An-Najah National University

Distributed Operating Systems

A Multi-tier Online Book Store

Instructor:

Samer Arandi.

Students:

Laila Abu Safiya.

“Mohammad Kareem” Kukhun.

Introduction:

In this lab we are required to design Bazar.com using two tier we design (front-end and back-end), also we are recommended to use a lightweight micro web framework.

So, in our solution we use a Node JS frame work, its uses JavaScript on the server also its run-on various platforms.

Also we are recommended to use virtual machine for different server, so I build 3 virtual machines with ubuntu operating system, two of them used for the backend tier and the third for frontend tier, first one we use it for catalog server, and the second one we use it for order server, then we create a NAT network to connect 3 machines with each other's and with our device network which work in this case as a router, then when we want to test our microservices we will send it with the IP address for the device and the port that listening for the request.

Note: here we convert the IP address for the machines to static IP's because we need to guarantee the IP in any request to avoid change it always if its dynamic.

In our program we have two files (catalog.csv & order.csv) which its consider as the database of servers that have the information for our program. At first in my program, we read each of them and store them in array of objects (catalog, Book_Sold) in JSON format to use it in our API's.

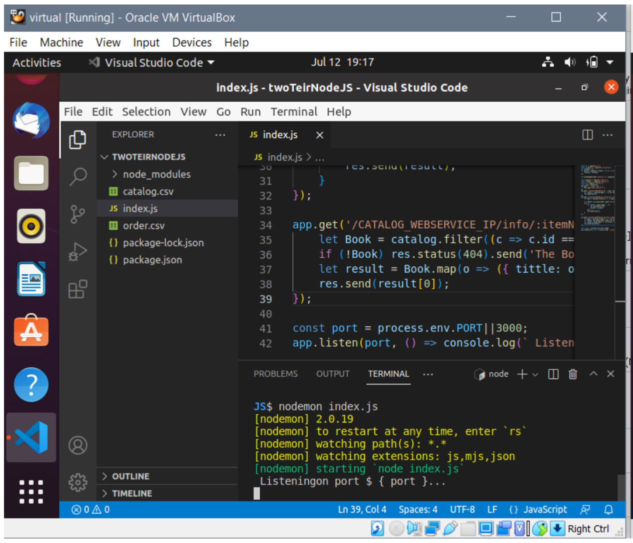
Also, we use Express which is flexible Node.js framework, and its help to create a robust API in quickly and easy way.

Procedure:

Then we build the microservices requested:

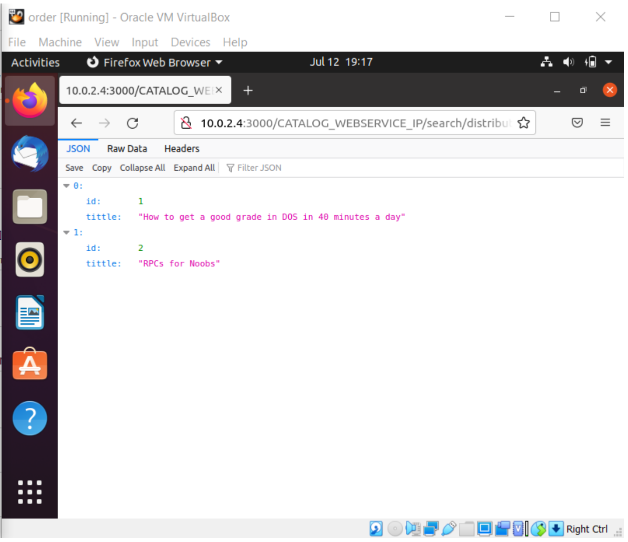
- /CATALOG_WEBSERVICE_IP/search/:itemName
  Using this we can search about specific books that belong to a specific group using the group topic, so it will return all information about the books, here it will return (id, tittle), in case the group not found it will return an empty array and set the status to **not found,** this microservice will done by catalog server.
  The request will send from frontend to catalog server using http request.

Test between virtual machines.



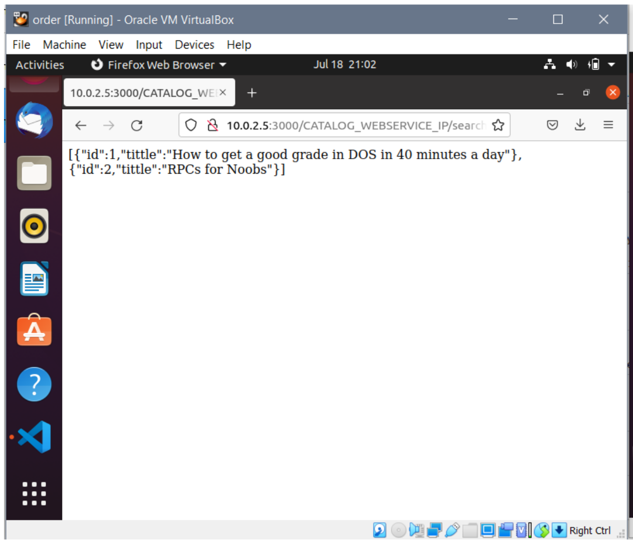
```
app.get('/CATALOG_WEBSERVICE_IP/info/:itemN
    let Book = catalog.filter((c => c.id ==
    if (!Book) res.status(404).send('The Bo
    let result = Book.map(o => ({ tittle: o
    res.send(result[0]);
});

const port = process.env.PORT||3000;
app.listen(port, () => console.log(` Listen
```

```
JS$ nodemon index.js
[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
 Listeningon port $ { port }...
```



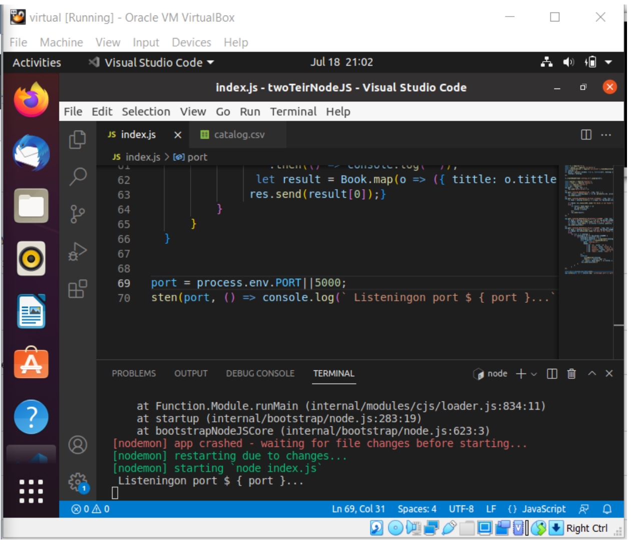
10.0.2.4:3000/CATALOG_WEBSERVICE_IP/search/distribu

```
0:
    id:      1
    tittle:  "How to get a good grade in DOS in 40 minutes a day"
1:
    id:      2
    tittle:  "RPCs for Noobs"
```



10.0.2.5:3000/CATALOG_WEBSERVICE_IP/searc

```
[{"id":1,"tittle":"How to get a good grade in DOS in 40 minutes a day"},
{"id":2,"tittle":"RPCs for Noobs"}]
```



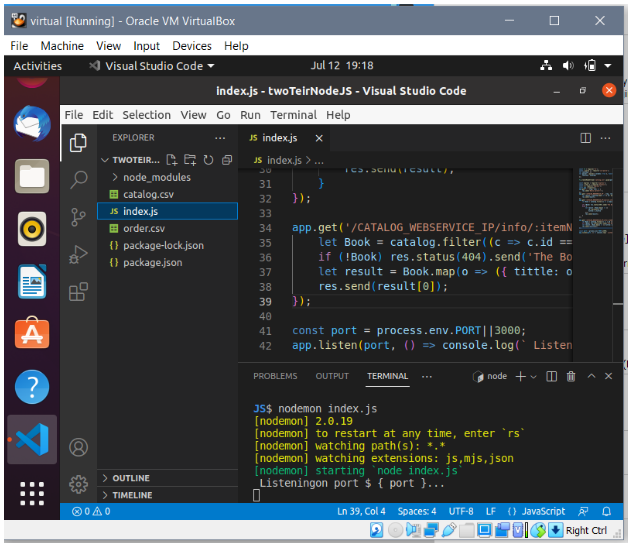
```
             let result = Book.map(o => ({ tittle: o.tittle
            res.send(result[0]);}
        }
    }
}


port = process.env.PORT||5000;
sten(port, () => console.log(` Listeningon port $ { port }...`
```

```
    at Function.Module.runMain (internal/modules/cjs/loader.js:834:11)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:623:3)
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
 Listeningon port $ { port }...
```

- /CATALOG_WEBSERVICE_IP/info/:itemNUM
  Using this microservices we can get all information about specific book by sending book number in URL, also it done by catalog server.

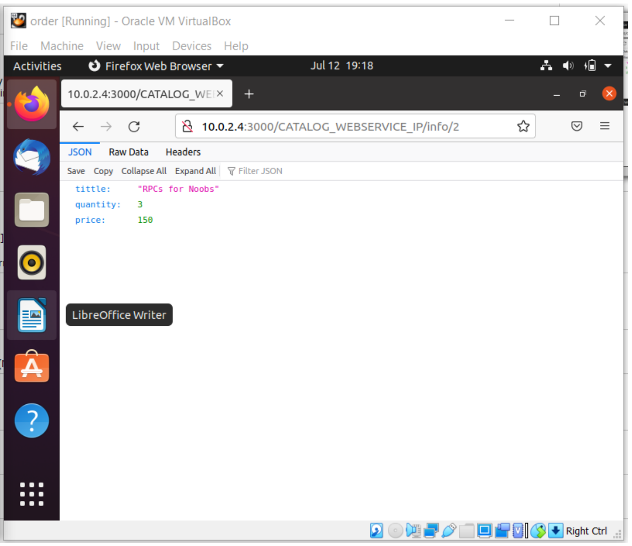
  The request sends from frontend to catalog server then its return the result to frontend.

Test between virtual machines:

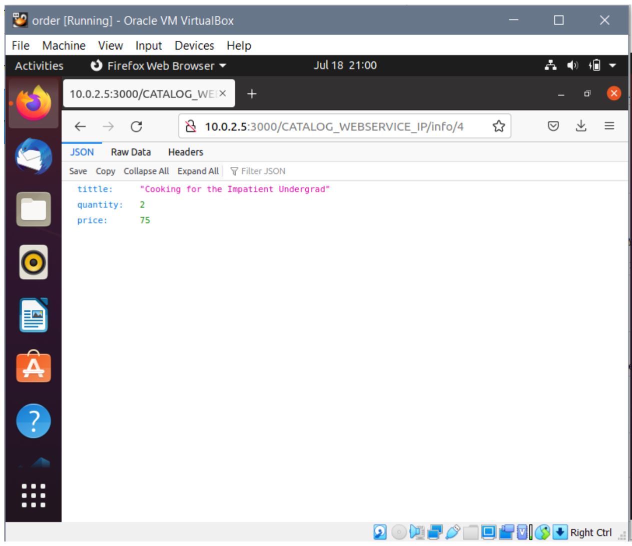

virtual [Running] - Oracle VM VirtualBox
index.js - twoTeirNodeJS - Visual Studio Code
catalog.csv
index.js
order.csv
package-lock.json
package.json

```
        }
});

app.get('/CATALOG_WEBSERVICE_IP/info/:itemN
    let Book = catalog.filter((c => c.id ==
    if (!Book) res.status(404).send('The Bo
    let result = Book.map(o => ({ tittle: o
    res.send(result[0]);
});

const port = process.env.PORT||3000;
app.listen(port, () => console.log(` Listen
```

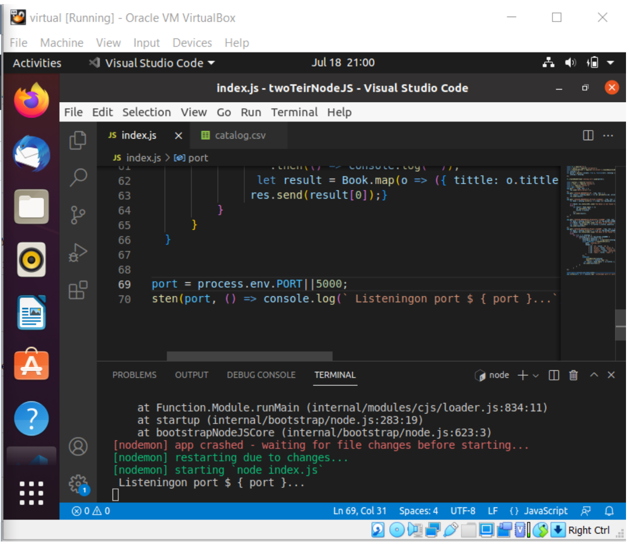
```
JS$ nodemon index.js
[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
 Listeningon port $ { port }...
```


order [Running] - Oracle VM VirtualBox
10.0.2.4:3000/CATALOG_WEBSERVICE_IP/info/2

```
tittle:     "RPCs for Noobs"
quantity:   3
price:      150
```


order [Running] - Oracle VM VirtualBox
10.0.2.5:3000/CATALOG_WEBSERVICE_IP/info/4

```
tittle:     "Cooking for the Impatient Undergrad"
quantity:   2
price:      75
```

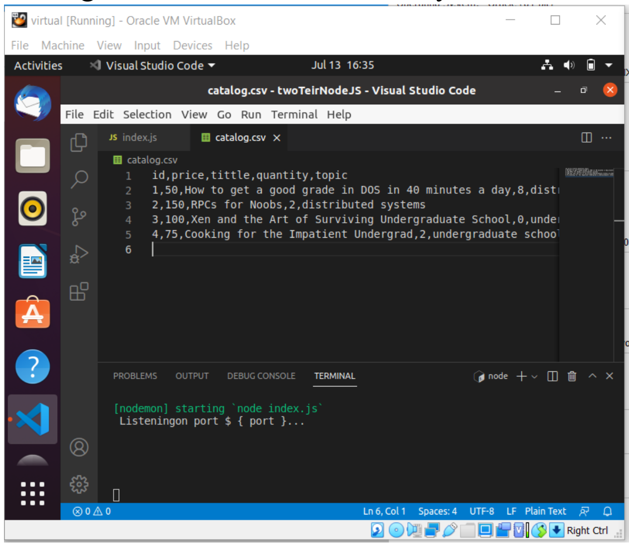

virtual [Running] - Oracle VM VirtualBox
index.js - twoTeirNodeJS - Visual Studio Code
catalog.csv

```
                let result = Book.map(o => ({ tittle: o.tittle
               res.send(result[0]);}
            }
        }
    }


port = process.env.PORT||5000;
sten(port, () => console.log(` Listeningon port $ { port }...`
```

```
    at Function.Module.runMain (internal/modules/cjs/loader.js:834:11)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:623:3)
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
 Listeningon port $ { port }...
```

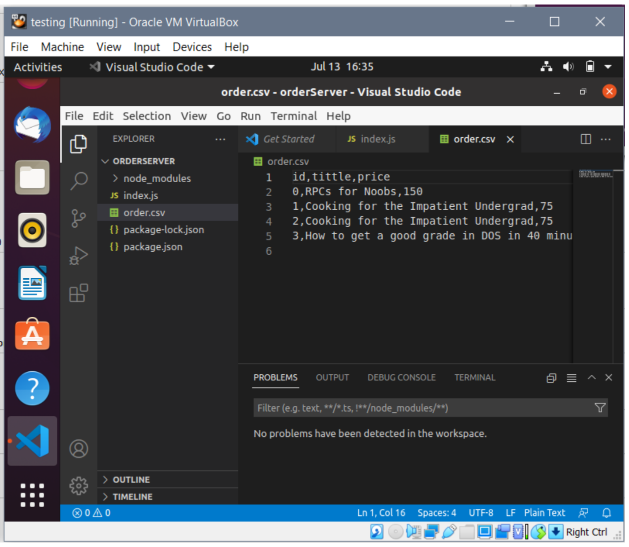
- /CATALOG_WEBSERVICE_IP/pruchase/:itemNUM
  We can use this microservices to specifies an item number for purchase, its will done by order server that will take the item number and send a query form catalog server to check if its available, then this query will return the information of the book if it's available and decrement the quantity available, and return null if it's not available, then the order server will save the purchase item in its data base.
  The request goes from frontend to order server, then the order server will request from catalog server and the result will back from catalog to order to frontend.

catalog.csv & order.csv before buy RPC's for Noobs



```
id,price,tittle,quantity,topic
1,50,How to get a good grade in DOS in 40 minutes a day,8,dist
2,150,RPCs for Noobs,2,distributed systems
3,100,Xen and the Art of Surviving Undergraduate School,0,unde
4,75,Cooking for the Impatient Undergrad,2,undergraduate schoo
```

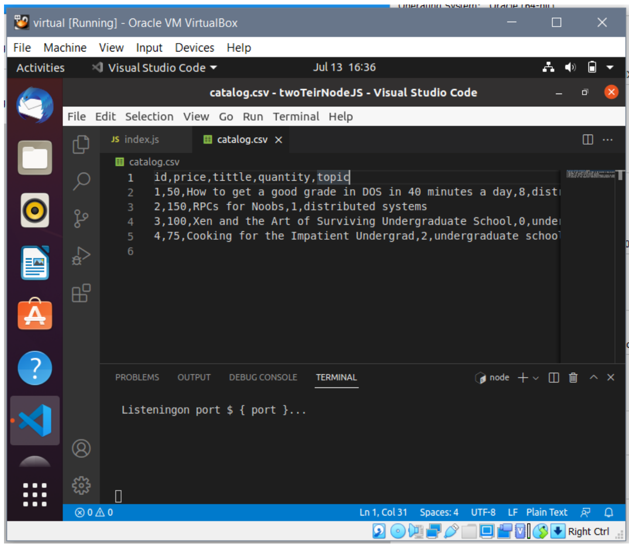
```
[nodemon] starting `node index.js`
 Listeningon port $ { port }...
```



```
id,tittle,price
0,RPCs for Noobs,150
1,Cooking for the Impatient Undergrad,75
2,Cooking for the Impatient Undergrad,75
3,How to get a good grade in DOS in 40 minu
```

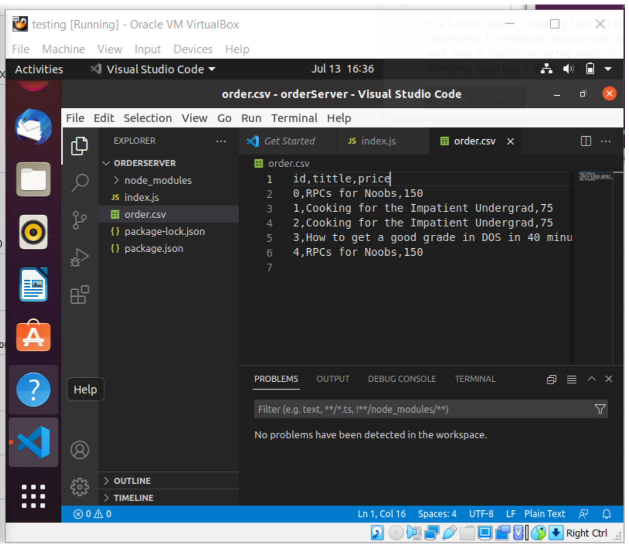
No problems have been detected in the workspace.

catalog.csv & order.csv after buy RPC's for Noobs



```
id,price,tittle,quantity,topic
1,50,How to get a good grade in DOS in 40 minutes a day,8,dist
2,150,RPCs for Noobs,1,distributed systems
3,100,Xen and the Art of Surviving Undergraduate School,0,unde
4,75,Cooking for the Impatient Undergrad,2,undergraduate schoo
```

```
 Listeningon port $ { port }...
```



```
id,tittle,price
0,RPCs for Noobs,150
1,Cooking for the Impatient Undergrad,75
2,Cooking for the Impatient Undergrad,75
3,How to get a good grade in DOS in 40 minu
4,RPCs for Noobs,150
```

No problems have been detected in the workspace.

Improvement:

We can improve our program by send the quantity of how many books we will buy, that's when we send a purchase in URL, we can send another number which consider of how many books we will buy.
It will be something like this:
CATALOG_WEBSERVICE_IP/purchase/2/3
which mean that we will buy 3 books from the book that have the id = 2.
Another improvement to my solution, I can use Joi library for testing and validation process.