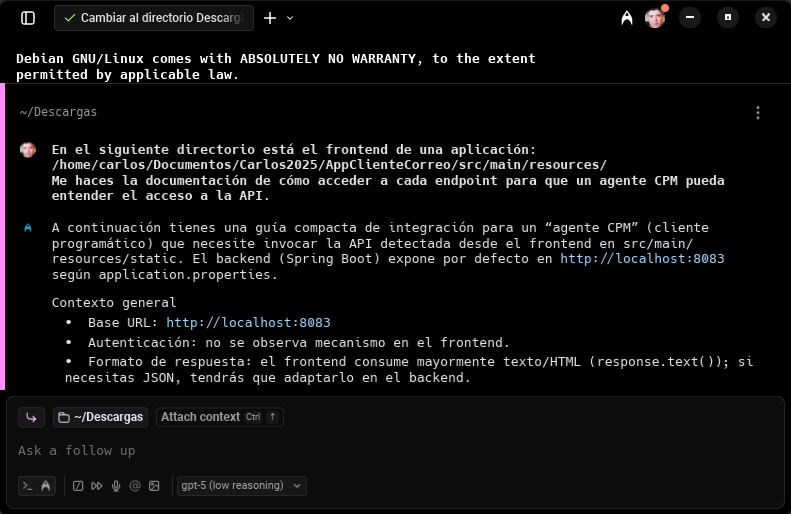
Descargar e instalar Warp terminal: https://www.warp.dev/download



Prompt 01:

En el siguiente directorio está el frontend de una aplicación:

/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/

Me haces la documentación de cómo acceder a cada endpoint para que un programador pueda entender el acceso a la API.

Prompt 02

En el siguiente directorio está el frontend de una aplicación:

/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/

Me haces la documentación de cómo acceder a cada endpoint para que un agente CPM pueda entender el acceso a la API.

Suggest new conversation rejected for message

Para generar la documentación de los endpoints, primero analizaré el código del frontend en ese directorio para identificar las llamadas a la API (fetch/axios, rutas /api, etc.). Empezaré por listar los archivos relevantes y buscar patrones de llamadas HTTP. Ejecutaré varios comandos en paralelo para hacerlo más rápido.

\*\*File Glob Results:\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/applicationCopia.properties\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/borrarClientes.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/altaCorreos.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/editarCorreos.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/editarClientes.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/index.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/borrarCorreos.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/listados.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/altaClientes.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/application.properties\*\*

\*\*Grep Results:\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/editarClientes.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/borrarClientes.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/borrarCorreos.html\*\*

- \*\*/home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/static/editarCorreos.html\*\*

Voy a abrir los archivos HTML y de configuración en ese directorio para identificar las llamadas fetch a la API y cualquier configuración de rutas, y así redactar la documentación precisa de los endpoints.

Perfecto. Con base en los archivos del frontend, estos son los endpoints que el cliente web llama. El servidor (Spring Boot) corre por defecto en http://localhost:8083 según application.properties.

Contexto general

- Base URL: http://localhost:8083

- Autenticación: no se observa ninguna en el frontend.

- Formato de respuesta: en varios casos el frontend usa response.text() o muestra directamente en iframes, por lo que las respuestas suelen ser texto/HTML (no JSON).

- Codificación de parámetros: cuando pases valores con espacios o caracteres especiales (por ejemplo correos o nombres), codifícalos en URL. Ejemplo: Juan Pérez → Juan%20P%C3%A9rez.

Clientes

1) Listar todos los clientes

- Método: GET

- Path: /clientes/listartodos

- Query params: ninguno

- Ejemplo curl:

curl -X GET http://localhost:8083/clientes/listartodos

2) Buscar cliente por DNI

- Método: GET

- Path: /clientes/buscarpordni

- Query params:

- dni: string (obligatorio para resultados; si se omite, el frontend indica que retorna vacío/nulo)

- Ejemplo curl:

curl -G http://localhost:8083/clientes/buscarpordni --data-urlencode "dni=12345678A"

3) Crear (alta) cliente

- Método: GET

- Path: /clientes/guardar/{dni}/{nombre}/{apellido}

- Path params:

- dni: string

- nombre: string

- apellido: string

- Nota: Aunque lo habitual sería usar POST, el frontend construye una URL GET para crear.

- Ejemplo curl:

curl -X GET "http://localhost:8083/clientes/guardar/12345678A/Juan/Perez"

4) Actualizar cliente

- Método: POST

- Path: /clientes/actualizar/{dni}/{nombre}/{apellido}

- Path params:

- dni: string

- nombre: string

- apellido: string

- Cuerpo: no se envía cuerpo; los datos van en la ruta.

- Ejemplo curl:

curl -X POST "http://localhost:8083/clientes/actualizar/12345678A/Juan/Lopez"

5) Borrar cliente

- Método: DELETE

- Path: /clientes/borrar/{dni}

- Path params:

- dni: string

- Ejemplo curl:

curl -X DELETE "http://localhost:8083/clientes/borrar/12345678A"

Correos

1) Listar todos los correos (con filtro opcional)

- Método: GET

- Path: /correos/listartodos

- Query params:

- buscarCorreoDni: string (opcional; puede ser parte de un correo o un DNI; si se omite, lista todo)

- Ejemplo curl (sin filtro):

curl -X GET "http://localhost:8083/correos/listartodos"

- Ejemplo curl (con filtro):

curl -G "http://localhost:8083/correos/listartodos" --data-urlencode "buscarCorreoDni=12345678A"

2) Ver clientes relacionados con correos (listado)

- Método: GET

- Path: /correos/verclientescorreos

- Query params:

- buscarClientesCorreosDNI: string (opcional; puede ser parte de un DNI o correo; si se omite, lista todo)

- Ejemplo curl:

curl -G "http://localhost:8083/correos/verclientescorreos" --data-urlencode "buscarClientesCorreosDNI=gmail.com"

3) Crear (alta) correo asociado a un DNI

- Método: GET

- Path: /correos/guardar/{dni}/{correo}

- Path params:

- dni: string

- correo: string (URL-encode si contiene @ u otros caracteres, p. ej. usuario%40servidor.com)

- Ejemplo curl:

curl -X GET "http://localhost:8083/correos/guardar/12345678A/usuario%40servidor.com"

4) Actualizar correo por ID

- Método: POST

- Path: /correos/actualizar/{idCorreo}/{correo}

- Path params:

- idCorreo: entero

- correo: string (URL-encode)

- Cuerpo: no se envía cuerpo; los datos van en la ruta.

- Ejemplo curl:

curl -X POST "http://localhost:8083/correos/actualizar/42/nuevo%40servidor.com"

5) Borrar correo por ID

- Método: DELETE

- Path: /correos/borrar/{idCorreo}

- Path params:

- idCorreo: entero

- Ejemplo curl:

curl -X DELETE "http://localhost:8083/correos/borrar/42"

Notas y recomendaciones

- CORS: como el frontend carga desde el mismo servidor (recursos estáticos de Spring), las rutas relativas funcionan sin CORS. Si se llama desde otro origen, activa y configura CORS en el backend o usa un proxy.

- Convenciones REST: actualmente se usan GET para crear recursos en algunos casos. Si más adelante quieres normalizar, podrías exponer variantes POST con cuerpo JSON.

- Formatos de respuesta: el frontend consume texto/HTML. Si necesitas uso programático, añade endpoints que devuelvan JSON o negocien Content-Type.

Si quieres, puedo generar una versión en Markdown o HTML para compartir con tu equipo, o añadir ejemplos en JavaScript (fetch) y en Postman/Insomnia.

Implementar CORS en Spring

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.RestController;

@CrossOrigin(origins = {

"https://frontend1.ejemplo.com",

"https://frontend2.ejemplo.com"

})

@RestController

public class ClientesController {

// ... tus mappings /clientes/...

}

Prompt 03

Anteriormente te había pedido la documentación del frontend: /home/carlos/Documentos/Carlos2025/AppClienteCorreo/src/main/resources/ En notas y recomendaciones me decís que en algunos casos uso get, parece que hay casos que están bien y otros casos hay que mejorar el uso de get. Me documentas los casos bien realizados y una estrategia para convertir los casos donde uso get de forma incorrecta a una forma correcta.

Suggest new conversation rejected for message

Voy a documentar los casos bien realizados y proporcionar una estrategia para mejorar los que usan GET incorrectamente. Aquí tienes el resumen:

### Casos Bien Realizados

1. \*\*Lectura de datos (GET)\*\*

- \*\*Descripción:\*\* Usar GET es apropiado cuando solo deseas recuperar datos sin alterar ningún estado en el servidor.

- \*\*Ejemplos de endpoints:\*\*

- `/clientes/listartodos`: Lista todos los clientes.

- `/clientes/buscarpordni?dni=...`: Busca un cliente por DNI.

- `/correos/listartodos`: Lista todos los correos.

2. \*\*Borrado de datos (DELETE)\*\*

- \*\*Descripción:\*\* Usar DELETE es correcto cuando deseas eliminar un recurso específico.

- \*\*Ejemplos de endpoints:\*\*

- `/clientes/borrar/{dni}`: Elimina un cliente por DNI.

- `/correos/borrar/{idCorreo}`: Elimina un correo por ID.

### Estrategia para Mejorar Casos de Uso de GET Incorrectos

3. \*\*Creación de datos (de GET a POST)\*\*

- \*\*Descripción:\*\* Cambia los endpoints a POST para crear nuevos recursos. POST es el método adecuado para envíos de datos cuando se modifican estados o se crean nuevos recursos en el servidor.

- \*\*Ejemplos de mejora:\*\*

- Cambiar `/clientes/guardar/{dni}/{nombre}/{apellido}` de GET a POST.

- Cambiar `/correos/guardar/{dni}/{correo}` de GET a POST.

- \*\*Estrategia:\*\*

- \*\*Actualizar el frontend:\*\* Cambia las llamadas fetch o formularios para usar POST.

- \*\*Modificar el backend:\*\* Ajusta el controlador para recibir el cuerpo de datos como JSON o parámetros del formulario.

- \*\*Validación adicional:\*\* Asegúrate de validar los datos del cuerpo antes de procesarlos.

4. \*\*Actualización de datos (de GET a PUT o PATCH)\*\*

- \*\*Descripción:\*\* Para actualizar recursos, utiliza PUT (para reemplazo completo) o PATCH (para actualización parcial).

- \*\*Ejemplos de mejora:\*\*

- Cambiar `/clientes/actualizar/{dni}/{nombre}/{apellido}` a PUT/PATCH.

- Cambiar `/correos/actualizar/{idCorreo}/{correo}` a PUT/PATCH.

- \*\*Estrategia:\*\*

- \*\*Actualizar el frontend:\*\* Cambia las llamadas pertinentes para usar PUT/PATCH.

- \*\*Modificar el backend:\*\* Ajusta los métodos en los controladores para manejar PUT/PATCH. Parsear el cuerpo de la solicitud para obtener los datos a actualizar.

- \*\*Validación y concurrencia:\*\* Implementa validaciones y gestión de concurrencia para garantizar la integridad de los datos.

### Beneficios de la Mejora

- \*\*Cumplimiento RESTful:\*\* Usar verbos HTTP apropiados mejora la capacidad de interoperación y entendimiento entre servicios.

- \*\*Seguridad y semántica:\*\* Al usar métodos correctos, el backend puede aplicar políticas de caché, autenticación más precisas y auditar operaciones correctamente.

- \*\*Desempeño:\*\* Un mejor alineamiento REST puede facilitar el caching y optimizar la comunicación entre cliente y servidor.

Si deseas que proceda a realizar cambios dentro de tu código, házmelo saber y te ayudaré con su implementación.