GRPC SERVER...............................................................................................................



-------

cartella "proto" dentro cartella "main" vicino "java": FILE.proto

syntax = "proto3";

option java\_multiple\_files = true;

package it.uniroma1.gRPCExample; //package diverso da quello con i codici

message ListRequest { //cambia nome messaggio

string id = 1; //ordine obbligatorio

}

message ListResponse { //cambia nome messaggio

string list = 1; //ordine obbligatorio

}

//uno o piu servizi con una o piu RPC dentro

service ListService { //cambia nome servizio (ad es col nome del proto)

rpc listOrInfo(ListRequest) returns (ListResponse); //cambia nome rpc e messaggi

}

-------

-------

ListServiceImpl.java //nomeservizioImpl.java

import io.grpc.stub.StreamObserver;

import it.uniroma1.gRPCExample.ListRequest; //occhio, classi autogenerate nel package indicato nel proto, cambia

import it.uniroma1.gRPCExample.ListResponse; //occhio, classi autogenerate nel package indicato nel proto, cambia

import it.uniroma1.gRPCExample.ListServiceGrpc.ListServiceImplBase; //occhio, classi autogenerate nel package indicato nel proto, cambia

public class ListServiceImpl extends ListServiceImplBase { //nomeservizioImpl extends nomeservizioImplBase

@Override

public void listOrInfo(ListRequest request, StreamObserver<ListResponse> responseObserver) { //nome dell rpc, cambia roba che entra

String id = request.getId(); //se vuoi salvati campi in ingresso in delle variabili, getNomeCampo()

String result = ""; //stringa da mandare come risposta parte vuota, oppure fai come stringa greeting in "HelloServiceImpl.java"

System.out.println("... the server has received id: " + id ); //facoltativo mostrarlo

//qui manca codice che costruisce stringa result, fai result += "roba" per aggiungere pezzi alla stringa

ListResponse response = ListResponse.newBuilder() //crea response, cambia

.setList(result) //setNomefieldrisposta(stringa di prima), occhio che ha maiuscola anche se nel proto il field era in minuscolo

.build();

responseObserver.onNext(response);

responseObserver.onCompleted();

}

}

-------

MyServer.java

package it.uniroma1.grpcexampleserver; //occhio

import io.grpc.Server;

import io.grpc.ServerBuilder;

import java.io.IOException;

public class MyServer {

public static void main(String[] args) throws IOException, InterruptedException {

Server server;

server = ServerBuilder

.forPort(8080) //cambia porta volendo

.addService(new ListServiceImpl()).build(); //ci va il nome della classe nomeservizioImpl

server.start();

server.awaitTermination();

}

}

------

GRPC CLIENT...............................................................................................................



------

file.proto uguale a quello del server

------

------

Client.java

package it.uniroma1.grpcexamplesimpleclient; //occhio

import io.grpc.ManagedChannel;

import io.grpc.ManagedChannelBuilder;

import it.uniroma1.gRPCExample.ListRequest; //occhio, classi autogenerate nel package indicato nel proto, cambia

import it.uniroma1.gRPCExample.ListResponse; //occhio, classi autogenerate nel package indicato nel proto, cambia

import it.uniroma1.gRPCExample.ListServiceGrpc; //occhio, classi autogenerate nel package indicato nel proto, cambia

public class Client {

public static void main(String[] args) throws InterruptedException {

ManagedChannel channel = ManagedChannelBuilder.forAddress("localhost", 8080) //occhio

.usePlaintext()

.build();

ListServiceGrpc.ListServiceBlockingStub stub

= ListServiceGrpc.newBlockingStub(channel);

ListResponse listResponse = stub.listOrInfo(ListRequest.newBuilder() //creazione e INVIO della richiesta grpc e metti il risultato in response

.setId("-1") //set dei field, cambia

.build());

System.out.println("Response received from server:\n" + listResponse.getList()); //stampa risposta, cambia

channel.shutdown();

}

}

------

JMS PRODUCER...............................................................................................................



qui metto quello di july in cui prende info da DB, se invece prendi jms stock market servant le prende da lista di stringhe scriptata

-------

ProduttoreQuotazioni.java //cambia

package it.uniroma1.jmspublisher; //occhio

//per jms

import javax.jms.\*;

import javax.naming.\*;

import java.util.Properties;

import java.util.Random;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

//per DB

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.logging.Level;

public class ProduttoreQuotazioni { //cambia

private List<String> titoli = new ArrayList<>(); //occhio se serve

private static final String DATABASE\_POSITION = "/home/biar/database"; //cambia

public ProduttoreQuotazioni() { //costruttore,cambia, lo usa per riempire array di titoli con roba presa dal DB, e' l unico punto in cui si usa DB

java.sql.Connection conn;

try {

Class.forName("org.sqlite.JDBC");

conn = DriverManager.getConnection("jdbc:sqlite:" + DATABASE\_POSITION);

PreparedStatement ps = conn.prepareStatement("select \* from stocks;"); //occhio, query

ResultSet rs = ps.executeQuery();

while(rs.next()) {

titoli.add(rs.getString("name")); //occhio a cosa prendi

}

rs.close();

} catch (SQLException ex) {

java.util.logging.Logger.getLogger(ProduttoreQuotazioni.class.getName()).log(Level.SEVERE, null, ex); //cambia

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(ProduttoreQuotazioni.class.getName()).log(Level.SEVERE, null, ex); //cambia

}

}

private String scegliTitolo() { //funz che sceglie titolo random

int whichMsg;

Random randomGen = new Random();

whichMsg = randomGen.nextInt(this.titoli.size());

return this.titoli.get(whichMsg); //come fai il get dipende da come hai dichiarato array (vedi i due es)

}

private float valore() { //facoltativa, crea float random

Random randomGen = new Random();

float val = randomGen.nextFloat() \* this.titoli.size() \* 10;

return val;

}

private static final Logger LOG = LoggerFactory.getLogger(ProduttoreQuotazioni.class); //facoltativo

public void start() throws NamingException, JMSException {

Context jndiContext = null;

ConnectionFactory connectionFactory = null;

javax.jms.Connection connection = null;

Session session = null;

Destination destination = null;

MessageProducer producer = null;

String destinationName = "dynamicTopics/students"; //cambia

try { //connessione al provider

Properties props = new Properties();

props.setProperty(Context.INITIAL\_CONTEXT\_FACTORY, "org.apache.activemq.jndi.ActiveMQInitialContextFactory");

props.setProperty(Context.PROVIDER\_URL, "tcp://192.168.50.83:61616"); //cambia

jndiContext = new InitialContext(props);

} catch (NamingException e) {

LOG.info("ERROR in JNDI: " + e.toString());

System.exit(1);

}

try { //connessione al topic?

connectionFactory = (ConnectionFactory) jndiContext.lookup("ConnectionFactory");

destination = (Destination) jndiContext.lookup(destinationName);

} catch (NamingException e) {

LOG.info("JNDI API lookup failed: " + e);

System.exit(1);

}

try {

connection = connectionFactory.createConnection();

session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

producer = session.createProducer(destination);

TextMessage message = null;

String stock = null; //cambia

message = session.createTextMessage();

float quotazione; //cambia

String myID = "1947850"; //cambia

int i = 0; //cambia volendo

while (true) {

i++; //cambia volendo

stock = scegliTitolo(); //cambia, nome della funz che sceglie random

quotazione = valore(); //cambia, nome della funz che crea float random

message.setStringProperty("student", myID); //cambia

message.setStringProperty("name", stock); //cambia

message.setFloatProperty("value", quotazione); //cambia

message.setText(

"Item " + i +" from student "+myID+": "+ stock + ", Valore: "

+ quotazione); //cambia

LOG.info(

this.getClass().getName()

+ "Invio quotazione -> " + message.getText()); //facoltativo

producer.send(message);

try { //un messaggio ogni 5 sec

Thread.sleep(5000);

} catch (Exception ex) {

ex.printStackTrace();

}

}

} catch (JMSException e) {

LOG.info("Exception occurred: " + e);

} finally {

if (connection != null) {

try {

connection.close();

} catch (JMSException e) {

}

}

}

}

}

------

------

ServerMain.java

package it.uniroma1.jmspublisher; //occhio

import javax.jms.JMSException;

import javax.naming.NamingException;

public class ServerMain {

public static void main(String[] args) throws NamingException, JMSException {

ProduttoreQuotazioni q = new ProduttoreQuotazioni(); //nome della classe producer

q.start();

}

}

------

JMS SUBSCRIBER...............................................................................................................



preso dall esame di july, quello che sta in jmsstockmarketclient e' piu tosto

------

JMSListener.java !!! senza connessione a provider perche in questo caso la mette nel ClientMain.java

package it.uniroma1.jmssubscriber; //occhio

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageListener;

public class JMSListener implements MessageListener {

private static JMSListener instance = null; //LISTENER SINGLETON

private static String myID = "1947850"; //serviva a lui

private JMSListener() { //costruttore vuoto, ci va se fai singleton

}

public static MessageListener getInstance() { //serve a prendere l'instance

if (instance == null) {

instance = new JMSListener();

}

return instance;

}

@Override

public void onMessage(Message msg) { //parte cruciale del listener di un subscriber, in questo caso faceva filtro su content

try {

String student = msg.getStringProperty("student"); //modo per prendere campi del msg

if(student.equals(myID)) { //FILTRO

String name = msg.getStringProperty("name");

float value = msg.getFloatProperty("value");

System.out.println("New stock update for student "+myID+" -> Name:"+name+" Value: "+value); //facoltativo

}

} catch (JMSException err) {

err.printStackTrace();

} catch (NullPointerException exc) {

//staysilent

}

}

}

-------

-------

ClientMain.java !!!con connessione a provider

package it.uniroma1.jmssubscriber; //occhio

import java.util.Properties;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageListener; //oppure javax.jms.\*

import javax.jms.MessageProducer;

import javax.jms.Session;

import javax.jms.Topic;

import javax.jms.TopicConnection;

import javax.jms.TopicSession;

import javax.jms.TopicSubscriber;

import javax.naming.Context;

import javax.naming.InitialContext;

import javax.naming.NamingException;

public class ClientMain {

private static TopicConnection topicConnection;

private static TopicSession topicSession = null;

private static Destination destination = null;

private static MessageProducer producer = null;

private static TopicSubscriber topicSubscriber = null;

public static void main(String[] args) {

System.out.println("Connecting to the JMS provider..."); //occhio

Context jndiContext = null;

ConnectionFactory topicConnectionFactory = null;

String destinationName = "dynamicTopics/students"; //cambia

try {

Properties props = new Properties();

props.setProperty(Context.INITIAL\_CONTEXT\_FACTORY, "org.apache.activemq.jndi.ActiveMQInitialContextFactory");

props.setProperty(Context.PROVIDER\_URL, "tcp://192.168.50.83:61616"); //cambia

jndiContext = new InitialContext(props);

topicConnectionFactory = (ConnectionFactory) jndiContext.lookup("ConnectionFactory");

destination = (Destination) jndiContext.lookup(destinationName);

topicConnection = (TopicConnection) topicConnectionFactory.createConnection();

topicSession = (TopicSession) topicConnection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

topicSubscriber = topicSession.createSubscriber((Topic) destination);

topicSubscriber.setMessageListener(JMSListener.getInstance()); //COLLEGHI IL LISTENER, se connessione a provider la facevi nel listener mettevi "this"

topicConnection.start(); //in azionijmslistener mette anche "stop" e gestisce eccezioni

System.out.println("Ready!\n--------------------------------------------------------------------------"); //occhio

} catch (JMSException err) {

err.printStackTrace();

} catch (NamingException err) {

err.printStackTrace();

}

}

}

------

SOAP SERVICE...............................................................................................................



preso da quello del prof, in cui usa xml adapt e Maps, vedi esame june in cui non servono xml annotations e usa arraylists,

gli adapter con marshal e unmarshal sono grevi, quindi se puoi usa solo roba base cone in june

quindi il core sono la WSInterface la WSImpl e il Server, poi ci vanno classi legate a quello che si trasferisce, se devi RETURNARE solo roba base ti basta una classe (es Book)

se invece devi RETURNARE roba complessa servono le altre classi

-------

WSInterface.java

package it.sapienza.softeng.soapws; //occhio

import javax.jws.\*; //unico import "obbligatorio"

import java.util.Map;

import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter; //xml only

@WebService

public interface WSInterface {

public String hello(String name); //cambia

public String helloStudent(Student student); //cambia

@XmlJavaTypeAdapter(StudentMapAdapter.class) //xml only, ci mette la classe adapter di quello che e' returnato (Map)

public Map<Integer, Student> getStudents(); //cambia

}

-------

-------

WSImpl.java

package it.sapienza.softeng.soapws; //occhio

import java.util.LinkedHashMap;

import java.util.Map;

import javax.jws.WebService; //unico import obbligatorio

@WebService(endpointInterface = "it.sapienza.softeng.soapws.WSInterface") //annotation obbligatoria, indica dove sta interface

public class WSImpl implements WSInterface {

private Map<Integer, Student> students = new LinkedHashMap<Integer, Student>(); //per map

public WSImpl() {} //costrutt vuoto

public String hello(String name) { //cambia

return "Hello " + name; //cambia

}

public String helloStudent(Student student) { //cambia

students.put(students.size() + 1, student); //cambia

return "Hello " + student.getName(); //cambia

}

public Map<Integer, Student> getStudents() { //cambia

return students; //cambia

}

}

-------

-------

Server.java

quasi totalmente scriptata

package it.sapienza.softeng.soapws;

import javax.xml.ws.Endpoint;

public class Server {

public static void main(String args[]) throws InterruptedException {

WSImpl implementor = new WSImpl();

String address = "http://0.0.0.0:8081/WSInterface"; //praticamente unica cosa che cambi

Endpoint.publish(address, implementor);

while(true) {}

//Thread.sleep(60 \* 1000);

//System.exit(0);

}

}

------

da qui in poi e' quasi tutta roba che sta solo nella versione del prof (di june manca solo classe Book)

-------

Student.java (di fatti sarebbe StudentInterface)

package it.sapienza.softeng.soapws; //occhio

import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter; //xml only

@XmlJavaTypeAdapter(StudentAdapter.class) //mette il suo adapter

public interface Student {

public String getName(); //qui mette solo get name non so perche'

}

-------

-------

StudentImpl.java

package it.sapienza.softeng.soapws; //occhio

import javax.xml.bind.annotation.XmlType; //non confonderla con XmlJavaTypeAdapter e XmlAdapter

@XmlType(name = "Student") //xml

public class StudentImpl implements Student {

private String name;

public String getName() { return name; }

public void setName(String n) { name = n; }

public StudentImpl(String n) { name = n; } //due costruttori

public StudentImpl() {}

}

-------

-------

StudentAdapter.java

marshal serve a passare da uno Student(interfaccia) a uno StudentImpl, unmarshal viceversa

package it.sapienza.softeng.soapws; //occhio

import javax.xml.bind.annotation.adapters.XmlAdapter; //non confonderla con XmlType e XmlJavaTypeAdapter

public class StudentAdapter extends XmlAdapter<StudentImpl,Student>{ //come campi di XmlAdapter ci vanno le due parti da convertire

public StudentImpl marshal(Student stud) throws Exception { //segue marshal tipico per passare da interfaccia a impl

if (stud instanceof StudentImpl)

return (StudentImpl) stud;

return new StudentImpl(stud.getName()); //occhio, cambia in base a come hai fatto il costrutt in StudentImpl

}

@Override

public Student unmarshal(StudentImpl v) throws Exception { //segue unmarshal tipico per passare da impl a interfaccia

return v;

}

public StudentAdapter(){}

}

------

StudentMap.java

possiamo vederla come impl, l' interfaccia non serve perche' l'elemento base e' la Map di util

package it.sapienza.softeng.soapws;

import java.util.ArrayList;

import java.util.List;

import javax.xml.bind.annotation.XmlElement; //qui usa anche questo per gli elementi dell xml type

import javax.xml.bind.annotation.XmlType;

@XmlType(name = "StudentMap")

public class StudentMap {

private List<StudentEntry> entries = new ArrayList<StudentEntry>();

public StudentMap() {}

@XmlElement(nillable = false, name = "entry")

public List<StudentEntry> getEntries() {

return entries;

}

@XmlType(name = "StudentEntry") //la singola entry e' a sua volta un xml type, sottoclasse

public static class StudentEntry {

private Integer id;

private Student student;

public Integer getId() { return id; }

public Student getStudent () { return student; }

public void setId(Integer i) { id = i; }

public void setStudent(Student s) { student = s; }

}

}

---------

--------

StudentMapAdapter.java

usata per passare da StudentMap a Map e viceversa

package it.sapienza.softeng.soapws;

import java.util.LinkedHashMap;

import java.util.Map;

import javax.xml.bind.annotation.adapters.XmlAdapter;

public class StudentMapAdapter extends XmlAdapter<StudentMap, Map<Integer,Student>>{

public StudentMapAdapter(){}

@Override

public Map<Integer,Student> unmarshal(StudentMap v) throws Exception {

Map<Integer,Student> boundMap = new LinkedHashMap<Integer, Student>();

for (StudentMap.StudentEntry studentEntry : v.getEntries()) {

boundMap.put(studentEntry.getId(), studentEntry.getStudent());

}

return boundMap;

}

@Override

public StudentMap marshal(Map<Integer, Student> boundMap) throws Exception {

StudentMap valueMap = new StudentMap();

for (Map.Entry<Integer, Student> boundEntry : boundMap.entrySet()) {

StudentMap.StudentEntry valueEntry = new StudentMap.StudentEntry();

valueEntry.setStudent(boundEntry.getValue());

valueEntry.setId(boundEntry.getKey());

valueMap.getEntries().add(valueEntry);

}

return valueMap;

}

}

---------

SOAP CLIENT...............................................................................................................



grazie al wsimport nel POM ha classi e funzioni del server autogenerate

-------

Client.java

usato solo per chiamare le funz offerte dal service

package it.uniroma1.soapclient; //occhio

import java.util.List;

import soapwsclient.generated.\*; //import chiave, forse cambia il package in base a dove hai messo roba autogenerata

public class Client { //tutto da cambiare in base a quello che devi fare

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

System.out.println("... adding new students ...");

Student s1 = new Student();

s1.setName("Andrea");

System.out.println(".. just added " + Client.helloStudent(s1));

Student s2 = new Student();

s2.setName("Carla");

System.out.println(".. just added " + Client.helloStudent(s2));

System.out.println("... and now recovering the whole StudentMap ...");

List<StudentEntry> result = Client.getStudents().getEntry();

for (int i = 0 ; i<result.size(); i++){

System.out.println(((StudentEntry)result.get(i)).getStudent().getName());

}

}

private static StudentMap getStudents() { // la vera implem sta nel service, il nome non deve per forza essere uguale

WSImplService service = new WSImplService(); //scriptato, nomeimplService()

WSInterface port = service.getWSImplPort(); //scriptato, getNomeImplPort()

return port.getStudents(); //qui chiami la vera funz col nome che avevi messo nel service

}

private static String helloStudent(Student arg0) { // la vera implem sta nel service, il nome non deve per forza essere uguale, arg0 non e' da command line

WSImplService service = new WSImplService(); // ...

WSInterface port = service.getWSImplPort(); // ...

return port.helloStudent(arg0); //qui chiami la vera funz col nome che avevi messo nel service

}

}

--------

REST SERVICE...............................................................................................................



apparte APIServer le altre classi cambiano contenuto a seconda di quello che devi fare

--------

APIServer.java

package it.sapienza.softeng.rest; //occhio

import org.apache.cxf.endpoint.Server;

import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;

import org.apache.cxf.jaxrs.lifecycle.SingletonResourceProvider;

public class APIServer {

public static void main(String args[]) throws Exception {

JAXRSServerFactoryBean factoryBean = new JAXRSServerFactoryBean();

factoryBean.setResourceClasses(CoursesRepository.class); //cambia classe

factoryBean.setResourceProvider(new SingletonResourceProvider(new CoursesRepository())); //cambia classe

factoryBean.setAddress("http://0.0.0.0:8080/"); //cambia

Server server = factoryBean.create();

System.out.println("Server ready...");

//Thread.sleep(60 \* 1000);

//System.out.println("Server exiting");

//server.destroy();

//System.exit(0);

while (true) {

}

}

}

-------

--------

CoursesRepository.java

usa alcune cose che stanno in classi dopo, falla contemporaneam alle altre

RICORDA: nella classe CoursesRepository hai tutti i metodi relativi ai corsi, nella classe Course hai i metodi relativi agli studenti ecc

anche se non servissero metodi su studenti, devi comunque fare classe Course perche la usi in CoursesRepository

package it.sapienza.softeng.rest; //occhio

import javax.ws.rs.\*;

import javax.ws.rs.core.Response;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@Path("/courses") //mettendo qui il path root e come se ce l avessi sempre

@Produces("text/xml")

public class CoursesRepository {

private Map<Integer, Course> courses = new HashMap<>(); //usa hash map per contenere roba

{ //cambia tutto, qui crea due corsi e in uno ci mette due studenti

Student student1 = new Student();

Student student2 = new Student();

student1.setId(1);

student1.setName("Student A");

student2.setId(2);

student2.setName("Student B");

List<Student> course1Students = new ArrayList<>();

course1Students.add(student1);

course1Students.add(student2);

Course course1 = new Course();

Course course2 = new Course();

course1.setId(1);

course1.setName("REST with Spring");

course1.setStudents(course1Students);

course2.setId(2);

course2.setName("Software Engineering");

courses.put(1, course1);

courses.put(2, course2);

}

@GET //get di un corso, cambia quello che segue

@Path("{courseId}")

public Course getCourse(@PathParam("courseId") int courseId) { //path param dice che il courseID che entra e' quello che arriva dal path

return findById(courseId); //separa la ricerca di un corso in funzione a parte

}

//se mettevi qui post senza path, era l aggiunta di un corso

@PUT //put di un corso, cambia quello che segue

@Path("{courseId}")

public Response updateCourse(@PathParam("courseId") int courseId, Course course) { //non entra solo l'id ma anche il corso (con i valori nuovi)

Course existingCourse = findById(courseId);

if (existingCourse == null) {

return Response.status(Response.Status.NOT\_FOUND).build();

}

if (existingCourse.equals(course)) {

return Response.notModified().build();

}

courses.put(courseId, course);

return Response.ok().build();

}

@Path("{courseId}/students") //POCO CHIARO, collega path con classe course, forse perche' si chiama pathToStudent?

public Course pathToStudent(@PathParam("courseId") int courseId) {

return findById(courseId);

}

private Course findById(int id) { //ricerca di un id nella hasmap

for (Map.Entry<Integer, Course> course : courses.entrySet()) {

if (course.getKey() == id) {

return course.getValue();

}

}

return null;

}

}

----------

---------

Course.java

DTO: override di hashCode e equals e no constructor e annotaz xml

package it.sapienza.softeng.rest; //occhio

import javax.ws.rs.\*;

import javax.ws.rs.core.Response;

import javax.xml.bind.annotation.XmlRootElement;

import java.util.ArrayList;

import java.util.List;

@XmlRootElement(name = "Course") //cambia

public class Course {

private int id;

private String name;

private List<Student> students = new ArrayList<>();

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public List<Student> getStudents() {

return students;

}

public void setStudents(List<Student> students) {

this.students = students;

}

@GET //get di uno stud cambia quello che segue

@Path("{studentId}") //la parte prima del path arriva da CoursesRepository

public Student getStudent(@PathParam("studentId") int studentId) {

return findById(studentId);

}

@POST //aggiunta di uno STUDENTE, no path (basta quello che arriva da ...)

public Response createStudent(Student student) { //arriva lo studente, no id (lo ricaveremo), cambia quello che segue

for (Student element : students) {

if (element.getId() == student.getId()) {

return Response.status(Response.Status.CONFLICT).build();

}

}

students.add(student);

return Response.ok(student).build();

}

@DELETE //delete di uno stud, cambia quello che segue

@Path("{studentId}")

public Response deleteStudent(@PathParam("studentId") int studentId) {

Student student = findById(studentId);

if (student == null) {

return Response.status(Response.Status.NOT\_FOUND).build();

}

students.remove(student);

return Response.ok().build();

}

private Student findById(int id) { //findById diversa da quella in CoursesRepository

for (Student student : students) {

if (student.getId() == id) {

return student;

}

}

return null;

}

@Override

public int hashCode() {

return id + name.hashCode();

}

@Override

public boolean equals(Object obj) {

return (obj instanceof Course) && (id == ((Course) obj).getId()) && (name.equals(((Course) obj).getName()));

}

}

------

------

Student.java

semplice DTO perche' dopo studenti non ci sta altro

package it.sapienza.softeng.rest; //occhio

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "Student") //cambia

public class Student {

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public int hashCode() {

return id + name.hashCode();

}

@Override

public boolean equals(Object obj) {

return (obj instanceof Student) && (id == ((Student) obj).getId()) && (name.equals(((Student) obj).getName()));

}

}

-------

REST CLIENT...............................................................................................................



composto da Client.java di core, poi una classe per ogni DTO (le classi saranno simili a quelle del service, ma non uguali

infatti non hanno l'implementaz degli http verbs)

Se serve fare un post hai bisogno di fare un file xml in src/main/resources

-------

Client.java

package it.sapienza.softeng.rest.client; //occhio

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.net.URL;

import javax.xml.bind.JAXB;

import org.apache.http.HttpResponse;

import org.apache.http.client.methods.HttpDelete;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.methods.HttpPut;

import org.apache.http.entity.InputStreamEntity;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClients;

public class Client {

private static final String BASE\_URL = "http://localhost:8080/courses/"; //cambia, url root

private static CloseableHttpClient client;

public static void main(String[] args) throws IOException{

client = HttpClients.createDefault();

// Example GET di un corso

Course course = getCourse(1); //dovrai poi ""implem"" getCourse, o meglio la richiesta GETcourse

System.out.println(course);

for (int i = 0; i < course.getStudents().size(); i++) { //non confonderti, getStudents e' una funzione semplice di Course non una del service

Student student = course.getStudents().get(i);

System.out.println(student);

}

// Example POST/PUT

course = getCourse(2);

System.out.println(course);

for (int i = 0; i < course.getStudents().size(); i++) {

Student student = course.getStudents().get(i);

System.out.println(student);

}

createValidStudent(); //dovrai poi ""implem"" createValidStudent, o meglio la richiesta POST

course = getCourse(2);

System.out.println(course);

for (int i = 0; i < course.getStudents().size(); i++) {

Student student = course.getStudents().get(i);

System.out.println(student);

}

client.close(); //occhio

}

private static Student getStudent(int courseOrder, int studentOrder) throws IOException { //lui non la usa mai

final URL url = new URL(BASE\_URL + courseOrder + "/students/" + studentOrder); //crei url

final InputStream input = url.openStream(); //apri stream (e manda request)

return JAXB.unmarshal(new InputStreamReader(input), Student.class); //unmarshal, cambia class con quella che ricevi

}

private static Course getCourse(int courseOrder) throws IOException {

final URL url = new URL(BASE\_URL + courseOrder); //crei url

final InputStream input = url.openStream(); //apri stream (e manda request)

return JAXB.unmarshal(new InputStreamReader(input), Course.class); //unmarshal, cambia class con quella che ricevi

}

private static void createValidStudent() throws IOException {

final HttpPost httpPost = new HttpPost(BASE\_URL + "2/students"); //crei url, il 2 in teoria dovrebbe arrivarmi come idCorso

final InputStream resourceStream = Client.class.getClassLoader().getResourceAsStream("newStudent.xml"); // qui lui prende il body della richesta dall'xml che sta in src/main/resources

httpPost.setEntity(new InputStreamEntity(resourceStream));

httpPost.setHeader("Content-Type", "text/xml");

final HttpResponse response = client.execute(httpPost); //in caso di put/post non va fatto openStream, ma questo

}

}

REST SERVICE, con DB e JSON TO MOVE DATA...............................................................................................................



DB e' un file, qui lui passa path come arg[0]

JSON cambia solo che vanno messe piu righe nel server, Produces e Consumes cambiano nel Repo, annotazione nei DTO

------

Server.java

apparte roba scriptata di json e due righe per il DB, e' uguale ad APIServer

package it.sapienza.softeng.complexrestservice; //occhio

import com.fasterxml.jackson.jaxrs.json.\*;

import java.util.\*;

import org.apache.cxf.binding.BindingFactoryManager;

import org.apache.cxf.jaxrs.\*;

import org.apache.cxf.jaxrs.lifecycle.\*;

public class Server {

public static void main(String args[]) throws Exception {

JAXRSServerFactoryBean factoryBean = new JAXRSServerFactoryBean();

factoryBean.setResourceClasses(FligthsRepository.class); //cambia classe

//interrompe un attimo roba di REST con due righe per la conn a DB

FligthsRepository fr = new FligthsRepository();

fr.setConnection(args[0]); //dara' la posizione del db in partenza, facoltativo farlo cosi

factoryBean.setResourceProvider(new SingletonResourceProvider(fr)); //riprende

factoryBean.setAddress("http://localhost:8080/"); //cambia

List<Object> providers = new ArrayList<Object>(); //da qui roba scriptata di json

providers.add(new JacksonJaxbJsonProvider());

factoryBean.setProviders(providers);

BindingFactoryManager manager = factoryBean.getBus().getExtension(BindingFactoryManager.class);

JAXRSBindingFactory restFactory = new JAXRSBindingFactory();

restFactory.setBus(factoryBean.getBus());

manager.registerBindingFactory(JAXRSBindingFactory.JAXRS\_BINDING\_ID, restFactory);

org.apache.cxf.endpoint.Server server = factoryBean.create();

System.out.println("Server ready...");

while (true) {

}

}

}

-------

-------

FligthsRepository.java

package it.sapienza.softeng.complexrestservice;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.ws.rs.\*;

import javax.ws.rs.core.Response;

@Path("/fligths") //mettendo qui il path root e come se ce l avessi sempre

public class FligthsRepository {

private Connection conn;

/\* Initial simple version, based on main memory, come CoursesRepository

final private Map<Integer, Fligth> fligths = new HashMap<>();

{

Fligth fl1 = new Fligth();

Fligth fl2 = new Fligth();

fl1.setId(1);

fl1.setName("AZ140");

fl2.setId(2);

fl2.setName("LH9120");

fligths.put(1, fl1);

fligths.put(2, fl2);

}

\*/

public void setConnection(String pos) { //per connetterti a DB, chiamata da Server.java

try {

try {

Class.forName("org.sqlite.JDBC");

} catch (ClassNotFoundException ex) {

Logger.getLogger(FligthsRepository.class.getName()).log(Level.SEVERE, null, ex); //cambia

}

conn = DriverManager.getConnection("jdbc:sqlite:"+pos); //usero conn per usare db

} catch (SQLException ex) {

Logger.getLogger(FligthsRepository.class.getName()).log(Level.SEVERE, null, ex); //cambia

}

}

@GET //get di un volo

@Path("{fligthId}")

@Produces("application/json") //diverso da xml

public Fligth getFligth(@PathParam("fligthId") int fligthId) { //come getCourse

return findById(fligthId);

}

@PUT //put di un volo

@Path("{fligthId}")

@Consumes("application/json") //in xml non c'era proprio

public Response updateFligth(@PathParam("fligthId") int fligthId, Fligth fligth) { //simile a put di un corso

Fligth existing = findById(fligthId);

if (existing == null) {

return Response.status(Response.Status.NOT\_FOUND).build();

}

if (existing.equals(fligth)) {

return Response.notModified().build();

}

// fligths.put(fligthId, fligth);

update(fligthId, fligth); //dovra' poi impl update

return Response.ok().build();

}

private Fligth findById(int id) { //simile a find di un corso

PreparedStatement stat = null;

Fligth fl = null;

try {

stat = conn.prepareStatement("select \* from fligth where id = ?"); //fa query a db

stat.setString(1, String.valueOf(id));

ResultSet rs = stat.executeQuery();

if (rs.next()) {

fl = new Fligth();

fl.setId(Integer.parseInt(rs.getString("id")));

fl.setName(rs.getString("name"));

Logger.getLogger(FligthsRepository.class.getName()).log(Level.INFO, "Accessed : " + fl);

}

rs.close();

} catch (SQLException ex) {

Logger.getLogger(FligthsRepository.class.getName()).log(Level.SEVERE, null, ex);

}

/\* simple version, come in CoursesRepository

for (Map.Entry<Integer, Fligth> fligth : fligths.entrySet()) {

if (fligth.getKey() == id) {

return fligth.getValue();

}

}

\*/

return fl;

}

private void update(int fligthId, Fligth fligth)

{ //cambia robe

PreparedStatement stat = null;

try {

stat = conn.prepareStatement("update fligth set name = ? where id = ?"); //query di update

stat.setString(1, fligth.getName());

stat.setString(2, String.valueOf(fligthId));

int affectedRow = stat.executeUpdate();

if (affectedRow == 1) {

Logger.getLogger(FligthsRepository.class.getName()).log(Level.INFO, "Updated : " + fligth);

return;

}

else throw new RuntimeException();

}

catch (Exception ex) {

Logger.getLogger(FligthsRepository.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

-------

-------

Flight.java

DTO semplice, perche' dopo i voli non c e altro, override del toString facoltativo?

package it.sapienza.softeng.complexrestservice; //occhio

import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;

@JacksonXmlRootElement(localName = "Fligth") //diversa da xml

public class Fligth {

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public int hashCode() {

return id + name.hashCode();

}

@Override

public boolean equals(Object obj) {

return (obj instanceof Fligth) && (id == ((Fligth) obj).getId()) && (name.equals(((Fligth) obj).getName()));

}

@Override

public String toString() {

return "fligth " + id + " " + name;

}

}

-------

REST CLIENT, con JSON TO MOVE DATA...............................................................................................................



simile ma non uguale a RestClient classico, cambia unmarshal e altro (non fa le cose in funzioni apparte, tutto nel main)

--------

Client.java

package it.sapienza.softeng.jsonclient; //occhio

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClients;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.OutputStream;

import org.apache.http.HttpResponse;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.methods.HttpPut;

import org.apache.http.entity.InputStreamEntity;

import org.apache.http.entity.StringEntity;

public class Client {

private static final String BASE\_URL = "http://localhost:8080/fligths/"; //path base

private static CloseableHttpClient client;

public static void main(String[] args) throws IOException {

client = HttpClients.createDefault();

// Example GET di un volo

ObjectMapper mapper = new ObjectMapper(); //lo usera per unmarshal

URL url = new URL(BASE\_URL + "2"); //crea url, 2 hardcoded (client non interattivo)

InputStream input = url.openStream(); //apre stream (e manda request)

Fligth fl = (Fligth)mapper.readValue(input, Fligth.class); //unmarshal in caso di get

System.out.println(fl);

// Example POST/PUT di un volo //fatto tutto qui, non come in RestClient che lo prendevamo da file xml

ObjectMapper objectMapper = new ObjectMapper(); //lo usera per unmarshal

Fligth newFl = new Fligth(); //prepara il volo

newFl.setId(4);

newFl.setName("XX000");

String json = objectMapper.writeValueAsString(newFl); //unmarshal/marshal in caso di post/put

HttpPut httpPut = new HttpPut(BASE\_URL + "2/"); //crea url

StringEntity entity = new StringEntity(json); //segue roba scriptata per post/put in json

httpPut.setEntity(entity);

httpPut.setHeader("Accept", "application/json");

httpPut.setHeader("Content-type", "application/json");

HttpResponse response = client.execute(httpPut); //in caso di put/post non va fatto openStream, ma questo

System.out.println(response);

//rifa' la medesima GET di prima

InputStream input2 = url.openStream(); //apre stream, (e manda request)

fl = (Fligth) mapper.readValue(input2, Fligth.class); //unmarshal in caso di get

System.out.println(fl);

}

}

------

------

Flight.java

semplice DTO

package it.sapienza.softeng.jsonclient; //occhio

import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;

@JacksonXmlRootElement(localName = "Fligth") //diversa da xml

public class Fligth {

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public int hashCode() {

return id + name.hashCode();

}

@Override

public boolean equals(Object obj) {

return (obj instanceof Fligth) && (id == ((Fligth) obj).getId()) && (name.equals(((Fligth) obj).getName()));

}

@Override

public String toString(){

return "fligth " + id + " " + name;

}

}

-------

DBSimpleManager...............................................................................................................



in JMS PRODUCER e REST COMPLEX SERVICE si usa DB, rispetto a questo cambia come creano l'oggetto conn e che fanno try catch

------

DBManager.java

molto da cambiare in base a cosa serve

package it.sapienza.softeng.dbsimplemanager; //occhio

import java.sql.\*; //unico import

public class DBManager {

public static void main(String[] args) throws Exception {

Class.forName("org.sqlite.JDBC"); //core

Connection conn = DriverManager.getConnection("jdbc:sqlite:"+args[0]); //core,cambia arg e metti una variabile stringa se non lo vuoi in start

Statement stat = conn.createStatement(); //usera questo per varie query senza '?'

if (args[1].equals("create")) {

stat.executeUpdate("drop table if exists fligth;"); //query senza '?' , non semplice select

stat.executeUpdate("create table fligth (id, name);"); //query senza '?' , non semplice select

PreparedStatement prep = conn.prepareStatement(

"insert into fligth values (?, ?);"); //per le query con il '?' usa un PreparedStatement

prep.setString(1, "1");

prep.setString(2, "AZ140"); //i set string fillano i '?'

prep.addBatch();

conn.setAutoCommit(false);

prep.executeBatch();

conn.setAutoCommit(true);

prep.setString(1, "2");

prep.setString(2, "LH999");

prep.addBatch();

conn.setAutoCommit(false);

prep.executeBatch();

conn.setAutoCommit(true);

prep.setString(1, "3");

prep.setString(2, "FR123");

prep.addBatch();

conn.setAutoCommit(false);

prep.executeBatch();

conn.setAutoCommit(true);

prep.setString(1, "4");

prep.setString(2, "US666");

prep.addBatch();

conn.setAutoCommit(false);

prep.executeBatch();

conn.setAutoCommit(true);

} else {

ResultSet rs = stat.executeQuery("select \* from fligth;"); //query senza '?', semplice query

while (rs.next()) {

System.out.print("Fligth = " + rs.getString("id") + " is : ");

System.out.println(rs.getString("name"));

}

rs.close();

}

conn.close();

}

}

------

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>3.2.4</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>it.uniroma1.grpcexampleserver.MyServer</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

<configuration>

<archive>

<manifest>

<mainClass>

com.baeldung.executable.ExecutableMavenJar

</mainClass>

</manifest>

</archive>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

</configuration>

</execution>

</executions>

</plugin>