

# DESARROLLO WEB



## Contenido

- 1 El proyecto4**
  - 1.1 Gestionar ofertas de empleo5**
    - 1.1.1 Página principal5
    - 1.1.2 Detalles de empleo5
    - 1.1.3 Nuevo empleo6
  - 1.2 Gestión de Curriculum vitae (CV)7**
    - 1.2.1 Añadir un nuevo CV7
  - 1.3 Gestión de usuarios7**
- 2 Persistencia8**
  - 2.1 El modelo de datos Entidad-Relación8**
- 3 Creación del proyecto10**
  - 3.1 Preparación del entorno10**
    - 3.1.1 Instalación de Symfony10
    - 3.1.2 Estructurando la aplicación10
    - 3.1.3 La filosofía de Symfony11
- 4 La base de datos13**
  - 4.1 Integración de la base de datos13**
  - 4.2 Generación de las entidades13**
  - 4.3 Valores iniciales14**
    - 4.3.1 Doctrine Fixture Bundle14
    - 4.3.2 DataFixture15
- 5 El layout17**
  - 5.1 Webpack Encore19**
    - 5.1.1 Configurando Encore/Webpack121
    - 5.1.2 Plantilla base24

## Desarrollo según el rol

- 6 Visitante26**
  - 6.1 Página principal26**
    - 6.1.1 Controlador26
    - 6.1.2 Vista28
- 7 Gestor de ofertas de empleo30**
  - 7.1 Publicar una oferta31**
  - 7.2 Visualizar Mis Ofertas31**

**8 Gestor de CVs32****8.1 Controladores de CV33**

- 8.1.1 Creación de un CV33
- 8.1.2 Edición de un CV37
- 8.1.3 Búsqueda de un CV40

**9 Gestión de usuarios41****9.1 Jerarquía de las entidades41**

- 9.1.1 Entidad base: Usuario42

**10 Gestión de acceso47****10.1 Formularios de acceso48**

- 10.1.1 Registro de usuario48
- 10.1.2 Acceso de usuario49

**11 Acceso a la web52****Manual de aplicación****12 Exalumno54**

- 12.1 Registro de CV54
- 12.2 Barra de herramientas55
- 12.3 Mi CV55
- 12.4 Ventana de edición55
- 12.5 Aplicar a una oferta56
- 12.6 Mis Candidaturas56

**13 Empresa58**

- 13.1 Registro58
- 13.2 Barra de herramientas58
- 13.3 Publicar ofertas58
- 13.4 Detalles de oferta59
  - 13.4.1 Candidatos60
- 13.5 Buscador de ofertas60

# ARTEAN: Bolsa de empleo

## 1 El proyecto

Todo el mundo está hablando de la crisis hoy en día. El desempleo está aumentando de nuevo.

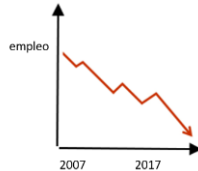


FIGURA 1.1. Gráfica de empleo de la última década.

Lo sé, los desarrolladores formados en *ITC Cuatrovientos* no están realmente preocupados y es por eso por lo que te estas esforzando al máximo en la asignatura de *Desarrollo Web*.

Dentro de poco te enfrentarás al mercado laboral, con la difícil tarea de encontrar un empleo que desarrolle tu potencial. Para ello, necesitas encontrar una plataforma que ofrezca un buen trabajo. ¿*InformaTrabajos* dices? Piensa otra vez. ¿*Dónde puedes encontrar un empleo de desarrollador de Symfony?* ¿*Dónde puedes anunciar tus habilidades de Symfony?*

Necesitas una plataforma que ofrezca trabajos orientada a desarrolladores. Una donde puedas encontrar a la mejor gente, los expertos. Una donde sea fácil, rápido y divertido buscar un trabajo, o proponer uno.

No busque más. *Artean* es el lugar. Debes de saber que no estás solo, desde *ITC Cuatrovientos* vamos a ayudarte. Pero, antes tienes la oportunidad de demostrar tus conocimientos. Vamos a desarrollar un software *Open-Source* que sólo hace una cosa, pero lo hace bien. Es fácil de usar, personalizar, ampliar e integrar.

Antes de sumergirnos en el código de cabeza, describamos el proyecto un poco más. Las siguientes secciones describen las características que queremos implementar.

El sitio web de *Artean* tiene cuatro tipos de usuarios:

- Visitantes: **anónimos**.
- Usuarios de rol **exalumno**.
- Usuarios de rol **empleador** (Empresa que oferta el empleo).
- Usuario **administrador** de la aplicación.

## 1.1 Gestionar ofertas de empleo

### 1.1.1 Página principal

#### Todos los usuarios

En la página de inicio, el usuario puede ver los últimos trabajos.

Cuando un usuario entra a la página web *Artean*, ve una lista de los puestos de trabajo activos. Los puestos de trabajo se clasifican por categoría y a continuación, por fecha de publicación (los nuevos puestos de trabajo primero). Para cada puesto de trabajo, sólo se muestran la ubicación, la posición, y la empresa.



FIGURA 2.1. Página principal.

### 1.1.2 Detalles de empleo

#### Todos los usuarios

Un usuario hace clic en un puesto de trabajo para ver información más detallada.

El usuario puede seleccionar un trabajo de la lista para ver información más detallada.



FIGURA 2.2. Detalles de empleo.

## 1.1.3 Nuevo empleo

### Usuario empleador

Un usuario registrado envía un puesto de trabajo.

Un usuario empleador puede enviar un puesto de trabajo. Un puesto de trabajo está formado por varias partes de información:

- Compañía
- Tipo (full-time, part-time, o freelance)
- Logo (opcional)
- URL (opcional)
- Posición
- Ubicación
- Categoría (el usuario elige una de una lista de posibles categorías)
- Descripción del trabajo (URL y correos electrónicos son enlazados de forma automática)
- Cómo aplicar (URL y correos electrónicos son enlazados de forma automática)
- Público (si el trabajo también puede ser publicados en sitios web afiliados)
- Email (email del ofertante)



The screenshot shows a web browser window with the URL <http://cuatrovientos.org>. The page header includes the ARTEAN logo and the text 'Bolsa de empleo.' Below the header, there is a search bar with the placeholder text 'Buscar...' and a 'Publicar' button. The main section is titled 'Publicar un trabajo' and contains a form with the following fields: 'Categoría' (a dropdown menu), 'Tipo' (radio buttons for 'Tiempo Completo', 'Media Jornada', and 'Freelance'), 'Compañía' (a text input field), 'Logo' (a text input field), 'URL' (a text input field), 'Cargo' (a text input field), 'Localización' (a text input field), and 'Descripción' (a text input field).

FIGURA 2.3. Nuevo empleo.

## 1.2 Gestión de Curriculum vitae (CV)

### 1.2.1 Añadir un nuevo CV

#### Usuario con rol de exalumno

Se considera que un CV puede estar compuesto por varias experiencias laborales previas, diferentes titulaciones de estudios y otros conocimientos, por ejemplo, idiomas. El CV es pues un conjunto de todos estos conceptos, por eso el *asistente CRUD* no nos ha facilitado un simple formulario.

Debemos indicar un flujo de pantallas que nos permita completar esta tabla mediante una introducción recursiva de registros en las tablas de *studies*, *work\_experiences*, *another\_abilities*.

## 1.3 Gestión de usuarios

#### Usuario con rol de administrador

Los visitantes son libres de registrarse como exalumno o empresa, la verificación la realizarán los administradores del sitio. Dependiendo del rol seleccionado deberán completar las propiedades correspondientes a su categoría.

#### El usuario puede solicitar convertirse en empleador

El usuario puede convertirse en Empleador para lo que debe completar el asistente de registro como empleador y ser validado por el administrador de *Artean*.

#### El usuario puede registrarse como exalumno

El usuario puede registrarse como exalumno para lo que debe completar el asistente de registro y ser validado por el administrador de *Artean*.

#### Administrador: configura el sitio web

El administrador puede:

- editar las categorías en el sitio web
- edita y elimina cualquier trabajo publicado
- Validar roles de usuario: activar y desactivar usuarios.

## 2 Persistencia

A continuación, se opresenta el esquema de tablas de base de datos con el que se va a estructurar la información la aplicación *Artean*.

### 2.1 El modelo de datos Entidad-Relación

Aquí está el diagrama de entidad-relación completo en el que se muestran los campos de las tablas de base de datos y sus relaciones.

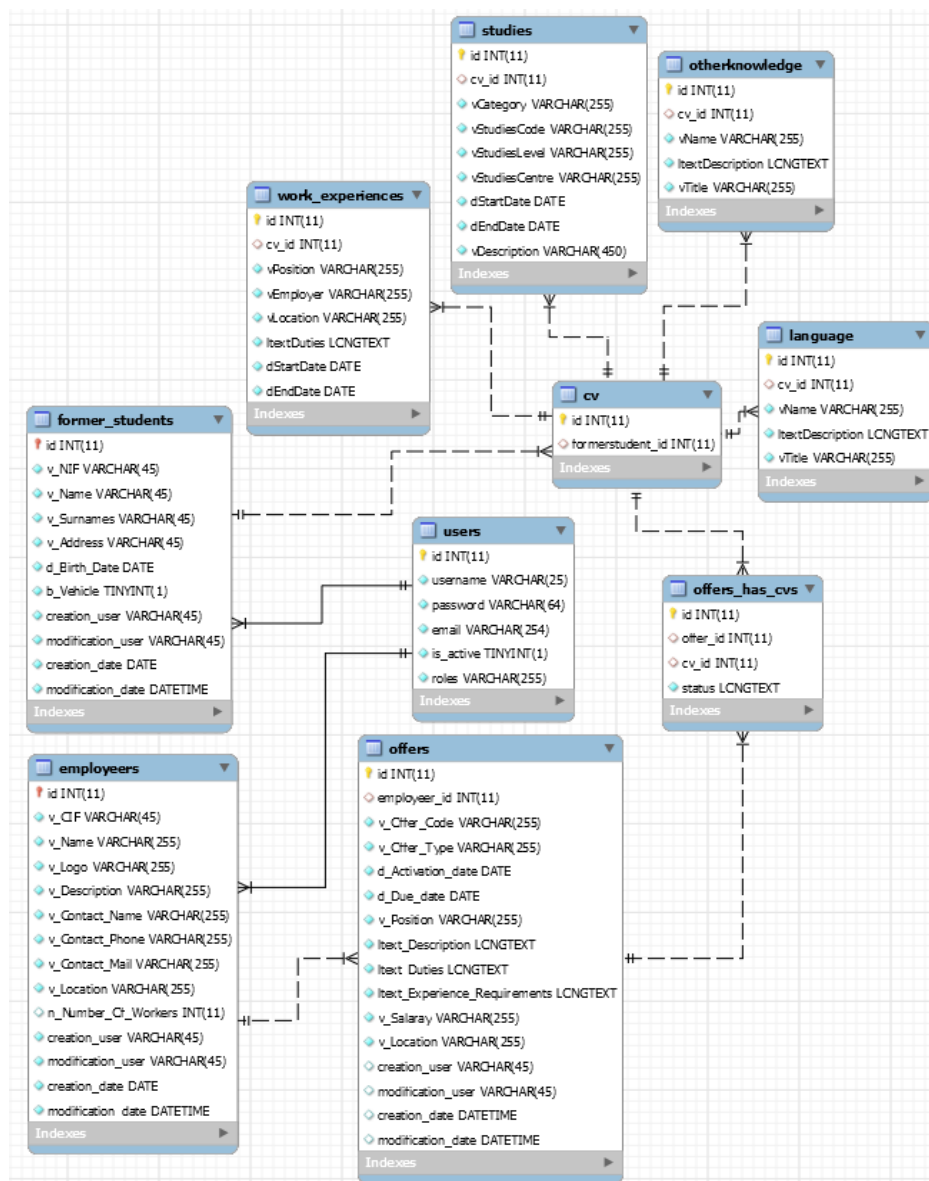


FIGURA 2.4. Entidad-Relación completo.



**NOTA:** Existe una tabla extra que sirve para almacenar valores de configuración en base de datos. Se menciona más adelante.

Las tablas principales cuentan con campos de auditoría (la idea es que se implementen mediante *triggers* o desde la aplicación) para establecer el usuario creador y modificador de los registros de base de datos. En un primer vistazo se pueden apreciar 3 subsistemas.

1. Exalumnos – CV

- Los Exalumnos tienen CVs.
- Los CVs están compuestos de estudios, experiencias laborales, idiomas y otros conocimientos.

2. Empresas - Ofertas

- Las empresas publican ofertas.
- Las ofertas tienen Curriculum Vitae (CVs).

Cuando los exalumnos se inscriben las ofertas lo hacen presentando un CV, este CV puede pasar por varios estados dentro de la oferta.

3. Gestión de usuarios:

- Establece los accesos de usuario y asigna roles de usuario. A nivel de aplicación se establecerán los permisos de cada usuaria a partir de su rol.
- Los usuarios *pueden* ser exalumnos.

## 3 Creación del proyecto

A continuación, se explica mediante una guía paso a paso como llevar a cabo este proyecto.

### 3.1 Preparación del entorno

El Entorno de Desarrollo Intergrado (IDE) utilizado ha sido PhpStorm al que se le ha añadido el *plugin de Symfony* y su dependencia del *plugin de Annotations*.

#### 3.1.1 Instalación de Symfony

La primera tarea que realizar es descargar el instalador de Symfony, lo hacemos desde la web oficial. <https://symfony.com/download>

Creamos nuestro proyecto.

```
C:\> symfony new artean-bolsa-empleo --version=1ts
```

Para ejecutar nuestra aplicación lanzaremos el comando

```
C:\> symfony console server:start
```

#### 3.1.2 Estructurando la aplicación

Después de crear la aplicación, vemos un número de archivos y directorios generados automáticamente:

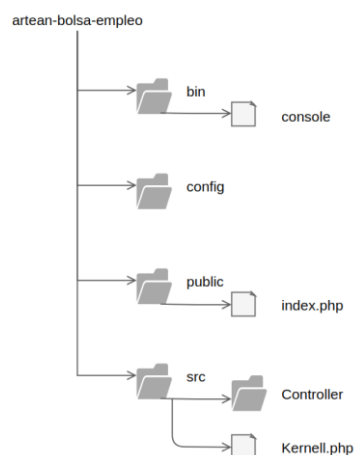


FIGURA 2.5. Estructura del proyecto

Esta jerarquía de archivos y directorios es la convención propuesta por Symfony para estructurar sus aplicaciones.

El propósito recomendado de cada directorio es el siguiente:

- `config/`, almacena toda la configuración definida para cualquier entorno;
- `src/`, almacena el código específico de Symfony (controladores y rutas), su código de dominio (por ejemplo, clases de doctrina) y toda su lógica de negocio;
- `var/cache/`, almacena todos los archivos caché generados por la aplicación;
- `var/logs/`, almacena todos los archivos de registro generados por la aplicación;
- `var/sessions/`, almacena todos los archivos de sesión generados por la aplicación;
- `vendor/`, este es el directorio donde Composer instala las dependencias de la aplicación y nunca debe modificar ninguno de sus contenidos;
- `public/`, almacena todos los archivos del controlador frontal y todos los activos web, como hojas de estilo, archivos JavaScript e imágenes.

### 3.1.3 La filosofía de Symfony

La filosofía de Symfony

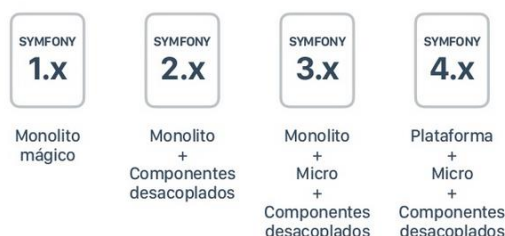


FIGURA 2.6. La filosofía de symfony. (Eguiluz, J. 2017. *deSymfony*)

Del mismo modo que la v2.x trajo componentes desacoplados y la v3 introdujo el **microkernel**, symfony 4 vendrá con **cambios importantes**:

- **Adiós al monolito:** Hasta ahora se buscaba que, de base, symfony fuese capaz de aportar todo lo necesario independientemente del tipo de aplicación a desarrollar: web, api, consola, etc. No en vano, la edición estándar de **symfony** consta de más de 40 componentes, 15 librerías y 11 bundles. Este enfoque ha muerto. A partir de ahora tendremos micro aplicaciones base a las que añadir aquello que requiramos. Tanto es así que la base de symfony 4 tiene un 70% menos de líneas de código que 3.3.

- **Nueva estructura de directorios:** Sin cambios revolucionarios, pero encontrarás varios directorios reubicados o renombrados. También se centralizan los contenidos de front: Assets, templates, etc.

### Estructura de directorios por defecto

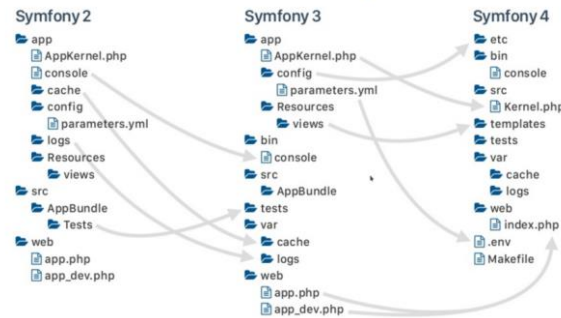


FIGURA 2.7. Estructura de directorios por defecto. (Madariaga, M. 2017)

No más bundles privados: La separación modular del código de aplicación se hará exclusivamente vía namespaces de PHP. El concepto bundle se conserva exclusivamente para software de terceros instalado vía composer.

`parameters.yml` desaparece para apostar por variables de entorno.

## 4 La base de datos

### 4.1 Integración de la base de datos

#### Instalación

Realizamos la instalación de orm a través del Symfony Pack y también añadimos MakerBundle, que nos ayudará a generar algo de código.

```
composer require symfony/orm-pack  
composer require symfony/maker-bundle
```

#### Configuración

Las opciones de configuración relacionadas con la base de datos van en el archivo `config/package/doctrine.yaml`. A continuación, modificamos el nombre de la base de datos en la configuración de parámetros.

```
doctrine:  
  dbal:  
    url: mysql://artean:artean@127.0.0.1:3306/bolsaartean
```

#### Creación

Lanzamos el comando generador de la base de datos.

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

### 4.2 Generación de las entidades

Las entidades deben cumplir con los requisitos establecidos en el análisis y en la definición de la tabla de base de datos.

Pasamos a crear las entidades, vamos a tener que ejecutar el siguiente comando por cada una de las tablas de nuestra tabla de base de datos.

```
C:\> php bin/console make:entity
```

Este comando lanza el asistente que tendremos que completar con los campos definidos a la hora de establecer la base de datos.

#### Ingeniería inversa

Podemos actualizar nuestro modelo de datos de múltiples formas; generando las clases *php* de forma manual y añadiendo las correspondientes anotaciones,

generando las entidades con el asistente o actualizando nuestra aplicación desde una base de datos ya generada.

En esta ocasión vamos a generar el resto de nuestro modelo de datos desde la base de datos creada con *MySQL Workbench* para lo que ejecutaremos el siguiente comando.

```
php bin\console doctrine:mapping:import App annotation
```

## Actualización del esquema

En caso de que realicemos modificaciones en las clases de nuestras entidades y queramos actualizar el esquema de base de datos lanzaremos el siguiente comando.

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

## 4.3 Valores iniciales

### 4.3.1 Doctrine Fixture Bundle

Para poner algunos datos iniciales en nuestra base de datos, crearemos el comando de la consola que nos permite llenar una base de datos. Pero antes de hacerlo, necesitaremos un componente adicional llamado *Fixtures*.

Vamos a instalar el *bundle* DoctrineFixturesBundle y a generar nuestro contenido de mentira.

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

Una vez que se actualizan las dependencias podemos comprobar que se han añadido las siguientes líneas en `composer.json`.

```
"require-dev": {
    "doctrine/doctrine-fixtures-bundle": "^3.0",
    ...
},
```

La dependencia se ha agregado en el kernel, en el archivo `config/bundles.php`

```
...
new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle();
...
```

Y con todo instalado, podemos empezar a crear fixtures.

Para generar valores aleatorios de las ofertas de empleo vamos a hacer uso de otro Bundle. *Faker* es una biblioteca PHP que genera datos falsos para nosotros.

Una manera sencilla de poner en marcha una base de datos para hacer pruebas de desarrollo.

```
composer require --dev fzaninotto/faker
```

```
"require-dev": {
    ...
    "fzaninotto/faker": "^1.7",
    ...
},
```

### 4.3.2 DataFixture

Los *DataFixture* son clases de PHP donde se crean objetos y los almacenamos de manera persistente en la base de datos. Imagina que deseamos inicializar nuestra base de datos. *¡No hay problema!*

Creamos una clase de AppFixtures dentro de una carpeta DataFixtures dentro de src.

```
<?php
namespace App\DataFixtures;

use Faker;
use App\Entity\OfferMgr\Offers;
use App\Entity\UserMgr\Employeers;

use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        // Creating 20 job offers
        for ($i = 0; $i < 20; $i++)
        {
            $jobFaker = Faker\Factory::create();

            // Employeeer
            $employeeer = new Employeers();
            $employeeer->setUsername("Cuatrovientos_{$i}");
            $employeeer->setEmail("artean_{$i}@cuatrovientos.org");
            $employeeer->setPassword("4Vientos");

            $employeeer->setVCIF("82102288A");
            $employeeer->setVName($jobFaker->company);
            $employeeer->setVLogo($jobFaker->imageUrl($width=640, $height=480));
            $employeeer->setVDescription($jobFaker->sentence);
            $employeeer->setVContactName($jobFaker->name);
            $employeeer->setVContactPhone($jobFaker->phoneNumber);
        }
    }
}
```

```

$employeeer->setVContactMail($jobFaker->companyEmail);
$employeeer->setVLocation($jobFaker->address);
$employeeer->setNNumberOfWorkers($jobFaker->numberBetween(0,255));
$employeeer->setCreationUser("InitialFixture");
$employeeer->setCreationDate(new \DateTime("2018-6-1"));
$employeeer->setModificationUser("InitialFixture");
$employeeer->setModificationDate(new \DateTime("2018-6-1"));

$manager->persist($employeeer);

// Offer
$offer = new Offers();
$offer->setVOfferCode("ACTIVE");
$offer->setVOfferType('full-time');
$offer->setDActivationDate(new \DateTime("2018-6-1"));
$offer->setDDueDate(new \DateTime("2018-6-1"));
$offer->setVPosition(implode(' ', $jobFaker->words(2)));
$offer->setLtextDuties($jobFaker->sentence);
$offer->setLtextDescription($jobFaker->paragraph);
$offer->setVSalaray("1200");
$offer->setLtextExperienceRequirements($jobFaker->paragraph);
$offer->setVLocation($jobFaker->city.', '.$jobFaker->country);

$offer->setEmployeeer($employeeer);

$offer->setCreationUser("InitialFixture");
$offer->setCreationDate(new \DateTime("2018-6-1"));
$offer->setModificationUser("InitialFixture");
$offer->setModificationDate(new \DateTime("2018-6-1"));

$manager->persist($offer);
}

$manager->flush();
}
}

```

Añadimos la siguiente configuración en el archivo `app/config/services.yml`

```

App\DataFixtures\
  resource: '../src/DataFixtures'
  tags: ['doctrine.fixture.orm']

```

Ya solo nos falta ejecutarlo, es muy fácil, pero mucho cuidado que hará un *truncate* de todas nuestras tablas:

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



# 5 Configurando un panel de administración

Un panel de administración es una sección protegida del sitio web donde los administradores del proyecto pueden gestionar los datos del sitio web. *¿Cómo podemos crearlos rápidamente?* Utilizando un *bundle* que genera un panel de administración a partir del modelo de datos. EasyAdmin es perfecto para esa tarea.

## 1.1 Configurando EasyAdmin

En primer lugar, añade EasyAdmin como una dependencia del proyecto.

```
symfony composer req admin
```

Para configurar EasyAdmin se ha generado un nuevo archivo de configuración a través de su receta Flex. El archivo `config/package/easy_admin.yaml`

La práctica totalidad de los paquetes instalados tienen una configuración similar a ésta en el directorio `config/packages`. Casi siempre los valores predeterminados habrán sido escogidos cuidadosamente para que funcione en la mayoría de las aplicaciones.

Quita la marca de comentarios del primer par de líneas y agrega las clases del modelo del proyecto.

```
easy_admin:
  entities:
#      # List the entity class name you want to manage
    - App\Entity\CvMgr\CVCategories
    - App\Entity\CvMgr\Languages
    - App\Entity\CvMgr\Otherknowledge
    - App\Entity\CvMgr\Studies
    - App\Entity\CvMgr\WorkExperiences
    - App\Entity\CvMgr\CV
    - App\Entity\OfferMgr\Offers
    - App\Entity\OfferMgr\Offers_has_CVs
    - App\Entity\UserMgr\Employeeers
    - App\Entity\UserMgr\FormerStudents
    - App\Entity\UserMgr\User
```

Accede al panel de administración generado en `/admin`. Ya disponer de una interfaz de administración rica en funciones.

## 6 El layout

En primer lugar, si realizas una mirada más cercana a los prototipos, *mockups*, te darás cuenta de que gran parte de cada página se ve igual. Ya sabes que la duplicación de código es mala, ya sea que hablemos de código HTML o PHP, por lo que necesitamos encontrar una forma de evitar que estos elementos de vista común resulten en duplicación de código.

Una forma de resolver el problema es definir un encabezado y un pie de página e incluirlos en cada plantilla:

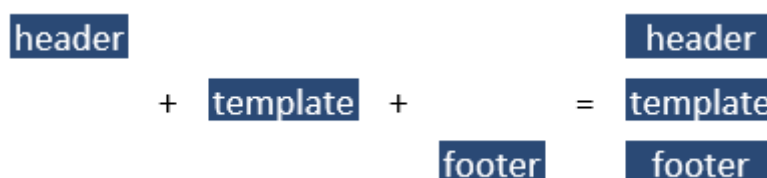


FIGURA 2.8. Layout.

Pero aquí el encabezado y los archivos de pie de página no contienen HTML válido. Debe haber una manera mejor. En lugar de reinventar la rueda, usaremos un patrón de diseño para resolver este problema: **el patrón de diseño del decorador**. El patrón de diseño del decorador resuelve el problema al revés: **la plantilla está decorada después**.

**Puedes analizar el siguiente ejemplo:**

[https://es.wikipedia.org/wiki/Decorator\\_\(patron\\_de\\_diseño\)#Ejemplo\\_de\\_PHP](https://es.wikipedia.org/wiki/Decorator_(patron_de_diseño)#Ejemplo_de_PHP)

A continuación, vamos a darle un aspecto más usable a nuestra web. Para ello comenzamos integrando los archivos CSS y JS de *Bootstrap* de una manera sencilla.

Lo primero que vamos a hacer es instalar el bundle de assets.

```
composer require symfony/asset
```

Symfony viene con una biblioteca de JavaScript, llamada Webpack Encore, que hace que trabajar con CSS y JavaScript sea una delicia.

## 6.1 Webpack Encore

*Webpack Encore* es una forma más simple de integrar *Webpack* en una aplicación. Webpack proporciona una API limpia y potente para agrupar módulos de JavaScript, preprocesar CSS y JS y compilar y minificar archivos.

Realizamos la instalación ejecutando el siguiente comando:

```
npm install @symfony/webpack-encore --save-dev
```

Instalamos las dependencias de *Webpack* con estos tres comandos.

```
npm install node-sass
npm install sass-loader@^8.0.0 --save-dev
npm install webpack-notifier
```

También instalamos las dependencias de Bootstrap y Bootstrap.

```
npm install --save jquery popper.js
npm install --save bootstrap
```

Además, añadimos unos iconos para usar en nuestro proyecto.

```
npm install --save webpack-icons-installer
```

Tenemos un simple proyecto, con una carpeta *assets* donde agrupamos las hojas de estilos SCSS siguiendo la siguiente arquitectura, para que el código CSS sea fácil de mantener y evolucionar y ayudando a evitar problemas relacionados con la herencia de estilos.

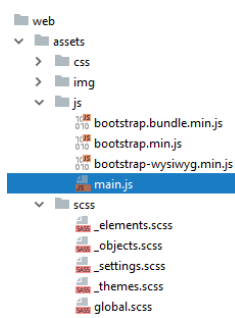


FIGURA 2.9. Carpeta de assets.

Una buena práctica de estructura de carpetas para los estilos es:

- **01\_settings:** Contiene las variables globales, configuración, colores, etc. común para todo el proyecto.
- **02\_tools:** Mixins y funciones comunes para todo el proyecto.
- **03\_generic:** Estilos para unificar la visualización en navegadores (por ejemplo normalize.css) e impresoras, declaración de fuentes, reglas globales para box-sizing, etc.
- **04\_elements:** Estilos a etiquetas de HTML.
- **05\_objects:** Patrones de layout estructurales repetitivos, sin ningún aspecto visual, por ejemplo, un [media object](#).
- **06\_components:** Piezas de la interfaz de usuario reconocibles, con estilos visuales propios. También llamados componentes.
- **07\_themes:** Estilos para cambiar el aspecto de componentes y elementos basándose en la clase del theme.
- **08\_specifics:** Estilos específicos de partes de una página que aún no se ha encontrado el patrón para convertirlo en componente.
- **09\_utilities:** Clases totalmente especializadas que aportan estilos que sobrescriben cualquiera de los anteriores si fuera necesario. Cada clase contiene un único estilo, a diferencia de los componentes.

Dentro de la carpeta css debemos tener un archivo app.scss donde importamos los elementos css.

```
@import 'settings';
@import 'elements';
@import 'objects';
@import 'themes';
...
```

Con *Encore*, debemos pensar en CSS como una dependencia del JavaScript. Esto significa que hay que requerir cualquier CSS que se necesite desde dentro de JavaScript:

```
require('../scss/global.scss');
// CSS
require('../css/bootstrap.min.css');
require('../css/bootstrap-grid.min.css');
require('../css/bootstrap-reboot.min.css');
// JS
require('./bootstrap.min.js');
require('./bootstrap.bundle.min.js');
// Icons
require('./webpack-icons-installer');
console.info('Your script is loaded.');
```

Con Encore, podemos minimizar fácilmente estos archivos, preprocesar app.scss a través de Sass y mucho más.

### 6.1.1 Configurando Encore/Webpack

Creamos un nuevo archivo llamado webpack.config.js en la ruta raíz de nuestro proyecto.

En el interior, establecemos el siguiente código de configuración.

```
var Encore = require('@symfony/webpack-encore');
Encore
    // directory where all compiled assets will be stored
    .setOutputPath('public/web/build/')
    // Windows??? <-----
    .setManifestKeyPrefix('build')
    // what's the public path to this directory (relative to your project's
    document root dir)
    .setPublicPath('/build')
    // empty the outputPath dir before each build
    .cleanupOutputBeforeBuild()
    // will output as web/build/app.js
    .addEntry('app', './web/assets/js/main.js')
    // will output as web/build/global.css
    .addStyleEntry('global', './web/assets/scss/global.scss')
    // allow sass/scss files to be processed
    .enableSassLoader()
    // allow legacy applications to use $/jQuery as a global variable
    .autoProvidejQuery()
    .enableSourceMaps(!Encore.isProduction())
    // create hashed filenames (e.g. app.abc123.css)
    // .enableVersioning()
;
// export the final configuration
```

```
module.exports = Encore.getWebpackConfig();
```

Debemos crear las carpetas *build* dentro la carpeta web. Para realizar la compilación de los assets utilizamos el siguiente comando, ejecutado desde la ruta donde se encuentra *Encore*.

```
call ./node_modules/.bin/encore dev --context ./
```

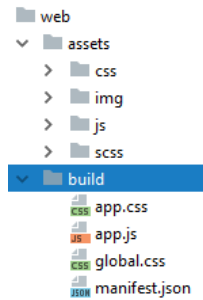


FIGURA 2.10. Webpack build.

Después de ejecutar uno de estos comandos, ahora se puede agregar las secuencias de comandos y etiquetas de enlace a los nuevos recursos compilados (por ejemplo, `/build/app.css` y `/build/app.js`).

Actualmente el documento `base.html.twig` se encuentra dentro de la carpeta `app` en `Resources>views`. Todas las páginas van a partir de este mismo punto raíz. Desde este archivo se enlazan las hojas de estilos.

## Iconos

Para hacer uso de los iconos instalados actualizamos dentro del directorio build el archivo manifest.json estableciendo las rutas para descarga de las fuentes y los iconos que vamos a utilizar.

```
{
  "build/app.css": "/build/app.css",
  "build/app.js": "/build/app.js",
  "build/global.css": "/build/global.css",
  "build/fonts/MaterialIcons-Regular.woff": "/build/fonts/MaterialIcons-Regular.012cf6a1.woff",
  "build/fonts/MaterialIcons-Regular.woff2": "/build/fonts/MaterialIcons-Regular.570eb838.woff2",
  "build/fonts/MaterialIcons-Regular.ttf": "/build/fonts/MaterialIcons-Regular.a37b0c01.ttf",
  "build/fonts/MaterialIcons-Regular.eot": "/build/fonts/MaterialIcons-Regular.e79bfd88.eot",
  "build/fonts/fontawesome-webfont.eot?v=4.7.0": "/build/fonts/fontawesome-webfont.674f50d2.eot",
  "build/fonts/fontawesome-webfont.woff?v=4.7.0": "/build/fonts/fontawesome-webfont.af7ae505.woff2",
  "build/fonts/fontawesome-webfont.ttf?v=4.7.0": "/build/fonts/fontawesome-webfont.b06871f2.ttf",
  "build/fonts/fontawesome-webfont.woff?v=4.7.0": "/build/fonts/fontawesome-webfont.fee66e71.woff",
  "build/fonts/glyphicons-halflings-regular.woff2": "/build/fonts/glyphicons-halflings-regular.448c34a5.woff2",
  "build/fonts/glyphicons-halflings-regular.ttf": "/build/fonts/glyphicons-halflings-regular.e18bbf61.ttf",
  "build/fonts/glyphicons-halflings-regular.eot": "/build/fonts/glyphicons-halflings-regular.f4769f9b.eot",
  "build/fonts/glyphicons-halflings-regular.woff": "/build/fonts/glyphicons-halflings-regular.fa277232.woff",
  "build/images/fontawesome-webfont.svg?v=4.7.0": "/build/images/fontawesome-webfont.912ec66d.svg",
  "build/images/glyphicons-halflings-regular.svg": "/build/images/glyphicons-halflings-regular.89889688.svg"
}
```

## 6.1.2 Instalar el motor de plantillas Twig

Realizamos la instalación de twig con el siguiente comando.

```
composer req twig
```

En nuestro proyecto se crea el directorio templates que es donde almacenaremos nuestras vistas.

## 6.1.3 Plantilla base

A continuación el código actualizado de este archivo desde donde se enlazan a los archivos css y js en el archivo base.html.twig:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>{% block title %}Artean{% endblock %}</title>
  {% block stylesheets %}
    <link rel="stylesheet"
      href="{{asset('build/app.css')}}">
  {% endblock %}
</head>
<body>

{% block navbar %}
  {% if is_granted('IS_AUTHENTICATED_FULLY') %}
    {% include 'navbar/private.html.twig' %}
  {% else %}
    {% include 'navbar/public.html.twig' %}
  {% endif %}
{% endblock %}

{% block body %}  {% endblock %}

{% block javascripts %}
  <script src="{{asset('build/app.js')}}"> </script>
{% endblock %}

</body>
<footer class="footer">
  {% block footer %}{% endblock %}
</footer>
</html>
```



# Desarrollo según el rol.

Aplicación desglosada según los roles de usuario

## 7 Visitante

### Usuario anónimo

A continuación, se desarrollan las pantallas dentro del ámbito de un usuario visitante.

### 7.1 Página principal

#### 7.1.1 Controlador

En la página principal se muestra un listado de las ofertas de empleo más recientes.

#### Repositorio de clases personalizadas

Con el objetivo de aislar, reutilizar y probar consultas, es una buena práctica crear una clase de repositorio personalizada para una entidad. Los métodos que contienen la lógica de la consulta pueden almacenarse en esta clase.

Para hacer esto, agregamos el nombre de clase del repositorio a la definición de mapeo de su entidad:

```
<?php
/** @title Offers ... */
namespace App\Entity\OfferMgr;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Table(name="offers")
 * @ORM\Entity(repositoryClass="App\Repository\OfferMgr\OffersRepository")
 */
class Offers {
    ...
}
```

Luego, creamos una clase `Repository\OfferMgr\OffersRepository` vacía que se extienda desde `DoctrineORMEntityRepository`.

Luego, agregamos un nuevo método - `findAllActive()` - a la clase `OffersRepository` recién generada. Este método consultará todas las entidades de ofertas activas, ordenadas alfabéticamente por fecha.

```
<?php
/** @title OffersRepository ...*/
namespace App\Repository\OfferMgr;

use Doctrine\ORM\EntityRepository;

class OffersRepository extends EntityRepository {

    public function findAllActive()
    {
        return $this->getEntityManager()
            ->createQuery(
                "SELECT o FROM App:OfferMgr\Offers o WHERE o.vOfferCode='ACTIVE'
ORDER BY o.modificationDate ASC"
            )
            ->getResult();
    }
}
```

Dentro de la carpeta de controladores establecemos la lógica de comunicación entre la petición desde URL de visualización de la pantalla principal, la obtención de los datos a través de los Repositorios de entidades y de la plantilla html.

Por cada una de las ofertas obtendremos la entidad del empleador formando un elemento compuesto que será el que renderizemos en la vista.

```
<?php

namespace App\Controller;

use App\Entity\OfferMgr\Offers;
use App\Entity\UserMgr\Employees;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(Request $request)
    {
        $offers = $this->getDoctrine()
            ->getRepository(Offers::class)
            ->findAllActive();

        return $this->render('default/index.html.twig', array(
            'offers' => $offers,
        ));
    }
}
```

## 7.1.2 Vista

La página principal es la página por defecto `index.html.twig` alojada dentro del directorio `default`.

```
{% extends 'base.html.twig' %}
{% block navbar %}
    {% include 'navbar/public.html.twig' %}
{% extends 'base.html.twig' %}

{% block body %}

    <div class="container-fluid alert alert-dark alert-dismissible fade show
pt-0 pb-2" role="alert" >
        <div class="row">
            <div class="col-10 mx-auto">
                <h2 class="alert-heading display-2">Bolsa de empleo</h2>
                <hr>
                <p class="mb-0 lead">Servicio gratuito cuya finalidad es
facilitar el acceso al mercado laboral de las personas que conforman la
Comunidad Cuatrovientos (alumnado, exalumnado...) dando respuesta a los
procesos de selección solicitados por las empresas y otras
organizaciones.</p>
            </div>
        </div>
        <button type="button" class="close" data-dismiss="alert" aria-
label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>

    <div class="container-fluid">
        <div class="row">
            <div class="col-10 mx-auto">
                <h6 class="lead">A continuación ofrecemos las ofertas de
empleo.</h6>
            </div>
        </div>
        <div class="row bg-white">
            <div class="col-8 mx-auto ">
                {% for offer in offers %}
                    <div class="media py-2">
                        
                        <div class="media-body">
                            <div class="mx-4 d-flex justify-content-between ">
                                <h4 class="p-0"> {{ offer.getVPosition() }}
                                <small ><i > {{ offer.getDActivationDate()
| date('d-m-Y') }}</i></small>
                            </div>
                            <a href="{{ path('offers_show', {'id':
offer.getId()}) }}" class="btn btn-outline-primary">More details...</a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

```
offer.getLtextDescription() }} </p>
    </div>
  </div>
</div>
<hr/>
{% endfor %}
</div>
</div>
{% endblock %}
```

## 8 Gestor de ofertas de empleo

Para el desarrollo de la funcionalidad Gestor de ofertas de empleo vamos a establecer un espacio de nombres, `offerMgr`. En este espacio de nombres la entidad principal es `Ofertas` y su relación con la entidad de `Empleadores`.

Las ofertas son publicadas por los empleadores de las empresas. Un empleador puede publicar múltiples ofertas. Por lo que la relación es *Many to one* en `Ofertas` y *One to many* en `Empleadores`.

Dentro de la entidad `Ofertas` establecemos el siguiente código responsable de realizar la relación.

```
/**
 * @ORM\ManyToOne(targetEntity="App\Entity\OfferMgr\Employeers",
inversedBy="offers",cascade={"persist"})
 * @ORM\JoinColumn(name="employee_id", referencedColumnName="id")
 */
private $employeeer;
```

Dentro de la entidad de empleadores establecemos la relación inversa.

```
/**
 * @ORM\OneToMany(targetEntity="App\Entity\OfferMgr\Offers",
mappedBy="employeeer")
 */
private $offers;

public function __construct()
{
    $this->offers = new ArrayCollection();
}
```

Haciendo uso del asistente vamos a crear un primer prototipo de lo que serían las acciones CRUD sobre la tabla de ofertas.

```
C:\> php bin/console doctrine:generate:crud
```

## 8.1 Publicar una oferta

### Usuario empleador

A partir del asistente CRUD disponemos de la mayor parte de la lógica de esta funcionalidad.

La actualización para implementar en el código autogenerado es que estas funcionalidades deben de ser personalizadas para el usuario empleador que ha iniciado sesión. A la hora de realizar la inserción en base de datos del usuario creador (campo de auditoría de la tabla de base de datos), debemos establecer este valor con el usuario que ha iniciado sesión.

## 8.2 Visualizar Mis Ofertas

### Usuario empleador

Lo mismo pasa al realizar la obtención de todas las ofertas, en el repositorio de ofertas debemos limitarlo a *todas las ofertas de un usuario determinado*.

```
/**
 * @Route("offers")
 */
class OffersController extends Controller
{
    /**
     * Lists all offer entities from Logged in Employee
     * @Route("/", name="offers_index")
     * @Method("GET")
     */
    public function indexAction(Request $request)
    {
        $em = $this->getDoctrine()->getManager();
        // Get all the offers published by the logged employer
        $loggedin_username = $request->getSession()-
>get(Security::LAST_USERNAME);
        $offers = $em->getRepository('App:OfferMgr\Offers')-
>findOffersFromEmployee($loggedin_username);

        return $this->render('offermgr/offers/index.html.twig', array(
            'offers' => $offers,
        ));
    }
}
```

## 9 Gestor de CVs

Se considera que un CV puede estar compuesto por varias experiencias laborales previas, diferentes titulaciones de estudios y otros conocimientos o idiomas. El CV es un conjunto de todos estos conceptos. Para aprovechar las funcionalidades de Symfony realizamos el CRUD de las tablas que conforman el CV; *studies*, *work\_experiences*, *languages* y *other\_knowledges*.

Para completar un CV debemos indicar un flujo de pantallas que nos permita completar esta tabla mediante la introducción recursiva de registros en las tablas de *studies*, *work\_experiences*, *languages* y *other\_knowledges*. Esta lógica debemos implementarla en el controlador.

Antes de poder ejecutar el asistente CRUD es necesario instalar el bundle de formularios de Symfony para la versión 4.4. En las aplicaciones que utilizan Symfony Flex, ejecutamos este comando para instalar la función de formulario antes de utilizarlo:

```
composer require symfony/form
```

Una vez realizada la instalación se genera una carpeta Forms en nuestro proyecto y podemos generar los formularios a partir de nuestras entidades

```
php bin/console make:crud Offers
```



## 9.1 Controladores de CV

### 9.1.1 Creación de un CV

#### Usuario exalumno

##### CVCreatorController

En la lógica de creación de un controlador debemos de tener en cuenta que el CV no es más que un contenedor, donde se agrupan diferentes registros, ya sean de experiencias laborales, estudios, idiomas u otros conocimientos. Además, el número de cada uno de estos elementos es indefinido y diferente en cada uno de los CVs.

Es por eso que debemos hacer uso de la recursividad a la hora de establecer un asistente de creación de CVs que contemple la creación de múltiples componentes asociados dando opción al usuario de registrar múltiples experiencias laborales, estudios, etc.

## Recursividad

A continuación, la lógica implementada para realizar una llamada recursiva al formulario de creación de múltiples experiencias laborales.

```
/**
 * @Route("/new_workexperience", name="workexperience_task_add")
 */
public function addWorkExperienceAction(Request $request, Session $session){
    //Check if there is any existing CV in session
    $this->checkIfCVinSessionExists($session);

    $workExperience = new WorkExperiences();
    $form = $this->createForm('App\Form\CvMgr\WorkExperiencesType',
    $workExperience);
    $form->handleRequest($request);

    if ($form->isSubmitted() ) {
        // 1. DateTime parse
        $startDate = $workExperience->getDStartDate();
        $workExperience->setDStartDate(new \DateTime($startDate));
        $endDate = $workExperience->getDEndDate();
        $workExperience->setDEndDate(new \DateTime($endDate));
        // 2. Set the relation to the CV
        $workExperience->setCv($this->actual_user_cv);
        // 3. Update CV in session
        $this->actual_user_cv = $session->get('actual_user_cv');
        $this->actual_user_cv->addWorkexperience($workExperience);
        $session->set('actual_user_cv', $this->actual_user_cv);

        // Depending on the request the user might want to enter another work-
        experience (true)
        // or continue to insert studies (false)
        $nextAction = $form->get('saveAndAdd')->isClicked()
            ? 'task_new'
            : 'task_success';

        if( $nextAction == 'task_success' ){
            // Next, the user adds his studies
            return $this->redirectToRoute('studies_task_add');
        }
        else{
            // Clears previous form
            unset($workExperience);
            unset($form);
            // recursive call to addWorkExperienceAction
            return $this->redirectToRoute('workexperience_task_add');
        }
    }
    return $this->render('cvmgr/workexperiences/new.html.twig', array(
        'workExperience' => $workExperience,
        'form' => $form->createView(),
    ));
}
```

## Almacenamiento en base de datos

Una vez el usuario a finalizado de crear su CV, introduciremos los valores gestionados en la sesión a la base de datos.

```
/**
 * @Route("/new_otherknowledge", name="other_task_add")
 */
public function addOtherKnowledgeAction(Request $request , Session $session){
    //Check existing CV
    //Check if there is any existing CV in session
    $this->checkIfCVinSessionExists($session);

    $other_knowledge = new Otherknowledge();
    $form = $this->createForm( 'App\Form\CvMgr\OtherknowledgeType',
    $other_knowledge);
    $form->handleRequest($request);

    if ($form->isSubmitted() ) {
        // Establish relation to CV and user
        $other_knowledge->setCv($this->actual_user_cv);
        // 2. Set the relation to the CV
        $other_knowledge->setCv($this->actual_user_cv);
        // 3. Update CV in session
        $this->actual_user_cv = $session->get('actual_user_cv');
        $this->actual_user_cv->addOtherknowledges($other_knowledge);
        $session->set('actual_user_cv', $this->actual_user_cv);

        // Depending on the request the user might want enter another studies
        (true)
        // or continue to insert studies (false)
        $nextAction = $form->get('saveAndAdd')->isClicked()
            ? 'task_new'
            : 'task_success';

        if( $nextAction == 'task_success' ){
            // Persist values
            $this->getDoctrine()
                ->getRepository(CV::class)
                ->store_cv_values($session, $this->actual_user_cv);
            // CV creation finished.

            // Add user values stored in session to the actual user cv
            $id = $session->get('formerStudentID');
            // Get formed student values from database
            $formerStudent = $this->getDoctrine()->getManager()-
            >getRepository(FormerStudents::class)->find($id);

            // Redirectio to main page after Login validation
            return $this->get('security.authentication.guard_handler')
                ->authenticateUserAndHandleSuccess(
                    $formerStudent,
                    $request,
                    $this->get('app.security.login_form_authenticator'),
                    'main'
                );
        }
    }
}
```

```

    );
}
else{
    // Clears previous form
    unset($other_knowledge);
    unset($form);

    // recursive call to addstudiesAction
    return $this->redirectToRoute('other_task_add');
}
}

return $this->render('cvmgr/otherknowledge/new.html.twig', array(
    'otherknowledge' => $other_knowledge,
    'form' => $form->createView(),
));
}

```

La función encargada de realizar la persistencia se encuentra en la clase de repositorio asociada a la entidad CV.

```

// Method used to persist values in database
public function store_cv_values( Session $session, CV $actual_user_cv)
{
    // Manage fields according to what the database expects:
    $em = $this->getEntityManager();
    // Add formed student (from session) to actual user cv
    $formerStudent = $session->get('formerStudent');
    $actual_user_cv->setFormerstudent($formerStudent);
    foreach ($actual_user_cv->getWorkexperiences() as $aux_workexperience) {
        $em->persist($aux_workexperience);
    }
    foreach ($actual_user_cv->getStudies() as $aux_studies) {
        $em->persist($aux_studies);
    }
    foreach ($actual_user_cv->getLanguages() as $aux_language) {
        $em->persist($aux_language);
    }
    foreach ($actual_user_cv->getOtherknowledges() as $aux_otherknowledge) {
        $em->persist($aux_otherknowledge);
    }
    $em->flush();
    // Delete CV in session
    $session->set('actual_user_cv', null);
}

```

## 9.1.2 Edición de un CV

### Usuario exalumno

#### CVEditorController

A la hora de actualizar un CV se muestran los valores obtenidos desde la base de datos para el CV de exalumno en cuestión.

```
/**
 * Update values of an edited work experience from a selected
 * FormerStudent's CV
 * @Route("/{id_std}/edit/workexperience", name="stdnts_wrkexp_edit")
 */
public function stdntsWorkExpEditAction($id_std, Request $request, Session
$session)
{
    // Get the Working experience id from session
    $id_wrkexp = $request->query->get('id_wrkexp');
    // Get WorkExperience by Id
    $em = $this->getDoctrine()->getManager();
    $wrkexp_db = $em->getRepository('App:CvMgr\WorkExperiences')-
>find($id_wrkexp);
    if(!$wrkexp_db){
        throw $this->createNotFoundException(
            'No product found for id '.$id_wrkexp
        );
    }
    // Update values taken from the form
    $wrkexp_db->setVPosition($request->request->get('position'));
    $wrkexp_db->setVEmployer($request->request->get('employer'));
    $wrkexp_db->setVLocation($request->request->get('location'));
    $wrkexp_db->setLtextDuties($request->request->get('duties'));
    $wrkexp_db->setDStartDate(new \DateTime($request->request-
>get('startdate') ));
    $wrkexp_db->setDEndDate(new \DateTime($request->request->get('enddate')
));
    // Update database
    $em->merge($wrkexp_db);
    $em->flush();
    // Back to former students editing page
    return $this->redirect($this->generateUrl('formerstudents_edit',
array('id' => $id_std)));
}
```

Dentro de la funcionalidad de edición hay que tener en cuenta la posibilidad de que el usuario quiera añadir un nuevo elemento de los que conforman su CV, ya sea una nueva experiencia laboral, estudios, etc.

Cuando el usuario pulsa la opción de añadir un nuevo elemento dentro de la ventana de edición de CV se crea un nuevo registro en base de datos con los valores de **to complete**.

Con esto simplificamos la lógica ya que al volver a realizar la llamada a edición a través de `stdnts_wrkexp_edit` el usuario visualiza estos valores para su edición.

```
/**
 * New work experience to add to an editing FormedStudent's CV
 * @Route("/{id_std}/{id_cv}/new/workexperience", name="stdnts_wrkexp_new")
 */
public function stdntsWorkExpNewAction($id_std, $id_cv, Request $request) {
    $em = $this->getDoctrine()->getManager();

    // Look for the editing CV
    $actual_user_cv = $em->getRepository(CV::class)->find($id_cv);

    // Get id from last workexperience
    $highest_id = $this->getDoctrine()
        ->getRepository(WorkExperiences::class)
        ->getMaxId();

    // Add another workexperience at the CV
    $newWrkExperience = new WorkExperiences();
    $newWrkExperience->setId($highest_id + 1);
    $newWrkExperience->setVPosition("to complete");
    $newWrkExperience->setVEmployer("to complete");
    $newWrkExperience->setVLocation("to complete");
    $newWrkExperience->setLtextDuties("to complete");
    $newWrkExperience->setDStartDate(new \DateTime());
    $newWrkExperience->setDEndDate(new \DateTime());

    // Saving the referencing entity
    $em->persist($newWrkExperience);
    $em->flush();
    // Add entity to actual user CV
    $actual_user_cv->addWorkexperience($newWrkExperience);

    $cv = $em->getRepository(CV::class)->find($id_cv);
    $newWrkExperience->setCv($cv);
    $em->merge($newWrkExperience);
    $em->flush();

    // Back to former students editing page
    $formerStudent = $em->getRepository(FormerStudents::class)->find($id_std);
    return $this->redirect($this->generateUrl('formerstudents_edit',
    ['username' => $formerStudent->getUsername()]));
}
```

**NOTA:** La aplicación está diseñada para permitir que un usuario pueda tener múltiples CVs, pero esta funcionalidad no ha sido implementada. Es por eso por lo que de una lista de CVs en la edición se obtiene el CV de índice 0.

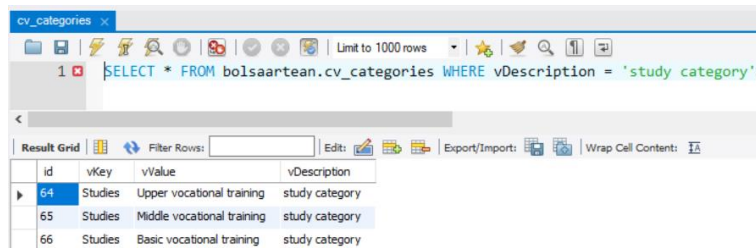


### 9.1.3 Búsqueda de un CV

#### Usuario empleador

CVSearchController

Para realizar la búsqueda de usuarios se establecen una serie de filtros. Los valores disponibles para estos filtros se almacenan en una tabla de base de datos, *categorías de CV*. Esta tabla se ha implementado con la idea de poder aglutinar todo tipo de variables de configuración, es por eso que su diseño sigue la pauta de Clave => Valor, típica de un array asociativo. Además, incluye un campo de descripción de manera que para obtener los valores de *categorías de estudio*, relacionadas con la entidad *Study* con la siguiente consulta.



The screenshot shows a database query tool interface. At the top, there's a toolbar with various icons and a 'Limit to 1000 rows' dropdown. Below the toolbar, a SQL query is entered: `SELECT * FROM bolsaartean.cv_categories WHERE vDescription = 'study category';`. The results are displayed in a table with the following data:

id	vKey	vValue	vDescription
64	Studies	Upper vocational training	study category
65	Studies	Middle vocational training	study category
66	Studies	Basic vocational training	study category

FIGURA 7.1. Consulta en table de categorías de CV.

A la hora de hacer uso de esta tabla se genera la respectiva entidad y su repositorio para obtener los valores.

```
use Doctrine\ORM\EntityRepository;

class CVCategoriesRepository extends EntityRepository {

    function getWorExperiencesPositions(){
        return $this->getEntityManager()->createQuery(
            "SELECT e.value FROM App:CvMgr\CVCategories e WHERE e.key='Work
Experience' AND e.description='position'")->getResult();
    }
    function getStudiesCategories(){
        return $this->getEntityManager()->createQuery(
            "SELECT e.value FROM App:CvMgr\CVCategories e WHERE e.key='Studies' AND
e.description='study category'")->getResult();
    }
}
```

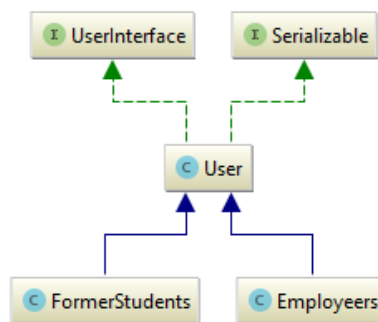


## 10 Gestión de usuarios

Para el desarrollo de las funcionalidades relacionadas con el control de usuarios vamos a establecer un espacio de nombres, UserMgr. Dentro de este espacio de nombres situamos las tablas de Usuario, Empleador y Exalumno siguiendo las especificaciones del apartado de diseño.

### 10.1 Jerarquía de las entidades

El diagrama de clases queda de la siguiente manera:



El mapeo relacional de objetos (ORM) de Doctrine se referencia mediante anotaciones.

Para utilizar la clase User como gestora de la seguridad se establece la implementación de UserInterface y Serializable.

La estructuración de la herencia mediante doctrine se realiza con las siguientes anotaciones:

```

/**
 * @ORM\Entity
 * @ORM\Table(name="user_mgr_users")
 * @ORM\InheritanceType("JOINED")
 * @ORM\DiscriminatorColumn(name="roles", type="string")
 * @ORM\DiscriminatorMap({
 *     "formerstudent" = "App\Entity\UserMgr\FormerStudents",
 *     "employeeer" = "App\Entity\UserMgr\Employeers",
 *     "user" = "App\Entity\UserMgr\User"
 * })
 */

```

## Modificaciones 4.4

```
use Symfony\Bridge\Doctrine\Security\User\UserLoaderInterface;
```

```
composer require symfony/validator doctrine/annotations
```

### 10.1.1 Entidad base: Usuario

La clase base User dispone de las siguientes propiedades.

#### Usuario

```
/**
 * @ORM\Column(type="integer")
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="AUTO")
 */
private $id;
/**
 * @ORM\Column(type="string", length=25, unique=true)
 */
private $username;
/**
 * @ORM\Column(type="string", length=64)
 */
private $password;
/**
 * @ORM\Column(type="string", length=254, unique=true)
 */
private $email;
private $roles=[];
```

Como se puede apreciar existe un array de roles de usuario el cuál no es trasladado a base de datos. En base de datos este campo se creará a partir del DiscriminatorMap establecido en la herencia de ORM.

Se crean los getter/setter de las propiedades y se implementan las funciones de Serialización utilizadas por el formulario de registro.

```
public function serialize() {
    return serialize([
        $this->id,
        $this->username,
        $this->email,
        $this->password,
        $this->roles
    ]);
}
```

```
public function unserialize($serialized) {
    list (
        $this->id,
        $this->username,
        $this->email,
        $this->password,
        $this->roles
    ) = unserialize($serialized, ['allowed_classes' => FALSE]);
}
```

## Entidades de segundo nivel

Los actores que participan en este apartado de la aplicación son dos, empleadores y exalumnos.

Estas clases contiene un constructor encargado de llamar al constructor de la clase base.

```
public function __construct() {
    parent::__construct();
}
```

## Exalumno

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

/**
 * @var string
 *
 * @ORM\Column(name="v_NIF", type="string", length=45)
 */
private $vNIF;

/**
 * @var string
 *
 * @ORM\Column(name="v_Name", type="string", length=45)
 */
private $vName;

/**
 * @var string
 *
 * @ORM\Column(name="v_Surnames", type="string", length=45)
 */
private $vSurnames;

/**
 * @var string
```

```

*
* @ORM\Column(name="v_Address", type="string", length=45)
*/
private $vAddress;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="d_Birth_Date", type="date")
 */
private $dBirthDate;

/**
 * @var bool
 *
 * @ORM\Column(name="b_Vehicle", type="boolean")
 */
private $bVehicle;

/**
 * @var string
 *
 * @ORM\Column(name="creation_user", type="string", length=45)
 */
private $creationUser;

/**
 * @var string
 *
 * @ORM\Column(name="modification_user", type="string", length=45)
 */
private $modificationUser;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="creation_date", type="date")
 */
private $creationDate;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="modification_date", type="datetime")
 */
private $modificationDate;

```

## Empleador

```

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

```

```
/**
 * @var string
 *
 * @ORM\Column(name="v_CIF", type="string", length=45)
 */
private $vCIF;

/**
 * @var string
 *
 * @ORM\Column(name="v_Name", type="string", length=255)
 */
private $vName;

/**
 * @var string
 *
 * @ORM\Column(name="v_Logo", type="string", length=255)
 */
private $vLogo;

/**
 * @var string
 *
 * @ORM\Column(name="v_Description", type="string", length=255)
 */
private $vDescription;

/**
 * @var string
 *
 * @ORM\Column(name="v_Contact_Name", type="string", length=255)
 */
private $vContactName;

/**
 * @var string
 *
 * @ORM\Column(name="v_Contact_Phone", type="string", length=255)
 */
private $vContactPhone;

/**
 * @var string
 *
 * @ORM\Column(name="v_Contact_Mail", type="string", length=255)
 */
private $vContactMail;

/**
 * @var string
 *
 * @ORM\Column(name="v_Location", type="string", length=255)
 */
private $vLocation;
```

```
/**
 * @var int
 *
 * @ORM\Column(name="n_Number_Of_Workers", type="integer", nullable=true)
 */
private $nNumberOfWorkers;

/**
 * @var string
 *
 * @ORM\Column(name="creation_user", type="string", length=45)
 */
private $creationUser;

/**
 * @var string
 *
 * @ORM\Column(name="modification_user", type="string", length=45)
 */
private $modificationUser;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="creation_date", type="date")
 */
private $creationDate;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="modification_date", type="datetime")
 */
private $modificationDate;
```

# 11Gestión de acceso

*Symfony* gestiona la seguridad mediante dos aspectos diferenciados: Autenticación y Autorización.

**Autenticación** verifica las credenciales de usuario. Su trabajo no es restringir el acceso, solo necesita saber quién eres tú. Podíamos pensar que se trata de un punto de seguridad en el acceso a un edificio en el que tienes que identificarte. Una vez verificado obtienes una tarjeta de acceso, o *token*. Este token puede ser utilizado para desbloquear puertas. Todo el mundo en el interior del edificio debe de disponer de un token, pero algunos tienen más acceso que otros.

**Autorización** es como el cerrojo que hay en cada puerta, impide que los usuarios puedan acceder a un determinado sitio. No necesita saber quién eres, solo si tienes permiso de acceso o no.

## 11.1 Formularios de acceso

### 11.1.1 Registro de usuario

Dentro del controlador de usuario se gestiona el registro de exalumnos y empleadores. Una vez introducidos los valores de acceso común a la web cada rol de usuarios sigue un asistente diferente.

```
/**
 * @Route("/register", name="user_register")
 */
public function registerAction(Request $request, UserPasswordEncoderInterface
$encoder) {
    $user_form = new UserRegistrationForm();
    $form = $this->createForm(UserRegistrationForm::class);
    $form->handleRequest($request);
    if ($form->isValid()) {
        // Values of the User object shared by FormerStudents and Employeers
        $user_form = $form->getData();
        // Save access values in session
        $session = $request->getSession();
        $session->set('username', $user_form->getUsername());
        $session->set('email', $user_form->getEmail());
        $plainPassword = $user_form->getPassword();
        $encoded = $encoder->encodePassword($user_form, $plainPassword);
        $session->set('password', $encoded);
        // Check user type
        if (in_array("ROLE_USER_FORMER_STUDENT", $user_form->getRoles())) {
            // Show cv creation form
            return $this->redirectToRoute('formerstudents_new', ['request' =>
$request]);
        }
        else {
            if (in_array("ROLE_USER_EMPLOYEEER", $user_form->getRoles())) {
                // Show employee creation form
                return $this->redirectToRoute('employeers_new', ['request' =>
$request]);
            }
        }
        return $this->get('security.authentication.guard_handler')
->authenticateUserAndHandleSuccess(
            $user_form,
            $request,
            $this->get('app.security.login_form_authenticator'),
            'main'
        );
    }
    return $this->render('usermgr/register.html.twig', [
        'form' => $form->createView(),
    ]);
}
```



## 11.1.2 Acceso de usuario

La lógica del controlador de usuario se encuentra en SecurityController.

```
/**
 * @Route("/register", name="user_register")
 */
public function registerAction(Request $request, UserPasswordEncoderInterface
$encoder) {
    $user_form = new UserRegistrationForm();
    $form = $this->createForm(UserRegistrationForm::class);

    $form->handleRequest($request);
    if ($form->isValid()) {
        // Values of the User object shared by FormerStudents and Employeers
        $user_form = $form->getData();

        // Save access values in session
        $session = $request->getSession();
        $session->set('username', $user_form->getUsername());
        $session->set('email', $user_form->getEmail());

        $plainPassword = $user_form->getPassword();
        $encoded = $encoder->encodePassword($user_form, $plainPassword);
        $session->set('password', $encoded);

        // Check user type
        if (in_array("ROLE_USER_FORMER_STUDENT", $user_form->getRoles())) {
            // Show cv creation form
            return $this->redirectToRoute('formerstudents_new', ['request' =>
$request]);
        }
        else {
            if (in_array("ROLE_USER_EMPLOYEEER", $user_form->getRoles())) {
                // Show employee creation form
                return $this->redirectToRoute('employeers_new', ['request' =>
$request]);
            }
        }

        return $this->get('security.authentication.guard_handler')
->authenticateUserAndHandleSuccess(
            $user_form,
            $request,
            $this->get('app.security.login_form_authenticator'),
            'main'
        );
    }
    return $this->render('usermgr/register.html.twig', [
        'form' => $form->createView(),
    ]);
}
```

Desde el formulario de registro se delega la responsabilidad de comprobar la validación de usuario a Security\LoginFormAuthenticator. Además, hay que configurar los parámetros de Autenticación y Autorización en la configuración de security.yml.

```
security:
    # En el caso que los usuarios tengan la contraseña cifrada configuramos
    # los encoders para la entidad en la que estén los usuarios
    encoders:
        App\Entity\UserMgr\User:
            algorithm: bcrypt
            cost: 4
    role_hierarchy:
        ROLE_USER: [ROLE_USER_FORMER_STUDENTS, ROLE_USER_EMPLOYEERS]
        ROLE_ADMIN: [ROLE_ARTEAN]

    # En providers le indicamos que los usuarios van a salir de la base de
    # datos y el username será email
    # https://symfony.com/doc/current/security.html#b-configuring-how-users-
    # are-loaded
    providers:
        our_db_provider:
            entity:
                class: App\UserMgr\User
                property: email
    firewalls:
        # disables authentication for assets and the profiler, adapt it
        # according to your needs
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: ~
            logout: ~

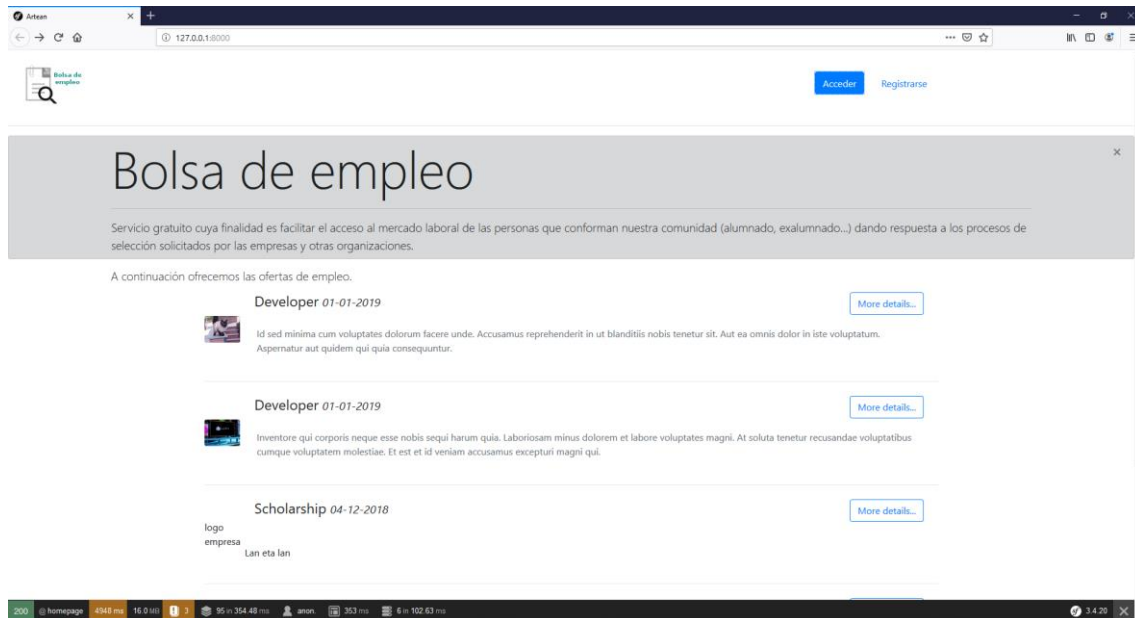
            guard:
                authenticators:
                    - App\Security\LoginFormAuthenticator
    access_control:
        - { path: ^/login, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin, role: [ROLE_ARTEAN] }
        - { path: ^/offers/(.+), role: [IS_AUTHENTICATED_FULLY] }
        - { path: ^/cv/new, role: [IS_AUTHENTICATED_ANONYMOUSLY] }
        - { path: ^/cv/search, role: [ROLE_USER_EMPLOYEERS] }
```

# Manual de la aplicación.

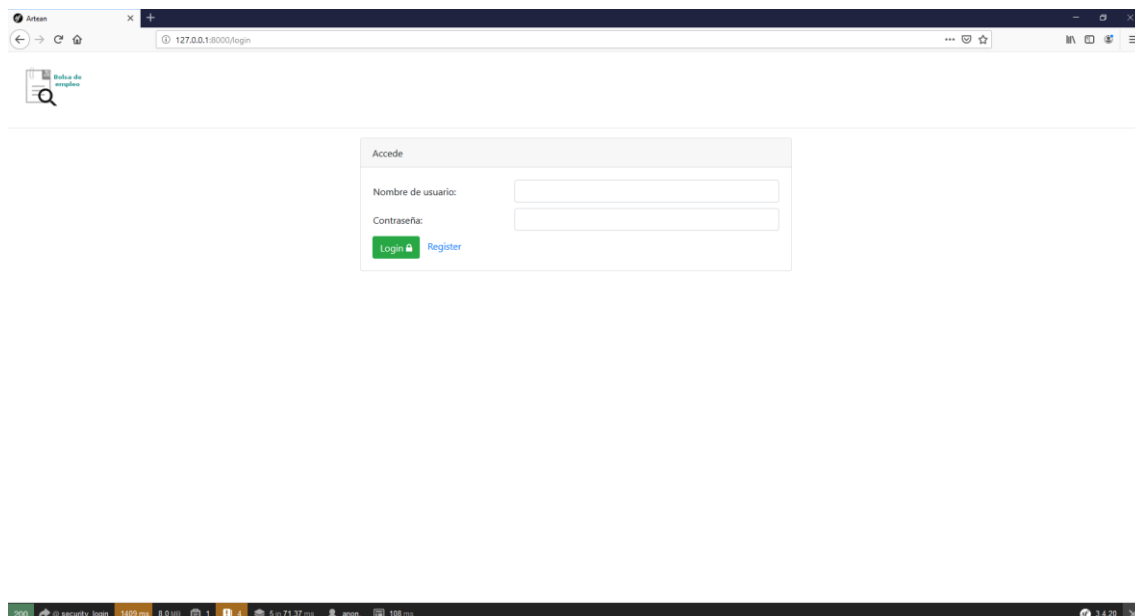
Resultado del primer prototipo.

## 12 Acceso a la web

En la primera pantalla lo que aparecen son las ofertas publicadas en la web, ordenadas por fecha.



Para poder acceder a una vista más detallada de las ofertas debemos registrarnos en la plataforma.



Desde la opción de registro vamos a tener que seleccionar entre los dos roles gestionados, exalumno y empresa.



### Access information

User name:

Email:

Password:

Repeat password:

User type: ☒ Former Student ☐ Employeeer

[register](#)

El primer formulario de información de acceso es común para ambos roles.

Generación de un Exalumno:

### Access information

• El correo electrónico de usuario ya está siendo usado.

User name:

Email:

Password:

Repeat password:

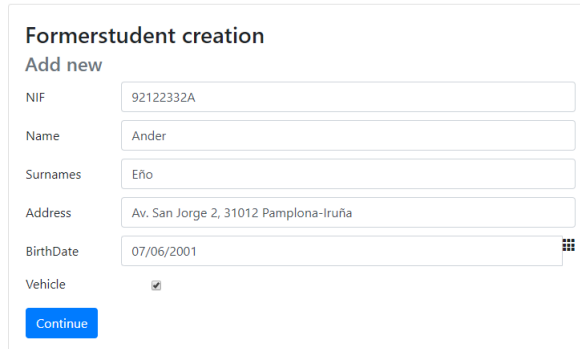
User type: ☒ Former Student ☐ Employeeer

[Register](#)

## 13 Exalumno

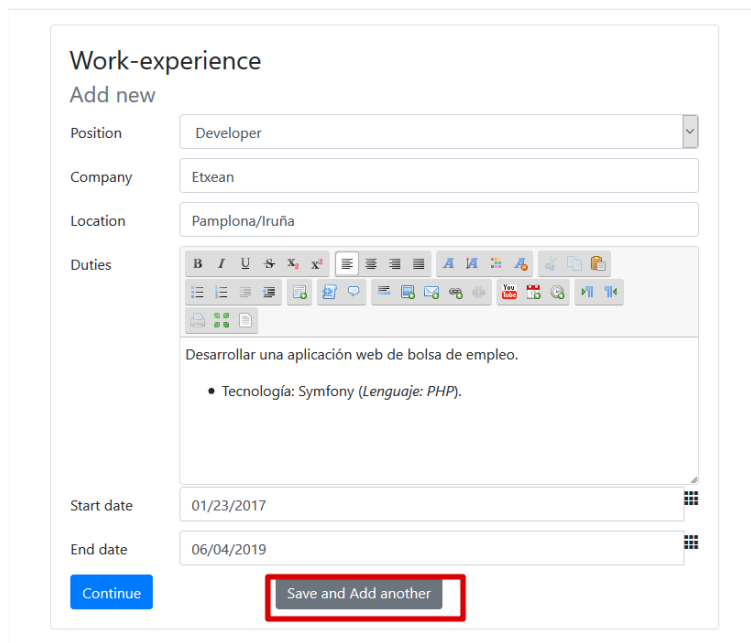
### 13.1 Registro de CV

Se registra en la aplicación introduciendo los datos de registro y los datos personales globales para todos los tipos de usuarios.



A continuación, se lanza el asistente de creación de CV. En este asistente se obliga a meter por lo menos un registro de cada uno de los componentes del CV, experiencia laboral, estudios, idiomas y otros intereses. El asistente permite introducir más de un campo de manera recursiva para cada uno de los componentes del CV.

Agregar una experiencia de trabajo:




## 13.2 Barra de herramientas

La barra de herramientas de Exalumno muestra las diferentes posibilidades que puede realizar un exalumno; Gestionar su CV y Controlar sus candidaturas.



## 13.3 Mi CV

Desde la pantalla de visualización de CV se puede visualizar el aspecto y pasar a la ventana de edición.


**Ander Eño**

92122332A 2001-07-06

Av, San Jorge 2, 31012 Pamplona-Iruña

Vehicle: Yes

edit

Work experiences

2019-01-23 - 2018-06-04 **Developer** Etxean Iruña

Desarrollo de una web de bolsa de empleo.

Studies

2016-09-18 - 2018-03-30 **Upper vocational training** Developer FP Cuatrovientos

Programación de una web para bolsa de empleo

Languages

**English** B1

Realización de Formación en Centro de Trabajo en Dublin, con la beca Erasmus.

Other knowledges

**Tocar la guitarra** No

Para mí la **guitarra** entra dentro de mis aficiones, lo tomo como un hobby

## 13.4 Ventana de edición

Desde la ventana de edición no solo somos capaces de modificar los registros de cada uno de los componentes del CV, también se puede crear nuevos registros.

Para que los cambios surjan efecto debe de pulsarse el botón de continuar.

## Former student edition

NIF

Name

Surnames

Address

BirthDate  ▾  ▾  ▾

Vehicle ☐

[Continue](#)

---

**Work experiences** [New](#)

Position:	<input type="text" value="Developer"/>
Employer	<input type="text" value="Etxean"/>
Location	<input type="text" value="Iruña"/>
Duties	<div style="border: 1px solid #ccc; padding: 5px;"> </div> <div style="border: 1px solid #ccc; padding: 5px; min-height: 100px;">           Desarrollo de una web de bolsa de empleo.         </div>
StartDate	<input type="text" value="01/23/2019"/>
EndDate	<input type="text" value="06/04/2018"/>

## 13.5 Aplicar a una oferta

Desde la pantalla principal un exalumno puede aplicar a una oferta al acceder a sus detalles.

Offer	
Offer code	ACTIVE
Offer type	full-time
Activation date	2018-12-04
Due date	2018-12-06
Position	Scholarship
Description	Lan eta lan
Duties	Gutxi gastau
Experience requirements	Ezer ez
Salary	122

Apply to offer

Una vez que se ha aplicado a una oferta la visualización de su estado se ve desde *Mis Candidaturas*.

## 13.6 Mis Candidaturas

Esta pantalla refleja las ofertas en las que está inscrito un exalumno. Desde aquí puede hacer el seguimiento de su estado.



**Job applications**

Application state
PENDING
Offer code
ACTIVE
Position
Scholarship

*i* show Offer

# 14 Empresa

## 14.1 Registro

La empresa pasa por un asistente parecido al de el exalumno al registrarse en el sistema.

### Employee creation

#### Add new

CIF:

Nombre:

Logo:

Descripción:

Persona de contacto  
(Nombre):

Telefono de contacto::

Email de contacto:

Localización:

Número de trabajadores:

Continue

## 14.2 Barra de herramientas

La barra de herramientas de la empresa muestra la opción de publicar oferta, gestionar los inscritos en las ofertas y el buscador de CVs.



## 14.3 Publicar ofertas

En lugar de publicar su CV la empresa lanza la pantalla de publicación de ofertas desde la barra de herramientas.

# Offer creation

## Add new

Offer code: ACTIVE

Working time: part-time

Activation date: 12/04/2018

Due date: 06/26/2019

Position: Developer

Description:

B I U X X'

Desarrollar una bolsa de empleo

Funciones:

B I U X X'

Programación en PHP

Experience requirements:

B I U X X'

CFPS Desarrollo Web

Salary: 1224€

Location: Rotxa

[Continue](#)

## 14.4 Detalles de oferta

Desde la pantalla de visualización de la oferta se pueden ver los candidatos, editar la oferta o borrar la oferta.

Offer	
Offer code	ACTIVE
Offer type	part-time
Activation date	2018-12-04
Due date	2019-06-26
Position	Developer
Description	Desarrollar una bolsa de empleo
Duties	Programación en PHP
Experience requirements	CFPS Desarrollo Web
Salary	1224€

[Candidates](#)
[Edit](#)
[Delete](#)

### 14.4.1 Candidatos

Desde la ventana de candidatos se visualizan los usuarios que se han registrado en la oferta. Desde aquí se puede cambiar el estado de cada uno de ellos.

**Job candidates**

Application state: PENDING

Candidate name: **Ander Eño**  
 Address: Av. San Jorge 2, 31012 Pamplona-Iruña  
 Birth date: 2001/07/06  
 Vehicle available

[i show CV](#)

## 14.5 Buscador de ofertas

El gran potencial de esta aplicación es el buscador de CVs, capaz de buscar una palabra en cualquiera de los campos registrados en el CV. Esto es debido a que todos los componentes del CV son registrados como tablas separadas y relacionadas. Cada uno de los campos de los diferentes componentes son columnas de las tablas de base de datos.

### Search filters

Key word:  [Search](#)

---

Work experiences:

- ☐ Scholarship
- ☐ System administrator
- ☐ Developer

---

Studies:

- ☐ FP
- ☐ Middle vocational training
- ☐ Basic vocational training

Name	Surnames	Address	Birthdate	Vehicle	
Ander	Eño	Av. San Jorge 2, 31012 Pamplona-Iruña	2001-07-06	Yes	<a href="#">show</a>