SHAKE THE FUTURE

CENTRALE NANTES

Web Programming

PHP - Symfony

JY Martin - JM Normand

Plan

Main objective

In this practical work, we will work on basic tools in PHP and SYMFONY

- First we will work on PHP basics
- Them we will build a small application using SYMFONY

Why a PHP Framework?

PHP is a very used language for web development.

- Easy to learn
- Easy to deploy
- Many functionalities
- Can be use on most environments with a single development

But, as it is easy to misuse it, it's a good idea to use a frameworK.
Symfony is a good one.

Tools

To write this application you will need a text editor or an IDE. Netbeans or NotePad++, BBEdit, Sublime Text, ... will be convenient.

- To display files, you will need a web browser with debugging tools.
- You will need an HTTP server for the first part.
- A Database Server. We will use PostgreSQL.
- And some data.

Have a look to prerequisites to install tools and data.

Plan

PHP Basics

PHP is a programming language. An interpreted langage.
When you launch a file, it is compiled on the fly.

There are 2 ways of using it:

- With an HTTP server. Scripts will be called by HTTP server to produce HTML files.
- Standalone. Scripts are launch with the terminal.

A bit of syntax

A php script is included in a tag:

```
<?php
... Here is the PHP script
?>
```

A bit of syntax

PHP is a programming language. Syntaxically, it is closed to PERL, C, ...

- instructions end with ;
- use { ... } to define blocks.
- variables start with $ followed by the variable name.
- variables types depends on the context, and their previous used. Since PHP 7 it is possible to force variable types definition.
- Your are not obliged to declare a variable. Using it is considered as a declaration.
- to set a variable value, use =
- comments on 1 line can use //
- comments on more that 1 line are set with /* ... */

A bit of syntax

Here is an example.

```
<?php
$myVariable = 5; // Set myVariable to 5
$myVariable = $myVariable + 1; // Add 1
$myVariable2 = "aString";
?>
```

A bit of syntax: types

You can manipulate standard type of data

- integers (1, 123, -42, ...)
- reals (12.34, -10.11, ...)
- boolean (true, false)
- characters, strings
  - '...'
  - "..."
- arrays
- objects

A bit of syntax: strings

Strings can be set with '...' or "...", but they do not have the same effects.

If the "..." string contains a variable ($...), the variable is replaced by its value.
'...' keeps variables in the string. $... will not be replaced.

You can concatenate strings using .

```
<?php
$i = 1;
$s1 = "i = $i"; // $s1 contains "i = 1"
$s2 = 'i = $i'; // $s2 contains "i = $i"
$s2 = $s1 . $s2; // $s3 contains "i = 1i = $i"
?>
```

A bit of syntax: operations

PHP implements

- standart arithmetic operations
  - +, −, *, /
  - %, * *
- usual functions
  - sin, cos, ..., asin, acos, ..., log, exp, ...
  - ceil, floor, round, ...
  - min, max, ..
  - rand, srand, ...
- standard boolean operations
  - &&, ||, !

And more.

A bit of syntax: arrays

- Arrays are created by their use or by an explicit declaration
- Multi dimensionnal arrays are build as arrays of arrays. That means index i and j are not obliged to have the same dimension.
- arrays are indexed by integers, reals, strings...

```
<?php
$myArray = array();
$myArray[0] = 1;
$myArray[1][2] = 3 + $myArray[0];
$myArray[2]["new"] = "aString";
?>
```

A bit of syntax: Control instructions

Control instructions cover:

- tests: if (...) { ... } else { ... }
- switch: switch ($expression) { case ...: .... break; default: ... }
- for loops: for ($var = ...; $var <... ; $var ++) { ... }
- while loops: while (...) { ... }

it also includes “foreach” to browse an array

A bit of syntax: Output

To print to the output flow you can use

- echo ...
- print ... There are only minor differences between these instructions, so you can use both.
- print_r(...) displays an array

```
<?php
$i = 1;
print "Result is ".$i;
?>
```

A bit of syntax: Functions

functions look a bit more like JS ones.

```
function myFunction(...parameters...) {
      ...
      return result; // eventually
}
```

- Using & before a parameter make it passed by address.
- You can set default values to parameters

And the way to call them looks like every function call

```
$variable = myFunction(... args ...)
```

A bit of syntax: objects

Since PHP 5, PHP can manage objects.
Syntax is a bit more clear since version 7.

```
class myClass {
    ... Attributes
    ... Methods
}
```

A bit of syntax: instances

Use new to create a new instance of a class.
Use ->to call a method

```
class myClass {
      ...
}
$myVar = new myClass();
$myVar->doSomething();
```

A bit of syntax: objects attributes

By default, attributes are public. You can define them as public, protected or private. That limits which class can directly access attributes.

- public: any class can access the attribute
- protected: acces for the class and any class that heritate the current one.
- private: acces limited to the class.

For protected and private access, you should add getters and setters for the attributes.

```
class myClass {
    ... private attribute;
    ...
}
```

A bit of syntax: objects methods

As for any class in any language, you can define methods. These methods will be applied to instance of their class.

```
class myClass {
    public function myFunction(...) {
        ...
    }
    ...
}
```

$this is current object $self for current class. Can be used for static methods or attributes.

A bit of syntax: constructors and destructors

Constructors are called when an instance is created.
Destructors are called when object are removed.

```
function __construct() {
    ...
}
function __destruct() {
    ...
}
```

A bit of syntax: Including files

Usually, program are not in a single file. You will split your program in several files, and include the ones you need.

In PHP, there are 2 ways of importing files

- require
- include

If file is missing, require will stop your program. include will not. You can also add "_once" to the instruction to be sure the action is done only once.

So, there are 4 instructions:
Require, Require_Once, Include and Include_Once.

```
Require_Once("MyFile.php");
```

A bit of syntax: libraries

There are many functions and many available libraries.

However you have to take care of some points:

- they are not deprecated
- your php version is compiled with the right library
- the library is activated in the .ini file

A bit of syntax: HTTP exchanges

PHP exchanges with browser through variables. More specifically arrays.

- $_GET contains data according to a GET call
- $_POST contains data according to a POST call
- $_SERVER contains context elements

Reply to the browser is made through the output flow (print)

A bit of syntax: HTTP exchanges

Example:

Suppose your browser launch this call:

```
http://myserver.com/action.php?myAction=1&name=DUPOND
```

This is a GET call. So your file action.php should retrieve data like this:

```
<?php
$myAction = $_GET["myAction"]; // will be set to 1
$personName = $_GET["name"]; // will be set to DUPOND
?>
```

A bit of syntax: HTTP exchanges

By default, with an HTTP exchange, reply will be an HTML stream. You can change this if necessary. Before any output, set the output format:

```
Header("Content-Type: any-mime-type");
Header("Content-disposition: attachment; filename=the-file-name");
print $Data;
```

Examples of mime types:

- text/html: a HTML file
- application/octet-stream: a text/binary stream
- application/msword: a word file
- ...

https://svn.apache.org/repos/asf/httpd/httpd/trunk/docs/conf/mime.typ

A bit of syntax: Some usefull functions

- isset($varieble) checks if a value has already been affected to this variable
- is_array(...), is_int(...), ... check if variable are arrays, int, ...

Our first PHP page

Let’s start with the usual hello world.

Here is the hello.php file:

```
<?php
print "Hello World !!";
?>
```

Put your file in your HTTP server doc location (htdocs, ...)

Call it with your browser: http://127.0.0.1/hello.php

Going a bit further...

In the materials for this practical work, you will find a HTML file.

This file is supposed calling action.php using GET method.
You will have to write the file action.php.

- You should get data from the form through the $_GET array.
- Check $_GET contains the "user" data
- Check "user" data is "admin"
  - if true, reply with an "ok"
  - if false reply with a "wrong".

Going a bit further...

Let's switch to POST.

Change form method in index.html to POST.

Change your action.php file to use $_POST instead of $_GET

Check it still works.

Going a bit further...

We will use a function.
Copy CheckLDAP.php from the materials to your HTTP server doc location.

In your file action.php, import CheckLDAP.php
Maybe Require_One should be good for that.

Using the function in the file CheckLDAP.php, check your login / password instead of “admin”

Plan

About Symfony

Symfony is a PHP framework, created in 2005 by SensioLabs.
It is one of the most used in France.

Symfony is free, but you can also pay to gain support.

Symfony current version is 4.3.2 (should have changed...)
Versions can be easilly upgraded since version 3.

Symfony and ORM

Symfony can use 2 ORMs, Doctrine and Propel.

By default, the version we use is designed for Doctrine.

Doctrine will manage the database through objects.
So we will have to define objects that map the tables in the database.

What will we do?

The objective is to build a basic application using symfony. We will connect to the application, get informations, add, remove modify informations.

These informations are stored in a database, so we will have to build a link between the application and the database.

What will we do?

First, we will install symfony and create a project.

Then we will build the link to the database, so that we can access data.

Next, we will implement functionalities -list, add, delete, modify-

And at last we will add a basic connection screen.

Installing Symfony

Check https://symfony.com/download
Follow instructions, especialy for adding path.

Have a look to prerequisites.

Installing Composer

Composer is a Symfony module that will help you to manage other modules / libraries

Have a look to https://getcomposer.org/download/

Run commands in your terminal / Invite de commandes to download it

That should create the file composer.phar locally

Creating a project

In your terminal / Invite de commande, ask composer to create a project

```
composer create-project symfony/website-skeleton prwebSYNFONY
```

Note:

- **prwebSYNFONY** is the name of your project
- That will download/install files. So it may take some time to create the project.

Still in your terminal / Invite de commande, go to your project.

```
cd prwebSYNFONY
```

Adding tools

We will add dev module. Maybe they can help you.
DO NOT COPY THESE LINES type then in your terminal. the

- 1rst time: That means you **NEVER** installed dev module on your computer.

```
composer require symfony/web-server-bundle --dev
```

- If already installed

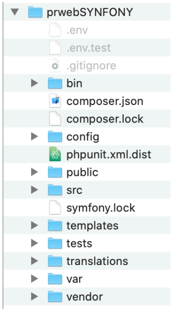
```
composer require server --dev
```

That should:

- Add elements in the file composer.json

Your architecture...

Here is what your folder should look like.



```
▼ prwebSYNFONY
    .env
    .env.test
    .gitignore
  ▶ bin
    composer.json
    composer.lock
  ▶ config
    phpunit.xml.dist
  ▶ public
  ▶ src
    symfony.lock
  ▶ templates
  ▶ tests
  ▶ translations
  ▶ var
  ▶ vendor
```

Look, there are hidden files, the grey ones.

Your architecture...

What are the files/folders used for?

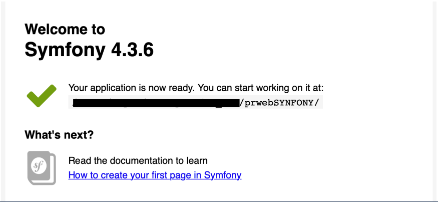
- .env: is a hidden global file for configuration
- bin: for symfony commands
- config: configuration files
- public: where we will put css, images, js files
- src: your files. The project source files. What you develop.
- templates: “HTML” templates for the application. It is not really HTML, call it TWIG.
- tests: for tests purpose
- translations: for multi-languages applications
- var: symfony temporary files, like caches
- vendor: symfony libraries

Let’s try...

Our project is ready to run. Use this command in your terminal / Invite de commande to run the server

```
php bin/console server:run
```

Now, open your web browser with the following URL:
http://127.0.0.1:8000
Check it works. Maybe you version is a bit newer.


Welcome to
Symfony 4.3.6

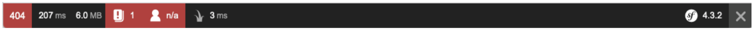
Your application is now ready. You can start working on it at:
[redacted]/prwebSYNFONY/

What's next?

Read the documentation to learn
How to create your first page in Symfony

In your browser...

Have a look bottom of your page.



Click on one of the icons to display tools.
You can have a overview of what happens in your application.

Nb: You can close the tool bar with the X icon located on right side.

server commands

- run the server: for test purpose. Use CTRL-C to stop the process

```
php bin/console server:run
```

- start server: launch server. Stop the server with server:stop

```
php bin/console server:start
```

- stop server: stop a launched server

```
php bin/console server:stop
```

- server status: check server is launched

```
php bin/console server:status
```

The application

Nice, we have the project infrastructure.

Now, we have to implement the link to the database.

Link with the database

We use a database. To build a link between our application and the database, we have to define some elements.

- Configuration files: to give informations about the server and the database
  - doctrine.yaml
  - .env
- Entities, php classes that map the tables in the database. They are located in src/Entity
- Repositories, php classes that manage sets of entities. They are located in src/Repository

Link with the database

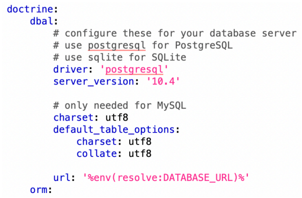
First step, we have to define with which kind of database we deal with.

By default, Doctrine is configured for a mysql database. As we use a postgresql one, we have to tell it to Doctrine.

Link with the database

Open file **config/packages/doctrine.yaml** and change dbal data:

- driver from ‘mysql’ to ‘postgresql’
- version to ... your database server version. You can use PgAdmin and connect to your database. PgAdmin would tell you the server version.
- charset and default_table_options /charset to utf8



```
doctrine:
    dbal:
        # configure these for your database server
        # use postgresql for PostgreSQL
        # use sqlite for SQLite
        driver: 'postgresql'
        server_version: '10.4'

        # only needed for MySQL
        charset: utf8
        default_table_options:
            charset: utf8
            collate: utf8

        url: '%env(resolve:DATABASE_URL)%'
    orm:
```

Link with the database

Next step, we have to define the database informations.
Have a look to the file **.env**, located root of your project. As it is an invisible file, use a text editor and open the file with the menu.

Have a look to the Doctrine section, around line 28. We have to change the DATABASE_URL item.

```
DATABASE_URL=pgsql://prweb:prweb@127.0.0.1:5432/prweb
```

- pgsql is the protocol for postgresql
- prweb (the first one) is the login to connect with
- prweb (the second one) is the password for prweb
- 127.0.0.1 is localhost and 5432 is the postgresql port
- prweb (the third one) is the database name

Link with the database

Next: we have to create files to manage informations and link them to the database. There are several ways, methods, to build this link:

- Create files and map them to the database
  This is the recommanded way of doing it with Doctrine.
  You can do it
  - 1: Manually: you create the files and their content
  - 2: Using configuration files -XML or YAML files-
  - 3: Using Symfony: interactive mode to create files
- Use an existing database and ask Doctrine to create the corresponding php files in your program.
  That means doing reverse engineering.

We will use the last one. So, do not be in an hurry writing files. Next slides explain main principles.

Link with the database - Entities file structure

Let’s talk about files contents.
Here is the structure of a php file to define an entity:

```
<?php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
* EntityName
* @ORM\Entity(repositoryClass="App\Repository\RepositoryName")
* @ORM\[] Table(name="tableName")
*/
class EntityName {
...
}
```

Link with the database - Entities file structure

In previous slide:

- We renamed **Doctrine\ORM\Mapping** as ORM, so that it is easier to manipulate it.
- EntityName = the class name, the entity name
- tableName = the table name in database. Should be written with lowercase characters
- RepositoryName = the repository class name that will manage EntityName entities

Link with the database - Creating an attribute

In a table, there are columns. They are defined as attributes in the class.
We have to take into account:

- Standart declaration for most of the attributes
- IDs
- External links to other entities / classes

https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/basic-mapping.html

Link with the database - Creating an attribute

Here is an example of column declaration:

```
/**
* @var string|null
* @ORM\Column(name="title", type="string", length=255, nullable=true)
*/
private $title;
```

Link with the database - Creating an attribute

- “@var string|null”: the column is a string, and it may contain null values.
- Second line if for the database. Column name is “title”, it is a string, its length is max 255 characters and it may contain null.
- last line is the attribute definition, as private. It’s name is title.

You will find possible data types in https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/basic-mapping.html

Link with the database - Creating an id

Now, an ID:

```
/**
* @var int
* @ORM\Column(name="id", type="integer", nullable=false)
* @ORM\Id
* @ORM\GeneratedValue
*/
private $id;
```

- "@ORM\Id", tells the column is the ID.
- "GeneratedValue", defines auto-increment.

GeneratedValue uses the default strategy to use auto-incrementation. You can change this strategy, but be sure of what you do.

Link with the database - Creating an link to another entity

And a last one: building a link with another entity.

```
/**
* @var \OtherEntity
* @ORM\OneToOne(targetEntity="OtherEntity", cascade={"persist"})
*/
private $otherEntity;
```

- OtherEntity is another entity class
- "OneToOne", tells this attribute is linked to an unique OtherEntity entity. And vice-versa.
  You can use other kind of declarations.
  https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/association-mapping.html

Link with the database - Creating an link

Here are the links you can create:

- OneToOne: one entity linked to another one, and this entity linked to the first one. The 2 linked entities use a OneToOne link.
- ManyToOne: one entity linked to another one, and this entity linked to many entities. The entity uses ManyToOne, and the other one uses OneToMany.
- OneToMany: one entity linked to many other ones, and these other ones linked to the first one. The entity uses OneToMany, and the other one uses ManyToOne.
- ManyToMany: one entity linked to many other ones, and these other ones linked to many entities too. The 2 linked entities use a ManyToMany link.

Link with the database - Creating an link: common syntax

Here are the elements you will find in the declarations

- targetEntity="..." defines the class that manages the entity we are linked to
- cascade={"persist"} ensure that if we apply on operation on the targetEntity entity, also it is mapped to current attribute. Take care, cascade do not solve every link problem.
- mappedBy="..." defines the name, in targetEntity, of the attribute we are linked to
- inversedBy="..." is the reverse link of mappedBy
- @JoinColumns and @JoinColumn define columns names in the database

Link with the database - UniDirectional and Bidirectional links

Most often, you do not have to define the link in both linked entities.

- If you use Unidirectional declaration, you have to declare the link in a file, the other link is implicit.
  In that case, you should not use mappedBy nor inversedBy.
- If you use Bidirectional declaration, you declare the link in both files.
  In that case, you have to use mappedBy and inversedBy.

Most often Unidirectional link can be used.
However, if you use reverse link, a bidirectional implementation may help you.

Link with the database - Creating an link

You can also define some informations.

JoinColumns will define informations about implementation.
It should be located at the end of the description, just before the attribute name definition.

```
* @JoinColumn(name="firstEntityId", referencedColumnName="id")
```

means current attribute will be implemented as a column named "firstEntityId". It is linked to a column "id" in the other entity.

Link with the database - OneToOne unidirectional link

- FirstEntity: This should be the one that access the link most often.

```
* @ORM\OneToOne(targetEntity="SecondEntity")
...
private $secondEntity;
```

- SecondEntity:
  Nothing to declare

Attribute $secondEntity in FirstEntity is linked to the ID of the class SecondEntity

Link with the database - OneToOne bidirectional link

- FirstEntity: This should be the one that access the link most often.

```
* @ORM\OneToOne(targetEntity="SecondEntity", mappedBy="firstEntity")
...
private $secondEntity;
```

- SecondEntity:

```
* @ORM\OneToOne(targetEntity="FirstEntity", inversedBy="secondEntity")
...
private $firstEntity;
```

Link with the database - ManyToOne unidirectional link

- FirstEntity: That should be the entity linked to an unique other entity.

```
* @ORM\ManyToOne(targetEntity="SecondEntity")
...
private $secondEntity;
```

- SecondEntity:
  Nothing to declare

$secondEntity in FirstEntity is linked to the ID of SecondEntity

Link with the database - ManyToOne bidirectional link

- FirstEntity: The most important entity uses ManyToOne. That should be the one linked to an unique other object.

```
* @ORM\ManyToOne(targetEntity="SecondEntity", inversedBy="firstEntities")
...
private $secondEntity;
```

- SecondEntity:

```
use Doctrine\Common\Collections\ArrayCollection;
...
* @ORM\OneToMany(targetEntity="FirstEntity", mappedBy="secondEntity")
...
private $firstEntities;
...
public function __construct() {
        $this->firstEntities = new ArrayCollection();
}
```

Link with the database - ManyToMany unidirectional link

- FirstEntity:

```
use Doctrine\Common\Collections\ArrayCollection;
...
* @ORM\ManyToMany(targetEntity="SecondEntity")
* @JoinTable(name="joinTableName",
* joinColumns={@JoinColumn(name="firstEntityId", referencedColumnName="id")},
* inverseJoinColumns={@JoinColumn(name="secondEntityId", referencedColumnName="id")}
* )
private $secondEntities;
...
public function __construct() {
        $this->secondEntities = new ArrayCollection();
}
```

- SecondEntity: As the other ones, you do not have to define anything, because it's already defined by FirstEntity.

Link with the database - ManyToMany unidirectional link

- @JoinTable tells
    - the link between the 2 tables is implemented through a table called joinTableName
    - joinTableName has 2 columns: firstEntityId and secondEntityId
    - there is an external link between firstEntityId and the primary key in FirstEntity: id
    - there is an external link between secondEntityId and the primary key in SecondEntity: id
- SecondEntitys entities will be managed in FirstEntity by $secondEntities

Link with the database - ManyToMany bidirectional link

- FirstEntity:

```
use Doctrine\Common\Collections\ArrayCollection;
...
* @ORM\ManyToMany(targetEntity="SecondEntity", inversedBy="firstEntities")
* @JoinTable(name="joinTableName")
...
private $secondEntities;
...
public function __construct() {
        $this->secondEntities = new ArrayCollection();
}
```

Link with the database - ManyToMany bidirectional link

- SecondEntity:

```
use Doctrine\Common\Collections\ArrayCollection;
...
* @ORM\ManyToMany(targetEntity="FirstEntity", mappeddBy="secondEntities")
...
private $firstEntities;
...
public function __construct() {
        $this->firstEntities = new ArrayCollection();
}
```

Link with the database - practical method

There are 4 methods to implement the php files for the entities:

- Method 1: Manual - From PHP to Database -
  You write php files, then you map them to the database
- Method 2: Semi-Automatic - From PHP to Database -
  You ask help to symfony to build php files. Then you map them to the database
- Method 3: Semi-Automatic - From PHP to Database -
  You use YAML or XML files to decribe data, Symfony build them. Then you map the files to the database
- Method 4: Automatic - From Database to PHP -
  Using reverse engineering, you build PHP files from database.

We will use the last one, but you should have a look to the first ones.

Link with the database - Method 1

Method 1: creating files and mapping them in the database

Nb: this is not the method we will use.
Create 2 files in src/Entity: Item.php and Category.php

Link with the database - Method 1

Use the infrastructure we describedto build entities Item and Category.

- EntityName = the object class name: Item or Category
- tableName = the table name in database: item or category
- RepositoryName = the repository class name: ItemRepository or CategoryRepository

We will define the "RepositoryName" files later.

Link with the database - Method 1

Add attributes for Item.

- id - an int, auto-incremented
- title - a string
- author - a string
- body - a string
- category_id a Category. You link it to the id of Category.
  the link type is ManyToOne. You can use a Unidirectional link.

Link with the database - Method 1

Add attributes for Category.

- id - an int, auto-incremented
- name - a String.

Link with the database - Method 2

Method 2: Create files using YAML or XML

Nb: this is not the method we will use.
we create a YAML file for each entity.

Link with the database - Method 2

Definitions are located in src/Resources/config/doctrine

Definition for FirstEntity is stored in src/Resources/config/doctrine/FirstEntity.orm.yml

Link with the database - Method 2 - YAML definition

YAML definition describe fields to implement.
Here is the example for Category

```
App\Entity\Category:
    type: entity
    repositoryClass: App\Repository\CategoryRepository
    table: category
    id:
        id:
            type: integer
            generator:
                strategy: AUTO
    fields:
        name:
            type: string
            length: 255
```

Link with the database - Method 3

Method 2: ask symfony to create files and to map them to the database

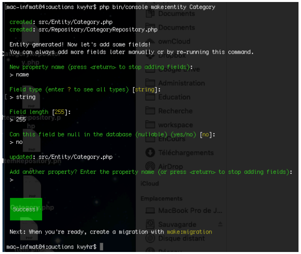
Nb: this is not the method we will use.
Use interactive command

Link with the database - Method 3

Use this command:

```
php bin/console make:entity Category
```

Then answer the question,
building attribute name.
name is a 255 characters
string.
id is built by default.



```
mac-infmat04:auctions kwyhr$ php bin/console make:entity Category

 created: src/Entity/Category.php
 created: src/Repository/CategoryRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

 New property name (press <return> to stop adding fields):
 > name

 Field type (enter ? to see all types) [string]:
 > string

 Field length [255]:
 > 255

 Can this field be null in the database (nullable) (yes/no) [no]:
 > no

 updated: src/Entity/Category.php

 Add another property? Enter the property name (or press <return> to stop adding fields):
 >


  Success!


 Next: When you're ready, create a migration with make:migration

mac-infmat04:auctions kwyhr$
```

Link with the database - Method 3

Use the same command to build Item attributes.

- title - a string
- author - a string
- body - a string
- category_id a Category. field type is a relation.

Link with the database - Method 3

This method generates Entities and Repositories.

However, have a look to the files. The ID generation should be a little more complex than it should be.
Change ID definitions according to what it should be.

Link with the database - Method 4

Method 4: import tables and ask Doctrine to create the required files.

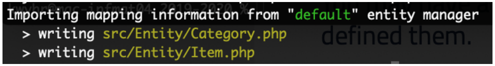
Some kind of reverse engineering.
This is not the recommended method, but it can be usefull when you still have a created database.
We use a command to generate files
This is the method we will use

Link with the database - Method 4

Run this command:

```
php bin/console doctrine:mapping:import "App\Entity" annotation
–path=src/Entity
```

This will create one file per table in src/Entity
You may have a look to the files and check if they are built as we defined them.



```
Importing mapping information from "default" entity manager
  > writing src/Entity/Category.php
  > writing src/Entity/Item.php
```

Link with the database - Method 4

Reverse engineering on tables usually do not take into account repositories.
Check if the entity files generated include links to the repository.

If not, we have to include them in both files.

If you do not understand what is a repository, please refer to previous slides. Restart reading from the beginning of “Link with the database”.

Link with the database - Method 4

If entity definition is

```
* @ORM\Entity
```

Replace it by:

```
* @ORM\Entity(repositoryClass="App\Repository\RepositoryName")
```

Where RepositoryName is the repository name for an entity: ItemRepository or CategoryRepository
We will define the RepositoryName files later.
Remember: do not copy lines: character " is not the right one

Link with the database - Method 4

By default, for PostgreSQL, Symfony uses specific “GeneratedValue” for IDs.

```
* @ORM\GeneratedValue(strategy=”SEQUENCE”)
* @ORM\SequenceGenerator(backslashsequenceName=”...”, allocationSize=1,
initialValue=1)
```

Change it by a more appropriate one:

```
* @ORM\GeneratedValue(strategy=”IDENTITY”)
```

This will tell symfony our sequences are used by default. SEQUENCE would be disconnected of the default values and managed a part from the ID.

Link with the database - All methods - Getters and Setters

We will need getters and setters for the attributes in our entities.

There are 2 ways of doing it:

- Manually
  Open each file and create getters and setters for your attributes
- Ask symfony to do it for you - recommanded method -

```
php bin/console make:entity --regenerate
```

Did you notice files modified / generated? In Entity? In Repository?

Link with the database - Repositories

We have the entities: Item and Category.
Now we have to build repositories to manage Items and Categories.

- if you did it manually, you will have to create 2 files in src/Repository: ItemRepository.php and CategoryRepository.php
- If you regenerate entities, these files should have been created and filled as they should be.

Link with the database - Repositories files structure

Here is the structure of a php file to define a repository:

```
<?php
namespace App\Repository;
use App\Entity\EntityName;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Symfony\Bridge\Doctrine\RegistryInterface;
/**
* @method EntityName|null find($id, $lockMode = null, $lockVersion = null)
* @method EntityName|null findOneBy(array $criteria, array $orderBy = null)
* @method EntityName[] findAll()
* @method EntityName[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
*/
class RepositoryName extends ServiceEntityRepository {
        public function __construct(RegistryInterface $registry) {
                parent::__construct($registry, EntityName::class);
        }
}
?>
```

Link with the database - Repositories files structure

In previous slide:

- EntityName is Item or Category
- RepositoryName is ItemRepository or CategoryRepository
- the "@method ..." lines define default methods. You don't have to implement them, ServiceEntityRepository will do it for you.
  - first method defines a method to find 1 object with the ID
  - second method defines a method to find 1 object with criteria
  - third method defines a method to find all objects
  - fourth method defines a method to find a set of objects, with criteria

Link with the database - Database creation

If you used one of the 3 first methods to create entities, you will need a database. So, you have to create it.

```
php bin/console doctrine:database:create
```

You will not have to if you used the last one, reverse engineering.

Link with the database - Table synchronization

Then we have to synchronize Entity files and the database.
You will have to do this each time you change something in the files.

- In Doctrine a modification in the database -in entities- is called a **migration**
- Doctrine generates a new file in src/Migrations to memorise what you did. If you remember to tell it to Doctrine.
- It will also keep modifications in the database in a specific table.

Link with the database - Table synchronization

Use this command to parse entity files and compare them to the database.

```
php bin/console doctrine:migrations:diff
```

You should have a result that explain what you can do with migrations.

- Have a look to src/Migrations.
  There should be a new file. You can change it if you need to.
  Have a look to the file name. What are the characters just after “Version”?
- Have a look to your database.
  There should be a new table: migration_versions

Link with the database - Applying Migrations

Once you created a migration, you have to apply it to the database.

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

migrate plays migrations files to update your database.
In our case, you should have a message that tells migrations did nothing.

Link with the database - Creating data

Sometimes, migrations remove data. The same way, if you create a new table, it is empty. But we need data.

We have to define some data for the database. Symfony call these “fixtures”.
First import the required symfony module. You will have to do it once, with composer.

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

This should create folder **DataFixtures** in src, and file **AppFixtures.php** in this folder.

Link with the database - Creating data

Have a look to file: AppFixtures.php in src/DataFixtures
In this file, there should be a method load. We will add instructions in that method.

First, import Categories and Items in the file. We will need objects and repositories.

```
namespace App\DataFixtures;
use App\Entity\Item;
use App\Entity\Category;
use App\Repository\ItemRepository;
use App\Repository\CategoryRepository;
use Doctrine\Bundle\FixturesBundle\Fixture;
```

Link with the database - Creating data

Then, modify method load to add your data.

For that, we create the list of data to insert.
The we loop on data, create objects, set data, and save it in the database.

To save an object, we persist it in the manager.

Link with the database - Creating data

Here is what it should look like:

```
public function load(ObjectManager $manager) {
    $categories = [ ['electronics'],
        ['appartment'],
        ['pets'],
        ['books'],
        ['smartphone'] ];
    foreach ($categories as $aCategory) {
        $category = new Category();
        $category->setName($aCategory[0]);
        $manager->persist($category);
    }
...
    $manager->flush();
}
```

Link with the database - Creating data

The array $categories contains values we want to insert in the database.
We parse the array, and for each value, we create a new Category.
The created category persists, that means it is stored in the database.

Do the same for Items.

```
$items = [ ['Computer', 'JY Martin', 'I sell my old computer. 3 years old'],
        ['Anime', 'JY Martin', 'Looking for the end of the "Fairy Tail" manga (>126)'],
        ['Elenium', 'JY Martin', 'Looking for the french version of "The Elenium" by David Eddings'],
        ['Kinect', 'JM Normand', 'I sell my new kinect that I can\'t connect to my computer'],
        ['Kikou', 'M Servieres', 'My dog Kikou gave me plenty of little dogs, who wants one?'],
        ['Mangas', 'M Magnin', 'I am looking for the first Alabator Mangas. Anyone get it?'] ];
```

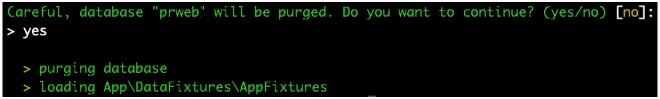
Take care method flush is applied after persisting items. It should be your last instruction.

Link with the database - Creating data

Now, we have to apply fixtures.

```
php bin/console doctrine:fixtures:load
```

Take care: each time you will apply it, existing data will be removed and new data will be created.



```
Careful, database "prweb" will be purged. Do you want to continue? (yes/no) [no]:
> yes

  > purging database
  > loading App\DataFixtures\AppFixtures
```

Building a page

We have data, we defined the entities we will have to manage, we created repositories to manipulate them.

Now, let’s create a page to display data.

Interactions between user and the program are handled by Controllers.
Controllers handle routes, application URL patterns.
Methods that handle routes may use templates to display a response.
Templates are implemented through twig files.

Building a page - About routes

Routes are URI patterns. When you call a route, symfony checks if there is a pattern that matches your URI and maps it.

This has some consequences about the routes:

- You can't use 2 patterns that would match the same URI. There should be only one single pattern that matches an URI.
- Each time you call a route there must be a pattern that matches.

You can see current possible routes with this call:

```
php bin/console debug:router
```

Building a page - About routes

Routes look like URI patterns. It is build as a list of keywords and parameters separated by /. Parameters are inside braces to recognize them from keywords.

Here are some examples:

- /edit/{id} is pattern route edit with parameter id
  if id=2, we call it with /edit/2
- /item/{id}/delete
- /item/list
- /{id}/delete

Always take care you cannot call 2 different paterns from the same URI.

Building a page - Controllers

Controllers are located in src/Controller
Here is the structure of a Controller. **Do not create it now**.

```
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
class ItemController extends AbstractController {
    /**
    * @Route("/item", name="item")
    */
    public function index(): Response {
        return $this->render('base.html.twig',
            [ 'title' =>'ItemController' ]);
    }
}
?>
```

Building a page - Controllers

In previous slide:

- Our controller is called ItemController.
- It implements a method called "index". This name is meaningless. If necessary, you can add parameters. Method index returns a response - an HTML response.
- Annotation **@Route** manages "/item". That means this method in this controller answers "/item" calls.
  Each method you create can handle a route.
- To handle the request, the method uses a render method that refers to a **twig** file. You can create as many twig files as you want.
- Do not forget to include libraries like Request, Response,...

Building a page - Templates

Symfony uses twig to create HTML template files. twig files are located in "templates" folder

Have a look to the file templates/base.html.twig

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %} Welcome! {% endblock %} </title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

Building a page - Templates

In the twig file:

- You may recognize the structure of a HTML file
- {% block ... %}{% endblock %} are parameters that we replace when calling "render", or by using inheritance

Building our first page

As always, there are 2 ways to create a controller:

- manually: you create all files by yourself
- asking symfony to do it for you

```
php bin/console make:controller ItemController
```

Use this second method.

This should create 2 files

- ItemController.php in src/Controller
- index.html.twig in templates/item

Building our first page

- Have a look to the controller.
  It renders “item/index.html.twig” with 1 parameter: ‘controller_name’
- Have a look to templates/item/index.html.twig
  It is not the same structure as base.html.twig
  - First line: we extend base.html.twig, that means this template inherits base.html.twig
  - {% block ... %}{% endblock %} blocks define the elements we have to insert into base.html.twig

Of course you can change the base.html.twig file content, add blocks, ... to implement more complex pages.

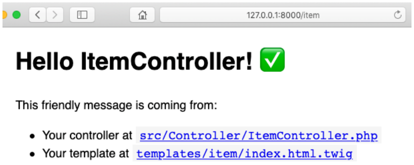
Building our first page

Start the server

```
php bin/console server:run
```

and call your page in the browser

```
http://127.0.0.1:8000/item
```



Hello ItemController! ✅

This friendly message is coming from:

- Your controller at `src/Controller/ItemController.php`
- Your template at `templates/item/index.html.twig`

Building our first page

What happens?

- Browser send a request to the server.
  - Method GET
  - URL: /item
- Server checks which method handles the "/item" route. Find it in ItemController, method index.
- Method Index in ItemController is called. It returns the response: the twig file... filled with parameters.
- response is sent back to the browser.

Display Items

There are 2 ways of writing forms to manage items.

- Manually: you write methods to control request answers
- Using predefined tools.

We will use first one, but let's have a look to the way managing an item with predefined tools.

Using predefined tools

Here is a way of managing forms.

This is not the way we will use. It is only for your information.

First let's install tools.
In the terminal use composer to install forms.

```
composer require symfony/form
```

Using predefined tools

This is not the way we will use. It is only for your information.

We use form with FormTypeInterface and ask Synfony to build our forms.
First we add a method in the controller.

```
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\HttpFoundation\Request;
public function new() {
        $item = new Item();
        // ...
        // Create and render the form
        $form = $this->createForm(ItemType::class, $item);
        return $this->render("items/new.html.twig", ["form" => $form->createView(),]);
}
```

Then we need 2 things: defining ItemType and the twig file.

Using predefined tools

This is not the way we will use. It is only for your information.

First, ItemType. Create folder Form in src and a file ItemType.php inside.

```
namespace App\Form\Type;
use App\Entity\Item;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
// ...
class ItemType extends AbstractType {
        public function buildForm(FormBuilderInterface $builder, array $options) {
                $builder ->add('title', TextType::class)
                                ->add('author', TextType::class)
                                ->add('body', TextType::class);
                                ->add('save', SubmitType::class);
        }
        public function configureOptions(OptionsResolver $resolver) {
                $resolver->setDefaults(['data_class' => Item::class, ]);
        }
}
```

Using predefined tools

This is not the way we will use. It is only for your information.

Then we have to manage form submissions.

```
public function new(Request $request) {
        $item = new Item();
        $form = $this->createForm(TaskType::class, $item);
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
                // get data
                $item = $form->getData();
                // ... manage item
                // ... then route to success
                return $this->redirectToRoute('item_success');
        }
        return $this->render('task/new.html.twig', ['form' => $form->createView(),]);
}
```

Display items

Here is the way we process.

We have to modify the files to display the item list.
First: the controller.

- We need the items list. So we will need the **ItemRepository as a parameter** to get them
- we add the **item list as a parameter to the render call**. We use findAll on the repository for that.
- do not forget to **import/use the ItemRepository** at the beginning of the file.
- Add Request and Response libraries. Default ones may be ok for most request, but they will give an error on some requests.

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

Display items

```
public function index(ItemRepository $itemRepository): Response {
    return $this->render('item/index.html.twig' ,
        [ 'items' =>$itemRepository->findAll(),
    ]);
}
```

Display items

Next: the template -twig- file

The {% block ... %} ... {% endblock %} define blocs in the base.html.twig files.
We can replace their content as we need in index.html.twig

Change the “body” block content by the one next slide

Display items

```
<h1>Auctions list </h1>
<table>
    <tr><th>Id </th><th>Title </th><th>Author </th><th>Body </th></tr>
    {% for item in items %}
    <tr>
        <td>{{ item.id }} </td>
        <td>{{ item.title }} </td>
        <td>{{ item.author }} </td>
        <td>{{ item.body }} </td>
    </tr>
    {% else %}
    <tr>
        <td colspan="4">no records found </td>
    </tr>
    {% endfor %}
</table>
```

Display items

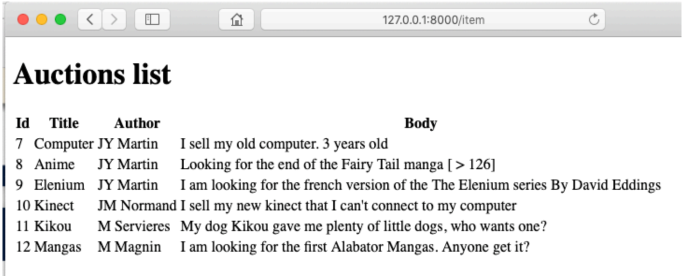
Oh, there are new things in the twig file...

- {% for item in items %}
  - This is a loop on items -which comes from the controller-
  - loop variable is item
- {% else %} inside the loop? Just in case items is empty
- {{ item.id }} means we get the id from item.
  In fact, this is not a direct call to item's id. item.id is converted to item.getId() to call the getter. This is the same for the other attributes.
  That's why we need getters.

Display items

Here should be your result. Maybe ids are not the same.
http://127.0.0.1:8000/item

Why /item? Because of the route we defined in the controller-remember the Route annotation?



127.0.0.1:8000/item

Auctions list

Id	Title	Author	Body
7	Computer	JY Martin	I sell my old computer. 3 years old
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer
11	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?

Public Folder

About public folder

The public folder may contain documents your html page is linked to:

- css files
- images
- js files

The documents will be available with URL /

So, if you create a css folder in public and a main.css file in the css folder, its URL is /css/main.css

Display items

Let’s add some colors.

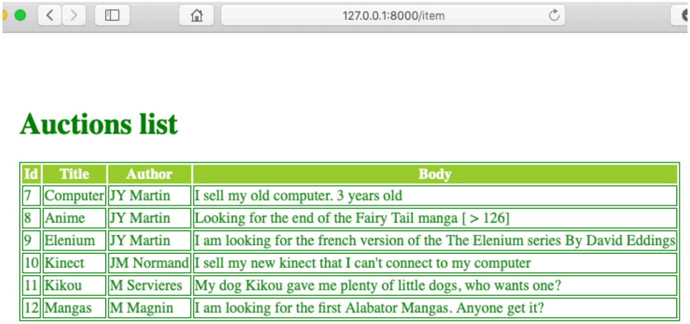
- create folder css in folder public
- copy file main.css from the material to public/css
- add a block ”stylesheets” in index.html.twig

```
{% block stylesheets %}
<link rel=”stylesheet” href=”/css/main.css” />
{% endblock %}
```

- use class ”table” for the table in the body block

Display items

Here should be your result



127.0.0.1:8000/item

Auctions list

Id	Title	Author	Body
7	Computer	JY Martin	I sell my old computer. 3 years old
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer
11	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?

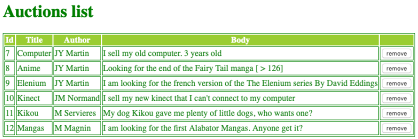
Actions on items

Now, let's add some actions.

In our twig file, add a column to the table, and add an action for each item.
Maybe, for the moment something like this:

```
<td><form><button>remove</button></form></td>
```

Maybe you should add a "th" in the table header too.



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	remove
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	remove
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	remove
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	remove
11	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	remove
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	remove

Actions on items - remove item

Now, we want our controller to do something when we click on the button. That means:

- Adding a method in our controller to remove an item
- Adding something to the form to call the method.

Actions on items - remove item

Let’s start with the method. Add a method in your controller file:

- Route is ”/removeItem”, name is ”removeItem”. Maybe it should be a good idea to add a parameter to the route to know which item we have to delete. ”id” would be a good choice.
- Method name is ... what you want, but should be a good idea to keep removeItem.
- Method has one parameter: the Item we want to remove. Let’s call it myItem. We do not need any other parameter. As we gave parameter id to the route, Symfony is able to find the Item with the right id.
- Maybe you should think about importing/using Item at the beginning of the file.

Actions on items - remove item

- For the method instructions, we have to remove the item. We will ask Doctrine's Entity Manager to do it for us. The Entity Manager is the object that manages entities and that will call the right Repository for that.

```
$em = $this->getDoctrine()->getManager();
$em->remove($myItem);
$em->flush();
```

Actions on items - remove item

- When item is deleted, we have to redisplay the item list. You can:
  - copy the lines from method index, but any change to index should be reported to every copy. Bad idea.
  - route the action so that it calls method index we defined earlier. We'll use method route name. item, not /item.

Let's add this at the end of our removeItem method

```
return $this->redirectToRoute("item");
```

NB: redirectToRoute can have up to 3 parameters, the route name, parameters, and an http code.

Actions on items - remove item

Here should be your method:

```
/**
* @Route("/removeItem/{id}", name="removeItem")
*/
public function removeItem(Item $myItem): Response {
    $em = $this->getDoctrine()->getManager();
    $em->remove($myItem);
    $em->flush();
    return $this->redirectToRoute("item");
}
```

Actions on items - remove item

Ok, now the form in the template, so that we call our method. We have to call “/removeItem”. Also, the URL should contain the item’s id to remove, or we will not be able to call the route in the controller.

Remember the annotation Route in the controller? Route is “/removeItem”, it has a parameter: “id”.
Do you remember route’s name? “removeItem”. We will ask symfony to get the route according to its name.

Actions on items - remove item

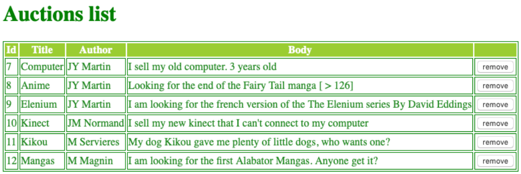
Change button like this:

```
<button formaction="{{ path('removeItem', {'id': item.id} ) }}">
```

item.id is a call to the getter, not a direct access to the id.
formaction replace action in the form by the one in formaction.
path gets route to removeItem, with parameter id equals to item.id.

Actions on items - remove item

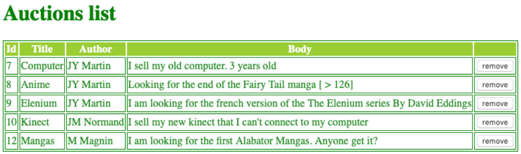
Let's try...



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	remove
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	remove
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	remove
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	remove
11	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	remove
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	remove

Click on "remove" on one item.



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	remove
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	remove
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	remove
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	remove
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	remove

Actions on items - remove item

Do you have an error about App\Controller\Response instead of App\Controller\HttpFoundation\Response?

Did you include the right elements?

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

Actions on items - add item

Now, let’s try to add a new item.
We need :

- elements in the HTML file, the twig file, to be able to add an item
- add a method to add the item

Actions on items - add item

Let’s start with the HTML file. Add a form and a new line to the table in block body in the index.html.twig file.

```
<form action="{{ path('addItem') }}" method="POST">
<tr>
    <td></td>
    <td><input type="text" name="title" id="title"
            size="20" style="background-color:lightgrey;"/></td>
    <td><input type="text" name="author" id="author"
            size="20" style="background-color:lightgrey;"/></td>
    <td><input type="text" name="body" id="body"
            size="60" style="background-color:lightgrey;"/></td>
    <td style="text-align:center"><button>add </button></td>
</tr>
</form>
```

Actions on items - add item

Now the method.

- Create an additem method.
  - Route is “/addItem”,
  - name is “addItem”. Name is the one we chose in the form - remember call to path in previous slide.
- We have to get elements sent by the form. For that, we we use Request informations.
  - add Request $request as a parameter

Actions on items - add item

Now, the method itself:

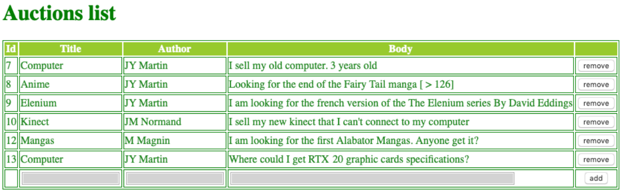
- We use request parameters to create a new item
  - use method get on $request to get informations. Like this:

```
$title = $request->request->get('title');
```

  - get title, author and body elements
  - Create new item and set informations
- Save to database
  - Get Item Manager
  - Persist information to the database - remember fixtures?
  - Do not forget to flush to database.
- And list the items... with a redirection

Actions on items - add item

Click on "add" button.



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	remove
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	remove
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	remove
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	remove
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	remove
13	Computer	JY Martin	Where could I get RTX 20 graphic cards specifications?	remove
				add

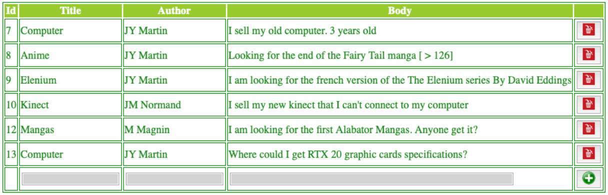
Actions on items - icons

Wouldn't it be a little bit more cute if we use icons?

Add icons in public/img

Replace buttons contents by icon call.

```
<img src="/img/plus.png" height="20" />
```



Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	🗑
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	🗑
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	🗑
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	🗑
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	🗑
13	Computer	JY Martin	Where could I get RTX 20 graphic cards specifications?	🗑
				⊕

Actions on items - modify item

Now, we have to modify items.
We will create 2 buttons. First one to edit, second one to save.

- First button will switch the line so that text will become editable text.
  Then it sets the other buttons as visible, and becomes invisible.
- Second button will gather informations and launch form to save the item content.
  Then it reloads the page.

You can use utilities.js in the materials. Maybe functions defined in this file will be useful.

Actions on items - modify item

First, we create a button in the form of the last TD of the line.
This button calls a script.

- Add a button for each item line, with the “edit” icon. Put it after the removeItem button.
  - Button calls a JS function on event “onclick” with 2 parameters:
    - “this”. Remember, “this” is the object, the button.
    - the auction ID
  - Button is in the form and we do not want this form to be submitted. So, you should return false after calling the editItem function.

```
<button onclick="editItem(this, item.id);return false;" ...
```

- Ensure button is visible, and has property “display” set, with something like that: style=”display:inline”.

Actions on items - modify item

Now, the JS function.

- Create a js folder in public.
- add utilities.js to public/js, if you use it, otherwise you will have to create your own functions.
- Create a JS file in public/js. Call it myScript.js
- Create a function editItem in **myScript.js** with 2 parameter called theButton and theID
- add a bloc “scripts” in **base.html.twig**. New block should be located just after block “stylesheets”.
  This block will allow us to add the JS scripts.
- define a block “scripts” in **index.html.twig** and import the JS files in this block, like you would do in any HTML file.

Actions on items - modify item

Now, the editItem function itself. theButton and theID are the parameters.

- get auction informations through an AJAX call. See next slide for that.
- get the TR element that contains the button. You can use “getNextParentTag” in utilities.js.
- parse TD children. Maybe “getTag” can help you.
  - skip first one,
  - process title, author and body: convert text to an editable element.
    maybe “convertTextToInput” can help you.
- Get FORM parent of the button, show all buttons inside, hide current one. Have a look to “showAll”.

Actions on items - modify item

The AJAX call to get auction needs:

- A JS function to get auction information
- It sends an AJAX call using JQuery. We you synchronous method, method is POST.
- We need a controller AjaxController, so create it with composer.
- And a method to give a response to our request.

Actions on items - modify item

Your JS script to get the auction informations may look like this.

```
function getItemValues(theID) {
        var returned = null;
        $.ajax({
                url: "/ajaxGetAuction",
                method: "POST",
                data: {
                        "id": theID
                },
                async: false,
                success: function (result) {
                        returned = result;
                },
                error: function (resultat, statut, erreur) {
                        console.log("error" + resultat.responseText + statut + erreur);
                }
        });
        return returned;
}
```

Actions on items - modify item

Now, the method in the AjaxController

- Route is "/ajaxGetAuction", Method is POST.
- We need Request as a parameter
- We will get the repository thanks to the Request parameter
- we build an array as a response
- to send it back as a JSON object we use **json_encode** and we change response header to tell it is a JSON response.

Actions on items - modify item

Here is the method:

```
public function ajaxGetAuction(Request $request) : Response {
        $itemRepository = $this->getDoctrine()->getRepository(Item::class);
        $id = $request->request->get('id');
        $categoryId = -1;
        $valItem = $itemRepository->find($id);
        if ($valItem->getCategory() != null) {
                $categoryId = $valItem->getCategory()->getId();
        }
        $item = array('id' => $valItem->getId(),
                        'author' => $valItem->getAuthor(),
                        'title' => $valItem->getTitle(),
                        'body' => $valItem->getBody(),
                        'category' => $categoryId
        );
        // Build response
        $response = new Response(json_encode($item));
        $response->headers->set('Content-Type', 'application/json');
        return $response;
}
```

Actions on items - modify item

Now, the second button.

- Add a button just beside the “edit” button, with the “save” icon.
- As removeItem, this button validates the form. Its route is “/modifyItem”, with parameter id, the same kind of route than removeItem.
- Make it invisible with something like that: style=”display:none”

When we click on the “edit” button, other buttons are shown, then the “edit” button is hidden. So, it looks like if the 2 buttons switch.

Actions on items - modify item

Now, the php function in the IndexController to save the auction.

- Create a new function with the route modifyItem. Route has a parameter id.
  This parameter is used to get the persistent item. The one in the database.
- Your function should have an Item has a parameter.
  Remember removeItem?
  Understand why we need id as a parameter?
- Your function should modify the title, author and body, and ask the Entity Manager to flush to save to the database.
- Then it redirects to the route that lists items.

Did you notice there is something missing?

Actions on items - modify item

How do we get new title, new author and new body?
Answer: from the request

But... they are not in the form!!!
Answer: So we have to add them to the form.

Yes but... they are edited outside the form
Answer: call a JS function that gets editable texts and add them to the form. When you click on the "save" button, to be sure we have the last values.

Actions on items - modify item

Ok, lets add a JS function to our "save" button.

- add a call to a JS function modifyItem on event "onclick".
- modifyItem has 1 parameter: the button itself
  - modifyItem gets parent TR of the button
  - get INPUT values in the TR - Maybe you should have a look to getCurrentValues

```
var values = getCurrentValues(trRef);
var title = values.title;
...
```

  - create hidden fields with title, author and body inside the form. Maybe you can have a look to addHidden.

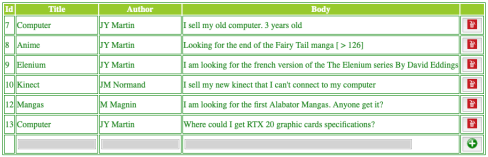
Actions on items - modify item

Now let's get our data in the controller function.

- Add request as a parameter for the function.
- Get title, author and body from request
  Take care:
  - when using post, you get data with $request->request.
  - when using get, you get data with $request->query.
- Set item fields to their new values
- flush entity manager.

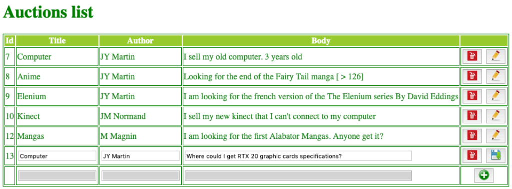
Actions on items - modify item

Let's try...



Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	
13	Computer	JY Martin	Where could I get RTX 20 graphic cards specifications?	

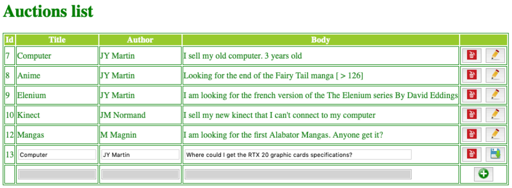
Click the edit button



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	
13	Computer	JY Martin	Where could I get RTX 20 graphic cards specifications?	

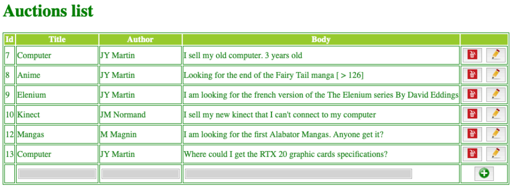
Actions on items - modify item



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	
13	Computer	JY Martin	Where could I get the RTX 20 graphic cards specifications?	

Change data and click save.



Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	
13	Computer	JY Martin	Where could I get the RTX 20 graphic cards specifications?	

Login Controller

Now, we need a login screen.

Add a LoginController in the controllers. use composer for that.
Add an index method in your controller with Route "/".
Call twig file in index.twig.html templates/login.

You should have a folder login and a file index.html.twig. Add login informations in the twig file. Maybe you can use materials.
Add form and call route login.

Add new method in LoginController that manages route "login".
Check login and password. Maybe with "admin", "admin" as (login/password).
if ok, route to index. If not, reload login page.

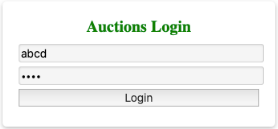
Login Controller

Now let’s consider using method POST to manage authentication. We have to add it in the form and take it into account in the annotation Route.

```
{% block body %}
        <form method="POST">
            <h1>Auctions Login</h1>
            <p><input type="login" name="user" placeholder="Login"></p>
            <p><input type="password" name="passwd" placeholder="Password"></p>
            <p><button formaction="{{ path('login') }}">Login</button></p>
        </form>
{% endblock %}
```

```
/**
 * @Route("/login", name="login", methods={"POST"})
 */
public function login(Request $request): Response {
    $login = $request->request->get('user');
    $passwd = $request->request->get('passwd');
```

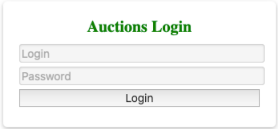
Login Controller



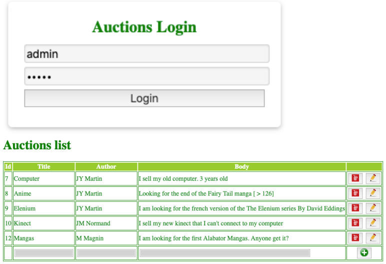
Auctions Login

abcd

••••



Auctions Login



Auctions Login

admin

•••••

Auctions list

Id	Title	Author	Body	
7	Computer	JY Martin	I sell my old computer. 3 years old	
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]	
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	

Use categories

Last step, using categories.

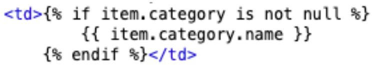
Our auctions may have a category so, we have to take this into account:

- in the twig file for each auction.
- still in the twig file to add an auction.
- in the controllers to save an auction

Use categories

First, column category in the twig file.

- Add a category column to the auction list
- for each auction add category name...
  ... but it could be null, so we have to test it.



```
<td>{% if item.category is not null %}
        {{ item.category.name }}
    {% endif %}</td>
```

Use categories

Then, still in the twig file, when we add an item we must be able to select a category.

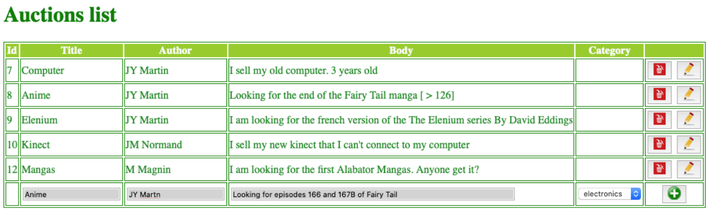
- In the controller, in the method that display the auction list, use CategoryRepository to get all categories.
- send then to the twig file
- use a tag select to display categories when you want to add an auction.
  - name is categoryId
  - option values are category IDs

Use categories

When we add an auction, we now get the category ID.

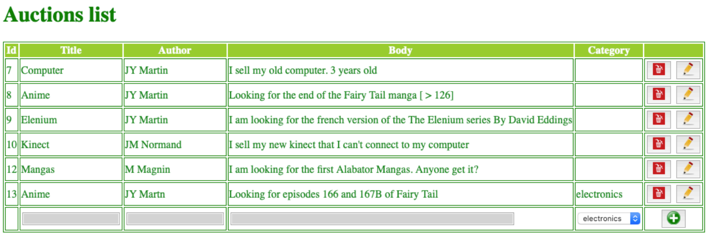
- get categoryId from the request
- use method find in the CategoryRepository to get the category
- Set it to the auction
- save auction

Use categories



Auctions list

Id	Title	Author	Body	Category	
7	Computer	JY Martin	I sell my old computer. 3 years old		
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]		
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
	Anime	JY Martn	Looking for episodes 166 and 167B of Fairy Tail	electronics	



Auctions list

Id	Title	Author	Body	Category	
7	Computer	JY Martin	I sell my old computer. 3 years old		
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]		
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
13	Anime	JY Martn	Looking for episodes 166 and 167B of Fairy Tail	electronics	
				electronics	

Use categories

Then we have to propose categories when editing an auction. We will do it with AJAX.

- in your JS script, get category list from the server to propose it as a selection. Use an AJAX call too.
- save data when clicking on “save”

Use categories

For the AJAX call to get the list of categories, you could do an AJAX call the same way we get the item. As we do not need to send any data, attribute data should be removed.

Then, have a look to utilities. Maybe you can find something to create a SELECT item.
Your SELECT name should be categoryId.

Use categories

For the AJAX response to get categories, you could do something like this:

```
$categoryList = $categoryRepository->findAll();
$categoryArray = array();
foreach ($categoryList as $category) {
        $categoryArrayElt = array('id' => $category->getId(), 'name' => $category->getName());
        $categoryArray[] = $categoryArrayElt;
}
$response = new Response(json_encode($categoryArray));
```

Use categories

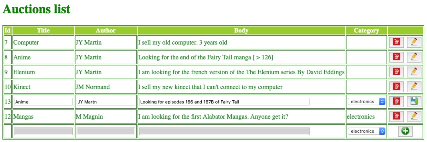
Then we have to save the category.

- In the JS file, in modifyItem, getCurrentValues should retrieve the categoryId value. If the SELECT name is categoryId.
- Add an hidden field your for categoryId with the SELECT value.
- in your ItemController, in method modifyItem, get categoryId value. Set it to the item you modify.

```
$theItem->setCategory($categoryRepository->find($categoryId));
```

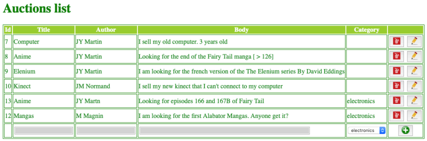
Use categories

Now, check everything is ok.



Auctions list

Id	Title	Author	Body	Category	
7	Computer	JY Martin	I sell my old computer. 3 years old		
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]		
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
13	Anime	JY Martn	Looking for episodes 166 and 167B of Fairy Tail	electronics	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	electronics	
				electronics	



Auctions list

Id	Title	Author	Body	Category	
7	Computer	JY Martin	I sell my old computer. 3 years old		
8	Anime	JY Martin	Looking for the end of the Fairy Tail manga [> 126]		
9	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
10	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
13	Anime	JY Martn	Looking for episodes 166 and 167B of Fairy Tail	electronics	
12	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	electronics	
				electronics	

Success
NEXT EXIT