SYMFONY

- Primero se intala composer y se dan los permisos necesarios
  En linux:
  curl -sS http://getcomposer.org/installer | php -- --filename=composer
  chmod a+x composer
  sudo mv composer /usr/local/bin/composer

  En Windows:
  se instala el fichero .exe

- Installar Symfony
  - En linux

```
$ sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

  - Ejecutar symfony

```
symfony new my_app
cd my_app
php bin/console server:run
```

  - En windows

```
c:\> php -r "readfile('http://symfony.com/installer');" > symfony
```

- Si no funciona

```
composer create-project symfony/website-skeleton my-project
cd my-project
composer require symfony/web-server-bundle --dev
```

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

- Más info en
  https://symfony.com/doc/current/setup.html

- Comandos importantes de symfony
  - **To start the server: php bin/console server:run**
  - **To clear cache: php bin/console cache:clear**
  - **To create the DB: php bin/console doctrine:database:create**
  - **To validate the DB: php bin/console doctrine:schema:validate**
  - **To update the DB: php bin/console doctrine:schema:update --force**
  - **To start wizard to create an entity: php bin/console doctrine:generate:entity**
  - **To create and entity: php bin/console doctrine:generate:entities AppBundle/Entity/Client**
  - **To generate missing getters and setters:**
    php bin/console doctrine:generate:entities AppBundle

- Pasos:
  - Primero se crea la tabla con doctrine:database:create
  - luego se crea la entidad con doctrine:generate:entity (nombre AppBundle:Client)
  - se dan nombre a los campos y se decide que tipo de campo y longitud

- despúes de crear cada entidad utilizamos los comandos update --force y clear:cache
- se añade el `manyToOne:` (para relacionar las tablas) y luego se utiliza el comando generate:entities AppBundle

- Composer
  - en el fichero json se añade en require `"guzzlehttp/guzzle": "^6.2"`
  - luego se actualiza composer -> composer update
- PhP Unit
  - en el fichero json se añade en require-dev `"phpunit/phpunit": "3.7.*"`
  - luego se lansa el comando php ./vendor/phpunit/phpunit/phpunit.php
  - luego se actualiza composer -> composer update

2. ==========

- Instalar xdebug
  - primero comprobar la versión de php
    http://localhost/dashboard/phpinfo.php
  - copiamos todo y vamos a esta web donde nos recomendara el fichero dll necesario para nuestro equipo (se copia y se pega toda la información del phpinfo)
    https://xdebug.org/wizard.php
  - luego ir a la página de xdebug y descargar la versión para nuestro php según las instrucciones
    https://xdebug.org/download.php
  - 1. Download php_xdebug-2.6.1-7.2-vc15.dll

2. Move the downloaded file to C:\xampp\php\ext
3. Edit `C:\xampp\php\php.ini` and add the line
`zend_extension = C:\xampp\php\ext\php_xdebug-2.6.1-7.2-vc15.dll`
4. Restart the webserver

  - en php.ini pegamos el código

```
; XDEBUG Extension
zend_extension = C:\xampp\php\ext\php_xdebug-2.6.1-7.2-vc15.dll

xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_host=127.0.0.1
xdebug.remote_port=9000
xdebug.remote_autostart=0
xdebug.remote_connect_back=0
xdebug.profiler_enable=0
xdebug.profiler_enable_trigger=0
xdebug.profiler_output_name=cachegrind.out.%s.%t
xdebug.profiler_output_dir="H:/xampp/tmp/xdebug"
xdebug.trace_output_dir="C:/xampp/tmp/xdebug"
```

xdebug.max_nesting_level=250

- hacemos las modificaciones correctas para que apunte en el directorio xampp/php/lib y que apunte al ficherito descargado
- volvemos a arrancar el xampp

3. ==========

- Instalar cygwin
  - se descarga cygwin y se instala https://cygwin.com/install.html
  - ante de finalizar la instalación se descargan y se instalan los paquetes:
    - git -> devel -> distributed version control system
    - ssh -> net -> openssh: the openssh server and client programs
    - nano -> editors -> nano: enhanced clone of pico editor
    - wget -> web -> wget: utility to retrive files from www ...

- Instalar Symfony
  - utilizando el comando dentro de cygwin
    `composer create-project symfony/framework-standard-edition symfony/ "3.0.0"`
  - luego completamos con los datos en el asistente
    - database_host: enter
    - database_name: prueba
    - enter a todo
  - ls -lh para ver todos los componentes instalados
  - limpiar el cache si da algún error
  - en php.ini hacer algún cambio
    - en intl.error lo ponemos a 0: ;intl.use_exceptions = 0
    - xdebug.max_nesting_level a 250
  - en caso de errores se cambia el fichero SymfonyRequirements.php

```php
protected function getRealpathCacheSize()
{
    $size = ini_get('realpath_cache_size');
    $size = trim($size);
    $unit = strtolower(substr($size, -1, 1));
    switch ($unit) {
        case 'g':
            return (int)$size * 1024 * 1024 * 1024;
        case 'm':
            return (int)$size * 1024 * 1024;
        case 'k':
            return (int)$size * 1024;
        default:
            return (int) $size;
    }
}
```

  - si sale el error de countable() borramos el cache y añadimos el código

```
/**
* Defines PHP required version from Symfony version.
*
* @return string|false The PHP required version or false if it could not be guessed
*/
protected function getPhpRequiredVersion()
{
        if (!file_exists($path = __DIR__.'/../composer.lock')) {
                return false;
        }
        $composerLock = json_decode(file_get_contents($path), true);
        foreach ($composerLock['packages'] as $package) {
                $name = $package['name'];
                if ('symfony/symfony' !== $name && 'symfony/http-kernel' !== $name) {
                        continue;
                }
                return (int) $package['version'][1] > 2 ? self::REQUIRED_PHP_VERSION :
                self::LEGACY_REQUIRED_PHP_VERSION;
        }
        return false;
}
```

- Para comprobar el hello word debemos acceder a la ruta de desarrollo app_dev http://localhost/symfony3_app/web/app_dev.php/hello-world
- Para poder acceder directamente en web -> app.php -> ponemos el AppKernel en true

- Para crear controladores vamos al directorio src
  - en el directorio por defecto AppBundle -> Controller y creamos un nuevo fichero PruebaController.php
  - en el nuevo fichero PruebaController.php indicamos los parámetros recibidos y el return

```
    public function indexAction(Request $request, $name, $page) {
        //redireccionar a la página de index
//        return $this->redirect($this->generateUrl("homepage"));
        //redireccionar a una página en concreto
//        return $this->redirect($this->container->get("router")->getContext()->getBaseUrl()."/hello-
world?hola=true");
//        return $this->redirect($request->getBaseUrl()."/hello-world?hola=true");

        //crear una vista
        // replace this example code with whatever you need
        return $this->render('AppBundle:pruebas:index.html.twig', array(
                'texto' => $name . " - " . $page,
        ));
    }
```

  - en app -> config -> route.yml especificamos la ruta del nuevo fichero yml routing

```
rutas_bundle:
    resource: "@AppBundle/Resources/config/routing.yml"
    prefix: /
```

```
app:
   resource: "@AppBundle/Controller/"
   type:     annotation
```

- en el directorio AppBundle creamos el directorio Resources -> contiene los directorios config y views
  - en config creamos el fichero routing.yml que contiene

```
pruebas_index:
   path: /pruebas/{lang}/{name}/{page}
   defaults: {_controller: AppBundle:pruebas:index, lang: es, name: Gonzalez, page: 1}
   methods: [GET, POST]
   requirements:
      #accepta caracteres y números
      #name: \w+
      #accepta solo caracteres
      name: "[a-zA-Z]*"
      page: \d+
      lang: es|en|fr
```

- en views se crea el directorio pruebas del controlador PruebasController que contiene el fichero index.html.twig

- Crear Bundles
  - con el comando
    `php bin/console generate:bundle --namespace=MiBundle --format=yml`
  - completamos el asistente de comandos
    - configuration format: yml

- Funciones
  - cuando se crean nuevas funciones hay que añadirlas en app -> config -> service.yml

- Filtros de Twig
  - https://twig.symfony.com/doc/2.x/

- Generar entidades a partir de la BD
  - se usa el comando, una vez que la bd ha sido creada

```
php bin/console doctrine:mapping:convert xml
./src/BlogBundle/Resources/config/doctrine/metadata/orm --from-database --force
```

- importar los datos en yml con el comando

```
php bin/console doctrine:mapping:import AppBundle yml
```

- generar las entidades para interactuar con el comando

```
php bin/console doctrine:generate:entities AppBundle
```

- Si no tenemos la BD las entidades se generan de este modo

```
php bin/console doctrine:generate:entity
```

- se le da un nombre, luego se completa el asistente para describir los campos, logitud etc
- luego se genera las tablas con el comando a partir de la entidad ya generada

```
php bin/console doctrine:schema:update --force
```

- si queremos eliminar todas las entidades y las tablas de la bd

```
php bin/console doctrine:schema:drop --force
```

- para crear las tablas utilizamos el comando create

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

- limpiar el cache

```
php bin/console cache:clear
```

- Si estas usando la versión 3.0.0 no tendrás problemas con esto en los siguientes vídeos, pero si usas la versión 3.0.6 o superior de Symfony date cuenta que cuando hacemos esto

```
 $em = $this->getDoctrine()->getEntityManager()
```

En una acción debes cambiarlo a esto

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

Para que funcione correctamente.

- Insertar datos en la BD con doctrine
  - en el controller.php se crea la acción

```
   public function createAction(){
      $curso = new Curso();
      $curso->setTitulo("Curso Symfony3");
      $curso->setDescripcion("Curso completo de Symfony3");
      $curso->setPrecio(12);

      //llamamos al entity manager
      $em = $this->getDoctrine()->getEntityManager();
      //persistimos los datos
      $em->persist($curso);
      //volcamos los datos en la bd
      $flush = $em->flush();

      if($flush != null){
         echo "El curso no se ha creado bien";
      }else{
         echo "Curso creado correctamente";
      }

      die();
   }
```

  - luego se crea la ruta en route.yml

```
pruebas_create:
   path: /pruebas/create
   defaults: { _controller: AppBundle:pruebas:create}
```

- Leer desde la bd
  - primero creamos la función en el controller

```
   public function readAction(){
```

```
        //llamamos al entity manager
    $em = $this->getDoctrine()->getEntityManager();

    //sacamos los cursos
    $cursos_repo = $em->getRepository("AppBundle:Curso");
    $cursos = $cursos_repo->findAll();

    //mostrar los cursos
    foreach ($cursos as $curso){
        echo $curso->getTitulo()."<br/>";
        echo $curso->getDescripcion()."<br/>";
        echo $curso->getPrecio()."<br/><hr/>";
    }

    die();
}
```

- luego creamos la ruta

```
pruebas_read:
  path: /pruebas/read
  defaults: { _controller: AppBundle:pruebas:read}
```

- Actualizar registros con doctrine
  - creamos la acción

```
public function updateAction($id, $titulo, $descripcion, $precio) {
    //llamamos al entity manager
    $em = $this->getDoctrine()->getEntityManager();
    //sacamos los cursos
    $cursos_repo = $em->getRepository("AppBundle:Curso");

    //buscar un registro por id
    $curso = $cursos_repo->find($id);
    $curso->setTitulo($titulo);
    $curso->setDescripcion($descripcion);
    $curso->setPrecio($precio);

    //persistimos los datos
    $em->persist($curso);
    //volcamos los datos en la bd
    $flush = $em->flush();

    if ($flush != null) {
        echo "El curso no se ha actualizado bien";
    } else {
        echo "Curso actualizado correctamente";
    }

    die();
}
```

  - luego creamos la ruta

- Eliminar
  - creamos la acción

```
public function deleteAction($id) {
    //llamamos al entity manager
    $em = $this->getDoctrine()->getEntityManager();
    //sacamos los cursos
    $cursos_repo = $em->getRepository("AppBundle:Curso");
    //buscar un registro por id
    $curso = $cursos_repo->find($id);

    $em->remove($curso);
    //volcamos los datos en la bd
    $flush = $em->flush();

    if ($flush != null) {
        echo "El curso no se ha borrado bien";
    } else {
        echo "Curso borrado correctamente";
    }

    die();
}
```

  - creamos la ruta

- Tipos de find
  - find($id)
  - findAll()
  - findBy(array("precio" => 14)
  - findOneByPrecio(12) -> la palabra Precio es variable, asi que se puede reemplazar con lo que se quiere buscar

- Consultas en SQL nativo

```
public function nativeSqlAction() {
    //llamamos al entity manager
    $em = $this->getDoctrine()->getEntityManager();
    //conexión a la bd
    $db = $em->getConnection();

    $query = "SELECT * FROM cursos";
    $stmt = $db->prepare($query);
    $params = array();
    $stmt->execute($params);

    $cursos = $stmt->fetchAll();

    //mostrar los cursos
    foreach ($cursos as $curso) {
        echo $curso["titulo"] . "<br/><hr/>";
    }

    die();
```

```
}
```

- y su correspondiende ruta

- DQL
  - hacer consulta a objetos con doctrine DOCTRINE QUERY LANGUAGE

```
public function docquerylangAction() {
    //llamamos al entity manager
    $em = $this->getDoctrine()->getEntityManager();

    $query = $em->createQuery(""
                . "SELECT c FROM AppBundle:Curso c "
                . "WHERE c.precio > :precio "
                . "")->setParameter("precio", "12");

    $cursos = $query->getResult();

    //mostrar los cursos
    foreach ($cursos as $curso) {
        echo $curso->getTitulo() . "<br/><hr/>";
    }

    die();
}
```

- y su ruta

- Query Builder
  - para construir consultas

```
public function createQueryBuilderAction() {
    //llamamos al entity manager
    $em = $this->getDoctrine()->getEntityManager();

    //sacamos los cursos
    $cursos_repo = $em->getRepository("AppBundle:Curso");

    $query = $cursos_repo->createQueryBuilder("c")
            ->where("c.precio > :precio")
//            ->orderBy("precio asc")
            ->setParameter("precio", "12")
            ->getQuery();

    $cursos = $query->getResult();

    //mostrar los cursos
    foreach ($cursos as $curso) {
        echo $curso->getTitulo() . "<br/>";
        echo $curso->getDescripcion() . "<br/>";
        echo $curso->getPrecio() . "<br/><hr/>";
    }

    die();
}
```

- y su ruta

- En el repositorio de symfony van todos los métodos y todas las consultas a la bd y en controller lo llamamos
- Para más información sobre BD y Doctine en http://symfony.com/doc/current/doctrine.html

- Crear formulario
  - `php bin/console doctrine:generate:form AppBundle:Curso`
  - hay que incluir en la cabecera

```
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
```

    - para más información sobre formularios https://symfony.com/doc/current/forms.html

4. ==========

Pasos para crear una aplicacación con symfony 3.0.*:

- para instalar symfony mirar según la versión http://symfony.com/doc/3.0/setup.html
- crear la base de datos y llenarla
- generar bundle: php bin/console generate:bundle
- cambiar parameters.yml
- posiblemente hace falta hacer un composer update
- generar las entidades en base a la bd:
  php bin/console doctrine:mapping:convert xml ./src/BlogBundle/Resources/config/doctrine/metadata/orm --from-database --force
- poner los nombres de las tablas en singular (los ficheros generados .orm.xml)
- abrimos los ficheros .orm.xml y tambien ponemos los name="" en singular
- generamos la configuaración de las entidades:
  php bin/console doctrine:mapping:import BlogBundle yml
- a los nuevos ficheros .yml generados también les cambiamos los nombres a singular
- también en singular los nombres de los ficheros
- generamos las clases php de las entidades:
  php bin/console doctrine:generate:entities BlogBundle
- en casos de error con generate:entities intentar
  - compser update
  - php bin/console make:entity --regenerate
  - `composer remove maker y composer require maker --dev`
  - volver a hacer composer update
- si hay un error 404 en web/app.php -> appKernel se pone en true
- generar formulario de registro
  - `php bin/console doctrine:generate:form BlogBundle:User`
- si al generar un formulario sale un error [Twig_Error_Runtime] vamos a vendor -> sesio -> generator-bundle -> Resources -> skeleton -> Form -> FormType.php.twig en la linea 29 posiblemente escribimos
  {%- if fields_mapping[field] is defined and fields_mapping[field]['type'] in ['date', 'time', 'datetime']%}
- se borra la linea de arriba y se vuelve a utilizar el comando para generar el formulario

- Crear, editar y eliminar sesiones

Para crear sesiones en Symfony tenemos que utilizar el servicio “Session” el cual podemos instanciar o llamarlo directamente con el método get accesible en cualquier controlador.
Creamos una sesión:

```
$this->get('session')->set("sesion_prueba", "conntenido de la sesión");
```

Conseguimos la sesión:

```
echo $this->get('session')->get("sesion_prueba");
```

Eliminamos la sesión:

```
$this->get('session')->remove("sesion_prueba");
```

Así de fácil, podemos trabajar con las sesiones.

- Instalar Bundles y Librerias

En Symfony podemos instalar nuevos bundles desarrollador por terceros. En http://knpbundles.com/ tenemos gran cantidad de ellos.
Para instalar alguno de ellos simplemente tendríamos que añadirlo al composer.json de la raíz del proyecto.
En el array require tenemos una serie de bundles que hemos instalado por defecto

```
"require": {
"php": ">=5.5.9",
"symfony/symfony": "3.0.*",
"doctrine/orm": "^2.5",
"doctrine/doctrine-bundle": "^1.6",
"doctrine/doctrine-cache-bundle": "^1.2",
"symfony/swiftmailer-bundle": "^2.3",
"symfony/monolog-bundle": "^2.8",
"sensio/distribution-bundle": "^5.0",
"sensio/framework-extra-bundle": "^3.0.2",
"incenteev/composer-parameter-handler": "^2.0"
},
```

Imaginate que queremos instalar un bundle para hacer migraciones. Pues tendriamos que añadir el repositorio y la versión a este array.

```
"doctrine/migrations": "v1.2.1"
```

Después tenemos que ir al directorio del proyecto desde la consola y lanzar el comando **composer update**, para que se descargue los nuevos paquetes y actualice los existentes si es necesario.
Si queremos instalar cualquier librería de PHP que tu estés acostumbrado a utilizar, el procedimiento es exactamente el mismo. Puedes buscarla en https://packagist.org/ y añadirla al require del composer.json de la misma forma.
En el caso que quisiéramos cargar una librería propia, podríamos crearnos un directorio library (en la raíz por ejemplo), y ahí meter nuestras clases propias. Añadiríamos al json el campo autoload:

```
"autoload":{
"classmap": ["library/"]
}
```

De igual forma que antes hay lanzar el comando **composer update.**

- Migraciones

Las migraciones de la base de datos **nos permiten crear diferentes versiones del esquema de la base de datos** mientras estamos desarrollando una aplicación.
Esto es muy útil cuando estás trabajando en un equipo y cada programador hace sus cambios en la base de datos, etc, cuando otro programador mezcle el código del compañero podrá lanzar las migraciones que ha generado el compañero y así tener **la misma versión de la base de datos de forma fácil**.
También cuando desplegamos la web, es decir cuando pasamos la aplicación de desarrollo a producción, en

este **proceso de deploy se lanzan las migraciones para tener la última base de datos** en producción.
Veamos cómo hacer migraciones con Symfony.
En primer lugar instalamos el bundle https://github.com/doctrine/DoctrineMigrationsBundle
DoctrineMigrationBundle con Composer.
En nuestro fichero composer.json, en la entrada require añadimos:
doctrine/doctrine-migrations-bundle": "dev-master"
Ahora lanzamos el comando:
composer update
Una vez que se instalen o actualicen todos los paquetes entramos al fichero **app/AppKernel.php** y añadimos al array bundles la instancia del objeto de las migraciones.
new Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle(),
Ahora en la consola tenemos disponibles una serie de comandos para hacer nuestras migraciones.
Si queremos generar una migración vacía para meterle lo que queramos podemos usar el comando
php bin/console doctrine:migrations:migrate

- Paginación con KnpPaginatorBundle

La mejor forma de hacer una paginación en Symfony es utilizando el bundle KnpPaginatorBundle, en este tutorial aprenderemos a usarlo:

https://victorroblesweb.es/2016/02/27/paginacion-con-knppaginatorbundle-en-symfony3/

- Ajax

Para comprobar en la acción del controllador si una petición es por AJAX, podemos usar el siguiente método:

```
if ($request->isXmlHttpRequest()) {

}
```

Recuerda que antes debes pasar por parámetro el objeto Request a método action.

- Generar/Actualizar una sola entidad

Si tienes que actualizar las entidades porque por ejemplo cambias un campo o lo añades a tu tabla, lo puedes hacer de dos formas. La recomendable sería añadir el campo a la definición de la entidad (yml) y añadir la propiedad, el getter y el setter a tu clase o modelo en PHP. La segunda opción sería re-generar la entidad de nuevo.

Si necesitas crear otra tabla simplemente la creas y lanzas los comandos de nuevo y en teoría te generara la entidad que no exista en el proyecto.

En nuestro caso Incluso indicándole a los comandos el parámetro --filter puedes especificar qué tabla vas a convertir a entidad en doctrine.

```
php bin/console doctrine:mapping:import MiBundle yml --filter="TuTabla"
```

http://stackoverflow.com/questions/10371600/generating-a-single-entity-from-existing-database-using-symfony2-and-doctrine

http://symfony.com/doc/3.0/doctrine.html

- Publicar un proyecto
  - en la carpeta var se eliminan los directorios cache, logs y sessions
  - eliminar carpeta nbproject
  - en el fichero sql se eliminan las lineas de create database y use database
  - en el hosting creamos la bd y damos credenciales a admin, al crear la bd se le pone un nombre y se nos da un usuario
  - vamos a phpmyadmin de la bd e importamos el fichero sql
  - entramos en app -> parameters.yml y ponemos el nombre de la bd y las credenciales del usuario para el acceso
  - en AppBundle -> Resource -> view -> layout.html.twig -> borramos nombre-proyecto-symfony/web/app_dev.php que es lo que se utilizaba en produción
  - utilizamos filezila para subir el proyecto
  - en el public_html subimos todo
  - reescribir url de .htaccess
    - creamos el fichero .htaccess

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteRule ^(.*)$ /web/$1 [QSA,L]
</IfModule>
```