# UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

## UNIDAD DE EDUCACIÓN A DISTANCIA

**Asignatura:**

**Aplicación de Tecnologias Web**

**Ing.**

**Angel Geovanny Cudco Pomagualli**

**TEMA:**

**Modelo MVC**

**NOMBRES:**

**Andres Alejandro Toledo Rojas**

**Carlos Ramiro Yánez Yazán**

**Luis David Flores Pillajo**

**NRC:**

**1385**

**Quito…20…febrero de 2025**

## *Modelo MVC*

### Objetivo General

Desarrollar una aplicación web funcional en PHP que haga uso de un modelo MVC con el fin de gestionar libros y autores de manera eficiente, para ello se va a integrar tecnología como Axios para las peticiones AJAX, con un diseño de interfaz centrado en Bootstrap, y configurando rutas mediante **.htaccess.**

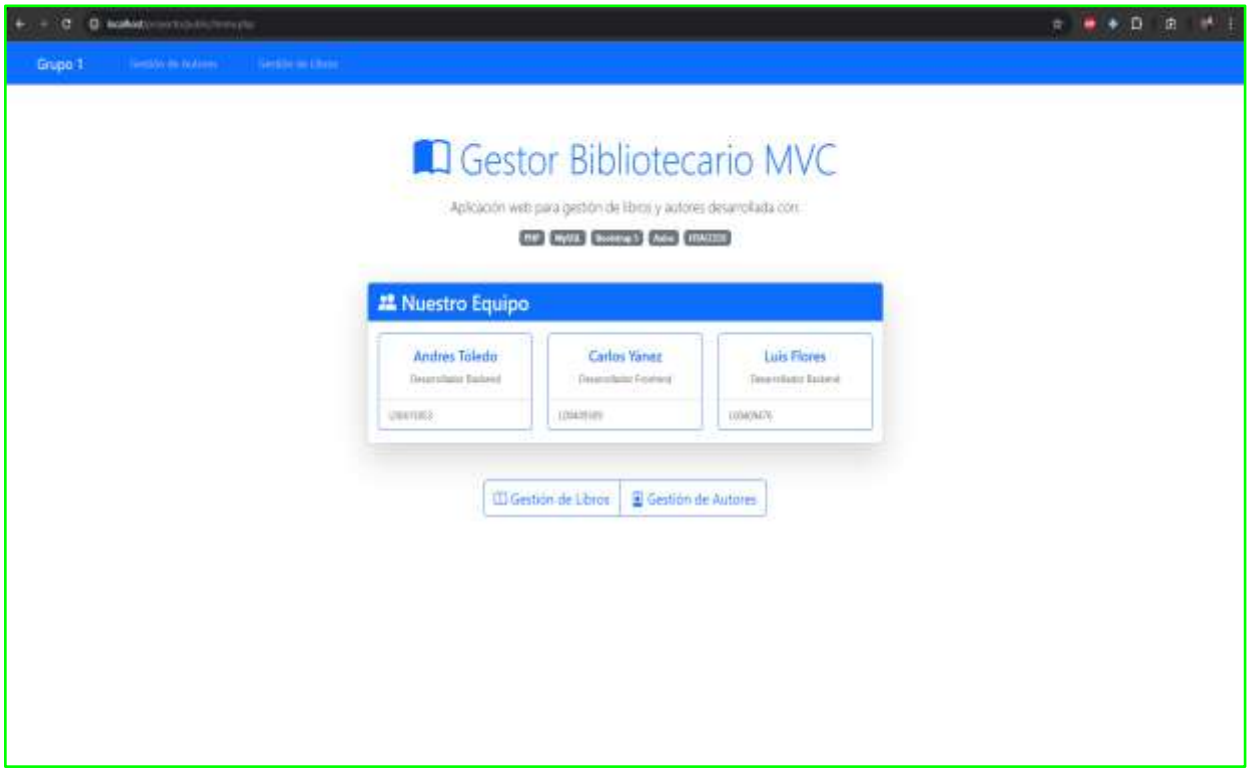
### Objetivos Específicos

- Diseñar la estructura de la aplicación con la implementación de un modelo MVC, que asegure la separación adecuada de responsabilidades entre los modelos, vistas y controladores.
- Implementar una gestión de rutas y URLs eficientes con la utilización de .htaccess y un router sencillo en PHP para dirigir las peticiones a los controladores que correspondan.
- Desarrollar una interfaz interactiva mediante Bootstrap y modales para la creación y edición de libros y autores.

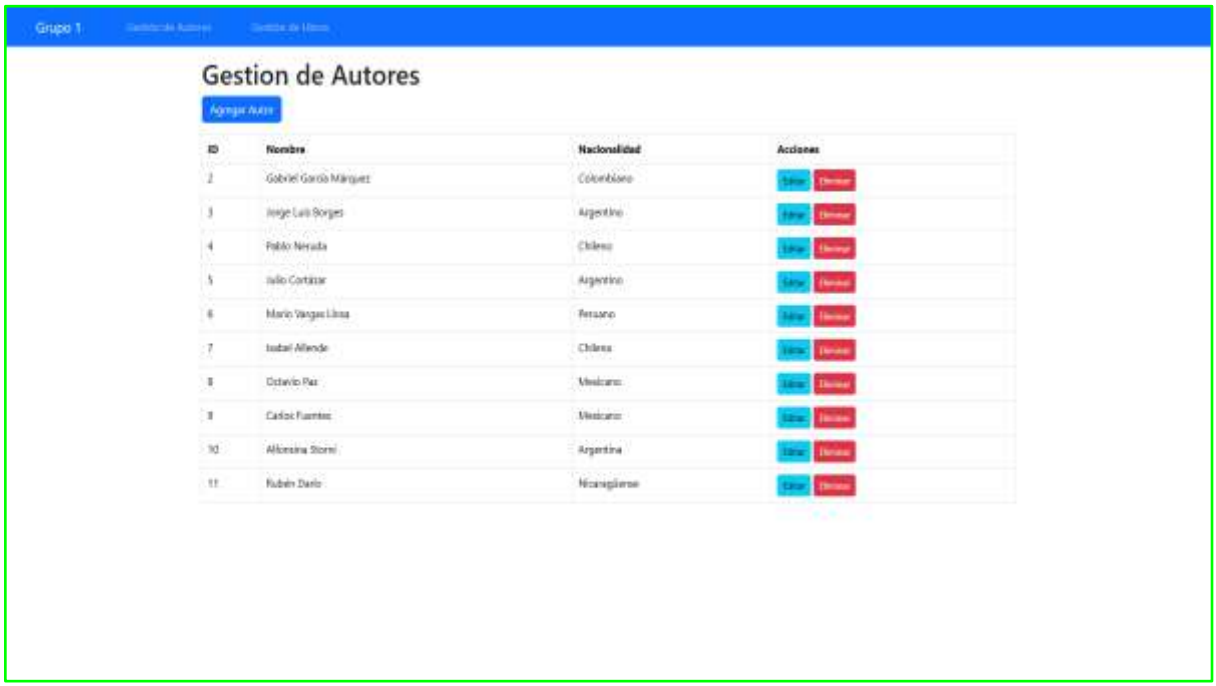
### Desarrollo

## *Maquetado general del aplicativo web*

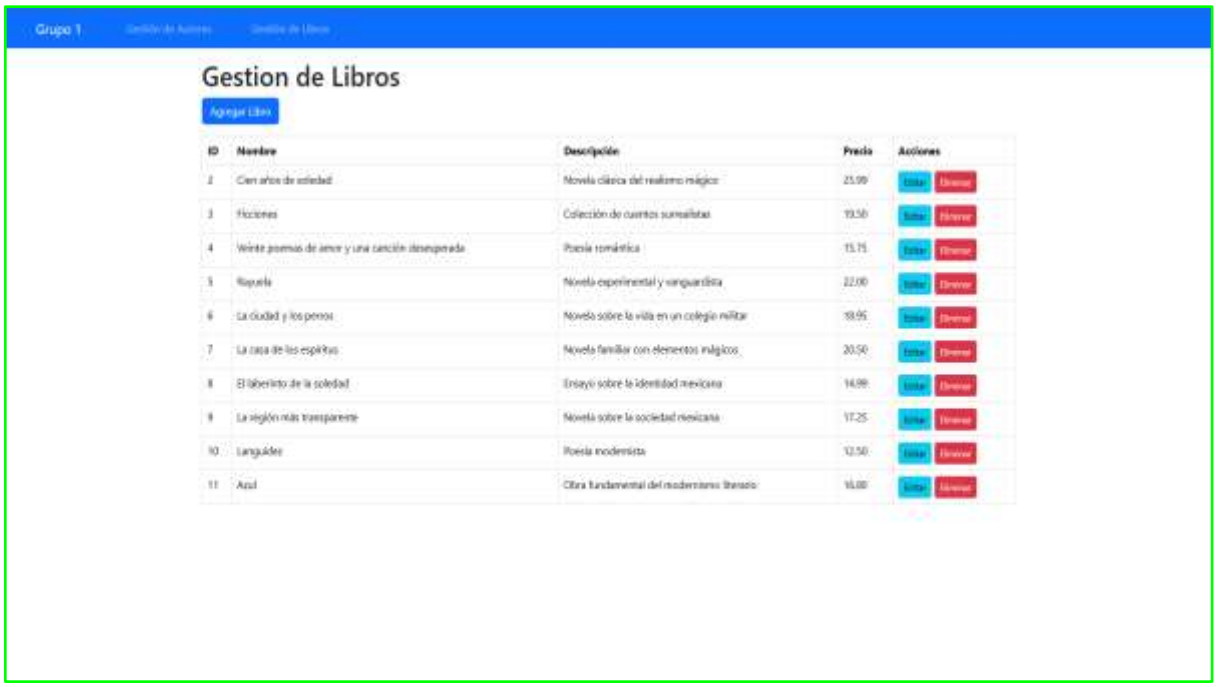
### *Vista home.php (Página de Inicio)*



***Vista gestion.php (Gestión de Autores)***

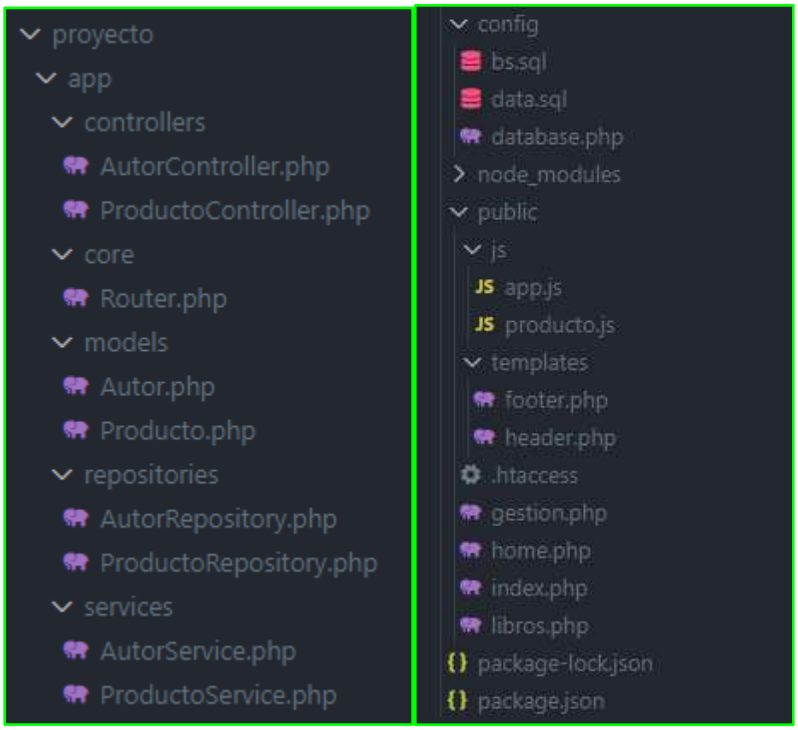



***Vista libros.php(Gestión de Libros)***




Como primer punto se busca explicar de forma general la estructura por la que se desarrolló el proyecto para tener una idea concisa sobre su funcionamiento de la siguiente manera:

***ESTRUCTURA APLICADA PARA EL APLICATIVO WEB***




Se propone el diseño del mismo de la siguiente forma:

**Estructura General:**

**proyecto/:** Esta es la carpeta raíz del proyecto. Contiene todos los archivos y subcarpetas necesarios para que la aplicación funcione.

**Carpetas Principales y su Función:**

**app/:**Esta carpeta alberga el código principal de la aplicación.

**controllers/:** Aquí se encuentran los controladores (Controllers). Los controladores son responsables de recibir las solicitudes del usuario, interactuar con los modelos y servicios para obtener los datos necesarios, y luego enviar esos datos a las vistas (templates) para generar la respuesta final, en este caso interactúa con gestion.php y libros.php ubicados en app/.

**AutorController.php:** Maneja las solicitudes relacionadas con los autores.

**ProductoController.php:** Maneja las solicitudes relacionadas con los libros.

**core/:** Esta carpeta contiene el enrutador (Router).

**Router.php:**Se encarga de dirigir las solicitudes entrantes al controlador y método adecuados.

**models/:** Los modelos (Models) representan las entidades de la aplicación y contienen la lógica para interactuar con la base de datos.

**Autor.php:**Representa la entidad "Autor" y sus propiedades.

**Producto.php:** Representa la entidad "Producto" y sus propiedades.

**repositories/:** Los repositorios (Repositories) son un patrón de diseño muy usado en Laravel, se utiliza para abstraer la lógica de acceso a datos y proporcionar una interfaz coherente para interactuar con la base de datos.

**AutorRepository.php:** Contiene la lógica para acceder y manipular los datos de los autores en la base de datos.

**ProductoRepository.php:** Contiene la lógica para acceder y manipular los datos de los libros en la base de datos.

**services/:** Los servicios (Services) contienen la lógica de negocio de la aplicación.

**AutorService.php:** Contiene la lógica de negocio relacionada con los autores.

**ProductoService.php:** Contiene la lógica de negocio relacionada con los libros.

**config/:** Aquí se almacenan los archivos de configuración de la aplicación.

**bs.sql:** Un script SQL para crear la estructura de la base de datos.

**data.sql:** Un script SQL para crear las tablas en caso de que no existan e insertar datos de ejemplo en la base de datos al mismo tiempo.

**database.php:** Contiene la configuración de la conexión a la base de datos.

**node_modules/:** Esta carpeta contiene los paquetes de Node.js instalados para el proyecto. Creada al instalar AXIOS.

**public/:** Esta carpeta contiene los archivos que son accesibles públicamente desde el navegador.

**js/:** Aquí se almacenan los archivos JavaScript.

**app.js:** El archivo JavaScript encargado de las funcionalidades para la Gestión de Libros

**producto.js:** JavaScript encargado de las funcionalidades para la Gestión de Libros.

**templates/:** Las plantillas (Templates) son archivos que contienen la estructura HTML de las páginas web**.**

**footer.php:** El pie de página común para todas las páginas.

**header.php:** La cabecera común para todas las páginas.

**.htaccess:** Archivo de configuración para el servidor web Apache.

**gestion.php:** Es la vista o interfaz que se mostrará para la Gestión de Autores

**home.php:** Es la vista o interfaz que se mostrará como Página de Inicio

**index.php:** El punto de entrada principal de la aplicación(sirve para conectar el enrutador y los controladores).

**libros.php:** Es la vista o interfaz que se mostrará para la Gestión de Libros.

**package-lock.json:** Un archivo generado por npm que contiene información detallada sobre las dependencias instaladas.

**package.json:** Un archivo que contiene metadatos sobre el proyecto y sus dependencias de Node.js.

Una vez unificada la estructura de cómo está compuesta podemos definir un resumen a través de aspectos claros sobre su patrón de diseño.

**Resumen:**

Esta estructura sigue un patrón de diseño Modelo-Vista-Controlador (MVC) con una separación clara de responsabilidades:

- Los modelos interactúan con la base de datos.
- Los controladores manejan las solicitudes del usuario.
- Las vistas, definidas en public, generan la interfaz de usuario.
- Los servicios contienen la lógica de negocio.
- Los repositorios manejan la interacción con la base de datos.

De esta manera buscamos definir cómo se compone la estructura de nuestro proyecto por lo cual vamos a partir a identificar puntos importantes como la configuración del Router y el uso de axios dentro del mismo donde se determine como es que se da su funcionamiento frente a la app web que buscamos desarrollar.

## FUNCIONAMIENTO DEL ROUTER.PHP.

Para el Router.php, tiene como objetivo manejar las rutas de la aplicación web siguiendo el modelo que se plantea en la actividad, que es el modelo **MVC**. Con esta clase podemos mapear las rutas a funciones específicas, lo que facilita la gestión de las peticiones HTTP entrantes.

### ***Funcionalidad***

Mediante el método **add()**, podemos añadir rutas específicas junto con las funciones que se deben ejecutar cuando se acceda a esa ruta. A través del método **dispatch()**, esta clase toma el método HTTP y la URI de la petición de busca la registrada que coincida.
Esto nos sirve para la gestión de rutas, enrutamiento de las solicitudes de la aplicación web, permitiendo asociar diferentes URIs con funciones específicas. Las rutas deben contener parámetros dinámicos, como **:id**, que se extraen y pasan a la función correspondiente.

## FUNCIONAMIENTO GENERAL DEL APP.JS

```
// Ruta base de la API (ajústala según tu entorno).
const apiUrl = 'http://localhost/proyecto/public';

// Al cargar la página, se ejecuta la función para obtener todos los autores.
document.addEventListener('DOMContentLoaded', () => getAutores());

/**
 * Obtiene todas los autores desde la API y actualiza la tabla.
 */
const getAutores = () => {
  axios.get(`${apiUrl}/autores`)
    .then(response => {
      const autores = response.data;
      const tbody = document.querySelector('#autoresTable tbody');
      tbody.innerHTML = '';
      autores.forEach(autor => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
          <td>${autor.id}</td>
          <td>${autor.nombre}</td>
          <td>${autor.nacionalidad}</td>
          <td>
            <button class="btn btn-sm btn-info"
onclick="editAutor(${autor.id})">Editar</button>
            <button class="btn btn-sm btn-danger"
onclick="deleteAutor(${autor.id})">Eliminar</button>
          </td>
        `;
        tbody.appendChild(tr);
      });
    })
    .catch(error => console.error(error));
};

/**
 * Abre el modal para agregar un nuevo autor.
```

```
 */
const openModal = () => {
  document.getElementById('autorForm').reset();
  document.getElementById('autorId').value = '';
  document.getElementById('autorModalLabel').innerText = 'Agregar Autor';
};



/**
 * Envía el formulario para crear o actualizar un autor.
 */
document.getElementById('autorForm').addEventListener('submit', e => {
  e.preventDefault();
  const id = document.getElementById('autorId').value;
  const nombre = document.getElementById('autorNombre').value;
  const nacionalidad = document.getElementById('autorNacionalidad').value;

  console.log('Datos del formulario:', { id, nombre, nacionalidad });

  if (id) {
    axios.put(`${apiUrl}/autores`, { id, nombre, nacionalidad })
      .then(response => {
        console.log('Respuesta del servidor (PUT):', response);
        $('#autorModal').modal('hide');
        getAutores();
      })
      .catch(error => console.error('Error en la solicitud PUT:', error));
  } else {
    axios.post(`${apiUrl}/autores`, { nombre, nacionalidad })
      .then(response => {
        console.log('Respuesta del servidor (POST):', response);
        $('#autorModal').modal('hide');
        getAutores();
      })
      .catch(error => console.error('Error en la solicitud POST:', error));
  }
});

/**
 * Carga los datos de un autor en el formulario para editar.
 */
const editAutor = id => {
  axios.get(`${apiUrl}/autores/${id}`)
    .then(response => {
      console.log(response);
      const autor = response.data;
      document.getElementById('autorId').value = autor.id;
      document.getElementById('autorNombre').value = autor.nombre;
      document.getElementById('autorNacionalidad').value = autor.nacionalidad;
      document.getElementById('autorModalLabel').innerText = 'Editar Autor';
      $('#autorModal').modal('show');
    })
    .catch(error => console.error(error));
};

const deleteAutor = id => {
```

```
  if (confirm('¿Estás seguro de eliminar este autor?')) {
    axios.delete(`${apiUrl}/autores`, { data: { id } })
      .then(response => getAutores())
      .catch(error => console.error(error));
  }
};
```

Este código está diseñado para que utilice **Axios** para realizar peticiones GET o PUT, según corresponda la clase, para efectuar cambios en el CRUD. Esto nos permite gestionar autores en una base de datos: obtener todos los autores, agregar uno nuevo, editar uno existente y eliminar un autor.

## ***Funcionalidad***

**Al cargarse la página, la función getAutores()**es ejecutable. Esta función obtiene todos los autores desde la API y actualiza la tabla en elHTML.

**La función getAutores():**
Realiza una solicitud GET para obtener los datos de todos los autores desde la API. Luego, recorre la lista de autores y actualiza el contenido de una tabla HTML. Cada autor tiene botones para editar o eliminar.

Se va abrir el modal para agregar un nuevo autor, esta función se llama cuando se desea agregar un nuevo autor. Resetea el formulario, establece el título del modal como "Agregar Autor" y limpia el campo de ID.

En este apartado también se va a enviar el formulario para crear o actualizar un autor para ello captura el evento de envío del formulario. Si el campo de ID tiene valor, se realiza una solicitud **PUT** para actualizar el autor existente. Si no tiene valor (es un nuevo autor), se hace una solicitud **POST** para agregarlo. En ambos casos, tras completar la operación, se cierra el modal y se actualiza la lista de autores.

**La función editAutor():**
Obtiene los datos del autor a través de una solicitud **GET** y los carga en el formulario para su edición. Luego, abre el modal para que el usuario pueda modificar los datos.

**La función deleteAutor():**
Confirma si el usuario desea eliminar el autor. Si se confirma, realiza una solicitud **DELETE** a la API para eliminar al autor. Luego, actualiza la lista de autores.

## FUNCIONAMIENTO GENERAL DEL PRODUCTO.JS

```
// Ruta base de la API (ajústala según tu entorno).
const apiUrl = 'http://localhost/proyecto/public';

// Al cargar la página, se ejecuta la función para obtener todos los productos.
document.addEventListener('DOMContentLoaded', () => getProductos());

/**
 * Obtiene todos los productos desde la API y actualiza la tabla.
```

```
 */
const getProductos = () => {
  axios.get(`${apiUrl}/productos`)
    .then(response => {
      const productos = response.data;
      const tbody = document.querySelector('#productosTable tbody');
      tbody.innerHTML = '';
      productos.forEach(producto => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
          <td>${producto.id}</td>
          <td>${producto.nombre}</td>
          <td>${producto.descripcion}</td>
          <td>${producto.precio}</td>
          <td>
            <button class="btn btn-sm btn-info"
onclick="editProducto(${producto.id})">Editar</button>
            <button class="btn btn-sm btn-danger"
onclick="deleteProducto(${producto.id})">Eliminar</button>
          </td>
        `;
        tbody.appendChild(tr);
      });
    })
    .catch(error => console.error("Error en la petición:", error));
};

/**
 * Abre el modal para agregar un nuevo producto.
 */
const openModalProductos = () => {
  document.getElementById('productoForm').reset();
  document.getElementById('productoId').value = '';
  document.getElementById('productoModalLabel').innerText = 'Agregar Producto';
};


/**
 * Envía el formulario para crear o actualizar un producto.
 */


document.getElementById('productoForm').addEventListener('submit', e => {
  e.preventDefault();
  const id = document.getElementById('productoId').value;
  const nombre = document.getElementById('productoNombre').value;
  const descripcion = document.getElementById('productoDescripcion').value;
  const precio = document.getElementById('productoPrecio').value;

  if (id) {
    axios.put(`${apiUrl}/productos`, { id, nombre, descripcion, precio})

    .then(response => {
        console.log('Respuesta del servidor (PUT):', response);
        $('#productoModal').modal('hide');
        getProductos();
      })
```

```
      .catch(error => console.error('Error en la solicitud PUT:', error));
  } else {
    axios.post(`${apiUrl}/productos`, { nombre, descripcion, precio})
      .then(response => {
        console.log('Respuesta del servidor (POST):', response);
        $('#productoModal').modal('hide');
        getProductos();
      })
      .catch(error => console.error('Error en la solicitud POST:', error));
  }
});

/**
 * Carga los datos de un producto en el formulario para editar.
 */
const editProducto = id => {
  axios.get(`${apiUrl}/productos/${id}`)
    .then(response => {
      const producto = response.data;
      document.getElementById('productoId').value = producto.id;
      document.getElementById('productoNombre').value = producto.nombre;
      document.getElementById('productoDescripcion').value = producto.descripcion;
      document.getElementById('productoPrecio').value = producto.precio;
      document.getElementById('productoModalLabel').innerText = 'Editar Producto';
      $('#productoModal').modal('show');
    })
    .catch(error => console.error(error));
};

/**
 * Elimina un producto.
 */
const deleteProducto = id => {
  if (confirm('¿Estás seguro de eliminar este producto?')) {
    axios.delete(`${apiUrl}/productos`, { data: { id } })
      .then(response => getProductos())
      .catch(error => console.error(error));
  }
};
```

***(Funcionamiento aplicado con Axios)***

El código interactúa con la API a través de **Axios**, generando peticiones GET o PUT según corresponda la clase, permitiendo realizar operaciones CRUD sobre los libros. Muestra los libros en una tabla, permite agregar nuevos productos, editar productos existentes y eliminarlos.

***Funcionalidad***

Cuando la página se carga, la función **getProductos()** se ejecuta automáticamente para obtener la lista de libros desde la API y actualizar la tabla en el HTML.

La función **getProductos()**

Realiza una solicitud **GET** a la API para obtener todos los productos. Luego, actualiza la tabla de la página web con los datos de los productos. Cada producto tiene botones para editar o eliminar.

En este mismo apartado se abre el modal para agrupar un nuevo producto, esta función se utiliza para abrir un formulario vacío para agregar un nuevo producto. Resetea los campos del formulario y establece el título del modal como "Agregar Producto".
De igual forma se envía el formulario para crear o actualizar un producto, en esta parte escucha el envío del formulario. Si el campo de ID tiene valor, se realiza una solicitud **PUT** para actualizar un producto existente. Si el campo de ID está vacío, se realiza una solicitud **POST** para crear un nuevo producto. Después de la operación, el modal se cierra y la lista de productos se actualiza.

**La función editProducto()**
Carga los datos de un producto seleccionado mediante una solicitud **GET** y los coloca en los campos del formulario para su edición. Luego, abre el modal de edición.

**La función deleteProducto()**
Confirma si el usuario desea eliminar el producto. Si confirma, realiza una solicitud **DELETE** a la API para eliminar el producto. Después de la eliminación, se actualiza la lista de productos.

## Conclusiones

- La implementación del modelo MVC en la aplicación web nos permite una separación de las responsabilidades dentro del código, la separación del modelo, la vista y el controlador mejora la organización del código, lo que nos proporciona una facilidad si se requiere modificar o ampliar el sistema.
- Teniendo una configuración adecuada de las rutas y la implementación de un archivo .htaccess para la gestión de las URLs nos proporcionan una estructura más clara y accesible a la aplicación, con esto garantizamos la navegación a través de rutas intuitivas facilitando la comprensión y el uso del sistema por parte de los usuarios.
- Con la utilización de Bootstrap en el diseño de la interfaz del usuario garantizamos que sea un aplicativo web responsiva, que se puede adaptar a los diferentes tamaños de pantallas.

## Recomendaciones

- Se recomienda ampliar las diversas funcionalidades de la aplicación, como implementar autenticación de usuario para la gestión de permisos de acceso a ciertas secciones de libros y autores. Con esto garantizamos un nivel de seguridad adicional y personalización en la gestión de los datos.
- Se recomienda revisar las consultas a la base de datos y la carga de los mismos a través de Axios. El uso de técnicas como la paginación o mediante el lazy loading se podría optimizar la carga de la lista de libros y los autores.

- Sería óptimo seguir mejorando la experiencia del usuario mediante la integración de las mejores prácticas de diseño de las interfaces, se puede mejorar en la inclusión de feedback visual como lo pueden ser las animaciones y transiciones, en la interacción con los formularios y botones.