Lab 7
RESTful Web Service

RESTful Web Service Programming with IntelliJ

Example of HTTP Request/Response

- **Client request (Header)**

```
GET /index.html HTTP/1.1 Host: www.example.com
```

- **Server response (Header)**

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux) Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b“
Accept-Ranges: none Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Example of RESTful Web Service

- Openstack API is composed of RESTful Web Service

Ex) Virtual Machine Instance Creation Request API & Response (With JSON Format)

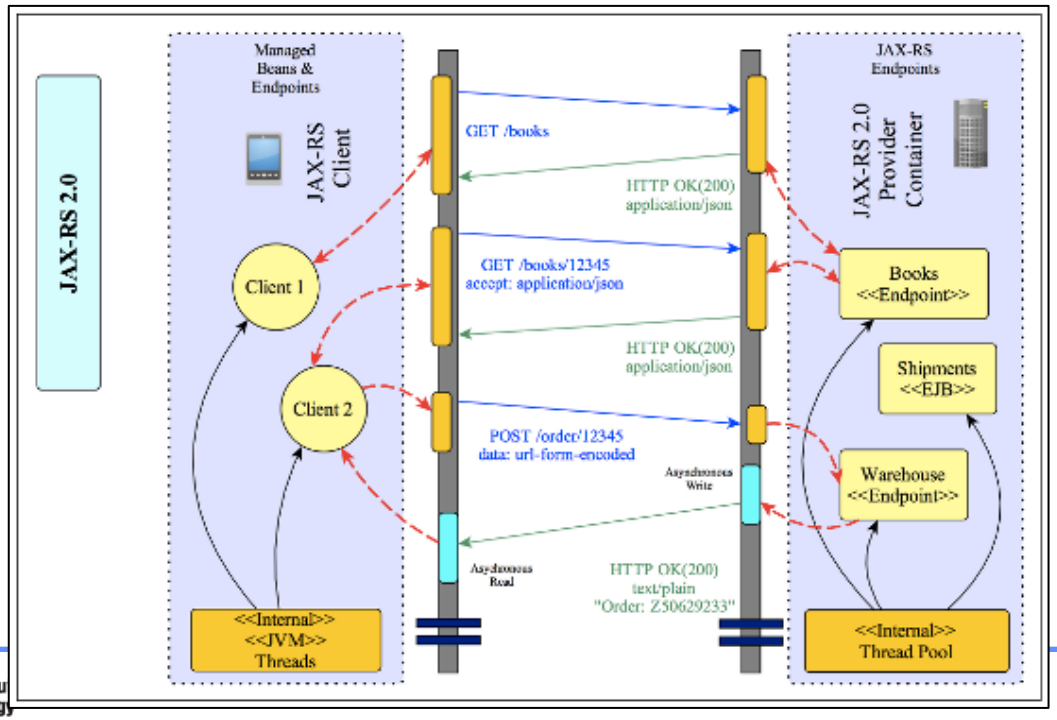
```
POST /v2/214412/servers HTTP/1.1 Host: servers.api.openstack.org Content-Type: application/json Accept: application/xml X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb

{
   "server" : {
      "name" : "new-server-test",
      "imageRef" :
      "http://servers.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f6f006e54",
      "flavorRef" : "52415800-8b69-11e0-9b19-734f1195ff37"
   }
}
```

```
HTTP/1.1 200 OK Date: Mon, 12 Nov 2007 15:55:01 GMT Server: Apache Content-Length: 1863 Content-Type: application/xml; charset=UTF-8

{
   "server": {
      "id": "52415800-8b69-11e0-9b19-734f565bc83b",
      "tenant_id": "1234",
      "user_id": "5678",
      "name": "new-server-test",
      "created": "2010-11-11T12:00:00Z",
      "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
      "accessIPv4" : "67.23.10.138",
      "accessIPv6" : "::babe:67.23.10.138",
      "progress": 0,
      "status": "BUILD",
      "adminPass": "GFf1j9aP",
      "image" : {
         "id": "52415800-8b69-11e0-9b19-734f6f006e54",
         "name": "CentOS 5.2",

...
```

For more API specification, refer the document http://docs.openstack.org/api/api-specs.html

JAX-RS

- JAX-RS: Java API for RESTful Web Services (JAX-RS)
  - A Java programming language API spec to support the Representational State Transfer (REST) architecture.
  - JAX-RS uses annotations, to simplify the development and deployment of web service clients and endpoints.
  - Specification in https://github.com/jax-rs



JAX-RS

- JAX-RS provides some annotations to aid in mapping a resource class (a POJO) as a web resource. The annotations include:
  - *@Path* specifies the relative path for a resource class or method.
  - *@GET, @PUT, @POST, @DELETE and @HEAD* specify the HTTP request type of a resource.
  - *@Produces* specifies the response Internet media types (used for content negotiation).
  - *@Consumes* specifies the accepted request Internet media types.

Implementation of JAX-RS

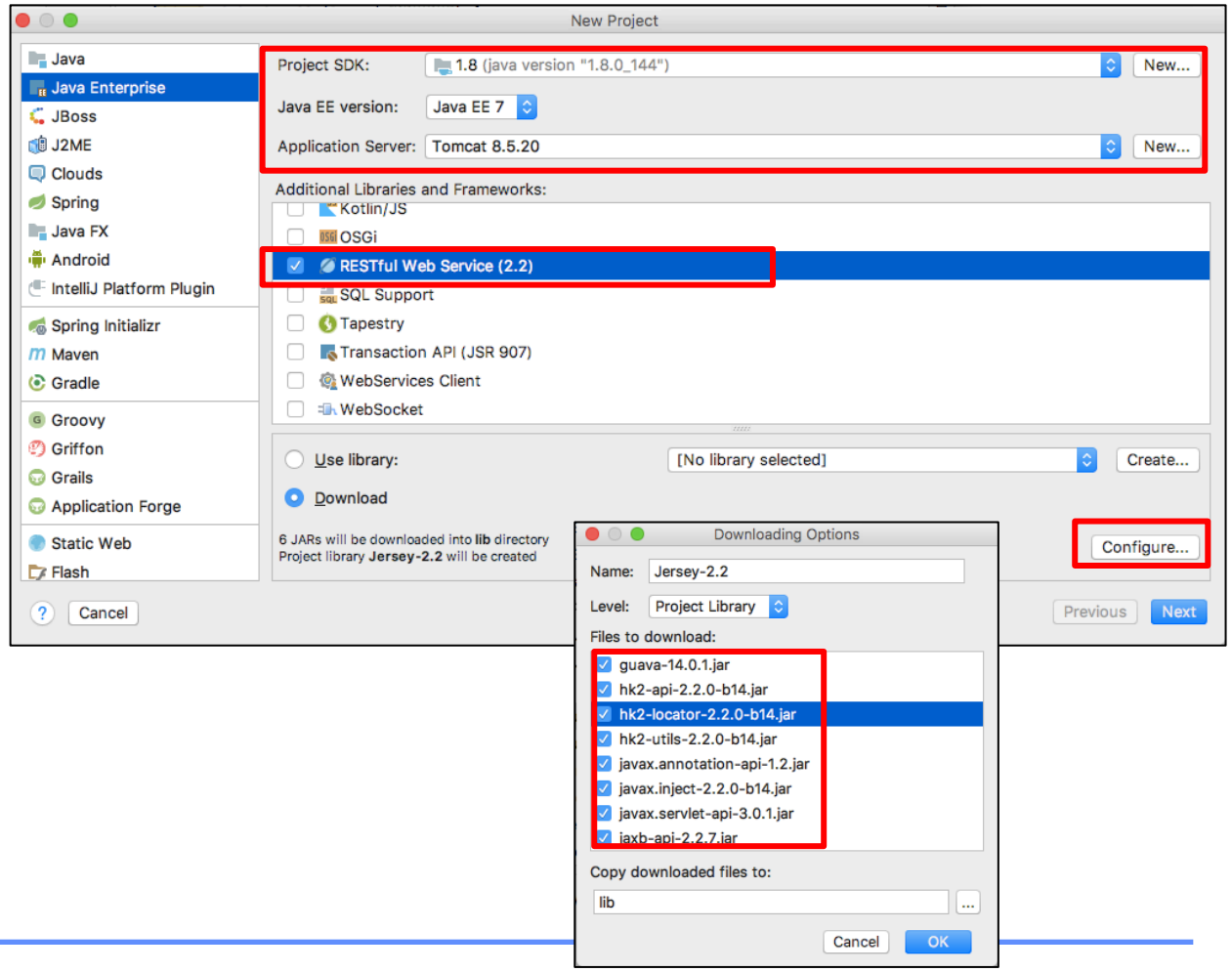
- Implementations of JAX-RS include:
  - Apache CXF, an open source Web service framework.
  - **Jersey**, the reference implementation from Sun (now Oracle).
  - RESTeasy, JBoss's implementation.
  - Restlet, created by Jerome Louvel, a pioneer in REST frameworks.
  - Apache Wink, Apache Software Foundation Incubator project, the server module implements JAX-RS.
  - WebSphere Application Server from IBM via the "Feature Pack for Communications Enabled Applications"
  - WebLogic Application Server from Oracle, see notes

Jersey

- http://jersey.java.net
- Jersey is the open source, production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.
- Also more than the Reference (JAX-RS) Implementation. Jersey provides an API so that developers may extend Jersey to suit their needs
- Download Link: http://jersey.java.net/nonav/documentation/latest/chapter_deps.html

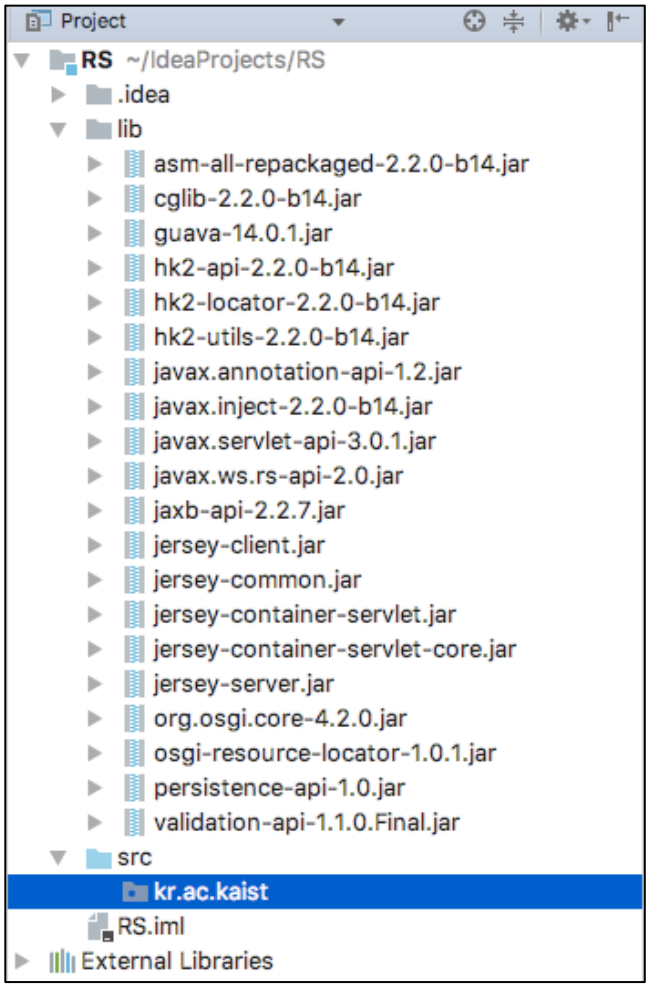
Create Web Service Project

1. Java Enterprise → Web Application → RESTful Web Service
2. Select Application Server as 'Tomcat'
3. In lib configuration, **select all possible libraries**



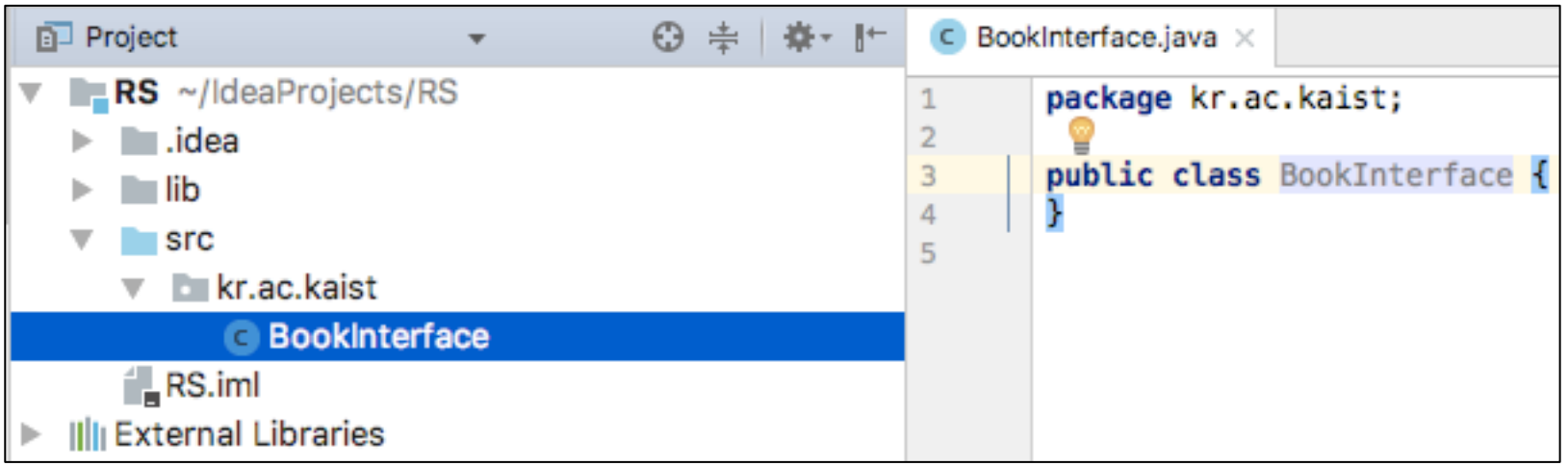
RESTful Service Implementation

4. Jersey library files will be downloaded to /lib directory
5. Create package ‘kr.ac.kaist’ and java class to /src directory



RESTful WS Programming

6. Under package ‘kr.ac.kaist’, create Java Class named ‘BookInterface’



Interface Description with JAX-RS Annotation

7. Copy and paste source code from lab material page

```
public class BookInterface {

    public String greeting() {
        System.out.println("User arrived!");
        return "Hello";
    }
}
```

Code at Lab Materials
- Lab7/src/BookInterface0.java

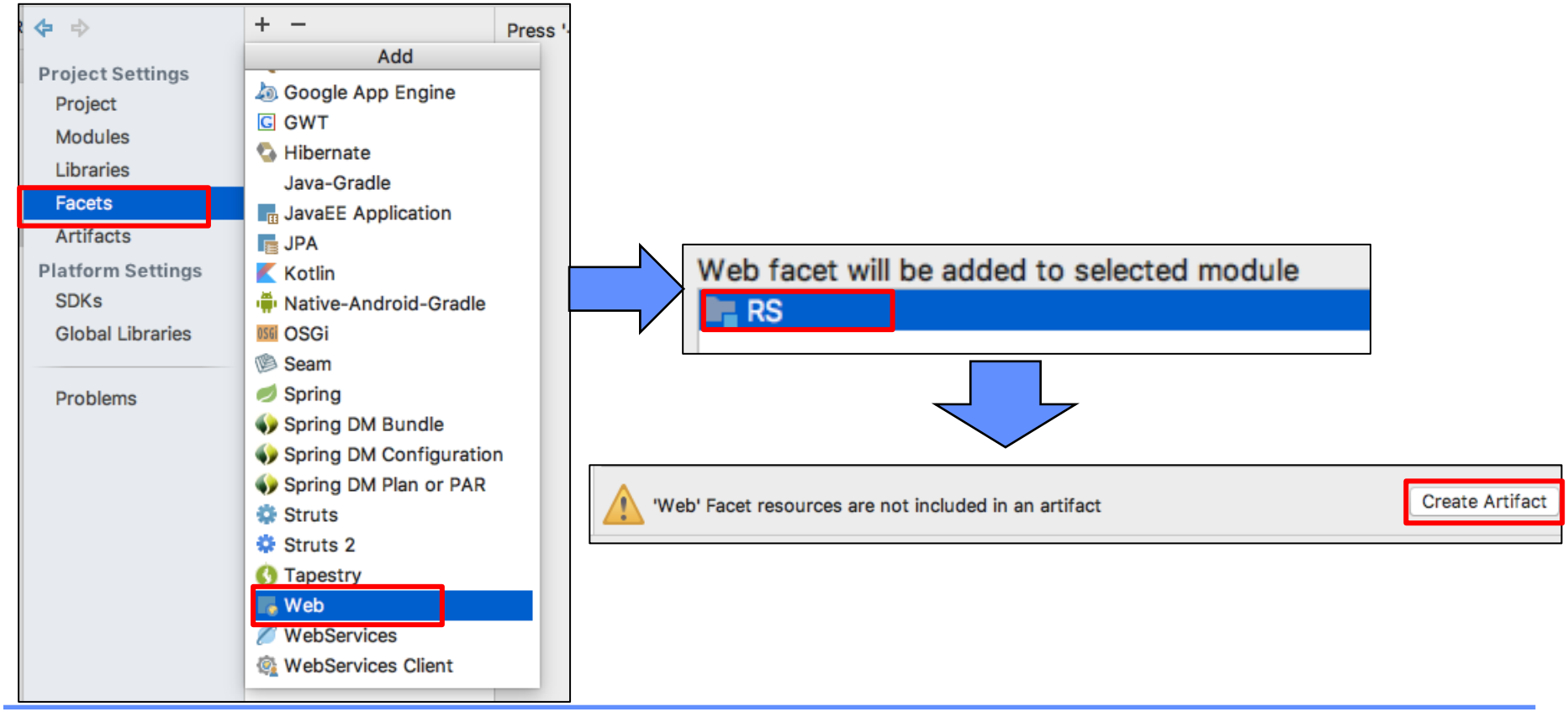
```
@Path("/Book")
public class BookInterface {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String greeting() {
        System.out.println("User arrived!");
        return "Hello";
    }
}
```

JAX-RS Annotation API for declaring router path

JAX-RS Annotation API for declaring method

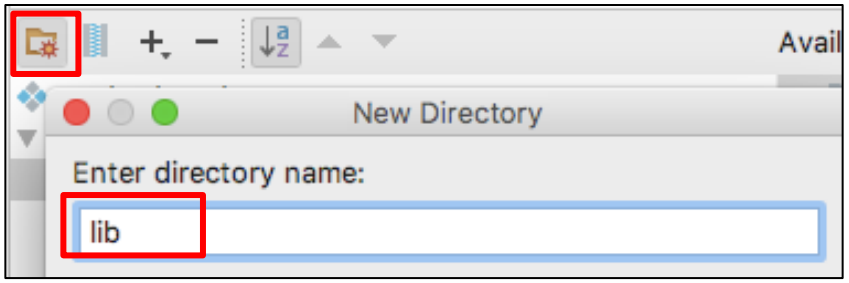
Add Facets

8. In the project setting, add ‘web’ facet to the project
9. Create ‘Web exploded’ artifact to the project

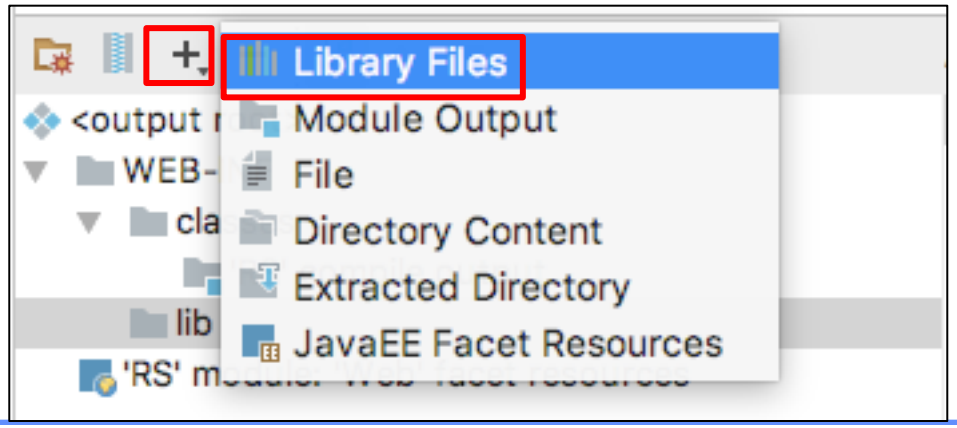


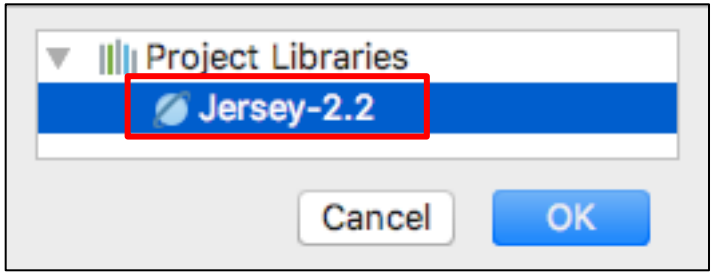
Configure Artifacts

10. Create lib dir into WEB-INF



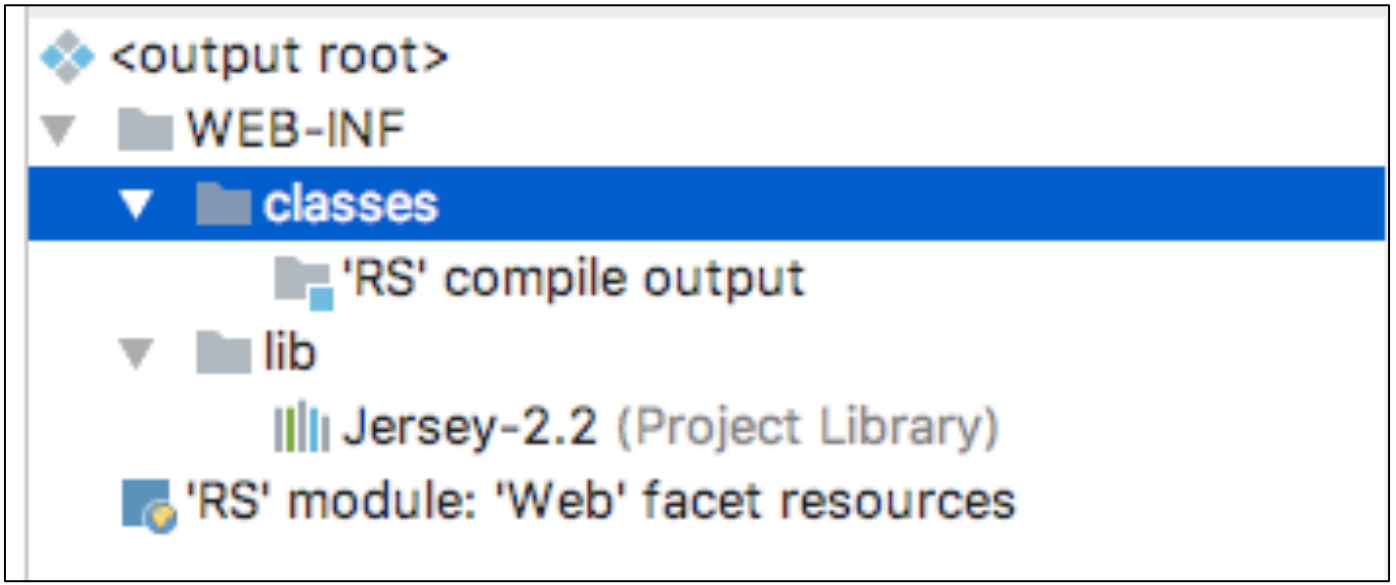
11. Add a jersey library into WEB-INF/lib Directory





Configure Artifacts

12. The artifacts should be set to the following structure



Web Configuration

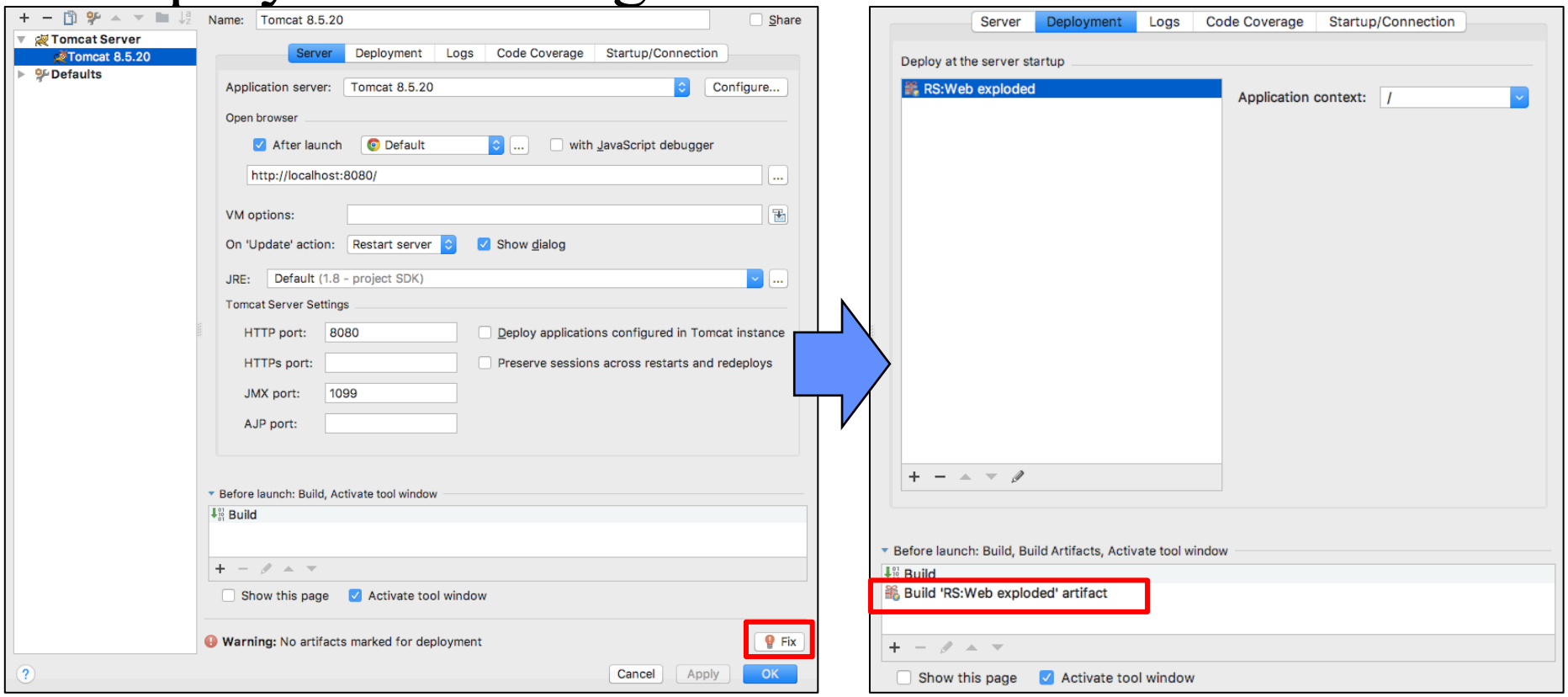
13. Edit web/WEB-INF/web.xml for declaring Web application

Code at Lab Materials
- Lab7/src/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>kr.ac.kaist</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

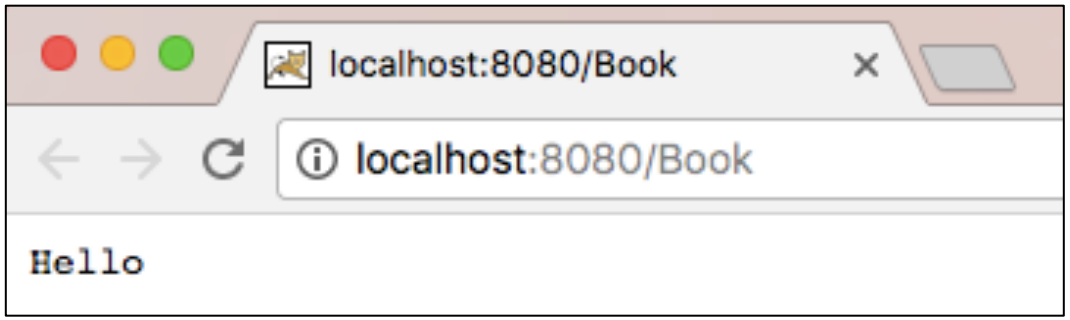
Deploy artifacts

14. Add ‘RS:Web exploded’ artifacts to deployment setting



Deploy service with IDE

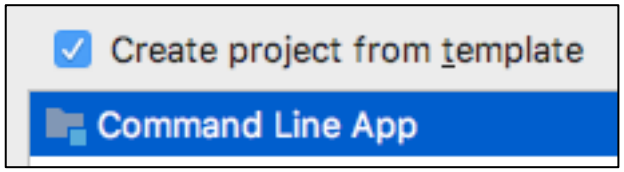
15. Run → Run 'Tomcat'
16. Access a page 'http://localhost:8080/Book' with your web browser



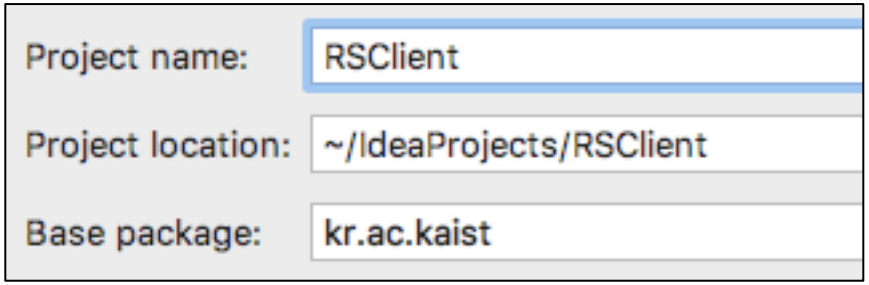
RESTful Client

Create default JAVA Project

1. File → New → Project → Next
2. Create project with template

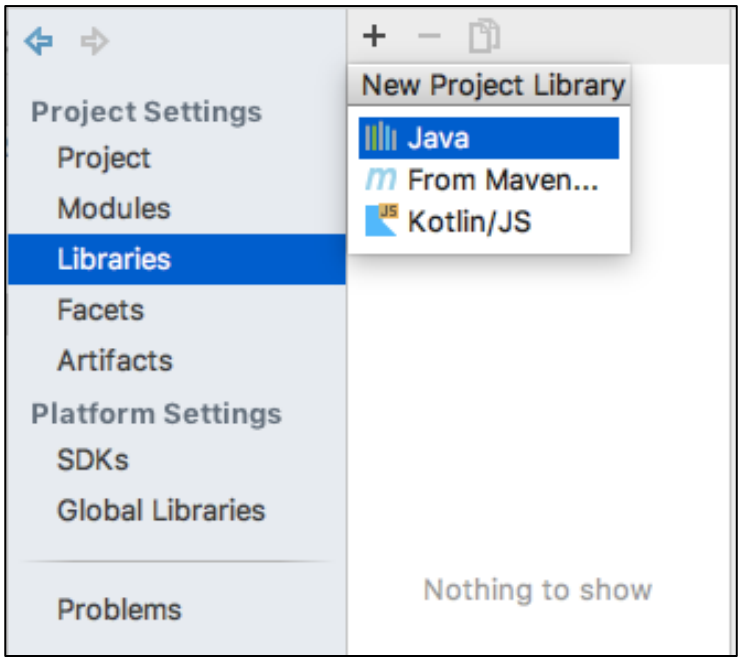


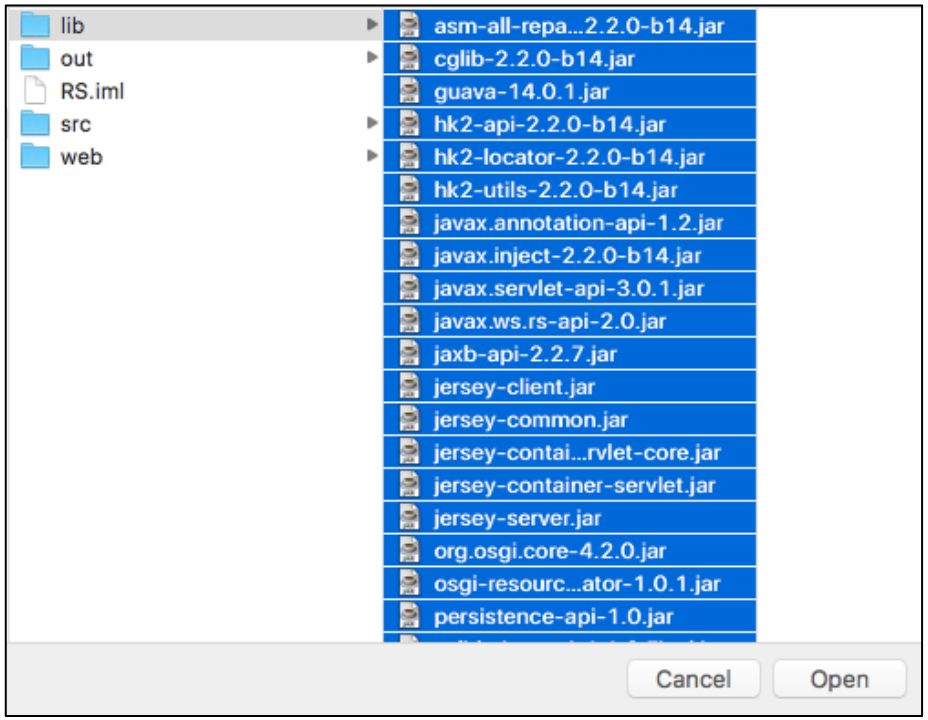
3. Specify package name



Library Import

4. Import new JAVA Libraries from previous 'RESTful server'





RESTful Client Implementation

5. Copy and paste source code from lab material page

```
package kr.ac.kaist;

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.Response;

public class Main {

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target( s: "http://localhost:8080/Book");
        Response res = target.request(MediaType.TEXT_PLAIN).get();
        String entity = res.readEntity(String.class);

        System.out.print(String.format("Status: %d\nEntity: %s\n", res.getStatus(), entity));
    }
}
```

Code at Lab Materials
- Lab7/src/Main0.java

6. Run client program

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java ...
Status: 200
Entity: Hello

Process finished with exit code 0
```

Object Representation with JSON

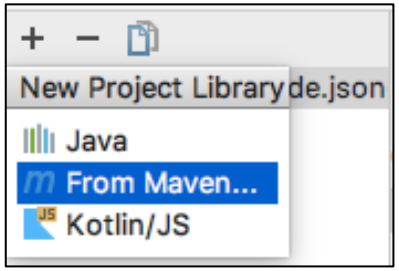
Book Class Example

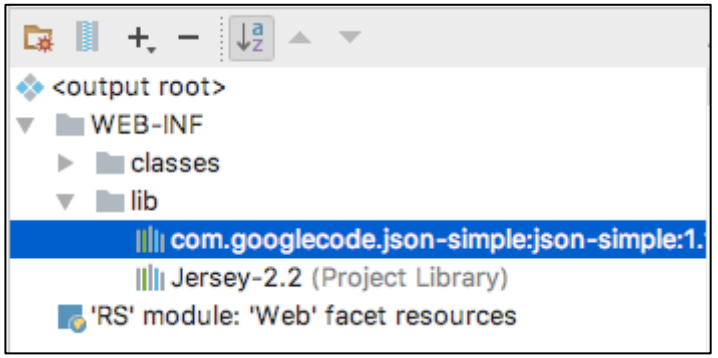
- Simple Book Class

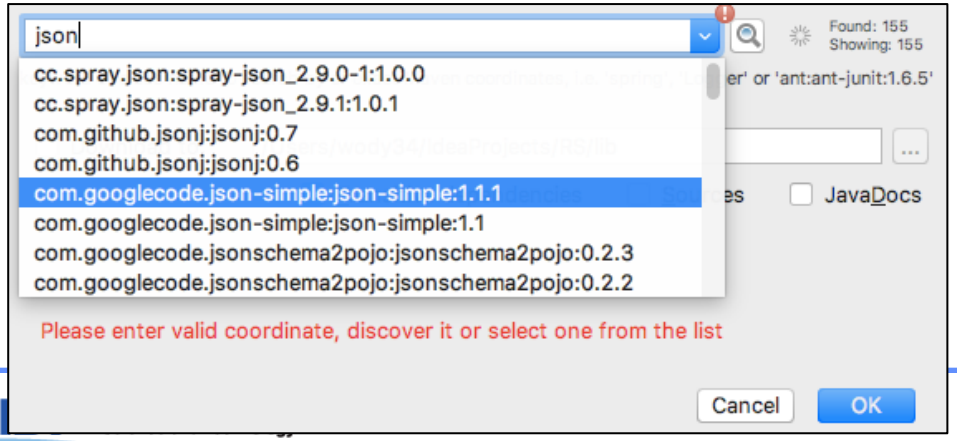
```
public class Book {
    private String name;
    private int price;

    public Book(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }
}
```

Add JSON Library

1. Download ‘json-simple’ library from maven repository and add it to the artifacts in both client and server project







JSON Enabling

2. Copy and paste source code from lab material page to server project and execute it

Code at Lab Materials
- Lab7/src/Book0.java

- New representation of book object
  - Interoperable
  - Text-based
  - Descriptive

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/C
{"price":1,"name":"UNP"}
```

```
public class Book {
    private String name;
    private int price;

    public Book(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }

    public String toJSON() {
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("name", this.name);
        jsonObject.put("price", this.price);
        return jsonObject.toJSONString();
    }

    public static void main(String[] args) {
        Book book = new Book( name: "UNP",  price: 1);
        System.out.println(book.toJSON());
    }
}
```

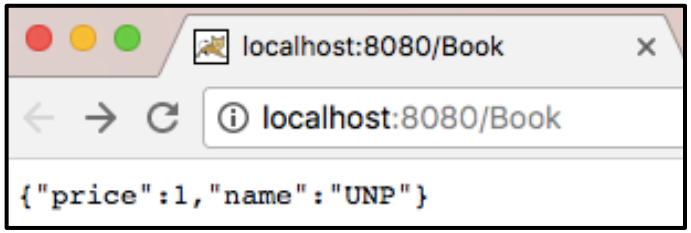
JSON in RESTful Server

3. Copy and paste source code from lab material page and access service using web browser

- We can get Book Object through RESTful Service

Code at Lab Materials
- Lab7/src/BookInterface1.java

```
package kr.ac.kaist;

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

@Path("/Book")
public class BookInterface {
    private Book book = new Book( name: "UNP",  price: 1);

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public String getBook() {
        return book.toJSON();
    }
}
```



JSON in RESTful Client

4. Copy and paste source code from lab material page to client project and execute it

- Book object is de-serialized using JSON Parser

Code at Lab Materials
- Lab7/src/Book1.java

```
public class Book {
    private String name;
    private int price;

    public Book(String json) {
        JSONParser parser = new JSONParser();
        try {
            JSONObject jsonObject = (JSONObject)parser.parse(json);
            this.name = (String) jsonObject.get("name");
            this.price = ((Long)jsonObject.get("price")).intValue();
        } catch(ParseException e) {
            e.printStackTrace();
        }
    }
```

Code at Lab Materials
- Lab7/src/Main1.java

```
public class Main {

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target( s: "http://localhost:8080/Book");
        Response res = target.request(MediaType.APPLICATION_JSON).get();
        String entity = res.readEntity(String.class);
        Book book = new Book(entity);
        System.out.print(String.format("Name: %s, Price: %d\n", book.getName(), book.getPrice()));
    }
}
```

Add functionalities

Add functionalities

1. Copy and paste source code ‘Book2’ and ‘BookInterface2’ to server project

- GET / POST / DELETE method is added
  - Create / Retrieve / Delete function is available

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getBookList() {...}

@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public String getBook(@PathParam("id") int id) {...}

@POST
@Consumes(MediaType.APPLICATION_JSON)
public String addBook(String json) {...}

@DELETE
@Path("/{id}")
public String deleteBook(@PathParam("id") int id) {...}
```

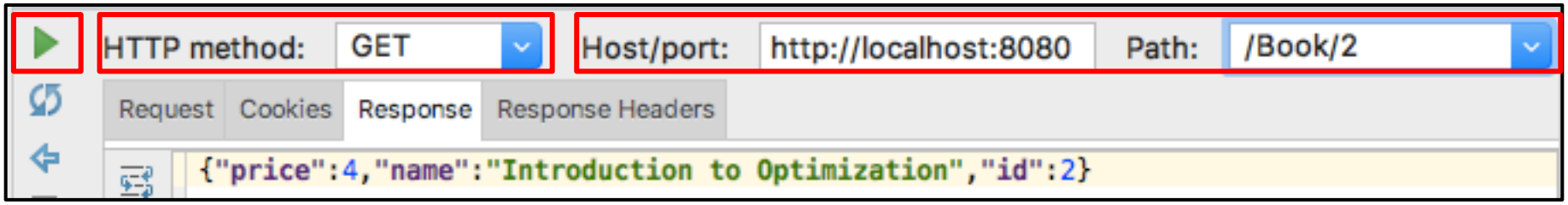
Code at Lab Materials
- Lab7/src/Book2.java
- Lab7/src/BookInterface2.java

API Specification

URL	Method	Request Body	Response Body
/Book	GET	N/A	[{"price":3,"name":"UNP","id":0}, {"price":1,"name":”LPG”,”id”:1}, {“price”:4,”name”:”ItO”,”id":2}]
/Book/{id}	GET	N/A	{"price":4,"name":"Introduction to Optimization","id":2}
/Book	POST	{"price":5, "name":"ABC"}	Result message in String
/Book/{id}	DELETE	N/A	Result message in String

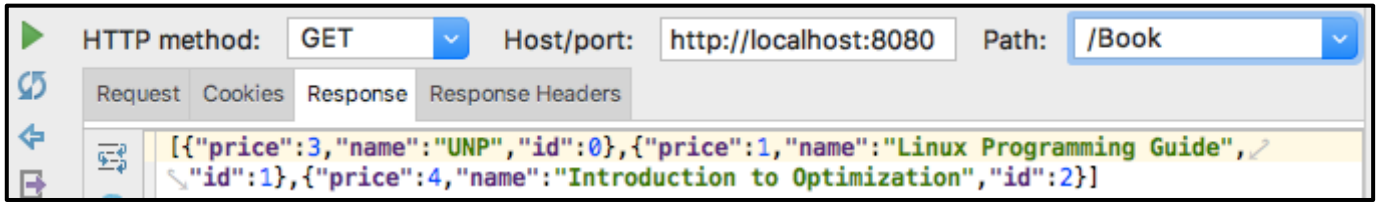
Test API with HTTP Client Tool

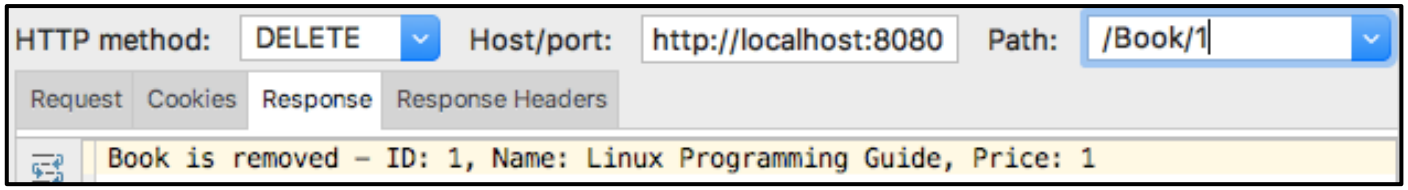
2. Tools → Test RESTful Web Service
3. Specify following attributes
   - Method (GET/POST/PUT/DELETE)
   - Access URL (IP / Port / Path)
   - Request Body (JSON Document)
4. Submit request

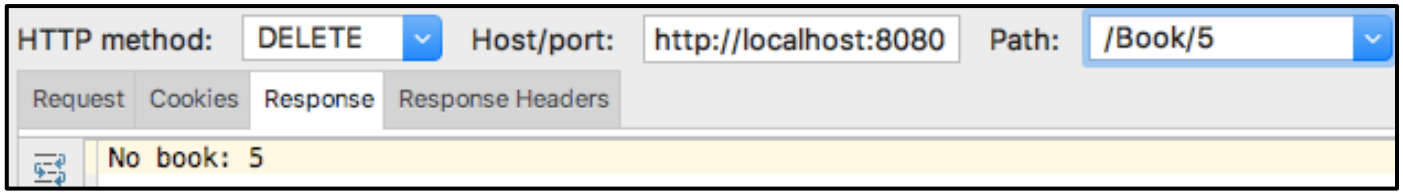


Test API with HTTP Client Tool

- Test API 1, 2 and 4

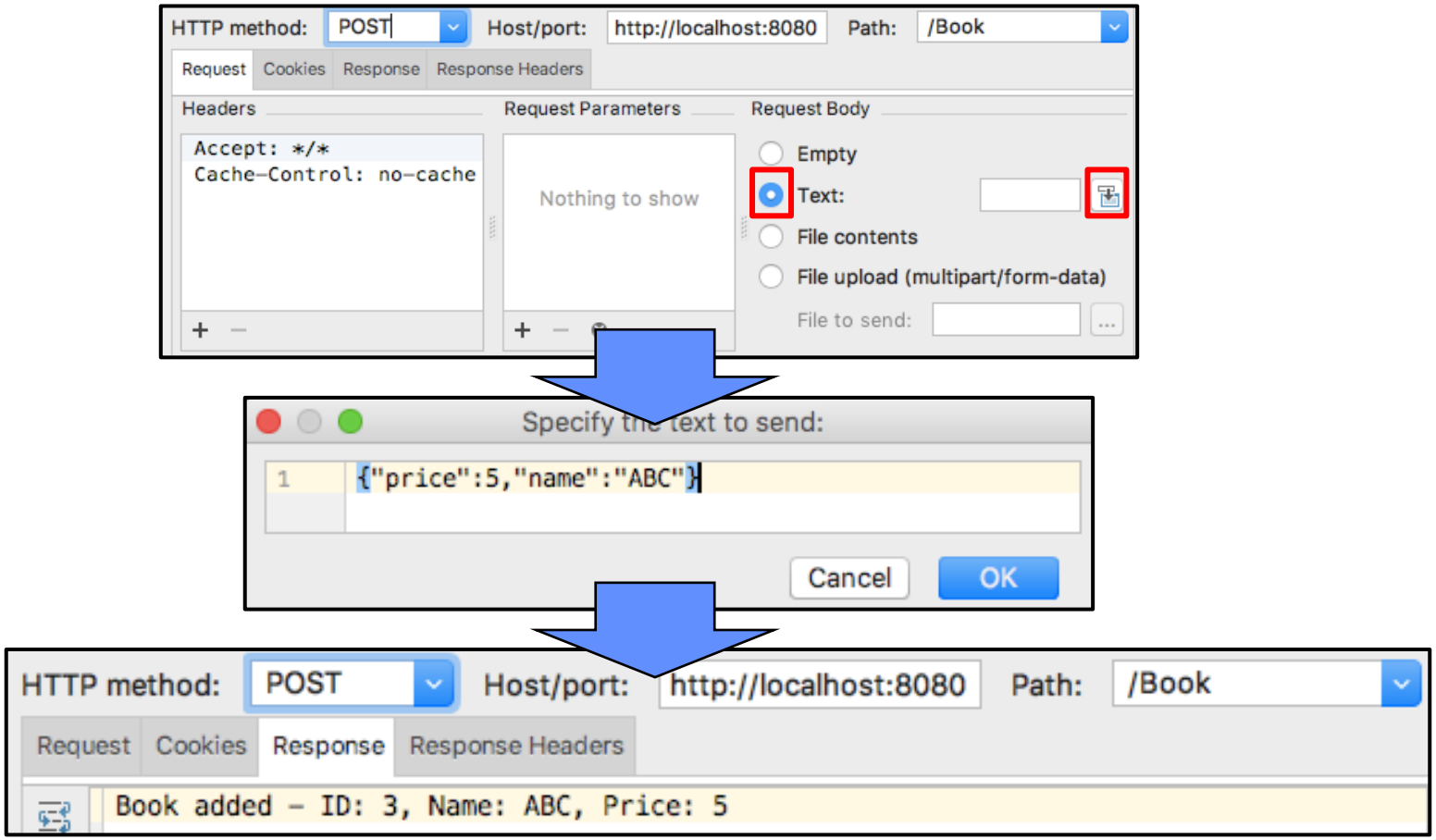






Test API with HTTP Client Tool

- Test API 3



Add client functionalities

5. Copy and paste source code ‘Book2’ and ‘Main2’ to client project
6. Restart server and run client code

- GET / POST / DELETE request handler is added
  - Create / Retrieve / Delete function is available

Code at Lab Materials
- Lab7/src/Book2.java
- Lab7/src/Main2.java

```
public static void main(String[] args) {
    Main client = new Main();
    ArrayList<Book> list;
    list = client.getBookList();
    client.addBook(new Book( name: "ABC",  price: 5));
    client.getBook( id: 0);
    list = client.getBookList();
    client.deleteBook( id: 5);
    client.deleteBook(list.get(1).getId());
    list = client.getBookList();
}
```

```
Get Book List from Server
 - ID: 0, Name: UNP, Price: 3
 - ID: 1, Name: Linux Programming Guide, Price: 1
 - ID: 2, Name: Introduction to Optimization, Price: 4
Book added - ID: 3, Name: ABC, Price: 5
ID: 0, Name: UNP, Price: 3
Get Book List from Server
 - ID: 0, Name: UNP, Price: 3
 - ID: 1, Name: Linux Programming Guide, Price: 1
 - ID: 2, Name: Introduction to Optimization, Price: 4
 - ID: 3, Name: ABC, Price: 5
No book: 5
Book is removed - ID: 1, Name: Linux Programming Guide, Price: 1
Get Book List from Server
 - ID: 0, Name: UNP, Price: 3
 - ID: 2, Name: Introduction to Optimization, Price: 4
 - ID: 3, Name: ABC, Price: 5
```

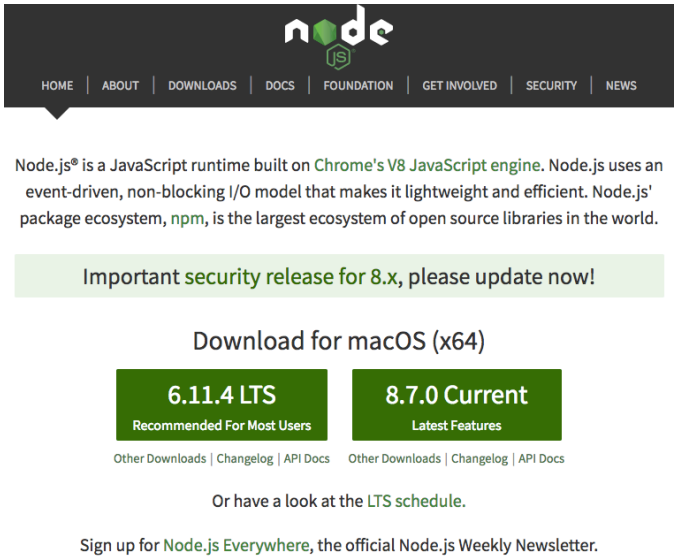
Lab 7. RESTful based Web Service

- Read text materials and test practices
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- Objectives
  - Understand the structure of RESTful Server
  - Learn how to implement a RESTful-based Web service using Node.js

HTTP Web Service using Node.js

Node.js

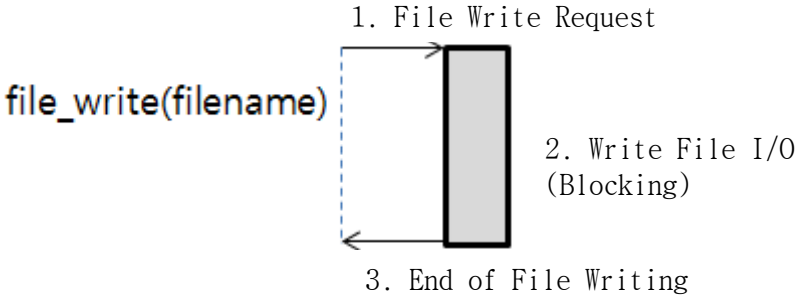
- Node.js is a software platform used to develop **scalable network applications**
- It utilizes JavaScript and has high throughput through non-blocking I/O and single-threaded event loops
- Efficient application for Node.js
  - Applications with frequent I / O
  - Data streaming applications
  - Application that handle data in real time
  - JSON API based applications
  - Single page application



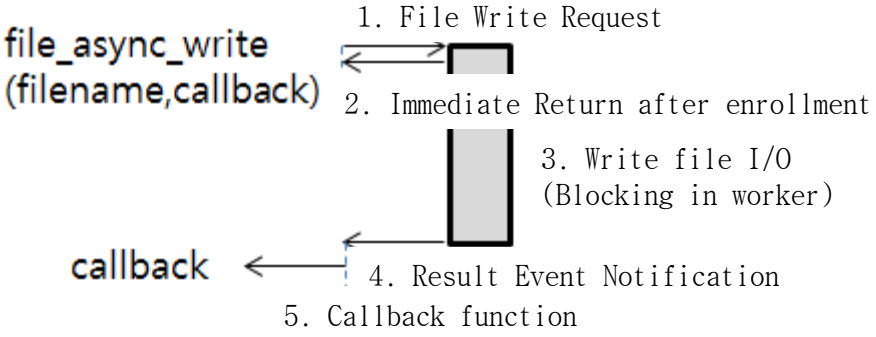
https://nodejs.org/en/

Sync / Async IO

- Synchronous I/O

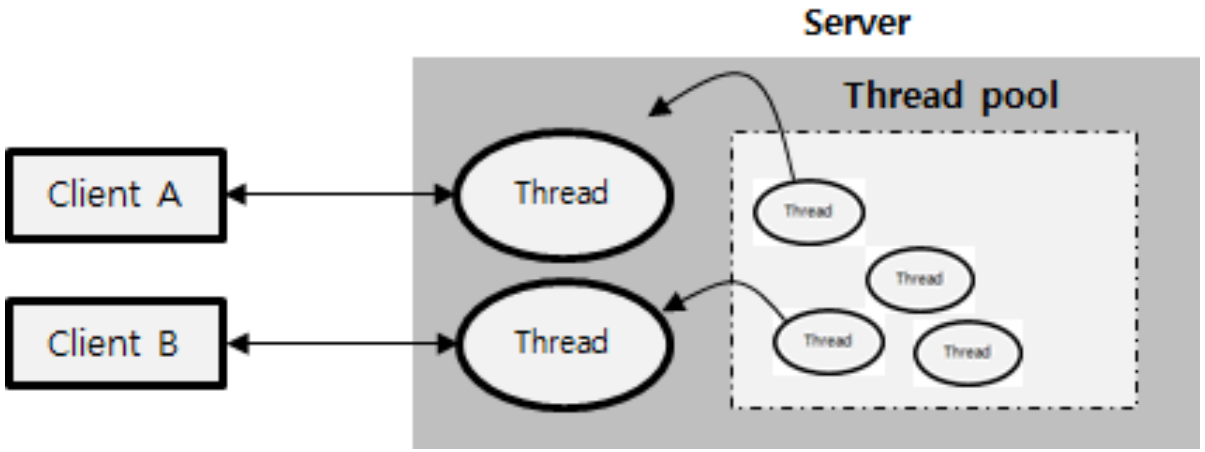


- Asynchronous I/O



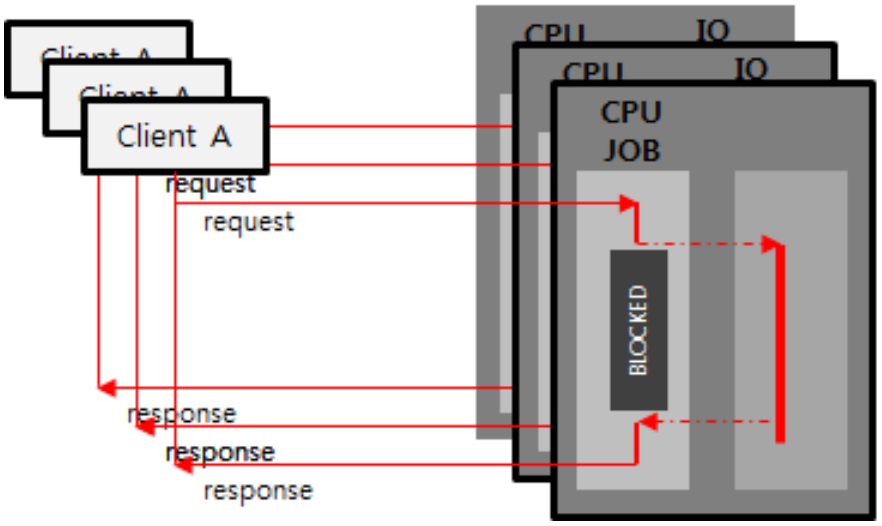
Tomcat vs Node.js

- Tomcat
  - Tomcat retrieves the thread from the thread pool, processes the client's request, and returns it to the thread pool.
  - The number of clients that can be serviced concurrently is equal to the number of threads, but the number of threads is physically limited in the system.
  - For example, Tomcat can create about 500 threads. There is a limit to the number of clients that can be processed at the same time.



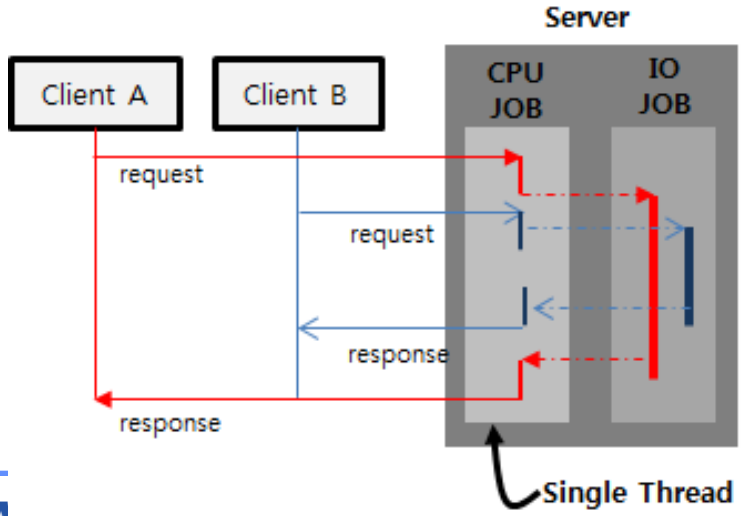
Tomcat vs Node.js

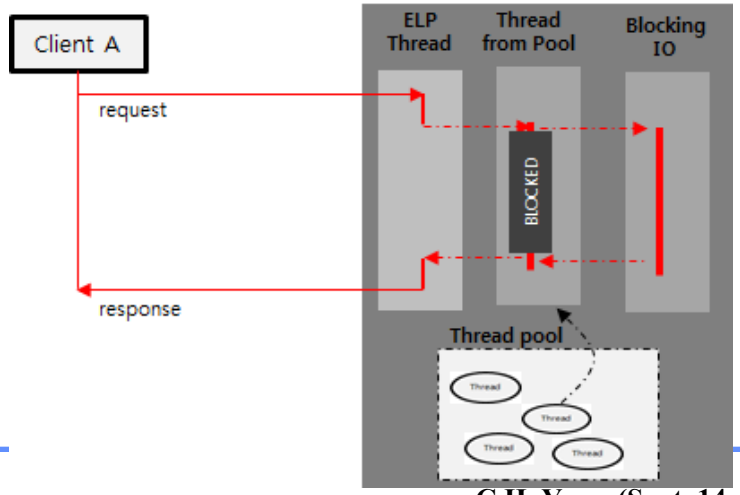
- Tomcat
  - In terms of IO efficiency, the Thread allocated to the Client maintains a Wait state when the IO task (DB, Network, File) exists and the Thread does not use the CPU.



Tomcat vs Node.js

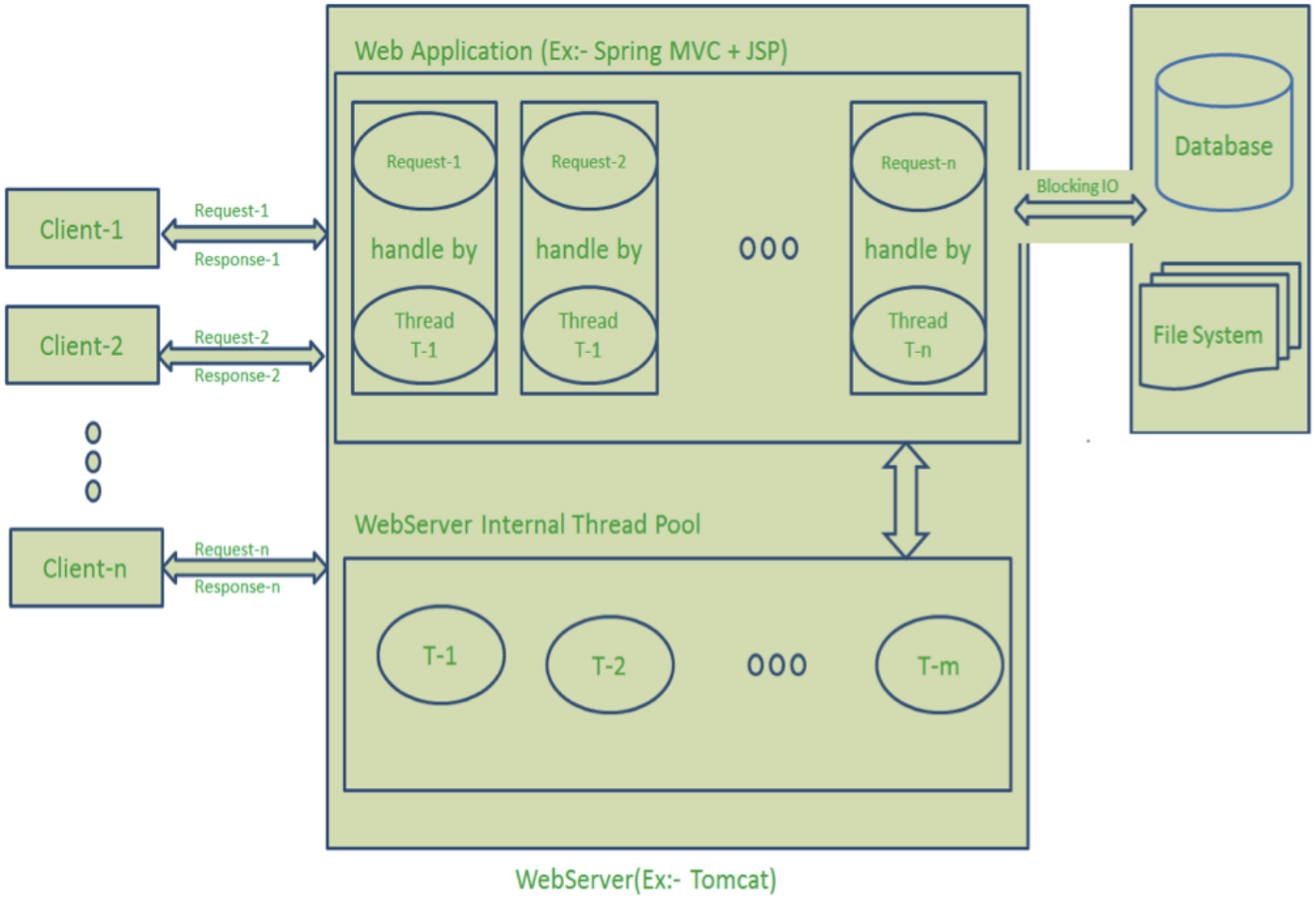
- Node.js
  - An asynchronous server based on single thread handles requests from multiple clients using only one thread.
  - The IO request is enrolled by the asynchronous IO method. While, the IO operation is processed, another operation is performed.
  - For example, some I/Os such as file open do not support the non-blocking function depending on the OS, so we utilize a multi-thread pool to prevent the blocking of the Event Loop Thread





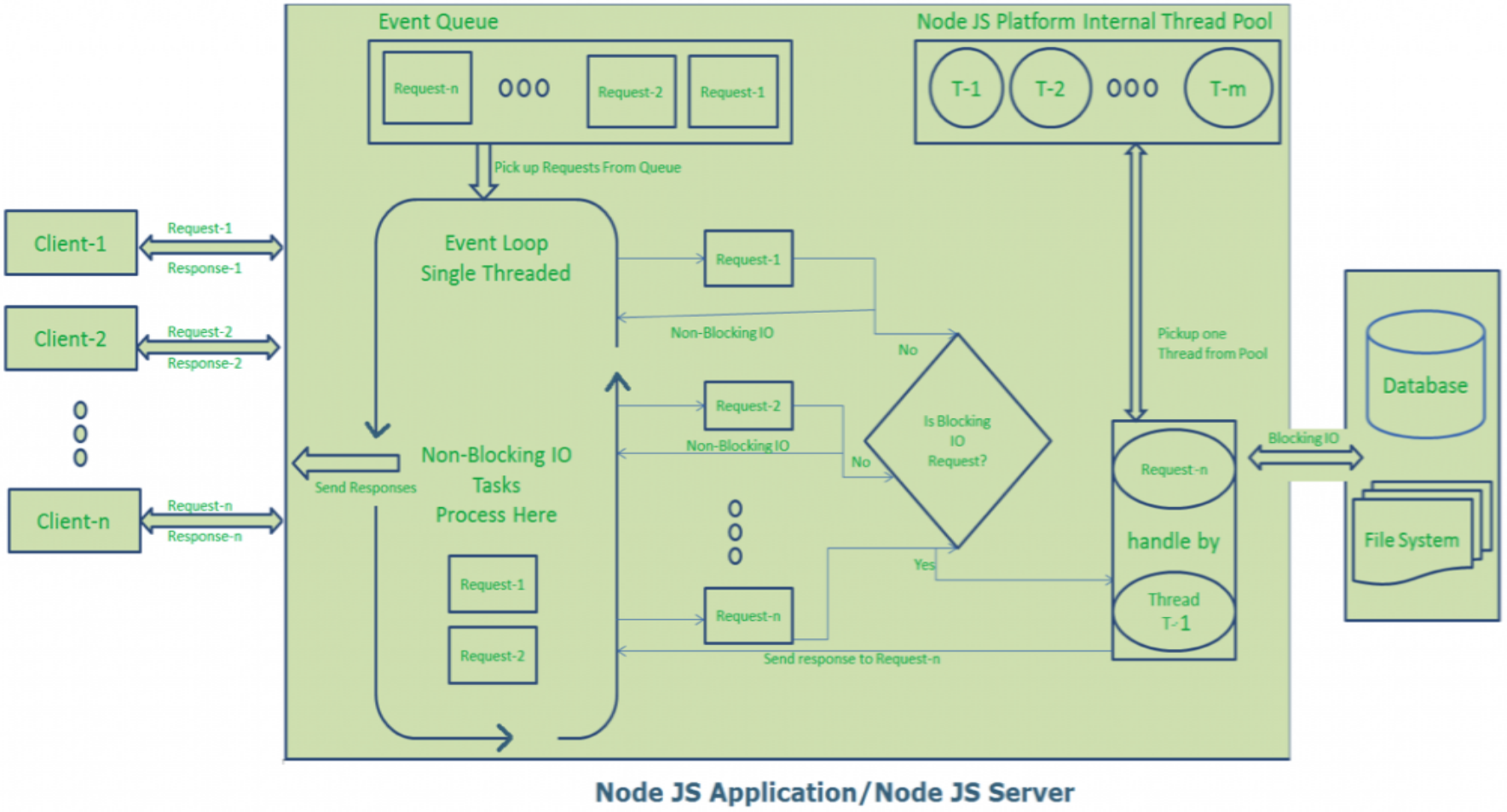
Detail Architecture

- Tomcat



Detail Architecture

- Node.js



NodeJS Installation

https://nodejs.org/en/download/package-manager/

- NodeJS HTTP server in Linux Environment
  - NodeJS(version 6.x) installation (http://nodejs.org/en/)

```
$ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
$ sudo apt-get install -y nodejs
```

  - NPM installation (http://docs.npmjs.com/getting-started/what-is-npm)

```
$ sudo apt-get install npm
```

- NodeJS HTTP server in MacOS

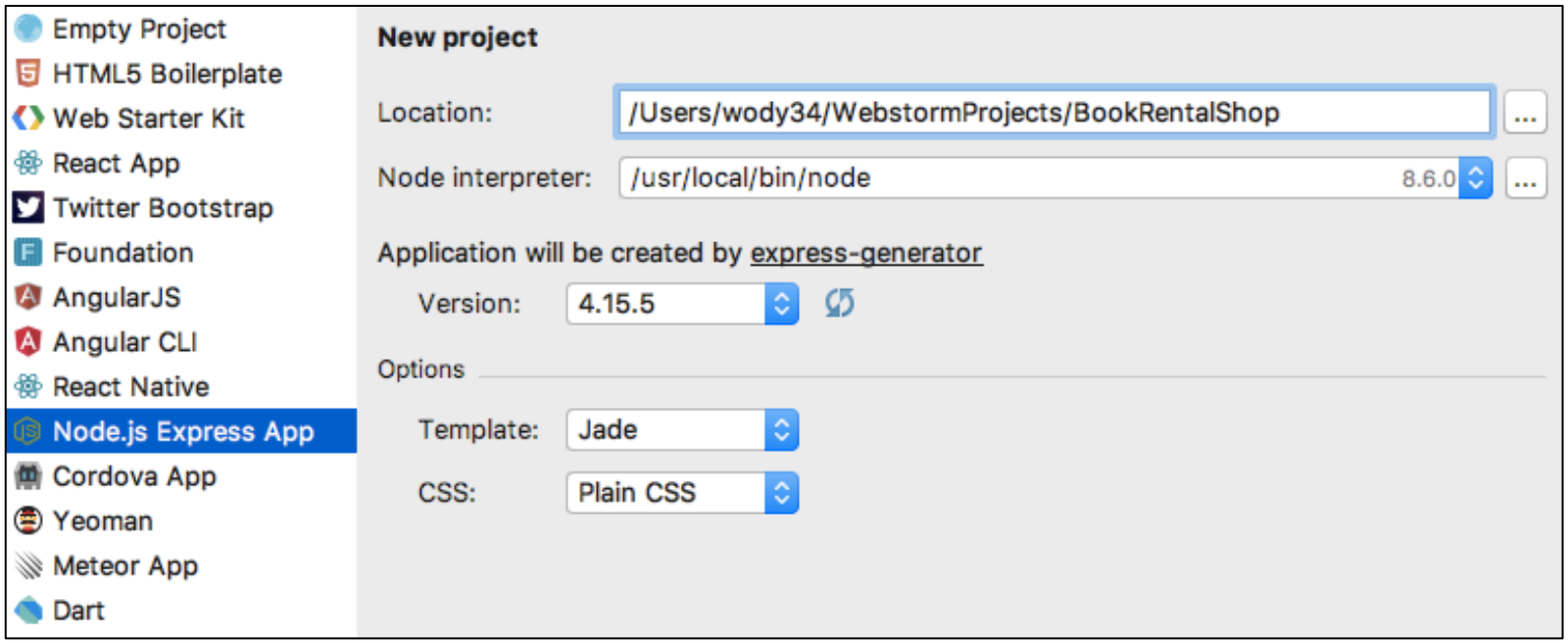
```
$ brew install node
```

IDE Installation

- Download and Install WebStorm from https://www.jetbrains.com/webstorm/

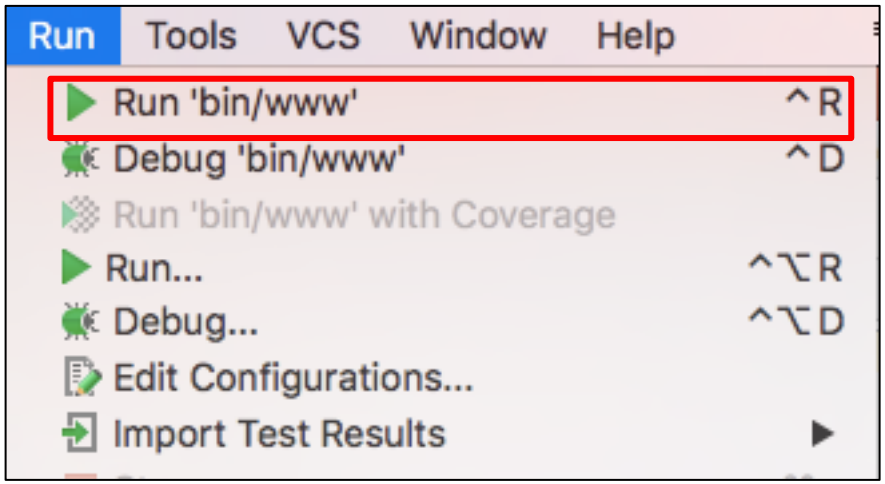
NodeJS based RESTful server

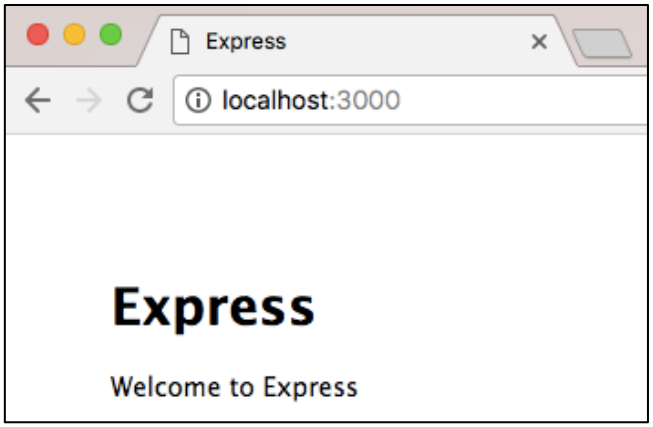
- Project Directory & initialize project
  - Create Project → Node.js Express App → Create

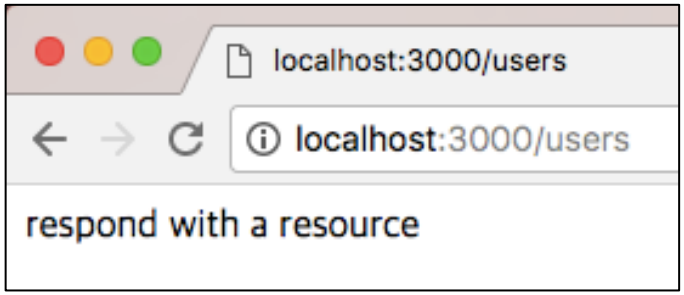


NodeJS based RESTful server

- Deploy server
  - Run → Run ‘bin/www’
  - Access to localhost:3000 and localhost:3000/users using Web Browser

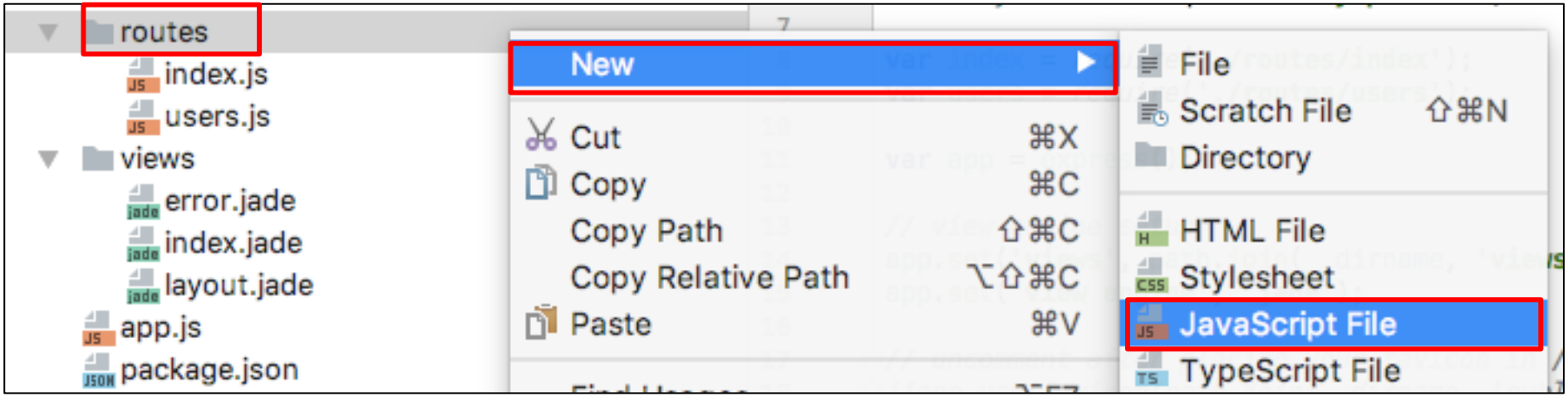






NodeJS based RESTful server

- Add a resource named book
  - Create book.js in routes directory



  - Link new routing rule to the application setting (Edit app.js)

```
var index = require('./routes/index');
var users = require('./routes/users');
var book = require('./routes/book');
```

```
app.use('/', index);
app.use('/users', users);
app.use('/book', book);
```

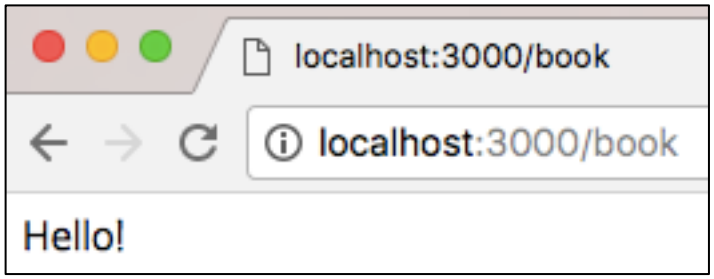
Entire Code at Lab Materials
- Lab7/src/app.js

NodeJS based RESTful server

- Create sample method in book resource
  - Copy and paste code from users.js to book.js and edit response message

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res, next) {
    res.send('Hello!')
});

module.exports = router;
```

  - Restart application and access http://localhost:3000/book using Web Browser



NodeJS based RESTful server

- Implementation of API

URL	Method	Request Body	Response Body
/Book	GET	N/A	[{"price":3,"name":"UNP","id":0}, {"price":1,"name":"LPG","id":1}, {"price":4,"name":"ItO","id":2}]
/Book/{id}	GET	N/A	{"price":4,"name":"Introduction to Optimization","id":2}
/Book	POST	{"price":5, "name":"ABC"}	Result message in String
/Book/{id}	DELETE	N/A	Result message in String

NodeJS based RESTful server

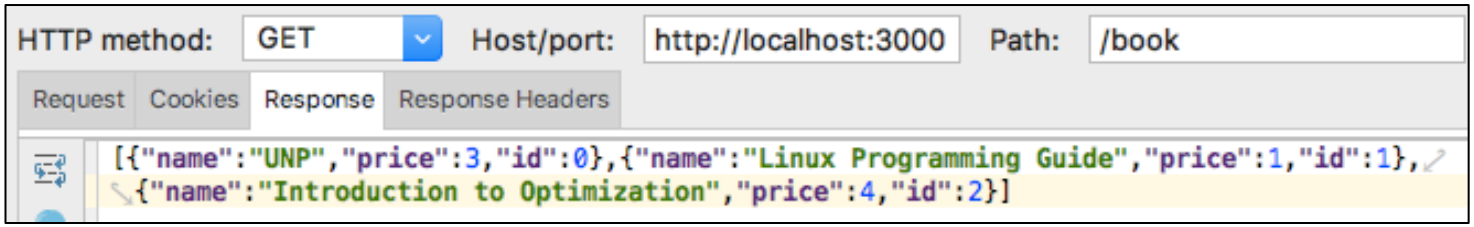
- Copy and paste source code from lab material page
- It is highly simpler than JAVA based RESTful implementation

```
router.get('/', function(req, res, next) {
    res.json(bookList)
});

router.get('/:id', function(req, res, next) {
    if((idx = findBook(req.params.id)))
        res.json(bookList[idx]);
    else
        res.json({error: 'no book exist'});
});

router.post('/', function(req, res, next) {
    addBook(req.body);
    res.json({msg: 'book enrolled'});
});

router.delete('/:id', function(req, res, next) {
    if((idx = findBook(req.params.id))) {
        bookList.splice(idx, 1);
        res.json({msg: 'book deleted'});
    }
    else
        res.json({error: 'no book exist'});
});
```

Entire Code at Lab Materials
- Lab7/src/book.js

NodeJS based RESTful server

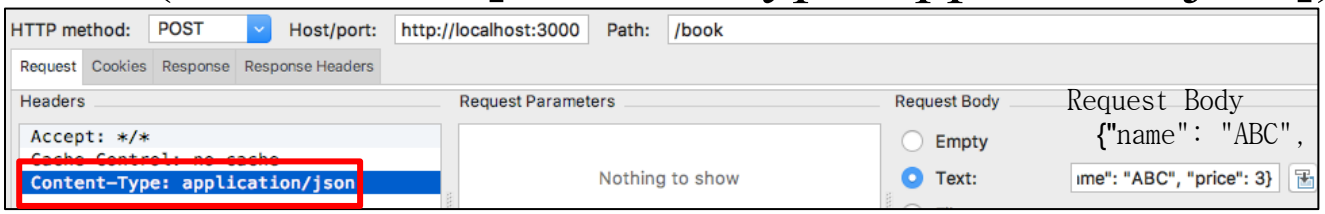
- Test API (Tools → Test RESTful Web Service)
  - API 1



  - API 2

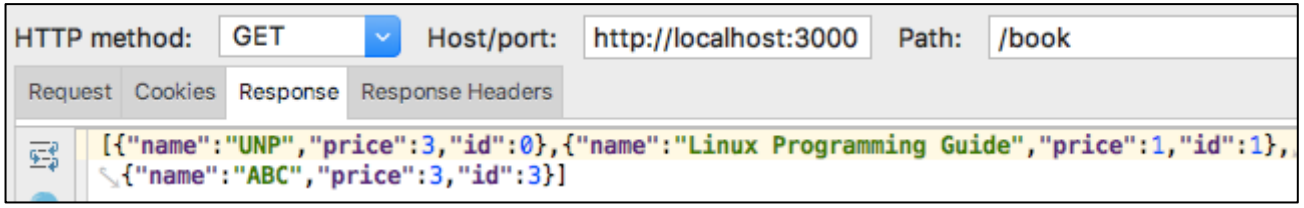
NodeJS based RESTful server

- Test API (Tools → Test RESTful Web Service)
  - API 3 (Add header [Contest-Type: application/json])



  - API 4

  - Result



RESTful Client Programming

- Add HTTP Client Library for Node.js
  - $ sudo npm --save install request
- Create client.js

```
var request = require('request');

request.get('http://localhost:3000/book', function (error, response, body) {
    console.log('error:', error);
    console.log('statusCode:', response && response.statusCode);
    console.log('body:', body);
});
```

- Launch client program
  - $ node client.js

```
SeongHwanKimui-MacBook-Air:BookRentalShop wody34$ node client.js
error: null
statusCode: 200
body: [{"name":"UNP","price":3,"id":0},{"name":"Linux Programming Guide","price":1,"id":1},{"name":"ABC","price":3,"id":3}]
```

RESTful Client Programming

- Add all capabilities about API

Entire Code at Lab Materials - Lab7/src/client0.js

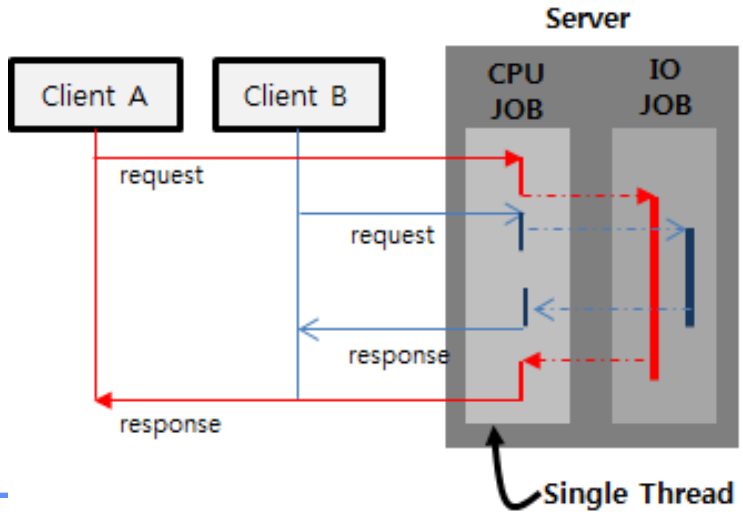
```
var request = require('request');

request.get('http://localhost:3000/book', function (error, response, body) {
    console.log(body);
});

request.get('http://localhost:3000/book/1', function (error, response, body) {
    console.log(body);
});

request.post('http://localhost:3000/book', {name: 'ABC', price: 3}, function (error, response, body) {
    console.log(body);
});

request.delete('http://localhost:3000/book/1', function (error, response, body) {
    console.log(body);
});
```

- What happened?

```
SeongHwanKimui-MacBook-Air:BookRentalShop wody34$ node client.js
[{"name":"UNP","price":3,"id":0},{"name":"Linux Programming Guide","price":1,"id":1},{"name":"ABC","price":3,"id":3}]
{"msg":"book deleted"}
{"msg":"book enrolled"}
{"error":"no book exist"}
```

Timing Problem in Asynchronous I/O

- We don't know which request get response firstly
- Unintentional procedure happened
- How can we know the end of request
  - When callback function called!



Solution for timing problem - callback

- Call request with callback function
- After request end, it call enrolled callback function

```
function getBookList(callback) {
    request.get('http://localhost:3000/book', function (error, response, body) {
        callback(body)
    });
}

function getBook(id, callback) {
    request.get('http://localhost:3000/book/' + id, function (error, response, body) {
        callback(body)
    });
}

function addBook(book, callback) {
    request.post('http://localhost:3000/book', book, function (error, response, body) {
        callback(body)
    });
}

function deleteBook(id, callback) {
    request.delete('http://localhost:3000/book/1', function (error, response, body) {
        callback(body)
    });
}
```

Entire Code at Lab Materials
- Lab7/src/client1.js

Solution for timing problem - callback

- When callback function called, we can know it is end of prior request
- Only after request end, create new request
- Sequence of request is implemented in cascaded way
- Hard to implement conventional program
  - Callback Hell!

```
getBookList(function (bookList) {
    console.log(bookList);
    //after end of getBookList Request

    getBook(1, function (book) {
        console.log(book);
        //after end of getBook Request

        addBook({name: 'ABC', price: 3}, function (msg) {
            console.log(msg);
            //after end of addBook Request

            deleteBook(2, function (msg) {
                console.log(msg);
                //after end of deleteBook Request

                getBookList(function (bookList) {
                    console.log(bookList);
                });
            });
        });
    });
});
```

Entire Code at Lab Materials
- Lab7/src/client1.js

```
SeongHwanKimui-MacBook-Air:BookRentalShop wody34$ node client.js
[{"name":"UNP","price":3,"id":0},{"name":"Linux Programming Guide","price":1,"id":1},{"name":"Introduction to Optimization","price":4,"id":2}]
{"name":"Linux Programming Guide","price":1,"id":1}
{"msg":"book enrolled"}
{"msg":"book deleted"}
[{"name":"UNP","price":3,"id":0},{"name":"Introduction to Optimization","price":4,"id":2},{"id":3}]
```

Performance Analysis on Web Services

Web server benchmarking test using AB (Apache HTTP server benchmarking tool)

- **Apache HTTP server benchmarking tool**
  - AB (Apache HTTP server benchmarking tool) is a very lightweight and useful web server benchmarking tool that uses the command line.
  - Benchmarking information can be obtained quickly and easily when testing simple REST APIs or static content.

AB HTTP Server Benchmarking

- **AB has known issues**
  - Do not interpret HTML, CSS, or image. It simply shows the response time.
  - Use an HTTP 1.0 client.
  - The -k KeepAlive option does not work because dynamic pages can not write Content-Length header content in advance.
  - You can not use the Transfer-Encoding: chunked option because it is an HTTP 1.0 client.
  - You can not use the option to delay between requests.
  - There is no option to delay between requests, so it can be considered a DDOS attack.

AB HTTP Server Benchmarking

Options	Exaplain
-n	The number of requests sent to check performance. By default, the request is sent only once, so you can not get a general performance check.
-c	The number of requests that are requested at the same time. Basically, it only sends one request at a time.
-g	Record all measured values in 'gnuplot' or TSV (tab separated values) file. The label refers to the first line of the output file.
-t	The maximum number of seconds to check for performance. Internally, we assume -n 50000. Used to check server performance for a specified time. Basically, it is checked without time limit.
-v	Specify the output level. If the value is greater than or equal to 4, information about the header is output. If the value is greater than or equal to 3, the response code is output (404, 202, etc.).
-A	Proxy provides BASIC Authentication information. Transmits base64 encoded user name and password separated by delimiter ‘:’.
-X	proxy[:port] Request using a proxy server.

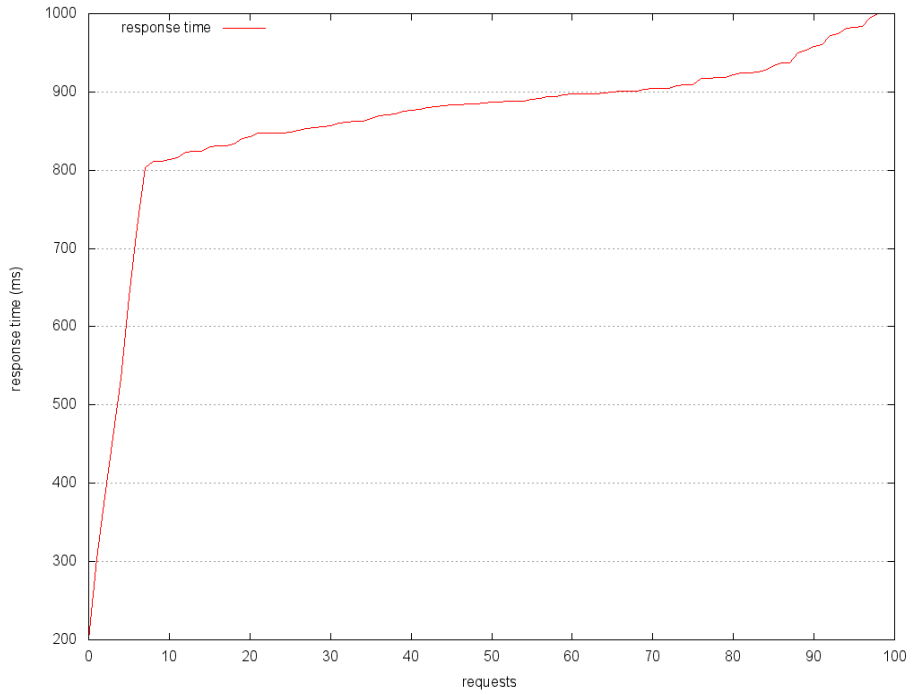
AB HTTP Server Benchmarking

```
$ ab -n 100 -c 10 -g result.plot http://lccalhost:3000
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation,
http://www.apache.org/

Benchmarking www.google.com (be patient).....done


Server Software:
Server Hostname:        www.google.com
Server Port:            80

Document Path:          /index.html
Document Length:        271 bytes

Concurrency Level:      10
Time taken for tests:   9.019 seconds
Complete requests:      100
Failed requests:        2
   (Connect: 0, Receive: 0, Length: 2, Exceptions: 0)
Non-2xx responses:      100
Total transferred:      49692 bytes
HTML transferred:       27096 bytes
Requests per second:    11.09 [#/sec] (mean)
Time per request:       901.890 [ms] (mean)
Time per request:       90.189 [ms] (mean, across all concurrent
requests)
Transfer rate:          5.38 [Kbytes/sec] received
```

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       75   89  11.7     86     136
Processing:    80  763 143.0    799     900
Waiting:       75  441 231.8    445     875
Total:        180  852 143.0    886    1000

Percentage of the requests served within a
certain time (ms)
  50%    886
  66%    900
  75%    909
  80%    919
  90%    953
  95%    981
  98%    995
  99%   1000
 100%   1000 (longest request)
```

Plot with gnuplot

```
starttime    seconds ctime   dtime   ttime   wait
Tue Jun 14 15:39:59 2016     1465886399   100 80
180 80
Tue Jun 14 15:39:59 2016     1465886399   102 100
201 100
Tue Jun 14 15:39:59 2016     1465886399   102 202
304 201
Tue Jun 14 15:39:59 2016     1465886399   78  305
383 304
Tue Jun 14 15:39:58 2016     1465886398   76  383
459 383
Tue Jun 14 15:39:58 2016     1465886398   75  459
534 459
Tue Jun 14 15:39:58 2016     1465886398   104 534
638 534
Tue Jun 14 15:39:58 2016     1465886398   89  638
727 638
...
```



Lab 8. Performance Comparison on Node.js and Tomcat Server

- Read text materials and test practices
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- Objectives
  - Learn how to use the apache benchmark to evaluate various web server performance
  - Learn how to obtain metrics that indices the performance of your Web service.
  - Analyze different performance according to the structure of web server and understand the reason