aasta (2015) novembris tuleb välja uus versioon nimega Symfony3.

Autori hinnangul täitis käesolev seminaritöö oma eesmärgi, kuna töö käigus jõuti vastavalt püstitatud eesmärkidele rahuldava tulemuseni.

Kasutatud kirjandus

Svatok, Y. (2015) *Symfony2 vs Yii2*. Loetud 16.10.2015 aadressil: http://stfalcon.com/en/blog/post/symfony2-vs-yii

Skvorc, B. (2015) *Best PHP framework 2015 Sitepoint survey results*. Loetud 23.10.2015 aadressil: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

Bundles, kuupäev teadmata. Loetud 10.10.2015 aadressil: http://symfony.com/doc/bundles/

Components, kuupäev teadmata. Loetud 24.10.2015 aadressil: http://symfony.com/components

Projects, kuupäev teadmata. Loetud 23.10.2015 aadressil: http://symfony.com/projects

The architecture, kuupäev teadmata. Loetud 20.10.2015 aadressil: http://symfony.com/doc/current/quick_tour/the_architecture.html

Installation, kuupäev teadmata. Loetud 15.10.2015 aadressil: http://symfony.com/doc/current/book/installation.html

Page creation, kuupäev teadmata. Loetud 20.10.2015 aadressil: http://symfony.com/doc/current/book/page_creation.html

Releases, kuupäev teadmata. Loetud 17.10.2015 aadressil: http://symfony.com/doc/current/contributing/community/releases.html

Forms, kuupäev teadmata. Loetud 29.10.2015 aadressil: http://symfony.com/doc/current/book/forms.html

LISAD

Lisa 1. Komponendid

- ***Asset*** – Haldab URL'i genereerimist ning CSS stiililehtede, JavaScript'i failide ning piltide versioneerimist.
- ***BrowserKit*** - Simuleerib veebilehitseja käitumist.
- ***ClassLoader*** - Laeb automaatselt projekti klassid, kui need järgivad mõnda PHP standardi kokkulepet.
- ***Config*** - Aitab leida, laadida, kombineerida ning autoomatselt täita ja valideerida konfiguratsiooni väärtusi.
- ***Console*** - Lihtsustab ilusate ja testitavate käsurealiideste loomist.
- ***CssSelector*** - Konverteerib CSS selektoreid x tee (*XPath*) avaldisteks.
- ***Debug*** - Pakub vajalikke tööriistu PHP koodi silumiseks.
- ***DependecyInjection*** - Võimaldab objektide loomise standardiseerimist ning tsentraliseerimist.
- ***DomCrawler*** – Pakub meetodeid, mis lihtsustavad HTML ning XML dokumentidega manipuleerimist.
- ***EventDispatcher*** - Implementeerib Mediaatori (*Mediator*) mustrit lihtsal ja efektiivsel viisil, mis teeb projektid lihtsasti laiendatavaks.
- ***ExpressionLanguage*** - Pakub mootorit, mis suudab kompileerida ja välja arvutada avaldisi.
- ***FileSystem*** - Pakub põhilisi utiliite failisüsteemile.
- ***Finder*** - Otsib faile ja katalooge intuitiivse liidese abil.
- ***Form*** - Pakub tööriistu HTML vormide lihtsaks loomiseks, töötlemiseks ning taaskasutuseks.
- ***Guard*** - Sisaldab mitmeid autentimise kihte, mis teevad keerukad autentimise süsteemid märksa lihtsamaks.
- ***HttpFoundation*** - Defineerib objekt-orienteeritud kihi HTTP spetsifikatsiooniks.
- ***HttpKernel*** - Pakub alustalasid paindliku ja kiire HTTP'l põhineva raamistiku loomiseks.
- ***Icu*** - Sisaldab ICU teeki(*library*), kindlas versioonis. On alates 2014. oktoobrist ebasoovitatav (deprecated), selle asemel tuleks kasutada *Intl* komponenti.
- ***Intl*** - Komponent, mida kasutatakse juhul, kui *intl* laiendust ei ole installeeritud.
- ***Ldap*** - Pakub *LDAP*'i klienti lisaks PHP *ldap*'i laiendusele.

- ***Locale*** - Samuti ebasoovitatav alates Symfony2.3 versioonist, tuleks kasutada *Intl*'i komponenti.
- ***OptionsResolver*** - Aitab lisavalikute massiivi abil objekte konfigureerida.
- ***Process*** - Käskude käivitamine alam-protsessides.
- ***PropertyAccess*** - Funktsionaalsus objektist ning massiivist andmete lugemiseks ja kirjutamiseks kasutades sõne(*string*) notatsiooni.
- ***PropertyInfo*** - Ekstraheerib informatsiooni PHP klasside atribuutide kohta, kasutades populaarsete allikate (*Doctrine, PHP Reflection, PHPDoc.*.) metaandmeid.
- ***Routing*** - Pakub keerulistele autentimissüsteemidele infrastruktuuri.
- ***Serializer*** - Muudab objektid kindlasse formaati (*XML, JSON, YAML,*..) ning vajadusel tagasi esialgsele kujule.
- ***Stopwatch*** - Pakub võimaluse koodi profileerimiseks(*profile).*
- ***Templating*** - Pakub erinevaid tööriistu mallisüsteemide loomiseks.
- ***Translation*** - Rakenduse rahvusvahelisustamiseks vajalikud tööriistad.
- ***Validator*** - Klasside valideerimiseks vajalikud tööriistad.
- ***VarDumper*** - Mehhanismid PHP juhuslike muutujate läbi käimiseks.
- ***Yaml*** - YAML failide laadimine.

(*Components*, kuupäev teadmata)