«Talento Tech»

Back-End

Java

Clase 14

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase 14: Integración del Backend con el Frontend

Índice

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Objetivos de la Clase

- Entender qué es CORS y por qué se necesita configurarlo para que el frontend (que corre en otro dominio) acceda a la API.
- Aprender a consumir la API desde el frontend utilizando `fetch()` u otros métodos, procesando las respuestas JSON.
- Integrar el carrito de compras del frontend con la API, mostrando productos reales y permitiendo operaciones (por ejemplo, crear productos, actualizar stock).
- Realizar pruebas integrales: verificar que el frontend muestre datos de la base, que se puedan agregar productos al carrito y que la API refleje esos cambios.
- Resolver problemas comunes de integración (errores, formateo de datos) y optimizar la API, considerando el rendimiento y la usabilidad.

Bienvenida a TechLab

Talento Lab

¡Les damos nuevamente la bienvenida a **TechLab**! Hemos recorrido un largo camino: desde los fundamentos de Java, POO, persistencia en MySQL y creación de APIs REST con Spring Boot. Ahora llega el momento de conectar ese backend con el frontend en HTML, CSS y JS que los alumnos ya tenían. La meta es que el frontend no dependa de datos estáticos, sino que obtenga productos reales desde la base de datos a través de la API, y que las acciones del carrito (agregar, quitar ítems) se reflejen en el backend.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Situación Inicial en TechLab

Silvia observa que el frontend muestra un carrito “falso”, cargado con datos estáticos o inexistentes. Ahora, Matías y Sabrina conectarán el frontend con la API Java, consumiendo endpoints como `GET /productos` para obtener el catálogo real. Una vez configurado CORS y ajustados los llamados desde JavaScript, el frontend podrá mostrar productos reales, agregar ítems al carrito y reflejar cambios persistentes en el backend.

Configuración de CORS para Permitir la Comunicación con el Frontend

Entendiendo el Problema CORS

Si el frontend (HTML, CSS, JS) corre en `http://localhost:3000` y la API en `http://localhost:8080`, el navegador, por seguridad, bloquea las solicitudes AJAX entre diferentes orígenes. CORS (Cross-Origin Resource Sharing) es un mecanismo que permite a la API indicar qué orígenes externos pueden acceder a sus recursos.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Soluciones con Spring Boot: @CrossOrigin y Configuración Global

En el controlador:

```
@RestController
@RequestMapping("/productos")
@CrossOrigin(origins = "http://localhost:3000")
public class ProductoController {
    // ...
}
```

`@CrossOrigin` permite solicitudes desde `http://localhost:3000`. Alternativamente, se puede configurar globalmente con un `WebMvcConfigurer` para habilitar CORS en todo el proyecto.

Consumo de la API desde el Frontend Existente

Fetch API, Axios u Otros Métodos de Consumo en JavaScript

En el frontend JS, podemos usar `fetch()`:

```
fetch("http://localhost:8080/productos")
 .then(response => response.json())
 .then(productos => {
   // Actualizar el DOM con la lista de productos
   mostrarProductosEnLaPantalla(productos);
 })
 .catch(error => console.error(error));
```

La respuesta vendrá en JSON, que podemos procesar para mostrar los productos en tarjetas, tablas, o como el frontend requiera.

Mostrando la Lista de Productos en la Web

Si el frontend ya tenía un contenedor para productos, ahora simplemente:

- Obtener el JSON con `fetch`.
- Generar el HTML dinámicamente.
- Incluir botones "Agregar al Carrito" que llamen a la API con `POST` a, por ejemplo, `POST /carrito` (si hubiésemos implementado un endpoint del carrito).

Integración del Carrito de Compras con la API

Si el carrito antes se manejaba localmente, ahora podemos:

- Consultar el stock antes de agregar un producto (`GET /productos/{id}`).
- Agregar productos al carrito si la API lo soporta. Si no, mantenemos el carrito en frontend, pero al finalizar la compra (futuro), enviamos una solicitud a la API para registrar el pedido.

Pruebas Integrales del Sistema Completo

Verificación de Flujos: Visualización, Agregado y Eliminación de Productos

- Abrir el frontend en `http://localhost:3000` (por ejemplo).
- Verificar que la lista de productos se cargue desde la API (`GET /productos`).
- Agregar un producto, ver que se refleja en el carrito (si el carrito es local, de momento no cambia en la base. Si implementamos endpoints para el carrito en el backend, se podrían enviar `POST /carrito`).

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Depuración de Problemas Comunes (Logs del Servidor, Herramientas del Navegador)

- Revisar la consola del navegador si hay errores CORS.
- Revisar logs en la consola de Spring Boot (`spring.jpa.show-sql=true` muestra queries, errores 500 se ven en logs).
- Ajustar `@CrossOrigin` o configuración global si el navegador bloquea las solicitudes.

Resolución de Problemas Comunes en la Integración

Errores 404, 500 y Mensajes Claros al Frontend

Si un producto no existe, el backend puede retornar 404. El frontend debe manejarlo mostrando un mensaje al usuario ("Producto no encontrado"). Si hay errores internos (500), registrar el error en logs y mostrar un mensaje genérico al usuario en el frontend.

Ajustando Formatos JSON, Nombres de Campos y Validaciones

Asegurar que los nombres de campos en la respuesta JSON coincidan con lo esperado por el frontend. Si el frontend espera `price` y la API envía `precio`, podríamos renombrar el campo o usar anotaciones como `@JsonProperty("price")`.

Tiempo de Respuesta y Optimización Inicial

Si la API se vuelve lenta, considerar paginación (`GET /productos?page=1&size=10`) o caching en el futuro. Por ahora, un catálogo pequeño no debería ser problema.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Optimización y Mejora en la API

Paginación, Filtros y Ordenamientos

En un futuro, podemos agregar parámetros de consulta para filtrar productos por categoría, ordenar por precio, o paginar resultados. Esto mejora la experiencia del frontend y la escalabilidad.

Manejo de Caché y Minificación de Recursos

Más adelante, si la API o el frontend requieren optimización, podemos agregar caches HTTP, ETAGs, o integrar CDNs. De momento, la prioridad es la integración básica.

Requerimiento en TechLab

El cliente Sibelius, reconocido por su expansión digital, está a punto de lanzar la versión final de su e-commerce. El frontend—desarrollado con HTML, CSS y JavaScript—está casi listo, pero al momento de conectarlo con la API del backend, han surgido errores inesperados: los navegadores bloquean las solicitudes al dominio del backend, y los desarrolladores no ven claramente cómo manejar esos códigos de error cuando un producto no existe en la base de datos.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Silvia, la Product Owner, explica que la experiencia del usuario es primordial. Necesita que el frontend muestre datos reales (productos con precio, stock y categoría), y que maneje cualquier error de forma amigable (“Producto no encontrado”, “Error de servidor”, etc.). Además, insiste en que si se olvida configurar CORS, el frontend falla silenciosamente. Este contratiempo se vuelve crítico justo cuando Sibelius se dispone a lanzar un catálogo especial para su nueva temporada de tés y cafés premium.

Para lograr la integración final entre frontend y backend, Matías y Sabrina, junto a vos, deben:

Ejercicios prácticos

1. Probar la configuración CORS en el backend (`@CrossOrigin`) y observar su impacto en las solicitudes desde el frontend.
2. Mostrar el catálogo real en la interfaz web, refrescando la lista de productos con un botón.
3. Manejar solicitudes a productos inexistentes (por ej., `GET /productos/9999`), mostrando un mensaje de error amigable y evitando que la aplicación muestre pantallas en blanco o datos inexistentes.

Con este paso, TechLab confía en que el ecosistema backend-frontend finalmente se integrará de forma robusta, satisfaciendo las necesidades de Sibelius y ofreciendo a sus clientes una experiencia estable y ágil al navegar por el catálogo gourmet. ¡Adelante con las pruebas y la configuración!

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Materiales y Recursos Adicionales

- Documentación de Spring sobre CORS:
- Ejemplos de `fetch()` en MDN: https://developer.mozilla.org/
- Videos recomendados en YouTube sobre integración de APIs REST con frontends HTML/JS.

Preguntas para Reflexionar

- ¿Cómo el manejo adecuado de CORS habilita la integración entre frontend y backend?
- ¿Qué ventajas ofrece tener una API REST genérica que el frontend consume en lugar de tener datos incrustados en el HTML estáticamente?

Cierre del Curso

¡Felicitaciones! Si llegaste hasta aquí, has recorrido un camino que abarcó desde los fundamentos de Java y POO, el manejo de excepciones, la organización del código, la creación de APIs REST con Spring Boot, la persistencia en MySQL, hasta la integración con el frontend y la resolución de problemas de comunicación. Ahora TechLab cuenta con un backend robusto y un frontend conectado a datos reales, formando una base sólida para un e-commerce funcional y listo para expandirse con nuevas funcionalidades.

Este curso ha sentado los cimientos de un stack completo y profesional. A partir de aquí, el camino está abierto para explorar temas más avanzados: seguridad con Spring Security, pruebas automatizadas, despliegue en la nube, microservicios, y mucho más. El mundo del desarrollo es amplio y desafiante, pero con las herramientas y conceptos adquiridos, estás preparado para afrontar nuevos retos.

Buenos Aires Ciudad