**@RequestMapping("/caracteristicas")**

El ejemplo proporcionado muestra un controlador de Spring que maneja las solicitudes HTTP para la gestión de características de productos. El controlador tiene los siguientes endpoints:

/agregar: endpoint POST para agregar una nueva característica.

/listar: endpoint GET para listar todas las características existentes.

/modificar/{id}: endpoint PUT para modificar una característica existente por su identificador.

/eliminar/{id}: endpoint DELETE para eliminar una característica existente por su identificador.

/buscar/{id}: endpoint GET para buscar una característica existente por su identificador.

/buscarNombre/{nombre}: endpoint GET para buscar una característica existente por su nombre.

La capa del controlador es responsable de recibir las solicitudes HTTP entrantes y dirigirlas al servicio correspondiente, que se encarga de la lógica de negocio. En este caso, el controlador utiliza la clase CaracteristicasService para llevar a cabo las operaciones de creación, modificación, eliminación y búsqueda de características.

En resumen, el controlador proporciona una interfaz RESTful para la gestión de características de productos y utiliza la capa de servicio para llevar a cabo las operaciones de negocio correspondientes.

**@RequestMapping("/categoria")**

Este es un ejemplo de un controlador de una aplicación que maneja solicitudes HTTP para una entidad de "Categoría". Aquí, el controlador tiene las siguientes anotaciones:

Dentro del controlador, hay varios métodos que manejan diferentes solicitudes HTTP relacionadas con la entidad de "Categoría". Por ejemplo:

@GetMapping("/listar"): se utiliza para manejar una solicitud GET a la ruta /categoria/listar, que devuelve una lista de todas las categorías.

@PostMapping("/agregar"): se utiliza para manejar una solicitud POST a la ruta /categoria/agregar, que agrega una nueva categoría a la base de datos.

@PutMapping("/modificar/{id}"): se utiliza para manejar una solicitud PUT a la ruta /categoria/modificar/{id}, que modifica una categoría existente en la base de datos.

@DeleteMapping("/eliminar/{id}"): se utiliza para manejar una solicitud DELETE a la ruta /categoria/eliminar/{id}, que elimina una categoría existente en la base de datos.

@GetMapping("/buscar/{id}"): se utiliza para manejar una solicitud GET a la ruta /categoria/buscar/{id}, que busca una categoría por su ID.

@GetMapping("/buscarTitulo/{titulo}"): se utiliza para manejar una solicitud GET a la ruta /categoria/buscarTitulo/{titulo}, que busca una categoría por su título.

@GetMapping("/buscarDescripcion/{descripcion}"): se utiliza para manejar una solicitud GET a la ruta /categoria/buscarDescripcion/{descripcion}, que busca una categoría por su descripción.

En general, este controlador maneja solicitudes HTTP relacionadas con la entidad de "Categoría" y realiza operaciones CRUD básicas en la base de datos.

**@RequestMapping("/ciudad")**

Este ejemplo muestra un controlador de la capa de controladores de una aplicación web. El controlador se encarga de manejar las solicitudes HTTP entrantes y dirigirlas al servicio correspondiente.

La clase CiudadController tiene varias anotaciones, cada una con un propósito específico:

@RequestMapping("/ciudad"): establece la ruta base para todas las solicitudes en este controlador.

El controlador tiene varios métodos que manejan solicitudes HTTP específicas:

@PostMapping("/agregar"): maneja las solicitudes POST en la ruta /ciudad/agregar. Este método valida la entrada con @Valid y luego comprueba si el nombre de la ciudad ya existe en la base de datos a través de la llamada al servicio CiudadService. Si el nombre de la ciudad ya existe, devuelve un mensaje de error; de lo contrario, agrega la ciudad a través de CiudadService.

@GetMapping("/listar"): maneja las solicitudes GET en la ruta /ciudad/listar. Este método devuelve una lista de todas las ciudades utilizando CiudadService.

@PutMapping("/modificar/{id}"): maneja las solicitudes PUT en la ruta /ciudad/modificar/{id}. Este método valida la entrada con @Valid y luego comprueba si el ID proporcionado existe en la base de datos a través de la llamada al servicio CiudadService. Si el ID no existe, devuelve un mensaje de error; de lo contrario, comprueba si ya existe una ciudad con el mismo nombre en la base de datos. Si una ciudad con el mismo nombre ya existe, devuelve un mensaje de error; de lo contrario, modifica la ciudad a través de CiudadService.

@DeleteMapping("/eliminar/{id}"): maneja las solicitudes DELETE en la ruta /ciudad/eliminar/{id}. Este método comprueba si el ID proporcionado existe en la base de datos a través de la llamada al servicio CiudadService. Si el ID no existe, devuelve un mensaje de error; de lo contrario, comprueba si hay productos o usuarios asociados con la ciudad antes de eliminar la ciudad a través de CiudadService.

@GetMapping("/buscar/{id}"): maneja las solicitudes GET en la ruta /ciudad/buscar/{id}. Este método busca la ciudad correspondiente al ID proporcionado a través de la llamada al servicio CiudadService. Si la ciudad no existe, devuelve un mensaje de error; de lo contrario, devuelve la ciudad.

@GetMapping("/buscarCiudad/{ciudad}"): maneja las solicitudes GET en la ruta /ciudad/buscarCiudad/{ciudad}. Este método busca la ciudad correspondiente al nombre proporcionado a través de la llamada al servicio CiudadService. Si la ciudad no existe, devuelve un mensaje de error; de lo contrario, devuelve la ciudad

**@RequestMapping("/imagen")**

El código proporcionado es un ejemplo de un controlador en una aplicación web basada en Spring Framework. El controlador maneja solicitudes HTTP entrantes relacionadas con la gestión de imágenes de productos y se comunica con un servicio de imágenes a través de inyección de dependencia.

método listar() devuelve una lista de todas las imágenes de productos

método buscarPorId() devuelve una imagen de producto específica según el id proporcionado en la URL.

método buscarTitulo() devuelve una lista de imágenes de productos que tienen un título específico.

método modificar() actualiza una imagen de producto existente según el id proporcionado en la URL.

En el caso de que la imagen no exista para el id proporcionado en la URL, el controlador devuelve una respuesta HTTP con un código de estado 404 Not Found y un mensaje de error. Si la imagen existe, el controlador devuelve una respuesta HTTP con un código de estado 201 Created y un mensaje de éxito.

En resumen, el controlador proporcionado es responsable de manejar solicitudes HTTP relacionadas con la gestión de imágenes de productos y se comunica con un servicio de imágenes a través de inyección de dependencia.

**@RequestMapping("/politicas")**

El código presentado es un ejemplo de un controlador llamado "PoliticasController" que maneja solicitudes HTTP entrantes y las dirige al servicio correspondiente. Este controlador tiene una serie de métodos para listar, buscar y eliminar políticas.

Finalmente, "@RequestMapping" se utiliza para mapear la ruta "/politicas" a este controlador en particular.

Los métodos en este controlador son:

"listar": devuelve una lista de todas las políticas.

"eliminar": elimina una política por su id. Si la política no existe, devuelve un error 404. Si la política tiene productos asociados, devuelve un error 400.

"buscarPorId": busca una política por su id y devuelve la política si la encuentra. Si la política no existe, devuelve un error 404.

"buscarNombre": busca una o varias políticas por su nombre y devuelve una lista de políticas que coinciden con el nombre. Si no hay coincidencias, devuelve un error 404.

"buscarDescripcion": busca una o varias políticas por su descripción y devuelve una lista de políticas que coinciden con la descripción. Si no hay coincidencias, devuelve un error 404.

**@RequestMapping("/productos")**

El controlador ProductoController maneja solicitudes HTTP relacionadas con los productos en una aplicación web de comercio electrónico. Los servicios asociados son ProductoService, CiudadService, CategoriaService, ImagenService, ReservaService y PuntuacionService.

La anotación @RequestMapping("/productos") establece la ruta base para todas las solicitudes en este controlador.

El controlador tiene varios métodos que manejan solicitudes HTTP específicas:

El método agregar() maneja las solicitudes POST en la ruta /productos/agregar. Este método valida la entrada con @Valid y luego comprueba si los ID de ciudad y categoría existen en la base de datos a través de las llamadas a los servicios CiudadService y CategoriaService. Luego, comprueba si los ID de características son válidos, buscando si existen en la base de datos y eliminando cualquier ID repetido. Si no se cumplen las condiciones anteriores, se devuelve un mensaje de error; de lo contrario, se agrega el producto a través de ProductoService.

El método modificarProductoCompleto() maneja las solicitudes PUT en la ruta /productos/modificarProductoCompleto/{id}. Este método busca un producto por ID a través de la llamada a ProductoService y, si existe, lo actualiza con la información proporcionada en el cuerpo de la solicitud a través de ProductoService.

El método modificar() maneja las solicitudes PATCH en la ruta /productos/modificar/{id}. Este método busca un producto por ID a través de la llamada a ProductoService y, si existe, lo actualiza con la información proporcionada en el cuerpo de la solicitud a través de ProductoService. Este método solo actualiza los campos que se proporcionan en el cuerpo de la solicitud y deja los campos no proporcionados sin cambios.

El método modificarCaracteristicas() maneja las solicitudes PATCH en la ruta /productos/modificarCaracteristicas/{id}. Este método busca un producto por ID a través de la llamada a ProductoService y, si existe, lo actualiza con la lista de nuevas características proporcionada en el cuerpo de la solicitud a través de ProductoService. Si los ID de características proporcionados no son válidos, se devuelve un mensaje de error.

El primer método modificarCiudad utiliza el verbo PATCH y se encarga de modificar la ciudad a la que pertenece un producto en particular, recibiendo como parámetros el objeto ProductoDto que contiene el nuevo valor de la ciudad, y el id del producto que se va a modificar. Si el producto existe en la base de datos, se actualiza la ciudad y se retorna un mensaje de éxito en el cuerpo de la respuesta. Si la ciudad proporcionada no existe en la base de datos, se retorna un mensaje de error con el código de estado HTTP NOT\_FOUND.

El segundo método modificarCategoria es similar al método anterior, pero en este caso se encarga de modificar la categoría a la que pertenece un producto. Recibe como parámetros el objeto ProductoDto que contiene el nuevo valor de la categoría, y el id del producto que se va a modificar. Si el producto existe en la base de datos, se actualiza la categoría y se retorna un mensaje de éxito en el cuerpo de la respuesta. Si la categoría proporcionada no existe en la base de datos, se retorna un mensaje de error con el código de estado HTTP NOT\_FOUND.

El método listar utiliza el verbo GET y retorna una lista de todos los productos almacenados en la base de datos.

El método listarProductosRandom también utiliza el verbo GET y retorna una lista de productos seleccionados aleatoriamente de la base de datos.

El método buscarCiudad utiliza el verbo GET y recibe como parámetro el nombre de una ciudad. Retorna una lista de productos que pertenecen a esa ciudad en particular. Si no se encuentra ningún producto que pertenezca a la ciudad proporcionada, se retorna un mensaje de error con el código de estado HTTP NOT\_FOUND.

El método buscarCaracteristicas utiliza el verbo GET y recibe como parámetro el nombre de una característica. Retorna una lista de productos que tienen la característica proporcionada. Si no se encuentra ningún producto con la característica proporcionada, se retorna un mensaje de error con el código de estado HTTP NOT\_FOUND.

El método buscarTituloCategoria utiliza el verbo GET y recibe como parámetro el título de una categoría. Retorna una lista de productos que pertenecen a la categoría con el título proporcionado. Si no se encuentra ningún producto que pertenezca a la categoría proporcionada, se retorna un mensaje de error con el código de estado HTTP NOT\_FOUND.

El método buscarDescripcionCategoria busca productos por su descripción de categoría, el método buscarTituloPoliticas busca productos por el título de políticas y el método buscarDescripcionPoliticas busca productos por la descripción de políticas.

El método buscarNombre busca productos por su nombre y el método buscarDescripcion busca productos por su descripción. Cada método devuelve una lista de productos que coinciden con los criterios de búsqueda especificados. Si la lista está vacía, devuelve un código de estado HTTP 404 (Not Found) y un mensaje que indica que no se encontraron productos. De lo contrario, devuelve la lista de productos y un código de estado HTTP 200 (OK).

El método eliminar elimina un producto por su ID. Si el producto no existe, devuelve un código de estado HTTP 404 (Not Found) y un mensaje que indica que el ID no pertenece a ningún producto. Si hay reservas asociadas al producto, devuelve un código de estado HTTP 400 (Bad Request) y un mensaje que indica que no se puede eliminar el producto porque tiene reservas asociadas. De lo contrario, elimina el producto y sus imágenes y puntuaciones asociadas y devuelve un código de estado HTTP 200 (OK) y un mensaje que indica que el producto se eliminó con éxito.

El método eliminarCaracteristicas utiliza el verbo DELETE y recibe el id del producto y una lista de ids de características para eliminar de dicho producto. Primero, se busca el producto por su id y, si existe, se verifican si los ids de características son válidos (existen y no están repetidos). Luego, se llama al servicio para eliminar las características del producto. Si se realiza la eliminación exitosamente, se retorna una respuesta con estado 200 OK y un mensaje indicando que las características se eliminaron correctamente. Si no se encuentra el producto, se retorna una respuesta con estado 404 Not Found y un mensaje indicando que el id del producto no existe. Si los ids de características son inválidos, se retorna una respuesta con estado 404 Not Found y un mensaje indicando los ids que son inexistentes o repetidos.

El método eliminarImagenes también utiliza el verbo DELETE y recibe el id del producto y una lista de ids de imágenes para eliminar de dicho producto. Primero, se busca el producto por su id y, si existe, se verifican si los ids de imágenes son válidos (existen y no están repetidos). Luego, se llama al servicio para eliminar las imágenes del producto. Si se realiza la eliminación exitosamente, se retorna una respuesta con estado 200 OK y un mensaje indicando que las imágenes se eliminaron correctamente. Si no se encuentra el producto, se retorna una respuesta con estado 404 Not Found y un mensaje indicando que el id del producto no existe. Si los ids de imágenes son inválidos, se retorna una respuesta con estado 404 Not Found y un mensaje indicando los ids que son inexistentes o repetidos.

El método findAllByCaracteristicas\_IdIn utiliza el verbo GET y recibe una lista de ids de características para buscar productos que tengan al menos una de estas características. Se llama al servicio para buscar los productos que cumplan esta condición y se retorna una respuesta con estado 200 OK y la lista de productos encontrados si existen. Si no se encuentran productos, se retorna una respuesta con estado 404 Not Found y un mensaje indicando que los ids de características no pertenecen a ningún producto.

El método findAllByCategoriasIds utiliza el verbo GET y recibe una lista de ids de categorías para buscar productos que pertenezcan a al menos una de estas categorías. Se llama al servicio para buscar los productos que cumplan esta condición y se retorna una respuesta con estado 200 OK y la lista de productos encontrados si existen. Si no se encuentran productos, se retorna una respuesta con estado 404 Not Found y un mensaje indicando que los ids de categorías no pertenecen a ningún producto.

El método findAllByCiudadesIds utiliza el verbo GET y recibe una lista de ids de ciudades para buscar productos que pertenezcan a al menos una de estas ciudades. Se llama al servicio para buscar los productos que cumplan esta condición y se retorna una respuesta con estado 200 OK y la lista de productos encontrados si existen. Si no se encuentran productos, se retorna una respuesta con estado 404 Not Found y un mensaje indicando que los ids de ciudades no pertenecen a ningún producto

El endpoint findAllByCiudad corresponde a una solicitud GET que recibe un idCiudad en la URL y devuelve una lista de productos que pertenecen a esa ciudad. Si la lista está vacía, devuelve un mensaje de error.

El endpoint eliminarDescripcionPolitica corresponde a una solicitud DELETE que recibe tres parámetros: productoId, politicaId y descripcion. Este endpoint elimina la descripción de una política en un producto y devuelve el producto actualizado. Si no se encuentra el producto, devuelve un mensaje de error.

El endpoint actualizarDescripcionPolitica corresponde a una solicitud PUT que recibe cuatro parámetros: productoId, politicaId, descripcion y newDescripcion. Este endpoint actualiza la descripción de una política en un producto y devuelve el producto actualizado. Si no se encuentra el producto, devuelve un mensaje de error.

El endpoint agregarDescripcionPolitica corresponde a una solicitud POST que recibe tres parámetros: productoId, politicaId y descripcion. Este endpoint agrega una nueva descripción de una política en un producto y devuelve el producto actualizado. Si no se encuentra el producto, devuelve un mensaje de error.

El endpoint agregarCaracteristica corresponde a una solicitud POST que recibe dos parámetros: productoId e idCaracteristicas. Este endpoint agrega una o más características a un producto y devuelve el producto actualizado. Si no se encuentra el producto, devuelve un mensaje de error.

El endpoint listarCaracteristicasNoAgregadas corresponde a una solicitud GET que recibe un productoId en la URL. Este endpoint devuelve una lista de características que aún no se han agregado a un producto en particular. Si no se encuentra el producto, devuelve un mensaje de error.

**@RequestMapping("/puntuacion")**

Este es un controlador de Spring que maneja las solicitudes relacionadas con las puntuaciones de productos. Tiene tres atributos: puntuacionService, usuarioService y productoService, que son instancias de servicios que se utilizan para realizar operaciones relacionadas con las puntuaciones, los usuarios y los productos, respectivamente.

La función agregar maneja la solicitud POST para agregar una nueva puntuación. Toma un objeto PuntuacionDto como parámetro, que contiene información sobre la puntuación, el usuario y el producto. Comprueba si el usuario y el producto existen en la base de datos y si ya existe una puntuación para ese usuario y producto. Si todo está bien, agrega la nueva puntuación y devuelve una respuesta con un estado HTTP 201 (CREATED) y un mensaje de éxito.

La función agregarPuntuacionRandom maneja la solicitud POST para agregar varias puntuaciones aleatorias. Genera aleatoriamente el ID del usuario, el ID del producto y la puntuación, y luego comprueba si ya existe una puntuación para ese usuario y producto. Si no existe, agrega la nueva puntuación y aumenta el contador. El ciclo for continúa mientras el contador sea menor que 20 y no se hayan hecho más de 100 iteraciones. Finalmente, devuelve una respuesta con un estado HTTP 201 (CREATED) y un mensaje que indica cuántas puntuaciones se agregaron con éxito.

La función listar maneja la solicitud GET para obtener todas las puntuaciones de la base de datos. Devuelve una lista de objetos Puntuacion.

El primer método modificarProducto actualiza el producto de una Puntuación con un nuevo id de producto, después de validar que el nuevo id de producto existe y que no hay otra Puntuación con el mismo id de producto. El segundo método modificarUsuario hace lo mismo, pero actualiza el usuario de la Puntuación con un nuevo id de usuario.

El tercer método modificarPuntuacion actualiza la puntuación de una Puntuación con un nuevo valor de puntuación, después de validar que el valor de puntuación es válido (entre 1 y 5).

El cuarto método eliminar elimina una Puntuación por su id, después de validar que la Puntuación existe.

El primer método, eliminarByUsuarioId, elimina todas las puntuaciones realizadas por un usuario determinado. El segundo método, eliminarByProductoId, elimina todas las puntuaciones asociadas a un producto determinado. El tercer método, buscarPorId, busca una puntuación por su ID. Los métodos buscarUsuarioId y buscarProductoId buscan todas las puntuaciones realizadas por un usuario o un producto, respectivamente. El método buscarPuntuacion busca todas las puntuaciones que tengan un valor determinado. Los métodos obtenerPromedioAlls y obtenerPromedioPuntuacionPorProducto calculan el promedio de las puntuaciones para todos los productos y para un producto determinado, respectivamente.

**@RequestMapping("/reservas")**

El método "agregar" utiliza el verbo HTTP POST y permite agregar una nueva reserva al sistema. Recibe como parámetro un objeto ReservaDto que contiene los datos de la reserva a agregar. Verifica que el producto y usuario especificados existan y que no exista una reserva que se superponga con las fechas especificadas. Si todo está correcto, agrega la reserva a la base de datos y envía un correo electrónico de confirmación al usuario.

El método "agregarReservaRandom" también utiliza el verbo HTTP POST y genera aleatoriamente 20 reservas a partir de una lista de 20 productos y 15 clientes. La fecha inicial de la reserva debe ser mayor o igual a la fecha actual y menor o igual a 30 días a partir de la fecha actual. También genera aleatoriamente la hora de inicio de la reserva. Retorna un mensaje de éxito si todas las reservas fueron creadas correctamente.

En el método modificarProducto, se está recibiendo un objeto ReservaDto en el cuerpo de la petición y un parámetro id en la URL. Luego, se busca la reserva correspondiente al id proporcionado y se valida si existe o no. Si la reserva existe, se busca el producto nuevo que se quiere asignar a la reserva y se valida si existe o no. Si el producto no existe, se devuelve una respuesta con el código de estado 404 (No encontrado) y un mensaje de error indicando que no se encontró el producto.

Si el producto existe, se valida si hay alguna otra reserva que ya tenga asignado el mismo producto y que se superponga en rango de fechas con la reserva que se está intentando modificar. Si se encuentra una reserva que cumple estas condiciones, se devuelve una respuesta con el código de estado 400 (Solicitud incorrecta) y un mensaje de error indicando que ya existe una reserva para el mismo producto y un rango de fechas que se superpone.

Si no se encuentra ninguna reserva que cumpla estas condiciones, se llama al método modificarProducto del servicio ReservaService y se devuelve una respuesta con el código de estado 201 (Creado) y un mensaje de éxito indicando que se ha modificado el producto para la reserva correspondiente al id proporcionado.

En el método modificar, se está recibiendo un objeto ReservaDto en el cuerpo de la petición y un parámetro id en la URL. Luego, se busca la reserva correspondiente al id proporcionado y se valida si existe o no. Si la reserva existe, se valida si hay alguna otra reserva que ya tenga asignado el mismo producto y que se superponga en rango de fechas con la reserva que se está intentando modificar. Si se encuentra una reserva que cumple estas condiciones, se devuelve una respuesta con el código de estado 400 (Solicitud incorrecta) y un mensaje de error indicando que ya existe una reserva para el mismo producto y un rango de fechas que se superpone.

Si no se encuentra ninguna reserva que cumpla estas condiciones, se llama al método modificar del servicio ReservaService y se devuelve una respuesta con el código de estado 201 (Creado) y un mensaje de éxito indicando que se ha modificado la reserva correspondiente al id proporcionado. Si la reserva no existe, se devuelve una respuesta con el código de estado 404 (No encontrado) y un mensaje de error indicando que no se encontró la reserva correspondiente al id proporcionado.

El primer método, eliminar, utiliza la anotación @DeleteMapping para indicar que maneja solicitudes DELETE en la ruta /eliminar/{id}, donde {id} es el ID de la reserva que se eliminará. El método busca la reserva por su ID utilizando el servicio reservaService, y si la reserva no existe, devuelve una respuesta con un estado HTTP 404 (No encontrado) y un mensaje de error. Si la reserva existe, crea y envía un mensaje de correo electrónico utilizando JavaMailSender, elimina la reserva utilizando reservaService y devuelve una respuesta con un estado HTTP 200 (Éxito) y un mensaje indicando que la reserva se eliminó con éxito.

El segundo método, eliminarByCliente\_Id, utiliza la anotación @DeleteMapping para indicar que maneja solicitudes DELETE en la ruta /eliminarByCliente\_Id/{id}, donde {id} es el ID del cliente cuyas reservas se eliminarán. El método busca todas las reservas asociadas con el cliente utilizando reservaService, y si no hay ninguna reserva asociada con el cliente, devuelve una respuesta con un estado HTTP 404 y un mensaje de error. Si hay reservas asociadas con el cliente, elimina cada reserva utilizando reservaService y elimina el cliente utilizando clienteService. Finalmente, devuelve una respuesta con un estado HTTP 200 y un mensaje indicando que las reservas se eliminaron con éxito.

El tercer método, eliminarByProducto\_Id, utiliza la anotación @DeleteMapping para indicar que maneja solicitudes DELETE en la ruta /eliminarByProducto\_Id/{id}, donde {id} es el ID del producto cuyas reservas se eliminarán. El método busca todas las reservas asociadas con el producto utilizando reservaService, y si no hay ninguna reserva asociada con el producto, devuelve una respuesta con un estado HTTP 404 y un mensaje de error. Si hay reservas asociadas con el producto, elimina cada reserva utilizando reservaService. Finalmente, devuelve una respuesta con un estado HTTP 200 y un mensaje indicando que las reservas se eliminaron con éxito.

El primer método, buscarPorId, recibe un ID y devuelve la reserva correspondiente, si existe, o un mensaje de error si no existe.

El segundo método, buscarClienteId, recibe un ID de cliente y devuelve una lista de todas las reservas asociadas a ese cliente.

El tercer método, buscarCorreoCliente, recibe un correo electrónico de cliente y devuelve una lista de todas las reservas asociadas a ese cliente.

El cuarto método, buscarProductoId, recibe un ID de producto y devuelve una lista de todas las reservas asociadas a ese producto.

El quinto método, findProductosDisponiblesByCiudadAndFechas, recibe una ciudad y un rango de fechas y devuelve una lista de todos los productos disponibles en esa ciudad en esas fechas.

El sexto método, findProductosDisponiblesByFechas, recibe un rango de fechas y devuelve una lista de todos los productos disponibles en esas fechas.

**@RequestMapping("/roles")**

El método que ya utiliza la anotación correspondiente es el método eliminar, que utiliza @DeleteMapping("/eliminar/{id}") para manejar solicitudes DELETE en la ruta /eliminar/{id}. Este método busca el rol por su ID utilizando rolService.buscarPorId(id). Si el rol no existe, devuelve una respuesta con un estado HTTP 404 (No encontrado) y un mensaje de error. Si el rol existe, verifica si hay usuarios asociados a este rol utilizando usuarioService.findByRol\_Id(id). Si hay usuarios asociados, devuelve una respuesta con un estado HTTP 400 (Solicitud incorrecta) y un mensaje de error. Si no hay usuarios asociados, elimina el rol utilizando rolService.eliminar(id) y devuelve una respuesta con un estado HTTP 200 (Éxito) y un mensaje indicando que el rol se eliminó correctamente.

Los otros métodos de la clase RolController que no utilizan anotaciones correspondientes para indicar el tipo de solicitud HTTP son:

agregar utiliza @PostMapping("/agregar") para manejar solicitudes POST en la ruta /agregar. Este método agrega un nuevo rol a través de rolService.agregar(rol). Si el rol ya existe, devuelve una respuesta con un estado HTTP 400 (Solicitud incorrecta) y un mensaje de error. Si se agrega correctamente, devuelve una respuesta con un estado HTTP 201 (Creado) y un mensaje indicando que el rol se guardó correctamente.

listar utiliza @GetMapping("/listar") para manejar solicitudes GET en la ruta /listar. Este método lista todos los roles utilizando rolService.listar().

modificar utiliza @PutMapping("/modificar/{id}") para manejar solicitudes PUT en la ruta /modificar/{id}. Este método modifica un rol existente a través de rolService.modificar(rol, id). Si el rol no existe, devuelve una respuesta con un estado HTTP 404 (No encontrado) y un mensaje de error. Si el nuevo nombre del rol ya existe, devuelve una respuesta con un estado HTTP 400 (Solicitud incorrecta) y un mensaje de error. Si se modifica correctamente, devuelve una respuesta con un estado HTTP 201 (Creado) y un mensaje indicando que el rol se modificó correctamente.

buscarPorId utiliza @GetMapping("/buscar/{id}") para manejar solicitudes GET en la ruta /buscar/{id}. Este método busca un rol por su ID utilizando rolService.buscarPorId(id). Si el rol no existe, devuelve una respuesta con un estado HTTP 404 (No encontrado) y un mensaje de error. Si el rol existe, devuelve una respuesta con un estado HTTP 200 (Éxito) y el rol encontrado.

buscarNombre utiliza @GetMapping("/buscarNombre/{nombre}") para manejar solicitudes GET en la ruta /buscarNombre/{nombre}. Este método busca un rol por su nombre utilizando rolService.getByNombre(nombre). Si el rol no existe, devuelve una respuesta con un estado HTTP 404 (No encontrado) y un mensaje de error. Si el rol existe, devuelve una respuesta con un estado HTTP 200 (Éxito) y el rol encontrado.

**@RequestMapping("/usuarios")**

El primer método es un controlador POST que maneja solicitudes para agregar un usuario a través de la ruta /usuarios/agregar. Si el correo electrónico del usuario ya existe en la base de datos, el método devuelve una respuesta con un estado HTTP 400 (Solicitud incorrecta) y un mensaje de error. El método crea un mensaje de correo electrónico utilizando JavaMailSender y envía una confirmación de registro al usuario después de agregarlo exitosamente a través de usuarioService. Si no se encuentra un rol o una ciudad con los IDs proporcionados en la solicitud, el método devuelve una respuesta con un estado HTTP 404 (No encontrado) y un mensaje de error.

El segundo método es un controlador POST que maneja solicitudes para agregar un usuario aleatorio a través de la ruta /usuarios/agregarUsuarioRandom. El método genera aleatoriamente valores para el nombre, el apellido, el ID de rol, el ID de ciudad y el correo electrónico del usuario. Luego, crea un nuevo usuario utilizando usuarioService y agrega el usuario a una lista de usuarios creados. Después de crear 15 usuarios, el método devuelve una respuesta con un estado HTTP 200 (Éxito) y la lista de usuarios creados en forma de mapa.

Este código es una implementación de una API REST que permite gestionar usuarios y clientes en un sistema. A continuación, se detallan las operaciones que pueden ser realizadas:

listar(): retorna una lista con todos los usuarios del sistema.

listarClientes(): retorna una lista con todos los clientes del sistema.

modificarRol(@RequestBody UsuarioDto usuarioDto, @PathVariable Long id): recibe como parámetros un objeto UsuarioDto que contiene el nuevo rol y el ID del usuario a modificar. Retorna un objeto ResponseEntity que indica si se realizó la operación con éxito o no.

modificarCiudad(@RequestBody UsuarioDto usuarioDto, @PathVariable Long id): recibe como parámetros un objeto UsuarioDto que contiene la nueva ciudad y el ID del usuario a modificar. Retorna un objeto ResponseEntity que indica si se realizó la operación con éxito o no.

modificar(@RequestBody UsuarioDto usuarioDto, @PathVariable Long id): recibe como parámetros un objeto UsuarioDto con los datos del usuario a modificar y el ID del usuario a modificar. Retorna un objeto ResponseEntity que indica si se realizó la operación con éxito o no.

eliminar(@PathVariable Long id): recibe como parámetro el ID del usuario a eliminar. Retorna un objeto ResponseEntity que indica si se realizó la operación con éxito o no.

buscarPorId(@PathVariable Long id): recibe como parámetro el ID del usuario a buscar. Retorna un objeto ResponseEntity que contiene al usuario si se encuentra en el sistema, y un mensaje de error si no se encuentra.

buscarClientePorId(@PathVariable Long id): recibe como parámetro el ID del cliente a buscar. Retorna un objeto ResponseEntity que contiene al cliente si se encuentra en el sistema, y un mensaje de error si no se encuentra.

buscarEmail(@PathVariable String email): recibe como parámetro el email del usuario a buscar. Retorna un objeto ResponseEntity que contiene al usuario si se encuentra en el sistema, y un mensaje de error si no se encuentra.

buscarNombre(@PathVariable String nombre): recibe como parámetro el nombre del usuario a buscar. Retorna un objeto ResponseEntity que contiene a los usuarios si se encuentra en el sistema, y un mensaje de error si no se encuentra.

En resumen, esta API permite listar, modificar, eliminar y buscar usuarios y clientes en un sistema, y provee diferentes formas de búsqueda, como por ID, email o nombre.