

Aplicación web para buscar aparcamiento y compartir coche

Grado en Ingeniería Informática



Trabajo Fin de Grado

Autor:

Ouadi Chamit

Tutor/es:

Estela Saquete Boro

Septiembre 2017



Tabla de contenido

[INTRODUCCION 3](#_Toc488779152)

[RESUMEN 3](#_Toc488779153)

[MOTIVACION Y ESTUDIO DE MERCADO 4](#_Toc488779154)

[ BlaBlaCar 4](#_Toc488779155)

[ Uber 4](#_Toc488779156)

[ Amovens 4](#_Toc488779157)

[ Carpooling 4](#_Toc488779158)

[ Wazypark 5](#_Toc488779159)

[ Parkopedia 5](#_Toc488779160)

[ Aparca&Go 5](#_Toc488779161)

[HERRAMIENTAS, TECNOLOGIAS Y LENGUAJES DE PROGRAMACION EMPLEADOS 6](#_Toc488779162)

[phpMyAdmin 6](#_Toc488779163)

[Github 6](#_Toc488779164)

[Balsamiq 7](#_Toc488779165)

[Codigneiter 7](#_Toc488779166)

[Ventajas de utilizar un framework como CodeIgniter 7](#_Toc488779167)

[REQUESITOS FUNCIONALES Y NO FUNCIONALES 9](#_Toc488779168)

[Requisitos funcionales 9](#_Toc488779169)

[Requisitos no funcionales Los requisitos no funcionales serán: 9](#_Toc488779170)

[DISEÑO 10](#_Toc488779171)

[Mockups 10](#_Toc488779172)

[DISEÑO FINAL 18](#_Toc488779173)

[ARQUETECTURA 19](#_Toc488779174)

[IMPLEMENTACION 20](#_Toc488779175)

[- Front end 20](#_Toc488779176)

[- Back end 20](#_Toc488779177)

[Base de datos 21](#_Toc488779178)

[Esquema conceptual 21](#_Toc488779179)

[Modelo Relacional 22](#_Toc488779180)

[Diccionario de Datos 26](#_Toc488779181)

# INTRODUCCION

## RESUMEN

El proyecto consiste en la creación de una aplicación web, que ayuda a los usuarios a publicar anuncios para compartir coche o buscar aparcamiento en la universidad de Alicante, con el fin de facilitar encontrar un aparcamiento libre, o poder compartir coche para ir y venir de la universidad.

El proyecto nace con la idea de crear una aplicación web dedicada a todos los usuarios de la universidad de Alicante, en concreto a aquellos usuarios que no tienen medios para venir a la universidad o los que no tienen suficiente gasto de los viajes y ayudar a los propietarios de los coches de poder ahorrar gastos. También sirve para buscar aparcamiento de forma cómoda, ya que la universidad consta de varios parkings, cada uno se identifica por una referencia y está ubicado en una zona, por lo que resulta difícil, encontrar un hueco en una zona predeterminada, según el interés de los usuarios, y como resultado, perder tiempo, gastos de energía etc.

## MOTIVACION Y ESTUDIO DE MERCADO

Siendo alumno de la universidad, siento esa necesidad, y la de los demás usuarios, por lo que me motiva desarrollar esa aplicación y poder resolver uno de los problemas que afrontan los usuarios de la universidad que se encuentran a día hoy, y poder proveer una herramienta, para poner fin a esa necesidad o por lo menos paliar el problema.

Al analizar la competencia nos hemos dado cuenta de que no existe una aplicación en concreto con el mismo objetivo que gestiona tanto el aparcamiento como compartir coche a la vez específicamente para el sector universitario. No obstante, hay algunas aplicaciones encontradas que coinciden en algún apartado/sección, y que nos ha servido para mejorar nuestra idea.

Las aplicaciones buscadas han sido, por ejemplo:

* **BlaBlaCar**

Es una red social de viajes de largas y cortas distancias en coche compartido. Permite a los usuarios ponerse en contacto cuando quieren realizar un trayecto común y coinciden para hacerlo el mismo día, así como permitir a los conductores ahorrar el coste del litro de gasolina por cada pasajero.

Los usuarios comparten los gastos del viaje sin que el conductor tenga beneficios, para eso la red social recomienda a los usuarios por cada viaje la aportación de 0.06 euros por cada kilómetro cada uno, y limita la aportación máxima que pueden solicitar los conductores de tal manera que no se superen estos gastos.

* **Uber**

Uber es una aplicación para compartir viajes rápidos y fiables en cuestión de minutos, de día o de noche. No hace falta aparcar ni esperar taxis o autobuses.

Mediante la cual, comunicas tu ubicación, el destino, el tipo de vehículo y el tiempo de espera estimado. Tras el viaje, lo puedes pagar con tarjeta, efectivo o con tu teléfono móvil y el 20% se lo queda la propia empresa a modo de comisión

* **Amovens**

es similar a Uber en el concepto urbano de la aplicación, pero el objetivo de esta herramienta es encontrar a gente que realice el mismo trayecto que tú a diario y poder crear vínculos de transporte entre las personas, para ir al trabajo o a la universidad, sin que la empresa se lleve ningún tipo de comisión ni facilite modos de pago.

* **Carpooling**

es una aplicación similar a BlaBlaCar. Sirve exactamente para lo mismo, consiste en compartir nuestros viajes en auto con otras personas de forma cotidiana. El carpooling es una práctica popular en Estados Unidos y Europa, donde se realiza de manera organizada para lograr aumentar el número de viajes compartidos y que estos sean concretados con otras personas además de nuestros vecinos y amigos.

Sin embargo ofrece elementos como la elección de viajes solo entre mujeres o fijar un radio de búsqueda a tu alrededor para empezar el viaje.

* [**Wazypark**](http://www.wazypark.com/)

Se trata de una aplicación de inteligencia colectiva en la que mediante geolocalización del móvil en un mapa, podemos saber dónde hay una plaza libre en las calles de nuestra ciudad para aparcar. Wazypark mejora a medida que aumenta el número de usuarios, que deben registrarse ellos mismos tanto como a su vehículo. Cada usuario avisa desde su móvil cuando deja un hueco libre en un aparcamiento, de modo que otro usuario cercano nos pueda detectar y aparcar en la plaza. Disponible para Android e iOS.

* [**Parkopedia**](http://www.parkopedia.es/)

Tiene el mismo espíritu que Wazypark pero con elementos de Foursquare, en la que los usuarios dejan información de zonas de aparcamientos libres en plena calle o de aparcamientos públicos o privados, con los precios por hora o día de cada parking. Su interés reside en que hay información de numerosas ciudades. Tanto en Android como en iOS.

* [**Aparca&Go**](http://www.aparcandgo.com/)

Disponible solo para Android como aplicación, pero se puede usar desde la web. Se ha especializado en la gestión de aparcamientos mediante acuerdos por los que el usuario puede ganar un descuento en su ticket. La aplicación gestiona los pagos -hay que dejar los datos de una tarjeta-, por lo que las molestias son mínimas; además se puede reservar la plaza por adelantado. Un valor fuerte de esta aplicación es la creación de aparcamientos propios y alternativos a los de aeropuertos y estaciones de tren, pero con precios reducidos. De modo que ellos gestionan el transporte hasta el aeropuerto desde sus aparcamientos y de vuelta.

# HERRAMIENTAS, TECNOLOGIAS Y LENGUAJES DE PROGRAMACION EMPLEADOS

Para llevar a cabo el proyecto he utilizado diversas tecnologías y herramientas seleccionadas para cumplir con ciertos propósitos concretos.

Para la realización de la aplicación, he utilizado el lenguaje de programación PHP en el lado Backend y el framework bootstrap, HTML, CSS, JAVASCRIPT, JQUERY en lado de FrontEnd.

Por último, para realizar la base de datos he utilizado SQL. La herramienta utilizada para manejar la base de datos es phpMyAdmin.

## 

## phpMyAdmin

phpMyAdmin es la herramienta que he utilizado para el manejo de la base de datos MySQL está alojado en el servidor junto con la página web. La versión de MySQL es la 10.1.25-MariaDB - mariadb.org binary distribution y la versión de phpMyAdmin utilizada es la 4.7.0 y está alojada sobre un servidor Apache/2.4.26 (Win32) OpenSSL/1.0.2l PHP/7.1.7.

## Github

GitHub es una plataforma de **desarrollo colaborativo de software** para alojar proyectos utilizando el sistema de control de versiones [Git](http://es.wikipedia.org/wiki/Git).

Sirve para alojar un repositorio de código y brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto.

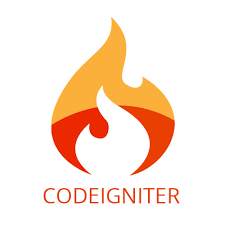
Además de todo eso, se ofrecen varias herramientas útiles para el trabajo en equipo

* Una wiki para el mantenimiento de las distintas versiones de las páginas.
* Un sistema de seguimiento de problemas que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
* Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
* Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio

## Balsamiq

He utilizado Balsamiq Mockups (versión escritorio). Balsamiq es la herramienta que he utilizado para realizar los mockups1 de las pantallas que va a tener mi aplicación. Con el objetivo claro de conseguir un enfoque sobre el diseño básico en etapas más tempranas, es decir, antes de comenzar a diseñar las pantallas ya sé cuál es el diseño que van a tener.

## Codigneiter

**CodeIgniter** es un entorno de desarrollo abierto que utiliza el **MVC** y permite crear webs dinámicas con [PHP](https://www.ecured.cu/PHP) . Su principal objetivo es ayudar a que los desarrolladores, puedan realizar proyectos mucho más rápido que creando toda la estructura desde cero, proveyendo un rico juego de librerías para tareas comúnmente necesarias, así como una interface simple y estructura lógica para acceder a esas librerías.

**Ventajas de utilizar un framework como CodeIgniter**

* Las páginas se procesan más rapido, el núcleo de CodeIgniter es bastante ligero.
* Es sencillo de instalar, basta con subir los archivos al ftp y tocar un archivo de configuración para definir el acceso a la bd.
* Reutilización de código, desarrollo ágil.
* Existe abundante documentación en la red.
* Facilidad de edición del código ya creado.
* Facilidad para crear nuevos módulos, páginas o funcionalidades.
* Acceso a librerías públicas y clases. Entre otras, hay librerías para el login, paginador, calendarios, fechas,….
* Estandarización del código. Fundamental cuando hay que tocar código hecho por otra persona o cuando trabaja más de una persona en un mismo proyecto.
* URLs amigables con SEO. Hoy en día creo que nadie duda de la importancia del posicionamiento web.
* Separación de la lógica y arquitectura de la web, el MVC.
* CodeIgniter es bastante menos rígido que otros frameworks. Define una manera de trabajar, pero podemos seguirla o no(esto puede convertirse en un inconveniente también)
* Cualquier servidor que soporte PHP+MySQL sirve para CodeIgniter.
* CodeIgniter se encuentra bajo una licencia open source, es código libre.
* CodeIgniter usa una versión modificada del Patrón de Base de Datos [Active Record](http://codeigniter.com/user_guide/database/active_record.html). Este patrón permite obtener, insertar y actualizar información in tu base de datos con mínima codificación. Permite queries más seguras, ya que los valores son escapadas automáticamente por el sistema.



## *PhpStorm*

es un IDE de programación desarrollado por [JetBrains](http://www.jetbrains.com/). Es uno de los entornos de programación más completos de la actualidad, permite editar código de varios lenguaje de programación aunque su nombre empieza por php.

algunas de las características principales.

* Proporciona un fácil autocompletado de código.
* Soporta el trabajo con PHP 5.5
* Sintaxis abreviada.
* Permite la gestión de proyectos fácilmente.

# REQUESITOS FUNCIONALES Y NO FUNCIONALES

## Requisitos funcionales

Los principales requisitos funcionales de la aplicación son:

* Cada usuario registrado es dado de alta, también existe el caso de dar de alta al coche si ese usuario es dueño de un coche en la cual comparte los viajes a/y aparca en la universidad.
* Cada usuario puede dar de alta a una notificación de compartir su coche cuando tiene plazas libres.
* Cada usuario puede buscar aparcamiento, poniendo la referencia del aparcamiento y ver las zonas libre.
* Cada usuario cuando sale del aparcamiento, debe actualizar la zona y ponerla como libre.
* Cuando usuario que se aparca, debe poner como ocupada la zona.
* Cada usuario puede comentar un coche cuando la comparte, así se lo permite a los demás usuarios conocer a las situaciones tanto del dueño como el coche .

## Requisitos no funcionales Los requisitos no funcionales serán:

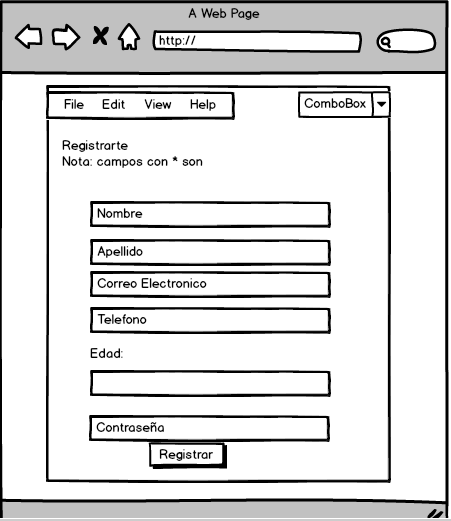
* Fiabilidad. La aplicación debe ser fiable en los datos que guarde. Si en algún momento no se puede acceder a internet, la aplicación deberá proporcionar otra forma alternativa de acceder a esos datos (guardados localmente).
* Usabilidad. La usabilidad en la aplicación será esencial ya que la mayoría de los usuarios que utilizarán la aplicación no estarán relacionados con el entorno informático. La tasa de errores cometidos por el usuario deberá ser menor. El sistema debe proporcionar mensajes de error que sean informativos y orientados a usuario final. El sistema debe poseer interfaces gráficas bien formadas.
* Eficiencia. El sistema debe ser capaz de procesar N transacciones por segundo. Toda funcionalidad del sistema y transacción de negocio debe responder al usuario en menor tiempo posible. El sistema debe ser capaz de operar adecuadamente con hasta N usuarios con sesiones concurrentes. Los datos modificados en la base de datos deben ser actualizados para todos los usuarios que acceden en menor tiempo.
* Dependibilidad. El sistema debe tener una disponibilidad del 99,99% de las veces en que un usuario intente accederlo. El tiempo para iniciar o reiniciar el sistema no podrá ser mayor. La tasa de tiempos de falla del sistema no podrá ser mayor. El promedio de duración de fallas no podrá ser mayor. La probabilidad de falla del Sistema no podrá ser mayor.

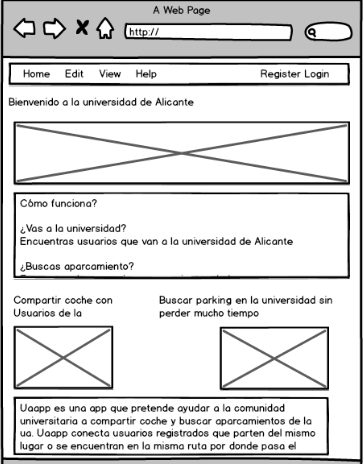
# DISEÑO

## Mockups

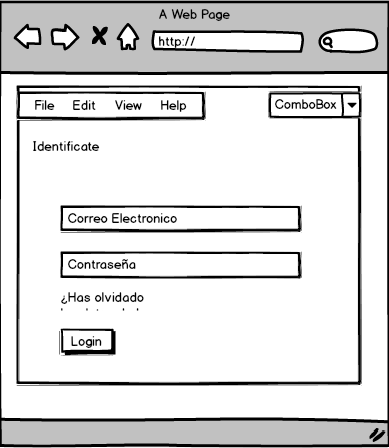
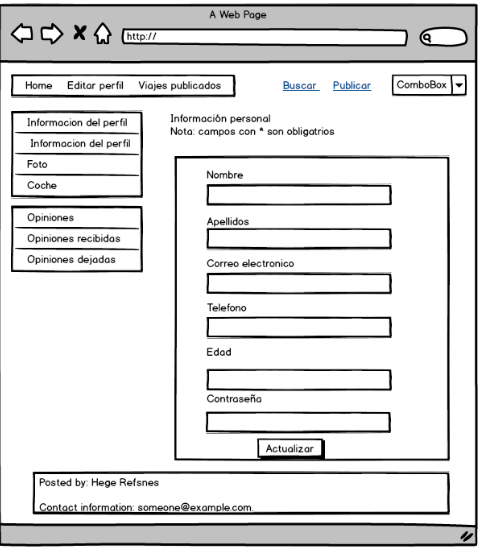
Antes de comenzar a implementar el proyecto, he creado los mockups de la aplicación. Consiguiendo así definir el diseño básico que tendrán las pantallas en fases tempranas del desarrollo .

Por ejemplo vamos a ver el mockup de la pantallas de la aplicación.

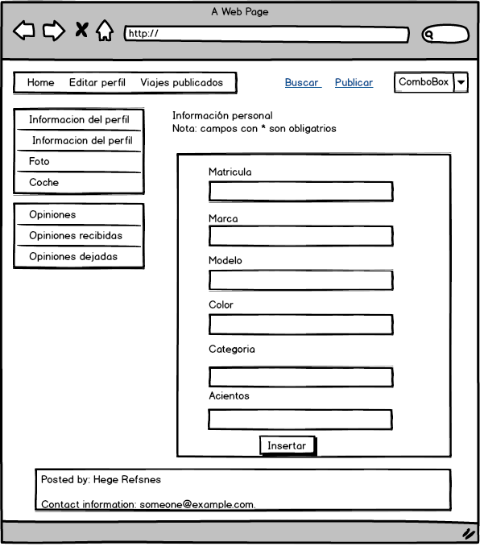
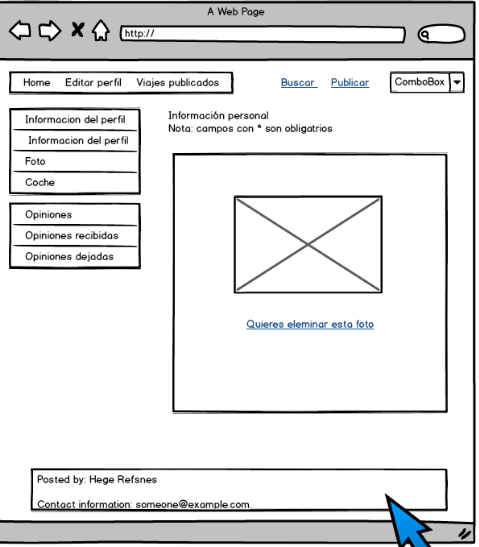


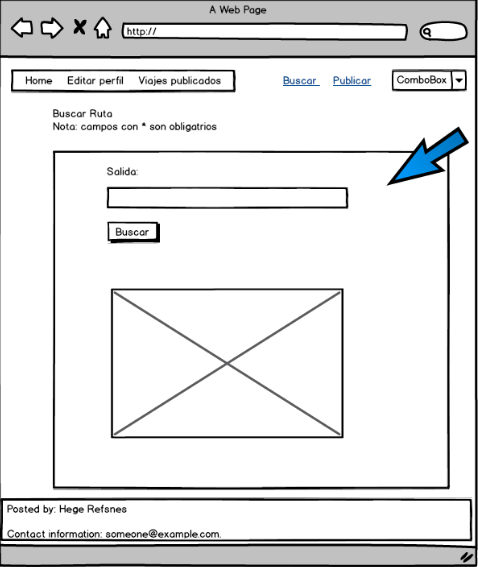


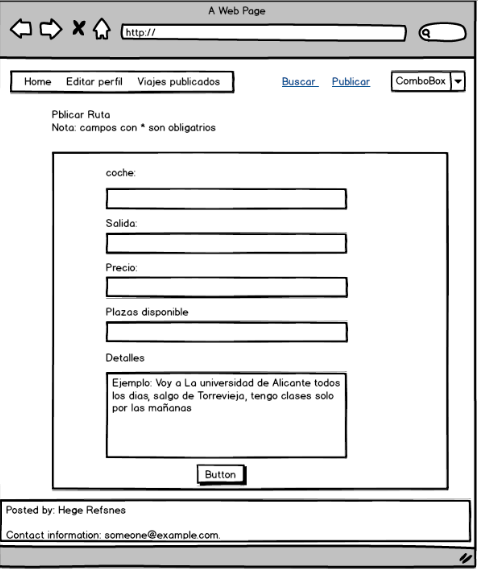
Pantalla Login/Actualizar datos usuario



Pantalla Coche/Foto usuario



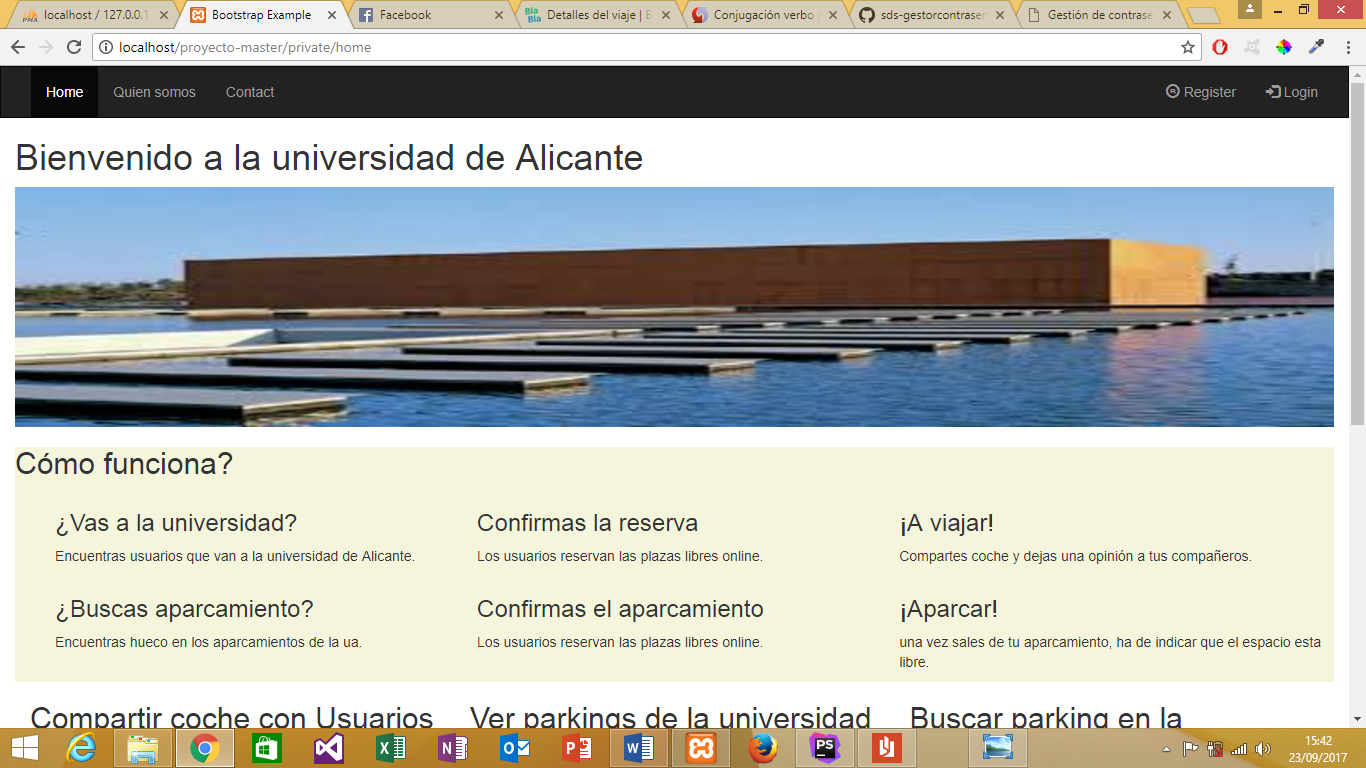
Pantalla Publicar anuncio/Buscar coche para compartir



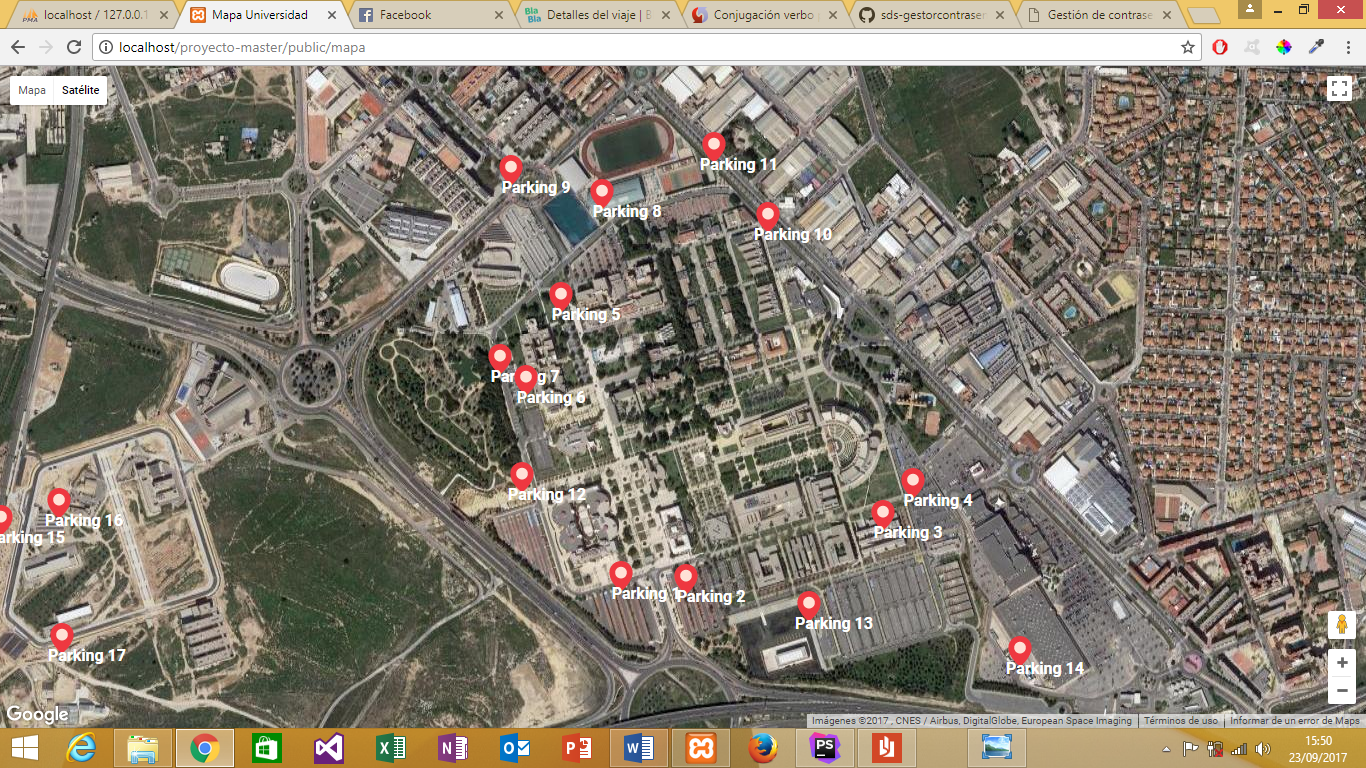
## DISEÑO FINAL

Pantalla principal de la aplicación

En la pantalla principal, se encuentra la formación que provee la aplicación, Como las funcionalidades, el registro y el login, Además cualquier usuario tiene la opción de ver los parkings de la ua, sin que este, este dado de alta.







Pantalla de Registro de Usuario

Esta pantalla permite a los usuarios crear una nueva cuenta. Como se puede observar,

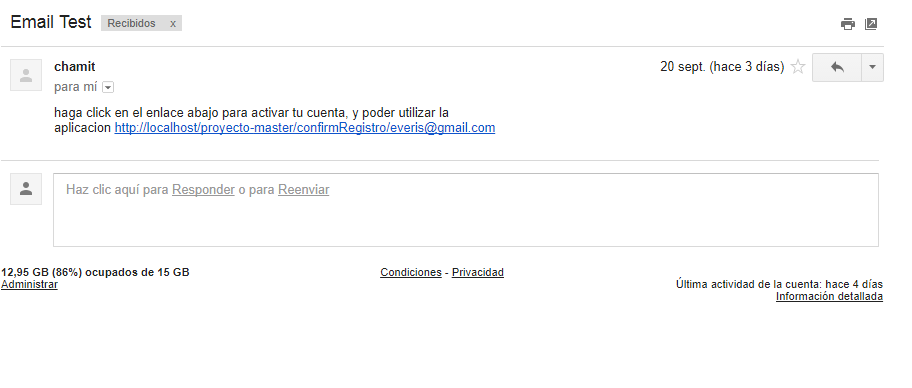
Consta de varios campos todos son obligatorios, indicados con el asterisco rojo, controlados con la propiedad de html required, la cual impide hacer un submit si un campo esta vacío.

Además, se comprueba el campo correo con Ajax para que no sea duplicado en la base de datos, ya que es un dato único, También el campo teléfono y contraseña son controlados con Jquery para que sean datos validos.

Una vez se comprueba todo que es correcto, cuando el controller registro recibe los datos de parte del cliente, se mantiene todos los datos en plano, menos la contraseña que se almacena de forma cifrada, para realizar el cifrado, se Utiliza SHA-512 como hash sobre la contraseña, se crea una salt aleatoria. Esta salt servirá para “mezclarla” con el hash creado y realizar un encriptado con “crypt”, y por ultimo realizar el almacenamiento en la base de datos.



Cuando se realiza el registro de forma correcta, se le envía al usuario un correo electrónico, para confirmar el registro pulsando sobre el enlace enviado, en caso contrario, no puede acceder al sistema.

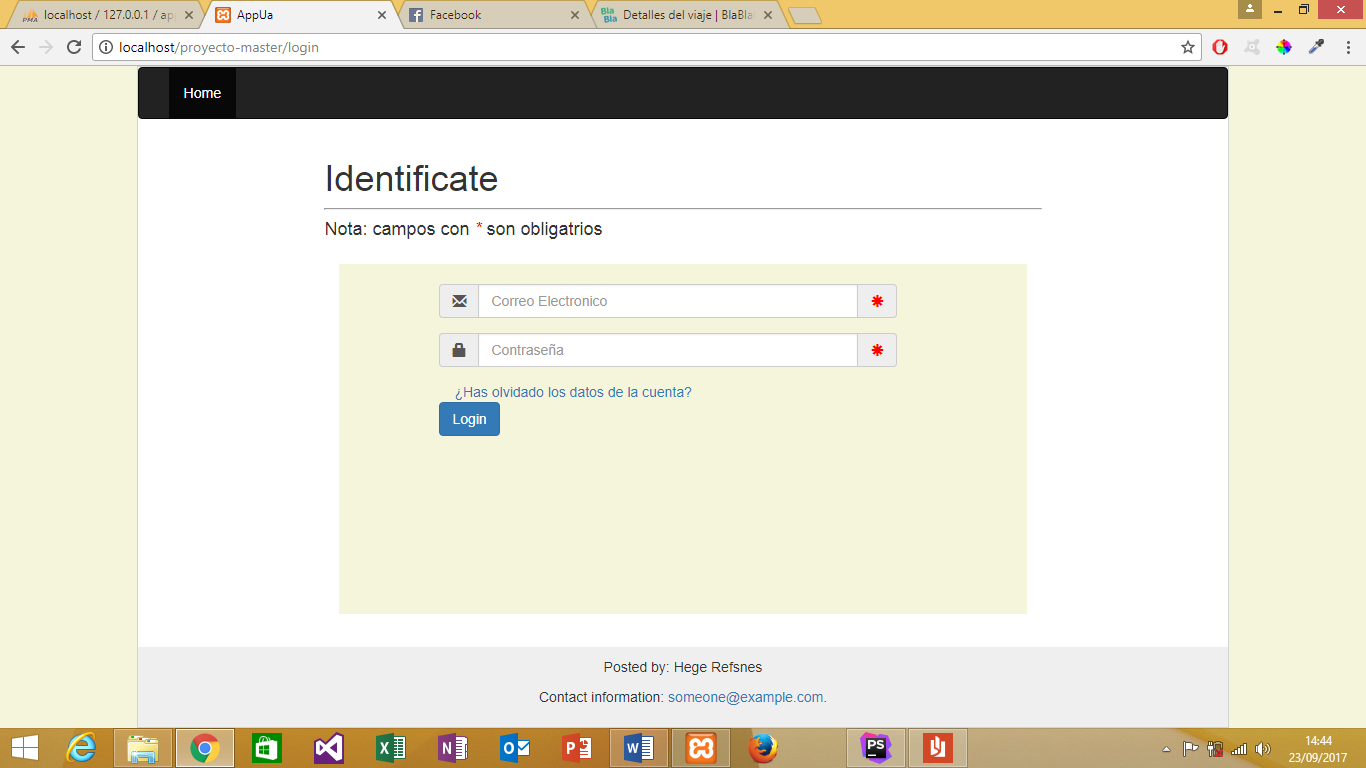


Pantalla de Inicio de Sesión

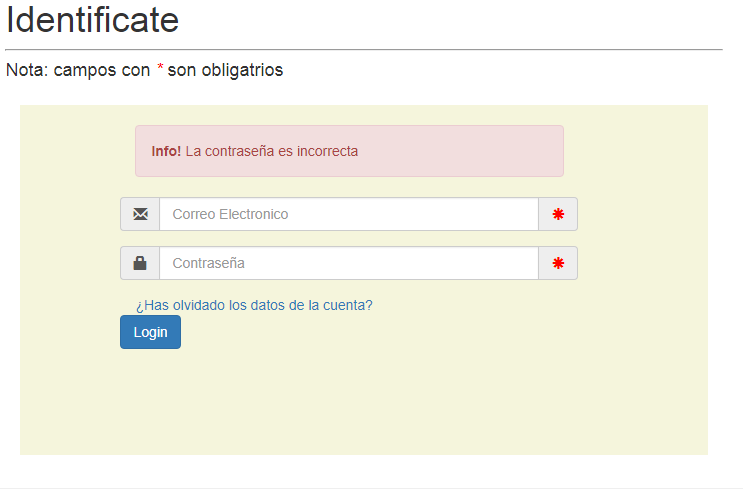
La pantalla de inicio de sesión sirve para que los usuarios puedan acceder con sus

credenciales a la aplicación.

La pantalla tendrá varias campos, un botón para realizar la petición, un enlace en caso de perder la contraseña y dos campos para introducir texto en el cuál se deberán introducir las credenciales de acceso del usuario, esto es, nombre de usuario y contraseña.



Una vez esta diseñada la pantalla, se implementa una función para comprobar la coincidencia de los datos proveídos con los datos que están en la base de datos, esta función Utiliza SHA-512 como hash sobre la contraseña, se recupera la salt aleatoria de la base de datos a base del correo introducido, porque cada usuario tiene un campo salt, con la cual se hace la mezcla con el hash con la función crypt y se guarda la contraseña mezclada. Esta salt servirá para “mezclarla” con el hash relalizado sobre la contraseña y realizar un encriptado con “crypt” y por tanto comprobar la coincidencia de las contraseñas si son correctas.



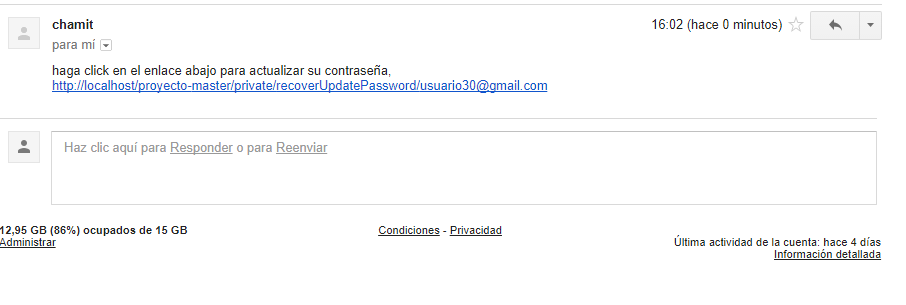
Como se observa en la captura, si la contraseña es incorrecta, se indica al usuario con el mensaje con color morado. En caso que el correo no existe en la base de datos se le indica con el mensaje reflejado en la captura siguiente:

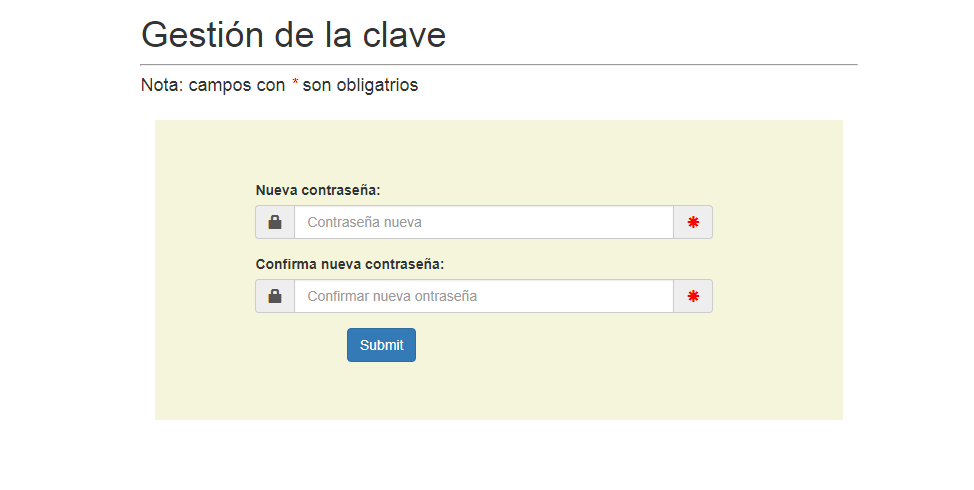


Pantalla recuperación de contraseña

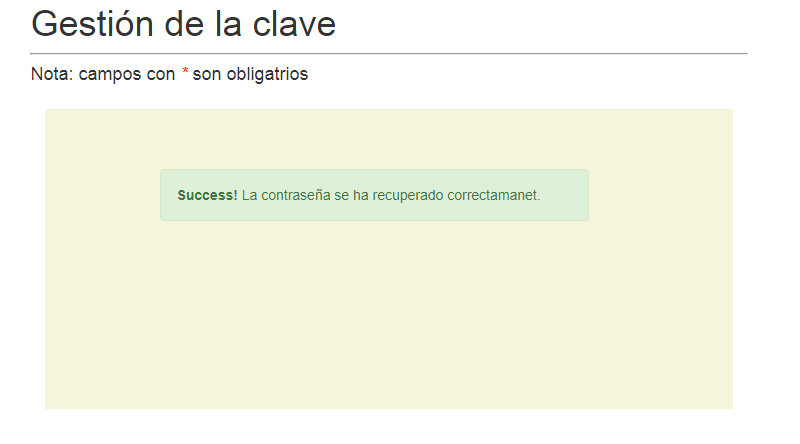
Si el usuario no se acuerda de su contraseña, bajo cualquier constancia, tiene la posibilidad de recuperar su cuenta, solo introduzca su correo, botón send, recibirá un correo con un enlace mediante el cual se lleva a otra vista donde introduzca su nueva contraseña.





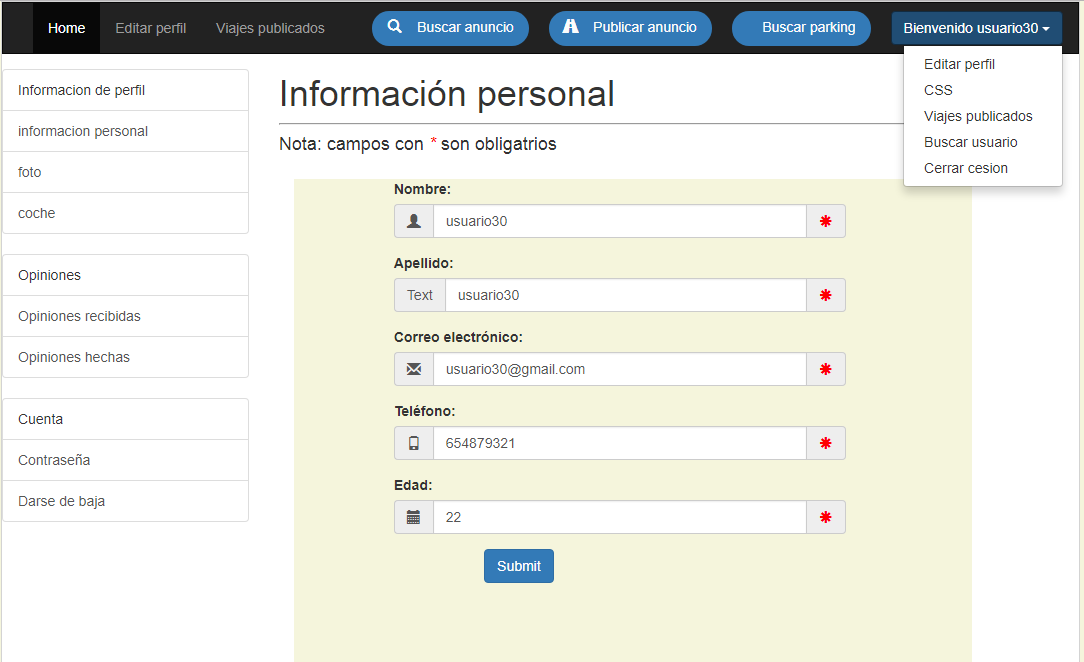


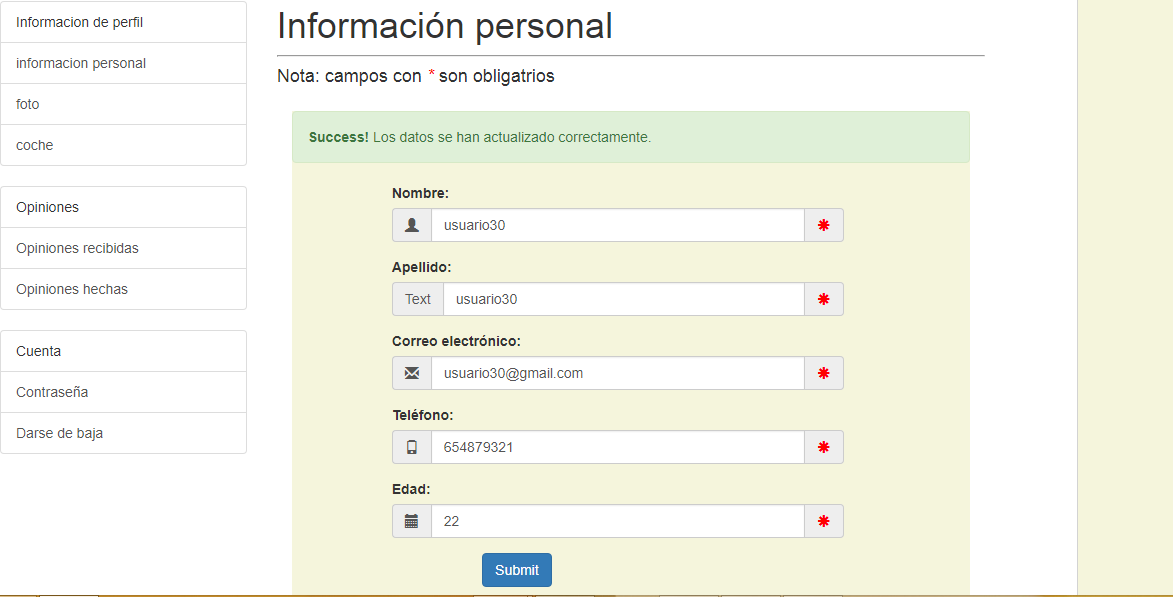
Como resultado de la recuperación de la cuenta, se le notifica al usuario con el mensaje “La contraseña se ha recuperado correctamente”



Pantalla Editar Perfil

Como se observa en la captura, cuando el usuario inicia sesión, aparecen los menús, dos horizontales y dos verticales, los cuales llevan a todas las funcionalidades de la aplicación, por ejemplo en la captura siguiente, el usuario puede editar la información personal, su foto, coche, ver comentarios hechos y recibidos y actualizar contraseña etc.





# 

Editar foto

En esta pantalla el usuario, puede poner foto, borrarla y cambiarla por otra.



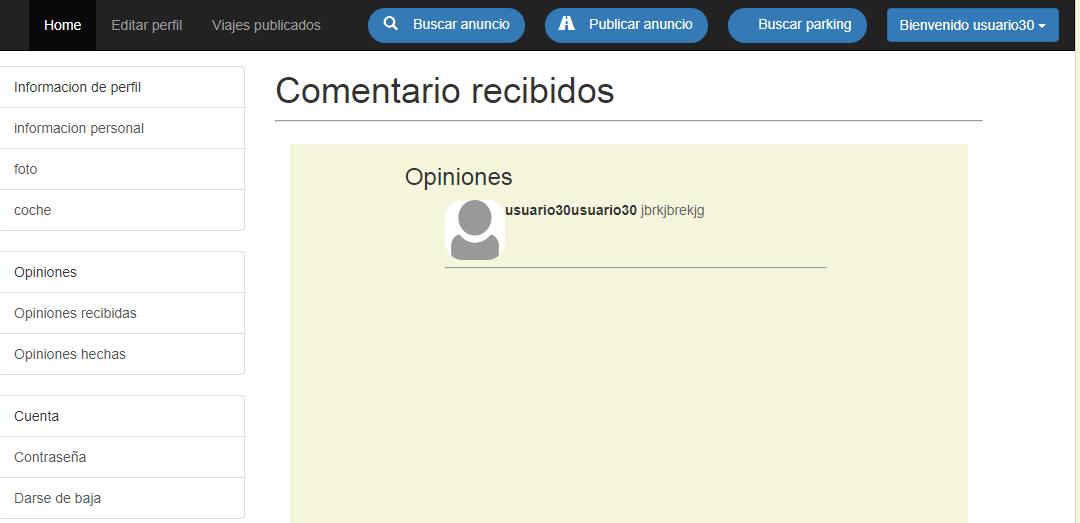
Cuando el usuario, no tiene foto o la borra, en esta vista puede incluir una.



Vista de opiniones realizados



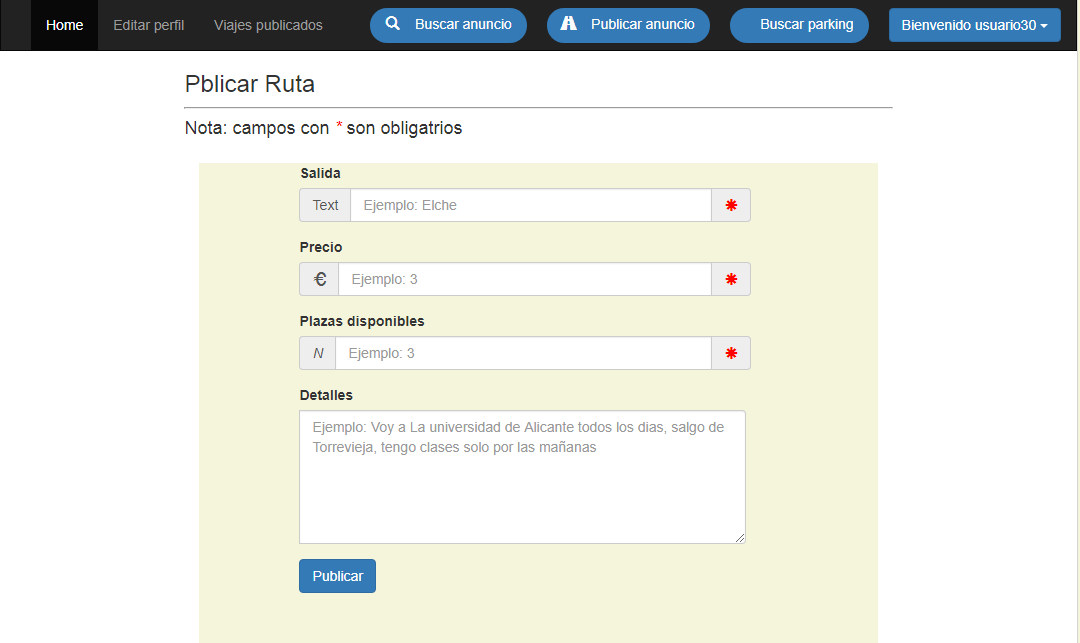
Comentarios recibidos



Pantalla publicar anuncio

En esta vista el usuario, publica un anuncio, para buscar usuarios con el fin de compartir el coche, el usuario, pone el punto de salida, el precio, las plazas disponibles y un detalles del anuncio.

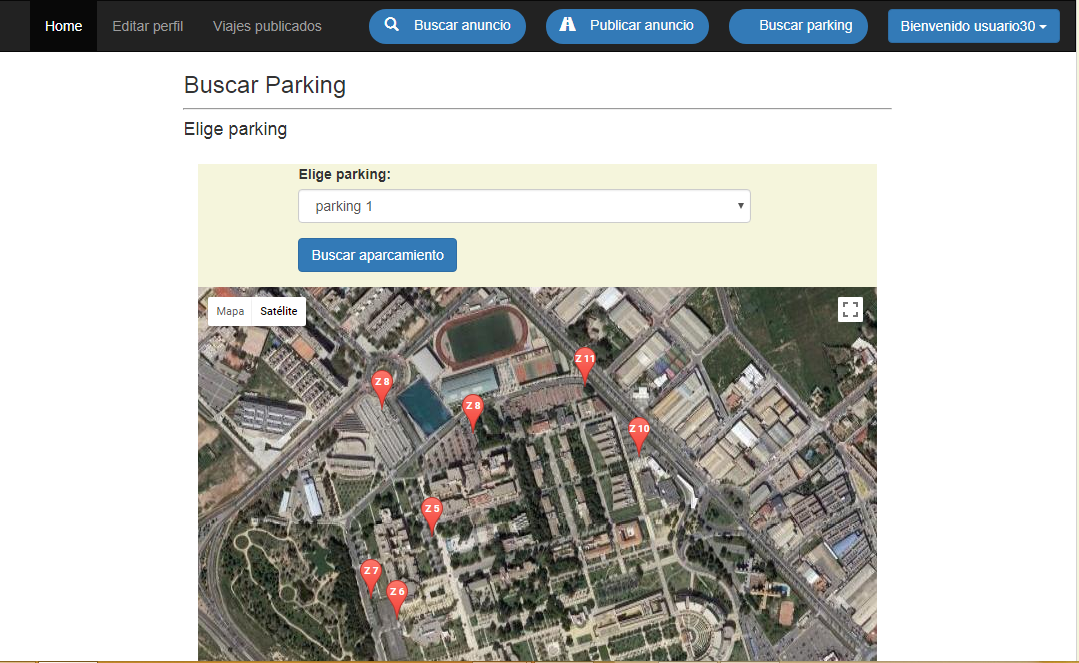
Si el usuario no dispone de un coche no puede publicar ningún anuncio, ya que cuando introduzca los datos y los envía al servidor, se le notifica con un mensaje de que debe introducir datos del su coche





Pantalla buscar parking

Cuando el usuario busca parking, la opción Buscar parking lo lleva a una vista con el mapa de la universidad, con los parkings indicados con los puntos rojos.



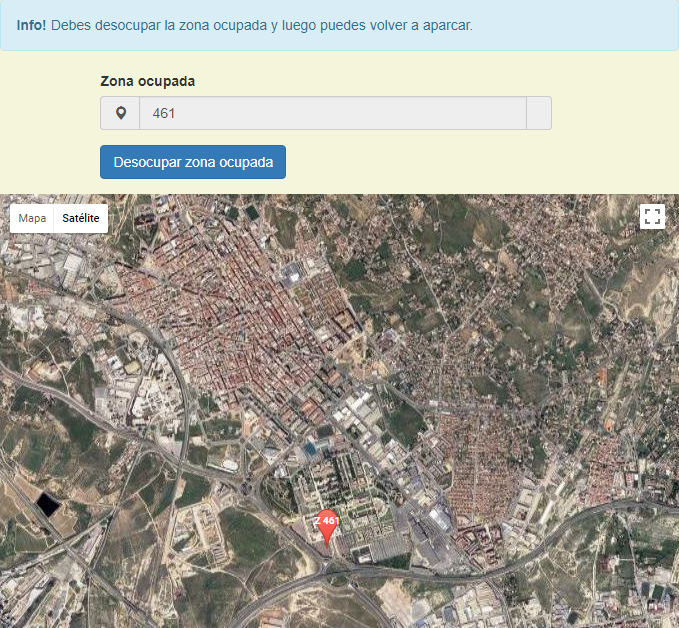
Pantalla Eligir zona.

Mediante la opción del select, cuando el usuario elige un parking mediante su numero, se le lleva otra vista con las zonas vacías de este parking.



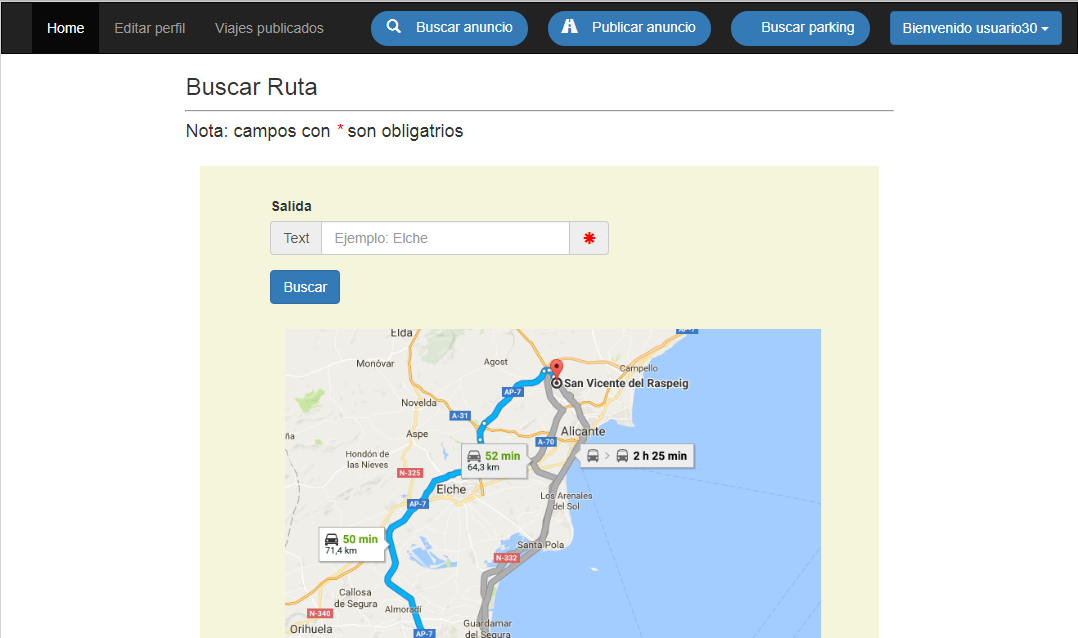
Pantalla Desocupar zona.

Cuando el usuario indica que ha ocupado una zona dada, se le lleva a una vista para desocupar esta zona, y además, si no la desocupa, cuando vuelve a buscar zonas vacías en los parkings, siempre se le lleva a esa vista para marcar como desocupada la zona, así puede volver a buscar huecos.



Pantalla buscar anuncio

En esta pantalla el usuario, pone un origen, en el campo origen del formulario, como resultado, le salen todos los anuncios desde ese origen con plazas disponibles.



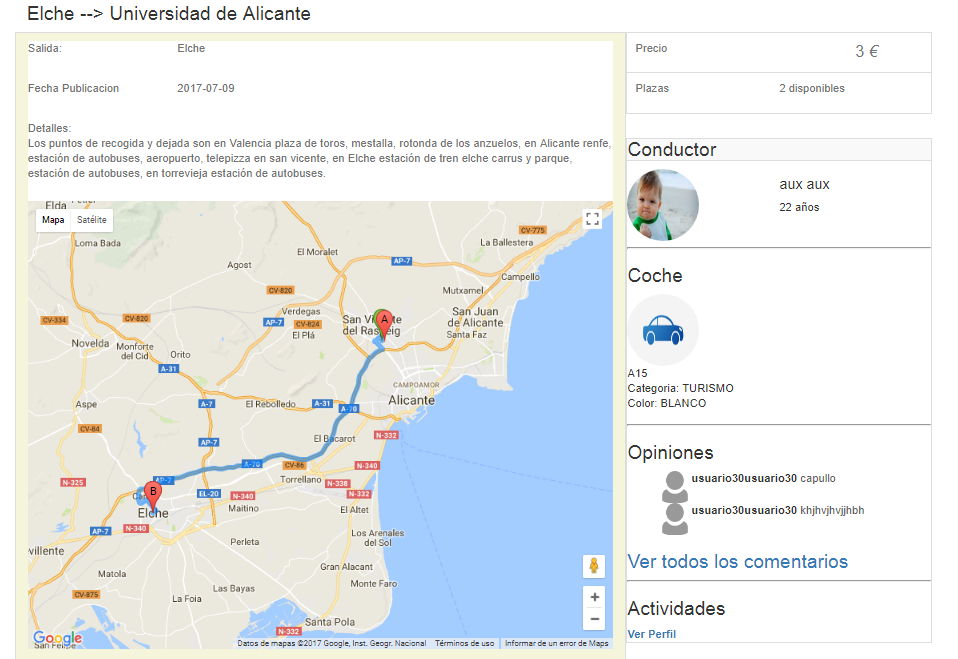
Como resultado, le sale una vista en modo paginación para ver todos los anuncios.





Cuando el usuario selecciona un anuncio le sale todos los datos del anuncio en otra vista

Con una mapa indicando la ruta desde el punto de salida hasta la universidad

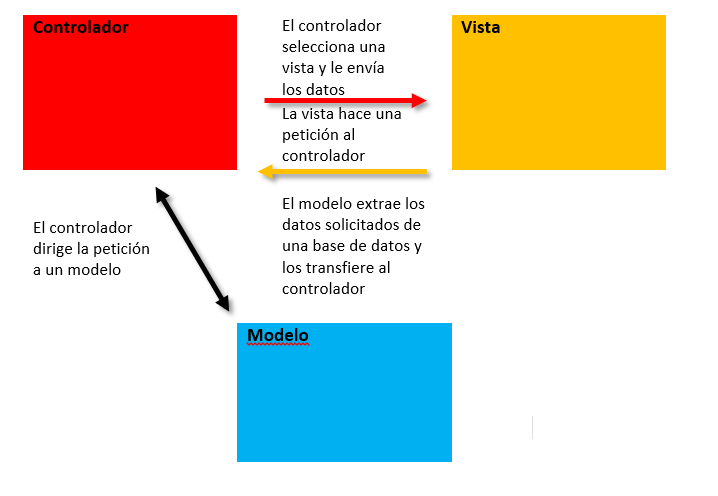


# ARQUETECTURA

El modelo utilizado en el desarrollo es Modelo vista controlador mediante el framework CodeIgniter, el cual hace uso del patrón MVC, un patrón de desarrollo que separa una aplicación en 3 capas:

* **Modelo:** es la capa de acceso a datos, en ella se realizan las conexiones y consultas a la base de datos, y los datos que esta obtiene son recibidos por la capa Controlador.
* **Controlador:** aquí encontramos la lógica del programa, esta capa manipula la información que recupera de la base de datos, y la envía a la Vista. También recibe las acciones del usuario capturadas por la Vista y responde a ellas.
* **Vista:** en esta parte se define la interfaz de usuario, especifica cómo se mostrará la información obtenida del Controlador y captura los eventos del usuario.





# 

# IMPLEMENTACION

## - Front end

Html/css

En cuanto a la vista, en el Front end, he utilizado el lenguaje de marcado HTML, es el elemento de construcción más básico de una página web y se usa para crear y representar visualmente una página web. Determina el contenido de la página web, pero no su funcionalidad. Otras tecnologías distintas de HTML son usadas generalmente para describir la apariencia/presentación de una página web ([CSS](https://developer.mozilla.org/es/docs/Web/CSS)) o su funcionalidad ([JavaScript](https://developer.mozilla.org/es/docs/Web/JavaScript)/jquery).

HTML le da "valor añadido" a un texto estándar en español. Hiper Texto se refiere a enlaces que conectan una página Web con otra, ya sea dentro de una página web o entre diferentes sitios web. los vínculos son un aspecto fundamental de la Web. Al subir contenido a Internet y vincularlo a páginas de otras personas, te haces participante activo de esta Red Mundial.

HTML usa "markup" o marcado para anotar textos, imágenes, y otros contenidos que se muestran en el Navegador Web. El lenguaje de marcado HTML incluye "elementos" especiales tales como [<head>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/head), [<title>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/title), [<body>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/body), [<header>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/header), [<article>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/article), [<section>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/section), [<p>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/p), [<div>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/div), [<span>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/span), [<img>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/img), y muchos otros más.

Para dar estilo a las paginas webs, he utilizado CSS Hojas de Estilo en Cascada (Cascading Style Sheets) y Bootstrap.

CSS es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos.

CSS funciona a base de reglas, es decir, declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas aplicadas a un documento HTML o XML. La regla tiene dos partes: un selector y la declaración. A su vez la declaración está compuesta por una propiedad y el valor que se le asigne.

**h1 {color: red;}**

**h1** es el selector

**{color: red;}** es la declaración

El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto. En el ejemplo anterior, el selector **h1** indica que todos los elementos h1 se verán afectados por la declaración donde se establece que la propiedad **color** va a tener el valor **red** (rojo) para todos los elementos **h1** del documento o documentos que estén vinculados a esa hoja de estilos.

Las tres formas que he utilizado dar estilo a un documento son las siguientes:

Utilizando una hoja de estilo externa que estará vinculada a un documento a través del elemento **<link>**, el cual debe ir situado en la sección **<head>**.

**<!DOCTYPE html>  
<html lang="en">**

**<html>**

**<head>**

**<title>Título</title>**

**<link href="<?php echo base\_url();?>assets/css/perfil.css" rel="stylesheet">**

**</head>**

**<body>**

**.**

**.**

**.**

**.**

**</body>**

**</html>**

Utilizando el elemento **<style>**, en el interior del documento al que se le quiere dar estilo, y que generalmente se situaría en la sección <head>. De esta forma los estilos serán reconocidos antes de que la página se cargue por completo.

**<!DOCTYPE html>  
<html lang="en">**

**<head>**

**<title>hoja de estilo interna</title>**

<**style**>  
.**navbar** {  
 **margin-bottom**: 0;  
 **border-radius**: 0;  
 }  
  
.**sidenav** {  
 **padding-top**: 20**px**;  
 **background-color**: **#f1f1f1**;  
 **height**: 100%;  
 }  
  
 */\* On small screens, set height to 'auto' for sidenav and grid \*/* **@media screen and** (**max-width**: 767**px**) {  
 .**sidenav** {  
 **height**: **auto**;  
 **padding**: 15**px**;  
 }  
  
  
 }  
 **ul** {  
 **list-style-type**: **none**;  
 }  
  
 .**paginacion**{  
 **background-color**: **#F5F5F5**;  
 **padding**: 20**px** 0 0 0;  
 **margin**: **auto**;  
 **width**: 500**px**;  
 **height**:80**px**;  
 **text-align**: **center**;  
 **margin-top**:40**px**;  
  
  
 }  
 .**paginacion a**{  
 **text-decoration**: **none**;  
 **padding**: 0;  
 **background-color**: **#4CAF50**;  
 **color**: **white**;  
 **border-radius**: 5**px**;  
 **margin**:0;  
 **width**:80%;  
  
 }  
 .**paginacion a**:**hover**{  
 **background-color**:**#333333**;  
 **color**: **#C0C0C0**;  
  
 }  
 .**pagination a**:**hover**:**not**(.**active**) {  
 **background-color**: **#ddd**;  
 **border-radius**: 5**px**;  
 }  
 .**pagination a**.**active** {  
 **background-color**: **#4CAF50**;  
 **color**: **white**;  
 **border-radius**: 5**px**;  
 }  
 .**actual**{  
 **color**: **#FFFFFF**;  
 **padding**: 4**px**;  
 }  
 .**flex-container** {  
 **display**: **-webkit-flex**;  
 **display**: **flex**;  
 **width**: 100%;  
 **height**: 100%;  
 */\* background-color: lightgrey;\*/* }  
  
</**style**>

**</head>**

**<body>**

**<h1>Aquí se aplicará el estilo de letra para el Título</h1>**

**</body>**

**</html>**

### Y por ultimo Incluir CSS en los elementos HTML.

**<div class="col-md-4" style="border-right-style:groove;height:80%">**

En cuanto al Bootstrap, para utilizarlo, se puede descargar de dos maneras, compilado o mediante el código fuente original. En mi caso he utilizado la versión compilada, la cual tiene la siguiente estructura de archivos y directorios:

bootstrap/

├── css/

│ ├── bootstrap.css

│ ├── bootstrap.min.css

│ ├── bootstrap-theme.css

│ └── bootstrap-theme.min.css

├── js/

│ ├── bootstrap.js

│ └── bootstrap.min.js

└── fonts/

├── glyphicons-halflings-regular.eot

├── glyphicons-halflings-regular.svg

├── glyphicons-halflings-regular.ttf

└── glyphicons-halflings-regular.woff

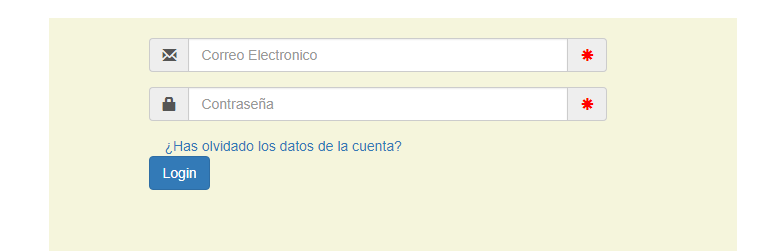
Después de descargarlo y ponerlo en una ruta especificada, para poder tener el acceso, y utilizar esas librerías, se debe incluir la ruta en el elemento <head> en el doc html como el ejemplo siguiente:

<**link href="<?php echo** base\_url();**?>assets/include/css/bootstrap.min.css" rel="stylesheet"**>

Con este mecanismo, se puede acceder y utilizar la librería, como en este ejemplo:

<**form action="<?php echo** site\_url(**'loginUser'**)**?>" name="myForm" class="form-horizontal" method="post"**>  
 <**div class="form-group"**>  
 <**div class="input-group"**>  
 <**span class="input-group-addon"**><**i class="glyphicon glyphicon-envelope"**></**i**></**span**>  
 <**input type="text" name="correo" class="form-control" id="correoElectronico" placeholder="Correo Electronico" required**><**span class="input-group-addon"**><**i style="color**:**red**;**" class="glyphicon glyphicon-asterisk"**></**i**></**span**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**div class="input-group"**>  
 <**span class="input-group-addon"**><**i class="glyphicon glyphicon-lock"**></**i**></**span**>  
 <**input class="form-control" name="contraseña" type="password" id="password1" placeholder="Contraseña" required**><**span class="input-group-addon"**><**i style="color**:**red**;**" class="glyphicon glyphicon-asterisk"**></**i**></**span**>  
 </**div**>  
 </**div**><**div**> <**a href=<?php echo** site\_url(**"public/recoverPass"**)**?>**>¿Has olvidado los datos de la cuenta?</**a**></**div**>  
 <**div class="form-group"**>  
 <**input type="submit" value="Login" class="btn btn-primary"**>  
 </**div**>  
</**form**>

Como se observa todas las clases son de la librería Bootstrap, con la cual se le da un estilo a los elementos de html, en este caso al los elementos input tienen dos elementos laterales, uno que demuestra el tipo del elemento de forma grafica y el otro como es un campo obligatorio, también el botón de login, mediante la clase btn btn-primary se le da buen estilo, con coclor azul y esquinas redondas.



Javascript/Jquery

para dar dinamismo a las páginas web y incorporar efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario, he utilizado la librería JQuery, es una [biblioteca](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) [multiplataforma](https://es.wikipedia.org/wiki/Multiplataforma) de [JavaScript](https://es.wikipedia.org/wiki/JavaScript), creada inicialmente por [John Resig](https://es.wikipedia.org/wiki/John_Resig), que permite simplificar la manera de interactuar con los documentos [HTML](https://es.wikipedia.org/wiki/HTML), manipular el árbol [DOM](https://es.wikipedia.org/wiki/Document_Object_Model), manejar eventos, desarrollar animaciones y agregar interacción con la técnica [AJAX](https://es.wikipedia.org/wiki/AJAX) a páginas web.

Se puede incluir de dos formas, una interna y otra externa:

La forma interna, se trata de incluir la librería descargada de la pagina <http://getbootstrap.com/> oficial, o cualquier fuente, que la contiene, una vez esta descargada la librería, se incluye un link que contiene la ruta hasta su ubicación, este ejemplo refleja su la manera como se incluye en la aplicación web:

<**script src="<?php echo** base\_url();**?>assets/include/js/jquery-3.2.1.min.js"**></**script**>

La forma externa, se puede utilizar desde Google or Microsoft. En mi caso he utilizado el enlace de Google.

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.2.1.min.js"></script>

En este ejemplo, se controla el campo nombre, mediante el cual se busca un usuario con su nombre, solo esta permitido nombres con letras, si no es asi, se le muestra un mensaje al usuario, y se desactiva el botón submit, hasta que el usuario corrija el error, una vez desaparece el error, el usuario puede realizar la consulta.

$("#boBuscarUsuarioN" ).click(function(event){

var valor= $("#usuarioNombre").val();

var characters = /^[a-z," "]+$/i;

if (!characters.test(valor)) {

$("#messageTipoN").html("El nombre solo debe contener letras");

return false

}else{

//$("#messageTipo").html("");

}

});

Peticiones Ajax/json

En cuanto a la realización de peticiones de consulta de datos, para evitar duplicidad, a la hora de almacenar información en la base de datos y además sin recarga de la pagina, he utilizado la tecnología Ajax, la cual permite intercambiar información entre el servidor y el cliente (navegadores) sin la necesidad de recargar la página. De esta forma, ganamos en usabilidad, experiencia y productividad del usuario final.

En este ejemplo cuando el usuario pone el email en el campo de registro, una vez sale del campo, se realiza una petición al servidor, para comprobar si existe el dato:

**$**(**function**(){  
 **$**(**"#correoElectronico"**).**blur**(**function**(event)  
 {  
  
 event.preventDefault();  
 **var** correo= **$**(**"#correoElectronico"**).val();  
 **var** url= **"<?php echo** base\_url(); **?>private/correo/"**+correo;  
  
 **$**.ajax(  
 {  
 **type**:**"GET"**,  
 **url**: url,  
 success:**function**(response)  
 {  
  
 **if**(response==**'false'**) {  
 **$**(**"#messageCorreo"**).html(**"Ya hay un usuario con esta cuenta, introduzca otra cuenta"**);  
 **$**(**'form input[id="boUsuario"]'**).prop(**"disabled"**, **true**);  
 }**else**{  
 **$**(**"#messageCorreo"**).html(**""**);  
 **$**(**'form input[id="boUsuario"]'**).prop(**"disabled"**, **false**);  
 }  
  
  
 },  
 error: **function**(error)  
 {  
 **$**(**"#message"**).html(error);  
 }  
 }  
 );  
 });  
});

El servicio que responde la consulta anterior en el lado del servidor:

**public function** correo($correo){  
 $resultado=$this->**RegistroModel**->consultarCorreo($correo);  
 **if**(!$resultado)  
 {  
 **echo "false"**;  
 }  
 **else** {  
 **echo "true"**;  
 }  
}

## - Back end

existe un procedimiento para atender una solicitud de página del cliente. Este proceso se realiza internamente por el propio CodeIgniter y de manera transparente s. Durante el proceso participan varios módulos como el enrutamiento de la solicitud, la caché interna, etc.



el flujo de aplicación que implementa CodeIgniter, puedes seguir los siguientes puntos:

1. Toda solicitud de una página a partir de CodeIgniter comienza en un index.php que hay en la raíz del framework.

2. Luego se realiza un filtrado de la URL para saber cuál es elemento que tiene que

procesar esta página.

3. Si la página se había generado antes y está en la caché de CodeIgniter, se devuelve el archivo de la caché ya generado, con lo que se ahorra procesamientos repetidos. La caché se puede configurar y si lo deseamos, incluso deshabilitar.

4. Antes de continuar con el proceso se realiza un tratamiento de seguridad sobre la entrada que tengamos, tanto de la información que haya en la URL como de la información que haya en un posible POST, si se ha configurado así.

5. El controlador adecuado realiza el procesamiento de la solicitud. CodeIgniter decide el controlador que debe procesar la solicitud en función de la URL solicitada.

6. El controlador comunica con una serie de módulos, los que necesite, para producir la página.

7. A través de las vistas adecuadas, el controlador genera la página, tal cual se tiene que enviar al navegador.

8. Si la página no estaba en la caché, se introduce, para que las futuras solicitudes de esta página sean más rápidas.

el funcionamiento empieza con con un URL: para dirigirse al controlador, como unidad de control central entre la vista y el modelo, es necesario introducir un URL en la barra de búsqueda del navegador web. Para ello, los desarrolladores crean las denominadas **clases de controlador**, archivos PHP con diversas **funciones**que permiten cargar librerías, extensiones o helpers, establecer conexiones a bases de datos, integrar un modelo o seleccionar una vista determinada.

El flujo de aplicación de CodeIgniter se basa en el siguiente esquema de URL básico:

*example.com/class/function/parameter*

el caso de class/function existe la posibilidad que el usuario, pueda darle palabras intuitivas, por ejemple si el usuario necesita registrarse, la clase puede ser Registro, y el método darseDeAlta, en el fichero de enrutamiento se puede relacionar cualquier ruta con la clase y el método, por ejemplo, si quisiera darse de alta, puede relacionar *esa ruta example.com/public/rigistro/parameter con el controlador y el método con la forma siguiente Registro/darseDeAlta, con eso el sistema llama al controlador Registro y al método darseDeAlta, prepara la vista y devuelve el resultado a la petición.*

CodeIgniter permite crear controladores individuales como clases definidas por el usuario. Para ello crean un archivo PHP separado para cada controlador en el directorio *application/controllers/.*Los controladores se crean como subclases de la *clase CI\_Controller*. En el código fuente se realiza con ayuda de la palabra clave *extend*s.

class Registro extends CI\_Controller {

}

*Registro hereda todas las funciones de la visibilidad*public*y*protected *de la clase CI\_Controller.*

*Cada clase también tiene un constructor, con lo cual se pueden integrar librerías, un modelo de datos, bases de datos o clases helper. Desde PHP5, se utiliza*\_\_construct() *como estándar.*

**public function** \_\_construct() {  
 **parent**::*\_\_construct*();  
  
 $this->**load**->database();*// podría hacerlo desde el autoload* $this->**load**->helper(**'url'**);  
 $this->**load**->library(**'session'**);  
 $this->**load**->model(**'CocheModel'**);  
  
  
}

Una vez creada la clase y definido el constructor, se crean los métodos, los cuales se invocan vistas o se interacciona con un modelo de datos integrado.

*//funcion borrar ruta***public function** borrarRutaCoches($id){  
 **if**($this->**session**->userdata(**'user'**)) {  
 $data = **array**(  
 **'id'** => $id,  
 );  
 $Resultado[**'ruta'**] = $this->**RutaModel**->borrarRutaUsuario($data);  
 *//echo "<pre>";  
 // print\_r($Resultado);  
 // echo "<pre>";* $data2 = **array**(  
 **'usuario'** => $this->**session**->userdata(**'user'**)->**correo**,  
 );  
 $output[**'rutas'**] = $this->**RutaModel**->obtenerRutasUsuario($data2);  
 **if** ($Resultado == **true**) {  
 $output[**'Exito'**] = **'exito'**;  
 } **else** {  
 $output[**'Error'**] = **'error'**;  
 }  
  
 $this->**load**->view(**'private/listadoRutas'**, $output);  
 }**else**{  
 $this->**load**->view(**'public/home'**);  
 }  
  
}

En el ejemplo anterior, después que un usuario intente borrar una ruta, el cual se llama el método borrarRutaCoches, se comprueba si el usuario ha iniciado sesión, se coge el parámetro, se llama al modelo y se borra la ruta, prepara la respuesta, levanta la vista pasándole los datos del proceso, para reflejarlos al usuario, como resultado de la acción borrar ruta.

Para conectarse con la base de datos, y recuperar información solicitada, se utilizan los Modelos, Las clases modelo permiten a los desarrolladores definir funciones de forma individual para las operaciones de bases de datos. En este ejemplo se crea el modelo coche con el cual se realizan todas las operaciones posibles con la entidad coche, por ejemplo insertar, consultar, borrar y actualizar.

**class** CocheModel **extends** CI\_Model  
{  
 **function** insertaCoche($data)  
 {  
 *//Inserta coche en la base de datos* $this->**db**->insert(**'coche'**, $data);  
 **return** ($this->**db**->affected\_rows() > 0);  
  
 }  
}

En el ejemplo anterior, en el modelo insertacoche, recibe los datos de un coche y los almacena en la base de datos.

En este ejemplo se hace una consulta para recuperar todos los datos del coche.

**function** get\_contents($data)  
{  
 $this->**db**->select(**'\*'**);  
 $this->**db**->from(**'coche'**);  
 $this->**db**->where(**'usuario'**, $data[**'usuario'**]);  
 *//$this->db->where('contraseña', $data['contraseña']);* $query = $this->**db**->get();  
 **return** $result = $query->row();  
}

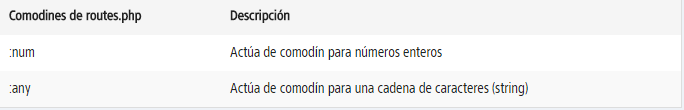
Enrutamiento con CodeIgniter

La dirección URL solicitada por el usuario, indica a CodeIgniter qué Ctrolador y que metodo son invocados. Para ello, el framework utiliza el esquema clase/función/parámetro, un esquema básico que se puede modificar según las necesidades. CodeIgniter dispone del archivo routes.php en el directorio application/config/ que contiene un array denominado $route, que es el que permite a los desarrolladores definir sus propios criterios de enrutamiento.

$route[**'registro'**] = **'registro/index'**;  
$route[**'registroUser'**] = **'registro/indexRegister'**;  
$route[**'confirmRegistro/(:any)'**] = **'registro/confirmRegistro/$1'**;  
  
$route[**'login'**] = **'login/index'**;  
$route[**'loginUser'**] = **'login/indexLogin'**;  
$route[**'cerrarSesion'**] = **'login/unset\_session\_data'**;*//cerrar sesion*$route[**'private/modifyPerfil'**] = **'modifyPerfil/index'**;  
$route[**'private/coche'**] = **'coche/index'**;  
*//foto del perfil*$route[**'private/fotoPerfil'**] = **'foto/index'**;  
*//borrar foto*$route[**'private/borrarFoto'**] = **'foto/borrarFoto'**;  
*//actualizar usuario*$route[**'updateUser'**] = **'modifyPerfil/updateUser'**;

Cuando se requiere crear una regla de enrutamiento nueva para una dirección dinámica, los desarrolladores web disponen con CodeIgniter de dos opciones: definir entradas de enrutamiento para URL dinámicos con Wildcards (comodines) o con expresiones regulares.

El documento routes.php soporta dos tipos de comodín:



Aquí hay ejemplos.

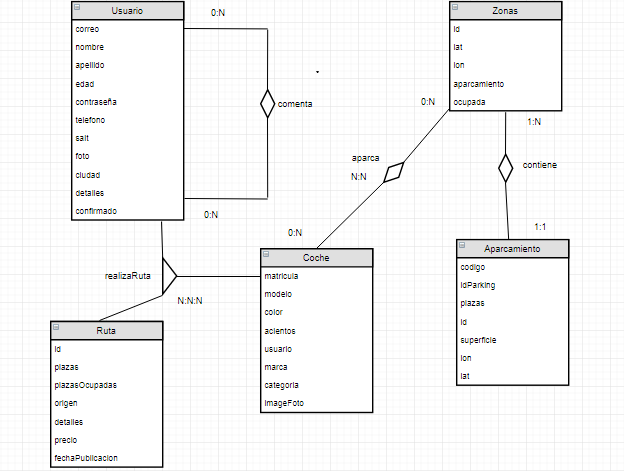
$route[**'private/buscarAnuncios/(:num)'**] = **'ruta/buscarAnuncios/$1'**;

*//cambiar foto coche*$route[**'private/cambiarFotoCoche/(:any)'**] = **'coche/fotoCoche/$1'**;

## Base de datos

### Esquema conceptual

Para visualizar la base de datos de una forma más clara he tomado una captura un esquema Entidad-Relación que representa las entidades de la base de datos así como las relaciones entre las distintas entidades



### Modelo Relacional

El modelo relacional nos aporta de una forma sencilla e intuitiva la representación de la base de datos.

**Usuario(**correo, nombre, apellidos, edad, contraseña, teléfono, salt, foto, ciudad, detalles, confirmado**)**

**C.P.:** correo

**V.N.N.:** correo, nombre, apellidos, edad, contraseña, teléfono, salt, ciudad, detalles, confirmado.

**Coche(**matricula, modelo, color, acientos, usuario, marca, categoría, iamgeFoto**)**

**C.P.:** matricula

**C.aj.:** usuario -> usuario

**VNN.:** modelo, color, asientos, usuario, marcia, categoría.

**Ruta(** id, plazas, plazasOcupadas, origen, detalles , precio, fechaPublicacion**)**

**C.P.:** Id

**V.N.N.:** plazas, plazasOcupadas, origen, destino, precio, fechaPublicacion.

**Aparcamiento(**Id, código, idParking, plazas, superficie, lon, lat**)**

**C.P.:** Id

**V.N.N.:** código, idParking, plazas, superficie, lon, lat.

**Realiza\_Ruta(**usuario, coche, ruta**)**

**C.P.:** (usuario, coche, ruta)

**C.aj.:** usuario -> usuario

**C.aj.:** coche -> coche

**C.aj.:** ruta -> ruta

**Comenta(**UsuarioA, UsuarioB, Comentario**)**

**C.P.:** (UsuarioA, UsuarioB)

**C. Ajena:** UsuarioA -> Usuario

**C. Ajena:** UsuarioB -> UsuarioB

**V.N.N.:** Comentario

**Zonas(**id, lat, lon, aparcamiento, ocupada**)**

**C.P.:** (id)

**C. Ajena:** aparcamiento -> aparcamiento

**V.N.N.:** lat, lon, aparcamiento, ocupada.

**UsuarioAparcaCoche(**coche, zona, fecha**)**

**C.P.:** (coche, zona)

**C. Ajena:** coche -> coche

**C. Ajena:** zona -> zonas

**V.N.N.:** fecha.

### Diccionario de Datos

Otra forma de representar la información que contienen las tablas es a través del diccionario de datos, por ejemplo el diccionario de datos de la tabla usuario es la siguiente:

Donde podemos observar el nombre de la columnas, el tipo de datos, si es posible que el valor de los campos sea nulo, el valor por defecto que tendrán los campos si no se establecen y el comentario que sirve como nota aclaratoria.

Usuario

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| Correo | Varchar | No |  | Correo del usuario |
| Nombre | Varchar | No |  | Nombre del usuario |
| Apellido | Varchar | No |  | Apellido del usuario |
| Edad | Int | No |  | Edad del usuario |
| Contraseña | Varchar | No |  | Contraseña del usuario para login |
| Telefono | Varchar | No |  | Teléfono del usuario |
| Salt | Varchar | No |  | Salt generada para crear la contraseña junto con el hash |
| Foto | Varchar | Sí | Null | Foto del usuario |
| detalles | Varchar | No |  | Detalles del usuario |
| Confirmado | Enum | No | 0 | 1 indica que el usuario ha confirmado su registro mediante el correo recibido, 0 todavia el usuario no ha confirmado su registro |

Coche

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| matricula | Varchar | No |  | Correo del coche |
| modelo | Varchar | No |  | Modelo del coche |
| Color | Varchar | No |  | Color del coche |
| Asientos | Int | No |  | Acientos del coche |
| Usuario | Varchar | No |  | Usuario dueño del coche |
| Marca | Varchar | No |  | Marca del coche |
| Categoría | Varchar | No |  | Categoria del coche |
| ImagenFoto | Varchar | Sí | Null | Foto del coche |

Ruta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| id | Int | No |  | Identificador de la ruta |
| Plazas | Int | No |  | Plazas disponibles |
| Plazas ocupadas | Int | No |  | Plazas ocupadas, es decir, las que han sido solicitadas por los usuarios |
| Origen | Varchar | No |  | Punto de partida, es el origen donde sale el usuario |
| Detalles | Varchar | No |  | Detalles del viaje |
| Precio | Decimal | No |  | Precio por cada persona |
| FechaPublicacion | Date | No |  | Fecha cuando el usuario se ha dado de alta a la publicación de la ruta |

Aparcamiento

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| Id | Int | No |  | Numero cremental, se asigna a cada parking |
| Codigo | Varchar | No |  | Identificador del parking |
| IdParking | Int | No |  | Identificador de actividad según la tipología del aparcamiento |
| Plazas | Int | No |  | Numero de plazas que contiene |
| Superficie | Double | No |  | La superficie del parking |
| Lon | Double | No |  | Cordinada y |
| Lat | Double | No |  | Cordinada X |

Zonas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| Id | Int | No |  | Identificador de la zona |
| Lat | Double | No |  | Cordinada x de la zona respecto del gps |
| Lon | Double | No |  | Cordinada y de la zona respecto a gps |
| Aparcamiento | Varchar | No |  | El aparcamiento al que pertenece la zona |
| Ocupada | Enum | No |  | 0 indica que la zona no esta ocupada, 1 indica el contrario |

Realiza ruta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| Usuario | Varchar | No |  | Usuario que publica la ruta |
| Coche | Varchar | No |  | Coche del usuario que publica la ruta |
| Ruta | Int | No |  | Identificador de la ruta |

Usuario aparca coche

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| Coche | Varchar | No |  | Coche la cual aparca en una zona de un aparcamiento dado |
| Zona | Int | No |  | Zona de un aparcamiento |
| Fecha | Date | No |  | Fecha cuando el coche se aparaco |

Comenta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|  |  |  |  |  |
| Id | Int | No |  | Identificador del comentario |
| UsuarioComentado | Varchar | No |  | Usuario que recibe el comentario |
| UsuarioComenta | Varchar | No |  | Usuario que realiza el comentario |
| Comentario | Varchar | No |  | El comentario introducido por el usuario |

SEGURIDAD

La seguridad de contraseñas es una de las partes mas impornates de la aplicación, asi para evitar daños, o robo de información y como consecuencia manipular y poner en peligro informacion de los usuarios, he utilizado un cifrado muy complejo, utiliza SHA-256 como HASH y un salt aleatoria que se mezcla con el hash y construir la contraeña mediante el método crypt.

Cuando se lleva acabo el desarrollo, se guarda el resultado después de aplicar el método crypt, como contraseña cifrada, además se guarda el salt, porque cuando se realiza el login, se aplica el mismo método, el usuario introduce su contraseña, se le aplica el hash, se consulta el salt creado mediante el correo cuando se hizo el registro, se hace la mezcla y se compara con la contraseña cifrada en la base de datos.

Aquí esta el ejemplo del registro

**public function** indexRegister()  
{$pass = *hash*(**'sha512'**, $this->**input**->post(**'password'**));  
 $salt = *uniqid*(*mt\_rand*(), **true**);  
 $pass = *crypt*($pass, $salt);  
 $correo = $this->**input**->post(**'correo'**);  
 $data = **array**(  
 **'correo'** => $this->**input**->post(**'correo'**),  
 **'nombre'** => $this->**input**->post(**'nombre'**),  
 **'apellido'** => $this->**input**->post(**'apellido'**),  
 **'edad'** => $this->**input**->post(**'bday'**),  
 **'contraseña'** => $pass,  
 **'telefono'** => $this->**input**->post(**'telefonoReg'**),  
 **'detalles'** => $this->**input**->post(**'detalles'**),  
 **'salt'** => $salt  
 );  
**if** ($this->send\_mail($correo)) {  
 $this->**RegistroModel**->insertaUsuario($data);  
 $Resultado = $this->**RegistroModel**->get\_contents($data);  
 $data[**'message'**] = $Resultado;  
 } **else** {  
 $data[**'message'**] = **"el correo introducido no es incorrecto"**;  
 }$this->**load**->view(**'public/confirmRegister'**, $data);}

Y aquí esta el ejemplo del login:

**public function** indexLogin()  
{  
  
 *//generar el hash a traves de a contraseña propuesta del usuario* $pass =*hash*(**'sha512'**, $this->**input**->post(**'contraseña'**));  
 $user = $this->**input**->post(**'correo'**);  
 $data = **array**(  
 **'correo'** => $user,  
 );  
 $this->**load**->model(**'LoginModel'**);$salt=$this->**LoginModel**->get\_salt($data);  
  
 **if**(!**empty**($salt)){$pass=*crypt*($pass,$salt);  
 $data2 = **array**(  
 **'correo'** => $this->**input**->post(**'correo'**),  
 **'contraseña'** => $pass  
 );  
 $Resultado = $this->**LoginModel**->get\_contents($data2);  
  
 **if** (!**empty**($Resultado)) {  
 **if** ($Resultado->**confirmado** == **'SI'**) {  
  
 *//Se guarda el objeto* $this->**session**->set\_userdata(**'user'**, $Resultado);  
redirect(**'/'**);  
 }**else**{  
  
 $output[**'Error'**] = **'Debes confirmar tu registro mediante el correo mandado con la direccion registrada'**;  
 }  
 }**else**{  
  
 $output[**'Error'**] = **'La contraseña es incorrecta'**;  
 }  
 }**else**{  
 $output[**'Error'**] = **'Los datos son incorrectos'**;  
 }  
 $this->**load**->view(**'public/log'**, $output);  
}

APIS UTILIZADAS

REFERENCIAS

https://www.formget.com/codeigniter-uri-segment/

https://validator.w3.org/nu/#textarea

https://www.w3schools.com/bootstrap/bootstrap\_templates.asp

https://www.w3schools.com/html/default.asp

https://www.w3schools.com/css/default.asp

https://www.w3schools.com/js/default.asp

https://www.w3schools.com/sql/default.asp

https://www.w3schools.com/php/default.asp

https://www.w3schools.com/jquery/default.asp

http://librosweb.es/

https://www.codeigniter.com/user\_guide/

https://codepen.io/benske/pen/iAgpq