2022

Paula Moreno Hermoso

2º DAW

4-12-2022

Ejercicios Objetos Ud. 3



ÍNDICE

1. Ejercicio 1 1

EJERCICIOS DE FORMS, OOP 1

QUIZ DE MYSQL 1

EJERCICIOS MYSQL 2

2. Ejercicio 2 3

3. Ejercicio 3 3

4. Ejercicio 4 4

PRIMERA KATA 4

SEGUNDA KATA 5

[5. PRÁCTICA BD: 5](#_Toc127852949)

[6. PRÁCTICA OBJETOS-BD: 9](#_Toc127852950)

[LEER ANTES 10](#_Toc127852951)

[Global.php 10](#_Toc127852957)

[Utils/Utils.php 10](#_Toc127852960)

[Product.php 14](#_Toc127852975)

[Controller/index\_product.php 18](#_Toc127852991)

[Add\_or\_edit\_product.php 20](#_Toc127853002)

[Details\_product.php 24](#_Toc127853028)

[View/index\_product.php 26](#_Toc127853040)

[Index.php](#_Toc127853028) 31

[En ejecución 32](#_Toc127853040)

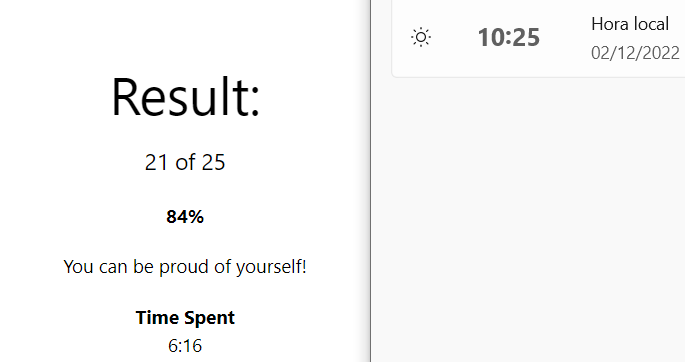
Puntuación (Los ejercicios tipo test y preguntas cortas valen la mitad que los ejercicios de programación, estos se valorarán utilizando la rúbrica de classroom). El alumno pondrá pantallazos de los ejercicios funcionando en este mismo fichero.

1. Ir a la web <https://www.w3schools.com/php/default.asp>  y realizar todos los apartados de los ejercicios de FORMS, OOP , el quiz de mysql <https://www.w3schools.com/mysql/mysql_quiz.asp> y los ejercicios <https://www.w3schools.com/mysql/exercise.asp>  , (pantallazos mostrando todas las tareas realizadas).

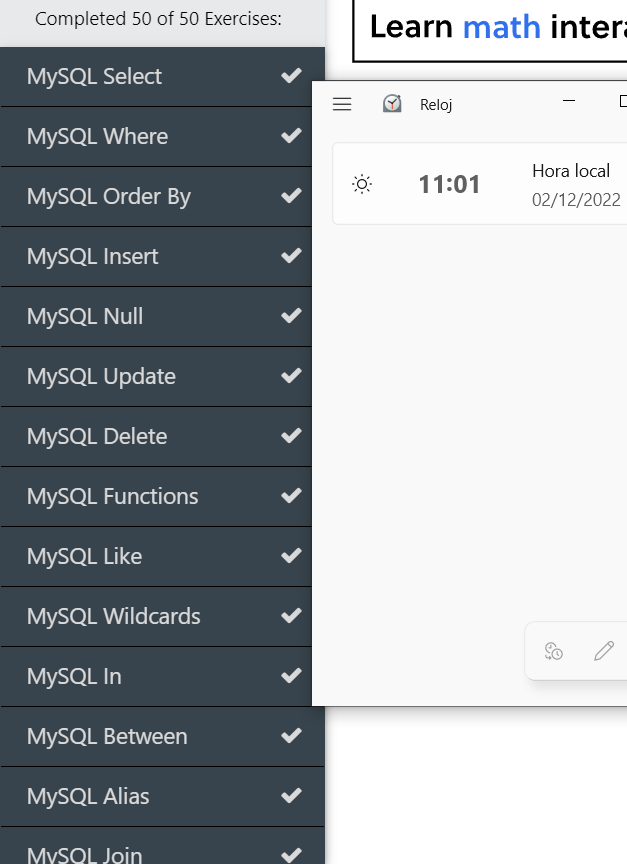
EJERCICIOS DE FORMS, OOP

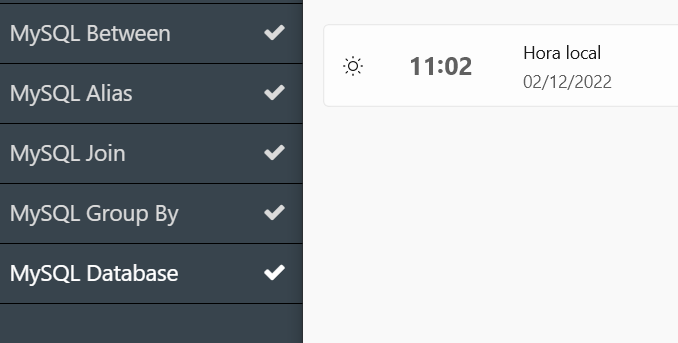


QUIZ DE MYSQL

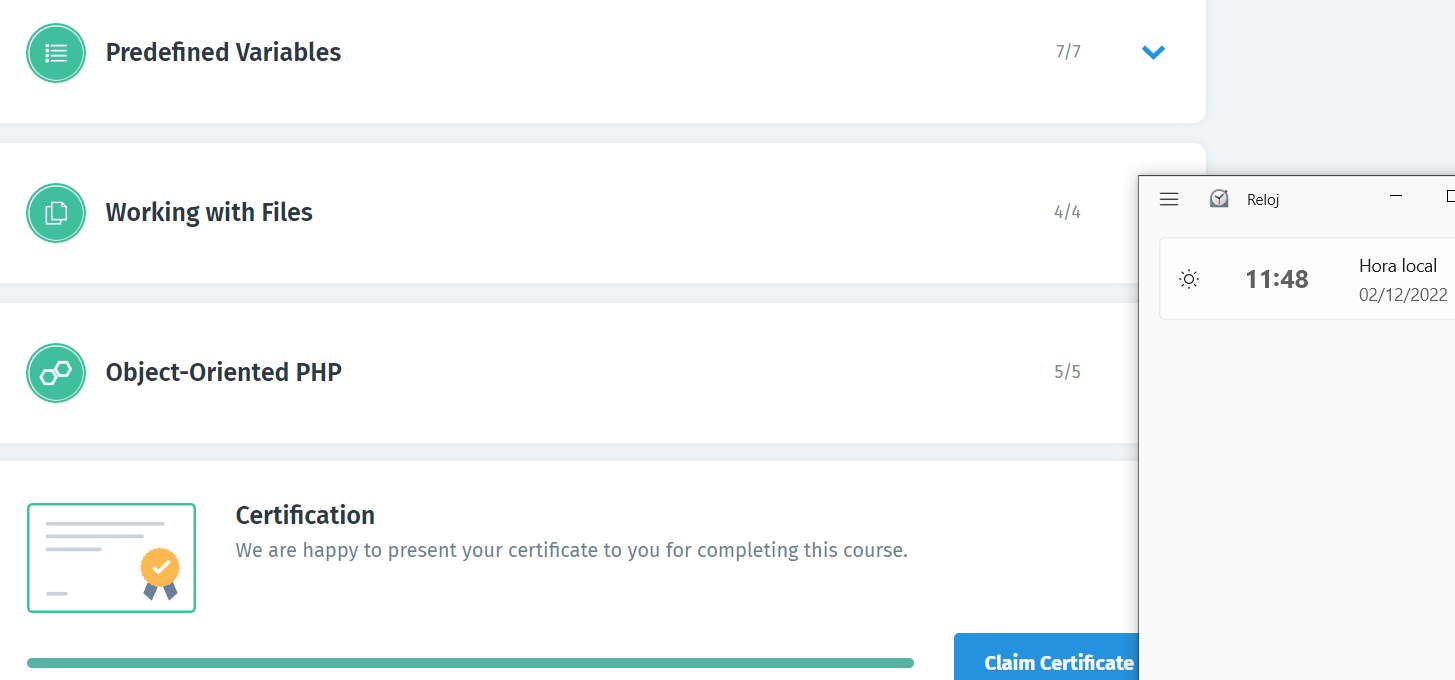


EJERCICIOS MYSQL

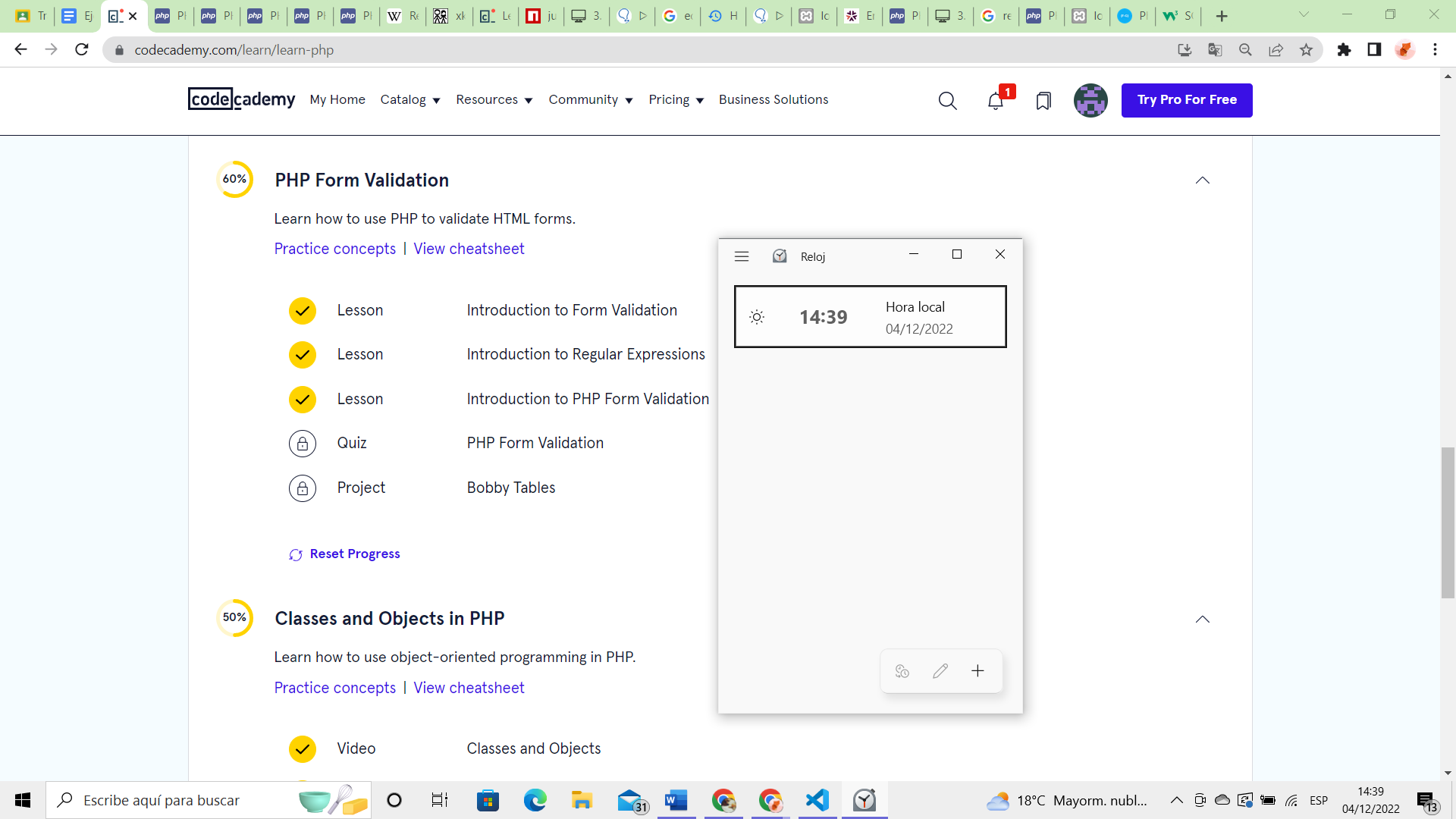


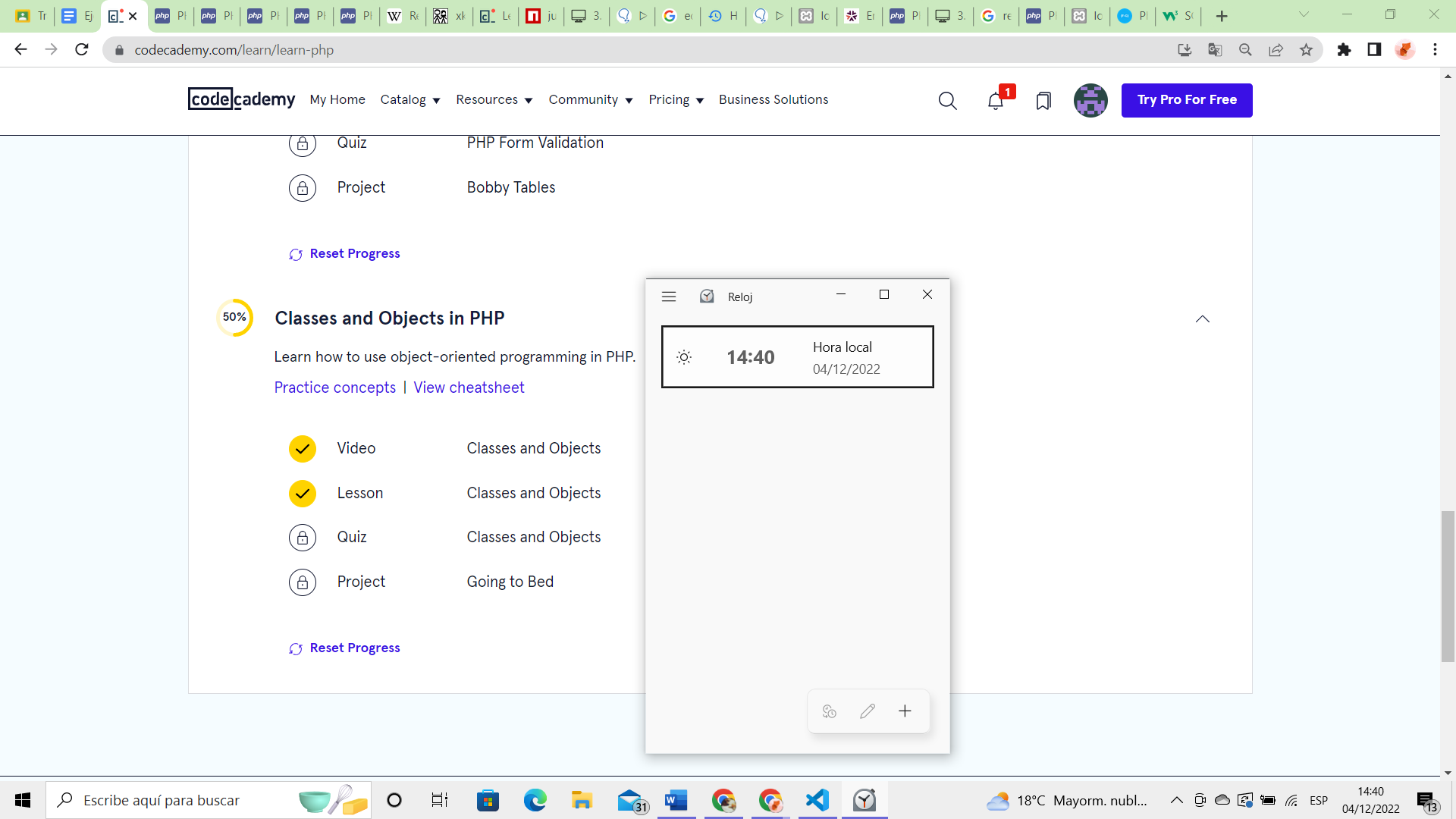


1. Realizar en <https://www.sololearn.com/learning/1059> los ejercicios de los bloques de predefined variables y de Object Oriented Php(pantallazos).



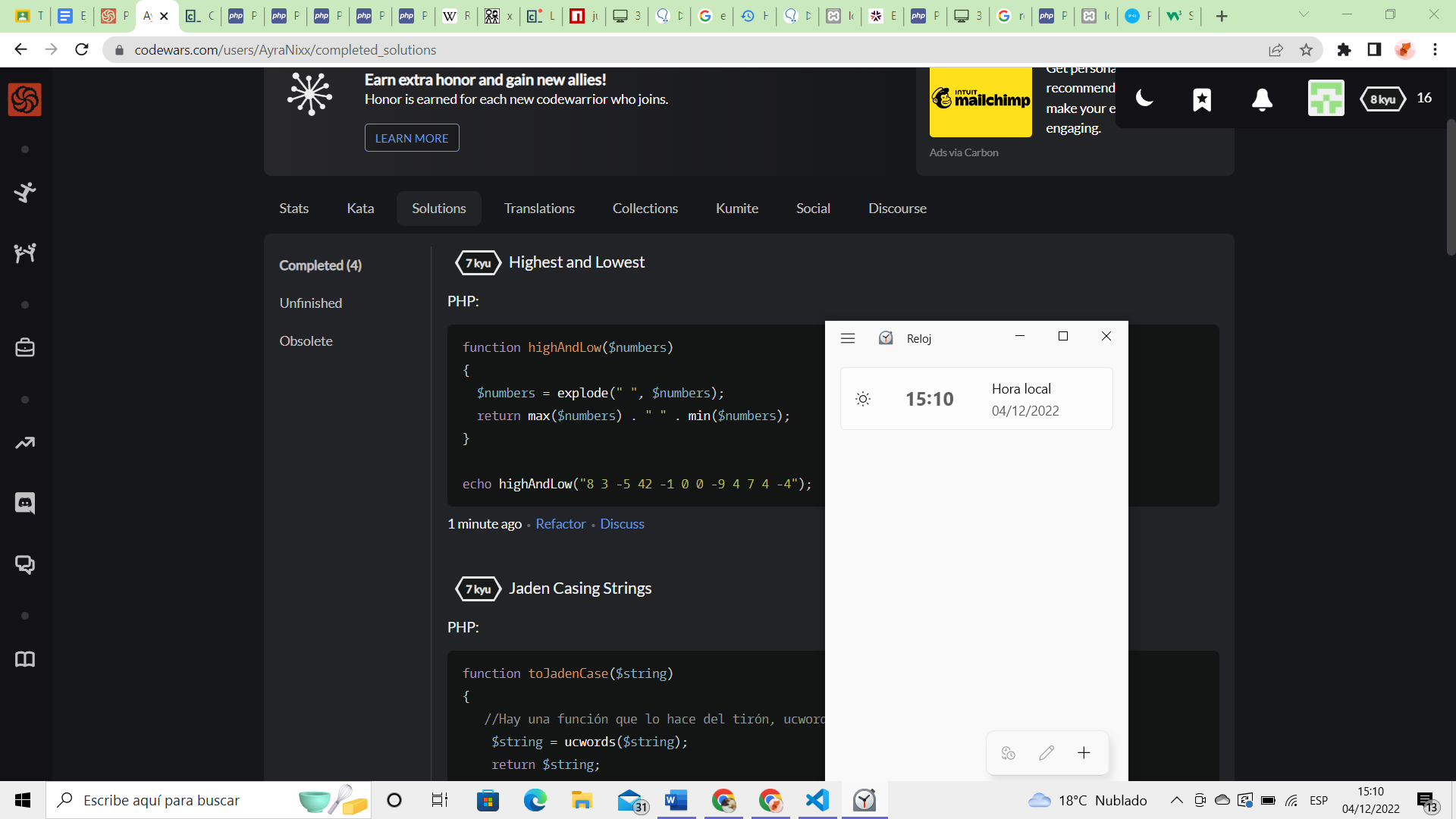
1. Realizar en <https://www.codecademy.com/courses/learn-php> todos los ejercicios posibles desde php form validations hasta classes and objects (Pantallazos).



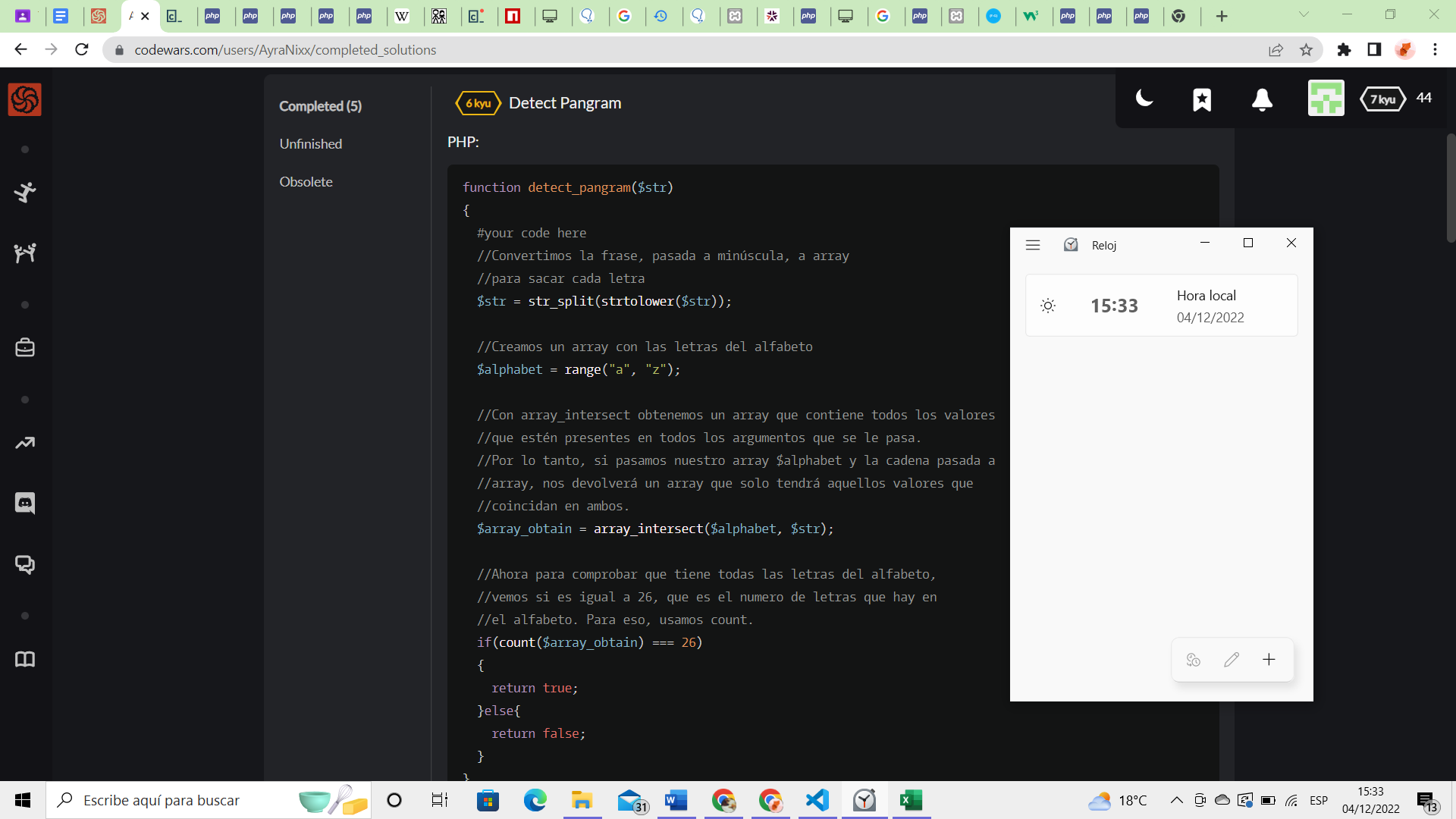


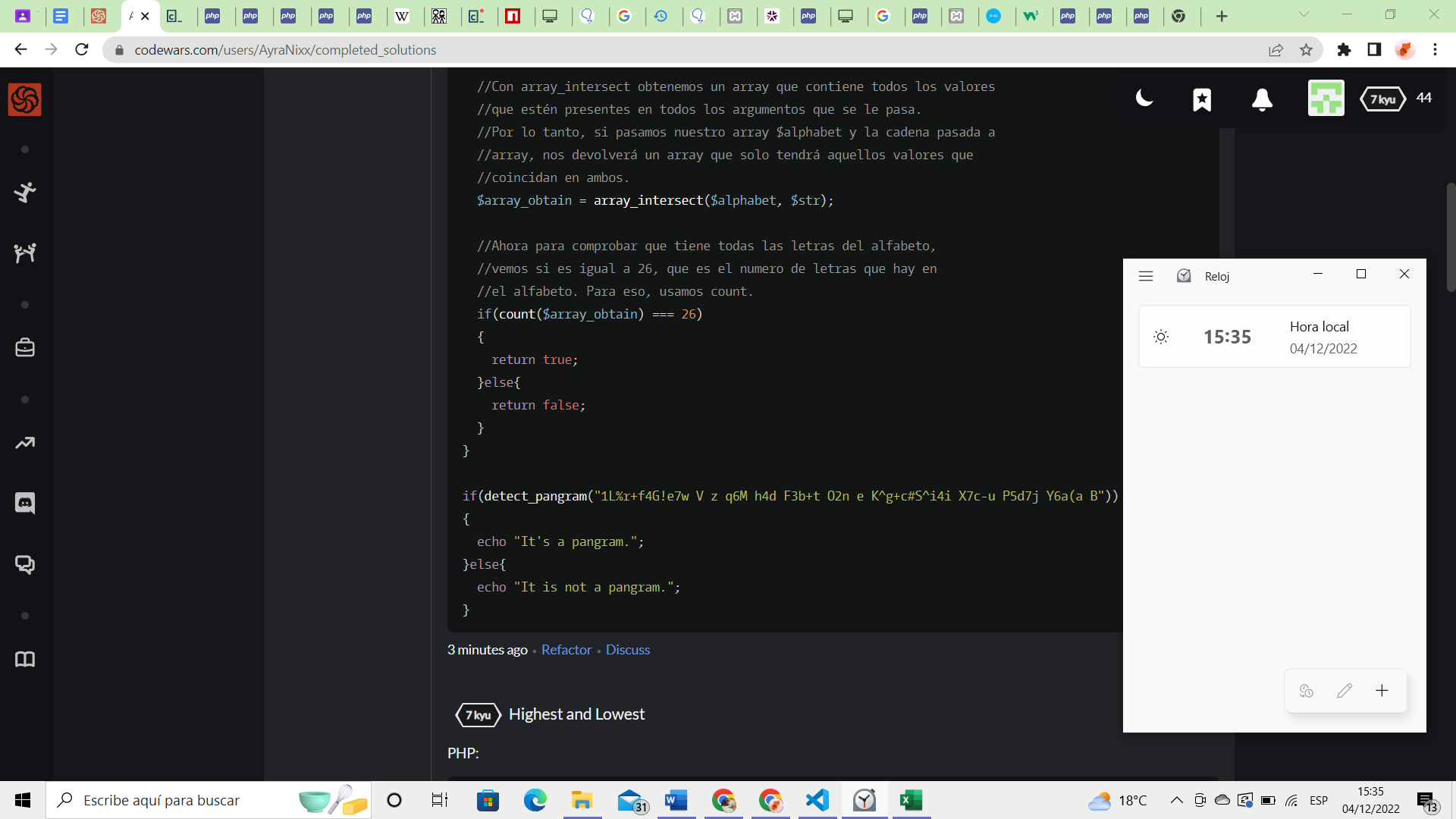
1. Realizar en <https://www.codewars.com/dashboard> 2 katas de nivel 7 o superior en php (pantallazos de vuestro perfil donde se vean dichos katas realizados).

PRIMERA KATA



SEGUNDA KATA





1. **PRÁCTICA BD**:

Realizar un diseño de una Base de datos de las siguientes características:

* Al menos tendrá 6 entidades, las cuales tendrán una clave primaria auto-incremental y al menos 4 campos excepto casos excepcionales.
* Deberá haber al menos dos relaciones de tipo n:m y otras dos 1:n.
* La temática será única para cada alumno. Tendrá que existir una entidad especial de la cual exista relaciones con otras entidades. La entidad principal tendrá un campo imagen(texto) que será el nombre de la imagen. (SUBIR NOTA: una entidad puede tener varias imágenes).

La base de datos realizada cuenta con las siguientes entidades:

* **Usuarios**. Posiblemente sea modificada durante el transcurso de la práctica de la Ud. 5. Consta de los siguientes atributos:
  + - **Id\_producto**. INT, AI.
    - **Nombre**. VARCHAR(100).
    - **Apellidos**. VARCHAR(200).
    - **Direccion**. VARCHAR(250).
    - **Telefono**. VARCHAR(11).
    - **Correo**. VARCHAR(100).
    - **Passwd**. VARCHAR(400).
    - **Salt**. VARCHAR(16).
    - **Foto**. VARCHAR(4000).
    - **Activo**. TINYINT.
    - **Cod\_activacion**. INT(5).
    - **Id\_rol**. INT.
* **Roles**. Formada por 3 atributos al no ser necesario añadir un cuarto: **id\_rol** *– INT, AI –*, **rol** *– VARCHAR(100) –*, **descripción** *– VARCHAR(300) –*.
* **Categoria**. Formada por 3 atributos al no ser necesario añadir un cuarto: **id\_categoria** *– INT, AI –*, **nombre** *– VARCHAR(100) –*, **descripción** *– VARCHAR(300) –*.
* **Pedidos**. Compuesta por 4 atributos: **id\_pedido** *– INT, AI –*, **fecha\_compra** *– DATE –*, **precio\_final** *– DECIMAL –,* ***id\_usuario*** *– INT –*.
* **Productos**. Atributos:
  + - **Id\_producto**. INT, AI.
    - **Nombre**. VARCHAR(100).
    - **Precio**. DECIMAL(10,2).
    - **Stock**. INT.
    - **Foto**. VARCHAR(4000).
    - **Categoría**. INT.
* **Proveedores**. Atributos:
  + - **Id\_proveedor**. INT, AI.
    - **Nombre**. VARCHAR(100).
    - **Direccion**. VARCHAR(250).
    - **Telefono**. VARCHAR(11).
    - **Correo**. VARCHAR(100).

Las relaciones que han surgido son las siguientes:

* **Productos - Proveedores**. N:M. Se crea una nueva tabla, **Detalles\_proveedores**.
* **Productos - Categoría**. 1:N.
* **Productos - Pedidos**. N:M. Se crea una nueva tabla, **Detalles\_pedido**.
* **Pedidos - Usuarios**. 1:N.
* **Usuarios - Roles**. 1:N.

A continuación, fotos del modelo entidad-relación:



1. **PRÁCTICA OBJETOS-BD**:

Realizar una web que separe el código utilizando el modelo vista controlador. Sí el código no está separado correctamente se tendrá que repetir. Desde la vista jamás se accederán a las funciones del modelo directamente, sino al controlador. Se deberá seguir la estructura de acceso a BD vista o una similar.

La web deberá cumplir los siguientes requisitos:

* Realizar una web que separe el código utilizando el modelo vista controlador. Sí el código no está separado correctamente se tendrá que repetir. Desde la vista jamás se accederán a las funciones del modelo directamente, sino al controlador. Se deberá seguir la estructura de acceso a BD vista o una similar.
* La web deberá cumplir los siguientes requisitos:
* Jamás se utilizarán variables directamente en las query, siempre se utilizará preparedStatements.
* Los datos de conexión estarán separados del código php
* Se crearán librerías para la conexión y manipulación de al menos 4 entidades de la BD. (Operaciones de borrado, inserción, modificación y consulta Paginada)
* La estética de la web utilizará elementos de Bootstrap
* Los datos del formulario se validarán que son del tipo y rango correspondiente y se les eliminarán todos los caracteres que puedan permitir inyección de código( mirar https://www.w3schools.com/php/php\_form\_validation.asp )
* Componentes:
* **Página Principal**: Mostrará un listado en tabla con los elementos de la entidad principal con paginación de forma adecuada. Cada elemento tendrá en su fila un botón para borrarlo y otro para modificarlo(deben de funcionar), El nombre será un enlace que abrirá un detalle del elemento al pulsarlo, en la misma página con ajax (subir nota) o en otra página distinta.. En la parte superior derecha mostrará un botón de “añadir … “.
* **Página Detalle**: Mostrará campos que no se muestran en la página principal, su foto y un listado con los datos de una de las entidades relacionadas. (por ejemplo, un coche al darle a detalle, un listado de concesionarios donde está disponible)
* **Página Modificación**: Al pulsar el botón modificación se mostrará un formulario dónde se cargarán los datos de la entidad correspondiente y se podrán modificar.
* **Página Inserción**: Igual que la anterior, sólo que aparecerán los campos vacíos y en lugar de modificar un registro en BD, se insertará uno nuevo.

LEER ANTES

Las entidades seleccionadas para la realización de esta práctica son: Roles, Categoría, Productos y Proveedores.

He intentado realizar un único controlador para cada entidad para intentar reducir el número de archivos, por lo que sé que el código será un caos.

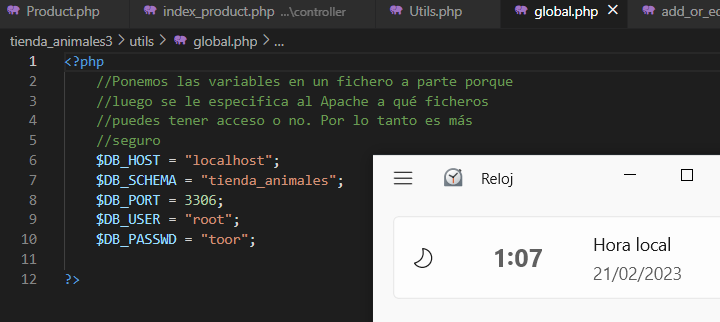
Mi página de Modificación y de Inserción es la misma, dependiendo de la opción elegida se hará una cosa u otra.

Al ser muchos archivos, explicaré la entidad que cuente con más detalles, como, por ejemplo, Productos en la que se sube la url de una imagen a la base de datos.

Empezaremos con global.php.

Global.php

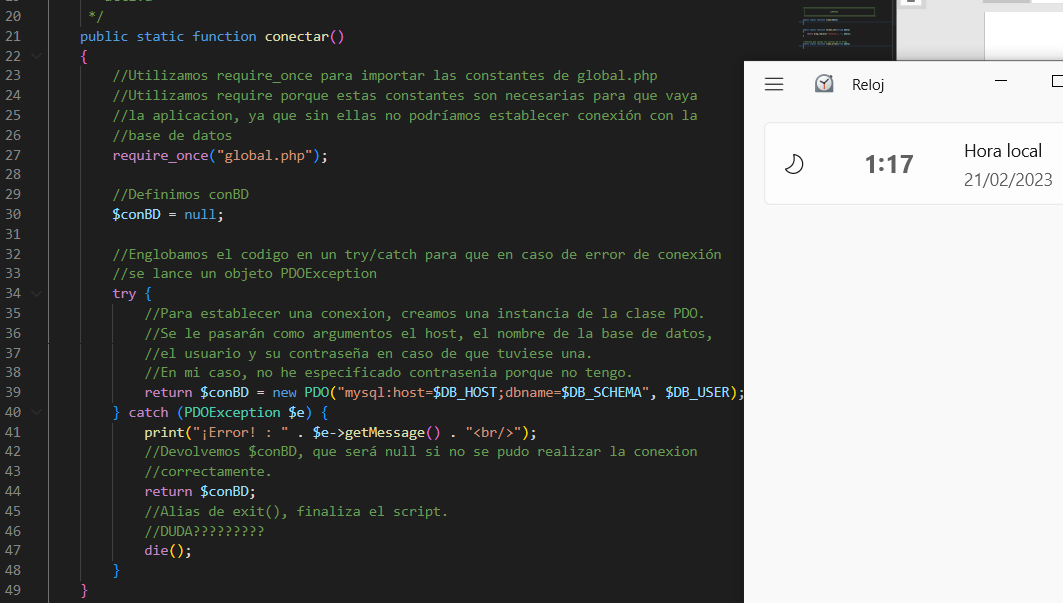
En este archivo guardaremos el nombre del host, el esquema, puerto, usuario y contraseña necesarios para el acceso a la base de datos.



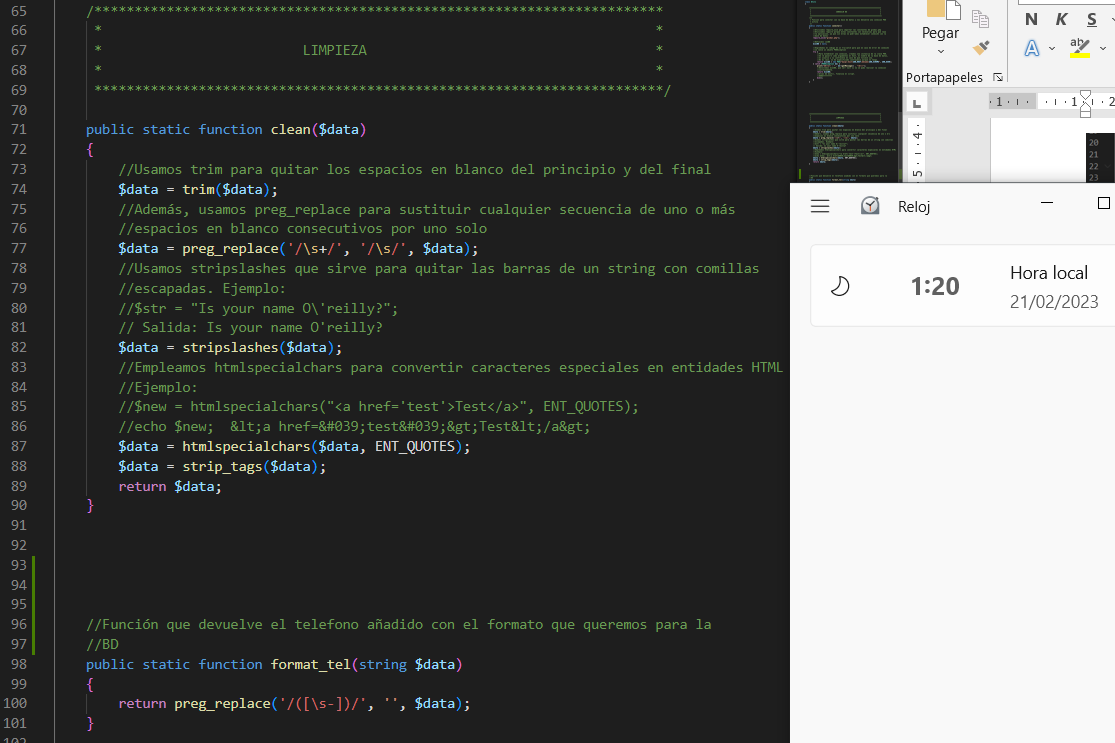
Utils/Utils.php

En este fichero, crearemos una clase denominada Utils en la que añadiremos funciones que nos puedan ser útiles en el futuro, por ejemplo, generar un SALT o guardar una imagen en un fichero.

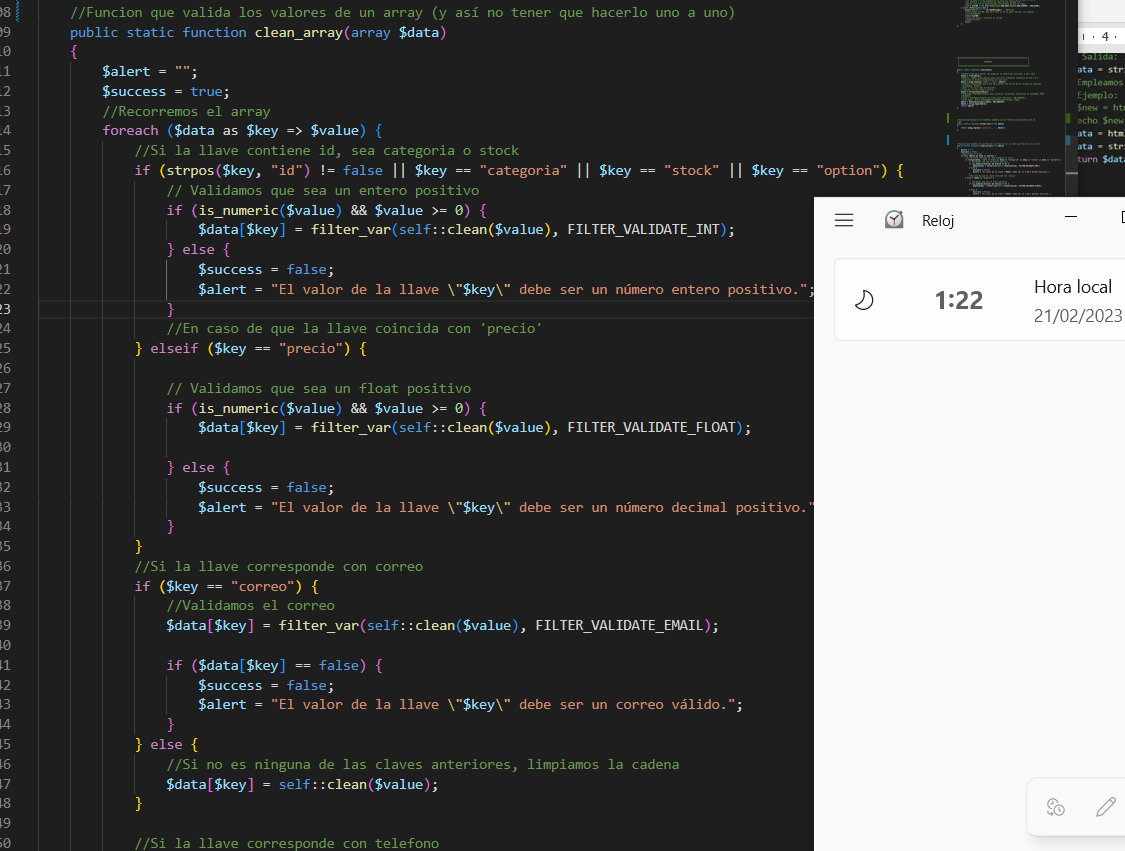
El primer método que crearemos es **conectar()** que será público y estático para que se pueda acceder a él sin tener que declarar un objeto de la clase Utils.

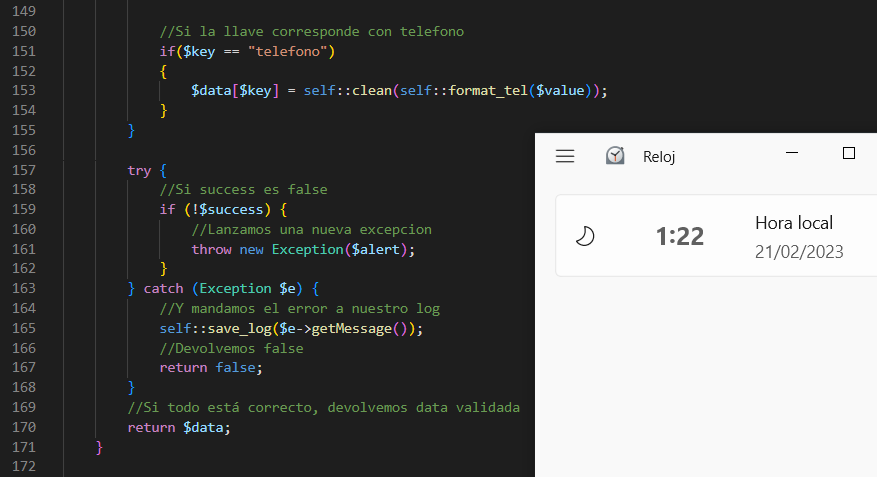


Seguidamente, crearemos tres métodos para la limpieza y validación de datos:



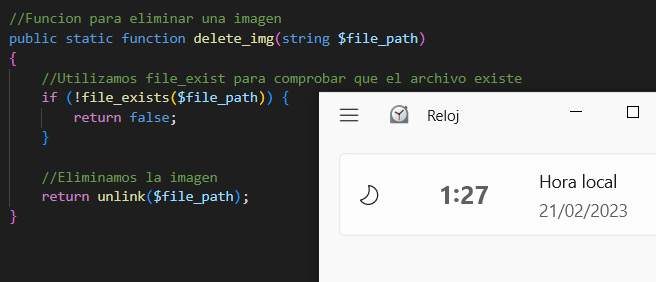
(La función clean\_array se creó para limpiar y validar los datos pasados por un array y así evitar tener que hacerlos uno a uno)



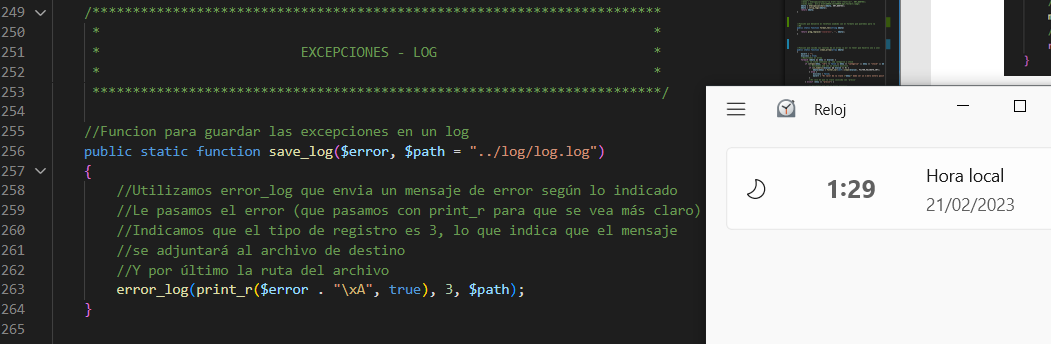


Ahora, vienen dos funciones una que devolverá la ruta de la imagen que se quiere subir y otra para eliminar la imagen indicada.





Por último, tenemos una función que guardará los errores generados en un log.



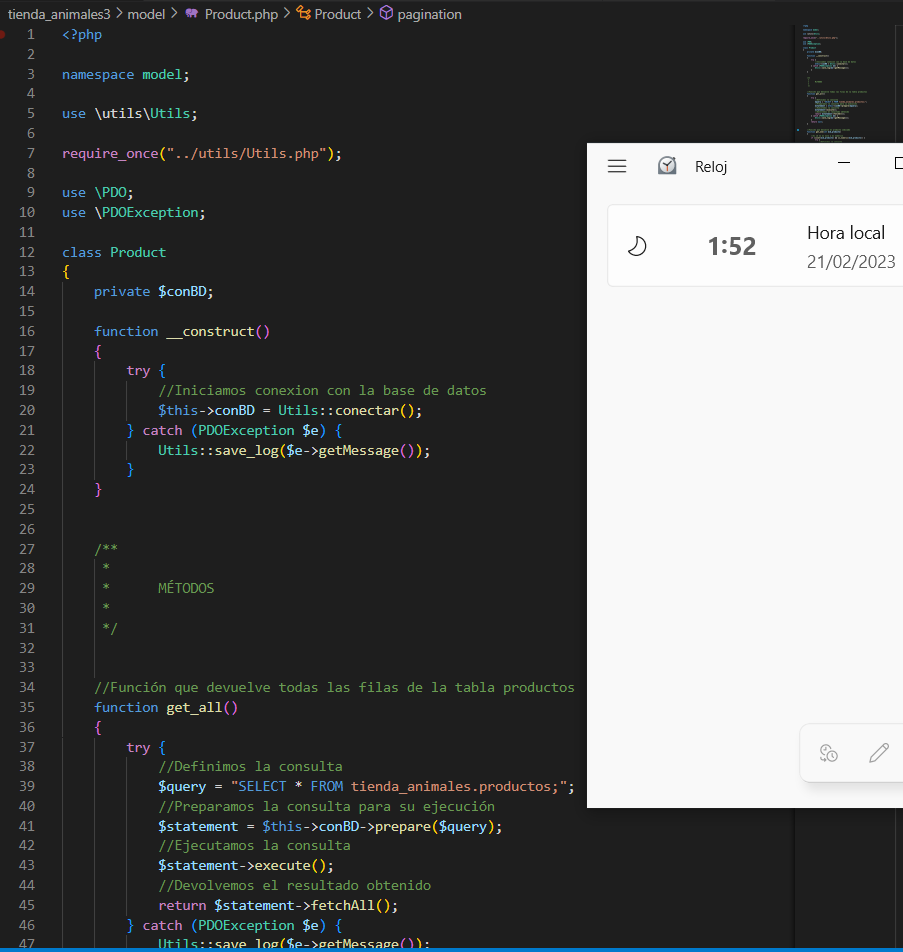
Más adelante, con la práctica de la Ud. 5 se añadirán más métodos.

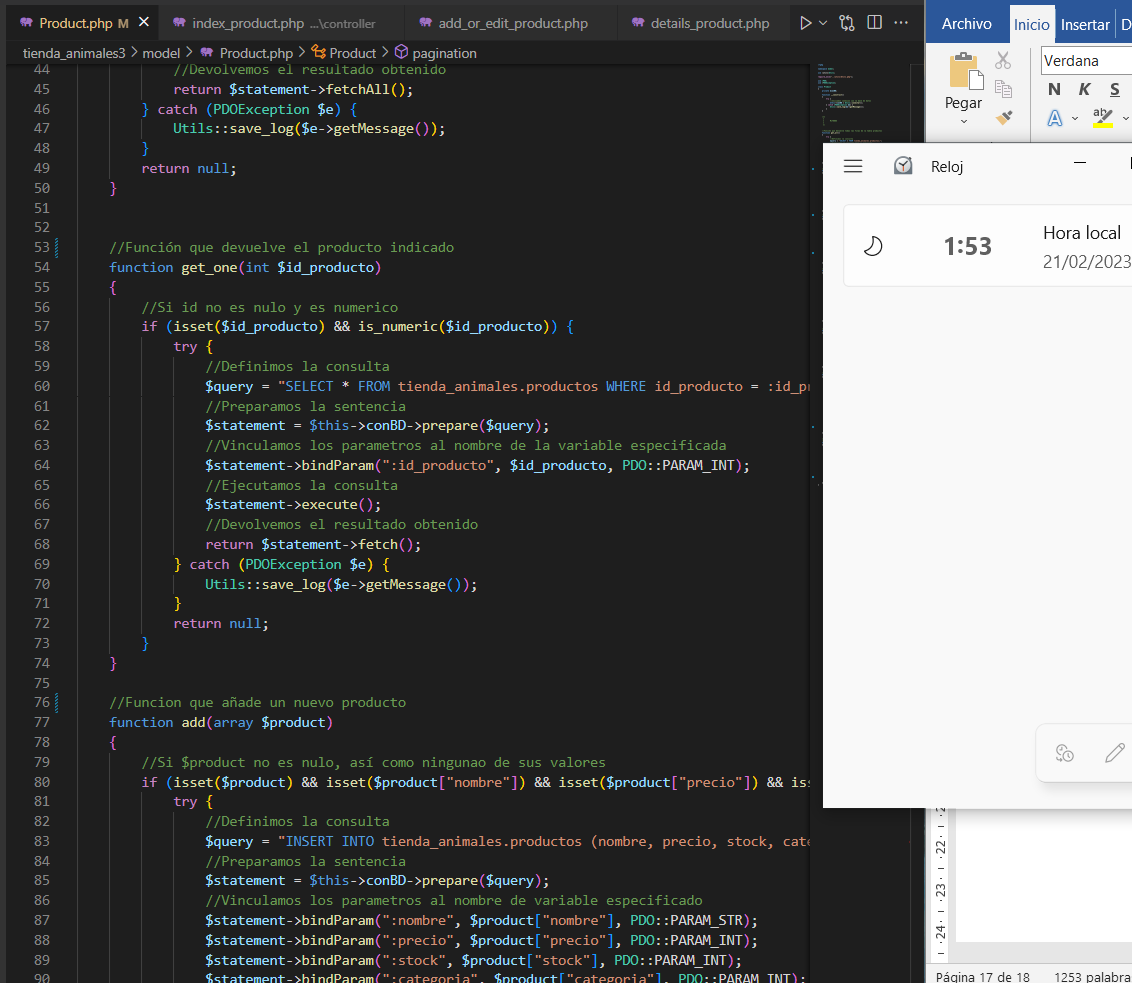
Product.php

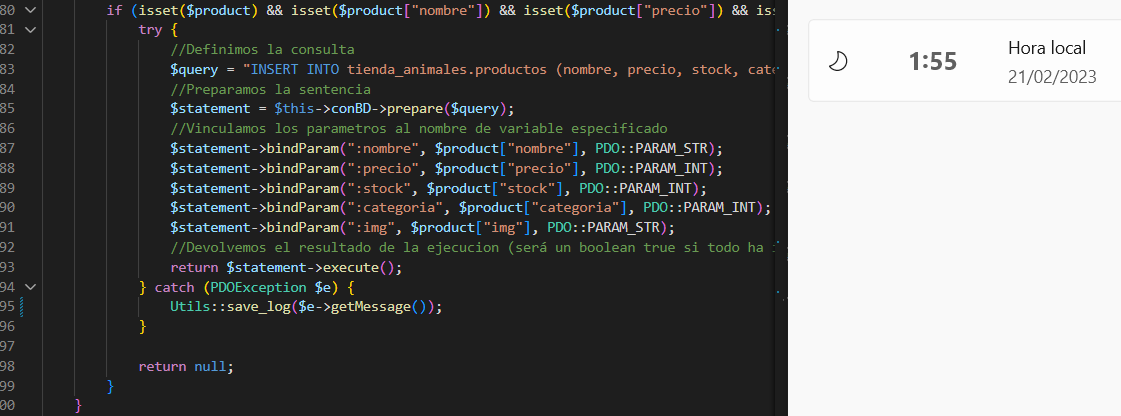
En este archivo crearemos una clase llamada Product (se hará lo mismo para las otras 3 entidades), que tendrá un atributo privado llamado $conBD para la conexión con la base de datos, que se inicializará en el constructor.

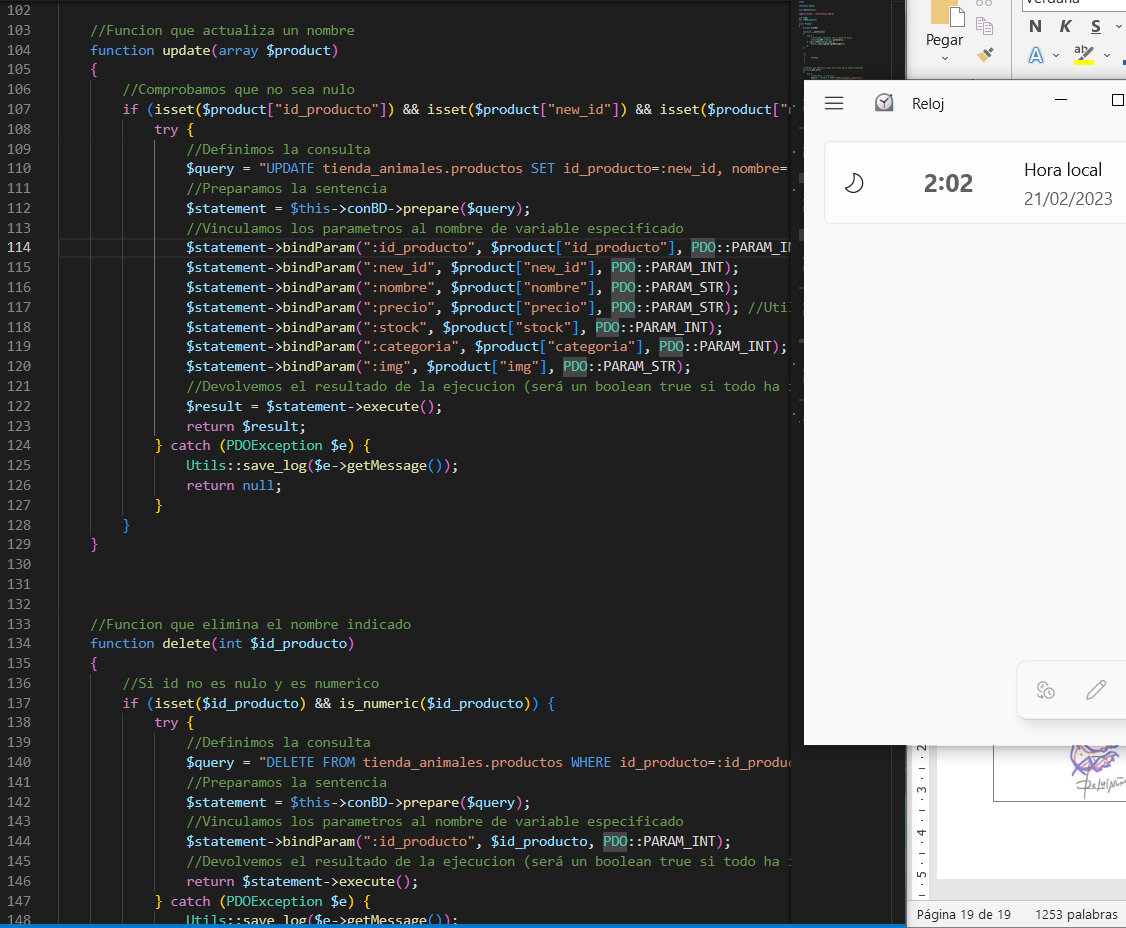
En esta clase crearemos varias funciones:

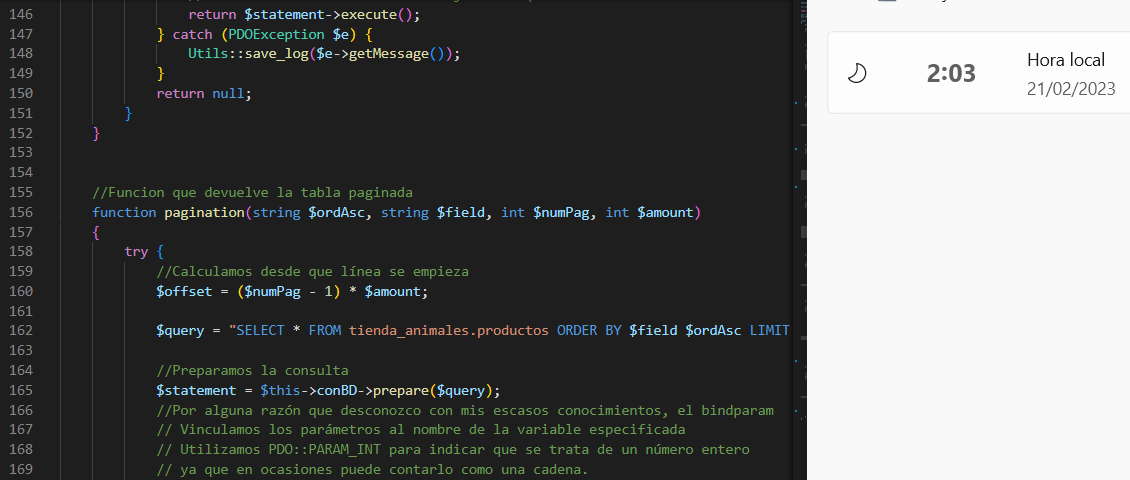
* **Get\_all**. Devuelve todas las filas de la tabla productos.
* **Get\_one**. Devuelve el product indicado.
* **Add**. Añade un nuevo producto.
* **Upadete**. Modifica un producto.
* **Delete**. Elimina un producto.
* **Paginación**. Devuelve los datos paginados.
* **Get\_total\_pages**. Devuelve el número de páginas que hay según la cantidad de elementos que se muestran en ella.

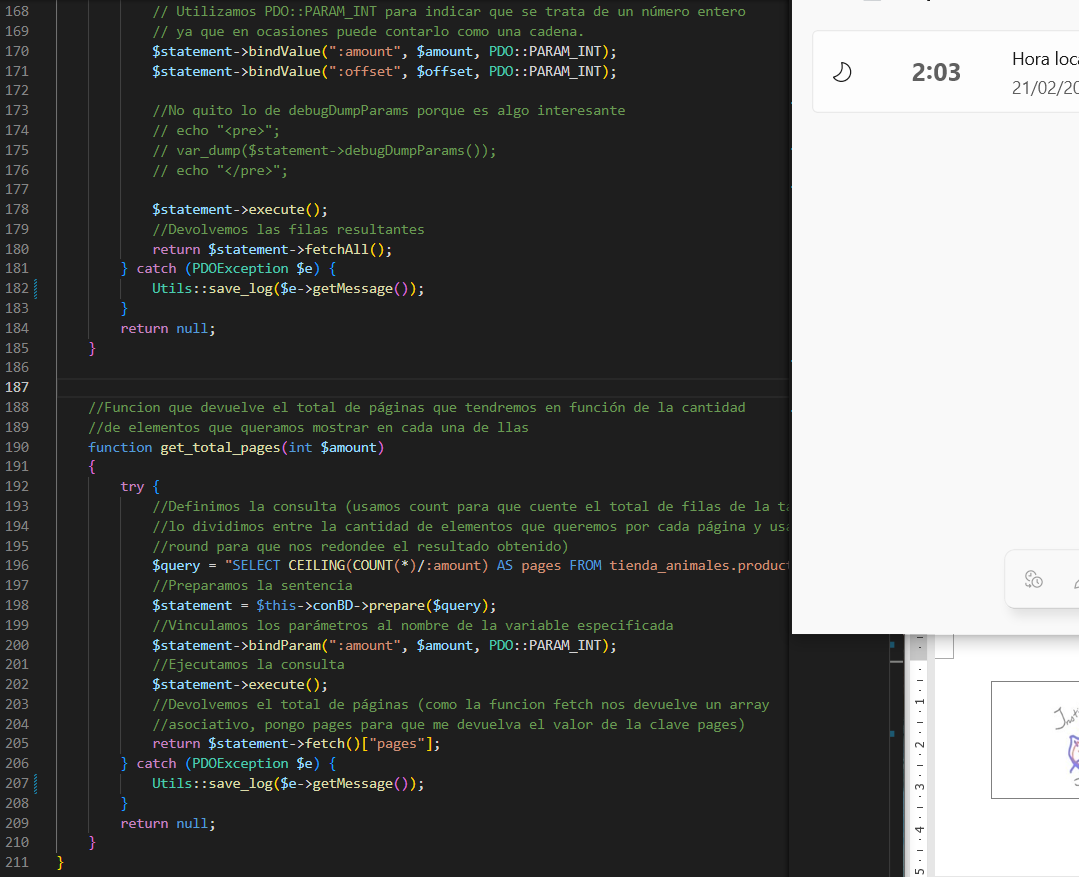








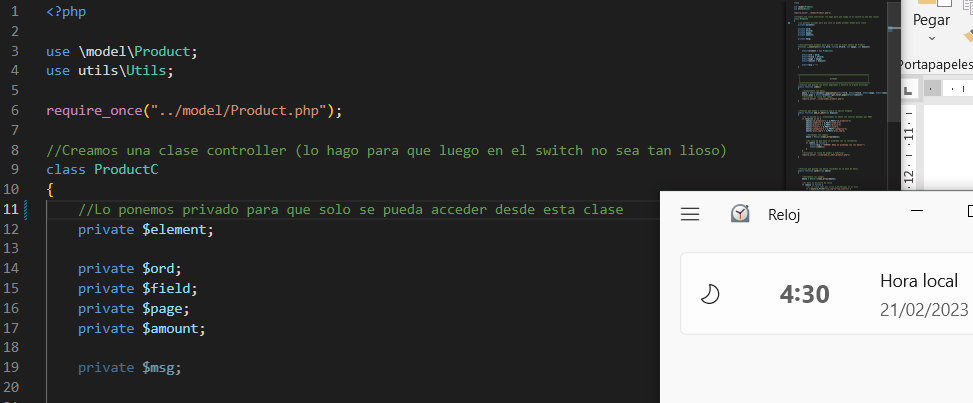


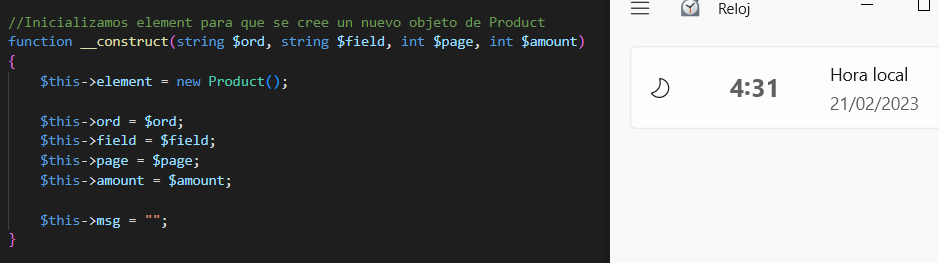


Controller/index\_product.php

Este archivo puede parecer lioso a simple vista, pero en realidad no lo es tanto.

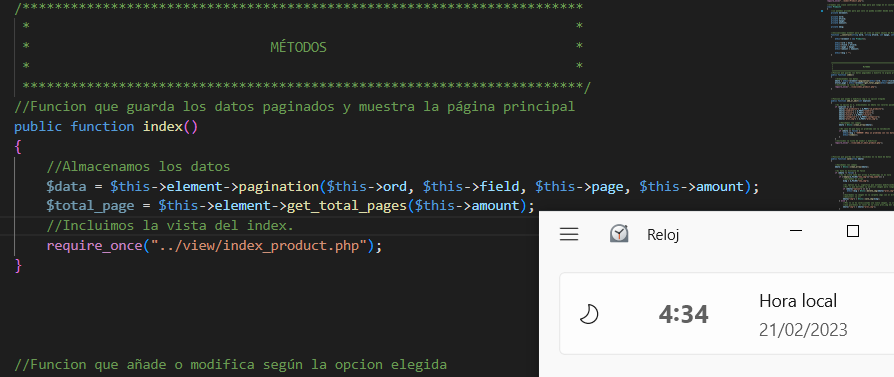
Creamos una clase llamada ProductC – *Controlador para Producto* –. En esta clase, definimos una variable llamada **$element** que inicializaremos en el constructor para crear un nuevo objeto de la clase **Product**, otros cuatros que corresponden al campo por el que queremos ordenar los datos, el tipo de ordenación, la página y la cantidad de elementos que queremos ver por cada página y que al crear un nuevo objeto de la clase **ProductC** se pasarán como argumentos y uno para almacenar el mensaje resultante.





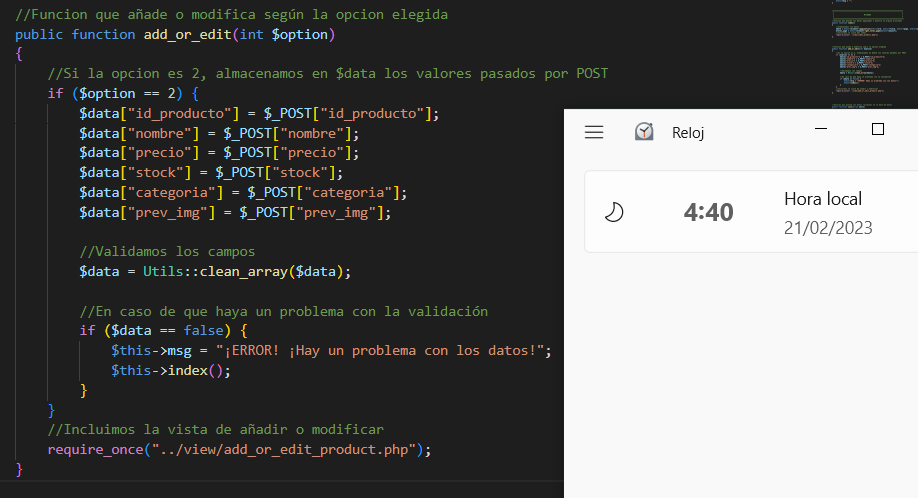
Tras esto, crearemos varios métodos.

La primera función es **index()** que nos llevará a la página principal de Producto, en la que se verán los datos paginados.



Luego, tenemos la función **add\_or\_edit**() que, realizará una acción u otra dependiendo de la opción elegida. Si queremos modificar un elemento preexistente, almacenamos los datos pasados por **POST** y validamos y limpiamos los datos con la función **clean\_array**() de **Utils**.

Si el resultado es false, nos saltará un mensaje de error y volveremos al índice. Si todo va bien, vamos al archivo **add\_or\_edit\_product.**php que se encuentra en **view**.



Como creo que es necesario ver el archivo add\_or\_edit\_product.php para continuar con la explicación de la siguiente función, lo mostraremos ahora mismo.

Este fichero, corresponde a la vista de Modificación e Inserción.

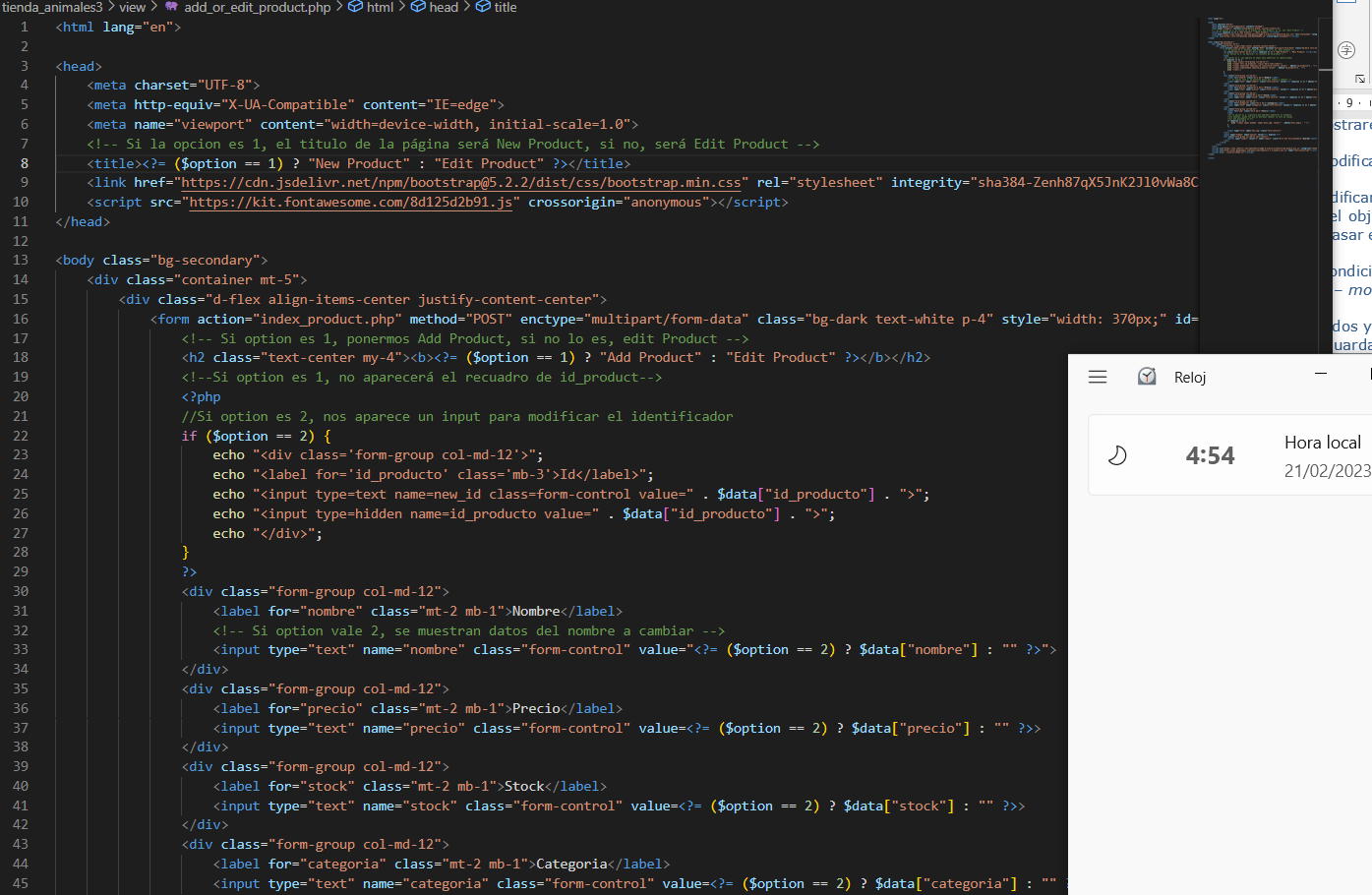
En caso de que la opción elegida sea modificar un elemento preexistente, **$option** valdrá 2 y se mostrará un input con el id\_producto del objeto seleccionado por si deseamos modificarlo. En caso de actualizar su id, deberemos de pasar el valor antiguo – *hidden* – y el nuevo por POST.

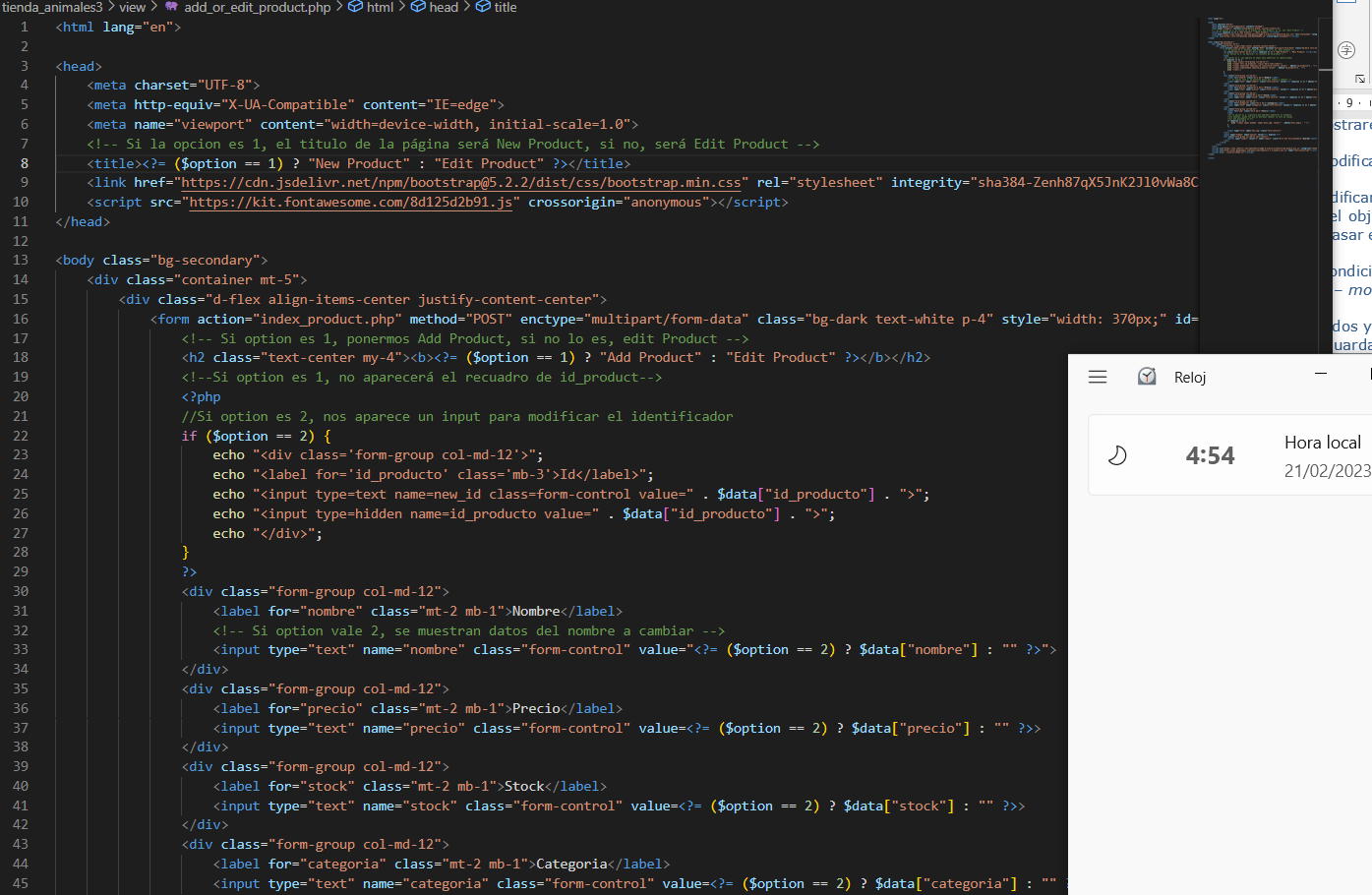
En el resto de inputs, realizaremos un condicional ternario para establecer el valor del input en función de si la opción seleccionada es 2 – *modificar* – o 1 – *añadir* –.

Por último, en caso de que la opción sea dos y tendremos que guardar la url previa y pasarla por **POST** por si no cambiamos la imagen guardada por otra. Por lo tanto, en caso de seleccionar modificar, pasaremos la ruta anterior con un **input hidden** y la nueva ruta con un **input file**.

Si nos fijamos, el botón submit tiene un valor 5, lo que nos indica que se debe de llamar a la función **save()** que veremos a continuación.

Y es muy importante que en el formulario pongas **enctype=”multipart/form-data”** para indicar que vas a pasar un archivo.







Volvemos de nuevo a **controller/index\_product.php** para continuar explicando la función **save()** que se llamará al enviar el formulario anterior.

A este método, se le pasará como argumento un array que validaremos y limpiaremos con la función **clean\_array()** de la clase **Utils**.

Si el resultado es **false**, nos dirigirá a la **página principal** de **Productos** y nos saltará una advertencia.

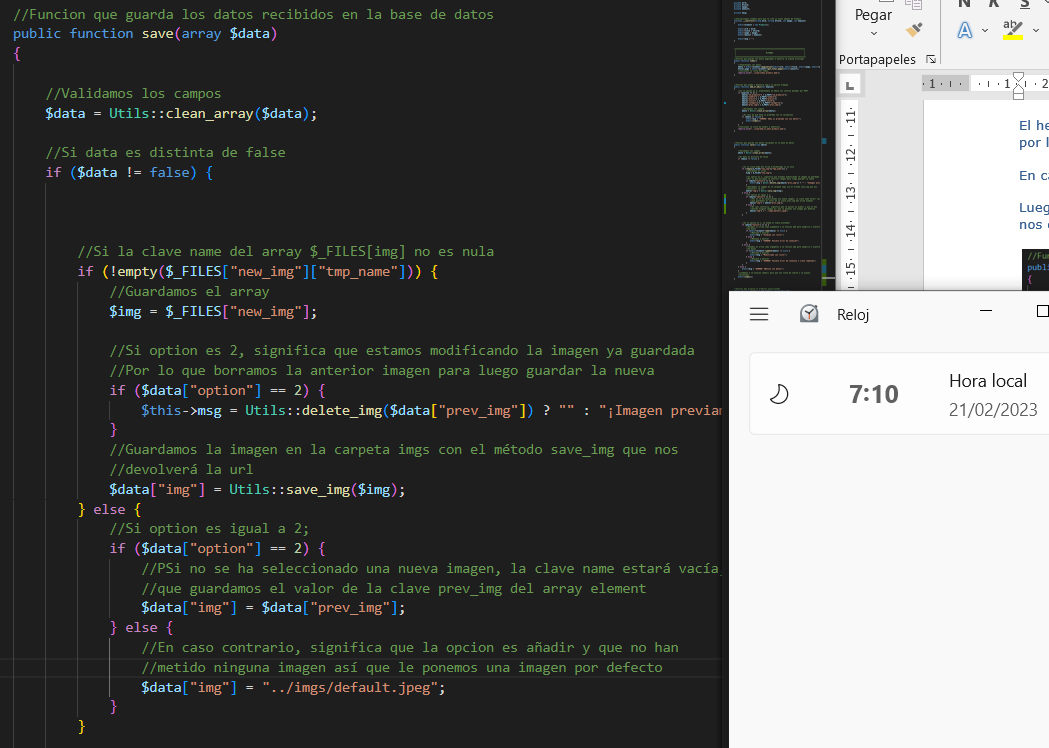
En caso de que no lo sea, se comprobará que **$\_FILES[“new\_img”][“tmp\_name”]** no esté vacío. Si está vacío, significa que la imagen no ha sido modificada (si fuese la opción de añadir, no debería de estar vacío, pero si lo está ponemos la ruta “../imgs/default.jpeg” para una imagen por defecto) y por lo tanto, seguimos almacenando la ruta anterior.

Si no lo está, alamacenamos el array $\_FILES[“new\_img”].

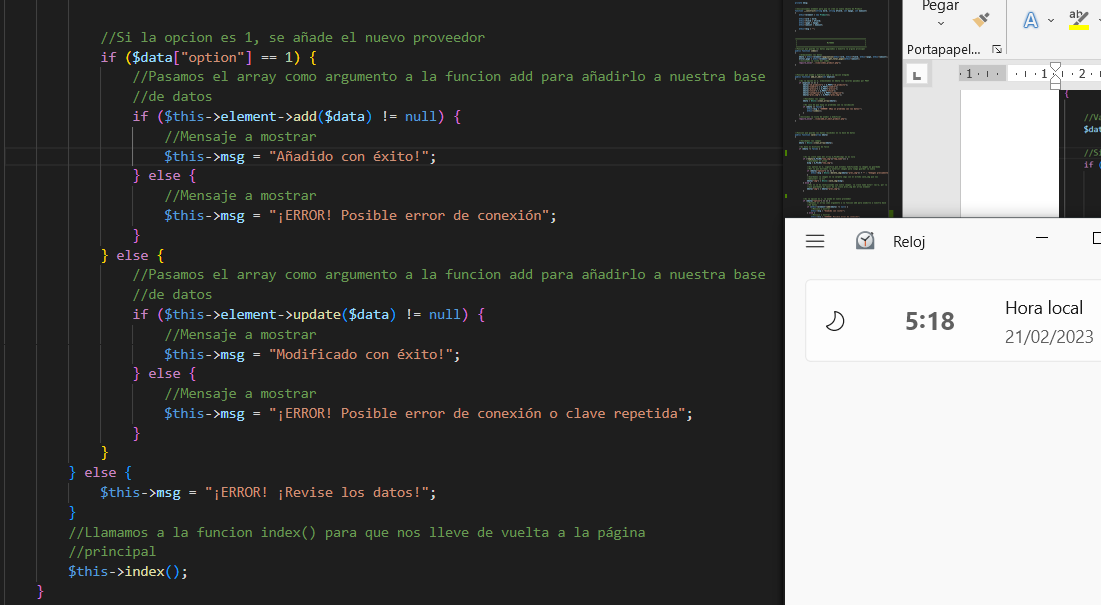
El hecho de que no esté vacío puede significar que o bien queremos añadir un nuevo producto y por lo tanto subir la imagen de este o bien cambiar la imagen de un elemento ya existente.

En caso de que se tratase de una modificación, eliminaríamos la imagen anterior.

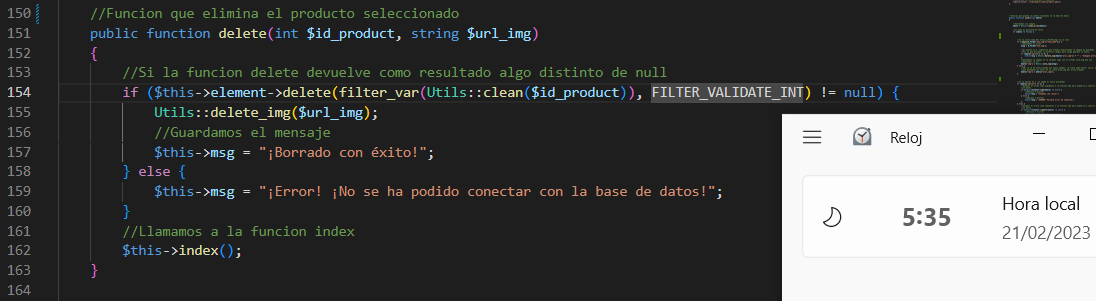
Luego, tanto si queremos añadir o modificar, guardamos la imagen con Utils::save\_img() que nos devolverá la ruta de la imagen.



Luego de hacer esto, se llamará a la función **add** o **update** de la clase **Product** en función de si hemos seleccionado una opción u otra.

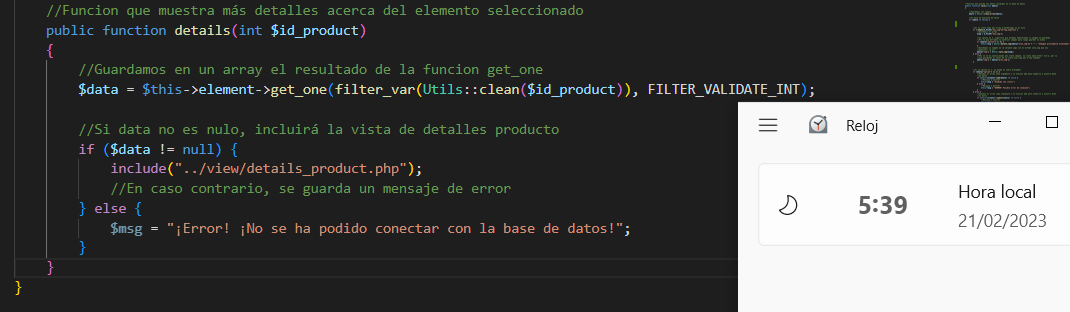


Tras esta función tenemos **delete()**, que eliminará el producto seleccionado.



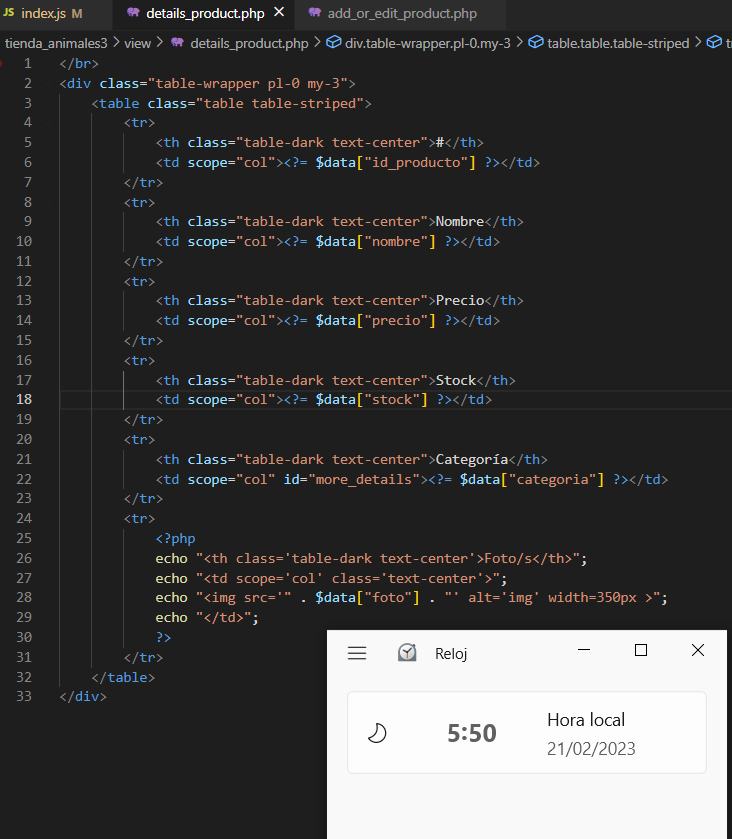
Y la función **details(),** que nos mostrará más detalles acerca del elemento seleccionado.

Esta función se llama con **AJAX** por lo que no se carga una nueva página y nos permitirá ver más detalles acerca del dato clicado. Más adelante, en este documento veremos el documento de JavaScript que hemos creado.



Al igual que con add\_or\_edit\_product.php, veo necesario mostrar el archivo details\_product.php ya que es lo que veremos cuando se llame a la función.

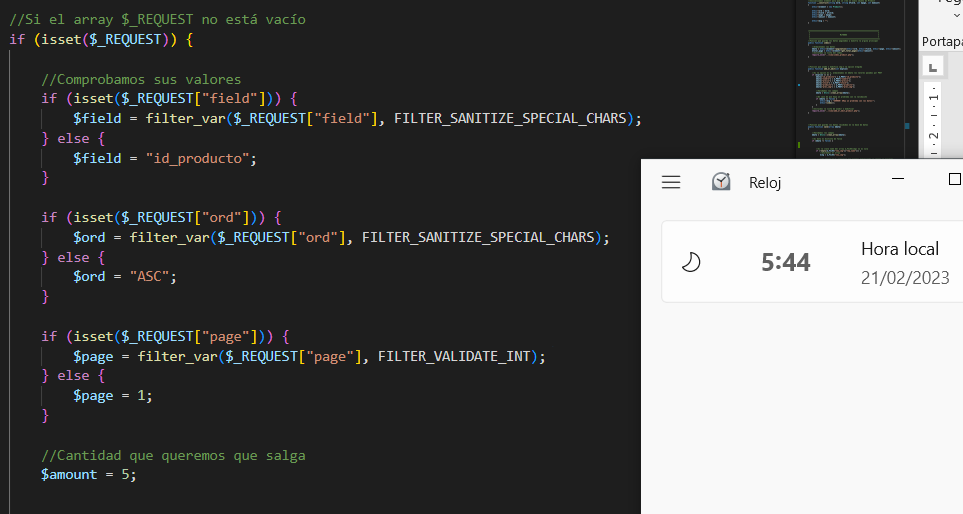
Es bastante simple, solo mostramos los valores de $data.

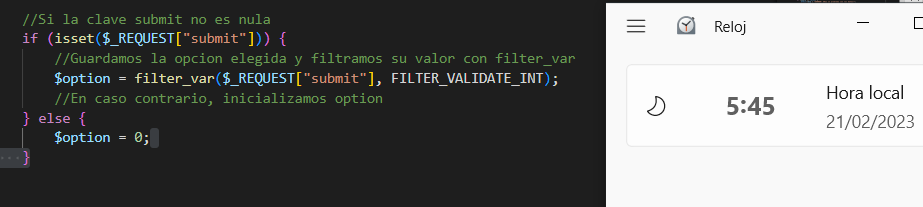


Con este último método, hemos terminado la clase **ProductC**.

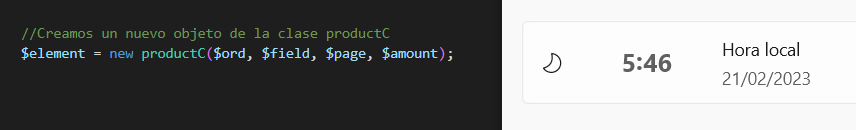
Ahora, fuera de la clase, comprobamos que **$\_REQUEST** no esté vacío con la función **isset()**.

Se comprueba que sus claves **field**, **ord**, **page y submit** no sean nulos y en caso de que así sea, se establece unos valores predeterminados.

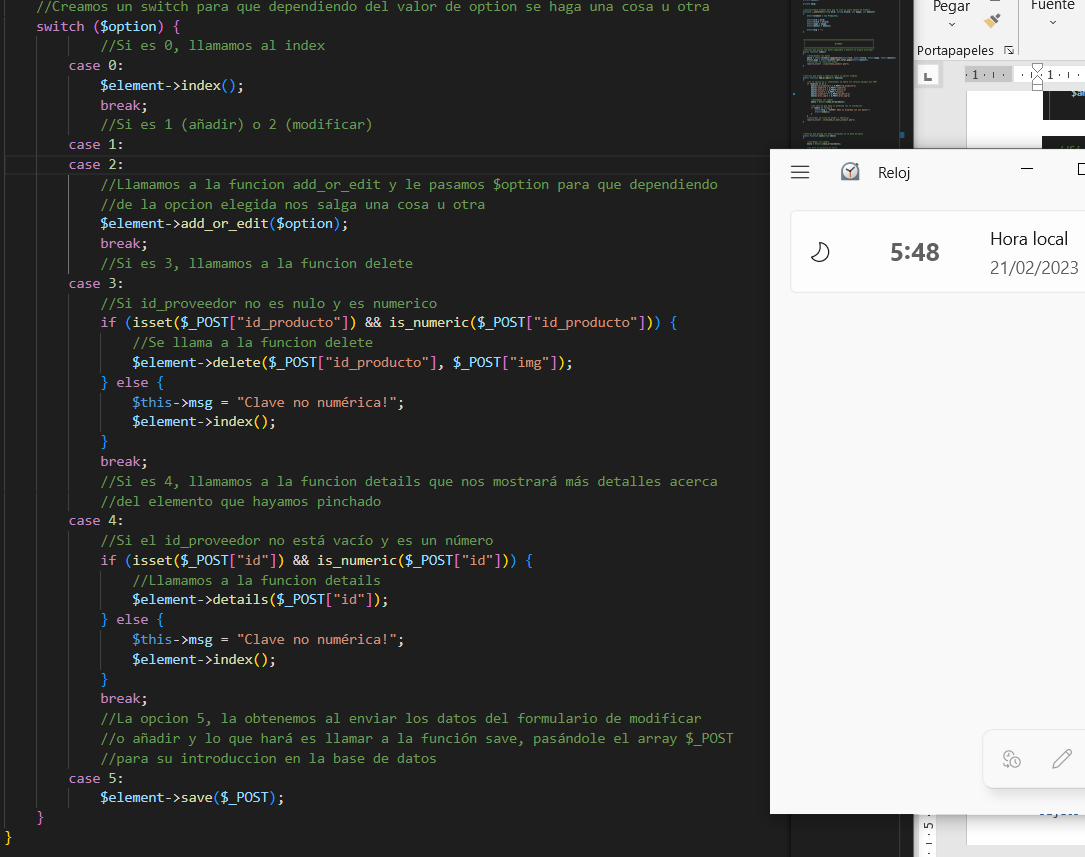




Luego, creamos un nuevo objeto de la clase **ProductC** y le pasamos como argumento las variables $**ord**, $**field**, $**page** y $**amount**.



Usamos un switch para que dependiendo del valor de $**option**, se llame a una función u otra del objeto de la clase **ProductC**.



view/index\_product.php

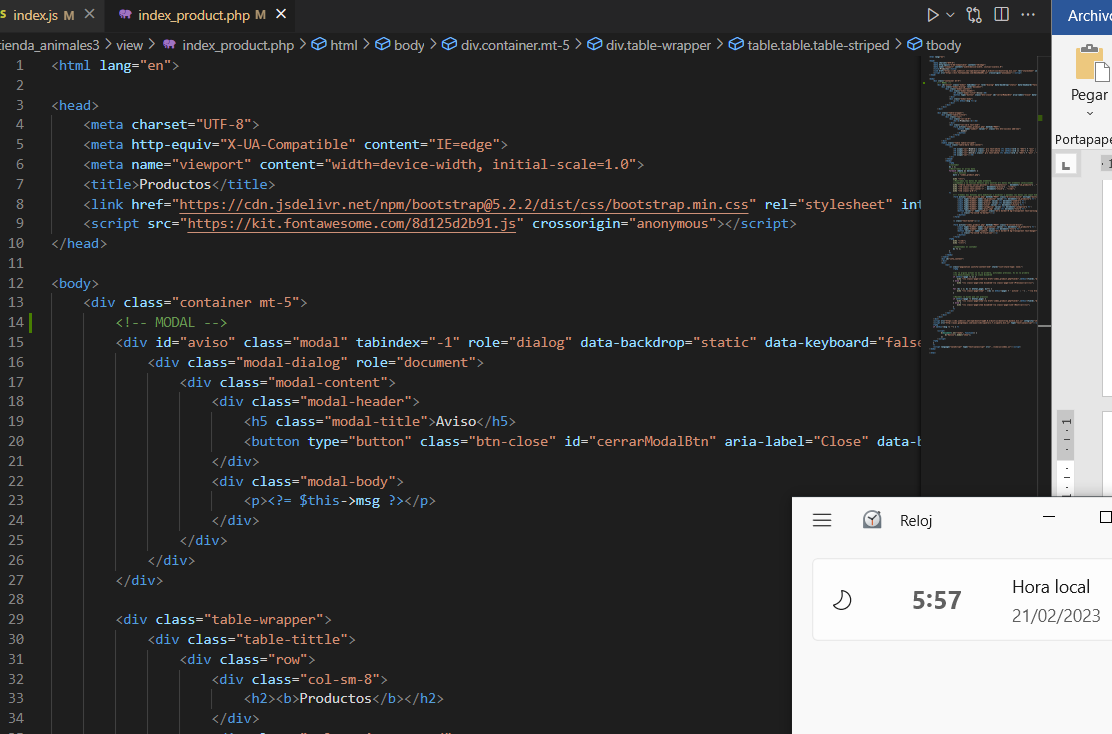
En la función index() de la clase ProductC, vemos el archivo view/index\_product.php que mostrará los datos paginados.

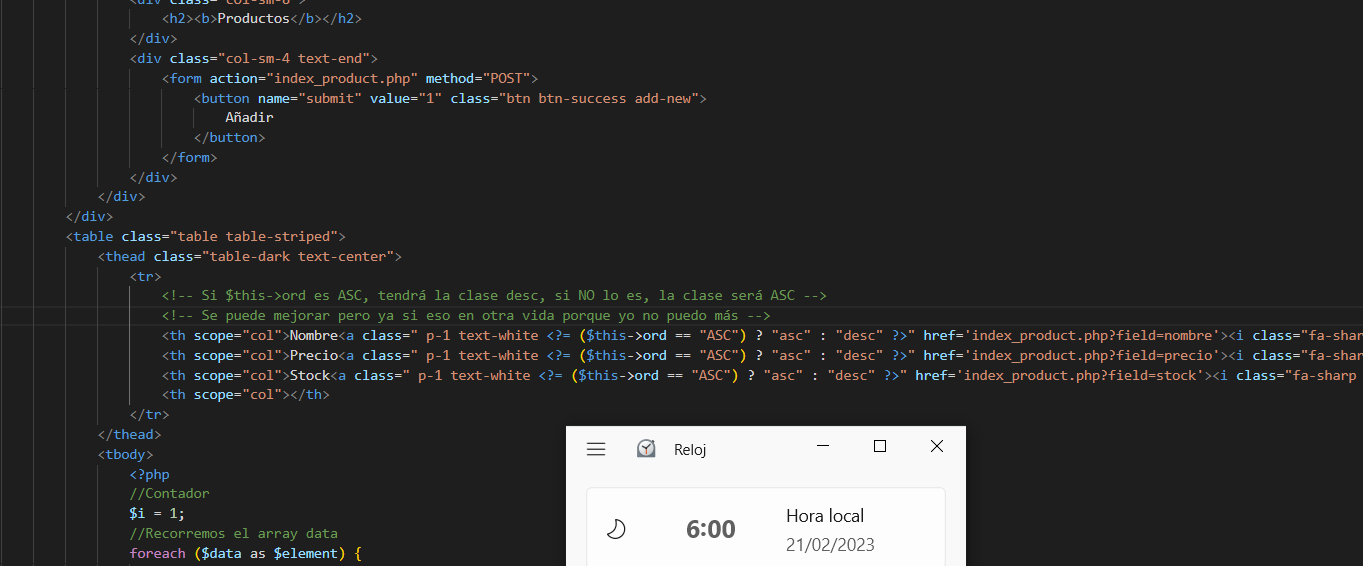
Esta vista se ha creado usando Bootstrap y es bastante simple.

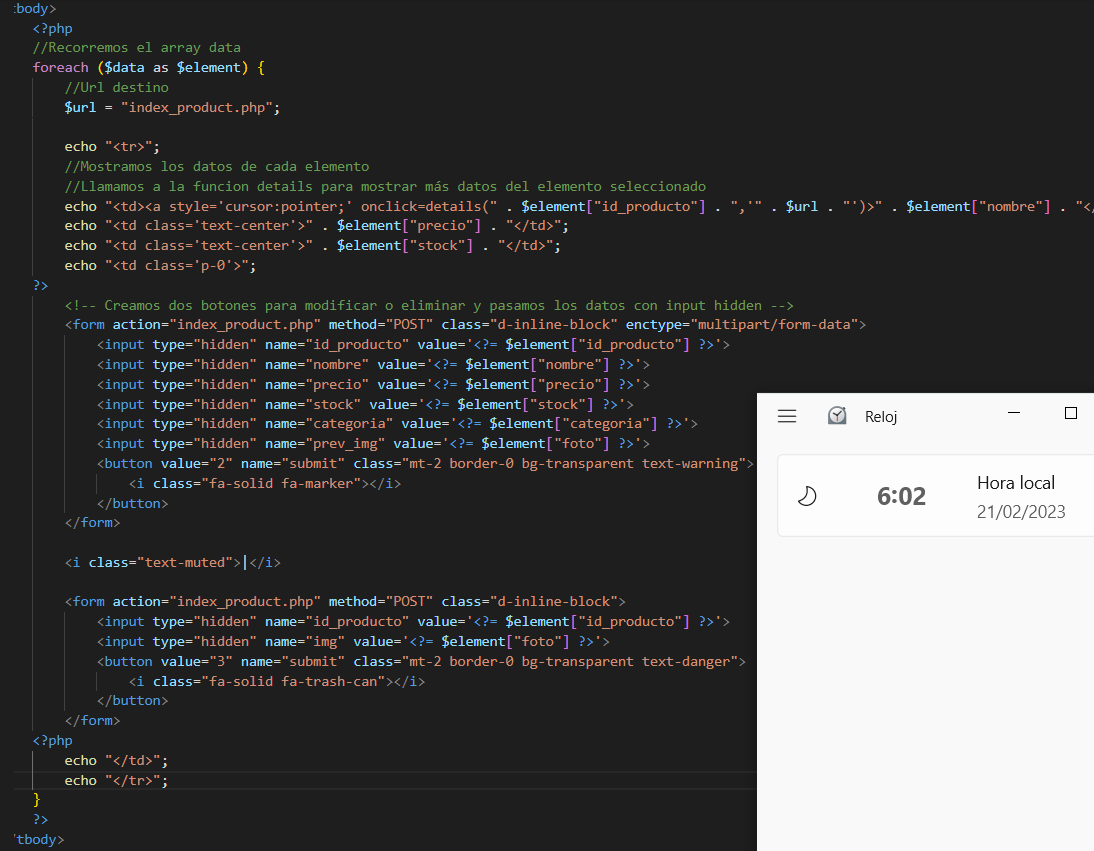
En ella tenemos un modal que se activa cuando tenemos un mensaje que mostrar.

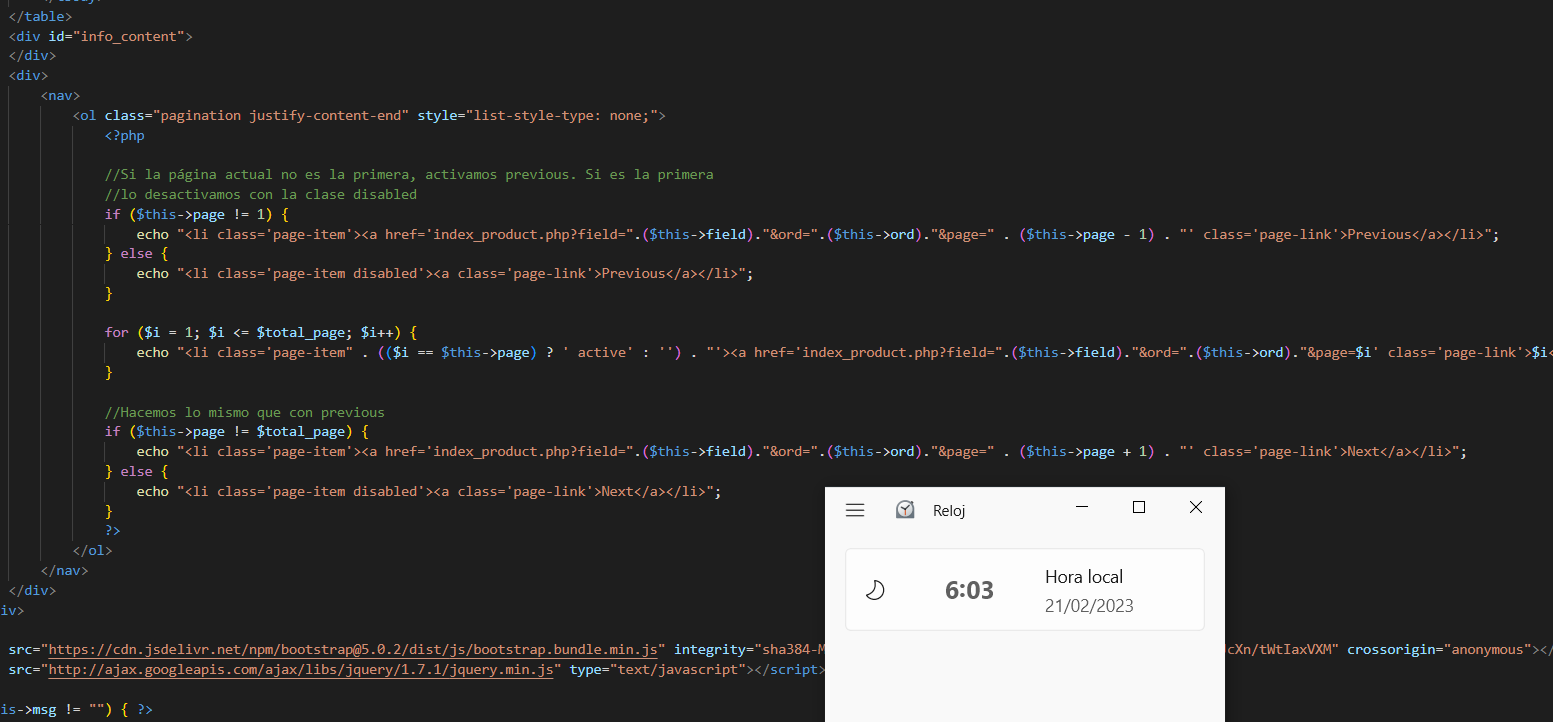
Una tabla con los datos, en la que se puede decidir si queremos que tengan un orden ascendente o descendente y que se ordenen según la casilla clicada y podremos añadir, modificar o eliminar un elemento.

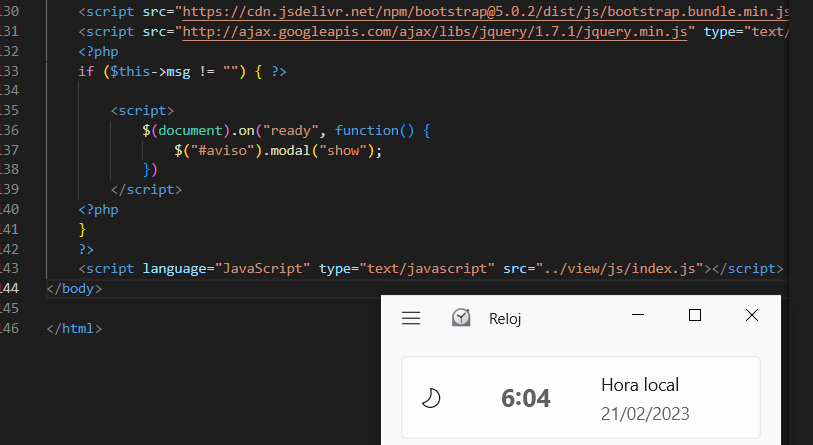
No hay mucho misterio en esto, así que pondré las capturas directamente.





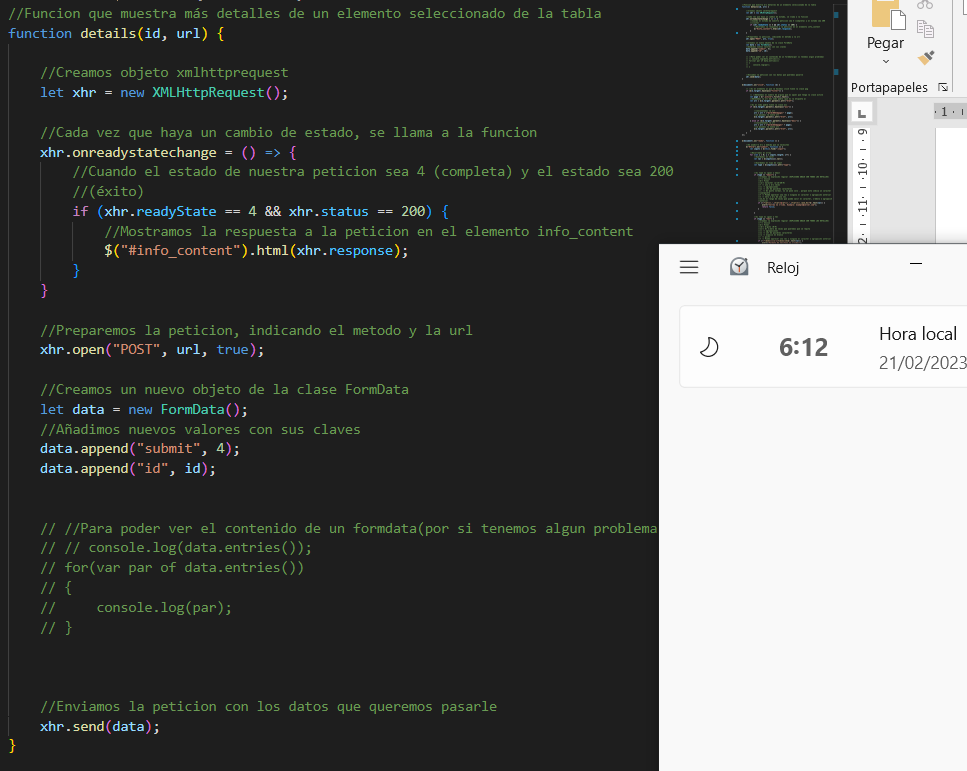




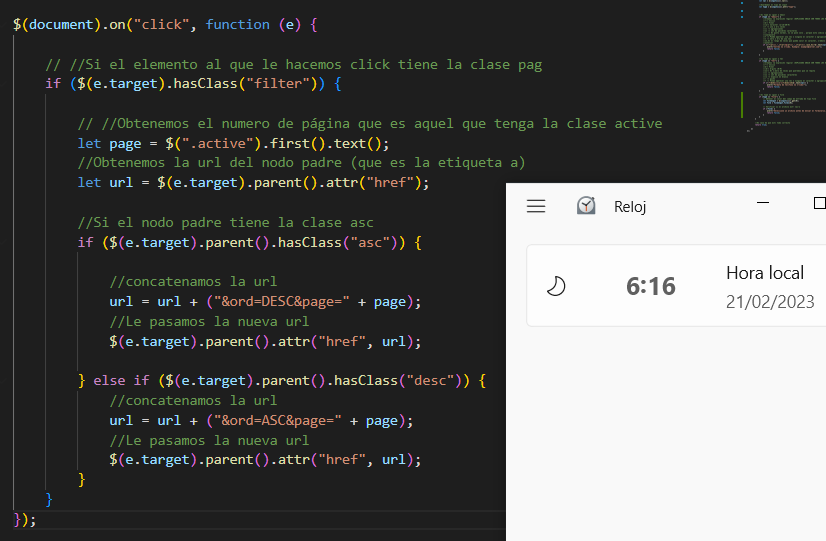


View/index.js

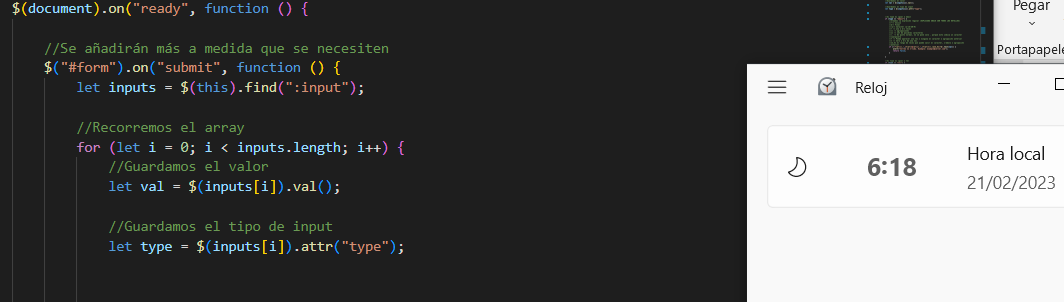
En este archivo tenemos la función **details()** en la que se utiliza AJAX para mostrar más datos del elemento seleccionado.

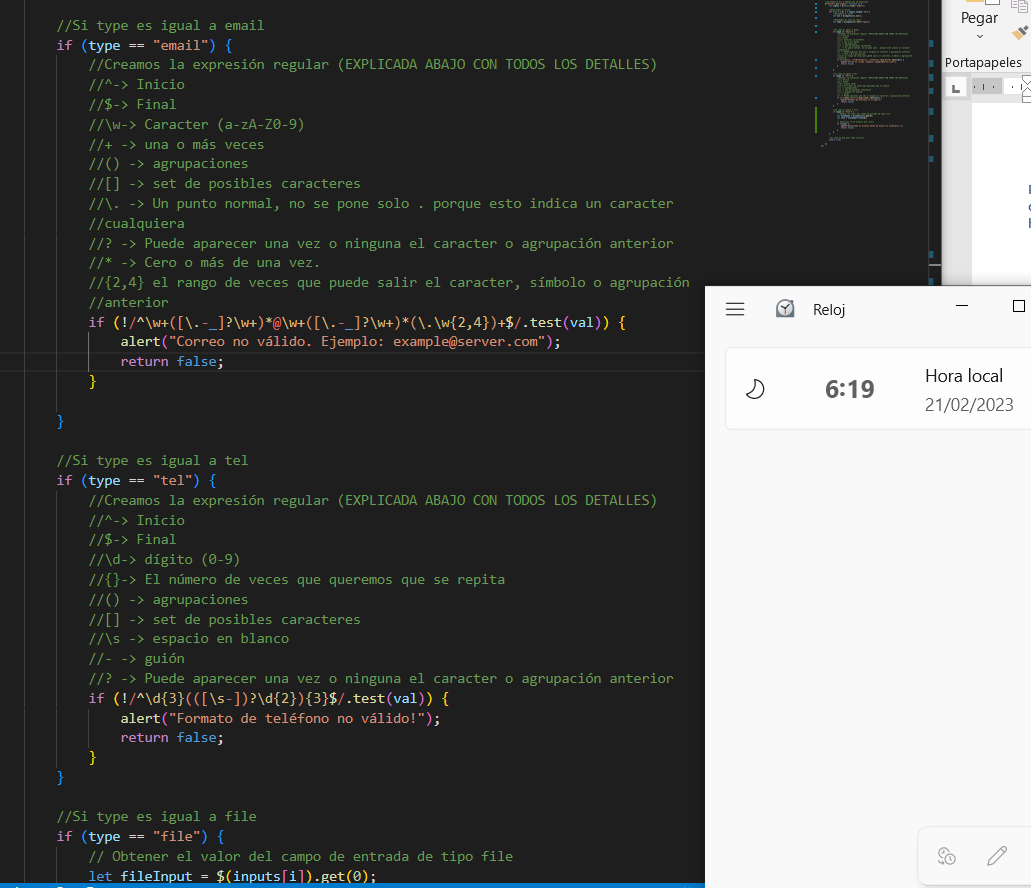


Luego, tenemos otra que cambia las clases asc y desc y, dependiendo de la clase que tenga el elemento al que se le ha realizado click, modificamos su atributo **href**.



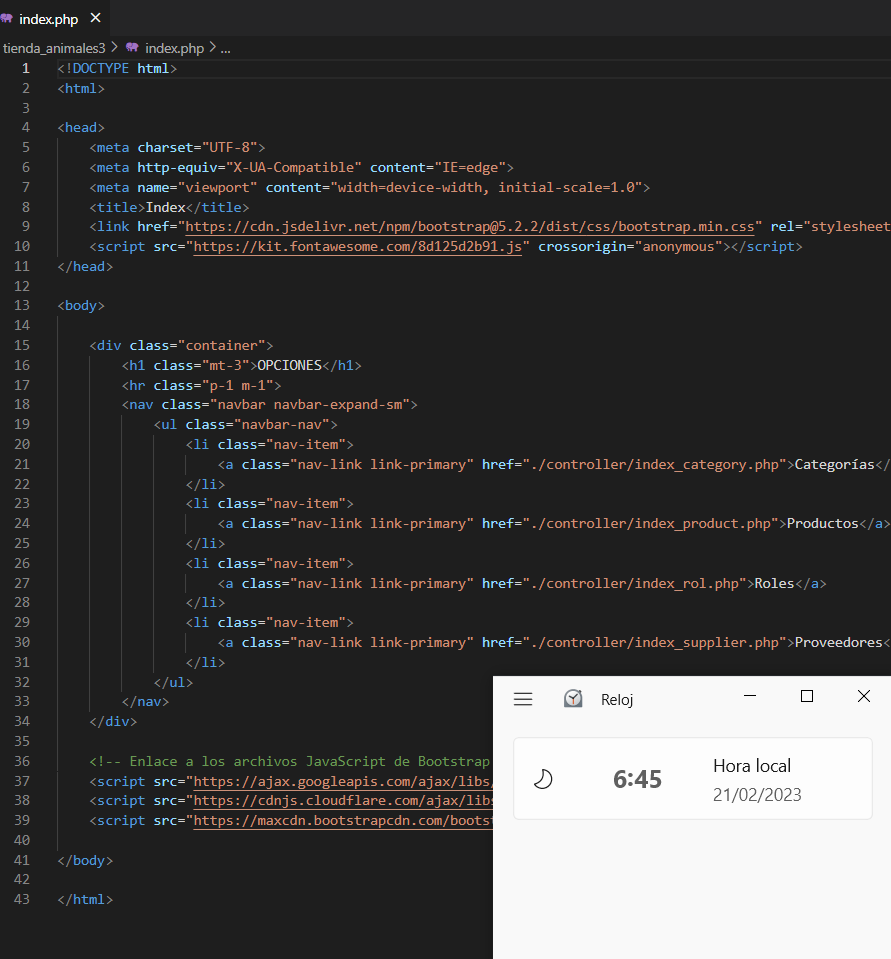
Por último, he creado otra para que se validen ciertos campos de un formulario antes de enviarlo como pueden ser un correo electrónico, el teléfono o la subida de una imagen obligatoriamente. Está hecho a lo vago, lo siento.



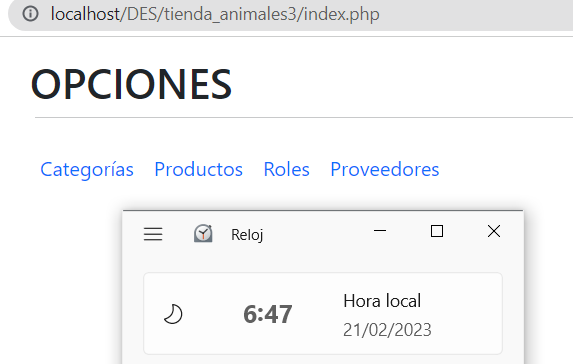


Index.php

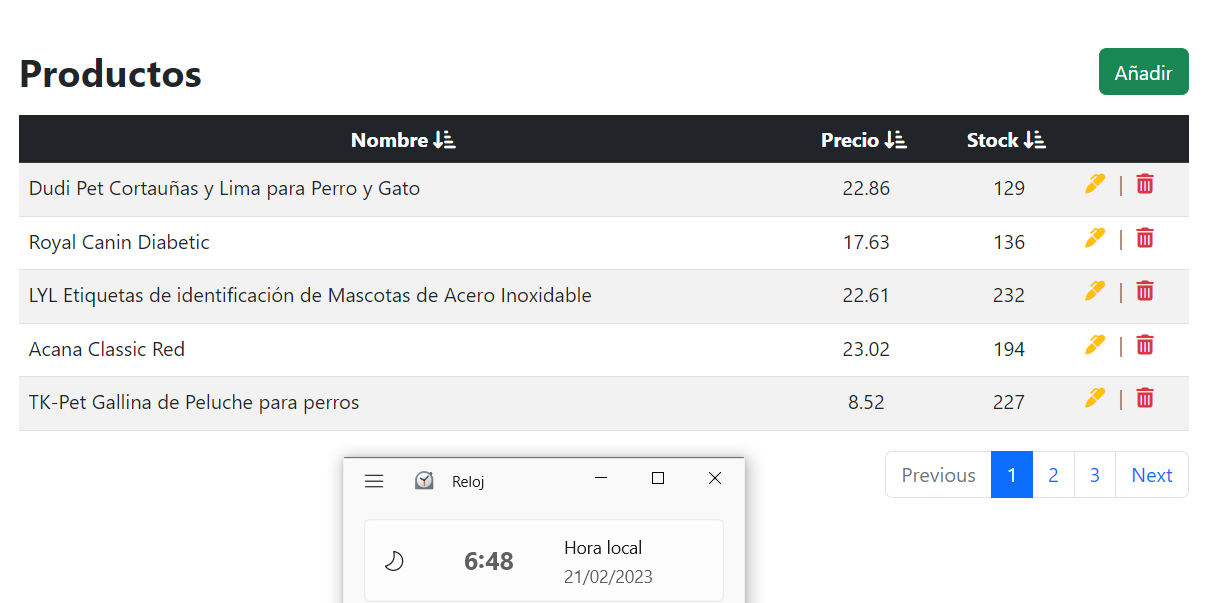
Archivo desde el cuál accederemos a las páginas principales de las 4 entidades seleccionadas para este ejercicio.



En Ejecución



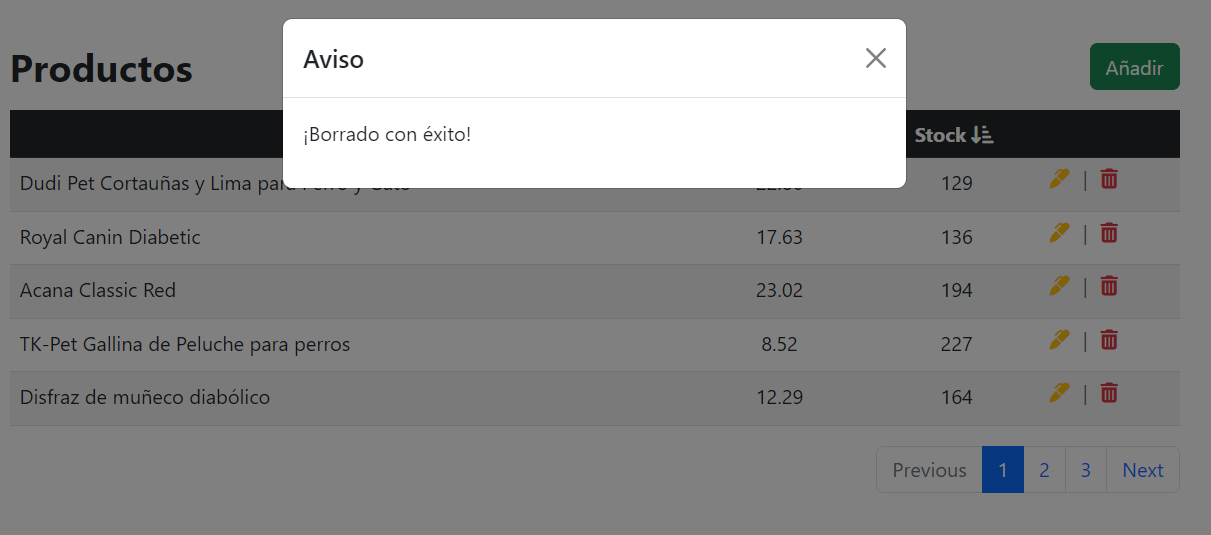
Si hacemos clic en Productos

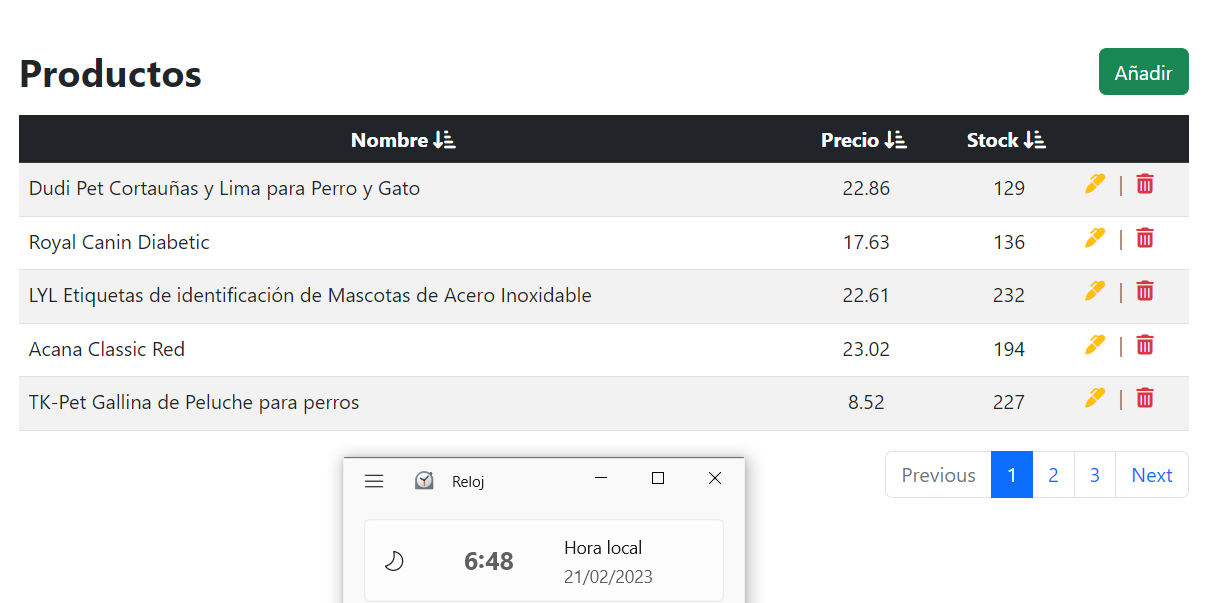


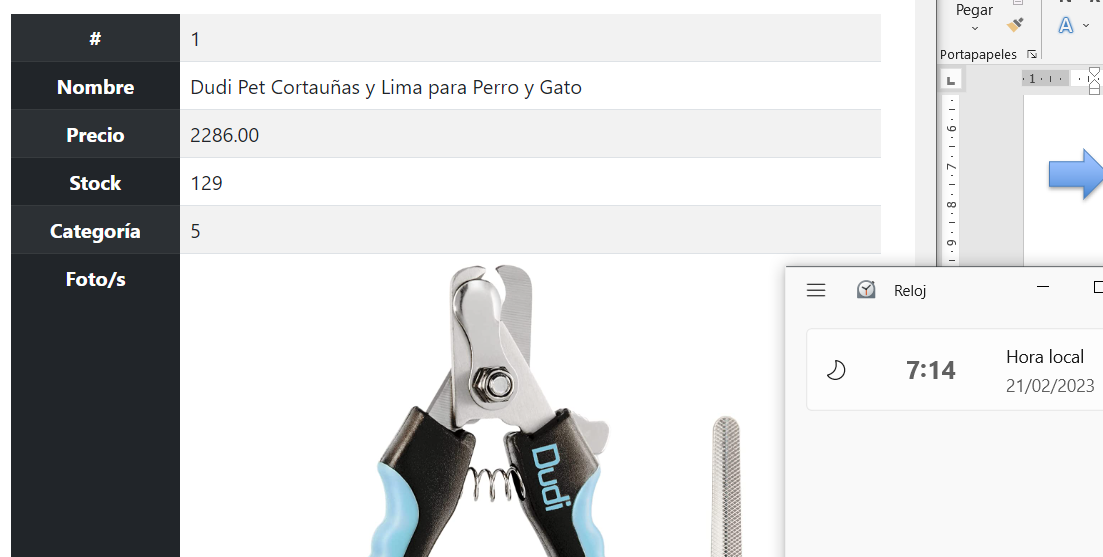
Si hacemos clic sobre el nombre



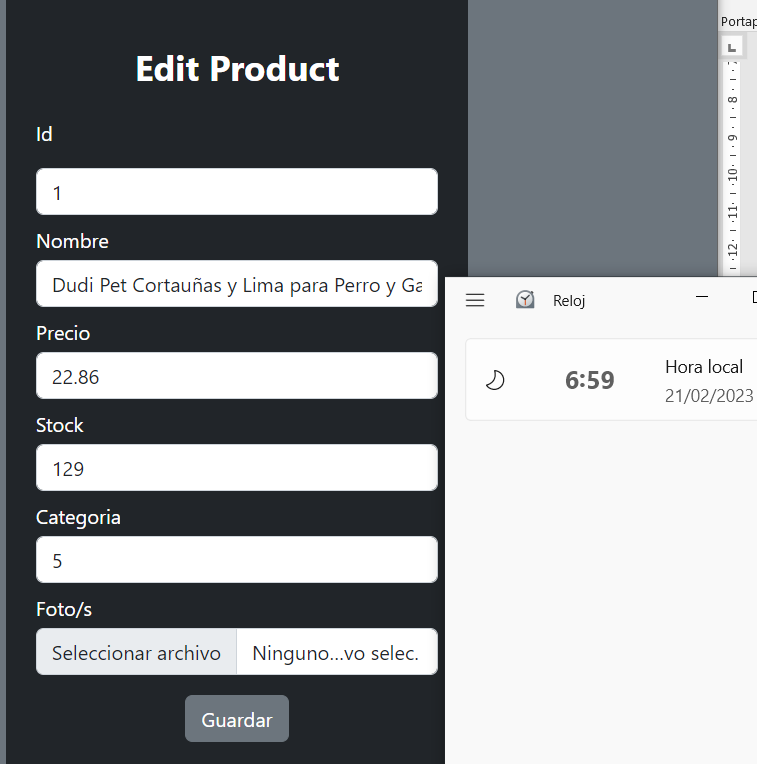
Si hacemos clic en la papelera, borramos este elemento



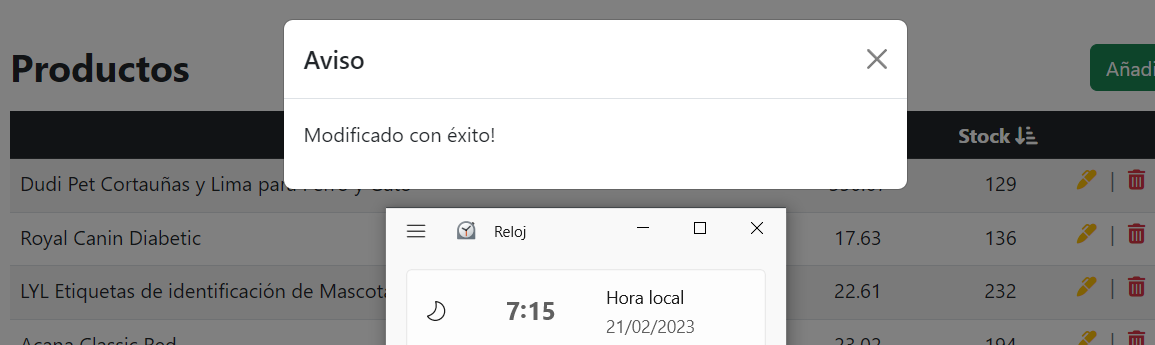
Si hacemos clic en modificar



Sale esto



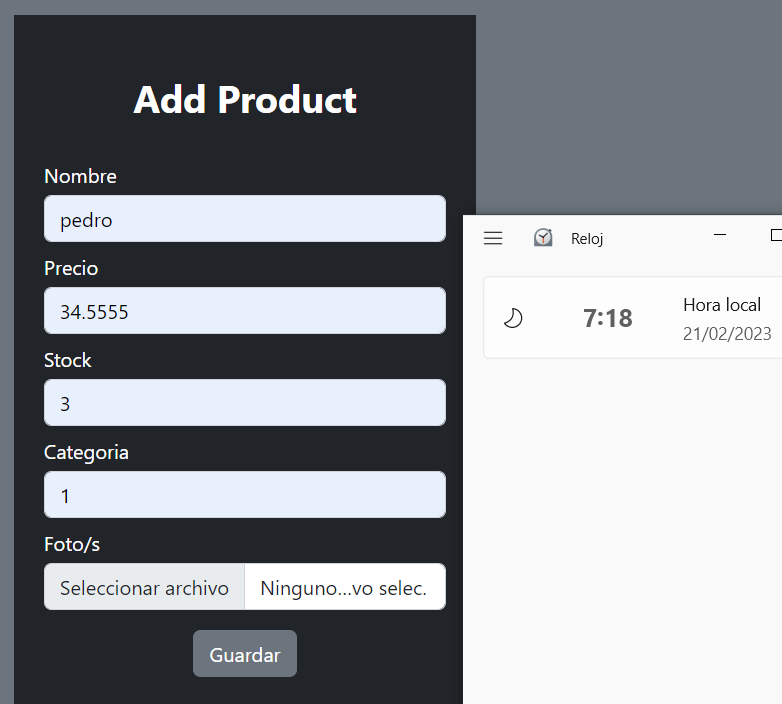
Si cambiamos el precio a 350.67 y dejamos la foto sin modificar, vemos que sigue manteniendo la misma foto.

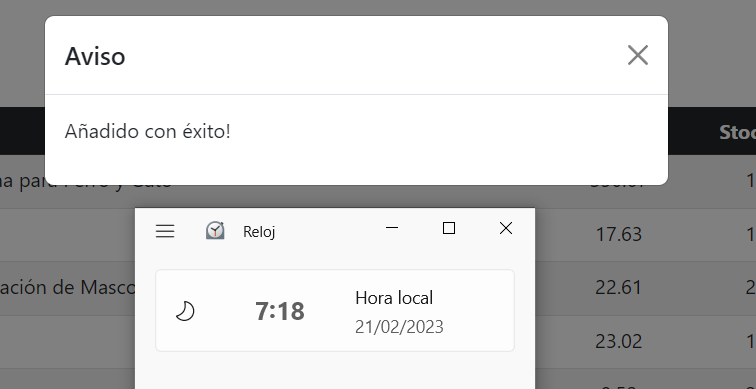


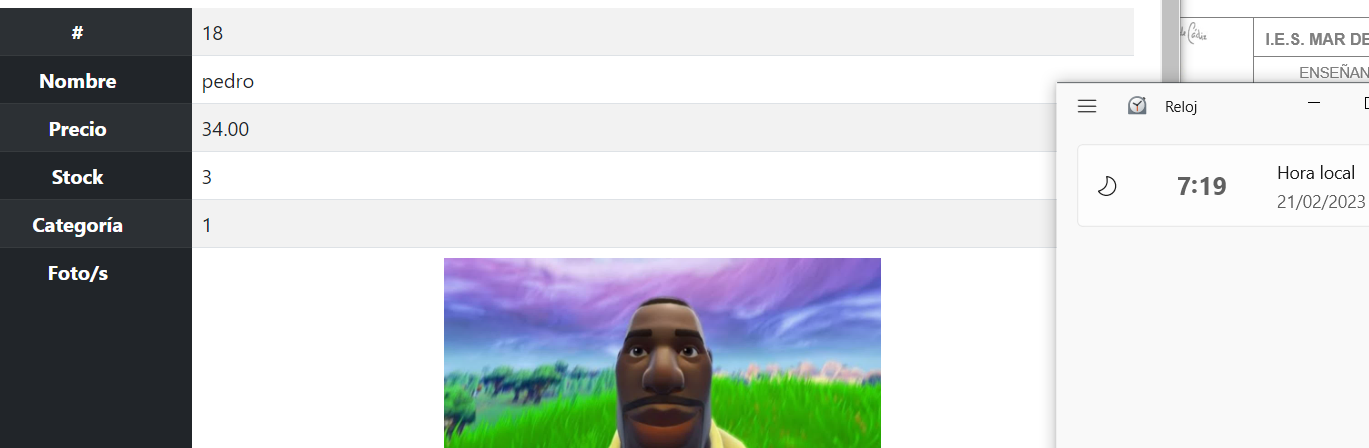
Vemos que funciona.



Si le doy a añadir y NO selecciono una foto, se pone la imagen de default.

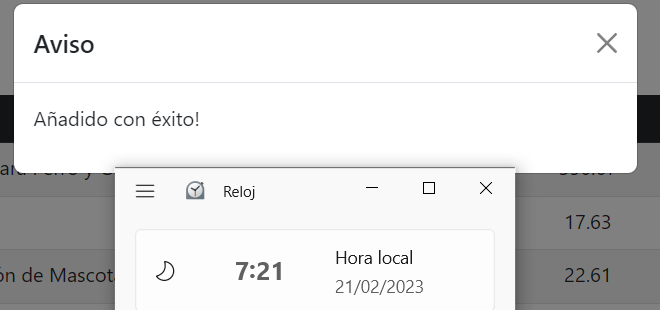


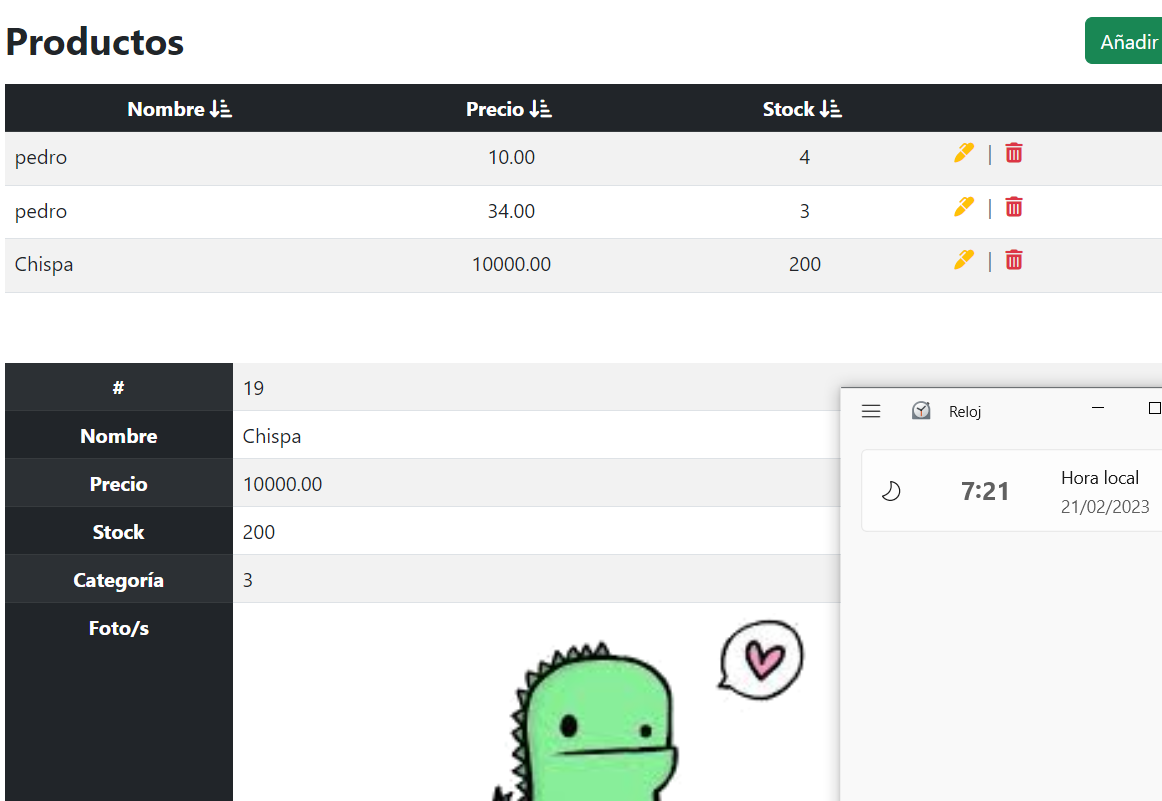




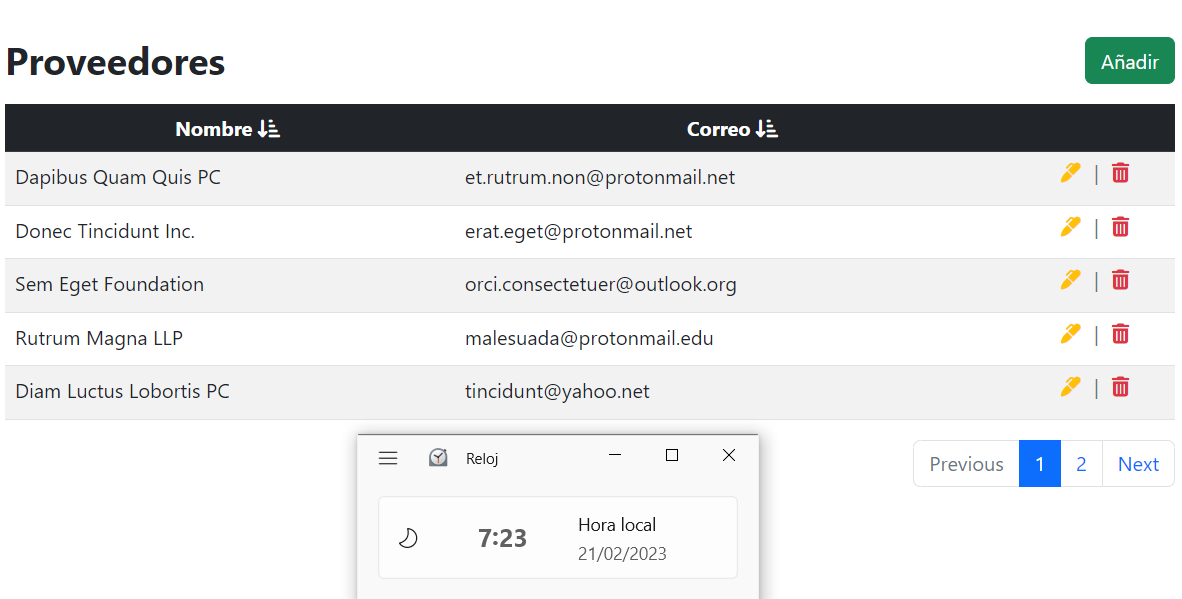
Si le doy a añadir de nuevo, pero esta vez añado una imagen







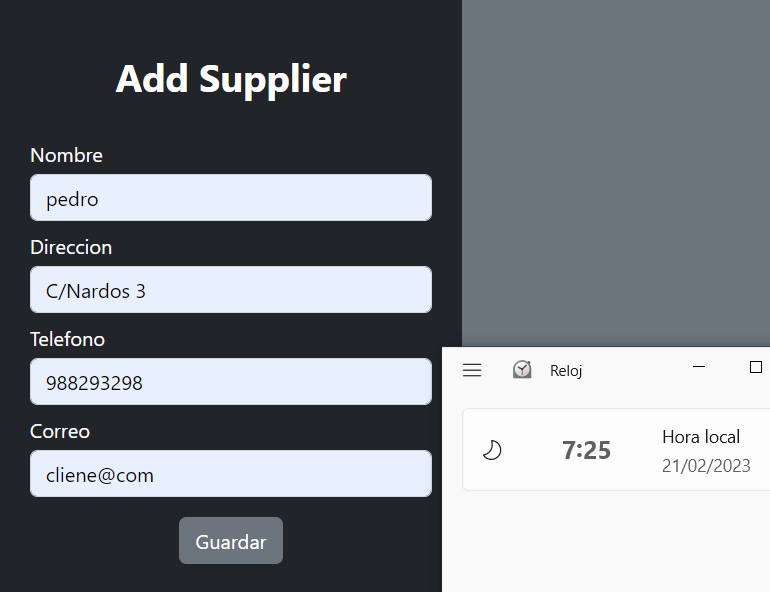
Para las otras entidades, es básicamente lo mismo. Sin embargo, hay algunas cosas más que quiero mostrar, por eso nos vamos a la página principal de proveedores.



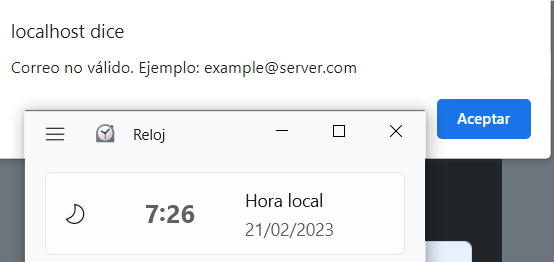
En proveedores, tenemos tanto correos electrónicos como números de teléfono.

Cómo hemos visto anteriormente, tengo una función que me valida y limpia los datos y un archivo js que también me valida correos y números de teléfono.

Veamos que pasa si intentamos introducir un correo electrónico erróneo.

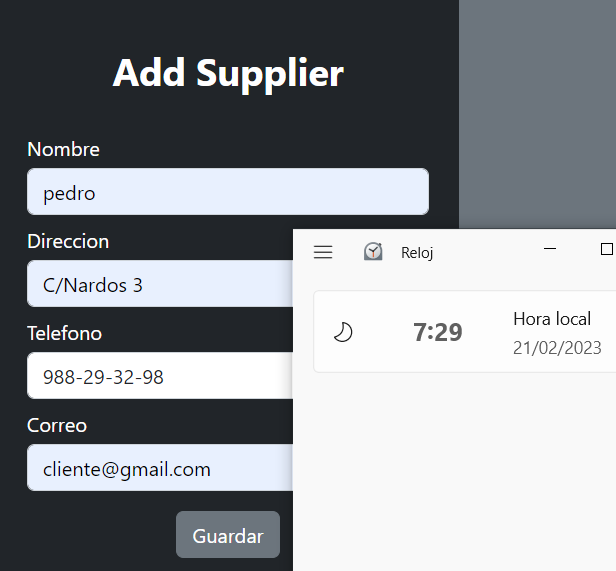


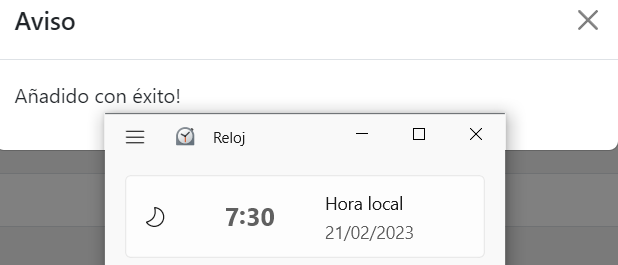
Nos salta una alerta, indicando que el formato del correo no es válido y no se manda el formulario.

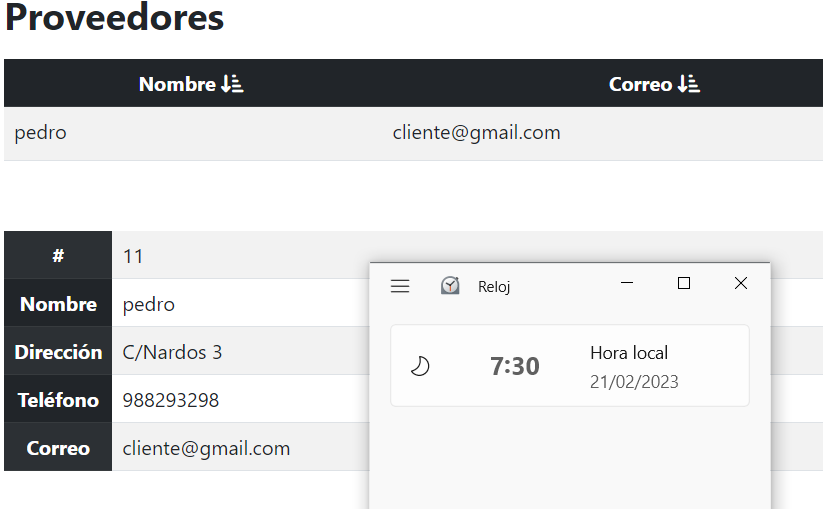


Lo mismo pasaría con el número de teléfono.

Además, el formato válido para el número de teléfono puede tener espacios, guiones o nada en grupos de 3222, pero a la hora de ser guardados en la base de datos aparecerán así 123456789.

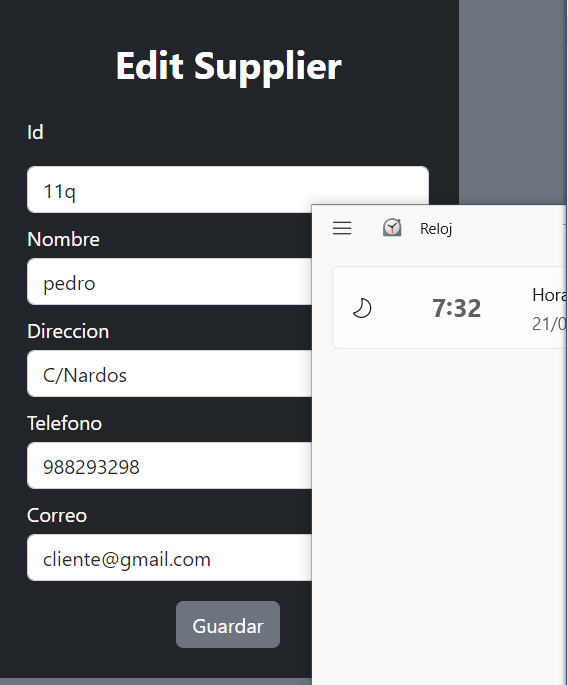


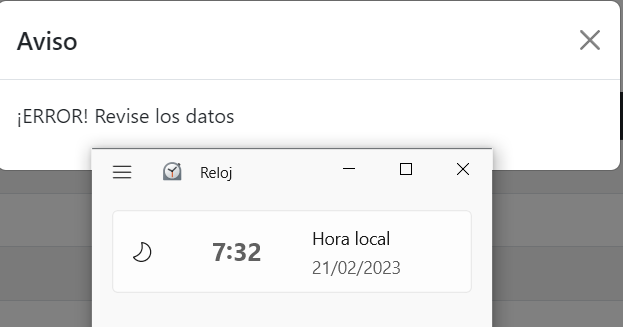




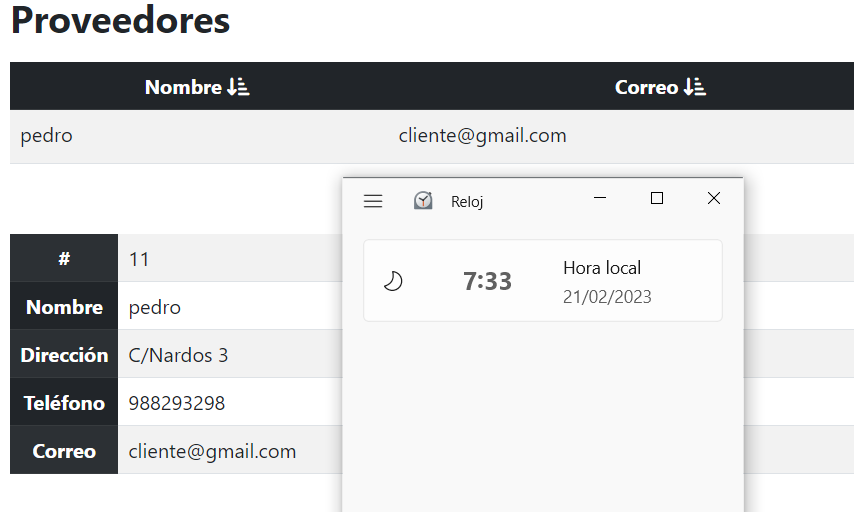
Ahora, que pasaría si intentásemos introducir un nuevo id que no fuese válido.

Pues que el error se recogería en el log y nos saltaría un aviso.





No cambia.



Y en el log nos aparece



Sé que no está muy bien ordenado ni optimizado el código pero he intentado hacer lo que he podido, lo siento.