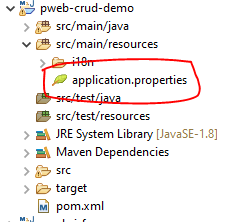
**Algunas consideraciones para demo de la integración Frontend y Backend**

**Frontend**

1. Antes de levantar las aplicaciones debe quedar configurado en el frontend la url del backend. Esto se define en el fichero “applicaction.properties” que está ubicado en “src/main/resources”.



1. Teniendo en cuenta la variante empleada para levantar el combo será la configuración que se ponga en este fichero.

|  |  |
| --- | --- |
| **Variante 1 (Usando tomcat embebido por el puerto 8085)** | **Variante 2 (Tomcat tradicional por el puerto 8080)** |
|  | **Nota**: En este caso se pone el nombre de la aplicación: pweb-backend. Este nombre se define en el pom.xml |

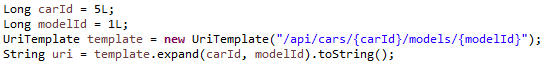
**Nota**: Recordar que cuando se usa la variante 2, que es un mismo tomcat9 para ambos proyectos es necesario agregar el nombre del proyecto. Si es la variante 1, que es usando el tomcat9 embebido que trae Spring Boot esto no es necesario.

Para realizar la lectura de este fichero y estas propiedades se implementó la clase “ApplicationProperties” que se encarga de esta gestión. Pueden ver en la implementación del RestService la manera en que se utiliza.

1. Siempre que se necesite formar Url para llamar un endpoint en el Backend, **NO CONCATENAR,** utilizar para eso la clase “UriTemplate” de Spring, perteneciente a la dependencia Spring-Web. A continuación, dos ejemplos:
   1. “/api/cars/{carId}” => /api/cars/5



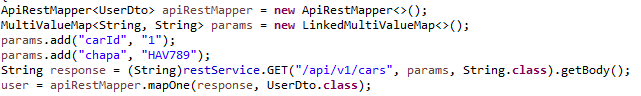
* 1. “/api/cars/{carId}/models/{modelId}” => /api/cars/5/models/1



1. Cuando la url tiene parámetros, porque en el RestController se utilizó @RequestParam, entonces pasar los parámetros utilizando:

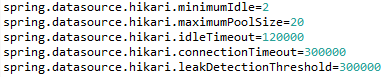
MultiValueMap<String, String> params = **new** LinkedMultiValueMap<>();

Por tanto, si tienen un endpoint con el siguiente formato: “/api/v1/cars?carId=1&chapa=HAV789” el Get lo deben hacer de la siguiente manera.

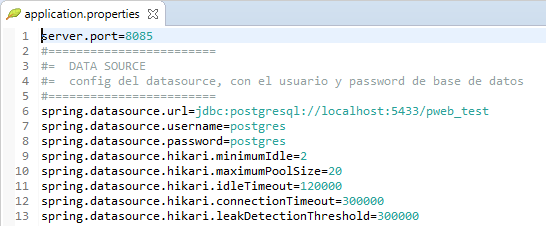


**Backend**

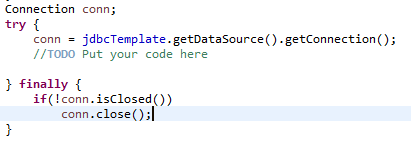
1. Agregar algunas configuraciones para evitar que se llene el pool de conexiones del gestor de base de datos. Para esto deben asegurarse de cerrar la conexiones de la base de datos y agregar la siguiente configuración en el fichero “application.properties” del backend.



De esta forma el fichero completo quedaría de la siguiente manera:



1. Para garantizar que las conexiones se cierren automáticamente basta con utilizar el principio “try-with-resources” que permite declarar recursos para ser usados ​​en un bloque try con la tranquilidad de que los recursos se cerrarán después de la ejecución de ese bloque. Antes se empleaba “try-with-finally” y este era el code:

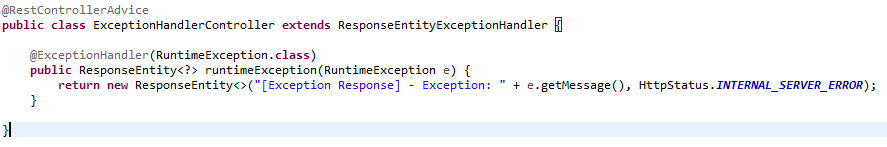


Ahora usando “try-with-resource”, disponible desde Java 7, el código quedaría de la siguiente manera:



Como se aprecia no es necesario poner el “Finally” ni cerrar la conexión.

1. Agregar un manejador de errores. Estas clases son muy útiles porque de manera transversal se encargan capturar todo tipo de errores y todo se hace en un mismo lugar. Spring para resolver este problema define una anotación llamada @RestControllerAdvice heredando de la clase ResponseEntityExceptionHandler.



De esta manera si ocurre un error en alguno de los enpoints se captura y además permite personalizar los mensajes dependiendo el tipo de error (Exception Type).