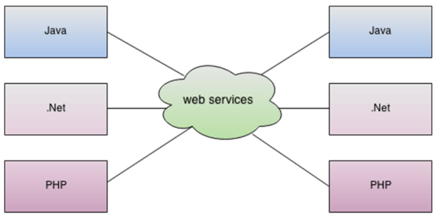
54

Web Service

W3C (World Wide Web Consortium) describes web service as a system of software allowing different machines to interact with each other through network.

A **Web Service** can be defined by following ways:

- It is a client-server application or application component for communication.
- The method of communication between two devices over the network.
- It is a software system for the interoperable machine to machine communication.
- It is a collection of standards or protocols for exchanging information between two devices or application.



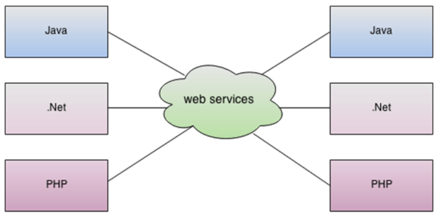
55

Web Service

W3C (World Wide Web Consortium) describes web service as a system of software allowing different machines to interact with each other through network.

Why Web Service?

- Modern day business applications use variety of programming platforms to develop web-based applications Java..Net, while some other in Angular JS, Node.js, etc.
- Most often than not, these heterogeneous applications need some sort of communication to happen between them. Since they are built using different development languages, it becomes difficult to ensure accurate communication between applications.
- Web services provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other.



56

56

Web Service

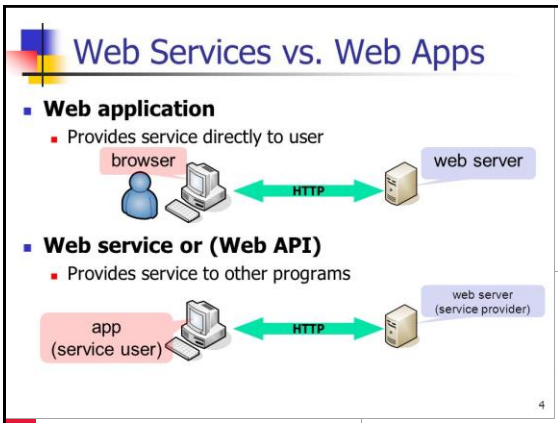
W3C (World Wide Web Consortium) describes web service as a system of software allowing different machines to interact with each other through network.

Web Services vs Web Applications

- Web Services can be used to transfer data between Web Applications.
- Web Services can be accessed from any languages or platform.
- A Web Application is meant for humans to read, while a Web Service is meant for computers to read.
- Web Application is a complete Application with a Graphical User Interface (GUI), however, web services do not necessarily have a user interface since it is used as a component in an application.
- Web Application can be access through browsers.

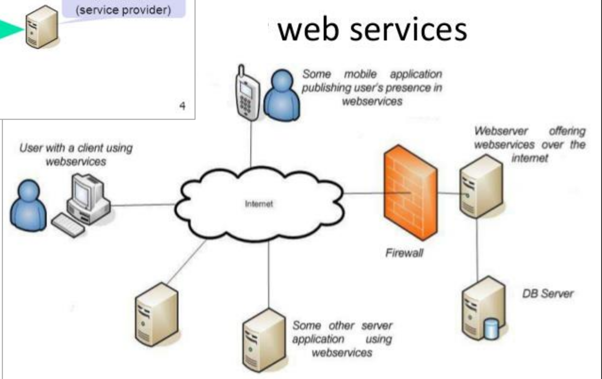
57

57



Web Services vs. Web Apps

- **Web application**
  - Provides service directly to user
- **Web service or (Web API)**
  - Provides service to other programs


web services
Some mobile application publishing user's presence in webservices
User with a client using webservices
Internet
Firewall
Webserver offering webservices over the internet
DB Server
Some other server application using webservices

58

58

Web Services vs Web Applications

A Web Application can consist of multiple Web Services. To differentiate between two, ask what interacts with it.

Web Application: End Users via a User Interface.

Web Service: Web App / Web Service Interaction via HTTP/S requests.

For example, an E-Commerce site is in its entirety a Web Application. It has users interact with it, to purchase items. It then speaks to its appropriate Web Services to achieve what the user wants:

- Order Service to place orders.
- Accounts Service to register a new Customer or update their details.
- Product Service to check if a particular item is in stock or to send results based on search criteria.
- ….

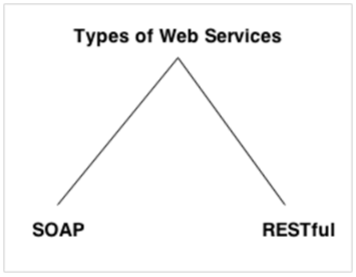
60

60

Types of Web Services

There are mainly two types of web services.

1. SOAP web services.
2. RESTful web services.



SOAP

SOAP is an acronym for Simple Object Access Protocol. SOAP is a XML-based protocol for accessing web services. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

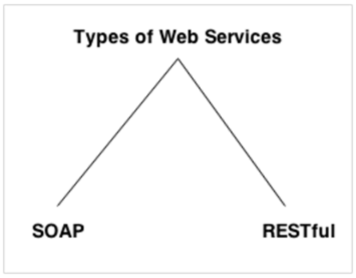
61

61

Types of Web Services

There are mainly two types of web services.

1. SOAP web services.
2. RESTful web services.



RESTful Web Services

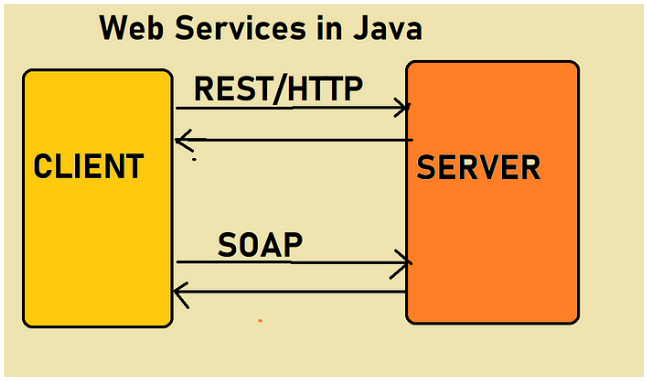
Fast: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.

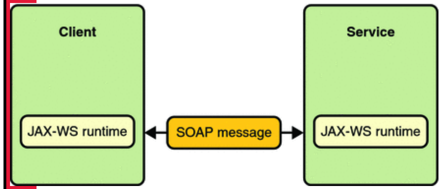
Language and Platform independent: RESTful web services can be written in any programming language and executed in any platform.

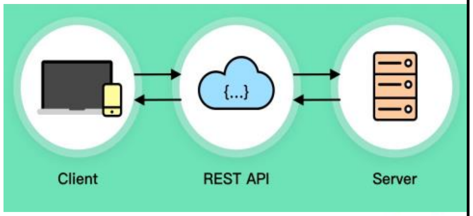
Permits different data format: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

62

62







63

What is REST?

RESTful Service: Representational State Transfer (REST). Has gained widespread acceptance across the Web as a simpler alternative to SOAP and Web Services Description Language (WSDL) based Web services.

REST defines a set of <u>architectural principles</u> by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.

If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use.

64

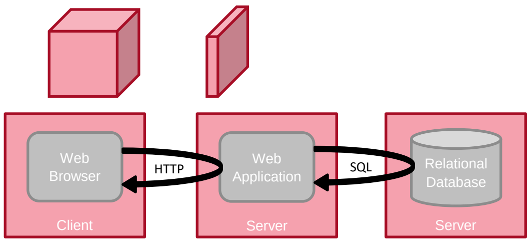
YORK U UNIVERSITÉ UNIVERSITY

What is REST?

An architectural style for *`distributed hypermedia systems`* described by Roy Thomas Fielding in his doctoral dissertation 2000.

Consists of constraints:

1. `Client - Server`
2. `Stateless`
3. `Cache`
4. `Uniform Interface`
5. `Layered System`
6. `Code-On-Demand`



Web Services that adhere to REST architectural style are characterized as *`RESTful web services,`*

66

What does REST mean?

`The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.`

`From Roy's dissertation.`

67

Using HTTP as the uniform interface

- Use URIs to identify resources.
- Use HTTP methods to specify operation:
  - Create: **POST**
  - Retrieve: **GET**
  - Update: **PUT** (or PATCH)
  - Delete: **DELETE**

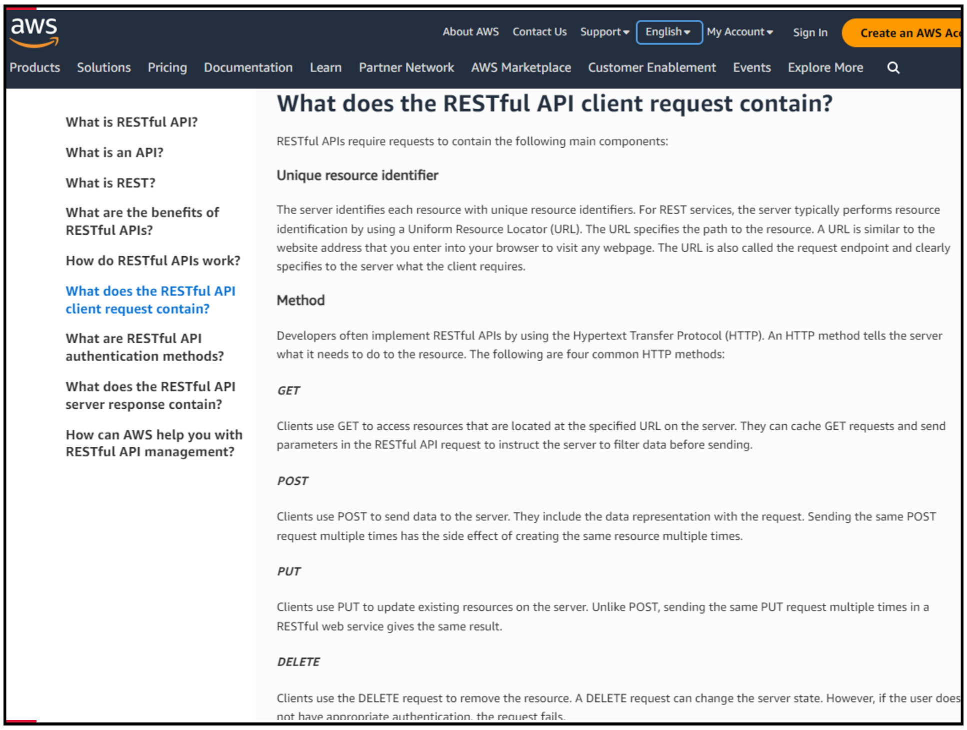
```
POST /book?id=abc&price=200
GET /books?id=abc
PUT /book?id=abc&price=200
```

- Use HTTP headers
  `Content-Type` and `Accept`
  to specify data format for the resources.

- Use HTTP status code to indicate success/failure.

YORK U UNIVERSITÉ UNIVERSITY

68



What is RESTful API?
What is an API?
What is REST?
What are the benefits of RESTful APIs?
How do RESTful APIs work?
What does the RESTful API client request contain?
What are RESTful API authentication methods?
What does the RESTful API server response contain?
How can AWS help you with RESTful API management?

What does the RESTful API client request contain?

RESTful APIs require requests to contain the following main components:

Unique resource identifier

The server identifies each resource with unique resource identifiers. For REST services, the server typically performs resource identification by using a Uniform Resource Locator (URL). The URL specifies the path to the resource. A URL is similar to the website address that you enter into your browser to visit any webpage. The URL is also called the request endpoint and clearly specifies to the server what the client requires.

Method

Developers often implement RESTful APIs by using the Hypertext Transfer Protocol (HTTP). An HTTP method tells the server what it needs to do to the resource. The following are four common HTTP methods:

GET

Clients use GET to access resources that are located at the specified URL on the server. They can cache GET requests and send parameters in the RESTful API request to instruct the server to filter data before sending.

POST

Clients use POST to send data to the server. They include the data representation with the request. Sending the same POST request multiple times has the side effect of creating the same resource multiple times.

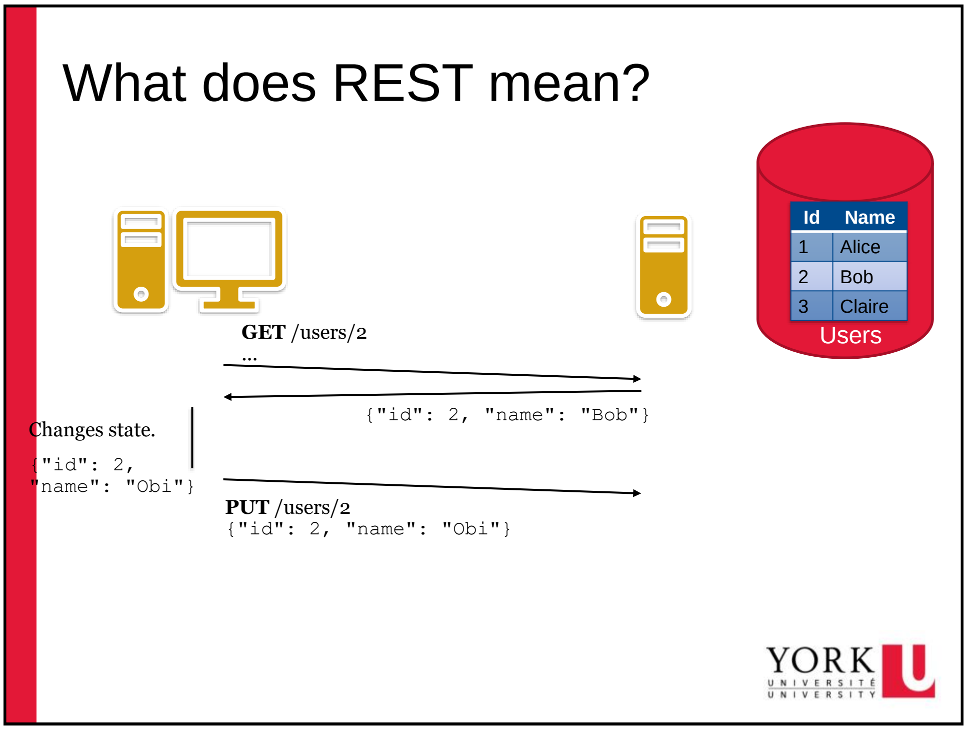
PUT

Clients use PUT to update existing resources on the server. Unlike POST, sending the same PUT request multiple times in a RESTful web service gives the same result.

DELETE

Clients use the DELETE request to remove the resource. A DELETE request can change the server state. However, if the user does not have appropriate authentication, the request fails.

69



What does REST mean?

GET /users/2
...

```
{"id": 2, "name": "Bob"}
```

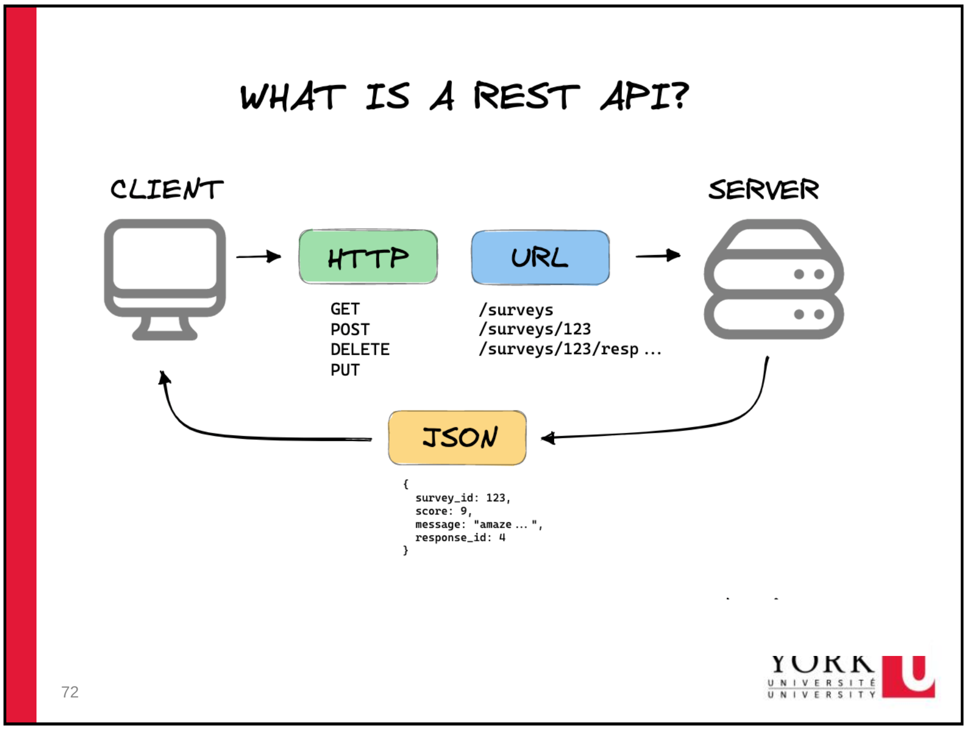
Changes state.

```
{"id": 2,
"name": "Obi"}
```

PUT /users/2

```
{"id": 2, "name": "Obi"}
```

70



WHAT IS A REST API?

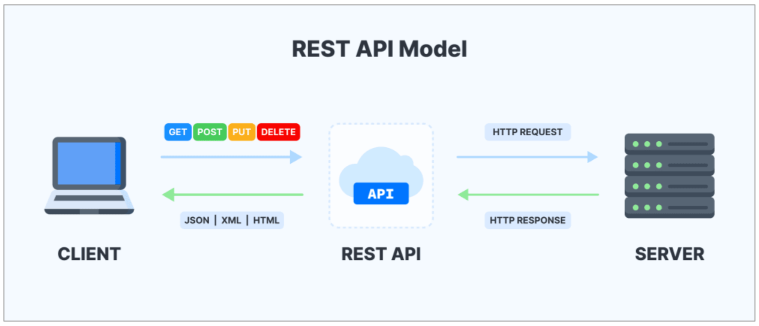
GET
POST
DELETE
PUT

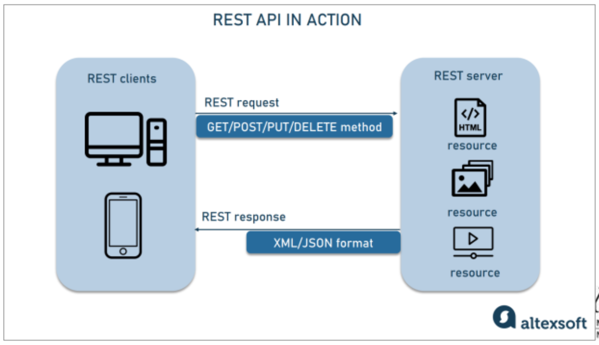
/surveys
/surveys/123
/surveys/123/resp ...

```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```

72

72

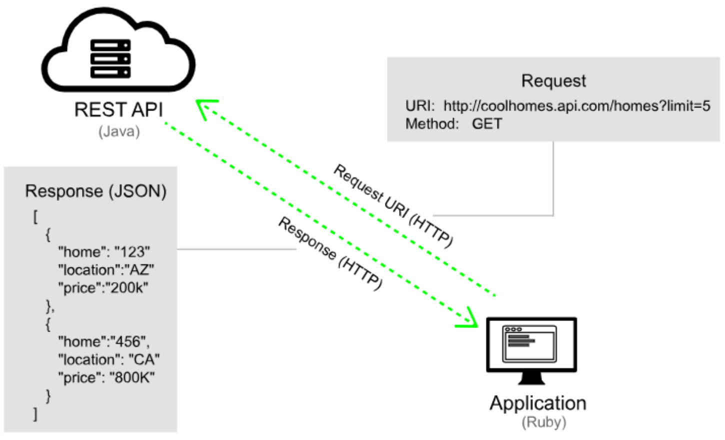




73

73

REST API model



Request
URI: http://coolhomes.api.com/homes?limit=5
Method: GET

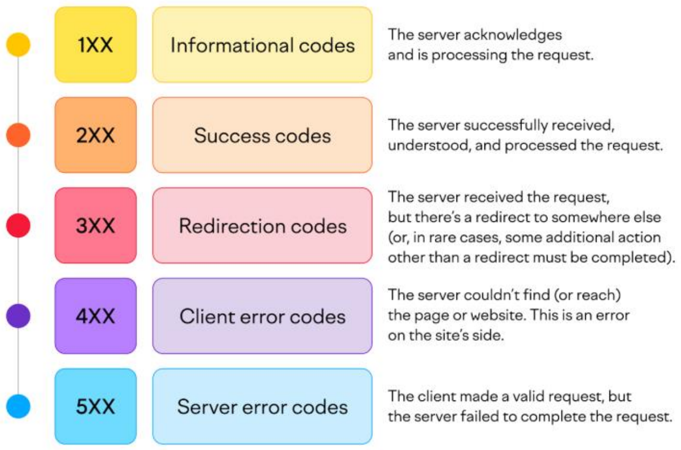
Response (JSON)

```
[
  {
    "home": "123"
    "location":"AZ"
    "price":"200k"
  },
  {
    "home":"456",
    "location": "CA"
    "price": "800K"
  }
]
```

74

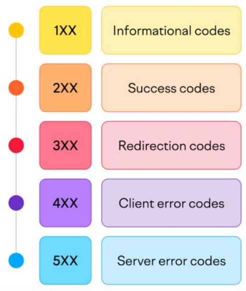
74

HTTP status code (recap)

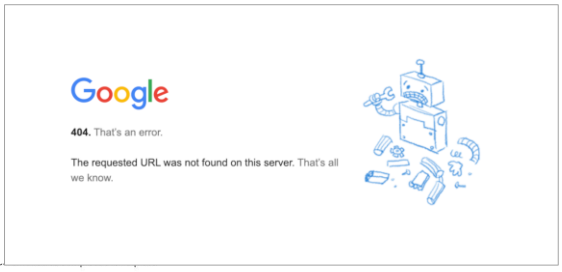


1XX	Informational codes	The server acknowledges and is processing the request.
2XX	Success codes	The server successfully received, understood, and processed the request.
3XX	Redirection codes	The server received the request, but there's a redirect to somewhere else (or, in rare cases, some additional action other than a redirect must be completed).
4XX	Client error codes	The server couldn't find (or reach) the page or website. This is an error on the site's side.
5XX	Server error codes	The client made a valid request, but the server failed to complete the request.

75



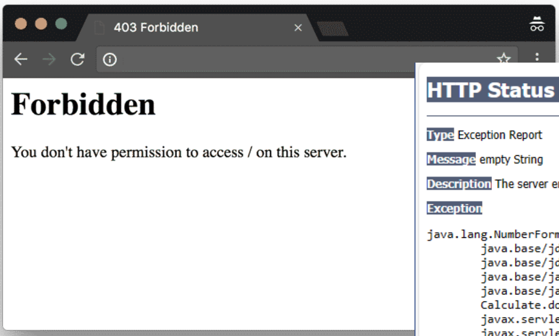
1XX	Informational codes
2XX	Success codes
3XX	Redirection codes
4XX	Client error codes
5XX	Server error codes



Google

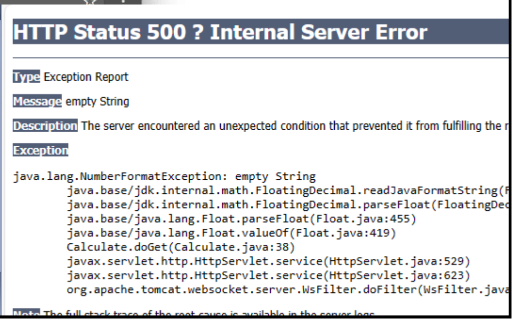
404. That's an error.

The requested URL was not found on this server. That's all we know.



Forbidden

You don't have permission to access / on this server.



HTTP Status 500 ? Internal Server Error

Type Exception Report

Message empty String

Description The server encountered an unexpected condition that prevented it from fulfilling the r

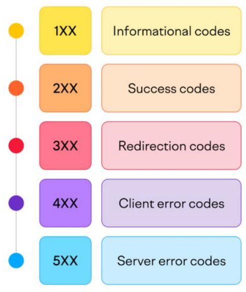
Exception

```
java.lang.NumberFormatException: empty String
        java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(F
        java.base/jdk.internal.math.FloatingDecimal.parseFloat(FloatingDec
        java.base/java.lang.Float.parseFloat(Float.java:455)
        java.base/java.lang.Float.valueOf(Float.java:419)
        Calculate.doGet(Calculate.java:38)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:529)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:623)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java
```

Note The full stack trace of the root cause is available in the server logs.

76

76



Successful responses

`200 OK`

The request succeeded. The result meaning of "success" depends on the HTTP method:

- `GET` : The resource has been fetched and transmitted in the message body.
- `HEAD` : The representation headers are included in the response without any message body.
- `PUT` or `POST` : The resource describing the result of the action is transmitted in the message body.

`201 Created`

The request succeeded, and a new resource was created as a result. This is typically the response sent after `POST` requests, or some `PUT` requests.

`202 Accepted`

The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

`204 No Content`

There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

specific case, the `200 OK` response is preferred to this status.

77

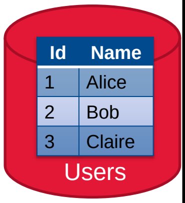
77

REST example

A server with information about users.

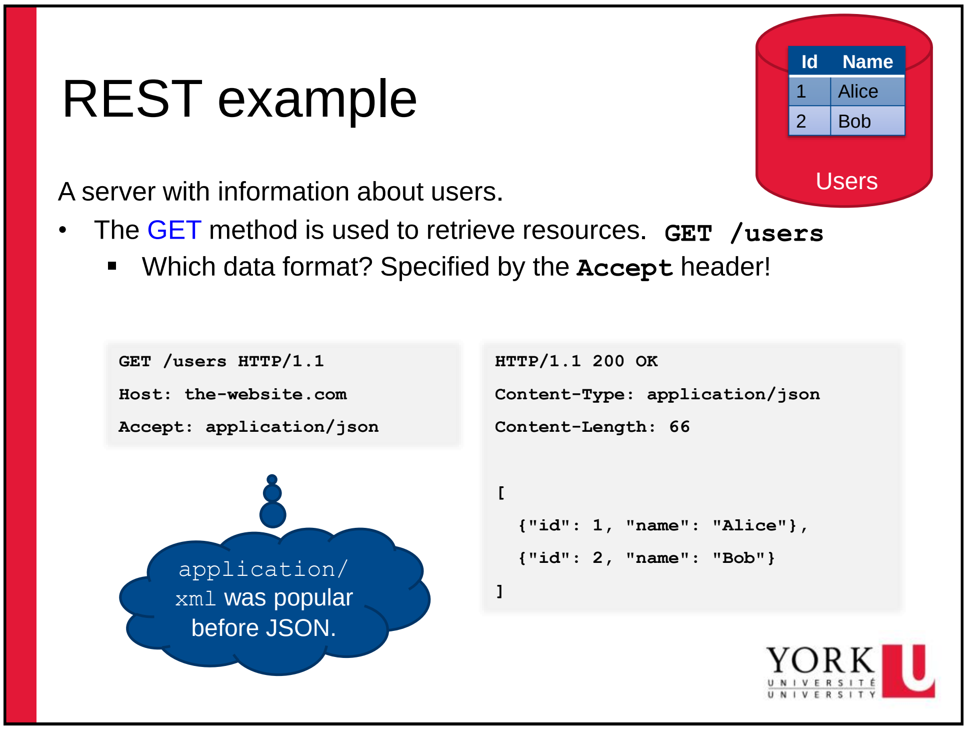
- The GET method is used to retrieve resources.

```
GET /users
GET /users/2
GET /users/pages/1
GET /users/gender/female
GET /users/age/18
GET /users/???
GET /users/2/name
GET /users/2/pets
```



Id	Name
1	Alice
2	Bob
3	Claire

78



REST example

Id	Name
1	Alice
2	Bob

Users

A server with information about users.

- The GET method is used to retrieve resources. **`GET /users`**
  - Which data format? Specified by the **`Accept`** header!

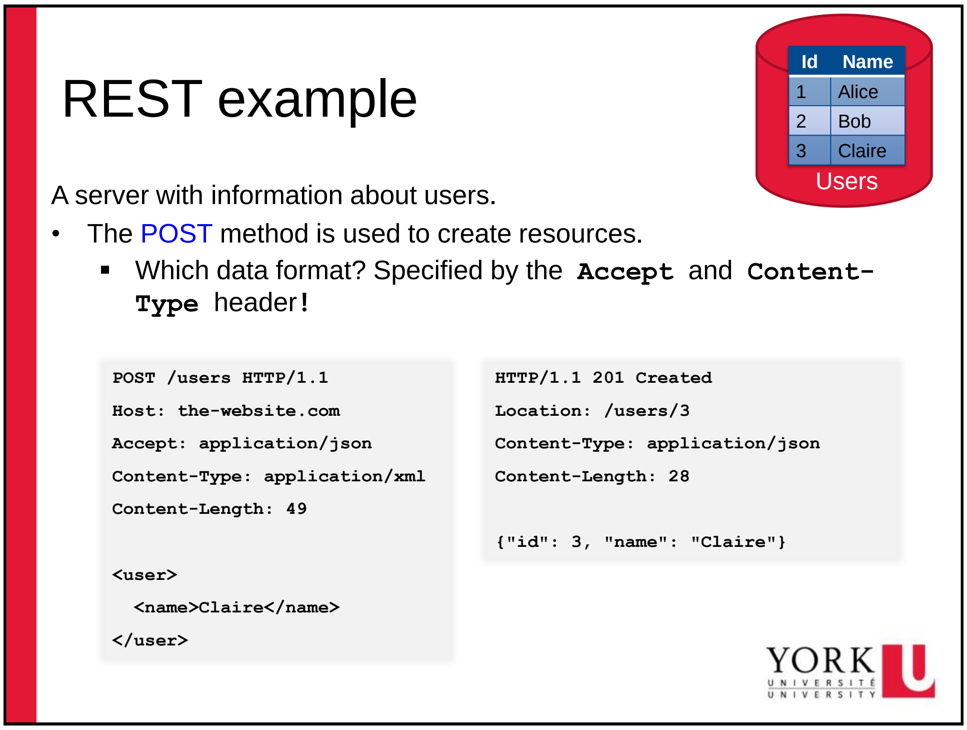
```
GET /users HTTP/1.1
Host: the-website.com
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 66

[
  {"id": 1, "name": "Alice"},
  {"id": 2, "name": "Bob"}
]
```

`application/xml` was popular before JSON.

YORK U UNIVERSITÉ UNIVERSITY

80



REST example

Id	Name
1	Alice
2	Bob
3	Claire

Users

A server with information about users.

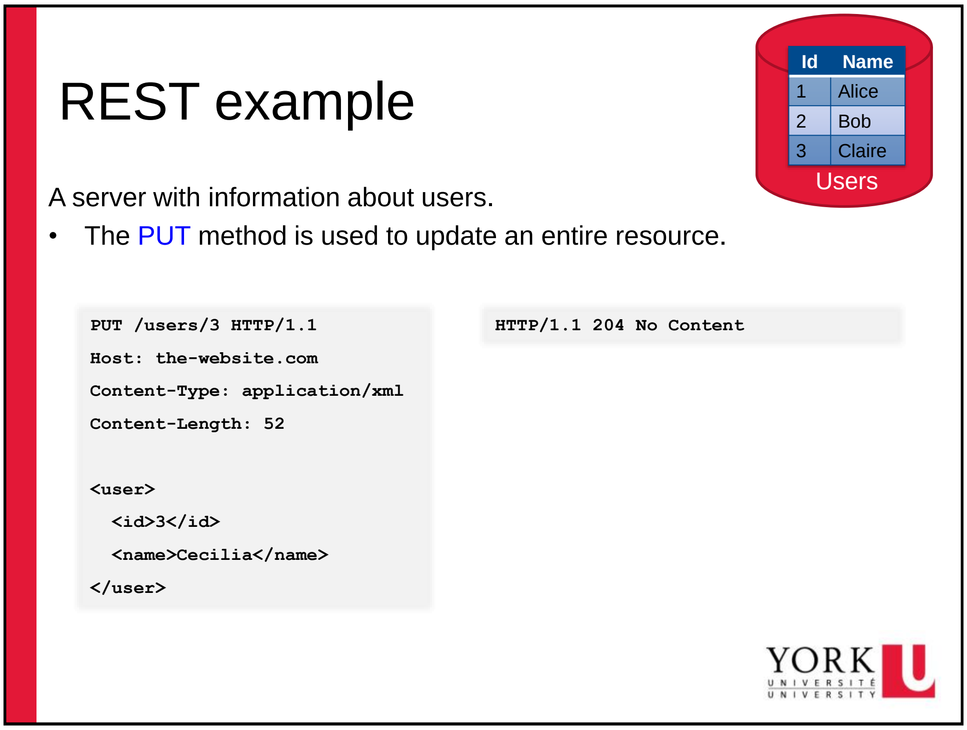
- The POST method is used to create resources.
  - Which data format? Specified by the **`Accept`** and **`Content-Type`** header**!**

```
POST /users HTTP/1.1
Host: the-website.com
Accept: application/json
Content-Type: application/xml
Content-Length: 49

<user>
  <name>Claire</name>
</user>
```

```
HTTP/1.1 201 Created
Location: /users/3
Content-Type: application/json
Content-Length: 28

{"id": 3, "name": "Claire"}
```

YORK U UNIVERSITÉ UNIVERSITY

81



REST example

Id	Name
1	Alice
2	Bob
3	Claire

Users

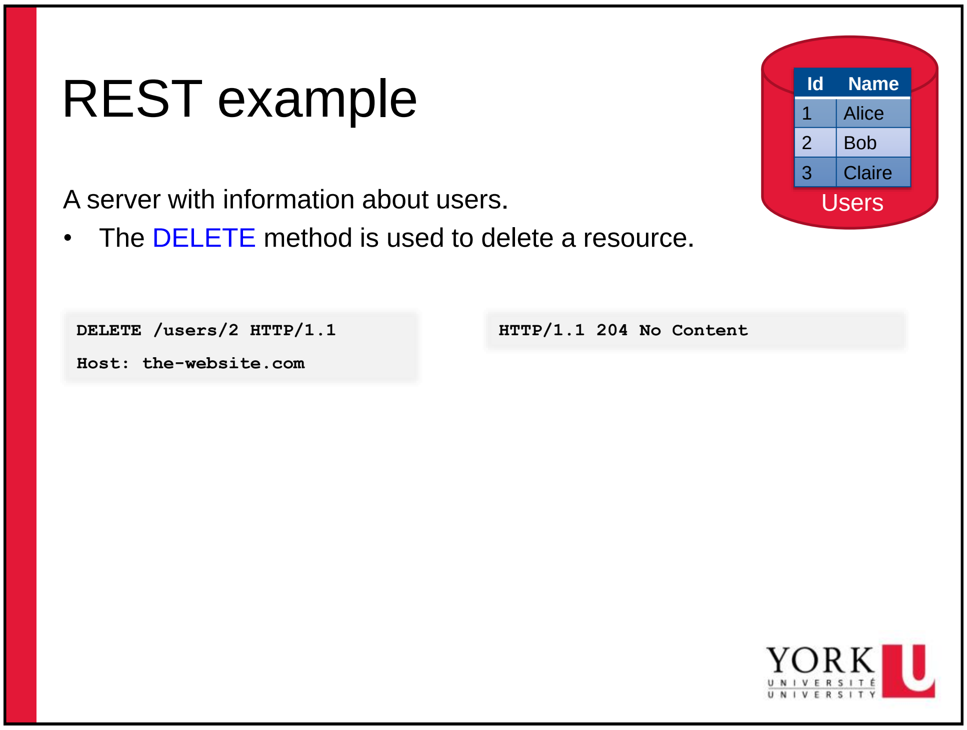
A server with information about users.

- The PUT method is used to update an entire resource.

```
PUT /users/3 HTTP/1.1
Host: the-website.com
Content-Type: application/xml
Content-Length: 52

<user>
  <id>3</id>
  <name>Cecilia</name>
</user>
```

```
HTTP/1.1 204 No Content
```

82



REST example

Id	Name
1	Alice
2	Bob
3	Claire

Users

A server with information about users.

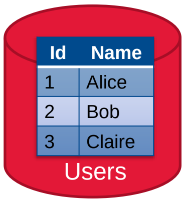
- The DELETE method is used to delete a resource.

```
DELETE /users/2 HTTP/1.1
Host: the-website.com
```

```
HTTP/1.1 204 No Content
```

83

REST example



Id	Name
1	Alice
2	Bob
3	Claire

A server with information about users.

- The PATCH method is used to update parts of a resource.

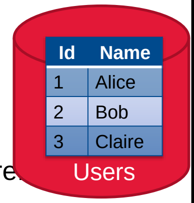
```
PATCH /users/1 HTTP/1.1
Host: the-website.com
Content-Type: application/xml
Content-Length: 37

<user>
  <name>Amanda</human>
</user>
```

```
HTTP/1.1 204 No Content
```

The PATCH method is only a proposed standard.

84

REST example



Id	Name
1	Alice
2	Bob
3	Claire

A server with information about users.

- What if something goes wrong?
  - Use the HTTP status codes to indicate success/failure.

```
GET /users/999 HTTP/1.1
Host: the-website.com
Accept: application/json
```

```
HTTP/1.1 404 Not Found
```

85

Designing a REST api

How should you think?

```
Make it as easy as possible to use by other programmers.
```

Twitter:

```
Only use GET and POST.
GET  /1.1/users/show.json?user_id=2244994945
POST /1.1/favorites/destroy.json?id=2431381289599
```

Facebook:

```
Always return 200 OK.
GET /v2.7/{user-id}
GET /v2.7/{post-id}
GET /v2.7/{user-id}/friends
GET /v2.7/{object-id}/likes
```

87

Java implementation of REST

JAX-RS:
Java API for RESTful Web Services (JAX-RS), is a set of APIs to developer REST service. JAX-RS is part of the Java EE6, and make developers to develop REST web application easily.

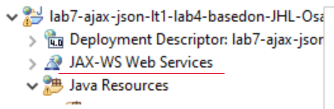
Jersey:
Jersey is the open source, production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services. But, it is also more than the Reference Implementation. Jersey provides an API so that developers may extend Jersey to suit their needs.

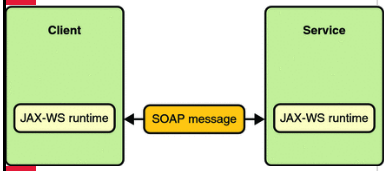
- extend/based on servlets

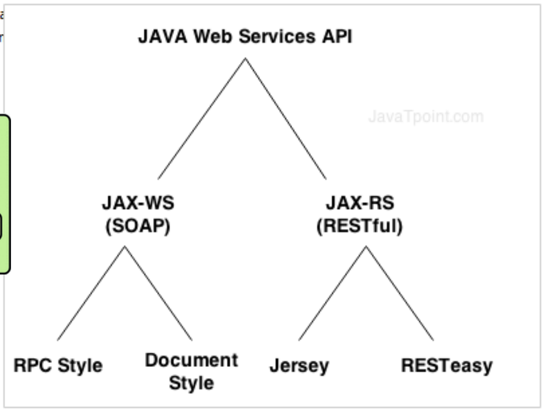
88

88

Java implementation of REST


lab7-ajax-json-lt1-lab4-basedon-JHL-Osa
Deployment Descriptor: lab7-ajax-jsor
JAX-WS Web Services
Java Resources





89

89

- **JAX-RS** stands for JAVA API for RESTful Web Services. JAX-RS is a JAVA based programming language API and specification to provide support for created RESTful Web Services.
- **JAX-RS** uses annotations available from Java SE 5 to simplify the development of JAVA based web services creation and deployment. It also provides supports for creating clients for RESTful Web Services.

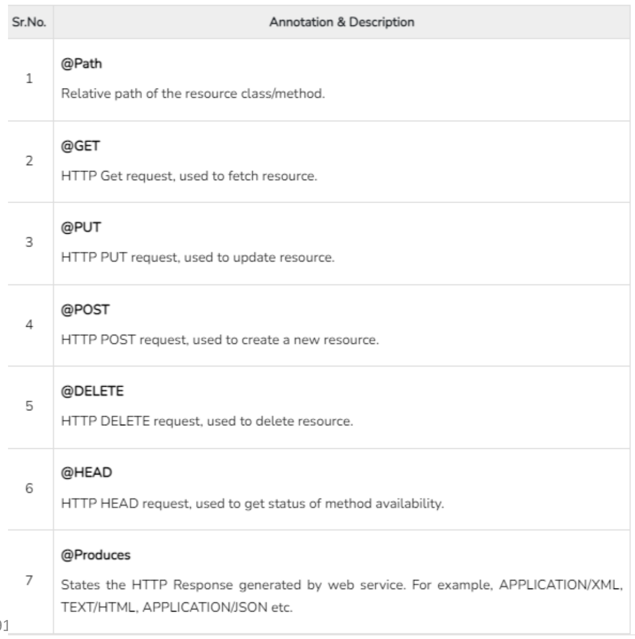
  - The **@Path** Annotation
  - The **@GET** HTTP Method Annotation
  - The @**POST** HTTP Method Annotation
  - The @**PUT** HTTP Method Annotation
  - The @**DELETE** HTTP Method Annotation
  - The @**Produces** Annotation
  - The @**Consumes** Annotation
  - Parameter Annotation - @**PathParam**
  - The @**QueryParam** Annotation
  - The **@FormParam** Annotation

  - The @MatrixParam Annotation
  - The @CookieParam Annotation
  - The @HeaderParam Annotation
  - The @Provider Annotation
  - The @OPTIONS HTTP Method Annotation
  - The @HEAD HTTP Method Annotation

90

90

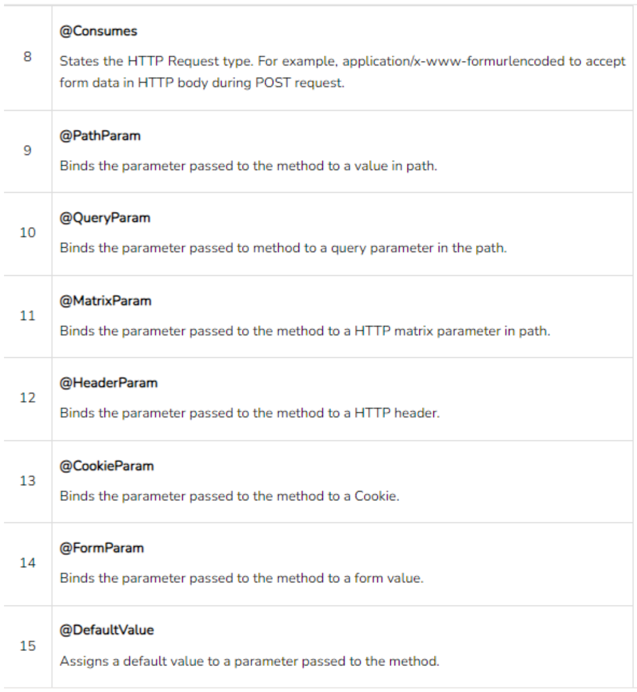
Following are the most commonly used annotations to map a resource as a web service resource.



Sr.No.	Annotation & Description
1	@Path Relative path of the resource class/method.
2	@GET HTTP Get request, used to fetch resource.
3	@PUT HTTP PUT request, used to update resource.
4	@POST HTTP POST request, used to create a new resource.
5	@DELETE HTTP DELETE request, used to delete resource.
6	@HEAD HTTP HEAD request, used to get status of method availability.
7	@Produces States the HTTP Response generated by web service. For example, APPLICATION/XML, TEXT/HTML, APPLICATION/JSON etc.

91

91



8	@Consumes States the HTTP Request type. For example, application/x-www-formurlencoded to accept form data in HTTP body during POST request.
9	@PathParam Binds the parameter passed to the method to a value in path.
10	@QueryParam Binds the parameter passed to method to a query parameter in the path.
11	@MatrixParam Binds the parameter passed to the method to a HTTP matrix parameter in path.
12	@HeaderParam Binds the parameter passed to the method to a HTTP header.
13	@CookieParam Binds the parameter passed to the method to a Cookie.
14	@FormParam Binds the parameter passed to the method to a form value.
15	@DefaultValue Assigns a default value to a parameter passed to the method.

92

92

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("hello")
public class HelloWorldResource {

    @GET
    @Produces("text/plain")
     public String getHello() {
          return "Hello World!";
     }
}
```

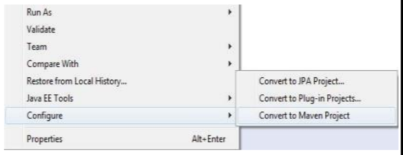
93

93

Set up

- Add jersey jars to project lib,
- Or, (create or), convert to Maven project, add dependencies on **pom.xml**
    - Maven will download jars for you

```
<dependencies>
 <dependency>
   <groupId>org.glassfish.jersey.containers</groupId>
   <artifactId>jersey-container-servlet</artifactId>
   <version>2.40</version>
 </dependency>

  <dependency>
   <groupId>org.glassfish.jersey.inject</groupId>
   <artifactId>jersey-hk2</artifactId>
   <version>2.40</version>
 </dependency>
```



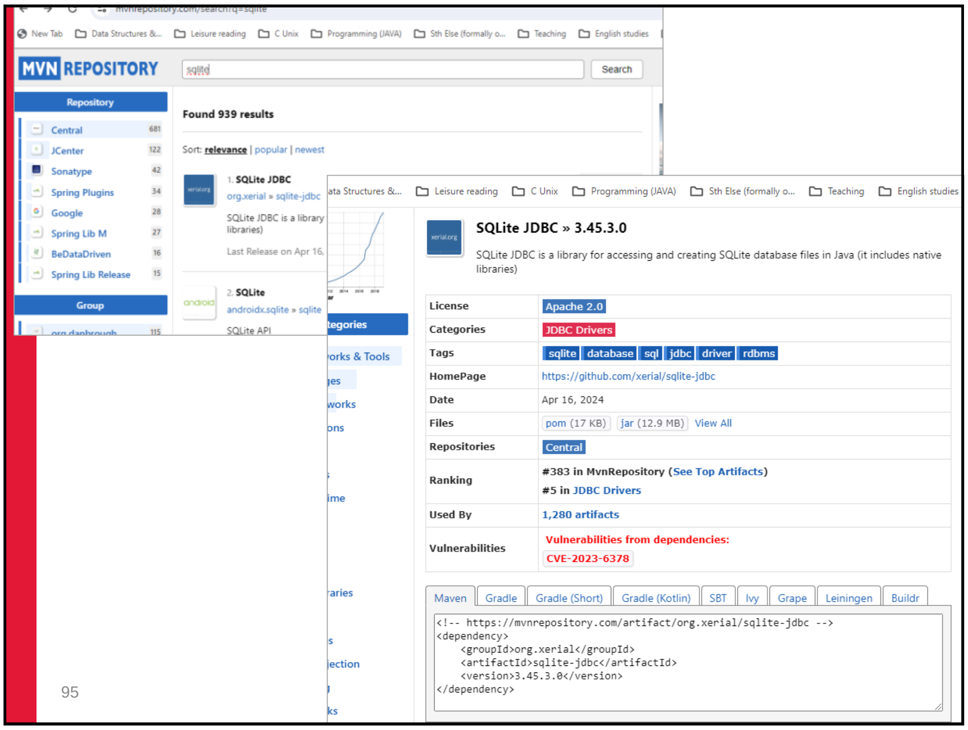
- Web.xml -- Add servlet dispatcher

```
<servlet>
   <servlet-name>Jersey REST Service</servlet-name>
   <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
       <param-name>jersey.config.server.provider.packages</param-name>
       <param-value>restService</param-value>
   </init-param>

 </servlet>
 <servlet-mapping>
   <servlet-name>Jersey REST Service</servlet-name>
   <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
```

Package name

94

94



mvnrepository.com/search?q=sqlite

MVN REPOSITORY sqlite Search

Repository

Central	681
JCenter	122
Sonatype	42
Spring Plugins	34
Google	28
Spring Lib M	27
BeDataDriven	16
Spring Lib Release	15

Group

Found 939 results

Sort: **relevance** | popular | newest

1. **SQLite JDBC**
org.xerial » sqlite-jdbc
SQLite JDBC is a library
libraries)
Last Release on Apr 16,

2. **SQLite**
androidx.sqlite » sqlite
SQLite API

SQLite JDBC » 3.45.3.0

SQLite JDBC is a library for accessing and creating SQLite database files in Java (it includes native libraries)

License	Apache 2.0
Categories	JDBC Drivers
Tags	sqlite database sql jdbc driver rdbms
HomePage	https://github.com/xerial/sqlite-jdbc
Date	Apr 16, 2024
Files	pom (17 KB) jar (12.9 MB) View All
Repositories	Central
Ranking	#383 in MvnRepository (See Top Artifacts) #5 in JDBC Drivers
Used By	1,280 artifacts
Vulnerabilities	Vulnerabilities from dependencies: CVE-2023-6378

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.45.3.0</version>
</dependency>
```

95

95

```java
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;


@Path("/helloM")
public class HelloM {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello() {
    return "HelloM Jersey Plain";
  }
  // This method is called if XML is request
  @GET
  @Produces(MediaType.TEXT_XML)
  public String sayXMLHello() {
    return "<?xml version=\"1.0\"?>" + "<hello> HelloM Jersey xml" +
    "</hello>";
  }

  // This method is called if HTML is request
   @GET
   @Produces(MediaType.TEXT_HTML)
   public String sayHtmlHello() {
     return "<html> " + "<title>" + "HelloM Jersey"
     + "</title>" + "<body><h1>" + "HelloM Jersey HTML" +
     "</h1></body>" + "</html> ";
```

curl localhost:8080/proj/rest/helloM -H "Accept: text/html"
<html> <title>Hello Jersey</title><body><h1>HelloM Jersey HTML</h1></body></html>

curl localhost:8080/proj/rest/helloM -H "Accept: text/plain"
HelloM Jersey Plain

curl localhost:8080/proj/rest/helloM -H "Accept: text/xml"
<?xml version="1.0"?><hello> HelloM Jersey xml</hello>

96

96

An example. Communicate with JSON

- Add a dependency in pom.xml
  - Maven will download jars for you

```
<dependencies>
 <dependency>
   <groupId>org.glassfish.jersey.containers</groupId>
   <artifactId>jersey-container-servlet</artifactId>
   <version>2.40</version>
 </dependency>

  <dependency>
   <groupId>org.glassfish.jersey.inject</groupId>
   <artifactId>jersey-hk2</artifactId>
   <version>2.40</version>
 </dependency>

 <dependency>
   <groupId>org.glassfish.jersey.media</groupId>
   <artifactId> jersey-media-json-jackson </artifactId>
   <version>2.40</version>
 </dependency>

</dependencies>
```

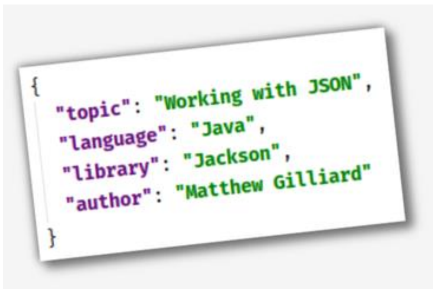
YORK UNIVERSITÉ UNIVERSITY

98

JSON

xml (eXtensible Markup Language)
a syntax to store and transport data

what is JSON?
JavaScript Object Notation
a simpler syntax to store and transport data

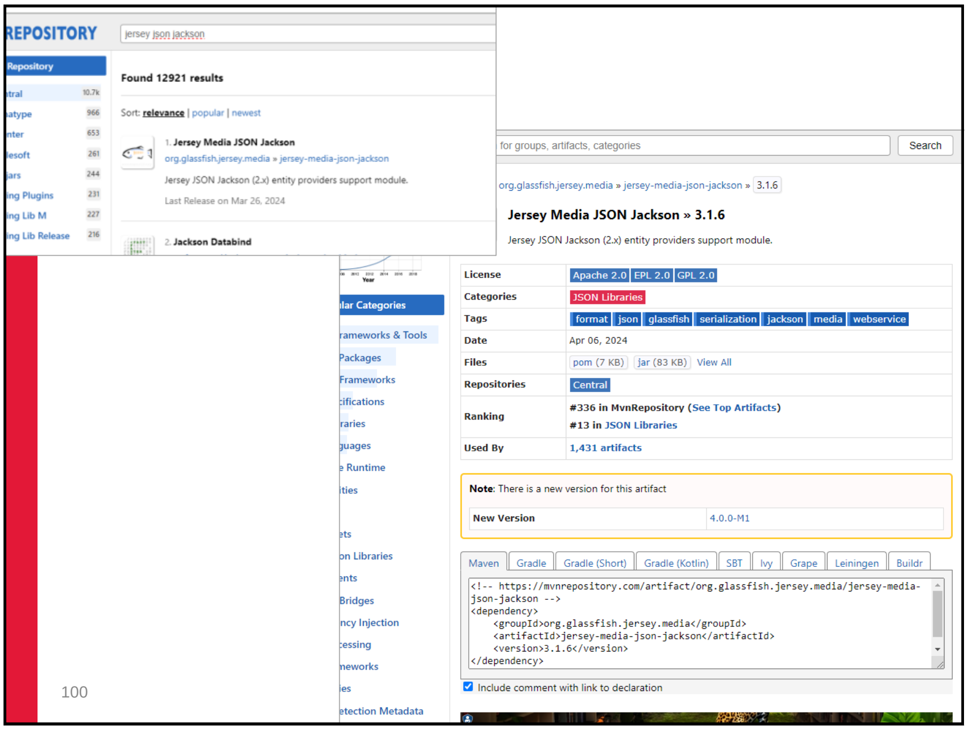


```
{
  "topic": "Working with JSON",
  "language": "Java",
  "library": "Jackson",
  "author": "Matthew Gilliard"
}
```

{"topic":"Working", "language":"Java", "library":"Jackson", "author":"Matthew Gilliard"}

1-99

99



REPOSITORY

jersey json jackson

Found 12921 results

Sort: **relevance** | popular | newest

1. **Jersey Media JSON Jackson**
org.glassfish.jersey.media » jersey-media-json-jackson
Jersey JSON Jackson (2.x) entity providers support module.
Last Release on Mar 26, 2024

2. **Jackson Databind**

org.glassfish.jersey.media » jersey-media-json-jackson » 3.1.6

Jersey Media JSON Jackson » 3.1.6

Jersey JSON Jackson (2.x) entity providers support module.

License	Apache 2.0 EPL 2.0 GPL 2.0
Categories	JSON Libraries
Tags	format json glassfish serialization jackson media webservice
Date	Apr 06, 2024
Files	pom (7 KB) jar (83 KB) View All
Repositories	Central
Ranking	#336 in MvnRepository (See Top Artifacts) #13 in JSON Libraries
Used By	1,431 artifacts

Note: There is a new version for this artifact

New Version	4.0.0-M1

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.glassfish.jersey.media/jersey-media-json-jackson -->
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>3.1.6</version>
</dependency>
```

Include comment with link to declaration

100

100

Project: PlantsRESTnew

Bean class

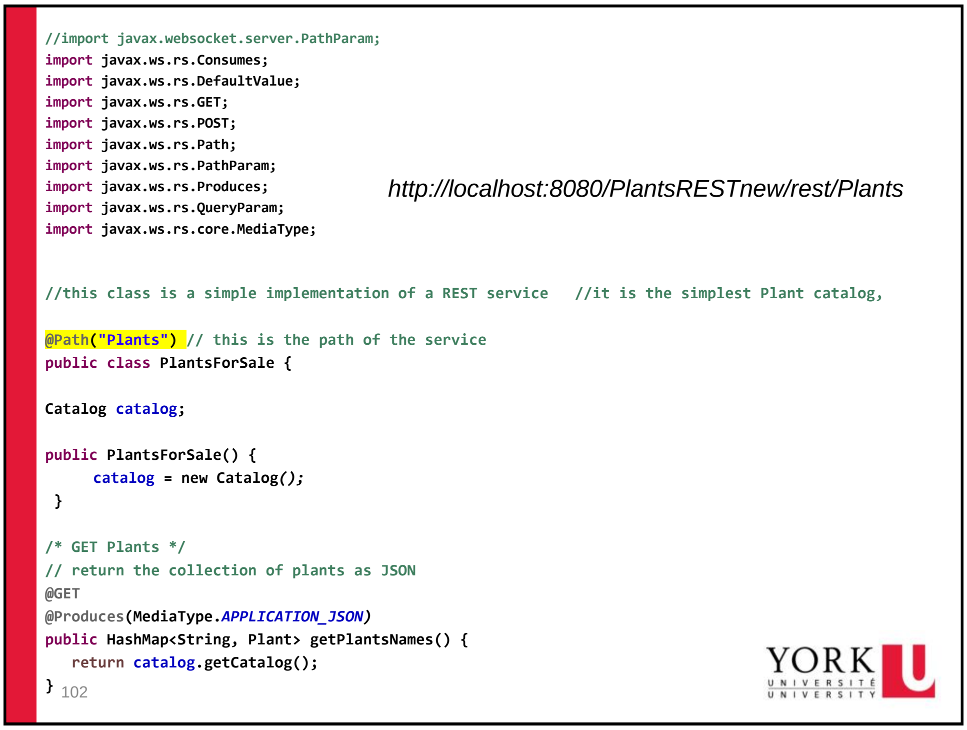
```
package restService;

public class Plant {
String name;
double price;
String description;

public Plant(String name, double price, String desc) {
    this.name = name;
    this.price=price;
    this.description=des;
}
public String getName() {
     return name;
}

public void setName(String nam
     this.name = name;
}
```

"DAO"

```
package restService;

import java.util.HashMap;

public class Catalog {

  public static Catalog instance;
  Static HashMap<String, Plant> catalog;
  Static {
      catalog=new HashMap<String, Plant>();
      catalog.put("rose", new Plant("rose", 10.9, "Most popular"));
      catalog.put("tulip", new Plant("tulip", 5.0,  "Discounted"));
      catalog.put("lily", new Plant("lily", 5.0,   "Available in Spring"));
   }

  public Catalog () {}

  public void addPlant (String id, String name, double price, String d) {
       this.catalog.put(id, new Plant(name, price, d));
  }

  public Plant getPlant (String id) {
      return catalog.get(id));
  }
  public HashMap<String, Plant> getCatalog() {  // get all plants
      return  this.catalog;
  }
```

101

101

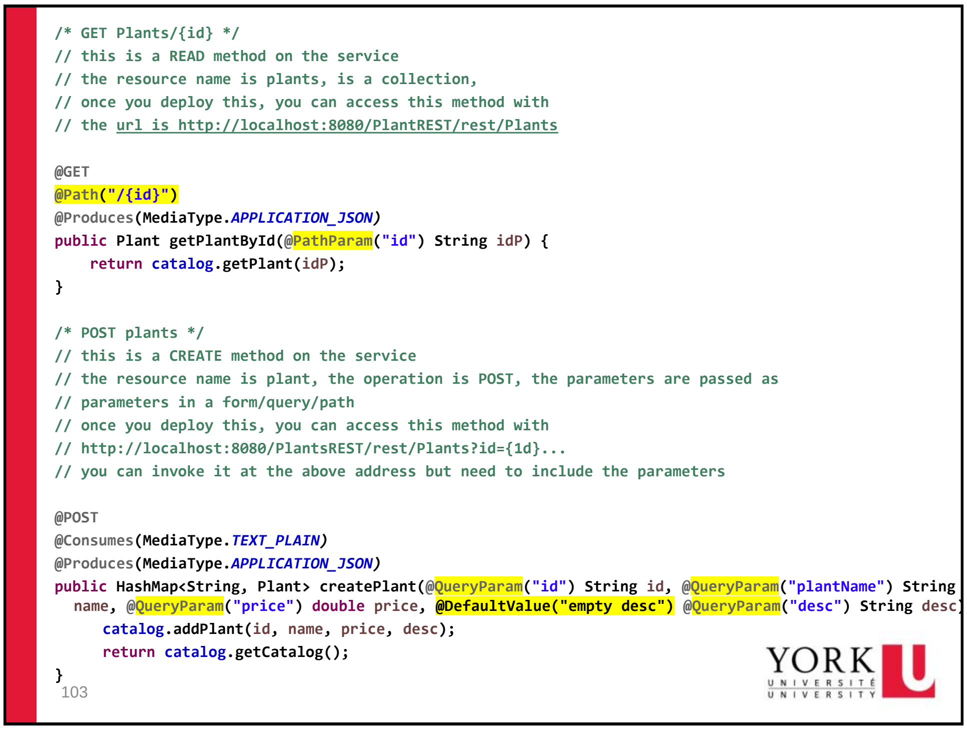


```
//import javax.websocket.server.PathParam;
import javax.ws.rs.Consumes;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;


//this class is a simple implementation of a REST service   //it is the simplest Plant catalog,

@Path("Plants") // this is the path of the service
public class PlantsForSale {

Catalog catalog;

public PlantsForSale() {
     catalog = new Catalog();
 }

/* GET Plants */
// return the collection of plants as JSON
@GET
@Produces(MediaType.APPLICATION_JSON)
public HashMap<String, Plant> getPlantsNames() {
   return catalog.getCatalog();
}
```

http://localhost:8080/PlantsRESTnew/rest/Plants

102

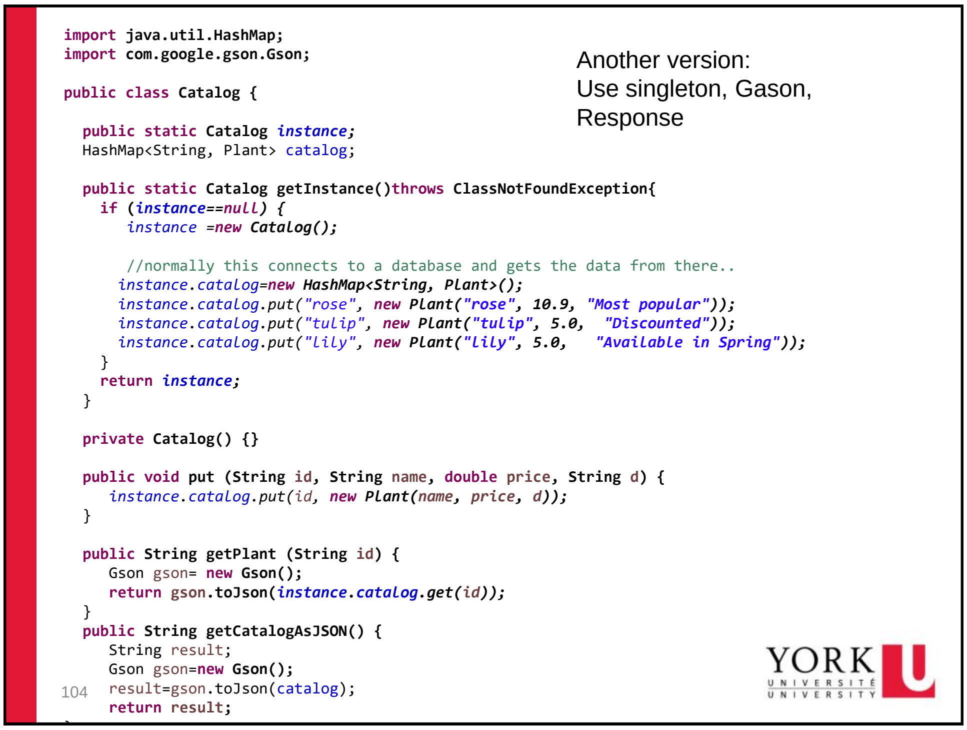
102



```
/* GET Plants/{id} */
// this is a READ method on the service
// the resource name is plants, is a collection,
// once you deploy this, you can access this method with
// the url is http://localhost:8080/PlantREST/rest/Plants

@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Plant getPlantById(@PathParam("id") String idP) {
    return catalog.getPlant(idP);
}

/* POST plants */
// this is a CREATE method on the service
// the resource name is plant, the operation is POST, the parameters are passed as
// parameters in a form/query/path
// once you deploy this, you can access this method with
// http://localhost:8080/PlantsREST/rest/Plants?id={1d}...
// you can invoke it at the above address but need to include the parameters

@POST
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.APPLICATION_JSON)
public HashMap<String, Plant> createPlant(@QueryParam("id") String id, @QueryParam("plantName") String
  name, @QueryParam("price") double price, @DefaultValue("empty desc") @QueryParam("desc") String desc
     catalog.addPlant(id, name, price, desc);
     return catalog.getCatalog();
}
```

103

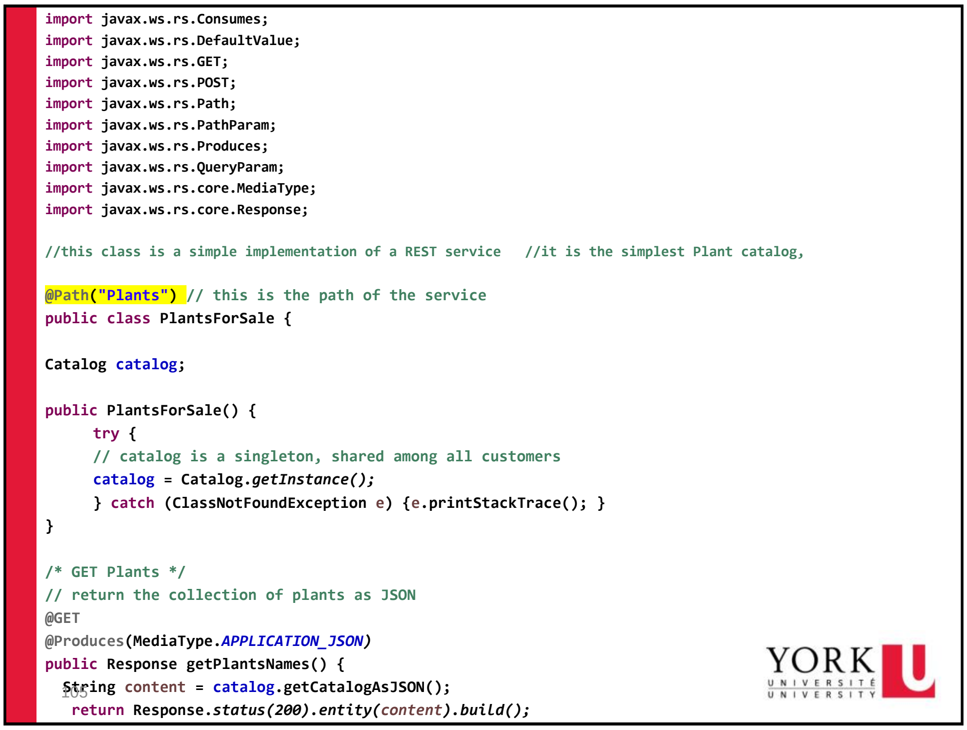
103



Another version:
Use singleton, Gason, Response

```
import java.util.HashMap;
import com.google.gson.Gson;

public class Catalog {

  public static Catalog instance;
  HashMap<String, Plant> catalog;

  public static Catalog getInstance()throws ClassNotFoundException{
    if (instance==null) {
       instance =new Catalog();

       //normally this connects to a database and gets the data from there..
      instance.catalog=new HashMap<String, Plant>();
      instance.catalog.put("rose", new Plant("rose", 10.9, "Most popular"));
      instance.catalog.put("tulip", new Plant("tulip", 5.0,  "Discounted"));
      instance.catalog.put("lily", new Plant("lily", 5.0,   "Available in Spring"));
    }
    return instance;
  }

  private Catalog() {}

  public void put (String id, String name, double price, String d) {
     instance.catalog.put(id, new Plant(name, price, d));
  }

  public String getPlant (String id) {
     Gson gson= new Gson();
     return gson.toJson(instance.catalog.get(id));
  }
  public String getCatalogAsJSON() {
     String result;
     Gson gson=new Gson();
     result=gson.toJson(catalog);
     return result;
```

104



```
import javax.ws.rs.Consumes;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

//this class is a simple implementation of a REST service   //it is the simplest Plant catalog,

@Path("Plants") // this is the path of the service
public class PlantsForSale {

Catalog catalog;

public PlantsForSale() {
      try {
      // catalog is a singleton, shared among all customers
      catalog = Catalog.getInstance();
      } catch (ClassNotFoundException e) {e.printStackTrace(); }
}

/* GET Plants */
// return the collection of plants as JSON
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response getPlantsNames() {
  String content = catalog.getCatalogAsJSON();
   return Response.status(200).entity(content).build();
```

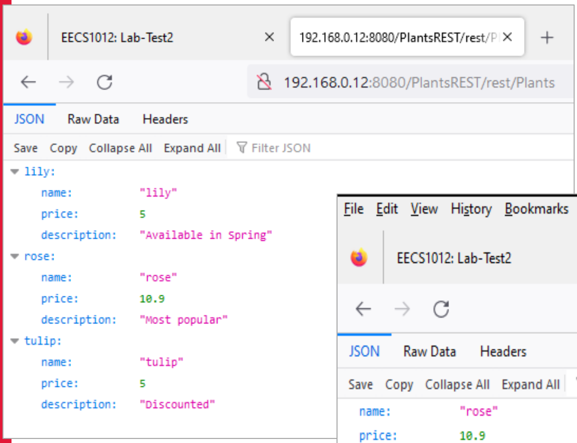
105

```java
/* GET Plants/{id} */
// this is a READ method on the service
// the resource name is plants, is a collection,
// once you deploy this, you can access this method with
// the url is http://localhost:8080/PlantREST/rest/Plants

@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Response getPrice(@PathParam("id") String id) {
    String content = catalog.getPlant(id);
    return Response.status(200).entity(content).build();
}

/* POST plants */
// this is a CREATE method on the service
// the resource name is plant, the operation is POST, the parameters are passed as
// parameters in a form/query/path
// once you deploy this, you can access this method with
// http://localhost:8080/PlantsREST/rest/Plants?id={1d}...
// you can invoke it at the above address but need to include the parameters

@POST
@Consumes(MediaType.TEXT_PLAIN)
public Response createPlant(@QueryParam("id") String id, @QueryParam("plantName") String name,
    @QueryParam("price") double price, @DefaultValue("empty desc") @QueryParam("desc") String desc) {
     System.out.println("received:" + name + " " + price);
     catalog.addPlant(id, name, price, desc);
     String content = catalog.getCatalogAsJSON();
     return Response.status(200).entity(content).build();
```

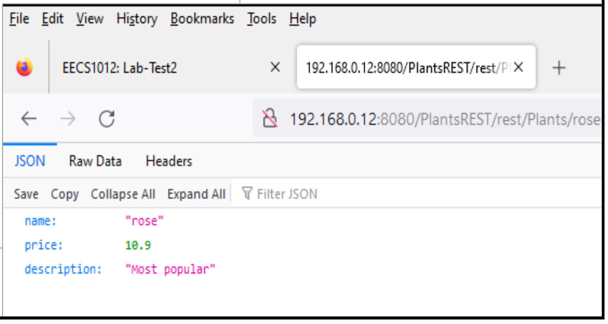
106

106

```
curl  http://localhost:8080/PlantsRESTnew/rest/Plants          or
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/rose
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/lily
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/tulip
curl -X POST
   'http://localhost:8080/PlantsRESTnew/rest/Plants?id=green&plantName=green&price=2.3&desc=XXX'
```



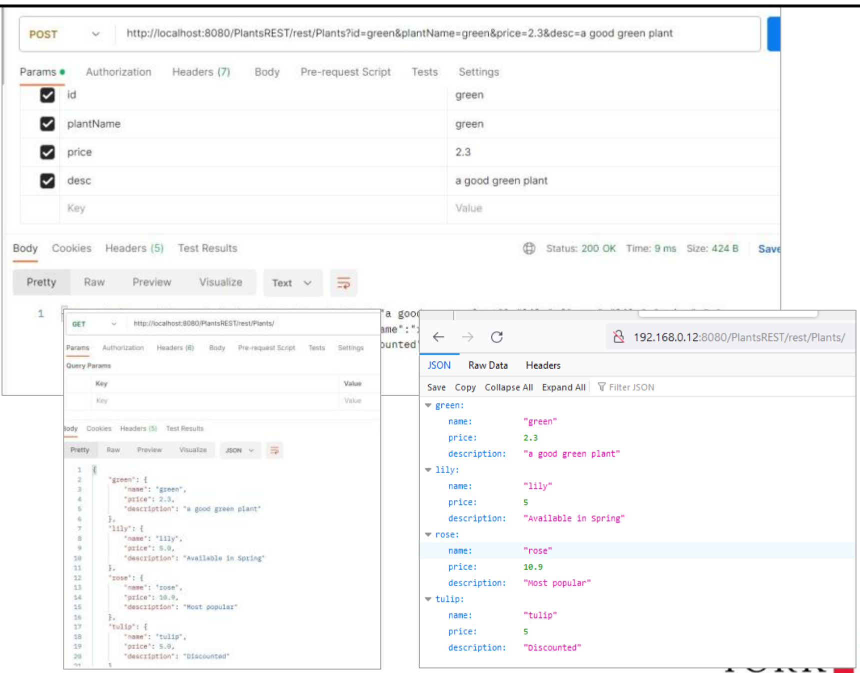
```
▾ lily:
    name:          "lily"
    price:         5
    description:   "Available in Spring"
▾ rose:
    name:          "rose"
    price:         10.9
    description:   "Most popular"
▾ tulip:
    name:          "tulip"
    price:         5
    description:   "Discounted"
```


File Edit View History Bookmarks Tools Help
EECS1012: Lab-Test2
192.168.0.12:8080/PlantsREST/rest/P
192.168.0.12:8080/PlantsREST/rest/Plants/rose
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
name:          "rose"
price:         10.9
description:   "Most popular"
```

107

107



```
POST   http://localhost:8080/PlantsREST/rest/Plants?id=green&plantName=green&price=2.3&desc=a good green plant
```

Params ● | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings

✓ id	green
✓ plantName	green
✓ price	2.3
✓ desc	a good green plant
Key	Value

Body | Cookies | Headers (5) | Test Results — Status: 200 OK Time: 9 ms Size: 424 B

Pretty | Raw | Preview | Visualize | Text

```
1  "a goo
   ame":"
   ounted
```

```
GET   http://localhost:8080/PlantsREST/rest/Plants/
```

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

Key	Value
Key	Value

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
{
    "green": {
        "name": "green",
        "price": 2.3,
        "description": "a good green plant"
    },
    "lily": {
        "name": "lily",
        "price": 5.0,
        "description": "Available in Spring"
    },
    "rose": {
        "name": "rose",
        "price": 10.9,
        "description": "Most popular"
    },
    "tulip": {
        "name": "tulip",
        "price": 5.0,
        "description": "Discounted"
    }
```

192.168.0.12:8080/PlantsREST/rest/Plants/

JSON | Raw Data | Headers

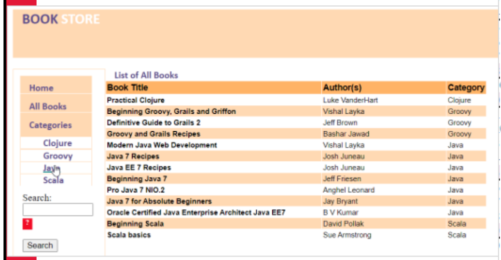
Save | Copy | Collapse All | Expand All | Filter JSON

```
▼ green:
    name:          "green"
    price:         2.3
    description:   "a good green plant"
▼ lily:
    name:          "lily"
    price:         5
    description:   "Available in Spring"
▼ rose:
    name:          "rose"
    price:         10.9
    description:   "Most popular"
▼ tulip:
    name:          "tulip"
    price:         5
    description:   "Discounted"
```

109

109

+ database



BOOK STORE

Home | All Books | Categories | Clojure | Groovy | Java | Scala

Search:

Search

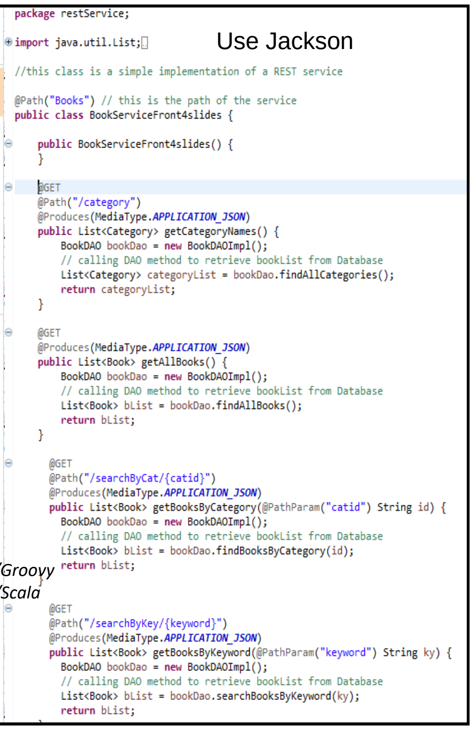
List of All Books

Book Title	Author(s)	Category
Practical Clojure	Luke VanderHart	Clojure
Beginning Groovy, Grails and Griffon	Vishal Layka	Groovy
Definitive Guide to Grails 2	Jeff Brown	Groovy
Groovy and Grails Recipes	Bashar Jawad	Groovy
Modern Java Web Development	Vishal Layka	Java
Java 7 Recipes	Josh Juneau	Java
Java EE 7 Recipes	Josh Juneau	Java
Beginning Java 7	Jeff Friesen	Java
Pro Java 7 NIO.2	Anghel Leonard	Java
Java 7 for Absolute Beginners	Jay Bryant	Java
Oracle Certified Java Enterprise Architect Java EE7	B V Kumar	Java
Beginning Scala	David Pollak	Scala
Scala basics	Sue Armstrong	Scala

```
<dependency>
        <groupId>org.xerial</groupId>
        <artifactId>sqlite-jdbc</artifactId>
        <version>3.34.0</version>
</dependency>
```

```
curl  http://localhost:8080/BooksREST/rest/Books
curl  http://localhost:8080/BooksREST/rest/Books/category
curl  http://localhost:8080/BooksREST/rest/Books/searchByCat/Groovy
curl  http://localhost:8080/BooksREST/rest/Books/searchByCat/Scala
```

Use Jackson

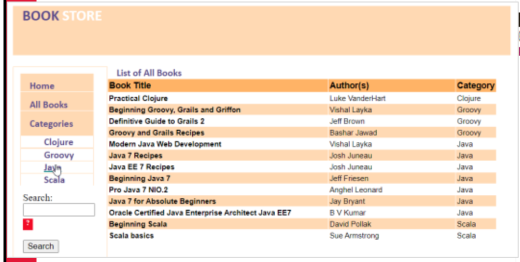


```
package restService;

import java.util.List;

//this class is a simple implementation of a REST service

@Path("Books") // this is the path of the service
public class BookServiceFront4slides {

    public BookServiceFront4slides() {
    }

    @GET
    @Path("/category")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Category> getCategoryNames() {
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Category> categoryList = bookDao.findAllCategories();
        return categoryList;
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Book> getAllBooks() {
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Book> bList = bookDao.findAllBooks();
        return bList;
    }

     @GET
     @Path("/searchByCat/{catid}")
     @Produces(MediaType.APPLICATION_JSON)
     public List<Book> getBooksByCategory(@PathParam("catid") String id) {
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Book> bList = bookDao.findBooksByCategory(id);
        return bList;
     }

     @GET
     @Path("/searchByKey/{keyword}")
     @Produces(MediaType.APPLICATION_JSON)
     public List<Book> getBooksByKeyword(@PathParam("keyword") String ky) {
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Book> bList = bookDao.searchBooksByKeyword(ky);
        return bList;
```

110

110

+ database


BOOK STORE

Home
All Books
Categories
Clojure
Groovy
Java
Scala

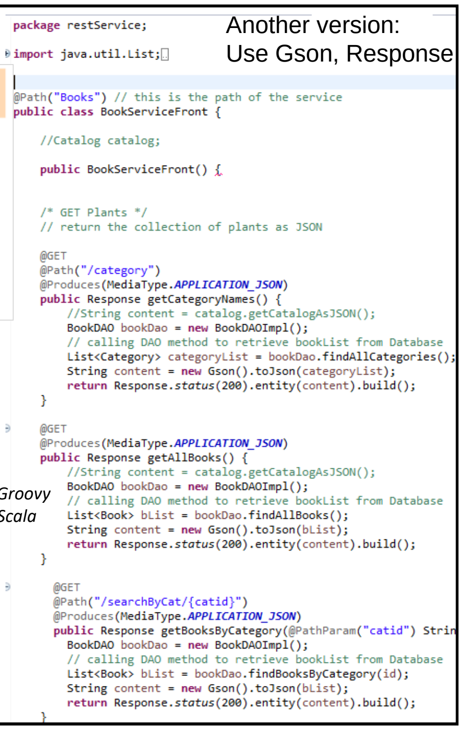
Search:

Search

List of All Books

Book Title	Author(s)	Category
Practical Clojure	Luke VanderHart	Clojure
Beginning Groovy, Grails and Griffon	Vishal Layka	Groovy
Definitive Guide to Grails 2	Jeff Brown	Groovy
Groovy and Grails Recipes	Bashar Jawad	Groovy
Modern Java Web Development	Vishal Layka	Java
Java 7 Recipes	Josh Juneau	Java
Java EE 7 Recipes	Josh Juneau	Java
Beginning Java 7	Jeff Friesen	Java
Pro Java 7 NIO.2	Anghel Leonard	Java
Java 7 for Absolute Beginners	Jay Bryant	Java
Oracle Certified Java Enterprise Architect Java EE7	B V Kumar	Java
Beginning Scala	David Pollak	Scala
Scala basics	Sue Armstrong	Scala

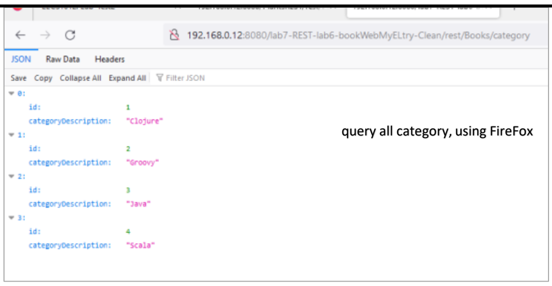
Another version:
Use Gson, Response


```java
package restService;

import java.util.List;

@Path("Books") // this is the path of the service
public class BookServiceFront {

    //Catalog catalog;

    public BookServiceFront() {


    /* GET Plants */
    // return the collection of plants as JSON

    @GET
    @Path("/category")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getCategoryNames() {
        //String content = catalog.getCatalogAsJSON();
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Category> categoryList = bookDao.findAllCategories();
        String content = new Gson().toJson(categoryList);
        return Response.status(200).entity(content).build();
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getAllBooks() {
        //String content = catalog.getCatalogAsJSON();
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Book> bList = bookDao.findAllBooks();
        String content = new Gson().toJson(bList);
        return Response.status(200).entity(content).build();
    }

      @GET
      @Path("/searchByCat/{catid}")
      @Produces(MediaType.APPLICATION_JSON)
      public Response getBooksByCategory(@PathParam("catid") Strin
        BookDAO bookDao = new BookDAOImpl();
        // calling DAO method to retrieve bookList from Database
        List<Book> bList = bookDao.findBooksByCategory(id);
        String content = new Gson().toJson(bList);
        return Response.status(200).entity(content).build();
    }
```

```
curl  http://localhost:8080/BooksREST/rest/Books
curl  http://localhost:8080/BooksREST/rest/Books/category
curl  http://localhost:8080/BooksREST/rest/Books/searchByCat/Groovy
curl  http://localhost:8080/BooksREST/rest/Books/searchByCat/Scala
```

111

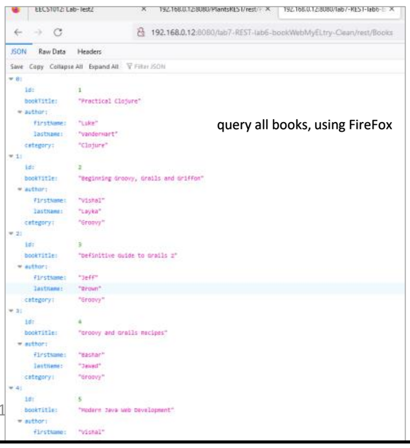
111


192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/category

JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

```
▼ 0:
    id:                     1
    categoryDescription:    "Clojure"
▼ 1:
    id:                     2
    categoryDescription:    "Groovy"
▼ 2:
    id:                     3
    categoryDescription:    "Java"
▼ 3:
    id:                     4
    categoryDescription:    "Scala"
```

query all category, using FireFox

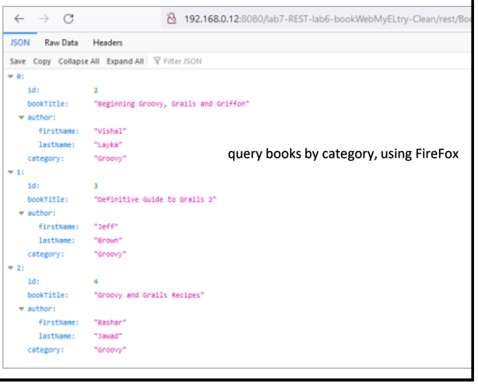

EECS1012: Lab-Test2 | 192.168.0.12:8080/PlantsREST/rest/ | 192.168.0.12:8080/lab7-REST-lab6-

192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books

JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

```
▼ 0:
    id:             1
    bookTitle:      "Practical Clojure"
  ▼ author:
      firstName:    "Luke"
      lastName:     "VanderHart"
    category:       "Clojure"
▼ 1:
    id:             2
    bookTitle:      "Beginning Groovy, Grails and Griffon"
  ▼ author:
      firstName:    "Vishal"
      lastName:     "Layka"
    category:       "Groovy"
▼ 2:
    id:             3
    bookTitle:      "Definitive Guide to Grails 2"
  ▼ author:
      firstName:    "Jeff"
      lastName:     "Brown"
    category:       "Groovy"
▼ 3:
    id:             4
    bookTitle:      "Groovy and Grails Recipes"
  ▼ author:
      firstName:    "Bashar"
      lastName:     "Jawad"
    category:       "Groovy"
▼ 4:
    id:             5
    bookTitle:      "Modern Java Web Development"
  ▼ author:
      firstName:    "Vishal"
```

query all books, using FireFox


192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Bo

JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

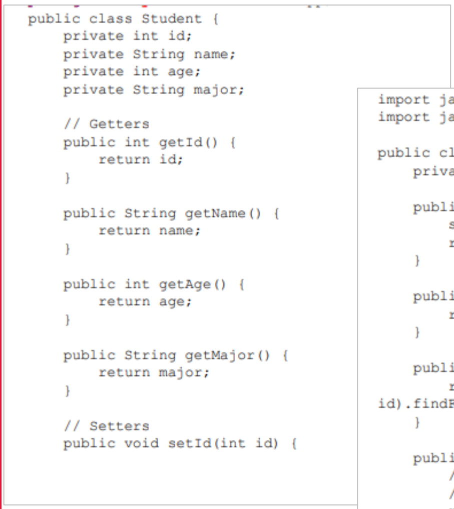
```
▼ 0:
    id:             2
    bookTitle:      "Beginning Groovy, Grails and Griffon"
  ▼ author:
      firstName:    "Vishal"
      lastName:     "Layka"
    category:       "Groovy"
▼ 1:
    id:             3
    bookTitle:      "Definitive Guide to Grails 2"
  ▼ author:
      firstName:    "Jeff"
      lastName:     "Brown"
    category:       "Groovy"
▼ 2:
    id:             4
    bookTitle:      "Groovy and Grails Recipes"
  ▼ author:
      firstName:    "Bashar"
      lastName:     "Jawad"
    category:       "Groovy"
```

query books by category, using FireFox

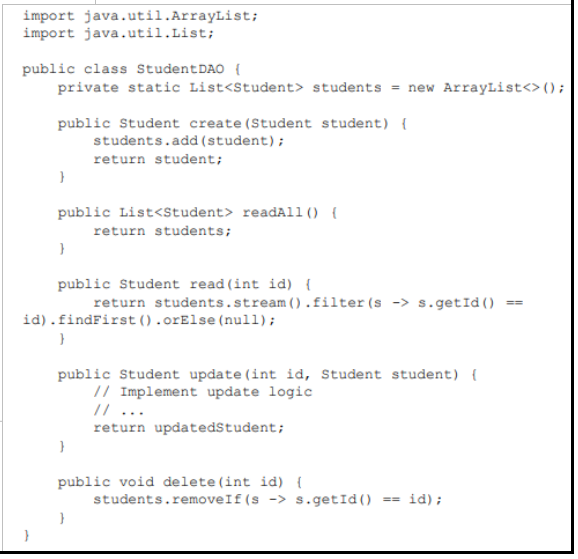
11

112

+ database another student system



```
public class Student {
    private int id;
    private String name;
    private int age;
    private String major;

    // Getters
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getMajor() {
        return major;
    }

    // Setters
    public void setId(int id) {
```

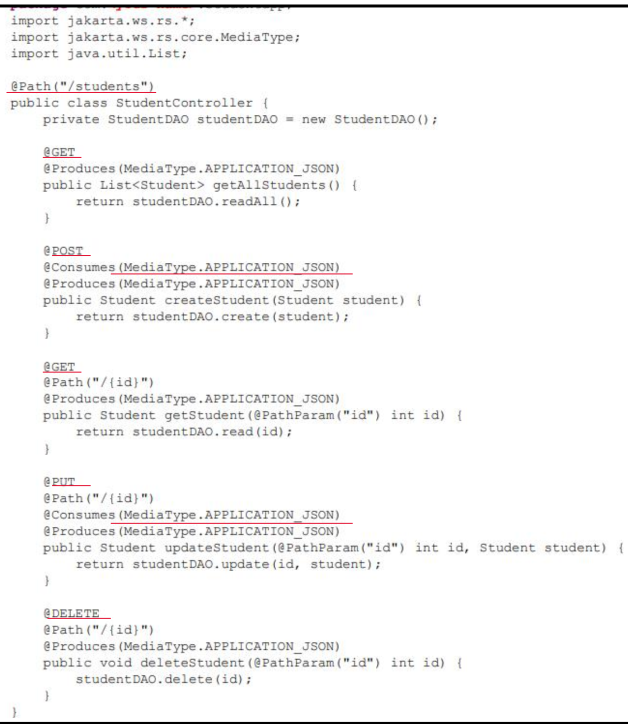


```
import java.util.ArrayList;
import java.util.List;

public class StudentDAO {
    private static List<Student> students = new ArrayList<>();

    public Student create(Student student) {
        students.add(student);
        return student;
    }

    public List<Student> readAll() {
        return students;
    }

    public Student read(int id) {
        return students.stream().filter(s -> s.getId() ==
id).findFirst().orElse(null);
    }

    public Student update(int id, Student student) {
        // Implement update logic
        // ...
        return updatedStudent;
    }

    public void delete(int id) {
        students.removeIf(s -> s.getId() == id);
    }
}
```

sqlite,
data source

114

114



```
import jakarta.ws.rs.*;
import jakarta.ws.rs.core.MediaType;
import java.util.List;

@Path("/students")
public class StudentController {
    private StudentDAO studentDAO = new StudentDAO();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Student> getAllStudents() {
        return studentDAO.readAll();
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Student createStudent(Student student) {
        return studentDAO.create(student);
    }

    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Student getStudent(@PathParam("id") int id) {
        return studentDAO.read(id);
    }

    @PUT
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Student updateStudent(@PathParam("id") int id, Student student) {
        return studentDAO.update(id, student);
    }

    @DELETE
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public void deleteStudent(@PathParam("id") int id) {
        studentDAO.delete(id);
    }
}
```

115

115

Testing with Curl

Run your application and use Postman or Curl commands to test the CRUD operations.

Here are some example Curl commands to test the different scenarios:

Create a Student

```
curl -X POST -H "Content-Type: application/json" -d '{"name":"John", "age":20,
"major":"Computer Science"}' http://localhost:8080/YourApp/students
      > Windows CMD can't read single quotes, so you may have to use \"
      to replace the single and quotes in your data payload

      curl -X POST -H "Content-Type: application/json" -d
      "{\"name\":\"John\", \"age\":20, \"major\":\"Computer Science\"}"
      http://localhost:8080/YourApp/students
```

List All Students

```
curl -X GET http://localhost:8080/YourApp/students
```

Get a Specific Student

```
curl -X GET http://localhost:8080/YourApp/students/1
```

Update a Student

```
curl -X PUT -H "Content-Type: application/json" -d '{"name":"John Doe", "age":21,
"major":"Software Engineering"}' http://localhost:8080/YourApp/students/1
```

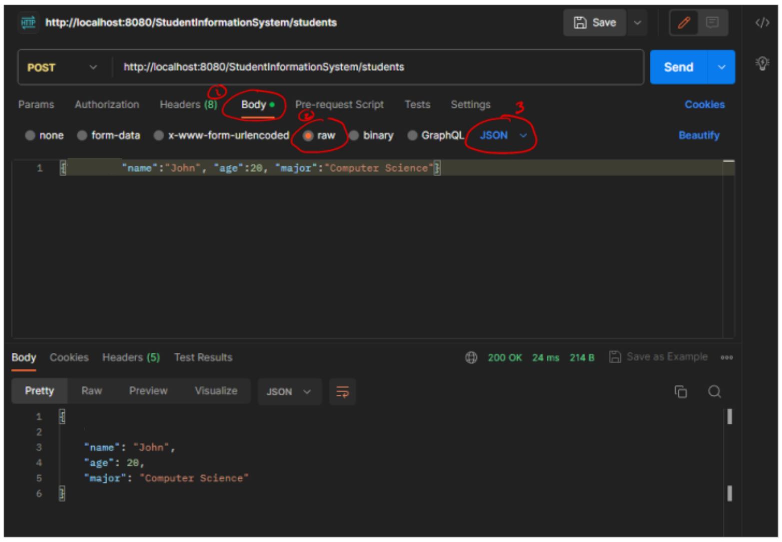
Delete a Student

```
curl -X DELETE http://localhost:8080/YourApp/students/1
```

116

116

With postman



http://localhost:8080/StudentInformationSystem/students

POST http://localhost:8080/StudentInformationSystem/students

```
{        "name":"John", "age":20, "major":"Computer Science"}
```

200 OK 24 ms 214 B

```
{

    "name": "John",
    "age": 20,
    "major": "Computer Science"
}
```

```
DELETE http://localhost:8080/YourApp/students/1
```

117

117

Talk to web services

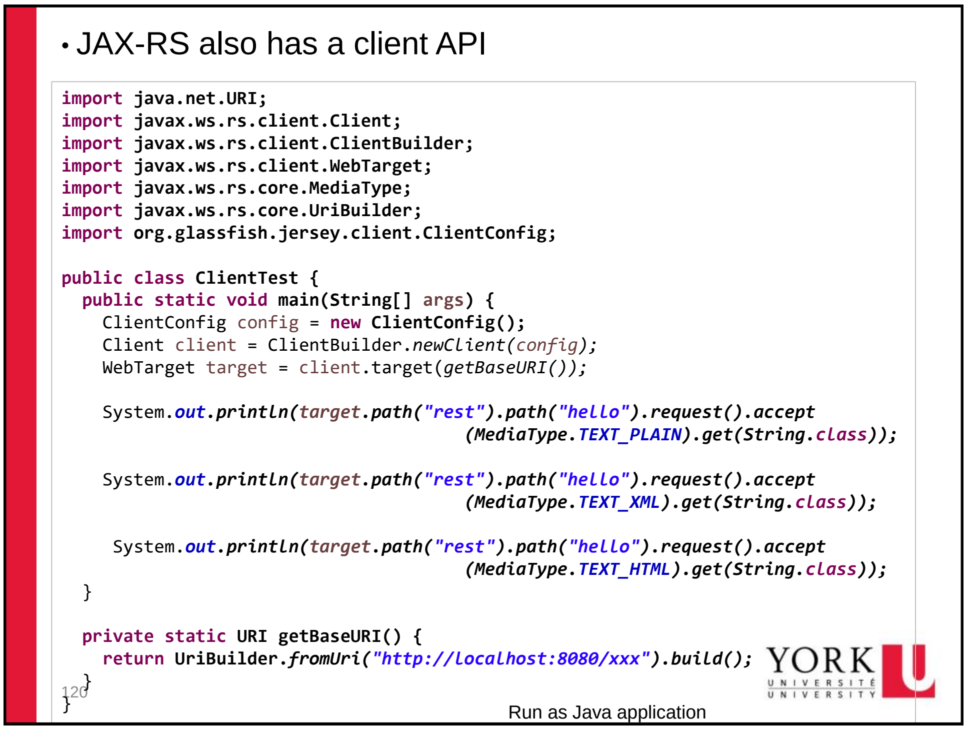
- Browser (GET only)
- curl
- Postman

- Java plain program e.g., using URL connection
- JAX-RS client API

For your information

- Use JavaScript/jQuery for (ajax) connection

```
public class httpCallClass {

public static void main(String[] args) {
// TODO Auto-generated method stub

String urlString = "http://localhost:8080/xxx/rest/helloM";


String inline = "";
try {
      URL url = new URL(urlString);
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.setRequestMethod("GET");
      conn.connect();
      int responsecode = conn.getResponseCode();
      System.out.println("Response code " + responsecode);
      if (responsecode == 200) {

      Scanner sc = new Scanner(url.openStream());
      while (sc.hasNext()) {
      inline += sc.nextLine();
      }

      System.out.println(inline);
      }

} catch (IOException e) {
              // TODO Auto-generated catch block e.printStackTrace();
}
   }
}
```



- JAX-RS also has a client API

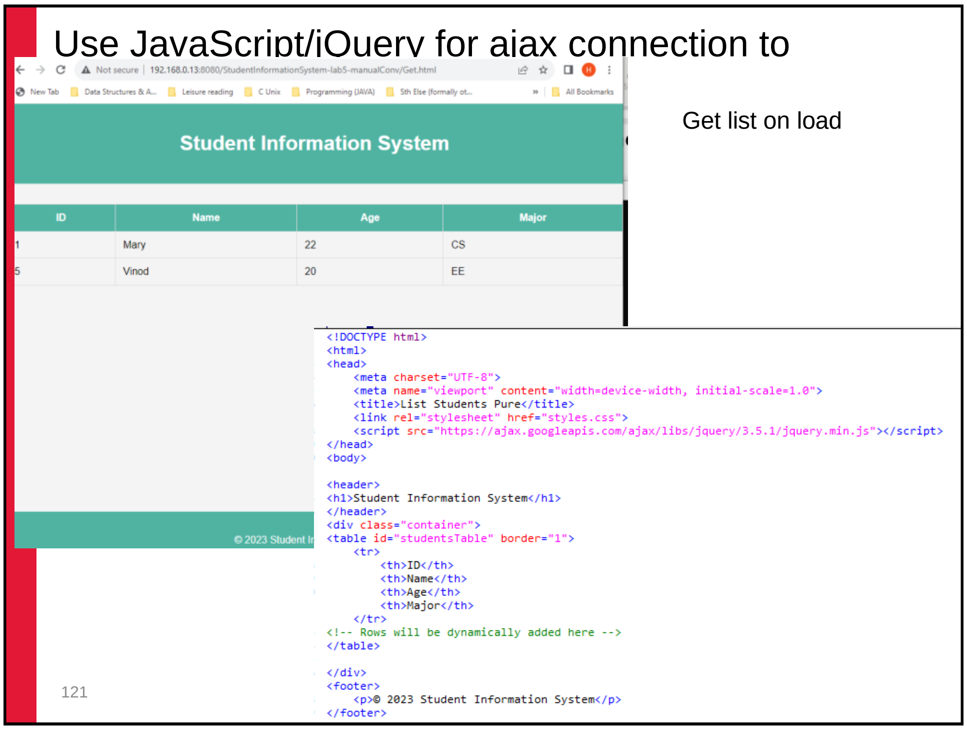
```
import java.net.URI;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;
import org.glassfish.jersey.client.ClientConfig;

public class ClientTest {
  public static void main(String[] args) {
    ClientConfig config = new ClientConfig();
    Client client = ClientBuilder.newClient(config);
    WebTarget target = client.target(getBaseURI());

    System.out.println(target.path("rest").path("hello").request().accept
                                   (MediaType.TEXT_PLAIN).get(String.class));

    System.out.println(target.path("rest").path("hello").request().accept
                                   (MediaType.TEXT_XML).get(String.class));

     System.out.println(target.path("rest").path("hello").request().accept
                                   (MediaType.TEXT_HTML).get(String.class));
  }

  private static URI getBaseURI() {
    return UriBuilder.fromUri("http://localhost:8080/xxx").build();
  }
}
```

YORK UNIVERSITÉ UNIVERSITY

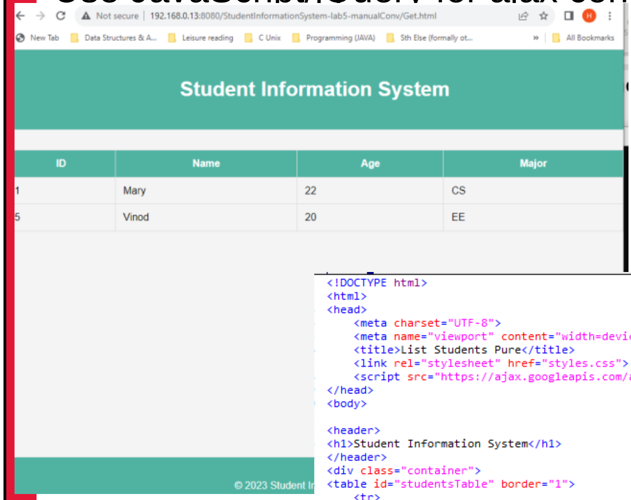
Run as Java application

120

120



Use JavaScript/jQuery for ajax connection to



Not secure | 192.168.0.13:8080/StudentInformationSystem-lab5-manualConv/Get.html

Student Information System

ID	Name	Age	Major
1	Mary	22	CS
5	Vinod	20	EE

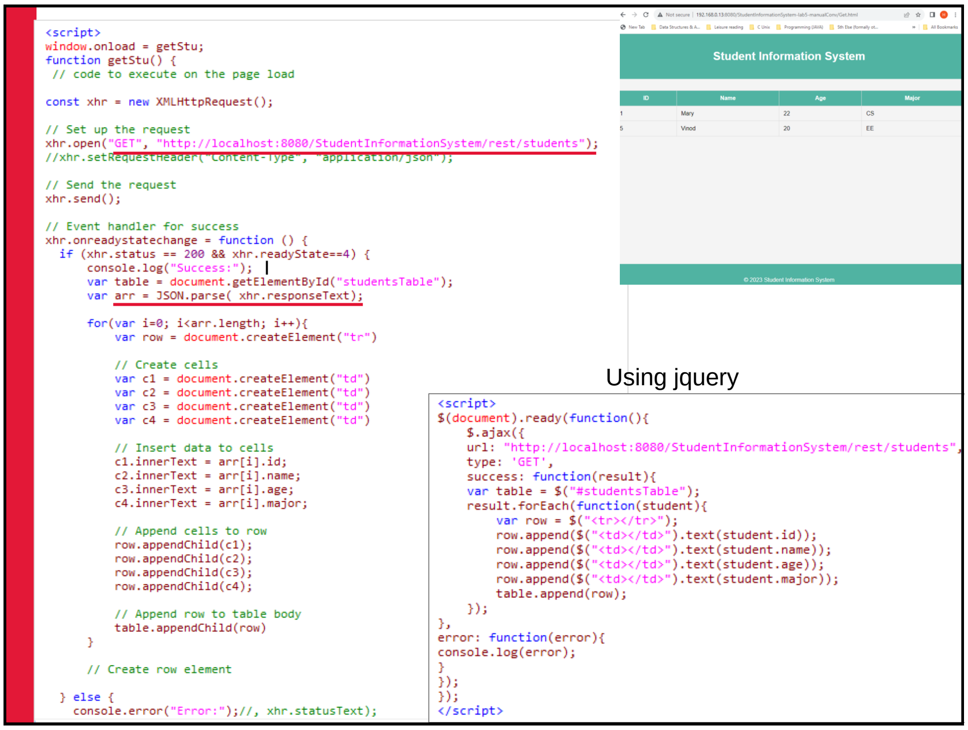
© 2023 Student In

Get list on load

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>List Students Pure</title>
    <link rel="stylesheet" href="styles.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>

<header>
<h1>Student Information System</h1>
</header>
<div class="container">
<table id="studentsTable" border="1">
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Age</th>
        <th>Major</th>
    </tr>
<!-- Rows will be dynamically added here -->
</table>

</div>
<footer>
    <p>© 2023 Student Information System</p>
</footer>
```

121

121



```
<script>
window.onload = getStu;
function getStu() {
 // code to execute on the page load

const xhr = new XMLHttpRequest();

// Set up the request
xhr.open("GET", "http://localhost:8080/StudentInformationSystem/rest/students");
//xhr.setRequestHeader("Content-Type", "application/json");

// Send the request
xhr.send();

// Event handler for success
xhr.onreadystatechange = function () {
  if (xhr.status == 200 && xhr.readyState==4) {
      console.log("Success:");
      var table = document.getElementById("studentsTable");
      var arr = JSON.parse( xhr.responseText);

      for(var i=0; i<arr.length; i++){
          var row = document.createElement("tr")

          // Create cells
          var c1 = document.createElement("td")
          var c2 = document.createElement("td")
          var c3 = document.createElement("td")
          var c4 = document.createElement("td")

          // Insert data to cells
          c1.innerText = arr[i].id;
          c2.innerText = arr[i].name;
          c3.innerText = arr[i].age;
          c4.innerText = arr[i].major;

          // Append cells to row
          row.appendChild(c1);
          row.appendChild(c2);
          row.appendChild(c3);
          row.appendChild(c4);

          // Append row to table body
          table.appendChild(row)
      }

      // Create row element

  } else {
    console.error("Error:");//, xhr.statusText);
```

Not secure | 192.168.0.13:8080/StudentInformationSystem-lab5-manualConv/Get.html

New Tab | Data Structures & A... | Leisure reading | C Unix | Programming (JAVA) | Sth Else (formally ot... | All Bookmarks

Student Information System

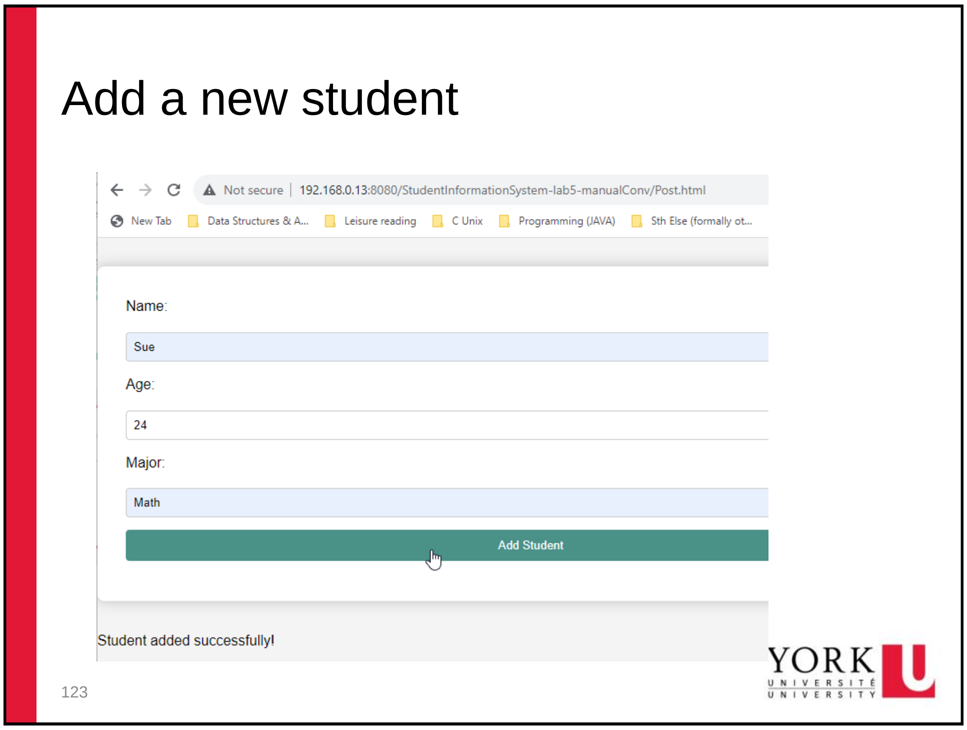
ID	Name	Age	Major
1	Mary	22	CS
5	Vinod	20	EE

© 2023 Student Information System

Using jquery

```
<script>
$(document).ready(function(){
    $.ajax({
    url: "http://localhost:8080/StudentInformationSystem/rest/students",
    type: 'GET',
    success: function(result){
    var table = $("#studentsTable");
    result.forEach(function(student){
        var row = $("<tr></tr>");
        row.append($("<td></td>").text(student.id));
        row.append($("<td></td>").text(student.name));
        row.append($("<td></td>").text(student.age));
        row.append($("<td></td>").text(student.major));
        table.append(row);
    });
},
error: function(error){
console.log(error);
}
});
});
</script>
```

122



Add a new student

Not secure | 192.168.0.13:8080/StudentInformationSystem-lab5-manualConv/Post.html

New Tab | Data Structures & A... | Leisure reading | C Unix | Programming (JAVA) | Sth Else (formally ot...

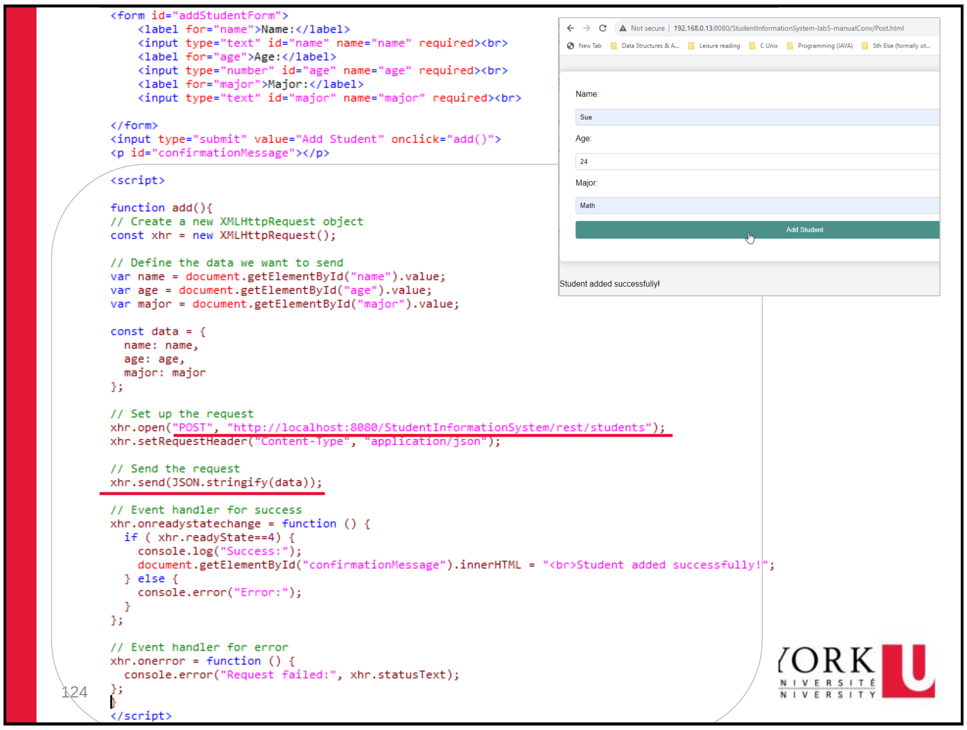
Name:	Sue
Age:	24
Major:	Math

Add Student

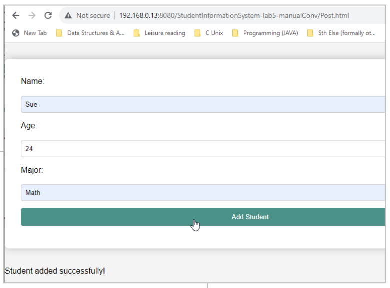
Student added successfully!

123

123



```
<form id="addStudentForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br>
    <label for="age">Age:</label>
    <input type="number" id="age" name="age" required><br>
    <label for="major">Major:</label>
    <input type="text" id="major" name="major" required><br>

</form>
<input type="submit" value="Add Student" onclick="add()">
<p id="confirmationMessage"></p>

<script>

function add(){
// Create a new XMLHttpRequest object
const xhr = new XMLHttpRequest();

// Define the data we want to send
var name = document.getElementById("name").value;
var age = document.getElementById("age").value;
var major = document.getElementById("major").value;

const data = {
  name: name,
  age: age,
  major: major
};

// Set up the request
xhr.open("POST", "http://localhost:8080/StudentInformationSystem/rest/students");
xhr.setRequestHeader("Content-Type", "application/json");

// Send the request
xhr.send(JSON.stringify(data));

// Event handler for success
xhr.onreadystatechange = function () {
  if ( xhr.readyState==4) {
    console.log("Success:");
    document.getElementById("confirmationMessage").innerHTML = "<br>Student added successfully!";
  } else {
    console.error("Error:");
  }
};

// Event handler for error
xhr.onerror = function () {
  console.error("Request failed:", xhr.statusText);
};
}
</script>
```



Name: Sue

Age: 24

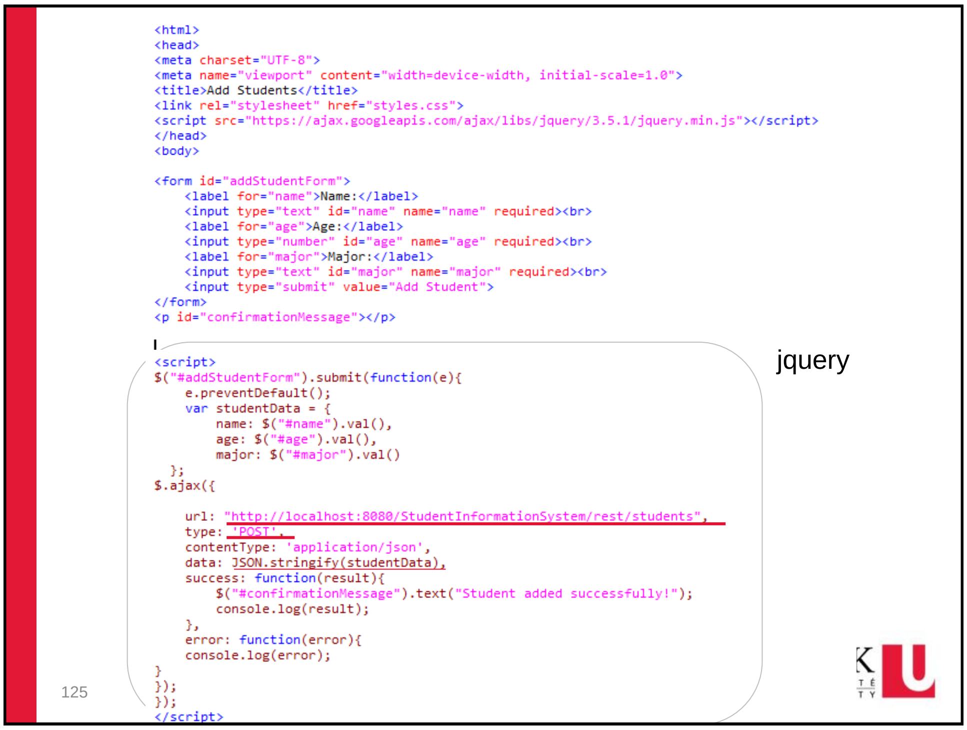
Major: Math

Add Student

Student added successfully!

124

124



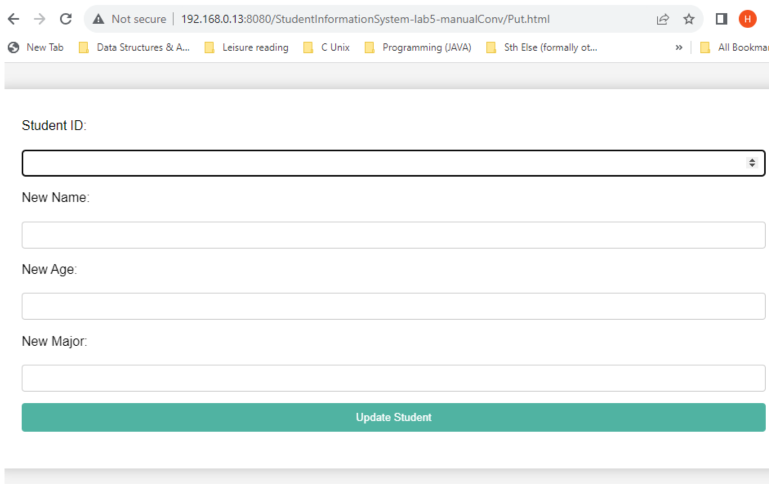
```
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Add Students</title>
<link rel="stylesheet" href="styles.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>

<form id="addStudentForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br>
    <label for="age">Age:</label>
    <input type="number" id="age" name="age" required><br>
    <label for="major">Major:</label>
    <input type="text" id="major" name="major" required><br>
    <input type="submit" value="Add Student">
</form>
<p id="confirmationMessage"></p>

<script>
$("#addStudentForm").submit(function(e){
    e.preventDefault();
    var studentData = {
        name: $("#name").val(),
        age: $("#age").val(),
        major: $("#major").val()
  };
$.ajax({

    url: "http://localhost:8080/StudentInformationSystem/rest/students",
    type: 'POST',
    contentType: 'application/json',
    data: JSON.stringify(studentData),
    success: function(result){
        $("#confirmationMessage").text("Student added successfully!");
        console.log(result);
    },
    error: function(error){
    console.log(error);
}
});
});
</script>
```

jquery

125

125

Update an existing student



Not secure | 192.168.0.13:8080/StudentInformationSystem-lab5-manualConv/Put.html

New Tab | Data Structures & A... | Leisure reading | C Unix | Programming (JAVA) | Sth Else (formally ot... | » | All Bookma

Student ID:

New Name:

New Age:

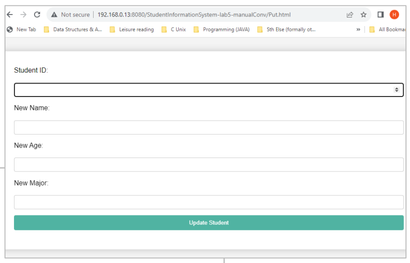
New Major:

Update Student

126

126

```
<form id="updateStudentForm">
    <label for="id">Student ID:</label>
    <input type="number" id="id" name="id" required><br>
    <label for="name">New Name:</label>
    <input type="text" id="name" name="name"><br>
    <label for="age">New Age:</label>
    <input type="number" id="age" name="age"><br>
    <label for="major">New Major:</label>
    <input type="text" id="major" name="major"><br>
    <input type="submit" value="Update Student" onclick="up">
</form>
<input type="submit" value="Update Student Ajax" onclick="updateStu()">

<p id="updateConfirmationMessage"></p>

 <script>
function updateStu(){
    const xhr = new XMLHttpRequest();

    // Define the data we want to send

    var id = document.getElementById("id").value;

    var name = document.getElementById("name").value;
    var age = document.getElementById("age").value;
    var major = document.getElementById("major").value;

    const data = {
      name: name,
      age: age,
      major: major
    };
    // Set up the request
    xhr.open("PUT", "http://localhost:8080/StudentInformationSystem/rest/students/" + id);
    xhr.setRequestHeader("Content-Type", "application/json");

    // Send the request
    xhr.send(JSON.stringify(data));

    // Event handler for success
    xhr.onreadystatechange = function () {
      if (/*xhr.status == 200 &&*/ xhr.readyState==4) {
        console.log("Success:"); //, JSON.parse(xhr.responseText));
        document.getElementById("updateConfirmationMessage").innerHTML = "<br>Student updated successfully!";
      } else {
        console.error("Error:");//, xhr.statusText);
      }
    };

    // Event handler for error
    xhr.onerror = function () {
      console.error("Request failed:", xhr.statusText);
    };
}
```



Not secure | 192.168.0.13:8080/StudentInformationSystem-lab5-manualConv/Put.html

Student ID:

New Name:

New Age:

New Major:

Update Student

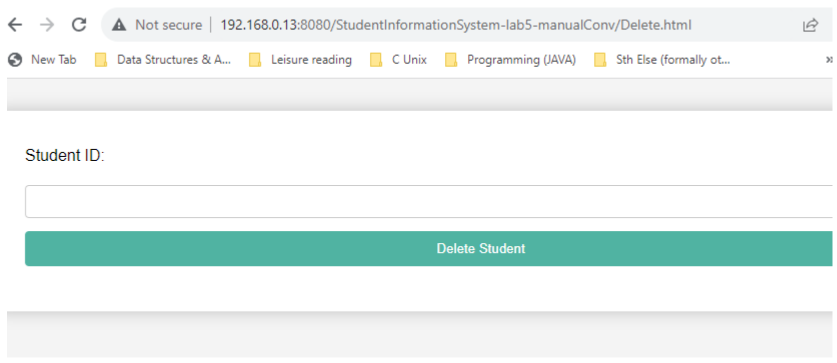
127

```
  <form id="updateStudentForm">
    <label for="id">Student ID:</label>
    <input type="number" id="id" name="id" required><br>
    <label for="name">New Name:</label>
    <input type="text" id="name" name="name"><br>
    <label for="age">New Age:</label>
    <input type="number" id="age" name="age"><br>
    <label for="major">New Major:</label>
    <input type="text" id="major" name="major"><br>
    <input type="submit" value="Update Student">
</form>
<p id="updateConfirmationMessage"></p>

<script>
    $("#updateStudentForm").submit(function(e){
        e.preventDefault();
            var studentData = {
            name: $("#name").val(),
            age: $("#age").val(),
            major: $("#major").val()
        };
        var studentId = $("#id").val();
        $.ajax({

            url: "http://localhost:8080/StudentInformationSystemSqlite_Test3/students/" + studentId,
            type: 'PUT',
            contentType: 'application/json',
            data: JSON.stringify(studentData),
            success: function(result){
                $("#updateConfirmationMessage").text("Student updated successfully!");
            },
    error: function(error){
        console.log(error);
    }
    });
    });
</script>
```

128

128

Delete a student

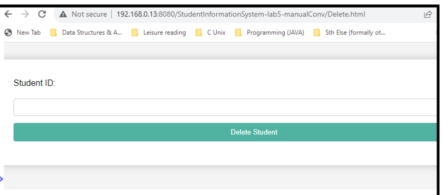

Not secure | 192.168.0.13:8080/StudentInformationSystem-lab5-manualConv/Delete.html

Student ID:

129

129

```
</head>
<body>

<!-- Inside body tag -->
<form id="deleteStudentForm">
<label for="id">Student ID:</label>
<input type="number" id="id" name="id" required><br>
<input type="submit" value="Delete Student">
</form>
<input type="submit" value="Delete Student" onclick="deleteStu()">
<p id="deleteConfirmationMessage"></p>

<script>

function deleteStu(){

    const xhr = new XMLHttpRequest();

    // Define the data we want to send
    var id = document.getElementById("id").value;


    // Set up the request
    xhr.open("DELETE", "http://localhost:8080/StudentInformationSystem/rest/students/" + id);
    //xhr.setRequestHeader("Content-Type", "application/json");

    // Send the request
    xhr.send();

    // Event handler for success
    xhr.onreadystatechange = function () {
      if (/*xhr.status == 200 &&*/ xhr.readyState==4) {
        console.log("Success:");
        document.getElementById("deleteConfirmationMessage").innerHTML =  "<br>Student deleted successfully!";
      } else {
        console.error("Error:");
      }
    };

// Event handler for error
xhr.onerror = function () {
  console.error("Request failed:", xhr.statusText);
};


}
```



130

```
<!DOCTYPE html>
<html>
<head>

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Delete Students</title>
<link rel="stylesheet" href="styles.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

</head>
<body>

<!-- Inside body tag -->
<form id="deleteStudentForm">
<label for="id">Student ID:</label>
<input type="number" id="id" name="id" required><br>
<input type="submit" value="Delete Student">
</form>
<p id="deleteConfirmationMessage"></p>

<script>
$("#deleteStudentForm").submit(function(e){
e.preventDefault();
var studentId = $("#id").val();
$.ajax({

    url: "http://localhost:8080/StudentInformationSystem/rest/students/" + studentId,
    type: 'DELETE',
    success: function(result){
    $("#deleteConfirmationMessage").text("Student deleted successfully!");
    },
    error: function(error){
console.log(error);
}
});
});
</script>


</body>
</html>
```

jquery

If run in different machine, need to change localhost

131

Main topics we covered

- Web App Architecture. Preliminary knowledge/Review
  - Client side: HTML CSS JavaScript
  - UML, design patterns, Java (cmd, thread, serialization),
- Client-Server, low level: socket programming
- Web applications (server side)
  - LAMP/CGI
  - Java Servlet
  - JSP, JavaBean, MVC pattern
  - SQL, Database access: JDBC. JPA
  - More: filter, ~~listener, Ajax~~, JSON
- Web (RESTful) services, micro services
- ~~SE topics: More design patterns, Performance, security~~
- Advanced topics (Tutorials): SpringBoot, React, Deployments: Docker container, cloud

Introduction 132

132

- Lab6 this Saturday.
  - Another TA session on Saturday noon
- Quiz 3 Sunday to Monday 24 hours
- Tutorials on eClass
  - Springboot
  - REST
  - React
  - Docker Docker container,
  - Cloud: AWS, Azure
- I am still around
- Final exam: May 25 1pm LAS1002 bring your laptop

133

133

Time to say goodbye ……

- Good luck with your exams and future studies
- Stay safe, and enjoy the coming summer!

134