FRAMEWORKS

- **Symfony** como framework de PHP
- **FosRestBundle.** Definición de controladores para las API REST
- **FosUserBundle.** Gestión de usuarios.
- **NelmioAPIDoc.** Documentación de la API
- **LexikJWTAuthentication**. Gestionar token y permisos de acceso a los diferentes path.
- **Doctrine.** Framework para conectarse al modelo de datos de sql.
- **PHPUnit** para testing.

CONTROLLERS

- **RestUserLoginController**. Endpoint para el logueo y obtención del token
- **RestUserRegisterController**. Endpoint para el registro de usuario
- **RestCarCoordinatesController**. Endpoint para obtener las coordenadas de los vehículos que se encuentran como máximo x km dadas unas coordenadas.
- **RestCarReservationController**. Endpoint para reservar un vehículo dado un identificador y una fecha de inicio y una de fin.
- **ExceptionController.** Cuando se produce o se provoca una APIException, este será el controlador que recoja la respuesta.

SERVICIOS

- **CarListService.** Servicio que nos permite obtener el listado de vehículos de la api rest de modo. Se ha centralizado aquí por si, el día de mañana se modifican los datos de respuesta de la api, tenerlos aquí y que sea fácil de mantener.
- **CarService.** Servicio de entrada de los controllers de car. La idea es que aquí esten las operaciones básicas de cada una de las funciones de los controllers.
- **ClientFactory.** Servicio que crea el cliente de la petición de Guzzle. Se ha separado del resto para que si se decide que sea otro servicio para hacer peticiones, con modificarlo aquí ya es suficiente.
- **DatabaseConnectionService.** Se ha creado este servicio para que si en un futuro se decide cambiar doctrine por otro framework, se haga directamente en esta clase.
- **LocationListService.** Servicio que nos permite obtener el listado de localizaciones de la api rest de modo. Se ha centralizado aquí por si, el día de mañana se modifican los datos de respuesta de la api, tenerlos aquí y que sea fácil de mantener.
- **RequestHandlerService.** Servicio que usa el cliente ClientFactory para realizar las peticiones que se precisen, en este caso, a la api modo para el listado de coches y localizaciones.
- **ResponseViewService.** Servicio donde se "crean" las vistas que recibirán los controladores. Se ha basado un poco en la salida de la api de modo para copiar sus salidas. Se ha separado para que, si se desea cambiar las salidas, esté centralizado en un solo sitio.
- **UserService.** Servicio de de entrada de los user controllers.

UTILIDADES

- **APIException.** Excepción que se lanzará en la aplicación y que será mostrada por el **ExceptionController.**

- **DatabaseQueries.** Me gusta separar en una clase estática las queries a la base de datos. Así se dónde buscarlas siempre.
- **ModoHandlerRequest.** Interficie para LocationListService y CarListService. Así nos aseguramos que en ambos servicios, tendrán el método getContents para poder ver el resultado de las request a la api modo.
- **Utilities.** Clase donde hay algunos métodos como el que calcula la distancia entre dos puntos o un validador de email.

ENTITIES

- **HTTPResponseEntity.** Guardamos la response del cliente en un objeto más amigable.
- **CarReservation.** DTO de la reserva de vehículos.
- **User.** DTO de usuario.

TESTING

Se han creado tests de los controladores. Además de una DataFixture para cargar un usuario root para los tests.

Antes de empezar cada uno de los tests, se crea una base de datos testing (test_modo) para no operar sobre la base de datos real (modo). Una vez acabo los tests, se destruye. Esto se hace en el setUp y el tearDown de la clase **FunctionalTestHelper**.