What is REST architecture?

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

A REST Server simply provides access to resources and REST client accesses and modifies the resources using HTTP protocol. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML but JSON is the most popular one.

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

- **GET** - This is used to provide a read only access to a resource.
- **PUT** - This is used to create a new resource.
- **DELETE** - This is used to remove a resource.
- **POST** - This is used to update a existing resource or create a new resource.

RESTful Web Services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability *e. g. , communicationbetweenJavaandPython, orWindowsandLinuxapplications* is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, which provides resource representation such as JSON and set of HTTP Methods.

Creating RESTful for A Library

Consider we have a JSON based database of users having the following users in a file **users.json**:

```
{
   "user1" : {
      "name" : "mahesh",
   "password" : "password1",
   "profession" : "teacher",
   "id": 1
   },
   "user2" : {
      "name" : "suresh",
   "password" : "password2",
   "profession" : "librarian",
   "id": 2
   },
   "user3" : {
      "name" : "ramesh",
   "password" : "password3",
   "profession" : "clerk",
   "id": 3
   }
```

```
}
```

Based on this information we are going to provide following RESTful APIs.

S. N.	URI	HTTP Method	POST body	Result
1	listUsers	GET	empty	Show list of all the users.
2	addUser	POST	JSON String	Add details of new user.
3	deleteUser	DELETE	JSON String	Delete an existing user.
4	:id	GET	empty	Show details of a user.

I'm keeping most of the part of all the examples in the form of hard coding assuming you already know how to pass values from front end using Ajax or simple form data and how to process them using express **Request** object.

List Users

Let's implement our first RESTful API **listUsers** using the following code in a server.js file:

```
var express = require('express');
var app = express();
var fs = require("fs");

app.get('/listUsers', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       console.log( data );
       res.end( data );
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

Now try to access defined API using *http://127.0.0.1:8081/listUsers* on local machine. This should produce following result:

You can change given IP address when you will put the solution in production environment.

```
{
   "user1" : {
      "name" : "mahesh",
      "password" : "password1",
      "profession" : "teacher",
      "id": 1
   },
   "user2" : {
      "name" : "suresh",
      "password" : "password2",
      "profession" : "librarian",
      "id": 2
   },
   "user3" : {
      "name" : "ramesh",
      "password" : "password3",
      "profession" : "clerk",
```

```
      "id": 3
   }
}
```

Add User

Following API will show you how to add new user in the list. Following is the detail of the new user:

```
user = {
   "user4" : {
      "name" : "mohit",
      "password" : "password4",
      "profession" : "teacher",
      "id": 4
   }
}
```

You can accept the same input in the form of JSON using Ajax call but for teaching point of view, we are making it hard coded here. Following is the **addUser** API to a new user in the database:

```
var express = require('express');
var app = express();
var fs = require("fs");

var user = {
   "user4" : {
      "name" : "mohit",
      "password" : "password4",
      "profession" : "teacher",
      "id": 4
   }
}

app.get('/addUser', function (req, res) {
   // First read existing users.
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       data = JSON.parse( data );
       data["user4"] = user["user4"];
       console.log( data );
       res.end( JSON.stringify(data));
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)

})
```

Now try to access defined API using *http://127.0.0.1:8081/addUsers* on local machine. This should produce following result:

```
{ user1:
   { name: 'mahesh',
     password: 'password1',
     profession: 'teacher',
     id: 1 },
  user2:
   { name: 'suresh',
     password: 'password2',
     profession: 'librarian',
     id: 2 },
  user3:
   { name: 'ramesh',
     password: 'password3',
```

```
      profession: 'clerk',
      id: 3 },
   user4:
    { name: 'mohit',
      password: 'password4',
      profession: 'teacher',
      id: 4 }
}
```

Show Detail

Now we will implement an API which will be called using user ID and it will display the detail of the corresponding user.

```
var express = require('express');
var app = express();
var fs = require("fs");

app.get('/:id', function (req, res) {
   // First read existing users.
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       data = JSON.parse( data );
       var user = users["user" + req.params.id]
       console.log( user );
       res.end( JSON.stringify(user));
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)

})
```

Now let's call above service using *http://127.0.0.1:8081/2* on local machine. This should produce following result:

```
{
   "name":"suresh",
   "password":"password2",
   "profession":"librarian",
   "id":2
}
```

Delete User

This API is very similar to addUser API where we receive input data through req.body and then based on user ID we delete that user from the database. To keep our program simple we assume we are going to delete user with ID 2.

```
var express = require('express');
var app = express();
var fs = require("fs");

var id = 2;

app.get('/deleteUser', function (req, res) {

   // First read existing users.
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       data = JSON.parse( data );
       delete data["user" + 2];

       console.log( data );
       res.end( JSON.stringify(data));
```

```
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)

})
```

Now let's call above service using *http://127.0.0.1:8081/deleteUser* on local machine. This should produce following result:

```
{ user1:
   { name: 'mahesh',
     password: 'password1',
     profession: 'teacher',
     id: 1 },
  user3:
   { name: 'ramesh',
     password: 'password3',
     profession: 'clerk',
     id: 3 }
}
```