UNIVERSITÀ DI PISA

Programmazione di reti
Corso B

13 Dicembre 2016
Lezione 12

Annunci

- Ricevimento mercoledì 14:30 - 16:00 **su appuntamento**
- Questionario anonimo online su Moodle
    - Per vedere quale argomento vi è piaciuto di più

Contenuti

- Servizi web
- Java EE teaser
- Server REST in Java

Java Enterprise Edition

- Java per applicazioni Web
- Robustezza e facilità di uso
  - modello basato su *container* - facili da configurare
    - programmatore deve solo sviluppare la parte logica dell'applicazione
    - si usano oggetti Java standard *annotati* - "*annotated POJO" (Plain Old Java Object)*
- Sviluppato anche dalla comunità
- HTML5, XML, JSON
- Ultima versione: Java EE 7

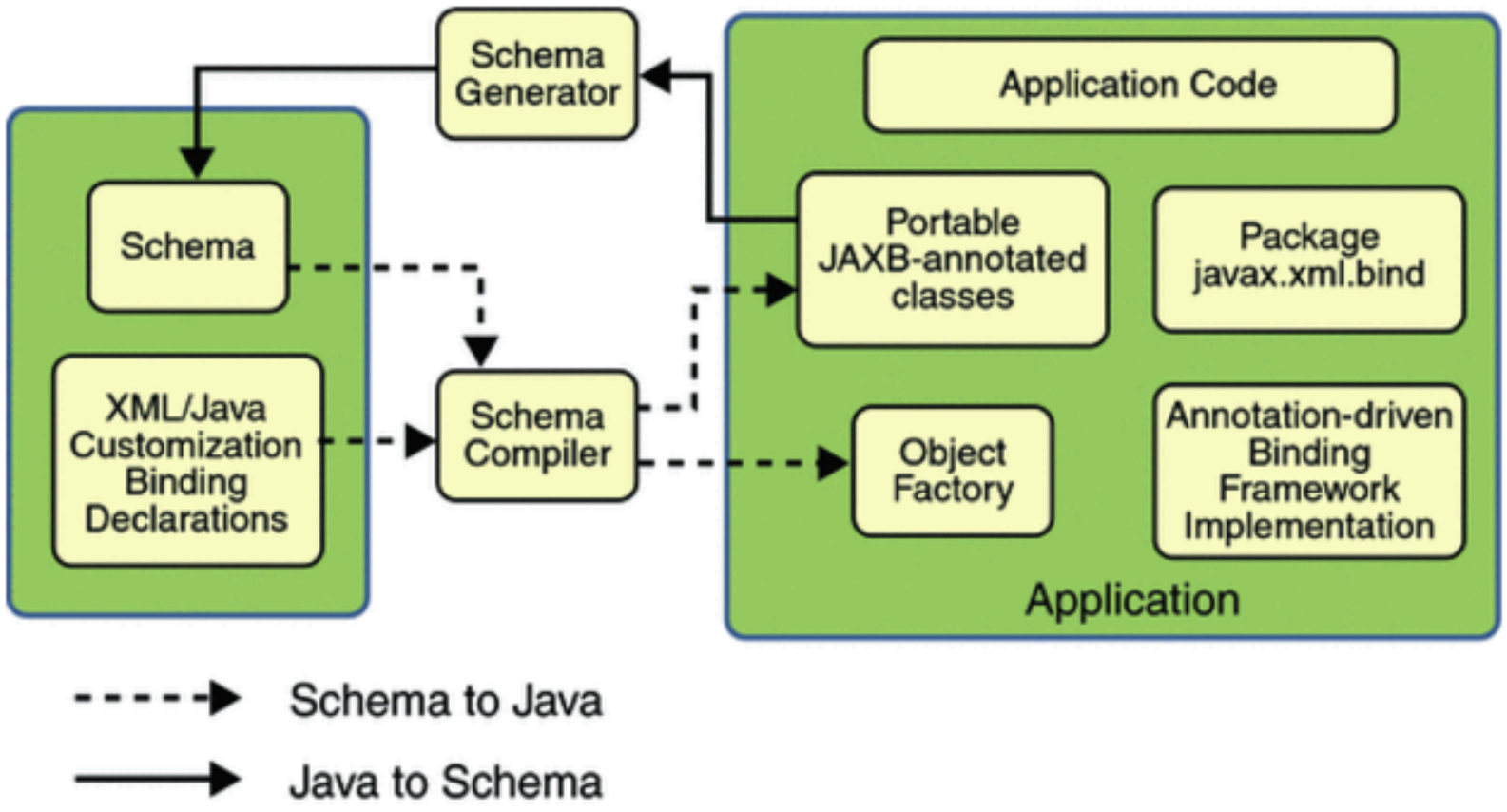
Servlet

- Oggetto Java in grado di rispondere a delle richieste sul web - applicazione web dinamica (lato server)
- Segue lo standard definito dalla `Java Servlet API`
- Implementato in pacchetto `javax.servlet` (classe di base `HTTPServlet`)
- Viene lanciato dentro un *container* web su un *web server* generale (e.g. Apache Tomcat)
- Può essere esportato come applicazione web - un file WAR (*Web ARchive*)
- Usato per servizi *web* - noi lo useremmo come *container* per il nostro servizio REST, senza creare una classe `Servlet` manualmente

Dati

- Dati (richieste, risposte) inviati sono in formato *XML/JSON/text*.
- Java EE contiene librerie per trasformare automaticamente oggetti Java in una rappresentazione XML e vice versa : *JAXB - Java Architecture for XML Binding*
  - binding tra una classe Java e un schema XML
    - *JAXB1* solo da XML a Java
    - *JAXB2* anche da Java a XML, usando pacchetto `javax.xml.bind.annotation`
  - *marshalling* e *unmarshalling*
- Un binding XML può essere usato anche per ottenere formato JSON

JAXB


Schema Generator
Application Code
Schema
Portable JAXB-annotated classes
Package javax.xml.bind
XML/Java Customization Binding Declarations
Schema Compiler
Object Factory
Annotation-driven Binding Framework Implementation
Application
Schema to Java
Java to Schema

JAXB

- Per creare un servizio *web*, ci interessa il *binding* da Java a XML
- Annotazioni disponibili:
  - per pacchetti
  - per classi
    - `@XMLRootElement, @XMLType`
  - per enum
  - per attributi e metodi
    - `@XMLElement , @XMLAttribute`, etc.

JAXB

- Modo più semplice:
  - Annotare la classe con `@XMLRootElement(name="nometag")`
  - Includere un costruttore senza parametri
  - Includere dei *setter* pubblici per gli attributi
  - Si ottiene automaticamente un schema XML:
    - ogni oggetto è rappresentato da un elemento XML <nometag>
    - l'elemento <nometag> contiene un elemento figlio per ogni attributo

JAXB

```
@XmlRootElement(name = "student")
public class Student{ …..
```

```
<student>
        <fname>Robert</fname>
        <grade>27.0</grade>
        <lname>Brown</lname>
        <street>Dawson</street>
</student>
```

JAX-WS

- *Java API for XML Web Services*
- Servizi web *heavy-weight* - usano SOAP e WSDL
- Per creare un web service: creare una classe annotata con `@WebService` (`javax.jws.WebService`)
  - metodi da pubblicare devono essere pubblici e non statici e non final, annotati con `@WebMethod` (`javax.jws.WebMethod`)
  - includere un costruttore senza parametri
  - esportare in un WAR e includere il servizio in un web server (per dettagli consultare la Javadoc)

JAX-RS

- *Java API for RESTful Web Services*
- Librerie per creare dei servizi web REST
  - Usano operazioni HTTP standard
  - Non usano SOAP e WSDL

Jersey

- https://jersey.java.net/

- Implementazione di riferimento di *JAX-RS* più tante estensioni

- Facile da usare - qui usiamo solo 2-3 funzionalità

Operazioni REST

- Operazioni seguono regole standard HTTP:
  - GET - *Read only, ripetibile and safe* - non deve avere effetti collaterali
  - PUT, DELETE - *Ripetibili*, hanno degli *effetti collaterali*
  - POST - *Non-ripetibile, con effetti collaterali*
- Non sono imposte da nessuna restrizione fisica, è la **responsabilità dello sviluppatore** di seguirle

JAX-RS

- Un servizio web REST è implementato tramite una *REST Root Resource Class*
  - POJO annotato con `@Path` (`javax.ws.rs.Path`)

    o

  - POJO con almeno un metodo annotato con `@Path` o con un '*request method designator*' (`@GET/@POST/@PUT/@DELETE`)
  - i metodi annotati si chiamano *'resource method'*
- Al runtime la piattaforma JAX-RS parsa le annotazioni e crea automaticamente delle classi per ottenere il servizio web.

Annotazioni JAX-RS

- `@Path`
  - annotazione per la *Root Resource Class* o per un *resource method*
  - determina la URI relativa dove la risorsa sarà disponibile.
  - può contenere dei parametri (*template URI*)
  - E.g. `@Path{“/”}` , `@Path{“/mywebservice/{studentID}”}`
- `@GET , @POST, @PUT, @DELETE` *(request method designators)*
  - annotazioni per un r*esource method*
  - questo metodo risponde ad una richiesta HTTP GET/POST/PUT/DELETE

Annotazioni JAX-RS

- Annotazioni per <u>parametri dei metodi:</u>
  - `@QueryParam` - questo parametro sarà trovato nel URL, nella parte di parametri (dopo il '?')
  - `@PathParam` - questo parametro sarà trovato nell'URL, nella parte di *path* (prima del '?') (e.g. `@PathParam{name=studentID}`)

Annotazioni JAX-RS

- `@Produces`
  - annotazione per classe o metodo
  - tipo di contenuto generato da una richiesta a questa risorsa
  - un MIME type (e.g. `@Produces(MediaType.APPLICATION_JSON)`)
- `@Consumes`
  - annotazione per classe o metodo
  - tipo di contenuto consumato da questa risorsa (nel body della richiesta)

Deployment

- Per istallare il nostro servizio web abbiamo bisogno di un *server web* generico (Tomcat, GlassFish, etc)

- Una volta sviluppate le classi, si esportano in un file WAR (Web ARchive) che viene copiato nel server web ad una locazione precisa (folder `webapps` per Tomcat)

- Per configurare il modo in cui il servizio viene pubblicato, dobbiamo includere una descrizione in XML

web.xml

- Elementi importanti:
  - `<servlet>`
    - Include il tipo di servlet da usare (generato automaticamente dalle POJO annotate) - usando elemento `<servlet-class>`
      - In Jersey1 possiamo usare `com.sun.jersey.spi.container.servlet.ServletContainer`
    - Identifica la classe che rappresenta la *Root Resource Class* - usando elemento `<init-param>`
  - `<servlet-mapping>`
    - Include l'URI relativo dove il servizio sarà disponibile.

Esempio

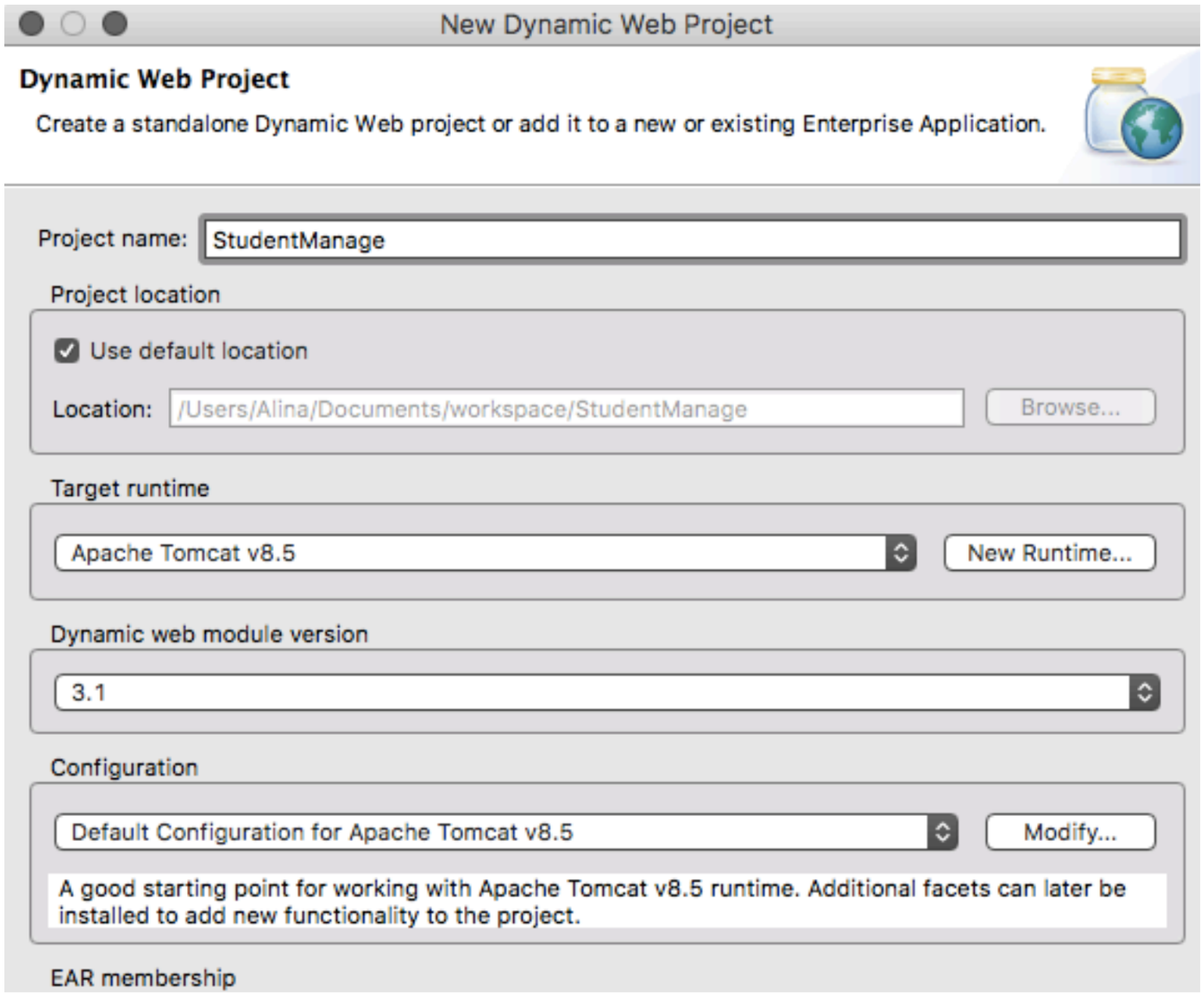
- *Student manager* come servizio web
- 2 operazioni
  - GET - ricerca utenti per cognome
  - POST - cambia voto

Setup

- Istalliamo `Eclipse` e `Java EE`
- Istalliamo `Apache Tomcat` - server che ci aiuta a fare deployment a Servlet Java
- Scarichiamo libreria `Jersey1.*`

Development

- Creiamo un progetto di tipo *Dynamic Web Project*, selezionando `Tomcat` come runtime



Development

- Aggiungiamo le librerie `Jersey` e `JsonSimple` al progetto (*Configure Build Path*)
- Creiamo un POJO `Students` :
  - Usato internamente per rappresentare gli studenti
  - Annotato usando `JAXB` per essere trasformato facilmente in XML o JSON - risposta alla ricerca

```
import javax.xml.bind.annotation.XmlRootElement;

import org.json.simple.JSONObject;


@XmlRootElement(name = "student")
public class Student{

   private String fname, lname;
   private String streetAddress;
   private long addressNo;
   private double grade;
   private long studentId;

   public Student(){}

   public Student(String fname, String lname,String street,
                  long no,long id, double grade){
      this.fname=fname;
      this.lname=lname;
      this.streetAddress= street;
      this.addressNo=no;
      this.grade=grade;
      this.studentId=id;
   }
```

```
public String getLname() {
   return this.lname;
}
public String getFname(){
   return this.fname;
}
public long getId(){
   return this.studentId;
}
public long getAddressNo(){
   return this.addressNo;
}
public String getStreet(){
   return this.streetAddress;
}
public double getGrade(){
   return this.grade;

}
```

```
public void setGrade(double grade){
     this.grade=grade;
}
 public void setFname(String fname){
     this.fname=fname;
 }
 public void setLname(String lname){
     this.lname=lname;
 }
public void setStreet(String street){
   this.streetAddress=street;
}
public void setStreetNo(int no){
   this.addressNo=no;
}
public void setId(int id){
   this.studentId=id;
}
```

```
public static Student fromJSON(JSONObject object) {
    return new Student(
          (String)object.get("fname"),
          (String)object.get("lname"),
          (String)((JSONObject)object.get("address")).get("street"),
          (long)((JSONObject)object.get("address")).get("no"),
          (long)object.get("id"),
          (double)object.get("grade"));

  }
public JSONObject toJson(){
    JSONObject result= new JSONObject();
    result.put("fname", this.fname);
    result.put("lname", this.lname);
    result.put("id", this.studentId);
    result.put("grade", this.grade);
    JSONObject address= new JSONObject();
    address.put("no",this.addressNo);
    address.put("street", this.streetAddress);
    result.put("address", address);
    return result;
  }}
```

Development

- Creiamo un POJO `StudentManager` :
  - Usato come *Root Resource Class*
  - 2 metodi - `searchByLastName` e `getGrade`
  - Annotato usando `JAX-RS`

```
@Path("/")
public class StudentManager  {

   public StudentManager() {}

   @GET
   @Path("/search")
   @Produces(MediaType.APPLICATION_JSON)
   public List<Student> searchByLastName(@DefaultValue("")
                                         @QueryParam("lname")
                                               String lastName) {
      List<Student> students=this.loadDatabase();
      ArrayList<Student> result= new ArrayList<>();
      for (Student s : students){
         if (s.getLname().equals(lastName)){
            result.add(s);
         }
      }
      return result;
}
```

```
@POST
@Path("/setGrade")
@Produces(MediaType.APPLICATION_JSON)
public MyResponse setGrade(@QueryParam("id") int id,
                           @QueryParam("grade") double grade) {
   List<Student> students=this.loadDatabase();
   for (Student s : students){
      if (s.getId()==id){
         s.setGrade(grade);
         this.dumpDatabase(students);
         return new MyResponse("student found");
      }
   }
   return new MyResponse("student not found");
}
```

```
private List<Student> loadDatabase(){
   List<Student> students=new  ArrayList<>();
   JSONParser parser= new JSONParser();
   try(FileReader registro= new FileReader("registro.json");){
      JSONArray array= (JSONArray) parser.parse(registro);
      for (Object jo: array){
         students.add(Student.fromJSON((JSONObject)jo));
         System.out.println(students.get(students.size()-1));
      }
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   } catch (ParseException e) {
      e.printStackTrace();
   }
   return students;
}
```

```
private void dumpDatabase(List<Student> students){
   JSONArray jsonStudents= new JSONArray();
   for (Student s : students){
      jsonStudents.add(s.toJson());
   }

   try(FileWriter registro= new FileWriter("registro.json");){
         jsonStudents.writeJSONString(registro);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

}
```

Development

- Creiamo un POJO `MyResponse` :
  - Rappresenta una risposta contenendo una stringa
  - Annotato usando `JAXB`

```java
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="response")
public class MyResponse {

   String message;

   public MyResponse(){
   }

   public MyResponse(String m){
      this.message=m;
   }

   public String getMessage() {
      return message;
   }

   public void setMessage(String message) {
      this.message = message;
   }
}
```

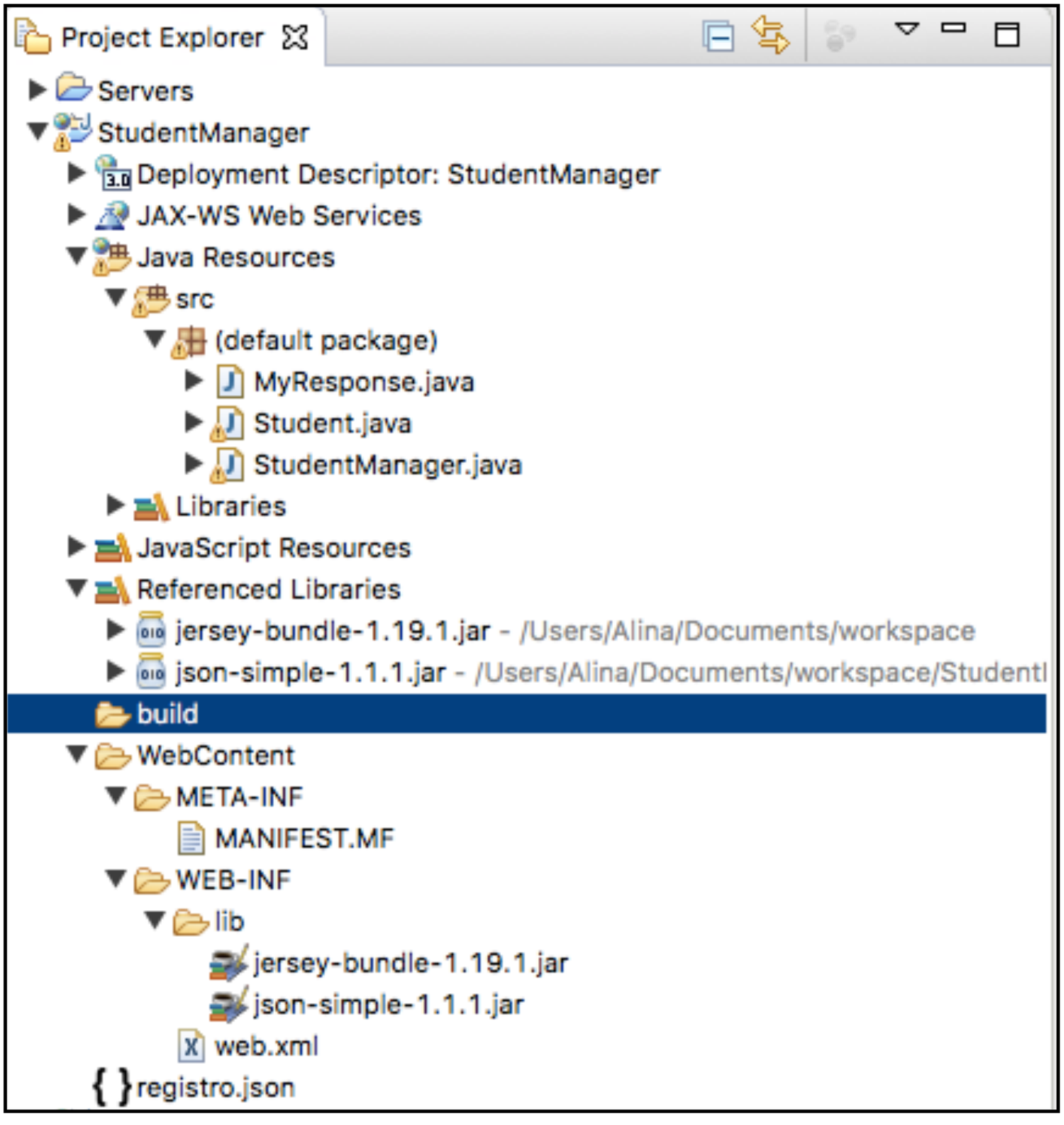
Deployment

- Creiamo file `web.xml` nel folder `WebContent/WEB_INF`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="Students" version="3.0">
    <display-name>Student Manager</display-name>
    <servlet>
        <servlet-name>Student Manager</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                </servlet-class>
         <init-param>
                <param-name>jersey.config.server.provider.packages</param-name>
                <param-value>StudentManager</param-value>
            </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>Student Manager</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

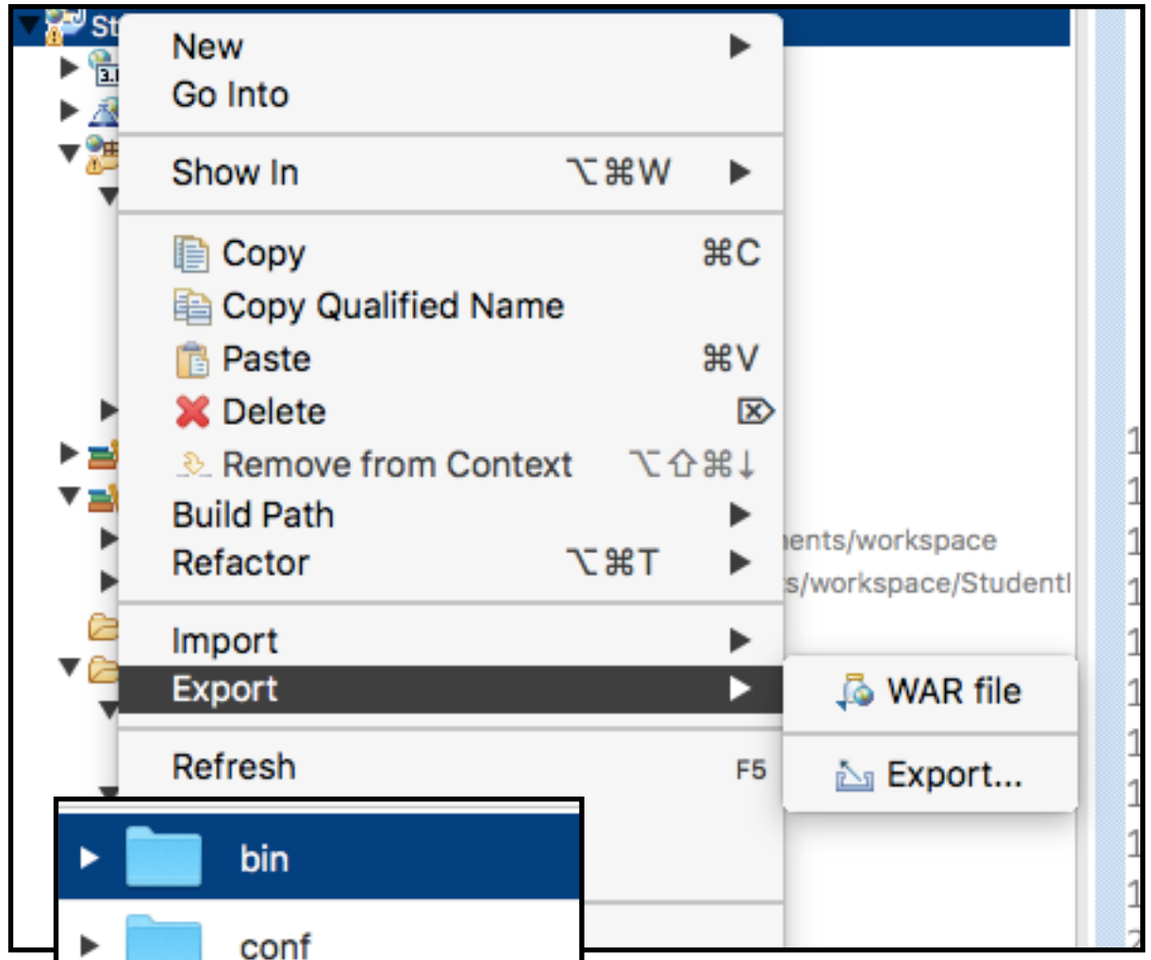
Deployment

- Copiamo le librerie nel folder WebContent/WEB_INF/lib



Deployment

- Esportiamo il progetto in un file WAR (Web ARchive)
- Includiamo il file WAR nel folder **`webapps`** di **`Tomcat`**
- Includiamo la database **`registro.json`** nel folder **`bin`** di **`Tomcat`** (o la aggiungiamo al CLASSPATH)
- Lanciamo **`Tomcat`**



Testing

- Usiamo `Postman Chrome` extension per inviare richieste GET e POST (GET funziona anche dal browser)

registro.json


```
[{"fname":"Robert","lname":"Brown","address":{"no":12,"street":"Dawson"},"grade":-1.0,"id":0},
{"fname":"Michael","lname":"Reds","address":{"no":40,"street":"Pearse"},"grade":-1.0,"id":1},
{"fname":"Joanna","lname":"Moore","address":{"no":62,"street":"Collins"},"grade":-1.0,"id":2},
{"fname":"Ann","lname":"Brown","address":{"no":132,"street":"Buffallo"},"grade":-1.0,"id":3}]
```

GET

http://localhost:8080/StudentManager/search?lname=Brown

GET | http://localhost:8080/StudentManager/search?lname=Brown | Params | Send

Pretty Raw Preview JSON

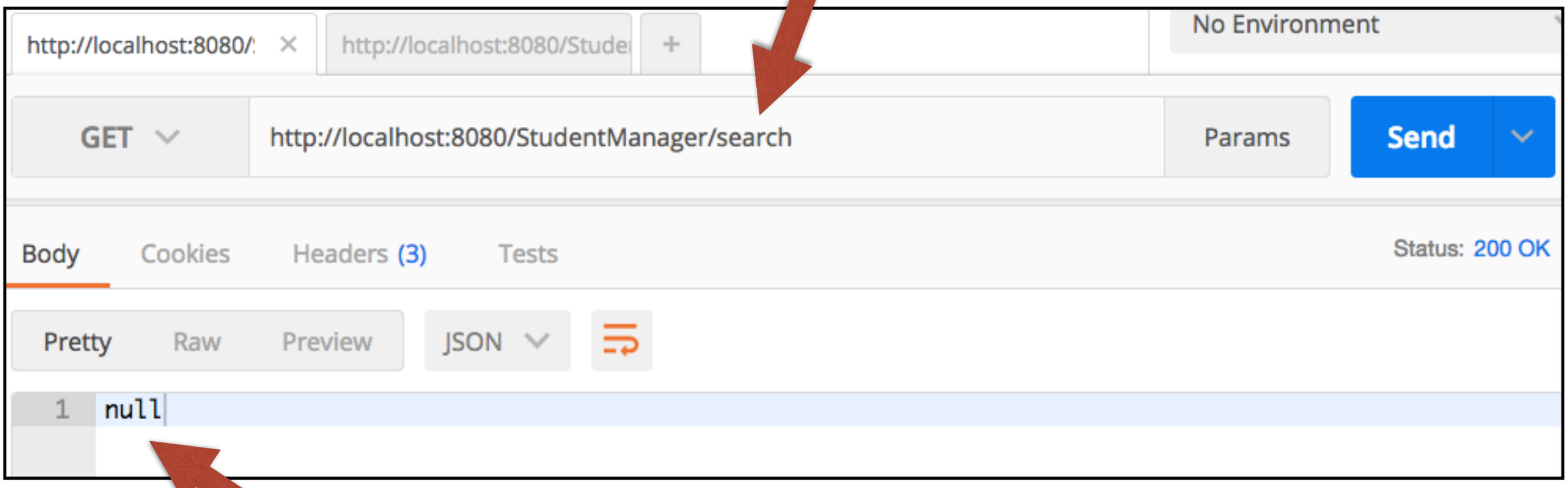

```
{
  "student": [
    {
      "addressNo": "12",
      "fname": "Robert",
      "grade": "-1.0",
      "lname": "Brown",
      "street": "Dawson",
      "studentId": "0"
    },
    {
      "addressNo": "132",
      "fname": "Ann",
      "grade": "-1.0",
      "lname": "Brown",
      "street": "Buffallo",
      "studentId": "3"
    }
  ]
}
```

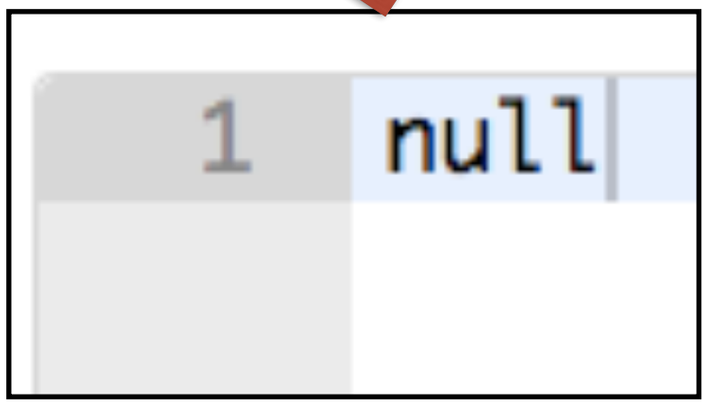


```
"student": [
  {
    "addressNo": "12",
    "fname": "Robert",
    "grade": "-1.0",
    "lname": "Brown",
    "street": "Dawson",
    "studentId": "0"
  },
```

GET

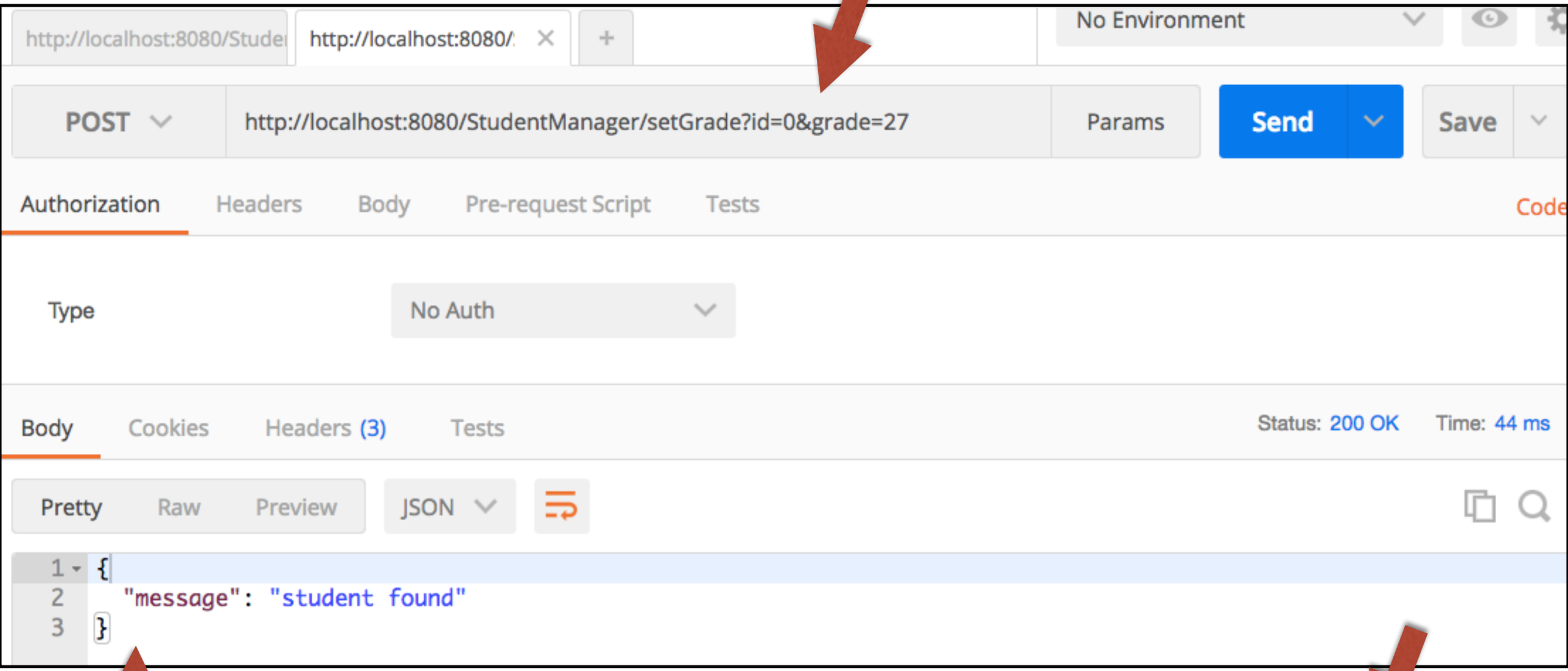
http://localhost:8080/StudentManager/search





POST

http://localhost:8080/StudentManager/setGrade?id=0&grade=27



POST

http://localhost:8080/StudentManager/setGrade?id=9&grade=27

http://localhost:8080/Stude | http://localhost:8080/ | No Environment

POST | http://localhost:8080/StudentManager/setGrade?id=9&grade=27 | Params | Send | Save

Authorization | Headers | Body | Pre-request Script | Tests | Code

Type | No Auth

Body | Cookies | Headers (3) | Tests | Status: 200 OK | Time: 35 ms

Pretty | Raw | Preview | JSON

```
1 {
2   "message": "student not found"
3 }
```


```
1 {
2   "message": "student not found"
3 }
```

GET

http://localhost:8080/StudentManager/search?lname=Brown

GET | http://localhost:8080/StudentManager/search?lname=Brown | Params | Send | Save

Pretty Raw Preview JSON


```
{
  "student": [
    {
      "addressNo": "12",
      "fname": "Robert",
      "grade": "27.0",
      "lname": "Brown",
      "street": "Dawson",
      "studentId": "0"
    },
    {
      "addressNo": "132",
      "fname": "Ann",
      "grade": "-1.0",
      "lname": "Brown",
      "street": "Buffallo",
      "studentId": "3"
    }
  ]
}
```



```
{
  "addressNo": "12",
  "fname": "Robert",
  "grade": "27.0",
  "lname": "Brown",
  "street": "Dawson",
  "studentId": "0"
},
```

```
@GET
   @Path("/search")
   @Produces(MediaType.APPLICATION_XML)
   public List<Student> searchByLastName(@DefaultValue("")
                                         @QueryParam("lname")
                                            String lastName) {
      List<Student> students=this.loadDatabase();
      ArrayList<Student> result= new ArrayList<>();
      for (Student s : students){
         if (s.getLname().equals(lastName)){
            result.add(s);
         }
      }
      return result;
}
```

- Export WAR file
- Restart Tomcat

GET

http://localhost:8080/StudentManager/search?lname=Brown

http://localhost:8080/

GET http://localhost:8080/StudentManager/search?lname=Brown Params Send

Pretty Raw Preview XML

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <students>
3      <student>
4          <addressNo>12</addressNo>
5          <fname>Robert</fname>
6          <grade>27.0</grade>
7          <lname>Brown</lname>
8          <street>Dawson</street>
9          <studentId>0</studentId>
10     </student>
11     <student>
12         <addressNo>132</addressNo>
13         <fname>Ann</fname>
14         <grade>-1.0</grade>
15         <lname>Brown</lname>
16         <street>Buffallo</street>
17         <studentId>3</studentId>
18     </student>
19 </students>
```

```
<student>
    <addressNo>12</addressNo>
    <fname>Robert</fname>
    <grade>27.0</grade>
    <lname>Brown</lname>
    <street>Dawson</street>
    <studentId>0</studentId>
</student>
```

Progetto

- GUI
  - potete usare qualsiasi libreria, anche esterna
  - se volete usare `Swing` ci sono le slide dell'anno scorso (ultima lezione su Moodle)

THE END!

- Buone vacanze e ci vediamo all’esame!
- Ricordatevi dei questionari
- Email per domande e/o ricevimento