Introduction to Symphony

Files	https://drive.google.com/drive/folders/1zHxsJq3vLxlmbsEn24-MvYOVvqoRelto
Tags	PHP

- One of the biggest PHP framework, most generic one ⇒ you can built anything in it like games, ecommerce site, api , crm system and so on

Point of using a framework

- to make repititive tasks like creating directions, validating forms more automatic
- it makes it easier to look for more developers (job ads) → there is less customised code so you can get up to speed very quickly
- symfony his big competitor is laravel

Getting started

INSTALL SEE LATER

▼ in terminal:

```
symfony new tutorial –full
```

▼ don't forget the full !!! otherwise its only good for an api and now its a full fledged web application

▼ You need composer

▼ composer is the apt-get of php, it's wat npm is voor nodejs. Its a packagemanager

▼ it installs all dependencies for you that you need when you want to intsall something else

▼ install plugin symfony support!!

▼ you will get symfony tab and need to enable it for this project

In PHP storm: directory structure

▼ vendor/var ⇒ these are in the gitignore files

- ▼ in the var ⇒ you have cache file (some calculation symfony remembers) and also logs
- ▼ vendor all packages installed by composer ⇒ composer manages these files
  - ▼ DONT CHANGE ANYTHING IN THOSE VENDOR DIRECTORIES cause when people do composer update ⇒you're changes will be lost

▼ bin

- ▼ shelsscripts in here
- ▼ console ⇒ nice terminal commands
- ▼ in terminal when you type bin/console you see all the possible terminal commands

```
bin/console
```

▼ config

- ▼ has all the config files
- ▼ yaml file ⇒ yet another markup language
  - ▼ it comes from Python originally
  - ▼ its tab and space sensitive! spaces have meaning just like in python so watch out
  - ▼ in these document you have glorified arrays which are divided by use of tree structure and tabs
- ▼ NORMALLY YOU DON4T have to change anything here to make it work

▼ migrations

- ▼ some form of sql scripts are here for your database ⇒ actually php files that stores the changes
- ▼ takes snapshot of currentdatabase and looks to the differences with the models and will write those changes into a php file
- ▼ by using git pull en run migrations he will have the same database as you

▼ public

- ▼ entry point in the browser
- ▼ if you go into the browser it will go to this index.php file

- ▼ FROM NOW ON WE DON'T have a root index.php file ANYMORE
- ▼ so people can't acces the other files in the directory !! WAY SAFER
  - ▼ this way hackers can not see for example all the vendor packages and can not see the vulnaribilities !
- ▼ NEVER CHANGE THE INDEX.PHP
- ▼ DON4T CHANGE ANYTHING HERE

▼ source - YOU SPEND 90% here

- ▼ controllers
  - ▼ where you place your controllers
- ▼ entity - here you place your models
- ▼ repository
  - ▼ those are the loaders the multiple ones
  - ▼ models for the single ones
  - ▼
- ▼ templates - view system
  - ▼ base.html.twig → extra layer to make more rapidly front end stuff
    - ▼ twig are prepared statements that filters and sanitizes a lot of you code
      - ▼ app will be more secure by usage of twig
- ▼ KENNEL ⇒ dont change this file!!

▼ test - for testing development

- ▼ for unit tests
- ▼ its green because its a test directory

▼ translation

- ▼ for translation concerning languages

Lets make some code

▼ controller file needed and view for saying hello world

▼ in terminal:

- ▼ here you can see the make bundle → things that symfony will write for you

```
make:controller
```

▼ you just have to chose a name that starts with a Capital and endswith Controller

▼ by this command it will create a controller and a view file

▼ in controller file

▼ namespace ⇒ new concept in OOP

▼ its kind away of categorising and has to do with categorising classes

▼ this way there is no conflict with classes with the same name

▼ use is for using classes in another namespace

▼ extends AbstractController

▼ EACH CONTROLLER NEEDS TO EXTEND FROM IT and will receive some handy features !!

▼ annotation

▼ is a special kind of comment that change the behaviour of the application

▼ adds meta-information

```
/**
* @route ("/home", name="homepage")
*/
```

name needs to be unique over ENTIRE APP

will render function below this when this route is loaded

▼ the render part will let you go to the view and gives an array of variables to this view

▼ how to use post and get

▼ make form in view put method on post

▼ use the request object ⇒ anything related to http request (organising file uploads, doing redirects and so far)

▼ you need to import it by using alt enter and then import class http foundation one

▼ use request

```php
/**
 * @Route("/products/{productname}", name="product")
 */
public function index(Request $request, string $productname)
{
    //post
    $firstname = $request->request->get('firstname');
    //get
    $lastname = $request->query->get( key: 'lastname');

    return $this->render( view: 'product/index.html.twig', [
        'controller_name' => $firstname,
        'productname' => $productname
    ]);
}
```

▼ use fancy links

```php
 * @Route("/products", name="productdefault")
 */
public function default()
{
    return $this->render( view: 'product/index.html.twig', [
        'controller_name' => 'ProductController',
        'productname' => 'No product'
    ]);
}

/**
 * @Route("/products/{productname}", name="product")
 */
public function index(string $productname)
{
    return $this->render( view: 'product/index.html.twig', [
        'controller_name' => 'ProductController',
        'productname' => $productname
    ]);
}
```

- in the view
  - builds site up with blocks
  - and kind of usage object oriented methodology for html
  - in the base file ⇒ you define what each base should have
    - the things in the block can be overwritten by your other pages
    - you can have several base files and then let the other files extend from it like in OOP
  - when you want to use a variable inside your view which you passed by the controller ⇒ you need {{name you gave in associative array in controller}}
    - twig automatically sanitised user input! so you don't have to worry about that anymore
  - use conditions and loops

```
{% if age >18 %}
<p> you're an adult </p>
{%else%}
<p> you're a child </p>
{% endif %}

{% for animal in animals %}
{{animal}}
{% endfor %}

{% for key, animal in animals %}
{{animal}} / {{key}}
{% else %}
<p>No animals found</p>
{% endfor %}
```

ELSE in the loop ⇒ executes when there is no animals

- in links use name of the route annotation

```
{{url('homepage')}}
```

- twig acces objects use . instead of →
  - if you just do .name ⇒ it will not use the name but the getter of the name because of symfony
- show flash message
  - flash message it is only displayed once

```
{% for flashMessage in app.session.flashbag.get('success') %}
    <p style="color: green;">{{ flashMessage }}</p>
{% endfor %}
```

Configure database

- copy file .env and name it .env.local ⇒ symphony will automattical gitignore this and it will be valued more and overwrite .env file
- create a database in the terminal or in dbBeaver
- in the env.local
  - APP_ENV=dev ⇒ developer mode it will not put anything in cache (in production mode it will rememeber it) ⇒ DOWNSIDE if change the code you need to rebuild the cache
    - in developer mode you also have a debugger bar in the browser
  - the url to database
- make model and repository

```
bin/console make:entity
```

- doctrine
  - ORM layer ⇒OBJECT RELATION MAPPER
    - it is mapping your classes with a database

▼ doctrine comes with annotations and by these it knows how to map it in my database

▼ create tables in db beaver

```
bin/console make:migration
```

▼ migrate migrations

▼ DRY RUN FIRST BEFORE DOING MIGRATE

▼ if you forgot to dry run and you have conflicts you need comment out what already is executed and after executions but it back out of the comments

```
bin/console doctrine:migrations:migrate --dry-run
```

```
bin/console doctrine:migrations:migrate
```

▼ if you're using mariaDb you need to .env.local file your specific version in the url database

▼ in db beaver you can see tables and also all the migrations that has been executed

Working with doctrine CRUD (for java it is spring)

▼ CREATE

▼ (watch out with the order of things when it match a pattern of something before in the route you will not see it)

▼ persist ⇒ doctrine aware of object that exists ⇒ like a git add

▼ flush ⇒ like git push and commit ⇒ send to database

- ▼ make sure there is a table (see configure database) ⇒ migration

▼ READ

- ▼ one line

```php
/**
 * @Route("/products/new/{categoryName}", name="product_new")
 */
public function create(string $categoryName)
{
    //$category = $this->getDoctrine()->getRepository(Category::class)->find(1);
    $category = $this->getDoctrine()->getRepository( persistentObject: Category::class)->findOneBy([
        'name' => $categoryName
    ]);
```

- ▼ load all info
  - ▼ see screenshot

▼ UPDATE

- ▼ typehint get variable as a product that has value of id ⇒ then sympfony is smart enough
- ▼ use the setters on the object and use flush
  - ▼ you don't need persist cause its already there in the database so doctrine is already tracking it
- ▼ SAVER WITH POST:

```php
/**
 * @Route("/products/update/{product}", name="product_update")
 */
public function update(Request $request, Product $product)
{
    $product->setPrice($request->request->get('price'));

    $this->getDoctrine()->getManager()->flush();//git push

    die($product->getPrice());
}
```

▼ DELETE

```php
/**
 * @Route("/products/delete/{product}", name="product_update")
 */
public function delete(Product $product)
{
    $this->getDoctrine()->getManager()->remove($product);
    $this->getDoctrine()->getManager()->flush();//git push

    $this->addFlash( type: 'Product is deleted');
    return $this->redirectToRoute( route: 'category');
}
```

- Use relations
  - when you make new entity and use doctrine you put type on relation
  - Joints are then managed by using the models

Styling

link in base file (in the style block) in the view or somewhere else

and in public you can add the css directory with the file and also js and images if needed

Make commands user login in 15 minutes

```
bin console | grep make
```

- Make a user with a password

```
bin/console make:user
```

  - in the entity that needs to implement UserInterface
    - id
    - password
    - roles
      - editorrole
      - new but no adjustments
      - adminrole
      - normal users and so one

```
bin/console make:migration
```

https://www.notion.so/Introduction-to-Symphony-47fba6b57e6f46c391271ca3a9c20636#9cbc8b383c334aab88313d50845d8120

When you want to adjust a field you need to do the make:user again with the same name

▼ make auth

```
make:auth
```

- in the authAthenticator you need to give which file can only been seen when you're logged in otherwise it will throw an error

▼ make: registration-form

```
make: registration-form
```

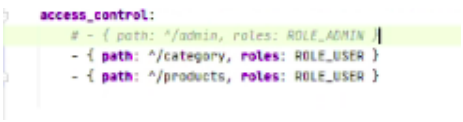
yes for unique validation

▼ in controller

▼ symfony will always use stronger hashing that is available on that server

▼ config file security.yaml to make sure for which pages you need to be logged in

access_control



Reset password make

▼ make: reset-password

```
make: reset-password
```

▼ install bundles that our needed (and other teammembers just need to use composer update) and try again

▼ migrations again

▼ look at the five steps in terminal

▼ mail catcher ⇒ you can install this to mimic what will happen if you would send a mail

Make:crud

▼ first make entity then make CRUD

```
make:crud
```

you see that / is there in the route in the controller but thats no problem because of the route above in the file