Universitat de les Illes Balears

escola politècnica superior

Trabajo Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA, ESPECIALIDAD INFORMÁTICA

comparteixUIB

JAVIER GONZÁLEZ AYLMAN

Tutor
Ramón Más

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 9 de septiembre de 2018

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

TFG Trabajo Final de Grado

UIB Universidad de las Islas Baleares

MVC Modelo Vista Controlador

HTML HyperText Markup Language

PHP Hypertext Preprocessor

JS JavaScript

CSS Cascading Style Sheets

SGBD sistema de gestión de base de datos

SMTP Simple Mail Transfer Protocol

RESUMEN

El propósito de este Trabajo Final de Grado (TFG) es la creación de una página web, llamada comparteixUIB(Figura 5.1), para que todos los usuarios relacionados con la UIB puedan, básicamente, compartir vehículo para desplazarse hacía la UIB. En concreto, esta aplicación permite únicamente realizar dos tipos de trayecto que consisten en viajar desde un pueblo de Mallorca hacía la Universidad de las Islas Baleareas (UIB) y viceversa. Esta aplicación se ha desarrollado para que funcione en todos los navegadores web. Asimismo, la web presenta una diseño denominado *Full Responsive* [1] que es una "técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos, desde ordenadores de escritorio hasta tablets y móviles". El hecho de haber escogido este tipo de diseño se debe a que hoy en día los seres humanos hacen un mayor uso de los *Smartphones* que de los ordenadores convencionales en la mayoría de las situaciones cotidianas (2) puesto que es más práctico, en este caso, comprobar si hay trayectos disponibles según la necesidad de cada estudiante.

Figura 1: Logo de la aplicación web comparteix

ComparteixUIB puede llegar a ofrecer múltiples funciones dependiendo de si el usuario está previamente registrado a la página web o si, por el contrario, no lo está. El servicio común que ofrece la plataforma a los dos tipos de usuario es buscar un trayecto según unos filtros que el usuario, en este caso, ha indicado previamente, como son el tipo de viaje, la fecha y la hora aproximada. Las funcionalidades extra que recibe un usuario registrado son: poder crear nuevos tipos de viaje, modificar dicho viaje, hacer reservas de un trayecto que se adecue a sus necesidades, eliminar un viaje siempre y cuando se indique el motivo por el cual no puede llevar a cabo el trayecto, registrar diferentes vehículos para hacer viajes y, por último, valorar un viaje efectuado mediante un sistema de puntuación.

CAPÍTULO 1

INTRODUCCIÓN

1.1 Contexto del proyecto

El cambio climático afecta a todas las regiones del mundo [2]. Los casquetes polares se están fundiendo y el nivel del mar está subiendo. En algunas regiones, los fenómenos meteorológicos extremos y las inundaciones son cada vez más frecuentes, y en otras se registran olas de calor y sequías. Es por esto que no se debe dar la espalda a un problema que afecta a todo ser vivo del planeta.

La actividad de los seres humanos tiene una influencia cada vez mayor en el clima y las temperaturas al quemar combustibles fósiles, talar las selvas tropicales y explotar ganado.

Las enormes cantidades de gases producidos de esta manera se añaden a los que se liberan de forma natural en la atmósfera, aumentando el efecto invernadero y el calentamiento global [2]. Estos y otro factores son los que producen el *cambio climático*.

Unos de los causantes del cambio climático es la emisión de CO2 por parte de los coches. Según los datos de la Aema, el 13 % [3] de las partículas contaminantes en los 28 países de la Unión Europea son ocasionados por el transporte por carretera.

Para comprobar si la Universidad de las Islas Baleares (UIB) era ajena a este problema, se decidió hacer un estudio de contaminación de CO2. Este estudio, consistía en contar el número de coches y el número de ocupantes que entraban a la Universidad de las Islas Baleares (UIB). El período de estudio fue durante un día de clase y el tiempo de estudio fue de tres horas. El resultado obtenido se puede observar en la **Figura** 1.1.

Se puede apreciar una significante diferencia entre los vehículos en los que viaja una sola persona y los vehículos en los que que viaja más de una persona. La emisión de CO2 de un vehículo de gasolina es de un 2,196 kg CO2/l [4]. Supongamos que de media un vehículo recorre 10 km para llegar a la UIB y que tiene un cosumo medio de

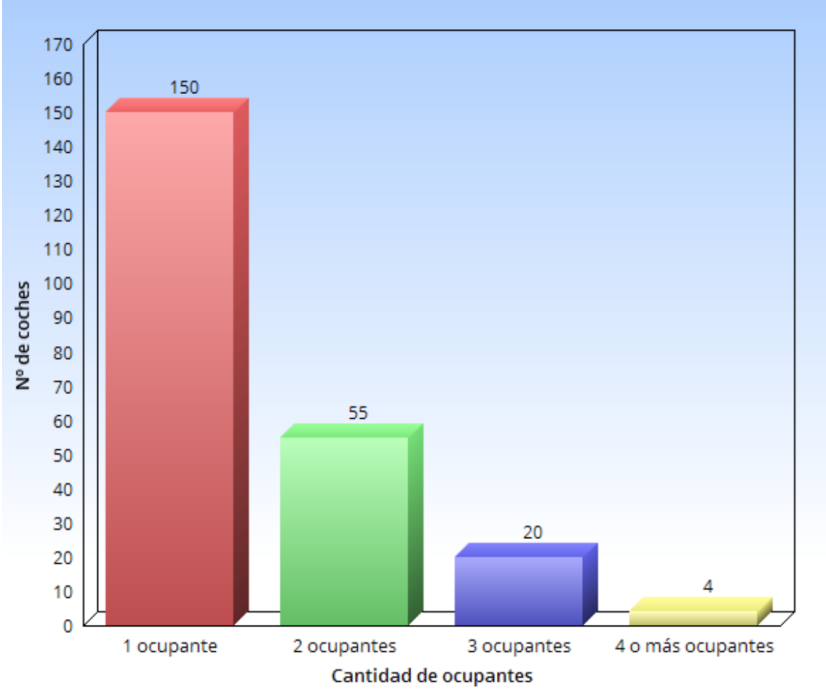

Figura 1.1: Logo de la aplicación web comparteix

3 litros. La emisión total del día en la que se llevó a cabo el estudio fue de:

(10 x 229) x 2,196 = 502,884 kg CO2

Tal y como está el mundo actual y dado un problema como es el cambio climático, la UIB no se puede quedar de brazos cruzados.

1.2 ¿De dónde surge la idea?

La idea de desarrollar la aplicacón web comparteixUIB nace del estudio que se llevó a cabo en la UIB para calcular las emisiones de CO2 producidas.

Para reducir esta emisión, era necesario crear un sistema para intentar minimizarlas. Después de hablar con el profesor Ramón Más sobre diferentes soluciones, se optó por la creación de una aplicación web que permitiera a los usuarios compartir coche para viajar hasta la UIB. A partir de este momento nace comparteixUIB (**Figura** 5.1).

1.3 Objetivos

1.3.1 Objetivos del proyecto

Uno de los objetivos principales de este proyecto es el desarrollar una web, sencilla y de fácil uso de cara al usuario, que permita a estudiantes y profesores de la UIB hacer

Figura 1.2: Logo de la web

reservas para viajar a la UIB. Otro de los grandes objetivos de este proyecto es que, con la utilización de la web, se puedan reducir las emisiones de CO2 que se emiten actualmente en la UIB. Los objetivos derivados del desarrollo son:

- Permitir, a personas asociadas a la UIB, registrarse en comparteixUIB para poder utilizar todas sus funciones.
- Personas que aún no forman parte de la UIB, puedan acceder a la aplicación web para que la conozcan.
- Crear, modificar o eliminar viajes de un usuario.
- Crear, modificar o eliminar vehículos de un usuario.
- Crear, modificar o eliminar reservas de un usuario.
- Permitir puntuar, por parte de los viajeros, trayectos que ya se hayan llevado a cabo.

1.3.2 Objetivos personales

Los objetivos personales que satisface el desarrollo de este proyecto son:

- Contribuir con la mejora del medio ambiente mediante la reduccción de CO2.
- Ayudar a mejorar la calidad de vida de las personas.
- Demostrar los conocimientos obtenidos durante la formación académica y aplicarlos a un desarrollo a gran escala.
- Obtener y aprender nuevos conocimientos que puedan ser reutilizados en un futuro para afrontar nuevos proyetos.

CAPÍTULO 2

ANÁLISIS

En este capítulo se analizan en detalle los requisitos de esta página web y las tecnologías utilizadas para su realización.

2.1 Especificación de requisitos

Para poder dar una visión global sobre comparteixUIB, a continuación se darán más detalles de las especificaciones técnicas que tendrá el proyecto. Para ello, haremos uso de lo que se conoce en el mundo de la ingeniería como los requisitos funcionales y no funcionales.

Como bien sabemos, comparteixUIB, es un proyecto que va dirigido a usuarios relacionados con la UIB . Debido a que el cliente final del producto está muy definido y sigue el estándar que nos brinda el libro PMBOOK [5], se decidió utilizar la técnica de obtención de requisitos denominada *Brainstorming* que tiene como objeto que los diferentes miembros del grupo aporten, durante un tiempo previamente establecido el mayor número de ideas posibles sobre un tema o problema determinado. Interesa, en primer lugar, la cantidad de ideas; conviene que las aportaciones sean breves, que nadie juzgue ninguna, que se elimine cualquier crítica o autocrítica y que no se produzcan discusiones ni explicaciones [6].

Para llevar a cabo esta técnica, se llevaron a cabo diferentes reuniones entre estudiantes de la UIB. En los primeros encuentros, lo que se hizo fue explicar en que consistía el proyecto y a qué usuarios finales iba dirigido. Una vez que los diferentes participantes conocían la información necesaria acerca de la aplicación, cada uno exponía de 3 a 5 ideas que se iban apuntando en una pizarra para ir separando los diferentes requisitos.

Después de que cada integrante había expuesto sus ideas, lo que se hizo fue realizar

una lista de *requisitos finales* para dejar definida la finalidad que iba a tener la aplicación web:

1. El usuario podrá ver los próximos viajes de los dos tipos de trayecto disponible.
2. El usuario podrá hacer reservas de un viaje.
3. El usuario podrá registrarse en la aplicación web.
4. El usuario podrá ver el ranking de los mejores conductores.

En la sección 5.2 se describirá el futuro de este proyecto con las funcionalidades que no se desarrollaron de las que se habían obtenido durante el *Brainstorming*.

2.2 Requisitos de negocio

Los requisitos de negocio, conocidos también como requisitos funcionales, son "los requerimientos funcionales o declaraciones de los servicios que proveerá el sistema, de manera que éste reaccionará a situaciones particulares"(3). En este apartado se han clasificado los requisitos en subgrupos: requisitos para usuario no registrados y requisitos para usuarios registrados.

Usuario no registrado

1. El usuario podrá ver los próximos viajes de los dos tipos de trayecto disponibles.
2. El usuario no podrá hacer reservas de un viaje.
3. El usuario podrá registrarse en la aplicación web.
4. El usuario podrá ver el ranking de los mejores conductores.

Usuario registrado

1. El usuario podrá recuperar su contraseña en caso de que la haya olvidado.
2. El usuario podrá modificar sus datos de contacto.
3. Un estudiante registrado debe poder hacer uso del panel de usuario, donde podrá gestionar los diferentes contenidos de su perfil.
4. El usuario podrá añadir vehículos a su perfil.
5. El usuario podrá hacer modificaciones de un vehículo que tenga creado.
6. El usuario podrá eliminar un vehículo que tenga creado siempre y cuando no tenga viajes asociados.
7. Un estudiante que haya llevado a cabo un viaje, podrá poner una puntuación al viaje mediante un número de estrellas del uno al cinco.

8. El usuario podrá hacer reservas de diferentes viajes.
9. A la hora de hacer una reserva, el estudiante recibirá un email con los datos de la reserva.
10. El usuario podrá modificar un viaje que haya creado.
11. El usuario podrá eliminar un viaje, siempre y cuando explique el motivo.
12. Un estudiante, que vaya a realizar un viaje y el conductor no pueda llevar a cabo dicho trayecto, recibirá un email a su correo con el motivo por el cual el conductor no ha podido realizar el viaje.
13. El usuario que haya creado un viaje no podrá hacer una reserva de su propio viaje.

2.3 Requisitos técnicos

Los requisitos de negocio, conocidos como requisitos no funcionales, son ”aquellos requerimientos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de este como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento.”

1. El sistema debe ser compatible con cualquier navegador web: *Google Chrome*, *Internet Explorer*, *Firefox* ...
2. El sistema debe ser capaz de soportar más de un usuario a la vez.
3. El sistema debe ser capaz de funcionar en multiidioma. Para este proyecto en concreto se da la posibilidad de poder utilizar la web en castellano y catalán. El número de lenguajes puede aumentar.
4. La web debe mostrar mensajes de confirmación por cada acción que realice un usario, por ejemplo al crear un viaje, hacer una reserva, etc.
5. El sistema debe mostrar en el mapa los viajes que están disponibles y cuales no lo están mediante una serie de colores.
6. La web debe ser compatible con la API de Google Maps [7].

2.4 Análisis tecnológico

Dado que en este proyecto web han ido apareciendo una serie de necesidades tecnológicas, se ha optado por la utilización de herramientas para poderlas llevar a cabo. A continuación, se justifica el uso de dichas herramientas:

1. *Lenguajes para desarrollar aplicaciones web*: debido a que se trata de un proyecto de una aplicación web, es necesario el uso de lenguajes que nos permitan llevar a cabo el desarrollo de manera satisfactoria.

2. *Soporte para el desarrollo de aplicaciones web*: necesitamos soporte que nos permite desarrollar de manera eficaz nuestro proyecto. Además, debe ser compatible con los lenguajes que se necesitan para desarrollar dicha aplicación.

3. *Tecnología que permita el almacenamiento y obtención de datos*: dado que en esta página web es necesario el guardado y la obtención de datos, se precisa de un sistema de almacenaje de datos así como de obtención de los mismos.

4. *Herramientas de soporte para el desarrollo*: control de versiones, uml, etc

2.4.1 Herramientas utilizadas

En este apartado se van a exponer las diferentes herramientas utilizadas que han permitido llevar a cabo este proyecto.

HTML

El lenguaje que predomina en el mundo actual y con el que se ha llevado a cabo prácticamente todo el desarollo de la página web, ha sido HyperText Markup Language (HTML) [8]. HTML significa **Lenguaje de Marcado de Hypertexto** por sus siglas en inglés **HyperText Markup Language**, es un lenguaje que pertenece a la familia de los **lenguajes de marcado** y es utilizado para la elaboración de páginas web. El estándar HTML lo define la W3C (World Wide Web Consortium) y actualmente HTML se encuentra en su versión HTML5. Un ejemplo sencillo de la estructura html sería la que podemos ver en la *Figura 2.1*:

```
<html>
 <head>
  <script type="application/javascript">
    function draw() {
      var canvas = document.getElementById("canvas");
      if (canvas.getContext) {
        var ctx = canvas.getContext("2d");

        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (10, 10, 55, 50);

        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 50);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

Figura 2.1: Estructura básica de HTML

Cabe destacar que HTML no es un lenguaje de programación ya que no cuenta con funciones aritméticas, variables o estructuras de control propias de los lenguajes de programación. HTML genera únicamente páginas web estáticas. Sin embargo, HTML se puede usar en conjunto con diversos lenguajes de programación para la creación de páginas web dinámicas.

CSS

A la hora de estructurar los elementos que contiene la web y hacerla uniforme, estilísticamente hablando, se ha hecho uso del lenguaje de estilos denominado Cascading Style Sheets (CSS).

CSS, es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML [9]. Además, este lenguaje es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos"). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes. En la *Figura 2.2*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>Mi primera página con estilo</title>
  <style type="text/css">
  body {
    color: purple;
    background-color: #d8da3d }
  </style>
</head>

<body>
[etc.]
```

Figura 2.2: Estructura básica de HTML combinado con CSS

JS

Para mejorar la usabilidad de la página web, se ha optado por utilizar JS. *JavaScript* [10] que es un lenguaje de programación que te permite realizar actividades complejas en una página web — cada vez más una página web hace más cosas que solo mostrar información estática — como mostrar actualizaciones de contenido en el momento, interactuar con mapas, animaciones gráficas 2D/3D etc. — puedes estar seguro que JavaScript está involucrado. En la *Figura 2.3*, podemos ver un ejemplo sencillo de lo que se puede hacer con el JS:

```
var para = document.querySelector('p');

para.addEventListener('click', updateName);

function updateName() {
  var name = prompt('Enter a new name');
  para.textContent = 'Player 1: ' + name;
}
```

Figura 2.3: Función simple con JavaScript

API Google Maps

La mejor manera de mostrar el funcionamiento de esta aplicación web es hacer uso de la API de Google Maps ya que la principal función de esta página web es poder buscar viajes. Si hablamos de mapas en la web, sin duda la referencia más conocida es Google Maps [7]. En su momento revolucionó la forma en que los mapas podían ser vistos en internet. Hoy en día mantiene su liderazgo en muchas cuestiones, como los servicios de geolocalización, datos de tráfico, cálculo de rutas...y además con cobertura global. En la *Figura* 2.4, podemos ver un sencillo ejemplo de inserción de un mapa de Google en un proyecto web:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- This stylesheet contains specific styles for displaying the map
         on this page. Replace it with your own styles as described in the
         documentation:
         https://developers.google.com/maps/documentation/javascript/tutorial -->
    <link rel="stylesheet" href="/maps/documentation/javascript/demos/demos.css">
  </head>
  <body>
    <div id="map"></div>
    <script>
      function initMap() {
        // Create a map object and specify the DOM element for display.
        var map = new google.maps.Map(document.getElementById('map'), {
          center: {lat: -34.397, lng: 150.644},
          zoom: 8
        });
      }

    </script>
```

Figura 2.4: Cómo insertar una mapa de manera fácil

PHP

Para llevar a cabo funciones internas de nuestra web, se ha hecho uso del lenguaje de programación *Hypertext Preprocessor* o más comunmente conocido como PHP. PHP es un lenguaje de muy potente que, junto con HTML], permite crear sitios web dinámicos. Php se instala en el servidor y funciona con versiones de Apache, Microsoft IIs, Netscape Enterprise Server y otros.

La forma de usar PHP es insertando código php dentro del código html de un sitio web. Cuando un cliente (cualquier persona en la web) visita la página web que contiene este código, el servidor lo ejecuta y el cliente sólo recibe el resultado. Su ejecución, es por tanto en el servidor, a diferencia de otros lenguajes de programación que se ejecutan en el navegador. En la *Figura* 2.5, podemos ver un ejemplo práctico para insertar lenguaje PHP junto con HTML.

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Ejemplo</title>
    </head>
    <body>

        <?php
            echo "¡Hola, soy un script de PHP!";
        ?>

    </body>
</html>
```

Figura 2.5: Ejemplo práctico de PHP junto con HTML

MySQL

Dado que esta página necesita de un sistema de almacenamiento de datos, es necesario un método para llevar a cabo tal fin. Además, otro requisito indispensable es que la web va a ser usada por varios usuarios a la vez, por ello, es de vital importancia que el sistema utilizado para almacenar dicha información sea multiusaurio. Para llevar a cabo el almacenado de datos se ha hecho uso de *MySQL*.

MySQL[11], es un sistema de gestión de base de datos (SGBD). Este gestor de base de datos es multihilo y multiusuario, lo que le permite ser utilizado por varias personas al mismo tiempo, e incluso, realizar varias consultas a la vez, lo que lo hace sumamente versátil. Para poder interactuar con nuestra *Base de datos*, utilizamos querys, que son consultas u operaciones para poder insertar, actualizar y/o borrar datos de nuestro sistema. En la Figura 2.6 podemos ver unos ejemplos prácticos que nos permiten interactuar con nuestra *base de datos*:

LARAVEL

Ya que este TFG se va a llevar a cabo mediante la técnica de *Modelo Vista Controlador (MVC)* que se explicará en el capítulo 3, se ha hecho uso del framework conocido

```
SELECT * FROM mi_tabla [Muestra todos los resultados
de nuestra base de datos]
SELECT id,nombre,apellidos FROM mi_tabla [Muestra todos
los resultados de nuestra base de datos, pero solo nos
muestra las columnas id, nombre y apellido]
SELECT * FROM mi_tabla WHERE nombre = 'Javi' [Muestra
todos los resultados de nuestra base de datos donde el
nombre sea 'Javi']
SELECT * FROM mi_tabla WHERE pais = 'Italia' ORDER by id ASC
[Muestra los resultados de nuestra base de datos que contengan
'Italia' en la columnas de pais y nos muestra los resultados de
manera ascendente por el id de cada registro]
```

Figura 2.6: Querys sencillas para interarctuar con nuestra base de datos

como *Laravel*.

Laravel [12] es un framework de código abierto para el desarrollo de aplicaciones web en PHP 5 que posee una sintaxis simple, expresiva y elegante. Fue creado en 2011 por Taylor Otwell, inspirándose en Ruby on Rails y Symfony, de los cuales ha adoptado sus principales ventajas.

Laravel facilita el desarrollo simplificando el trabajo con tareas comunes como la autenticación, el enrutamiendo, gestión de sesiones, el almacenamiento en caché, etc. Algunas de las principales características y ventajas de Laravel son:

1. Está diseñado para desarrollar bajo el patrón MVC (modelo - vista - controlador), centrándose en la correcta separación y modularización del código, lo que facilita el trabajo en equipo, así como la claridad, el mantenimiento y la reutilización del código.

2. Integra un sistema ORM de mapeado de datos relacional llamado Eloquent aunque también permite la construcción de consultas directas a base de datos mediante su Query Builder.

3. Permite la gestión de bases de datos y la manipulación de tablas desde código, manteniendo un control de versiones de las mismas mediante su sistema de Migraciones.

4. Utiliza un sistema de plantillas para las vistas llamado Blade, el cual hace uso de la cache para darle mayor velocidad. Blade facilita la creación de vistas mediante el uso de layouts, herencia y secciones.

5. Facilita la extensión de funcionalidad mediante paquetes o librerías externas. De esta forma es muy sencillo añadir paquetes que nos faciliten el desarrollo de una aplicación y nos ahorren mucho tiempo de programación.

6. Incorpora un intérprete de línea de comandos llamado Artisan que nos ayudará con una gran cantidad de tareas rutinarias como la creación de distintos componentes de código, trabajo con la base de datos y migraciones, gestión de rutas, cachés, colas, tareas programadas, etc.

BLADE

A modo de complemento para el uso de *Laravel*, se ha hecho uso de un motor de plantillas para poder trabajar de manera más sencilla, ágil y efectiva. Este motor de plantillas es conocido como *Blade*. Este motor nos permite una infinidad de posibilidades a la hora de programar nuestra web de manera más cómoda. Para ver un ejemplo de la sencillez de utilizar blade, fijémonos en la figura 2.7, donde podemos ver como insertar código dentro de etiquetas HTML:

```
<div class="panel-body" align="center">

    <!—Aquí vendrá el resultado de la búsqueda—>
    {!!Form::open()!!}

        {{— @if (!$resultados) —}}
        @if(count($viajes) == 0)
            <h1 style="color:red;">Cap viatge trobat</h1>
        @elseif(count($viajes) == 1)
            <!—<h1>{{ count($viajes) }} viaje encontrado.</h1>—>
        @else
            <!—<h1>{{ count($viajes) }} viajes encontrados.</h1>—>
        @endif
```

Figura 2.7: Contador básico entre etiquetas HTML

CAPÍTULO 3

IMPLEMENTACIÓN

Esta web se basa en el modelo **cliente-servidor** [13], que es un modelo que describe el proceso de interacción entre el sistema local (el cliente) y el remoto (el servidor). Generalmente, los clientes y los servidores se comunican entre sí a través de una red, pero también pueden residir ambos en un mismo sistema (el mismo hardware). Es por ello que, comparteixUIB, se ha desarrollado mediante la técnica de desarollo conocida como Modelo Vista Controlador (MVC). Esto se debe a que este es un proyecto web donde predomina el uso de las **interfaces de usuario**. Otro aspecto importante para hacer uso de esta técnica es que, gracias a ella, el proyecto podrá aumentar el número de funcionalidades de manera ágil y sencilla.

En líneas generales [14], el estilo de arquitectura MVC es una propuesta de diseño de software utilizada para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos. MVC divide las aplicaciones en tres niveles de abstracción:

1. **Modelo**: es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la *vista* aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al *modelo* a través del *controlador*.

2. **Vista**: esta capa es la encargada de presentar el *modelo* (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de

usuario), por tanto requiere de dicho *modelo* la información que debe representar como salida.

3. **Controlador**: esta capa es la encargada de responder a los eventos (usualmente acciones del usaurio) e invoca las peticiones al *modelo* cuando se hace alguna solicitud sobre la información (por ejemplo crear un documento o editar un registro de la base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el *modelo* (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el *controlador* hace de intermediario entre la *vista* y el *modelo*.

El funcionamiento básico del patrón MVC [15], puede resumirse en:

1. El usuario realiza una petición.
2. El **controlador** captura la petición.
3. El **controlador** hace la llamada al **modelo** correspondiente.
4. El **modelo** sera el encargado de interactuar con la base de datos.
5. El **controlador** recibe la información y la envía a la vista.
6. La **vista** muestra la información.

En la figura 3.1, tenemos una visión general del funcionamiento básico del patrón de diseño MVC:

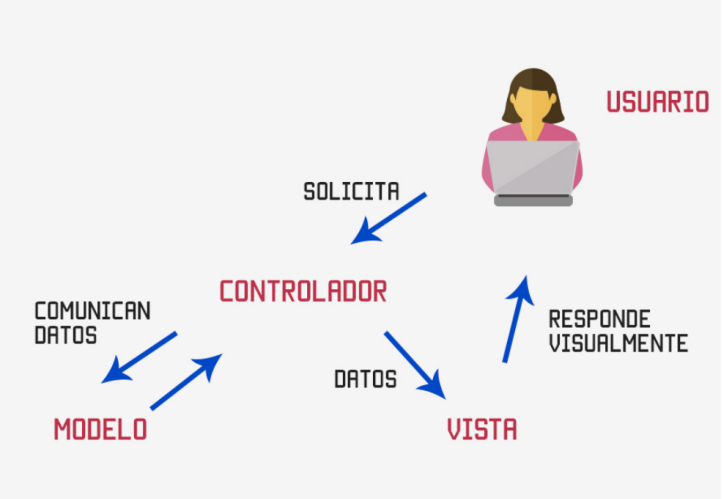

Figura 3.1: Comunicación entre las diferentes secciones del modelo MVC

Para esta aplicación web, en comparteixUIB está bien definido qué parte representa cada elemento del diseño MVC. En la siguiente figura 3.2 podemos ver como quedaría representado cada elemento de esta estructura:

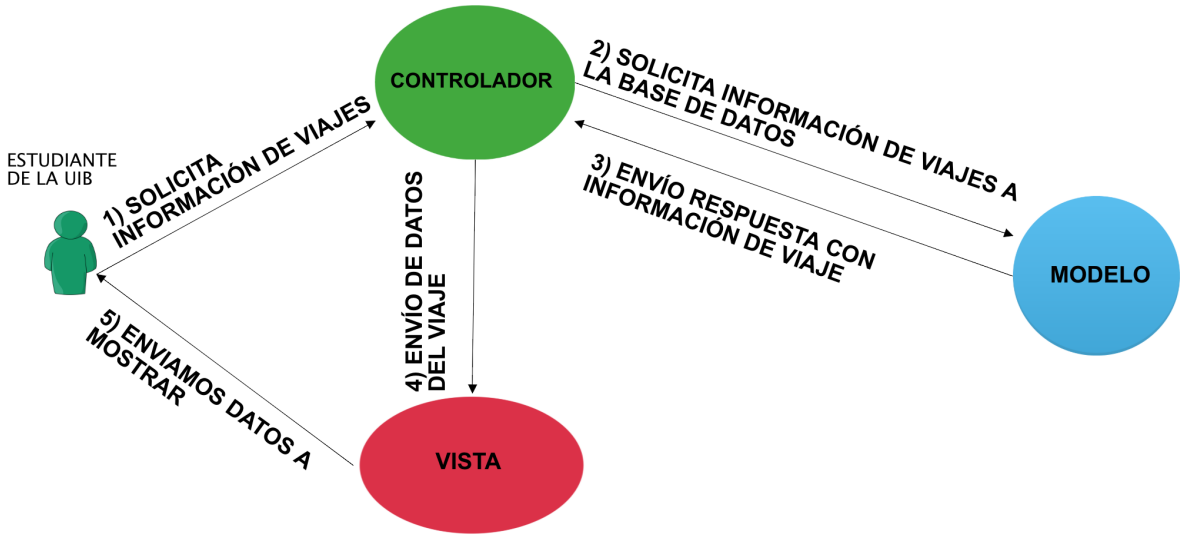

Figura 3.2: Comunicación entre los diferentes componentes del modelo MVC

La parte de **cliente** se ve representada por el nivel de la **vista** en el modelo MVC. Ésta ha sido desarrollada mediante el uso de los lenguajes de programación web más conocidos, como pueden ser HTML, JS, etc.

Además, para llevar a cabo la aplicación, como bien se ha dicho antes, se ha hecho uso del framework Laravel, que a su vez utiliza un motor de plantillas llamado Blade para que la interacción de las **vistas** con los diferentes **controladores** del sistema sea más accesible. El uso de **Blade** nos permite evitar que dupliquemos código de manera innecesaria ya que Laravel [16] usa unos archivos llamados *plantillas* o *templates*, que tienen los segmentos de código que se repiten en más de una **vista**, como por ejemplo la barra de navegación, un menú de opciones, la estructura del acomodo de nuestro proyecto, y como deben de estar prácticamente presentes en todas las pestañas, pues no tiene sentido tener que ir repitiendo las definiciones de estos elementos en todas las pestañas de la web.

Por ejemplo, la barra de navegación o el footer son elementos repetitivos en toda la aplicación, es por eso que se ha creado un fichero llamado *app.blade.php* (template), donde se han definido todos los CSS, JS, llamadas a la API de google maps, etc. En este template, tenemos una sentencia que nos sirve para indicar que en ese punto se van a añadir otras vistas como puede ser el buscador de viajes, la creación de un nuevo vehículo, etc.

En la **Figura** 3.3, podemos ver la definición de nuestro *template* principal y en la línea 131 tenemos una sentencia para indicar que en ese punto se incluye otra vista:

app.blade.php

```
1   <!DOCTYPE html>
2   <html lang="{{ config('app.locale') }}">
3       <head> ...
48      </head>
49      <body>
50          <header id="header">
51              <div class="container">
52                  <div class="row"> ...
76                  </div>
77              </div>
78              <div class="navbar navbar-inverse" role="banner">
79                  <div class="container">
80                      <div class="navbar-header">
81                          <button type="button" class="navbar-toggle" data-toggle="collapse" data-
                            target=".navbar-collapse">
82                          <span class="sr-only">Toggle navigation</span>
83                          <span class="icon-bar"></span>
84                          <span class="icon-bar"></span>
85                          <span class="icon-bar"></span>
86                          </button>
87                      </div>
88                      <div class="collapse navbar-collapse"> ...
125                     </div>
126                 </div>
127             </div>
128         </header>
129         <!--/#header-->
130
131         @yield('content')
132
133         <footer id="footer">
134             <div class="container">
135                 <div class="row">
136                     <div class="col-sm-12 text-center bottom-separator">
137                         <img src="{{ asset('images/home/under.png') }}" class="img-responsive inline
```

Figura 3.3: Template principal del proyecto

En la **Figura** 3.4 se puede apreciar un ejemplo del funcionamiento de las plantillas mediante el motor de plantillas **Blade**. En este ejemplo en concreto, tenemos un fichero PHP que es la página inicial de nuestra web. En ella se puede ver la llamada que realizamos para insertar el código de la página **index2** dentro de la clase **app.blade**. Por tanto, una de las funciones que tienen las **vistas** es presentar la información al usuario en las diferentes ventanas de la web. En la *Figura* 3.5, vemos una ejemplo de cómo una vista procesaría la información que le llega desde un **controlador**.

Otra de las funciones principales que tienen las **vistas** es la de realizar las diferentes llamadas o peticiones a los **controladores** mediante peticiones HHTPS para obtener la diferente información que el usuario solicita.

En la parte de cliente también se engloba al **controlador**, que es el encargado de recibir las peticiones provenientes de las **vistas**. Los **controladores**, como ya se ha dicho, son un mecanismo que nos permite agrupar la lógica de peticiones HHTP relacionadas y de esta forma organizar mejor nuestro código. Estos elementos del proyecto no son más que simples archivos PHP que engloban las diferentes funciones encargadas de la comunicación con las **vistas** y con los **modelos**. En la **Figura** 3.6 podemos ilustrar de una manera clara cómo se propaga la información de las peticiones de las **vistas** hasta los **modelos**. En el ejemplo podemos ver una función que se encarga de obtener un tipo de viajes. Para ello, el **controlador** obtiene todos los parámetros enviados por

```
index2.blade.php
1  @extends('layouts.app')
2  @section('content')
3  <section id="home-slider">
4     @if(Session::has('message'))
5     <div class="alert alert-success alert-dismissible" role="alert">
6        <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-
         hidden="true">&times;</span></button>
7        {{ Session::get('message') }}
8     </div>
9     @endif
10    <div class="container">
11       <div class="row">
12          <div class="main-slider">
13             <div class="slide-text">
14                <h1><img src="{{ asset('images/home/imagen_principal.png') }}" alt="
                  img_principal" width="75%" height="75%" style="margin-top: 1%;"></h1>
15                @if (Auth::guest())
16                <a href="{{ url('/login') }}" class="btn btn-common">INICIA SESSIÓ</a>
17                @else
18                <a href="{{ url('/searchTravel') }}" class="btn btn-common">CERCA VIATGE</a>
19                @endif
20             </div>
21             <img src="images/home/slider/mallorca_isla_opt.png" class="slider-hill" alt="
               slider image">
22             <img src="images/home/slider/coche_opt.png" class="slider-house" alt="slider image
               >
23             <img src="images/home/slider/sun.png" class="slider-sun" alt="slider image">
24             <img src="images/home/slider/birds1.png" class="slider-birds1" alt="slider image">
25             <!--<img src="images/home/slider/birds2.png" class="slider-birds2" alt="slider
```

Figura 3.4: Llamada al template app.blade en la vista del index

```
index2.blade.php
31 <!--/#home-slider-->
32 <section id="services">
33    <div align="center" class="wow scaleIn" data-wow-duration="500ms" data-wow-delay="300ms">
34       <div class="slide-text">
35          <h1>Propers viatges fins la UIB</h1>
36       </div>
37    </div>
38    <div class="container">
39       <div class="row">
40          @foreach ($viajesHacia as $travel)
41          <div class="col-sm-4 text-center padding wow fadeIn" data-wow-duration="1000ms" data-
            wow-delay="300ms">
42             <div class="single-service">
43                <div class="wow scaleIn" data-wow-duration="500ms" data-wow-delay="300ms">
44                   <img src="{{ asset('images/tipo0.png') }}" alt="">
45                </div>
46                <h2>Cap la UIB</h2>
47                <p>Origen: {{ $travel->origen }}</p>
48                <p>Dia: {{ $travel->FECHA }}</p>
49                <p>Hora: {{ $travel->HORA }}</p>
50                <a class="btn icon-btn btn-info" href="{{ route('reservaTravel.show',$travel->id
                  ) }}">
51                <span class="glyphicon btn-glyphicon glyphicon-plus img-circle text-success"></
                  span>Detalls
52                </a>
53             </div>
54          </div>
55          @endforeach
56       </div>
```

Figura 3.5: Tratamiento básico de datos

parte de la **vista** y los procesa.

El elemento que se encarga de gestionar la parte del servidor también serían los **controladores** ya que son los encargados de procesar las diferentes peticiones realizadas por el usuario. En la **Figura** 3.6, podemos ver cómo en la función se obtienen todos los parámetros de la petición y se hace una llamada al **modelo** para obtener estos datos. Una vez hemos obtenido respuesta por parte del **modelo**, el **controlador** se encarga de enviar estos datos hacia la vista tal y como podemos.

```php
public function showProfile($id)
{
    $user = User::findOrFail($id);
    return view('user.profile', ['user' => $user]);
}
```

Figura 3.6: Tratamiento de petición de una vista y llamada al modelo encargado de proporcionar la información

Por último, para obtener los datos o información requerida por parte del usuario, se acceden a estos a través de los **modelos**. En Laravel [17] se hace uso de un ORM llamado Eloquent. Un **ORM** es un Mapeo Objeto-Relacional por sus siglas en inglés (Object-Relational mapping), que es una forma de mapear los datos que se encuentran en la base de datos almacenados en un lenguaje de script SQL a objetos de PHP y viceversa. En la **Figura** 3.7 tenemos un ejemplo de un **modelo** que es el encargado de obtener los datos correspondientes de la tabla de los viajes creados por parte de los usuarios.

Viajes.php

```php
28      */
29      protected $fillable = [
30          'origen', 'destino', 'latitud', 'longitud', 'plazas_disponibles', 'descripcion','fecha', 'hora',
31          'user_id', 'id_vehiculo', 'tipo_viaje', 'estado_viaje', 'motivo_eliminacion'
32      ];
33
34
35      protected $dates = ['deleted_at'];
36
37
38      /*Funciones para el filtro de busqueda*/
39      public function scopeOrigen($query, $lugar)
40      {
41          return $query->where('origen', 'LIKE', '%'.$lugar.'%');
42      }
43
44      public function scopeFecha($query, $fecha)
45      {
46          //Sacamos el día de hoy y el día de hoy con 5 semanas adelantadas
47          $hoy = Carbon::today();
48          $adelantado = Carbon::today()->addWeeks(5);
49          if ($fecha != ""){
50              return $query->where('fecha', '=', $fecha);
51          }else{
52              /*Si no nos ponen fecha en el combo, sacamos los viajes de hoy, hasta 5 semanas adelante*/
53              return $query->whereBetween('fecha', [$hoy, $adelantado]);
54          }
55      }
```

Figura 3.7: Modelo *Viajes*, encargado del tratamiento de los datos correspondientes a los trayectos

3.1 Estructura del proyecto

A lo largo de este documento se ha introducido el concepto de comparteixUIB y cómo se iba a utilizar por parte de los usuarios. Una vez definidas las tecnologías utilizadas para el desarrollo del proyecto, se va a proceder a explicar cómo se ha desarrollado este proyecto desde un punto de vista de programación. Hay que recordar que comparteixUIB se pensó como una aplicación **cliente-servidor** siguiendo el patrón de programación *MVC*. Por esta razón, se ha decidido que la explicación sobre la implementación de la web se haga dividiendo cada uno de los elementos que componen el patrón.

- **Cliente**

Vistas

Las **vistas**, como ya se ha explicado anteriormente, son los elementos encargados de mostrar la información al estudiante de manera gráfica. De este modo, el estudiante podrá interactuar con la pantalla y volver a realizar peticiones. Por tanto, las **vistas** son los elementos encargados de trasformar los datos obtenidos del **controlador** y mostrarlos en formato **HTML**.

A la hora de generar un nuevo proyecto, *Laravel* crea una carpeta para guardar las diferentes **vistas** de nuestro proyecto. Esta carpeta es *comparteixUIB/resources/views*. En la **Figura** 3.8 podemos ver el esquema mencionado. Las **vistas** que se han ido generando como ficheros **PHP** y contienen código **HTML** mezclado con los diferentes *assets* como pueden ser los CSS, imágenes y código PHP.

Figura 3.8: Estructura de las vistas en el proyecto

En la carpeta *public* es donde se almacenan los diferentes *assets* que utiliza el proyecto. En la carpeta *resources/views* se encuentran todas las páginas generadas para la web. Además, esta carpeta está subdividida en varias carpetas donde se guardan los formularios, el formato de los emails, los distintos layouts, etc. En la **Figura** 3.9

podemos ver un ejemplo de como se realizan las peticiones en las **vistas** para solicitar la información. En esa petición, vemos cómo hacemos una petición al **controlador** *gestionCar* sobre el método *update* además de pasarle cómo parámetro el **id** del vehículo.

```
{!!Form::model($vehiculo,['route'=> ['gestionCar.update', $vehiculo->id], 'method'=>'
    PUT'])!!}

    <!--CON ESTO LLAMAMOS AL FORMULARIO DE CREACION/ACTUALIZACION DEL VEHICULO-->
    @include('autenticado.forms.carFormUpdate')

    <div align="center">
        <a class="btn btn-info" href="{{ URL::previous() }}">&laquo; Enrere</a>

        <button type="submit" class="btn icon-btn btn-info" ><span class="glyphicon btn
        -glyphicon glyphicon-share img-circle text-info"></span>Actualizar</button>


        <!--{!!Form::submit('Actualizar Perfil',['class'=>'glyphicon btn-glyphicon
        glyphicon-share img-circle text-info'])!!}-->
    </div>
{!!Form::close()!!}
```

Figura 3.9: Ejemplo de petición de datos al controlador desde la vista

Otra de las acciones que realizan las vistas es el tratar la información recibida por parte del **controlador**. Como se ha explicado anteriormente, *Laravel* hace uso de unas plantillas llamadas **Blade**. Esta librería nos permite realizar todo tipo de operaciones con los datos recibidos. Un ejemplo sencillo de la muestra de los datos por pantalla lo podemos ver en la **Figura** 3.10.

En la imagen podemos ver cómo se realiza un simple *foreach* para ir recorriendo una lista de viajes obtenidos por parte del controlador. Para poder acceder a un dato del objeto JSON como por ejemplo el destino del viaje, podemos realizar la siguiente instrucción {{ $travel->destino }}.

```
<div class="container">
   <div class="row">
      @foreach ($viajesDesde as $travel)
      <div class="col-sm-4 text-center padding wow fadeIn" data-wow-duration="1000ms" data-wow-delay="
      300ms">
         <div class="single-service">
            <div class="wow scaleIn" data-wow-duration="500ms" data-wow-delay="300ms">
               <img src="{{ asset('images/tipo1.png') }}" alt="">
            </div>
            <h2>Des de la UIB</h2>
            <p>Destí: {{ $travel->destino }}</p>
            <p>Dia: {{ $travel->FECHA }}</p>
            <p>Hora: {{ $travel->HORA }}</p>
            <a class="btn icon-btn btn-info" href="{{ route('reservaTravel.show',$travel->id) }}">
            <span class="glyphicon btn-glyphicon glyphicon-plus img-circle text-success"></span>Detalls
            </a>
         </div>
      </div>
      @endforeach
   </div>
</div>
```

Figura 3.10: Ejemplo de manipulación de datos obtenidos del controlador

Controladores

Las peticiones que se realizan por parte de las **vistas** y la gestión de los datos obtenidos por parte de los **modelos** y el envío de estos se ha realizado mediante el uso de los **controladores** que nos proporciona *Laravel*.

Debido a que el sistema gestiona diferentes tipos de datos, se han creado diferentes **controladores** que son los elementos encargados en *Laravel* de enviar los datos obtenidos hacia la **vista** que solicita un tipo de dato. Como se ha explicado anteriormente, el **controlador** es el elemento intermedio entre los **modelos** y las **vistas**. En cada uno de estos **controladores** se han creado las diferentes funciones para crear, modificar y/o consultar los datos.

El hecho de poder agilizar el tratamiento de datos y consultas es importante ya que la web estará gestionando muchas peticiones simultáneas, es por eso que se ha hecho uso de un **OR!** (**OR!**) llamado Eloquent que nos permite mapear los datos obtenidos de base de datos en objetos PHP.

La estructura para poder gestionar los diferentes **controladores** la gestiona *Laravel* creando una estructura de carpetas. Para este proyecto se han gestionado los **controladores** en la siguiente ruta *comparteixUIB/app/Http/Controllers/Autenticado* que podemos ver en la Figura 3.11.

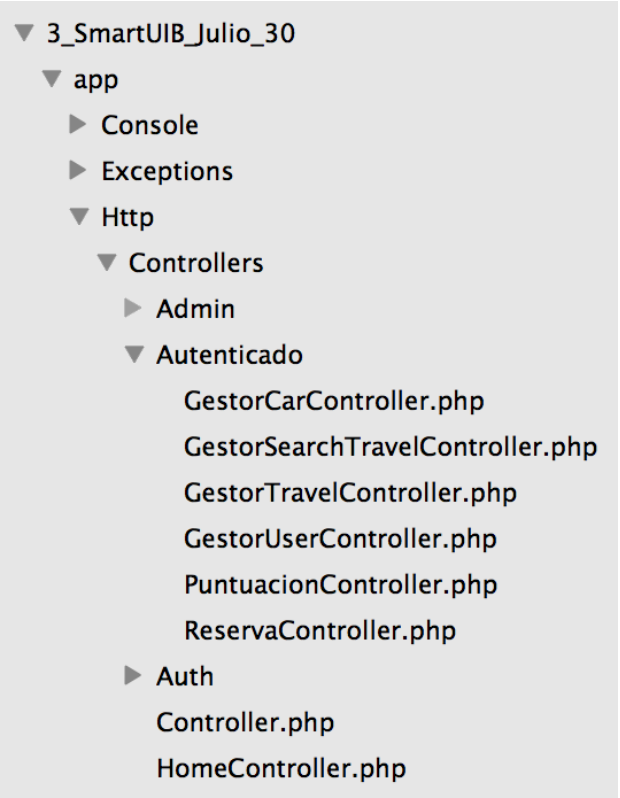

Figura 3.11: Estructura de controladores del proyecto

Un ejemplo sencillo para ver el funcionamiento del tratamiento de las peticiones y envío de datos por parte de un **controlador** sería la obtención de la información de un vehículo registrado por parte de un usuario. El usuario haría click sobre el vehículo del cual quiere ver los datos y automáticamente el sistema envía al **controlador** la matrícula del vehículo en cuestión. Un vez el **controlador** ha recibido está información, va a solicitar al **modelo** encargado de la gestión de vehículos toda la información que tenga sobre un vehículo que tenga esa matrícula. Una vez hemos obtenido respuesta por parte del **modelo**, el **controlador** se encarga de devolver la respuesta a la vista para que esta la pueda tratar. En la **Figura** 3.12 podemos ver el funcionamiento descrito anteriormente.

GestorCarController.php

```
138     public function edit($matricula)
139     {
140
141         /*Primero tenemos que encontrar el vehiculo que tenemos que modificar
142           y ese será el que tendremos que modificar.
143           Utilizaremos la matrícula que nos la pasan por parámetro para obtener todos
144           los datos del vehículo en cuestión.
145         */
146         $vehiculo = Vehiculos::where('matricula','=',$matricula)->get()->first();
147         $marca_modelo = DB::table('marca_modelo')
148             ->join('Vehiculos', 'marca_modelo.id', '=', 'Vehiculos.id_MM')
149             ->get();
150         return view("autenticado.vehiculo.edit", ['vehiculo' => $vehiculo],
151             ['marca_modelo' => $marca_modelo]);
152     }
153
```

Figura 3.12: Ejemplo de tratamiento y envío de datos por parte del controlador

- **Servidor**

Modelos

Debido a que la web necesita poder almacenar información sobre los viajes, vehículos, etc, se requiere el uso de una estructura que permitiera gestionar de manera eficaz y rápida el tratado de datos. Por eso se ha hecho uso de bases de datos mediante el gestor MySQL que es un sistema de gestión de bases de datos relacional. Asimismo, para poder manejar de manera mas eficaz este gestor, se ha hecho uso de una herramienta llamada phpMyAdmin. Cada tipo de información se guarda en diferentes tablas que a su vez están relacionadas entre sí. Al ser una base de datos **relacional**, las tablas se han creado siguiendo el modelo **relacional**. Una base de datos **relacional** tiene las siguientes características [18]:

1. Una base de datos se compone de varias tablas o relaciones.
2. No pueden existir dos tablas con el mismo nombre ni registro.
3. Cada tabla es a su vez un conjunto de campos (atributos) y registros (filas).
4. La relación entre una tabla padre y una tabla hija se lleva a cabo por medio de las claves primarias y claves foráneas (o ajenas).
5. Las claves primarias son la clave principal de un registro dentro de una tabla y éstas deben cumplir con la integridad de datos.
6. Las claves ajenas se colocan en la tabla hija, contienen el mismo valor que la clave primaria del registro padre; por medio de estas se hacen las formas relacionales.

En la **Figura** 3.13 podemos ver representado el modelo de datos para este proyecto.

A continuación, se explicarán las tablas creadas para este proyecto. La tabla **Usuario** contiene los datos de los estudiantes como por ejemplo el nombre, los apellidos, el

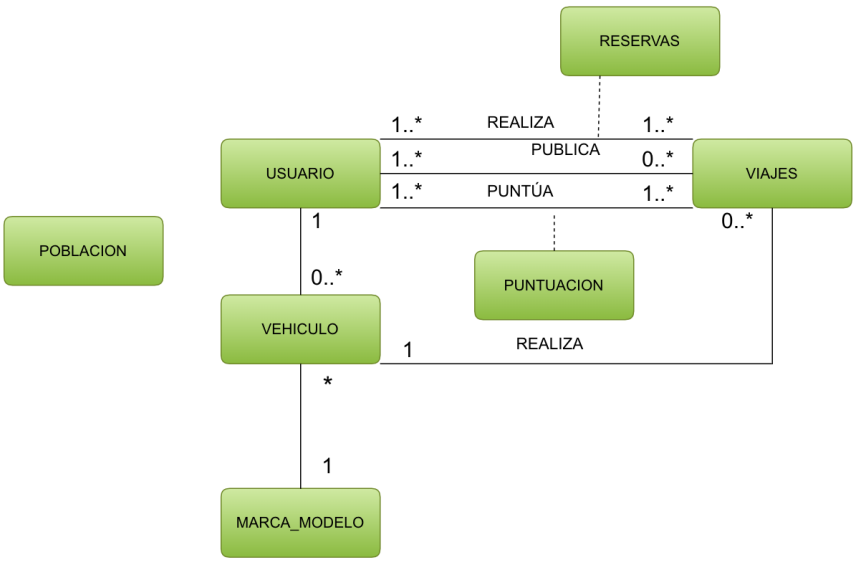

Figura 3.13: Representación del modelo de datos de comparteixUIB

Usuario	
ID (PK)	Este ID es la *primary key* de la tabla
Nombre	En esta columna guardaremos el nombre del estudiante
Apellidos	En esta columna guardaremos los apellidos del estudiante
Email	En esta columna guardaremos el correo del usuario
Teléfono	En esta columna guardaremos el teléfono del estudiante
Verificado	Columna que nos dice si el usuario ha sido validado a la hora de registrarse

Cuadro 3.1: Tabla **Usuario**

email, etc. En la *Tabla* 3.1 podemos observar los datos que se van a guardar de los estudiantes.

Cabe destacar que el **Email** con el que se registrarán los estudiantes debe ser un correo válido de la UIB, como por ejemplo *javi.gonzalez@uib.es*. La columna *Verificado* nos sirve para saber si un usuario está registrado en el sistema. Una vez nos registramos en la página, se enviará automáticamente un correo para poder validar la cuenta y así poder estar validados en la web.

Dado que es obligatorio asignar un vehículo a la hora de crear un viaje, ha sido necesario crear una tabla que contenga los datos de los diferentes vehículos de los estudiantes. La tabla **Vehículo** es la encargada para esta función. En la *Tabla* 3.2 podemos observar los datos que se van a guardar de los vehículos.

Como hemos visto en la *Figura* 3.13, un vehículo pertenece a un tipo de marca y modelo. Por otro lado, una marca y modelo puede estar en varios vehículos. La tabla *marca_modelo* es la encargarda de guardar la información sobre los diferentes modelos

Vehículo	
ID	Este ID es la *primary key* de la tabla
Matrícula	En esta columna guardaremos la matrícula del vehículo que es de tipo **UNIQUE**
Plazas	En esta columna guardaremos el número de plazas que tiene ese vehículo
Color	En esta columna guardaremos el color del vehículo
ID_MM (PK)	Este ID es la *primary key* de la tabla **Marca_modelo**
ID_user (PK)	Este ID es la *primary key* de la tabla **Usuario**

Cuadro 3.2: Tabla **Vehículo**

y marcas que existen hoy en día: Renault Clio, Peugeot 206, etc. En la tabla 3.3 podemos ver la estructura que tiene esta tabla para guardar los datos.

Marca_Modelo	
ID	Este ID es la *primary key* de la tabla
Matrícula	En esta columna guardaremos una marca y su modelo correspondiente

Cuadro 3.3: Tabla **Marca_Modelo**

A la hora de hacer una búsqueda de un viaje para una fecha y hora concreta es importante tener una lista con las poblaciones que hay en Mallorca para que se pueda seleccionar la zona que más convenga. Es por eso que se ha creado la tabla **Población** donde tendremos una lista con las poblaciones de Mallorca. En la *Tabla* 3.4 podemos ver la estructura que tendrá.

Poblacion	
ID	Este ID es la *primary key* de la tabla
Poblacion	En esta columna guardaremos cada una de las poblaciones de Mallorca

Cuadro 3.4: Tabla **Poblacion**

Una de las principales características de comparteixUIB es la de dar al usuario la posibilidad de crear sus propios viajes para que otros estudiantes puedan hacer reservar. Por eso ha sido necesario crear una tabla que contuviera toda la información necesaria para poder gestionar estos datos. La tabla creada se llama *Viaje* que contiene la información necesaria para poder gestionar los viajes que van creando y/o modificando los usuarios. En la *Tabla* 3.5 veremos la estructura de la tabla.

Cabe destacar que un viaje puede ser de dos tipos: que tenga el origen en la UIB o que el destino sea la UIB. La web no permite crear un viaje que no sea uno de esos dos tipos. Además, tenemos que tener en cuenta que el estado de un viaje puede ser: pendiente de realizar o realizado.

A medida que se vayan creando viajes, es necesario tener una estructura para almacenar las reservas que se van haciendo sobre estos trayectos. Por eso se ha creado una

Viajes	
ID	Este ID es la *primary key* de la tabla
Origen	En esta columna guardaremos el origen del viaje
Destino	En esta columna guardaremos el destino del viaje
Latitud	En esta columna guardaremos las coordenadas correspondientes a la latitud ya sea del origen o del destino en función de si es un viaje hacia la UIB o que el origen sea la UIB
Longitud	En esta columna guardaremos las coordenadas correspondientes a la longitud ya sea del origen o del destino en función de si es un viaje hacia la UIB o que el origen sea la UIB
Plazas_disponibles	En esta columna guardaremos las plazas de las que dispone el viaje para poder hacer reservas
Descripcion	En esta columna guardaremos una breve descripción del viaje
Fecha	En esta columna guardaremos la fecha en la que se realizará el viaje
Hora	En esta columna guardaremos la hora en la que se realizará el viaje
ID_Usuario	En esta columna guardaremos el ID del conductor del viaje. Este ID es la *primary key* de la tabla **Usuario**
ID_vehiculo	En esta columna guardaremos el ID del vehículi que realizará el viaje. Este ID es la *primary key* de la tabla **Vehiculo**
Tipo_viaje	En esta columna guardaremos el tipo de viaje al que pertenece el trayecto. Los valores serán 0 o 1
Estado_viaje	En esta columna guardaremos el estado en el que se encuentra un viaje. Los valores serán 0 o 1
Motivo_eliminacion	En esta columna guardaremos un breve motivo por el cual un viaje ha sido cancelado

Cuadro 3.5: Tabla **Viajes**

tabla llamada *Reserva* que nos permite ir almacenando información sobre una reserva de un viaje. En la *Tabla* 3.6 podemos ver la estructura de la tabla.

Reserva	
ID	Este ID es la *primary key* de la tabla
ID_Viaje	En esta columna guardaremos el viaje que por el que se ha hecho la reserva. Este ID es la *primary key* de la tabla **Viaje**
ID_Conductor	En esta columna guardaremos el conductor que va a llevar a cabo el viaje. Este ID es la *primary key* de la tabla **Usuario**
ID_Viajero	En esta columna guardaremos el usuario que ha hecho la reserva. Este ID es la *primary key* de la tabla **Usuario**

Cuadro 3.6: Tabla **Reserva**

Otra de las características que tiene comparteixUIB es la de poder poner una valoración al viaje que ha realizado. Las valoraciones son muy importantes de cara a la elección de un viaje ya que los estudiantes siempre querrán reservar un viaje en el

que el conductor tenga una alta puntuación. Se ha creado la tabla **Puntuacion** para poder almacenar todos estos datos. En la *Tabla* 3.7 podemos ver la estructura de la tabla.

Puntuacion	
ID	Este ID es la *primary key* de la tabla
Puntuacion	En esta columna guardaremos la puntuación que se le ha dado al viaje por parte de un usuario
ID_Viaje	En esta columna guardaremos el viaje que por el que se ha hecho la reserva. Este ID es la *primary key* de la tabla **Viaje**
IDD_Conductor	En esta columna guardaremos el conductor que va a llevar a cabo el viaje. Este ID es la *primary key* de la tabla **Usuario**
ID_Viajero	En esta columna guardaremos estudiante que ha hecho la reserva. Este ID es la *primary key* de la tabla **Usuario**

Cuadro 3.7: Tabla **Puntuacion**

El framework *Laravel* nos crea una estructura predefinida cuando creamos un nuevo proyecto. La gestión de todas las bases de datos se encuentran bajo una carpeta llamada */comparteixUIB/app*. En la Figura 3.14 podemos ver la estructura comentada anteriormente.

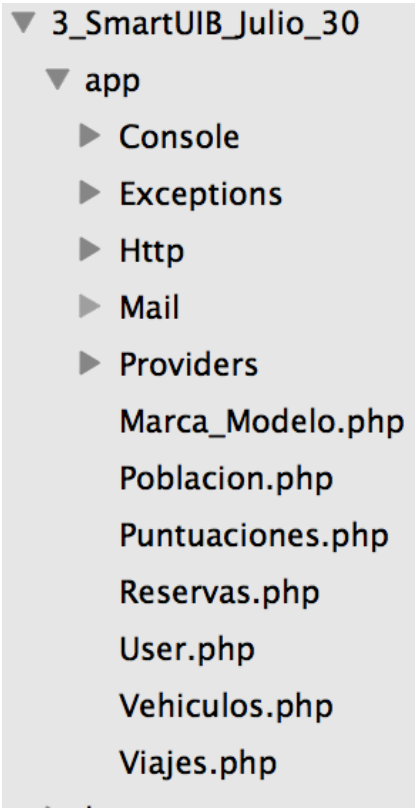

Figura 3.14: Estructura de los modelos en *Laravel*

En cada una de estas clases se defininen los atributos que tiene cada tabla así como las diferentes funciones para gestionar la diferente información de éstas. Un ejemplo de este tipo de clases en *Laravel* lo podemos vemos en la Figura 3.15.

Viajes.php

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Foundation\Auth\User as Authenticatable;

use Carbon\Carbon;

class Viajes extends Model
{
    use SoftDeletes;


    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'viajes';

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     *
     */
    protected $fillable = [
        'origen', 'destino', 'latitud', 'longitud', 'plazas_disponibles', 'descripcion','fecha', 'hora',
        'user_id', 'id_vehiculo', 'tipo_viaje', 'estado_viaje', 'motivo_eliminacion'
    ];


    protected $dates = ['deleted_at'];


    /*Funciones para el filtro de busqueda*/
    public function scopeOrigen($query, $lugar)
    {
        return $query->where('origen', 'LIKE', '%'.$lugar.'%');
    }
```

Figura 3.15: Ejemplo de un modelo en *Laravel*

Controlador

Una vez finalizada la creación de la base de datos y de los diferentes **modelos**, se ha llevado a cabo la creación de funciones que nos permitieran manipular los diferentes datos almacenados en la base de datos (o **Modelos**).

Para llevar a cabo la obtención y manipulación de los datos, se ha hecho uso de los **controladores** que además de enviar información hacia las vistas como se ha explicado antes, también tiene la función de comunicarse con los **modelos** para obtener la información solicitada por parte del usuario.

Además, para tratar de manera eficaz la información que nos proporcionan los **modelos** a través de la base de datos, se ha hecho uso de un **ORM** llamado Eloquent que, como bien se ha comentado anteriormente, es un *Mapeo Objeto-Relacional* que es una forma de mapear los datos que se encuentran en la base de datos almacenados en un lenguaje de script SQL a objetos de PHP y viceversa. Estas clases están implementadas en PHP y gracias a ellas se permite realizar un tipo de consultas llamadas *querys*.

A continuación, se detallaran los diferentes **controladores** que se han creado para este proyecto. Además, para cada uno de ellos se explicarán las funciones que tiene cada **controlador**.

En la **Tabla** 3.8 se explican las funciones encargadas de manipular los datos de los vehículos de los estudiantes.

GestorCarController	
index()	Esta función sirve para obtener los vehículos que tiene registrado un estudiante
create()	Esta función es la encargada de redirigirnos a la página para crear un nuevo vehículo. Además esta función se encarga de obtener todos los modelos de vehículos para que el estudiante pueda seleccionar uno
store()	Esta función es la encargada de obtener y tratar los datos para crear nuevos vehículos
edit()	Esta función es la encargada de obtener los datos del vehículo de un estudiante y nos redirige a la pantalla que nos muestra estos datos para editarlos
update()	Esta función es la encargada de actualizar los datos de un vehiculo
destroy()	Esta función es la encargada de eliminar un vehículo de un estudiante

Cuadro 3.8: Tabla **Controlador Vehículo**

En la **Tabla** 3.9 se explican las funciones necesarias para llevar a cabo las diferentes búsquedas de los viajes que han ido creado los estudiantes. Este controlador es uno de los más importantes ya que tiene que llevar a cabo unas de las características principa-

les de la web que es la búsqueda de viajes y el mostrado de dicha información.

Como información añadida, este controlador se encarga de cambiarle el estado a los viajes para que no aparezcan en los resultados del buscador. Esto se ha hecho así debido a que más del 70 % de las acciones que se llevarán a cabo en la web serán búsquedas.

GestorSearchTravelController	
index()	Esta función es la encargada de redirigr al usuario a la pantalla de selección de tipo de búsqueda; si hacia la UIB o desde la UIB
filtroTipo0()	Esta función es la encargada de obtener los viajes que se dirijan hacía la UIB según los criterios de búsqueda que haya solicitado un usuario
filtroTipo1()	Esta función es la encargada de obtener los viajes que salgan desde la UIB según los criterios de búsqueda que haya solicitado un usuario
show()	Esta función es la encargada de mostar la información sobre el viaje que el usuario ha seleccionada para ver sus detalles

Cuadro 3.9: Tabla **Controlador Buscador de Viajes**

En la **Tabla** 3.10 se explican las funciones que se han desarrollado para gestionar toda la información con respecto a los viajes, ya sea la creación, modificación y/o eliminación de un viaje.

A modo informativo, cuando un usuario elimina un viaje por un motivo justiciado, automáticamente se envía un correo a los estudiantes afectados de este viaje explicando que el viaje que tenían que realizar se ha cancelado.

En la **Tabla** 3.11 se explican las funciones para llevar a cabo el mantenimiento de los datos del usuario. Además, también se encarga de mostrarnos las diferentes opciones que tiene el usuario de hacer modificaciones en el sistema.

En la **Tabla** 3.12 se explican las funciones que se llevan a cabo para gestionar las puntuaciones de los viajes. El sistema escogido para dar puntuación a un viaje ha sido mediante estrellas, es decir, el usuario debe dar una puntuación en forma de estrellas siendo una estrella la puntuación más baja y cinco estrellas, la puntuación máxima.

GestorTravelController	
index()	Esta función es la encargada de redirigir al usuario a la pantalla de selección del tipo de viaje que quiere hacer una modificación
verviajesTipo0()	Esta función es la encargada de obtener una lista de viajes creadas por el usuario logueado y que sean viajes hacia la UIB
verviajesTipo1()	Esta función es la encargada de obtener una lista de viajes creadas por el usuario logueado y que sean viajes desde la UIB
create()	Esta función es la encargada de redirigir al usuario a la página de creación de nuevo viaje hacía la UIB. Además, esta función se encarga de obtener todos los vehículos que el usuario tiene creados y los envía a la pantalla de creación de nuevo viaje para que el usuario seleccione un vehículo.
create2()	Esta función es la encargada de redirigir al usuario a la página de creación de nuevo viaje desde la UIB. Además esta función se encarga de obtener todos los vehículos que el usuario tiene creados y los envia a la pantalla de creación de nuevo viaje para que el usuario seleccione un vehículo.
store()	Esta función es la encargada de obtener los datos del formulario de creación y de hacer el *insert* en base de datos de los viajes hacia la UIB
store2()	Esta función es la encargada de obtener los datos del formulario de creación y de hacer el *insert* en base de datos de los viajes desde la UIB
edit()	Esta función es la encargada de obtener los datos del viaje que el usuario ha seleccionado para hacer una modificación. Además, esta función se encarga de obtener todos los vehículos del usuario por si la modificación es sobre el vehículo
update()	Esta función se encarga de obtener los datos del formulario para hacer un *update* del viaje seleccionado
destroy()	Esta función se encarga de eliminar un viaje seleccionado por el usuario. Además, esta función se encarga de enviar un correo a los usuarios que estaban inscritos a este viaje para avisarles de que el viaje no se llevará a cabo

Cuadro 3.10: Tabla **Controlador Viajes**

En la **Tabla** 3.13 se explican las funciones necesarias que se han desarrollado para gestionar todo lo relacionado con las reservas. El usuario puede hacer tantas reservas como quiera pero si el cupo está completo, no podrá realizar la reserva. Además, este controlador sirve al estudiante para tener un control de las reservas que tiene que llevar a cabo o por el contrario eliminar una reserva de un viaje futuro.

GestorUserController	
index()	Esta función es la encargada de redirigir al usuario a la pantalla de modificaciones de usuario. Estas modificaciones pueden ser sobre reservas, viajes creados y/o vehículos registrados
edit()	Esta función es la encargada de redirigir al usuario a la pestaña de modificación de datos del usuario
update()	Esta función se encarga de la obtención de los datos del formulario y de hacer el correspondiente *update* en la base de datos con la nueva información de usuario

Cuadro 3.11: Tabla **Controlador Usuario**

PuntuacionController	
index()	Esta función es la encargada de obtener una lista de viajes realizados por el estudiante y que están pendientes de puntar
store()	Esta función es la encargada de crear un nuevo registro en base de datos con la puntuación que el usuario le ha dado al viaje

Cuadro 3.12: Tabla **Controlador Puntuaciones**

ReservaController	
index()	Esta función es la encargada de obtener una lista de reservas que el estudiante para realizar
crearReserva()	Esta función es la encargada de crear los diferentes registros en base de datos de las nuevas reservasa. Este método además hace una serie de comprobaciones para ver si es viable el hacer una reserva como por ejemplo si quedan plazas disponibles o si el viaje ya no está disponible
show()	Esta función es la encargada de mostrar los datos del viaje que el usuario ha seleccionada. Esta función puede ser llamada de dos lugares diferentes. Por un lado puede ser llamada desde el panel de reservas del usuario o en el buscador de viajes. Además esta función se encarga de comprobar que este usuario ya tiene una reserva de este viaje o no, en caso de que ya tenga una reserva en la página de reservas no le aparecerá el botón de reservar
quedan_plazas()	Esta función es la encargada de comprobar si la reserva por la que está consultando el usuario tiene plazas o no
tieneReserva()	Esta función, dado un usuario y un viaje, comprueba si el usuario que está solicitando información sobre el viaje tiene una reserva o no
destroy()	Esta función es la encargada de eliminar una reserva por parte de un usuario e incrementa el número de plazas del viaje relacionado

Cuadro 3.13: Tabla **Controlador Reservas**

CAPÍTULO 4

DISEÑO Y RESULTADOS

Todas las pruebas realizadas sobre comparteixUIB, se ha hecho en un entorno local. Se ha utilizado el programa **MAMP** [19], que nos ofrece una serie de características para poder desarrollar de manera satisfactoria este proyecto. Una de éstas es que incluye un servidor web HTTP Apache.

Dado que la web envía correos a los estudiantes en algunas ocasiones, se ha utilizado un Simple Mail Transfer Protocol (SMTP) para poder simular el envío de correos, en este caso, a los estudiantes de la UIB. El SMTP utilizado ha sido **mailtrap**.

4.1 Página principal de comparteixUIB

La primera vista que tiene un usuario al ingresar a comparteixUIB por primera vez lo podemos ver en la **Figura** 4.1. De la estructura de está página, y de las demás, se encarga el *layout* **app.blade** que es el fichero que contiene tanto la estructura general de todas las pestañas como los diferentes *assets* de la web como son el JS, CSS, etc. De manera más técnica, este *layout* contiene el *header* y el *footer* que se utilizará en toda la web. Por tanto, las demas pestañas hacen una llamada interna para utilizar dicho *layout*.

Para este caso de la página princial, en la **Figura** 4.2 podemos ver como la fichero que contiene los elementos de la página princial (**index.php**), hace una llamada a este *layout*.

Finalemente, podemos dividir la página principal en tres secciones: cabecera, cuerpo y pie.

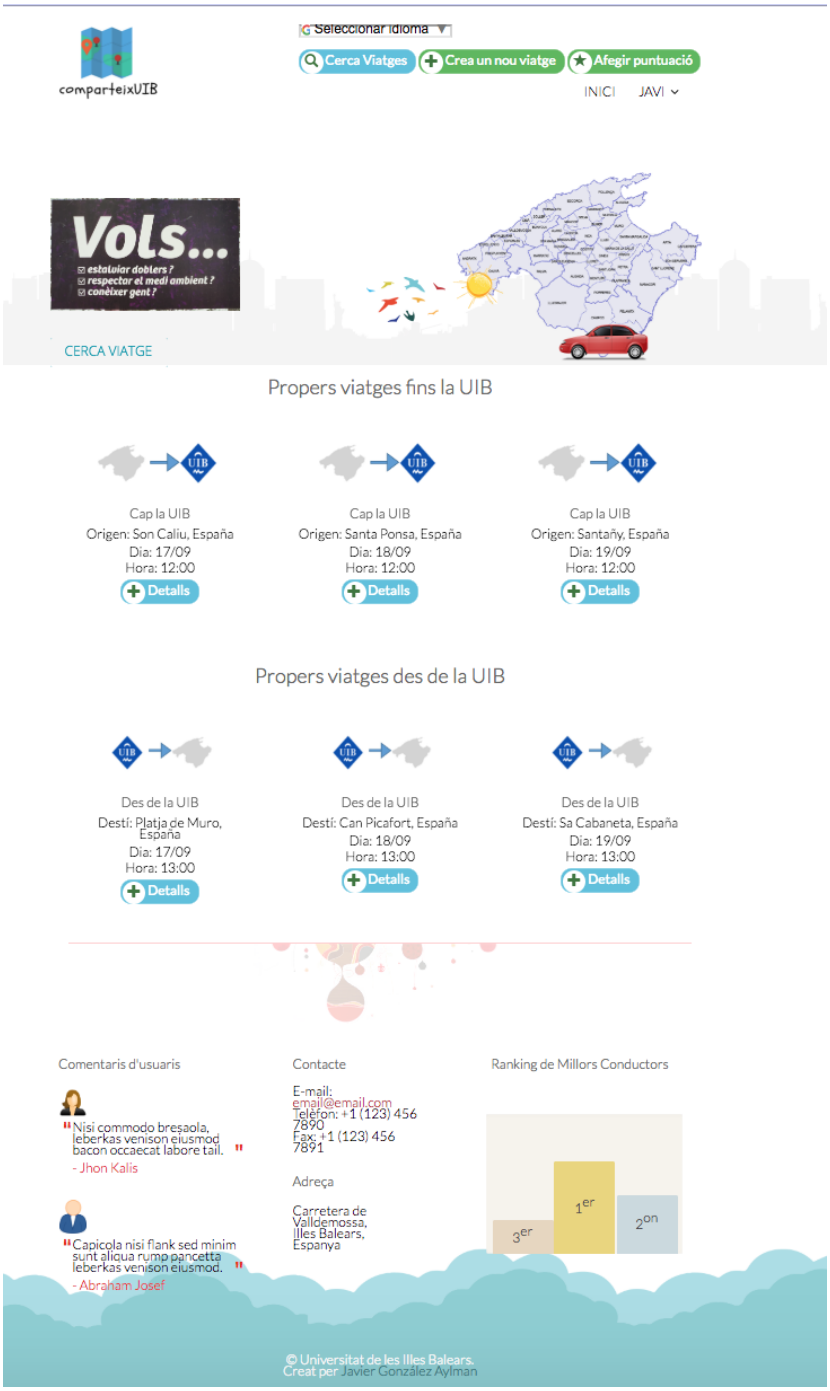

Figura 4.1: Página principal de comparteixUIB

4.1.1 Cabecera

En esta sección de la página tendríamos el logo de comparteixUIB. Si hacemos click sobre el logo, automáticamente redirige al usuario a la pantalla principal de la web. Otro elementos que tenemos en la cabecera es un selector de idiomas, que se ha

```
@extends('layouts.app')
@section('content')
<section id="home-slider">
   @if(Session::has('message'))
   <div class="alert alert-success alert-dismissible" role="alert">
      <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">
      &times;</span></button>
      {{ Session::get('message') }}
   </div>
   @endif
   <div class="container">
      <div class="row">
         <div class="main-slider">
            <div class="slide-text">
               <h1><img src="{{ asset('images/home/imagen_principal.png') }}" alt="img_principal" width="
               75%" height="75%" style="margin-top: 1%;"></h1>
               @if (Auth::guest())
               <a href="{{ url('/login') }}" class="btn btn-common">INICIA SESSIÓ</a>
               @else
               <a href="{{ url('/searchTravel') }}" class="btn btn-common">CERCA VIATGE</a>
               @endif
            </div>
            <img src="images/home/slider/mallorca_isla_opt.png" class="slider-hill" alt="slider image">
            <img src="images/home/slider/coche_opt.png" class="slider-house" alt="slider image">
            <img src="images/home/slider/sun.png" class="slider-sun" alt="slider image">
            <img src="images/home/slider/birds1.png" class="slider-birds1" alt="slider image">
            <!--<img src="images/home/slider/birds2.png" class="slider-birds2" alt="slider image">-->
```

Figura 4.2: Llamada al layout principal

implementado utilizando *Google Traslator* [20]. También contamos con una serie de botones para gestionar la creación de nuevas reservas, búsqueda de viajes y gestión de puntuaciones (**Figura** 4.3). Por último, esta cabecera cuenta con un menú desplegable en el que el usuario puede utilizarlo para: entrar al panel de gestión de usuario, edición de los datos de usuario o para cerrar la sesión.

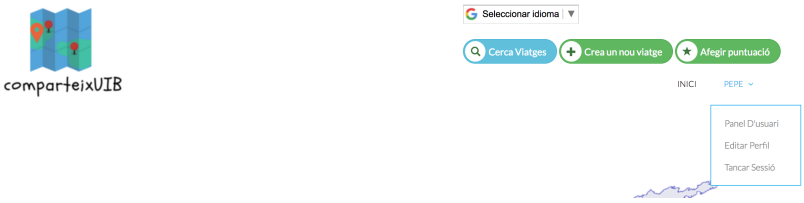

Figura 4.3: Cabecera (*Header* de la página inicial)

4.1.2 Cuerpo

En esta sección de la página se cargan dinámicamente los próximos tres viajes de cada tipo de trayecto: tres viajes hacía la UIB y tres viajes con origen en la UIB. Además, es posible acceder a los detalles de cada viaje haciendo click en el botón *Detalles* del viaje en cuestión (**Figura** 4.4). El **controlador** de obtener estos viajes es el **HomeController**.

4.1.3 Pie

En el pie de la página o *Footer*, nos encontramos con comentarios de conductores o de viajeros que el administrador del sistema puede ir cambiando modificando el código HTML del fichero **app.blade.php** que es el layout que general el *footer*. Además en esta sección encontramos un podium de mejores conductores (**Figura** 4.5). Gracias a las puntuaciones que los los usuarios, se hace una media y, los usuarios con las tres mejores medias, aparecen en el podium.

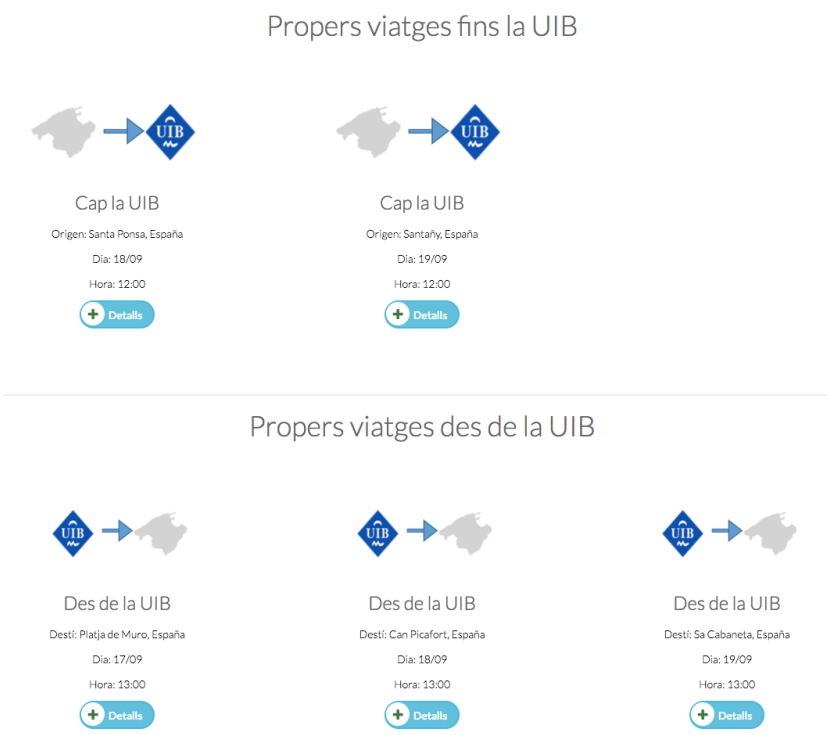

Figura 4.4: Viajes de la página principal

Figura 4.5: Pie de la página principal

4.2 Registro

Como bien se ha explicado anteriormente, los usuarios no registrados en la web, solo pueden realizar búsquedas sin posibilibidad de realizar reservas. En caso de que estos usuarios quieran realizar reservas y/o crear sus propios viajes, deben registrarse en la web. Para ello, se debe ir al apartado de registro de nuevos usuarios. Se puede acceder haciendo click en el botón *Registra't* (**Figura** 4.6).

Una vez se ha hecho click en el botón, la web, automáticamente redirige al usuario a un formulario de registro. La **Figura**4.7 muestra este formulario. Hay que recordar que los únicos usuarios, por el momento, que se pueden registrar en la web, son los

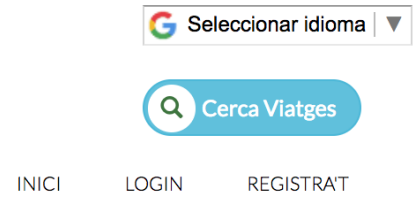

Figura 4.6: Botón registrar

estudiantes, profesores y otros trabajadores de la UIB, es por eso que en el selector de terminación de correo solo tendremos: *@uib.es* y *@uibdigital.uib.es*.

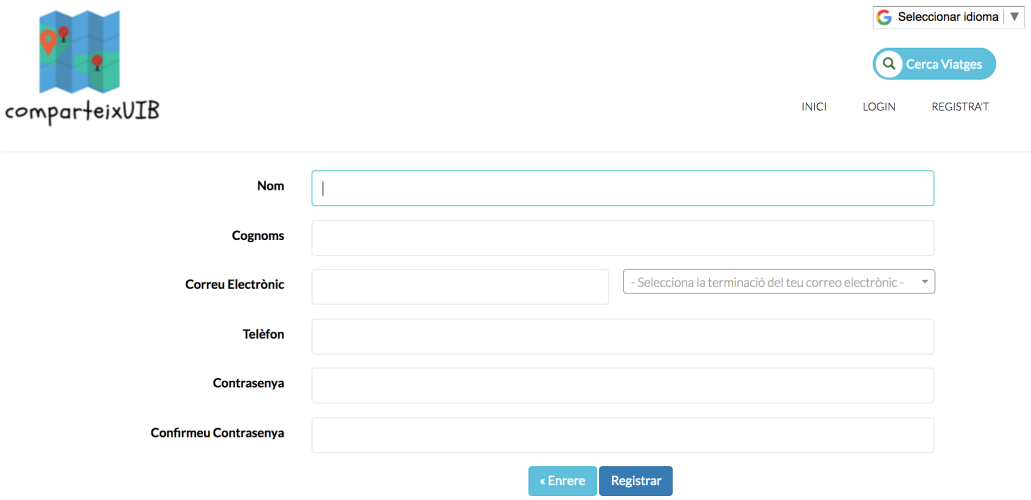

Figura 4.7: Botón registrar

Todos los campos son obligatorios, si el usuario, a la hora de hacer el registro, se olvide de rellenar uno, el sistema le avisará que algún campo está pendiente de rellenar.

Una vez rellenados todos los campos, se hace click en *Registrar*. Al realizar esta acción, esta pantalla (**vista**), se comunica con el **controlador** encargado del registro de usuarios (**RegisterController**) envíando los datos del formulario. El **controlador** procesa los datos y los verifica. Una vez hecho esto, hace una llamada a una clase llamada *Mail* que se encarga de enviar el correo de confirmación al usuario. Además, el **controlador** también tiene la tarea de avisar al usuario de que se le ha enviado un correo con un link para activar su cuenta y automáticamente redirige al usuario a la pantalla de *login*. Un ejemplo de correo que recibe el usuario puede verse en la **Figura** 4.9:

Al hacer click sobre el enlace, se redirige al usuario a la página de *login* y el sistema

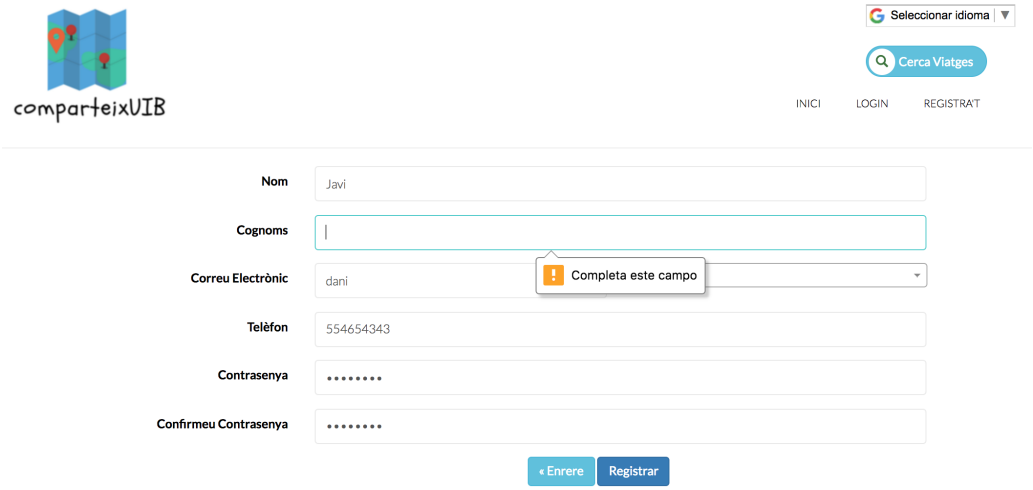

Figura 4.8: Faltan campos

Figura 4.9: Correo de confirmación

avisará mediante un mensaje por pantalla que la cuenta ha sido verificada (**Figura** 4.10). El **controlador** encargado de activar la cuenta del usuario y mostrar dicho mensaje es **RegisterController**.

4.3 Login

Una vez el usuario se ha registrado con éxito en comparteixUIB, debe hacer el login en la web para poder desbloquear nuevas funciones. En caso de que el usuario se haya equivocado al poner algún dato, la web avisará con un mensaje al usuario (**Figura** 4.11).

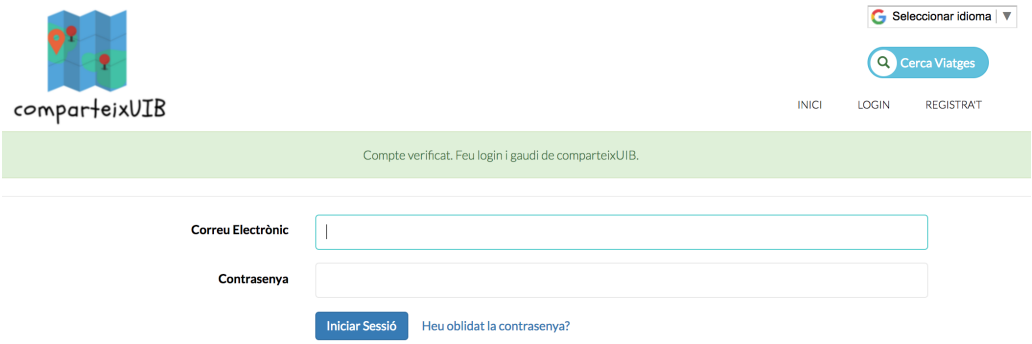

Figura 4.10: Cuenta verificada

El **controlador** que realiza la validación de usuario es **LoginController**.

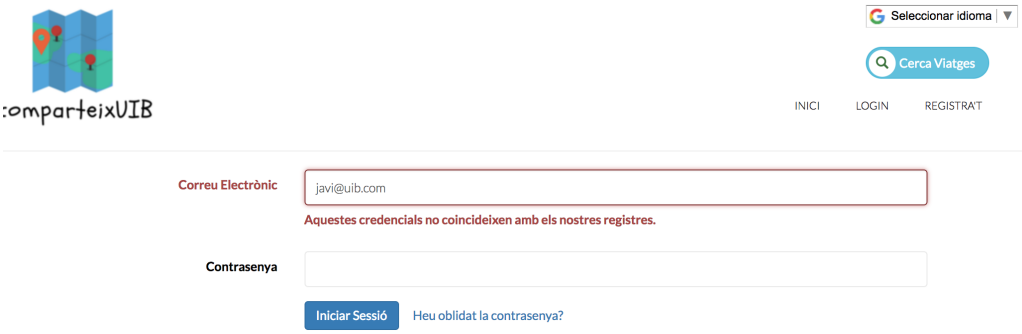

Figura 4.11: Login incorrecto

En caso de que el usuario haya introducido correctamente los datos e inicie sesión, el sistema le redirige a la página principal. Al haber iniciado sesión correctamente, se añade automáticamente una serie de botones en el menú de la página principal: *crear viajes* y *añadir puntuación*. En la **Figura** 4.12 se puede apreciar la diferencia entre un menú cuando no hay ningún usuario que haya iniciado sesión y el contrario, es decir, cuando hay un usuario que ha iniciado sesión.

Figura 4.12: Comparativa de menú

4.3.1 Recuperación de contraseña

En el caso de que un usuario haya olvidado sus credenciales de acceso, el sistema permite realizar una recuperación de la misma. Para ello el usaurio debe hacer click en *"Heu oblidat la contrasenya?"* en la ventana del *login* (**Figura 4.13**).

Figura 4.13: Botón *Heu oblidat la contrasenya?*

Al hacer click, se redirige al usuario a una pantalla donde tiene que indicar su correo y darle al botón enviar. Al darle al botón, se enviará un mensaje al usuario con el asunto *Reset Password* (**Figura** 4.14).

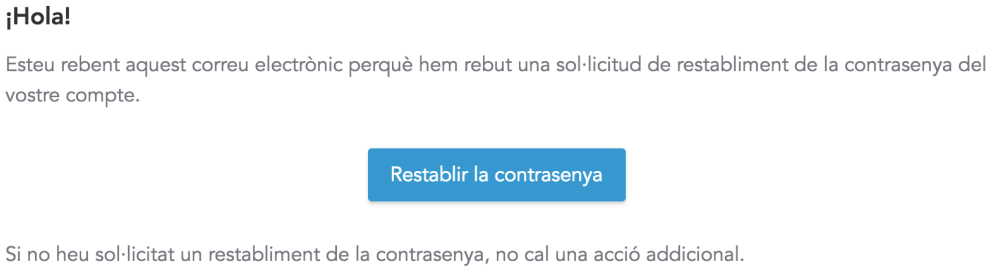

Figura 4.14: Correo restablecer contraseña

Una vez el usuario haya pulsado sobre el botón, se le redirige a una pantalla para crear una nueva contraseña (**Figura** 4.15). Una vez el usuario restablece su contraseña

se le redirige a la página home con la sesión de ususario iniciada.

Figura 4.15: Formulario restablecer contraseña

4.4 Panel de usuario

Una vez el usuario se ha registrado en el sistema, aparecerán nuevos botones en la cabecera de la página, así como un nuevo menú con el nombre del usuario. Cuando este menú se despliega, aparecén tres submenús que son: *Panel de usuario*, *Editar perfil* y *Cerrar sesión*.

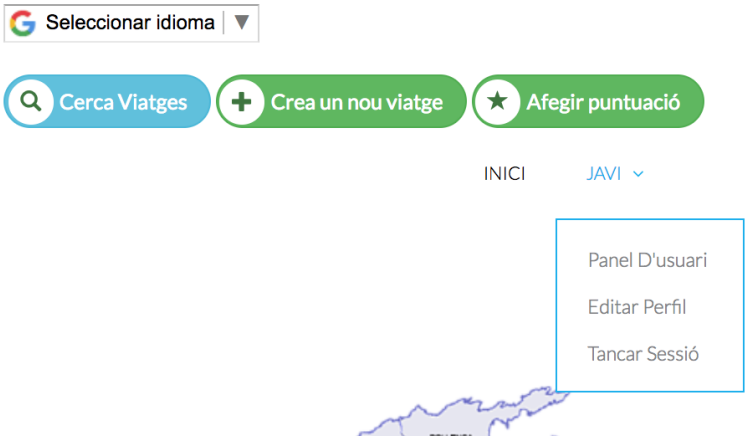

Figura 4.16: Menú y submenú de usuario

4.4.1 Editar Perfil

Si el usuario desea realizar alguna modificación con respecto a sus datos personales, ha de entrar al menú *Editar perfil*. Al hacer click, se redirige al usuario a un formulario para cambiar sus datos (**Figura** 4.17).

El **controlador** encargado de manipular los datos obtenidos del formulario y a su vez hacer un *update* en la base de datos con los nuevos datos es el **GestorUserController** mediante la función *update*.

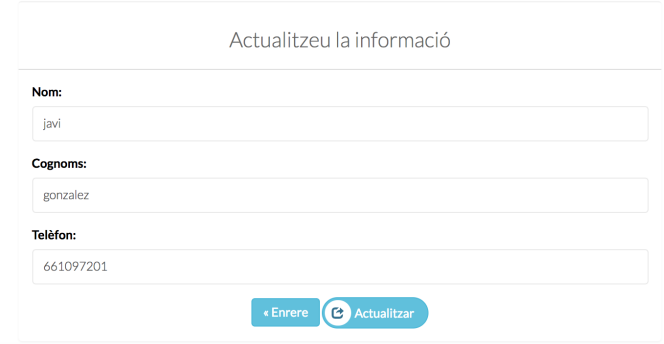

Figura 4.17: Formulario para cambiar datos personales

4.4.2 Panel de usuario

CompparteixUIB ofrece un cómodo sistema de gestión de elementos que tenga el usuario como pueden ser: reservas, vehículos y viajes. Para ello el usuario hará click sobre *Panel de ususario* y automáticamente se le redirigirá a un panel para administrar los elementos comentados anteriormente (**Figura** 4.18). Cada uno de los botones redirige al usuario a una ventana de gestión diferente. Cada una de las redirecciones es controlada por el controlador correspondiente: **GestorCarController**, **GestorTravelController**, **ReservaController**. Cada uno de estos controladores, se encarga de obtener el **ID** del usuario logueado y, mediante este, se obtiene la información correspondiente de cada tabla.

Figura 4.18: Panel de gestión

4.5 Gestión vehículos

Al entrar a la gestión de vehículos, el usuario se puede encontrar con dos escenarios: no tener ningún vehículo registrado o tener varios vehículos registrados. Esto se ve representado en la **Figura** 4.19.

En el primer caso, en la pantalla aparecería un mensaje comunicando al usuario que no tiene vehículos registrados. Además, le dará la opción de poder crear un vehículo. En el segundo caso, el usuario vería una lista de los vehículos que tiene registrados y podría acceder a cada uno de ellos pulsando el botón *Editar Vehiculo*.

Figura 4.19: Diferentes escensarios en la gestión de vehículos

4.5.1 Crear Vehículo

Si el usuario decide crear un nuevo vehículo, deberá hacer click, según si tiene o no tiene vehículos ya creados, en *Nuevo Vehículo* o *Registrar Vehículo*. Al pulsar sobre el botón, se redirigirá al usuario al formulario de creación de nuevo vehículo (**Figura** 4.20). El usuario tiene que rellenar todos los datos que aparecen, ya que si, por el contrario, no rellena algún dato, el sistema no le permitirá crear un nuevo vehículo y se mostrará un aviso en la cabecera. El selector de *marca y modelo* se carga con datos obtenidos de la tabla **marca_modelo**. Para facilitar la búsqueda de una marca en el selector, se ha creado un sistema de texto predicitivo que va generando resultados a medida que se va escribiendo sobre el combo.

El **controlador** encargado de la obtención de las opciones del selector de *marca y modelo*, tratamiento de los datos del formulario y su posterior inserción en la base de datos es **GestorCarController**.

Figura 4.20: Formulario nuevo vehículo

4.5.2 Editar Vehículo

El sistema también permite al usuario realizar modificaciones sobre un vehículo ya creado o eliminarlo del sistema. Para ello, el usuario debe hacer click sobre el botón *Editar Vehículo* del vehículo que se quiere modificar. El formulario mostrado es el de la **Figura** 4.21. El formulario solo permite modificar los siguientes datos: la matrícula, el color y el número de pasajeros.

Tanto si el usuario ha actualizado datos o eliminado el vehículo, será redirigido a la pestaña de gestión de vehículos y, en la cabecera, aparecerá un mensaje de la acción

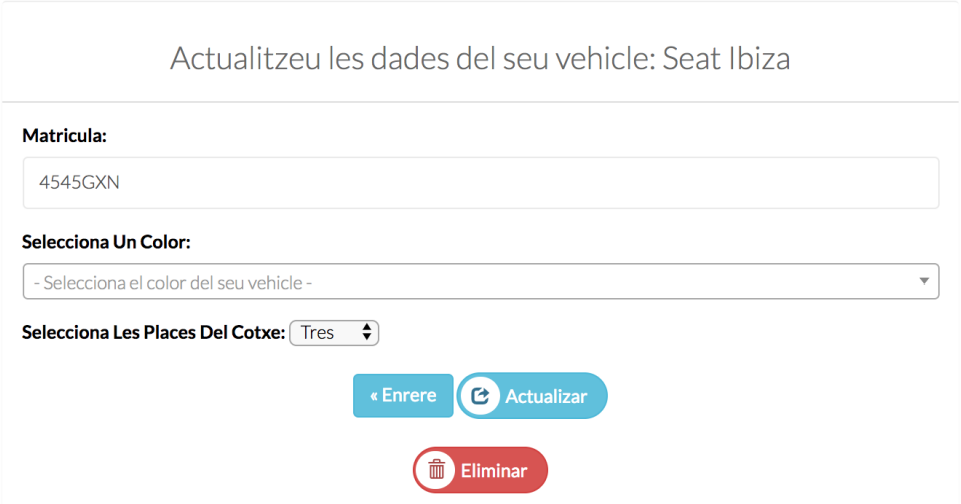

Figura 4.21: Formulario modificación de un vehículo

realizada (**Figura** 4.22).

Además, el sistema prohibe el intentar eliminar un vehículo que tenga reservas asociadas. En caso de tenerlas, se redirige al usuario a la pestaña de gestión de vehículos y, en la cabecera, aparecerá un mensaje de que no ha sido posible eliminar el vehículo ya que tenía reservas asociadas.

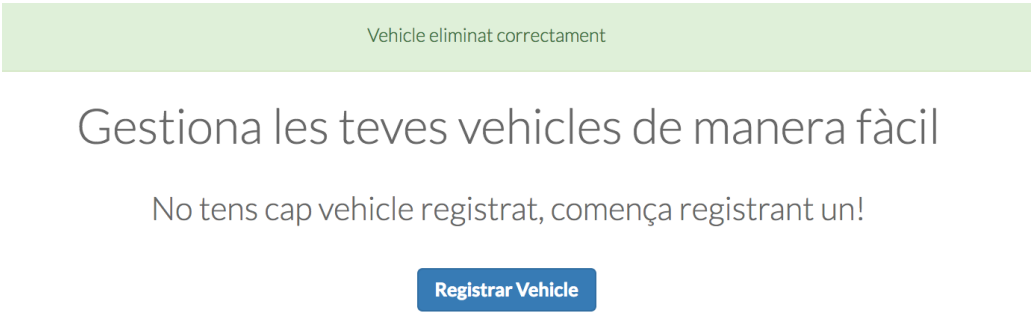

Figura 4.22: Mensaje eliminación de vehículo

4.6 Gestión de reservas

Si el usuario opta por gestionar sus reservas y, en ese momento, no tiene ninguna reserva hecha, le aparecerá la siguiente pantalla (**Figura** 4.23). El **contolador ReservaController**, es el que se encarga de comprobar si el usuario tiene reservas en ese momento o no. Para ello, hace una petición para comprobar si el usuario logueado tiene reservas activas, es decir, que no sean de fechas pasadas, sino reservas de viajes

que el usuario tenga que realizar.

En este escenario donde no existen reservas, el usuario podría optar por realizar una búsqueda de un viaje, para ello, haría click sobre el botón **Buscar Viajes** y automáticamente se le redirigirá a la pantalla de buscador de viajes.

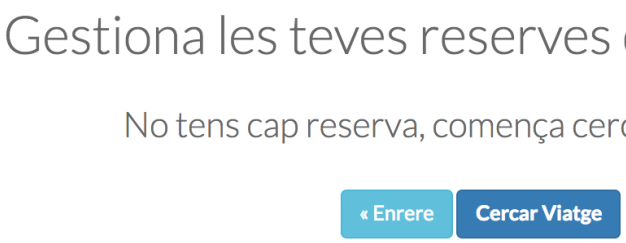

Figura 4.23: Pantalla cuando no hay reservas

El otro escensario posible que puede presentar esta pantalla es, que el usuario, tenga reservas activas. En este caso dichas reservas se visualizarán de la la manera que se muestra en la **Figura** 4.24. Para que pueda ver los detalles o cancelar la reserva, el usuario debe hacer click en *Detalles*. La función que realiza el **controlador** es obtener los detalles del viaje mediante el **ID** de viaje que nos ha proporcionado la ventana de gestión de viajes (**vista**).

Figura 4.24: Pantalla cuando hay reservas

La información que se le muestra al usuario sobre el viaje, lo podemos ver en la **Figura** 4.25. Los datos que se muestran en esta pestaña son: origen del viaje, mapa con trazado entre el origen y el destino, la fecha y hora, las características del vehículo y una breve descripción del viaje.

La realización del mapa se ha hecho utilizando la API de Google Maps [7]. Para el pintado del trazado entre el origen y el destino, se ha guardado, en base de datos, la latitud y longitud del origen o destino (según el tipo de viaje) y estos valores son recuperados por parte del controlador y enviados a la vista que se encarga de pintar esta ventana. Una vez tenemos los valores, una función de JS, los recupera y los utiliza para marcar un punto sobre el mapa. Una vez que tenemos los dos puntos sobre el mapa, se llama a una función para unir el origen y el destino.

4.6.1 Eliminación de reservas

En caso de que el usuario quiera eliminar una reserva, debe hacer click en **Eliminar Reserva**. Una vez se pulsa el botón, aparecerá una pop-up para confirmar que la acción que desea realizar el usuario es la correcta (**Figura** 4.26). Una vez el usuario acepte el borrado de la reserva, se le redirigirá a la pantalla de gestión de reservas.

4.7 Gestión viajes

Si el usuario opta por gestionar sus viajes, al hacer click en *Gestionar tus viajes*, se redigirá al usuario a la pantalla de gestión. En esta pestaña podrá seleccionar el tipo de viaje que desea modificar. En caso de que se intente gestionar un tipo de viaje del que no se tiene ninguno creado, se le mostrará al usuario un mensaje de que no tiene viajes creados. Además, se le da la oportunidad, al usuario, de crear un viaje (**Figura** 4.27).

En caso de que el usuario quisiera crear un viaje, debe escoger el tipo de viaje y hacer click en el botón **Crear viaje** correspondiente. El formulario que le aparecerá al usuario es como el de la **Figura** 4.28. Los datos a rellenar, en la figura de ejemplo, son: destino, fecha y hora, vehículo, número de plazas y descricpión del viaje.

Para poder mejorar la facilidad de uso de la creación de viajes, se ha creado un *combobox* predictivo para añadir el origen o el destino. Esto permite al usuario ir escribiendo una zona y en el combo irán apareciendo diferentes opciones que coincidan con lo que se está escribiendo. Esto se ha podido llevar a cabo gracias al uso de la API de Google Maps [7] (**Figura** 4.29).

Como se ha comentado anteriormente, cuando se guarda un nuevo viaje, se ha de guardar tanto la longitud como la latitud, según el tipo de viaje, del origen o destino. Para obtener estos datos, se ha creado una función **JS** que, dada una zona, obtiene la latitud y longitud de ésta (**Figua** 4.30).

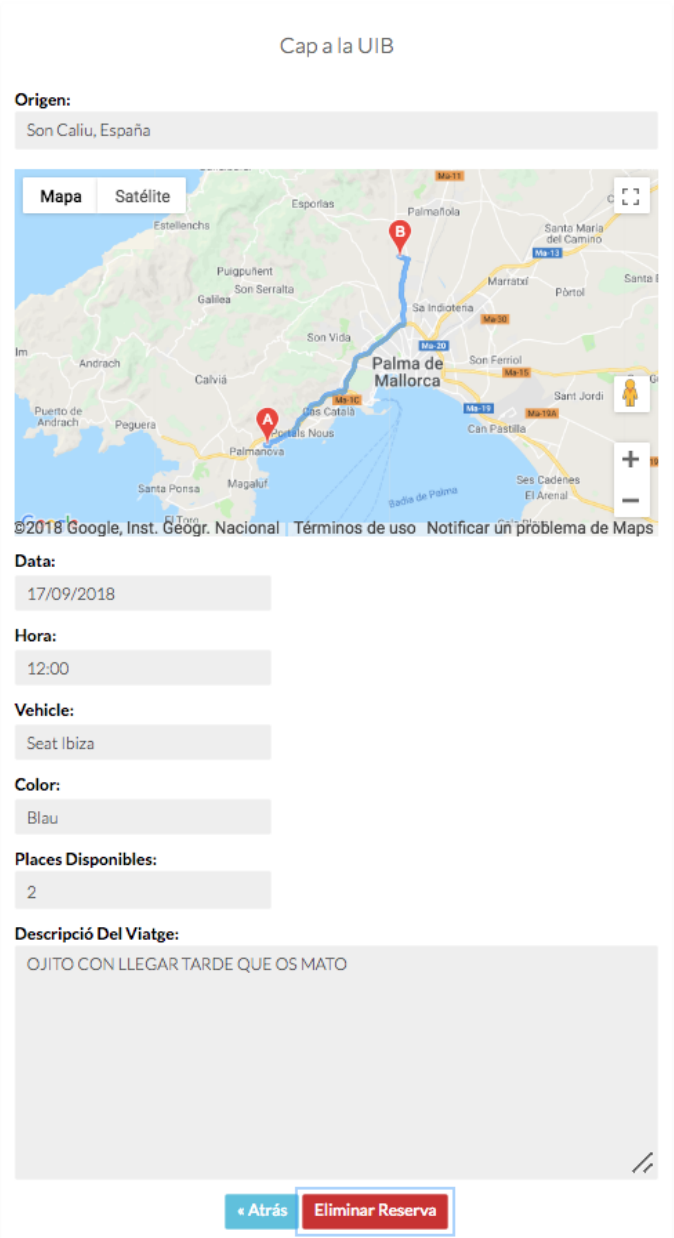

Figura 4.25: Datos de una reserva

El **controlador** encargado de gestionar la creación de un nuevo viaje es el **GestorTravelController**; más concretamente la función *store*.

4.7.1 Eliminación de viajes

En el caso de que un usuario necesite eliminar uno de sus viajes, puede entrar en los detalles del viaje y hacer click sobre **Eliminar viaje**, esto hará que se depliegue un

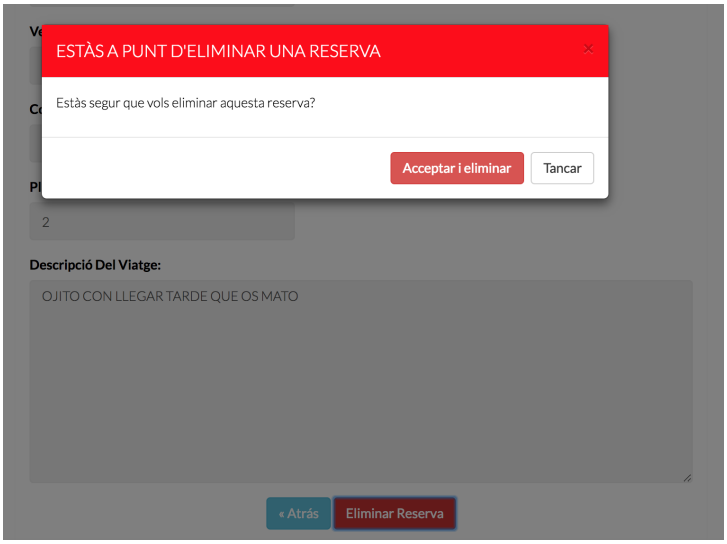

Figura 4.26: Pop-up confirmación de eliminación de reserva

Figura 4.27: No hay viajes creados

combo (**Figura** 4.31) para redactar el motivo por el cual ese viaje no se va a realizar. Este mensaje es el que se le enviará a cada uno de los usuarios que tenían que llevar a cabo ese viaje. Por tanto este motivo no puede quedar vacío. Si se intenta dejar vacío, el sistema avisa a usuario que es obligatorio poner un motivo de cancelación.

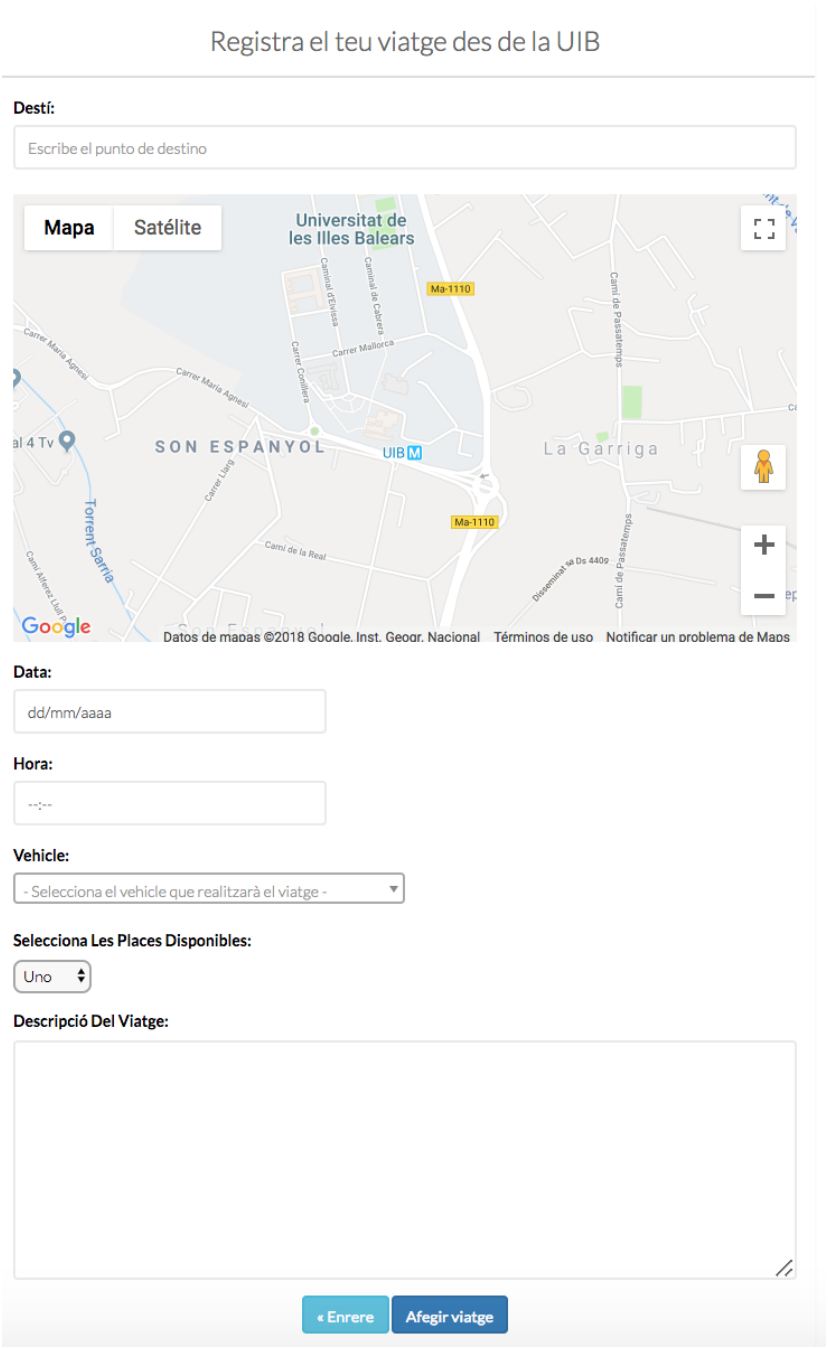

Figura 4.28: Formulario creación viaje

4.8 Buscador de viajes

A la hora de realizar búsquedas de viajes, es necesario que el sistema de contenga un diseño sencillo y de fácil uso para facilitar al usuario el uso de este buscador.

Una vez el usuario ha hecho click sobre *Buscar Viajes*, se encontrará con la ventana

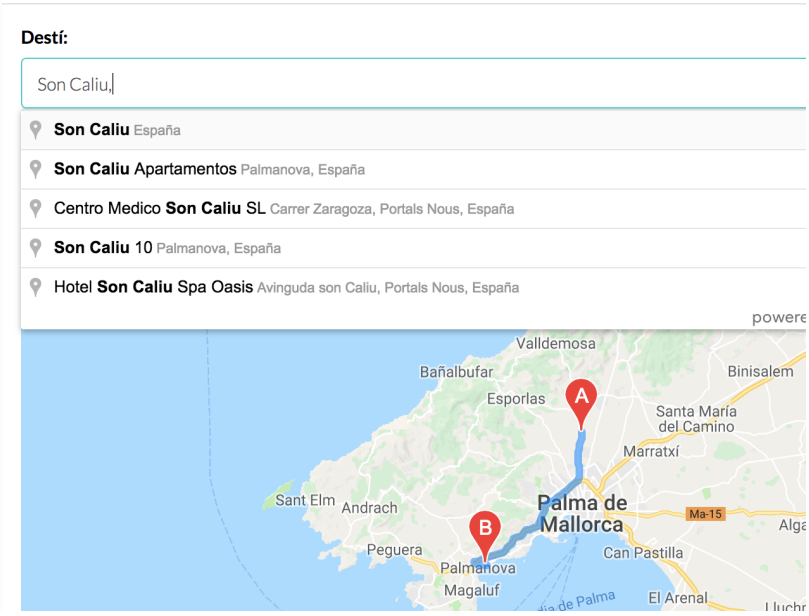

Figura 4.29: Combo predictivio para añadir la zona

```
/*ESTA FUNCIÓN LA USAREMOS PARA PASAR DE UNA DIRECCIÓN A LAT Y LNG (PARA LA BD)*/
function obtenerLatLng(){
    var geocoder = new google.maps.Geocoder();
    var address =  document.getElementById('destino').value;

    geocoder.geocode( { 'address': address}, function(results, status) {

        if (status == google.maps.GeocoderStatus.OK) {
            var latitude = results[0].geometry.location.lat();
            var longitude = results[0].geometry.location.lng();

            //SETEAMOS LOS DATOS
            $('#lat').val(latitude);
            $('#lng').val(longitude);
        }
    });

}
```

Figura 4.30: Función JS para el cálculo de latitud y longitud

que se muestra en la **Figura** 4.32.

El buscador de viajes se ha separado en dos tipos de búsqueda para simplicarlo, así el usuario puede decidir qué tipo de búsqueda lleva a cabo. Al hacer click sobre un tipo de búsqueda, se redirigirá al usuario al buscador de viajes del tipo seleccionado. Lo que primero visualiza el usuario al acceder al buscador, son unos filtros de búsqueda y un mapa donde hay marcados una serie de marcadores. Estos marcadores, pueden tener dos colores: verde y rojo.

El color verde representa que ese viaje tiene plazas disponibles para reservar. Por otra parte, el color rojo indica que ese viaje no es reservable ya que no tiene plazas disponibles (**Figura** 4.33).

Los filtros comentados anteriormente, sirven para que el usuario pueda filtrar de

Figura 4.31: Motivo de cancelación de viaje.

Figura 4.32: Buscador de viaje

diferentes maneras. Por un lado, tenemos un filtro de orígenes en el que el usuario selecciona una población del desplegable (**Figura** 4.34).

Otro de los filtros de los que dipone el usuario son, los de *Fecha y hora aproximada*. Aquí el usuario es libre de seleccionar una fecha válida, es decir, que sea del mismo día o posterior. Podemos ver un ejemplo de selección de estos filtros en la **Figura** 4.35.

Figura 4.33: Mapa de resultados

Figura 4.34: Selección población

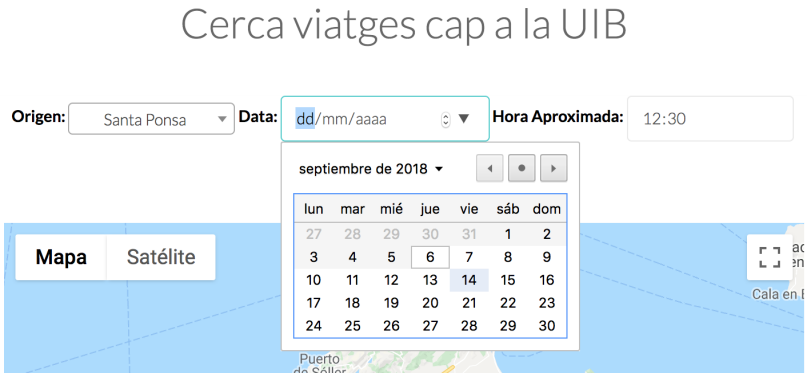

Figura 4.35: Selección fecha y hora

Debajo del mapa, tenemos un cuadro de resultados (**Figura** 4.36) donde se muestran los detalles de los viajes que aparecen en el mapa. En este cuadro aparecen los siguiente datos: nombre del conductor, valoración media del conductor, el origen del viaje, el destino del viaje, las plazas disponibles y operaciones disponibles.

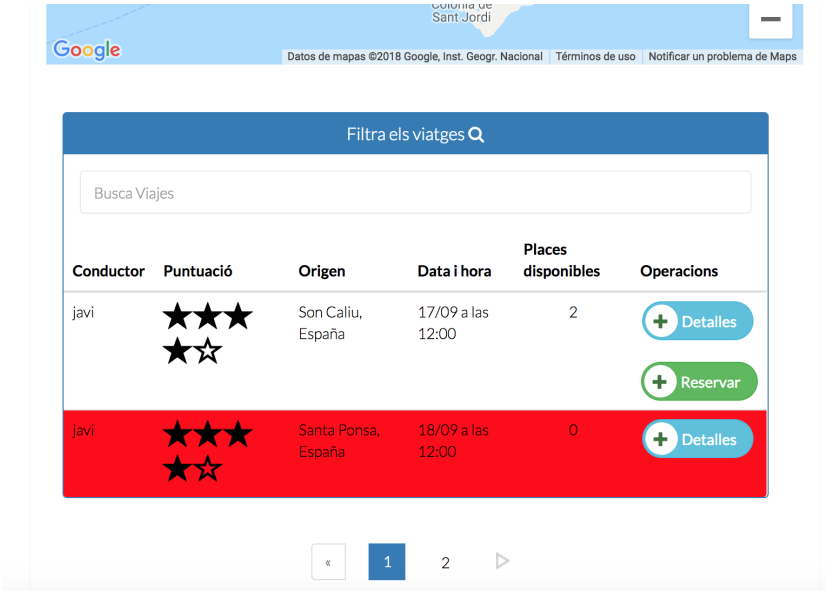

Figura 4.36: Cuadro de resultados

Los resultados de la tabla, pueden tener dos colores: blanco y rojo. El color blanco indica que el viaje tiene plazas disponibles para realizar una reserva. Este estado, permite al usuario al usuario tener dos acciones a realizar con este viaje, o bien va a ver sus detalles o hace una reserva. Por el contrario, el colo rojo, indica que ese viaje no tiene plazas disponibles y por eso solo se le permite ver los datos del viaje.

Finalmente, si el usuario desea realizar una reserva, tienes tres posibilidades de hacerlo:

1. Hacer click sobre un marcador verde del mapa de resultados. Esto hace que el usuario sea redirigido a la ventana de los datos de la reserva, donde podrá realizar la reserva.

2. Hacer click sobre el botón **Detalles** de la tabla de resultados. Esta acción hace que el usuario sea redirigido a la pantalla de los datos del viaje, donde podrá realizar la reserva.

3. Hacer click sobre el botón *Reservar* de la tabla de resultados. Esta acción permite hacer una reserva sin tener que ir a la ventana de detalles del viaje. Una vez la reserva se ha completado, aparecerá un mensaje en la cabecera de la página indicando al usuario que ha hecho una reserva de manera satisfactoria.

Si el usuario opta por el primer o segundo método, como se ha dicho anteriormente, será redirigido a la ventana que se muestra en la **Figura** 4.37.

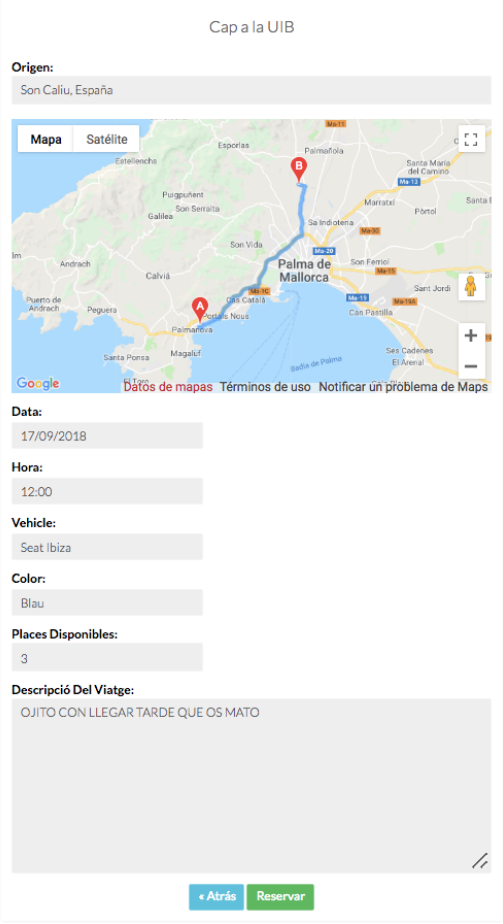

Figura 4.37: Ventana de detalles de viaje

Una vez el usuario ha decidido hacer una reserva y hace click en el botón *Reservar*, el **controlador GestorTravelController** se encarga de crear la nueva reserva en la base de datos y envía a la ventana un mensaje de confirmación de reserva como podemos ver en la **Figura**

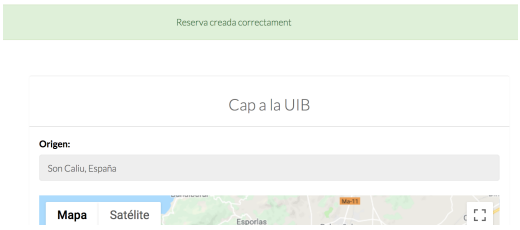

Figura 4.38: Mensaje confirmación de reserva

4.9 Puntuaciones

Una vez el usuario va finalizando sus viajes, puede puntar cada uno de ellos. Para ello, debe hacer click en el botón *Añadir puntuación* y automáticamente se le redigirá a la pantalla de gestión de puntuaciones.

En caso de que el usuario no tengo ningún viaje finalizado, la pantalla tendrá un listado vacío y, además, se le indicará que no tiene ningún viaje para puntar (**Figura** 4.39).

Figura 4.39: Pantalla puntuaciones sin ningún viaje

Por el contrario, si el usuario tiene viajes finalizados, estos aparecerán en la pantalla. El sistema de puntuación utilizado, como ya se ha dicho anteriormente, se basa en un sistema de estrellas, siendo una la menor puntuación y cinco la mayor puntuación (**Figura** 4.40).

Figura 4.40: Puntuar un viaje

Una vez el usuario ha escogido la puntuación del viaje, hará click en el botón *Puntuar* y automáticamente la web le notifica por pantalla que la puntuación se ha registrado correctamente (**Figura** 4.41).

Figura 4.41: Mensaje de confirmación de puntuación

CAPÍTULO 5

CONCLUSIONES

En este capítulo se resumirá el proyecto explicando brevemente sus características más importantes y cómo éstas se han cumplido. Además, se ofrecerán posibles mejoras de cara al futuro del proyecto. Finalmente, se hará una reflexión personal sobre lo que este proyecto ha aportado a la hora de llevar a cabo este (TFG), esto es, la experiencia de haber realizado este proyecto.

5.1 Resultados del proyecto

El objetivo de este TFG ha sido crear una plataforma web, conocida como comparteixUIB (Figura 5.1), con la idea principal de permitir a los estudiantes de la Universidad de las Islas Baleareas (UIB) desplazarse compartiendo vehículo entre sus miembros. Otro de los objetivos de este proyecto es que, con el lanzamiento de la web, se puedan llegar a minimizar en un futuro próximo las emisones de *CO2* que emiten los vehículos. Además, todos los objetivos de proyecto, comparteixUIB planteados en el apartado `2.1`, han sido implementados. Esto ha sido gracias a que se han conseguido desarrollar todos los requisitos funcionales de la web.

Para comparteixUIB se han desarrollado una serie de funcionalidades. La función principal de la web es permitir a los usuarios realizar búsquedas en tiempo real de trayectos que se vayan a llevar a cabo en un día y hora concretos. Para ello, se han creado dos buscadores diferentes: un buscador de viajes hacia la UIB y otro buscador con viajes que tengan como origen la UIB. Además, la web debe indicar al usuario que un viaje está completo o si, por el contrario, quedan plazas libres para reservar.

Otras de las funciones que ofrece la web es permitir a los usuarios añadir vehículos que realizarán viajes. Para ello, se ha creado una ventana donde el usuario añadirá diferentes datos relacionados con su vehículo como son la matrícula, el número de plazas, etc. También, se ha creado una tabla en base de datos para tener actualizados los datos con respecto a marcas y modelos de coches de actualidad.

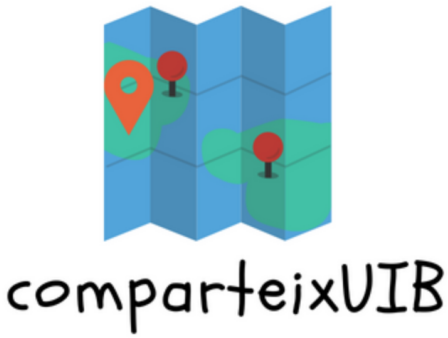

Figura 5.1: Logo de la aplicación web comparteixUIB

Por último, se ha desarrollado un sistema de puntuación para valorar los viajes ofrecidos por los usuarios basado en un número de estrellas: una estrella indica la menor puntuación y, cinco estrellas, la máxima puntuación. Se hará una media de todas las puntuaciones que tenga un usuario, que será visible en los resultados del buscador de viajes. Los tres mejores conductores aparecerán en el ranking que encontramos en el pie de la página principal de comparteixUIB.

Cabe destacar incluso que la web es accesible a personas que aún no forman parte de la UIB pero con ciertas limitaciones: este tipo de usuarios solo pueden relizar búsquedas de trayectos para ver los detalles sin poder reservar por motivos de seguridad.

5.2 Trabajo futuro

A lo largo del desarrollo han ido surgiendo ideas y nuevas funciones que podrían implementarse para comparteixUIB:

Mensajería a través de la web

La web no permite el uso de mensajes entre usuarios registrados. Esta funcionalidad sería útil para una mejor comunicación entre conductores y viajeros. Se podría implementar un sistema de mensajería en tiempo real o un sistema de mensajes por correo entre estudiantes.

Notificaciones de nuevos viajes

Actualmente, si un estudiante quiere saber si un viaje está disponible, puede entrar en la web y hacer una consulta en el buscador de viajes. Esto no sería necesario si se implementara un sistema de notificaciones en el que el estudiante guarde en él una serie de características de viaje y cuando el sistema detectara un viaje con las mismas características, este, de manera automática, mande un aviso al estudiante en forma de

mensaje a su correo.

Migración hacia plataformas de telefonía móvil

Actualmente, la web presenta una diseño denominado *Full Responsive* [1], por lo que es posible utilizar la web de manera cómoda en un dispositivo móvil. Aún así, se debe tener en cuenta esta migración, ya que con ella comparteixUIB podría llegar a ser utilizada por más usuarios. Se podría migrar la aplicación a sistemas *Android* e *iOS*. Además, el mantenimiento de la aplicación sería menos costoso.

5.3 Opinión personal

El desarrollo de comparteixUIB ha supuesto una gran experiencia.

Esta ha sido la primera aplicación web a gran escala que se ha desarrollado, ya que durante la carrera las aplicaciones web que se han realizado han sido de carácter más sencillo o, simplemente, se han hecho aplicaciones con interfaces de usuario.

Desde un punto de vista técnico, se han aplicado los conocimientos adquiridos durante la carrera como, por ejemplo, hacer uso de la técnica de *Brainstorming* para la obtención de requisitos. Para realizar la web, se han tenido que estudiar diferentes patrones de diseño web que se amoldaran mejor al proyecto. Esto ha permitido aprender un nuevo patrón de diseño de arquitectura de software. En este caso se ha aprendido el patrón **Modelo - Vista - Controlador** (MVC).

El hecho de poder trabajar conjuntamente con la UIB ha supuesto una motivación extra a la vez que ha sido un reto a nivel personal, ya que uno de los grandes objetivos del uso de la web es poder hacer que la contaminación de *CO2* disminuya y así poder contribuir a la reconstrucción de la capa de ozono. Finalmente, siempre es satisfactorio el poder realizar proyectos que ayuden a mejorar la calidad de vida de las personas y, en este caso, también la movilidad de los universitarios de la UIB.

BIBLIOGRAFÍA

[1] (2013) ¿qué es el diseño responsive? [Online]. Available: https://www.40defiebre.com/que-es/diseno-responsive/ (document), 5.2

[2] (2015) Consecuencias del cambio climático. [Online]. Available: https://ec.europa.eu/clima/change/consequences_es/ 1.1

[3] (2017) Contaminación por co2. [Online]. Available: http://www.elmundo.es/motor/2017/01/26/5889f3f7e2704e98418b4678.html 1.1

[4] (2017) Calculo de contaminación de co2. [Online]. Available: https://noticias.eltiempo.es/calculadora-emisiones-de-co2-cuanto-emite-coche/ 1.1

[5] N. Cognoms-autor, *PMBOOK*. PROJECT MANAGEMENT INSTITUTE, 2000. 2.1

[6] (2013) Técnica de obtención de requisitos. [Online]. Available: http://www.cge.es/portalcge/tecnologia/innovacion/4112brainstorming.aspx 2.1

[7] (2018) Google maps como referencia de los mapas web. [Online]. Available: https://mappinggis.com/2018/02/primeros-pasos-con-la-api-javascript-de-google-maps/ 6, 2.4.1, 4.6, 4.7

[8] (2008) Definición de html. [Online]. Available: http://www.acercadehtml.com/manual-html/que-es-html.html 2.4.1

[9] (201) Definición de css. [Online]. Available: http://librosweb.es/libro/css/capitulo_1.html 2.4.1

[10] (2018) Definición de javascript. [Online]. Available: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Que_es_JavaScript 2.4.1

[11] (2018) Qué es y para que sirve mysql. [Online]. Available: http://culturacion.com/que-es-y-para-que-sirve-mysql/ 2.4.1

[12] (2016) Laravel. [Online]. Available: https://ajgallego.gitbooks.io/laravel-5/ 2.4.1

[13] (2016) Modelo cliente-servidor. [Online]. Available: http://www.alegsa.com.ar/Dic/cliente_servidor.php 3

[14] (2014) Mvc. [Online]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html 3

[15] (2014) ¿qué es mvc? [Online]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html 3

[16] (2016) Motor de plantillas blade. [Online]. Available: https://richos.gitbooks.io/laravel-5/content/capitulos/chapter10.html 3

[17] (2016) Modelos orm en laravel. [Online]. Available: https://richos.gitbooks.io/laravel-5/content/capitulos/chapter7.html 3

[18] (2017) Caracterísitcas bd relacional. [Online]. Available: https://www.ibm.com/support/knowledgecenter/es/SSEPGG_8.2.0/com.ibm.db2.udb.doc/admin/c0004099.htm 3.1

[19] (2010) Mamp. [Online]. Available: http://oldblog.jesusyepes.com/administracion-de-sistemas/el-servidor-web-perfecto-macos-mamp/ 4

[20] (2013) Google traslador. [Online]. Available: https://www.milcursosgratis.com/como-poner-google-translation-en-mi-web-o-blog/ 4.1.1