Universidad Zaragoza

Proyecto Fin de Carrera

Aplicación web PHP completa y automatizada para alquiler de vehículos

Autor

David Velilla Alegre

Directores

Paula Lema Camean
José Manuel Bermejo Ruiz

Ponente

Manuel González Bedia

Escuela de Ingeniería y Arquitectura
2013

Aplicación web PHP completa y automatizada para alquiler de vehículos

RESUMEN

El alquiler de vehículos tradicional está en declive. La necesidad de mantener personas físicas en cada una de las oficinas donde haya vehículos, sumada a las esperas por parte del cliente mientras toman sus datos y escanean sus documentos de identidad, más la imposibilidad de reservar menos de 24h, hacen que la alternativa, el denominado “Car Sharing”, esté en auge.

Con este sistema, desde una aplicación web se llevan a cabo las verificaciones de identidad que permiten dar de alta un usuario, que después podrá realizar reservas en menos de un minuto y conducir de inmediato. Además, se puede reservar el tiempo deseado: sólo se paga por lo que se conduce. El coche recibe la orden a través de su router 3G, y permite que el usuario, con su tarjeta magnética, abra el vehículo a la hora adecuada.

Este proyecto emprendedor de la empresa de innovación tecnológica FringesCT ha cubierto el análisis, diseño e implementación de una aplicación web completa para poner en funcionamiento este nuevo paradigma de alquiler de vehículos en nuestro país.

La implementación se ha realizado usando el framework para PHP Symfony (en su versión 2.0), que facilita el modelo vista controlador y el desarrollo modular, poniendo a tu disposición múltiples herramientas como Twig: un lenguaje generador de plantillas que facilitan la interacción con la base de datos. Doctrine, que acerca las entidades de la base de datos al modelo, de modo que las sentencias SQL son generadas a partir de funciones DQL en PHP. Symfony2 también aleja los ficheros de configuración, parámetros y aspectos de la seguridad y control de acceso del resto de la aplicación en ficheros YAML o XML.

La aplicación es completa y autosuficiente para controlar toda la parte software del sistema carsharing (el hardware instalado en los vehículos no forma parte de este proyecto). La base de datos diseñada consta de 18 tablas, y se ha implementado utilizando MySQL. De cara a un administrador, la aplicación ofrece un back-end para interaccionar con las distintas entidades de la base de datos con las restricciones impuestas por los requisitos.

De cara al usuario, éste puede realizar las funciones que se podrían esperar de un front-end, como registrarse, realizar búsquedas de vehículos en las localizaciones que desee y reservarlos, modificar y cancelar reservas cuando se le permita, gestionar sus datos, abrir incidencias...

La aplicación también genera las facturas de los clientes dependiendo de la forma de pago elegida (domiciliación bancaria o tarjeta de crédito), la tarifa escogida por el usuario (normal, premium, ...), la categoría del vehículo (deportivo, familiar, económico, ...), la duración de reserva y distancia recorrida. Las facturas generadas son almacenadas en el servidor y enviadas por email a los clientes. También se mandan emails a los administradores para advertir de múltiples eventos, como nuevos usuarios registrados, incidencias abiertas por usuarios, adjuntos de un permiso de conducción subidos por un usuario para que se le valide y se le deje hacer reservas...

Resumiendo, el proyecto se ha basado en los siguientes puntos: diseño, implementación, e instalación en el servidor de la aplicación web en PHP que permite una funcionalidad completa y automatizada del sistema de carsharing descrito.

Índice general

I. Memoria

II. Anexos

Índice de figuras

Parte I

MEMORIA

Capítulo 1

Introducción

1.1 El “Car Sharing” como nuevo modelo de transporte

El alquiler de vehículos tradicional está en declive. La necesidad de mantener personas físicas en cada una de las oficinas donde haya ubicaciones de vehículos, sumada a las esperas por parte del cliente mientras toman sus datos y escanean sus documentos de identidad, más la imposibilidad de reservar menos de 24h, hacen que la alternativa, el denominado carsharing, esté en auge.

Con este sistema, desde una aplicación web se llevan a cabo las verificaciones de identidad que permiten dar de alta un usuario, que después podrá realizar reservas en menos de un minuto y conducir de inmediato. Además, se puede reservar el tiempo deseado: sólo se paga por lo que se conduce. El coche recibe la orden a través de su router 3G, y permite que el usuario, con su tarjeta magnética, abra el vehículo a la hora adecuada.

El carsharing está extendido por Estados Unidos y ganando fuerza en Europa. La empresa referente a nivel mundial es Zipcar, que para septiembre de 2012 contaba con una flota de 11.000 vehículos y 730.000 miembros registrados [1]. Para aquellos conductores que no necesiten un vehículo más que para pequeños desplazamientos diarios, o para un único largo desplazamiento al año (e.g vacaciones de verano), la alternativa del carsharing ofrece un gran ahorro frente a los grandes costes de mantener un vehículo todo el año: el precio del vehículo mismo, el seguro, impuesto de circulación, plaza de garaje si no se desea tener en la calle, etc.

La empresa de innovación tecnológica FringesCT, tras un estudio de mercado de las posibilidades de este nuevo paradigma de alquiler de vehículos en nuestro país, ha puesto en marcha un proyecto para iniciar un servicio de carsharing en España. Este proyecto tratará de llevar a cabo el análisis, diseño e implementación de una aplicación web completa para poner en marcha el sistema carsharing con el hardware que la empresa adquiera e incorpore a los vehículos.

1.2 Herramientas a utilizar durante el proyecto

El proyecto se realizará usando el framework para PHP Symfony (en su versión 2.0), que facilita el modelo vista controlador (MVC) y el desarrollo modular, poniendo a tu disposición múltiples herramientas como Twig: un lenguaje generador de plantillas que facilita la interacción con la base de datos. Doctrine, que acerca las entidades de la base de datos al modelo, de modo que las sentencias SQL son generadas a partir de funciones DQL en PHP. Symfony también aleja los ficheros de configuración, parámetros y aspectos de la seguridad y control de acceso del resto de la aplicación en ficheros YAML o XML [2]. Todas estas facilidades que ofrece Symfony pueden no obstante ser sustituidas por el uso exclusivo de PHP, aunque este proyecto se ha beneficiado de sus

claras ventajas.

Se estudiaron otras posibilidades para la implementación, entre ellas usar el lenguaje de programación Ruby mediante el framework Ruby-on-Rails, o el mismo PHP mediante el popular framework Zend. La primera se descartó por un mayor conocimiento inicial de PHP frente a Ruby, la segunda por contar Symfony con unos manuales explicativos del framework que no sólo te muestran las diferentes funcionalidades que ofrece con ejemplos, sino que también explican al usuario inexperto todo el esquema conceptual del Modelo Vista Controlador en una aproximación muy didáctica que puede resultar positiva para quien busca introducirse en este tipo de tecnologías web.

Se empleará el IDE (Integrated Development Environment) NetBeans 7.1 para poder tener el código más fácilmente accesible y de paso contar con su detección de errores para sintaxis PHP. En realidad es una herramienta que aporta comodidad, pues al ser ésta una aplicación web, no se necesitará más que un navegador abierto donde pulsemos F5 (actualizar) para ver los cambios.

Con respecto a los navegadores, se emplearán Internet Explorer (versión 7 o superior), Firefox, Chrome, Safari y Opera para comprobar la validez del código escrito, siendo especialmente útiles sus respectivos modos para desarrolladores a la hora de depurar html y hojas de estilos css.

La metodología de trabajo a seguir a lo largo del desarrollo del proyecto será Scrum[3], proponiendo objetivos concretos que ir cumpliendo y avanzar así en la lista de requisitos de la aplicación, con reuniones periódicas de control con los responsables de la empresa.

1.3 Responsables del proyecto

El proyecto se realiza bajo la guía de los responsables técnico, Paula Lema Camean, y financiero, José Manuel Bermejo Ruiz, de la empresa FringesCT, siendo estos los responsables de la planificación de los plazos del proyecto, así como de aportar todo el conocimiento relacionado con sistemas carsharing fruto de sus estudios de negocio y de mercado.

Una vez con la idea clara de qué hacer -el ámbito del proyecto-, todo el código fuente, así como las decisiones de desarrollo e implementación de las distintas partes de éste software, son fruto del trabajo del autor de ésta memoria, David Velilla Alegre.

Por último, la instalación (deploy en el servidor contratado) se realiza con la asesoría de David Navarro Estruch, ingeniero informático y matemático por la Universidad Autónoma de Madrid además de buen amigo mío, a quien agradezco todo lo aprendido respecto a configuración de servidores, mailers, SMTP, hostings dedicados vs compartidos, certificados SSL, etc.

1.4 Objetivos del proyecto

El proyecto tiene como objetivo el diseño e implementación de la aplicación web en PHP que permita una funcionalidad completa y automatizada del sistema descrito, y su posterior instalación (deploy) en un servidor contratado a tal efecto por la empresa FringesCT para la puesta en marcha del servicio carsharing.

Resumiendo la lista de objetivos del proyecto, éstos se pueden dividir en los referentes al

back-end (parte de administración de la aplicación) y front-end (parte de usuario, ya sea un miembro registrado o cualquier visitante de la web).

Desde el punto de vista de los administradores, desde la aplicación se podrá gestionar la apertura de vehículos y el cierre automático cuando termine la reserva o el cliente elija. Se ha de poner en marcha un sistema de alertas mediante envío de emails a los administradores: cuando un coche esté alejado del lugar donde debe estacionar; cuando se pierda comunicación con algún coche; cuando se aproximan las fechas de mantenimiento o ITV; cuando se ha registra un nuevo cliente; cuando un cliente registra un nuevo conductor; cuando a un conductor le va a expirar pronto el permiso de conducción y aún no ha enviado la fotocopia del renovado; cuando no le quedan puntos en su permiso, etc. Además, todo cambio en la base de datos será registrado asimismo en un log o histórico de cambios.

Existirán distintos grupos administrativos para poder abstraer al usuario de otras localizaciones que los gestores quieran ampliar. Por ejemplo, si la actividad empresarial se desarrolla en Madrid pero se decide ampliar a Zaragoza, los nuevos usuarios de Zaragoza tendrán distintas tarifas acordes a los precios de cada ciudad. La aplicación tampoco tiene que estar restringida al territorio nacional, no habrá problema en añadir un grupo para otros países, si se desea. La web se ofrecerá en castellano e inglés, y se dejará documentado cómo añadir nuevos idiomas.

Por tanto, cada grupo tendrá sus administradores, usuarios, tarifas, coches, ubicaciones, promociones, información de reservas de sus usuarios y posibles multas de tráfico, y su propia gestión de incidencias. Un administrador debe poder interactuar con todas estas entidades de su grupo, con una interfaz web adecuada que facilite las tareas de gestión.

La aplicación también generará las facturas de forma automática, dependiendo de la forma de pago elegida por un usuario, que puede ser mensualidad a su cuenta corriente, o reserva por reserva mediante tarjeta de crédito. Para facturar se tomarán en cuenta múltiples factores como la tarifa del usuario, si ésta tiene una componente mensual, posible reducción de la franquicia del seguro que el usuario pueda escoger contratar, duración de reserva en días y/o horas y minutos, distancia recorrida. Para la extracción de la distancia será necesario preguntar al coche cada poco tiempo por su posición GPS, y calcularla al final. Las facturas se almacenarán en el servidor y serán enviadas por email a los clientes, que podrán descargarlas además desde la aplicación.

Un usuario podrá registrarse, realizar, modificar y cancelar reservas, gestionar sus datos, abrir incidencias, reportar problemas, introducir códigos de promociones, etc.

Resumiendo los objetivos, Front-end y Back-end tendrán funcionalidad completa, y la apariencia de la web será profesional, teniendo que generarse todas las hojas de estilos CSS a tal efecto. La aplicación deberá funcionar correctamente en los distintos navegadores, incluyendo móviles, aunque no será necesario realizar una interfaz especial para estos dispositivos (app).

1.5 Estructura del documento

Las siguientes secciones de este documento tratarán de abordar las diferentes fases que se han seguido a lo largo del proyecto junto con las complicaciones a las que se ha tenido que hacer frente durante las mismas.

El framework Symfony tiene un papel fundamental en este proyecto, y como tal se le dedicará un capítulo, donde también se repasarán los fundamentos de una aplicación web. Como punto explicativo del modo de proceder que se siguió para fraccionar el listado de requisitos en algo mucho más claro y tangible, se proseguirá con una exposición del diseño de la base de datos, sus entidades, y los mapas de rutas de la aplicación.

A continuación se pasará a una descripción de los diferentes controladores PHP, formularios, plantillas Twig-HTML-CSS, adaptación multilenguaje, javascripts, históricos de cambios, alertas por email... es decir, toda una serie de elementos que es necesario aprender a utilizar para el desarrollo de una aplicación web; una numerosa lista de herramientas que hay que llegar a dominar para que, como si de piezas de un puzzle se trataran, se pueda construir un software profesional de este tamaño y características.

Por último, se expondrán los resultados obtenidos y algunas posibilidades de ampliación a este trabajo que quedarían abiertas para futuros desarrollos.

Capítulo 2

Marco de trabajo: Symfony

2.1 Symfony: aprendizaje del framework

Symfony (en su versión 2.0) es el framework en cuyo marco se ha llevado a cabo todo el desarrollo. Por ello, merece una sección introductoria del mismo, resaltando las ventajas que ha aportado al proyecto.

Como se ha adelantado en la introducción, Symfony facilita el modelo vista controlador (MVC) y el desarrollo modular, poniendo a tu disposición múltiples herramientas como Twig: un lenguaje generador de plantillas que facilita la interacción con la base de datos. Doctrine, que acerca las entidades de la base de datos al modelo, de modo que las sentencias SQL son generadas a partir de funciones DQL en PHP. Symfony2 también aleja los ficheros de configuración, parámetros y aspectos de la seguridad y control de acceso del resto de la aplicación en ficheros YAML o XML [2].

En primer lugar, la decisión de utilizar Symfony2 como marco de trabajo para el proyecto viene principalmente de su gran facilidad de uso para el usuario no experto en el trabajo con frameworks. Su curva de aprendizaje es muy suave, permitiendo enseguida que un usuario vagamente familiarizado con el desarrollo web básico, con nociones de HTML y CSS, pueda construir poco a poco un sistema que va creciendo con el tiempo sin desbordar al programador.

La documentación de Symfony, que puede encontrarse online (y gratuita, como todo en Symfony), empieza con una introducción a los fundamentos de la programación web: las peticiones y respuestas HTTP, para pasar a explicar las ventajas aportadas a programar con PHP plano. Sólo con leer hasta aquí cualquier desarrollador poco familiarizado con el uso de frameworks estará convencido de la ventaja enorme que aporta cuando se trata de construir un sistema de tamaño medio-grande.

Siguiendo las instrucciones de la documentación de Symfony, se procedió a descargar e instalar el mismo tras configurar en local sobre Linux un servidor Apache 2.0 con PHP 5.3 y MySQL.

Symfony permite el uso de una consola de comandos que facilitan algunas tareas de desarrollo. Ejecutando el comando: “`php app/console`” desde la ruta raíz de Symfony obtenemos una lista con todas las posibilidades ofrecidas. Por ejemplo, no hay que mantener actualizadas las sentencias SQL de generación de las tablas de la base de datos. Symfony ofrece un comando que crea una base de datos con los parámetros introducidos en su fichero de configuración. Y aún más útil, otro comando crea o actualiza las tablas en la base de datos acorde a la información (metadata) que se desee colocar en las entidades php que serán el modelo. Es decir, sólo hay que asegurarse de crear las clases entidad: Symfony será quien se encargue de crear la estructura de la base de datos construyendo las sentencias SQL pertinentes. Esta abstracción permite migrar sin ningún coste entre

sistemas gestores de bases de datos.

Para poder probar de inmediato los conceptos aprendidos de la documentación de Symfony y entender mejor el MVC, se procedió a descargar el curso “deSymfony” [4], una serie de ponencias realizadas anualmente en España sobre el uso de Symfony que facilitan enormemente el entendimiento del mismo realizando paso a paso una aplicación sencilla pero completa.

2.2 Modelo Vista Controlador con Symfony

Como ya se ha mencionado, Symfony facilita el uso del MVC al desarrollador. A continuación se va a explicar brevemente cómo trabaja desde que recibe una petición HTTP hasta que da una respuesta, generalmente en forma de página web.

El motor de Symfony recoge las rutas de las peticiones (request) que le llegan al servidor, y las compara con su tabla de rutas, que se sitúa en ficheros de routing. Si detecta un patrón que encaja con una ruta reconocida, ejecuta la función del 'Controlador' correspondiente a esa ruta. El controlador es una función php cuyo objetivo final (su return) es devolver una respuesta HTTP a la petición recibida. Ésta respuesta suele contener simplemente una página web, que tendrá que ser renderizada a partir de una plantilla. Aquí entra en juego la 'Vista', que a su vez podría necesitar de información de la base de datos. Por ejemplo, si el usuario le pide al servidor que le muestre su información personal, preferencias de su cuenta, etc. Éste sería el 'Modelo'.

Una vez explicadas estas nociones, se puede ver que una aplicación web enmarcada en Symfony consta de construir cuantas entidades sean requeridas por el modelo la base de datos, más la jerarquía de rutas y los controladores que las atenderán, más las plantillas a renderizar. De querer ampliar la aplicación, basta con añadir nuevas rutas, controladores y plantillas.

2.3 Esqueleto de una aplicación construida sobre Symfony

En el siguiente apartado se va a indicar *qué* va *dónde* en el esqueleto facilitado por Symfony. De esta manera cuando más adelante en el documento se hable de que “se ha introducido una nueva ruta que ha llamado a cierto controlador”, se podrá entender mejor qué es lo que se está haciendo realmente, y sobre qué ficheros en particular podrían encontrarse estos cambios.

CarSharing - *Directorio raíz de la aplicación.*
- **app**
  - **cache** - *Cache de la aplicación. Symfony la gestiona.*
  - **Config**
    - security.yml - *Fichero de seguridad. Autenticación y permisos.*
    - parameters.ini - *Parámetros definidos (mailer, motor de la base de datos...)*
  - **logs** - *Logs de la aplicación. Symfony los gestiona.*
  - **Resources** - *En nuestra aplicación, utilizada para almacenar uploads.*
- **bin**
- **nbproject**
- **src** - *Contiene todo el código fuente, agrupado en bundles.*
  - **CarSharingBundle** - *Bundle único de la aplicación.*
    - **Controller** - *Controladores...*
      - BookingController.php - *...de rutas de reservas (e.g: /booking/list-all, /booking/new)*

CarController.php - *...de rutas de coches (e.g: /admin/car/edit/Z1234BM)*
ClientController.php
…

Entity - *Entidades (muchas de ellas con mapeo directo en la bb.dd)*
Booking.php - *Clase, sus atributos y métodos, y metadata para la bb.dd.*
Car.php
Client.php
…

Extension - *Funciones PHP para Twig usadas en varias plantillas.*

Form - *Formularios que se mostrarán en las distintas plantillas*
newBookingType.php
adminEditBookingType.php - e.g: permite editar cualquier atributo de una reserva
clientEditBookingType.php - e.g: sólo edita los campos “fecha inicio” y “fecha fin” (un cliente no ha de poder modificar más cosas)
…

Repository - *Repositorios de funciones de acceso a bb.dd de cada Entidad*
BookingRepository.php
CarRepository.php
ClientRepository.php

Resources

config - *Archivos de rutas*
routing.yml
routing_booking.yml
routing_car.yml
…

translations - *Ficheros de textos mostrados en las plantillas*
messages.en.yml - *Todos los textos de la aplicación, en inglés*
messages.es.yml

views - *Plantillas...*

Booking - *...referentes a reservas.*
edit.html.twig
new.html.twig
list.html.twig
show.html.twig

Car -*...referentes a coches.*
edit.html.twig
…

…

vendor - *Bundles de terceros, como Doctrine, Twig, Swiftmailer...*

doctrine
twig
swiftmailer
…

web - *Recursos web*

css - *Hojas de estilos (CSS: Cascading Style Sheet)*
img - *Imágenes públicas*
js - *Scripts (javascript, jQuery, ...)*

Este es el esqueleto de Symfony en el caso particular de la nuestra aplicación. La estructura de carpetas, de cuyos direccionamientos internos se ocupa el motor de Symfony, permite abstraerse de problemas que no sean el desarrollo mismo de la aplicación cumpliendo requisito a requisito.

2.4 Doctrine y los repositorios de entidades

Doctrine es un conjunto de librerías PHP centradas en proporcionar servicios de persistencia. Sus principales proyectos son ORM (Object Relational Mapper), y el nivel de abstracción de la base de datos sobre el que está construido[9].

El primero permite generar el SQL necesario para la creación de una base de datos con todas sus tablas y relaciones a partir de la información de mapeo de las entidades. El segundo, realizar consultas a la base de datos utilizando DQL (Doctrine Query Language) en lugar de SQL, el cual está más orientado a objetos, y que ofrece la ventaja de permitir portabilidad entre sistemas gestores de bases de datos sin que haya que adaptar todo el SQL.

2.4.1 Creación de la base de datos a partir de la consola de Symfony

Para crear una base de datos desde la consola de Symfony necesitaremos tener configurado el fichero “app/config/parameters.ini” con el controlador, host, puerto, y credenciales del sistema gestor de bases de datos deseado. A continuación la configuración que se utilizó durante el desarrollo en local. (Nota: dicho fichero se excluyó del control de versiones Subversion, de manera que el servidor tuviera su propia configuración de la base de datos y no hubiera que reconfigurarlo tras cada commit/update).

```
database_driver="pdo_mysql"
database_host="localhost"
database_port=""
database_name="CarSharing"
database_user="root"
database_password="********"
```

Con este fichero listo, se ejecuta el siguiente comando de la consola de Symfony desde el directorio raíz del proyecto (aquél que contiene app, src, vendors... consultar capítulo 1.3 para más información): `app/console doctrine:database:create`

2.4.2 Creación de tablas a partir de meta-datos en las entidades php

El siguiente fragmento de código muestra una pequeña parte de la entidad Coche. De las entidades y cómo se ha llegado a su diseño se hablará con más detalle en el capítulo 3.2. Lo que pueden parecer simples comentarios sobre código php, son en realidad meta-datos utilizados por Doctrine para la generación de la base de datos.

```
/**@ORM\Table(name="car")
 * @ORM\Entity
 * @ORM\Entity(repositoryClass="FringesCT\CarSharingBundle\Repository\CarRepository") */
class Car
{
   /**@ORM\Id
    *  @ORM\Column(type="integer")
    *  @ORM\GeneratedValue(strategy="IDENTITY") */
   protected $id;

   /**@ORM\ManyToOne(targetEntity="AdminGroup", inversedBy="cars")
    *  @ORM\JoinColumn(name="idAdminGroup", referencedColumnName="id")  */
   protected $adminGroup;

   /**@ORM\Column(type="string", length="20")  */
   protected $category;
```

```
    /**@ORM\Column(type="date") */
    protected $next_ITV;

    /**@ORM\Column(type="integer", nullable="true")
     * @Assert\Min(0)
     * @Assert\Max(100) */
    protected $fuelLevel = NULL;

    [...]
}
```

Resumiendo, se le está diciendo a Doctrine que cree una tabla llamada “car” mapeada por esta entidad, que tendrá las funciones de consulta en el fichero CarRepository. Su identificador será el atributo “id”, de tipo entero “integer”, tendrá una columna de tipo texto con capacidad máxima para 20 caracteres llamada “category”, otra columna de tipo fecha “date” llamada “next_ITV”, y otra de tipo entero que tendrá que estar comprendido entre 0 y 100, y que podrá ser NULL.

Además, y esto es lo más potente que ofrece Doctrine de cara a la programación orientada a objetos, la entidad tendrá una relación N→1 hacia la entidad AdminGroup, invertida por el atributo “cars”. Es decir, se podrá obtener el objeto “grupo administrativo” con un simple método get, que no te devolverá el identificador del grupo (lo que se almacena realmente en la base de datos y que sirve para referenciar), sino que devolverá el objeto entero. De igual manera, cualquier objeto “AdminGroup” podrá hacer un get sobre su atributo “cars” y conseguir un vector con todos los objetos “Car” que lo tengan como grupo administrativo.

Para decir a la consola de Symfony que cree las tablas de la base de datos a partir de los meta-datos, hay que escribir desde la ruta raíz de la aplicación el siguiente comando: `app/console doctrine:schema:create`

2.4.3 DQL: Repositorios de entidades

La mejor forma de ver lo que aporta el DQL (Doctrine Query Language) a la aplicación es estudiar una función del repositorio BookingRepository, la función php que devuelve todas las reservas del grupo administrativo introducido desde la fecha $from hasta la fecha $to.

Como se puede ver, la consulta es una string muy similar a la sintaxis SQL [10], a la que le introducimos los parámetros “grupo”, “desde” y “hasta” con funciones PHP sobre los placeholders introducidos en la cadena como palabras precedidas por dos puntos (:)

En lugar de utilizar nombres de columnas de las tablas, gracias a Doctrine podemos abstraernos por completo de esa capa y utilizar entidades PHP y sus atributos. La llamada a “createQuery” convertirá la consulta en el SQL requerido por el sistema gestor de bases de datos que utilices, y “getResult” lanzará esa consulta, y te devolverá los resultados como un array de objetos de la clase entidad que estás buscando.

```
public function findBookingsBetweenDates($group, $from, $to) {
  $em = $this->getEntityManager();
  $consulta = 'SELECT x FROM CarSharingBundle:Booking x,CarSharingBundle:Car y,
                              CarSharingBundle:Driver  z,  CarSharingBundle:Client  w,
                              CarSharingBundle:Place   p
              WHERE x.car = y  AND  x.driver = z  AND   z.client = w  AND   y.place  = p
                        AND   y.adminGroup = :group   AND   x.startDateTime >= :from
                        AND   x.startDateTime <= :to   ORDER BY x.startDateTime DESC';
  $query = $em->createQuery($consulta) ->setParameter('group', $group)
                      ->setParameter('from', $from)->setParameter('to', $to);
  return $query->getResult();}
```

Capítulo 3

Diseño de entidades y sus relaciones

3.1 Esquema de la base de datos

A partir de los requisitos se tenía una idea del problema en su conjunto, y la mejor forma de poner sobre el papel esta idea era en forma de un esquema de base de datos. Tras varios diagramas en sucio sobre las entidades necesarias y las relaciones entre ellas, se llegó al modelo relacional que se puede apreciar en la figura 1. La notación seguida refleja el identificador de cada entidad en negrita.

Por orden alfabético, las entidades resultantes son: AdminGroup (grupo administrativo), Bill (factura), Booking (reserva), Car (coche), CarPosition (posición gps de un coche), Card (tarjeta de crédito/débito), Client (cliente, usuario registrado), Driver (conductor), Fare (tarifa), Fine (multa), Incident (incidencia), IncidentNote (comentario de una incidencia), Manager (administrador), Promotion (promoción).

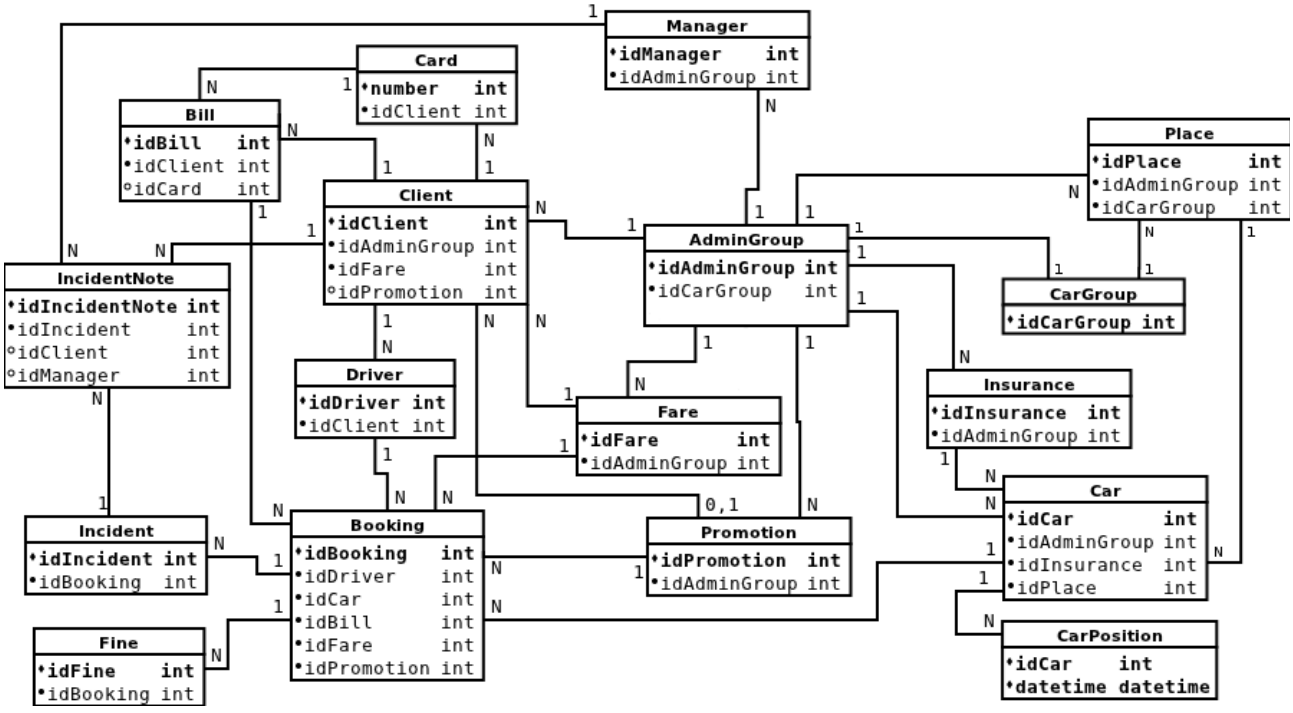


Figura 1: Esquema de la base de datos

Se pueden contar un total de 17 entidades. Como se verá más adelante, finalmente se decidieron implementar los históricos de cambios de la aplicación en una tabla de la base de datos, por su sencillez a la hora de buscar y filtrar sucesos. Con ello, al final la base de datos cuenta con 18 tablas.

3.2 Entidades

Las entidades pensadas para la base de datos tienen su correspondencia directa con una clase php con su nombre. Es más: como se ha mencionado en el capítulo introductorio de Symfony, un comando de la consola de éste permite la creación de las sentencias SQL para la formación de la base de datos sólo a partir de las clases entidad en php.

A continuación se listan los atributos de cada una de las entidades, describiendo las razones que han llevado a su diseño, con énfasis en aquellas que no sean tan intuitivas de ver.

3.2.1 Grupo administrativo (AdminGroup)

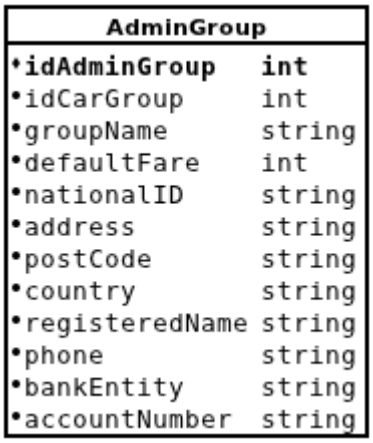


Figura 2: Grupo Administrativo

Los grupos administrativos tendrán, aparte de su propio identificador, un grupo de coches. Varios grupos administrativos pueden tener el mismo grupo de coches. Esto será útil en el caso de que se quiera permitir, por ejemplo, tanto a usuarios de “Zaragoza” y “Huesca” ver y reservar coches de ambos grupos manteniendo las tarifas, managers, etc, independientes.

Aparte del atributo groupName, el nombre del grupo, i.e “Zaragoza”, el resto son atributos necesarios para la facturación. Cada grupo constituye una entidad independiente, que necesita una razón social, teléfono, dirección, cuenta bancaria y el banco al que pertenece.

3.2.2 Factura (Bill)

Las facturas deben estar mantenidas por motivos legales un mínimo de 5 años en la base de datos. Primero se pensó en que cuando se generase físicamente una factura, es decir, se extrajese la información de reservas y se crease un excel/pdf/el formato que se pensase, almacenar tan sólo el mismo fichero. Después pensamos que no costaba nada crear una entidad adicional y almacenar parte de esta información, para que los managers pudieran llevar el control de cuándo una factura había sido pagada y cuando no.

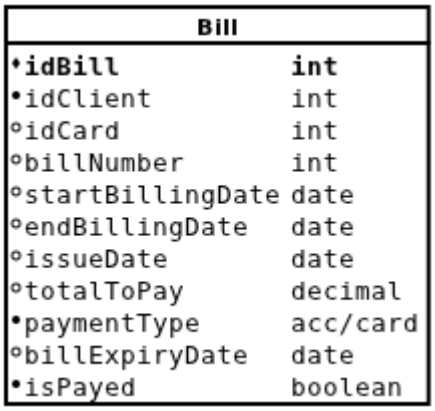


Figura 3: Factura

Los atributos precedidos por una circunferencia blanca en lugar de un punto negro relleno, significan que pueden ser nulos. En este caso, idCard puede ser nulo si la reserva no se ha pagado con una tarjeta sino con cuenta corriente. El resto de atributos pueden ser nulos si no se ha generado todavía realmente la factura. Esto sucede en el caso de pago por tarjeta. Cuando se hace una reserva, con “tarjeta” como forma de pago, se crea también una entidad Factura, y la reserva queda enlazada a ella. Entonces, la entidad Factura es enlazada con la entidad Tarjeta que realizará el pago una vez finalice la reserva y se genere la factura “física”.

3.2.3 Reserva (Booking)

Booking	
*idBooking	int
*idDriver	int
*idCar	int
*idBill	int
*idFare	int
°pickUpPlaceComments	string
*idPromotion	int
°bookingNumber	string
*startDateTime	datetime
*endDateTime	datetime
°actualReturnDateTime	datetime
°billingEndDateTime	datetime
*leaveCarInWrongPlaceCharge	decimal
*lowFuelTankCharge	decimal
*sendCarCharge	decimal
*sendCarAddress	string
*extras	decimal
*cdw	decimal
*isActive	boolean
°startKm	int
°endKm	int
°cancelCharge	decimal
°cancelDateTime	datetime

Figura 4: Reserva

La entidad Reserva es una de las más relevantes en la aplicación, como cabe esperar en una aplicación web de reservas de vehículos. Se compone, como se puede apreciar en el diagrama de la base de datos (figura 1), de un Conductor (que a su vez estará en una cuenta de usuario registrado), un Coche, una Factura (generada posteriormente a la reserva), una Tarifa aplicada, y opcionalmente una Promoción.

La tarifa (fare) es necesaria para persistir en el tiempo la reserva sin perder información. La tarifa aplicada a una reserva dependerá de si el cliente pertenece al mismo grupo administrativo que el coche (se pone la tarifa del cliente), o el coche pertenece a otro AdminGroup, que por tener el mismo CarGroup que el AdminGroup del Cliente, se le está mostrando también y por tanto ha podido reservarlo. En este caso, se aplica la tarifa por defecto “default fare” del AdminGroup del coche reservado.

Si el usuario tenía alguna promoción activa a la hora de realizar la reserva, se aplica también en ésta.

Las fechas de inicio y fin de reserva (start y endDateTime) son las seleccionadas por el usuario al reservar. ActualReturnDateTime es la fecha/hora a la que el conductor realmente devuelve el coche. Se utilizará para facturar al cliente si devuelve el coche pronto o tarde. BillingEndDateTime es la hora hasta la que se facturará. Sirve para que si el cliente ha reducido el tiempo de reserva cuando faltaba menos de media hora para el final, se le siga facturando toda como penalización por un cambio tan tardío que no permite a otros usuarios tener tiempo de poder reservar ese vehículo.

Existen unos atributos necesarios para facturación, como los cargos por devolver el vehículo en un lugar distinto al punto de devolución estipulado, por dejar el depósito de combustible a menos de 1/3 de su capacidad, por envío a una dirección de recogida en particular (véase un aeropuerto), y otros posibles extras.

De cancelar la reserva, podría tener que aplicarse un cargo por hacerse con poca antelación (los requisitos exactos se marcaron en las múltiples reuniones mantenidas a lo largo del desarrollo), y entonces habrá que registrarlo en la entidad Reserva.

StartKm y EndKm marcan el kilometraje del vehículo al comienzo y al final de la reserva, lo cual se lee de los vehículos cuando sea posible, o se computa a partir de las posiciones GPS almacenadas cada poco tiempo del vehículo, para trazar la ruta seguida y poder facturar por kilómetros (ver “Hardware de los vehículos”, anexo A.1).

El parámetro isActive indica si la reserva está activa, es decir, si se ha mandado ya la orden al coche de que se abra al usuario correspondiente cuando éste pase su tarjeta magnética por el lector.

3.2.4 Coche (Car)

El número de atributos a registrar de un coche es elevado. Como referencias a otras entidades están el grupo administrativo al que pertenece, el lugar (Place) donde está ubicado, y su seguro.

Los siguientes atributos: marca (brand), modelo (model), color, matrícula (carPlateNumber), categoría (category), combustible (fuelType), fechas de las próximas ITV, cambios de ruedas y mantenimiento (inspection, wheelChange y revision), y fechas y resultados las anteriores.

La IP es estática, y por lo tanto se puede almacenar como un atributo. Servirá para poder conectarnos al coche y leer su kilometraje (km) y el nivel de combustible del depósito (fuelLevel). El campo isActive permite deshabilitar un coche sin tener que borrarlo de la base de datos, y dejaría de ser mostrado a los clientes. Útil para darlo de baja o para llevarlo un día al taller. Los campos feat_ son características: si lleva o no aire acondicionado, el tipo de transmisión, nº de plazas y puertas, capacidad del maletero, emisiones CO_2, y GPS.

Los últimos 3 atributos, “en reserva activa”, “última latitud” y “última longitud” no fueron inicialmente diseñados, y sirven para agilizar la carga del servidor en una serie de procesos que se ejecutan cada pocos minutos sobre todos los coches de la aplicación.

Car	
•**idCar**	**int**
•idAdminGroup	int
•idInsurance	int
•idPlace	int
•brand	string
•model	string
•color	string
•carPlateNumber	string
•category	string
•next_inspection_date	date
•last_inspection_date	date
•last_inspection_result	string
•next_revision_date	date
•last_revision_date	date
•last_revision_result	string
•next_wheelChange_date	date
•last_wheelChange_date	date
•fuelType	string
◦fuelLevel	%
◦km	int
•ip	int
•isActive	boolean
•feat_airConditioning	boolean
•feat_transmission	manual/autom
•feat_seats	int
•feat_doors	int
•feat_bigBaggagesCapacity	int
•feat_smallBaggagesCapacity	int
◦feat_CO2emissions	string
•feat_GPS	string
•inActiveBooking	boolean
◦lastLatitude	decimal
◦lastLongitude	decimal

Figura 5: Coche

3.2.5 Tarjeta de crédito/débito (Card)

Card	
•**number**	**string**
•idClient	int
•frequent	boolean
•saved	boolean
•cardType	string
•expiryDate	date
•cvc	int
•cardHolderName	string
•cardHolderAddress	string
•cardHolderAddress_postCode	string
•cardHolderAddress_country	string

Figura 6: Tarjeta de crédito/débito

La entidad “Tarjeta de crédito/débito” tiene su número, cliente, tipo (crédito o débito), caducidad (expiry date), cvc, y nombre y dirección completa del titular.

Frequent indica si se muestra la tarjeta al escoger forma de pago en reservas entre sus “tarjetas frecuentes” para ahorrarse introducir los detalles de nuevo.

Y por último el atributo “guardada” (saved) indica si la tarjeta está permanentemente almacenada en la base de datos, o si se va a borrar en cuanto termine la reserva que está pagando y se cobre. Esto permite al usuario tener la seguridad de que no se va a almacenar información de sus tarjetas sin su consentimiento.

3.2.6 Grupo de coches (CarGroup)

Un grupo de coches es una entidad de muy alto nivel. No posee referencias a otras entidades. Los únicos atributos son su nombre, que tan sólo sirve como ayuda al administrador para no tratar con un simple identificador numérico; y los campos latitud, longitud y zoom, que marcan el punto exacto donde se centrará, y el zoom inicial con el que se mostrará el mapa de coches de la aplicación.

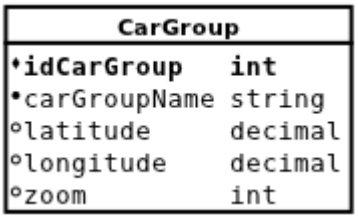


Figura 7: Grupo de coches

3.2.7 Posición de un coche (CarPosition)

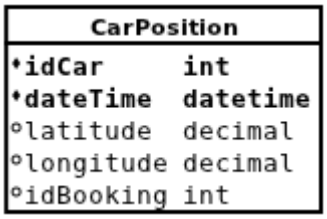


Figura 8: Posición de coche

La entidad “Posición de un coche” registra dónde ha estado (latitud y longitud) un coche (idCar) en un determinado instante (dateTime). Posteriormente al diseño inicial se añadió el atributo “idBooking”, que se rellena con el identificador de una reserva en ese instante el coche estaba en una. Así, a la hora de recopilar todas las posiciones gps de un vehículo en una reserva para calcular los km que facturar, se podrán buscar todas las posiciones de esa reserva mucho más rápidamente.

3.2.8 Cliente registrado (Client)

El cliente tiene como identificador su email, que a su vez sirve para que haga login en la aplicación, junto con su contraseña (password). Se comprueba que el email haya sido validado (email_isChecked). Se registra una referencia a su grupo administrativo (idAdminGroup), a su tarifa escogida (idFare) y a una promoción si la tiene suscrita (idPromotion).

Se guarda también si es una empresa o no (isCompany), y en caso afirmativo, el nombre de ésta. El nombre y apellidos (firstName y lastName) son los del cliente, en caso de individual, o del responsable en caso de empresa. Se registra el DNI o VAT (u otro documento oficial si se es extranjero, de ahí el nombre genérico national_ID). Asimismo, todos los datos de contacto: dirección (address), código postal (postCode), ciudad (city), país (country), teléfonos (phoneNumber), fax.

Se registra si el cliente quiere recibir o no las facturas por email (sendBillsToEmail), el tipo de pago escogido (paymentType), y su nº de cuenta, de ser pago por domiciliación bancaria. Si es pago por tarjeta, habrá una o más entidades tarjeta de crédito ligadas al Cliente. El idioma escogido (language), será en el que se le manden los emails, y el que se muestre por defecto al hacer login.

Se guardará su número de tarjeta magnética RFID, y si ésta ha sido activada o no (card_isActive, por seguridad hay que asegurarse de que el cliente la reciba y lo confirme antes de activar una tarjeta que puede abrir vehículos). Como al usuario se le permiten dos cambios de tarifa (configurable) libres de cargo,

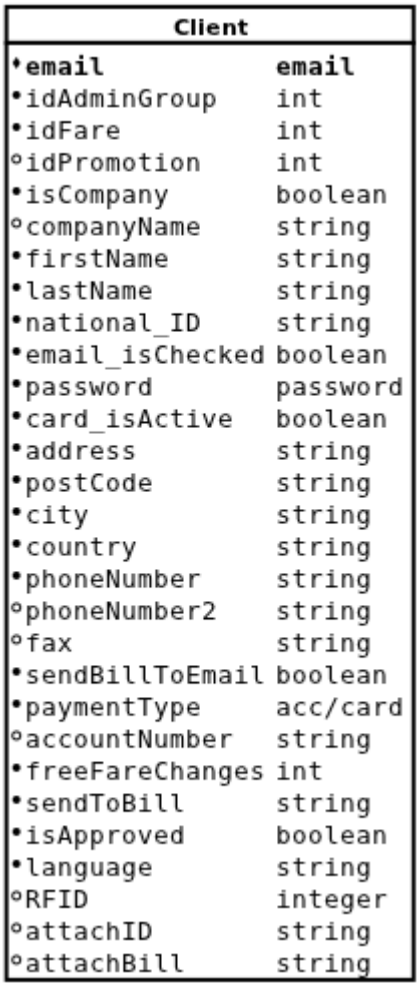


Figura 9: Cliente

hay que registrar los cambios restantes que le quedan (freeFareChanges).

Por último, si ha subido una fotocopia del DNI o de una factura domiciliada a la cuenta bancaria introducida, hay que registrar el nombre de estos archivos (principalmente por su extensión) para que desde su interfaz, un manager las pueda buscar, descargar, y validar al usuario.

3.2.9 Conductor (Driver)

Driver	
•**idDriver**	**int**
•idClient	int
•firstName	string
•lastName	string
•birthdate	date
•gender	male/fem
•national_ID	string
•dL_issueDate	datetime
•dL_expiryDate	datetime
◦dL_points	int
◦dL_pointsQueryPermit	boolean
•dL_issueCountry	string
•dL_isInternational	boolean
•isApproved	boolean
◦attachDL	string

Figura 10: Conductor

Los conductores son una entidad aparte a los clientes porque un cliente puede ser una empresa que registre tantos conductores como quiera, o bien un particular que se registre a sí mismo y a un conductor extra. Por ello, tendrá una referencia al cliente (idClient).

Los atributos son el nombre y apellidos (first y lastName), fecha de nacimiento (birthdate, es necesario saber la edad del conductor para ver si está autorizado a llevar cierto coche dado el seguro que tenga contratado), género (gender) y DNI (national_ID).

Además, se tendrán que registrar los siguientes datos del permiso de conducción (driving license, dL): fecha de expedición (issueDate), de caducidad (expiryDate), puntos (points), si da permiso para que se consulten los puntos electrónicamente (pointsQueryPermit), país de expedición (issueCountry), y si es internacional.

Por último, se marcará si está aprobado, que inicialmente al dar de alta un nuevo conductor está registrado como falso. Es el manager quien se tiene que ocupar de validar los datos introducidos y que el adjunto del permiso de conducción concuerde.

3.2.10 Tarifa (Fare)

Las tarifas (fare) pertenecen a un grupo administrativo, tendrán un nombre, un par de características descriptoras (feature, description), si son o no para empresa o para particular (isCompany), y a continuación muchos atributos de los distintos precios.

Precio por hora (facturadas siempre de 10 en 10 minutos), precio por km, por día (cuenta como un día aquella reserva de más de 12h de duración), por km, por cada 10 minutos que se devuelva el coche tarde, precio por km en caso de que se devuelva tarde, y descuento por cada 10 minutos de devolución adelantada.

Cargos fijos: precio de la tarjeta que se le vende al cliente (cardPrice), cuota mensual (montlyPrice), fianza a poner al reservar un coche (deposit), precio de la reducción de franquicia del seguro, de contratarse (excessDeduction).

Fare	
•**idFare**	**int**
•idAdminGroup	int
•name	string
•pricePerHour	decimal
•pricePerKm	decimal
•pricePer10MinsLate	decimal
•pricePerKmLate	decimal
•pricePerDay	decimal
•cardPrice	decimal
•monthlyPrice	decimal
•deposit	decimal
•excessDeduction	decimal
•deductionPer10MinsEarly	decimal
•isCompany	boolean
•feature1	string
•feature1description	string
•feature2	string
•feature2description	string

Figura 11: Tarifa

3.2.11 Multa (Fine)

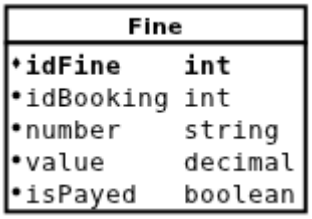


Figura 12: Multa

Guardar información sobre las multas que puedan llegar es necesario, ya que los responsables de su abono serán los propios conductores, y se asociarán siempre a la reserva en la que haya resultado multado el vehículo. Tendrán el número de multa asignado por la autoridad competente, el valor a pagar, y si ha sido pagada ya o no.

3.2.12 Incidencia (Incident)

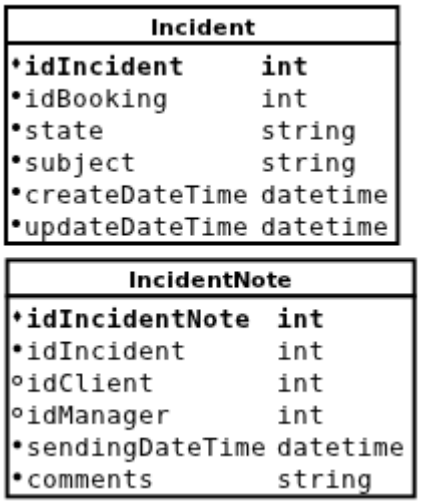


Figura 13: Incidencia

Las incidencias pueden suceder durante una reserva, ya sea antes del comienzo de ésta, e.g. que el usuario reporte que el vehículo tenía una rueda pinchada, o tras ella, e.g. un accidente. Estos podrían ser los asuntos de la reserva (subject).

Tiene visibilidad sobre ella la cuenta del usuario del conductor que haya realizado la reserva en la que se produjo el incidente. Por ello, con almacenar en la entidad "idBooking" es suficiente. Tendrá un estado (state), que podrá ser pendiente, a la espera de respuesta, o cerrada.

Un manager puede ver todas las incidencias sobre reservas de su grupo administrativo para intentar resolverlas. Por eso, tanto el cliente responsable de la reserva como los managers podrán enviar notas sobre la incidencia, comentarios (comments) que se almacenarán en una tabla independiente, IncidentNote. Una nota es sobre una única incidencia, por lo que se almacenará "idIncident". Puede ser enviada o bien por el cliente (en cuyo caso idManager sería nulo), o bien por un manager (lo que dejaría nulo idClient), y hay que registrar la fecha/hora de envío.

3.2.13 Seguro (Insurance)

Un seguro (insurance) estará asociado a un grupo administrativo. Se guardará la compañía aseguradora, el tipo del seguro (e.g. Todo riesgo), la franquicia (excess) i.e. a partir de qué cantidad el seguro cubre el coste de una reparación.

Por último, hay que registrar la edad mínima que tendrá que tener un conductor que intente reservar un coche con éste seguro, y la antigüedad mínima de su permiso de conducción.

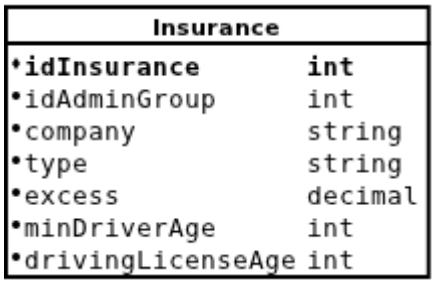


Figura 14: Seguro

3.2.14 Administrador de grupo (Manager)

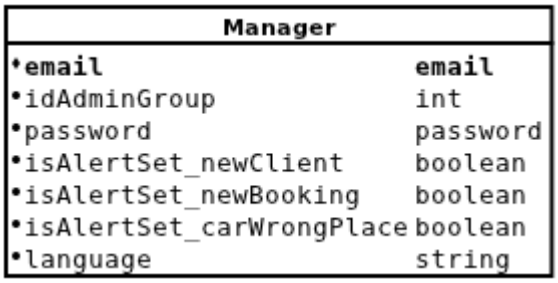


Figura 15: Manager de grupo

Los managers administran a los usuarios de su grupo, por ello tienen que tener una referencia a su admin group. El email y la password les permitirán hacer login de idéntica manera a los clientes, y el idioma preferido (language) será en el que reciban los emails y visualicen la aplicación. Tres campos booleanos registrarán bajo qué eventos quieren recibir alertas al email (_newClient para nuevos clientes, _newBooking para reservas, _carWrongPlace para coches fuera de su lugar habitual).

3.2.15 Ubicación de vehículos (Place)

Las ubicaciones de vehículos (place) registradas pertenecen a un grupo administrativo. El grupo de coches (idCarGroup) es el que tiene visión y por tanto puede reservar los vehículos estacionados ahí. Latitud y longitud permitirán mostrar en el mapa visualmente la ubicación.

El resto de atributos, muchos de ellos opcionales, son el tipo de lugar (placeType: parking, local privado, calle...), el nombre (e.g. de ser un parking), teléfono y horario de apertura, de ser un parking (timetable). Los comentarios podrán aclarar su facilidad de acceso, visibilidad...

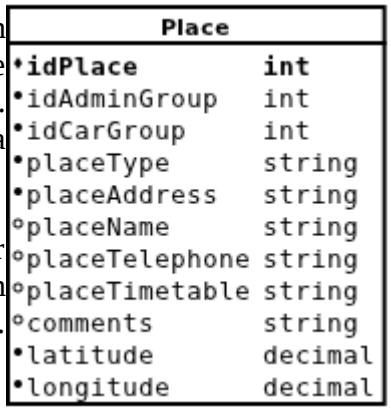


Figura 16: Ubicación de vehículos

3.2.16 Promoción (Promotion)

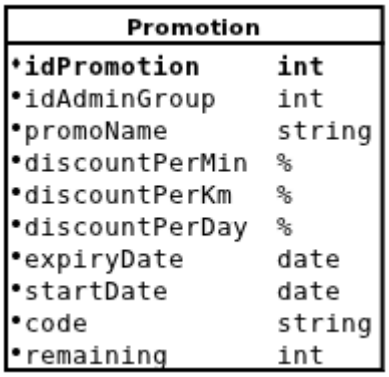


Ilustración 17: Promoción

Las promociones pertenecen a un grupo administrativo, y tienen un nombre que las identifique más fácilmente (promoName).

El código (code) es lo que permite que un usuario la introduzca. Al introducirla, se descontará una unidad de la cuenta de disponibilidad (remaining).

Consistirán en un porcentaje de descuento sobre el precio a pagar por minuto, km, y día de reserva.

Las fechas de comienzo (startDate) y caducidad (expiry) permitirán acotar la duración de las promociones.

Capítulo 4

Enrutamiento

Así como el diseño de la base de datos permite ver cómo se van a estructurar las entidades de información del sistema, el diseño del mapa de rutas de una aplicación web es otro punto clave del desarrollo. Permite tener una visión muy concreta de qué es lo que hay que hacer, cómo se va a estructurar funcionalmente la aplicación.

Lo primero a tener en cuenta en el diseño fue el requisito de mantener los grupos administrativos independientes, de manera que pudieran ofrecer distintas interfaces. Para ello, una ruta debía de permitir siempre saber en qué grupo administrativo nos encontrábamos.

Cuando un usuario ha hecho login, automáticamente se guardan sus datos en la sesión, y podemos saber su grupo administrativo sin necesidad de que esté presente en la ruta. Lo mismo sucede para un manager. Por eso, sólo para usuarios no registrados hay que poner explícitamente en la ruta en qué grupo nos encontramos.

A continuación veremos unos ejemplos básicos de rutas reales de la aplicación, para ver cómo trabaja Symfony con ellas. Después veremos un resumen del mapa de rutas de la aplicación, especialmente aquellas que representan una funcionalidad clara para alguno de los actores (usuario no registrado, cliente, manager, administrador general).

4.1 Resolución de URL (routing) en Symfony

Las URL (Uniform Resource Locator), también conocidas como direcciones web, son lo que entendemos por las rutas que permitirán acceder a las distintas funcionalidades de la aplicación.

Vamos a ver unos ejemplos sencillos de rutas de la aplicación para ver las diferentes características de Symfony a la hora de resolverlas. Todos los ficheros de rutas se albergan en “src/Resources/config” (ver Capítulo 2.3, Esqueleto de una aplicación construida sobre Symfony).

El usuario no registrado puede acceder a todas las rutas públicas. Sólo unas pocas de éstas serán únicas entre los distintos grupos administrativos: la raíz, que por requisito del sistema estará libre para albergar una página web estática que servirá de selector de zona (grupo administrativo). La forma que tienen los patrones de rutas en los ficheros de enrutamiento yaml es la siguiente:

```
home:
   pattern:  /
   defaults: { _controller: CarSharingBundle:Public:home }
```

Según trabaja el motor de Symfony, la aplicación responderá a la ruta raíz / con la función

“homeAction()” del controlador “PublicController.php”. Pocas otras rutas no contienen el nombre del grupo administrativo. Estas son las que muestran los términos y condiciones (/terms), privacidad (/privacy), y cómo funciona (/how), todo ello igual para cualquier grupo.

El siguiente caso que se explicará son las rutas de la página de contacto. Éstas son dependientes del grupo, así que como se puede ver a continuación, tienen en la ruta “/{group}”. Una palabra entre llaves en la ruta significa una variable que tomará el valor que se le de en la URL y que se pasará al controlador. La primera ruta (contact_us) será atendida por un controlador que renderizará el formulario de contacto y lo presentará al usuario. La segunda página tiene una línea extra, “requisito: método POST”. Esto asegura que esa ruta sólo pueda atender envíos del formulario de contacto (para más información sobre los métodos GET y POST de una request HTTP, consultar la web de w3schools)[5]. *(Imagen del formulario de contacto en Anexo B.3)*

```
contact_us:
   pattern:  /{group}/contact-us
   defaults: { _controller: CarSharingBundle:Public:contactUs }

contact_submit:
   pattern:  /{group}/contact-submit
   defaults: { _controller: CarSharingBundle:Public:contactSubmit }
   requirements: { _method: post }
```

4.2 Rutas públicas (de usuario no registrado)

/	*- Raíz, se deja vacía.*
/terms	*- Términos y condiciones.*
/privacy	*- Privacidad.*
/how	*- Cómo funciona.*
/{group}	*- Index del grupo.*
/{group}/where	*- Mapa de coches del grupo. (Anexo B.10, B.11)*
/{group}/where/filter (post)	*- Envío de filtros seleccionados en el mapa. (Anexo B.12)*
/{group}/where/address (post)	*- Búsqueda de dirección en el mapa.*
/{group}/book/{car}	*- Selección de un coche desde el mapa. (Anexo B.13)*
/{group}/business	*- Index de la web viéndola en modo “para empresas”.*
/{group}/search	*- Búsqueda de vehículo.*
/register/{group}	*- Escoger tipo de usuario. (Anexo B.5)*
/register/{group}/individual	*- Escoger tarifa (usuario particular). (Anexo B.6)*
/register/{group}/company	*- Escoger tarifa (usuario empresa).*
/register/{group}/individual/{fare}	*- Muestra el formulario de registro para el tipo de usuario y*
/register/{group}/company/{fare}	*- la tarifa escogida. (Anexo B.7)*
/confirm/{group}/{email}/{hash}	*- Confirmación de email. Link enviado al cliente. (Anexo B.24)*

4.3 Rutas de cliente registrado

/user/index	*- Página de inicio de cliente.*
/user/profile/{what}	*- Ver parte de su perfil (datos personales, de pago o de cuenta) (Anexo B.17)*
/user/edit/{what}	*- Modificar parte de sus datos de perfil. {what} marca el qué.*
/user/map	*- Mostrar el mapa. Equivale a /{group}/where, pero para user.*

/user/map/filter (post) - *Mapa con filtros introducidos.*
/user/map/address (post) - *Mapa con búsqueda de dirección*
/user/update/personal (post) - *Envío del formulario de edición de detalles personales.*
/user/update/account (post) - *Envío del formulario de edición de datos de la cuenta.*
/user/update/password (post) - *Envío del formulario de cambio de contraseña.*
/user/update/payment (post) - *Envío del formulario de cambio de forma de pago.*
/user/promotion - *Muestra la promoción del usuario, de tenerla. (Anexo B.16)*
/user/promotion/add (post) - *Envío del formulario de introducir código de promoción.*

/user/bill - *Muestra las facturas del usuario (Anexo B.15)*
/user/bill/list/{m1}-{y1}_{m2}-{y2} - *Muestra las facturas desde mes1-año1 hasta m2-y2*
/user/bill/show/{id} - *Descarga de la factura con identificador "id".*

/user/booking/index - *Lista las reservas del cliente*
/user/booking/show/{id} - *Muestra los detalles de la reserva {id}*
/user/booking/new - *Muestra el formulario de nueva reserva (Anexo B.4)*
/user/booking/create (post) - *Envío del formulario de reserva*
/user/booking/edit/{id} - *Muestra el formulario de edición de reserva (Anexo B.20)*
/user/booking/update/{id} (post) - *Envío del formulario de edición*
/user/booking/cancel/{id} - *Pide confirmación para cancelar la reserva*
/user/booking/delete/{id} (post) - *Cancela la reserva (si está permitido dada la fecha/hora)*
/user/booking/book/{idcar} - *Reserva de coche desde el mapa.*

/user/driver/index - *Muestra todos los conductores registrados del usuario.*
/user/driver/show/{id} - *Muestra los detalles del conductor {id} de su cuenta.*
/user/driver/new - *Muestra el formulario de alta de conductor. (Anexo B.14)*
/user/driver/create (post) - *Envío del formulario de alta de conductor.*
/user/driver/delete/{id} - *Da de baja un conductor de su cuenta. No se borra realmente.*

/user/fare/choose/{id} - *Cambiarse a la tarifa {id}*
/user/fare/update/{id} (post) - *Envío del formulario de cambio de tarifa*
/user/plans - *Muestra los planes disponibles, puede cambiarse a otro si le quedan*
- *"freeFareChanges", cambios de tarifa libres de cargo.*

/user/incident/index - *Lista las incidencias del usuario.*
/user/incident/show/{id} - *Muestra los detalles de una incidencia. (Anexo B.18)*
/user/incident/new - *Muestra el formulario de apertura de una nueva incidencia.*
/user/incident/create (post) - *Controla el envío del formulario de nueva incidencia.*
/user/incident/newnote/{id} (post) - *Controla el envío de una nueva nota en una incidencia.*

/user/card/index - *Lista las tarjetas de crédito/débito del usuario. (Anexo B.19)*
/user/card/show/{id} - *Muestra los detalles de una tarjeta.*
/user/card/new - *Muestra el formulario de registro de una tarjeta.*
/user/card/create (post) - *Controla el envío del formulario de nueva tarjeta.*
/user/card/edit/{id} - *Muestra el formulario de edición de una tarjeta.*
/user/card/update/{id} (post) - *Envío del formulario de edición.*
/user/card/delete/{id} - *Borra una tarjeta.*

Como se puede ver, las rutas de cliente son numerosas. Éstas son exactamente todas las funcionalidades que pueden realizar los usuarios registrados en la aplicación. No se han resumido porque era interesante mostrar exactamente el ámbito de competencias de un cliente, y porque las

web que acaban renderizando los controladores de todas las rutas expuestas tienden a estar más trabajadas que las de manager. Éste era un requisito del sistema: que las pantallas mostradas a un usuario no administrador fuesen especialmente atractivas visualmente (al fin y al cabo hay que captar nuevos usuarios).

4.4 Rutas de administrador de grupo (manager)

Para evitar llenar innecesariamente el presente documento con rutas repetitivas, se empleará la terminología CRUD para las rutas de administración -el back-end- que son aproximadamente cuatro veces más numerosas que las del front-end (usuario no registrado y cliente registrado juntos).

Las siglas CRUD indican Create, Read, Update, Delete (Crear, Leer, Actualizar, Borrar), las cuatro funciones básicas de administración que realizar sobre una entidad.

- C – Create: Crear nuevo objeto de la entidad. (e.g: manager da de alta un vehículo)
- R – Read: Mostrar un objeto existente de la entidad. (e.g: manager ve una reserva)
- U – Update: Editar algún atributo de un objeto (e.g: manager aprueba a un conductor)
- D – Delete: Eliminar objeto (e.g: manager borra a un usuario fraudulento no autorizado)

/admin/index - *Página inicial de manager.*

/admin/bill/list/{m1}-{y1}_{m2}-{y2} - *Lista facturas entre dos fechas.*
/admin/bill/show/{id} - *Descarga la factura {id}. (Anexo B.25, B.26)*
/admin/bill/changestate/{id} (AJAX) - *Cambia el estado de la factura.*

Nota: el cambio de estado {pagado ← → no pagado} se realiza de manera asíncrona para evitar un formulario sólo para este campo, por medio de una petición AJAX a través de jQuery [16].

/admin/booking/RUD - *Ver, editar o borrar Reservas.(Anexo B.21)*
/admin/card/RUD - *Ver, editar o borrar Tarjetas de crédito/débito.*
/admin/client/RUD - *Ver, editar o borrar Clientes.*
/admin/driver/RUD - *Ver, editar o borrar Conductores.*
/admin/fare/CRUD - *Crear, ver, editar o borrar Tarifas.*
/admin/fine/CRUD - *Crear, ver, editar o borrar Multas.*
/admin/incident/CRUD - *Crear, ver, editar o borrar Incidencias.*
/admin/insurance/CRUD - *Crear, ver, editar o borrar Seguros.*
/admin/place/CRUD - *Crear, ver, editar o borrar Ubicaciones.*
/admin/promotion/CRUD - *Crear, ver, editar o borrar Promociones.*
/admin/car/CRUD - *Crear, ver, editar o borrar Coches. (Anexo B.23)*
/admin/car/category/list - *Muestra las categorías de vehículos. (Anexo B.22)*
/admin/car/category/upload/{category} - *Sube una nueva imagen de una categoría.*

4.5 Rutas de administrador general (super-admin)

/superadmin/index - *Página inicial del administrador general*
/superadmin/manager/CRUD - *Crear, ver, editar o borrar Administradores de grupos.*
/superadmin/admingroup/CRUD - *Crear, ver, editar o borrar Grupos administrativos.*

4.6 Rutas de acceso

/access/login - *Muestra el formulario de login. (Anexo B.9)*
/access/logout-redirect - *Escoge página a la que redireccionarte tras hacer logout.*
/access/goto_group - *Ruta redireccionada por Symfony en caso de login correcto. Carga*
- del idioma preferido y el grupo del usuario que ha iniciado sesión.
/access/check - *Ruta de seguridad de Symfony para la comprobación del login.*
/access/logout - *Ruta de seguridad de Symfony para salir de la sesión.*
/access/denied - *Ruta redireccionada por Symfony de no pasar el control de acceso.*
/access/remember-password - *Ruta que muestra el formulario de recuperación de contraseña.*
/access/remember-check (post) - *Controla el envío del formulario de recuperación de pass.*
/access/es - *Cambia a locale: español.*
/access/en - *Cambia a locale: inglés.*

Capítulo 5

Seguridad

Al final del capítulo 4 podíamos ver listadas las rutas de acceso. Algunas de estas rutas son algo especiales: no hay que capturarlas con ningún controlador, sino que se indica a Symfony en su fichero de seguridad que debe ocuparse de gestionarlas.

La seguridad de una aplicación web Symfony se realiza en dos pasos, autenticación y autorización [2]. Primero, se verifica que seas quien dices ser (login), y después se decide si tienes permiso para acceder a un recurso (control de acceso).

5.1 Autenticación de usuarios

El primer paso comienza por enviar el formulario con tus credenciales. Si observamos la figura 18, el formulario de acceso tiene como destino (action) la ruta de nombre “login_check”, y los inputs son “_username” “_password” y “_remember_me”. Lo que hay entre llaves es el lenguaje propio del generador de plantillas Twig. Cuando el controlador renderiza esta plantilla, busca la ruta llamada “login_check” y sustituye {{ path(“login_check”) }} por el pattern de la ruta, en este caso “/access/check”.

```
<form id="register-form" action="{{ path("login_check") }}" method="post" id="login">
    <span>{% trans %}access.login.email{% endtrans %}</span>
    <input type="text" id="username" name="_username" value="{{ last_username }}" />

    <span>{% trans %}access.login.password{% endtrans %}</span>
    <input type="password" id="password" name="_password" />

    <input type="checkbox" id="remember_me" name="_remember_me" checked />
    <label for="remember_me">{% trans %}access.login.remember{% endtrans %}</label>
</form>
```

Figura 18: Formulario de acceso (tanto para clientes como managers)

```
form_login:
    login_path:  /access/login
    check_path:  /access/check
    # login success redirecting options
    always_use_default_target_path: false
    default_target_path:            /access/goto_group
    target_path_parameter:          _target_path
    # field names for the username and password fields
    username_parameter:             _username
    password_parameter:             _password
logout:
    path:   /access/logout
    target: /access/logout-redirect
access_denied_url: /access/denied
```

Figura 19: Autenticación en security.yml (seguridad de Symfony)

Si nos fijamos en la figura 19 (un fragmento del fichero de seguridad) vemos que /access/check es el check path del login, así que Symfony capturará esta ruta y pondrá en marcha la autenticación.

Además, '_username' y '_password' son los campos que espera recibir en el formulario, y que utilizará en el proceso.

Entonces, comprobará si son válidos

comparándolos a pares (usuario, contraseña) extraídos de sus entidades proveedoras de usuarios (figura 20, otro fragmento del fichero de seguridad), en este caso las que definimos como client_db y manager_db, entidades Client y Manager a través de su identificador, el email.

De ser válidos, te lleva a la ruta /access/goto_group, la cual es capturada por el controlador que hemos escrito a tal efecto, que entre otras cosas comprueba el atributo "language" del cliente o manager que ha iniciado sesión, y cambia el locale (idioma actual en que se muestra la aplicación) al suyo.

```
providers:
    main:
        providers: [client_db, manager_db]
    client_db:
        entity: { class: FringesCT\CarSharingBundle\Entity\Client, property: email }
    manager_db:
        entity: { class: FringesCT\CarSharingBundle\Entity\Manager, property: email }
```

Figura 20: Proveedores de usuarios para el proceso de autenticación

En la figura 19 también se observa que se define la ruta de logout de la aplicación como /access/logout, con lo que tampoco habrá que escribir un controlador que responda a esa ruta. Tras un logout la aplicación te lleva a la ruta /access/logout-redirect. El controlador de tal ruta se limita a mandarte de vuelta a la página inicial de tu grupo administrativo.

5.2 Autorización (control de acceso)

La primera declaración que veíamos en la figura 19 es "login_path: /access/login". Esa ruta es registrada como la de control de acceso para toda la aplicación. Esto significa que cada vez que un usuario intente acceder a una ruta protegida por el control de acceso (figura 21, otro fragmento más del fichero de seguridad), se comprobará si tiene los roles necesarios para que se le permita.

Como se puede apreciar (figura 21), todas las rutas bajo la rama /user requieren del rol "ROLE_USER". Las clases proveedoras de usuarios, Client y Manager, tendrán que estar definidas como "implements UserInterface", y construir la función "getRoles()". En el caso de Client, la función "getRoles()" devuelve: "array('ROLE_USER'); ", en el caso de Manager devolverá rol de admin o de superadmin dependiendo de si tiene grupo administrativo (se tomó la decisión de que el administrador general o superadmin sería almacenado en la base de datos como cualquier manager, pero con la peculiaridad de que tendría nulo el atributo "idAdminGroup").

```
access_control:
    - { path: ^/user          , roles: ROLE_USER }
    - { path: ^/superadmin    , roles: ROLE_SUPER_ADMIN }
    - { path: ^/admin         , roles: ROLE_ADMIN }
    - { path: ^/register      , roles: IS_AUTHENTICATED_ANONYMOUSLY, requires_channel: https }
    - { path: ^/access        , roles: IS_AUTHENTICATED_ANONYMOUSLY, requires_channel: https }
```

Figura 21: Control de acceso

Si un cliente intenta acceder a una ruta bajo la rama /admin, como muestra la figura 21, el control de acceso comprobará que no tiene el rol necesario, y te llevará a la ruta definida en la figura 19 como: "access_denied_url: /access/denied". La dicha ruta es capturada por un controlador que sí se ha escrito y que renderiza una plantilla con un sencillo mensaje del tipo: "No tiene los permisos necesarios para acceder aquí".

5.3 Cifrado de conexiones: HTTPS

Lo último a remarcar de la figura 21 es el “requieres_channel: https”. Esto fuerza a que todas las rutas bajo esa rama usen el protocolo HTTPS (Protocolo de Transferencia de Hipertexto Seguro), es decir, vayan cifradas. Todo enlace de la aplicación generado a partir del motor de plantillas Twig: {{ path('nombre-ruta') }} cuya ruta esté bajo una rama que fuerce el https automáticamente utilizará este protocolo.

Cabe destacar que para que esto funcione, el servidor deberá ser capaz de trabajar con el protocolo SSL (Secure Socket Layer). Las primeras pruebas realizadas en localhost no permitieron trabajar con el cifrado activo, pero tras el deploy en el servidor, que tenía instalado SSL, los dueños del hosting comenzaron a tramitar la obtención de un certificado digital por medio de una Autoridad de Certificación.

Para más información sobre HTTPS[6], SSL[7] y certificados digitales y autoridades de certificación[8], consultar las referencias.

Capítulo 6

Controladores

A lo largo de la memoria se ha utilizado el término controlador, definido como la función que ejecuta la aplicación cuando captura una petición con un patrón reconocido entre los archivos de enrutamiento. Este capítulo tratará de presentar la estructura de un controlador de la aplicación, y los diferentes elementos envueltos en él.

6.1 Estructura básica

La estructura que sigue un controlador es la siguiente: fichero PHP que reúne funciones públicas que se ocupan de responder ante las rutas de una determinada rama. Por ejemplo, vamos a centrarnos en CarController, el controlador de las rutas de manager relacionadas con gestión de coches: /admin/car/CRUD (consultar el capítulo 4.4 para más información sobre rutas y CRUD).

```
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use FringesCT\CarSharingBundle\Entity\Car;
use FringesCT\CarSharingBundle\Form\CarType;

class CarController extends Controller
{
        public function indexAction() {...}         /admin/car/index →Lista coches
        public function showAction($id) {...}      /admin/car/show/{id} →Detalles del coche {id}
        public function newAction() {...}           /admin/car/new →Muestra form. de creación
        public function createAction() {...}       /admin/car/create → Gestiona envío del form.
        public function editAction($id) {...}      /admin/car/edit →Formulario de edición de {id}
        public function updateAction($id) {...}  /admin/car/update →Envío del form. de {id}
        public function deleteAction($id) {}       /admin/car/delete → Elimina el coche {id}
}
```

La consola de Symfony tiene un comando que genera la CRUD más básica posible para una entidad. Por ejemplo en este caso, simplemente escribiría estas tres líneas para indexAction:

```
$em = $this->getDoctrine()->getEntityManager();
$entities = $em->getRepository('CarSharingBundle:Car')->findAll();
return $this->render('CarSharingBundle:Car:index.html.twig', array('entities' =>$entities));
```

Es decir, cuando un manager accediese a la ruta /admin/car/index, el servidor ejecutará el controlador indexAction, que no recibe parámetros, y extrae todos los coches de la base de datos a través de una función definida en el repositorio de “coche”. Los almacena sobre una variable ($entities), y pasa ésta como parámetro a la función “render”. Ésta renderizará la plantilla index.html.twig para a continuación formar una respuesta HTTP con la página web html formada. Dicha web será lo único que el navegador del manager a la espera reciba.

Por desgracia esta aplicación tiene un importante requisito que nos impide generar todas las

CRUDs de forma automática: la separación de grupos administrativos. Un manager, pese a contar con el rol requerido por una ruta /admin, no deberá poder listar coches de grupos administrativos diferentes al suyo. Esto se soluciona leyendo de la sesión el nombre de usuario, y buscándolo en la base de datos, para extraer el objeto completo "Manager". Mediante él, podemos acceder a su grupo administrativo, y pasarlo a una nueva función que definiremos en el repositorio CarRepository, que realizará una búsqueda de coches sólo en el grupo introducido.

```
$username = $this->get('security.context')->getToken()->getUsername();
$manager = $em->getRepository('CarSharingBundle:Manager')->find($username);
$group = $manager->getAdminGroup();
$entities = $em->getRepository('CarSharingBundle:Car')->findAllInGroup($group);
```

6.2 Formularios

6.2.1 Constructor de formularios. Clases Type.php

Los formularios en Symfony funcionan a través de los archivos que terminan en Type.php (consultar el capítulo 2.3: esqueleto de una aplicación). Una entidad podrá tener múltiples formularios.

Por ejemplo para el caso de las reservas, "Booking", habrá un formulario de creación de nueva reserva (UserNewBookingType.php) para clientes registrados, los cuales podrán escoger el coche a reservar, conductor, fechas de inicio y fin, si contratar una reducción de franquicia para el seguro del vehículo,etc. Pero en cambio el formulario de edición (UserEditBookingType.php) sólo tendrá la posibilidad de cambiar la fecha de inicio y la de final de reserva. Un último formulario, el de managers (BookingType.php) contendrá todos los atributos de la entidad reserva, puesto que los administradores de un grupo están autorizados para cambiar cualquier atributo de una reserva sobre un coche de su grupo administrativo. A continuación se muestra un fragmento de esta clase.

```
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilder;

class BookingType extends AbstractType{
   protected $client, $translator;

   public function __construct($client, $translator){
      $this->client = $client;   $this->translator = $translator;
   }

   public function buildForm(FormBuilder $builder, array $options){
        $builder
          ->add('driver', 'entity', array(
            'label' => $this->translator->trans('form.booking.driver'),
            'class'=>'\Entity\Driver',
            'query_builder'=>function(\DriverRepository $repository) use ($this->client){
                 return $repository->createQueryBuilder('s')
                      ->where('s.client = ?1')
                      ->andwhere('s.removed = ?2')
                      ->setParameter(1, $this->client )
                      ->setParameter(2, FALSE);
               }
          ))
          ->add('startDateTime', 'datetime', array(
               'label' => $this->translator->trans('form.booking.start'),
               'minutes' => array('00','10','20','30','40','50')          ))
   }
   public function getName(){ return 'bookingform'; }
}
```

Resumiendo, la clase ha de extender AbstractType, al constructor hay que pasarle las variables que necesitemos utilizar dentro, getName devuelve el atributo html nombre (“name”) con el que se mostrará el formulario una vez se renderice, y la función buildForm creará el formulario añadiendo los atributos de la entidad reserva que queramos.

Se añaden dos atributos. El primero es el conductor, “driver”, de tipo “entity” de la clase “\Entity\Driver”. En el formulario html se mostrará un selector con tantas opciones como salgan de la consulta a la base de datos allí descrita. En la consulta se utiliza el parámetro “client”, por eso hay que pasarlo al constructor de la clase.

Es decir, cuando el manager edita una reserva, tendrá un elemento donde seleccionar al conductor de entre la lista de conductores del cliente que no hayan sido dados de baja. También tendrá un conjunto de selects para la fecha y hora de inicio de la reserva, que limitará las opciones para el select “minutos” a slots de 10 min.

La cantidad de opciones que permiten los ficheros de generación de formularios Type.php es muy elevada. Toda la información al respecto se puede encontrar en la documentación de Symfony, sección formularios [11].

6.2.2 Trabajo con formularios en los controladores

A continuación se muestra un fragmento del controlador de reservas para “editAction($id)”. Previo al código mostrado, se realizaría la búsqueda en base de datos del Manager a partir del username de la sesión, y se extraería su grupo a la variable $group.

```
$booking = $em->getRepository('CarSharingBundle:Booking')->findInGroup($id,$group);
$editForm = $this->createForm(new BookingType($booking->getDriver()->getClient(),
                                                $this->get('translator')), $booking);
return $this->render('CarSharingBundle:Booking:edit.html.twig', array(
    'booking'=>$booking, 'edit_form'=>$editForm->createView() ));
```

Es decir, se busca en la base de datos la reserva que se pretender editar, y se utiliza para crear una instancia de formulario junto con el traductor de mensajes de la aplicación (ver apartado 6.3) y la entidad encontrada en la base de datos $booking en sí misma. Esto permitirá que el formulario de edición renderizado en el html aparezca relleno con los valores actuales de la reserva que se quiere editar. Si en vez de estar modificando una reserva se quisiera crear una de cero, en lugar de buscar la reserva a editar por su ID en la base de datos, y pasar esta entidad a “createForm”, le pasaríamos una nueva instancia de la entidad: “`$booking = new Booking()`”.

Una vez el usuario de la aplicación ha rellenado un formulario y pulsa submit, vuelven a entrar en juego los controladores, esta vez atendiendo peticiones POST. En estos casos su estructura será la siguiente:

1. Comprobar que el usuario tiene permiso para interaccionar con esa entidad: mismo caso que el del manager intentando acceder a coches de un grupo administrativo distinto al suyo.
2. Comprobar que el formulario sea válido: “`if($form->isValid())`”. Si un atributo de la entidad no puede ser nulo, deberá aparecer en el formulario, y con el tipo adecuado. En el proceso de validación de formularios entran en juego las aserciones (Assert) de los meta-datos de las entidades (ver en el capítulo 2.4.2 los comentarios sobre el atributo “fuelLevel”)
3. Ligar la entidad que se quiere crear (create) o editar (update) al formulario recibido:

“`$editForm->bindRequest($this->getRequest());`”

4. Persistir los cambios en la entidad sobre la base de datos:
```
$em = $this->getDoctrine()->getEntityManager();
$em->persist($booking);
```
5. Redireccionar al usuario a otra ruta:
```
return $this->redirect($this->generateUrl('booking_show', array('id' => $id)));
```

6.3 Traducciones

Cuando en el apartado anterior repasábamos el fragmento de código de BookingType.php para entender los formularios, había otra opción a pasar al constructor de la clase: $translator. Esta es una instancia de la clase traductora, que nos permitirá que las etiquetas que acompañan a cada uno de los campos del formulario html salgan en español, inglés, o el idioma actual de la sesión.

Las traducciones en Symfony se realizan extrayendo todos los textos mostrados desde las plantillas, controladores y formularios a un único fichero con una estructura multinivel que permita facilidad de búsqueda y edición. Por ejemplo, en lugar de que el atributo 'label' para el conductor de la reserva muestre: “Seleccione al conductor de la reserva: ”, lo sustituimos por una etiqueta multinivel, llamamos al traductor: `$this->translator->trans('form.booking.driver')`, y extraemos el texto a dos ficheros, messages.es.yml y messages.en.yml:

```
[...]
controller:
    [...]
form:
    booking:
        driver: Conductor de la reserva
        start: Hora de inicio de la reserva
    car:
        [...]
[...]
```

El fichero equivalente en inglés será idéntico: con mismo número de líneas y mismas etiquetas, pero con los textos traducidos. La aplicación cargará los textos de uno u otro fichero dependiendo de la locale que haya en la sesión. De hecho, para cambiar mediante un controlador el idioma de la aplicación basta con ejecutar en un controlador: “`$this->get('request')->getSession()->setLocale('es');`” Para más información, visitar la sección de internacionalización (i18n) de la documentación de Symfony [2].

6.4 Subida de ficheros

Hay ciertos momentos en los que es necesario subir un fichero al servidor. Por ejemplo, a la hora de registrarse un usuario puede escoger subir un adjunto con su DNI, o mandarlo por correo electrónico o postal.

Symfony ofrece soporte para gestionar los adjuntos (attachments) que puedan acompañar peticiones POST. Se puede comprobar su tamaño para ver si excede límites (muy necesario para prevenir ataques de este tipo al servidor por medio de sobrecarga), computar su extensión (no hay que fiarse de que un fichero sea lo que dice ser), y almacenarlo en la carpeta deseada. En la aplicación se ha puesto en funcionamiento un directorio, “app/Resources/upload”, donde se prepara una carpeta por cada grupo administrativo, como se muestra en la figura 22.

Nota: los impresos de reservas y facturas también se almacenan bajo app/Resoureces, dentro de carpetas habilitadas para tal efecto, además de ser enviadas como adjuntos por correo electrónico a los clientes.

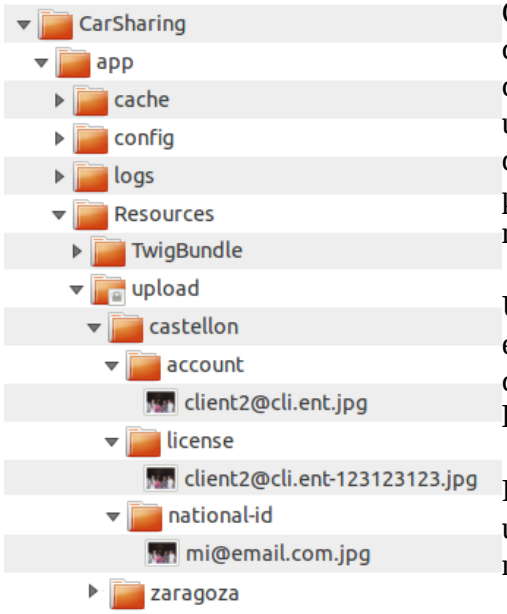


Figura 22: Directorio de subidas de ficheros

Como se puede apreciar, el grupo "castellon" tiene una carpeta "account" donde el cliente registrado cuyo email es client2@cli.ent ha subido una imagen con una fotocopia de una factura domiciliada a la cuenta bancaria con la que quiere realizar los pagos. Gracias a esto, el manager verá tal prueba y aprobará al cliente para que pueda empezar a realizar reservas.

Un conductor dado de alta por el mismo cliente, cuyo DNI es 123123123 ha subido una imagen de su permiso de conducción, y por esto ha quedado almacenada en la carpeta license con ese nombre.

Por último, un cliente con email "mi@email.com" ha subido una fotocopia de su DNI, el cual ha ido a parar a la carpeta national-id.

Ésta forma de organizar los ficheros garantiza que no existirán dos con el mismo nombre porque varios clientes intenten subirlos, que se podrán subir en el formato deseado de forma segura, adivinando su extensión: "`$ext = $file->guessExtension();`", y que estarán disponibles para descargar a través de rutas dispuestas a tal efecto y sus correspondientes controladores.

6.5 Históricos de cambios

Se tomó la decisión de implementar los históricos de cambios en la propia base de datos en lugar de en un fichero externo, por la simple razón de que de la base de datos se puede extraer información de manera precisa gracias al SQL (y exportarla a un fichero de ser necesario). Así pues, se puso en marcha una nueva entidad: Log, que tendría su mapeo en base de datos. Se extendió el esquema de la base de datos con el comando de la consola de Symfony, ejecutado desde la carpeta raíz del proyecto: `php app/console doctrine:schema:update --force`

Y por último se creó una nueva entrada al log cada vez que tuviéramos un evento que lo requisiera. Por ejemplo, al gestionar el formulario de registro de un cliente nuevo, entre otras cosas hay que almacenar la fotocopia del DNI que podría haber subido (como se ha explicado en el apartado anterior). Si esto ha de ser reportado en el log, se realiza de la siguiente manera:

```
$logEntry = new Log();
$logEntry->setAdminGroup($adminGroup->getGroupName());
$logEntry->setAuthor($client->getEmail());
$logEntry->setAction('UPLOAD ID attachment');
$em->persist($logEntry);
```

El log automáticamente crea un parámetro de tipo fecha/hora y inicializa a la hora actual. Se añade manualmente el grupo administrativo al que afecta la entrada del log, el responsable (por ejemplo, el manager que haya creado un coche nuevo), y la acción del suceso que se está reportando. Para facilitar la búsqueda, la primera palabra que se escribirá en el campo Action será NEW, EDIT, UPLOAD, DELETE, etc, seguido por una descripción más detallada.

Capítulo 7

Plantillas

7.1 El lenguaje Twig

Las plantillas son ficheros en los que hay un esqueleto preparado para rellenar con información y formar un documento completo. Este documento puede ser de cualquier tipo texto: un .txt o .csv, una página web html, xml, un email en formato de texto plano...

Si bien PHP se originó precisamente para generar páginas web dinámicas, conforme ha evolucionado ha dejado de dar tanto soporte a esta característica, y se ha convertido más en un lenguaje de programación orientado a objetos. El lenguaje Twig, de SensioLabs (los creadores de Symfony)[17], ofrece muchas utilidades para el renderizado de plantillas con una sintaxis muy clara.

Los elementos del lenguaje Twig se distinguen por las llaves: {% for coche in coches %}. Las variables se imprimen poniéndolas entre dobles llaves, e.g. {{ coche.category }}.

Entonces, si desde el controlador CarController recogemos de la base de datos todas las entidades "car" de un grupo administrativo bajo un array llamado "coches", le pasamos ese array al template que lista los coches, "list.html.twig":

```
return $this->render('CarSharingBundle:Car:index.html.twig', array('coches' => $coches));
```

Desde el template podremos muy fácilmente, por ejemplo, crear una tabla con la marca y modelo en la primera columna, y la matrícula en la segunda, de todos los coches del array que estén activos:

```
<table><tr><th>Coche</th><th>Matrícula</th></tr>
{% for coche in coches %}
	{% if coche.isActive %}
		<tr>
			<td>{{ coche.brand}} {{ coche.model }}</td>
			<td>{{ coche.carPlate }}</td>
		</tr>
	{% endif %}
{% endfor %}
</table>
```

Renderizar esta plantilla daría la salida html que un navegador mostraría así:

Coche	Matrícula
Ford Escort	Z-1234-DK
Mercedes C180	9876-WZT

Como se puede apreciar, la sintaxis Twig combinada con Doctrine permiten un control de las entidades de la base de datos desde su extracción hasta darles la forma deseada en la plantilla.

7.2 Estructura multinivel

Twig permite la herencia, de manera que unas plantillas extiendan otras. Esta posibilidad ha permitido un desarrollo multinivel de la aplicación web. En particular, se ha optado por un desarrollo en tres niveles. Una plantilla base contiene los elementos básicos que se repiten en todas las páginas web del sitio: el logo, menú, cuerpo, y pie de página. Unas plantillas intermedias extienden la base, y rellenan el cuerpo con un layout que puede ser:

- layout.html.twig: todo el cuerpo sin dividir.
- layoutprofile.html.twig: una barra lateral a la izquierda y el contenido principal a la derecha, donde la barra lateral contiene el menú de navegación del perfil de usuario: “Mis conductores, mis facturas, mis datos personales, mis preferencias de cuenta, mis promociones...”.
- layoutmanager.html.twig: igual que la anterior, pero la barra lateral tendrá las opciones del menú de administración: “Tarifas, coches, clientes, facturas, reservas, promociones, multas...”

Cualquier plantilla web de la aplicación hereda una de las anteriores tres, extendiendo su bloque “contenido”, para rellenarlo con la información que quiera mostrar.

7.3 Formularios en las plantillas

Los formularios también tienen su soporte en las plantillas. Se puede mostrar todos los campos de un formulario (ver 6.2.1) pasado a una plantilla de la siguiente manera: {{ form.rest }}, aunque le faltarían los tags de <form action=... method=... enctype=...> y </form> antes y después.

El campo action (así como cualquier tag anchor <a>, un hiperenlace), requiere una ruta (consultar sintaxis html para más información), a la cual será enviado el formulario una vez pulsado el botón de envío (submit). Las rutas se pueden construir en plantillas con la función path(), a la cual se le pasa el nombre de un patrón (pattern) de enrutamiento. Por ejemplo, poner lo siguiente: `action="{{ path('contact_submit', {'group': user.adminGroup.groupName}) }}"` (consultar capítulo 4.2 para enrutamientos), cogerá la variable “user”, obtendrá la entidad grupo administrativo que tenga referenciada, y de éste su nombre, por ejemplo, digamos que sea “zaragoza”. Entonces, el html generado será: `action="/zaragoza/contact-us"`.

Un formulario también puede mostrarse campo a campo en el caso de que nos interese situarlos en diferentes celdas de una tabla, por ejemplo. Para este caso, se puede emplear {{ form.row }}. Incluso yendo más allá, se puede mostrar por separado el label (etiqueta que acompaña al campo del formulario, por ejemplo: “Introduzca la matrícula”), el widget (campo en sí, por ejemplo, un input para introducir el texto propiamente dicho que será la matrícula), y los errores de validación, en caso de que el formulario sea mostrado nuevamente remarcando los errores que el usuario haya introducido para que los cambie y vuelva a enviar.

Se puede customizar la forma en que se muestran formularios en toda la aplicación, es decir, el html generado al ejecutar alguna de las funciones form.row, form.rest, etc. En nuestro caso se ha utilizado esta posibilidad para que los inputs se muestren con una estructura que responda a algunos scripts de validación a nivel global que se han preparado para responder a ciertos patrones html.

7.4 Traducciones y filtros en plantillas Twig

Las plantillas son el principal punto de la aplicación donde se ha tenido que traducir texto. Estos se han extraído y sustituido por etiquetas multinivel (ver el capítulo 6.3), cuyo texto se mostrará tal y como aparezca en el fichero del idioma correspondiente. Para este proceso, las etiquetas siguen la siguiente sintaxis: `<h1>{% etiqueta.multi.nivel %}</h1>`

Otra opción es utilizar el filtro traductor. Éste es útil para traducir el contenido textual de ciertas variables. Por ejemplo, si sabemos que `{{ coche.fuelType }}` puede contener "Diesel", "Gasoline", "Hybrid" o "Electric", se puede traducir este contenido sacado directo de la base de datos pasándole el filtro traductor: `{{ coche.fuelType|trans }}` escribiendo las correspondientes etiquetas en el fichero de traducciones al español:

```
Gasoline:    Gasolina
Diesel:      Diesel
Hybrid:      Híbrido
Electric:    Eléctrico
```

Los filtros son otra de las herramientas que ofrece Twig [18]. Son funciones php normales a las que podemos pasar como entrada el objeto filtrado (y más opciones). Hay algunas por defecto, y se pueden definir más si se quiere mediante las denominadas "Extensiones".

En la aplicación se han desarrollado como extensiones a Twig múltiples filtros que ayudan al desarrollo mucho más rápido y eficaz de plantillas, entre ellos:

- Filtro que oculta todos los números de una tarjeta de crédito o una cuenta bancaria con asteriscos excepto los últimos cuatro.
- Filtro que deja como máximo dos cifras decimales a un número.
- Filtro que dado un campo booleano y la ruta de dos imágenes, devuelve un tag <img src=...> con la primera si es FALSE o la segunda si es TRUE. Las imágenes utilizadas son un pequeño gif con una cruz roja, o con un tick verde. Esto permite mostrar visualmente mucho mejor el contenido de una variable:

```
<td>{{ coche.isActive|tick_or_cross(
   asset('bundles/carsharing/image/icon_tick.gif'),
   asset('bundles/carsharing/image/icon_cross.gif') )|raw }}</td>
```

7.5 Plantillas de facturas y correos electrónicos

Las facturas son plantillas especiales que se renderizan sin heredar menús ni layouts. Contienen las hojas de estilos dentro del propio html en lugar de en un archivo externo css, para así ser auto-contenidas y poder enviarse a los clientes en un sólo fichero. Mantenerlas como HTML permite que tengan referencias a las reservas que referencian, a la vez que pueden ser convertidas a PDF con herramientas externas a la aplicación. El anexo B.25 y su continuación, B.26, muestran una factura mensual de un usuario que ha realizado 4 reservas a lo largo del mes.

Además de la plantilla de reservas y la de facturas, existen las facturas "inter-company", que registran reservas de vehículos por parte de usuarios externos al grupo administrativo. De esta manera, si el grupo "Zaragoza" y el grupo "Huesca" comparten grupo de coches "Aragón", un usuario del primer grupo que reserve un vehículo del segundo será facturado por su grupo administrativo, el cual a su vez será facturado por el segundo grupo, dueño del coche. Ésta factura emitida de un grupo a otro será una factura "inter-company".

Los correos electrónicos también están en formato HTML. Por ser tan numerosos (todas las diferentes alertas a administradores, los envíos de facturas a clientes...) se ha optado por diseñar una plantilla base de la que extiendan todos, email.html.twig, con estilos optimizados para los distintos navegadores y aplicaciones frecuentes. En el anexo B.24 se puede apreciar un correo electrónico de confirmación de usuario visualizado en Mozilla Thunderbird.

7.6 Javascripts

Javscript es un lenguaje de programación del lado del cliente que permite, entre otras muchas cosas, que el navegador web que recibe el trozo de código o "script" pueda dar lugar a cambios en el contenido de la página, comunicarse asíncronamente con el servidor e interaccionar con el usuario [12]. A efectos de este proyecto, las antes listadas son las características del lenguaje de las que se hace uso.

Las diferentes plantillas que generen html pueden tener códigos javascript embebidos. Se ha optado por destinar un bloque vacío en la plantilla base para que lo extiendan las plantillas que necesiten colocar ahí sus scripts, de manera que queden ordenados y sean fáciles de consultar.

Tener javascript habilitado es obligatorio para el correcto funcionamiento del sitio web. La aplicación también hace uso de la popular librería jQuery [13], de ahí que se incluya en todas las plantillas web a través de la base.

Algunos javascript son relativamente sencillos, como por ejemplo hacer que unas pestañas muestren u oculten una sección de una página web. Otros dominan el comportamiento entero de esa página web con el papel que desempeñan. Algunos ejemplos, en complejidad creciente:

- En el mapa de coches podemos mostrar los filtros o volver al mapa mediante las pestañas superiores. (Ilustración en el Anexo B.10, B.12)
- En el formulario de registro, si has escogido "particular" en vez de empresa, y eliges una tarifa que no tenga componente mensual, tendrán que generarse dinámicamente (condiciones controladas del lado del servidor mediante Twig) dos radio-buttons que permitan el pago por tarjeta (y por consiguiente mostrar el formulario de creación de una tarjeta), o por cuenta corriente (con su correspondiente input para introducirla). Además dependiendo de la opción escogida tendrá que ponerse o quitarse de algunos campos del formulario el atributo html5 required [14]. (Anexo B.7, B.8)
- En el mapa, al introducir una dirección y pulsar el botón correspondiente o la tecla intro se realiza una serie de tratamiento sobre la cadena de caracteres introducida, añadiéndole el nombre del grupo administrativo si procede, o el país. Después, se manda ese texto a Google para que devuelva las coordenadas encontradas, y se pinta un "pin" sobre el mapa en dicho punto, centrándolo además sobre él.
- Al pinchar sobre un pin que representa una ubicación de vehículos sobre el mapa, se genera en la barra lateral una sección con todos los coches del lugar, con sus formularios correspondientes para poderlos reservar. (Anexo B.11)
- Cuando se intenta reservar un coche desde el citado mapa, se muestra una pantalla en la que si estás registrado como usuario te aparecerá una tabla para seleccionar gráficamente el intervalo de tiempo de la reserva, con un código de colores para cada 10 minutos que mostrará si el coche está libre u ocupado para ese intervalo. Esto ha requerido numerosos cómputos temporales para asegurar una cómoda usabilidad por parte del usuario, que pueda

seleccionar el intervalo a la inversa (primero la hora final y luego la inicial), aparte de trabajo con las hojas de estilos, y generación dinámica de URLs. (Anexo B.13).

- En las búsquedas de coches, se realiza desde el servidor para ahorrar tiempos de espera una carga de todos los disponibles que cumplan los parámetros de la búsqueda, y después se paginan con javascript, dependiendo de las variables: nºcoches resultantes de la búsqueda, nºcoches por página. Los mismos índices cambian dependiendo de la página en que te encuentres. (Anexo B.1).
- En las búsquedas de vehículos se han incorporado calendarios selectores de fechas (datepickers). Se han utilizado los de la librería jQueryUI, para lo cual se ha tenido que realizar una precarga de librerías, y un script de inicialización con las opciones de configuración deseadas. Además, era necesario traducir manualmente los nombres de los meses a la hora de realizar la internacionalización (i18n) de la aplicación [15].
- El “select” de la búsqueda de localizaciones (también el del filtro de reservas del manager) se ha sustituido por medio de un script por un combo-box de jQueryUI, que además de la flecha (dropdown) que muestra todas las opciones del selector, permite escribir para que se filtren.
- Las esperas en las búsquedas de coches y su posterior reserva podían llegar a alargarse un poco debido a que el servidor tiene que realizar comprobaciones en la base de datos con grandes volúmenes de registros. Por ello, se puso en marcha una pantalla “Estamos buscando...” para que el usuario no se cansase de esperar y quizá pulsase “atrás”. Para que dichas pantallas se mantuviesen en todos los navegadores, fue necesario realizar el envío del formulario de forma asíncrona. Gracias a las funciones de soporte para AJAX de jQuery[16], esto fue mucho más rápido y no se tuvo que rediseñar ningún controlador, sólo incorporar los cambios en algunas plantillas. (Anexo B.2).

Capítulo 8

Conclusiones

8.1 Resultados obtenidos

Se puede concluir que los resultados obtenidos son satisfactorios, habiéndose cubierto cada punto del listado de requisitos del que se partió. En la introducción se describía un resumen de éste, siendo el listado completo el que se ha adjuntado al presente documento en el anexo A. Dichos requisitos han podido ser modificados a lo largo del desarrollo conforme ha sido necesario, para adaptarse a las necesidades cambiantes que implica un proyecto de un año de duración.

El gran volumen del software desarrollado ha imposibilitado una explicación más detallada de las diferentes decisiones tomadas cada vez que se presentaba un problema durante el proyecto. Las aproximadamente 700 horas invertidas han producido más de 50.000 líneas de código y 156 plantillas para renderizar páginas web diferentes bajo más de 200 rutas y otros tantos controladores. Una selección de imágenes ilustrativas de la aplicación se ha adjuntado al presente documento en el anexo B.

Además del código, era requisito preparar una documentación que detalla cómo se llevó a cabo la instalación (deploy) del producto en el hosting preparado para tal efecto por la empresa. Aquí se deja constancia de cómo bajarse el código desde el repositorio de Subversion con el que se fue trabajando para guardar los avances en sucesivos commits. También se explican conceptos de configuración: cómo cambiar el mailer de la aplicación, cómo instalar nuevos grupos de coches o administrativos, nuevos idiomas, cómo cambiar el zoom por defecto con el que se muestra el mapa de Google de un grupo administrativo, etc. Al contener información privada del acceso a servidor y bases de datos no se ha podido incluir como anexo a la presente memoria

8.2 Desarrollos futuros

El desarrollo de ésta aplicación deja abierto un camino hacia su constante mejora. Los más evidentes y sencillos añadidos que se pueden llevar a cabo sobre la aplicación: incluir nuevos idiomas (que ya es un trabajo costoso de por sí, dados los varios miles de líneas que ocupa cada fichero contenedor de los textos en un idioma). Algunos de éstos posibles nuevos desarrollos:

- Nuevas funcionalidades, atajos para el administrador.
- Trabajar más sobre la API de Google utilizada para los Maps para incluir opciones como trazar gráficamente la ruta seguida por un vehículo, para estudios estadísticos de hábitos de consumo (rutas más frecuentes donde podría interesar colocar más vehículos).
- Ver en tiempo real la ubicación cambiante de los vehículos.
- Implementación de algoritmos de búsqueda más rápidos.
- Desarrollo de una web más sencilla para los dispositivos móviles con opciones reducidas.

Parte 2

ANEXOS

Anexo A

Requisitos del sistema

A.1 Hardware de los vehículos

El hardware que se instalará en los coches se compondrá de los siguientes elementos:

(NOTA: Este proyecto no comprende la instalación del hardware en los coches, este apartado es meramente informativo para ayudar a la comprensión del sistema que se desea desarrollar).

- Router 3G con IP fija.
- Lector de tarjetas de proximidad al que se accede a través del router.
- Opcional: botón de fin de reserva (lo pulsaría el usuario al finalizar la misma). NOTA: Configuraremos una variable adicional booleana que se active o no con este botón.
- Opcional: señal de bloqueo de puertas (se envía desde la app en remoto – se programará por si alguien deja el coche abierto, forma parte del SDK del lector de tarjetas; simplemente configuramos una variable booleana con esta acción, la forma de enviarla se configurará una vez estudiado el SDK). Simplemente corresponderá a un atributo de cada coche, que será booleano y tendrá un click desde los interfaces de manager. El método al que llamará quedará vacío con un comentario de cuál es el motivo (indicar dónde está en la documentación).
- Señal del cuentakilómetros, cuando se pueda recoger del coche, nos permitirá el calculo de los kilómetros de cada reserva.
- GPS, se utilizará la localizar el coche en cada momento y para calcular los km de las reservas si la señal de cuentakilómetros no está disponible.

A.2 Grupos administrativos y grupos de coches

- Un grupo administrativo es un grupo de usuarios, coches, tarifas y promociones (y todas las correspondientes variables secundarias derivadas de su funcionalidad) identificados por un código de grupo administrativo, e independiente del resto de grupos.
- Cada grupo administrativo es gestionado por uno o varios managers.
- Un manager puede gestionar un único grupo administrativo.
- Los usuarios, coches, promociones, etc. nunca cambiarán de grupo administrativo a lo largo de la vida de la aplicación.
- Adicionalmente se definen los grupos de coches (CarGroup). Cada coche tendrá un idCarGroup (identificador de grupo de coches), y un idAdminGroup (del grupo administrativo).
- A lo largo de la vida de la aplicación, es posible modificar el grupo de coches de un grupo administrativo actuando directamente sobre la bb.dd (especificar los objetos que es

necesario cambiar en la documentación).

- El parámetro idCarGroup indica los coches puede visualizar (define los posibles valores del desplegable LUGAR DE RECOGIDA) un usuario no registrado que accede a la Web en el portal de un grupo en particular.
- Si un usuario registrado escoge un coche que no pertenece a su grupo administrativo, se le asignará la tarifa por defecto de ese grupo administrativo (será un atributo de AdminGroup).

La idea de la dualidad de grupos administrativos y grupos de coches es el poder ampliar ubicaciones de vehículos en una nueva ciudad, sin que sean visibles para un usuario de otro grupo. Por ejemplo, si hay un grupo “Madrid”, y la empresa comienza a hacer pruebas de expansión a “Guadalajara”, se crea un nuevo grupo administrativo, transparente a los viejos usuarios, con distintas tarifas. Si llegado el momento los gestores deciden que ambos grupos puedan ver y reservar coches tanto de una como de otra ciudad, se crearía un nuevo “grupo de coches”. Todos los coches de ambas ubicaciones cambiarían su idCarGroup al nuevo grupo común. De esta manera, los usuarios, coches y tarifas seguirían siendo administradas por separado por sus managers, distintos e independientes, pero los usuarios tendrían visibilidad y posibilidad de realizar reservas sobre todos los vehículos. Si un usuario de Madrid hace una reserva sobre un coche de Guadalajara, se aplicaría la tarifa “por defecto” de Guadalajara.

A.3 Usuarios no registrados

- Un usuario no registrado podrá ver los coches (de igual modo que un usuario registrado) pero, cuando este usuario pulsa el botón reservar, el sistema le mostrará dos opciones: Login y Aún no estás registrado?
- Se mostrará un mapa con las ubicaciones de los vehículos, donde se podrá ver sus características y pasar a reservarlos, previo login o registro.
- Cuando un usuario no registrado accede, su pantalla principal será de modo similar a Hertz: verá una pestaña desplegable donde aparecen las posibles ubicaciones a elegir, fecha y hora de inicio, fecha y hora final, o tiempo total de reserva (intervalos de 10 min), código de promoción.
- Una vez elegidos estos parámetros, se mostrará un listado de coches (con los intervalos horarios disponibles, cambiando de color el intervalo elegido –leyenda) con este orden:
  1º. Coches disponibles en el lugar elegido y de la categoría elegida.
  2º. Coches disponibles en el lugar elegido de otras categorías (del más barato al más caro).
  3º. Coches disponibles en el lugar más cercano al lugar elegido y de la categoría elegida.
  4º. Coches disponibles en el lugar más cercano al lugar elegido de otras categorías.
  5º. Coches disponibles del segundo elegido y de la categoría elegida.
  ... (Hasta el cuarto lugar más cercano).
- Habrá un código de colores para indicar el estado de los intervalos temporales. La documentación debe incluir cómo y dónde cambiar estos colores.

A.4 Registro de usuarios y tipos de usuarios

- Aparecerá una pantalla preguntando si es empresa o particular.
- Los particulares pueden dar de alta un conductor adicional y las empresas a todos los que quiera. Cada vez que se dé de alta a un conductor adicional es necesario que nos llegue una notificación para comprobar la documentación.

- Para enviar el formulario de registro, el usuario debe aceptar las condiciones del contrato (un clic sin marcar por defecto).
- Si hay algún error de formato en el formulario o algún campo obligatorio vacío, el usuario vuelve al formulario y puede ver marcado en rojo el error. (Extensible a cualquier formulario de la aplicación).
- Si el formulario se ha rellenado correctamente, se mostrará un mensaje que indicará al usuario que recibirá un email de confirmación en unos minutos.
- El correo electrónico enviado al usuario permitirá a la aplicación verificar que el email introducido es correcto. Pinchar en el enlace dispuesto en tal correo activará la cuenta del usuario.
- También se envía notificación de aviso al administrador de grupo, para que compruebe los datos del usuario y lo valide definitivamente.
- En el formulario de registro siempre se comprueban los formatos y, en caso de error, aparecen los mensajes de ayuda correspondientes. Esto se extiende a cualquier formulario de la aplicación.
- Al finalizar el registro, se comprueba la tarifa elegida. Si el valor de la cuota mensual es igual a cero, podrá escoger entre dar su número de cuenta para domiciliar los pagos o utilizar la pasarela de pago con tarjeta. Si hay una cuota mensual, la única forma de pago será domiciliación bancaria.
- En caso de domiciliación, aparecerá un adjunto para que suba un recibo de la cuenta en la que quiere domiciliar los pagos.

A.5 Reservas de vehículos

- Los slots temporales están divididos en 10 minutos. Es decir, un usuario puede reservar un coche a en punto, y diez, y veinte, y media, menos veinte o menos diez. Éste será un parámetro configurable. Añadir en la documentación cómo cambiarlo.
- Cada reserva deja inutilizado el intervalo de 10 minutos siguientes. De este modo, evitamos que las reservas solapen.
- Cada reserva tiene 10 min de cortesía: 5 min antes y 5 min después, en los que la reserva no contará como “devuelta tarde”, y por tanto no se cargará un plus.
- La reserva mínima es de 10 minutos. No se puede reservar menos tiempo ya que es el intervalo mínimo.
- Se permitirá al usuario escoger el conductor que llevará el vehículo (un usuario tipo empresa puede tener muchos conductores registrados bajo la misma cuenta).
- Se permitirá contratar una reducción de franquicia del seguro del vehículo, con 2 opciones: una pequeña cuota sólo para ésta reserva, o una contratación para 1 año de reducción de franquicia en todas sus reservas.
- Si no hay domiciliación, sale la pasarela de pago y un mensaje indicando que si quiere cambiar la forma de pago vaya a sus datos.

A.6 Manager de grupo

- La interfaz de manager permitirá modificar cualquier parámetro de las reservas de los usuarios de su grupo, ver las facturas de éstos y marcarlas como pendientes de pago o pagadas; dar de alta, de baja o modificar parámetros de sus coches, promociones, tarifas y usuarios; revisar permisos de conducción de nuevos conductores para autorizarlos, así como

de DNI o adjuntos de facturas para comprobar las cuentas corrientes para domiciliar pagos.

A.7 Facturas

- Si el cliente escoge el pago por tarjeta, la factura se genera instantáneamente en el caso del registro o al finalizar la reserva.
- Si el cliente escoge la domiciliación, se va guardando en su ficha los gastos que hace durante el mes (puede verlos). A las 00.00 del día 1 de cada mes, se generará una factura igual que la del pago con tarjeta que incluya la suma de todos los importes (y el detalle) borrándose de su ficha.

A.8 Alertas por email

- Se enviarán alarmas por email a los administradores o managers de grupo en los siguientes casos: prioridades (indicadas en el asunto): grave, media, leve, notificación .
- Si se pierde comunicación con el router (se comprueba cada 5min): GRAVE, hay que indicar el coche y su última ubicación guardada.
- Si el coche no está en su ubicación y no hay reserva activa (se comprueba cada 5 min): GRAVE
- Si finaliza una reserva y el coche no está en su ubicación: GRAVE, hay que indicar qué coche y su posición actual.
- Si hay una reserva nueva: NOTIFICACIÓN
- Si se da de alta un nuevo cliente: NOTIFICACIÓN
- Si un cliente nuevo dado de alta pero no confirmado, realiza una reserva: GRAVE (necesitamos confirmarlo para que le funcione la tarjeta de acceso al coche)
- Si se da de alta un nuevo conductor: NOTIFICACIÓN
- Si un conductor nuevo dado de alta pero no confirmado, realiza una reserva: GRAVE (necesitamos confirmarlo para que le funcione la tarjeta de acceso al coche).
- Apertura de una nueva incidencia: tiene el nivel que le haya dado el usuario a la incidencia.
- Si le toca pasar la ITV al coche: NOTIFICACIÓN
- Si a un coche le toca mantenimiento: NOTIFICACIÓN

A.9 Producto final

- El proyecto desarrollado se instalará en el hosting dejando libre la raíz para que pueda albergar una página web. El proceso de instalación ha de quedar perfectamente documentado y se entregará el paquete de la versión final del código fuente.
- El sistema de Car Sharing ha de tener un funcionamiento correcto en Internet Explorer (todas las versiones superiores a 7.0), Firefox, Opera, Chrome, Safari y en plataformas móviles.
- Es imprescindible que la aplicación sea ágil y los tiempos de respuesta sean buenos (por supuesto, esta funcionalidad se probará en local). Por este motivo, será necesario detallar en la documentación qué imágenes pesan demasiado así como el resto de requisitos que necesita la aplicación para un funcionamiento correcto.

A.10 Documentación

- Todos los textos que se muestran en los interfaces, bien sean de usuarios no registrados, registrados, privilegiados y administradores, deben ser fácilmente localizables, siendo preferible que se encuentre en el mismo fichero. Debe estar documentado cómo cambiarlos.
- La contraseña de administrador ha de ser única y en la documentación debe constar cómo modificar la misma directamente en la base de datos. Éste será el único modo de cambiar la clave de administración.
- Explicar cómo configurar nuevas alarmas y crear históricos de las mismas .
- La documentación debe incluir la descripción de las variables principales, sus atributos y los métodos.
- Para cambiar los coches de grupo de coches a otro ya existente, es necesario detallar en la documentación: a) Cómo parar la bb.dd. b) Los objetos que contienen idgroup y el comando SQL necesario para el cambio del valor del atributo idgroup. c) Cómo arrancar la bb.dd. d) Cómo comprobar que no hay incongruencias en la bb.dd.
- Incluir en la documentación cómo se puede restringir en el acceso a todas las pantallas de la aplicación sólo para usuarios registrados .
- Se documentará con construir la URL para que un buscador independiente con los parámetros: lugar de recogida, fecha de recogida, hora de recogida, fecha de entrega, hora de entrega.
- Los resultados de la búsqueda están paginados con 5 coches por página (indicar dónde se puede cambiar este parámetro en la documentación).
- Indicar cómo cambiar el número máximo de cambios de tarifa por cliente; por defecto, 2.
- Indicar cómo modificar el número de minutos de antelación que el usuario puede modificar la reserva sin penalización.

Este listado es resultado de las reuniones, intercambio de emails y negociaciones que se fueron manteniendo con la empresa, en las que se buscaba precisar hasta dónde llegaba el ámbito del problema: qué se buscaba resolver exactamente. También se generó un documento en el que se define en una primera fase qué información se debe tener en cuenta sobre los distintos actores que entran en juego en el sistema.

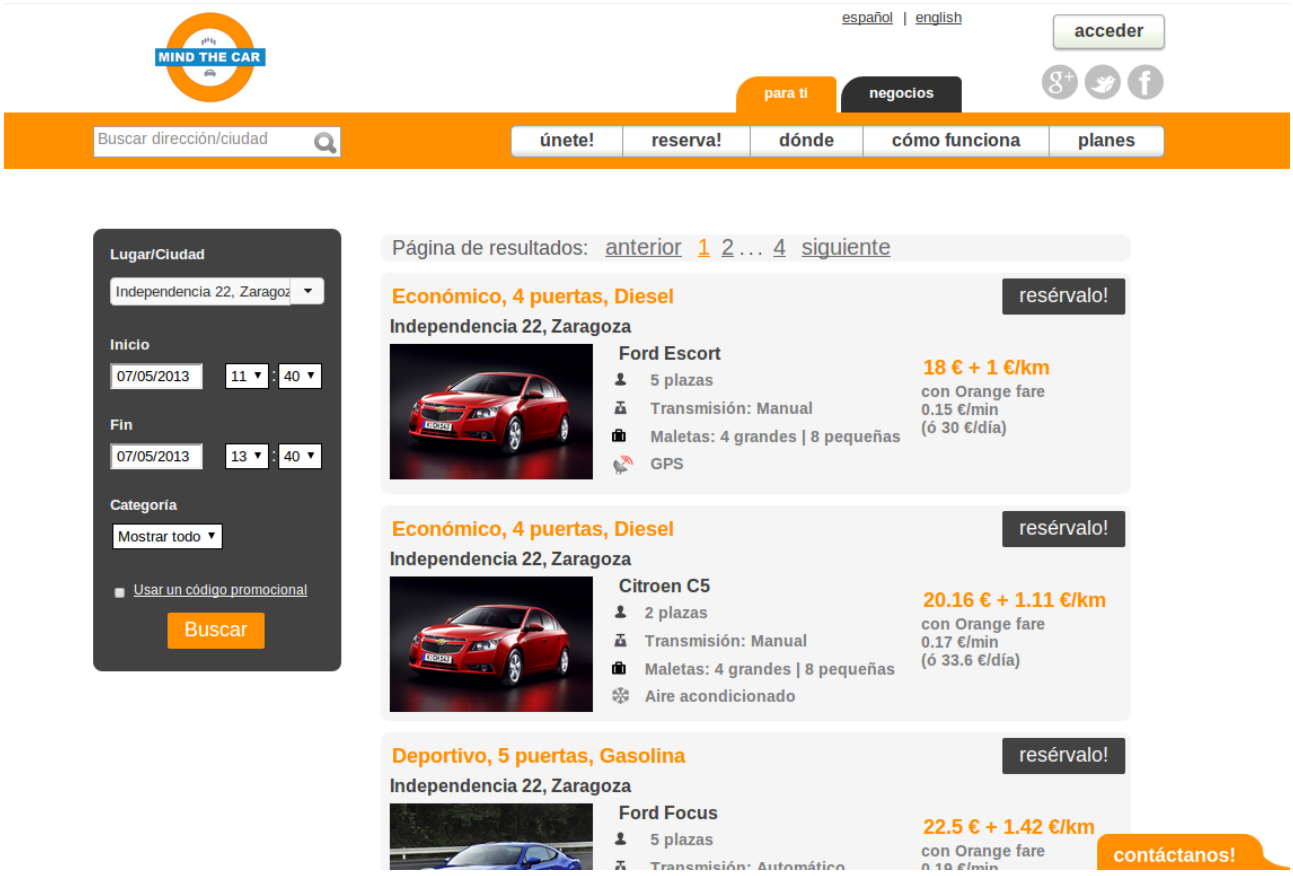
A partir de ambos documentos se tenía una idea del problema en su conjunto, y se pudo pasar a su análisis para el diseño de un esquema de base de datos que pudiera satisfacer las necesidades del problema.

Anexo B

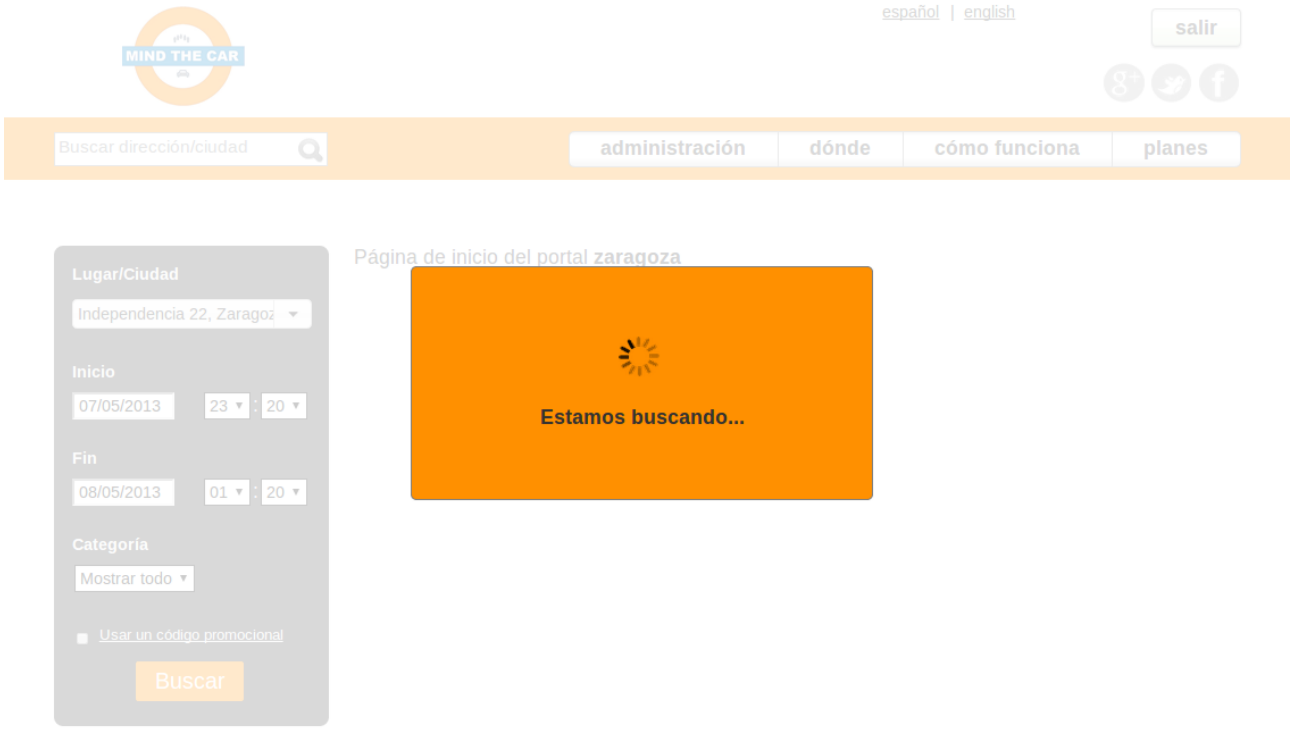
Capturas de pantalla de la aplicación

A continuación se muestran unas imágenes que ilustran algunas de las pantallas que la aplicación ofrece, pretendiendo cubrir una parte representativa de las 156 plantillas diferentes de los interfaces que se han ido describiendo a lo largo de la memoria, incluyendo algún ejemplo de correo electrónico y factura generadas.

B.1 Búsqueda de vehículos



B.2 Pantalla de carga “Estamos buscando...”



B.3 Formulario de contacto

B.4 Nueva reserva

Ubicación

Dirección: Paseo de la Independencia 22, Zaragoza
Tipo: Parking subterráneo
Teléfono: 6949182841
Comentarios: Esquina con el Simply

Hora

Recogida: 07-05-2013 23:40
Devolución: 08-05-2013 01:40

Ford Focus

Categoría: Deportivo
Combustible: Gasolina
Transmisión: Automático

- 5 puertas
- 5 plazas
- Aire acondicionado
- GPS
- Capacidad: 4 maletas grandes
- Capacidad: 8 maletas pequeñas
- Emisiones CO2: 8ml/h

Seguro

Tipo: Todo riesgo joven
Franquicia: 400€
Edad mínima del conductor: 20
Antigüedad mínima del permiso de conducción: 2

Conductor

Si el permiso de conducción del conductor ha caducado o su antigüedad o la edad del conductor no concuerdan con los requeridos por el seguro del coche, la reserva no podrá ser llevada a cabo.

34613513F - David Garcia Atun

Tarifa aplicada

La tarifa aplicada considerando tu actual plan de pagos y la tarifa del coche:
Precio por minuto: 0.17 €
Precio por km: 0.99 €
Precio diario (si más de 12h de reserva): 38.75 €

Fecha de caducidad de la reducción de franquicia que tiene contratada 26-09-2012
¿Desea contratar una reducción de la franquicia del seguro?

- (•) No contratar reducción de franquicia para esta reserva.
- ( ) 600€ - Reducción de franquicia para todas las reservas en los próximos 12 meses.
- ( ) 100€ - Reducción de franquicia sólo para esta reserva.

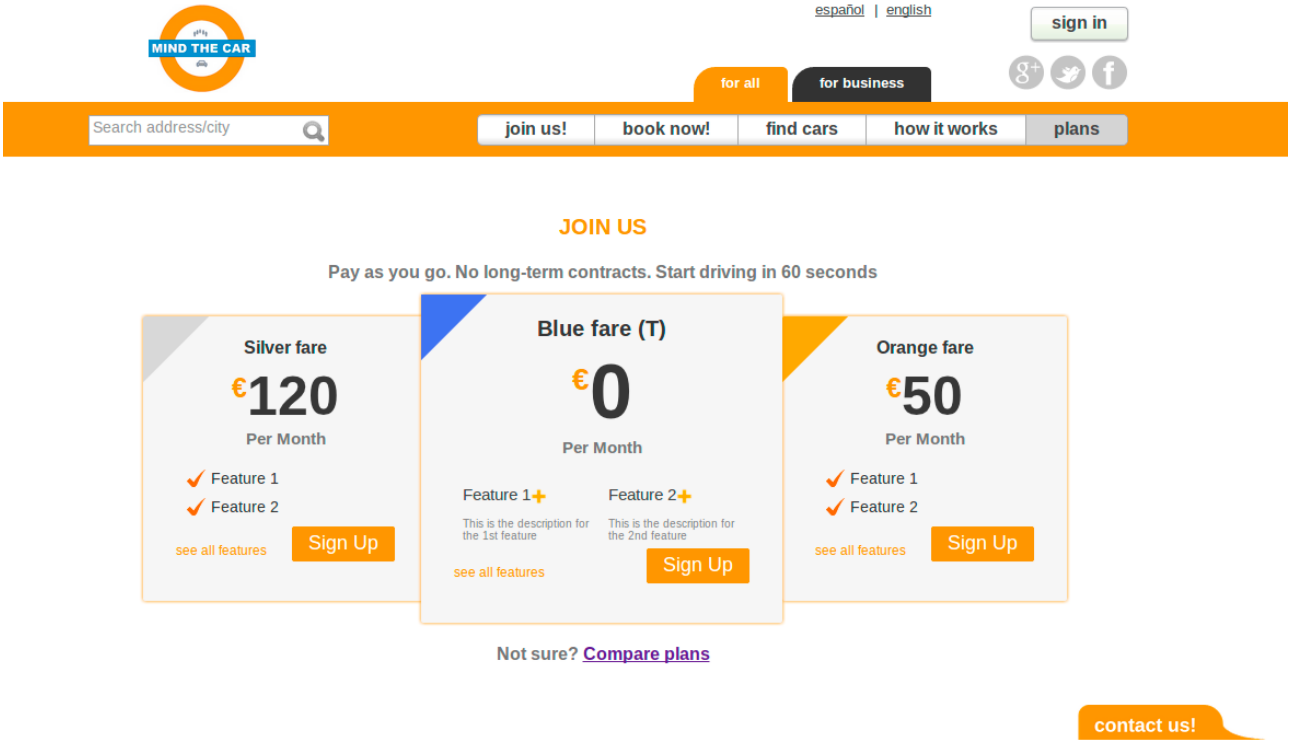
Detalles de pago

Tarjeta de crédito/débito Credit - 9876543210987654

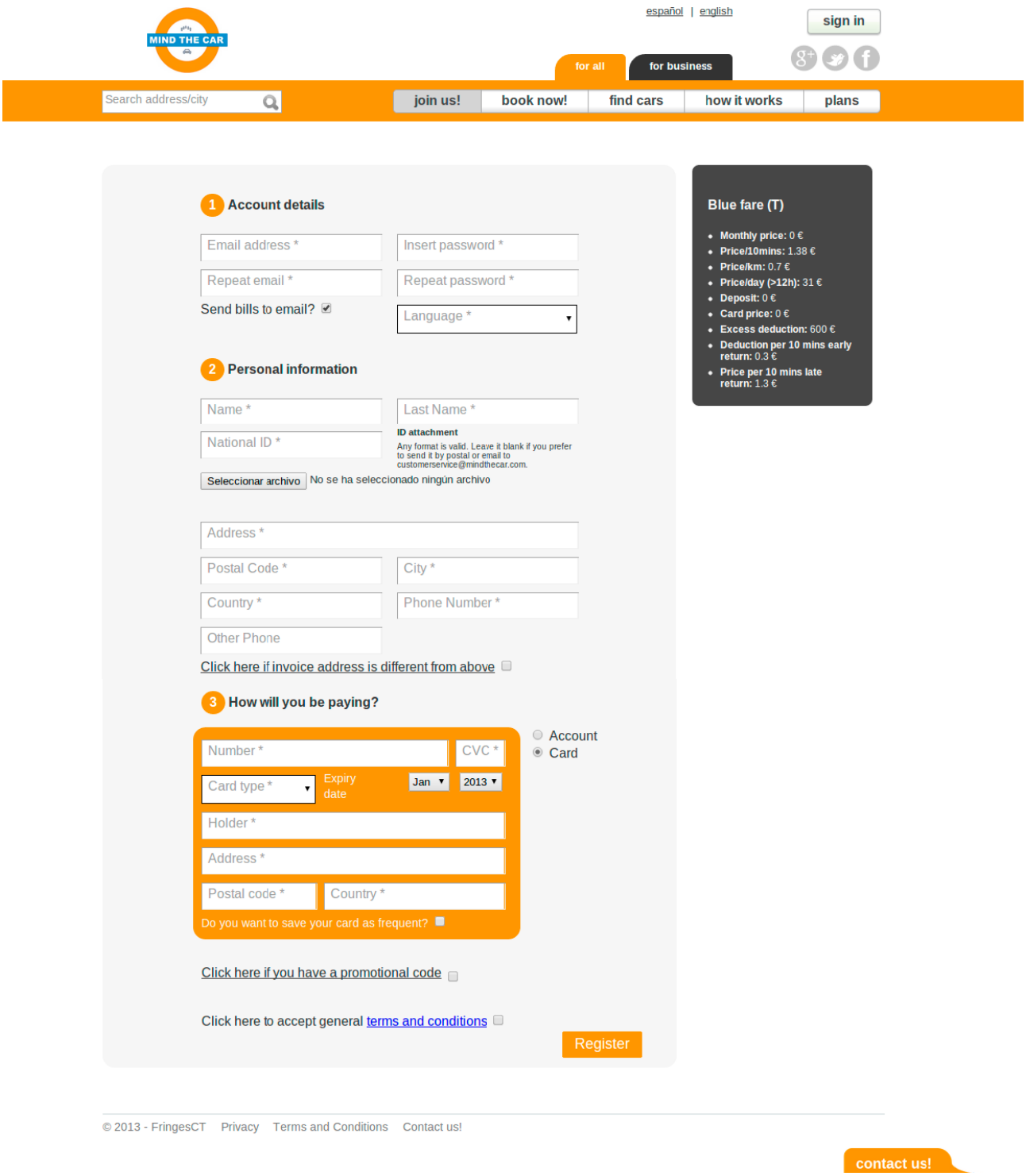
¿Desea confirmar la reserva?
Confirmar Cancelar

B.5 Elección de tipo de usuario al registrarse

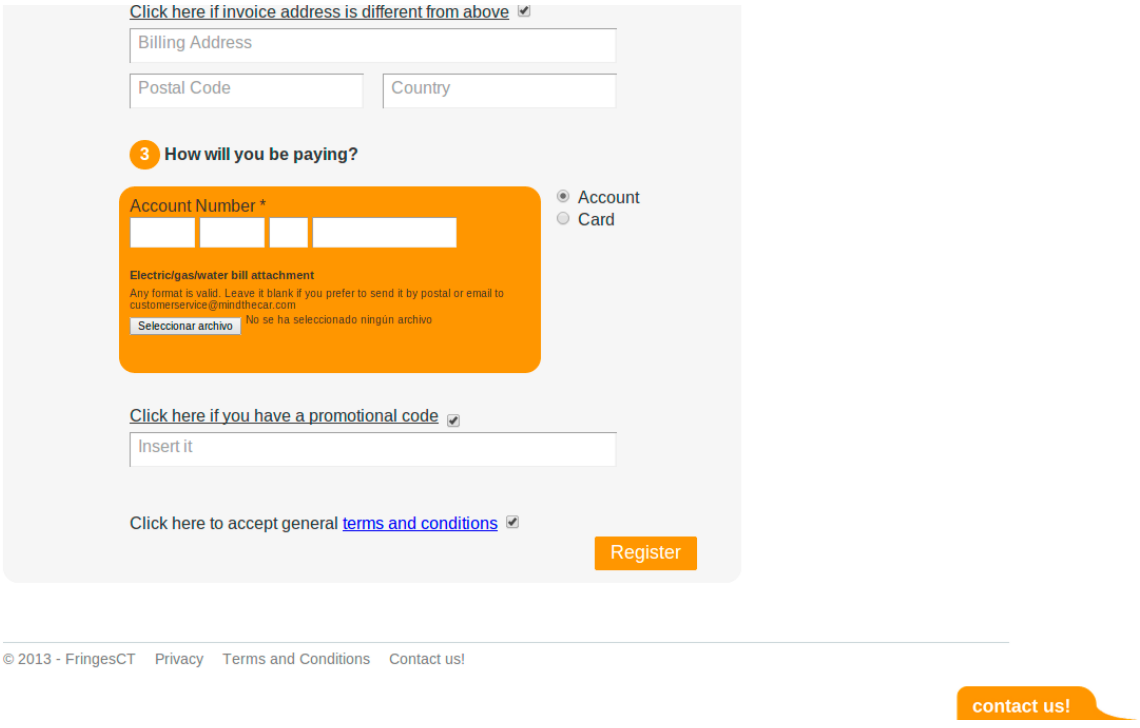
B.6 Elección de tarifa al registrarse



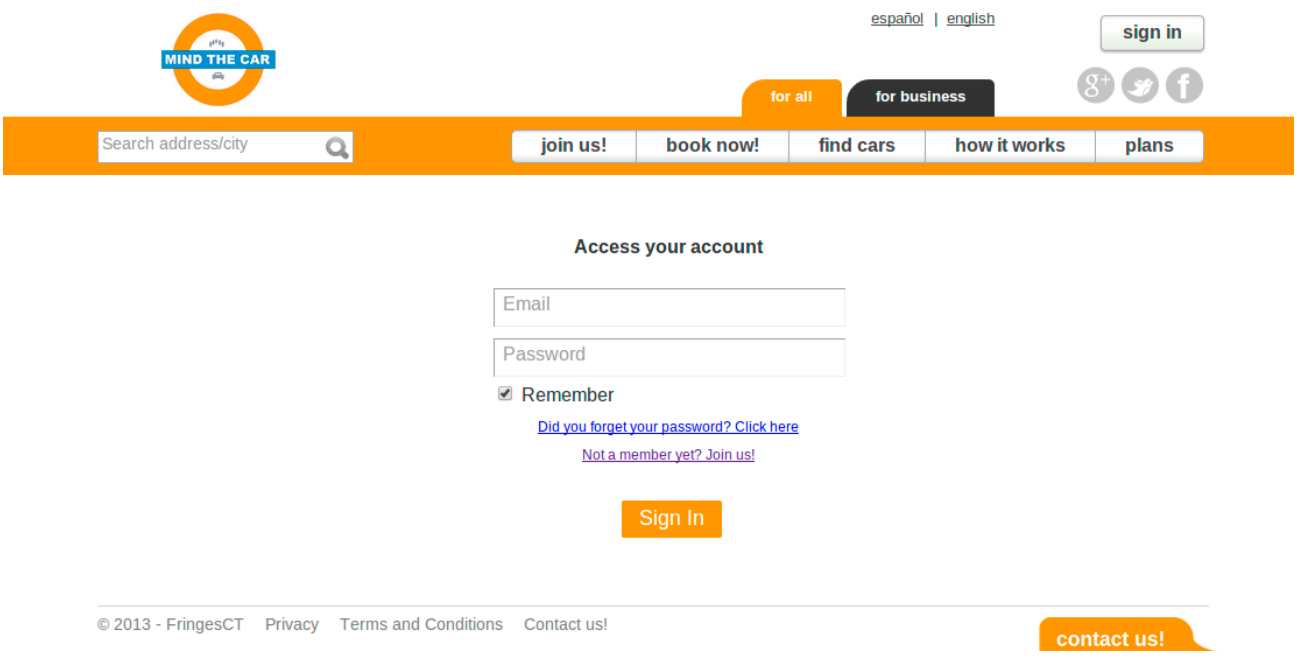
B.7 Registro de usuario



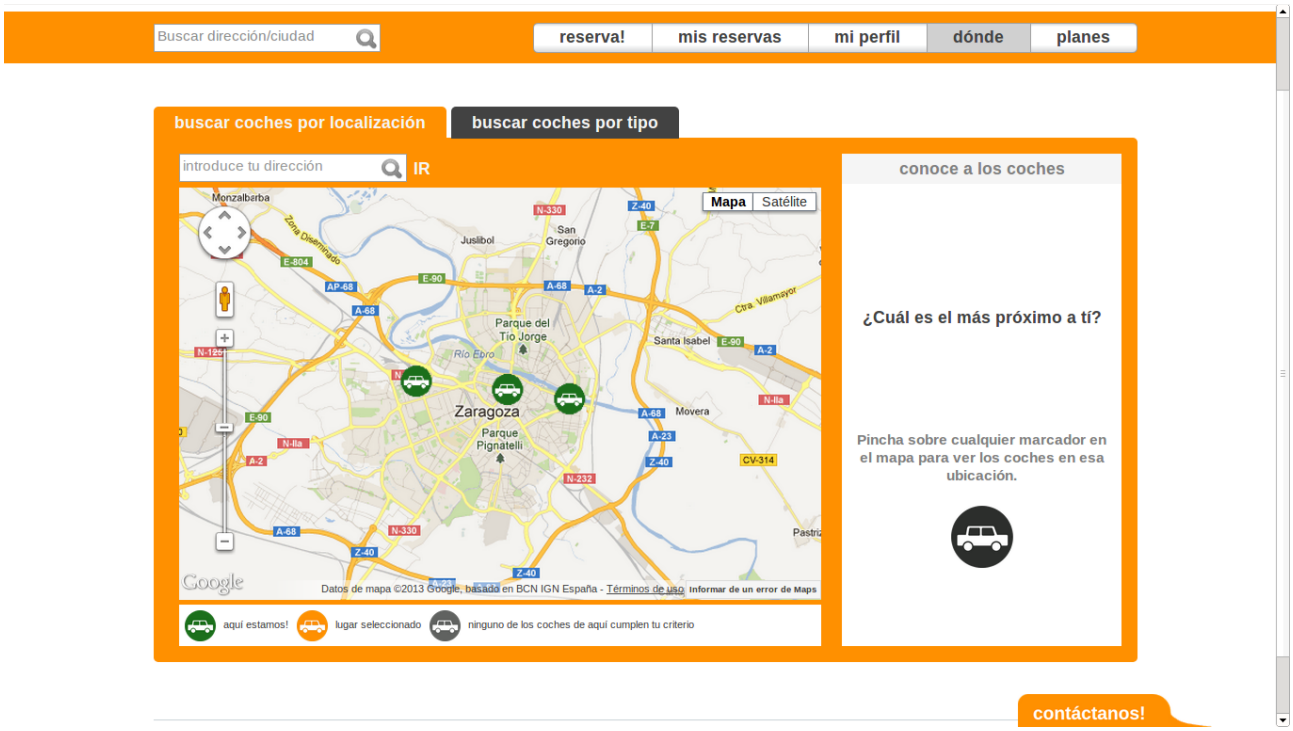
B.8 Registro de usuario (forma de pago: domiciliación)



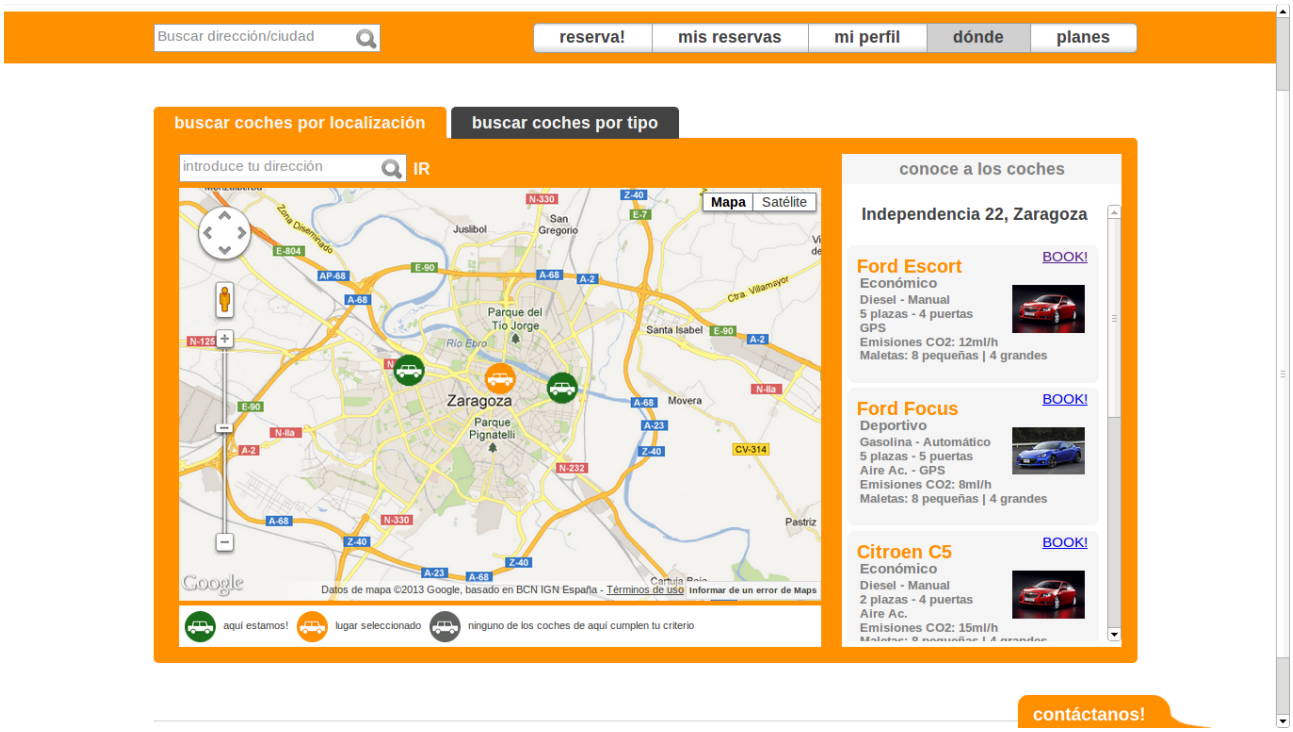
B.9 Login (formulario de acceso)



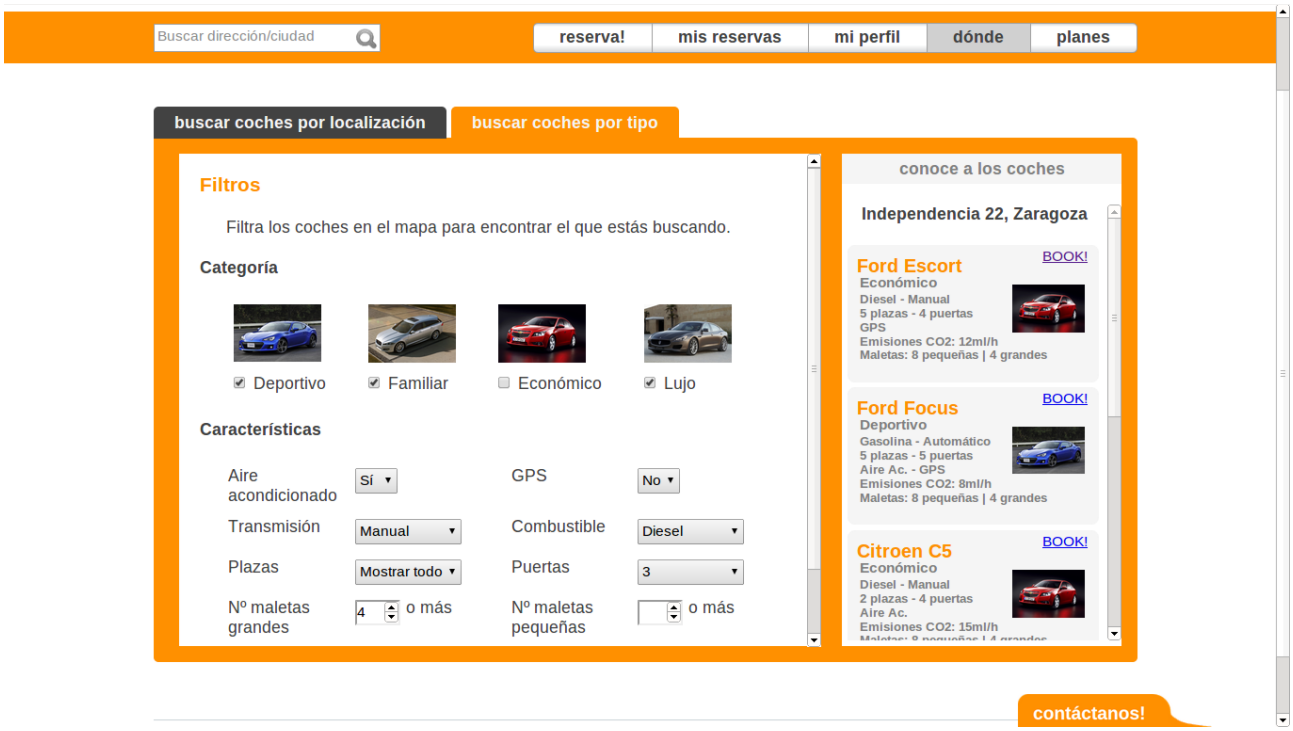
B.10 Mapa de vehículos de un grupo



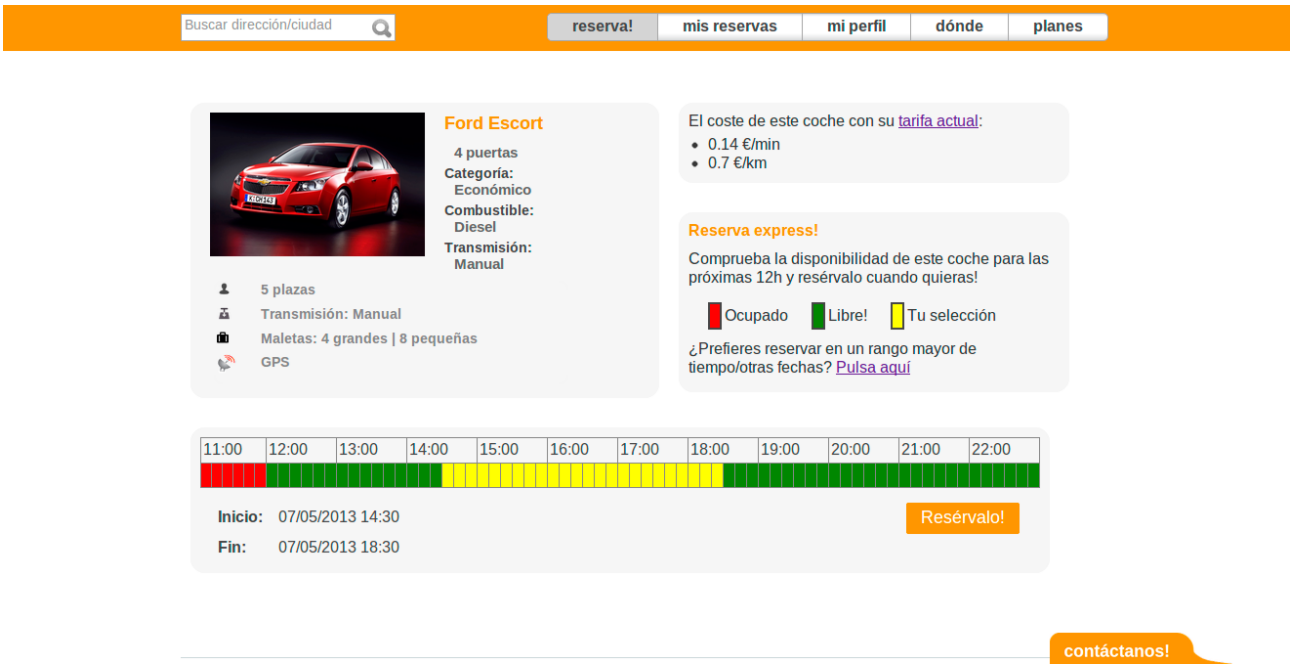
B.11 Mapa: vehículos de una ubicación



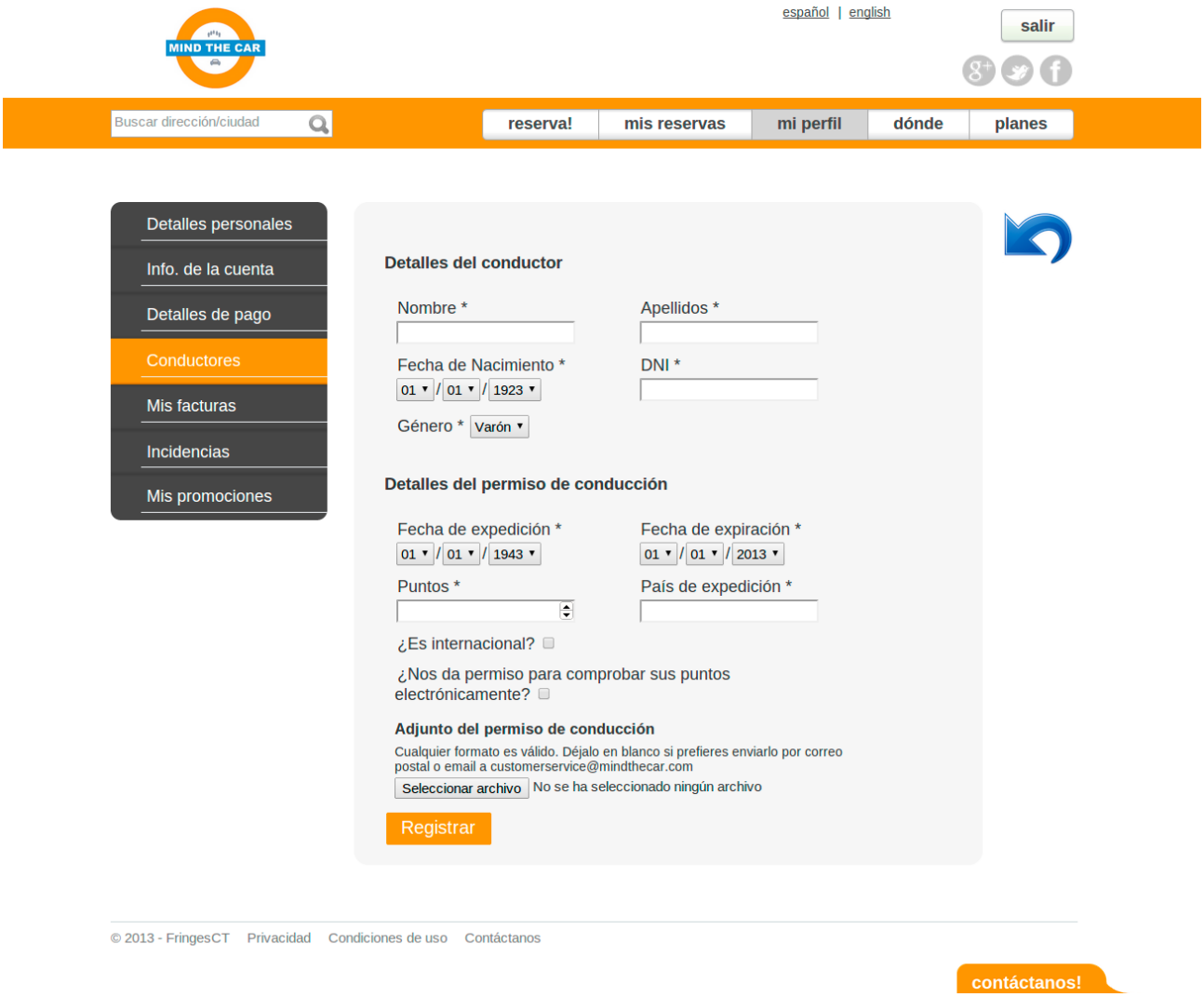
B.12 Filtros del mapa



B.13 Reserva de vehículos desde el mapa

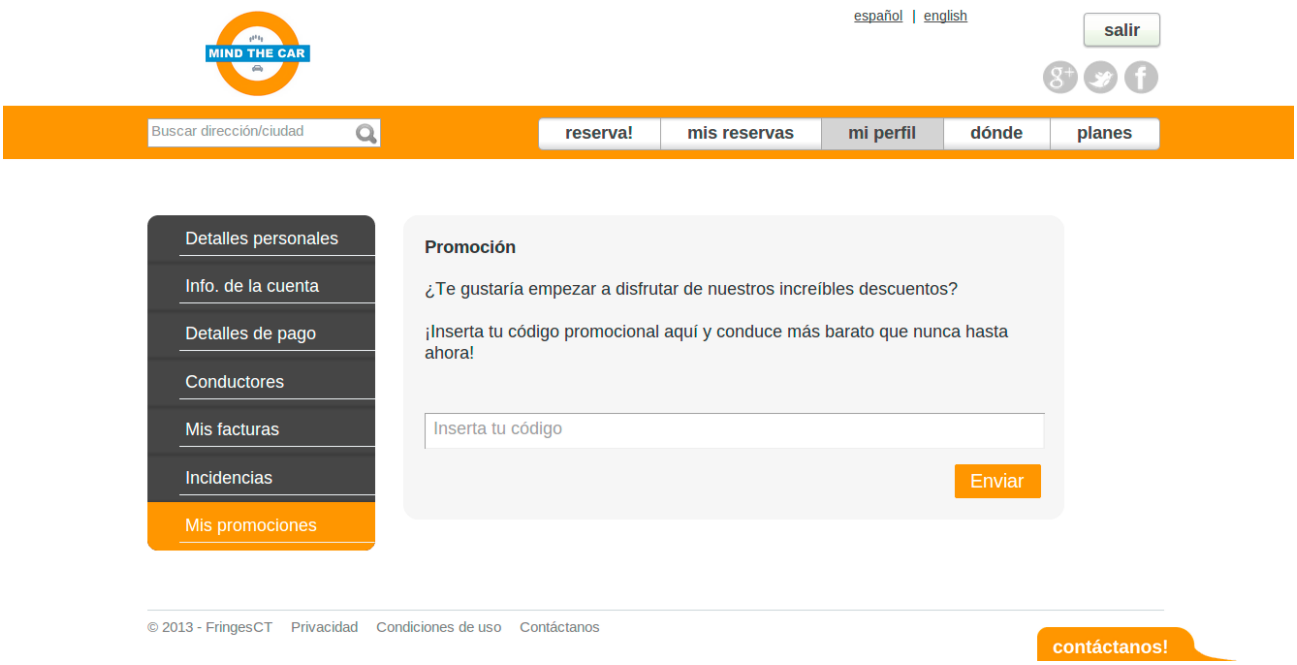


B.14 Formulario de alta de conductor



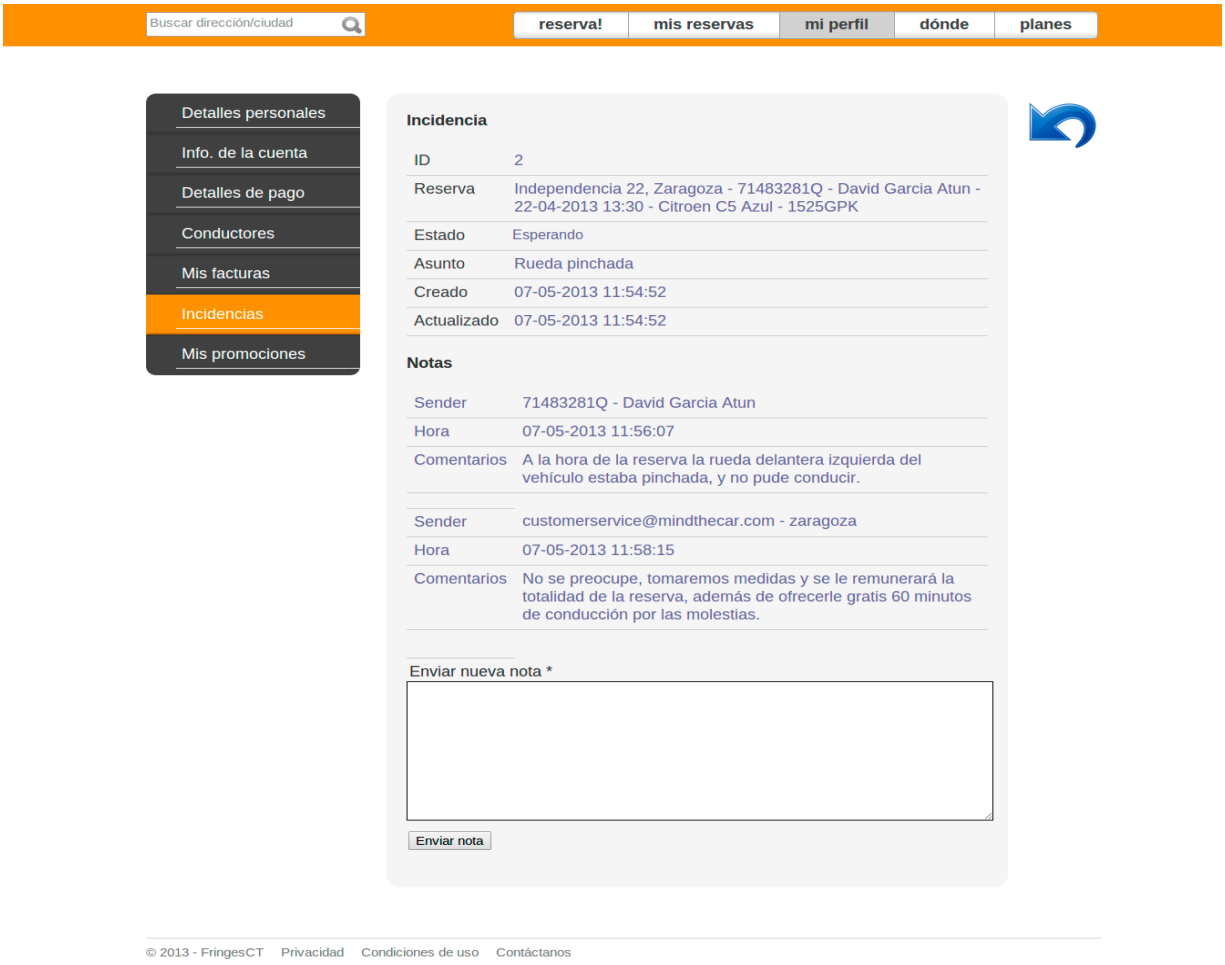
B.15 Filtro de facturas del usuario

B.16 Promociones del usuario



B.17 Información de la cuenta del usuario

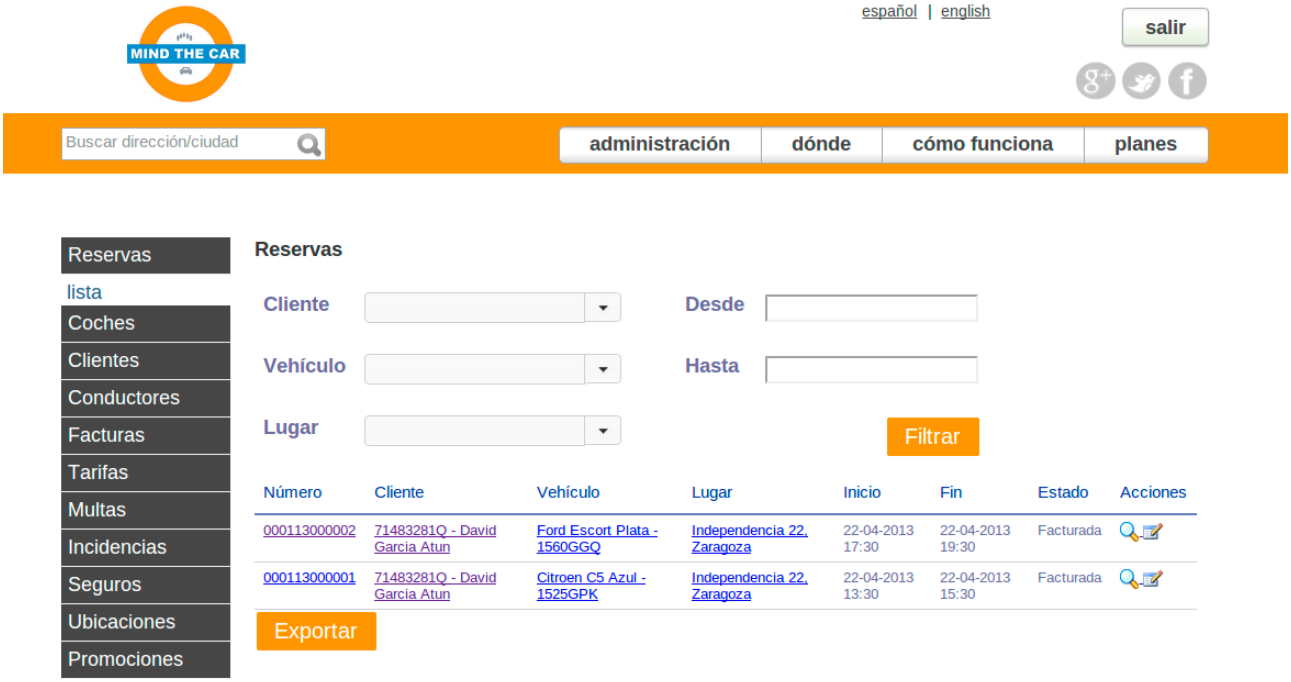
B.18 Gestión de una incidencia en la interfaz de usuario



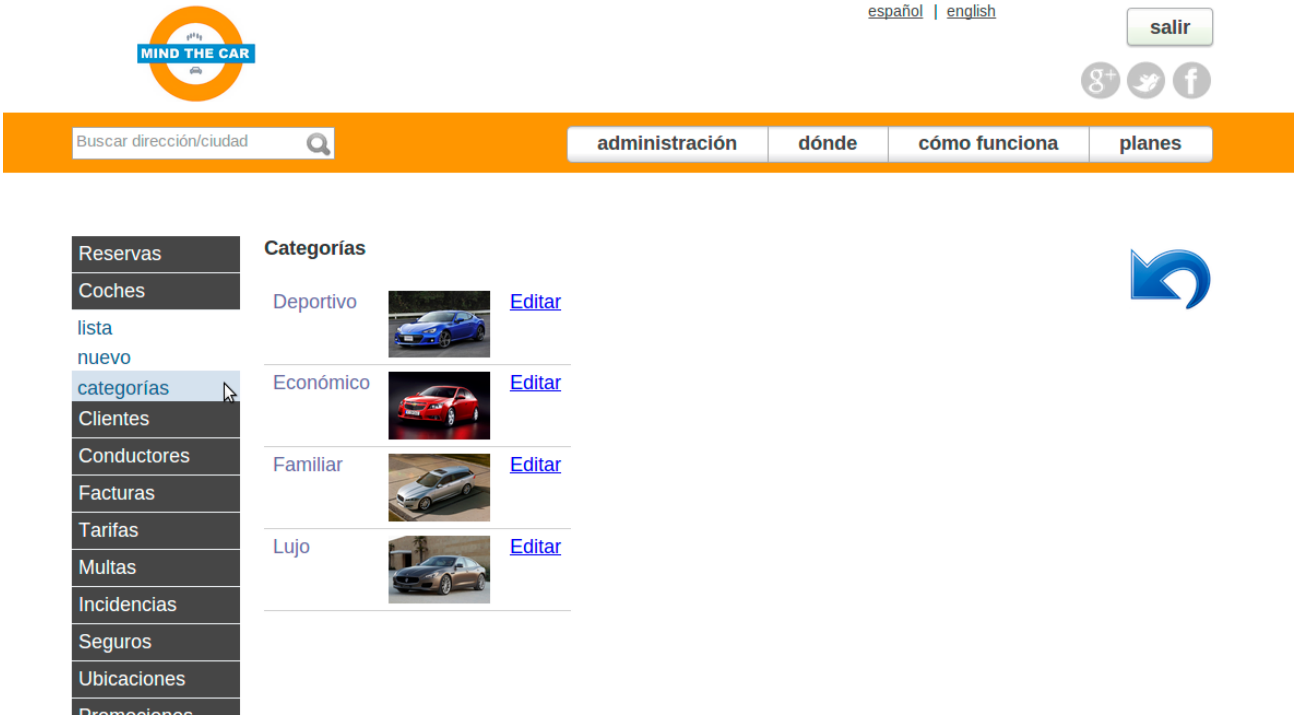
B.19 Gestión de tarjetas de un usuario

B.20 Modificación de reserva por un usuario

B.21 Filtro de reservas de un administrador

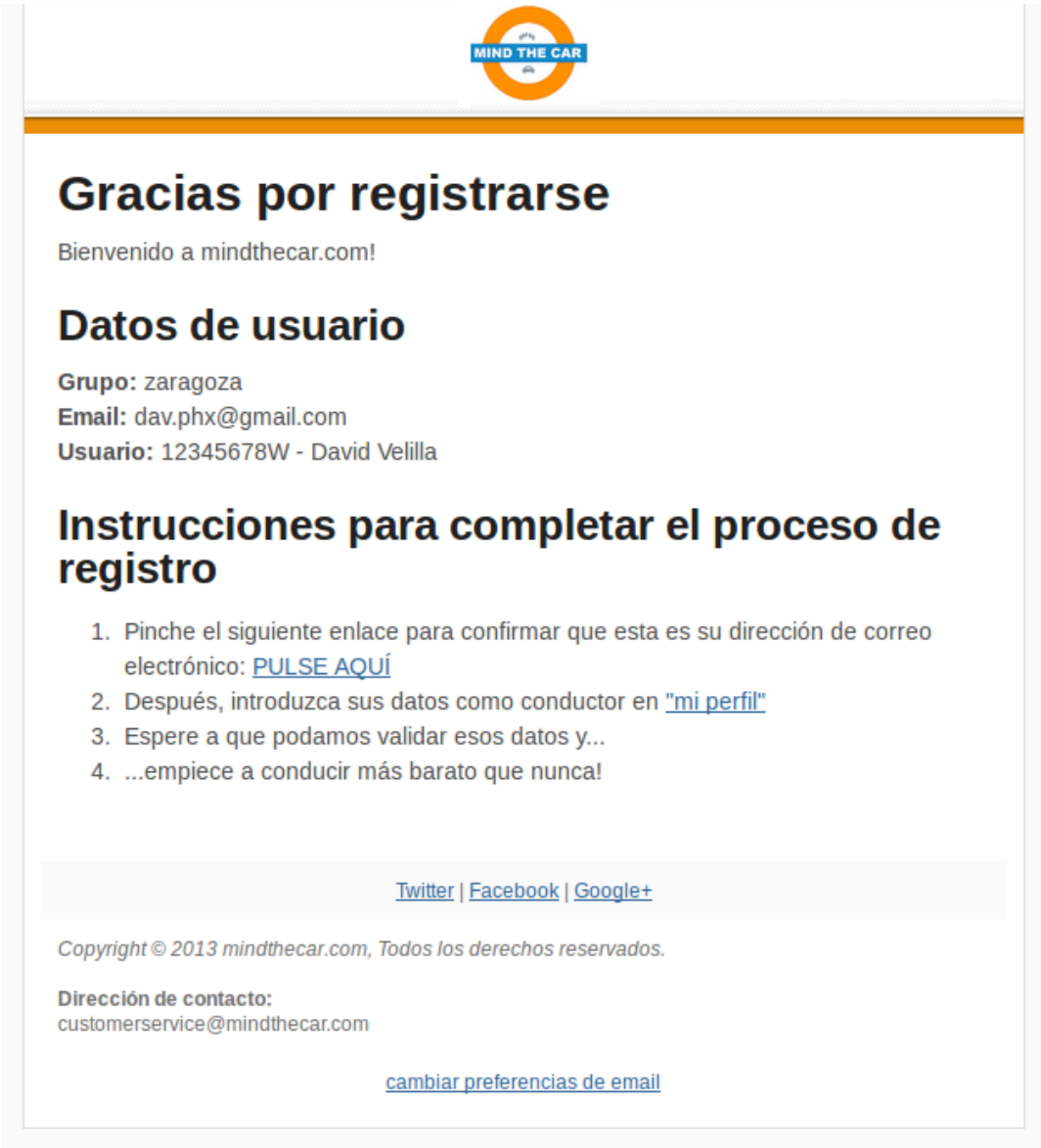


B.22 Categorías de los vehículos



B.23 Listado de vehículos de un grupo

B.24 Email de confirmación de usuario



Gracias por registrarse

Bienvenido a mindthecar.com!

Datos de usuario

Grupo: zaragoza
Email: dav.phx@gmail.com
Usuario: 12345678W - David Velilla

Instrucciones para completar el proceso de registro

1. Pinche el siguiente enlace para confirmar que esta es su dirección de correo electrónico: PULSE AQUÍ
2. Después, introduzca sus datos como conductor en "mi perfil"
3. Espere a que podamos validar esos datos y...
4. ...empiece a conducir más barato que nunca!

Twitter | Facebook | Google+

Copyright © 2013 mindthecar.com, Todos los derechos reservados.

Dirección de contacto:
customerservice@mindthecar.com

cambiar preferencias de email

B.25 Factura (página 1)

David Garcia Atun
C/Facturación 1
12491, España

DATOS

NÚMERO FACTURA:	000113000001	FECHA EMISIÓN:	13/05/2013
PERÍODO:	01/04/2013 - 30/04/2013	VENCIMIENTO:	13/05/2013
CÓDIGO CLIENTE:		TOTAL A PAGAR:	822.11 Euros
TITULAR:	David Garcia Atun	FORMA DE PAGO:	Domiciliación bancaria
NIF/CIF:	71483281Q	NÚMERO DE CUENTA:	**** **** ** ******3412

RESUMEN

Importe factura del mes de	April 2013	
Cliente nº		
Total (Base imponible):	679.43 Euros	Euros
Impuestos (IVA 21% S/ B.I.)	142.68 Euros	Euros
TOTAL FACTURA	822.11 Euros	Euros

SERVICIOS FACTURADOS

CONCEPTO	CANTIDAD	IMPORTE	TOTAL	
Cuota Mensual	1	39.9	39.9	Euros
Reservas	4		635.02	Euros

Nota: Todos los datos del cliente, reservas, coches, y facturas son ficticios y de prueba.

B.26 Factura (página 2)

Detalle de reservas del cliente

Nº DE RESERVA	Matrícula	TARIFA	CATEGORÍA	GRUPO
000113000003	9564HKR	Blue fare (T)	Deportivo	zaragoza
Fecha recogida	**Fecha entrega**	**Duración**	**Importe**	
08/05/2013	08/05/2013	0 días	0 Euros	
Recogida	**Entrega**	130 minutos	22.12 Euros	
20:30	22:38			
KMS iniciales	**KMS finales**	**KMS recorridos**		**Total**
42122	42179	57	97.13 Euros	119.25 Euros

Nº DE RESERVA	Matrícula	TARIFA	CATEGORÍA	GRUPO
000113000004	9564HKR	Blue fare (T)	Deportivo	zaragoza
Fecha recogida	**Fecha entrega**	**Duración**	**Importe**	
10/05/2013	10/05/2013	0 días	0 Euros	
Recogida	**Entrega**	120 minutos	20.92 Euros	
21:20	23:24			
KMS iniciales	**KMS finales**	**KMS recorridos**		**Total**
79248	79300	52	52.78 Euros	73.7 Euros

Nº DE RESERVA	Matrícula	TARIFA	CATEGORÍA	GRUPO
000113000005	1134GHT	Blue fare (T)	Lujo	zaragoza
Fecha recogida	**Fecha entrega**	**Duración**	**Importe**	
12/05/2013	12/05/2013	0 días	0 Euros	
Recogida	**Entrega**	440 minutos	129.04 Euros	
13:20	20:40			
KMS iniciales	**KMS finales**	**KMS recorridos**		**Total**
60035	60129	94	140.15 Euros	269.19 Euros

Nº DE RESERVA	Matrícula	TARIFA	CATEGORÍA	GRUPO
000113000006	1525GPK	Blue fare (T)	Economico	zaragoza
Fecha recogida	**Fecha entrega**	**Duración**	**Importe**	
13/05/2013	14/05/2013	1 días	34.72 Euros	
Recogida	**Entrega**	0 minutos	0 Euros	
14:40	09:37			
KMS iniciales	**KMS finales**	**KMS recorridos**		**Total**
114900	115001	101	138.17 Euros	172.89 Euros

OTROS CONCEPTOS

Penalizaciones	4.51	Euros		

TOTAL (Base imponible):	679.43	Euros
Impuestos (IVA 21% s/ B.I.)	142.68	Euros
TOTAL FACTURA	822.11	Euros

Referencias

[1]. http://ir.zipcar.com/releasedetail.cfm?ReleaseID=719904 "Zipcar Reports 2012"

[2]. http://www.symfony.com/ Framework Symfony. Info, instalación, y documentación.

[3]. http://www.scrum.org/Resources/What-is-Scrum Metodología ágil Scrum.

[4]. http://www.desymfony.com/ Curso de aprendizaje de Symfony.

[5]. http://www.w3schools.com/tags/ref_httpmethods.asp Métodos HTTP: GET y POST.

[6]. http://en.wikipedia.org/wiki/HTTPS Protocolo HTTP seguro (cifrado).

[7]. http://en.wikipedia.org/wiki/Secure_Sockets_Layer SSL (prot. seguridad sobre internet)

[8]. http://www.verisign.com/ Verisign. Autoridad de Certificación SSL.

[9]. http://www.doctrine-project.org/ Proyecto Doctrine. Librerías PHP para bases de datos.

[10]. http://www.w3schools.com/sql/ Información de SQL: Structured Query Language.

[11]. http://symfony.com/doc/2.1/reference/forms/types.html Documentación completa de los “tipos” de los atributos de entidades en campos de formularios de Symfony.

[12]. http://en.wikipedia.org/wiki/JavaScript Información general sobre JavaScript.

[13]. http://jquery.com/ Información y documentación de la librería jQuery.

[14]. http://www.w3schools.com/html/html5_form_attributes.asp Atributos de los formularios en html5 y sus diferencias de compatibilidad entre navegadores.

[15]. http://jqueryui.com/ Información y documentación de la librería jQuery User Interface.

[16]. http://api.jquery.com/jQuery.ajax/ Métodos jQuery de soporte de asincronía AJAX.

[17]. http://sensiolabs.com/en Información de SensioLabs, creadores de Symfony y Twig.

[18]. http://twig.sensiolabs.org/documentation Documentación de Twig.