DAM
Desarrollo de Aplicaciones Multiplataforma
2º Curso

AD
Acceso a Datos

UD 5
Programación de componentes
de acceso a datos
Parte 4

IES BALMIS
Dpto Informática
Curso 2022-2023
Versión 2 (12/2022)

12. EJERCICIOS – API Rest

UD5Ejer1201 – ApiRestMathAmpliado (2)

Copiar el proyecto **ApiRestMath** creado en los apuntes como **ApiRestMathAmpliado** con el context root **/xmath** y añadir dos nuevos métodos:

- **Restar**: también recibirá dos parámetros y devolverá el resultado de restar los dos parámetros recibidos.
  Ejemplo: 20 - 12
  - http://localhost:8080/xmath/apirest/aritmetica/restar/20/12
- **Multiplicar**: también recibirá dos parámetros y devolverá el resultado de multiplicar los dos parámetros recibidos.
  Ejemplo: 7 * 5
  - http://localhost:8080/xmath/apirest/aritmetica/multiplicar/5/7
- **Porcentaje**: también recibirá dos parámetros con el opcional de decimales (por defecto 0) y devolverá el resultado de calcular el porcentaje de los del primer parámetro indican en el segundo.
  Ejemplo: calcular el 9% de 55 con 3 decimales
  - *http://localhost:8080/xmath/apirest/aritmetica/porcentaje/55/9?decimales=3*
  - http://localhost:8080/xmath/apirest/aritmetica/porcentaje/55/9

UD5Ejer1202 – ApiRestBiblioteca1Ampliado (1)

Ampliar proyecto **ApiRestBiblioteca** creado en los apuntes como **ApiRestBiblioteca1Ampliado** con el context root **/xapirest1** para que disponga además del método:

- **DELETE /libros/{id}**

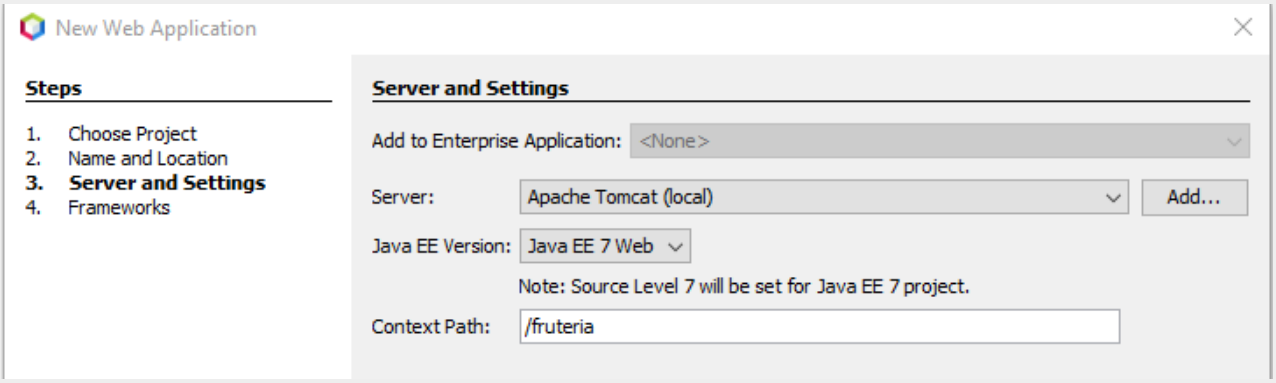
Se deberá crear el método de acceso a datos en **DAOLibros** y atender en servicio **DELETE** en **ServiceRestLibros.**

Realiza pruebas con Postman para comprobar el correcto funcionamiento.

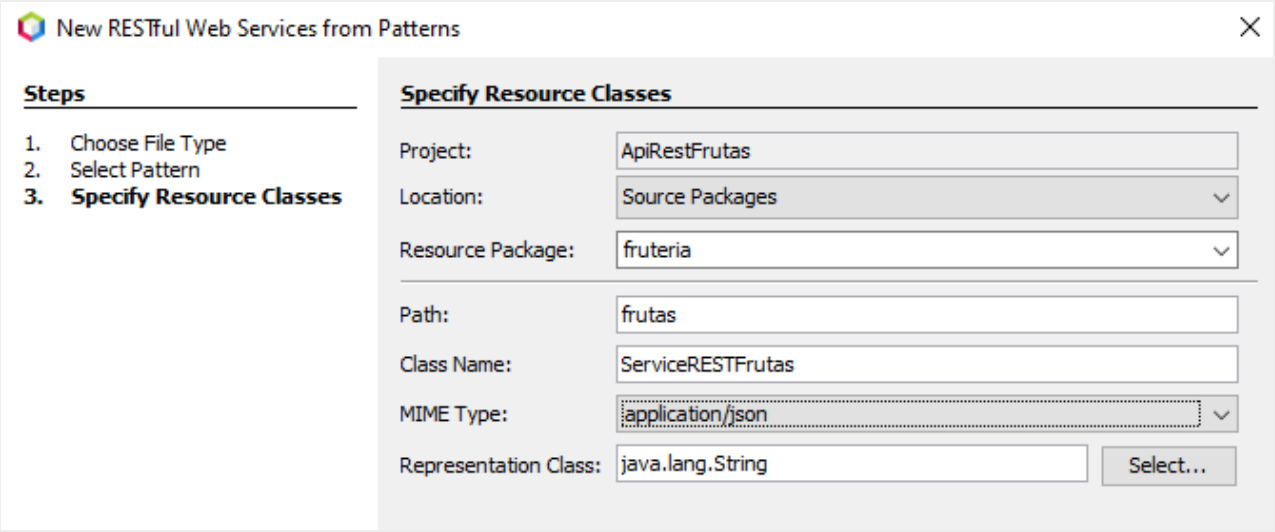
UD5Ejer1203 – ApiRestFrutas (1)

Crear el proyecto **ApiRestFrutas** de tipo **Java Web** para **Apache Tomcat** con la URL es:

http://localhost:8080/fruteria/recursos/frutas



Y le añadimos el **ServiceRESTFrutas** con el asistente **RESTful Web Services from Patterns** de tipo **Simple Root Resource**:



Para completar el path deberemos:

- Editar **ApplicationConfig** y cambiar **webresources** por **recursos**

Al proyecto le crearemos (o copiaremos de lo entregado por el profesor):

- copiar **index.html** para la información del API
- quitaremos las librerías que por defecto nos ha añadido NetBeans y añadiremos nuestro **lib** con las librerías de **jakarta-ws.rs-3.1.0** que ya hemos usado anteriormente
- copiar la carpeta **clases**, que contiene las clases Frutas
- copiar la clase **DAOFrutas.java** en el java package **fruteria**
- añadiremos a **ServiceRestFrutas** el código para atender los métodos que están en **GET (todos y uno), POST, PUT y DELETE.**

Si observas el código verás que **DAOFrutas** es una **clase de datos no persistente**, pues al iniciarse se carga con 3 registros en un ArrayList en RAM y al parar el servidor web o parar la App se perderán todos los datos almacenados.

Prueba el proyecto con **Postman** haciendo GET de todas, GET de una, POST, PUT y DELETE.

UD5Ejer1204 – ApiRestFrutasPvp (1)

Copia el proyecto **ApiRestFrutas** anterior como **ApiRestFrutasPvp** y configúralo para que funcione con la URL es:

http://localhost:8080/tienda/fruteria/producto

Deberás cambiar:

1. **Context Root** (**tienda** en context.xml),
2. **Application Path** (**fruteria** en ApplicationConfig.java)
3. El **Path** del Service REST (**producto** en ServiceRESTFrutas.java)

Edita el **index.html** y cambia la información del API con la nueva URL.

Prueba su correcto funcionamiento.

Una vez funcione la nueva URL se desea:

1. Añadir a la clase **Frutas** el campo **precio** de tipo **double**.
   - Al añadirlo deberás eliminar todos sus métodos y añadirlos de nuevo todos con **Insert Code**, incluyendo también el **equals** y **hashCode**, dos métodos que pueden ser usados por **REST** o por **DAO**.
   - Además, deberás añadir un valor del **precio** a las 3 frutas que se añaden al iniciar en **DAOFrutas**.
   - Modificar el **put** de **DAOFrutas** para que actualice también el precio si ha cambiado:

```
if (fruta.getPrecio() != fruit.getPrecio()) {
    fruta.setPrecio(fruit.getPrecio());
    resul = true;
}
```

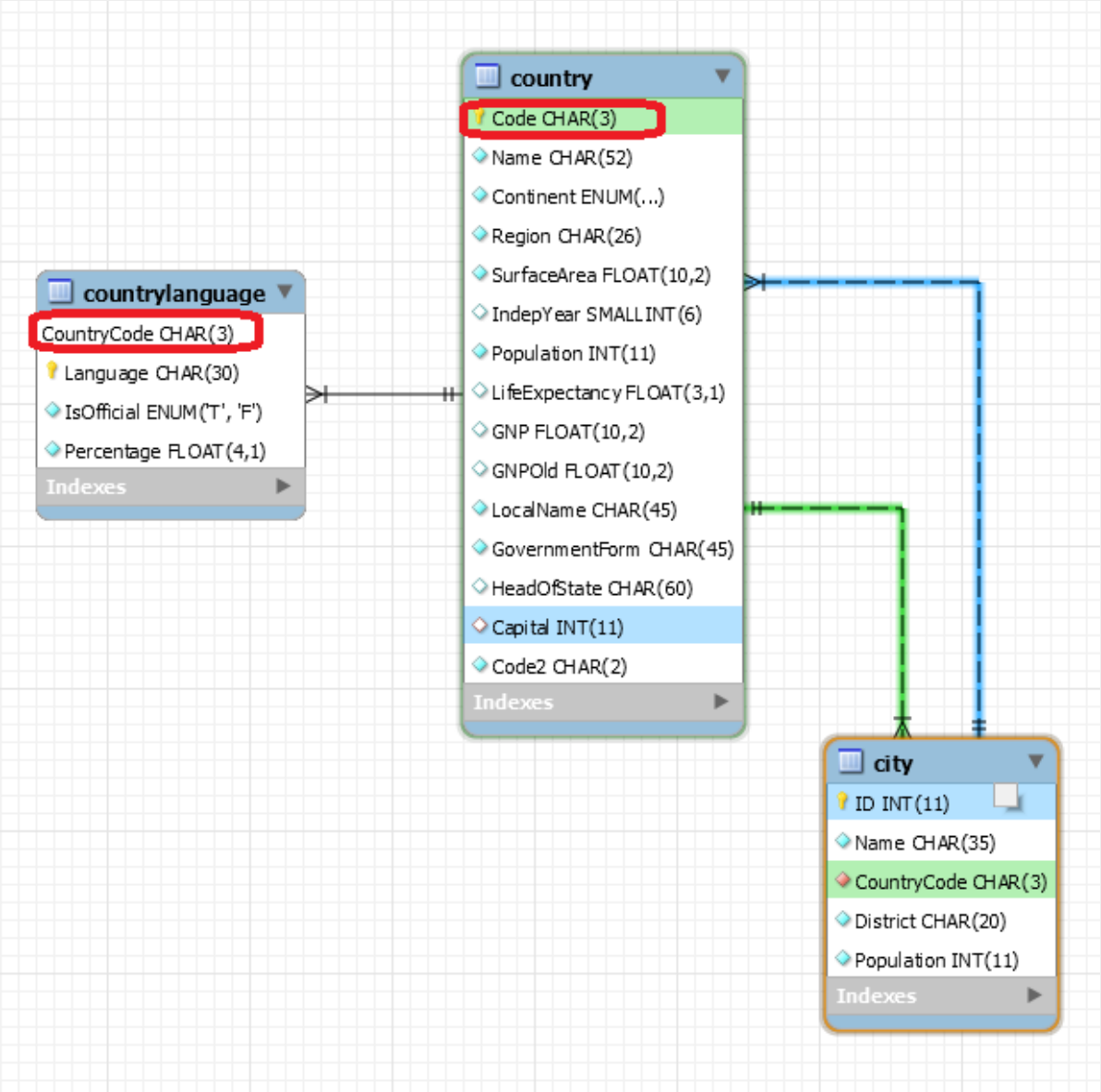
2. Añadir la funcionalidad también con XML para lo que deberemos:
   - Añadir las clases **ListaFrutas** y **Mensaje**
   - Realizar en las clases de datos las anotaciones de XML para realizar el mapeo de **JAXB**.
   - Añadir el **MediaType.APPLICATION_XML** en los **Produces** y **Consumes** métodos de **ServiceRESTFrutas**.
   - Sustituir el código de **HashMap** por el de la clase **Mensaje**
   - Modificar el método **getAll** para que devuelva la lista o el objeto según sea **JSON** o **XML** lo solicitado:

Se puede probar cada paso según se complete.

UD5Ejer1205 – ApiRestWorld (2)

Tenemos una BD en MySQL denominada **worldnew**. El profesor entregará el script para crearla.

El esquema relacional es:



Un país (**country**) tiene muchas ciudades (**city**), pero además tiene una capital (**city**).
En un país (**country**) se hablan muchas lenguas (**contrylanguage**).

Crea el proyecto **Java Web** para el servidor Web **Apache Tomcat** denominado **ApiRestWorld** que ofrecerá un servicio de API Rest en el formato **JSON** para acceder a la base de datos **worldnew** de **MySQL.**

La url será:

http://localhost:8080/apidata/world/XXXX

El proyecto no trabaja con XML por lo que **no son necesarias las anotaciones con JAXB**.

Será necesario añadir el **driver de MySQL** y las librerías de **jakarta-ws.rs-3.1.0**.

Se deberán crear solo las clases **City** y **Country** para mapear el resultado en Java.

Los métodos a implementar son:

- **GET => /ciudad/{id}**
  - Devuelve los datos de la ciudad cuyo id es el indicado
  - Por elemplo:
    - http://localhost:8080/apidata/world/ciudad/3
  - Los campos a devolver en JSON son:
    - int **id**;
    - String **name**;
    - String **district**;
    - int **population**;
    - String **countrycode**;
- **GET => /pais/{code}**
  - Devuelve los datos del país cuyo **code** es el indicado
  - Por ejemplo:
    - http://localhost:8080/apidata/world/pais/FIN
  - Los campos a devolver en JSON son:
    - String **code**;
    - String **name**;
    - String **continent**;
    - Double **surfacearea**;
    - City **capital**;
    - ArrayList<City> **ciudades**;

Lo mejor es hacer un **ServiceREST** para cada tabla, que se añadirán en el método **addRestResourceClasses** de **ApplicationConfig**.

El **DAOWorld** puede contener los métodos para las dos tablas. Podéis consultar el DAOLibros de ejemplo de los apuntes.

UD5Ejer1301 – ApiRestWorldJPA (2)

Realizar la misma aplicación anterior pero utilizando JPA

Crea el proyecto **Java Web** para el servidor Web **Apache Tomcat** denominado **ApiRestWorldJPA** que ofrecerá un servicio de API Rest en el formato **JSON** para acceder a la base de datos **worldnew** de **MySQL.**

La url será:
http://localhost:8080/apijpa/world/XXXX

Se desea:
- Al mostrar una ciudad, no se mostrarán los datos del país, solo se verá su código (utiliza **@JsonbTransient**)
- Al mostrar un país se mostrarán todos los datos de la capital y los de todas sus ciudades

UD5Ejer1302 – ApiRestWorldJPQL (2)

Haz una copia del proyecto **ApiRestWorldJPA** como **ApiRestWorldJPQL**.

La nueva url será:
http://localhost:8080/apijpql/world/XXXX

En este nuevo proyecto añadiremos un nuevo API que muestre una lista de objetos con los datos:
- String **code**; → código del país
- Long **numciudades**; → número de ciudades del país en la BD

Los métodos a implementar son:
- **GET => /pais/total/{id}**
  - Devuelve los datos del código de país y el número de ciudades
  - Por ejemplo:
    - http://localhost:8080/apijpql/world/pais/total/FIN
- **GET => /pais/total**
  - Devuelve de todos los países los datos del código de país y el número de ciudades
  - Por ejemplo:
    - http://localhost:8080/apijpql/world/pais/total

Para obtener estos datos usaremos JPQL.

Para **GET => /pais/total/{id}** se deberá crear un **JsonObject** con
```
String countrycode;
Long numciudades;
```
La consulta JPQL a realizar sobre la clase City podría ser:
```
SELECT ct.country.code, COUNT(ct.id) as numciudades
FROM City ct
WHERE ct.country.code='"+code+"' GROUP BY ct.country.code
```

Para **GET => /pais/total** se deberá crear **JsonArray** con la consulta JPQL
```
SELECT ct.country.code, COUNT(ct.id) as numciudades
FROM City ct
GROUP BY ct.country.code
```