

---

# MEMORIA DSW

---

Sara del Pino Cabrera Sánchez

2º DAW

---



---

# ÍNDICE

---

Introducción.....	3
Enlaces.....	3
Diagrama Lógico.....	4
Análisis del Proyecto.....	5
Diseño e Implementación .....	8
Refactorización.....	8
Aspectos de Interés.....	9
Clases y métodos utilizados .....	10
Problemas encontrados.....	12
Trabajo a futuro y conclusiones.....	13
Documentación.....	13
Requerimientos.....	13
Instalación .....	13

# Introducción

---

Trabajo realizado por:

- Sara del Pino Cabrera Sánchez.

Temática:

- Red social.

Para la realización de este proyecto he decidido realizar un foro, donde los usuarios podrán crear diferentes posts y responder a los posts de otras personas.

Para ello he hecho uso de diferentes tecnologías:



→ Para toda la parte del diseño web.



→ Para toda la parte de la gestión de la Base de Datos



→ Para toda la parte funcional de la web.



→ Para algún pequeño script.



→ Para la realización del proyecto.



→ Para el despliegue de la web.



→ Para la realización del README.md.

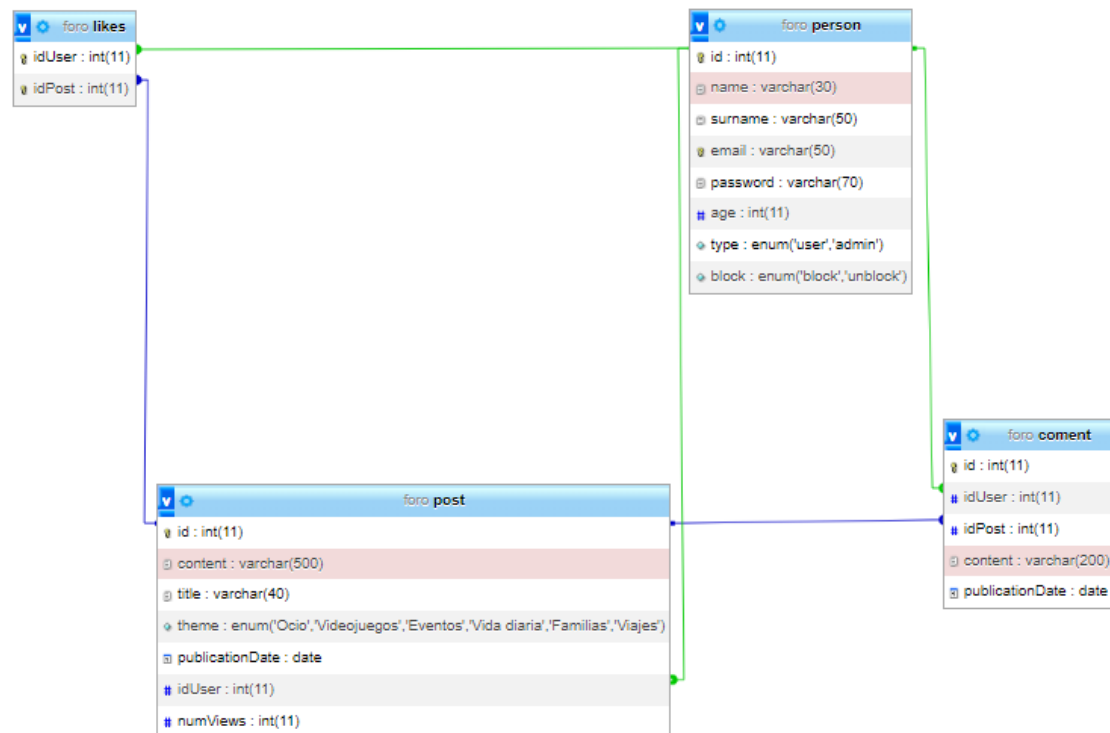


→ Para el control de versiones del proyecto.

## Enlaces

- [Enlace a la web.](#)
- [Enlace a la documentación.](#)
- [Enlace al vídeo documentado.](#)

## Diagrama lógico

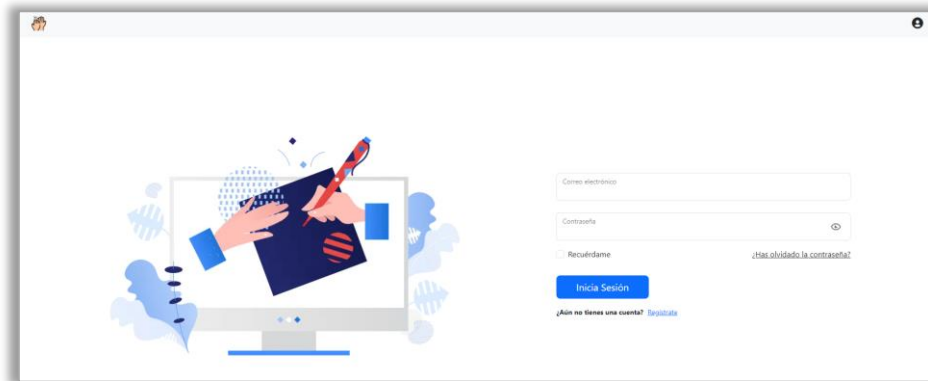


- **Likes**: una persona puede dar likes a uno o muchos posts. De forma que cada post individualmente tendrá una cierta cantidad de likes.
- **Person**: una persona puede ser tanto un usuario, como un administrador. Ambos pueden comentar, crear posts y darle likes a estos.
  - Usuario: podrá editar y eliminar sus posts, comentarios y likes.
  - Administrador: podrá editar y eliminar sus posts, comentarios y likes. También podrá editar y eliminar los posts de los usuarios normales y editar los datos de los usuarios.
- **Post**: un post puede ser escrito y editado por personas. Almacenará un número de visualizaciones que se incrementará a la hora de visitar el post.
- **Coment**: un post contendrá comentarios escritos por las personas. Estos pueden ser editados y eliminados, únicamente, por aquellas personas que los escriban.

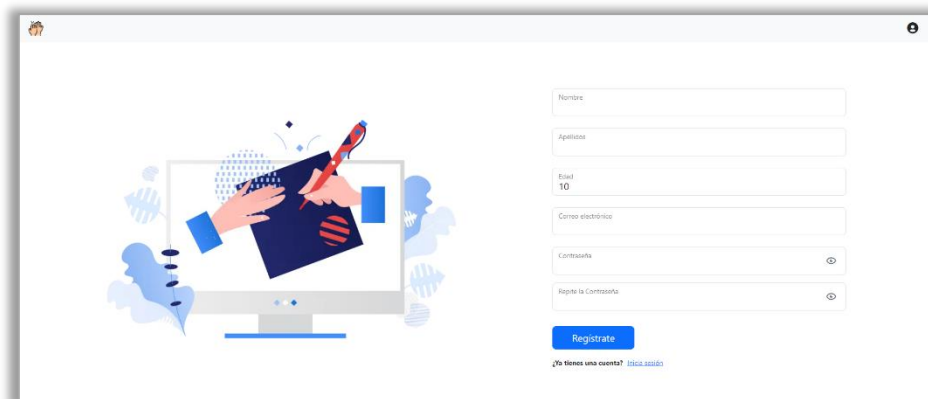
## Análisis del Proyecto

El proyecto consiste en un Foro, donde tanto usuarios como administradores podrán crear diferentes Posts. Dentro de cada Posts, cualquiera podrá realizar comentarios referentes al foro.

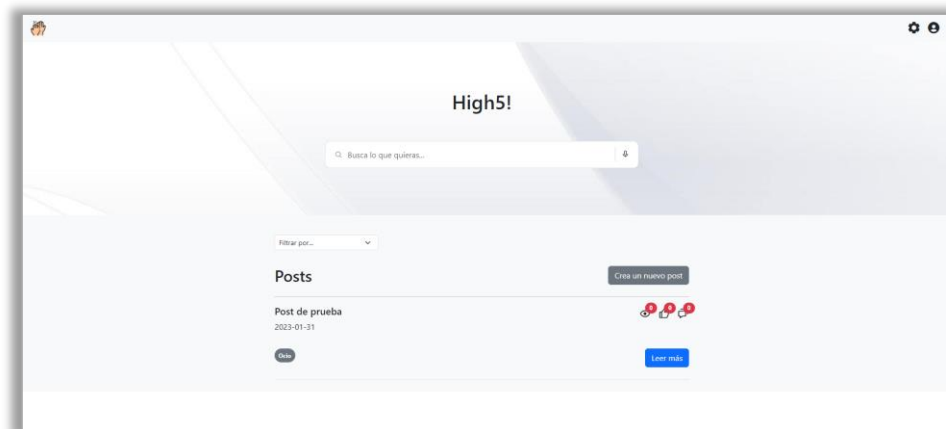
Primeramente, en la web nos encontraremos con un [Login](#), donde podremos iniciar sesión y entrar a la web.

The image shows a login form within a web browser window. On the left, there is a stylized illustration of a hand holding a pen and writing on a document displayed on a computer monitor. The form on the right contains a text input field for 'Correo electrónico', a password input field for 'Contraseña' with a toggle icon, a 'Recuérdame' checkbox, and a '¿Has olvidado la contraseña?' link. Below these is a blue 'Inicia Sesión' button and a link for '¿Aún no tienes una cuenta? Regístrate'.

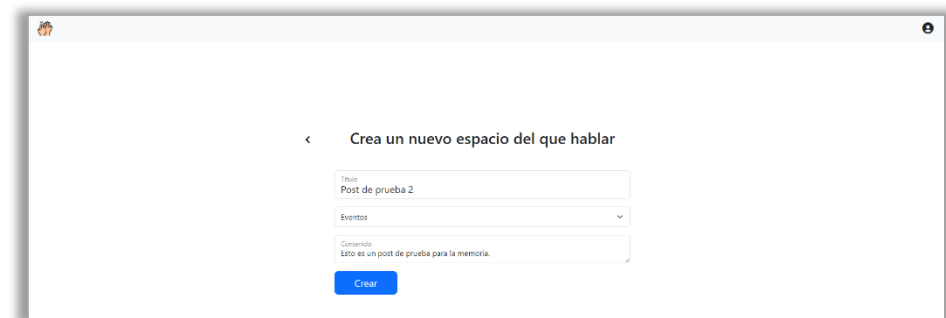
En caso de no disponer de una cuenta, podremos ir a la página de [Registro](#) y crear una cuenta nueva, siempre teniendo en cuenta que el email del usuario debe ser único dentro de la Base de Datos.

The image shows a registration form within a web browser window. On the left, there is a stylized illustration of a hand holding a pen and writing on a document displayed on a computer monitor. The form on the right contains input fields for 'Nombre', 'Apellidos', 'Edad' (with the value '10' pre-filled), 'Correo electrónico', 'Contraseña', and 'Repita la Contraseña' (with toggle icons). Below these is a blue 'Regístrate' button and a link for '¿Ya tienes una cuenta? Inicia sesión'.

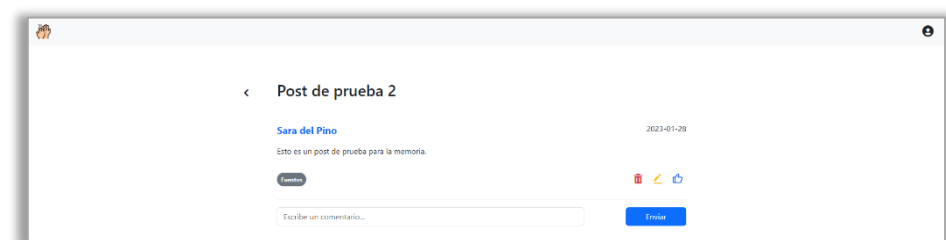
Una vez tengamos nuestra cuenta creada e iniciemos sesión, estaremos en la [Página Principal](#). Aquí podremos ver todos los Posts subidos hasta la fecha en nuestra web.



Podremos crear nuevos posts desde el botón que pone **“Crear un nuevo post”**. Esto nos llevará a una nueva página en donde tendremos que rellenar los datos del formulario.

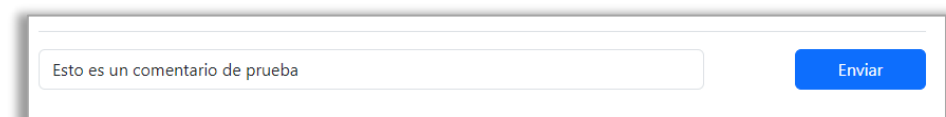


Una vez tengamos nuestro post creado, volveremos a la página principal y podremos darle al botón de **“Leer más”** para ver en mayor profundidad nuestro post, o cualquier otro post que esté publicado.

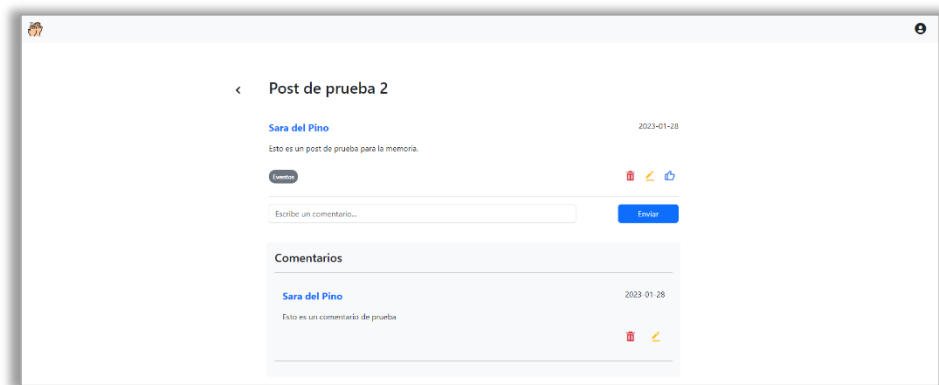



Al haber creado nosotros este post, veremos que podemos eliminarlo y editarlo. En caso de que otra persona hubiera escrito el post no tendríamos disponibles estos iconos.

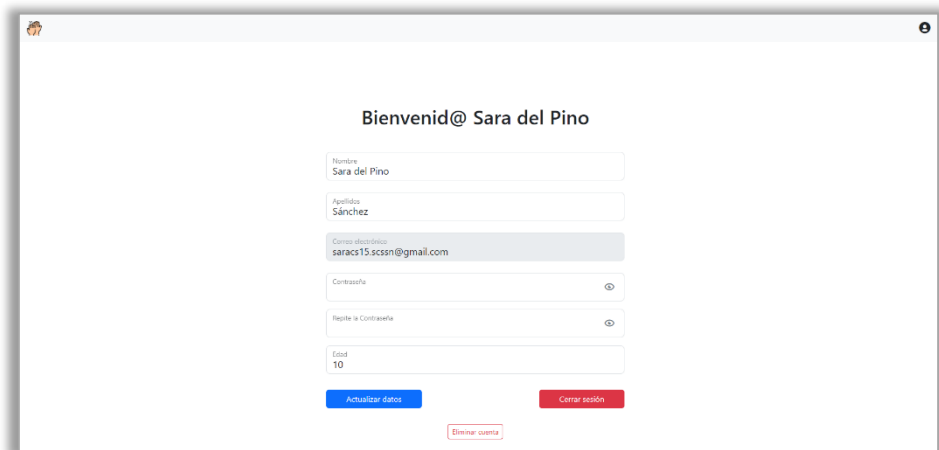
Dentro del post podremos darle a [Like](#) y también podremos realizar diferentes comentarios:



Y al darle al botón de “Enviar” veremos como el comentario se sube al post. Al igual que pasaba con los posts, al crear nosotros el comentario podremos editarlo o eliminarlo cuando queramos.




Ahora que ya hemos visto toda la página principal y sus funcionalidades principales, podremos pasar a la parte de Configuración del Usuario, llegando a este a través del icono del header .

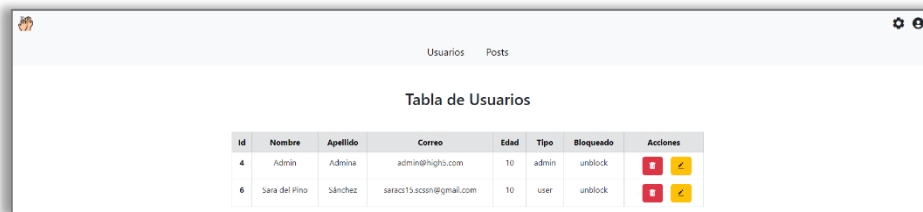






Aquí podremos darle al botón de “Actualizar datos” para modificar los datos anteriores del usuario y modificarlos por los que queramos.

También podremos darle al botón de “Cerrar sesión” para cerrar la sesión actual y volver a la página de [Login](#).

Por último, podemos darle al botón de “Eliminar cuenta” para eliminar la cuenta, cerrando la sesión actual y eliminando al usuario de la Base de Datos.

A continuación, si iniciamos sesión como administrador, veremos un nuevo icono en el header  que nos llevará al [Panel de Administración](#), al que solo tiene acceso el administrador.



Id	Nombre	Apellido	Correo	Edad	Tipo	Bloqueado	Acciones
4	Admin	Admina	admin@high0.com	10	admin	unblock	 
6	Sara del Pino	Sánchez	saract5.acson@gmail.com	10	user	unblock	 

Aquí podremos editar y eliminar tanto usuarios como posts dentro de la web. También tendremos la posibilidad de bloquear usuarios y que estos no puedan entrar a la web hasta que se lo volvamos a permitir.

## Diseño e Implementación

Para toda la parte visual de la web he utilizado plantillas de Bootstrap, sacadas de [esta página web](#).

### Refactorización

Al comienzo de la creación de la web, tenía toda la lógica con *programación funcional*, dividida en diferentes ficheros dependiendo de que hacía cada uno. Con el paso del tiempo fui cambiando la lógica de *programación funcional* y cambiándola a *programación orientada a objetos*, permitiéndome tener una sintaxis mucho más ordenada y, además, darme la posibilidad de usar en [autoload](#).

```
1 <?php
2
3 /** ...
4 */
5 function createComent(){
6     try {
7         $connection = connect();
8
9         $querySQL = $connection->prepare
10         ("INSERT INTO coment (idUser, idPost, content, publicationDate) VALUES
11         (:idUser, :idPost, :content, :publicationDate)");
12
13         $querySQL->bindValue(':idUser', $_SESSION["idUser"], PDO::PARAM_INT);
14         $querySQL->bindValue(':idPost', $_REQUEST["idPost"], PDO::PARAM_INT);
15         $querySQL->bindValue(':content', $_REQUEST["content"], PDO::PARAM_STR);
16         $querySQL->bindValue(':publicationDate', date('Y-m-d'), PDO::PARAM_STR);
17
18         $querySQL->execute();
19
20     } catch(PDOException $error) {
21         $error = $error->getMessage();
22     }
23 }
```

Primera versión

```
1 <?php
2
3 namespace Controllers;
4
5 use PDO;
6 use PDOException;
7 use Config\ConnectDB;
8
9 /** ...
10 */
11 class Coment {
12     /** ...
13     */
14     public static function createComent(){
15         try {
16             $connection = ConnectDB::connect();
17
18             $querySQL = $connection->prepare
19             ("INSERT INTO coment (idUser, idPost, content, publicationDate) VALUES
20             (:idUser, :idPost, :content, :publicationDate)");
21
22             $querySQL->bindValue(':idUser', $_SESSION["idUser"], PDO::PARAM_INT);
23             $querySQL->bindValue(':idPost', $_REQUEST["idPost"], PDO::PARAM_INT);
24             $querySQL->bindValue(':content', $_REQUEST["content"], PDO::PARAM_STR);
25             $querySQL->bindValue(':publicationDate', date('Y-m-d'), PDO::PARAM_STR); // Fecha actual
26
27             $querySQL->execute();
28
29         } catch(PDOException $error) {
30             $error = $error->getMessage();
31         }
32     }
33 }
```

Última versión



Además de esto, cambié algunos *condicionales clásicos* por *condicionales ternarios*, simplificando así la lectura de muchas partes del código.

Primeramente, llamaba a la Base de Datos pasándole los parámetros de manera manual y poco segura, pero más adelante cambié esto por variables de entorno, asegurando la privacidad de estos datos.

```
1 <?php
2
3 $dotenv=Dotenv\Dotenv:: createImmutable(__DIR__ . "\..");
4 $dotenv->safeLoad();
5
6 > /** ...
7
8 */
9 function create(){
10     $config = require "../config/config.php";
11     [
12         "host" => $host,
13         "user" => $user,
14         "pass" => $pass,
15         "name" => $name,
16         "options" => $options
17     ] = $config["db"];
18
19     $connection = new PDO("mysql:host=$host", $user, $pass, $options);
20     return $connection;
21 }
22
23
24
25
26
27
28
29
30
31
```

```
1 <?php
2
3 // Datos de configuración para conectarnos a la Base de Datos
4 return [
5     "db" => [
6         "host" => "localhost",
7         "user" => "root",
8         "pass" => "",
9         "name" => "Foro",
10        "options" => [
11            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
12        ]
13    ]
14 ];
15 ?>
```

Primera versión

```
1 <?php
2
3 namespace Config;
4
5 use PDO;
6 use PDOException;
7 use Dotenv;
8 use Controllers\Person;
9 use Controllers\Post;
10
11 $dotenv=Dotenv\Dotenv:: createImmutable(__DIR__ . "\..");
12 $dotenv->safeLoad();
13
14 > /** ...
15
16 */
17 class ConnectDB {
18     /** ...
19
20     */
21     public static function create(){
22         $options = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION];
23         $connection = new PDO("mysql:host=" . $_ENV['DB_HOST'], $_ENV['DB_USER'], $_ENV['DB_PASS'], $options);
24         return $connection;
25     }
26 }
```

Última versión

```
1 # Variables de entorno para la Base de Datos
2 DB_HOST=localhost
3 DB_USER=root
4 DB_PASS=
5 DB_DB=Foro
```

.env

## Aspectos de Interés

Como aspectos de interés cabe recalcar el uso de [Composer](#) para poder utilizar las variables de entorno y el autoload.

El [autoload](#) funciona declarando un espacio de nombres a las clases que queramos importar y, en aquellos ficheros que queramos que lo usen, tenemos que definírselo.

Clase [SelectPage](#) con espacio de nombre [Config](#)

```
1 <?php
2
3 namespace Config;
4
5 > /** ...
6
7 */
8 class SelectPage {
```

```
10 use Config\SelectPage;
11
12 ConnectDB::install(); Llamada a la clase SelectPage ubicada
13 ?>
14
15 <?php
16 // Definimos la ruta a la que queremos movernos dentro de la web
17 $routes = require "../static/constant/routes.php";
18
19 // En caso de no tener una url a la que ir, o que el
20 // usuario no haya iniciado sesión...
21 if (!isset($_REQUEST['url']) || !isset($_SESSION['idUser'])){
22     SelectPage::selectPage();
```

## Clases y métodos utilizados

- Dentro del espacio de nombres de **Config** podemos encontrar las siguientes clases:
  - **ConnectDB**: es la clase encargada de conectarse con la Base de Datos. Sus métodos son:
    - **connect()**: se encarga de conectarse a la base de datos.
    - **create()**: se encarga de crear la base de datos.
    - **install()**: se encarga de instalar la base de datos y de inicializar ciertos valores por defecto.
  - **Debug**: es la clase encargada de debuggear ciertas partes de la web cuando haga falta. Su método es:
    - **debugArray()**: recibe un array y se encarga de devolverlo en formato **string** (clave – valor).
  - **SelectPage**: es la clase encargada de devolver la url correspondiente en cada momento. Su método es:
    - **selectPage()**: comprueba los parámetros de la sesión para verificar a que punto de la web debe dirigirse.
  - **Validate**: es la clase encargada de validar los parámetros pasados a través de los distintos formularios. Sus métodos son:
    - **comprobeInsert()**: recibe varios parámetros y se encarga de comprobar que estén escritos correctamente.
    - **hashPassword()**: recibe una contraseña y la encripta usando el algoritmo **PASSWORD\_BCRYPT**.
    - **validate()**: recibe varios parámetros, comprueba que estén escritos correctamente y que estén todos los valores necesarios.
    - **validateEmail()**: recibe un email y se encarga de validar que esté bien escrito.
    - **validatePassword()**: recibe dos contraseñas, comprueba que la primera esté bien escrita y, después, que ambas son iguales.
    - **verifyPassword()**: recibe una contraseña y la compara con la contraseña encriptada de la base de datos.
- Dentro del espacio de nombres de **Controllers** podemos encontrar las siguientes clases:
  - **Coment**: es la clase encargada del CRUD de los comentarios. Sus métodos son:
    - **countComents()**: devuelve la cantidad de comentarios que tiene un post.
    - **createComent()**: crea un nuevo comentario en la base de datos.
    - **deleteComent()**: elimina un comentario de la base de datos.
    - **selectComentPost()**: selecciona un comentario de la base de datos y devuelve sus datos.
    - **updateComent()**: actualiza los datos de un comentario en la base de datos.
  - **Likes**: es la clase encargada del CRUD de los likes. Sus métodos son:
    - **countLikesPost()**: cuenta la cantidad de likes que tiene un post.
    - **deleteLike()**: elimina un like de la base de datos.

- **getLikesPost()**: busca likes en la base de datos y devuelve sus valores.
- **setLike()**: crea un nuevo like en la base de datos.
- **Person**: es la clase encargada del CRUD de las personas. Sus métodos son:
  - **allEmailPerson()**: devuelve todos los email de la base de datos.
  - **comprobePerson()**: comprueba que exista el email de un usuario y, además, que su contraseña coincida con la que tiene en la base de datos.
  - **createAdmin()**: crea un administrador por defecto en la web.
  - **createPerson()**: crea una nueva persona en la base de datos.
  - **deletePerson()**: elimina una persona de la base de datos.
  - **selectAllPerson()**: busca a todas las personas y devuelve sus datos.
  - **selectBlockPerson()**: devuelve el estado del bloqueo de una persona.
  - **selectEmailPerson()**: devuelve el email de una persona.
  - **selectIdPerson()**: devuelve el id de una persona.
  - **selectNamePerson()**: devuelve el nombre de una persona.
  - **selectPerson()**: devuelve todos los datos de una persona.
  - **selectTypePerson()**: devuelve el tipo de una persona.
  - **updatePerson()**: actualiza los datos de una persona en la base de datos.
- **Post**: es la clase encargada del CRUD de los posts. Sus métodos son:
  - **createDefaultPost()**: crea un post por defecto en la web.
  - **createPost()**: crea un nuevo post en la base de datos.
  - **deletePost()**: elimina un post de la base de datos.
  - **filterPost()**: devuelve los datos de todos los post filtrados en el orden que el usuario solicita.
  - **getViews()**: devuelve la cantidad de vistas que tiene un post.
  - **incrementViews()**: incrementa en 1 la cantidad de visitas en el post.
  - **selectAllPost()**: devuelve los datos de todos los posts.
  - **selectPost()**: devuelve los datos de un post.
  - **updatePost()**: actualiza los datos de un post en la base de datos.

## Problemas encontrados

---

- El primer problema con el que me encontré, fue a la hora de modificar el index.php para llamar a las otras páginas de la web.

En el caso de mi web, cuando me muevo de Login a Landing, por ejemplo, la ruta no cambia de login.php a landing.php, sino que siempre se queda estática como index.php.

Para poder realizar esto tuve que ir pasando siempre una url, en un input de tipo hidden, a la hora de ir cambiando de página. (En algunos casos tuve que pasar la url a través de un Header).

```
<!-- Url a la que iremos cuando se recargue la página -->
<input type="hidden" name="url" value="login">
```

De forma que, al recargar la página, volvíamos a index.php y aquí, establecía un require del trozo de html que quería hacer visible.

```
require $routes["routes"][$_REQUEST["url"]];
```

Teniendo en un fichero aparte todas las rutas y sus claves para acceder a ellas de manera más cómoda y rápida.

```
// Todas las rutas de las diferentes páginas de la web
return [
    "routes" => [
        "login" => "./parts/login.php",
        "adminPanel" => "./parts/adminPanel.php",
        "landing" => "./parts/landing.php",
        "register" => "./parts/register.php",
        "newTheme" => "./parts/newTheme.php",
        "post" => "./parts/post.php",
        "userConfig" => "./parts/userConfig.php",
        "updatePost" => "./parts/updatePost.php",
        "updatePerson" => "./parts/updatePerson.php"
    ]
];
```

- Otro problema con el que me encontré a medida que avanzaba con el proyecto fue a la hora de usar las variables de entorno y el autoloader, pues era algo que tuve que investigar por mi cuenta y ver de qué manera poder realizarlo correctamente.

## Trabajo a futuro y conclusiones

---

Como mejoras a futuro, me gustaría:

- Poder llevar un seguimiento de los cambios que realizan tanto usuarios como administradores, pues ahora mismo pueden eliminar y editar cosas, pero no queda un registro de esto.
- Mejorar la interfaz visual a una mucho más elaborada y atractiva.
- Poder responder también a los comentarios, de forma que se puede llevar una conversación en el post.
- Notificar a un usuario cuando alguien ha comentado en su post.

Como conclusión me llevo que, para la correcta realización de un proyecto ambicioso, es mucho mejor disponer de más tiempo, pues muchas partes quedarían más pulidas y con un mejor acabado técnico.

## Documentación

---

### *Requerimientos*

Para poder inicializar nuestro proyecto de manera local debemos tener:

- PHP 7.4.10 o superior.
- MariaDB 10.4.14.
- Apache/2.4.46.
- Bootstrap 5.
- Composer 2.5.1

### *Instalación*

Primeramente, debemos clonar el repositorio para tenerlo de manera local:

```
1 | $ git clone https://github.com/SaraCS21/High5-.git
```

Debemos tener en cuenta que nuestra aplicación hace uso de variables de entorno, así que para poder usar nuestra Base de Datos de manera local tendremos que crear un fichero `.env` en el directorio raíz, siguiendo la siguiente estructura:

```
1 | DB_HOST=host_que_queramos
2 | DB_USER=user_que_queramos
3 | DB_PASS=password_que_queramos
4 | DB_DB=Foro
```

Una vez lo tengamos listo, tendremos que instalar las dependencias para que nuestra aplicación pueda funcionar, para ello tendremos que ejecutar [composer](#):

```
1 | $ composer install
```

Tras eso, ejecutaremos el [index.php](#), este instalará de manera automática la base de datos del programa, además de inicializar una serie de parámetros por defecto para no ver la web vacía al comienzo.

Podemos ejecutar nuestra aplicación de la siguiente manera, estableciendo como puerto el que nosotros queramos:

```
1 | $ php -S localhost:3000
```

Con todo esto ya estaríamos listos para usar nuestra aplicación.

---

En caso de que queramos crear la documentación, tan solo tendremos que ejecutar el siguiente comando en la carpeta de nuestro proyecto:

```
1 | $ docker run --rm -v "$(pwd):/data" "phpdoc/phpdoc:3" --title High5! --ignore "vendor/"
```