UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Fleet Manager: Software de Gestión de Flotas desarrollado con Symfony2

Proyecto Final de Carrera

Ingeniería Informática

Autor: Miguel Ángel Sánchez Chordi

Director: Sergio Sáez Barona

18/04/2013

Resumen

Fleet Manager es una aplicación web de gestión desarrollada para la empresa Paul Günther España, dedicada al alquiler de vehículos de transporte de alto tonelaje. Este proyecto ha sido desarrollado íntegramente con software libre. La realización de este proyecto ha servido para aprender a desarrollar aplicaciones web con una metodología ágil basada en el framework Symfony2.

Palabras clave: symfony2, jquery, apache, gestión, vehículos

Tabla de contenidos

1. Introducción

1.1 Propósito del proyecto

El propósito del siguiente proyecto consiste en el desarrollo de una aplicación destinada a la gestión interna de una empresa dedicada al *renting* de vehículos de alto tonelaje.

La empresa en cuestión, *Paul Günther S.L*, había estado realizando todas las gestiones de contratos, altas de vehículos, gastos derivados, revisiones obligatorias de los vehículos... etc, mediante hojas de cálculo que ellos mismos habían convenido a usar.

Con el paso del tiempo y el crecimiento de la empresa, esta solución se volvió inviable: Cada trabajador llevaba sus propias hojas de cálculo, que no tenían porqué estar actualizadas respecto a las de sus compañeros, el número de vehículos y clientes a gestionar iba en aumento y el desorden era cada vez mayor, además, debido a la necesidad de desplazamiento de algunos de los comerciales, que fácilmente podían pasarse fuera hasta una semana, en la oficina central la desactualización de los contenidos era cada vez mayor.

Dada la situación, *Paul Günther S.L* se puso en contacto con *Pro Solutions - Soluciones Informáticas Integrales* con el fin de ver qué opciones tenían para facilitar su trabajo. Desde Pro Solutions, se decidió ofrecer al cliente la creación de un software de gestión integrada de la empresa, donde todos pudieran acceder y manejar la misma información. Además, el desarrollo iba a hacerse en una plataforma web, permitiendo así el acceso desde cualquier punto y dispositivo a la herramienta de gestión.

1.2 Estructura de este documento

El documento realiza una descripción progresiva sobre las nociones básicas para comprender la naturaleza del proyecto.

En el primer capítulo realiza una introducción a las necesidades concretas del cliente de la aplicación.
En el segundo capítulo se trata el análisis funcional de la aplicación mediante una Especificación de Requisitos Software (ERS a partir de ahora) basada en el estándar IEEE 830/1998.
En el tercer capítulo se abordan aspectos como el modelo de datos usado y los casos de uso que la aplicación deberá contemplar.
El cuarto capítulo describe el diseño de la aplicación y el patrón usado, en este caso *Modelo Vista Controlador* (MVC a partir de ahora).
El quinto capítulo aborda la implementación de la aplicación, describiendo las tecnologías usadas para el desarrollo de manera que el lector pueda comprender los entresijos de su funcionamiento.
El sexto capítulo está destinado a comentar el resultado de una batería de pruebas realizadas a la aplicación para determinar si cumple correctamente con su cometido.
El manual básico de usuario esta en el capítulo siete.

Las conclusiones finales, tanto los aspectos personales como técnicos, son tratadas en el octavo capítulo.

Por último, el noveno capítulo contiene la bibliografía usada durante todo el desarrollo de esta aplicación.

1.3. Estudio Previo

En una fase previa al desarrollo de la aplicación se decidió buscar qué otras soluciones había ya implementadas y analizarlas para determinar si eran útiles para *Paul Günther*.

El mercado actual contempla algunas aplicaciones comerciales de escritorio como el *Visual Rentacar* o *Remis Manager* que resultaron ser un software demasiado simple, poco intuitivo, nada ampliable y que poco o nada tenía que ver con el sistema de gestión que había estado planteado *Paul Günther* desde sus inicios.

Dado que el cliente era poco receptivo a cambiar su *modus operandi* y que el software encontrado no satisfacía el requisito de la movilidad, se planteó la opción de desarrollar un software de gestión a medida, basado íntegramente en su sistema de gestión actual.

2.Especificación de Requisitos Software

2.1. Introducción

La ERS o *Especificación de Requisitos Software* es un documento que sirve para que tanto el cliente, como el diseñador, el analista y el desarrollador, entienda la naturaleza del proyecto y acuerden los requisitos funcionales que la misma habrá de cumplir.
Para hacer esta ERS, se ha seguido el estándar IEEE 830/98.

2.1.1. Propósito

La finalidad de este proyecto es servir de sistema de gestión centralizado para la empresa cliente, permitiéndoles así mantener sus datos coherentes y actualizados.

Se pretende plasmar el funcionamiento actual de la empresa, de modo que la adaptación al software por parte de los futuros usuarios se haga de la manera más transparente posible y la curva de aprendizaje sea lo más baja posible.

2.1.2. Alcance

El producto a desarrollar es un sistema de información (desde ahora SI) que plasme de la manera más fiel el funcionamiento de la empresa.
Se diseñará una base de datos que represente claramente la información usada por la empresa, y se desarrollará una interfaz web que permita acceder a esta información de forma sencilla e intuitiva, así como editar, eliminar y añadir más información al SI.

2.1.3. Apreciación global

Una vez finalizado, el producto consistirá en una aplicación web que permitirá realizar el conjunto de tareas CRUD sobre la base de datos diseñada, además de ofrecer otras facilidades como estadísticas basadas en datos históricos o informes de alertas varias.

2.2. Descripción global

2.2.1. Perspectiva del producto

El producto se divide en 5 módulos básicos de gestión. Cada uno de los módulos representa un área de negocio del cliente en la que se agrupan las tareas CRUD básicas más algunas tareas propias de cada módulo.
Los módulos a implementar son:

Módulo Vehículos
Gestiona toda la información relacionada con los vehículos. Permite dar de alta nuevos vehículos, asignarles un contrato, modificar datos de la ficha del vehículo, consultar datos asociados a un vehículo en concreto, como gastos derivados de su contrato, los propios datos de la contratación... etc.

Módulo Clientes
Mantiene toda la información relativa a los clientes, así como los datos principales de contacto.

Módulo Contratos

Los contratos unen vehículos con clientes, y definen el periodo de duración de esta unión, así como las condiciones bajo las que se presta un vehículo a un cliente.

Módulo Gastos
Desde el módulo de gastos se gestionan todos los desembolsos llevados a cabo por la empresa para el mantenimiento de cada vehículo y cada contrato.

Módulo Alertas
Este módulo se nutre de la información del resto de los módulos para generar una serie de alertas o avisos que pueden resultar de interés para el usuario encargado de la gestión, como revisiones obligatorias de vehículos cercanas o caducadas, finalizaciones próximas de contratos... etc.
Al contrario que el resto de módulos, este no permite realizar tareas de inserción ni modificación, sólo de acceso a datos.

2.2.2. Funciones del producto
Una vez finalizado, el producto será capaz de ofrecer a los usuarios una gestión completa de su negocio.

2.2.2.1 Funciones Comunes
Como antes se ha comentado, el producto se divide en 5 módulos, cada uno de ellos, a excepción del módulo de Alertas, ofrece las siguientes funcionalidades

Búsqueda de registros
Cada uno de los módulos ofrece como vista principal una lista tabular paginada de registros guardados. Dado que la cantidad de registros puede llegar a ser bastante elevada, una funcionalidad básica que ofrece el producto es el filtrado de resultados en base a unos parámetros (customizados para cada módulo) que el propio usuario define, de manera que los resultados mostrados en la lista, sólo sean resultados que cumplan con el filtro.

Nuevo registro
Otra de las funciones comunes a todos los módulos es dar de alta nuevos registros. Mediante un sencillo formulario (customizado para cada módulo con los campos necesarios), el usuario puede dar de alta nuevos registros que pasarán a la base de datos y serán accesibles para todos los usuarios.

Editar un registro
El producto ofrece también la posibilidad de modificar los registros que ya han sido guardados previamente. Al igual que en el caso de dar de alta nuevos registros, se realiza a través de un sencillo formulario adaptado para cada módulo.

Eliminar un registro

Como última funcionalidad común, los usuarios pueden eliminar registros que ya han dejado de ser útiles en el sistema. El borrado puede realizarse individualmente, registro a registro o seleccionando un conjunto de registros y eliminandolos en grupo.

2.2.2.2 Funciones específicas del módulo de Vehículos

Asociar vehículos a un contrato
Una opción específica de los vehículos es asociar uno o varios de ellos a un mismo contrato.

Estadísticas
El producto ofrece una serie de estadísticas mostradas en gráficos que resultan útiles para la redacción de informes comerciales o la toma de decisiones.
Los gráficos pueden ser exportados sencillamente a archivos PNG, además de ser interactivos permitiendo ocultar información a voluntad del usuario.

2.2.2.3 Funciones específicas del módulo de Gastos

Estadísticas
El producto ofrece una serie de estadísticas mostradas en gráficos que resultan útiles para la redacción de informes comerciales o la toma de decisiones.
Los gráficos pueden ser exportados sencillamente a archivos PNG, además de ser interactivos permitiendo ocultar información a voluntad del usuario.

2.2.2.4 Funciones específicas del módulo de Contratos

Marcar como finalizados
Existe la posibilidad de marcar una serie de contratos como “Finalizados”, resulta útil para rescindir un contrato antes de que se haga naturalmente por caducidad de fecha.

Estadísticas
El producto ofrece una serie de estadísticas mostradas en gráficos que resultan útiles para la redacción de informes comerciales o la toma de decisiones.
Los gráficos pueden ser exportados sencillamente a archivos PNG, además de ser interactivos permitiendo ocultar información a voluntad del usuario.

2.2.2.5 Funciones específicas de Alertas

Alertas de ITV
Realiza un listado de los vehículos cuya revisión ITV ha caducado y de a los que les caducará próximamente, solo cuando su contrato no incluye mantenimiento.
Las revisiones caducadas se alertan en color rojo.
Las revisiones a las que les quedan menos de 2 meses para caducar, se alertan en amarillo.

Alertas Semestrales
Realiza un listado de los vehículos cuya revisión semestral ha caducado y de a los que les caducará próximamente, solo cuando su contrato no incluye mantenimiento.
Las revisiones caducadas se alertan en color rojo.
Las revisiones a las que les quedan menos de 2 meses para caducar, se alertan en amarillo.

Alertas de Termógrafos
Realiza un listado de los vehículos frigoríficos cuya revisión de termógrafos ha caducado y de a los que les caducará próximamente, solo cuando su contrato no incluye mantenimiento.
Las revisiones caducadas se alertan en color rojo.
Las revisiones a las que les quedan menos de 1 año para caducar, se alertan en amarillo.

Alertas de ATP
Realiza un listado de los vehículos frigoríficos cuya revisión de ATP ha caducado y de a los que les caducará próximamente, solo cuando su contrato no incluye mantenimiento.
Las revisiones caducadas se alertan en color rojo.
Las revisiones a las que les quedan menos de 1 año para caducar, se alertan en amarillo.

2.2.3. Características del usuario
El usuario objetivo de esta aplicación es exclusivamente el personal administrativo de *Paul Günther*. Únicamente debe conocer el modelo de trabajo propio de la empresa.

2.2.4. Restricciones

2.2.4.1 Seguridad y consideraciones de seguridad
La aplicación solo puede ser accedida por personal previamente autorizado por *Paul Günther*.
Los usuarios pueden ser definidos previamente por un usuario administrador en el propio sistema.

Las contraseñas se guardan en la base de datos codificadas en SHA-512 con 5000 iteraciones del algoritmo, usando como entrada para cada iteración el resultado de la anterior y codificando el resultado final en BASE64.

Los datos de los clientes se encuentran cifrados en AES con una frase de paso de 1024 bits, para que en caso de obtener la base de datos, un atacante no pueda acceder a la información de los clientes.

2.2.5. Atención y dependencias

2.2.5.1 Parte servidor
El servidor que albergue la aplicación deberá de disponer de un servidor web instalado previamente con soporte para PHP5 y, con un sistema gestor de base de datos, también accesible por el servidor web.

Es altamente recomendable disponer de un acelerador de PHP, tal como APC, ya que mejora notablemente el rendimiento de la aplicación.

La configuración recomendada es una arquitectura LAMP, es decir, un servidor con sistema operativo basado en Linux, con un servidor web Apache, base de datos MySQL e intérprete de PHP.

2.2.5.2 Parte cliente
En la parte cliente, solo es necesario contar con un navegador compatible con el estándar HTML5 y el estándar CSS3.

Se recomienda el uso de navegadores que cumplan correctamente con estos estándares, como Google Chrome o Mozilla Firefox en sus versiones más recientes.

2.3. Requisitos específicos

2.3.1 Requisitos del módulo Vehículos

2.3.1.1 Alta de un vehículo
Para dar de alta un nuevo vehículo, es imprescindible definir como mínimo el valor valor de los campos:

- No. de Flota
- Matrícula
- Modelo
- Bastidor

El campo matrícula además deberá de contener una matrícula válida con el formato europeo de matrícula, esto significa cumplir con la máscara:

DDDD-LLL

donde “D” representa un dígito cualquiera del 0 al 9, y “L” una consonante cualquiera del alfabeto internacional.

De la misma manera, el número de bastidor debe de cumplir el estándar internacional VIN (Vehicle Identification Number), compuesto por una cadena de 17 caracteres que no incluya las letras I, O, Q y Ñ.

Al guardar el nuevo vehículo, automáticamente se le asigna el valor de la fecha actual a los campos:

- Última ITV
- Última Semestral
- Última ATP
- Última Termógrafos

2.3.1.2 Edición de los datos de un vehículo
Cuando se editen los datos de un vehículo previamente dado de alta, se deben de cumplir las mismas restricciones que cuando se da de alta un vehículo nuevo.

A diferencia de en el alta del vehículo, en la edición si se podrá cambiar las fechas de las revisiones, y estas deben de cumplir el formato de fecha francesa y ser válidas. Para facilitar la introducción de estas fechas, el usuario deberá de disponer de un calendario que le permita seleccionar con facilidad la fecha deseada.

La ficha de edición de un vehículo, debe de mostrar además un listado de los gastos asociados a dicho vehículo, no siendo editables estos datos, pero dando la opción de acceder a la ficha el gasto para realizar las modificaciones deseadas.

2.3.1.3 Filtrado del listado de vehículos

Al realizar una búsqueda mediante los filtros, la lista resultante deberá incluir únicamente elementos que coincidan con todos los parámetros definidos en los filtros.

Dicho de otra forma, la operación que encadena los filtros deberá ser un AND, no un OR.

2.3.1.4 Asociar un contrato a varios vehículos

Para realizar la asociación de un conjunto de vehículos a un mismo contrato, estos deben de estar previamente seleccionados mediante un control de tipo checkbox o similar.

En caso de no haber ningún elemento seleccionado, el sistema debe advertir al usuario que la operación no puede ser llevada a cabo.

Si solamente hay seleccionado un vehículo, la acción debe realizarse igualmente.

2.3.1.5 Eliminar un vehículo

Al eliminar un vehículo, los contratos que tengan asociado el vehículo en cuestión deberán ser marcados automáticamente como *Finalizados* después de eliminarse el vehículo.

Los gastos asociados al vehículo no se eliminarán, pero dejarán de referenciarse con el vehículo eliminado, quedando referenciado únicamente por el contrato.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.1.6 Eliminar un conjunto de vehículos

Al eliminar un conjunto de vehículos, se deben de cumplir las restricciones mencionadas en el apartado 2.3.1.5, además de haberse seleccionado previamente los vehículos a eliminar mediante un control de tipo checkbox o similar.

En caso de no haber ningún elemento seleccionado, el sistema debe advertir al usuario que la operación no puede ser llevada a cabo.

Si solamente hay seleccionado un vehículo, la acción debe realizarse igualmente.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.2 Requisitos del módulo de Clientes

2.3.2.1 Alta de un nuevo cliente

Para dar de alta un nuevo cliente, es imprescindible definir como mínimo el valor valor de los campos:

- No. de Cliente
- Razón Social
- Persona de contacto

Si se rellena la información de contacto, se incluyen restricciones en los siguientes campos:

- Teléfono
- Email
- Sitio Web

El teléfono debe ser un número de teléfono válido.

El email debe de cumplir con el formato válido de email, además, se realizará una consulta DNS para consultar que el dominio especificado en la dirección exista y posea registros MX.

El sitio web debe de ser una URL válida, incluyendo el protocolo, que debe ser http o https.

2.3.2.2 Edición de los datos de un cliente

Cuando se editen los datos de un cliente, todas las restricciones definidas en el apartado 2.3.2.1 deben de cumplirse igualmente.

La ficha de edición de un cliente debe de mostrar además, un listado con los contratos vigentes y los vehículos contratados, no pudiendo editar ninguno de estos listados.

2.3.2.3 Filtrado del listado de clientes

Al realizar una búsqueda mediante los filtros, la lista resultante deberá incluir únicamente elementos que coincidan con todos los parámetros definidos en los filtros.

Dicho de otra forma, la operación que encadena los filtros deberá ser un AND, no un OR.

2.3.2.4 Eliminar un cliente

Cuando un cliente es eliminado, los contratos que tuviese asociados y permanecieran aún vigentes, serán marcados como finalizados, y los vehículos asociados por tanto no tendrán asociado ya ningún contrato.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.2.5 Eliminar un conjunto de clientes

Al eliminar un conjunto de clientes, se deben de cumplir las restricciones mencionadas en el apartado 2.3.2.4, además de haberse seleccionado previamente los clientes a eliminar mediante un control de tipo checkbox o similar.

En caso de no haber ningún elemento seleccionado, el sistema debe advertir al usuario que la operación no puede ser llevada a cabo.

Si solamente hay seleccionado un cliente, la acción debe realizarse igualmente.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.3 Requisitos del módulo de Gastos

2.3.3.1 Alta de un nuevo gasto

Para dar de alta un nuevo gasto, es imprescindible definir como mínimo el valor valor de los campos:

- Fecha
- Categoría
- Vehículo
- Importe
- Importe Facturado al cliente
- Descripción

Si se rellena la información adicional, se puede añadir información como

- Kilómetros (lectura del cuentakilómetros)
- Observaciones (impresiones personales del agente que da de alta el gasto)
- Foto (testigo gráfico de los posible daños del vehículo)

La fecha debe de seguir un formato de fecha francesa y ser además válida.
El importe debe de ser un número positivo en todos los casos.
El importe facturado al cliente puede ser un número negativo, indicando un reembolso al cliente.

Para facilitar al usuario la introducción del vehículo, se le mostrará un listado de los vehículos que actualmente tienen un contrato todavía vigente.

La foto adjunta debe de tener un formato gráfico válido (jpg, png, gif) y no tener un tamaño mayor a 1MB.

Al guardarse el gasto, automáticamente se guardará el contrato al que se asociará el gasto, consultando previamente el contrato que tiene activo el vehículo seleccionado.

2.3.3.2 Edición de los datos de un gasto

Cuando se editen los datos de un gasto, todas las restricciones definidas en el apartado 2.3.3.1 deben de cumplirse igualmente.

Si una foto se hubiese guardado previamente, deberá de mostrar una miniatura de la misma, permitiendo verla ampliada y también eliminarla. Al eliminarla, se ofrecerá de nuevo la posibilidad de subir una foto, con las mismas restricciones que en el punto 2.3.3.1.

2.3.2.3 Filtrado del listado de gastos

Al realizar una búsqueda mediante los filtros, la lista resultante deberá incluir únicamente elementos que coincidan con todos los parámetros definidos en los filtros.

Dicho de otra forma, la operación que encadena los filtros deberá ser un AND, no un OR.

2.3.2.4 Eliminar un gasto

Cuando un gasto es eliminado no se realiza ninguna tarea extra.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.2.5 Eliminar un conjunto de gastos

Al eliminar un conjunto de gastos, se deben de cumplir las restricciones mencionadas en el apartado 2.3.3.4, además de haberse seleccionado previamente los gastos a eliminar mediante un control de tipo checkbox o similar.

En caso de no haber ningún elemento seleccionado, el sistema debe advertir al usuario que la operación no puede ser llevada a cabo.

Si solamente hay seleccionado un gasto, la acción debe realizarse igualmente.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.4 Requisitos del módulo de Contratos

2.3.4.1 Alta de un nuevo contrato

Para dar de alta un nuevo contrato, es imprescindible definir como mínimo el valor valor de los campos:

- No. de Contrato
- Cliente
- Tipo de contrato
- Fecha de Inicio
- Fecha fin

Si se rellena la información adicional, se puede añadir información como

- Kilómetros (cantidad de kilómetros al año para la que se contrata)
- Con ruedas (el contrato incluye mantenimiento de ruedas)
- Con mecánica (el contrato incluye mantenimiento mecánico)

Las fechas deben de seguir un formato de fecha francesa y ser además válidas.

El cliente se seleccionará de un desplegable para evitar errores.

El tipo de contrato se elegirá de un desplegable, si marca las opciones “Con Mantenimiento” o “Full Service”, automáticamente se marcarán también los checkbox de “Con ruedas” y “Con mecánica” del detalle del contrato.

El número de kilómetros debe ser obligatoriamente un número positivo.

2.3.4.2 Edición de los datos de un contrato

Cuando se editen los datos de un contrato, todas las restricciones definidas en el apartado 2.3.4.1 deben de cumplirse igualmente.

En la ficha del contrato, debe de mostrarse además un listado con los vehículos asociados.
Se podrá también asociar nuevos vehículos al contrato, seleccionandolos y añadiéndolos al contrato.

2.3.4.3 Asociar nuevos vehículos a un contrato

Para asociar nuevos vehículos a un contrato, desde la ficha debe de mostrarse un listado de vehículos sin contrato vigente, poder seleccionar los que se desee añadir al contrato y añadirlos mediante un click en un botón.

Al hacer esto, los vehículos seleccionados quedarán asociados al contrato en cuestión.

2.3.4.4 Filtrado del listado de contratos

Al realizar una búsqueda mediante los filtros, la lista resultante deberá incluir únicamente elementos que coincidan con todos los parámetros definidos en los filtros.

Dicho de otra forma, la operación que encadena los filtros deberá ser un AND, no un OR.

2.3.4.5 Marcar como finalizados un conjunto de contratos

Desde la pantalla de búsqueda de contratos se debe de poder seleccionar un conjunto de contratos mediante un control de tipo checkbox o similar y marcarlos como finalizados.

Antes de realizar la acción, el sistema deberá preguntar al usuario si realmente desea realizar la acción. En caso positivo, los contratos quedarían marcados como finalizados, y los vehículos asociados a dichos contratos, quedarían desvinculados de los mismos.

2.3.4.6 Eliminar un contrato

Cuando un contrato es eliminado los gastos asociados a dicho contrato quedarán únicamente referenciados por el vehículo al que fueron asociados. Asimismo, los vehículos asociados a dicho contrato quedarían desvinculados del mismo.

El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

2.3.4.7 Eliminar un conjunto de contratos

Al eliminar un conjunto de contratos, se deben de cumplir las restricciones mencionadas en el apartado 2.3.4.6, además de haberse seleccionado previamente los gastos a eliminar mediante un control de tipo checkbox o similar.

En caso de no haber ningún elemento seleccionado, el sistema debe advertir al usuario que la operación no puede ser llevada a cabo.

Si solamente hay seleccionado un gasto, la acción debe realizarse igualmente.
El sistema deberá preguntar al usuario si realmente desea eliminar el registro para evitar posibles errores.

3. Análisis

Durante la fase de análisis de una aplicación, el analista determina cómo se interrelacionan todos los componentes de la aplicación, así como las entradas y salidas que se obtienen en las distintas llamadas a procedimientos o funciones.

Para facilitar la labor del analista este se apoya en una serie de diagramas que permiten ver de una forma mucho más visual y atractiva las relaciones entre componentes, las funcionalidades definidas y los casos de uso, la definición de los datos manejados... etc.

Otro gran apoyo para el analista son las herramientas de diseño de software asistido por computador (desde ahora *Herramientas CASE)*, que permiten realizar todo este tipo de documentación de una manera muy sencilla.

Muchas de estas herramientas son editores de UML (*Unified Modeling Language)*, un lenguaje gráfico de modelado de software muy extendido y declarado estándar en 2005.

Entre todos los diagramas que permite realizar UML se han usado dos en concreto para definir la interacción entre los objetos de la aplicación y el conjunto de funcionalidades que pueden ser realizadas por cada usuario según su rol.

Aunque no pertenezca al lenguaje UML, otro diagrama que también se ha usado para analizar esta aplicación ha sido el diagrama *Entidad-Relación* que define las conexiones entre tablas de una base de datos y los atributos de las mismas.

3.1. Diagrama de clases

El diagrama de clases es un gráfico que representa las distintas clases que componen un software o aplicación y cómo se relacionan entre ellas.
En el diagrama se representan las clases, sus atributos y sus métodos. También se representa el ámbito de los métodos y de los atributos (públicos, privados, protegidos...) .

También quedan definidas las relaciones de herencia y especialización de las clases entre ellas.
Este tipo de documento es muy usado sobretodo cuando el desarrollo se basa en el paradigma de *Orientación a Objetos*, como es el caso de la aplicación objeto de este documento.


TiposContratos
- id : int
- nombre : string
+ getId() : int
+ setNombre(nombre : string) : string
+ getNombre() : string
-tipoContrato
Contrato
- id : int
- cliente : Cliente
- numContrato : string
- fechaInicio : DateTime
- fechaFin : DateTime
- tipoContrato : TiposContratos
- kilometros : double
- conRuedas : bool
- conMecanica : bool
+ getId() : int
+ setCliente(cliente : Cliente) : Cliente
+ getCliente() : Cliente
+ setNumContrato(numContrato : string) : string
+ getNumContrato() : string
+ setFechaInicio(fechaInicio : DateTime) : DateTime
+ getFechaInicio() : DateTime
+ setFechaFin(fechaFin : DateTime) : DateTime
+ getFechaFin() : DateTime
+ setTipoContrato(tipoContrato : TiposContratos) : TiposContratos
+ getTipoContrato() : TiposContratos
+ setKilometros(kilometros : double) : double
+ getKilometros() : double
+ setConRuedas(conRuedas : bool) : bool
+ getConRuedas() : bool
+ setConMecanica(conMecanica : bool) : bool
+ getConMecanica() : bool
-contrato
Vehiculo
- id : int
- numFlota : string
- contrato : Contrato
- modelo : string
- bastidor : string
- matricula : string
- constructor : string
- seguro : bool
- depot : string
- site : string
- financiera : string
- ultimaITV : DateTime
- ultimaSemestral : DateTime
- ultimaATP : DateTime
- ultimaTermografos : DateTime
+ getID() : int
+ setNumFlota(numFlota : string) : Vehiculo
+ getNumFlota() : string
+ setContrato(contrato : Contrato) : Contrato
+ getContrato() : Contrato
+ setModelo(modelo : string) : string
+ getModelo() : string
+ setBastidor(bastidor : string) : string
+ getBastidor() : string
+ setMatricula(matricula : string) : string
+ getMatricula() : string
+ setConstructor(constructor : string) : string
+ getConstructor() : string
+ setSeguro(seguro : bool) : bool
+ getSeguro() : bool
+ setDepot(depot : string) : string
+ getDepot() : string
+ setSite(site : string) : string
+ getSite() : string
+ setFinanciera(financiera : string) : string
+ getFinanciera() : string
+ setUltimaITV(ultimaITV : DateTime) : DateTime
+ getUltimaITV() : DateTime
+ setUltimaSemestral(ultimaSemestral : DateTime) : DateTime
+ getUltimSemestral() : DateTime
+ setUltimaATP(ultimaATP : DateTime) : DateTime
+ getUltimaATP() : DateTime
+ setUltimaTermografos(ultimaTermografos : DateTime) : DateTime
+ getUltimaTermografos() : DateTime
-vehiculo
Usuario
- id : int
- nombre : string
- apellidos : string
- email : string
- password : string
- salt : string
- role : string
- activo : bool
- foto : string
+ getId() : int
+ setNombre(nombre : string) : string
+ getNombre() : string
+ setApellidos(apellidos : string) : string
+ getApellidos() : string
+ setEmail(email : string) : string
+ getEmail() : string
+ setPassword(password : string) : string
+ getPassword() : string
+ setSalt(salt : string) : string
+ getSalt() : string
+ setRole(role : string) : string
+ getRole() : string
+ setActivo(activo : bool) : bool
+ getActivo() : bool
+ setFoto(foto : string) : string
+ getFoto() : string
-cliente
Cliente
- id : int
- numCliente : string
- razonSocial : string
- direccion : string
- email : string
- telefono : string
- web : string
- personaContacto : string
+ getId() : int
+ setNumCliente(numCliente : string) : string
+ getNumCliente() : string
+ setRazonSocial(razonSocial : string) : string
+ getRazonSocial() : string
+ setDireccion(direccion : string) : string
+ getDireccion() : string
+ setEmail(email : string) : string
+ getEmail() : string
+ setTelefono(telefono : string) : string
+ getTelefono() : string
+ setWeb(web : string) : string
+ getWeb() : string
+ setPersonaContacto(personaContacto : string) : string
+ getPersonaContacto() : string
Gasto
- id : int
- vehiculo : Vehiculo
- contrato : Contrato
- categoria : CategoriaGasto
- descripcion : string
- observaciones : string
- importe : double
- importeFacturadoCliente : double
- fecha : DateTime
- foto : string
- kilometros : double
+ getId() : int
+ setVehiculo(vehiculo : Vehiculo) : Vehiculo
+ getVehiculo() : Vehiculo
+ setContrato(contrato : Contrato) : Contrato
+ getContrato() : Contrato
+ setCategoriaGasto(categoria : CategoriaGasto) : CategoriaGasto
+ getCategoriaGasto() : CategoriaGasto
+ setDescripcion(descripcion : string) : string
+ getDescription() : string
+ setObservaciones(observaciones : string) : string
+ getObservaciones() : string
+ setImporte(importe : double) : double
+ getImporte() : double
+ setImporteFacturadoCliente(importe : double) : double
+ getImporteFacturadoCliente() : double
+ setFecha(fecha : DateTime) : DateTime
+ getFecha() : DateTime
+ setFoto(foto : string) : string
+ getFoto() : string
+ setKilometros(kilometros : double) : double
+ getKilometros() : double
-categoria
CategoriaGasto
- id : int
- nombre : string
+ getId() : int
+ setNombre(nombre : string) : string

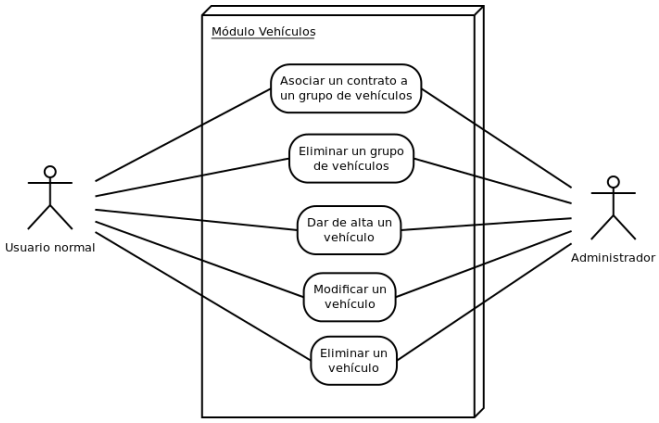
3.2. Casos de Uso

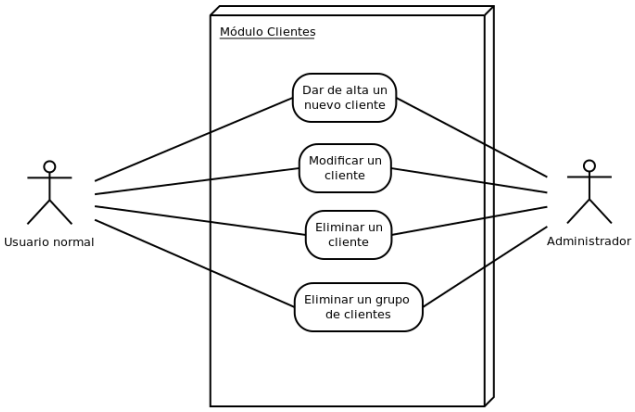
El diagrama de casos de uso es también uno de los esquemas más utilizados para definir qué funcionalidades tendrá la aplicación y quién podrá usar dichas funcionalidades.

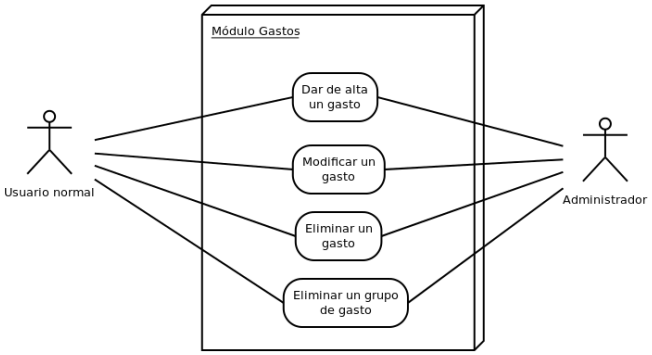
Los distintos roles de usuarios son representados por “Actores”, en el diagrama, puede haber tantos actores como tipos de usuario haya en el sistema que se está diseñando/analizando.

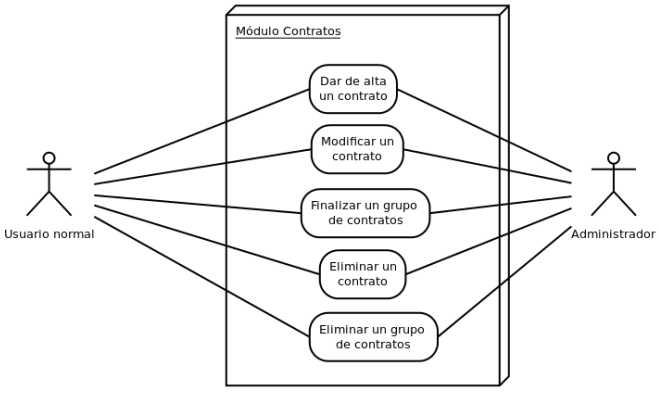
Los casos de uso pueden tener dependencia de unos entre otros, y esto también es posible expresarlo en el diagrama.

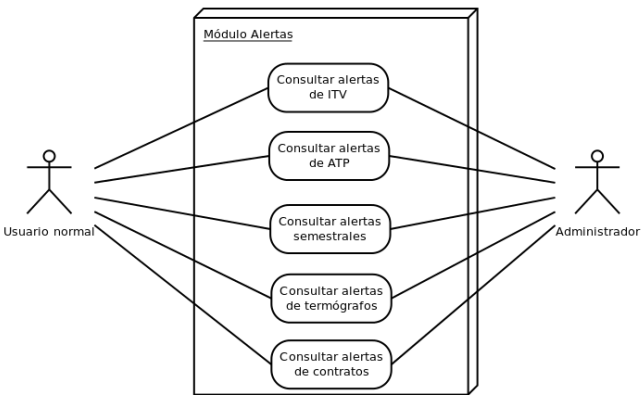
Usualmente, estos diagramas escapan del lenguaje técnico para usar un lenguaje más próximo al del cliente o consumidor final.

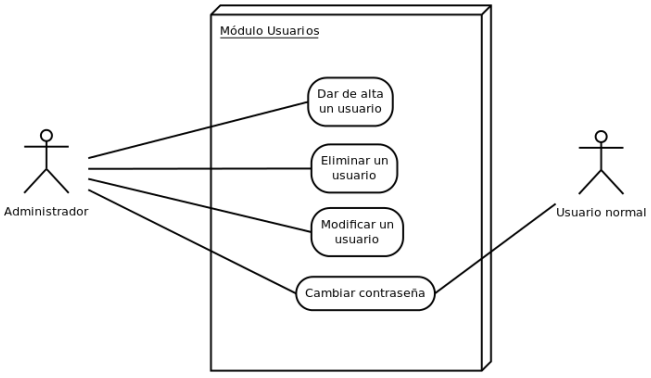












Otra forma de presentar los casos de uso

Los casos de uso también pueden ser documentados con una lista numerada de los pasos que sigue el actor para interactuar con el sistema.

A continuación se mostrarán algunos de los casos de uso anteriores en este nuevo formato propuesto.

Caso de uso: Asociar un contrato a un grupo de vehículos	
Actor: Usuario Normal	
Curso Normal	**Alternativas**
1) El usuario accede al listado de vehículos	1.1 Si no ha iniciado sesión se le pedirá que la inicie para acceder al listado
2) El usuario realiza un filtrado de la tabla con tal de conseguir obtener el conjunto de vehículos deseado	2.1 Si no hay resultados debe de probar otros criterios de filtrado.
3) Seleccionar mediante el selector situado en la primera columna de la tabla los vehículos deseados.	3. 1 También puede usarse el selector de "Seleccionar Todos" situado al pie de la tabla.
4) Seleccionar en el selector de contratos del pie de la tabla el número de contrato al que se quieren asociar los vehículos	
5) Hacer click sobre "Asociar estos vehículos al contrato"	

Caso de uso: Dar de alta un vehículo	
Actor: Usuario Normal	
Curso Normal	**Alternativas**
1) El usuario accede al formulario de alta de vehículos	1.1 Si no ha iniciado sesión se le pedirá que la inicie para acceder al listado
2) El usuario rellena los datos del vehículo mediante el formulario.	2.1 Puede reiniciar el formulario con el botón de "Restablecer" situado al pie del formulario
3) Hacer click sobre el botón "Guardar"	3.1 Puede darse el caso de que el usuario no haya rellenado todos los campos obligatorios. En este caso se informará de los campos faltantes que el usuario debe rellenar obligatoriamente.
4) El usuario es redirigido a la ventana de edición del registro recién insertado.	4.1 El usuario puede modificar los datos de nuevo, además aparecen dos nuevos campos de fecha que se han rellenado automáticamente: "Ultima ITV" y "Última Semestral"

Caso de uso: Eliminar uno o más vehículos	
Actor: Usuario Normal	
Curso Normal	**Alternativas**
1) El usuario accede listado de vehículos	1.1 Si no ha iniciado sesión se le pedirá que la inicie para acceder al listado
2) El usuario realiza un filtrado de la tabla con tal de conseguir obtener el vehículo o el conjunto de vehículos deseado	2.1 Si no hay resultados debe de probar otros criterios de filtrado.
3) Si quiere eliminar un conjunto de vehículos puede seleccionar el conjunto y eliminarlos mediante el comando “Eliminar seleccionados”, si desea eliminar un único vehículo, puede hacerlo con el botón de eliminar situado en la última celda de la fila.	3.1 Puede darse el caso de que el vehículo tenga contratos asociados, en cuyo caso, el sistema devolverá una advertencia y pedirá confirmación. En caso positivo, todos los contratos asociados se anularán.
4) El usuario recibe la confirmación del borrado.	4.1 Si se produce algún error, se le informará mediante un mensaje emergente.

3.3 Diagrama Entidad-Relación

El diagrama Entidad-Relación es el tercero de los diagramas usados en el desarrollo de la aplicación objeto de este documento.

Este tipo de diagrama se encarga de definir las tablas de las que se compone la base de datos usada y las relaciones que existen entre ellas.

En el diagrama Entidad-Relación, cada tabla se representa como una entidad con sus atributos, marcando debidamente los que sean clave primaria o foránea.

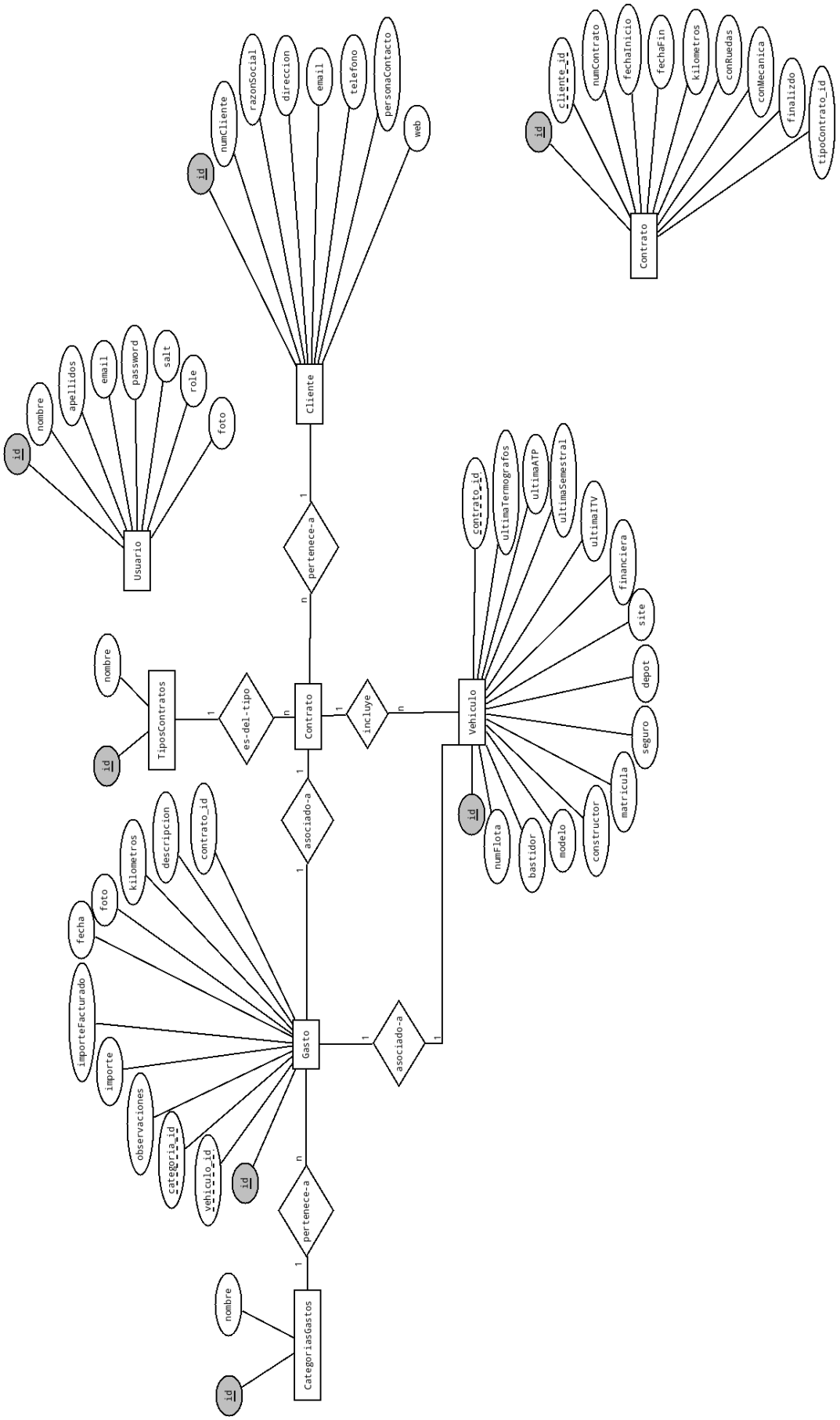
Hablamos de clave primaria cuando se trata de un atributo que identifica de manera unívoca en una tabla a un elemento en concreto. Suelen usarse índices autonuméricos para estos menesteres.
En muchos casos puede llegar a ser necesario identificar con más de un atributo a un elemento de una tabla, creando así claves combinadas o superclaves.

Una clave foránea en una tabla A es aquella se usa para referenciar a un elemento de otra tabla B que tiene una relación directa con un elemento de la tabla A.
Por ejemplo, en la tabla Vehículo existe un atributo *contrato_id* que relaciona el vehículo con un contrato.

Uno de los puntos fuertes de este diagrama es resaltar las relaciones existentes entre distintas tablas, indicando la cardinalidad de las mismas y la existencia de atributos derivados de las relaciones muchos a muchos.

Los tipos de relaciones que pueden darse entre entidades son:

- Uno a uno (1...1)
    - Una entidad de A se relaciona con una única entidad de B.
    - Por ejemplo, un gasto se asocia a un único contrato.
- Uno a muchos (1...n)
    - Una entidad de A se relaciona con más de una entidad de B.
    - Por ejemplo, un cliente puede tener uno o más contratos.
- Muchos a uno (n...1)
    - Varias entidades de A se relacionan con una entidad de B
    - Por ejemplo, varios vehículos pueden asociarse a un mismo contrato.
- Muchos a muchos (n...m)
    - Varias entidades de A se relacionan con 0 o más entidades de B y viceversa.


Contrato
id
cliente_id
numContrato
fechaInicio
fechaFin
kilometros
conRuedas
conMecanica
finalizdo
tipoContrato_id
Cliente
id
numCliente
razonSocial
direccion
email
telefono
personaContacto
web
Usuario
id
nombre
apellidos
email
password
salt
role
foto
pertenece-a
1
n
TiposContratos
id
nombre
es-del-tipo
1
n
Contrato
1
incluye
n
Vehiculo
id
numFlota
bastidor
modelo
constructor
matricula
seguro
depot
site
financiera
ultimaITV
ultimaSemestral
ultimaATP
ultimaTermografos
contrato_id
asociado-a
1
1
Gasto
id
importeFacturado
importe
observaciones
categoria_id
vehiculo_id
fecha
foto
kilometros
descripcion
contrato_id
asociado-a
1
1
pertenece-a
1
n
CategoriasGastos
id
nombre

4. Diseño

El diseño de la aplicación ha seguido el patrón de desarrollo *Modelo Vista Controlador* (desde ahora MVC).

El MVC es un patrón o modelo de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos.

Esta separación implica que cualquier petición al sistema sea tratada por el controlador, que obtendrá la información necesaria del modelo, le pasará a la vista adecuada esta información para realizar una presentación consistente que será devuelta como respuesta, uniendo así los datos con la presentación.

El siguiente esquema resume gráficamente lo anteriormente comentado.

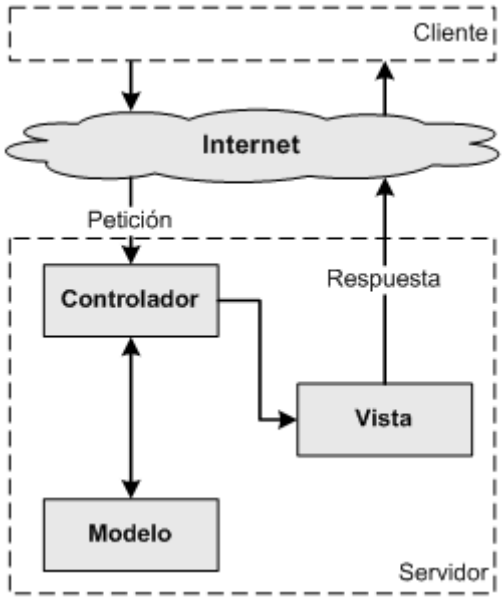


Figura 4.1 - Modelo MVC

En el caso de una aplicación web, la vista es la página HTML y el código provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la lógica de negocio, y el controlador es el responsable de los eventos de entrada desde la vista.

Este patrón de diseño aporta mucha sencillez a los desarrollos de cierto tamaño, ya que permite trabajar separando claramente los tres conceptos antes mencionados, de manera que nunca se mezcla la lógica de negocio con la presentación, obteniendo así una interfaz de usuario limpia y fácilmente sustituible.

Para el desarrollo de *FleetManager* se ha utilizado un framework de desarrollo en PHP llamado *Symfony2*, que está diseñado para utilizar el patrón MVC en todos los proyectos que se realicen con él.

En los próximos capítulos se verá en mayor profundidad los tres componentes del MVC.

4.1. Capa de presentación. La vista

La capa de presentación o vista es la parte más superficial de la aplicación, es el interfaz en el que se apoya el usuario para interactuar con el sistema que hay por debajo.

La presentación de la aplicación es un aspecto muy importante, ya que un interfaz atractivo y sencillo de utilizar atraerá más al usuario y le permitirá realizar sus tareas de forma fácil.

4.1.1 Usabilidad

Un concepto muy a tener en cuenta es la llamada "usabilidad" del diseño. Una interfaz con una buena usabilidad es aquella que se desprende de todo lo innecesario, dejando a la vista y con un fácil acceso, sólo los elementos que el usuario puede llegar a necesitar.

Estos interfaces maximizan la productividad del usuario en la aplicación y convierten el tiempo invertido frente a la misma en una actividad menos pesada y tediosa.

Una regla no oficial del diseño web es la llamada **regla de los tres clicks**. Esta regla sugiere que todo usuario debería de poder acceder a toda la información que desee consultar en la aplicación en un máximo de 3 clicks desde la página de inicio.

Es un principio bastante popular y seguido, el mismo Steve Jobs hizo uso de esta regla durante el diseño del iPod en 2004.

4.1.2 Diseño adaptable

Otro de los estandartes de los interfaces modernos es la adaptabilidad del diseño.
Tiempo atrás solo existían pantallas de determinados tamaños: 15", 17",19"…
Con el paso del tiempo, esto ha cambiado muchísimo, las pantallas ahora pueden ser de 3", 7", 10", panorámicas…
La diversidad de dispositivos está crea la necesidad de no generar más diseños estáticos que sean inservibles ante determinados tamaños de pantalla.
Ahora la adaptabilidad de un entorno a los distintos tamaños de pantalla es la diferencia entre un buen interfaz y uno mediocre.

4.1.3 Tecnologías

En el caso de las aplicaciones web y más concretamente en la capa de presentación, las tecnologías más habituales son HTML + CSS.

Actualmente, el estándar HTML se encuentra en su el borrador de su versión 5, a pesar de que aún no es definitiva, su uso ya es muy extendido dadas las nuevas prestaciones que esta aporta respecto a otros estándares como XHTML.

Por su parte, CSS se encuentra en su versión 3, con grandes mejoras respecto a su sucesora, la 2.1, como degradados, bordes redondeados, animaciones…

Un único punto negativo a todo esto es que no todos los navegadores han sabido implementar estas características, o lo han hecho a su manera y no son compatibles con el estándar, por lo que a veces se vuelve impredecible el resultado que tendrá ver una misma interfaz en dos navegadores distintos.

El desarrollo de esta aplicación se realiza con Symfony2, que incluye el motor de plantillas Twig, el cual permite usar un esquema de herencia de plantillas de tres niveles, lo cual facilita muchísimo el desarrollo y los posteriores cambios. Existen otros motores de plantillas, como Smarty o Haanga.

En el capítulo 5 de este documento se tratará en mayor profundidad el tema de los motores de plantillas, concretamente Twig.

4.2. Capa de negocio. El controlador

El centro neurálgico de toda aplicación basada en el patrón MVC es sin duda la capa de negocio, mejor conocida como *controlador*.

El controlador es la parte de la aplicación que se encarga de gestionar el acceso a los datos, si es necesario, consultando directamente la base de datos y obteniendo lo que necesita para pasarlo a la vista y componer una respuesta válida a la petición.

Aunque se suele referir al controlador en singular, esto no es del todo cierto, ya que una aplicación no tiene porqué limitarse a un único controlador. De hecho, una misma aplicación puede tener tantos controladores como desee si es que le son necesarios. Esto puede venir bien en aras de la organización del código y distribución de las funciones de una manera que el diseñador de la aplicación considere oportuna.

En el caso de la aplicación objeto de este documento, cuenta con 6 controladores distintos:

- Controlador del módulo de vehículos
- Controlador del módulo de clientes
- Controlador del módulo de gastos
- Controlador del módulo de contratos
- Controlador del módulo de alertas
- Controlador del contenido estático

Más adelante en el capítulo 5 se verá como Symfony2 se encarga de gestionar los controladores y el porqué de este número de controladores.

4.3. Capa de persistencia. El modelo

Todas las aplicaciones necesitan guardar información, de manera que una vez finalicen, esté disponible en un nuevo arranque.
Esto se conoce como persistencia de la información, y en el modelo MVC es la capa del modelo la que se encarga de esto.

Un modelo es un esquema de base de datos, no importa el Sistema de Gestión de Base de Datos (SGBD desde ahora) que haya por detrás de este, puede ser un sistema

relacional o una base de datos documental NoSQL, un MySQL o un PostgreSQL, pero debe de ser suficiente para manejar los datos relativos al funcionamiento de la aplicación.

En el modelo, se debe definir adecuadamente la información que vamos a usar, teniendo en cuenta las posibles restricciones (longitud de cadena, tipo de datos guardado, si el valor debe ser único en la tabla...) de manera que se mantenga siempre la integridad de los datos.

En el caso que se trate de una base de datos relacional, deben ser definidas las relaciones entre tablas que y el comportamiento por defecto al eliminar valores vinculados (eliminar en cascada, actualizar, dejar como NULL...).

4.4 Conectando las capas: Un ejemplo en código.

Para poder apreciar de una manera más clara la interconexión existente entre las tres capas del patrón MVC, se incluyen a continuación dos porciones de código, uno relativo a la vista y otro relativo al controlador, se mostrará una tabla de ejemplo que representará una tabla del modelo, y finalmente el resultado devuelto por la aplicación.

El código mostrado corresponde a un listado de vehículos de la aplicación objeto de este documento, por lo que el código sigue el esquema y la nomenclatura de Symfony2.

4.4.1 Código relativo a la Vista

```
<div id="results-count">
        {{ total }} vehículos encontrados
</div>
<table>
        <tr>
                <th>#</th>
                <th>No. de Flota</th>
                <th>Matricula</th>
                <th>Modelo</th>
                <th>Constructor</th>
                <th>Última ITV</th>
                <th>Última Semestral</th>
                <th> </th>
        </tr>
{% for vehiculo in vehiculos %}
        <tr>
                <td><input type="checkbox" id="vehiculo-{{ vehiculo.id }}"
                        value="{{ vehiculo.id }}" name="vehiculo-{{ vehiculo.id }}" /></td>
                <td>{{ vehiculo.numFlota }}</td>
                <td>{{ vehiculo.matricula }}</td>
                <td>{{ vehiculo.modelo }}</td>
                <td>{{ vehiculo.constructor }}</td>
                <td>{{ vehiculo.ultimaITV | date('d/m/Y') }}</td>
                <td>{{ vehiculo.ultimaSemestral | date('d/m/Y') }}</td>
                <td>
                        <a data-rol="boton-editar" href="{{ path('editar_vehiculo', { "num_flota":
vehiculo.numFlota } ) }}">
                                <img src="{{ asset('bundles/vehiculo/img/edit.png') }}" />
                        </a>
                        <a data-rol="boton-eliminar" href="{{ path('eliminar_vehiculo', { "num_flota":
```

```
vehiculo.numFlota } ) }}">
                                        <img src="{{ asset('bundles/vehiculo/img/delete.png') }}" />
                                </a>
                        </td>
                </tr>
{% else %}
                <tr>
                        <td colspan="7" style="text-align: center;">
                                No hay vehiculos registrados
                        </td>
                </tr>
{% endfor %}
</table>
```

VehiculoBundle:Default:listado.html.twig

Como se puede apreciar, se trata de un fragmento de código HTML que mediante un objeto recibido por el controlador (vehículos), itera el contenido del mismo realizando un listado en una tabla.
Este código tiene fragmentos de Twig resaltados en negrita, ya que son los que aplican cierta lógica funcional al código.

4.4.2 Código relativo al Controlador

```
public function listadoAction(){
        $em = $this->getDoctrine()->getEntityManager();
        $entities = $em->getRepository("VehiculoBundle:Vehiculo")->queryFindVehiculos();

        return $this->render('VehiculoBundle:Default:listado.html.twig', array(
                "vehiculos" => $entities,
                "total" => count($entities)
        ));
  }

public function queryVehiculos(){
        $em = $this->getEntityManager();
        $dql = "SELECT v FROM VehiculoBundle:Vehiculo v ORDER BY v.numFlota DESC";
        $query = $em->createQuery($dql);
        return $query->getResult();
}
```

VehiculoBundle:DefaultController.php

En este caso se pueden apreciar dos funciones, una que es la que gestiona la llamada (*listadoAction()*) y otra que es la que contiene la consulta a la base de datos *(queryVehiculos()*).

Normalmente, la segunda función se encontrará en otro tipo de documento, llamado *Repositorio*, que es donde se engloban todas las funciones que realizan consultas a la base de datos.

4.4.3 Contenido de la base de datos

En la imagen adjunta puede apreciarse una captura de los datos contenidos en la base de datos, aunque sólo se trata un fragmento y que además no coincidan con los mostrados en la presentación final, sirve para apreciar los campos de los que se conforma la tabla Vehículo de este modelo.

id	numFlota	modelo	bastidor	constructor	matricula	seguro	depot	site	financiera	ultimaITV	ultimaATP	ultimaTermografos	ultimaSemestral	contrato_id
5001	588740	20"	49124009468594760	MAN	6416-FRR	0	CARPA - 29	VIGO	ING	2005-02-16	0000-00-00	0000-00-00	2001-07-03	NULL
5002	570803	Volquete	42545849413042204	MAN	9745-DWL	0	CARPA - 33	SEVILLA	Cetelem	2004-03-01	0000-00-00	0000-00-00	2011-04-05	1724
5003	584961	Volquete	73600418331704009	PEGASO	2001-HKZ	1	CARPA - 42	VALENCIA	Cetelem	2011-08-08	0000-00-00	0000-00-00	2005-08-10	NULL
5004	468654	20"	13459953397262893	MAN	7926-FVR	1	CARPA - 7	SEVILLA	ING	2000-02-19	0000-00-00	0000-00-00	2007-01-15	NULL
5005	800093	20"	28317432579443118	MACK	7557-FSW	1	CARPA - 30	MADRID	Cetelem	2000-01-10	0000-00-00	0000-00-00	2005-01-14	NULL
5006	315738	Frigo MT	39109889213396056	IVECO	8856-BKT	1	CARPA - 20	SEVILLA	OpenBank	2012-11-05	2006-09-09	2004-06-27	2009-01-19	NULL
5007	831862	Frigo	97484720133226665	MACK	8786-GLZ	0	CARPA - 31	MADRID	OpenBank	2006-08-24	2002-06-04	2002-03-28	2003-03-05	1740

Figura 4.2 - Contenido de la base de datos

4.4.4 Respuesta del sistema

494 vehículos encontrados

#	No. de Flota	Matricula	Modelo	Constructor	Última ITV	Última Semestral	
☐	999999	0108-DNY	Lona	IVECO	17/10/2002	04/05/2003	✎ ✖
☐	999371	8731-GVX	Frigo	PEGASO	29/09/2011	24/06/2000	✎ ✖
☐	998741	0706-FPY	Frigo MT	IVECO	20/01/2000	21/10/2002	✎ ✖
☐	997132	4587-DDZ	Volquete	PEGASO	20/03/2011	17/07/2010	✎ ✖
☐	995724	5958-FLR	Cisterna	MAN	28/07/2011	13/09/2012	✎ ✖
☐	994920	7184-GQW	Cisterna	MGM	01/03/2010	01/07/2011	✎ ✖
☐	989926	4908-HMW	Volquete	MAN	02/11/2000	13/05/2004	✎ ✖
☐	989292	2380-CQS	Frigo MT	MAN	12/05/2004	13/05/2006	✎ ✖
☐	987946	7462-FCB	20"	MACK	19/04/2001	02/01/2005	✎ ✖
☐	985937	4730-DZN	Lona	MAN	03/11/2012	20/04/2003	✎ ✖
☐	985747	3509-CMQ	Lona	IVECO	16/12/2011	23/12/2007	✎ ✖
☐	985052	8385-GTW	Lona	MACK	18/12/2000	11/10/2006	✎ ✖
☐	983418	0917-CQS	Frigo MT	MGM	27/02/2006	07/05/2006	✎ ✖
☐	981796	3381-FKR	20"	MAN	17/05/2005	27/09/2011	✎ ✖
☐	979985	3851-GTH	Frigo	MGM	26/10/2009	01/11/2012	✎ ✖

Primera Anterior 1 2 3 4 5 6 7 8 9 10 Siguiente Última

Figura 4.3 - Muestra de la respuesta del sistema

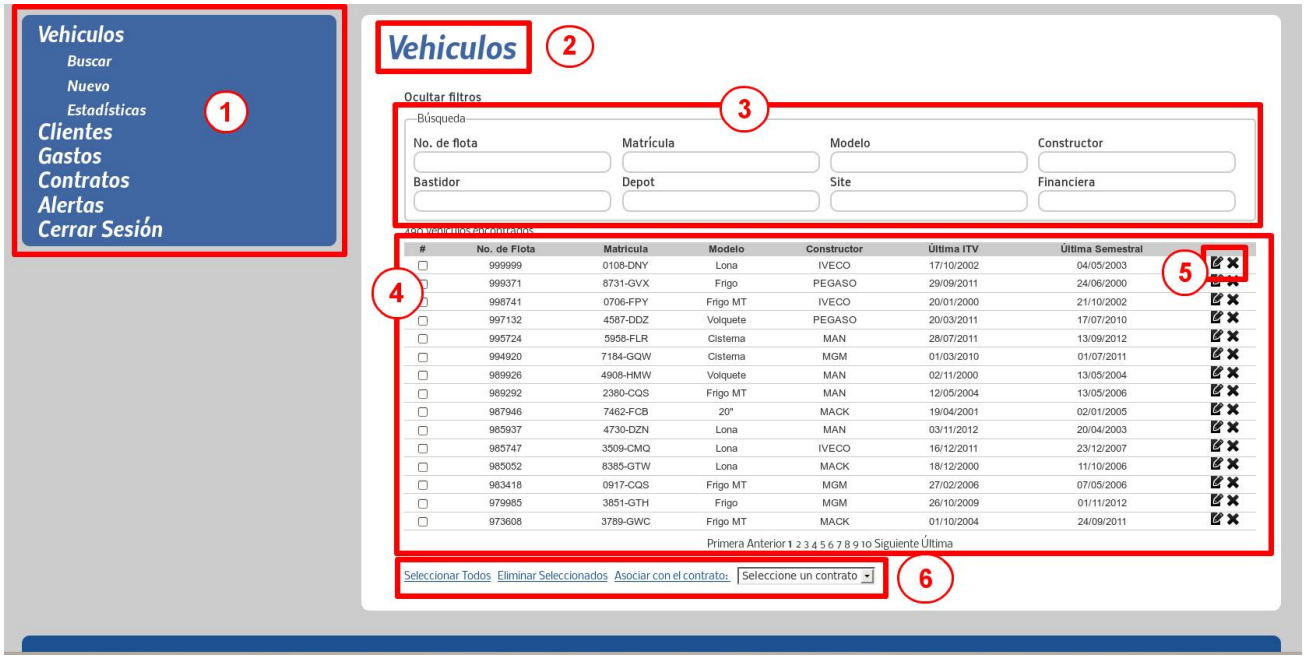
Esta última captura demuestra cómo quedaría la respuesta despues de unir con la vista los datos obtenidos del modelo por el controlador.

4.5 Wireframe de la aplicación

Respecto al diseño de la aplicación, lo último que vamos a ver es un pequeño esquema de la la distribución del contenido en la pagina media de la aplicación Fleet Manager.

Aunque no todas las páginas son iguales, si que siguen la misma estructura básica, formada por dos zonas: una pequeña franja lateral y una zona más grande con el contenido principal de la página.

A continuación, sirviéndose como referencia del siguiente esquema, podrá verse como está distribuido el contenido:



Leyenda

1. **Menú Desplegable**
   - Contiene el menú de navegación de la página, desde aquí se puede acceder a todas las secciones de la página
2. **Título de la página**
   - Nos sirve para identificar donde estamos en cada momento
3. **Filtros de Búsqueda**
   - Dada la ingente cantidad de información que puede llegar a manejarse con Fleet Manager, es posible filtrar el contenido para encontrar lo que buscamos más sencillamente.
   - Con estos filtros es posible realizar búsquedas en tiempo real sobre la base de datos, con solo escribir en cualquiera de los criterios de búsqueda la tabla inferior **(4)** comenzará a mostrar los resultados coincidentes.
4. **Tabla de resultados**
   - La tabla de resultados nos muestra los registros coincidentes con los criterios de búsqueda. En caso de no definirse ningún filtro, la tabla mostrará todos los registros de la base de datos.

- Puesto que pueden llegar a ser muchos resultados, la tabla se encuentra paginada a 15 resultados por página. Además, estos registros pueden seleccionarse mediante el selector situado en la primera columna de la tabla (#) para realizar operaciones en lote sobre ellos. También es posible realizar operaciones individuales sobre cada registro mediante los controles situados en la última columna de la tabla **(5).**

5. **Operaciones individuales**
    - Con cada registro podemos realizar un conjunto de operaciones reducido, aunque depende de cada modulo la disponibilidad de estas operaciones, e incluso el tipo de operación, las dos más comunes son editar y borrar.
6. **Operaciones en lote**
    - Al igual que ocurre con las operaciones individuales, la disponibilidad del tipo de operación depende del módulo en cuestión, las únicas operaciones en lote comunes a todos los módulos son "Seleccionar/Deseleccionar todos" y "Eliminar seleccionados".
    - Para poder realizar este tipo de operaciones debe de haber seleccionado mínimo un registro mediante el selector que se encuentra en la primera columna por la izquierda de la tabla de resultados **(4).**

Mapa de navegación

El mapa completo de navegación es el siguiente:

- Vehículos
    - Buscar
        - Ver/Editar
        - Eliminar
    - Nuevo
    - Estadísticas
- Clientes
    - Buscar
        - Ver/Editar
        - Eliminar
    - Nuevo
- Gastos
    - Buscar
        - Ver/Editar
        - Eliminar
    - Nuevo
    - Estadísticas
- Contratos
    - Buscar
        - Ver/Editar
        - Eliminar
    - Nuevo
- Alertas
    - Alertas ITV
    - Alertas Semestrales
    - Alertas Contratos
    - Alertas ATP
    - Alertas Termógrafos
- Cerrar Sesión

5. Implementación

El desarrollo de la aplicación se ha llevado a cabo sobre una arquitectura LAMP, más específicamente sobre una Ubuntu Server 12.1, con Apache 2.2, MySQL 5.2 y PHP5.

La elección del lenguaje de desarrollo ha sido en base a la comodidad de usar un lenguaje ya conocido y tener la oportunidad de trabajar con Symfony2, un framework para PHP especialmente orientado para aplicaciones con un perfil similar a la que se ha desarrollado.

Aunque en los siguientes apartados se aborda Symfony2 con más detalle, el lector debe saber que Symfony2 es un framework RAD (*Rapid Application Development*) que facilita mucho al desarrollador tareas muy rutinarias (inserción y manipulación en bases de datos) y otras de mayor envergadura que suelen dar mayores quebraderos de cabeza (refactorizar código, modificar o ampliar entidades y actualizar los esquemas de la base de datos automáticamente...).
Además, no se trata de la única tecnología, Symfony2 incluye otros componentes, como Twig, Doctrine, Composer... que también serán descritos en los próximos apartados.

5.1. Tecnologías usadas en la implementación

5.1.1 Symfony2

Como ya se ha comentado, el núcleo de la aplicación es Symfony2, un framework RAD para PHP.
Aunque no es el único, y a nivel mundial ni siquiera el más conocido, Symfony2 goza en España de una popularidad altísima, con enormes comunidades de desarrolladores por todo el país, conferencias anuales y muy buen soporte.

Otras opciones que se han barajado son CakePHP y Yii. Son frameworks también muy reconocidos y con una comunidad también bastante amplia, pero la fuerza que ha cogido Symfony y la cantidad de mejoras que ofrece y que saca cada 6 meses puntualmente, dan una sensación de que es un proyecto sólido y con mucha proyección.
Symfony2 está desarrollado por Fabien Potencier, de SensioLabs, una firma francesa de software.

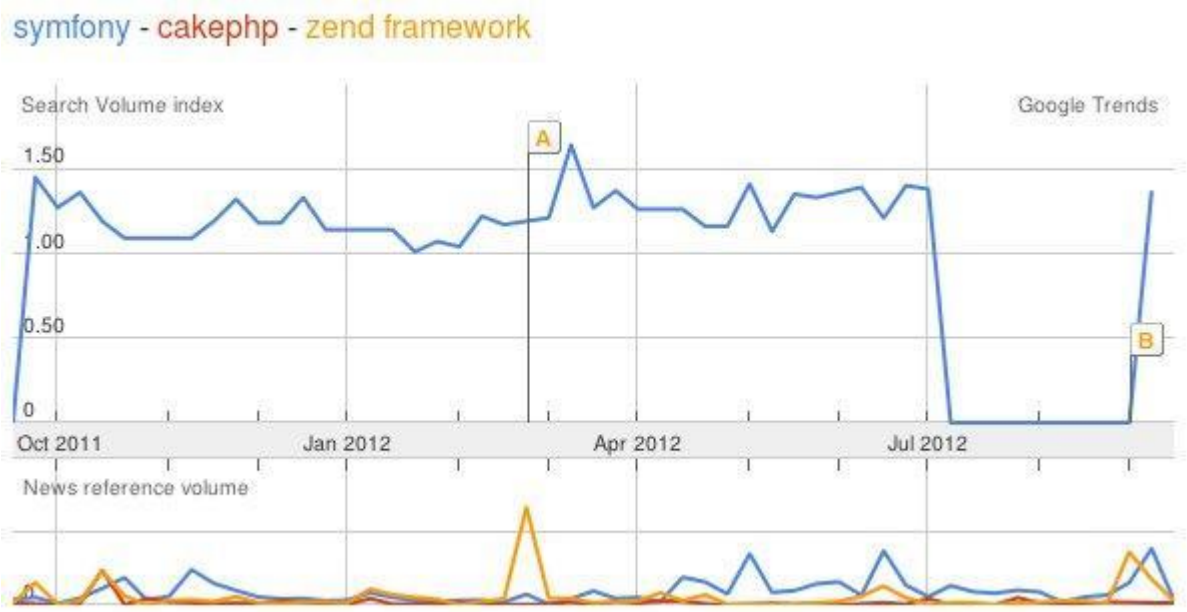


Figura 5.1 - Popularidad de Symfony2 en España

5.1.1.1 Componentes

Entre los componentes que integra Symfony2 se encuentran:

- Twig
  - Un potente motor de plantillas desarrollado también por SensioLabs
- Sistema de Routing
  - El sistema encargado de interpretar las URL's y enviar la petición al controlador adecuado
- Doctrine
  - Un ORM muy versátil y con muchas posibilidades
- Composer
  - Gestor de dependencias de Symfony2, permite mantener actualizado el sistema e instalar nuevo componentes fácilmente.
- Assetic
  - Sistema de compresión, unifica y minimiza CSS y JavaScript bajo demanda para optimizar el tráfico de la aplicación.

5.1.1.2 Configuración

Una de las mayores ventajas de Symfony es la sencillez de su configuración, todos los ficheros de configuración están en YAML, un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python o Perl. Este formato permite que los ficheros puedan leerse por el programador de una manera muy sencilla y fácil de interpretar.

5.1.1.3 Contenedor de inyección de dependencias

Otra gran ventaja que ofrece Symfony2, y que es sin duda lo que le dota de rapidez y flexibilidad, es el *contenedor de inyección de dependencias*.
La inyección de dependencias consiste en *inyectar* a las clases todos los objetos que necesitan para funcionar ya creados y configurados.

El contenedor de inyección de dependencias es pues un objeto que sabe cómo crear los objetos de las aplicaciones, conoce todas las relaciones entre las clases y la configuración necesaria para instanciar correctamente cada clase.

5.1.1.4 La consola de Symfony2

Symfony2 ofrece un interfaz de consola desde el cual es posible controlar todos los aspectos del proyecto.

Desde la consola de Symfony se pueden dar de alta nuevos bundles (actualizando automáticamente el kernel de Symfony2 y las tablas de rutas), crear nuevas entidades (generando automáticamente el código asociado a los *getters* y los *setters*), vaciar la caché del sistema (indicando el entorno a utilizar), actualizar los *assets*, actualizar el esquema de la base de datos en base a las *Entities* de nuestro proyecto, volcar la tabla de rutas... y un largo etcétera.
A continuación se muestra el listado completo de acciones que se pueden realizar con la versión 2.1 de Symfony a través de la consola.

```
Available commands:
  help                                Displays help for a command
  list                                Lists commands
assetic
  assetic:dump                        Dumps all assets to the filesystem
assets
  assets:install                      Installs bundles web assets under a public web directory
cache
  cache:clear                         Clears the cache
  cache:warmup                        Warms up an empty cache
config
  config:dump-reference               Dumps default configuration for an extension
container
  container:debug                     Displays current services for an application
doctrine
  doctrine:cache:clear-metadata       Clears all metadata cache for an entity manager
  doctrine:cache:clear-query          Clears all query cache for an entity manager
  doctrine:cache:clear-result         Clears result cache for an entity manager
  doctrine:database:create            Creates the configured databases
  doctrine:database:drop              Drops the configured databases
  doctrine:ensure-production-settings Verify that Doctrine is properly configured for a production
environment.
  doctrine:fixtures:load              Load data fixtures to your database.
  doctrine:generate:crud              Generates a CRUD based on a Doctrine entity
  doctrine:generate:entities          Generates entity classes and method stubs from your mapping
information
  doctrine:generate:entity            Generates a new Doctrine entity inside a bundle
  doctrine:generate:form              Generates a form type class based on a Doctrine entity
  doctrine:mapping:convert            Convert mapping information between supported formats.
  doctrine:mapping:import             Imports mapping information from an existing database
  doctrine:mapping:info               Shows basic information about all mapped entities
  doctrine:query:dql                  Executes arbitrary DQL directly from the command line.
  doctrine:query:sql                  Executes arbitrary SQL directly from the command line.
  doctrine:schema:create              Executes (or dumps) the SQL needed to generate the database schema
  doctrine:schema:drop                Executes (or dumps) the SQL needed to drop the current database
schema
  doctrine:schema:update              Executes (or dumps) the SQL needed to update the database schema
to match the current mapping metadata
  doctrine:schema:validate            Validates the doctrine mapping files
generate
  generate:bundle                     Generates a bundle
  generate:doctrine:crud              Generates a CRUD based on a Doctrine entity
  generate:doctrine:entities          Generates entity classes and method stubs from your mapping
information
  generate:doctrine:entity            Generates a new Doctrine entity inside a bundle
  generate:doctrine:form              Generates a form type class based on a Doctrine entity
init
  init:acl                            Mounts ACL tables in the database
  init:jms-secure-random
router
  router:debug                        Displays current routes for an application
  router:dump-apache                  Dumps all routes as Apache rewrite rules
  router:match                        Helps debug routes by simulating a path info match
swiftmailer
  swiftmailer:spool:send              Sends emails from the spool
translation
```

translation:update	Updates the translation file
twig	
twig:lint	Lints a template and outputs encountered errors

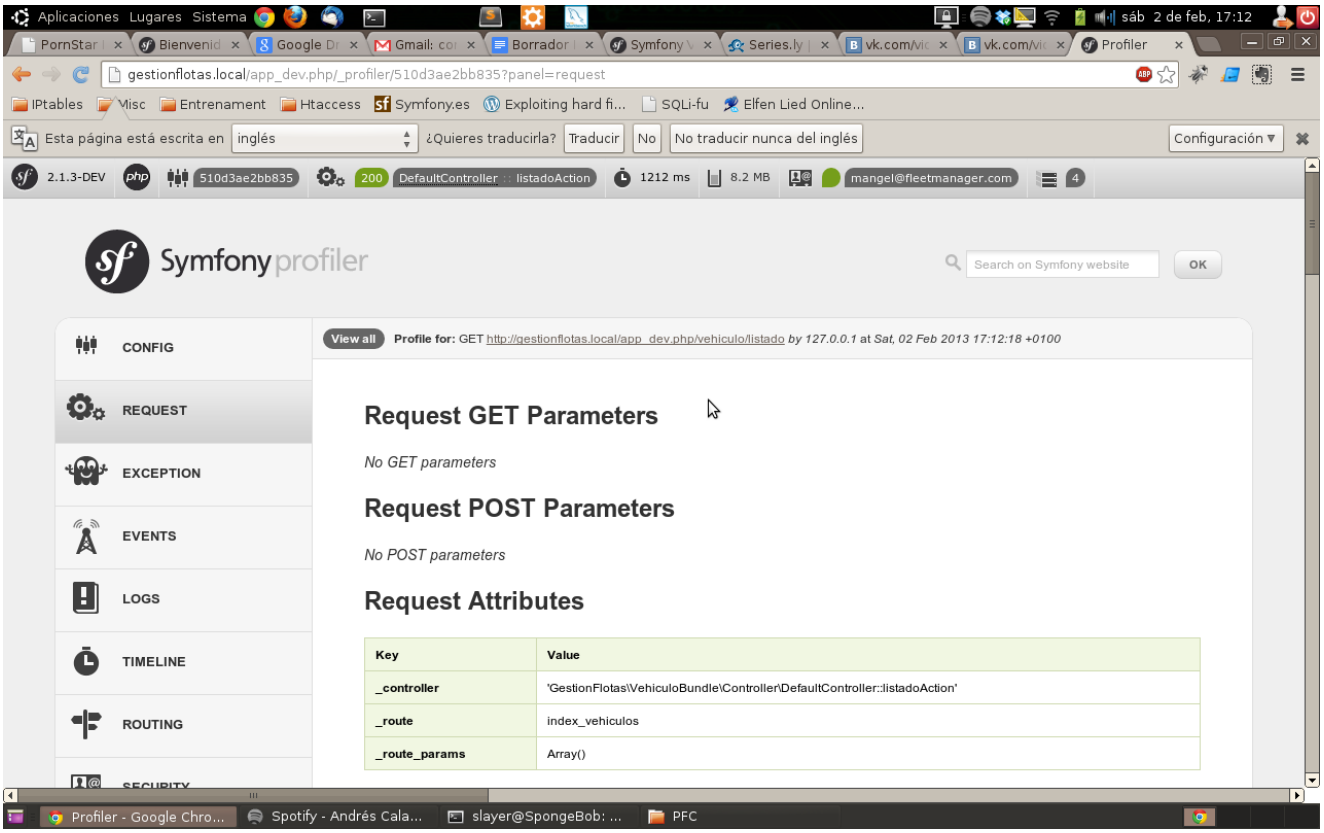
Listado 5.1 - Listado de opciones de la consola de Symfony2

5.1.1.5 Profiler

Symfony2 cuenta también con un *Profiler*, una aplicación propia de Symfony que permite analizar la aplicación:

- Ver que parámetros se han pasado, tanto por GET como por POST,
- Tiempo que ha tardado en cargar
- Número de consultas ejecutadas
- Qué consultas se han ejecutado
- Tiempo de respuestas de las consultas
- Ruta por la que ha entrado
- Excepciones que ha generado el código
- Emails enviados
- Estado del usuario (logueado, deslogueado)

y un larguísimo etcétera de opciones que hace sencillo depurar el código cuando falla.



5.1.1.6 Entornos de desarrollo

Otra de las características que hacen de Symfony2 un entorno ideal para el desarrollo es la posibilidad de usar varios entornos.

Por defecto Symfony2 propone dos entornos: producción (prod) y desarrollo (dev). Estos dos entornos funcionan de forma muy diferente, por ejemplo, el entorno de producción realiza un cacheado de consultas y respuestas mucho mayor que el realizado en desarrollo, además no ofrece acceso al profiler para depurar el código. Este entorno además ofrece un mayor rendimiento, ya que es la versión final de la aplicación y la que utilizan los usuarios finales.

Por otra parte, el entorno de desarrollo da mayores facilidades para el desarrollo, pero su rendimiento está ligeramente penalizado respecto al entorno de producción, ya que realiza un mayor número de escrituras en logs y carga el profiler con cada llamada.

5.1.1.7 El sistema de Routing

El sistema de routing de Symfony es uno de los componentes más exportables de Symfony2, de hecho debido al débil acoplamiento de los componentes de Symfony2, es bastante sencillo incorporar algunos de estos componentes a otros proyectos PHP no basados en Symfony2.

El sistema de routing nos permite crear fácilmente rutas asociadas a controladores que se encargaran de procesar la petición y devolver el contenido adecuado.

Las rutas se pueden definir en ficheros YML, además cada bundle dispone de su propio fichero routing.yml en el que podemos dar de alta las rutas necesarias. Existe un fichero de routing central en la carpeta app, en este se incluyen el resto de ficheros configuración de rutas de los bundles, de hecho, este es el único fichero de rutas que lee Symfony2, por lo que si no hemos incluido algún routing.yml de algún bundle, este no será interpretado.

Las rutas se definen como mínimo con dos parámetros: el **pattern** y el **defaults**. En el primero definimos la “forma” o aspecto que tendrá la URL y las variables que se le pasarán al controlador, en el segundo se especifica el controlador que manejara esta ruta y los valores por defecto de las variables de la ruta, si es el caso.

Existe un tercer parámetro muy útil también que nos permite hilar más fino en la definición de la ruta. Se trata de **requirements**, una serie de condiciones que entre otras cosas pueden especificar el “tipo” de una variable mediante expresiones regulares (numérica, alfanumérica, una opción entre un conjunto cerrado…), el protocolo usado para acceder (http o https), el formato de la petición (html, xml, rss, json…) o el idioma en el que se servirá la petición.

Otra forma de definir rutas es en el propio controlador, haciendo uso de anotaciones en la cabecera del método que gestionará una ruta.

El debugging de las rutas es bastante sencillo gracias al profiler de Symfony2 que incluye una sección de “Router” en el que se puede ver todas las rutas que han sido

evaluadas y las que han realizado matching, así como la que ha resultado elegida para procesar la petición.

Otra forma de ver lo que hay definido es utilizar la consola de Symfony2 para "volcar" un listado con todas las rutas existentes en el sistema mediante el comando **router:debug**.

5.1.2 Twig

Twig es el motor de plantillas oficial de Symfony2.
Aunque no es exclusivo para ser usado con Symfony, ya que puede usarse en cualquier proyecto PHP que deseemos.

Twig incluye una gran cantidad de interesantes características que lo convierten en una opción muy atractiva para el desarrollo de front-ends.
Entre estas características, destaca especialmente la herencia de plantillas, un mecanismo con el cual es posible ampliar una plantilla de la que se está heredando previamente. Esto permite seguir un patrón conocido como *Herencia a tres niveles*.

Este patrón se basa en crear una primera plantilla que contenga la etiquetas básicas que conforman la estructura común de la página (Doctype, html, head, body y las inclusiones básicas de hojas de estilos y scripts comunes a todas las páginas).

Una segunda plantilla que define la estructura de las páginas (se pueden definir por ejemplo una estructura para las páginas del front-end y otra para las del back-end). Además, pueden incluirse más hojas de estilos y más scripts comunes a las páginas que posteriormente hereden de esta.

Finalmente, se pueden crear más plantillas, una por cada vista del front-end o del back-end que hereden de la plantilla correspondiente y que incluyen también las hojas de estilos necesarias y específicas de cada vista, así como scripts con funciones únicas para cada vista.

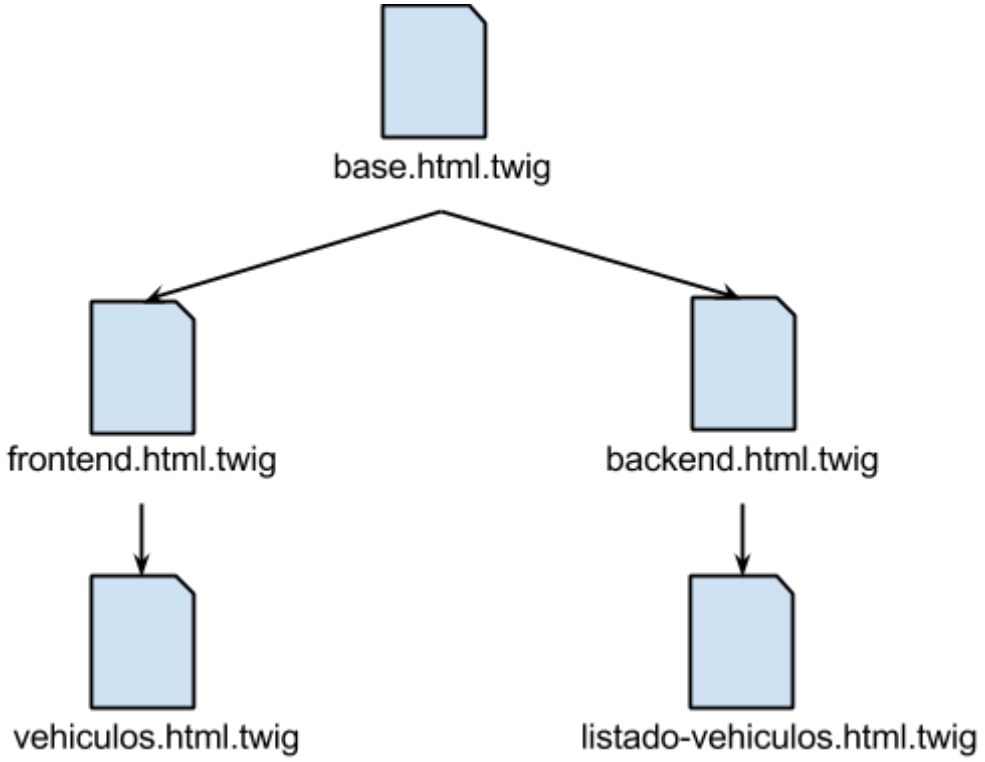


Figura 5.2 - Ejemplo de herencia a tres niveles

Además de esta característica, Twig cuenta con bastantes funciones, estructuras de control de flujo como bucles, condicionales... etc, y filtros, que permiten, por ejemplo, dar formato a fechas.

Cabe también destacar la posibilidad de desarrollar funciones y filtros propios para Twig, lo cual lo convierte en una herramienta muy ampliable en cuanto a funcionalidad.

Por último destacar que Twig es considerado uno de los motores de plantillas más seguros y rápidos de la actualidad.
Como características de seguridad destacan el escapado global automático del contenido de las variables PHP y el modo *sandbox* donde se pueden definir conjuntos reducidos de etiquetas, limitando así la funcionalidad de Twig para otros desarrolladores que trabajen en el mismo proyecto.

5.1.3 Doctrine

Según lo visto hasta ahora podría decirse que Symfony2 es la tecnología que representa a la capa *controlador* del modelo MVC, y que Twig, la que representa a la *vista*, así que Doctrine será la que represente al *modelo*.
Y no podía ser de otro modo, ya que Doctrine es un *Object-Relational Mapping* (desde ahora ORM).

5.1.3.1 Object-Relational Mapping

El ORM es una técnica de programación que permite convertir del modelo relacional con objetos tipados, propios de los lenguajes de programación orientada a objetos.

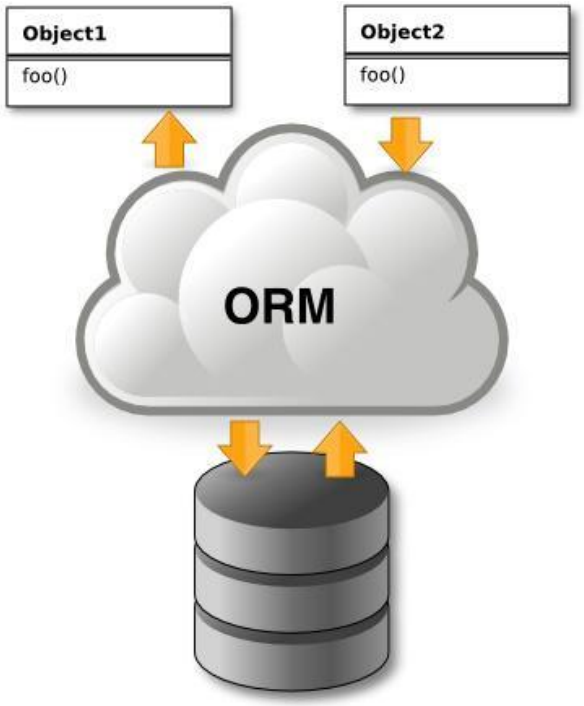


Figura 5.3 - Representación de un ORM

Su funcionamiento consiste básicamente en, por ejemplo, obtener datos de un cliente de la la tabla *Cliente* de la base de datos y convertirlo en un *objeto* de la *entidad Cliente* que hemos definido en nuestro código. De este modo, es mucho más sencillo trabajar con los datos, ya que al ser objetos, estos poseen métodos con los que realizar ciertas acciones.

Otra de las ventajas que nos da un ORM es que abstrae por completo al programador del motor de base de datos usado, por lo que podría pasarse de usar, por ejemplo, MySQL a PostgreSQL sin tener que tocar ni una línea del código de los controladores o las vistas.

Trabajar con un ORM nos permite pensar íntegramente en objetos, olvidándonos temporalmente de la base de datos que soportará la persistencia de la aplicación, ya que será el propio ORM el que mantenga y defina esto.

5.1.3.2 Acerca de Doctrine

Doctrine es un ORM para PHP que viene por defecto incluido con Symfony2, a pesar de que no es proyecto del mismo fabricante. En anteriores versiones de Symfony, el ORM por defecto era Propel.

Doctrine es bastante sencillo de utilizar y muy ampliable, y cuenta con una buena comunidad detrás suyo.

Entre las características más destacadas de Doctrine están el lenguaje DQL, las transacciones ACID, el *lazy loading,* y la *hidratación* del resultado de la consulta.

5.1.3.3 Doctrine Query Language (DQL)

No es más que un subconjunto del lenguaje SQL estándar con algunas modificaciones de sintaxis.
Con DQL realizamos consultas *orientadas a objetos*. Esto nos permite obtener un resultado fácilmente convertible en un objeto de la aplicación.
Carece de funciones típicas de SQL, como funciones de strings o fechas. No obstante, permite realizar joins con otras tablas, así como usar funciones de agregación.

5.1.3.4 Transacciones ACID

Doctrine es *ACID Compliant*, lo cual quiere decir que todas las transacciones (entendemos como transacción un conjunto de consultas con un propósito común) que realiza cumplen los siguientes requisitos:

- Atomicidad (Atomicity)
    - La transacción se realiza como un todo, o tiene éxito o fracasa, pero no se queda a medias.
- Consistencia (Consistency)
    - Solo se empieza lo que puede acabarse.
    - Si existen errores de integridad entre las operaciones no se ejecutará.
    - La transacción parte de una base de datos válida y tiene que dejar como resultado una base de datos igualmente válida.
- Aislamiento (Isolation)
    - Una transacción no afecta a otra.
    - Dos transacciones sobre un mismo conjunto de datos son independientes y no genera error.
- Durabilidad (Durability)
    - Asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer.

5.1.3.5 Lazy Loading

Una de las características más propias de Doctrine. El lazy loading o *carga perezosa* puede ser un arma de doble filo, ya que bien usado es una característica deseable, pero fuera de control puede sobrecargar una aplicación hasta su saturación.

Como es frecuente, entre las tablas de una base de datos hay relaciones, por lo que un atributo de una tabla puede referenciar a todo un registro de otra tabla. Esto implica que podamos acceder a la información contenida en la tabla referenciada.

Symfony y Doctrine permiten acceder a cualquier información que solicites y no esté disponible, haciendo que Doctrine busque automáticamente en la base de datos.

Si por ejemplo obtenemos un listado de 500 elementos de la tabla *Vehículo* obtendremos entre otros datos, una referencia al contrato actual de cada vehículo. Si se

quisiese acceder a los datos del contrato para, digamos, consultar las fechas de inicio y fin, no haría falta modificar la consulta, Doctrine buscará esos datos automáticamente. Teniendo en cuenta que el resultado es de 500 elementos, se realizarán 500 consultas adicionales para resolver los datos de los contratos.

Esto se puede evitar especificando en la consulta que tambien queremos los datos del contrato, uniendo las tablas con un JOIN.

5.1.3.6 Hidratación de resultados

La hidratación de los resultados consiste en especificar a Doctrine cómo queremos que devuelva los datos consultados.
Las opciones son:

- Objeto (Object)
  - Valor por defecto
  - Devuelve un array donde cada elemento es un objeto.
  - Este objeto puede incluir objetos de sus entidades asociadas
- Array
  - Devuelve un array donde cada elemento es un array asociativo simple
  - Este array puede incluir arrays de sus entidades asociadas
- Escalar (Scalar)
  - Devuelve un array unidimensional.
  - Las entidades relacionadas no se incluyen en forma de array, sino que sus propiedades se añaden al único array devuelto
- Escalar unidimensional (Single Scalar)
  - Devuelve únicamente el valor solicitado.
  - Solo se puede utilizar si se consulta una única propiedad de una sola entidad.
- Objeto Simple (Simple Object)
  - Solo se puede usar si la consulta devuelve una única entidad, sin importar el número de resultados.

Eligiendo bien el tipo de hidratación a usar, puede mejorarse considerablemente el rendimiento de la aplicación.

5.1.4 Composer

En un afán por simplificar la creación y mantenimiento de proyectos Symfony, SensioLabs desarrolló otra herramienta complementaria para Symfony: Composer.

Composer es un gestor de dependencias configurable mediante JSON que, entre otras cosas, nos permite crear nuevos proyectos, actualizar las versiones de Symfony de proyectos ya creados, instalar bundles de terceros en nuestro proyectos sin complicarse... etc.

Esta programado con PHP y hace uso de Git para acceder a los repositorios de código de las dependencias que necesita instalar.

5.1.5 MySQL

MySQL es un sistema de gestión de bases de datos (SGBD) relacionales.
Es ampliamente usado en en el desarrollo de aplicaciones web dada su sencillez y rapidez al resolver consultas.

Se ha elegido como motor de persistencia para la aplicación objeto de este documento por su sencillez, fácil integración en PHP, fácil gestión a través de interfaces web como phpMyAdmin y disponibilidad en varias plataformas.

5.1.6 Git

Para mantener un control exhaustivo de los cambios llevados a cabo durante el desarrollo de la aplicación, se ha optado por usar un software de control de versiones.
La herramienta elegida ha sido Git.
Git es una herramienta de control de versiones creada por Linus Torvalds, creador asimismo del kernel Linux.
Git está pensado para el mantenimiento del control de versiones de proyectos con grandes cantidades de ficheros.
En el caso de los proyectos con Symfony2 es altamente recomendado trabajar con un repositorio Git.

Por otra parte, el IDE utilizado durante el desarrollo de este proyecto es un fork de Eclipse llamado Aptana, optimizado para el desarrollo web.
Aptana lleva integración de serie con repositorios Git, haciendo extremadamente simple llevar un control absoluto sobre las revisiones de código.

5.1.7 HTML 5

El lenguaje de marcado HTML 5 ha sido el elegido para realizar el interfaz de la aplicación de Fleet Manager.

Es la quinta revisión del lenguaje HTML, el lenguaje con el que está definido la inmensa mayoría del contenido de Internet. HTML fue creado por Tim Berners-Lee a principios de 1990, considerado a día de hoy el padre de la WWW.
Se trata de un lenguaje de marcado, no de un lenguaje de programación, ya que carece de elementos como variables, funciones, iteradores... propios de los lenguajes de programación. Es una variante de SGML ampliada y su finalidad es dotar a los textos de una estructura.

Para dotar a las páginas de estilos gráficos o interactuar con el usuario se utilizan otras tecnologías como CSS o Javascript

5.1.8 CSS

Las Cascade Style Sheets o simplemente CSS son un conjunto de reglas de estilo que permiten dar una mejor apariencia a los documentos HTML.
El lenguaje HTML por si mismo solo permite estructurar contenidos, pero poco puede hacer por el aspecto visual. Ahí es donde entra CSS, dando estilos a los textos con tipografías, colores, tamaños, fuentes, o dando formato a los bloques de contenido con márgenes, bordes, colores de fondo... etc.

Se han convertido en un recurso íntimamente ligado con el código HTML, ya que es extraño ver webs que no implementen estilos con CSS.
A pesar de ser tan utilizado, el maquetado con CSS es una ardua tarea debido a que durante una época, casi cada navegador interpretaba de una manera las reglas, haciendo imposible que un mismo recurso se viera igual en distintos navegadores.
A día de hoy este problema perdura de manera bastante notable, siendo Internet Explorer de Microsoft el navegador peor valorado y con una interpretación más "libre" del estándar CSS.

La solución a la que parece ser han llegado muchos navegadores es implementar prefijos para las reglas que todavía están en proceso de estandarización, por ejemplo –moz-border-radius para añadir bordes redondeados en Mozilla Firefox o –webkit-border-radius para Chrome y Safari.

5.1.9 Javascript

Este lenguaje desarrollado basado en el estándar ECMAScript fue desarrollado inicialmente por Netscape, llamándose LiveScript en un principio y Javascript después debido a la popularidad del lenguaje Java como estrategia comercial.

Se trata de un lenguaje orientado a objetos, basado en prototipos, imperativo y débilmente tipado.

En un principio se usó para dar mayor dinamismo a las paginas web, ya que permitían al usuario interactuar con la página.

Al igual que con CSS, ha habido muchas "formas de ver" este lenguaje por los distintos navegadores, y códigos que funcionaban correctamente en uno no tenían porque funcionar en otro de otro fabricante (de nuevo, Microsoft queda en último lugar).

Con el paso del tiempo han aparecido frameworks Javascript que abstraen al programador de estas incompatibilidades y lo centran en desarrollar su código y su lógica de negocio.

Algunos ejemplos de estos frameworks son Mootools, jQuery, Angular JS… aunque no todos son iguales, Javascript es un lenguaje con un enorme crecimiento y existen implementaciones MVC como Angular o Backbone, implentanciones para ejecutar Javascript en servidores, como Node.js, librerías gráficas como Kendo UI, o jQuery UI, frameworks de testing como Jasmine, frameworks de desarrollo móvil como jQueryMobile, Sencha o Rachett, librerías de primitivas como _underscore.js…

En definitiva, una infinidad de frameworks distintos en propósito y tecnología hacen de Javascript uno de los lenguajes con mayor crecimiento y desarrollo a día de hoy.

5.2. Estructura de ficheros de Fleet Manager

5.2.1 Bundles de Symfony2

Los proyectos de Symfony2 se organizan en bundles. Un bundle es un directorio del proyecto que representa una división lógica del proyecto y contiene en una estructura de directorios jerarquizada todo lo relativo a la misma.
Los bundles contienen clases PHP y archivos web o *assets* (JavaScript, CSS e imágenes).

Una forma de división podría ser utilizar tantos bundles como partes tiene el proyecto, como por ejemplo *frontend, backend* y *extranet*. Esto no es un estándar y de hecho, se puede hacer al gusto del programador, incluso se podría meter todo en un único bundle.

Para el desarrollo de *Fleet Manager* se ha dado un enfoque distinto y más modular, ya que la estructura lo permitía.
El proyecto se ha organizado con un bundle por cada entidad principal, aunque algunos bundles contienen más de una entidad, como por ejemplo el bundle de *Contrato* que contiene la entidad Contrato y TiposContratos.

Esta división de bundles permite tener una visión más modular del proyecto y entender las partes de las que se compone. Además también permite hacer la aplicación configurable en cuanto a funcionalidad, lo que permitiría crear un producto adaptable para futuros clientes; ocultando módulos o añadiendo nuevos.

El uso de bundles es nuevo en Symfony2, en versiones anteriores solo existía una estructura de *frontend* y *backend*, con un módulo para cada entidad. Esta nueva forma de gestionar el contenido favorece el desarrollo en equipo de proyectos.

La creación de bundles es un proceso sencillo, guiado por un asistente si utilizamos la consola de Symfony2 y la instrucción *generate:bundle*.
Después de una serie de preguntas mayormente relacionadas con el nombre que tendrá el bundle, el namespace, directorio destino y formato de configuración, se crea automáticamente el bundle y se registra en el kernel para ser utilizado. También se importan las rutas al archivo de rutas general.
Durante la creación del bundle se ofrece la posibilidad de generar el bundle con una estructura de directorios básica o generarlo con la estructura completa, suele recomendarse hacer la estructura básica y a partir de ahí seguir ampliando el bundle según necesidades.

5.2.2 Estructura de directorios de un Bundle

Aunque la estructura de carpetas de un bundle puede variar, siempre existe una estructura mínima de directorios que se crean cuando generamos el bundle sin especificar que queremos la estructura completa.

Figura 5.4 - Estructura de carpetas del bundle Contrato

5.2.2.1 Controller

En esta carpeta es donde se guardan los controladores, en plural. Un bundle no tiene porqué limitarse a un único controlador. Se pueden crear tantos controladores como se consideren oportunos en cada bundle.
En el bundle de Vehículos se han creado dos controladores, uno destinado a las acciones relacionadas con la entidad Vehículo y otro destinado a las acciones relacionadas con el contenido estático del sitio.

Aunque se hubiera podido optar por crear un nuevo bundle para el contenido estático, finalmente se decidió fusionarlo con el bundle que puede considerarse el más destacado.

5.2.2.2 Entity

En esta carpeta es donde se guardan las Entidades. Como ya se ha visto antes, se pueden generar Entidades directamente desde la consola de Symfony2.
Una Entidad no es más que una clase que representa un objeto de la aplicación. Esta entidad después se plasma en el modelo como una tabla relacional.

De esta manera cuando le pedimos al ORM que nos devuelve un resultado de una consulta, cada registro de la base de datos se corresponde a un objeto instancia de la entidad correspondiente.

Las entidades pueden configurarse cómodamente mediante anotaciones PHP. Las anotaciones son unos comentarios que se añaden en la cabecera de la clase o de sus atributos, de manera que el ORM les confiere ciertas propiedades relacionales, como hacer que sean un índice o un clave foránea, que sea un valor autoincremental, que no pueda guardarse como nulo... etc.

En la carpeta Entity también puede encontrarse repositorios. Los repositorios son unas clases que extienden la funcionalidad de la entidad respecto a las búsquedas con consultas.
Suelen contener métodos con consultas en DQL que facilitan las búsqueda más complejas.
Sin crear un repositorio, Doctrine nos permite por ejemplo buscar vehículos que sean de cierto modelo, o que tengan cierta matrícula, pero con los repositorios podemos resolver otras consultas como *qué vehículos tienen un contrato activo entre dos fechas dadas?*.

5.2.2.3 Form

La carpeta *Form* es la que contiene las clases que generan los formularios. En Symfony2, los formularios se definen así, como una clase más. El vocablo usado para referirse a ellos es *Type*.
En la definición del formulario se asocia el formulario con una entidad ya creada, y se *añaden* los atributos que tiene que mostrar, con su tipo, clase CSS que vestirá, nombre de la etiqueta, algunas restricciones... etc.
Lo habitual es crear un formulario por cada entidad, o bien dos formularios, pudiendo usarse uno para la inserción y otro distinto para la modificación.
También es posible generar un formulario por cada rol de usuario, si es que se muestran diferencias entre los niveles de acceso.

5.2.2.4 Resources

En la carpeta *Resources* podemos encontrar principalmente tres tipos de contenido. Por una parte podemos encontrar ficheros de configuración o de enrutamiento, por otra podemos encontrar *assets* o contenido auxiliar como imágenes, iconos, CSS, javascript... etc, y por último también podemos encontrar *vistas* de la aplicación. Todo esto se encuentra ordenado en un jerarquía de subcarpetas que permite ordenar el contenido.

Config

En esta carpeta podemos encontrar ficheros de configuración y enrutamiento.
Principalmente hay dos ficheros que podemos encontrarnos:

- routing.yml
  - Fichero de rutas del bundle
  - Suelen importarse sus rutas al fichero de rutas principal, en la carpeta *app*
- services.yml
  - Aquí es donde se dan de alta los servicios propios del Bundle
  - Un servicio es un *Listener*, un script que se ejecuta al producirse determinado evento y realiza una acción entre medias.

Public

Esta carpeta contiene elementos auxiliares, como imágenes, CSS o javascript.
Habitualmente contiene otras tres subcarpetas:

- img
  - Contiene imágenes o iconos
- css

    - Contiene ficheros de estilos propios del bundle
- js
    - Contiene los scripts en JavaScript propios del bundle

Views

Aquí se guardan las vistas. Habitualmente los ficheros siguen esta nomenclatura:

nombre.formato.twig

Por ejemplo:

listado.html.twig

Se usa esta nomenclatura porque Symfony2 puede gestionar varios formatos, y desde el controlador, basándose en la URL, podría llamar a cualquier vista, generando el documento resultante en el formato deseado.
Así podría existir:

perfil.html.twig
perfil.pdf.twig
perfil.json.twig
perfil.xml.twig

Simplemente tendríamos que generar el tipo de documento apropiado para ello, y en esto Twig da muchas facilidades.

5.2.2.5 Tests

Otro de los puntos fuertes de Symfony2 es la integración de PHPUnit como motor de tests unitarios y funcionales.
Un test unitario no es más que una prueba de un pequeño trozo de código para comprobar que funciona correctamente, como por ejemplo, probar que un método de una clase funciona correctamente.
Los tests funcionales van un poco más allá y prueban requisitos funcionales enteros, como por ejemplo, validar que se insertan nuevos elementos en la base de datos y que se redirige al usuario al sitio adecuado después de hacerlo.

Symfony ayuda a testear fácilmente la navegación incorporando un objeto *Client* que actúa como un navegador web, pero que podemos controlar mediante código.
Cada vez que se lanza el cliente contra una URL de la aplicación, devuelve un *crawler,* que mediante expresiones XPath permite acceder a los elementos de la página cargada y asegurarnos de que está bien formada, incluso rellenar formularios y mandarlos.

La mayor ventaja de realizar estos tests es que después de hacer un cambio importante se pueden volver a ejecutar los test y comprobar que nada ha cambiado y todo sigue intacto, sin tener que hacer de nuevo a mano todas las pruebas.

5.2.2.7 Listeners

Es donde se guardan las clases que implementan servicios. Por ejemplo, podría tratarse de un servicio que mantiene un log de auditoría de una tabla del modelo. y cada vez que se produce una nueva inserción guarda un nuevo registro en el log con la fecha, usuario, IP, contenido... etc.

Los servicios son muy usados en Symfony2 para ampliar la funcionalidad del sistema, y su uso conjunto con el contenedor de inyección de dependencias, hace muy fácil su integración

5.2.2.7 Otros directorios

Además de los anteriores, un Bundle de Symfony2 puede tener más directorios aún si son necesarios.
Uno de los más socorridos durante el desarrollo de aplicaciones es el de **DataFixtures.**

Las *fixtures* son unas clases que nos sirven para llenar la base de datos con información de pruebas, suelen consistir en un bucle que crea objetos de la entidad adecuada, y los persiste en la base de datos, permitiendo así por ejemplo llenar una base de datos de pruebas con 500 usuarios falsos para realizar pruebas.

Las fixtures se cargan desde consola, y si no especificamos ruta busca en el directorio DataFixtures/ORM de cada bundle.
Como en un proyecto medio suele haber dependencias en la base de datos, podemos enumerarlas para que se ejecuten de forma ordenada y la base de datos no dé errores.

6. Pruebas

Antes de servir el proyecto al cliente, este debe de pasar por una serie de pruebas que garanticen que satisface todos los requisitos y que realiza correctamente todas las operaciones y procedimientos.

Además de pruebas basadas en el funcionamiento, al tratarse de una aplicación web también debe de superar algunas pruebas extra, como la validación del consorcio W3C que define los estándares de la web, varios test de cómo se verá la aplicación en diferentes resoluciones de pantalla o cómo se verá en diferentes navegadores.

6.1. W3C

El W3C es el consorcio encargado de la regulación de los estándares web a través de recomendaciones. Este consorcio fue fundado en 1994 por Tim Berners-Lee, creador del URL y de HTML.

6.1.1 Validación HTML

Es posible validar el contenido de una página web mediante el validador del W3C, el cual reflejará todos los errores que contiene el documento HTML analizado. La dirección del validador es:

http://validator.w3.org/

Dado que HTML es un lenguaje de marcado, al igual que XML necesitan de una DTD o *Definición de Tipo de Documento*, en la que están descritas la estructura y la sintaxis de un documento.
Las DTD se emplean generalmente para determinar la estructura de un documento mediante etiquetas. Una DTD describe:

- Elementos: indican qué etiquetas son permitidas y el contenido de dichas etiquetas.
- Estructura: indica el orden en que van las etiquetas en el documento.
- Anidamiento: indica qué etiquetas van dentro de otras.

Mediante esta DTD, el validador del W3C puede comprobar que el documento que está analizando cumple con lo especificado por el estándar.

Un documento HTML especifica en una etiqueta llamada DOCTYPE qué DTD va a utilizar. En el caso de HTML5, estándar bajo el que se ha desarrollado este proyecto, el DOCTYPE no incluye ninguna DTD, por lo que no es posible validar este tipo de documentos contra una DTD.

Aún así, a fecha de este documento el W3C ofrece una *validación experimental* basada en el motor de validación de documentos sin DTD *Validation.Nu*

6.1.2 Validación de CSS

Del mismo modo que W3C permite validar un documento HTML, permite también validar los documentos CSS.

El servicio de validación de CSS del W3C se encuentra en la siguiente dirección:
http://jigsaw.w3.org/css-validator/

Esta validación permite comprobar si se ha utilizado alguna propiedad no válida, o si se han aplicado correctamente los valores adecuados a cada etiqueta.

Los ficheros CSS de este proyecto han sido analizados con este validador obteniendo una validación satisfactoria, obteniendo el sello que lo demuestra:

6.2. Resoluciones

Este proyecto ha sido desarrollado siguiendo un principio de diseño conocido como *Responsive Web Design*. A pesar de que no hay una traducción exacta al castellano, podría traducirse como *diseño adaptable* o *diseño fluido*.

Este principio de desarrollo se fundamenta en que la página debe de ser cómodamente visible desde cualquier tamaño de pantalla, teniéndose que adaptar por tanto su diseño a las medidas de la pantalla del dispositivo utilizado.

Se puede conseguir este efecto sustituyendo los valores fijos de anchos y altos por valores porcentuales, y controlar su crecimiento o decrecimiento con anchos y altos mínimos y máximos, de forma que tengamos siempre controlada la apariencia de la aplicación.

Otra forma más elaborada y que ofrece mayor flexibilidad para redistribuir los contenidos de la página es el uso de Media Queries.
Las Media Queries son unas hojas de estilos especiales que sólo se aplican bajo unas circunstancias que previamente hemos definido, como por ejemplo una densidad de pixel específica en la pantalla del dispositivo visor, un ancho de pantalla comprendido entre dos valores fijos... etc.

Este proyecto ha sido desarrollado usando el primer método descrito. En la siguiente tabla se muestran los resultado de un conjunto de pruebas realizadas a distintas resoluciones de pantalla bajo el navegador Mozilla Firefox.

Dispositivo ejemplo	Ancho	Alto	Scroll	Valoración
Pantalla 22”	1920	1080	No	Visibilidad correcta
Portátil HD	1366	768	Vertical	Visibilidad correcta
Portátil 15”	1280	800	Vertical	Visibilidad correcta
Monitor 14”	1024	768	Vertical	Visibilidad correcto
Tablet en horizontal	960	640	Vertical	Visibilidad Correcta
Tablet en vertical	640	960	Vertical Horizontal	Contenido demasiado apretado y scroll necesario
Móvil	320	480	Vertical/Horizontal	Contenido demasiado apretado y scroll necesario

6.3. Navegadores

Esta quizás es la más dura de las pruebas que tiene que superar una aplicación web. Históricamente la incompatibilidad entre navegadores ha dado mil quebraderos de cabeza a los desarrolladores web. Esto se debe a la diferente forma de implementar e interpretar las propiedades CSS que tienen los navegadores.

Aunque pueda parecer poco, en realidad era un asunto bastante complejo de resolver, ya que ajustar los estilos para un navegador implicaba hacer que en los demás no se viera bien en la gran mayoría de las veces.

Especialmente eran problemáticas las versiones 5 y 6 de Internet Explorer. Aunque ha llovido bastante desde entonces, Internet Explorer en sus últimas versiones (a fecha de este documento, las versiones 8 y 9, la 10 sigue en beta) sigue siendo el navegador más incompatible y que menos parte del estándar CSS3 ha implementado.

La mayoría de navegadores además han implementado parte de las propiedades CSS3 con prefijos. Esto quiere decir que si la propiedad para añadir bordes redondeados es *border-radius* en estándar CSS, podemos encontrar propiedades como *-moz-border-radius* para Mozilla Firefox, *-webkit-border-radius* para navegadores con motor Webkit como Chrome o Safari, o *-o-border-radius* para Opera.
Esto se debe a que han añadido ya la propiedad pero sigue en desarrollo y el fabricante no se “atreve” a usar la propiedad oficial hasta tener el desarrollo finalizado.

Los desarrolladores web encontraron algunas maneras o “tricks” que permitían arreglar un poco los desperfectos ocasionados por estas incompatiblidades.

Hojas de estilo Reset

Una de las formas más eficaces es el uso de hojas de estilo "reset".
Son hojas de estilo preparadas para dejar a 0 todos los estilos de todos los elementos HTML. De esta forma el programador se ve obligado a redefinir todos los estilos evitando así que el navegador agregue los estilos por defecto de cada elemento HTML que añade automáticamente y que son distintos en cada navegador (incluso entre versiones de un mismo navegador)

Fallbacks

Otra forma un tanto más tediosa y sólo recomendable cuando en un proyecto debe primar la visibilidad y el diseño ante la funcionalidad o el rendimiento, es el uso de Fallbacks.

Los Fallbacks son scripts en JavaScript que corrigen un comportamiento no deseado de un navegador. También pueden detectar carencias en características (como por ejemplo la interpretación de ciertas propiedades CSS) y añadir la funcionalidad a través del script en JavaScript.

Hay librerías JavaScript dedicadas exclusivamente a esto, como por ejemplo *Modernizr* (http://modernizr.com/docs/#whatis).
Una de las características más usadas de Modernizr es la posibilidad de detectar características y en su defecto suplirla con fallbacks o *polyfills* que es como llama Modernizr a estos scripts.

Preprocesadores de CSS

El uso de preprocesadores CSS también se ha extendido en los últimos años, ya que permiten definir hojas de estilos con un pseudo-lenguaje con elementos interesantes como variables o condicionales.
Otro punto fuerte es la herencia de propiedades o *mixin*. Consiste en indicar que una regla hereda de otra y ésta obtiene sus propiedades, sin tener que volver a reescribirlas. Esto es especialmente útil cuando necesitamos añadir prefijos para algunas propiedades. Véase un ejemplo en código LESS.

```
// LESS

.rounded-corners (@radius: 5px) {
  -webkit-border-radius: @radius;
  -moz-border-radius: @radius;
  -ms-border-radius: @radius;
  -o-border-radius: @radius;
  border-radius: @radius;
}

#header {
  .rounded-corners;
}
#footer {
  .rounded-corners(10px);
}
```

```
/* Compiled CSS */

#header {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
  -ms-border-radius: 5px;
  -o-border-radius: 5px;
  border-radius: 5px;
}
#footer {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -ms-border-radius: 10px;
  -o-border-radius: 10px;
  border-radius: 10px;
}
```

Imagen 6.1 - Ejemplo de código LESS

Como se puede observar en la imagen, se ha definido en LESS una clase CSS *rounded-corners* que acepta un parámetro *@radius* con un valor por defecto de 5px y que define la propiedad *border-radius* (bordes redondeados) con todos los prefijos posibles (Mozilla, Webkit, Microsoft y Opera), y además, la propiedad sin prefijos. De este modo, cualquier navegador debería de ser compatible con los bordes redondeados.
A continuación define dos id's, *header* y *footer*, que también necesitan tener bordes redondeados. En lugar de volver a escribir todos los prefijos y la propiedad, simplemente ha añadido dentro del cuerpo de la regla la clase *rounded-corners*, heredando así sus propiedades.
Además, en el caso del *footer* ha pasado un parámetro *10px* por lo que la definición se ajusta a esta medida.
A la derecha, se puede ver el resultado en CSS compilado.

Gestión de flotas

En este proyecto se ha hecho uso de hojas de estilos reset, concretamente la del proyecto Boilerplate de HTML5.

No se ha necesitado de Fallbacks para suplir carencias del navegador, ya que la especificación de requisitos exigía que fuera usable en Firefox y Chrome, que no presentan carencias respecto a las características usadas.

Al no tratarse de un proyecto con un contenido eminentemente gráfico no ha habido necesidad de entrar en demasiadas complicaciones.

Los resultados obtenidos han sido reflejados en la siguiente tabla, donde se puede ver una descripción de las incompatiblidades en los distintos navegadores.

A la vista de los resultados se puede observar que no todas las incompatibilidades se han resuelto, especialmente en las versiones más antiguas de Internet Explorer.

A pesar de esto, la hoja de estilo reset ha mantenido intacta la estructura de la página en todos los navegadores y el hecho de que los bordes se vean rectos en lugar de redondeados o que se cargue una fuente distinta, no altera el funcionamiento de la aplicación, por lo que no se ha prestado mayor atención a esto.

Navegador	Versión	Valoración
Mozilla Firefox	19	• Todas las características CSS se ven correctamente
Mozilla Firefox	18	• Todas las características CSS se ven correctamente
Google Chrome	24	• Todas las características CSS se ven correctamente
Google Chrome	23	• Todas las características CSS se ven correctamente
Safari	6	• Todas las características CSS se ven correctamente
Opera	12.12	• Todas las características CSS se ven correctamente
Microsoft IE	7	• No interpreta bordes redondeados. • No carga webfonts. • No interpreta correctamente la propiedad *display:inline-block*
Microsoft IE	8	• No interpreta bordes redondeados. • No carga webfonts. • No interpreta correctamente la propiedad *display:inline-block*
Microsoft IE	9	• No interpreta correctamente la propiedad *display:inline-block*

Resultados de pruebas de renderizado en distintos navegadores

6.4. Pruebas de velocidad: PageSpeed

El WPO o Web Performance Optimization es una disciplina que ha ido cobrando fuerza en los últimos años.
El afán del WPO es conseguir sitios web rápidos, que rendericen lo antes posibles para dar al usuario una buena experiencia de uso.

6.4.1 Antecedentes

Aunque a priori no pueda parecer importante, esto tiene un gran peso a día de hoy, ya que la mayoría de la gente consume contenidos web desde gran diversidad de dispositivos, como tablets o smartphones, que habitualmente no disponen de conexiones de datos demasiado rápidas. Y no solo eso, a nivel de SEO (*Search Engine Optimization*) también cobra especial importancia, ya que, entre otras cosas, un sitio rápido permite al crawler indexar antes, y por tanto, más contenido. También influye mucho en la conversión y fidelización de usuarios a un sitio web o tienda online, los datos hablan por sí solos:

- **Amazon**: 0,1 segundos de retraso implican una pérdida del 1% de los ingresos [1].
- **AOL**: hizo una revisión del número de páginas vistas en muchos sitios web [2] y concluyó que aquellos que funcionan rápido tienen unas 7-8 páginas vistas por usuario y que las lentas tan sólo 3-4 páginas vistas por usuario.
- **Bing**: 1 segundo de retraso implica una caída del 2,8% de los ingresos; 2 segundos de retraso implican una bajada del 4,3% de los ingresos[3] por usuario.
- **Facebook**: 0,5 segundos más lento provoca una caída de tráfico del 3%; 1 segundo provoca una caída del 6%.
- Google: 0,4 segundos de retraso causan una caída del 0,59% de las búsquedas[4][0] por usuario; 0,5 segundos más en cargar implica un 25% menos de búsquedas.
- **Google Maps**: redujo un 30% el tamaño de sus ficheros y el número de peticiones aumentó un 30%[5] .
- **Hotmail**: 6 segundos de retraso en la carga implica 40 millones de anuncios menos al mes, lo que supone 6 millones de dólares menos al año.
- **Mozilla**: hizo su página de descargas 2,2 segundos más rápida y hubo un crecimiento de descargas de un 15,4%[6] .
- **Netflix**: activó el sistema gzip en sus servidores consiguiendo un aumento de entre el 13% y 25% de velocidad de carga y reducción de un 50% del volumen de tráfico[7].
- **Shopzilla**: consiguió reducir el tiempo de carga de las páginas de 7 segundos a 2 segundos y la conversión se incrementó entre un 7% y un 12%[8] , además de aumentar un 25% las páginas vistas del sitio y pudiendo reducir la cantidad de servidores a la mitad.
- **Yahoo!**: 0,4 segundos de retraso causan una caída entre el 5% y el 9% del tráfico[9].

---

[1] *Marissa Mayer at Web 2.0*
http://glinden.blogspot.com.es/2006/11/marissa-mayer-at-web-20.html

[2] *The Secret Weapons of the AOL Optimization Team*
http://es.scribd.com/doc/16878352/The-Secret-Weapons-of-the-AOL-Optimization-Team

[3] *The User and Business Impact of Server Delays, additional bytes and HTTP chunking in web Search*
http://velocityconf.com/velocity2009/public/schedule/detail/8523

[4] *The User and Business Impact of Server Delays, additional bytes and HTTP chunking in web Search*
http://velocityconf.com/velocity2009/public/schedule/detail/8523

[5] *We're all Guinea Pigs in Google's Search Experiment*
http://news.cnet.com/8301-10784_3-9954972-7.html

[6] *Mozilla Blog of Metrics*
http://blog.mozilla.org/metrics/category/website-optimization/

[7] *Improving Netflix Performance*
http://velocityconf.com/velocity2008/public/schedule/detail/3632

[8] *Shopzilla's Site Redo, You get what you measure*
http://velocityconf.com/velocity2009/public/schedule/detail/7709

De forma general ya nos encontramos con:

- El 47% de los usuarios esperan que una página cargue en menos de 2 segundos.
- El 14% cambia de tienda online si la página tarda en cargar.
- El 40% de los usuarios abandona una página que tarda más de 3 segundos en cargar.
- El 64% de los compradores que no están satisfechos cambia de sitio para su próxima compra.
- El 52% de los compradores afirman que un sitio que carga rápido los fideliza.

6.4.2 Algoritmos de medición

A fecha de este documento, existen dos algoritmos clave para valorar el rendimiento de una web. *Google PageSpeed* y *Yahoo YSlow* son conjunto la mejor forma de testear el rendimiento de una web.
Ambos se centran en el tiempo de carga de la página y sugieren una serie de mejoras que pueden acelerar la carga.

Estas mejoras se suelen basar en:

- Reducción del número de peticiones HTTP.
- Comprimir el contenido CSS y JavaScript de forma que los ficheros a cargar tengan menor peso y por tanto menor tiempo de carga.
- Activar ciertas directivas del servidor como las conexiones persistentes.
- Servir las imágenes previamente reescaladas desde el servidor y con los atributos de ancho y alto para evitar que el navegador las reescale.
- Incluir el JavaScript al final para evitar el bloqueo de la carga de la página.
- Postergar el análisis del Javascript lo máximo posible.
- Evitar redirecciones.
- Servir el contenido web comprimido con Gzip.
- Uso de sistemas de caché.
- Servir el contenido estático desde CDN's o dominios sin cookies.

y un largo etcétera que podemos consultar por ejemplo en la web oficial de YSlow[10] o en la web oficial de Page Speed[11].

A pesar de que no siempre es viable seguir todas las recomendaciones, es interesante al menos analizar el sitio web una vez en producción para comprobar su rendimiento y tratar de mejorar la experiencia del usuario.

---

[9] '*Don't make me wait' or Building High Performance Web Applications*
http://www.slideshare.net/stoyan/dont-make-me-wait-or-building-highperformance-web-applications

[10] Web oficial de YSlow en castellano
http://yslow.es/

[11] Web oficial de Page Speed
https://developers.google.com/speed/pagespeed/

Un punto especialmente crítico son las imágenes, ya que al tratarse de una web responsive, estas deben ajustarse siempre al contenido, siendo por tanto necesario que estas sean de un tamaño lo bastante grande como para poder soportar un amplio conjunto de dispositivos.

6.4.3 Gestión de flotas

Dado que el rendimiento de la aplicación es un punto importante en este proyecto y puesto que va a ser utilizada por un número elevado de usuarios de forma simultánea, se han realizado los tests de *PageSpeed* y *YSlow* y se han seguido algunos de sus consejos para eliminar tiempos de espera innecesarios.

Ambos algoritmos generan una puntuación en 0 y 100, siendo 0 la más baja y por tanto la peor puntuación y 100 la máxima.

6.4.3.1 Resultados prévios

Contenido CSS y JavaScript

En unas primeras mediciones, ambos algoritmos delataron que había demasiadas hojas de estilo que cargar y además sin comprimir, y exactamente lo mismo pasaba con los ficheros Javascript.

Aprovechando que Symfony2 cuenta con herramientas para optimizar esto, se decidió que eran unos cambios viables.

La solución pasaba por el uso de *Assetic*, una herramienta que hace uso de filtros de compresión que nosotros mismos podemos descargar y configurar en Symfony2.
En la propia web de Yahoo Developers podemos encontrar uno de estos compresores, el *YUI Compressor*, que además cuenta con una buena reputación y sirve tanto para CSS como para Javascript.

Con estas dos herramientas, se consiguió dos cosas: minificar el contenido de las hojas de estilo de forma individual y además juntarlas en un único fichero. Esto tiene dos ventajas claras: el contenido CSS ocupa el menor tamaño posible y se sirve en una única conexión, evitando así aumentar el número de descargas paralelas.

En el caso de Javascript, el procedimiento y el resultado es el mismo.

Servir el contenido comprimido y habilitar la caché

Otra característica mejorable que detectaron ambos algoritmos era el servir el contenido comprimido con Gzip desde el servidor, de manera que las conexiones transfieran una menor cantidad de información y sean más rápidas.

Estas modificaciones debían realizarse en el lado servidor, configurando el VirtualHost para que realizase estas tareas.

A continuación se adjunta el código del fichero de configuración del VirtualHost, donde se puede apreciar la activación del sistema de caché y la compresión de datos.

```
<VirtualHost *:80>
    ServerName      gestionflotas.com
    ServerAlias     www.gestionflotas.com
    DocumentRoot    "/var/www/gestionFlotas/web"
    DirectoryIndex  app.php

    <Directory "/var/www/gestionflotas/web">
        AllowOverride None
        Allow from all

        <IfModule mod_rewrite.c>
            Options -Multiviews
            RewriteEngine On
            RewriteCond %{REQUEST_FILENAME} !-f
            RewriteRule ^(.*)$ app.php [QSA,L]
        </IfModule>
    </Directory>

    KeepAlive            On
    MaxKeepAliveRequests    200
    KeepAliveTimeout        5

    AddOutputFilterByType DEFLATE text/css text/plain text/html application/xhtml+xml
text/xml application/xml

    <IfModule mod_headers.c>
        Header append Vary User-Agent env=!dont-vary
        ExpiresActive On
        ExpiresDefault "now plus 1 week"
        ExpiresByType image/x-icon "now plus 1 month"
        ExpiresByType image/gif    "now plus 1 month"
        ExpiresByType image/png    "now plus 1 month"
        ExpiresByType image/jpeg   "now plus 1 month"
    </IfModule>
</VirtualHost>
```

Optimizar imágenes

Un último punto en el que coinciden ambos algoritmos era en la optimización de imágenes.
En este proyecto el uso de imágenes es más bien esporádico, y a parte de algunos pequeños iconos, la única imagen a resaltar es el banner de la cabecera.
Esta imagen tenía un peso y dimensiones considerables, pero también se trataba de la única imagen de la web.

Se decidió recurrir a servir la imagen desde los servidores del servicio *Sencha*. Sencha ofrece un servicio de reescalado de imágenes online.

6.4.3.1 Resultados Finales

Las pruebas de estos algoritmos se realizaron únicamente sobre las páginas que cargan listados, ya que estas son las que más se usan y las que tienen más posibilidades de estar ralentizadas debido a la carga de datos.
Para calcular la nota obtenido en cada algoritmo se ha sacado la media de las puntuaciones obtenidas en cada página de las analizadas.

La siguiente tabla muestra los resultados:

Página	Page Speed	YSlow
Listado de Vehículos	89	92
Listado de Clientes	92	93
Listado de Gastos	88	91
Listado de Contratos	89	91
Listado de Alertas ITV	87	90
Listado de Alertas de Semestrales	88	90
Listado de Alertas de Termógrafos	87	91
Listado de Alertas de ATP	87	91
Listado de Alertas de Contratos	87	91
Resultado Medio	**88.2**	**91.1**

Tabla de Resultados de los tests de Velocidad

6.5. Pruebas unitarias y funcionales

Por último, dado que Symfony2 ofrece facilidades para el desarrollo de pruebas unitarias y funcionales, se decidió llevar un control de ciertos procesos y trozos de código que se consideraban sensibles, como por ejemplo la inserción de nuevos registros y la eliminación de los mismos.

El tener una batería de pruebas listas para lanzar simplifica mucho el desarrollo de una aplicación, ya que después de modificarla o ampliarla, verificar que todo sigue funcionando correctamente con solo ejecutar un comando en la consola de Symfony2 es muy rápido y sencillo.

6.5.1 Tests Unitarios

Los tests unitarios prueban que un pequeño trozo de código de la aplicación funciona tal y como
debería hacerlo. Idealmente, los trozos de código son la parte más pequeña posible que se pueda probar. En la práctica suelen probarse clases enteras, a menos que sean muy complejas y haya que probar sus métodos por separado.

Estos tests consisten en *aserciones* que deben de cumplirse para que el test se considere válido.
Una de las finalidades más claras de las pruebas unitarias es el testeo de las entidades. Asegurar que la información creada por la aplicación sea correcta es crítico para su buen funcionamiento. Para ello, no basta con añadir reglas de validación a las entidades, sino que es imprescindible comprobar que todas se están cumpliendo escrupulosamente.

Estos test se han realizado sobre cada una de las entidades de la aplicación. En ellos se comprueban que todas las propiedades de las entidades cumplan con sus restricciones (valor no nulo, longitud máxima de una cadena, formato de matrícula correcto... etc).

```
public function testValidarSlug()
{
    $cliente = new Cliente();
    $cliente->setRazonSocial('Transportes Químicos Valencia');
    $slug = $cliente->getSlug();
    $this->assertEquals('transportes-quimicos-valencia', $slug,
        'El slug se asigna correctamente'
    );
}
```

En el anterior código se comprueba que el slug se asigne correctamente al cliente tras añadirle la razón social.

6.5.2 Tests Funcionales

Los tests funcionales se diseñan para probar partes enteras de la aplicación, también llamados escenarios.
Estos tests son un poco más complejos de definir, ya que se trata de controlar incluso el aspecto visual de la aplicación dado un escenario.

Por ejemplo, un escenario a probar sería la generación del listado de vehículos. Entre otras cosas para saber que esta página ha sido correctamente generada debería contener una tabla con al menos dos filas, y además deberían de existir unos filtros de búsqueda.

Por suerte, Symfony2 nos ofrece algunas facilidades que pueden ayudar bastante a realizar estas pruebas. Una de estas facilidades en un cliente o navegador *especial*, ya que se trata de un navegador que controlamos mediante el código. A continuación algunos ejemplos:

```
$client = static::createClient();  /*Creamos el cliente */

$client->request('GET', '/vehiculos'); /*Accedemos por GET a la ruta /vehiculos */
$client->reload(); /*Recargamos la página*/
$client->back(); /*Volvemos atrás */
$client->forward(); /*Avanzamos una página*/

$peticion = $client->getRequest(); /*Recuperamos la petición */
$respuesta = $client->getResponse();  /*Recuperamos la respuesta*/
$historial = $client->getHistory(); /*Recuperamos el historial*/
$cookies = $client->getCookieJar(); /*Recuperamos las cookies*/
```

Además, el cliente devuelve después de cada llamada al método *request* un *crawler* que nos permite analizar el contenido HTML de la respuesta.
Por tanto, podemos comprobar si existen ciertos elementos que nos indiquen que la carga de una página se ha producido correctamente. Incluso rellenar formularios y enviarlos.

```
public function elListadoDeVehiculosCargaCorrectamente()
{
   $client = static::createClient();
   $client->followRedirects(true);
   $crawler = $client->request('GET', '/vehiculos');
   $filas = $crawler->filter(
      'article div#table-wrapper table tr'
   )->count();
   $this->assertGreaterThan(1, $filas,
      'Existen al menos dos filas en la tabla del listado'
   );
}
```

El código anterior comprueba que existe una tabla en la página, y que esta tiene más de una fila (esto es importante, ya que la primera fila es para los textos, y la segunda puede indicar que no existen resultados o ser un resultado, en cualquiera de los dos casos, sería válido).

La expresión `'article div#table-wrapper table tr'` selecciona todos los *tr* que estén dentro de un *table* que este dentro de un div con el atributo *id* igual a *table-wrapper* que a su vez esté dentro de cualquier etiqueta *article*.

Algunos de los tests funcionales que se han hecho en Gestión de Flotas son:

- La portada carga correctamente
- El formulario de contacto carga y se envía
- Un usuario sin loguear no puede acceder a los listados
- Un usuario se puede loguear y ver los listados
- Guardar un nuevo registro mediante el formulario de “Nuevo”

7. Manual de usuario

En este capítulo se hará un breve recorrido por la aplicación viendo las opciones que ofrece de forma general.
En algunos casos en los que las características propias de algún módulo se consideran interesantes, se mostrarán también estas.

7.1 Acceso de usuarios

El primer paso es acceder a la aplicación, y puesto que se trata de una aplicación privada para uso sólo de la empresa contratante, se requiere ingresar un usuario y contraseña válido en el sistema para poder acceder.
Esto queda controlado por el firewall de Symfony2 que se encarga de que a las zonas protegidas por él solo se accede si se tienen credenciales válidos.

Imagen 7.1 - Cuadro de login

7.1 Listados

Los listados son una de las zonas más usadas de la aplicación.
Los listados se presentan como tablas paginadas a 15 registros por página, en ellos se pueden consultar los registros con los que los empleados trabajan, por lo que es necesario poder localizar fácilmente la información que buscan.

Para lograr esto, se han implementado unos filtros que recargan el contenido automáticamente a medida que el empleado teclea en ellos.

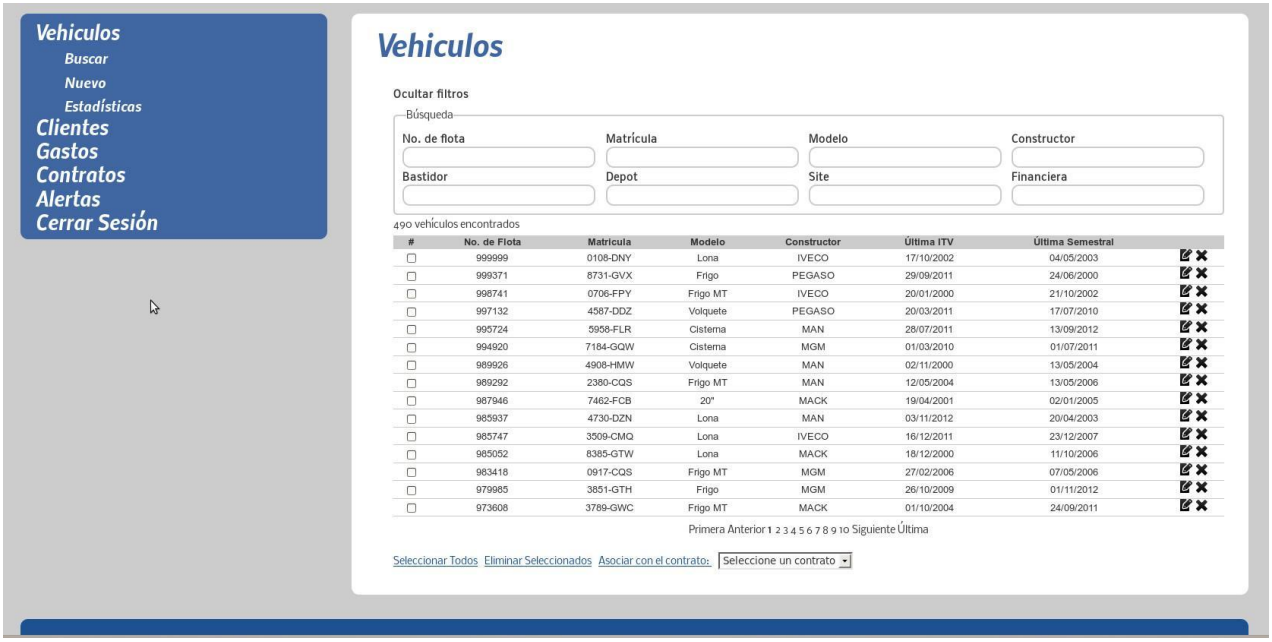


Imagen 7.2 - Listado de Vehículos

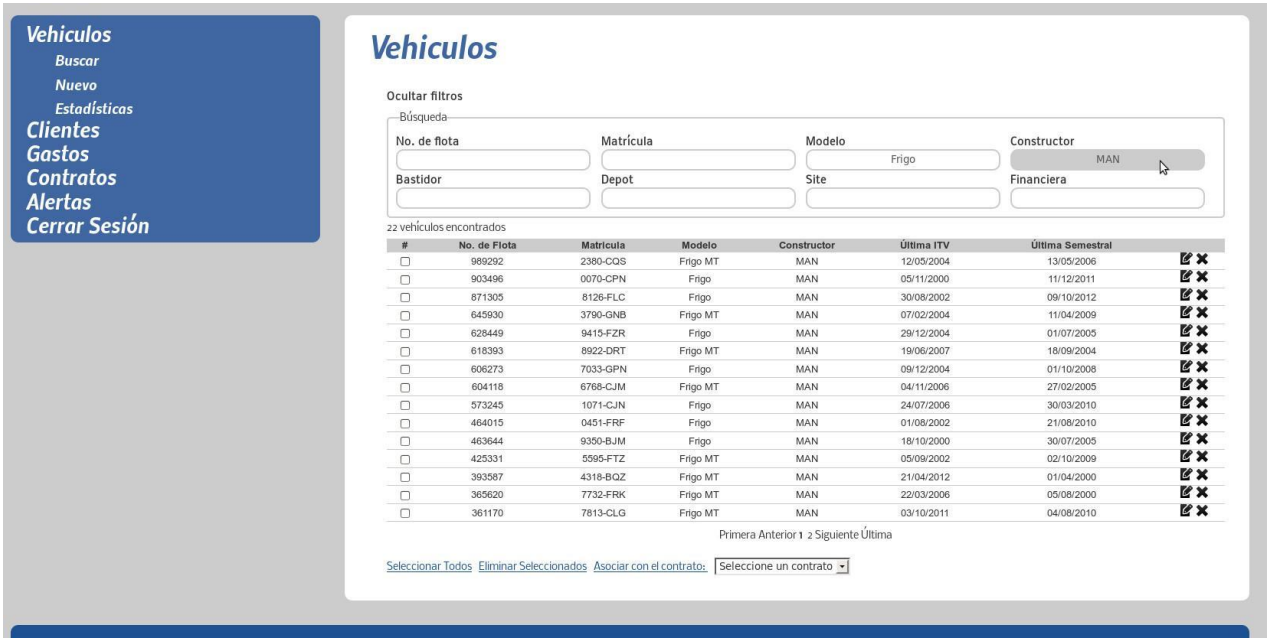


Imagen 7.3 - Listado filtrado por Modelo y Constructor

Por comodidad, estos filtros se pueden ocultar dejando más espacio en la pantalla. Esto es especialmente útil cuando se trabaja en pantallas pequeñas o portátiles.

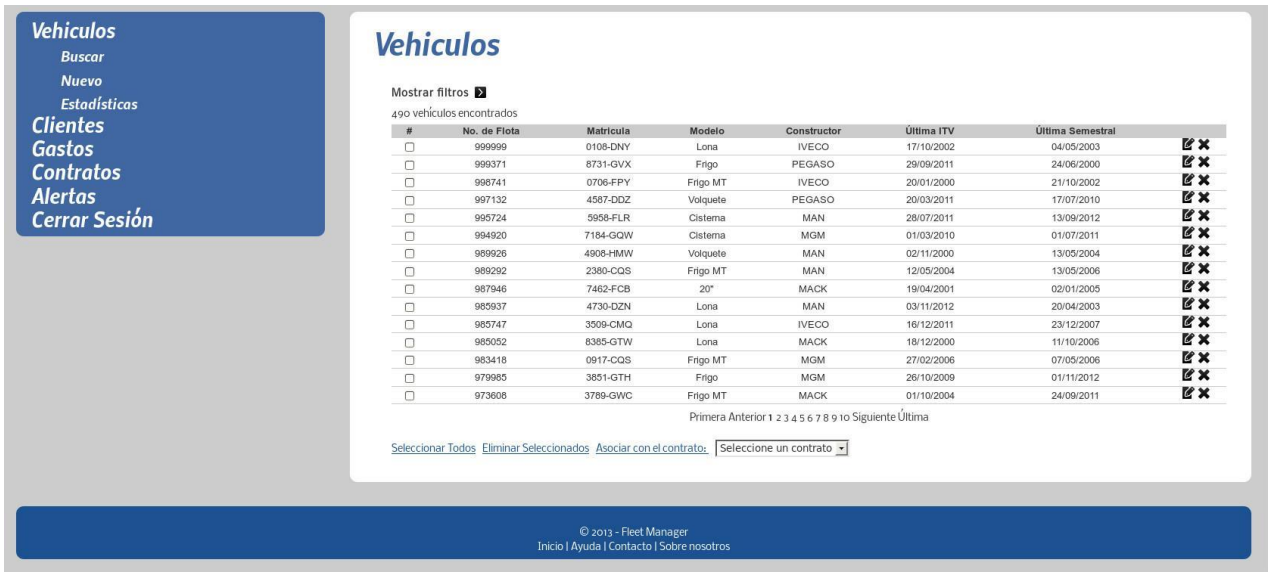


Imagen 7.4 - Listado con filtros ocultos

Como se puede observar, a la parte derecha de cada listado se pueden encontrar una serie de iconos que representan ciertas acciones que se pueden realizar con los registros. Normalmente estas son *Editar* y *Borrar*, aunque pueden variar dependiendo de módulo.

Por otra parte, en la primera columna existe un selector de registros que nos permite realizar acciones en lote, como borrar múltiples registros u otras operaciones específicas de cada módulo (asociar varios vehículos a un mismo contrato, finalizar un conjunto de contratos... etc.).

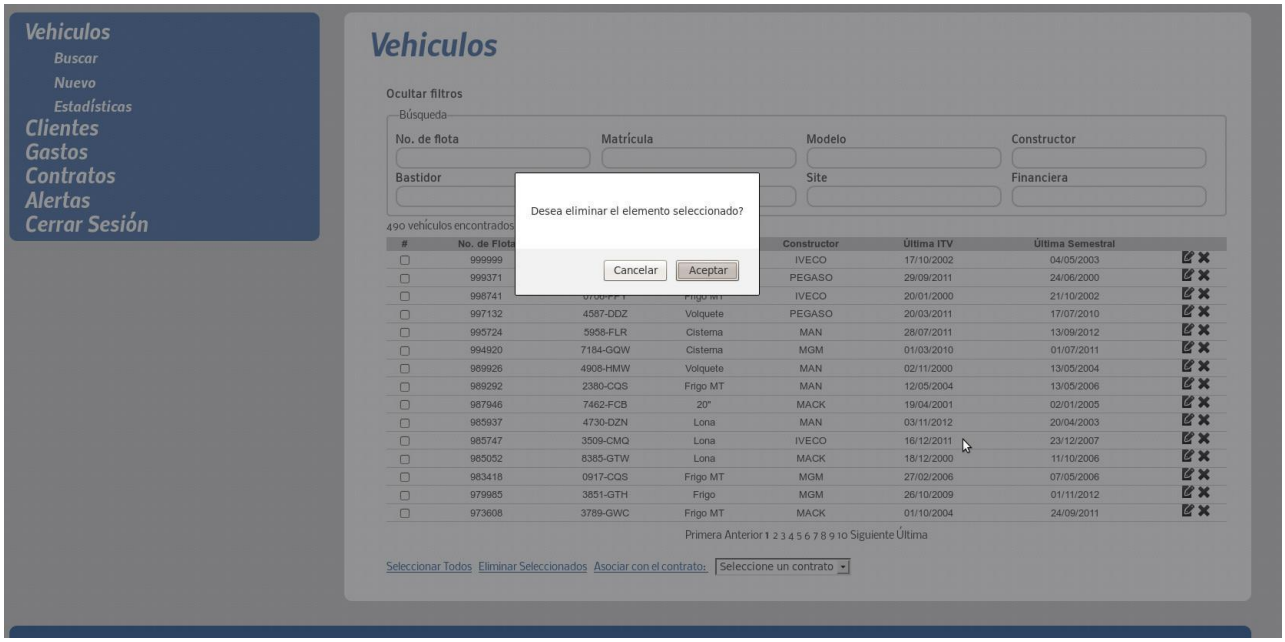


Imagen 7.5 - Eliminar un único registro

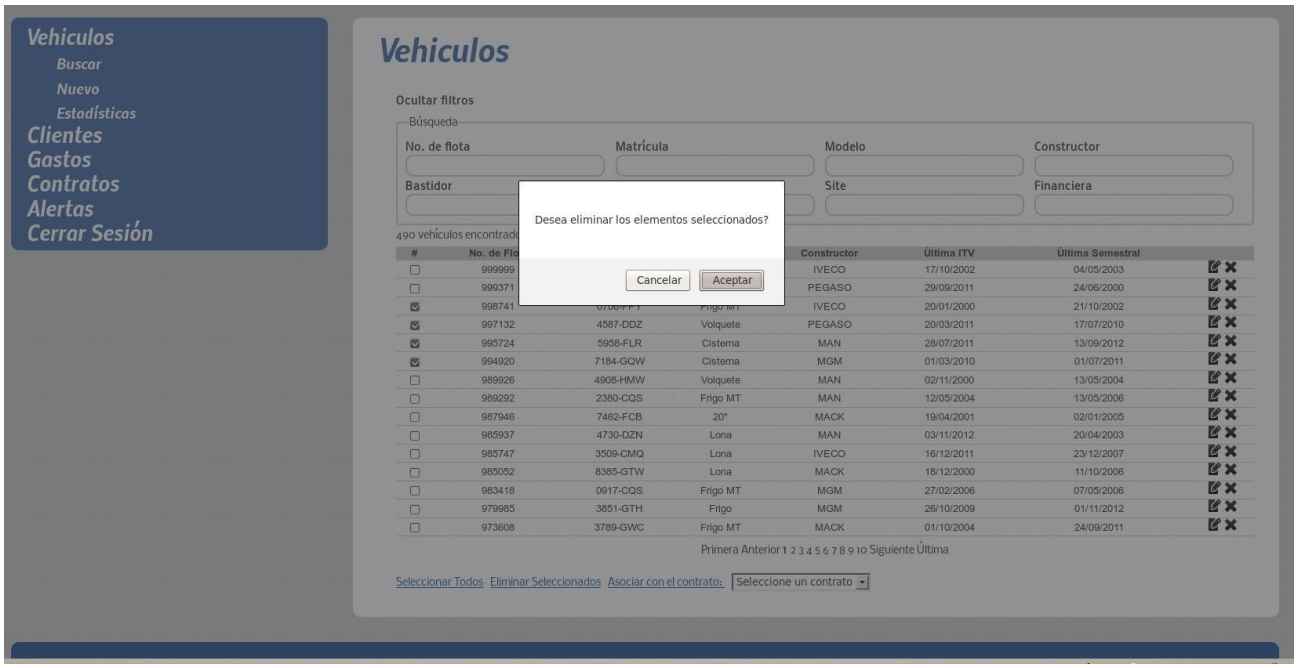


Imagen 7.6 - Eliminar un lote de registros

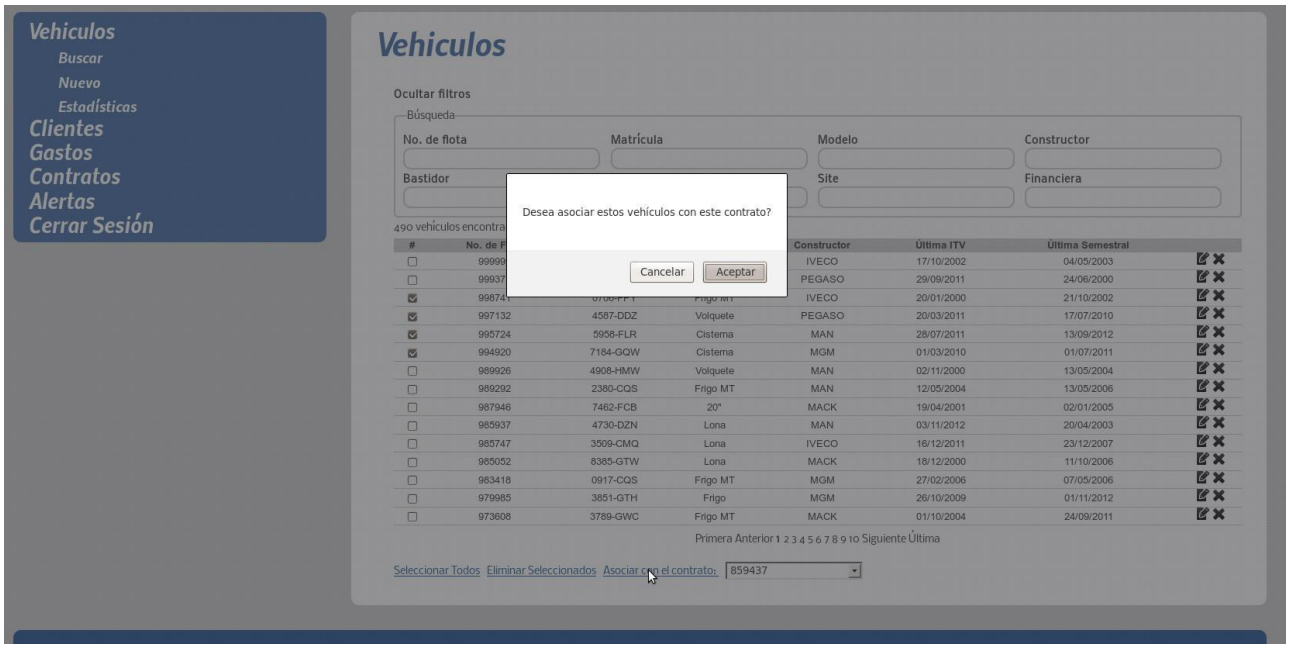


Imagen 7.7 - Operación en lote: Asociar varios vehículos a un mismo contrato

7.3 Nuevo Registro

Otra de las operaciones más habituales que realizan los empleados consiste en introducir nuevos registros al sistema: nuevos vehículos, nuevos contratos, nuevos clientes, nuevos gastos...

Todos los módulos, a excepción de *Alertas* permite añadir nuevos registros mediante un formulario adaptado para cada caso. Por sencillez se mostrará solo el formulario de nuevo vehículo, aunque el resto de formularios siguen el mismo diseño y funcionalidad.

Imagen 7.8 - Nuevo vehículo

Todos los formularios tienen control de errores, para evitar en la medida de lo posible que se introduzca información errónea o inconsistente por accidente o voluntariamente.

Los errores pueden ser de dos tipos:

1 Errores de tipo
2 Errores de entidad

Los errores de tipo son aquellos que se producen debido a que el contenido de los campos no coincide con su tipo.
Los errores de entidad son aquellos que se producen debido a restricciones de integridad de las entidades, por ejemplo *la matrícula no cumple con el formato deseado, la fecha de fin de contrato no es válida por ser anterior a la fecha de inicio del contrato, el número de bastidor no puede ser nulo,* etc.

Imagen 7.9 - Formulario detectando errores

7.4 Editar un registro

Del mismo modo que pueden darse de alta nuevos registros, estos también pueden ser modificados.
Normalmente los formularios de edición de datos están sujetos a las mismas restricciones que los formularios de nuevo registro, aunque esto no tiene siempre por qué ser así.

Por otra parte, la ficha de edición de un registro suele incluir más información a parte de la editable. Esta información son datos relacionados de otros módulos con el registro actual, por ejemplo: gastos asociados a un vehículo, contratos/vehículos asociados a un cliente, vehículos asociados a un mismo contrato... etc.

Imagen 7.10 - Ficha de un cliente con datos asociados de contratos y vehículos

Imagen 7.11 - Ficha de un contrato con vehículos asociados y la opción de asociar más vehículos

Imagen 7.12 - Ficha de vehículo con gastos asociados

7.5 Estadísticas

Algunos módulos cuentan con una sección de estadísticas basados en los datos contenidos en la aplicación. Esta información es útil para poder ofrecer un servicio más personalizado a los clientes o mejores ofertas.

Los gráficos son interactivos, por lo que se puede ocultar ciertos valores para simplificar el gráfico, o resaltar otros valores. Se han realizado mediante un plugin de jQuery llamado HighCharts & HighStocks.

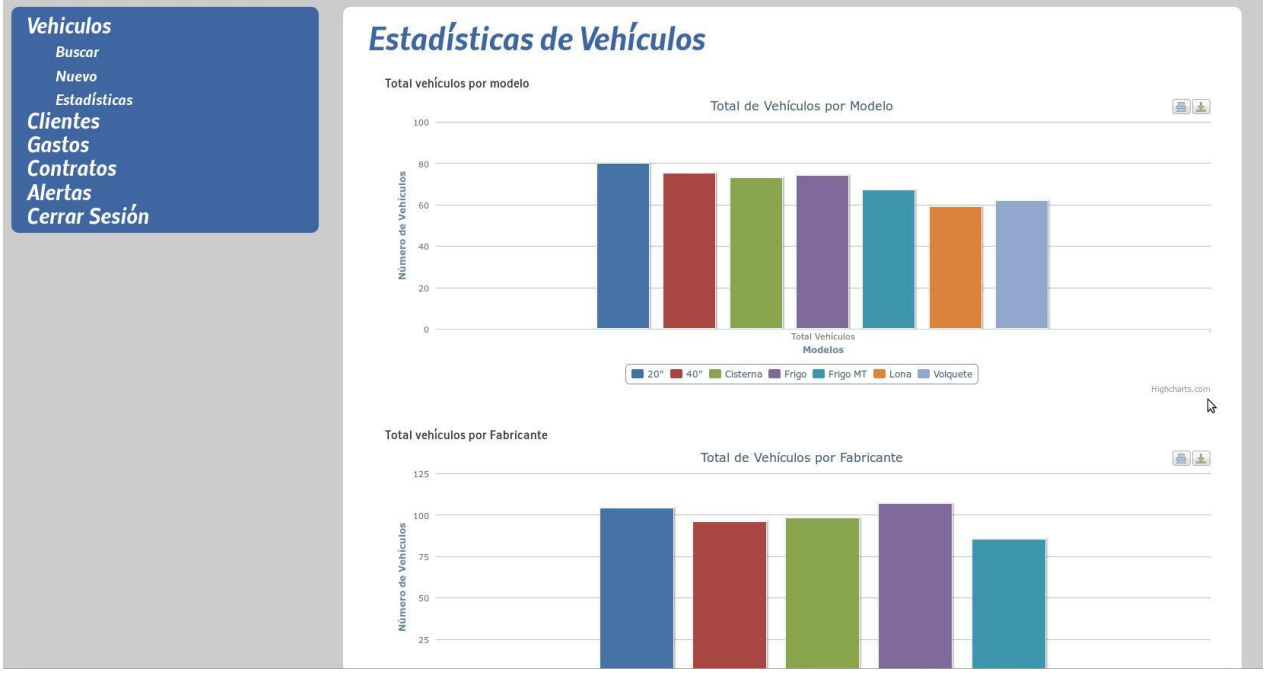


Imagen 7.13 - Estadísticas de vehículos (1 de 2)

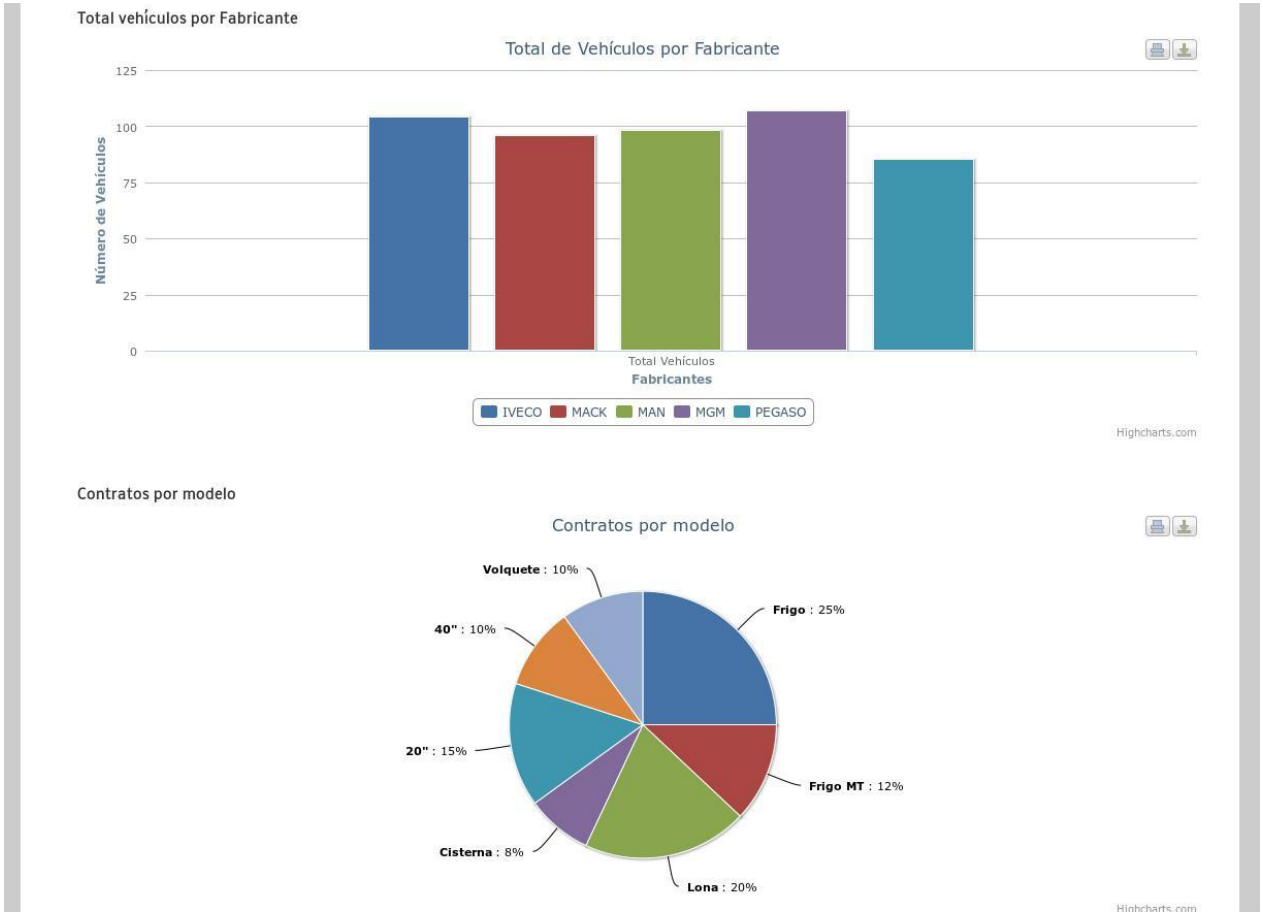


Imagen 7.14 - Estadísticas de vehículos (2 de 2)

Imagen 7.15 - Estadísticas de gastos

7.6 Alertas

Por último existe un módulo de *Alertas* que consiste en una recopilación de información de los otros módulos, indicando por ejemplo qué contratos van a caducar o han caducado o que revisiones hay proximamente o ya han caducado.

Hay tres niveles de alerta:

1. **Verde:** No hay urgencia, en breve se requerirá prestar atención a esta alerta
2. **Amarillo:** Urgencia media, la revisión o contrato caduca en pocos meses
3. **Rojo:** Urgencia alta, la revisión o contrato ha expirado su fecha de finalización.

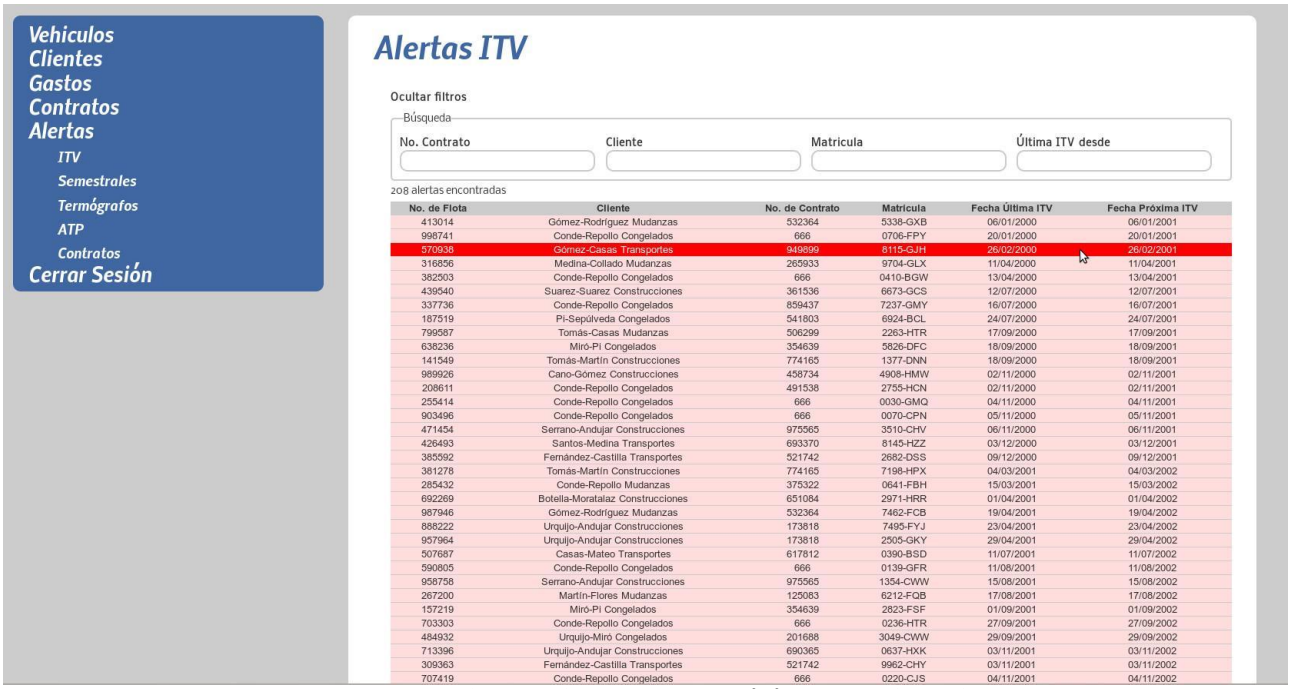


Imagen 7.16 - Alertas de ITV con revisiones caducadas

8. Conclusiones

En este capítulo se incluyen las conclusiones tanto personales como técnicas sobre lo que ha supuesto desarrollar este proyecto.

8.1. Conclusiones Personales

El desarrollo de este proyecto ha representado para mí un pequeño reto personal. Después de 3 años dedicado al desarrollo de aplicaciones web en PHP, he visto código muy variado, desde código muy limpio y claro hasta código ilegible e inmantenible. Pasado un tiempo dedicándome al desarrollo de aplicaciones a medida sin más herramientas que un editor de texto, empecé a echar de menos paradigmas y conceptos que aprendí a lo largo de la carrera, como Programación Orientada a Objetos, arquitectura Modelo Vista Controlador... etc.

Por mi propia cuenta empecé a crear proyectos en mi trabajo de una forma más ambiciosa, usando orientación a objetos en PHP, creando mis propias clases reutilizables, creando interfaces para aumentar la reutilización de componentes, separando la lógica de negocio de la presentación y de los datos para hacer el código más mantenible... etc.

Después de esto inevitablemente me encontré con los frameworks PHP orientados al Modelo Vista Controlador más conocidos, como CakePHP, Zend Studio, Symfony2 y otros muchos más.
Especialmente me llamó la atención Symfony2, ya que tiempo atrás había oído hablar de él por algunos compañeros y encontré muchísima documentación, libros, artículos... etc.
Poco a poco fuí conociendo Symfony2 y pude observar que todo lo que yo había estado haciendo no era incorrecto, de hecho, la idea no difería de la de Symfony2, pero era mucho menos amigable y reutilizable que la de Symfony2.

Estudié otros frameworks, como CakePHP o CodeIgniter, pero siempre encontré carencias en ellos respecto a todo lo visto en Symfony2, y paralelamente a todo esto, Symfony2 no había dejado de crecer, seguía incorporando componentes nuevos, como Composer, que hacían aún más sencillo el uso de Symfony2.

Estaba ante un proyecto vivo, en pleno crecimiento y era el momento de formar parte de él, así que decidí aprender Symfony2.

Lo más difícil era convencer en mi empresa para que usáramos Symfony2 en los proyectos y así abandonáramos el desarrollo casi artesanal de aplicaciones web y pudiéramos realizar proyectos en pequeños equipos de desarrollo, siguiendo un esquema de desarrollo común y conocido y con tecnologías punteras.

Aunque suena goloso para cualquier programador, en los oídos de un empresario o de algún responsable de proyectos suena a desastre innecesario y evitable, por lo que no suelen dar luz verde a este tipo de cosas.

Por eso, el desarrollo de este PFC era un reto, pretendía coger una aplicación que ya hubiéramos desarrollado antes y rehacerla desde cero con Symfony2, y así demostrar que era una tecnología viable y útil.

Los resultados convierten este reto en un éxito. No solo ha sido desarrollado en menos de la mitad de tiempo que tardó el otro proyecto, sino que el rendimiento es ampliamente superior, el frontend es más rápido, el código es mucho más modular y ampliable, contamos con un entorno de pruebas y otro de producción... en definitiva: considero que vale la pena el uso de frameworks en el desarrollo de aplicaciones web.

Por último puedo decir que durante el desarrollo de esta aplicación he aprendido muchos conceptos y técnicas de desarrollo que antes desconocía: he podido trabajar con tecnologías punteras como YUI Compressor, Doctrine ORM, sistemas de control de versiones como Git... etc.
También he aprendido sobre una rama muy importante en el desarrollo actual de aplicaciones web como es el WPO, pero todo esto queda un poco en segundo plano comparado con la experiencia de haber *dirigido* por mi cuenta un proyecto web desde su planteamiento hasta su puesta en producción, pasando por las fases de análisis, diseño, implementación y testeo del mismo.

8.2. Conclusiones Técnicas

El punto de vista técnico tiene vital importancia en este proyecto, ya que la decisión inicial fue elegir el framework adecuado para desarrollarlo. Era una decisión importante, ya que este tenía que cubrir todas las necesidades del desarrollo, tener una curva de aprendizaje razonable y demostrar que era útil.

Personalmente creo que el conjunto de utilidades y facilidades de las que provee Symfony2 hacen el desarrollo bastante más simple. Deja al programador que se centre en la tarea más importante: plasmar la lógica de negocio en forma de código. El resto de aspectos técnicos como el filtrado de variables, la gestión de errores, el control de usuarios... etc, queda relegado al framework, que ofrece facilidades para desarrollar estos aspectos.

Por otra parte, la forma de trabajar con Symfony simplifica mucho el diseño de la aplicación, ya que el desarrollador solo debe pensar en objetos y las interacciones entre ellos, sin preocuparse en otras cosas como por ejemplo la base de datos, ya que Doctrine se encargará de crear la base de datos adecuada para dar soporte al proyecto.

Una vez empezado el desarrollo, realizar cambios en el esquema de la base de datos es simple, basta con modificar las clases que representan las entidades y ejecutar una orden para actualizar el esquema, sin más.

Llenar la base de datos de datos de prueba es muy sencillo también mediante el uso de fixtures con Doctrine. Es muy cómodo poder recargar por completo la base de datos después de realizar algunas pruebas.

Toda esta gestión de la base de datos automatizada facilita mucho el desarrollo en general, ya que nos abstrae mucho de los pormenores o el trabajo repetitivo y pesado.

Uno de los mayores temores que tenia era si el rendimiento de la aplicación quedaría penalizado por el propio peso del framework, ya que estamos hablando de un sistema bastante complejo y pesado y esto puede repercutir en el rendimiento de la aplicación.

Es bien sabido que los sistemas CMS más populares adolecen de este mismo problema: Drupal acaba volviéndose lento y pesado con el paso del tiempo y el aumento del tamaño de la base de datos, y Joomla no queda atrás en este aspecto.
Otros sistemas, como los de e-commerce, especialmente Magento o Prestashop, también son lentos en muchos aspectos, y esto siempre se agrava con el aumento del tamaño de la base de datos.

Claro está que este no era el mismo caso, con Symfony2 es el desarrollador el que diseña la base de datos, aunque después sea Doctrine el que haga las consultas y la mantenga, el diseño es un punto muy importante que denota si la estructura de la base de datos ha sido correctamente definida. También Doctrine colabora a realizar las consultas de manera más eficiente, y a tratar los datos devueltos mediante modelos de hidratación que se ajusten lo mejor posible al tipo de datos devueltos, evitando crear objetos innecesarios en el código.

Leyendo esto se podría pensar que simplemente integrando Doctrine en mis proyectos web obtendría ya la estabilidad de la que habló, pero Symfony no solo da estabilidad a nivel de persistencia, sino que con sus sistemas de caché, plantillas, gestores de *assets* y demás componentes, hacen el sistema todo lo robusto que puede ser.

Otro punto curioso que quiero mencionar es que durante el desarrollo de este proyecto he estado conociendo y estudiando los sistemas de API's RESTful. Los sistemas REST son webservices que basan el acceso a los recursos de sistema en URL's que identifican de manera unívoca el recurso dentro del sistema, basando las acciones a realizar sobre el mismo según el tipo de petición HTTP (GET, PUT, POST, DELETE,...) .

Estos sistemas son a día de hoy ampliamente utilizados por grandes empresas como Google, Facebook, Amazon, Twitter, en sustitución de otros servicios más pesados basados en arquitecturas como SOAP.

Después de mucho investigar sobre este tipo de arquitectura de servicios, empecé a pensar en cómo implementar una API REST con Symfony, y me sorprendí al darme cuenta de lo realmente sencillo que podía llegar a ser, y después de buscar, encontré casos reales de API's REST creadas con Symfony por empresas locales, y tuve la suerte de poder hablar con los desarrolladores y compartir un poco las experiencias y debatir sobre los pros y los contras.

Con esto quiero decir que Symfony es un sistema que viene preparado para quedarse mucho tiempo ya que como un ponente dijo en una conferencia sobre desarrollo de webservices "*hablar de PHP a día de hoy es hablar de Symfony*".

9. Bibliografía

En el siguiente apartado se va a hacer referencia a la bibliografía usada para la composición de este documento y el desarrollo del proyecto.

[1] *Manual de instalación de Symfony 2.1* [online] disponible en http://symfony.es/documentacion/guia-de-actualizacion-de-symfony-2-1/

[2] *Manual de instalación y uso de Composer* [online] disponible en http://symfony.es/documentacion/guia-de-instalacion-de-composer/

[3] *Manual de instalación y uso de Git* [online] disponible en http://symfony.es/documentacion/guia-de-instalacion-de-git/

[4] *Manual de configuración de Apache para Symfony2* [online] disponible en http://symfony.es/documentacion/como-configurar-bien-apache-para-las-aplicaciones-symfony2/

[5] *Referencia de filtros y funciones de Twig* [online] disponible en http://twig.sensiolabs.org/documentation

[6] *Documentación oficial de Composer* [online] disponible en http://getcomposer.org/doc/

[7] *Desarrollo Web ágil con Symfony2* [epub, pdf, papel] libro autopublicado, disponible en http://symfony.es/libro/

[8] *Documentación oficial del proyecto Symfony2* [online] disponible en http://symfony.com/doc/current/index.html

[9] *Libro de recetas rápidas de Symfony2* [online] disponible en http://symfony.com/doc/current/cookbook/index.html

[10] *Documentación de jQuery* [online] disponible en http://api.jquery.com/

[11] *Documentación de PHPUnit* [online] disponible en http://www.phpunit.de/manual/3.7/en/

[12] Aprende jQuery 1.3 [papel] de Jonathan Chaffer y Karl Swedberg, Anaya Multimedia ISBN 978-84-415-2665-5

[13] Integrando Ajax en Symfony2 [online] disponible en http://www.maestrosdelweb.com/editorial/curso-symfony2-integrando-ajax/

[14] Google PageSPeed, Make the web faster [online] disponible en https://developers.google.com/speed/pagespeed/?hl=es

[15] Yahoo YSlow Web Performace Best Practices and Rules [online] disponible en http://yslow.org/

[16] Highcharts API Documentation [online] disponible en http://api.highcharts.com/highcharts

[17] Highstock API Documentation [online] disponible en http://api.highcharts.com/highstock

[18] Documentación de jQuery UI [online] disponible en http://api.jqueryui.com/

[19] Guía Web Performace Optimization de Javier Casares http://javiercasares.com/wpo

[20] Steve Souders Web Performance Optimization [online] disponible en http://www.stevesouders.com/blog/2010/05/07/wpo-web-performance-optimization/

[21] Apache Server Configuration Files [online] disponible en http://httpd.apache.org/docs/2.2/configuring.html

[22] Apache Server VirtualHost Examples [online] disponible en http://httpd.apache.org/docs/2.2/configuring.html

[23] Marissa Mayer at Web 2.0 [online] disponible en http://glinden.blogspot.com.es/2006/11/marissa-mayer-at-web-20.html

[24] The Secret Weapons of the AOL Optimization Team [online] disponible en http://es.scribd.com/doc/16878352/The-Secret-Weapons-of-the-AOL-Optimization-Team

[25] The User and Business Impact of Server Delays, additional bytes and HTTP chunking in web Search [online] disponible en http://velocityconf.com/velocity2009/public/schedule/detail/8523

[26] The User and Business Impact of Server Delays, additional bytes and HTTP chunking in web Search [online] disponible en http://velocityconf.com/velocity2009/public/schedule/detail/8523

[27] We're all Guinea Pigs in Google's Search Experiment [online] disponible en http://news.cnet.com/8301-10784_3-9954972-7.html

[28] Mozilla Blog of Metrics [online] disponible en http://blog.mozilla.org/metrics/category/website-optimization/

[29] Improving Netflix Performance [online] disponible en http://velocityconf.com/velocity2008/public/schedule/detail/3632

[30]Shopzilla's Site Redo, You get what you measure [online] disponible en http://velocityconf.com/velocity2009/public/schedule/detail/7709

10. Glosario de Términos

- **AES**
  - Advanced Encryption Standard
  - Es un algoritmo simétrico de cifrado. Convertido en estándar por el gobierno de Estados Unidos en 2002.
- **APC**
  - Alternative PHP Cache
  - Sistema de caché alternativo para PHP, aumenta notablemente el rendimiento de las aplicaciones cuando se hace buen uso de él.
- **BASE64**
  - Sistema de numeración posicional que usa 64 como base
  - Ampliamente usado para codificar información de gran tamaño en pocos bits.
- **CASE**
  - Computer Aided Software Development
  - Herramientas para el desarrollo de software asistido por computador
  - Mejoran la productividad y simplifican el diseño y el modelado.
- **CSS3**
  - Cascade Style Sheets, versión 3
  - Lenguaje usado para la definición de estilos de presentación de documentos HTML o XML.
- **CRUD**
  - Create, Read, Update, Delete
  - Conjunto básico de operaciones que se puede realizar sobre una base de datos
- **ERS**
  - Especificación de Requisitos Software
- **HTML5**
  - HyperText Markup Language, versión 5.
  - Quinta revisión del lenguaje HTML. Todavía en fase de desarrollo
- **IEEE**
  - Institute of Electrical and Electronic Enginers
  - Asociación técnico-profesional mundial dedicada a la estandarización entre otras cosas
- **IEEE 830/98**
  - Especificación estándar de la IEEE para las Especificaciones de Requisitos Software o ERS
- **LAMP**
  - Linux, Apache, MySQL y PHP
  - Arquitectura habitual de las aplicaciones web
- **MySQL**
  - Sistema de gestión de bases de datos relacional.
  - Permite integración con PHP de forma nativa
- **PHP**
  - PHP Hypertext Pre-processor
  - Lenguaje de programación de scripting del lado del servidor.

- **SHA-512**
  - Secure Hash Algorithm
  - Algoritmo de resumen con una salida de 512 bits
- **SI**
  - Sistema de Información
  - Hace referencia al conjunto de datos + lógica de negocio
- **UML**
  - Unified Modeling Language
  - Lenguaje gráfico para el modelado y análisis de software.
  - Muy extendido y estándar desde 2005
- **VIN**
  - Vehicle Identification Number
  - Formato internacional que define el número de chasis para la identificación de vehículos.