Ex No: 02	Implementation of SOAP based and RESTful Web Services inJava

Aim:

To implement SOAP and RESTful Web Services in Java.

Web Services:

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java,HTML, and XML.

SOAP Web Service:

SOAP is an acronym for Simple Object Access Protocol. It is an XML-based messaging protocol for exchanging information among computers. SOAP is an application of the XML specification. Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP.
Other frameworks including CORBA, DCOM, and Java RMI provide similar functionality to SOAP, but SOAP messages are written entirely in XML and are therefore uniquely platform- and language- independent.

RESTFUL Webservices

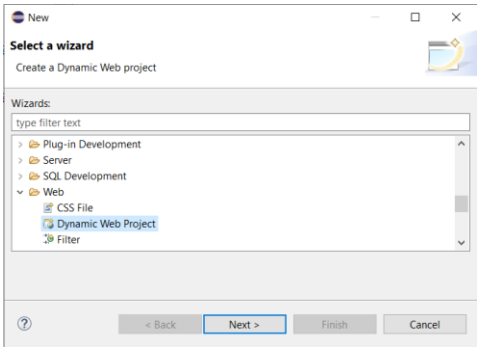
REST stands for **REpresentational State Transfer**. Restful Web Services is a lightweight, maintainable, and scalable service that is built on the REST architecture. Restful Web Service, expose API from your application in a secure, uniform, stateless manner to the calling client. The calling client can perform predefined operations using the Restful service. The underlying protocol for REST is HTTP.

Four HTTP methods are commonly used in REST based architecture.

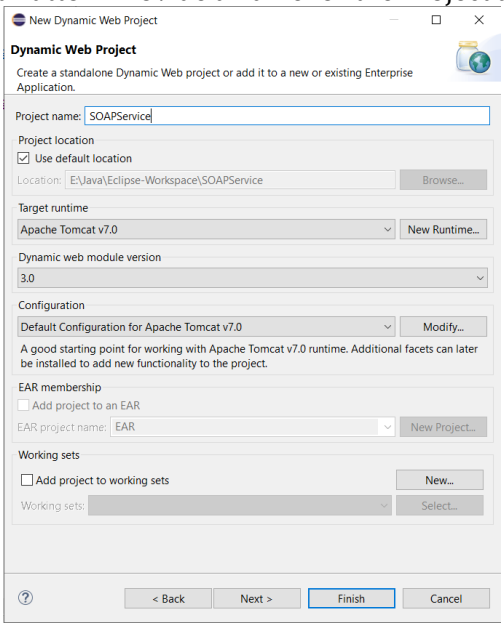
- **GET** − Provides a read only access to a resource.
- **POST** − Used to create a new resource.
- **DELETE** − Used to remove a resource.
- **PUT** − Used to update an existing resource or create a new resource.

SOAP WEB SERVICE ENVIRONMENT CONFIGURATION

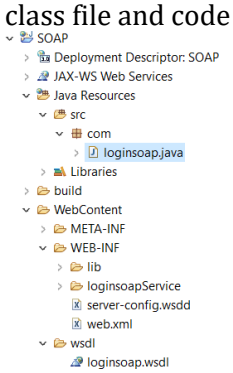
1. Install Eclipse IDE and add Web Project during Installation
2. Install the tomcat server
3. Create a Dynamic Web Project from the File Menu



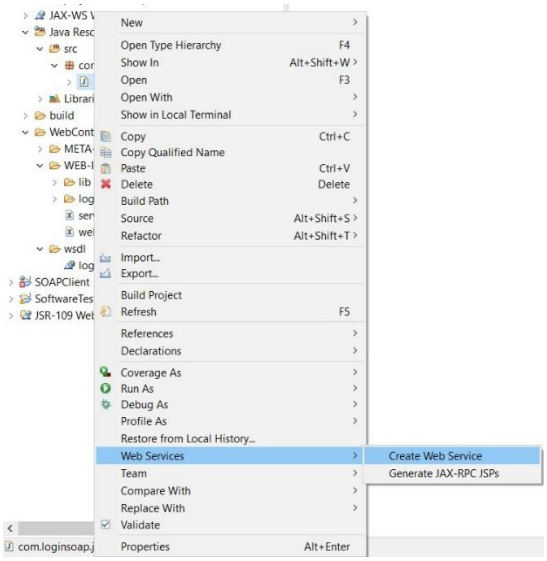
4. Click on the Next Button. Provide a name for the Project and select the server



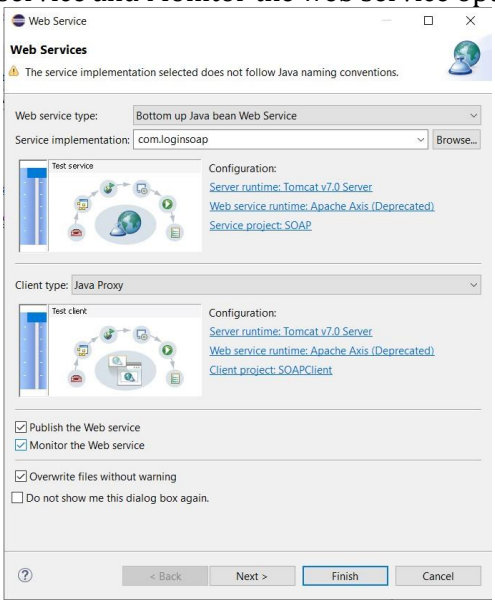
5. Click on the Finish Button. The project has been created successfully.
6. In the package explorer view, expand Java Resources -> src and create a package
7. In the package create a java class file and code



8. In order to run the service, right click on the java file and select the Web service - > Create Web Service



9. Select Bottom up Java bean Web service,set Test Service and Test Client High, Check Publish the web service and Monitor the web service option



10. Click on the Finish button
11. Now the web service will run on the server

SOAP WEB SERVICE PROGRAMMING

loginsoap.java package com;

public class loginsoap {

public String login(String username, String password) { int domainPos = username.indexOf("@");

String domain = username.substring(domainPos+1,domainPos+4); if(domain.equals("tce") && password.equals("password")) {

return "Login Successful";

}

return "Invalid Credentials";

}

}

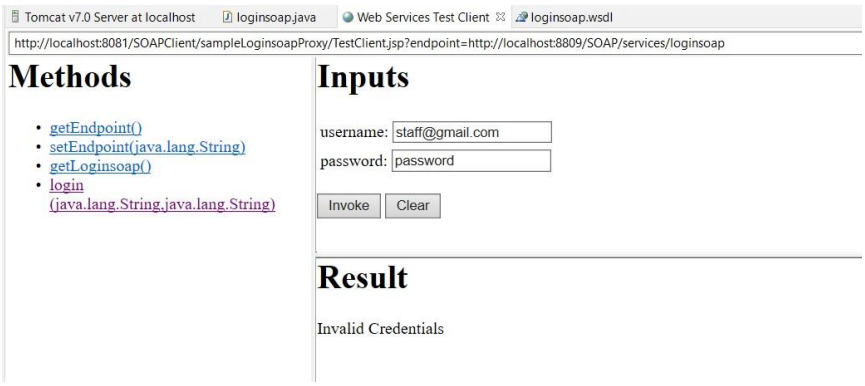
loginsoap.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://com"
xmlns:apachesoap="http://xml.apache.org/xml- soap" xmlns:impl="http://com"
xmlns:intf="http://com" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48 PDT)-->
<wsdl:types>
<schema elementFormDefault="qualified" targetNamespace="http://com"
xmlns="http://www.w3.org/2001/XMLSchema">
<element name="login">
<complexType>
<sequence>
<element name="username" type="xsd:string"/>
<element name="password" type="xsd:string"/>
</sequence>
</complexType>
</element>
<element name="loginResponse">
<complexType>
<sequence>
<element name="loginReturn" type="xsd:string"/>
</sequence>
</complexType>
</element>
</schema>
</wsdl:types>
<wsdl:message name="loginRequest">
<wsdl:part element="impl:login" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="loginResponse">
<wsdl:part element="impl:loginResponse" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:portType name="loginsoap">
<wsdl:operation name="login">
<wsdl:input message="impl:loginRequest" name="loginRequest">
</wsdl:input>
```

```
    <wsdl:output message="impl:loginResponse" name="loginResponse">
    </wsdl:output>
    </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="loginsoapSoapBinding" type="impl:loginsoap">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="login">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="loginRequest">
    <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="loginResponse">
    <wsdlsoap:body use="literal"/>
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="loginsoapService">
    <wsdl:port binding="impl:loginsoapSoapBinding" name="loginsoap">
    <wsdlsoap:address location="http://localhost:8081/SOAP/services/loginsoap"/>
    </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Screenshots

Invalid domain in email



Invalid password

Valid domain and password

RESTFUL WEB SERVICE ENVIRONMENT CONFIGURATION

Installation

1. Install nodejs and have npm (Node Package Manager)
2. Install Postman API tool
3. To install express into application inside folder, open command prompt and use this command
4. npm install express -save Installing express
5. Create a package.json file to add name of the application, version, description, author and dependencies with the help of npm init
6. Creating package.json file.

```
D:\19IT077>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (19it077)
version: (1.0.0)
description:
git repository:
keywords:
author:
license: (ISC)
About to write to D:\19IT077\package.json:

{
  "name": "19it077",
  "version": "1.0.0",
  "main": "script.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.1"
  },
  "devDependencies": {},
  "description": ""
}


Is this OK? (yes) yes
```

7. Installing nodemon

```
D:\19IT077>npm nodemon -g

Usage: npm <command>

where <command> is one of:
    access, adduser, audit, bin, bugs, c, cache, ci, cit,
    clean-install, clean-install-test, completion, config,
    create, ddp, dedupe, deprecate, dist-tag, docs, doctor,
    edit, explore, fund, get, help, help-search, hook, i, init,
    install, install-ci-test, install-test, it, link, list, ln,
    login, logout, ls, org, outdated, owner, pack, ping, prefix,
    profile, prune, publish, rb, rebuild, repo, restart, root,
    run, run-script, s, se, search, set, shrinkwrap, star,
    stars, start, stop, t, team, test, token, tst, un,
    uninstall, unpublish, unstar, up, update, v, version, view,
    whoami

npm <command> -h  quick help on <command>
npm -l            display full usage info
npm help <term>   search for help on <term>
npm help npm      involved overview

Specify configs in the ini-formatted file:
    C:\Users\HP\.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@6.13.4 C:\Program Files\nodejs\node_modules\npm
```

RESTFUL WEB SERVICE PROGRAMMING – script.js

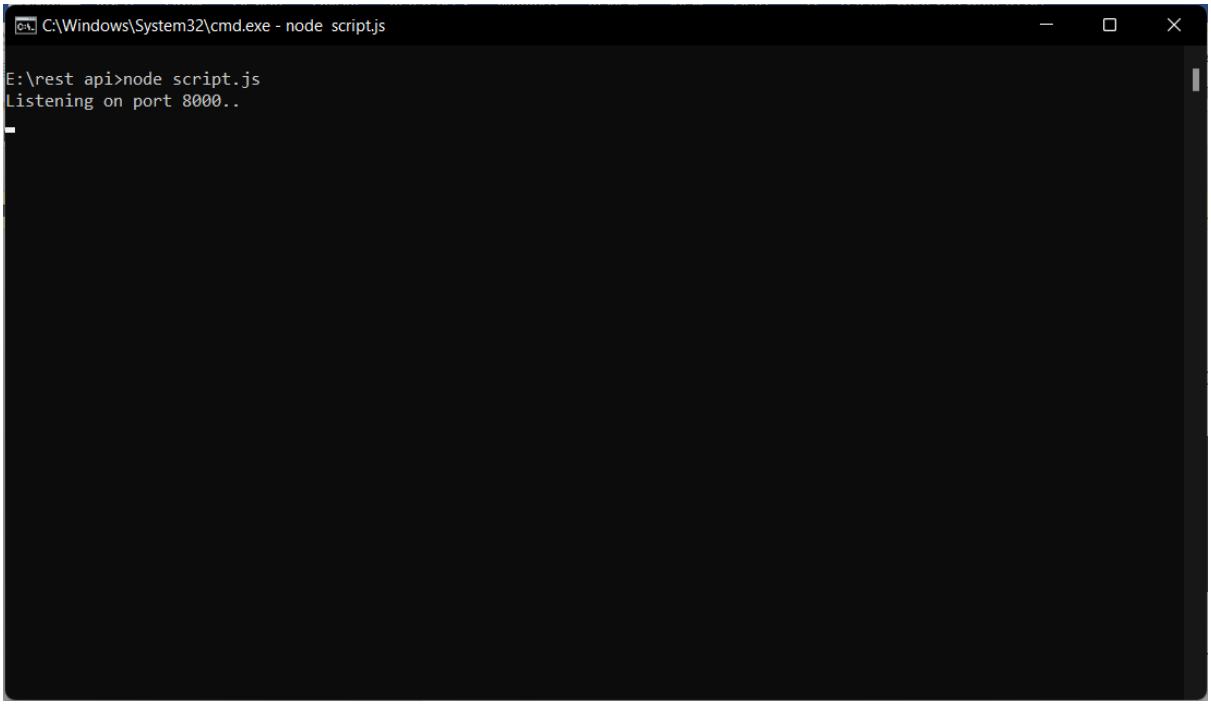
```
const express = require('express');
const app = express();
const Joi = require('joi')
app.use(express.json());

const books = [
{title: 'Harry Potter', id: 1},
{title: 'Twilight', id: 2},
{title: 'Lorien Legacies', id: 3}
]
function validateBook(book) {
  const schema = Joi.object({
  title: Joi.string().min(3).required()
  });
  // return Joi.validate(book, schema);
```

```
   return schema.validate(book)
   // const validation = schema.validate(book);
   // res.send(validation);
}
 //READ Request Handlers
app.get('/', (req, res) => {
res.send('Welcome to Edurekas REST API with Node.js Tutorial!!');
});

app.get('/api/books', (req,res)=> {
res.send(books);
});

app.get('/api/books/:id', (req, res) => {
const book = books.find(c => c.id === parseInt(req.params.id));

if (!book) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Ooops... Cant find what you are looking for!</h2>');
res.send(book);
});

//CREATE Request Handler
app.post('/api/books', (req, res)=> {
const { error } = validateBook(req.body);
if (error){
res.status(400).send(error.details[0].message)
return;
}
const book = {
id: books.length + 1,
title: req.body.title
};
books.push(book);
res.send(book);
});
//UPDATE Request Handler
app.put('/api/books/:id', (req, res) => {
const book = books.find(c=> c.id === parseInt(req.params.id));
if (!book) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Not
```

```
Found!! </h2>');
console.log(req.body);
console.log("bla");
const { error } = validateBook(req.body);
if (error){
res.status(400).send(error.details[0].message);
console.log(error)
return;
}
book.title = req.body.title;
res.send(book);
});
//DELETE Request Handler
app.delete('/api/books/:id', (req, res) => {
 const book = books.find( c=> c.id === parseInt(req.params.id));
if(!book) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;"> Not Found!! </h2>');
 const index = books.indexOf(book);
books.splice(index,1);
 res.send(book);
});

//PORT ENVIRONMENT VARIABLE
const port = process.env.PORT || 8000;
app.listen(port, () => console.log(`Listening on port ${port}..`));
```

Screenshots

Create a collection and add 4 requests for CRUD operations(Create – POST, Read – GET, Update – PUT,Delete – DELETE)
Run the code js code in cmd prompt or in visual studio

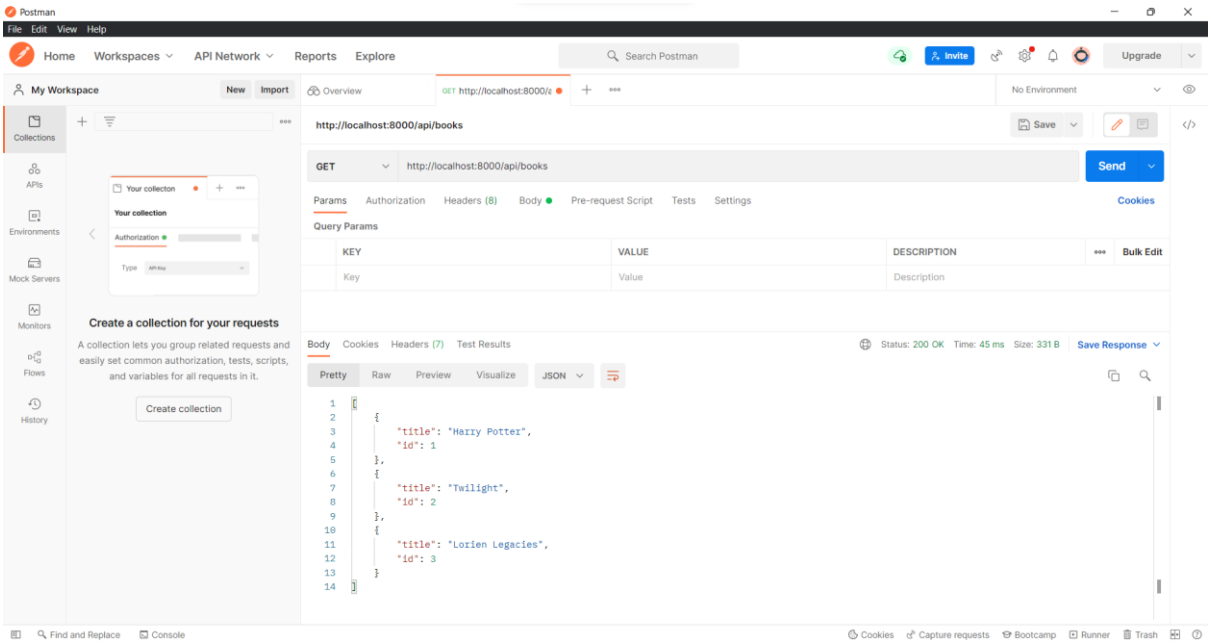


GET

Open postman then select the corresponding request name “GET” and type localhost:5001/ and click the send button.

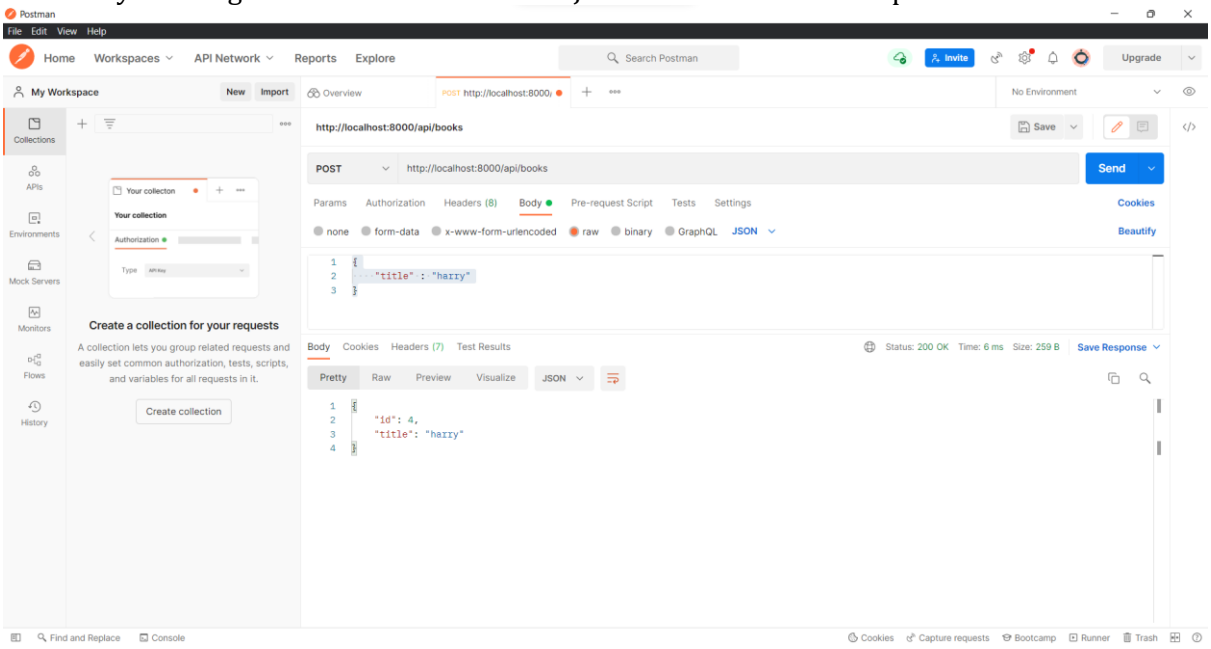
Select Preview in the response section.

To list all the students “http://localhost:8000/api/books” and click the send button. To see the response in JSON format, select Pretty and JSON from the dropdown list.



POST

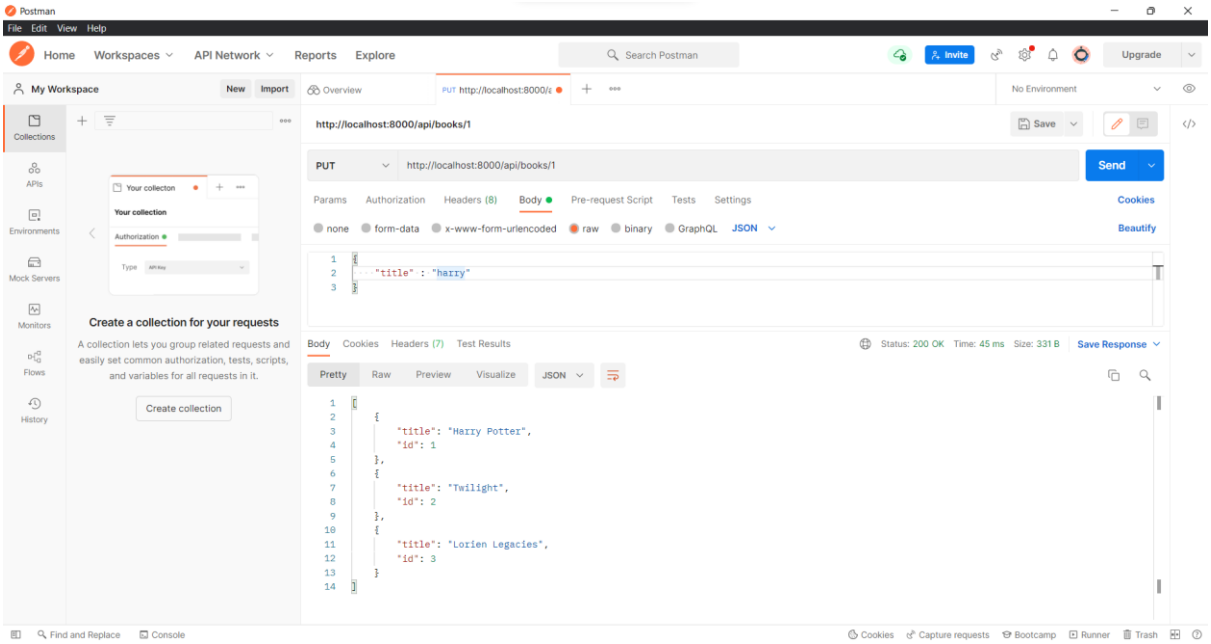
Create a post request in postman and type the URL and give parameters in the body section byselecting the raw radio button and JSON format from the drop down list.

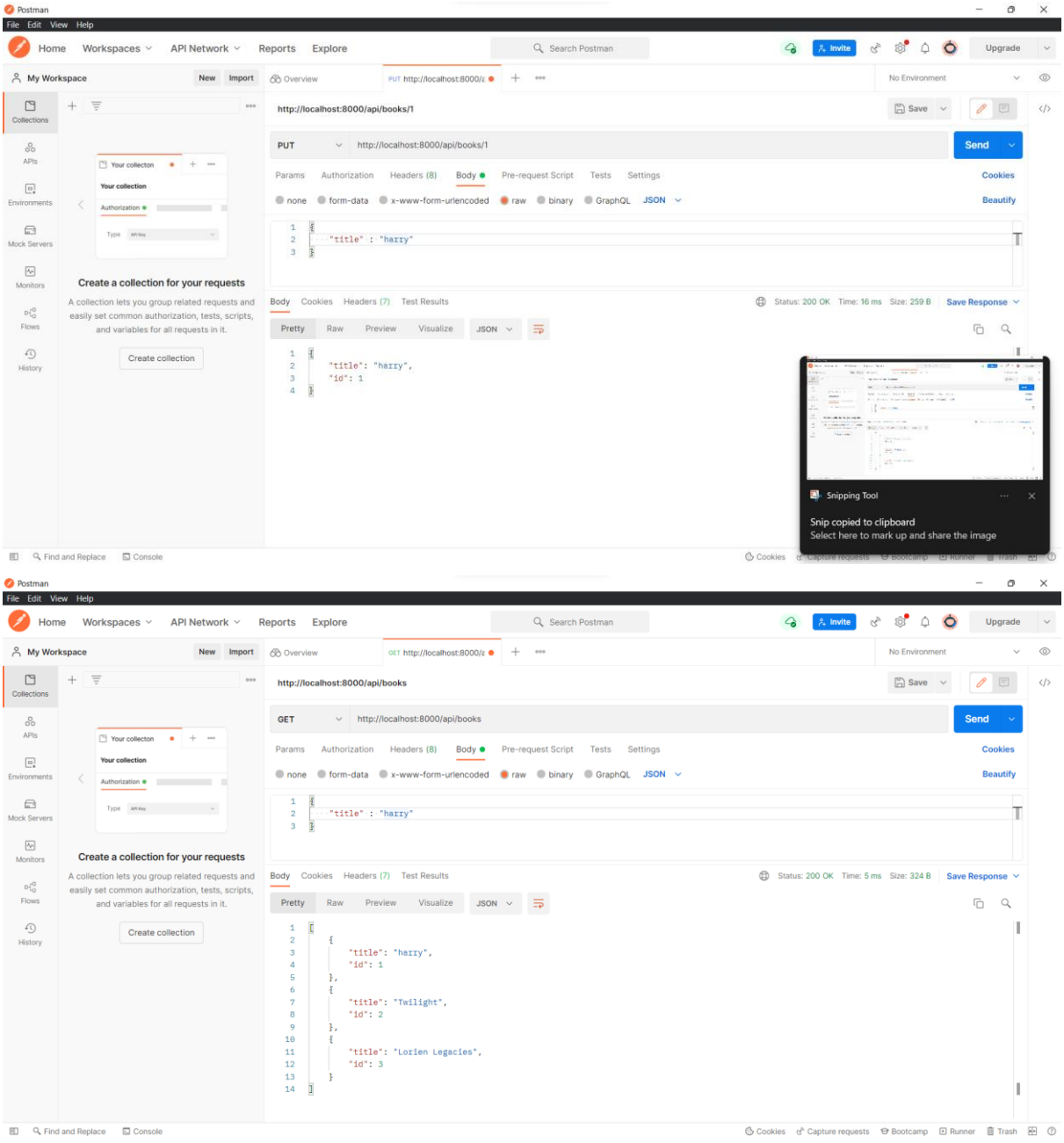


PUT

Create a PUT request in postman tool and type URL with the student ID as “localhost:800/api/books/1” and give parameter (i.e. status) in body section as a raw data and in JSON format.

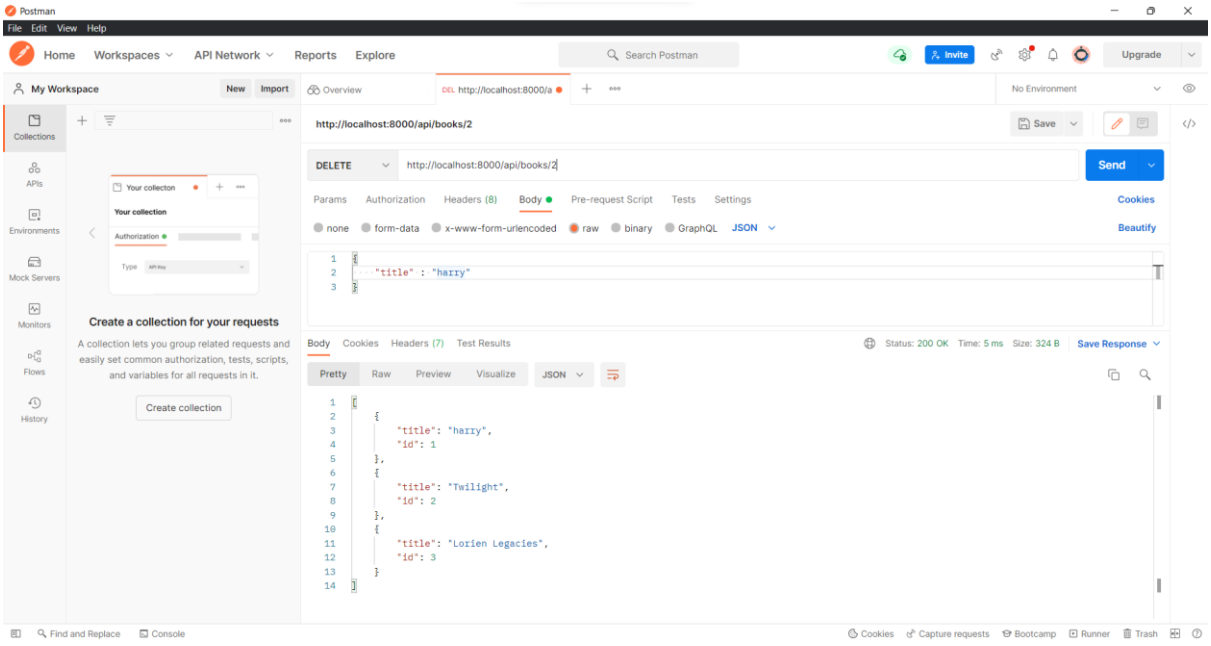
Click the send button and it shows the corresponding device details after updating.



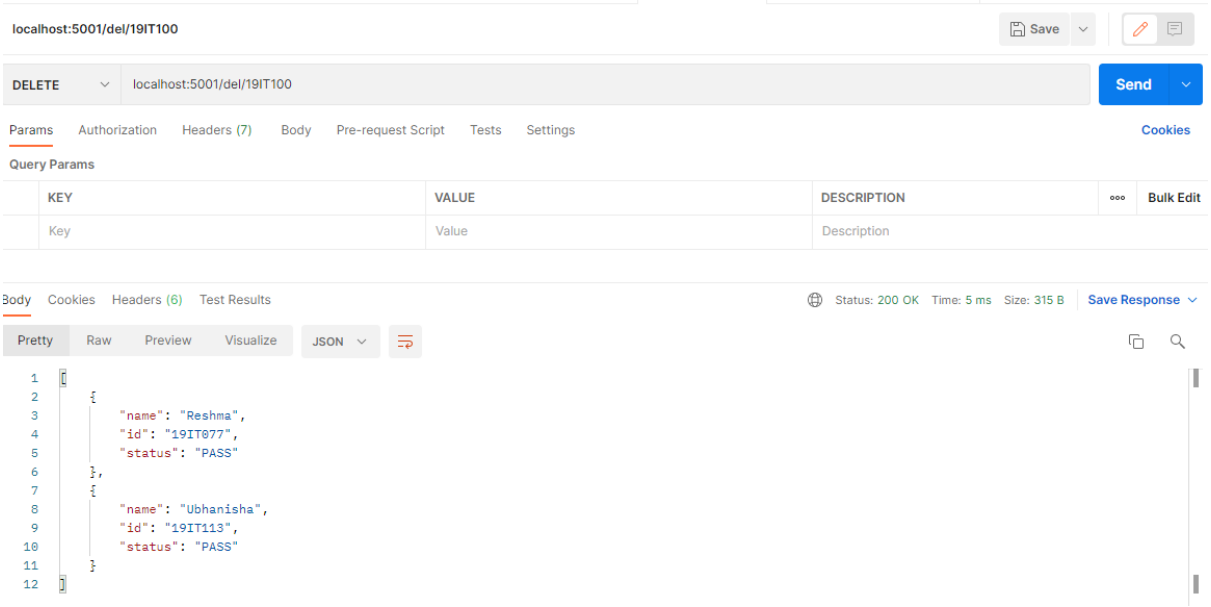


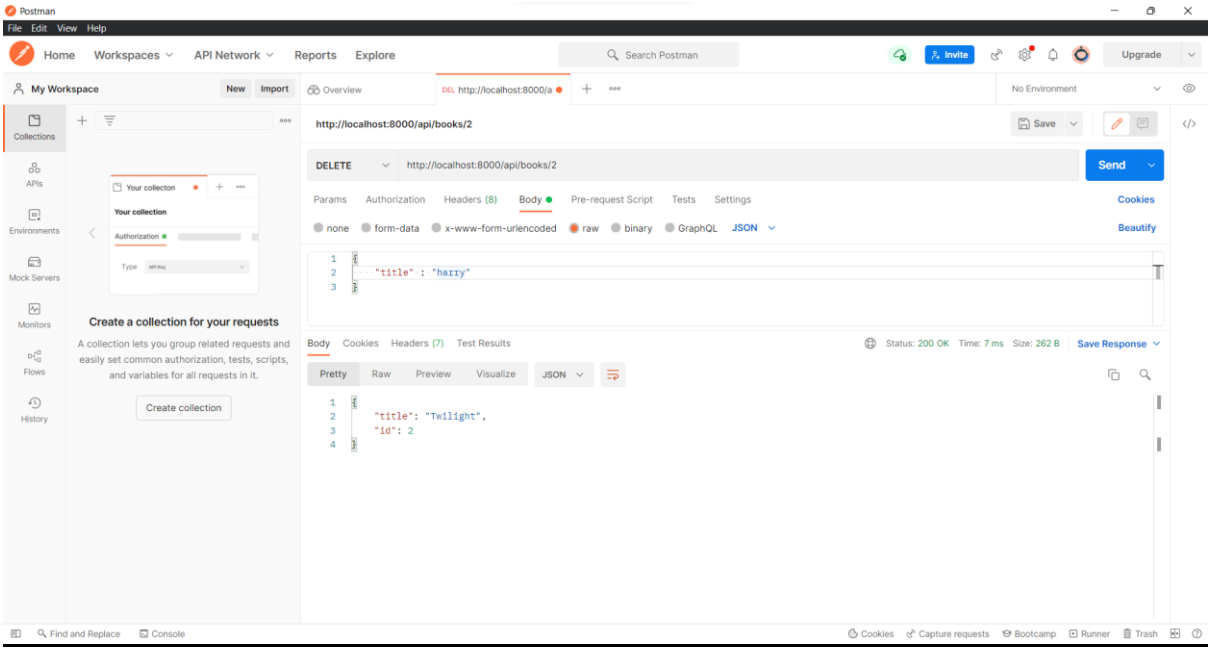
DELETE

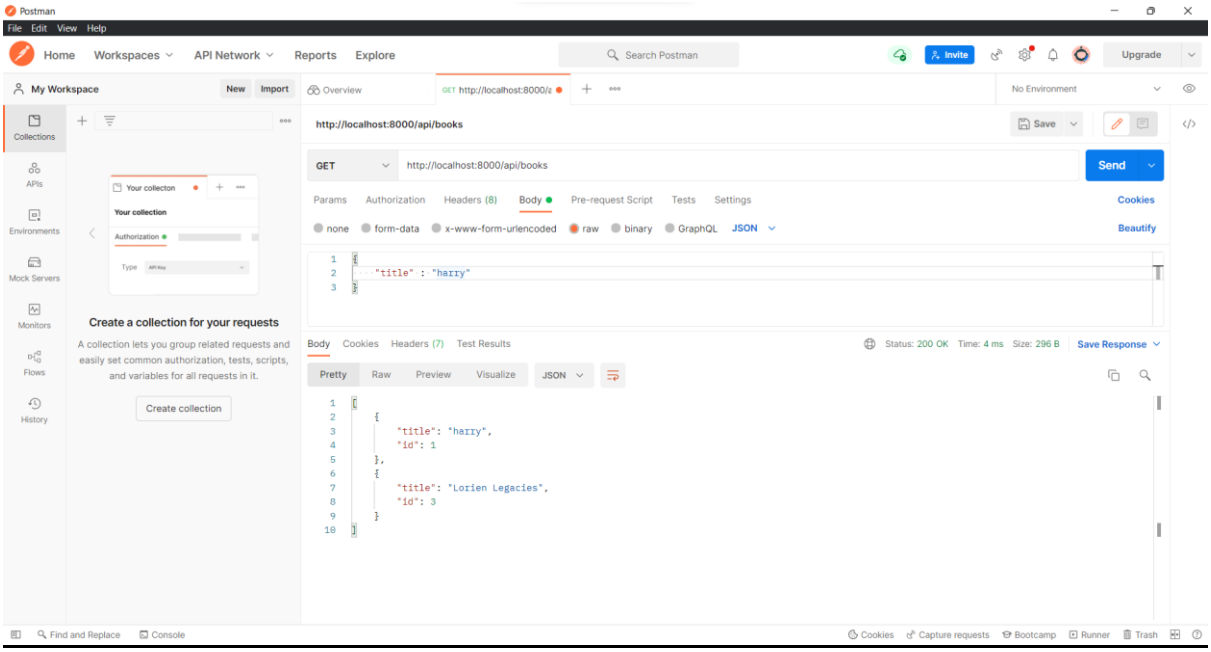
To delete a particular record, the corresponding URL with id and click send button localhost:800/api/books/2



Verifying the deletion via get request.







Result:

Thus, SOAP and RESTful Web services have been successfully implemented and the output is verified