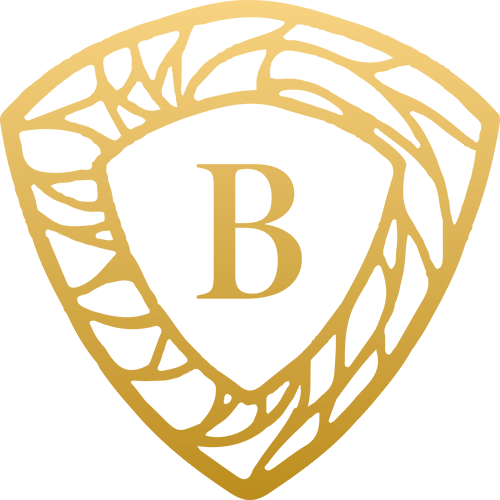
SPRINT 2



BHEC, S.L

Best High-End Cars

Josep Ferrer

Juancho Moles

Jordi Aparicio

Andreu Soriano

***IES Enric Valor de Pego* *Professorat participant***

***Curs escolar 2023-2024* Vicent Jordà**

***Desenvolupament d’aplicacions web* Toni Morant**

***2n curs* Joan Orquín**

**Javi Signes**

[**Introducción 2**](#_wg270yu4v2ze)

[**1. Diagrama E-R Unificado 3**](#_4a4ijraldf7m)

[**2. UML Unificado 4**](#_2cpwcuevd3iw)

[**3. Formularios PHP 5**](#_bd7gvjz35gn0)

[3.1 Mecanismos autenticación 5](#_e0f7ake79ilt)

[3.2 Jerarquía de excepciones 6](#_wtvvgqhke0pu)

[3.3 PDO 7](#_ggf7jlu9hgi)

[3.3.1 Transacciones PDO 7](#_acoh8opx2w69)

[3.4 PDF 8](#_tyqeuftjhhhg)

[3.5 Patrones de diseño 9](#_c6thi2dw5de0)

[3.6 Validaciones formularios 10](#_xweifcunm89h)

[3.6.1 Registro empleados 10](#_yxujhvrrt6qe)

[3.6.2 Registro particular 11](#_4lvci2vb4jza)

[3.6.2.1 CRUD 12](#_65wsz2bvwrmt)

[3.6.3 Registro profesional 13](#_965pbxr5w5mx)

[3.6.3.1 CRUD 16](#_qqh0rwfe5b89)

[3.7.1 Login 20](#_sdyeujpsmrfd)

[**4. Formularios HTML, CSS Y JS 25**](#_v51xcstp2e2l)

[4.1 Cookies 25](#_48eyrrt45x1x)

[4.2 Registro empleados 26](#_9dls2p5zc3f7)

[4.2.1 Eventos 28](#_ebk1tmbkw2db)

[4.3 Registro de particular 28](#_aq0grlga4cbk)

[4.3.1 :Valid e :Invalid 28](#_gvotqfkec1jb)

[4.3.2 Validación API 30](#_udfhhcy9mbbo)

[4.3.3 Gestión de Eventos 30](#_bmbo72od70q0)

[4.3.4 Mensajes con SweetAlert 32](#_cn1ta6gqzxcc)

[4.3.5 Mensajes con Modal 32](#_6mum3ehq15g4)

[4.4 Login 34](#_l997ddbb7ahm)

[**5. Maquetación 37**](#_jm2jixp4bolh)

[5.1 Login 37](#_isi3ibhgixn7)

[5.1.1 Escritorio 37](#_cwg93yyuk2ef)

[5.1.2 Móvil 38](#_cy4y0y787816)

[5.2 Domicilio 39](#_31g1rgwkyxp4)

[5.2.1 Escritorio 39](#_9xbgg0n7qdz8)

[5.2.2 Móvil 39](#_uxqoxbhlcezs)

[5.3 Compras 40](#_wp2fqdwb7q86)

[5.3.1 Escritorio 40](#_bqehmoxipi3z)

[5.3.2 Móvil 40](#_6b5t80xtct6p)

[5.4 Movimientos 41](#_ze30zm4fc1f6)

[5.4.1 Escritorio 41](#_5luod51z19hh)

[5.4.2 Móvil 42](#_mnezxtzfr7tb)

[**6. Conclusión 43**](#_ths1xmz6x0tf)

[**7. Webgrafía 44**](#_bcszzdaf2ctd)

# Introducción

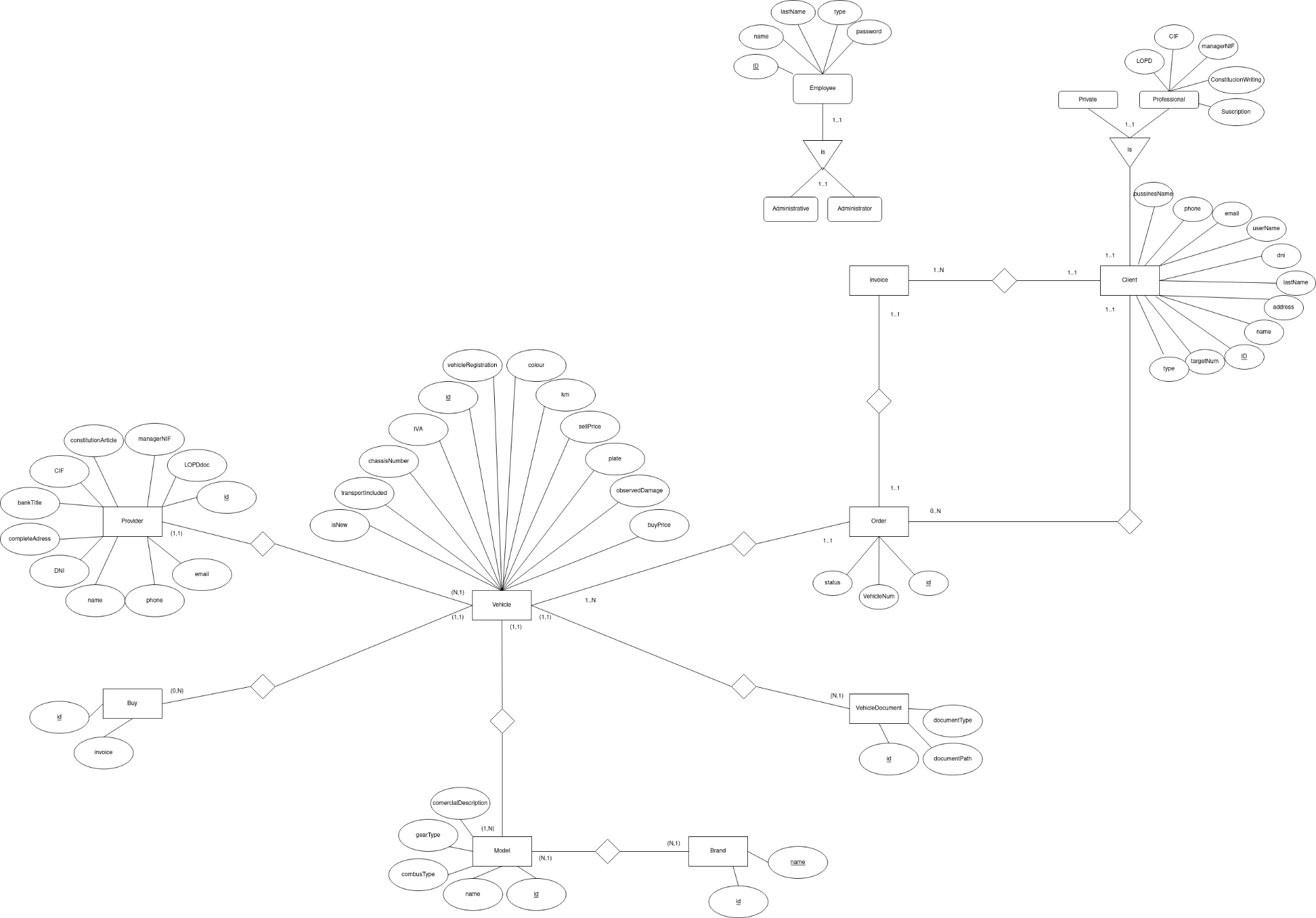
En este trabajo, exploramos un enfoque integral para la validación de formularios en el desarrollo web, utilizando tecnologías fundamentales como PHP, HTML, CSS y JavaScript. Nos sumergimos en el mundo de la programación orientada a objetos (POO) en PHP, empleando clases y funciones para validar formularios de manera efectiva y eficiente. Además, daremos un paso más allá al implementar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en cada clase PHP, permitiendo así la manipulación completa de los datos.

La validación de formularios no se limita solo al lado del servidor. Reconociendo la importancia de la experiencia del usuario, también explicaremos las técnicas de validación en el lado del cliente mediante HTML, CSS y JavaScript. Estas tecnologías trabajan en armonía para proporcionar retroalimentación instantánea a los usuarios, mejorando la usabilidad y previniendo errores antes de que lleguen al servidor.

A lo largo de este trabajo, nos adentraremos en los principios teóricos y prácticos de la validación de formularios, explorando ejemplos concretos y casos de uso del mundo real, además de maquetar y dar forma a las páginas tanto para el uso de escritorio y móvil.

# Diagrama E-R Unificado

Para la realización de este diagrama se arreglaron por separado cada uno de los diagramas ER de los distintos equipos, lo que se arregló fueron los nombres, siendo cambiados al inglés. Además se arregló la lógica de cada uno de los diagramas individuales para luego poder ser juntados todos en un diagrama único ya arreglado.



## 

# UML Unificado

# Formularios PHP

## 3.1 Mecanismos autenticación

La autenticación es importante en la protección de la información y la identidad de los usuarios en las aplicaciones web. Consiste en verificar quiénes son los usuarios antes de permitirles acceder a cuentas, los mecanismos más importantes de autenticación son:

* **Nombre de usuario y contraseña** → Método estándar donde los usuarios ingresan credenciales que deben coincidir con registros en la base de datos. Requiere gestión cuidadosa para garantizar seguridad.
* **Autenticación de dos factores (2FA)** → Añade capa de seguridad con una segunda forma de autenticación, como códigos generados en aplicaciones móviles o enviados por correo electrónico, mejorando la protección ante accesos no autorizados.
* **Autenticación basada en tokens** → Utiliza tokens temporales o permanentes para verificar la identidad del usuario. Pueden ser empleados para acceder a recursos específicos o recordar sesiones.
* **Autenticación biométrica** → Emplea rasgos físicos únicos, como huellas dactilares o reconocimiento facial, para verificar la identidad del usuario, ofreciendo un nivel adicional de seguridad y comodidad.
* **Autenticación de terceros** → Permite a los usuarios autenticarse mediante cuentas de redes sociales o servicios externos como Google o Facebook, simplificando el proceso de inicio de sesión.
* **Autenticación de certificados digitales** → Utiliza certificados emitidos por entidades confiables para verificar la identidad del usuario, común en entornos empresariales y servicios bancarios en línea.
* **Autenticación de clave única (FIDO U2F / WebAuth)** → Se basa en hardware de seguridad, como llaves USB, para autenticar al usuario, ganando popularidad por su alta seguridad.
* **Autenticación de SMS o correo electrónico** → Envía códigos de verificación al usuario a través de SMS o correo electrónico, añadiendo un paso adicional de seguridad.
* **Autenticación por tiempo de sesión** → Los usuarios se autentican una vez y pueden permanecer autenticados durante un período específico antes de requerir autenticación nuevamente, equilibrando seguridad y conveniencia.
* **Biometría en línea** → Examina el comportamiento del usuario en línea, como patrones de navegación y uso del teclado, para verificar la identidad, ofreciendo un alto nivel de seguridad, aunque menos común.

## 3.2 Jerarquía de excepciones

En PHP, las excepciones siguen una jerarquía que permite organizar y manejar diferentes tipos de errores de manera efectiva. La jerarquía de excepciones en PHP se organiza de la siguiente manera (de manera ascendente, es decir, desde las clases más específicas a las más generales):

* **Exception** → Esta es la clase base para todas las excepciones en PHP. Todas las demás clases de excepciones están directa o indirectamente derivadas de esta clase.
* **ErrorException** → Esta clase es una excepción que representa errores generados por el manejador de errores tradicional de PHP. Puedes lanzar esta excepción en tu código para convertir errores estándar en excepciones.
* **ArithmeticError** → Esta es una clase abstracta que representa errores aritméticos, como divisiones por cero. Las clases derivadas de esta incluyen “DivisionByZeroError”.
* **ParseError** → Esta clase representa errores de análisis, que ocurren durante la interpretación del código PHP. Los errores de análisis son fatales y no se pueden capturar.
* **Exception Classes específicas** → Además de estas clases base, PHP también incluye varias clases de excepciones específicas para manejar situaciones comunes, como “PDOException” para errores de base de datos, “FileNotFoundException” para errores de archivo no encontrado, etc.

## 3.3 PDO

PDO, o PHP Data Objects, es una interfaz de acceso a bases de datos para PHP. Básicamente, proporciona una capa de abstracción para interactuar con diversas bases de datos, permitiendo que tu código sea más portable y flexible. Con PDO, puedes trabajar con diferentes sistemas de gestión de bases de datos utilizando la misma sintaxis, lo que facilita el cambio entre ellas sin tener que modificar significativamente tu código. Además, ofrece medidas de seguridad incorporadas, como el uso de consultas preparadas, ayudando a prevenir ataques de inyección SQL. En resumen, PDO simplifica la conexión y manipulación de bases de datos en entornos PHP.

### 3.3.1 Transacciones PDO

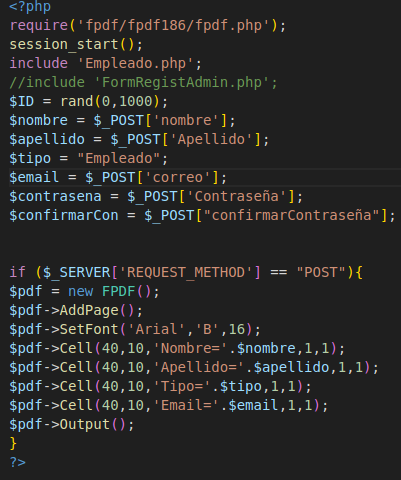
Las transacciones son secuencias de operaciones de base de datos que se ejecutan como una unidad atómica. Las transacciones en PDO se utilizan para garantizar la integridad de la base de datos, lo que es crucial en aplicaciones donde la precisión de los datos es fundamental.

* **beginTransaction()** → Este método inicia una transacción. Una vez llamado, todas las consultas subsiguientes se consideran parte de la misma transacción.
* **commit()** → Cuando todas las operaciones se han ejecutado con éxito, se utiliza el método commit() para confirmar la transacción. Esto aplica todos los cambios a la base de datos.
* **rollBack()** → Si ocurre un error o una operación falla, se utiliza el método rollBack() para deshacer todos los cambios realizados durante la transacción.

## 

## 3.4 PDF

El formato de fichero [**PDF**](https://en.wikipedia.org/wiki/PDF)fue creado por Adobe en el año 1992 y puesto a disposición de forma gratuita al año siguiente. La intención de la compañía era desarrollar algo que permitiese presentar documentos con texto e imágenes de forma que pudiesen ser visualizados independientemente de la aplicación de software, el hardware y el sistema operativo. Sin embargo, y a pesar de haber sido puesto a disposición de forma gratuita, PDF fue publicado totalmente como software privativo hasta 2008, año en el que la propia Adobe movió ficha para convertirlo en un estándar ISO.

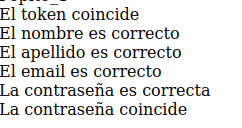
De entre todas las librerías para generar PDF decidí usar el FPDF que mediante la importación de su librería y el siguiente código puede extraer los datos de un formulario y generar un PDF:

## 

## 3.5 Patrones de diseño

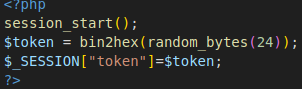
## 3.6 Validaciones formularios

### 3.6.1 Registro empleados



El siguiente formulario validará los siguientes campos comprobando que es pattern es correcto, además generará un toque en el archivo donde se encuentra el formulario,para que en la siguiente página sea imposible acceder desde el exterior.

Aquí se genera el token:

.

### 

### 

Aquí se comprueba el token:

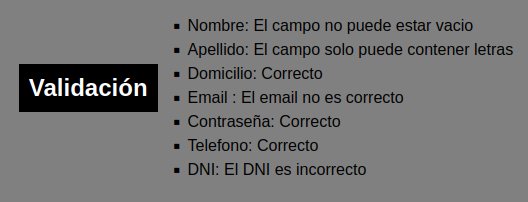
### 

### 

### 3.6.2 Registro particular

Para realizar la validación de formularios a través de las clases en PHP, se optó por un formulario simple en HTML llamado “contact.php”, siendo estos no validados ahí (aplicando el “novalidate en el form”). Luego se procedió a crear una Clase llamada “ValidarParticular” en la cual están todos los métodos que validan los datos ingresados. Esta contiene todas las variables del formulario y un constructor con las mismas. Algunas de las validaciones aplicadas fueron, empty(para saber si los campos estan vacios), *filter\_var* (usando *FILTER\_VALIDATE\_EMAIL* para validar el email), e usos de expresiones reguladoras para validar la contraseña. Estos métodos cuentan con excepciones personalizadas para poder captar el error al momento, además todos los métodos son ejecutados en un método el cual tiene como parámetro un objeto y es ahí donde se llaman a todos los métodos.

Luego la clase que es llamada por el HTML obtendrá los datos de los inputs y con el POST luego estos datos pasan a la clase “*Particular” creando un objeto para luego pasar este objeto al método de la clase “ValidarParticular”* para luego ser mostrados en un mensaje si existe un error o si todo esta bien.



#### 

#### 3.6.2.1 CRUD

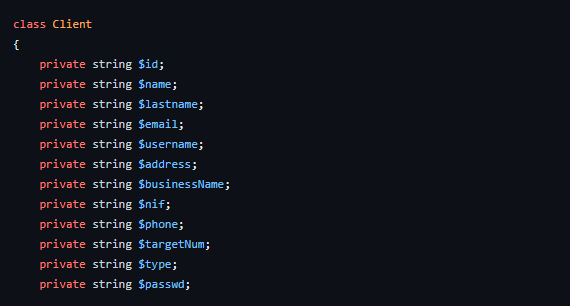
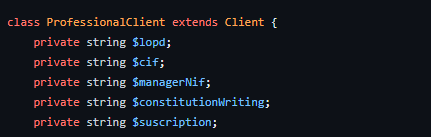
El **CRUD** se implementó a través del uso de PDO, se empezó con un try/catch en caso de que la inicialización del pdo diera error, poder tenerlo capturado dicho error. El significado de CRUD es CREATE, READ, UPDATE, DELETE. Una vez aclarado eso, se pasará a explicar la implementación de cada uno en el código php:

* **CREATE:** En el caso de crear los datos, primero se requirió de una base de datos, esta fue creada mediante un script ya explicado. Ya con la base de datos en PHPMyAdmin, la cosa se complica, pues el usuario particular es un hijo del cliente, es por esto que primero hay que crear la tabla cliente para luego con su id crear la tabla de particulares, es por ello que se procede a obtener los datos ya validados del formulario, y con pdo estos se ingresan en una consulta de INSERT INTO, la cual luego es ejecutada, lo que hace esta consulta es agregar los datos ingresados en el formulario dentro de la tabla cliente, pero, lo que se necesita es añadir el registro en particulares(cuyo único es la id). La solución propuesta a este problema fue, una vez creada el registro en clientes, obtener su id(la cual es autoincremental) con el método de PDO lastInsertId(este regresa la última id registrada), una vez obtenida la id ya se puede crear el registro de particulares añadiendo la id en el INSERT INTO con una consulta, creando de esta manera el registro. Todo esto se realiza con una transacción, usando “beginTransaction” inicio, luego “commit” al final y por último el rollback en caso de soltar una excepción, de manera que si algo da error no se realice ningún paso
* **READ:** Para leer los datos se creó una tabla HTML(table) la cual a través de echo y un array asociativo mostraba los datos en dicha tabla, para hacer esto posible se creó un php denominado “Crud.php”, en este se inicializa el PDO de la misma manera que el CREATE, sólo que luego se crea una consulta de búsqueda que muestra todos los datos de la tabla cliente mostrada según la id, luego es ejecutada y es guardada en una variable(array) con “fetchAll” el cual devuelve un array que contiene todas las filas restantes del conjunto de resultados.
* **UPDATE:** Este se podría considerar el más complicado, pero no imposible, para realizar el UPDATE se añade un botón llamado editar en la tabla, dicho botón es de tipo submit y cuenta con un value el cual tiene asignado la id de los datos en los que está. Este botón al hacerle click por ser de tipo submit te lleva a los inputs del formulario html(el cual es un .php) esto a través de un form que tiene cuyo action lo redirige ahí y pasa como parámetro una id. Una vez en el formulario “contact.php” hay una consulta que pregunta si se pasó como parámetro en la url un id, en caso de ser cierto una variable llamada “editMode” cambia a true, en caso de que sea true, cambia el funcionamiento del formulario, cuya función era crear tablas, siendo ahora la de editar los datos. Para poder lograr esta acción el modo editar en verdadero consigue inicializar un objeto de la clase PDO con el cual se obtendrán todos los datos de la tabla cuya id es igual a la que se pasó como parámetro en la URL, una vez obtenido los datos con una consulta se aplica el “fetch(PDO::FETCH\_ASSOC)” el cual guarda los datos en un array asociativo, acto seguido se rellena el formulario con estos datos, dejando el formulario lleno con los datos de la base de datos. Luego se agrega un input invisible que pasa el valor de “editMode” y otro que pasa la id. Después al darle a submit se envían todos los datos como en el formulario de siempre, pero, lo que cambia es que en el php de las validaciones también hay una condición que pregunta si el modo Editar es true, es false, entonces la tabla se crea normal, pero como en este caso venimos del botón editar, esta variable es true, cambiando el codigo, logrando que el PDO en vez de hacer un INSERT INTO haga un UPDATE en el registro cuyo id es igual a al que fue pasado actualizando de esta manera los datos existentes.
* **DELETE:** Para realizar el DELETE se creo un botón submit en la tabla que muestra los datos está refresca la página “crud.php” y pasa el valor de la id en el registro en el que está, dentro de este php hay una condición que evalúa que si se se paso un parámetro “delete” en caso de ser cierto, se borrará la tabla cuya id sea igual a la del valor del delete presionado.

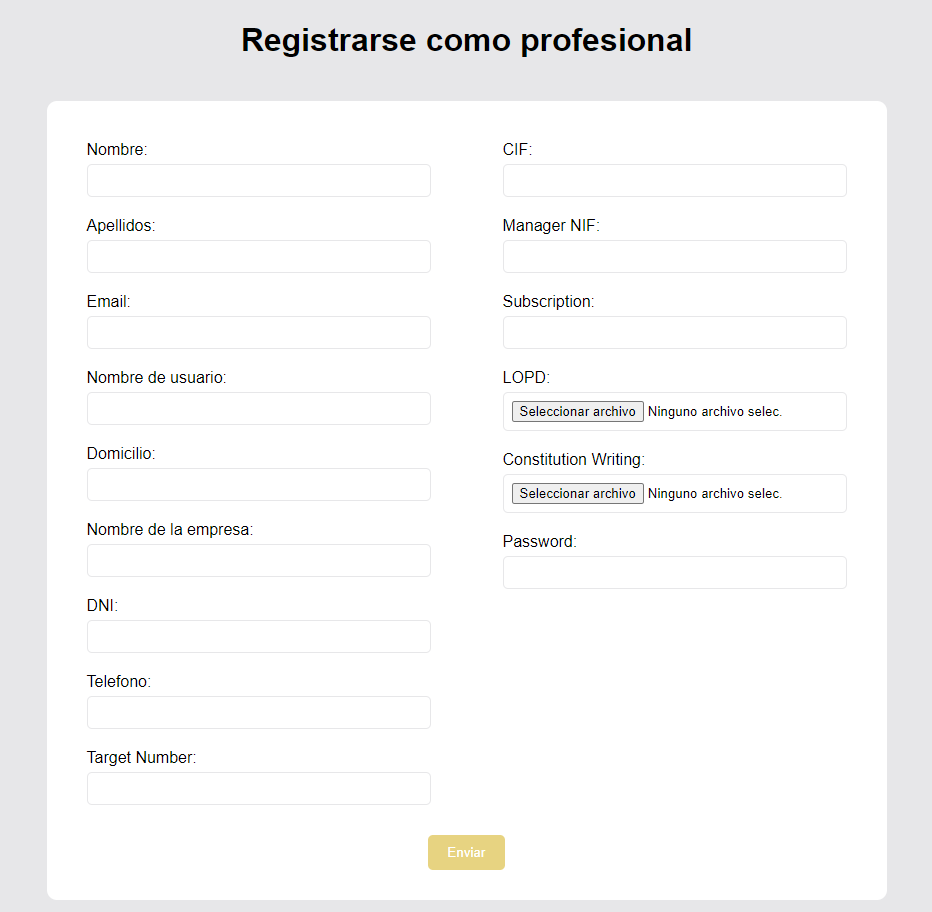
### 3.6.3 Registro profesional

Para el registro de clientes profesionales se ha optado por la utilización de un formulario que contenga todos los atributos de la clase Client más todos los atributos de la clase Professional.

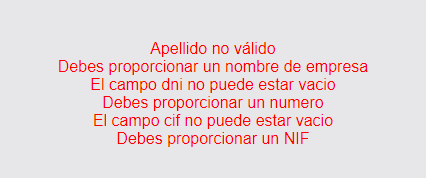
Por parte de la clase Client mandaremos los atributos id, name, lastname, email, username, address, businessName, nif, phone, targetNum, type y passwd, y por parte de la clase Professional enviaremos los atributos lopd, cif, managerNif, constitutionWriting, subscription.



En el formulario todos los campos se validaron gracias a la clase Validation, que contiene funciones específicas para cada campo.



### 

Si hay algún error a la hora de introducir los datos, en el momento que intentemos guardar el nuevo cliente no mostrará por pantalla aquel error que el sistema de validación haya encontrado.

### 

### 

### 

### 

### 

### 

#### 3.6.3.1 CRUD

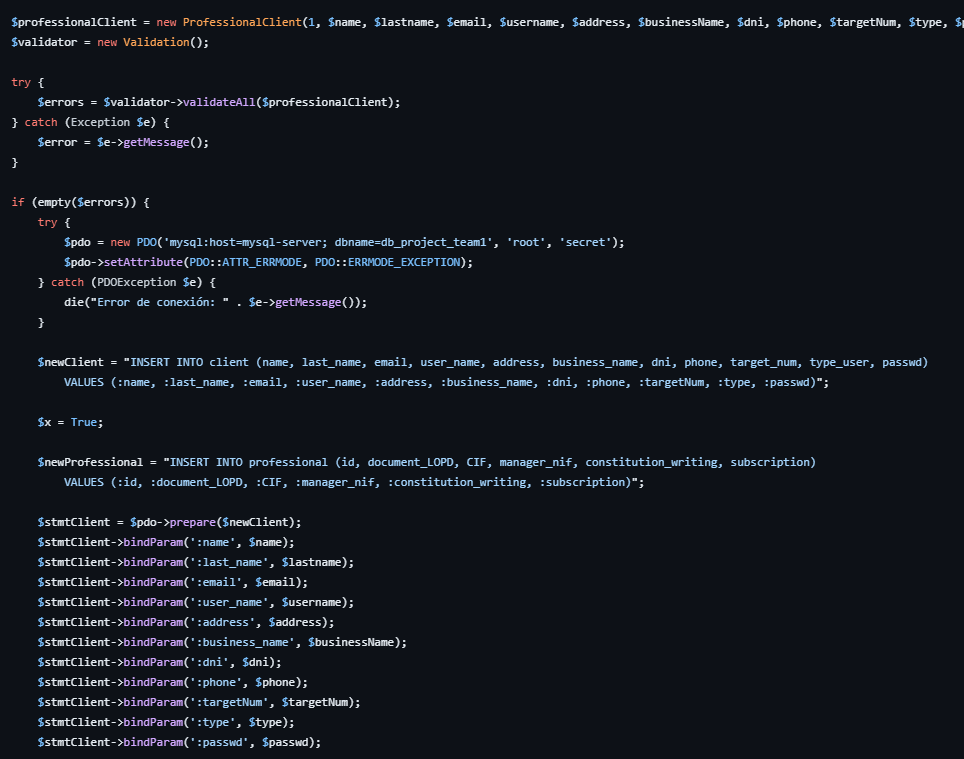
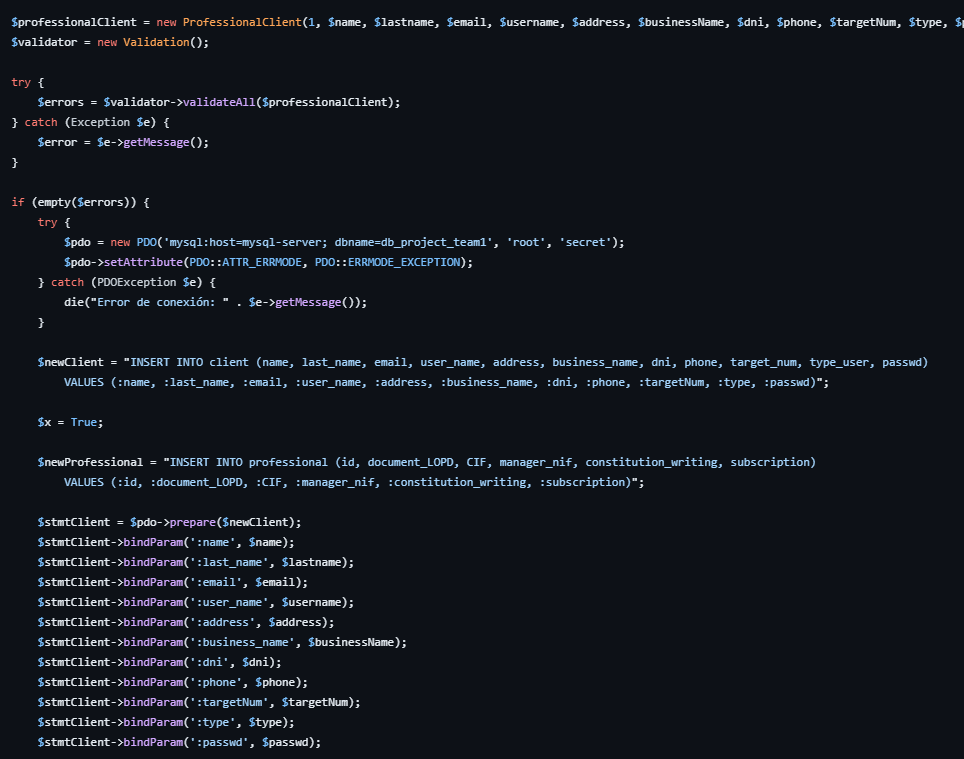
Una vez creada la base de datos, ya podemos implementar CRUD en nuestro código.

**CREATE:** Para crear nuevos clientes profesionales utilizamos el formulario anterior. Para una correcta creación del cliente, se tiene que crear primero el cliente, y después el profesional, ya que el profesional es hijo de cliente. Estos están enlazados por una id.

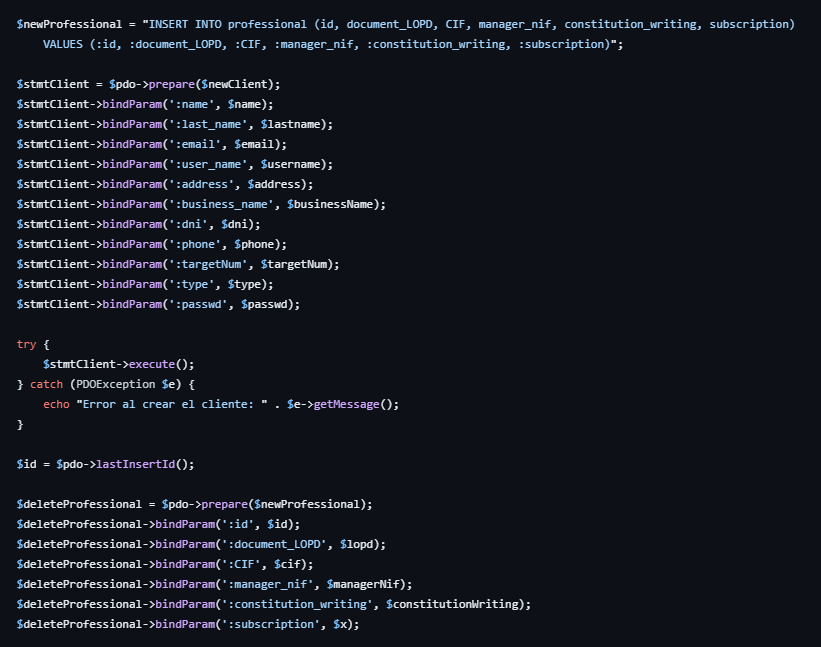
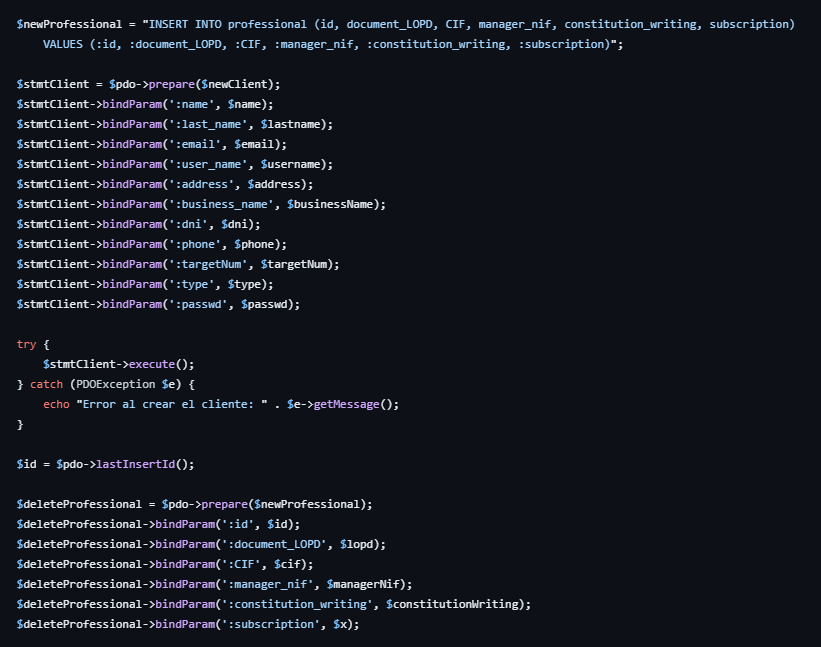
Primero se crea un nuevo ProfessionalClient con los datos del formulario y una nueva validación para validar todos los campos.

### 

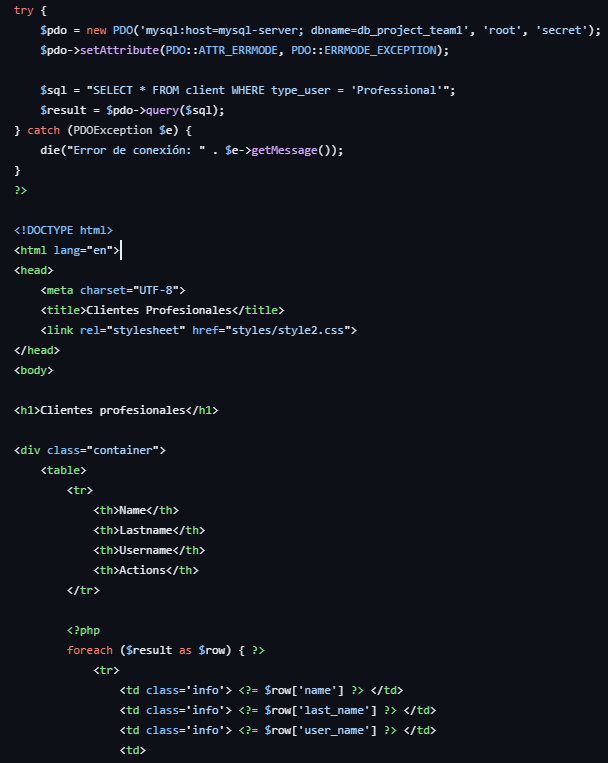
Una vez tengamos todos los campos validados y sin errores, se inserta primero el cliente en la base de datos.



Luego se utiliza la función lastInsertId() para obtener la id del cliente que se acaba de crear, y se pasa al profesional.



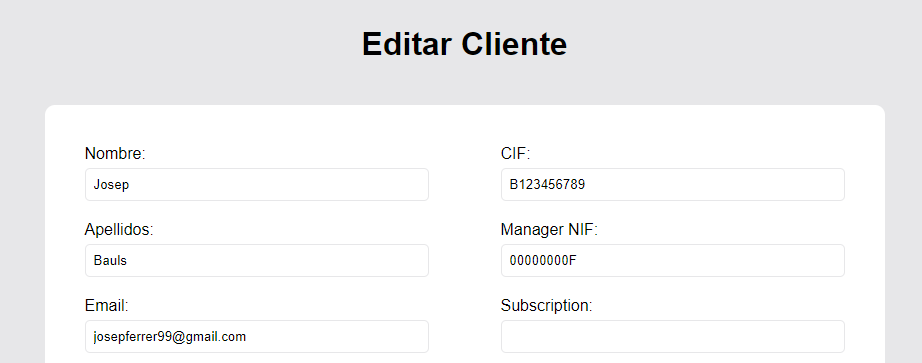
**READ:** Para leer la información de la base de datos primero creamos la conexión con la base de datos. Una vez creada la conexion, obtendremos todos los campos de cliente con la petición SELECT \* junto a una restricción, que solo nos muestre los clientes que sean de tipo Professional.

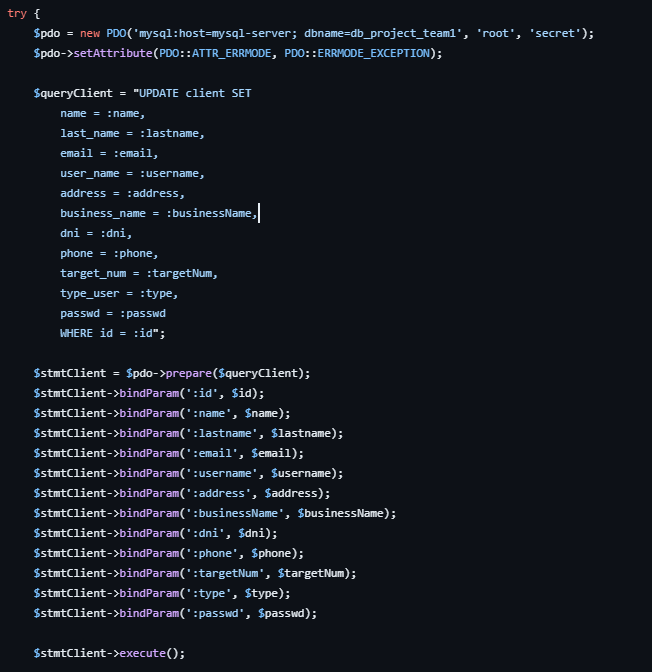


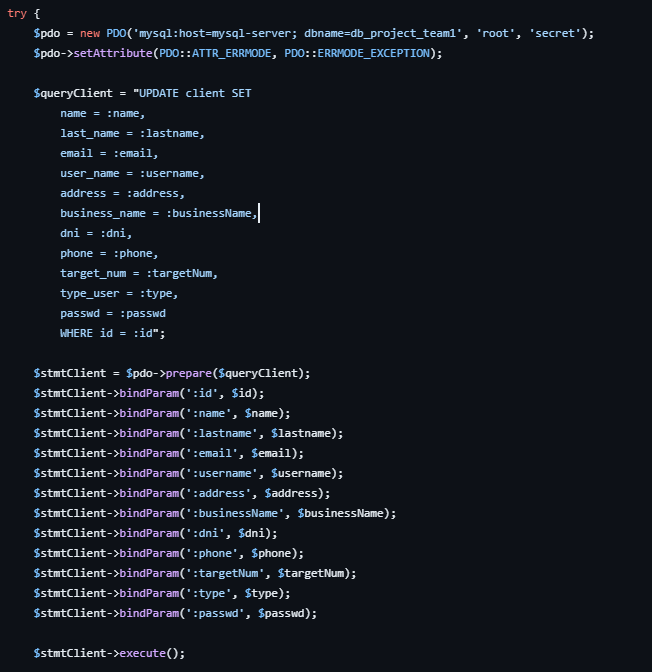
Una vez tengamos acceso a la información de la tabla cliente, mostraremos la información en forma de tabla, junto a dos botones, uno para modificar los datos y otro para eliminar al cliente.

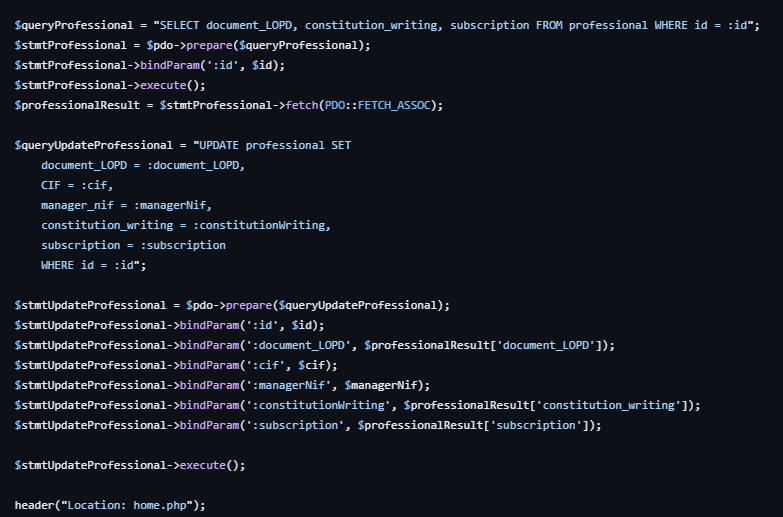


**UPDATE:** Al pulsar en el botón de modificar de algún cliente, tendremos acceso a un formulario con la información del cliente seleccionado donde podremos editarla.

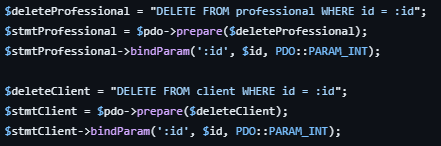
Una vez se hayan modificado los datos, al pulsar el botón de guardar, se hará un UPDATE tanto de la información relacionada con el cliente como de la de profesional.







**DELETE:** Al pulsar el botón delete de la tabla de usuarios, obtendremos la id de ese usuario. Para eliminar a un cliente en la base de datos, primero debemos de eliminar al profesional.



### 

### 

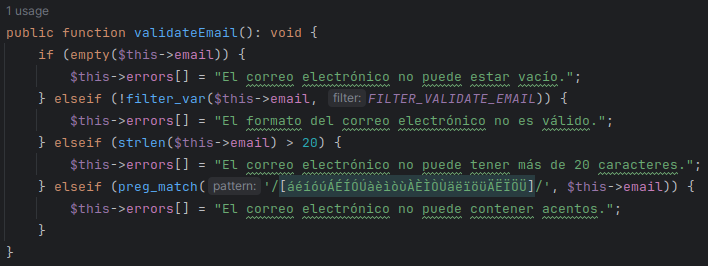
### 

### 3.7.1 Login

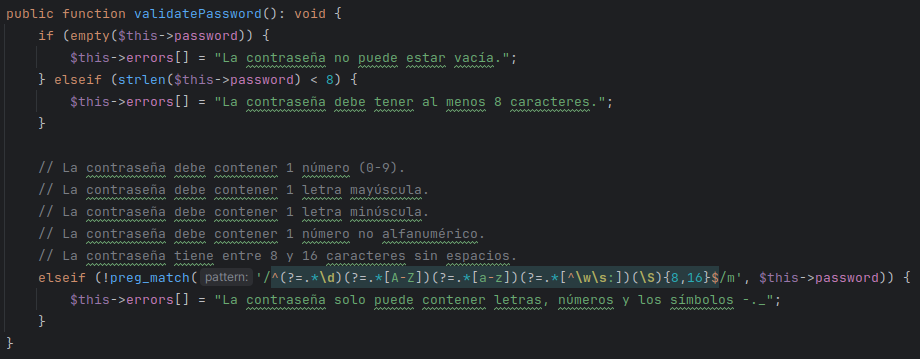
Aquí hemos hecho un formulario del iniciar sesión, poniendo un correo electrónico y una contraseña

A partir de ahí hemos creado una clase para poder validar cada campo con sus correspondientes patterns. Por ejemplo en el correo electrónico debe ser obligatorio el @, además hemos añadido que el correo no puede tener acentos ni tampoco puede estar en blanco o tener más de 20 caracteres.



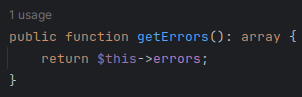


También en el apartado de la contraseña hemos puesto validaciones para que el usuario no pueda poner cualquier cosa. Por ejemplo, en el campo de la contraseña no puede estar vacía y debe tener entre 8 y 16 caracteres, debe contener 1 número del 0 al 9, una letra mayúscula, una minúscula ni tampoco espacios en blanco.





Y para poder capturar esos errores, dentro de la clase con una función.

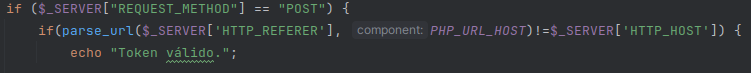


Y al final, si los dos campos están bien, te mostrará el resultado.

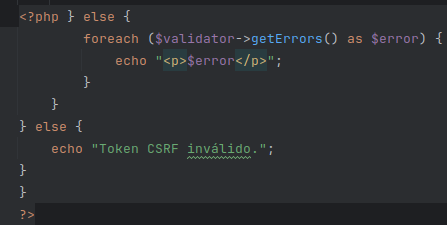


Luego, tendremos que añadir un token para no sufrir ataques CSRF. Como hemos explicado ya anteriormente.



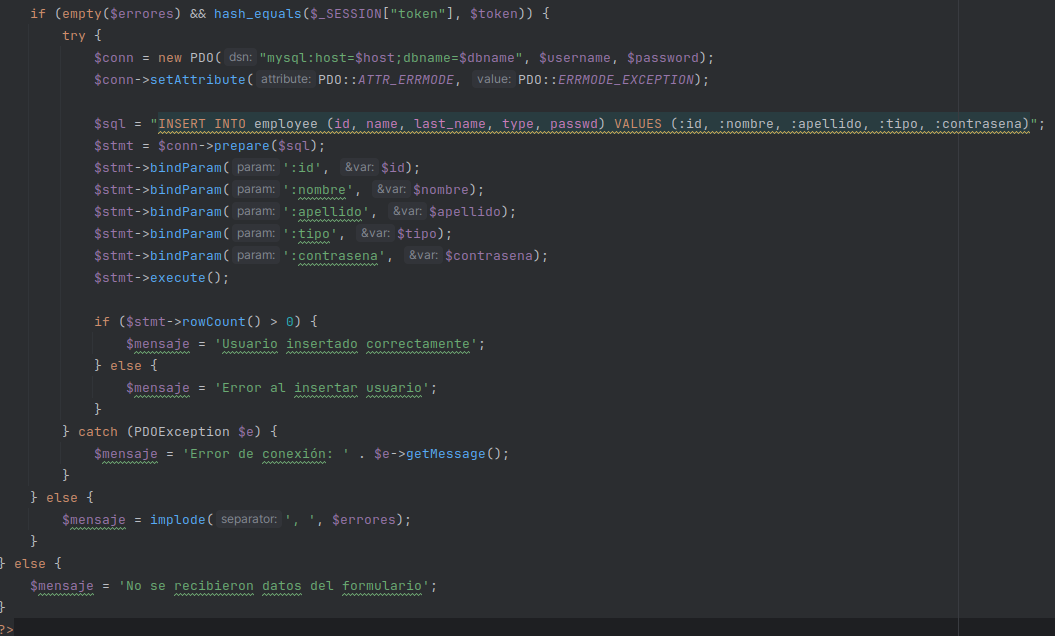


Y si el token no coincide saldrá el mensaje diciendo que el token es invalido.



Luego nos pondremos hacer el CRUD para poder crear, leer, actualizar y borrar empleados. Para hacer eso nos teníamos que conectar a la base de datos. En este caso es la clase Empleado donde se genera todo.

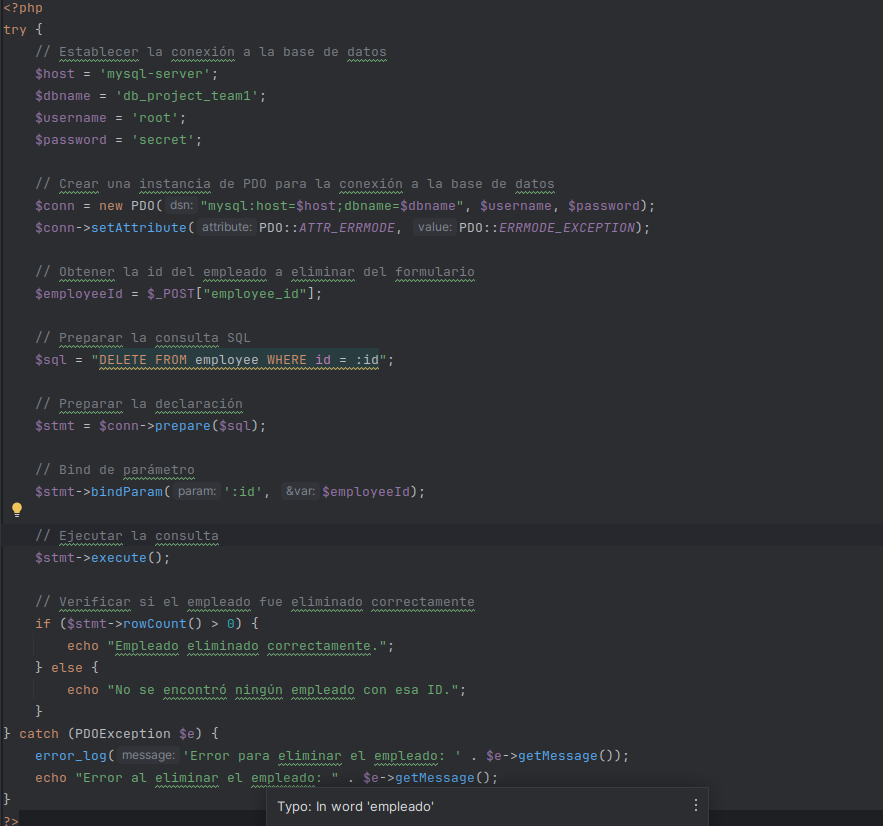
* **Create**

****

* **Read**



* **Delete**



# Formularios HTML, CSS Y JS

## 4.1 Cookies

Las **cookies** son pequeños archivos de texto que los sitios web colocan en los dispositivos de los usuarios para diversas finalidades, como recordar preferencias, personalizar la experiencia, realizar seguimientos y proporcionar datos analíticos.

Algunas formas comunes en las que las cookies se utilizan en las plataformas incluyen:

* **Inicio de sesión y autenticación** → Las cookies pueden almacenar información de inicio de sesión, lo que permite a los usuarios permanecer conectados a sus cuentas sin tener que ingresar sus credenciales cada vez que visitan la plataforma.
* **Personalización** → Las cookies se utilizan para recordar las preferencias de los usuarios, como el idioma, la ubicación y las configuraciones de visualización. Esto permite personalizar la experiencia del usuario de acuerdo con sus preferencias.
* **Publicidad dirigida** → Las cookies de seguimiento se utilizan para recopilar datos sobre el comportamiento de navegación de un usuario en diferentes sitios web. Esta información se utiliza para mostrar anuncios personalizados a los usuarios, basados en sus intereses y actividades en línea previas.
* **Análisis del usuario** → Las cookies pueden recopilar datos sobre cómo los usuarios interactúan con la plataforma, como las páginas que visitan con más frecuencia y el tiempo que pasan en el sitio. Estos datos se utilizan para realizar análisis y mejorar la usabilidad del sitio.
* **Seguridad** → Las cookies también pueden utilizarse para mantener la seguridad de la plataforma, detectando actividades sospechosas y protegiendo contra fraudes y ataques cibernéticos.

## 4.2 Registro empleados

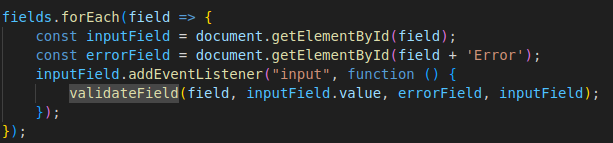
## 

## 

## 

## 

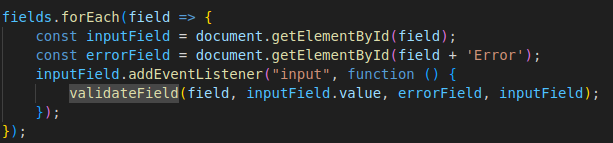
## 



Este formulario recorre los diferentes inputs y se validaron mediante la siguiente función, mostrando mensajes de error en cada campo oculto si este está equivocado, añadiendo la clase “error” a los diferentes InPuts (a la vez lanzará un error personalizado):

## 

Y la siguiente función se encargará de comprobar que todos los campos estén correctos:



### 

### 4.2.1 Eventos

* **DOMContentLoaded** → En el momento que haya cargado la página ejecutará la función.
* **Click** →En el momento que hagamos click sobre el elemento se ejecutará la función.
* **Submit** → En el momento que hagamos submit se ejecutará la función.
* **Input** → Cada vez que se añada algo a un input, se ejecutará la función.

## 4.3 Registro de particular

### 4.3.1 :Valid e :Invalid

Para la creación del registro de particulares se han aplicado los patterns (expresiones regulares), además de validaciones de :valid e :invalid en el css, estas se aplicaron usando Javascript mediante eventos ‘input’ que validaron en tiempo real cada uno de los datos pasados en el formulario, intercambiando la clase valid/invalid mediante el JS.

-Aquí se puede ver una lista de los patrones usados para cada input:

* **Nombre** → *'^[a-zA-Z ]{0,25}$'*

El texto solo debe contener mayúsculas o minúsculas con un máximo de 25 caracteres.

* **Apellido** → *'^[a-zA-Z ]{0,25}$'*

Tiene el mismo pattern que el nombre.

* **Domicilio** → *'^[a-zA-Z ]{0,50}[0-9]{0,5}$'*

Puede contener mayúsculas o minúsculas con un máximo de 50 caracteres, además puede contener números que no superen los 5 dígitos.

* **Email** → *‘^[a-zA-Z0-9]+@[a-z0-9]+\.[a-z]{2,4}$’*

Puede contener mayúsculas y minúsculas además de números, luego debe llevar una arroba obligatoria seguido de letras minúsculas o números junto a un punto y luego texto de entre 2 a 4 caracteres.

* **Contraseña** → *‘((?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[\W]).{8,})’*

Debe contener como mínimo un carácter especial, una minúscula, una mayúscula y un número, y la contraseña debe tener como mínimo 8 caracteres.

* **Teléfono** → ‘*^[0-9]{9}$'*

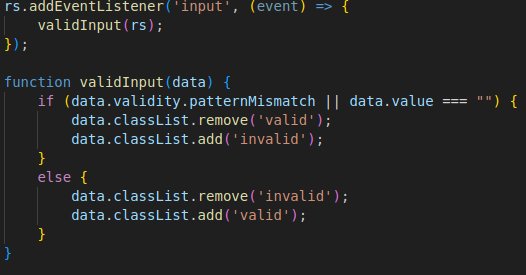
Debe contener números de 9 dígitos.

* **DNI** → *‘^[0-9]{8}[A-Z]$’*

Debe contener números de 8 dígitos además de una letra.

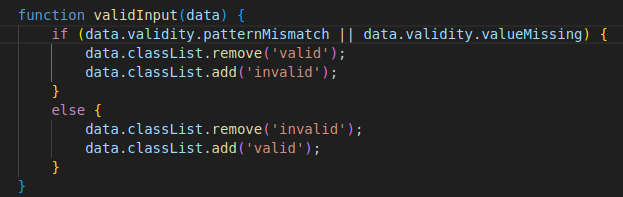
* **Razón social** →*‘^[a-zA-Z .\_\-,]{0,25}$’*

Puede contener minúsculas mayúsculas, algunos caracteres especiales como el punto “.”, la coma “,”, la barra baja “\_” y el guión “-”, y debe tener entre 1 y 25 caracteres.



### 

### 4.3.2 Validación API

Las expresiones regulares creadas en el HTML se validaron en JS mediantes APIs, siendo usadas algunas como ‘Validity.patternMismatch’ con la que sevalidaron los pattern o ‘Validity.MissValue’ para evaluar si el campo está vacío.

### 

### 

### 

### 4.3.3 Gestión de Eventos

Para el formulario se aplicaron varios eventos de JS, estos eventos serían:

* **Evento DOMContentLoaded** → Este evento HTML se usó para evitar errores a la hora de obtener los datos del formulario, ya que si la página no estaba cargada a la hora de pedirlos esto podría llevar a varios problemas en la página.
* **Evento mousemove/mouseout** → Son 2 eventos de ratón cuya funcionalidad es la de cambiar de color los nombre de los inputs en caso de que el cursor este sobre el y devolviendo su color normal en el caso contrario.

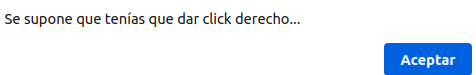
# 

# 

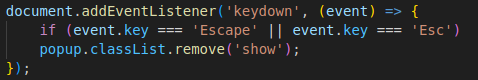
* **Evento contextmenu** → Este evento consiste en captar el click derecho del mouse, en el caso del formulario, existe un evento llamado “click derecho aqui”, con el cual al hacer click derecho te despliega un popup con las instrucciones para rellenar el formulario.

# 

* **Evento dblclick** → Este evento se aplicó en el mismo botón de “click derecho aqui” usando un addEventListener y el evento(‘dbclick’). Lo que hace es que cuando se da doble click en el botón se despliega un alert que te indica que esa no era la acción.



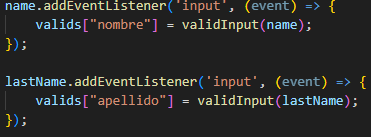
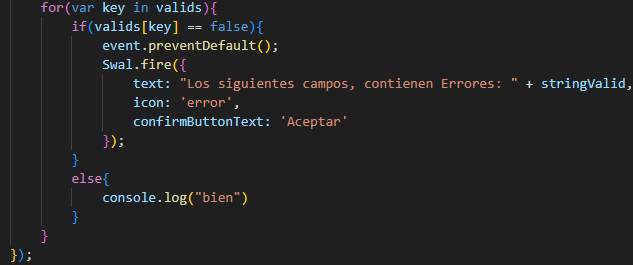
* **Evento keydown** → este evento de teclado se usó para poder cerrar el popup, siendo el botón el ‘ESC’.



## 

### 4.3.4 Mensajes con SweetAlert

El **sweetAlert** fue implementado en un submit de manera que cuando el formulario contenga errores y se le dé a enviar aparezca. Para conseguir lo dicho anteriormente, se tuvo que crear un array asociativo que almacenará el valor true o false de cada validación, consiguiendo saber qué inputs estaban malos. Una vez realizado y teniendo el array lleno, a este se le hace un forEach para buscar dentro de él los validaciones que dieron false(las incorrectas), mostrando el sweetAlert en caso de que un solo valor del array, no importa cual, sea incorrecto.



## 

## 

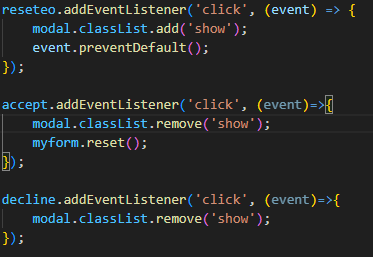
## 

## 

## 

### 4.3.5 Mensajes con Modal

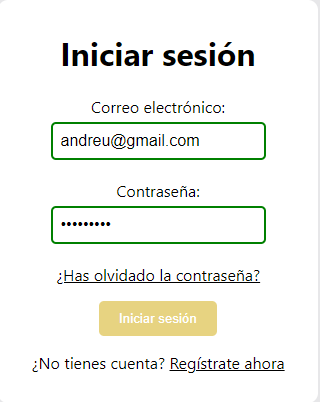
Para implementar el **modal** en javaScript, se optó por ser aplicado en un botón de reset en el formulario, este se activa con un evento ‘click’ en el botón que le da al modal(el cual está invisible) una clase ‘show’ de manera que este se muestre y un eventPreventDefault para evitar que se resetee el formulario para poder preguntar. Acto seguido aparecerán 2 opciones una para aceptar el reseteo, y otra para cancelarlo, si este se cancela se oculta el modal y no sucede nada, en caso contrario al darle a aceptar este hará un form.reset() que forzara el reinicio del formulario, ocultando también el modal.



## 

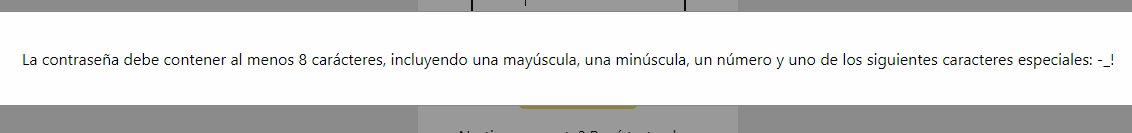
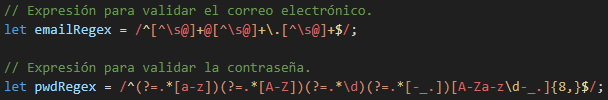
## 4.4 Login

Para este formulario, primero hemos aplicado **:valid** e **:invalid** usando simplemente CSS y generando un formulario con HTML para comprobar los campos. Si el campo está bien el borde se pondrá en verde y si está mal el borde del input se pondrá en rojo.



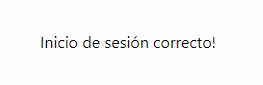
# 

Luego con JS añadiremos los patterns necesarios para que el usuario no pueda poner cualquier cosa. Si el usuario pone bien su correo o la contraseña sale un mensaje avisando que el correo electrónico o la contraseña está mal y se debe poner bien.

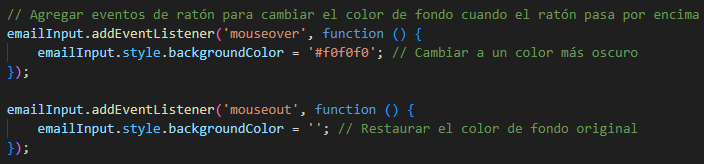


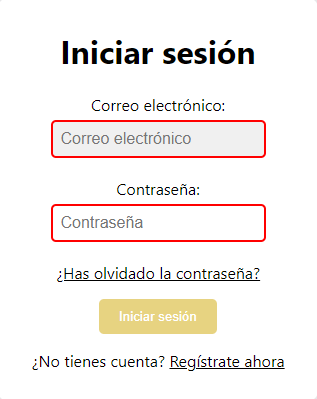


Aquí podemos ver los mensajes de que la contraseña está mal puesta y que el correo también. Aquí se puede observar que el mensaje sale como **MODAL** pero también puede salir con Sweet Alert o con un mensaje simple. Y si el inicio de sesión es correcto sale un mensaje diciendo que el inicio de sesión ha sido correcto.

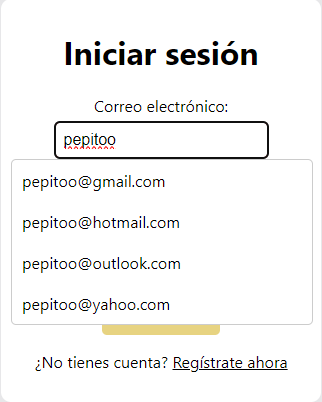


Luego, con el ratón con **mouseOver** y **mouseOut** que el input se cambie a un color más oscuro usando JS para poder modificar el CSS.





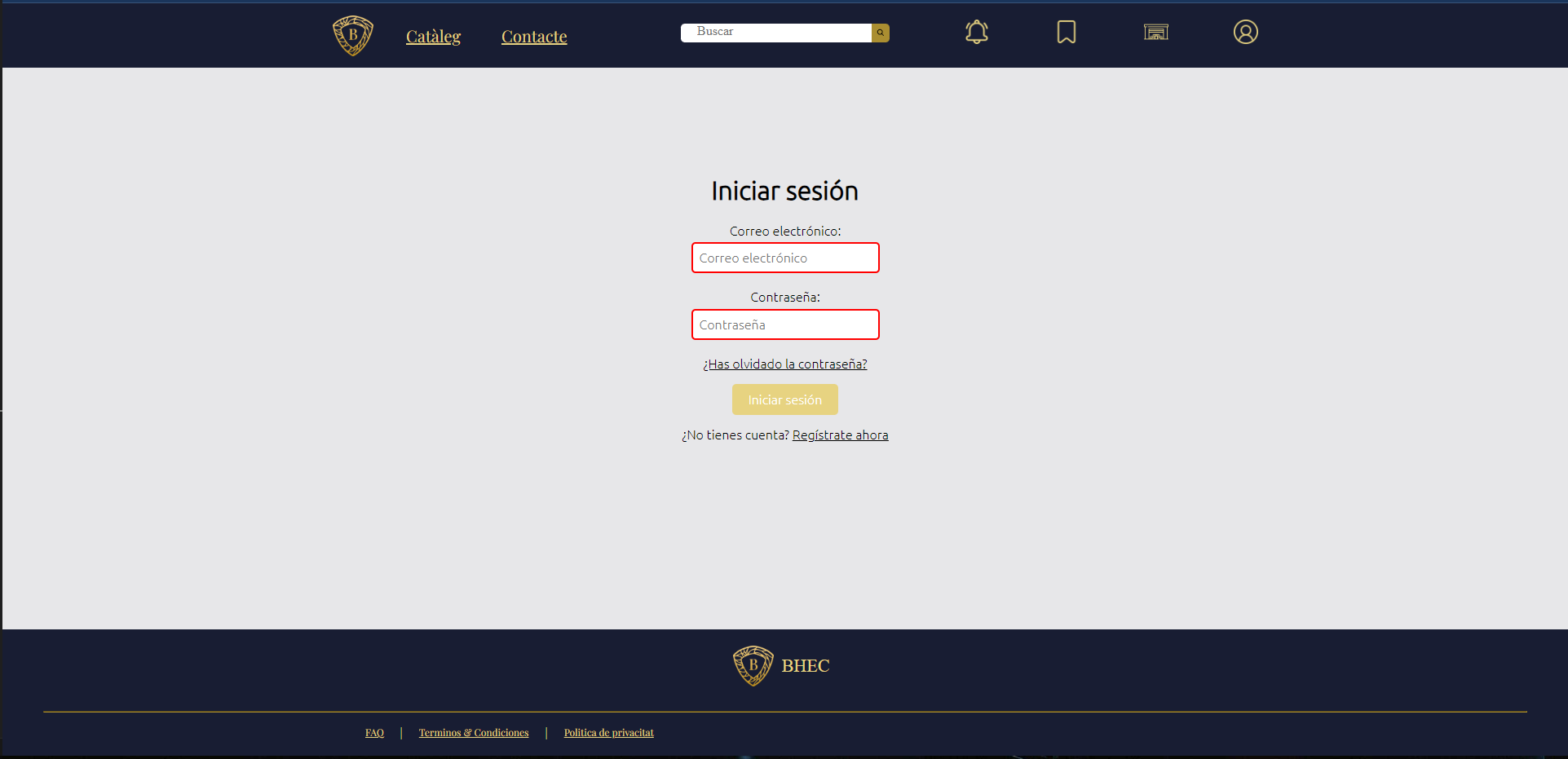
También cuando estás escribiendo el correo electrónico sale una sugerencia de dominios de correos electrónicos para que el usuario no tenga que escribir todo, simplemente lo tiene que seleccionar.



# Maquetación

## 5.1 Login

### 5.1.1 Escritorio



### 5.1.2 Móvil

## 

## 5.2 Domicilio

### 5.2.1 Escritorio

### 

### 5.2.2 Móvil

## 

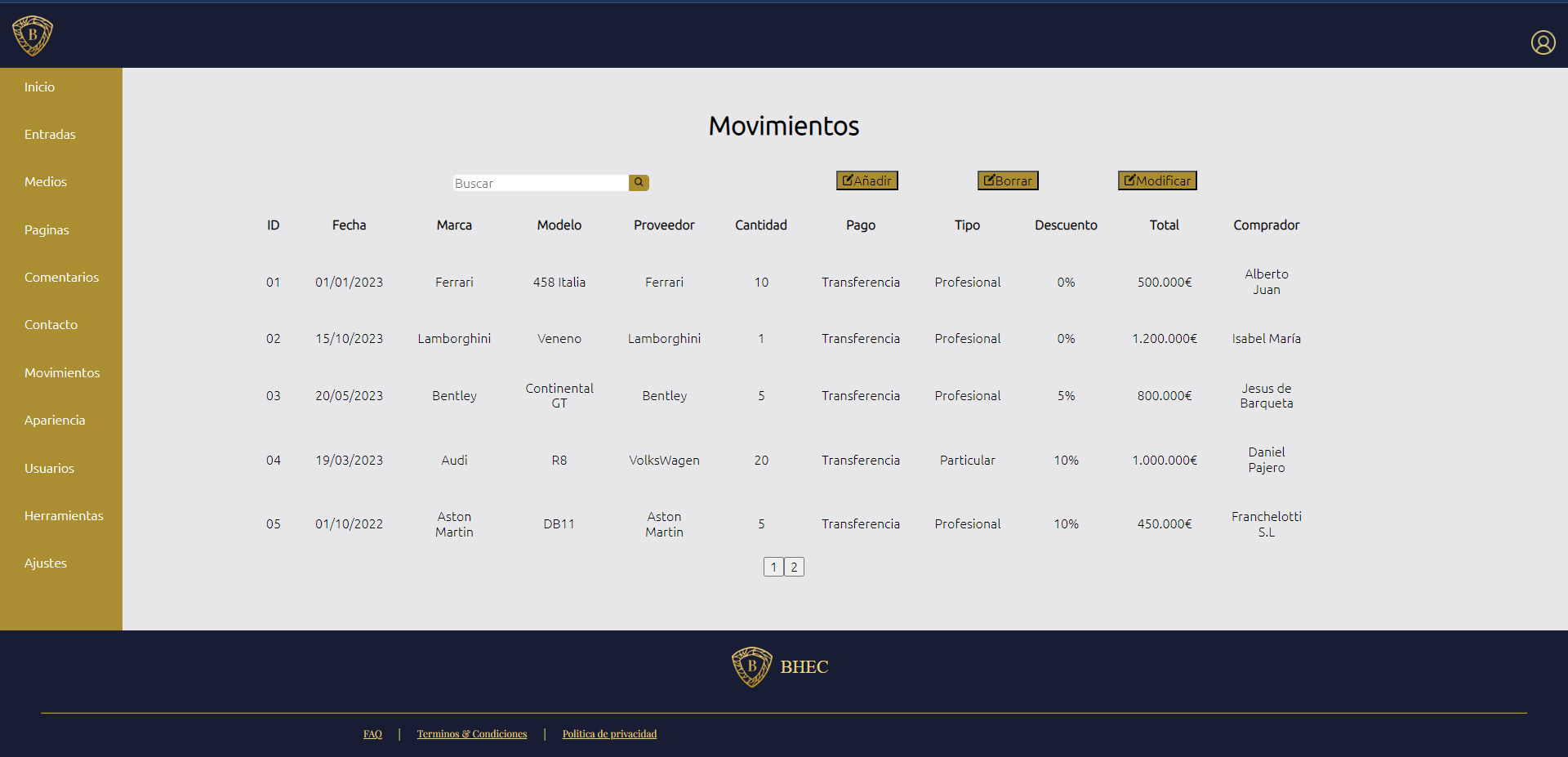
## 5.3 Compras

### 5.3.1 Escritorio

### 5.3.2 Móvil

## 5.4 Movimientos

### 5.4.1 Escritorio



### 5.4.2 Móvil



# 

# Conclusión

En este sprint hemos aprendido a utilizar Normalize.css para corregir bugs en las estilizaciones de los navegadores además de grid que nos permite estructurar nuestro código HTML muy eficazmente además hemos podido comprobar que añadiendo la etiqueta @ se puede crear reglas que se apliquen en el momento que se cumpla una o varias reglas.

En JavaScript hemos hecho validaciones utilizando estados en los elementos (:valid/:invalid), además hemos podido utilizar diferentes eventos (como el .submit que nos permite realizar el submit solo en caso de que los campos estén correctos) que nos han ayudado a hacer nuestra páginas más funcionales. Además hemos creado ventanas modales personalizadas que sólo aparecen como consecuencia de un evento y que permiten al usuario confirmar la acción que vaya a realizar o rechazarla si no está conforme.

Además en PHP hemos aprendido a hacer validaciones mediante funciones,varias formas de crear código seguro (Ya sea variables seguras, seguridad csrf, tokens de sesión, etc…), generar PDF mediante código PHP y acceder a la base de datos ya sea para consultar o modificar esta.

# Webgrafía

HTML/CSS

Iconos <https://www.ionos.es/digitalguide/paginas-web/diseno-web/como-se-pueden-integrar-icon-fonts/>

Modulos <https://mmteam.controldedominios.com/entradaBlog.asp?blog=3&cod=54>

Modal <https://www.silocreativo.com/crea-ventanas-modales-de-forma-sencilla-con-el-elemento-html/> 30-10-23

Icono flotante <https://codepen.io/MarioDesigns/pen/woJgeo> 19-10-23

Audio html <https://developer.mozilla.org/es/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content> 19-10-23

Expresiones regulares <https://regexr.com/> 17-10-23

Expresiones regulares <https://www.w3schools.com/tags/att_input_pattern.asp> 28-9-23

Popups <https://www.w3schools.com/howto/howto_js_popup.asp> 23-10-23

CSS estilos de “a” <https://www.adaweb.es/eliminar-subrayado-enlaces-css/> 23-10-23

Diferentes tipos de display:

<https://developer.mozilla.org/es/docs/Web/CSS/display> 31-10-23

JAVASCRIPT

Valores vacios <https://webplatform.github.io/docs/dom/ValidityState/valueMissing/> 23-10-23

Extracción de Valores de un string <https://www.php.net/manual/es/function.substr.php> 23-10-23

Validation API <https://www.w3schools.com/js/js_validation_api.asp> 23-10-23

Ejemplos de evento click <https://www.tutorialesprogramacionya.com/javascriptya/temarios/descripcion.php?cod=64&inicio=> 23-10-23

Eventos de teclado <https://learn.microsoft.com/es-es/dotnet/desktop/winforms/input-keyboard/events?view=netdesktop-7.0> 23-10-23

Eventos de mouse <https://es.javascript.info/mouse-events-basics> 23-10-23

Comprobar expresiones regulares: <https://regex101.com/> 1-11-23

Funcionamiento del :Valid :Invalid: \_

<https://developer.mozilla.org/es/docs/Web/CSS/:invalid> 2-11-23

Validaciones con JS:

<https://medium.com/stackanatomy/form-validation-using-javascripts-constraint-validation-api-fd4b70720288> 3-11-23

COOKIES

informacion cookies <https://clickage.es/implementar-politica-cookies-ecommerce/> 20-10-23

informacion cookies <https://vento.es/aviso-legal-politica-de-privacidad-y-cookies/> 20-10-23

PHP

informacion php <https://corriol.github.io> 02-10-23

metodos de validacion php <https://www.php.net/manual/es/filter.examples.validation.php> 18-10-23

metodos de validacion php <https://informaticapc.com/tutorial-php/validar-formulario.php> 18-10-23

toda la tarea <https://chat.openai.com> siempre

Seguridad csrf: <https://brightsec.com/blog/csrf-token/> 6-11-23

Creación de Clases (con sus constructores), getters, setters, conexión con BD, creación de excepciones y errores, relacions entre clases y creación de variables:

<https://corriol.github.io> 7-11-23

Mecanismos de Autenticacion <https://redtrust.com/mecanismos-autenticacion/> 25-10-23