

# Documentación Técnica

---

Proyecto Fin de Ciclo

**María Frades Castro**

**16/04/2018**

## TABLA DE CONTENIDOS

1	Introducción .....	3
2	Arquitectura de la aplicación .....	4
2.1	Frontend de la aplicación.....	4
2.1.1	Tecnologías utilizadas en el frontend.....	4
2.1.2	Entorno de desarrollo .....	5
2.1.3	Referencia de clases del frontend .....	7
2.2	Backend de la aplicación .....	10
2.2.1	Tecnologías utilizadas en el backend .....	10
2.2.2	Entorno de desarrollo .....	10
2.2.3	Referencia de clases del backend .....	11
2.2.4	Diagrama de clases del backend.....	15
2.2.5	Referencia de ficheros de configuración del backend .....	17
3	Modelo de datos.....	18
3.1	Explicación de los datos utilizados .....	18
3.2	Diagrama Entidad – Relación .....	20
4	Entorno de ejecución del proyecto – guía de despliegue .....	21
4.1	Requisitos para la instalación .....	21
4.2	Procedimiento de instalación .....	21
4.2.1	Copia de los ficheros en el servidor .....	21
4.2.2	Configuración del servidor web.....	21
4.2.3	Carga de datos de prueba en la base de datos.....	22

# 1 INTRODUCCIÓN

El presente documento contiene la documentación técnica de la aplicación web FPal. Para la documentación sobre funcionalidad, pueden visitarse los manuales de usuario.

Está estructurado (aparte de esta introducción) en tres partes cuya finalidad conjunta es proporcionar una visión general de la aplicación de tal forma que cualquier persona con los conocimientos técnicos adecuados pueda continuar el desarrollo de la aplicación antes mencionada o llevar a cabo su instalación.

1. En primer lugar, se lleva a cabo una aproximación a la arquitectura de la aplicación, identificando sus componentes a más alto nivel (frontend y backend) con las funciones que lleva a cabo cada uno de ellos y las tecnologías que se han seleccionado para implementar cada componente. Podemos separar esta parte en sus dos elementos:
  - a. Frontend: sobre este primer componente, se hace un recorrido justificado por las tecnologías utilizadas en cada uno de los elementos, se detalla el entorno de desarrollo que se ha utilizado durante la fase de elaboración y se incluye una referencia con las clases del frontend, agrupándolas (tal como se recomienda dadas las características de Angular) en “clases que representan elementos de la vista” (lo que se conoce como componentes), “clases que representan elementos de negocio” (lo que se conoce como modelos y, por último, las “clases que acceden al servidor para el intercambio de datos” (lo que se conoce en el contexto de Angular como servicios).
  - b. Backend: pasaremos a la parte que se encarga de almacenar y proporcionar los datos que se muestran en la aplicación web. Estas son las funciones de las que se encarga el backend de la aplicación, a través de una API REST. Para este componente, al igual que el anterior, haremos un repaso de las tecnologías utilizadas y el entorno de desarrollo en el que se ha implementado el proyecto. Pasaremos luego a hacer un repaso de las clases que implementan la funcionalidad del backend y, por último, de algunos ficheros de configuración que se utilizan en este componente.
2. Una vez vista la arquitectura general de la aplicación y comprendida la interacción entre sus dos elementos fundamentales, pasamos a estudiar el **modelo de datos**, que nos proporcionará una visión general de cuáles son los “elementos de negocio” de la aplicación y cómo se han modelado
3. Por último, podremos comenzar la actualización o extensión de la aplicación, partiendo de la **guía de desarrollo**, que paso a paso muestra cómo instalar la aplicación y llevar a cabo la configuración necesaria en el servidor.

## 2 ARQUITECTURA DE LA APLICACIÓN

El diseño de la aplicación sigue el paradigma Model – View – Controller.. Con el fin de simplificar la lectura de esta documentación, trataremos de manera separada la Vista (que constituye el FrontEnd de la aplicación web) mientras que el Modelo y el Controlador se tratarán en el apartado de Backend.

El diagrama de la arquitectura, incluyendo las tecnologías utilizadas, es el siguiente:



### 2.1 Frontend de la aplicación

La función principal que tiene el frontend de FPal pasa por constituir el punto de acceso del usuario a los servicios que presta la aplicación. Por lo tanto, recoge las acciones del usuario traduciéndolas a peticiones hacia el backend y adapta la interfaz para representar los resultados obtenidos.

#### 2.1.1 Tecnologías utilizadas en el frontend

Como tecnología fundamental para el desarrollo del Frontend hemos seleccionado Angular. Se trata de un framework escrito en Typescript (un lenguaje de programación basado en JavaScript pero que presenta características distintas, como el hecho de que es fuertemente tipado).

Entre los beneficios de utilizar Angular, podemos destacar los siguientes:


- Proporciona una estructura muy limpia y estandarizada para las aplicaciones web que facilita su mantenimiento.
- Incluye gran cantidad de código reutilizable y que libera a los programadores de dedicar tiempo a código que no es propio de la aplicación (boilerplate).
- Promueve las pruebas integradas durante el desarrollo.
- Incluye no solo herramientas sino también patrones de diseño que favorecen la creación de proyectos de forma mantenible.

- Está escrito utilizando TypeScript, que es un superconjunto de JavaScript mejorado con características como el manejo de tipos estáticos, decoradores, interfaces...
- Proporciona soluciones para problemas frecuentes como validación en formularios que liberan a los desarrolladores de escribir código repetitivo
- Desacoplamiento: los componentes están desacoplados lo que permite que el código sea independiente del DOM, manteniendo la lógica totalmente separada de la presentación
- Proporciona un ecosistema de herramientas amplio que agiliza muchas tareas como el servicio de una aplicación en desarrollo o la compilación continua

El uso de Angular se ha complementado, como no puede ser de otra forma, con HTML (Lenguaje de Marcado de Hipertexto) y CSS (Hojas de Estilo en Cascada) así como JQuery (una librería de JavaScript que facilita el uso de este lenguaje a través de un framework estable y ligero) y BootStrap (una librería construida orientada al desarrollo web responsive que proporciona los estilos para distintos elementos de la aplicación).

### 2.1.2 Entorno de desarrollo

Para el desarrollo del frontend se ha utilizado NPM (Node Package Manager) para la instalación de la herramienta de línea de comandos de Angular (Angular CLI)



```
Símbolo del sistema
C:\Users\María>ng --version

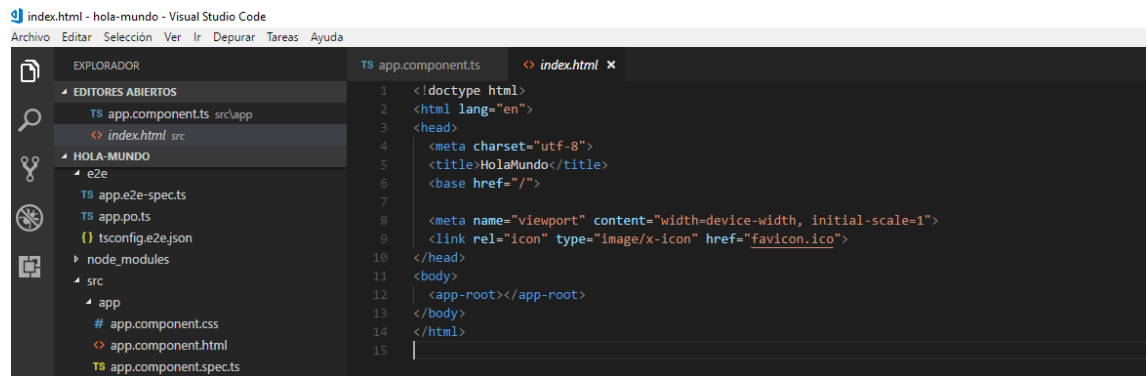
Angular CLI
Angular CLI: 1.7.4
Node: 9.1.0
OS: win32 x64
Angular:
...
C:\Users\María>
```

Angular CLI es un intérprete de comandos que permite llevar a cabo las tareas asociadas a las acciones de gestión de proyectos en Angular más frecuentes:

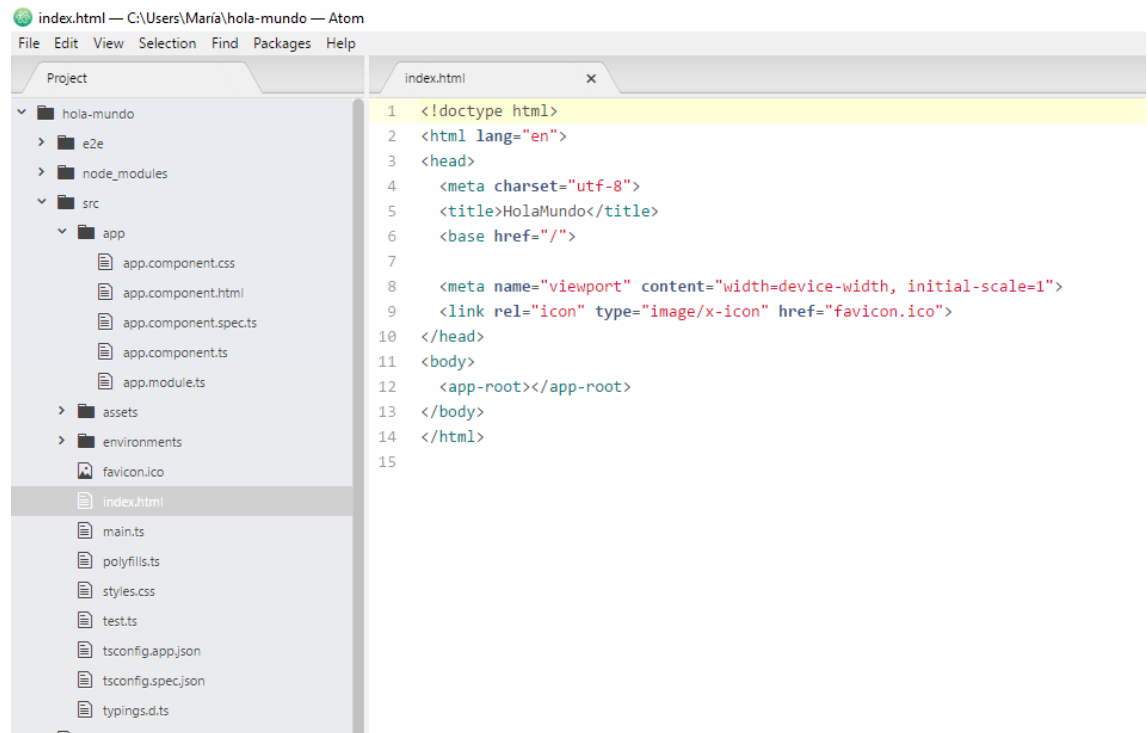
- Ng new: crea una nueva aplicación con una estructura que se ajusta a las guías de buenas prácticas de Angular y funciona “out of the box”
- Ng generate: genera componentes, rutas, servicios y tuberías además de los esqueletos de los tests para cada uno de ellos.
- Ng serve: lanza un servidor de pruebas local a efectos del desarrollo.

WebPack: es una herramienta de empaquetado que toma todo el código typescript, lo compila y combina el javascript con css, crea un bundle con él y lo minimiza para optimizarlo y finalmente lo despliega.

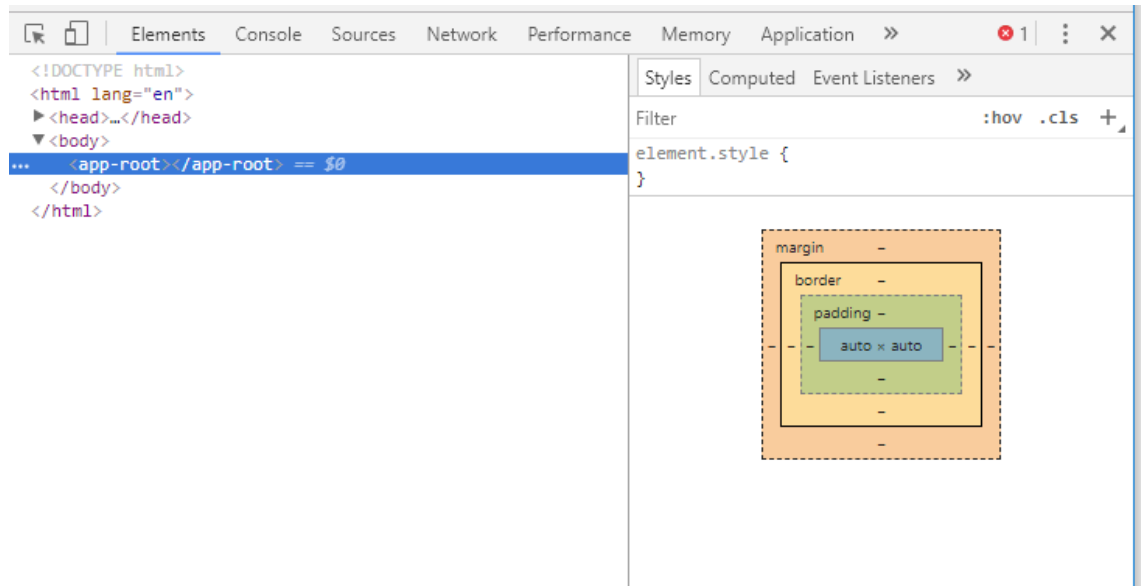
Como editor de código para esta parte de la aplicación se ha seleccionado Visual Studio Code.



Aunque en ocasiones se ha complementado con Atom:



También ha resultado de mucha utilidad el uso de las herramientas de desarrollador incluidas en Google Chrome, que permiten ver el código fuente (tanto HTML como CSS y JavaScript) que se ha servido en cada momento así como realizar cambios “en vivo”.



### 2.1.3 Referencia de clases del frontend

Podemos dividir, conforme a la clasificación que propone Angular, las clases que contituyen FPal en dos grandes grupos: las clases que “muestran algo”, que son Componentes y las clases que “representan algo”, que son los modelos.

Entre los componentes, se encuentran las siguientes clases:

- **Clase DashboardComponent:** se trata de la clase encargada de presentar los elementos que proporcionan al usuario los mensajes recibidos, en el caso de que sea un alumnos mostrará también su itinerario y, si no es alumno, mostrará el listado de usuarios de tipo alumno. Tiene acceso, proporcionado en el constructor, a los servicios de Routing (que le permite saber en qué parte de la aplicación se encuentra) y Login (que le permite saber el tipo de usuario que está conectado, así como su información asociada).
- **Clase CompetenciasComponent:** se trata de la clase que muestra el marcado relacionado con la gestión de las competencias (listar las competencias, borrarlas, acceder al detalle...) Utiliza la vista de tabla y los elementos de Angular que, en la plantilla, permiten definir criterios de ordenación en función del contenido.
- **Clase set-CompetenciaComponent:** es la clase que muestra y gestiona la información necesaria para definir una nueva Competencia a través del fronted de FPal o para mostrar los detalles de una competencia concreta. Utiliza algunas características de Angular que facilitan mucho el desarrollo, como la definición y validación de formularios mediante los métodos `createForm` y `validateForm`, que los construyen en base a las características de la lógica de negocio y con desacoplamiento de la presentación.
- **Clase CualificacionesComponent:** se trata de la clase que muestra el marcado relacionado con la gestión de las cualificaciones (listar las cualificaciones borrarlas, acceder al detalle...) Utiliza la vista de tabla y los

elementos de Angular que, en la plantilla, permiten definir criterios de ordenación en función del contenido.

- **Clase setCualificaciónComponent:** es la clase que muestra y gestiona la información necesaria para definir una nueva Cualificación Profesional a través del fronted de FPal o para mostrar los detalles de una cualificación concreta. Utiliza algunas características de Angular que facilitan mucho el desarrollo, como la definición y validación de formularios mediante los métodos `createForm` y `validateForm`, que los construyen en base a las características de la lógica de negocio y con desacoplamiento de la presentación.
- **Clase ItinerarioComponent:** comprueba si el usuario logado es un Alumno o un Profesor. Dependiendo del caso muestra los itinerarios formativos que tiene asociados. La plantilla HTML asociada permite además reordenar los itinerarios formativos y gestionarlos (añadirlos, eliminarlos y modificarlos)
- **Clase LoginComponent:** se trata del componente que muestra el formulario de login y se encarga de autenticar a los usuarios que quieren acceder a la aplicación.
- **Clase MessagesComponent:** permite manejar la bandeja de mensajes del usuario que está autenticado (para ello está suscrito al servicio de mensajería, `MessagesService`). Se permite al usuario ver los mensajes recibidos, marcarlos como leídos o borrarlos. La plantilla asociada utiliza las funciones condicionales y las ordenaciones de Angular para tratar de forma distinta a los enviados de los recibidos.
- **Clase newMessageComponent:** permite la redacción de un nuevo mensaje, llevando a cabo la validación del formulario a través del cual el usuario introduce los datos mediante las directivas de Angular.
- **Clase navBarComponent:** es el componente que muestra y gestiona la barra de navegación que puede utilizarse para acceder a las diferentes funcionalidades de la aplicación. Utiliza el sistema de permisos que gestiona el servicio de Login para determinar qué partes de la aplicación están disponibles en función de si el usuario es un Alumno o un Formador.
- **Clase UserComponent:** este componente muestra las funcionalidades asociadas a la creación de usuarios: más concretamente, muestra el formulario a través del que los usuarios se registran en la plataforma (que es el mismo a través del cual los formadores pueden editar los datos de un alumno).
- **Clase UsersComponent:** mediante este componente, se muestra al administrador y los formadores la lista de usuarios de la plataforma, permitiéndoles llevar a cabo funciones como la activación de usuarios o la definición de formadores.

Entre los modelos, se encuentran las siguientes clases:

- **Clase Realización:** modela una realización profesional. Como tal, tiene un título y un array de criterios de realización.



- Clase **Competencia**: modela una competencia profesional. Como tal, tiene un código, nivel, nombre, medios, productos, información y un array de realizaciones profesionales.
- Clase **Realización**: modela una realización profesional. Como tal, tiene un título y un array de criterios de realización.
- Clase **Cualificación**: modela una cualificación profesional. Como tal, tiene un código, nivel, nombre, familia profesional, descripción y entorno productivo (todos son cadenas menos el nivel) y un array de competencias profesionales.
- Clase **Itinerario**: modela un itinerario formativo. Cada itinerario formativo tiene un identificador, un número de identificación del usuario al que está asociado, la cualificación profesional a la que se asocia, el estado (si se ha superado o no), las fechas de comienzo y fin y el número de orden que ocupan dentro del plan formativo del usuario.
- Clase **Message**: modela un mensaje. Como tal, tiene un identificador C:\Users\María\Desktop\Proyectino\Desplegado\desplegado\fpal\src\app\models\user.tsr único de mensaje, el identificador del remitente y el receptor así como sus nombres de usuario, un asunto, un cuerpo, una fecha en la que se creó y si se quiere mostrar o no (porque se haya eliminado) así como si ya se ha leído o no por su destinatario.
- Clase **User**: modela un usuario de la plataforma. Para cada uno de ellos, tiene su identificación, su correo, password, el tipo de usuario, el nombre, el apellido, la foto, la formación, los intereses, las perspectivas y un flag que indica si el usuario está activo o no.

Por último, las clases que quedan son aquellas que proporcionan servicios a los demás elementos de la aplicación. Estas clases están marcadas como inyectable e implementan la interfaz CanActivate. A través de estas clases, además, se lleva a cabo el acceso a los servicios prestados por el backend, lo que favorece el desacoplamiento de ambos elementos de la aplicación. Las más destacables son las siguientes:

- Clase-servicio **Ensenanza**: se encarga del acceso al servidor en lo relativo a competencias y cualificaciones, posibilitando las acciones CRUD sobre estos elementos
- Clase-servicio **Itinerario**: se encarga del acceso al servidor en lo relativo a itinerarios formativos de un usuario, posibilitando las acciones CRUD sobre estos elementos
- Clase-servicio **Login**: se encarga de la autenticación de los usuarios y de permitir que los demás servicios accedan a la información sobre los permisos de los distintos usuarios de la plataforma. También se ocupa de la gestión de las sesiones, utilizando el token para comprobar si los usuarios están autenticados.
- Clase-servicio **Messages**: se encarga del acceso al servidor en lo relativo a mensajes, posibilitando las acciones CRUD sobre estos elementos

- Clase-servicio **Permisos:** decide, para cada ruta de petición (dirigida al servidor), los roles que debe tener el usuario para poder acceder a ella.
- Clase-servicio **User:** permite llevar a cabo las operaciones CRUD sobre los usuarios registrados en la plataforma.

## 2.2 Backend de la aplicación

La función que tiene el backend de FPal es proporcionar una API a través de la cual el frontend le envíe las peticiones (iniciadas en términos generales por acciones del usuario) y ofrecer respuesta a estas peticiones llevando a cabo el acceso a los almacenes de datos

### 2.2.1 Tecnologías utilizadas en el backend

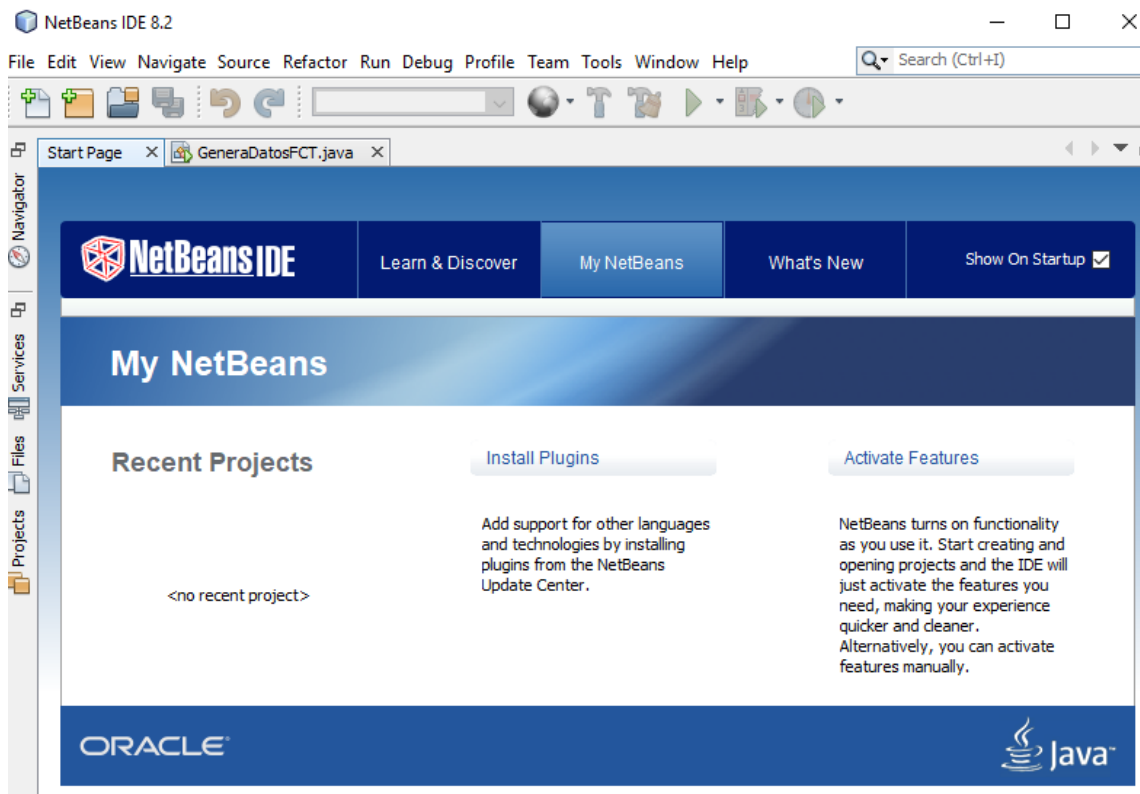
Como tecnología fundamental para el desarrollo del backend hemos seleccionado PHP (PHP:Hypertext Preprocessor). Se trata de un lenguaje de script del lado del servidor con soporte para orientación a objetos, de código abierto, gratuito y ampliamente utilizado.

Para el almacenamiento de datos y su gestión y manejo se ha seleccionado el sistema gestor de base de datos de MySQL. Se trata de un sistema gestor de bases de datos relacionales bien conocido, publicado bajo licencia pública general/licencia comercial por Oracle Corporation y es considerado el gestor de bases de datos de código abierto más popular del mundo, específicamente para entornos de desarrollo web.

En cuanto al envío de datos desde el backend al frontend, se lleva a cabo en formato JSON (Java Script Object Notation). Se trata de un formato de texto ligero para el intercambio de datos que constituye una alternativa más fácil de parsear y que produce un overhead de comunicaciones más reducido que alternativas como XML.

### 2.2.2 Entorno de desarrollo

Para el desarrollo del backend se ha utilizado NetBeans, dado que es el entorno de desarrollo utilizado en el módulo de Desarrollo Web en Entorno de Servidor, en el que se utiliza PHP y por ser ya conocido y adecuado para nuestros fines.



### 2.2.3 Referencia de clases del backend

En primer lugar analizaremos las clases del paquete **framework**:

- **Clase Controller:** representa el contexto de ejecución del backend. Almacena referencias estáticas al núcleo de la aplicación (clase core), al servicio de análisis de rutas (routing), a la base de datos (dataBase) y al array de errores que se hayan producido. En cuanto a sus métodos, sirven para inicializar los atributos y modificar el contenido del array de errores.
- **Clase Core:** clase que representa el núcleo de la aplicación, accediendo a la configuración almacenada en conf/core.json (que determina si estamos en producción o en desarrollo) y lanzando a ejecución las clases que se encargan de la lógica de negocio.
- **Clase Routing:** en esta clase se lleva a cabo el análisis de las URIs de las peticiones recibidas de forma que se determina la clase y el método que hay que ejecutar así como los parámetros que deben pasársele para la ejecución (en caso de que sean necesarios). Lee el fichero conf/routing.json que es donde se establecen las correspondencias entre rutas, métodos y clases a ejecutar.
- **Clase Database:** en esta clase se almacena la conexión a la base de datos (cuyos datos de configuración se establecen en el fichero conf/dataBase.json). Contiene métodos para ejecutar consultas, actualizaciones, salvar objetos a la base de datos, hacer logs de los accesos a la base de datos con error, así como el manejo de transacciones.

- **Clase Session:** se encarga del manejo de las sesiones, analizando a partir de un token que identifica al fichero de sesión si una sesión es válida o está caducada.
- **Clase Files:** Clase que sirve para manejar los ficheros y eliminarlos si es preciso eliminando el directorio que los contiene.
- **Clase dbObject:** se trata de una clase abstracta que hace de puente entre los datos que vienen del cliente (que parsea mediante el método `getFromPost`) y los datos que vienen de la base de datos, que obtiene mediante el método `getFromDB`. Se encarga de almacenar, validar, modificar y eliminar los objetos en la base de datos. Todos los objetos que se almacenan en la base de datos heredan de esta clase.

El siguiente paquete cuyo contenido veremos es el denominado **dbObjects**. Este paquete almacena todas las clases que heredan de la clase `dbObject` y que se utilizan para representar los objetos de negocio en nuestra aplicación:

- **Clase dbCompetencia:** La clase `dbcompetencia` hereda de la clase `dbObject`. Particulariza la función `format` de dicha clase abstracta con el formato de los campos propios de una competencia profesional representada en la base de datos. También proporciona atributos para almacenar en memoria los datos propios de la competencia profesional, así como la información sobre la tabla en la que se almacenan las competencias profesionales dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.
- **Clase dbCualificacion:** La clase `dbCualificacion` hereda de la clase `dbObject`. Particulariza la función `format` de dicha clase abstracta con el formato de los campos propios de una cualificación profesional representada en la base de datos. También proporciona atributos para almacenar en memoria los datos propios de la cualificación profesional, así como la información sobre la tabla en la que se almacenan las cualificaciones profesionales dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.
- **Clase dbItinerario:** La clase `dbItinerario` hereda de la clase `dbObject`. Particulariza la función `format` de dicha clase abstracta con el formato de los campos propios de un itinerario profesional representado en la base de datos. También proporciona atributos para almacenar en memoria los datos propios del itinerario profesional, así como la información sobre la tabla en la que se almacenan los itinerarios profesionales dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.
- **Clase dbMessage:** La clase `dbMessage` hereda de la clase `dbObject`. Particulariza la función `format` de dicha clase abstracta con el formato de los campos propios de un mensaje representado en la base de datos. También proporciona atributos para almacenar en memoria los datos propios del mensaje, así como la información sobre la tabla en la que se almacenan los mensajes dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.

- **Clase dbUser:** La clase dbUser hereda de la clase dbObject. Particulariza la función format de dicha clase abstracta con el formato de los campos propios de un usuario representado en la base de datos. También proporciona atributos para almacenar en memoria los datos propios del usuario, así como la información sobre la tabla en la que se almacenan los usuarios dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.
- **Clase dbUserType:** La clase dbUserType hereda de la clase dbObject. Particulariza la función format de dicha clase abstracta con el formato de los campos propios de un tipo de usuario representado en la base de datos. También proporciona atributos para almacenar en memoria los datos propios del tipo de usuario, así como la información sobre la tabla en la que se almacenan los tipos de usuarios dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.
- **Clase dbCualifCompet:** La clase dbCualifCompet hereda de la clase dbObject. Particulariza la función format de dicha clase abstracta con el formato de los campos propios de un tipo de una relación entre cualificaciones y competencias representada en la base de datos. También proporciona atributos para almacenar en memoria los datos propios del tipo de dicha relación, así como la información sobre la tabla en la que se almacenan estas relaciones dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos.

Por último, pasamos a estudiar el paquete Controllers. Se trata de subclases de la clase Controller, que como vimos con anterioridad, proporciona acceso al entorno de ejecución de la aplicación. Cada una de las subclases se instancia según el tipo de objeto de negocio que vaya a tratarse a través de la aplicación, implementando la lógica de negocio en cada caso concreto.

- **Clase competencias:** La clase competencias hereda de la clase Controller y se instancia cuando se van a llevar a cabo acciones sobre las competencias profesionales. de la clase dbObject. Particulariza la función format de dicha clase abstracta con el formato de los campos propios de un tipo de una relación entre cualificaciones y competencias representada en la base de datos. También proporciona atributos para almacenar en memoria los datos propios del tipo de dicha relación, así como la información sobre la tabla en la que se almacenan estas relaciones dentro de la base de datos y el nombre del atributo que actúa como identificador de estos elementos. Lleva a cabo, a través de la referencia a la base de datos y al array de errores las transacciones relacionadas con las competencias, la transformación de los datos de las competencias entre json e instancias de la clase dbCompetencia cuando resulta necesario.
- **Clase cualificaciones:** La clase cualificaciones hereda de la clase Controller y se instancia cuando se van a llevar a cabo acciones sobre las cualificaciones profesionales. Lleva a cabo, a través de la referencia a la base de datos y al

array de errores las transacciones relacionadas con las competencias, la transformación de los datos de las competencias entre json e instancias de la clase dbCompetencia cuando resulta necesario. Almacena también información sobre las familias profesionales.

- **Clase itinerarios:** La clase itinerarios hereda de la clase Controller y se instancia cuando se van a llevar a cabo acciones sobre las cualificaciones profesionales asociadas al itinerario formativo de un usuario (por lo que almacena un apuntador al usuario al que se refiere). Lleva a cabo, a través de la referencia a la base de datos y al array de errores las transacciones relacionadas con los itinerarios y sus elementos (cualificaciones profesionales), la transformación de los datos entre json e instancias de la clase dbCompetencia cuando resulta necesario.
- **Clase login:** La clase login hereda de la clase Controller y se instancia cuando se van a llevar a cabo acciones relacionadas con el control de acceso de los usuarios (autorización y autenticación). Para ello, analiza la sesión actual determinando si el usuario está o no logado o comprobando los datos de acceso del usuario que quiere hacer login.
- **Clase messages:** La clase messages hereda de la clase Controller y se instancia cuando se van a llevar a cabo acciones relacionadas con el control de mensajes (envío, recuperación, borrado). Para ello, instancia la clase dbMessage y accede a la base de datos para actualizar sus atributos o bien actualiza la base de datos, según corresponda.
- **Clase user:** La clase user hereda de la clase Controller y se instancia cuando se van a llevar a cabo acciones relacionadas con el control de usuarios (por ejemplo, la validación de los campos proporcionados en el registro, la actualización de la información correspondiente comprobando los permisos...). Para ello, instancia la clase dbUser cuando es necesario y accede a la base de datos para actualizar sus atributos o bien actualiza la base de datos, según corresponda.

## 2.2.4 Diagrama de clases del backend

## Framework

Controller
<ul style="list-style-type: none"> <li>+ core : Core</li> <li>+ routing: Routing</li> <li>+ database: DataBase</li> <li>+ errors: []</li> </ul>
<ul style="list-style-type: none"> <li>+ public static function setCore(Core \$core)</li> <li>+ public static function setDataBase(DataBase \$dataBase)</li> <li>+ public static function setRouting(Routing \$routing)</li> <li>+ public static function setRouting(Routing \$routing)</li> <li>+ public function response(\$body = '', \$code = 200)</li> <li>+ public function addError(\$code, \$mess)</li> <li>+ public function checkErrors()</li> </ul>

DataBase
<ul style="list-style-type: none"> <li>- private static \$mysqli;</li> <li>+ public \$queries = [];</li> <li>- private \$config = [];</li> <li>+ public \$schema;</li> <li>+ public \$error;</li> <li>+ public \$erro;</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function __destruct()</li> <li>+ public function lastError()</li> <li>+ public function lastId()</li> <li>- private function refValues(\$arr)</li> <li>- private function query(\$query, \$params = [], \$hasResult = false)</li> <li>- private function log()</li> <li>- private function saveQuery(\$query, \$params)</li> <li>+ public function select(\$query, \$params = [])</li> <li>+ public function selectOne(\$query, \$params = [])</li> <li>+ public function modify(\$query, \$params = [])</li> <li>+ public function insert(\$tableName, \$object)</li> <li>+ public function update(\$tableName, \$object, \$idName)</li> <li>+ public function save(\$tableName, \$object, \$idName)</li> <li>+ public function startTrans()</li> <li>+ public function endTrans()</li> <li>+ public function getError(\$objeto)</li> </ul>

Session
<ul style="list-style-type: none"> <li>- private \$token;</li> <li>- private \$sessionsRoute;</li> <li>- private \$object;</li> <li>- private \$caducidadMin = 60;</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function save()</li> <li>+ public function caducada()</li> <li>+ public function token(\$length)</li> <li>- private function saveSession()</li> </ul>

Core
<ul style="list-style-type: none"> <li>+ ajax : boolean</li> <li>+ config: []</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>- private function autoloader()</li> <li>- private function debugSetup()</li> <li>- private function checkAjax()</li> <li>+ public function error(\$code)</li> <li>+ public function launch(\$className, \$method, \$param)</li> <li>+ public function shutdown()</li> <li>+ public function cleanBuffer()</li> </ul>

Files
<ul style="list-style-type: none"> <li>+ static public function deleteDir(\$dir, \$deleteDir = true)</li> </ul>

Routing
<ul style="list-style-type: none"> <li>+ public \$protocol = '';</li> <li>+ public \$protocolURL = '';</li> <li>+ public \$host = '';</li> <li>+ public \$uri = '';</li> <li>+ public \$segments = [];</li> <li>+ public \$routes = [];</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function getRoute()</li> <li>+ public function ruta(\$class, \$method)</li> <li>+ public function baseUrl()</li> </ul>

dbObject
<ul style="list-style-type: none"> <li>+ public \$tableName = 'prueba';</li> <li>+ public \$idName = 'pruebaid';</li> <li>+ public static \$dataBase = null;</li> <li>protected \$format;</li> <li>+ public \$errors = [];</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public static function setDataBase(DataBase \$dataBase)</li> <li>- private function setError(\$type, \$data)</li> <li>abstract protected function format()</li> <li>+ public static function createClasses()</li> <li>+ public function setObject(\$object)</li> <li>+ public function getFromPost()</li> <li>+ public function getFromDB(\$id)</li> <li>+ public function getObject()</li> <li>+ public function save()</li> <li>+ public function insert()</li> <li>+ public function update()</li> <li>+ public function validar(\$format, \$messages)</li> <li>+ public function delete()</li> </ul>

## dbObjects

dbcompetencia
<ul style="list-style-type: none"> <li>+ public \$tableName = 'competencia';</li> <li>+ public \$idName = 'codigo';</li> <li>+ public \$codigo;</li> <li>+ public \$nivel;</li> <li>+ public \$nombre;</li> <li>+ public \$medios;</li> <li>+ public \$productos;</li> <li>+ public \$informacion;</li> <li>+ public \$realizaciones;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>
dbcualif_compet
<ul style="list-style-type: none"> <li>+ public \$tableName = 'cualif_compet';</li> <li>+ public \$idName = 'cualifCompetid';</li> <li>+ public \$cualifCompetid;</li> <li>+ public \$cualificacion;</li> <li>+ public \$competencia;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>
dbmessage
<ul style="list-style-type: none"> <li>+ public \$tableName = 'message';</li> <li>+ public \$idName = 'messageld';</li> <li>+ public \$messageld;</li> <li>+ public \$senderid;</li> <li>+ public \$receiverid;</li> <li>+ public \$fechaAdd;</li> <li>+ public \$body;</li> <li>+ public \$senderDelete;</li> <li>+ public \$receiverDelete;</li> <li>+ public \$leido;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>
dbusertype
<ul style="list-style-type: none"> <li>+ public \$tableName = 'usertype';</li> <li>+ public \$idName = 'usertypeid';</li> <li>+ public \$usertypeid;</li> <li>+ public \$userid;</li> <li>+ public \$type;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>
dbcualificacion
<ul style="list-style-type: none"> <li>+ public \$tableName = 'cualificacion';</li> <li>+ public \$idName = 'codigo';</li> <li>+ public \$codigo;</li> <li>+ public \$nivel;</li> <li>+ public \$nombre;</li> <li>+ public \$familia;</li> <li>+ public \$descripcion;</li> <li>+ public \$sentomo;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>
dbitinerario
<ul style="list-style-type: none"> <li>+ public \$tableName = 'itinerario';</li> <li>+ public \$idName = 'itinerariold';</li> <li>+ public \$itinerariold;</li> <li>+ public \$userid;</li> <li>+ public \$cualificacion;</li> <li>+ public \$terminada;</li> <li>+ public \$fechaFin;</li> <li>+ public \$orden;</li> <li>+ public \$fechaAdd;</li> <li>+ public \$userAdd;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>
dbuser
<ul style="list-style-type: none"> <li>+ public \$tableName = 'user';</li> <li>+ public \$idName = 'userid';</li> <li>+ public \$userid;</li> <li>+ public \$email;</li> <li>+ public \$passwd;</li> <li>+ public \$formacion;</li> <li>+ public \$intereses;</li> <li>+ public \$perspectivas;</li> <li>+ public \$activo;</li> <li>+ public \$foto;</li> <li>+ public \$nombre;</li> <li>+ public \$apellido;</li> <li>+ public \$fechaAdd;</li> </ul>
<ul style="list-style-type: none"> <li>+ function format()</li> </ul>

## Controllers

competencias
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function get(\$id = null)</li> <li>+ public function getAll()</li> <li>+ public function set(\$id = null)</li> <li>+ public function mover(\$itinerarios)</li> <li>+ public function delete(\$id = null)</li> <li>+ public function checkId()</li> </ul>
cualificaciones
<ul style="list-style-type: none"> <li>+ public static \$familias</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function get(\$id = null)</li> <li>+ public function getAll()</li> <li>+ public function set(\$id = null)</li> <li>+ public function delete(\$id = null)</li> <li>+ public function checkId()</li> </ul>
login
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function autoLogin</li> <li>+ public function login()</li> </ul>
messages
<ul style="list-style-type: none"> <li>+ public \$user = null</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function messageValidation()</li> <li>+ public function save(\$userid = null)</li> <li>+ public function get(\$userid = null)</li> <li>+ public function getAll()</li> <li>+ public function delete(\$userid)</li> <li>+ public function activar(\$userid)</li> <li>+ public function esFormador(\$userid)</li> </ul>
messages
<ul style="list-style-type: none"> <li>+ public \$user</li> </ul>
<ul style="list-style-type: none"> <li>+ public function __construct()</li> <li>+ public function send()</li> <li>+ public function getAll(\$sent = null)</li> <li>+ public function delete(\$messageld)</li> <li>+ public function leer(\$messageld)</li> </ul>

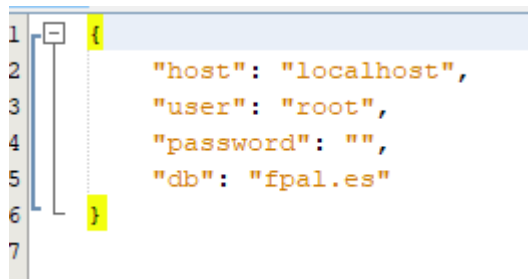


### 2.2.5 Referencia de ficheros de configuración del backend

- **Fichero conf/core.json:** determina si el backend está en producción o en desarrollo (solo tiene la propiedad production).
- **Fichero conf/routing.json:** almacena para cada URI y el método con el que se llame la ruta donde está la clase que debe ejecutarse y dentro de la clase, el método que se debe invocar.

```
"/messages":{  
  "GET": "messages/getAll",  
  "POST": "messages/send",  
  "DELETE": "messages/delete"  
},
```

- **Fichero conf/dataBase.json:** contiene los parámetros de configuración necesarios para conectarse a la Base de Datos:



```
1 {  
2   "host": "localhost",  
3   "user": "root",  
4   "password": "",  
5   "db": "fpal.es"  
6 }  
7
```

## 3 MODELO DE DATOS

### 3.1 Explicación de los datos utilizados

Se trata de una representación conceptual de los elementos sobre los que FPAL almacena y procesa información, así como de las relaciones entre ellos.

Las principales entidades que en él se reflejan son las siguientes:

- **Usuario:** es la figura central en la aplicación, dado que dependiendo del tipo de usuario se le permitirá o denegará la realización de múltiples acciones en la plataforma. Algunos de los datos que se almacenan sobre los usuarios son su nombre, apellidos, correo electrónico, foto, perspectivas e intereses... Así como el tipo de usuario (puede ser formador, estudiante o administrador) y el estado (un usuario no estará activo hasta que así lo determine el administrador).
- **Cualificación Profesional:** es el conjunto de competencias profesionales con significación en el empleo que pueden ser adquiridas mediante formación o experiencia laboral.
- **Competencia:** es el conjunto de conocimientos y capacidades que permiten el ejercicio de la actividad profesional conforme a las exigencias de la producción y del empleo.
- **Itinerario:** es un agregado de cualificaciones que, asociado a un usuario, permite proporcionarle un plan de formación con una trayectoria que amplíe su empleabilidad en vez de adquirir competencias y cualificaciones sin una orientación profesional específica.
- **Mensaje:** son los que se intercambian entre los usuarios de la plataforma. Van cambiando de estado cuando los usuarios interactúan con ellos.

Para el modelado de Competencias y Cualificaciones se ha tomado como referencia los datos mínimos que establece la Ley 5/2002 sobre las Cualificaciones y la Formación Profesional.:

## ¿QUÉ ES UNA CUALIFICACIÓN PROFESIONAL?

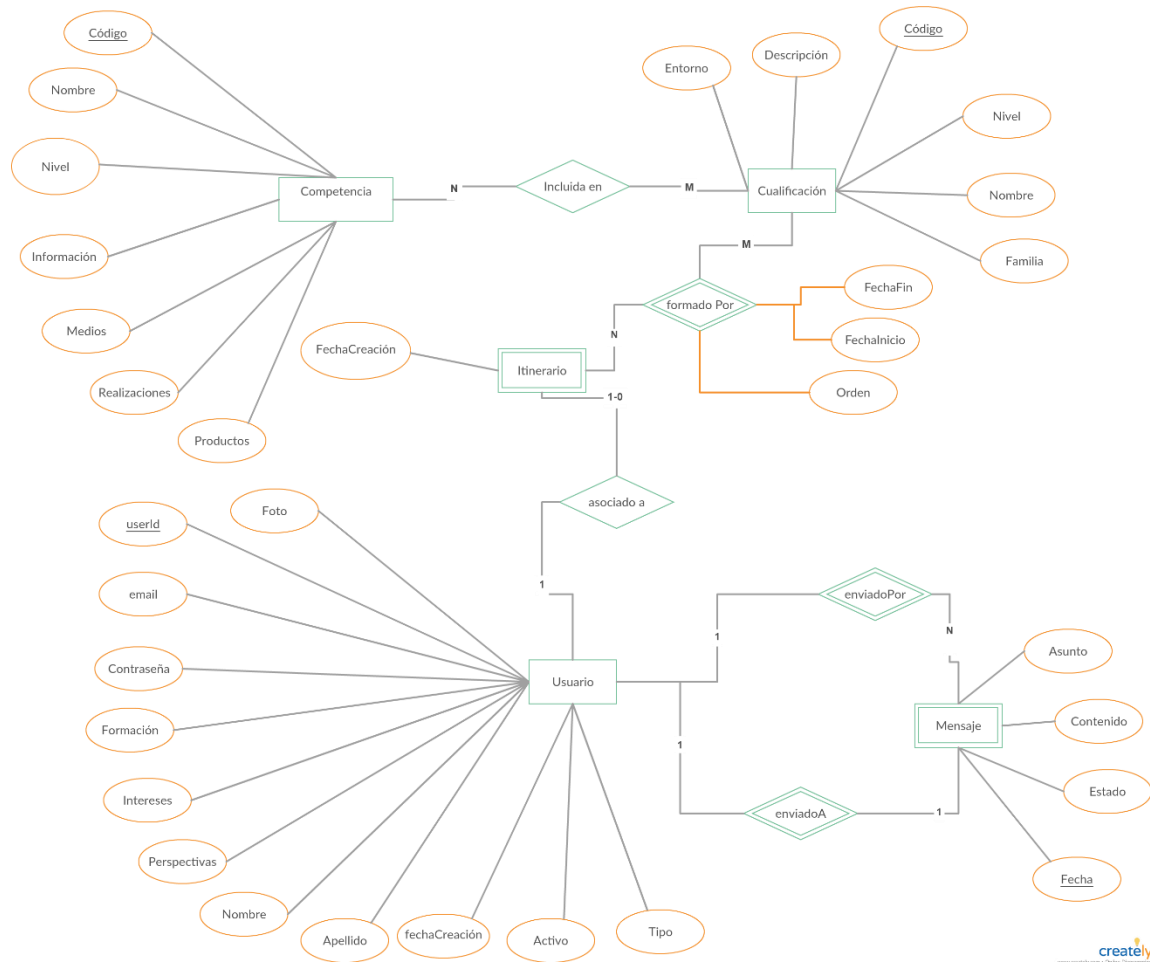
Es el conjunto de competencias (conocimientos y capacidades) válidas para el ejercicio de una actividad profesional que pueden adquirirse a través de la formación o de la experiencia laboral. Las cualificaciones se agrupan en 26 familias y 3 niveles.



## ESTRUCTURA DE LA UNIDAD DE COMPETENCIA



### 3.2 Diagrama Entidad – Relación



## 4 ENTORNO DE EJECUCIÓN DEL PROYECTO – GUÍA DE DESPLIEGUE

### 4.1 Requisitos para la instalación

La aplicación debe instalarse en un entorno WAMP (Windows – Apache – PHP – MySQL). Se recomiendan las siguientes versiones:

- La **versión del sistema operativo** es indiferente, siempre que se trate de una plataforma de 64 bits que dé soporte a los demás programas necesarios. Cualquier **hardware capaz de ejecutar estas versiones del sistema operativo podrá ejecutar la aplicación sin problemas**, si bien para entornos de alto rendimiento, alta disponibilidad o de elevada carga de usuario pueden elevarse los requisitos hardware.
- **Apache**: la versión recomendada es la 2.4.27 o posterior.
- **PHP**: se recomienda utilizar la versión 5.6.31 o posterior
- **MySQL**: la versión probada es la 5.7.19, por lo que se recomienda esa misma o posterior.

Además, resulta recomendable utilizar un frontend para gestionar la base de datos, que durante el desarrollo ha sido PHPMyAdmin en su versión 4.7.4.

Por último, debe crearse una asociación de nombres, sea local o por DNS, en la que las peticiones se dirijan a fpal.ddns.net.

### 4.2 Procedimiento de instalación

La aplicación se distribuye en un fichero comprimido que contiene dos directorios (fpal y fpal-serv) que se corresponden respectivamente con el frontend y el backend así como el fichero de configuración de vhosts del servidor y el dump de la base de datos.

#### 4.2.1 Copia de los ficheros en el servidor

Una vez instalados los programas antes mencionados, deben copiarse las carpetas fpal y fpal-serv (proporcionadas en un fichero comprimido) en el directorio web de Apache (el directorio donde Apache busque los contenidos que debe servir ante una petición).

#### 4.2.2 Configuración del servidor web

Es necesario crear un host virtual que atienda las peticiones dirigidas a fpal.ddns.es y fpal-serv.ddns.net.

La configuración a aplicar en el fichero de hosts virtuales es la siguiente:

```

1  <VirtualHost *:80>
2      ServerName fpal.es
3      ServerAlias fpal.ddns.net
4      DocumentRoot c:/wamp64/www/fpal/src
5      <Directory "c:/wamp64/www/fpal/src">
6          Options +Indexes +Includes +FollowSymLinks +MultiViews
7          AllowOverride All
8          Require all granted
9      </Directory>
10 </VirtualHost>
11
12 <VirtualHost *:80>
13     ServerName fpal-serv.es
14     ServerAlias fpal-serv.ddns.net
15     DocumentRoot c:/wamp64/www/fpal-serv
16     <Directory "c:/wamp64/www/fpal-serv">
17         Options +Indexes +Includes +FollowSymLinks +MultiViews
18         AllowOverride All
19         Require all granted
20     </Directory>
21 </VirtualHost>
22

```

El fragmento anterior debe pegarse en el fichero de configuración de hosts virtuales (en la versión de apache antes mencionada, dentro de .... conf/extra/httpd-vhosts.conf

### 4.2.3 Carga de datos de prueba en la base de datos

El siguiente script permite la creación del esquema de base de datos y su población por un conjunto de datos de prueba. En todos los usuarios, la contraseña es Aasdf1234.

Debe existir una base de datos vacía llamada fpal.es en el sistema gestor de base de datos.

```

-- phpMyAdmin SQL Dump
-- version 4.7.4
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1:3306
-- Generation Time: May 16, 2018 at 05:59 PM
-- Server version: 5.7.19
-- PHP Version: 5.6.31

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `fpal.es`
--

--
-- Table structure for table `competencia`
--

```

```

DROP TABLE IF EXISTS `competencia`;
CREATE TABLE IF NOT EXISTS `competencia` (
  `codigo` varchar(45) NOT NULL,
  `nivel` int(11) NOT NULL,
  `nombre` varchar(255) NOT NULL,
  `medios` varchar(2000) DEFAULT NULL,
  `productos` varchar(2000) DEFAULT NULL,
  `informacion` varchar(2000) DEFAULT NULL,
  `realizaciones` varchar(10000) DEFAULT NULL,
  PRIMARY KEY (`codigo`),
  UNIQUE KEY `nombre_UNIQUE` (`nombre`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Dumping data for table `competencia`
--

INSERT INTO `competencia` (`codigo`, `nivel`, `nombre`, `medios`,
`productos`, `informacion`, `realizaciones`) VALUES
('UC0255_1', 1, 'Ejecutar operaciones básicas de aprovisionamiento,
preelaboración y conservación culinarios', 'Equipos e instrumentos de
medida. Almacenes. Equipos de refrigeración. Mobiliario y maquinaria
propia\nde un cuarto frío: abatidor de temperatura, máquinas de vacío,
batidoras de brazo, máquina corta\nfiambres, vasos batidores y otros.
Equipos de cocción. Pilas estáticas y móviles para lavar
géneros\ncrudos. Utensilios y herramientas propios de la
preelaboración culinaria. Materias primas, mercancías y\nmaterial de
acondicionamiento y etiquetado.', 'Limpieza y mantenimiento de
equipos, máquinas y útiles propios de cuarto frío.
Registro\ncumplimentado con datos correspondientes a recepción,
almacenamiento y distribución en los soportes\nestablecidos.
Mercancías y géneros preelaborados, acondicionados y conservados para
su consumo\ninmediato o posterior distribución.', 'Instrucciones de
seguridad, uso y manipulación de productos de limpieza y desinfección.
Manuales de\nfuncionamiento de equipos, maquinaria e instalaciones
propias de cuarto frío. Documentos\nnormalizados como \'relevés\',
vales de pedidos, albaranes y fichas de almacén. Órdenes de
trabajo.\nTablas de temperaturas de conservación de alimentos.
Normativa aplicable de manipulación de\nalimentos, medioambiental y
riesgos laborales', '[{"title": "Ejecutar las diferentes operaciones
de limpieza y puesta a punto de equipos y utillaje en el \u00e1rea de
producci\u00f3n culinaria, respetando la normativa aplicable
higi\u00e9nico\u2010sanitaria y las instrucciones
recibidas", "criterios": ["La limpieza de superficies, equipos y
utillaje propios del \u00e1rea de producci\u00f3n culinaria
se\nefect\u00faa seg\u00fan el sistema establecido, con los
productos y m\u00e1todos determinados."]]]'),
('UC0256_1', 1, 'Asistir en la elaboración culinaria y realizar y
presentar preparaciones sencillas', 'Equipos e instrumentos de medida.
Almacenes. Equipos de refrigeración. Mobiliario y maquinaria
propia\nde un cuarto frío: abatidor de temperatura, máquinas de vacío,
batidoras de brazo, máquina corta\nfiambres, vasos batidores y otros.
Equipos de cocción. Pilas estáticas y móviles para lavar
géneros\ncrudos. Utensilios y herramientas propios de la
preelaboración culinaria. Materias primas, mercancías y\nmaterial de
acondicionamiento y etiquetado. Material de limpieza.', 'Limpieza y
mantenimiento de equipos, máquinas y útiles propios de cuarto frío.
Registro\ncumplimentado con datos correspondientes a recepción,
almacenamiento y distribución en los soportes\nestablecidos.
Mercancías y géneros preelaborados, acondicionados y conservados para

```

```

su consumo\ninmediato o posterior distribución.', 'Instrucciones de
seguridad, uso y manipulación de productos de limpieza y desinfección.
Manuales de\nfuncionamiento de equipos, maquinaria e instalaciones
propias de cuarto frío. Documentos\nnormalizados como \'relevés\',
vales de pedidos, albaranes y fichas de almacén. Órdenes de
trabajo.\nTablas de temperaturas de conservación de alimentos.
Normativa aplicable de manipulación de\nalimentos, medioambiental y
riesgos laborales.', ' [{\"title\": \"Ejecutar las diferentes
operaciones de limpieza y puesta a punto de equipos y utillaje en el
\\u00elrea de producci\\u00f3n culinaria, respetando la normativa
aplicable higi\\u00e9nico\\u2010sanitaria y las instrucciones
recibidas.\", \"criterios\": [\" La limpieza de superficies, equipos y
utillaje propios del \\u00elrea de producci\\u00f3n culinaria
se\\nefect\\u00faa seg\\u00fan el sistema establecido, con los
productos y m\\u00e9todos determinados\", \"Las instrucciones de
seguridad, uso y manipulaci\\u00f3n de productos utilizados en
la\\nlimpieza y puesta a punto se cumplen, teniendo en cuenta su
posible toxicidad y contaminaci\\u00f3n\\nmedioambiental\"]}]'),
('UC1769_2', 2, 'Conducir Black Jack', 'Medios', 'Productos', 'Info',
' [{\"title\": \"Conducir el turno de apuestas con clientes en el juego
de Black Jac\", \"criterios\": [\"Criterio 1\\\", \"Criterio 2\\\", \"Criterio
3\\\"]}]'),
('UC69_2', 2, 'Conducir el juego de Black Jack', 'Medios de
producción', 'Productos y resultados', 'Info usada o generada',
' [{\"title\": \"Organizar los elementos materiales
necesarios\", \"criterios\": [\"Las fichas se exponen sobre la mesa de
juego ordenad\\\", \"Las fichas se cuentan en lo que se refiere a su
cantida\\\", \"Las fichas se guardan de manera ordenada, seg\\u00fan el
proced\\\"]}]');

```

```

-- -----
--
-- Table structure for table `cualificacion`
--

```

```

DROP TABLE IF EXISTS `cualificacion`;
CREATE TABLE IF NOT EXISTS `cualificacion` (
  `codigo` varchar(45) NOT NULL,
  `nivel` int(11) NOT NULL,
  `nombre` varchar(255) NOT NULL,
  `familia` varchar(255) NOT NULL,
  `descripcion` varchar(2000) DEFAULT NULL,
  `entorno` varchar(2000) DEFAULT NULL,
  PRIMARY KEY (`codigo`),
  UNIQUE KEY `nombre_UNIQUE` (`nombre`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

```

--
-- Dumping data for table `cualificacion`
--

```

```

INSERT INTO `cualificacion` (`codigo`, `nivel`, `nombre`, `familia`,
`descripcion`, `entorno`) VALUES
('HOT541_2', 2, 'Actividades para el juego en mesas de casinos',
'Hostelería y turismo', 'Facilitar el desarrollo de los juegos de
Black Jack, Póquer con descarte y Póquer sin descarte, Pu',
'Desarrolla su actividad pro'),
('HOT91_1', 1, 'Operaciones básicas de cocina', 'Hostelería y
turismo', 'Preelaborar alimentos, preparar y presentar elaboraciones
culinarias sencillas y asistir en la preparación de elaboraciones más

```



complejas, ', 'Desarrolla su actividad profesional como auxiliar o ayudante, tanto en grandes como en medianas y pequeñas empresas, principalmente del sector de hostelería');

```
-- -----

--
-- Table structure for table `cualif_compet`
--

DROP TABLE IF EXISTS `cualif_compet`;
CREATE TABLE IF NOT EXISTS `cualif_compet` (
  `cualifCompetId` int(11) NOT NULL AUTO_INCREMENT,
  `cualificacion` varchar(45) NOT NULL,
  `competencia` varchar(45) NOT NULL,
  PRIMARY KEY (`cualifCompetId`),
  UNIQUE KEY `comp_cualif` (`cualificacion`,`competencia`),
  KEY `competenciaId_idx` (`competencia`),
  KEY `cualificacionId_idx` (`cualificacion`)
) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8;

--
-- Dumping data for table `cualif_compet`
--

INSERT INTO `cualif_compet` (`cualifCompetId`, `cualificacion`,
`competencia`) VALUES
(14, 'HOT541_2', 'UC1769_2'),
(15, 'HOT541_2', 'UC69_2'),
(9, 'HOT91_1', 'UC0255_1'),
(10, 'HOT91_1', 'UC0256_1');

-- -----

--
-- Table structure for table `itinerario`
--

DROP TABLE IF EXISTS `itinerario`;
CREATE TABLE IF NOT EXISTS `itinerario` (
  `itinerarioId` int(11) NOT NULL AUTO_INCREMENT,
  `userId` int(11) NOT NULL,
  `cualificacion` varchar(45) NOT NULL,
  `terminada` tinyint(4) DEFAULT NULL,
  `fechaFin` datetime DEFAULT NULL,
  `orden` int(11) NOT NULL,
  `fechaAdd` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `userAdd` int(11) NOT NULL,
  PRIMARY KEY (`itinerarioId`),
  UNIQUE KEY `alumno_cualif` (`userId`,`cualificacion`),
  UNIQUE KEY `orden` (`userId`,`orden`),
  KEY `alumno_idx` (`userId`),
  KEY `cualificacion_idx` (`cualificacion`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;

--
-- Dumping data for table `itinerario`
--

INSERT INTO `itinerario` (`itinerarioId`, `userId`, `cualificacion`,
`terminada`, `fechaFin`, `orden`, `fechaAdd`, `userAdd`) VALUES
```

```

(1, 5, 'HOT91_1', NULL, NULL, 1, '2018-04-09 20:32:10', 4),
(2, 5, 'HOT541_2', NULL, NULL, 2, '2018-04-09 21:04:44', 1),
(3, 7, 'HOT91_1', NULL, NULL, 1, '2018-04-10 13:19:00', 1),
(4, 8, 'HOT541_2', NULL, NULL, 1, '2018-04-11 09:48:51', 6);

-- -----

--
-- Table structure for table `message`
--

DROP TABLE IF EXISTS `message`;
CREATE TABLE IF NOT EXISTS `message` (
  `messageId` int(11) NOT NULL AUTO_INCREMENT,
  `senderId` int(11) NOT NULL,
  `receiverId` int(11) NOT NULL,
  `fechaAdd` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `subject` varchar(255) DEFAULT NULL,
  `body` varchar(2000) DEFAULT NULL,
  `senderDelete` tinyint(4) DEFAULT NULL,
  `receiverDelete` tinyint(4) DEFAULT NULL,
  `leido` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`messageId`),
  KEY `sender_idx` (`senderId`),
  KEY `receiver_idx` (`receiverId`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8;

--
-- Dumping data for table `message`
--

INSERT INTO `message` (`messageId`, `senderId`, `receiverId`,
`fechaAdd`, `subject`, `body`, `senderDelete`, `receiverDelete`,
`leido`) VALUES
(9, 1, 1, '2017-06-08 21:23:28', 'Otro asunto', 'Y un texto', 1, 1,
NULL),
(10, 1, 1, '2017-06-12 20:20:33', 'asdf', 'asdf', 1, 1, NULL),
(11, 5, 4, '2018-04-09 18:27:45', 'Inicio de mi formación', 'Me
gustaría obtener mi itinerario formativo', NULL, NULL, 1);

-- -----

--
-- Table structure for table `user`
--

DROP TABLE IF EXISTS `user`;
CREATE TABLE IF NOT EXISTS `user` (
  `userId` int(11) NOT NULL AUTO_INCREMENT,
  `email` varchar(255) CHARACTER SET latin1 NOT NULL,
  `passwd` varchar(255) CHARACTER SET latin1 NOT NULL,
  `formacion` varchar(3000) CHARACTER SET latin1 DEFAULT NULL,
  `intereses` varchar(3000) CHARACTER SET latin1 DEFAULT NULL,
  `perspectivas` varchar(3000) CHARACTER SET latin1 DEFAULT NULL,
  `activo` tinyint(4) DEFAULT NULL,
  `foto` varchar(255) CHARACTER SET latin1 DEFAULT NULL,
  `nombre` varchar(255) CHARACTER SET latin1 NOT NULL,
  `apellido` varchar(255) CHARACTER SET latin1 NOT NULL,
  `fechaAdd` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`userId`),
  UNIQUE KEY `email` (`email`)

```

```

) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;

--
-- Dumping data for table `user`
--

INSERT INTO `user` (`userId`, `email`, `passwd`, `formacion`,
`intereses`, `perspectivas`, `activo`, `foto`, `nombre`, `apellido`,
`fechaAdd`) VALUES
(1, 'admin@fpal.es',
'$2y$10$rcjv7TKjET9fXHgaiWSR9uD5bAiNJvwuO8IOrhWwMqcrAg6/9EqGC', NULL,
NULL, NULL, 1, '', 'Admin', '-', '2017-05-08 21:35:48'),
(3, 'admin2@fpal.es',
'$2y$10$xiWVVGv9qwdTIVflrc51N.5xAmPdoDhQvfYtKjFkZnig.cws.Dkga', '',
'', '', NULL, '', 'A', 'S', '2018-04-05 14:56:04'),
(4, 'maria.frades.3@fpal.es',
'$2y$10$0J2X6NpLOWlQcArBe6FqFurLdattajh6R7HCDFaII.8g8tRZSoY7G', 'Otra
formación', 'Otros intereses', 'Otras perspectivas', 1,
'files/profile5.png', 'María', 'Frades', '2018-04-09 20:17:04'),
(5, 'julia.frades@fpal.es',
'$2y$10$WUigvGR2kiCqcmYqbIIm9.TfmJ2oAK3NTur6HEHynLNgZhnSDDV8m',
'Formación de Julia', 'Intereses de Julia', 'Perspectivas de Julia',
1, 'files/profile2.png', 'Julia', 'Frades Castro', '2018-04-09
20:25:28'),
(6, 'irene.frades@fpal.es',
'$2y$10$XBqq2xGA.OkWyhCeqTzble9zd8eG/58w/WBUT278s05pSvUcl4Yi.',
'Formación de Irene', 'Intereses de Irene', 'Perspectivas de Irene',
1, 'files/profile4.png', 'Irene', 'Frades', '2018-04-09 20:54:01'),
(7, 'rodrigo.sanchez@fpal.es',
'$2y$10$HYiAcSBKRN8sVfHah84USezelvqjnfCohRLT17fa9b7NiU5kEsLN.',
'Formación de Rodrigo', 'Intereses de Rodrigo', 'Perspectivas de
Rodrigo', 1, 'files/profile_0.png', 'Rodrigo', 'Sánchez', '2018-04-09
20:54:47'),
(8, 'carlos@fpal.es',
'$2y$10$h/VXXCmv65BfQUyumxCFX.dtkSnBLSfhIWkX41Ie4oHV6dVOjipOe',
'Ingeniería en Informática', '', '', 1, '', 'Carlos ', 'González
Contreras', '2018-04-11 09:46:41'),
(9, 'julia.frades2@fpal.es',
'$2y$10$uV4l1j/3ejw53lSM1Rkf5.ual6OSSKyXDJdcnWQK8UED5fxfqF4Xy', '',
'', '', NULL, '', 'Julia2', 'Frades', '2018-04-23 17:15:23'),
(10, 'mail@de.prueba',
'$2y$10$BXeKpGbdV77Iot2z0PtKX.e6ijnTA.f3hhoLDaB3aXmotlojyMZiC', '',
'', '', NULL, '', 'Usuariode', 'Prueba', '2018-04-23 17:33:19'),
(11, 'email@deprue.ba',
'$2y$10$G1Wsmthbc4/hquP4PRHZevztFx.Q.rE8lV2CSg2JpIeKb7QfKqEu', '',
'', '', NULL, '', 'Nombre', 'Apellidos', '2018-04-23 17:34:44'),
(12, 'correo@deprue.ba',
'$2y$10$1oTteUtD0DEINQeOXNoeBuoqnTZexEpXUosZCi7jTUKPn7PHk.dZW', '',
'', '', NULL, '', 'Usuario', 'Apellidos', '2018-04-23 17:36:11');

--
-----

--
-- Table structure for table `usertype`
--

DROP TABLE IF EXISTS `usertype`;
CREATE TABLE IF NOT EXISTS `usertype` (
  `userId` int(11) NOT NULL AUTO_INCREMENT,
  `userId` int(11) NOT NULL,

```

```

    `type` enum('Administrador','Formador','Alumno') CHARACTER SET
latin1 NOT NULL,
    PRIMARY KEY (`userId`),
    UNIQUE KEY `userType` (`userId`,`type`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

--
-- Dumping data for table `usertype`
--

INSERT INTO `usertype` (`userId`, `type`) VALUES
(3, 1, 'Administrador'),
(8, 4, 'Formador'),
(7, 5, 'Alumno'),
(11, 6, 'Formador'),
(10, 7, 'Alumno'),
(12, 8, 'Alumno'),
(13, 9, 'Alumno'),
(14, 10, 'Alumno'),
(15, 11, 'Alumno'),
(16, 12, 'Alumno');

--
-- Constraints for dumped tables
--

--
-- Constraints for table `cualif_compet`
--
ALTER TABLE `cualif_compet`
  ADD CONSTRAINT `compet` FOREIGN KEY (`competencia`) REFERENCES
`competencia` (`codigo`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `cualif` FOREIGN KEY (`cualificacion`) REFERENCES
`cualificacion` (`codigo`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `itinerario`
--
ALTER TABLE `itinerario`
  ADD CONSTRAINT `alumno` FOREIGN KEY (`userId`) REFERENCES `user`
(`userId`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `cualificacion` FOREIGN KEY (`cualificacion`)
REFERENCES `cualificacion` (`codigo`) ON DELETE CASCADE ON UPDATE
CASCADE;

--
-- Constraints for table `message`
--
ALTER TABLE `message`
  ADD CONSTRAINT `receiver` FOREIGN KEY (`receiverId`) REFERENCES
`user` (`userId`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `sender` FOREIGN KEY (`senderId`) REFERENCES `user`
(`userId`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `usertype`
--
ALTER TABLE `usertype`
  ADD CONSTRAINT `user-type` FOREIGN KEY (`userId`) REFERENCES `user`
(`userId`) ON DELETE CASCADE ON UPDATE CASCADE;
COMMIT;

```

```
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```