



Proyecto de desarrollo de aplicaciones web

Jonay Martel Martel - 2ºDAW

Índice

1.- Descripción del proyecto	3
2.- Desarrollo del proyecto	4
1.1.- Requisitos	4
1.2 .- Análisis y diseño	4
1.3 .- Implementación	6
1.4.- Pruebas	22
1.4.- Despliegue	23
1.4.- Problemas, dificultades y tiempo dedicado	24
3.- TECNOLOGÍAS EMPLEADAS	25
4.- INTEGRACIÓN Y COSTE EMPRESARIAL	25
4.- ANEXOS	26
4.1.- Anexo I. Guía o Manual de Usuario	26
4.2.- Anexo II. DB Coding Standards	32
4.3.- Anexo III. DOC Guía Desarrollador	36
1 Objetivo	36
2 Alcance	36
3 Dependencias	36
4 Integración en la aplicación	36
4.1 Añadir referencias al proyecto	37
4.2 Uso de instancia de clases	37
4.3 Ejemplos de USO	37
4.4.- Anexo IV. DTS Guía Técnica	37
1. ARQUITECTURA DE LA SOLUCIÓN	37
1.1 Dependencias	38
1.1.1 Librerías	38
1.2 Entornos	38
2 INFRAESTRUCTURA UTILIZADA	38
2.1 Plataforma Software	38
3 NameSpace XXXXXXXXXXXX	38
3.1 Class Hospital	38
3.1.1 Constructores	38
3.1.2 Métodos	38

1.- Descripción del proyecto

Para mi proyecto, decidí enfocarme en el sistema de un hospital. Me pareció un tema súper interesante del que puedo obtener un montón de información útil y tablas que seguro serán interesantes. Los hospitales tienen un montón de datos importantes, como los historiales médicos de los pacientes, sus diagnósticos y tratamientos.

Lo que más me llama la atención es cómo manejan y guardan toda esa información en un lugar tan movido como un hospital. Es clave mantener seguros los datos de los pacientes y asegurarse de que todo el personal pueda acceder a la información que necesitan para cuidar a los pacientes de la mejor manera posible.

Además, analizar estos datos podría ayudar a descubrir patrones o tendencias que podrían hacer que los hospitales funcionen mejor. Por ejemplo, podría ayudar a los médicos a tomar decisiones más informadas o a los administradores a gestionar los recursos de manera más eficiente. En resumen, elegí el sistema de un hospital para mi proyecto porque creo que es un área súper interesante y llena de datos útiles que podrían mejorar la atención médica para todos.

He elegido este tema, debido a que es un tema que ya tengo fresco del año anterior. El cuál he usado esta misma temática para la asignatura BAE (base de datos), para hacer el proyecto de la asignatura que engloba cosas que vamos a usar en este mismo proyecto como viene siendo base de datos, mysql, cruds de tablas, etc.

Lo que vamos hacer es simular una pequeña gestión de una empresa llamada Salus Martel, que tiene en su poder varios hospitales por la isla de Gran Canaria, en la cuál vamos a llevar a cabo introducir datos de pacientes, del personal del hospital (médicos, etc). Las salas de espera con las que cuenta el hospital y muchas más.

Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

Una vez que hemos establecido la estructura básica de nuestro proyecto, es el momento de adentrarnos en el diseño arquitectónico. Aquí es donde entra en juego una herramienta fundamental en el desarrollo de software: el diagrama UML.

El diagrama UML Define proyectos por clases, atributos y funciones. Nos brinda una vista panorámica y detallada de las clases que componen nuestro sistema, así como de las relaciones entre ellas. Esta representación visual nos permite no solo visualizar la anatomía de nuestro proyecto, sino también entender cómo se relacionan sus partes móviles.

Al adentrarnos en la creación de un diagrama UML, estamos dando un paso crucial hacia la claridad y la coherencia en nuestro diseño. Cada clase, cada relación, cada método y cada atributo son como piezas de un rompecabezas que deben encajar perfectamente para que nuestro proyecto funcione sin problemas.

Una de las ventajas más significativas del diagrama UML es su capacidad para anticipar problemas potenciales antes de que ocurran. Al identificar con precisión los métodos necesarios en cada clase, podemos prever posibles puntos de fricción o inconsistencias en nuestro diseño. Esto nos permite abordar estos problemas en las etapas iniciales del desarrollo, cuando aún son más fáciles de corregir y menos costosos en términos de tiempo.

Al hacer este mapa, estamos construyendo los cimientos de nuestro proyecto. Cada parte es como un ladrillo en un edificio: necesitamos que encajen bien para que todo funcione correctamente. También nos ayuda a pensar en grande, a planificar cómo el proyecto puede crecer y adaptarse con el tiempo. En resumen, este mapa no es solo un dibujo bonito, sino una herramienta poderosa para tener claro nuestro proyecto.

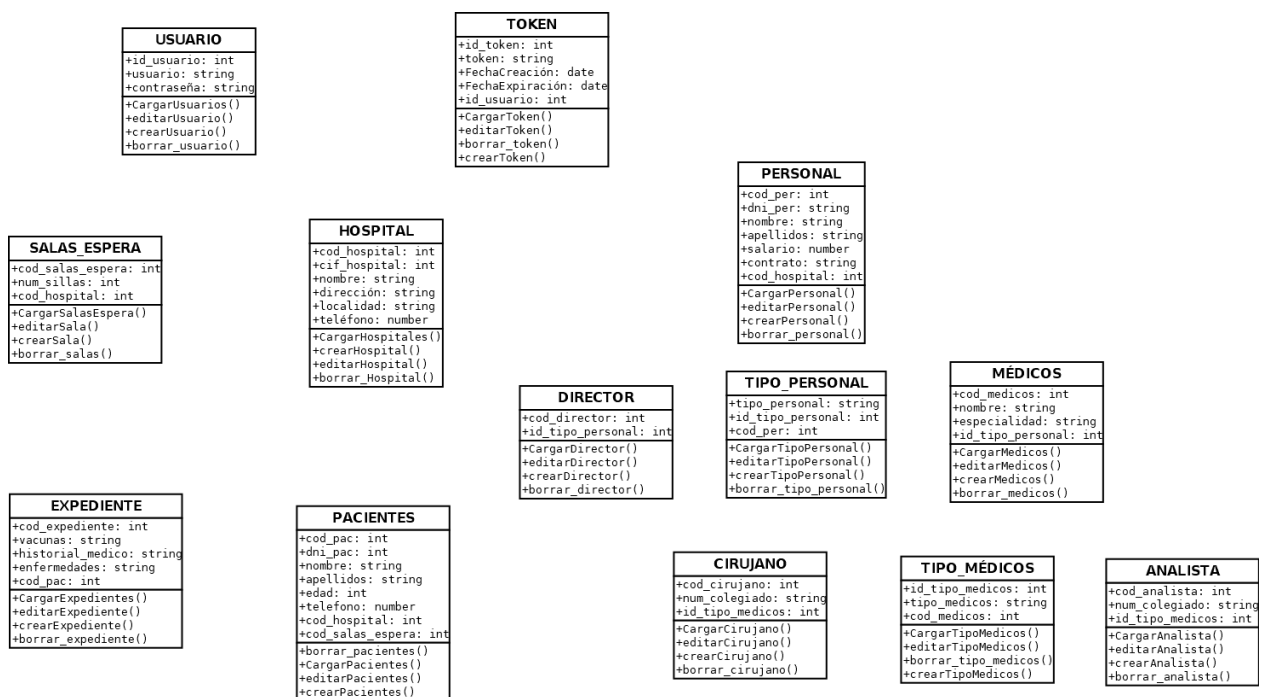


Imagen Diagrama E/R

1.3 .- Implementación

El siguiente paso sería abrir el Visual Studio Code que es el programa que he usado durante todo el curso y durante todo el ciclo para hacer mis proyectos. Una vez abierto vamos a ir creando la estructura de carpetas que vamos a necesitar para nuestro proyecto:

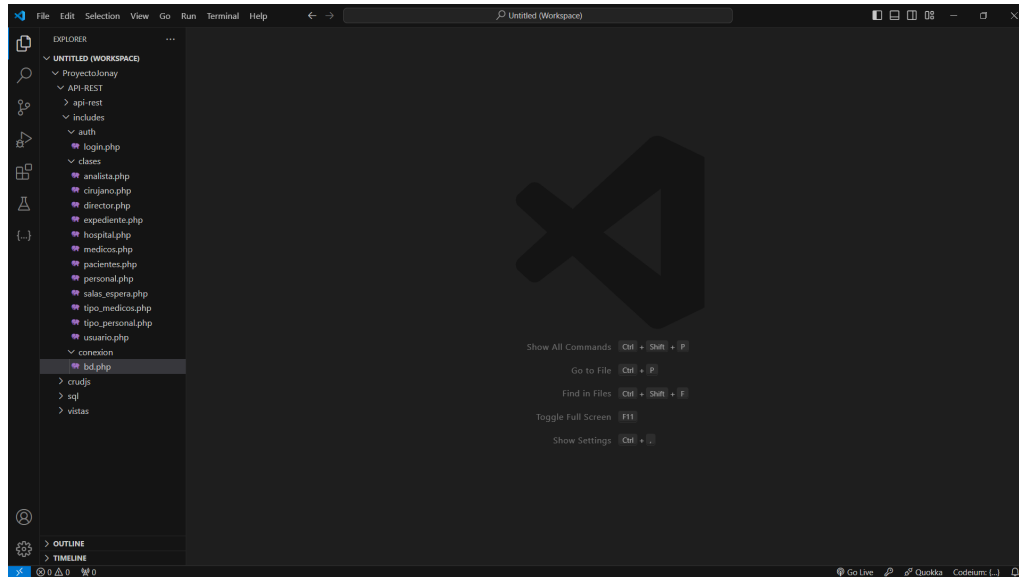


Imagen: Estructura de carpetas

A continuación, vamos a comenzar con la API-REST, primero necesitamos definir todas las clases que vamos a necesitar para manejar los diferentes recursos de nuestra aplicación. Estos recursos podrían ser usuarios, productos, publicaciones, etc. Cada recurso generalmente tiene su propia clase que define cómo interactuar con él a través de la API.

Una vez que hemos definido nuestras clases, el siguiente paso es establecer una conexión con la base de datos, ya que la mayoría de las API-REST operan sobre datos almacenados en una base de datos. En tu caso, mencionas que has creado un archivo de conexión con la base de datos llamado bd.php. Este archivo será responsable de establecer la conexión con tu base de datos MySQL a través de phpMyAdmin.

Dentro de bd.php, incluimos el código para establecer la conexión con la base de datos utilizando la extensión MySQLi o PDO en PHP. En él pondremos todos los detalles de conexión, como el nombre de usuario, la contraseña, el nombre de la base de datos y la dirección del servidor de la base de datos:

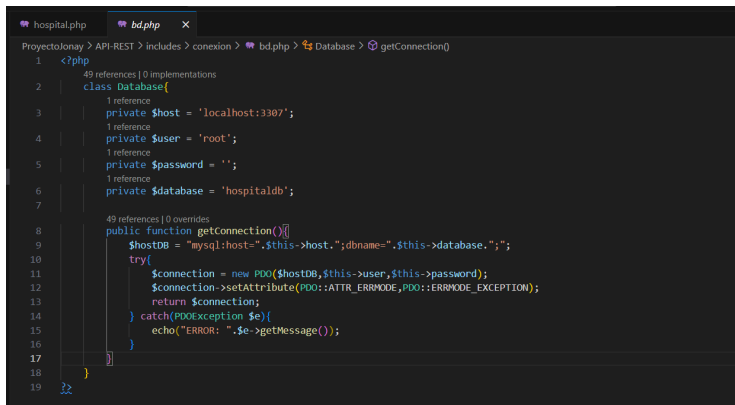


Imagen bd.php

Una vez que la conexión se haya establecido con éxito, podrás ejecutar consultas SQL para recuperar, actualizar, insertar o eliminar datos de tu base de datos según las solicitudes recibidas a través de tu API-REST.

La clase principal del proyecto es "Hospital". En la función mostrar_hospital(), se utiliza la clase "Database" para obtener la conexión con la base de datos.

Posteriormente, se ejecuta una consulta SQL para recuperar todos los registros de la tabla "hospital". Una vez obtenidos los datos, se devuelven en formato JSON para su posterior procesamiento. Esta función proporciona una manera de obtener una vista general de todos los hospitales almacenados en la base de datos.

En cuanto a la función crear_hospital(), el proceso es similar. Se utiliza nuevamente la clase "Database" para establecer la conexión con la base de datos. Luego, se prepara una consulta SQL para insertar un nuevo registro en la tabla "hospital" con los datos proporcionados. Después de ejecutar la consulta, se crea un nuevo hospital en la base de datos. Esta función permite agregar nuevos hospitales al sistema de manera eficiente y segura.

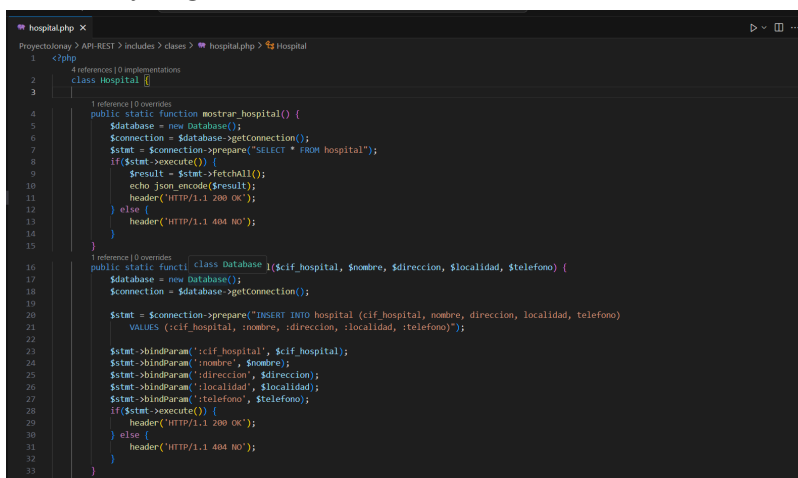
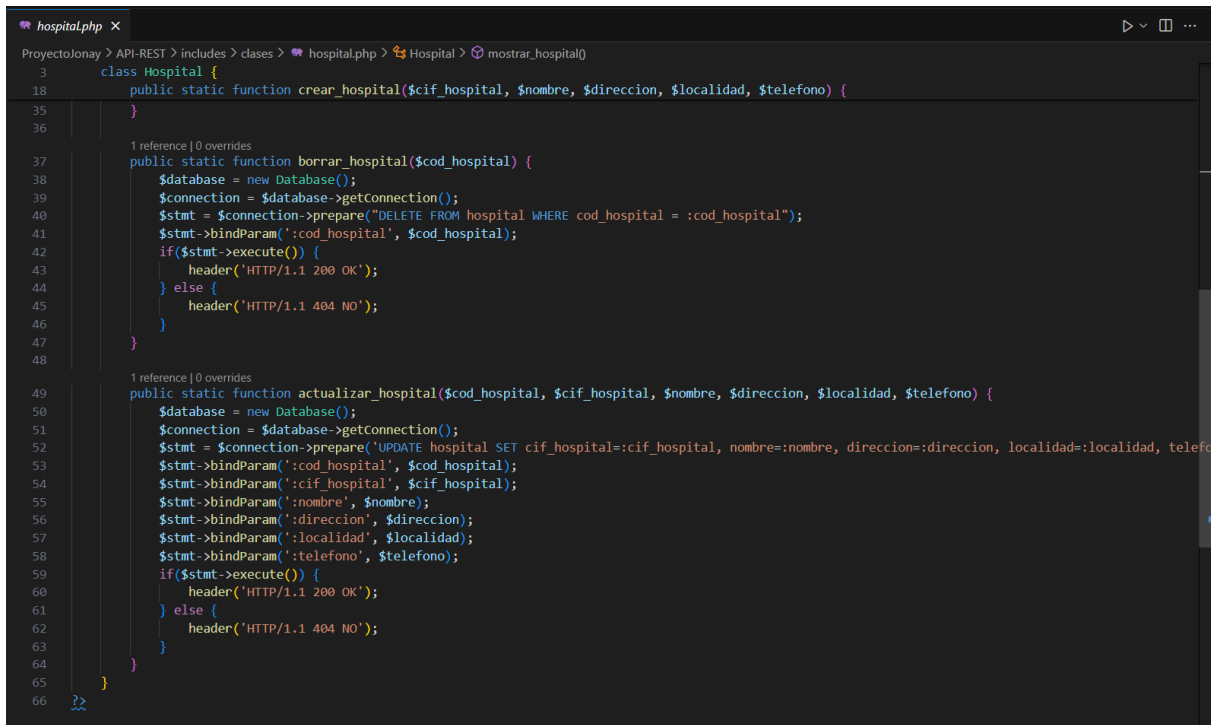


Imagen: hospital.php

En la función `borrar_hospital()`, se utiliza la clase "Database" para establecer la conexión con la base de datos. Una vez obtenida la conexión, se prepara una consulta SQL para eliminar un registro específico de la tabla "hospital" según el código proporcionado. Luego, se ejecuta la consulta, lo que resulta en la eliminación del hospital correspondiente en la base de datos. Esta función proporciona una manera segura de eliminar hospitales existentes del sistema según su código.

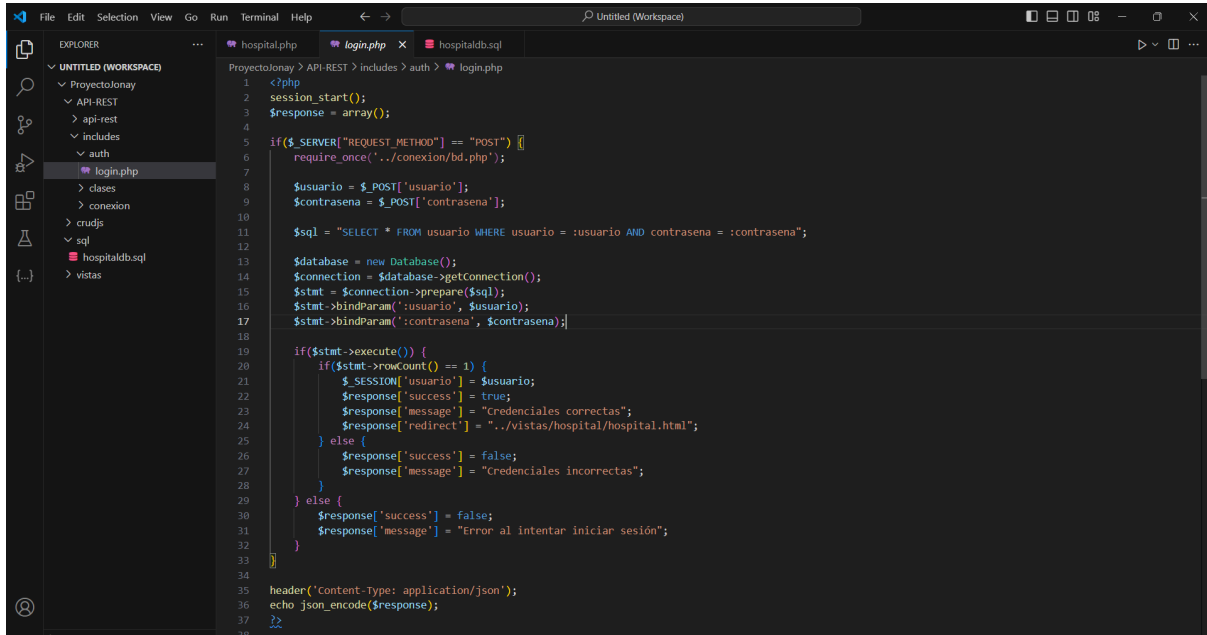
En cuanto a la función `actualizar_hospital()`, el proceso es similar a las anteriores. Se utiliza la clase "Database" para obtener la conexión con la base de datos. Luego, se prepara una consulta SQL para actualizar un registro específico de la tabla "hospital" con los nuevos datos proporcionados, incluyendo el código del hospital a actualizar. Después de ejecutar la consulta, se realizan los cambios en el registro correspondiente en la base de datos. Esta función es útil para modificar la información de un hospital existente, como su CIF, nombre, dirección, localidad o número de teléfono.



```
hospital.php x
ProyectoJonay > API-REST > includes > clases > hospital.php > Hospital > mostrar_hospital()
3      class Hospital {
18      public static function crear_hospital($cif_hospital, $nombre, $direccion, $localidad, $telefono) {
35      }
36
1 reference | 0 overrides
37      public static function borrar_hospital($cod_hospital) {
38          $database = new Database();
39          $connection = $database->getConnection();
40          $stmt = $connection->prepare("DELETE FROM hospital WHERE cod_hospital = :cod_hospital");
41          $stmt->bindParam(":cod_hospital", $cod_hospital);
42          if($stmt->execute()) {
43              header('HTTP/1.1 200 OK');
44          } else {
45              header('HTTP/1.1 404 NO');
46          }
47      }
48
1 reference | 0 overrides
49      public static function actualizar_hospital($cod_hospital, $cif_hospital, $nombre, $direccion, $localidad, $telefono) {
50          $database = new Database();
51          $connection = $database->getConnection();
52          $stmt = $connection->prepare("UPDATE hospital SET cif_hospital=:cif_hospital, nombre=:nombre, direccion=:direccion, localidad=:localidad, telefono=:telefono WHERE cod_hospital=:cod_hospital");
53          $stmt->bindParam(":cod_hospital", $cod_hospital);
54          $stmt->bindParam(":cif_hospital", $cif_hospital);
55          $stmt->bindParam(":nombre", $nombre);
56          $stmt->bindParam(":direccion", $direccion);
57          $stmt->bindParam(":localidad", $localidad);
58          $stmt->bindParam(":telefono", $telefono);
59          if($stmt->execute()) {
60              header('HTTP/1.1 200 OK');
61          } else {
62              header('HTTP/1.1 404 NO');
63          }
64      }
65  }
66  }
```

Imagen: hospital.php

También de paso, antes de mostrar los datos más adelante en las vistas html. Hay que iniciar sesión mediante un login mediante el siguiente código.



```
1 <?php
2 session_start();
3 $response = array();
4
5 if($_SERVER["REQUEST_METHOD"] == "POST") {
6     require_once('../conexion/bd.php');
7
8     $usuario = $_POST['usuario'];
9     $contrasena = $_POST['contrasena'];
10
11     $sql = "SELECT * FROM usuario WHERE usuario = :usuario AND contrasena = :contrasena";
12
13     $database = new Database();
14     $connection = $database->getConnection();
15     $stmt = $connection->prepare($sql);
16     $stmt->bindParam(':usuario', $usuario);
17     $stmt->bindParam(':contrasena', $contrasena);
18
19     if($stmt->execute()) {
20         if($stmt->rowCount() == 1) {
21             $_SESSION['usuario'] = $usuario;
22             $response['success'] = true;
23             $response['message'] = "Credenciales correctas";
24             $response['redirect'] = "../vistas/hospital/hospital.html";
25         } else {
26             $response['success'] = false;
27             $response['message'] = "Credenciales incorrectas";
28         }
29     } else {
30         $response['success'] = false;
31         $response['message'] = "Error al intentar iniciar sesión";
32     }
33 }
34
35 header('Content-Type: application/json');
36 echo json_encode($response);
37
38
```

Imagen: login.php

Este código PHP maneja una solicitud POST para iniciar sesión de usuario. Primero, verifica si la solicitud es de tipo POST y luego obtiene los datos de usuario y contraseña enviados. Luego, realiza una consulta SQL para buscar un usuario en la base de datos que coincida con los datos proporcionados. Si encuentra un usuario, establece una variable de sesión y devuelve una respuesta JSON con un indicador de éxito y un mensaje personalizado dependiendo de la respuesta que reciba, así como la URL del fichero html a la que se redirigirá después del inicio de sesión. En caso contrario, devuelve un mensaje de error y seguirá en la misma pantalla a la espera que se introduzcan unos datos correctos para poder iniciar sesión. Finalmente, envía la respuesta JSON al cliente. Más adelante cuando lleguemos a la parte de explicar los crud en js, explicaré la función que usamos para tratar el tema del inicio de sesión.

Vamos a continuar con la creación de la API-REST, ya tenemos hechas las clases, he de comentar primero de todo que solo he mostrado cómo sería la clase “Hospital” pero realmente las otras 10 clases que tengo es exactamente lo mismo. Lo único que cambiaría serían los atributos de cada una.

Cada clase tendría sus propios métodos que permitirían realizar acciones específicas relacionadas con esa clase, como agregar, recuperar, actualizar o eliminar registros de la base de datos. Pero el patrón general de cómo se implementaría las clases en la API REST para interactuar sería el mismo.

Ahora seguimos con el “crud” de la API-REST de la tabla “Hospital” que es la que estamos llevando a cabo como ejemplo, esta sería la carpeta donde tengo guardado los ficheros de la tabla, en el resto de las carpetas es totalmente lo mismo lo que los ficheros adaptados a su tabla:

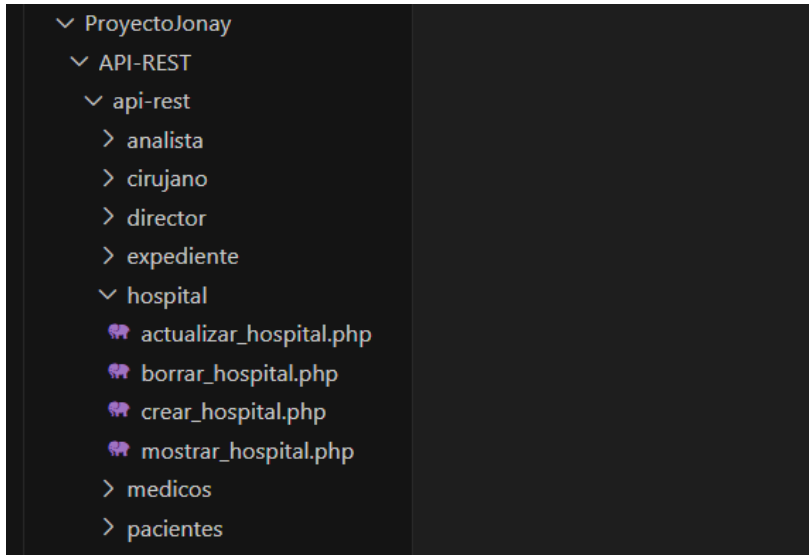


Imagen: Estructura de fichero de la tabla “Hospital”

En este fichero nos vamos a encargar de escribir el código necesario para actualizar la información de nuestro hospital. Para hacer esto, primero de todo incluimos en el fichero la clase “hospital”, también vamos a incluir el fichero de conexión. Seguido de esto nos aseguramos que la solicitud sea del tipo correcto, tipo PUT que sería el correcto para el momento de cuando tengamos que editar una tabla. Si todo está en orden, utiliza esos datos para llamar al método que actualiza la información del hospital en la base de datos.

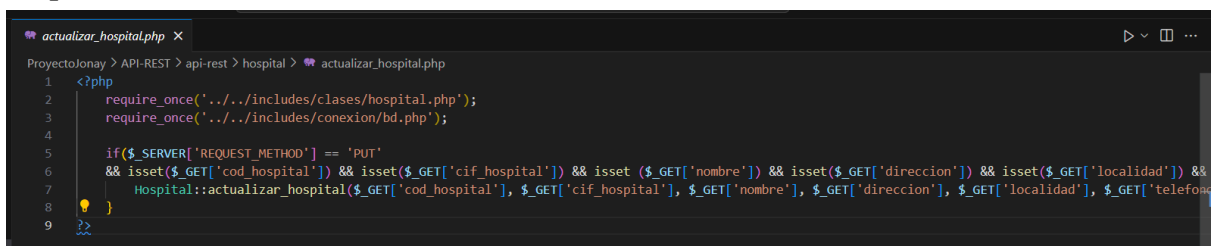
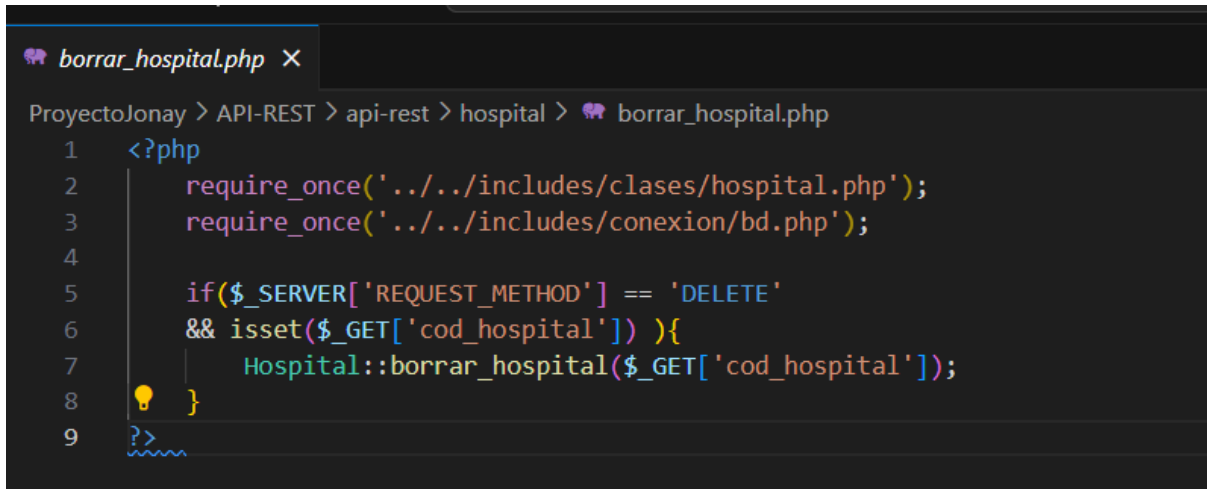


Imagen: actualizar_hospital.php

En este fichero nos vamos a encargar de escribir el código necesario para eliminar la información de nuestro hospital cuando recibe una solicitud DELETE. Primero, incluye dos archivos necesarios: uno para importar la clase Hospital, que contiene los métodos para interactuar con la base de datos, y otro para la conexión a la base de datos. Luego, verifica si la solicitud HTTP es de tipo DELETE y si contiene el parámetro 'cod_hospital' en la URL. Si ambas condiciones se cumplen, llamamos al método necesario para borrar nuestro hospital “borrar_hospital()” de la clase Hospital,

pasándole el código del hospital a eliminar. Este método se encarga de eliminar el registro correspondiente de la base de datos.



```
borrar_hospital.php X
ProyectoJonay > API-REST > api-rest > hospital > borrar_hospital.php
1  <?php
2      require_once('../includes/clases/hospital.php');
3      require_once('../includes/conexion/bd.php');
4
5      if($_SERVER['REQUEST_METHOD'] == 'DELETE'
6      && isset($_GET['cod_hospital'])) ){
7          Hospital::borrar_hospital($_GET['cod_hospital']);
8      }
9  ?>
```

Imagen:borrar_hospital.php

En el archivo crear_hospital.php, nos enfocamos en facilitar la creación de nuevos registros hospitalarios. Comenzamos incluyendo la clase "Hospital" y el archivo de conexión necesario para interactuar con la base de datos. Luego, verificamos si la solicitud entrante es del tipo POST. Si se cumple esta condición, procedemos a decodificar el JSON recibido en la solicitud. Posteriormente, validamos si se han proporcionado todos los datos necesarios para crear un nuevo hospital, como el CIF, nombre, dirección, localidad y teléfono. En caso afirmativo, llamamos al método crear_hospital() de la clase, pasando los datos correspondientes. De esta manera, aseguramos un proceso claro y estructurado para la incorporación de nuevos registros en nuestra base de datos.

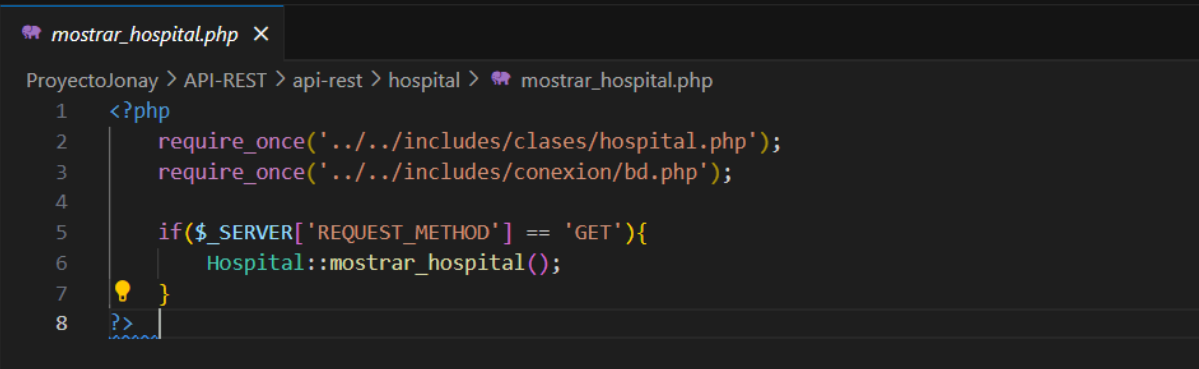


```
crear_hospital.php X
ProyectoJonay > API-REST > api-rest > hospital > crear_hospital.php
1  <?php
2      require_once('../includes/clases/hospital.php');
3      require_once('../includes/conexion/bd.php');
4
5      if($_SERVER['REQUEST_METHOD'] == 'POST' ){
6
7          $json = json_decode(file_get_contents("php://input"), true);
8
9          if(isset($json['cif_hospital']) && isset($json['nombre']) && isset($json['direccion']) && isset($json['localidad']) && isset($json['telefono'])) ){
10             Hospital::crear_hospital($json['cif_hospital'], $json['nombre'], $json['direccion'], $json['localidad'], $json['telefono']);
11          }
12      }
13
14  ?>
```

Imagen: crear_hospital.php

En este fichero nos vamos a encargar de escribir el código necesario para actualizar la información de nuestro hospital cuando recibe una solicitud PUT. Primero, incluye dos archivos necesarios: uno para importar la clase Hospital, que contiene los métodos para interactuar con la base de datos, y otro para la conexión a la base de datos. Luego, verifica si la solicitud HTTP es de tipo PUT y si contiene el parámetro. Al

verificar que la solicitud entrante sea de tipo GET, hacemos la llamada al método `mostrar_hospital()` de nuestra clase, el cual contiene la lógica para recuperar y mostrar en el formulario html los datos hospitalarios. Esta estructura garantiza un proceso coherente y seguro para acceder a la información hospitalaria, proporcionando una visión ordenada y validada de los registros disponibles.



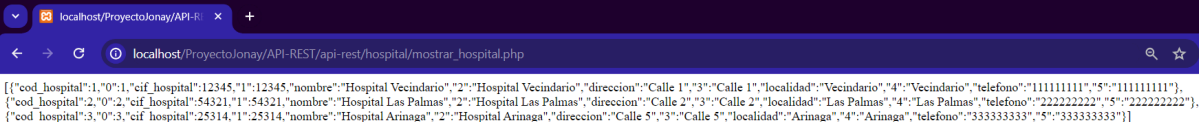
```
1 <?php
2     require_once('../includes/clases/hospital.php');
3     require_once('../includes/conexion/bd.php');
4
5     if($_SERVER['REQUEST_METHOD'] == 'GET'){
6         Hospital::mostrar_hospital();
7     }
8     ?>
```

Imagen: actualizar_hospital.php

Después de haber implementado el CRUD para la clase "Hospital", he seguido el mismo proceso para las otras 10 clases del proyecto. Ahora, llega el momento crucial de probar la funcionalidad de la Api-rest. Existen varias formas de hacerlo, pero he identificado dos métodos especialmente efectivos.

Primera, poniendo la ruta de la Api-rest en tu buscador. Ya sea chrome, opera o el mismo edge, todos sirven. Ponemos la ruta por ejemplo:

`localhost/ProyectoJonay/API-REST/api-rest/hospital/mostrar_hospital.php`, como la siguiente imagen:



```
[{"cod_hospital":1,"0":1,"cif_hospital":12345,"1":12345,"nombre":"Hospital Vecindario","2":"Hospital Vecindario","direccion":"Calle 1","3":"Calle 1","localidad":"Vecindario","4":"Vecindario","telefono":"111111111","5":"111111111"}, {"cod_hospital":2,"0":2,"cif_hospital":54321,"1":54321,"nombre":"Hospital Las Palmas","2":"Hospital Las Palmas","direccion":"Calle 2","3":"Calle 2","localidad":"Las Palmas","4":"Las Palmas","telefono":"222222222","5":"222222222"}, {"cod_hospital":3,"0":3,"cif_hospital":25314,"1":25314,"nombre":"Hospital Arinaga","2":"Hospital Arinaga","direccion":"Calle 5","3":"Calle 5","localidad":"Arinaga","4":"Arinaga","telefono":"333333333","5":"333333333"}]
```

La segunda opción implica el uso de una aplicación llamada Postman. Postman es una herramienta popular entre los desarrolladores para probar APIs. Permite enviar diversas peticiones a servicios web y visualizar las respuestas de manera clara y organizada, facilitando así la depuración y el análisis de la funcionalidad de la API. Postman, es una gran herramienta que facilita el trabajo a la hora de probar y comprobar el funcionamiento de una API-REST. Porque si, se podría hacer como en la primera opción pero lo bueno de Postman es que viene con la opción de que tu mismo puedes seleccionar la solicitud (DELETE, POST, PUT o GET), lo que hace que todo sea más profesional y puedes comprobar el funcionamiento por ejemplo del fichero `mostrar_hospital.php` con la solicitud "DELETE" por ejemplo, cuando para mostrar tendría que ser el GET:

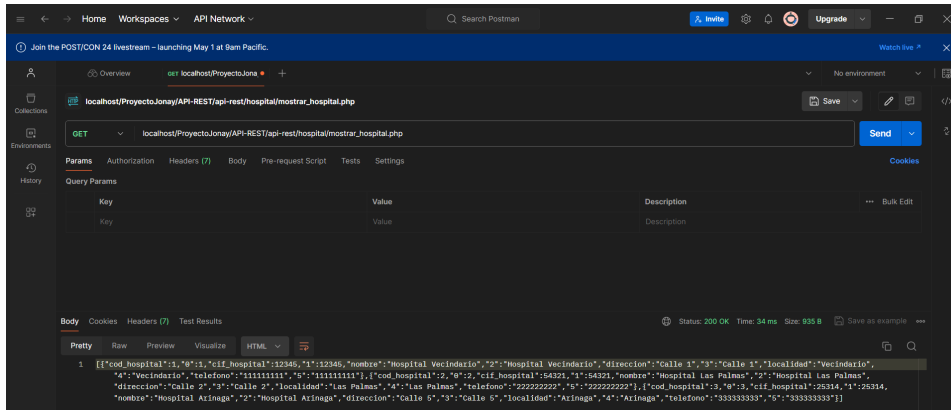


Imagen: Comprobación método get para mostrar_hospital.php

Hasta el momento, solo hemos realizado una comprobación utilizando la función `mostrar_hospital()`, pero es necesario extender esta comprobación para todas las funciones de la clase `hospital` y para todas las funciones de las clases de nuestro proyecto. Se planifica realizar un conjunto completo de pruebas para cada función definida en el sistema. Esto incluye pruebas exhaustivas para asegurar que cada función se comporte según lo esperado en diversas situaciones.

Una vez completadas las pruebas de las funciones, el siguiente paso será iniciar el desarrollo del archivo `funciones.js`. En este archivo, se implementará un conjunto completo de operaciones CRUD (Crear, Leer, Actualizar y Borrar) para la clase en cuestión. Estas operaciones se realizarán a través de llamadas a la API desarrollada hasta el momento.

El objetivo final es tener un sistema completamente funcional que permita interactuar con la base de datos desde el frontend. Esto implica la capacidad de crear, leer, actualizar y eliminar registros, todo ello manipulando la base de datos desde los ficheros `.html` que mostraremos al final del todo.

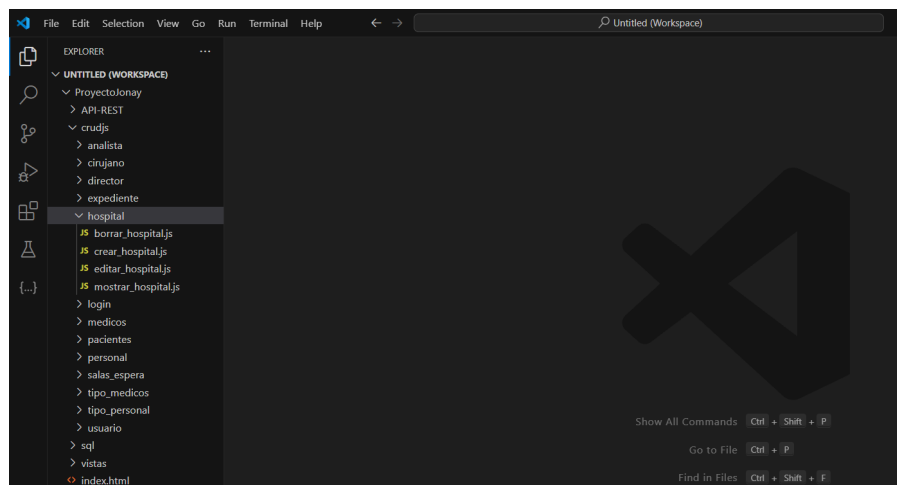


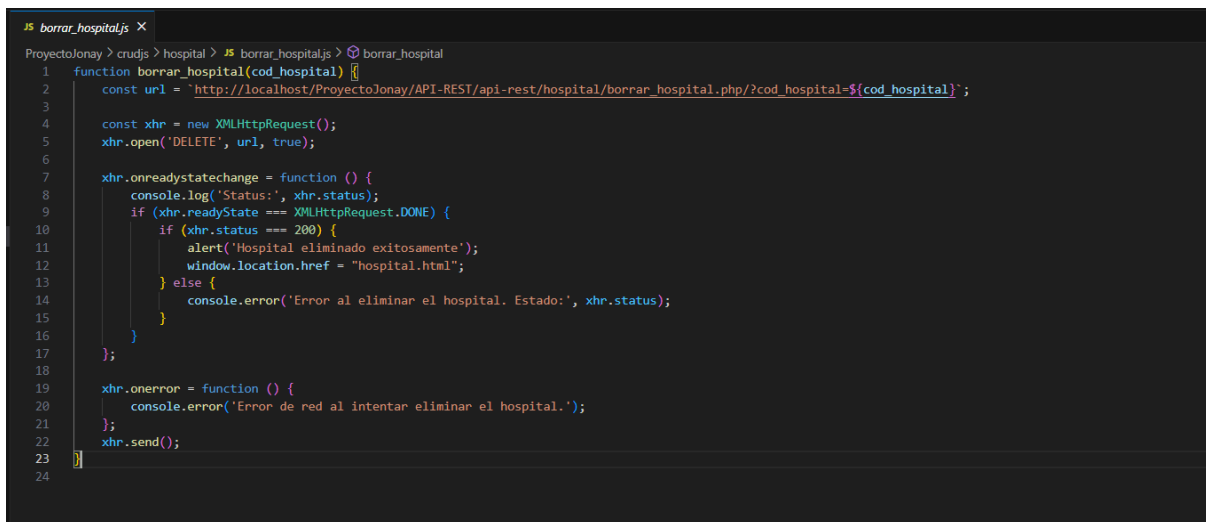
Imagen: Estructura de las carpetas .js

Como se puede comprobar en la imagen tengo todo organizado en las carpetas y cada carpeta es una clase de las que tenemos en el proyecto. Al final esto lo que hace es que tienes también los ficheros separados y cada función separada por su fichero. La parte buena de esto es que cualquier error de una función o de una funcionalidad pues al tenerlo todo más organizado es más fácil localizarla y a la hora de intentar explicarle a alguien es más fácil o si simplemente le envías tu proyecto a alguien puede localizar las funciones/métodos más fácilmente.

Ahora vamos a empezar a explicar los métodos de la clase de los ficheros.js.

La función `borrar_hospital` implementada en JavaScript se encarga de enviar una solicitud DELETE a una API REST con el objetivo de eliminar un hospital específico en función del código de hospital proporcionado. Esta función primero de todo hace una llamada a la Api mediante la url y realiza la solicitud DELETE utilizando la interfaz XMLHttpRequest.

Una vez realizada la solicitud, se manejan diferentes escenarios: si la eliminación es exitosa, se muestra una alerta al usuario informando sobre el éxito de la operación y se redirige a la página `hospital.html` que es la página principal de la clase. En caso de errores durante el proceso, como fallos en la red o problemas en la API, se muestran diversos mensajes de error que se le muestra a los usuarios que están poniendo en práctica el programa para facilitar el diagnóstico y la depuración del problema.



```
1 function borrar_hospital(cod_hospital) {
2   const url = 'http://localhost/ProyectoJonay/API-REST/api-rest/hospital/borrar_hospital.php?cod_hospital=${cod_hospital}';
3
4   const xhr = new XMLHttpRequest();
5   xhr.open('DELETE', url, true);
6
7   xhr.onreadystatechange = function () {
8     console.log('Status:', xhr.status);
9     if (xhr.readyState === XMLHttpRequest.DONE) {
10      if (xhr.status === 200) {
11        alert('Hospital eliminado exitosamente');
12        window.location.href = "hospital.html";
13      } else {
14        console.error('Error al eliminar el hospital. Estado:', xhr.status);
15      }
16    }
17  };
18
19  xhr.onerror = function () {
20    console.error('Error de red al intentar eliminar el hospital. ');
21  };
22  xhr.send();
23 }
24
```

Imagen: `borrar_hospital.js`

Seguimos ahora con la función crearHospital está diseñada para enviar una solicitud POST a una API REST con el fin de crear un nuevo hospital en el sistema. Primero de todo hacemos una llamada a la API mediante la URL y establece la configuración de la solicitud utilizando la interfaz XMLHttpRequest. Se especifica que el tipo de contenido que se enviará en la solicitud es JSON. Después de enviar la solicitud, se manejan los eventos onreadystatechange y onerror para gestionar diferentes escenarios de respuesta del servidor y posibles errores de red. Si la creación del hospital es exitosa, se muestra una alerta al usuario informándole del éxito de la operación y se redirige a la página hospital.html que es la página principal. En caso de errores, se registran mensajes detallados de error en la consola del navegador para facilitar la identificación y resolución de problemas.

Además, se adjunta un evento al evento DOMContentLoaded del documento, que se dispara cuando se ha cargado completamente la estructura del documento HTML. Este evento escucha el envío del formulario de creación de hospital y evita su envío predeterminado utilizando event.preventDefault(). Luego, extrae los datos del formulario, los organiza en un objeto JavaScript y llama a la función crearHospital para iniciar el proceso de creación del hospital mediante la API REST. Este enfoque asegura una interacción fluida con el usuario y una gestión eficiente de la creación de hospitales en la aplicación.

```
JS crear_hospital.js X
ProyectoJonay > crudjs > hospital > JS crear_hospital.js > crearHospital
1 function crearHospital(HospitalNuevo) {
2   const url = 'http://localhost/ProyectoJonay/API-REST/api-rest/hospital/crear_hospital.php';
3   const xhr = new XMLHttpRequest();
4   xhr.open('POST', url, true);
5   xhr.setRequestHeader('Content-Type', 'application/json');
6
7   xhr.onreadystatechange = function () {
8     console.log('Status:', xhr.status);
9     if (xhr.readyState === XMLHttpRequest.DONE) {
10      if (xhr.status === 200) {
11        alert('Hospital creada exitosamente');
12        window.location.href = 'hospital.html';
13      } else {
14        console.error('Error al crear la hospital. Estado:', xhr.status);
15      }
16    }
17  };
18
19  xhr.onerror = function () {
20    console.error('Error de red al intentar crear la hospital.');
```

Imagen: crear_hospital.js

Continuamos con el crud vamos con la función `editarHospital` que se encarga de la actualización de la información de un hospital específico existente en la base de datos a través de una solicitud PUT a la API REST. Esta función construye la URL de la API para la actualización del hospital, configura la solicitud utilizando `XMLHttpRequest` y envía los datos actualizados del hospital en formato JSON. Una vez completada la solicitud, se manejan los eventos `onreadystatechange` y `onerror` para gestionar diferentes escenarios de respuesta del servidor y posibles errores de red. Si la actualización del hospital es exitosa, se muestra una alerta al usuario informándole del éxito de la operación y se redirige a la página `hospital.html` que es la página principal. En caso de errores, se registran mensajes detallados de error en la consola del navegador para facilitar la identificación y resolución de problemas.

Además, se define la función `obtenerDatosHospital`, que realiza una solicitud GET a la API REST para obtener los datos del hospital correspondientes al código de hospital proporcionado. Una vez obtenidos los datos del hospital, se llamará a la función `llenarFormulario` para poblar automáticamente el formulario de edición con la información del hospital. Finalmente, se adjunta un evento al evento `DOMContentLoaded` del documento para iniciar el proceso de carga de datos del hospital y actualizarlo según corresponda. Este enfoque garantiza una experiencia fluida al usuario durante la edición de hospitales en la aplicación, al tiempo que proporciona una gestión eficiente de los datos del hospital en el backend y frontend.

```
1 function editarHospital(cod_hospital, hospitalActualizado) {
2     const url = 'http://localhost/ProyectoJonay/API-REST/api-rest/hospital/actualizar_hospital.php/?cod_hospital=${cod_hospital}';
3     const xhr = new XMLHttpRequest();
4     xhr.open('PUT', url, true);
5     xhr.setRequestHeader('Content-Type', 'application/json');
6
7     xhr.onreadystatechange = function () {
8         console.log('Status:', xhr.status);
9         if (xhr.readyState === XMLHttpRequest.DONE) {
10             if (xhr.status === 200) {
11                 alert('Hospital actualizado exitosamente');
12                 window.location.href = "hospital.html";
13             } else {
14                 console.error('Error al actualizar el hospital. Estado:', xhr.status);
15             }
16         }
17     };
18
19     xhr.onerror = function () {
20         console.error('Error de red al intentar actualizar el hospital.');
```

```
21 };
22
23 xhr.send(JSON.stringify(hospitalActualizado));
24
25
26 function obtenerDatosHospital(cod_hospital) {
27     const url = 'http://localhost/ProyectoJonay/API-REST/api-rest/hospital/mostrar_hospital.php/?cod_hospital=${cod_hospital}';
28     const xhr = new XMLHttpRequest();
29     xhr.open('GET', url, true);
30
31     xhr.onreadystatechange = function () {
32         if (xhr.readyState === XMLHttpRequest.DONE) {
33             if (xhr.status === 200) {
34                 const hospital = JSON.parse(xhr.responseText);
35                 llenarFormulario(hospital);
36             } else {
37                 console.error('Error al cargar los datos de la sala. Estado:', xhr.status);
38             }
39         }
40     };
41
42     xhr.onerror = function () {
43         console.error('Error de red al intentar cargar los datos de la sala.');
```

```
44 };
45
46 xhr.send();
47 }
```



```
function llenarFormulario(hospital) {
    document.getElementById('cod_hospital').value = hospital.cod_hospital;
    document.getElementById('cif_hospital').value = hospital.cif_hospital;
    document.getElementById('nombre').value = hospital.nombre;
    document.getElementById('direccion').value = hospital.direccion;
    document.getElementById('localidad').value = hospital.localidad;
    document.getElementById('telefono').value = hospital.telefono;
}

document.addEventListener('DOMContentLoaded', function () {
    const editarHospitalBtn = document.getElementById('editarHospitalBtn');
    if (editarHospitalBtn) {
        editarHospitalBtn.addEventListener('click', function () {
            const formData = new FormData(document.getElementById('formularioHospital'));
            const hospitalActualizado = {};

            for (const [key, value] of formData.entries()) {
                hospitalActualizado[key] = value;
            }

            const urlParams = new URLSearchParams(window.location.search);
            const cod_hospital = urlParams.get('cod_hospital');

            if (cod_hospital) {
                editarHospital(cod_hospital, hospitalActualizado);
            } else {
                console.error('No se proporcionó un ID válido para editar el hospital.');
```

Imagen: editar_hospital.js

La función CargarHospitales se encarga de realizar una solicitud GET a la API REST para obtener la lista de hospitales almacenados en la base de datos. Una vez obtenida la respuesta del servidor, la función maneja diferentes escenarios de respuesta utilizando los eventos onreadystatechange y onerror de XMLHttpRequest. Si la solicitud se completa con éxito (status === 200), los datos de los hospitales se procesan y se muestra una tabla en el elemento HTML con id "Resultados" esto irá al hospital.html para que se muestren los datos. Cada fila de la tabla representa un hospital y contiene información como el CIF, nombre, dirección, localidad y teléfono del hospital.

Además, ponemos enlaces que se dirigen a los ficheros/formularios para editar y otro que dirige al fichero que contiene la función para eliminar un hospital. Si la solicitud no se completa con éxito, se muestran mensajes de error dependiendo del error con mensajes de alerta para facilitar la comprensión sobre lo que está ocurriendo por parte del Usuario. También se adjunta un evento al evento DOMContentLoaded del documento para llamar a la función CargarHospitales() una vez que se ha cargado completamente la estructura del documento HTML. Esto sirve para que nada más el usuario inicie sesión y se vaya a hospital.html se le muestre directamente el listado de todos los hospitales guardados en la base de datos.

```
hospital.php > # mostrar_hospitals X
ProyectoJonay > crud > hospital > # mostrar_hospitals > ...
1 function CargarHospitales() {
2     const xhr = new XMLHttpRequest();
3     const url = 'http://localhost/ProyectoJonay/API-REST/api-rest/hospital/mostrar_hospital.php';
4     xhr.open('GET', url, true);
5
6     xhr.onreadystatechange = function () {
7         console.log('Estado:', xhr.status);
8         if (xhr.readyState === XMLHttpRequest.DONE) {
9             if (xhr.status === 200) {
10                 const hospitales = JSON.parse(xhr.responseText);
11                 const hospitalesContainer = document.getElementById('Resultados');
12                 hospitalesContainer.innerHTML = '';
13
14                 let tableHTML =
15                     <table class="table">
16                         <tr>
17                             <th>C.I.F./th>
18                             <th>Nombre/th>
19                             <th>Dirección/th>
20                             <th>Localidad/th>
21                             <th>Teléfono/th>
22                         </tr>
23                     </table>
24
25                 hospitales.forEach(function (hospital) {
26                     tableHTML +=
27                         <tr>
28                             <td>${hospital.cif_hospital}</td>
29                             <td>${hospital.nombre}</td>
30                             <td>${hospital.direccion}</td>
31                             <td>${hospital.localidad}</td>
32                             <td>${hospital.telefono}</td>
33                             <td>
34                                 <a href="formularioeditahospital.html?cod_hospital=${hospital.cod_hospital}" class="btn btn-primary">Editar</a>
35                                 <button class="btn btn-danger" onclick="borrar_hospital(${hospital.cod_hospital})">Eliminar</button>
36                             </td>
37                         </tr>
38                     </tr>
39                 </tr>
40             </tr>
41             </table>
42             hospitalesContainer.innerHTML = tableHTML;
43         } else {
44             console.error('Error al cargar los hospitales. Estado:', xhr.status);
45         }
46     };
47
48     xhr.onerror = function () {
49         console.error('Error de red al cargar los hospitales.');
```

Imagen: crear_hospital.js

Vale, pues ya hemos acabado con el crud de hospital. Pero antes de continuar voy a sacar un ejemplo de un selector que tengo en la tabla pacientes, esto lo que hace y para lo que sirve es para cuando tu vayas a crear un paciente y lo quieras asignar a un hospital ya existente te salga un selector con todos los hospitales que hay y el nombre del hospital lo que hace más sencilla su elección.

Esto se hace con una función que la mía se llama CargarHospitales(), realiza una solicitud GET a la API mediante una llamada por URL y configura la solicitud usando “XMLHttpRequest” y luego envía esos datos mediante un formato JSON

Y así se vería los 3 hospitales que se muestran son los 3 hospitales existentes en la base de datos:

Código de hospital

Selecciona un hospital

Selecciona un hospital

1 - Hospital Vecindario

2 - Hospital Las Palmas

3 - Hospital Arinaga

Y esta sería la imagen del código:

```
hospital.php JS crear_pacientes.js
ProyectoJonay > crudjs > pacientes > JS crear_pacientes.js > document.addEventListener('DOMContentLoaded') callback > cargarHospitales > onload > data.forEach() callback
26 document.addEventListener('DOMContentLoaded', function () {
27   form.addEventListener('submit', function (event) {
28     crearPacientes(PacienteNuevo);
29   });
30 });
31
32 function cargarHospitales() {
33   const url = 'http://localhost/ProyectoJonay/API-REST/api-rest/hospital/mostrar_hospital.php';
34   try {
35     const xhr = new XMLHttpRequest();
36     xhr.open('GET', url, true);
37     xhr.onload = function() {
38       if (xhr.status === 200) {
39         const data = JSON.parse(xhr.responseText);
40         const selectHospital = document.getElementById('cod_hospital');
41         data.forEach(hospital => {
42           const option = document.createElement('option');
43           option.value = hospital.cod_hospital;
44           option.textContent = `${hospital.cod_hospital} - ${hospital.nombre}`;
45           selectHospital.appendChild(option);
46         });
47       } else {
48         throw new Error('Error al cargar los hospitales.');
```

Bueno, una vez terminado el crud.js vamos con los ficheros html, primero muestro la estructura usada, es la que se muestra en la siguiente imagen. De paso aprovecho y muestro uno de los ficheros más importante el index.html que es el que sirve para iniciar sesión nada más pongas en el buscador localhost/ProyectoJonay:

```
File Edit Selection View Go Run Terminal Help
ProyectoJonay > index.html > html
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
7   <link rel="stylesheet" href=".../ProyectoJonay/vistas/css/css.css">
8   <script src=".../ProyectoJonay/crudjs/login/login.js"></script>
9   <title>Iniciar Sesión</title>
10 </head>
11 <body>
12   <h2>Iniciar Sesión</h2>
13   <form id="login-form" method="post">
14     <div>
15       <label for="usuario">Usuario:</label>
16       <input type="text" id="usuario" name="usuario">
17     </div>
18     <div>
19       <label for="contrasena">Contraseña:</label>
20       <input type="password" id="contrasena" name="contrasena">
21     </div>
22     <div>
23       <button type="submit">Iniciar Sesión</button>
24     </div>
25   </form>
26   <script>
27     document.addEventListener('DOMContentLoaded', function () {
28       const scripts = document.querySelectorAll('script[src$=".js"]');
29       if (scripts.length === 0) {
30         const messageContainer = document.createElement('div');
31         messageContainer.classList.add('alert', 'alert-danger');
32         messageContainer.textContent = 'No se encontraron archivos .js, sin ellos no se podrán mostrar los datos ni hacer nada';
33         document.body.appendChild(messageContainer);
34       }
35     });
36   </script>
37 </body>
38 </html>
```

Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

Luego, empezamos con el hospital.html, primero de todo abrimos la etiqueta html, luego metemos el link de estilos de css y de bootstrap. Y a continuación, seguimos con los ficheros js que nos serán necesarios para poder mostrar los resultados, cuando mostremos esta vista, si seguimos nos encontramos con la cabecera para que la accesibilidad sea total y se pueda navegar de una vista a otra con un solo click, si seguimos bajando nos encontramos con un div que contiene un id="resultados" esto sirve para mostrar los datos que se cargan en el mostrar_hospital.js.

Y por último tenemos un script cuya finalidad es que salte un mensaje de error cuando no se detecten ficheros.js.

```
hospital.html X
ProjectJonay > vistas > hospital > hospital.html > html > body > div.cabecera > a
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
7 <link rel="stylesheet" href="css/css.css">
8 <title>Lista de Hospitales</title>
9 <script src=".../crudjs/hospital/borrar_hospital.js"></script>
10 <script src=".../crudjs/hospital/mostrar_hospital.js"></script>
11
12 </head>
13 <body>
14 <div class="cabecera">
15 <a href=".../usuario/usuario.html">Usuarios</a>
16 <a href=".../hospital/hospital.html">Hospitales</a>
17 <a href=".../salas_espera/salas_espera.html">Salas de Espera</a>
18 <a href=".../pacientes/pacientes.html">Pacientes</a>
19 <a href=".../expediente/expediente.html">Expedientes</a>
20 <a href=".../personal/personal.html">Personal</a>
21 <a href=".../tipo_personal/tipo_personal.html">Tipo Personal</a>
22 <a href=".../medicos/medicos.html">Medicos</a>
23 <a href=".../director/director.html">Directores</a>
24 <a href=".../tipo_medicos/tipo_medicos.html">Tipo Medicos</a>
25 <a href=".../analista/analista.html">Analistas</a>
26 <a href=".../cirujano/cirujano.html">Cirujanos</a>
27 <a href="formularioaddhospital.html">Añadir Hospital</a>
28 </div>
29
30 <div>
31 <h2>Lista de Hospitales</h2>
32 <div id="Resultados"></div>
33 </div>
34 <script>
35 document.addEventListener('DOMContentLoaded', function () {
36   const scripts = document.querySelectorAll('script[src$=".js"]');
37   if (scripts.length === 0) {
38     const messageContainer = document.createElement('div');
39     messageContainer.classList.add('alert', 'alert-danger');
40     messageContainer.textContent = 'No se encontraron archivos .js, sin ellos no se podrán mostrar los datos';
41     document.body.appendChild(messageContainer);
42   }
43 });
44 </script>
45 </body>
46 </html>
```

Este es el formulario para editar los campos de la clase Hospital en la base de datos:

```
hospital.html
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
7 <link rel="stylesheet" href="css/css.css">
8 <title>Proyecto PMJ Jonay</title>
9 <script src=".../crudjs/hospital/editar_hospital.js"></script>
10 </head>
11 <body>
12 <div class="cabecera">
13 <a href=".../usuario/usuario.html">Usuarios</a>
14 <a href=".../hospital/hospital.html">Hospitales</a>
15 <a href=".../salas_espera/salas_espera.html">Salas de Espera</a>
16 <a href=".../pacientes/pacientes.html">Pacientes</a>
17 <a href=".../expediente/expediente.html">Expedientes</a>
18 <a href=".../personal/personal.html">Personal</a>
19 <a href=".../tipo_personal/tipo_personal.html">Tipo Personal</a>
20 <a href=".../medicos/medicos.html">Medicos</a>
21 <a href=".../director/director.html">Directores</a>
22 <a href=".../tipo_medicos/tipo_medicos.html">Tipo Medicos</a>
23 <a href=".../analista/analista.html">Analistas</a>
24 <a href=".../cirujano/cirujano.html">Cirujanos</a>
25 </div>
26
27 <div class="container">
28 <h2>Datos de Hospital</h2>
29 <form id="form-editar-hospital">
30 <div class="form-group">
31 <label for="cod_hospital">Código del Hospital</label>
32 <input type="text" class="form-control" id="cod_hospital" name="cod_hospital" placeholder="" readonly>
33 </div>
34 <div class="form-group">
35 <label for="cif_hospital">CIF del Hospital</label>
36 <input type="text" class="form-control" id="cif_hospital" name="cif_hospital" placeholder="" required>
37 </div>
38 <div class="form-group">
39 <label for="nombre">Nombre del Hospital</label>
40 <input type="text" class="form-control" id="nombre" name="nombre" placeholder="" required>
41 </div>
42 <div class="form-group">
43 <label for="direccion">Dirección del Hospital</label>
44 <input type="text" class="form-control" id="direccion" name="direccion" placeholder="" required>
45 </div>
46 <div class="form-group">
47 <label for="localidad">Localidad del Hospital</label>
48 <input type="text" class="form-control" id="localidad" name="localidad" placeholder="" required>
49 </div>
50 <div class="form-group">
51 <label for="telefono">Teléfono</label>
52 <input type="text" class="form-control" id="telefono" name="telefono" placeholder="" required>
53 </div>
54 <button type="button" class="btn btn-primary" id="editarHospital">Editar Hospital</button>
55 <a href="hospital.html" class="btn btn-secondary">Volver</a>
56 </form>
57 </div>
```

Y este es el formulario para añadir un hospital nuevo:

```
hospital.html  formularioadHospital.html X
ProjectJonay > vistas > hospital > formularioadHospital.html > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
7      <link rel="stylesheet" href="/css/css.css">
8      <title>Proyecto PWA Jonay Villal</title>
9      <script src="/js/crudjs/hospital/crear_hospital.js"></script>
10 </head>
11 <body>
12     <div class="navbar">
13         <a href="/usuario/usuario.html">Usuarios</a>
14         <a href="/hospital/hospital.html">Hospitales</a>
15         <a href="/salas_espera/salas_espera.html">Salas de Espera</a>
16         <a href="/pacientes/pacientes.html">Pacientes</a>
17         <a href="/expediente/expediente.html">Expedientes</a>
18         <a href="/personal/personal.html">Personal</a>
19         <a href="/tipo_personal/tipo_personal.html">Tipo Personal</a>
20         <a href="/medicos/medicos.html">Medicos</a>
21         <a href="/director/director.html">Directores</a>
22         <a href="/tipo_medicos/tipo_medicos.html">Tipo Medicos</a>
23         <a href="/analista/analista.html">Analistas</a>
24         <a href="/cirujano/cirujano.html">Cirujanos</a>
25     </div>
26
27     <div class="container">
28         <h2>Añadir Hospital</h2>
29         <form id="formularioHospital">
30             <div class="form-group">
31                 <label for="cif_hospital">CIF del hospital</label>
32                 <input type="number" class="form-control" id="cif_hospital" name="cif_hospital" placeholder="" required>
33             </div>
34             <div class="form-group">
35                 <label for="nombre">Nombre del hospital</label>
36                 <input type="text" class="form-control" id="nombre" name="nombre" placeholder="" required>
37             </div>
38             <div class="form-group">
39                 <label for="direccion">Dirección del hospital</label>
40                 <input type="text" class="form-control" id="direccion" name="direccion" placeholder="" required>
41             </div>
42             <div class="form-group">
43                 <label for="localidad">Localidad del hospital</label>
44                 <input type="text" class="form-control" id="localidad" name="localidad" placeholder="" required>
45             </div>
46             <div class="form-group">
47                 <label for="telefono">Teléfono</label>
48                 <input type="number" class="form-control" id="telefono" name="telefono" placeholder="" required>
49             </div>
50             <button type="submit" class="btn btn-primary">Crear Hospital</button>
51             <a href="/hospital.html" class="btn btn-secondary">Volver</a>
52         </form>
53     </div>
54 </body>
55 </html>
```

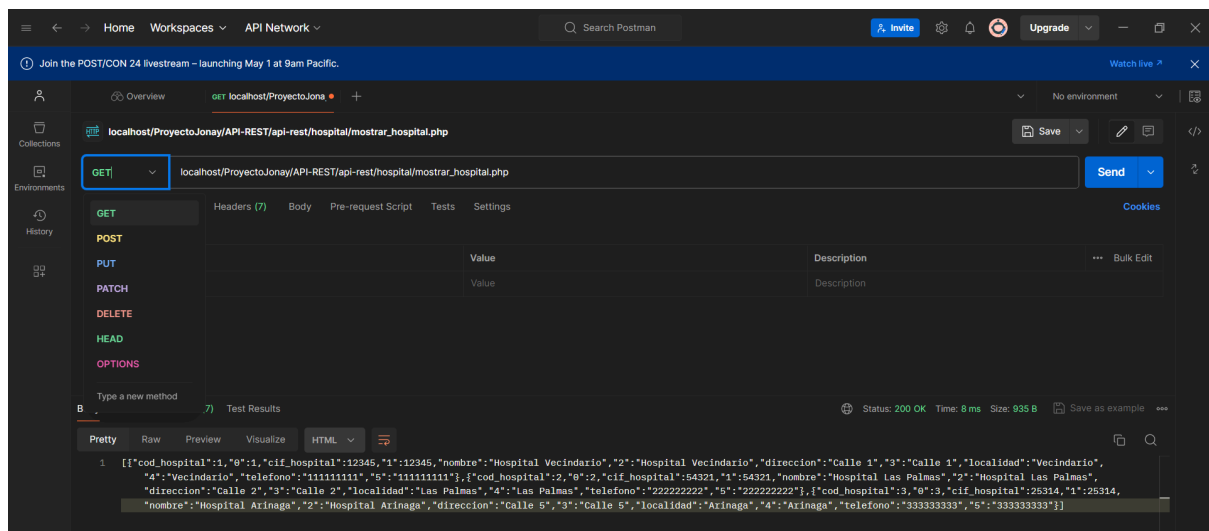
Además, voy a hacer un inciso aquí como hice para mostrar el tema del selector, si nos vamos por ejemplo al formulario de pacientes, así se mostraría un selector en html:

```
<div class="form-group">
    <label for="cod_hospital">Codigo de hospital</label>
    <select class="form-control" id="cod_hospital" name="cod_hospital" required>
        <option value="" selected disabled>Selecciona un hospital</option>
    </select>
</div>
```

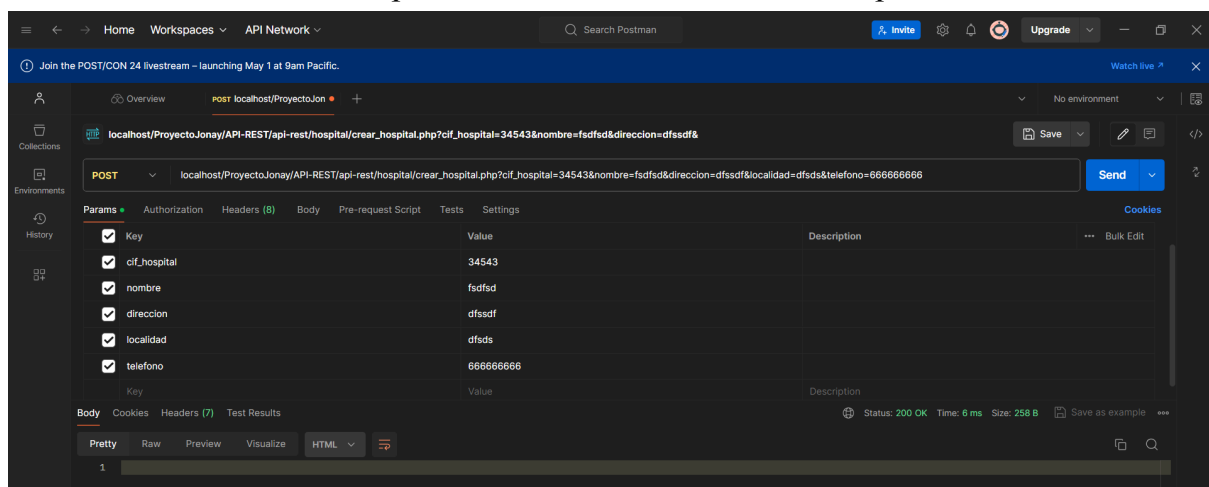
1.4.- Pruebas

Para hacer las pruebas he usado sobre Postman. Ayuda mucho, Postman es una herramienta popular entre los desarrolladores para probar APIs. Permite enviar diversas peticiones a servicios web y visualizar las respuestas de manera clara y organizada, facilitando así la depuración y el análisis de la funcionalidad de la API. Una de las partes principales de postman es que viene con la opción de que tu mismo puedes seleccionar la solicitud (DELETE, POST, PUT o GET), lo que hace que todo sea más profesional y puedas comprobar el funcionamiento. Y puedes comprobar el funcionamiento de una solicitud DELETE con un fichero que tiene en su código una solicitud POST por ejemplo.

He aquí un ejemplo de lo que digo tienes un panel con diversas opciones de solicitudes, esta por ejemplo se trata de una solicitud GET que muestra los datos de todos los hospitales que dispone mi base de datos



Ahora vamos a crear un hospital nuevo con la solicitud POST, ponemos todos :



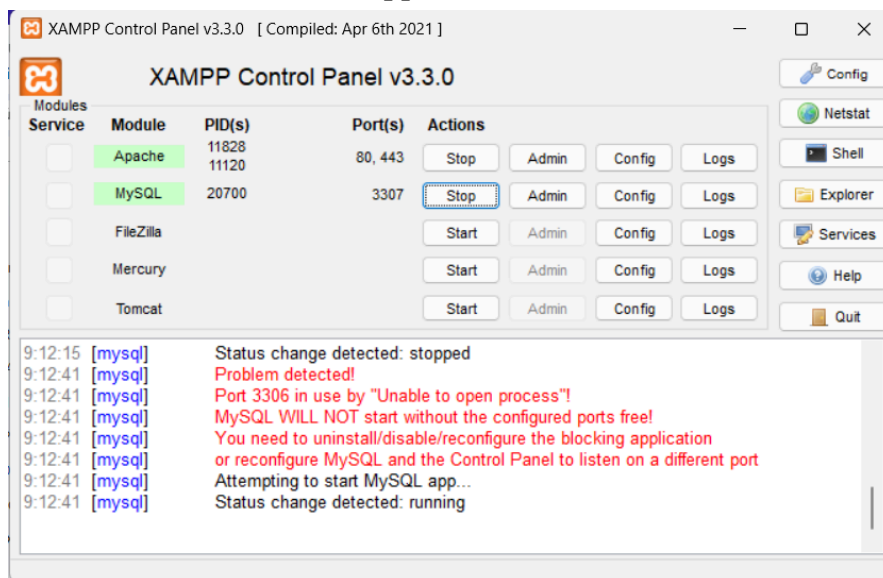
Y ya estaría creada si ponemos otra vez la solicitud GET y comprobamos:

```
15 {
16     "cod_hospital": 4,
17     "g": 4,
18     "cif_hospital": 34543,
19     "i": 34543,
20     "nombre": "Esdfsd",
21     "2": "Esdfsd",
22     "direccion": "dfssdf",
23     "3": "dfssdf",
24     "localidad": "dfsds",
25     "4": "dfsds",
26     "telefono": "666666666",
27     "5": "666666666"
28 }
```

Y lo mismo con editar y con borrar

1.4.- Despliegue

Para ello vamos a usar xampp:

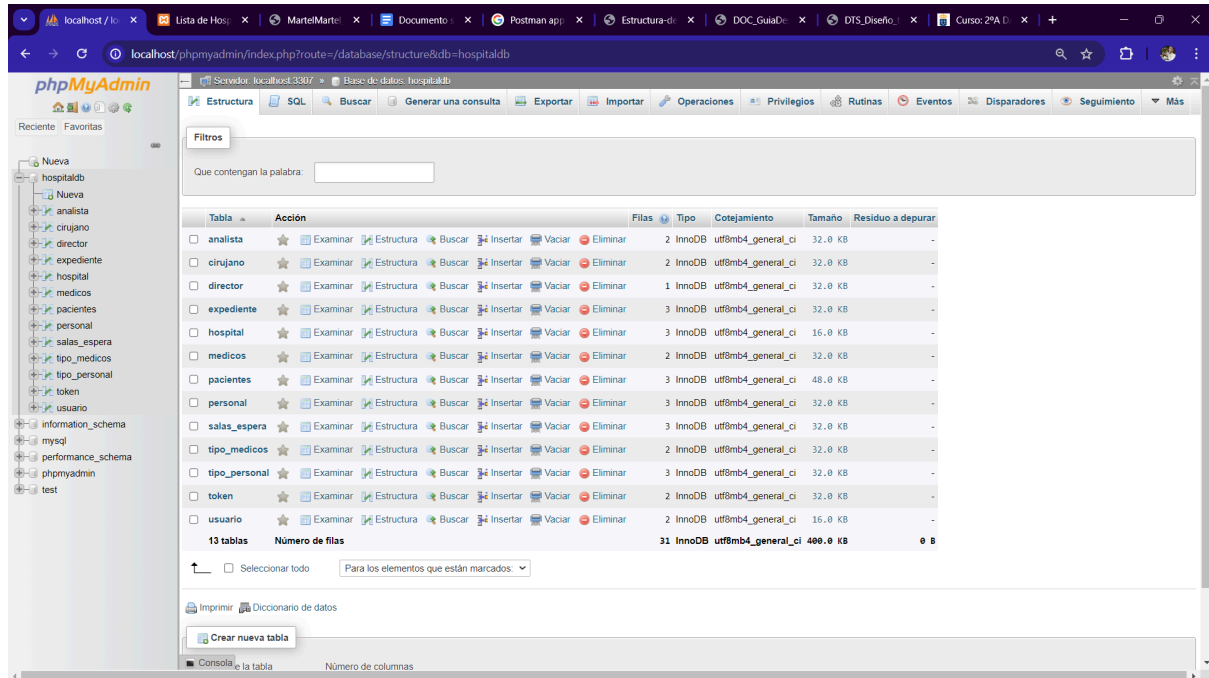


Antes de nada, nos tenemos que fijar muy bien en el puerto ya que es importante para la conexión con la base de datos en phpmyadmin, vamos a ir a bd.php y ponerle el puerto que tengamos en el xampp, y ponemos el usuario y la contraseña si es que tenemos en nuestro caso pues no tenemos contraseña puesta:

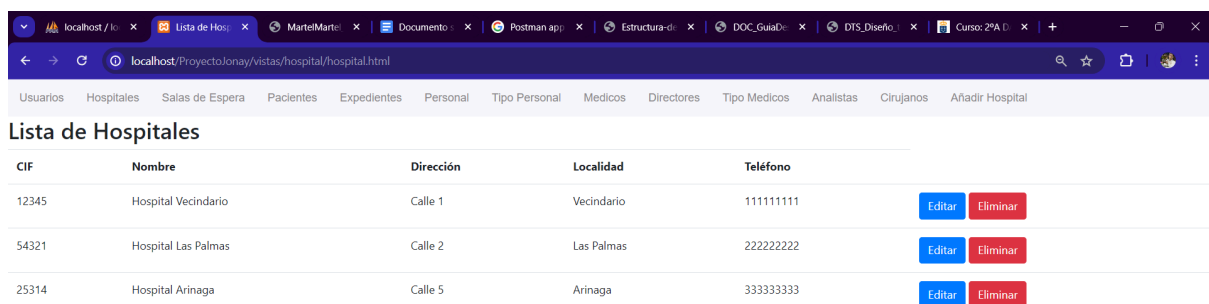
```
<?php
49 references | 0 implementations
class Database{
    1 reference
    private $host = 'localhost:3307';
    1 reference
    private $user = 'root';
    1 reference
    private $password = '';
    1 reference
    private $database = 'hospitaldb';
```

Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

Ahora entramos en el panel de control de phpmyadmin y seleccionamos crear base de datos y le ponemos el mismo nombre que el que tenemos en bd.php y metemos el sql dentro y aquí tendríamos nuestra base de datos importada:



Y también ya podríamos ver los datos en nuestra web:



1.4.- Problemas, dificultades y tiempo dedicado

La verdad, los problemas vinieron con el paso del tiempo porque al principio los diagramas no me costaron nada porque ya era una idea que la tenía clara desde el año pasado que hice un trabajo del mismo tipo. Pero desde que empecé con la API ya se empezó a complicar el asunto. Al final del todo solo me quedó hacer el inicio de sesión mediante Token y la parte de editar que al final nunca conseguí hacerla por un problema que la Api no lee bien los campos y me muestra “undefined” cuando muestro el formulario con los datos.

Y luego en cuanto a tiempo pues desde que empezemos en febrero hasta el último día, lo más tiempo ha llevado es el tema de la documentación pero al final es una parte importante para el trabajo y el mundo de la programación.

3.- TECNOLOGÍAS EMPLEADAS

Sobre todo videos de Youtube, tutoriales, etc. Me han servido mucho aparte de que puedes ver como hacen algunos ejemplos y además pues lo explican detalladamente y al igual que otros videos pues salen errores que a ti también te salían y pues lo solucionas

Otra tecnología que he empleado y que recomiendo mucho es <https://stackoverflow.com/> es una web muy interesante y en resumen se trata de una comunidad de preguntas y respuestas para programadores y aficionados a la programación. Está diseñado para ayudar a las personas a encontrar soluciones a problemas de programación y compartir conocimientos sobre el lenguaje de programación y la tecnología de la información. En los últimos años la verdad que se ha vuelto muy famosa, es verdad que no resuelve problemas muy grande pero para pequeños remedios esta de lujo

Stack Overflow fue creado para ayudar a los programadores a encontrar respuestas a sus preguntas, compartir conocimientos y aprender nuevas habilidades informáticas. Cuando alguien hace una pregunta en Stack Overflow, la responde una comunidad de programadores y expertos en informática. Estas respuestas se organizan y votan para garantizar que los usuarios reciban la mejor información posible.

4.- INTEGRACIÓN Y COSTE EMPRESARIAL

La integración de la API-REST de un hospital dentro de una empresa ofrece numerosos beneficios y aportaciones. En primer lugar, al brindar una plataforma centralizada para acceder y gestionar los datos del hospital, la API facilita la coordinación entre diferentes departamentos y profesionales médicos. Esto mejora la eficiencia operativa y la atención al paciente, al tiempo que reduce los errores y duplicaciones de información.

Además, al implementar esta solución tecnológica, la empresa se posiciona a la vanguardia en términos de innovación y digitalización en el sector de la salud. Esto puede resultar atractivo para los clientes y socios comerciales, fortaleciendo la reputación de la empresa y generando nuevas oportunidades de negocio.

En cuanto a los costos, es importante considerar tanto el hardware como el software necesarios para implementar y mantener nuestra API. Esto incluye servidores, almacenamiento de datos, licencias de software, seguridad informática, entre otros. Realizar una comparativa y elección de precios entre diferentes opciones de hosting, servidores, VPS, licencias y otros servicios relacionados nos permitirá optimizar los recursos y reducir los costos operativos a largo plazo.

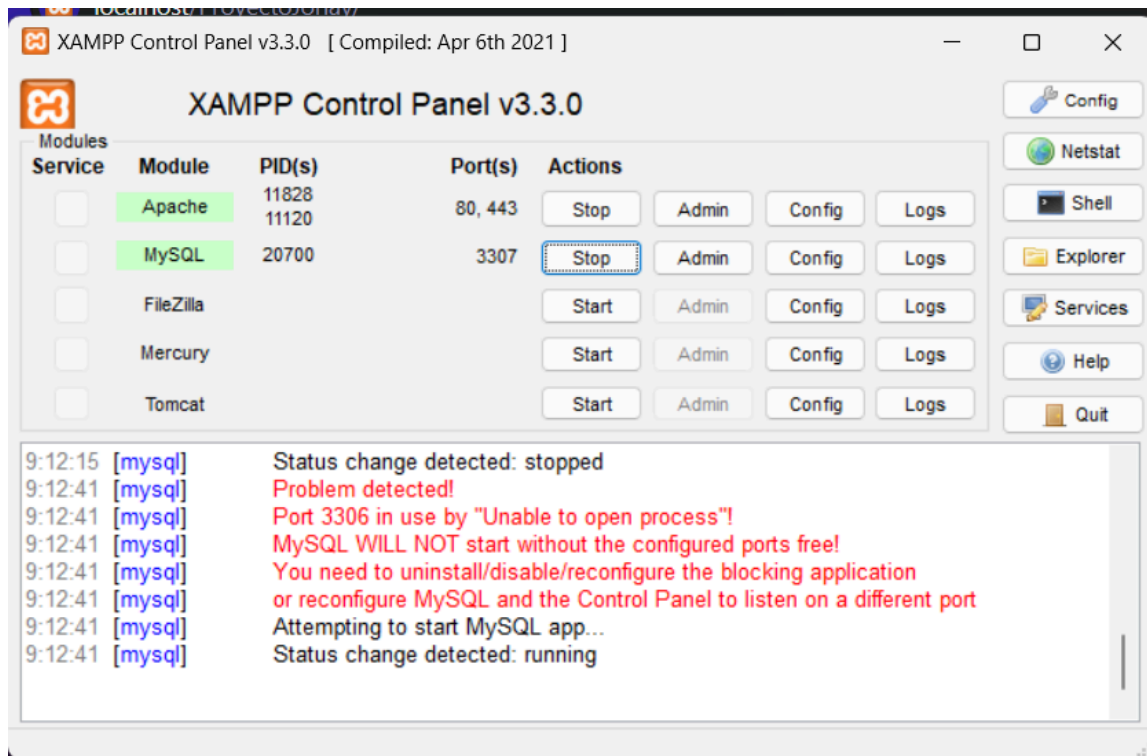
Además, no podemos pasar por alto los costos asociados con la seguridad y la protección de datos, que son especialmente críticos en el sector de la salud. La implementación de medidas de seguridad robustas y copias de seguridad regulares garantizará la integridad y confidencialidad de la información del hospital, protegiendo tanto a los pacientes como a la empresa de posibles riesgos y sanciones legales.

En cuanto a costes para hacer este trabajo; Económicamente ninguno pero en cuanto a tiempo bastante, ya no solo horas sino días y meses de trabajo.

4.- ANEXOS

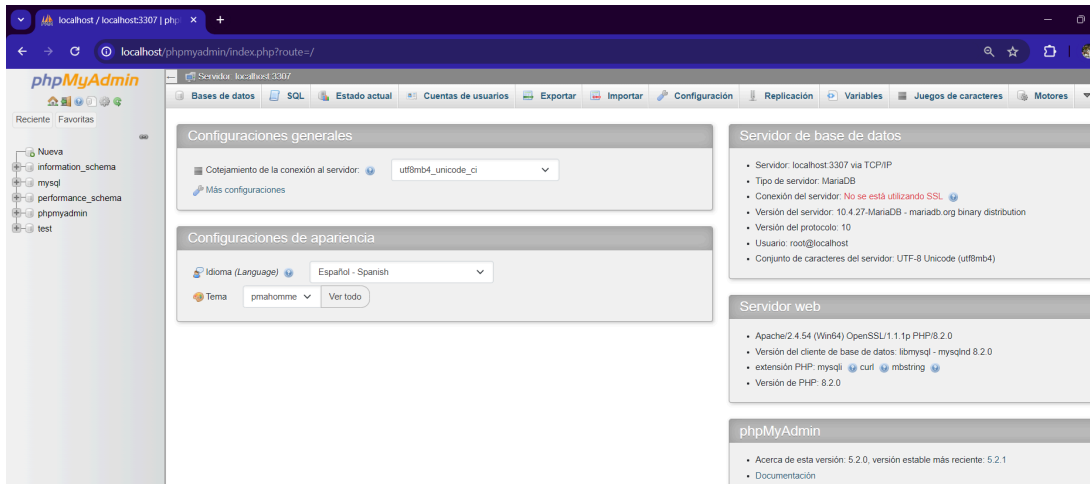
4.1.- Anexo I. Guía o Manual de Usuario

- Primero de todo vamos a encender el xampp, paso fundamental:

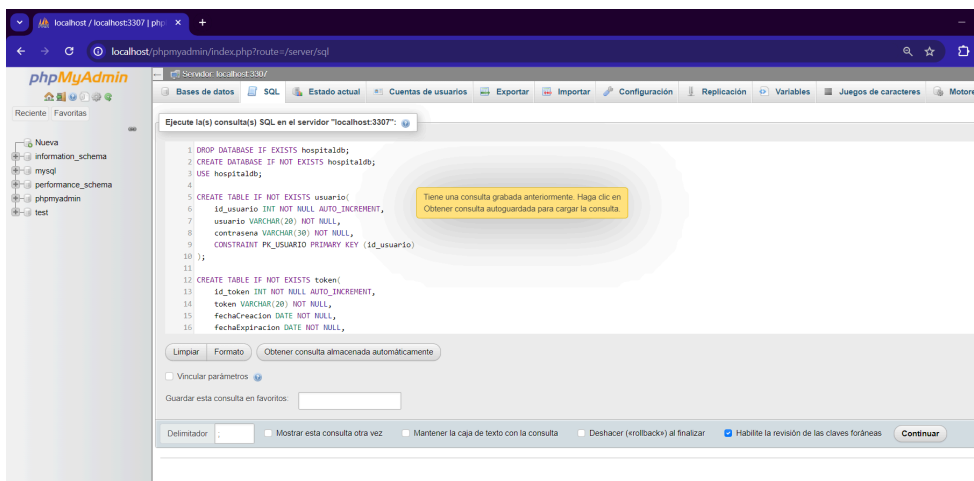


Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

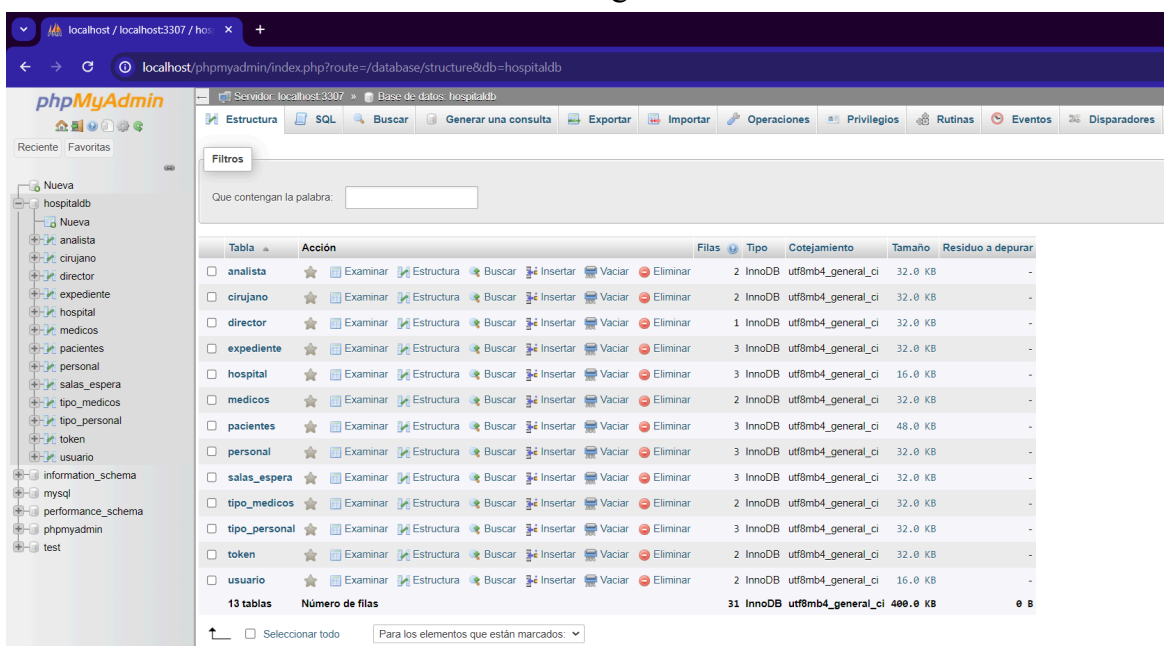
- Luego de ello, pulsamos en el botón admin en la fila de MySQL y nos mandara al panel de control de MySQL:



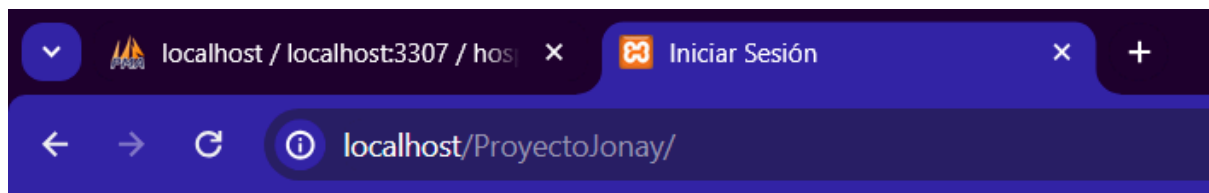
- Ahora vamos al apartado SQL y metemos nuestro código SQL:



- Ya tenemos nuestra base de datos cargadas:



- Ahora vamos a abrir otra pestaña y ponemos localhost/ProyectoJonay:

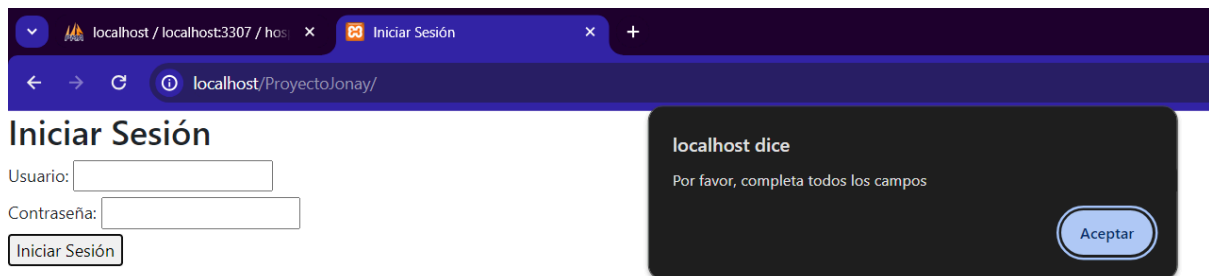


Iniciar Sesión

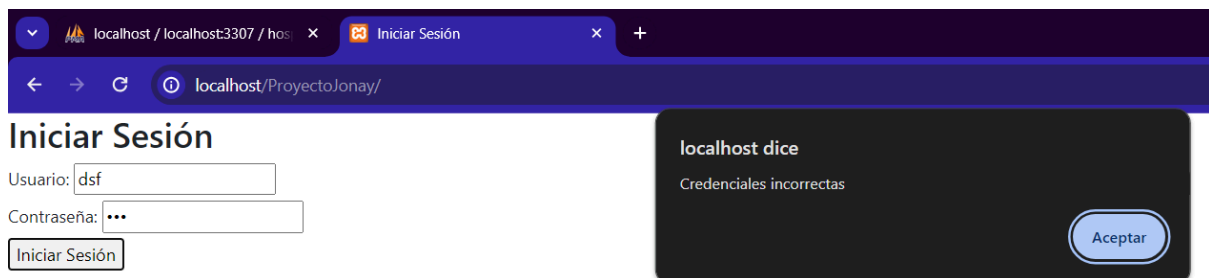
Usuario:

Contraseña:

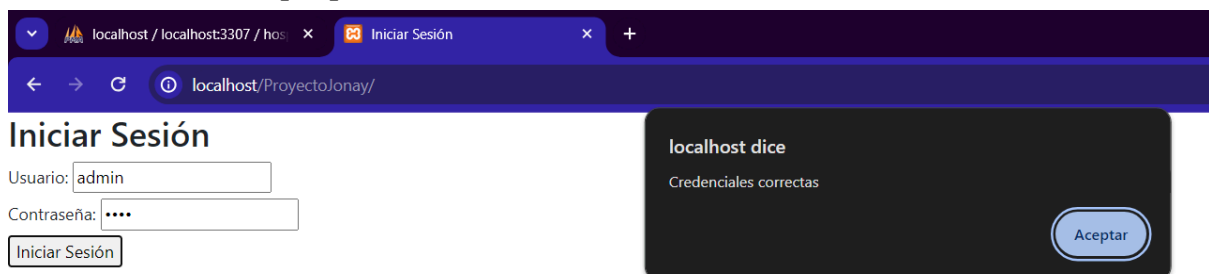
- Esto es lo que pasa si están los campos vacíos:



- Esto es lo que pasa si las credenciales son incorrectas:

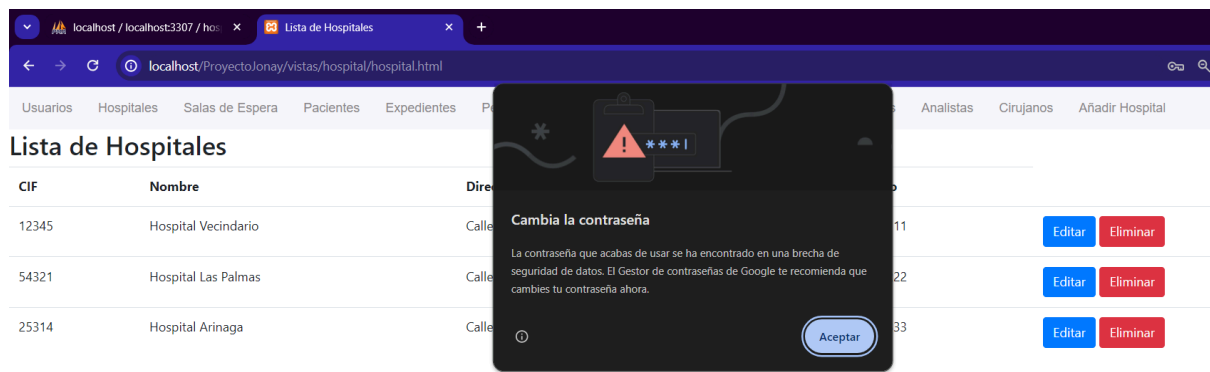


- Y esto es lo que pasa cuando metemos las credenciales correctas:



Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

- Tras iniciar sesión, nos redirige a la página principal del proyecto hospital.html:



- Ahora vamos a crear un nuevo hospital:

The screenshot shows the 'Añadir Hospital' form in the web application. The form has the following fields:

- CIF del hospital: 24234
- Nombre del hospital: Hospital CIPF Villa de Agüimes
- Dirección del hospital: Polígono Residencial de Arinaga, C. Alcorac, 50, 35118 Agüimes, Las Palmas
- Localidad del hospital: Agüimes
- Teléfono: 928599141

At the bottom of the form are two buttons: 'Crear Hospital' (blue) and 'Volver' (grey).

- Aquí lo tenemos ya creado:

The screenshot shows the 'Lista de Hospitales' page after adding a new hospital. The table now includes the new entry for 'Hospital CIPF Villa de Agüimes'.

CIF	Nombre	Dirección	Localidad	Teléfono	Acciones
12345	Hospital Vecindario	Calle 1	Vecindario	111111111	Editar Eliminar
54321	Hospital Las Palmas	Calle 2	Las Palmas	222222222	Editar Eliminar
25314	Hospital Arinaga	Calle 5	Arinaga	333333333	Editar Eliminar
24234	Hospital CIPF Villa de Agüimes	Polígono Residencial de Arinaga, C. Alcorac, 50, 3	Agüimes	928599141	Editar Eliminar

Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

- Vamos a intentar editarlo pulsando en el botón de editar, tendrían que salir los datos del hospital 4, pero no he encontrado la manera de solucionar el error:

Datos de Hospital

Código del hospital

CIF del hospital

Nombre del hospital

Dirección del hospital

Localidad del hospital

Teléfono

[Editar Hospital](#) [Volver](#)

Console Errors:

- editor_hospital.js:50: The specified value "undefined" cannot be parsed, or is out of range.
- editor_hospital.js:51: The specified value "undefined" cannot be parsed, or is out of range.
- editor_hospital.js:55: The specified value "undefined" cannot be parsed, or is out of range.

- Ahora vamos a borrar el hospital creado:

Lista de Hospitales

localhost dice
Hospital eliminado exitosamente

[Aceptar](#)

CIF	Nombre	Dirección	Localidad	Teléfono	
12345	Hospital Vecindario	Calle 1	Vecindario	111111111	Editar Eliminar
54321	Hospital Las Palmas	Calle 2	Las Palmas	222222222	Editar Eliminar
25314	Hospital Arinaga	Calle 5	Arinaga	333333333	Editar Eliminar
24234	Hospital CIF Villa de Aguimes	Poligono Residencial de Arinaga, C. Alcorac, 50, 3	Aguimes	928599141	Editar Eliminar

- Ya se ha borrado correctamente

Lista de Hospitales

CIF	Nombre	Dirección	Localidad	Teléfono	
12345	Hospital Vecindario	Calle 1	Vecindario	111111111	Editar Eliminar
54321	Hospital Las Palmas	Calle 2	Las Palmas	222222222	Editar Eliminar
25314	Hospital Arinaga	Calle 5	Arinaga	333333333	Editar Eliminar

Jonay Martel Martel 2ºDAW - Proyecto de desarrollo de aplicaciones web

- También voy a enseñar la tabla Pacientes ya que es una de las que contiene selectores

DNI	Nombre	Apellidos	Edad	Teléfono	Código de Hospital	Código de Sala de Espera	Acciones
54270592E	Jonay	Martel Martel	19	666376019	1	1	Editar Eliminar
54070070Z	Maite	Martel Ventura	47	658930822	2	2	Editar Eliminar
65432198C	Isabelino	Martel Martel	44	606930689	3	3	Editar Eliminar

- Esto sería al crear un paciente que saldría los hospitales que hay disponibles

Añadir Paciente

DNI

55555555W

Nombre

Roberto Carlos

Apellidos

Dos Santos Aveiro

Edad

Teléfono

666666666

Codigo de hospital

Selecciona un hospital

- 1 - Hospital Vecindario
- 2 - Hospital Las Palmas
- 3 - Hospital Arinaga

[Añadir Paciente](#) [Volver](#)

Y ya estaría no hay mucho más de lo que he enseñado, con el resto de las tablas es exactamente lo mismo.

4.2.- Anexo II. DB Coding Standards

Mi base de datos consiste en 13 tablas para formar un sistema de gestión hospitalaria. Esas 13 tablas son:

- Usuario, que cuenta con los campos
 - id_usuario de tipo INT auto increment
 - usuario de tipo VARCHAR(100)
 - contraseña de tipo VARCHAR(100)

```
CREATE TABLE IF NOT EXISTS usuario(  
    id_usuario INT NOT NULL AUTO_INCREMENT,  
    usuario VARCHAR(100) NOT NULL,  
    contrasena VARCHAR(100) NOT NULL,  
    CONSTRAINT PK_USUARIO PRIMARY KEY (id_usuario)  
);
```

- Token, que cuenta con los campos
 - id_token de tipo INT auto increment
 - token de tipo VARCHAR(100)
 - fechaCreacion de tipo DATE
 - fechaExpiracion de tipo DATE
 - id_usuario de tipo INT

```
CREATE TABLE IF NOT EXISTS token(  
    id_token INT NOT NULL AUTO_INCREMENT,  
    token VARCHAR(100) NOT NULL,  
    fechaCreacion DATE NOT NULL,  
    fechaExpiracion DATE NOT NULL,  
    id_usuario INT NOT NULL,  
    CONSTRAINT PK_TOKEN PRIMARY KEY (id_token)  
);
```

- Hospital, que cuenta con los campos
 - cod_hospital de tipo INT auto increment
 - cif_hospital de tipo INT
 - nombre de tipo VARCHAR(100)
 - direccion de tipo VARCHAR(100)
 - localidad de tipo VARCHAR(100)
 - telefono de tipo NUMERIC(9,0)


```
CREATE TABLE IF NOT EXISTS hospital(  
    cod_hospital INT NOT NULL AUTO_INCREMENT,  
    cif_hospital INT NOT NULL,  
    nombre VARCHAR(100) NOT NULL,  
    direccion VARCHAR(100) NOT NULL,  
    localidad VARCHAR(100) NOT NULL,  
    telefono NUMERIC(9,0) NOT NULL,  
    CONSTRAINT PK_HOSPITAL PRIMARY KEY (cod_hospital)  
);
```

- Salas_espera, que cuenta con los campos
 - cod_salas_espera de tipo INT auto increment
 - num_sillas de tipo INT
 - cod_hospital de tipo INT

```
CREATE TABLE IF NOT EXISTS salas_espera(  
    cod_salas_espera INT NOT NULL AUTO_INCREMENT,  
    num_sillas INT NOT NULL,  
    cod_hospital INT NOT NULL,  
    CONSTRAINT PK_SALAS_ESPERA PRIMARY KEY (cod_salas_espera)  
);
```

- Pacientes, que cuenta con los campos:
 - cod_pac de tipo INT auto increment
 - dni_pac de tipo VARCHAR(9)
 - nombre de tipo VARCHAR(100)
 - apellidos de tipo VARCHAR(100)
 - edad de tipo NUMERIC (3,0)
 - telefono de tipo NUMERIC (9,0)
 - cod_hospital de tipo INT
 - cod_salas_espera de tipo INT

```
CREATE TABLE IF NOT EXISTS pacientes(  
    cod_pac INT NOT NULL AUTO_INCREMENT,  
    dni_pac VARCHAR(9) NOT NULL,  
    nombre VARCHAR(100) NOT NULL,  
    apellidos VARCHAR(100) NOT NULL,  
    edad NUMERIC(3,0) NOT NULL,  
    telefono NUMERIC(9,0) NOT NULL,  
    cod_hospital INT NOT NULL,  
    cod_salas_espera INT NOT NULL,  
    CONSTRAINT PK_PACIENTES PRIMARY KEY (cod_pac)  
);
```

- Expediente, que cuenta con los campos:
 - cod_expediente de tipo INT auto increment
 - vacunas de tipo VARCHAR(100)
 - historial_medico de tipo VARCHAR(200)
 - enfermedades de tipo VARCHAR(100)
 - cod_pac de tipo INT

```
CREATE TABLE IF NOT EXISTS expediente(  
    cod_expediente INT NOT NULL AUTO_INCREMENT,  
    vacunas VARCHAR(100) NOT NULL,  
    historial_medico VARCHAR(200) NOT NULL,  
    enfermedades VARCHAR(100) NOT NULL,  
    cod_pac INT NOT NULL,  
    CONSTRAINT PK_EXPEDIENTE PRIMARY KEY (cod_expediente)  
);
```

- Personal, que cuenta con los campos:
 - cod_per de tipo INT auto increment
 - dni_per de tipo VARCHAR(9)
 - nombre de tipo VARCHAR(100)
 - apellidos de tipo VARCHAR(100)
 - salario de tipo NUMERIC (7,0)
 - contrato de tipo VARCHAR(100)
 - cod_hospital de tipo INT

```
CREATE TABLE IF NOT EXISTS personal(  
    cod_per INT NOT NULL AUTO_INCREMENT,  
    dni_per VARCHAR(9) NOT NULL,  
    nombre VARCHAR(100) NOT NULL,  
    apellidos VARCHAR(100) NOT NULL,  
    salario NUMERIC(7,0) NOT NULL,  
    contrato VARCHAR(100) NOT NULL,  
    cod_hospital INT NOT NULL,  
    CONSTRAINT PK_PERSONAL PRIMARY KEY (cod_per)  
);
```

- tipo_personal, que cuenta con los campos
 - id_tipo_personal de tipo INT auto increment
 - tipo_personal de tipo VARCHAR(100)
 - cod_per de tipo INT

```
CREATE TABLE IF NOT EXISTS tipo_personal(  
    id_tipo_personal INT NOT NULL AUTO_INCREMENT,  
    tipo_personal VARCHAR(100) NOT NULL,  
    cod_per INT NOT NULL,  
    CONSTRAINT PK_TIPO_PERSONAL PRIMARY KEY (id_tipo_personal)  
);
```

- medicos, que cuenta con los campos
 - cod_medicos de tipo INT auto increment
 - nombre de tipo VARCHAR(100)
 - especialidad de tipo VARCHAR(100)

- id_tipo_personal de tipo INT

```
CREATE TABLE IF NOT EXISTS medicos(  
    cod_medicos INT NOT NULL AUTO_INCREMENT,  
    nombre VARCHAR(100) NOT NULL,  
    especialidad VARCHAR(100) NOT NULL,  
    id_tipo_personal INT NOT NULL,  
    CONSTRAINT PK_MEDICOS PRIMARY KEY (cod_medicos)  
);
```

- Director, que cuenta con los campos
 - cod_director de tipo INT auto increment
 - id_tipo_personal de tipo INT

```
CREATE TABLE IF NOT EXISTS director(  
    cod_director INT NOT NULL AUTO_INCREMENT,  
    id_tipo_personal INT NOT NULL,  
    CONSTRAINT PK_DIRECTOR PRIMARY KEY (cod_director)  
);
```

- Tipo_medicos, que cuenta con los campos
 - id_tipo_medicos de tipo INT auto increment
 - tipo_medicos de tipo VARCHAR(100)
 - cod_medicos de tipo INT

```
CREATE TABLE IF NOT EXISTS tipo_medicos(  
    id_tipo_medicos INT NOT NULL AUTO_INCREMENT,  
    tipo_medicos VARCHAR(100) NOT NULL,  
    cod_medicos INT NOT NULL,  
    CONSTRAINT PK_TIPO_MEDICOS PRIMARY KEY (id_tipo_medicos)  
);
```

- Analista, que cuenta con los campos
 - cod_analista de tipo INT auto increment
 - num_colegiado de tipo INT
 - id_tipo_medicos de tipo INT

```
CREATE TABLE IF NOT EXISTS analista(  
    cod_analista INT NOT NULL AUTO_INCREMENT,  
    num_colegiado INT NOT NULL,  
    id_tipo_medicos INT NOT NULL,  
    CONSTRAINT PK_ANALISTA PRIMARY KEY (cod_analista)  
);
```

- Cirujano, que cuenta con los campos
 - cod_cirujano de tipo INT auto increment
 - num_colegiado de tipo INT
 - id_tipo_medicos de tipo INT

```
CREATE TABLE IF NOT EXISTS cirujano(  
    cod_cirujano INT NOT NULL AUTO_INCREMENT,  
    num_colegiado INT NOT NULL,  
    id_tipo_medicos INT NOT NULL,  
    CONSTRAINT PK_CIRUJANO PRIMARY KEY (cod_cirujano)  
);
```

Y estos son las foreign key que tengo:

```
ALTER TABLE token ADD CONSTRAINT FK_TOKEN FOREIGN KEY (id_usuario) REFERENCES usuario(id_usuario) ON DELETE CASCADE;  
ALTER TABLE salas_espera ADD CONSTRAINT FK_SALAS_ESPERA FOREIGN KEY (cod_hospital) REFERENCES hospital(cod_hospital) ON DELETE CASCADE;  
ALTER TABLE pacientes ADD CONSTRAINT FK_PACIENTES FOREIGN KEY (cod_hospital) REFERENCES hospital(cod_hospital) ON DELETE CASCADE;  
ALTER TABLE pacientes ADD CONSTRAINT FK_PACIENTES1 FOREIGN KEY (cod_salas_espera) REFERENCES salas_espera(cod_salas_espera) ON DELETE CASCADE;  
ALTER TABLE expediente ADD CONSTRAINT FK_EXPEDIENTE FOREIGN KEY (cod_pac) REFERENCES pacientes(cod_pac) ON DELETE CASCADE;  
ALTER TABLE personal ADD CONSTRAINT FK_PERSONAL FOREIGN KEY (cod_hospital) REFERENCES hospital(cod_hospital) ON DELETE CASCADE;  
ALTER TABLE tipo_personal ADD CONSTRAINT FK_TIPO_PERSONAL FOREIGN KEY (cod_per) REFERENCES personal(cod_per) ON DELETE CASCADE;  
ALTER TABLE medicos ADD CONSTRAINT FK_MEDICOS FOREIGN KEY (id_tipo_personal) REFERENCES tipo_personal(id_tipo_personal) ON DELETE CASCADE;  
ALTER TABLE director ADD CONSTRAINT FK_DIRECTOR FOREIGN KEY (id_tipo_personal) REFERENCES tipo_personal(id_tipo_personal) ON DELETE CASCADE;  
ALTER TABLE tipo_medicos ADD CONSTRAINT FK_TIPO_MEDICOS FOREIGN KEY (cod_medicos) REFERENCES medicos(cod_medicos) ON DELETE CASCADE;  
ALTER TABLE analista ADD CONSTRAINT FK_ANALISTA FOREIGN KEY (id_tipo_medicos) REFERENCES tipo_medicos(id_tipo_medicos) ON DELETE CASCADE;  
ALTER TABLE cirujano ADD CONSTRAINT FK_CIRUJANO FOREIGN KEY (id_tipo_medicos) REFERENCES tipo_medicos(id_tipo_medicos) ON DELETE CASCADE;
```

4.3.- Anexo III. DOC Guía Desarrollador

Manual para el desarrollador Clase Hospital

1 Objetivo

El objetivo de este documento es explicar el funcionamiento de la clase “Hospital”, a nivel de desarrollador y comentar un poco sobre ella.

2 Alcance

Esta guía va dirigida a todo el mundo que este viendo este proyecto y quiero entender en profundidad la clase Hospital

3 Dependencias

No se han detectado dependencias de bibliotecas para la utilización de esta clase.

4 Integración en la aplicación

En este documento explicaremos este apartado con la clase “Hospital”. Para conocer todas las clases y métodos disponibles, ver documentación del DTS

4.1 Añadir referencias al proyecto

En el apartado 3 no se han encontrado dependencias ninguna

4.2 Uso de instancia de clases

```
CLASS HOSPITAL {  
  mostrar_hospital(){  
    - Con este método obtenemos y mostramos la información de todos los  
      hospitales en la base de datos. Realiza una consulta SQL para seleccionar todos  
      los registros de la tabla hospital y luego muestra los resultados en formato  
      JSON.  
  }  
  crear_hospital($cif_hospital, $nombre, $direccion, $localidad, $telefono){  
    - Este método crea un nuevo registro de hospital en la base de datos.  
    Prepara y ejecuta una sentencia SQL de inserción en la tabla hospital con los datos  
    proporcionados  
  }  
  borrar_hospital($cod_hospital){  
    - Este método elimina un hospital específico de la base de datos. Prepara y  
    ejecuta una sentencia SQL de eliminación en la tabla hospital utilizando el  
    código de hospital seleccionado  
  }  
  
  actualizar_hospital($cod_hospital, $cif_hospital, $nombre, $direccion, $localidad,  
  $telefono){  
    - Este método actualiza la información de un hospital existente en la base de  
    datos.  
    Prepara y ejecuta una sentencia SQL de actualización en la tabla hospital con los datos  
    proporcionados y el código de hospital seleccionado.  
  }  
}
```

Atributos de la clase: cod_hospital, cif_hospital, nombre, direccion, localidad y telefono

4.3 Ejemplos de USO

4.4.- Anexo IV. DTS Guía Técnica

Diseño técnico del sistema Clase Hospital

1. ARQUITECTURA DE LA SOLUCIÓN

Clase Hospital se trata de una clase desarrollada en PHP.

Se ha decidido estructurar la aplicación separando en clases cada uno de los tipos de validación y valores por defecto .

1.1 Dependencias

No existen

1.1.1 Librerías

No existen

1.2 Entornos

El alojamiento en subversion/GIT es el siguiente:

2 INFRAESTRUCTURA UTILIZADA

2.1 Plataforma Software

Visual Studio Code

3 NameSpace XXXXXXXXXXXX

3.1 Class Hospital

A través de la clase estática Hospital

3.1.1 Constructores

No dispongo de constructores en esta clase

3.1.2 Métodos

public static function mostrar_hospital()

- Descripción: Con este método obtenemos y mostramos la información de todos los hospitales en la base de datos. Realiza una consulta SQL para

seleccionar todos los registros de la tabla hospital y luego muestra los resultados en formato JSON.

- Parámetros: Ninguno
- Retorno: Retorna un código de estado HTTP en función del resultado
- Excepciones: No hay

public static function crear_hospital(\$cif_hospital, \$nombre, \$direccion, \$localidad, \$telefono)

- Descripción: Este método crea un nuevo registro de hospital en la base de datos.

Prepara y ejecuta una sentencia SQL de inserción en la tabla hospital con los datos proporcionados

- Parámetros: cif_hospital, nombre, direccion, localidad, telefono.
- Retorno: Retorna un código de estado HTTP en función del resultado
- Excepciones: No hay

public static function borrar_hospital(\$cod_hospital)

- Descripción: Este método elimina un hospital específico de la base de datos.

Prepara y ejecuta una sentencia SQL de eliminación en la tabla hospital utilizando el código de hospital seleccionado

- Parámetros: cod_hospital
- Retorno: Retorna un código de estado HTTP en función del resultado
- Excepciones: No hay

public static function actualizar_hospital(\$cod_hospital, \$cif_hospital, \$nombre, \$direccion, \$localidad, \$telefono)

- Descripción: Este método elimina un hospital específico de la base de datos.

Prepara y ejecuta una sentencia SQL de eliminación en la tabla hospital utilizando el código de hospital seleccionado

- Parámetros: cod_hospital, cif_hospital, nombre, direccion, localidad, telefono.
- Retorno: Retorna un código de estado HTTP en función del resultado
- Excepciones: No hay