Request Post

En el contexto de una API REST, una solicitud POST (POST request) es uno de los métodos de solicitud HTTP utilizados para enviar datos al servidor.

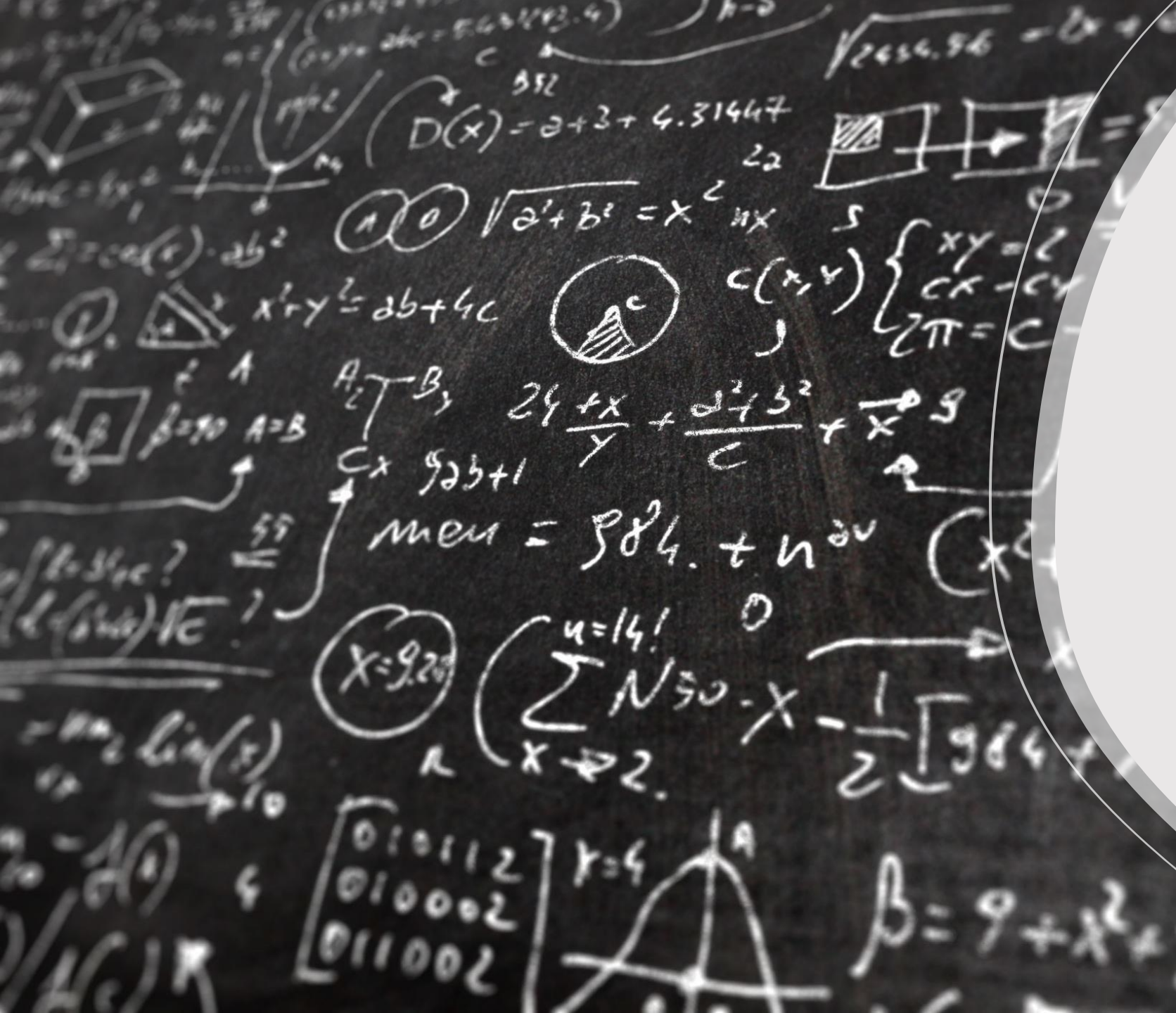
Se utiliza para enviar información del cliente al servidor y solicitar que se realice alguna acción en base a esos datos.

La información enviada en una solicitud POST se incluye en el cuerpo (body) de la solicitud HTTP en lugar de en la URL, como ocurre con las solicitudes GET.

Suele usarse para enviar datos como formularios, carga de archivos, envío de mensajes o cualquier otro tipo de información que deba ser procesada por el servidor.

Cuando se realiza una solicitud POST, los datos se envían en el cuerpo de la solicitud en un formato específico, como JSON o XML.

El servidor procesa los datos enviados y realiza las acciones correspondientes, como almacenar la información en una base de datos, actualizar registros existentes o generar una respuesta que indique el resultado de la operación.



Respuesta por parte del Servidor

- ResponseEntity es una clase que representa una respuesta HTTP, incluyendo los encabezados, el cuerpo y el estado; util cuando se necesita manipular el código de estado o agregar encabezados personalizados a la respuesta.
- Al utilizarlo, Se puede especificar el código de estado deseado, como 200 OK, 201 Created, 400 Bad Request, etc.

```java
@GetMapping("/example")
public ResponseEntity<String> getExample() {
    String responseBody = "Hello, World!";
    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "Sending information");

    return new ResponseEntity<>(responseBody, headers, HttpStatus.OK);
}
```

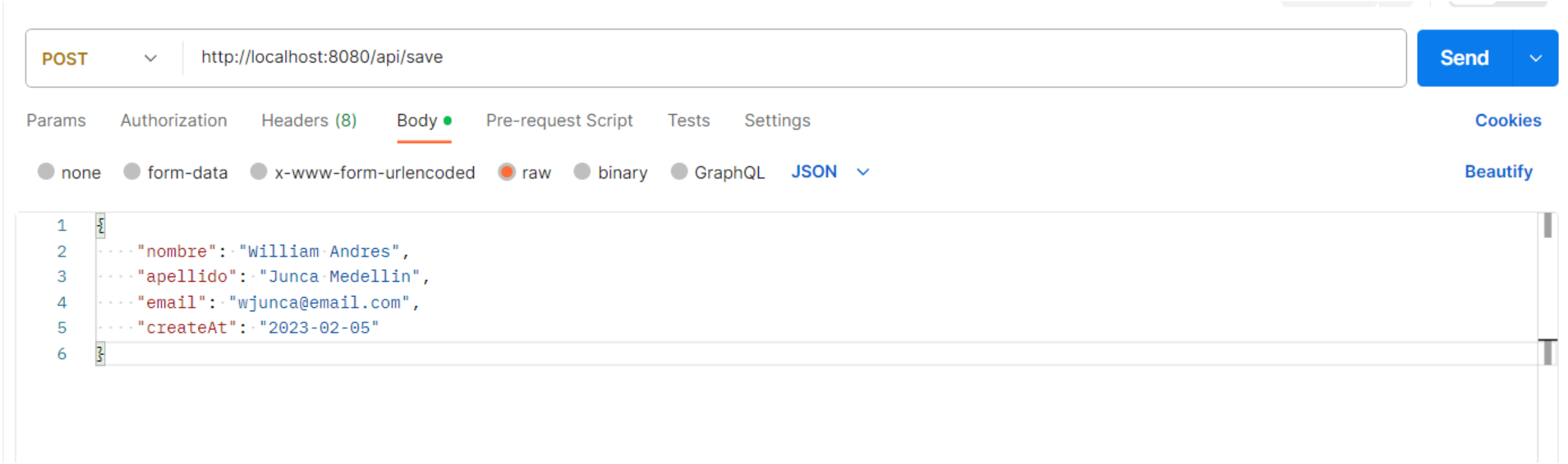
Notación @PostMapping

- Se utiliza para mapear una solicitud HTTP POST a un método específico en un controlador. Esta anotación se utiliza para manejar las solicitudes POST y definir la lógica de procesamiento correspondiente.

```java
@PostMapping("/save")
public ResponseEntity<String> crearEjemplo(@RequestBody Cliente cliente) {
    // Lógica para procesar el objeto ejemplo recibido en el cuerpo de la solicitud
    // Puedes guardar los datos en la base de datos, realizar alguna operación, etc.

    return ResponseEntity.ok("Ejemplo creado exitosamente");
}
```

Configuración en Postman



Request Body

(cuerpo de la solicitud) es la parte de una solicitud HTTP que contiene los datos enviados por el cliente al servidor. Se utiliza para enviar información adicional junto con la solicitud, como parámetros, datos estructurados o contenido de un formulario.

El cuerpo de la solicitud puede contener datos en diferentes formatos, como JSON, XML, formularios codificados o incluso archivos binarios. El tipo de contenido del cuerpo de la solicitud se especifica mediante el encabezado "Content-Type".

En el servicio

```java
@Service
public class ClienteServiceImpl implements ClienteService{

    @Autowired
    private ClienteDao clienteDao;


    @Override
    public List<Cliente> findAll() {
        return (List<Cliente>) clienteDao.findAll();
    }

    @Override
    public Cliente save(Cliente cliente) {

        return clienteDao.save(cliente);
    }

}
```

```java
public interface ClienteService {

    public List<Cliente> findAll();
    public Cliente save(Cliente cliente);

}
```

En el controlador

```java
@PostMapping("/save")
public ResponseEntity<Cliente> crearEjemplo(@RequestBody Cliente cliente) {
    // Lógica para procesar el objeto ejemplo recibido en el cuerpo de la solicitud
    // Puedes guardar los datos en la base de datos, realizar alguna operación, etc

    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "Sending Client information");
    Cliente responseBody = clienteServiceImpl.save(cliente);
    return new ResponseEntity<>(responseBody, headers, HttpStatus.OK);

}
```

```java
@Override
public void delete(Long id) {
    clienteDao.deleteById(id);
}
```

```java
public interface ClienteService {

    public List<Cliente> findAll();
    public Cliente save(Cliente cliente);
    public void delete(Long id);

}
```

En el servicio - Delete

@PathVariable

- @PathVariable en Spring Boot se utiliza para vincular y capturar partes variables de una URL en los parámetros de un método de controlador. Permite extraer valores dinámicos de la URL y utilizarlos dentro del método para procesar la solicitud de manera adecuada.

DELETE http://localhost:8080/api/deleteclient/11 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Description	ooo	Bulk Edit
	Key	Value	Description		

```java
@DeleteMapping("/deleteclient/{id}")
public String deleteExample(@PathVariable Long id) {
    // Lógica para eliminar el recurso con el ID especificado
    clienteServiceImpl.delete(id);
    return "Recurso eliminado con éxito";
}
```

¿Proceso para crear una vista?

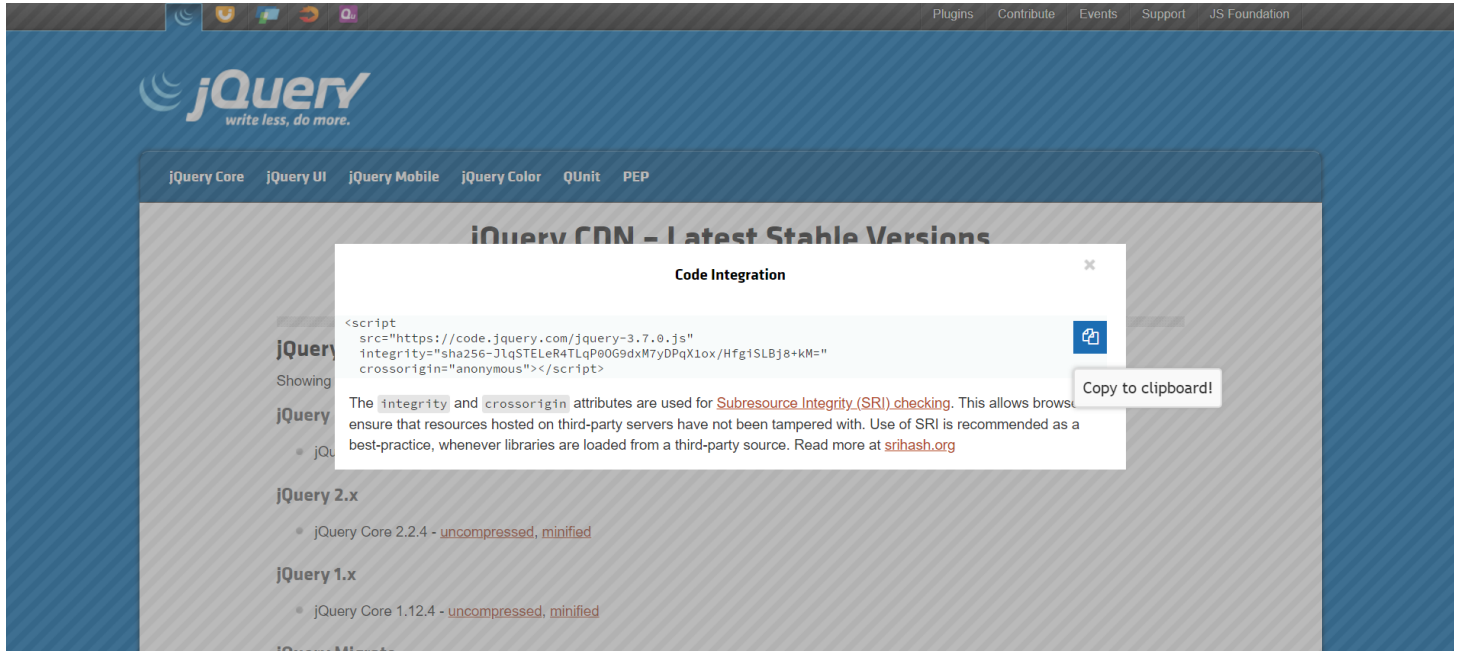
- 1.Crear un archivo HTML en este caso en el escritorio.
- 2.Descargar la plantilla básica de Bootstrap.

Involucrando a la vista

```html
<div class="container">
  <h1>Lista de Clientes</h1>
  <table class="table">
    <thead>
      <tr>
        <th>ID</th>
        <th>Nombre</th>
        <th>Apellido</th>
        <th>Email</th>
        <th>Fecha de Creación</th>
      </tr>
    </thead>
    <tbody id="clientes-body">
      <!-- Los clientes se cargarán aquí dinámicamente -->
    </tbody>
  </table>
</div>
```

Incluir CDN

- acrónimo de "Content Delivery Network", que se traduce al español como "Red de Distribución de Contenido". Un CDN es una red global de servidores distribuidos geográficamente que se utilizan para entregar contenido web de manera eficiente a los usuarios finales.




```
    </table>
  </div>
  <script src="https://code.jquery.com/jquery-3.7.0.js" integrity="sha256-JlqSTELeR4TLqP0OG9dxM7y
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js" integr
</body>
```

Utilizando Ajax

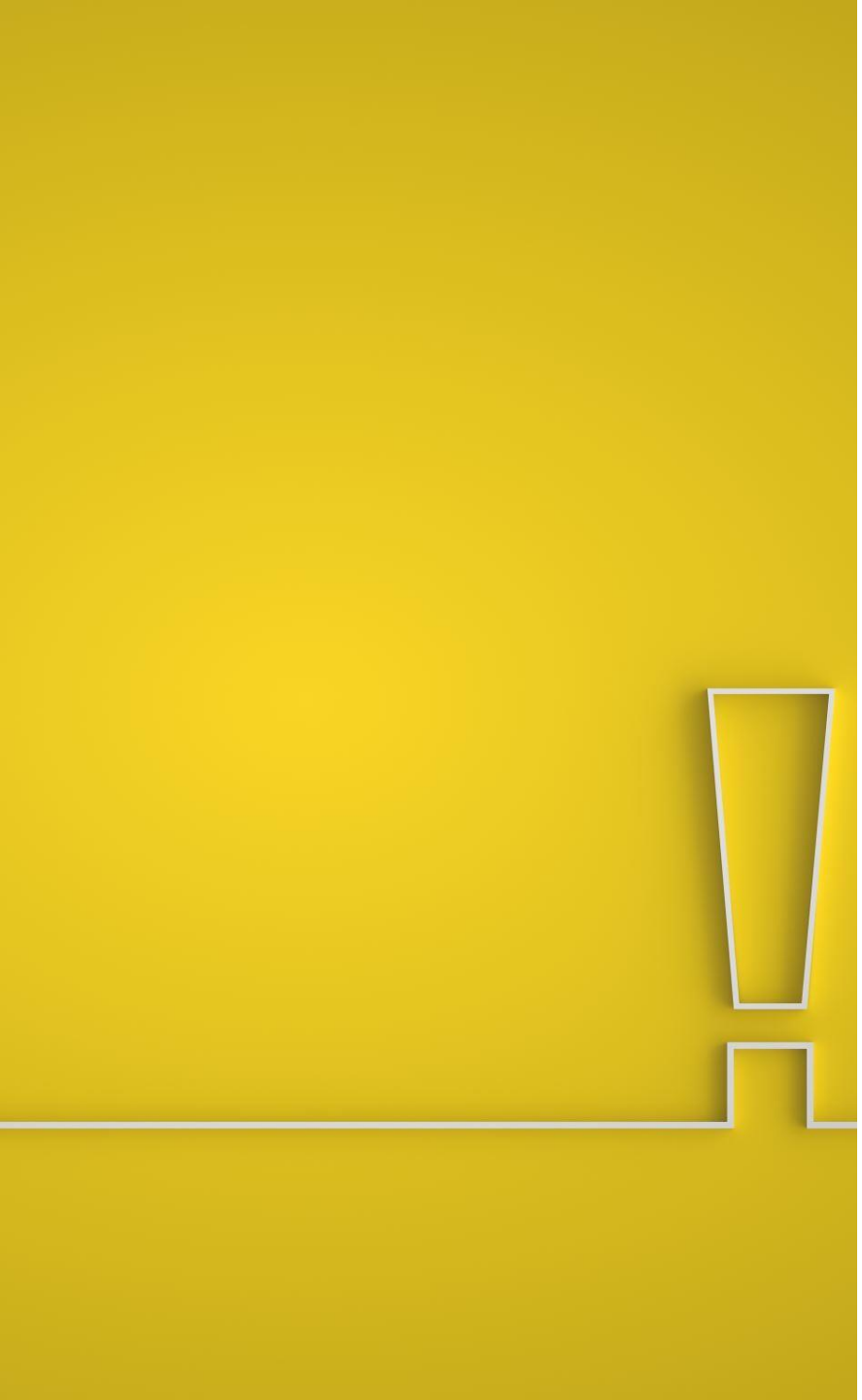
```
<script>
    // Utilizar JavaScript para obtener la lista de clientes mediante una solicitud AJAX
    $(document).ready(function() {
      $.ajax({
        url: "http://localhost:8080/api/clientes", // La ruta debe coincidir con el endpoint defi
        type: "GET",
        dataType: "json",
        success: function(response) {
          // Iterar sobre la lista de clientes y construir las filas de la tabla dinámicamente
          var clientesBody = $("#clientes-body");
          for (var i = 0; i < response.length; i++) {
            var cliente = response[i];
            var row = "<tr>" +
              "<td>" + cliente.id + "</td>" +
              "<td>" + cliente.nombre + "</td>" +
              "<td>" + cliente.apellido + "</td>" +
              "<td>" + cliente.email + "</td>" +
              "<td>" + cliente.createAt + "</td>" +
              "</tr>";
            clientesBody.append(row);
          }
        }
      });
    });
  </script>
```

Explicación

- $(document).ready(function() { ... }): Esta función se ejecuta cuando el documento HTML ha sido completamente cargado. Es una forma común de asegurarse de que el código JavaScript se ejecute después de que se haya cargado todo el contenido HTML.

- $.ajax({ ... }): Esta es una llamada AJAX que utiliza jQuery para realizar una solicitud HTTP al servidor. Aquí se configuran los parámetros de la solicitud, como la URL, el tipo de solicitud, el tipo de datos esperado y las funciones de éxito y error.

- for (var i = 0; i < response.length; i++) { ... }: Este bucle itera sobre la lista de clientes en la respuesta y construye dinámicamente las filas de la tabla.
- var row = "<tr> ... </tr>": Se construye una cadena HTML que representa una fila de la tabla utilizando los datos de cada cliente en la iteración actual.
- clientesBody.append(row): La fila construida se agrega al final del elemento <tbody> utilizando la función append() de jQuery.



Explicación

- url: "http://localhost:8080/api/clientes": Aquí se especifica la URL a la que se enviará la solicitud. Asegúrate de que coincida con el endpoint definido en el controlador.

- type: "GET": Esto indica que se realizará una solicitud HTTP GET para obtener la lista de clientes.
- dataType: "json": Esto especifica que se espera recibir datos en formato JSON como respuesta.

- success: function(response) { ... }: Esta función se ejecuta cuando la solicitud AJAX se completa con éxito y se obtiene la respuesta del servidor. El parámetro response contiene los datos de los clientes devueltos por el servidor en formato JSON.
- var clientesBody = $("#clientes-body"): Aquí se selecciona el elemento <tbody> de la tabla con el ID "clientes-body" utilizando jQuery.

¿Qué es Ajax?

- AJAX es un acrónimo de "Asynchronous JavaScript and XML" (JavaScript y XML Asíncronos). Se trata de una técnica de desarrollo web que permite realizar solicitudes HTTP asíncronas desde una página web sin tener que recargarla por completo. AJAX se basa en varias tecnologías web, como JavaScript, XML, HTML y CSS.

¿Algún ajuste sobre el controlador?

¿Qué es ?

- Es una cabecera HTTP utilizada en el contexto de las políticas de control de acceso entre orígenes (CORS, por sus siglas en inglés). Esta cabecera se utiliza para especificar qué orígenes (dominios, esquemas o combinaciones) tienen permiso para acceder a los recursos de un servidor web mediante una solicitud HTTP.

```
@GetMapping("/clientes")
@CrossOrigin(origins = "*")
public List<Cliente> listaClientes(){
    return clienteServiceImpl.findAll();
}
```

Registro

```
<div class="container">
  <h1>Formulario</h1>
  <form id="formulario">
    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre" placeholder="Ingrese su nombre">
    </div>
    <div class="form-group">
      <label for="apellido">Apellido</label>
      <input type="text" class="form-control" id="apellido" placeholder="Ingrese su apellido">
    </div>
    <div class="form-group">
      <label for="email">Email</label>
      <input type="email" class="form-control" id="email" placeholder="Ingrese su email">
    </div>
    <button type="submit" class="btn btn-primary">Enviar</button>
  </form>
</div>
```

```
<script>
  $(document).ready(function() {
    // Escuchar el evento de envío del formulario
    $("#formulario").submit(function(event) {
      event.preventDefault(); // Evitar que se envíe el formulario de forma tradicional

      // Obtener los valores de los campos del formulario
      var nombre = $("#nombre").val();
      var apellido = $("#apellido").val();
      var email = $("#email").val();

      // Crear el objeto JSON con los datos del formulario
      var datos = {
        nombre: nombre,
        apellido: apellido,
        email: email
      };
```

```
$.ajax({
  url: "http://localhost:8080/api/save",
  type: "POST",
  dataType: "json",
  contentType: "application/json",
  data: JSON.stringify(datos),
  success: function(response) {
    // La solicitud se ha realizado con éxito
    console.log("Solicitud enviada correctamente");
    // Aquí puedes realizar alguna acción adicional, como mostrar un mensaje de éxito o redir
  },
  error: function(xhr, status, error) {
    // Ocurrió un error al enviar la solicitud
    console.log("Error al enviar la solicitud: " + error);
    // Aquí puedes manejar el error de acuerdo a tus necesidades
  }
});
```

```java
@PostMapping("/save")
@CrossOrigin(origins = "*")
public ResponseEntity<Cliente> saveClient(@RequestBody Cliente cliente){

    LocalDateTime localDateTime = LocalDateTime.now();
    Date date = Date.from(localDateTime.atZone(ZoneId.systemDefault()).toInstant());
    cliente.setCreateAt(date);

    Cliente responseBody =clienteServiceImpl.guardar(cliente);
    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-header","Saving Client");

 return  new ResponseEntity<>(responseBody, headers, HttpStatus.OK);

}
```