



ASSIGNMENT 4

CSCI 5409

Sneha Jayavardhini Doguparthi

B00846995

1. Deficiencies

None

2. Excess Requirements

The web service uses Azure Database for MySQL. Figure 1 shows the overview of the created azure database.[1]

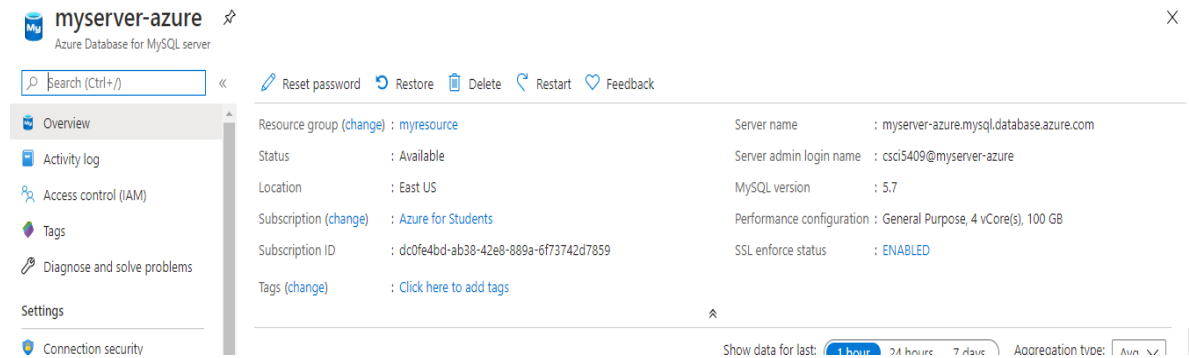


Figure 1 Successful creation of Azure DB for MySQL server

After the creation of the Azure database, the SQL connection is made from the MySQL workbench to create the database and table.

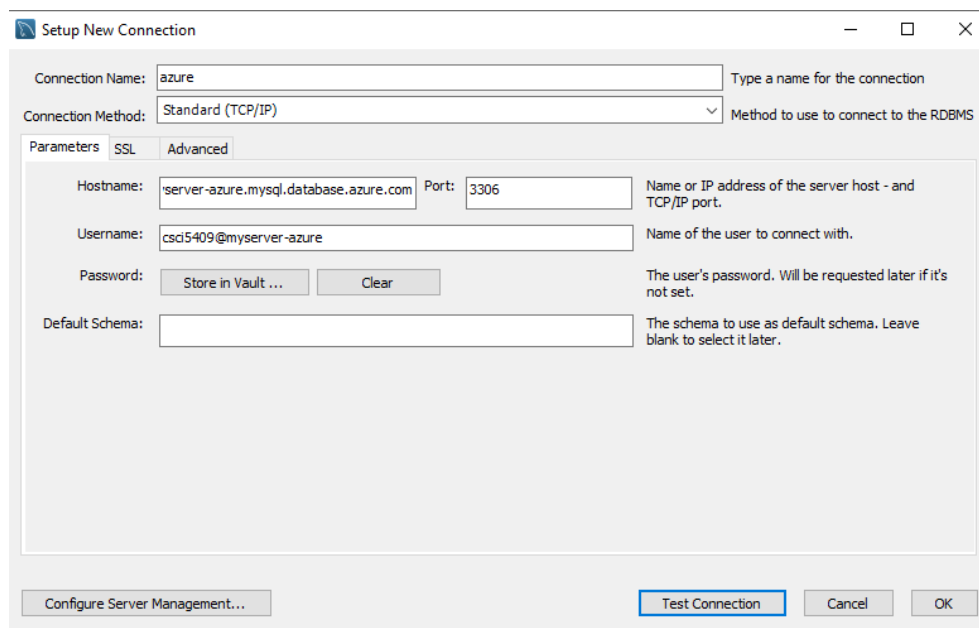
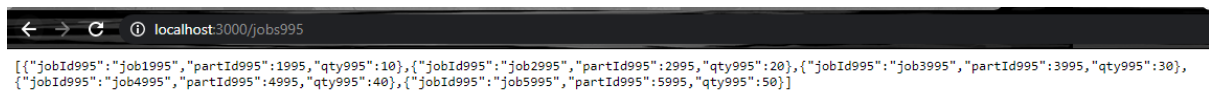


Figure 2 Connection to Azure Database using MySQL workbench

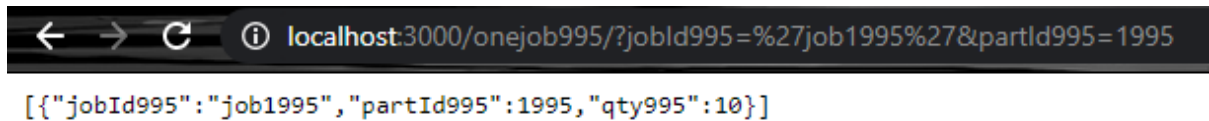
A database 'csci5409' is created, and then the table 'jobs995' is created with job Id and part Id as primary keys. Appendix A1 shows the sql script which is executed to create the table with columns jobId995, partId995, qty995.

The web service is ran locally and tested to check whether the data is retrieved from azure database.



```
localhost:3000/jobs995
[{"jobId995":"job1995","partId995":1995,"qty995":10}, {"jobId995":"job2995","partId995":2995,"qty995":20}, {"jobId995":"job3995","partId995":3995,"qty995":30}, {"jobId995":"job4995","partId995":4995,"qty995":40}, {"jobId995":"job5995","partId995":5995,"qty995":50}]
```

Figure 3 Checking locally to retrieve data form Azure database



```
localhost:3000/onejob995/?jobId995=%27job1995%27&partId995=1995
[{"jobId995":"job1995","partId995":1995,"qty995":10}]
```

Figure 4 Checking locally to retrieve data form Azure database

3. Steps to perform this task

- Initially, I created an Azure Database for MySQL server. The server level firewall rule is cofigured so that the external web service can communicate with the azure database.
- Then the database connection in the code is modified in order to connect to the database. The code which is used to develop the web service is shown in Appendix A2.
- An Azure registry is created from Visual Studio code, Figure 5 shows that 'csci5409registry' is successfully created.

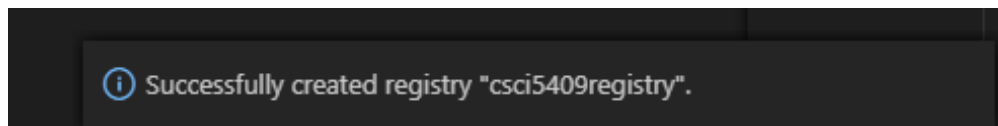


Figure 5 Registry creation successful

- The creation of registry is checked by going to Registries section in Visual Studio code (Figure 6).

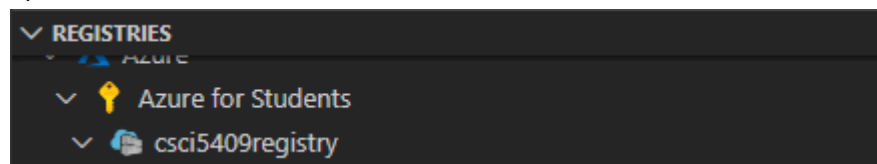


Figure 6 Azure registry

- Using azure cli, we login into the azure portal, and then execute the following command to build the docker image and push into the azure registry i.e. csci5409registry.

az login

az acr build --file Dockerfile --registry csci5409reg --image dockerimage .

Figure 7 and Figure 8 show the docker image is built and pushed into the azure registry.

```

PS D:\MyDream\Dalhousie\Term 3\Cloud Computing\assignment4> az acr build --file Dockerfile --registry csci5409registry --image dockerimage .
Packing source code into tar to upload...
Uploading archived source code from 'C:\Users\SNEHA\AppData\Local\Temp\build_archive_074688026a7c499482fd655c0bb70c9f.tar.gz'...
Sending context (764.426 KiB) to registry: csci5409registry...
Queued a build with ID: cw2
Waiting for an agent...
2020/06/09 20:51:54 Downloading source code...
2020/06/09 20:51:55 Finished downloading source code
2020/06/09 20:51:56 Using acb_vol_be49849a-69f7-4107-bbcd-241839c253a5 as the home volume
2020/06/09 20:51:56 Setting up Docker configuration...
2020/06/09 20:51:57 Successfully set up Docker configuration
2020/06/09 20:51:57 Logging in to registry: csci5409registry.azurecr.io
2020/06/09 20:51:58 Successfully logged into csci5409registry.azurecr.io
2020/06/09 20:51:58 Executing step ID: build. Timeout(sec): 28800, Working directory: '', Network: ''
2020/06/09 20:51:58 Scanning for dependencies...

```

Figure 7 Building the docker image

```

- image:
  registry: csci5409registry.azurecr.io
  repository: dockerimage
  tag: latest
  digest: sha256:8a0f4f0c189859d63302558b81d0de1a5c8f6b57c0fb9faca7f191d56525b4e4
runtime-dependency:
  registry: registry.hub.docker.com
  repository: library/node
  tag: "7"
  digest: sha256:af5c2c6ac8bc3fa372ac031ef60c45a285eeba7bce9ee9ed66dad3a01e29ab8d
  git: {}

Run ID: cw2 was successful after 1m21s
PS D:\MyDream\Dalhousie\Term 3\Cloud Computing\assignment4>

```

Figure 8 Docker image in Azure registry

- f) The name of the image pushed into the registry is 'dockerimage'. Figure 9 shows that the docker image is built and pushed into the azure registry.

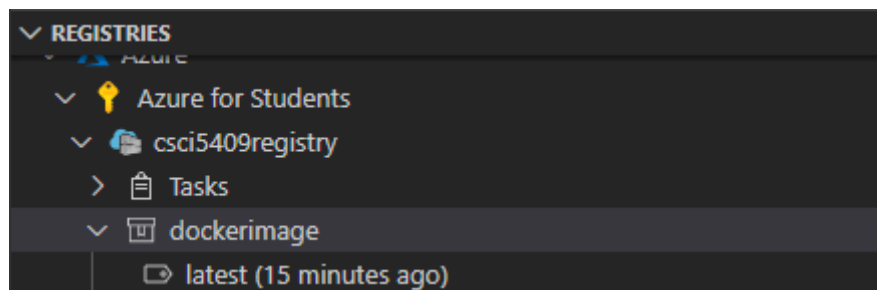


Figure 9 Docker image in Azure Registry

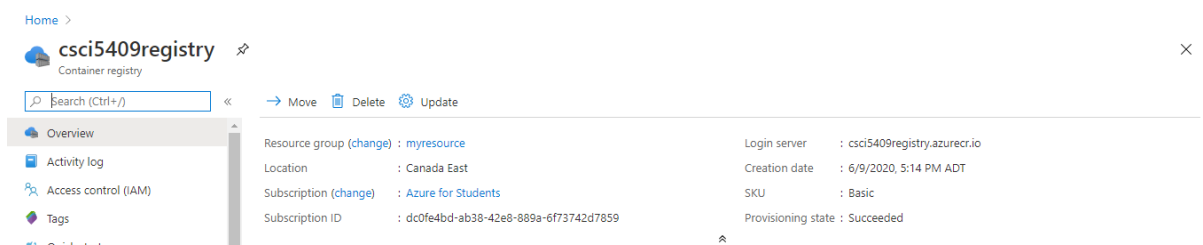


Figure 10 Azure Registry

Figure 10 shows the overview of the created registry on Azure console.

- g) A web app is created using the Microsoft azure console, the image source is selected as 'Azure Container Registry' and the recently created registry is selected in order to create the web app. [2]

Basics **Docker** Monitoring Tags Review + create

Pull container images from Azure Container Registry, Docker Hub or a private Docker repository. App Service will deploy the containerized app with your preferred dependencies to production in seconds.

Options

Image Source

Azure container registry options

Registry *

Image *

Tag *

Startup Command ⓘ

Figure 11 Web app creation

- h) After the successful creation of the web application, the overview of the application is checked. (Figure 12)

Home > App Services >

csci5409-azureapp App Service

Search (Ctrl+/) << Browse Stop Swap Restart Delete Get publish profile Reset publish profile

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Security

Resource group (change) : myresource URL : <https://csci5409-azureapp.azurewebsites.net>

Status : Running App Service Plan : [ASP-myresource-aefc \(F1: Free\)](#)

Location : Australia Central FTP/deployment username : No FTP/deployment user set

Subscription (change) : [Azure for Students](#) FTP hostname : <ftp://waws-prod-cbr20-003.ftp.azurewebsites.windows.net>

Subscription ID : dc0fe4bd-ab38-42e8-889a-6f73742d7859 FTPS hostname : <https://waws-prod-cbr20-003.ftp.azurewebsites.windows.net>

Tags (change) : [Click here to add tags](#)

Figure 12 Overview of webapp

- i) Now, finally the Browse option is clicked to view the output from the database, or the URL can also be used for the same.

4. Directory structure and Docker file

Figure 13 shows the files in the directory used to create the content.

```
PS D:\MyDream\Dalhousie\Term 3\Cloud Computing\assignment4> ls

Directory: D:\MyDream\Dalhousie\Term 3\Cloud Computing\assignment4

Mode                LastWriteTime         Length Name
----                -
d-----          08-06-2020    19:54             node_modules
-a-----          08-06-2020    23:57           3687 app.js
-a-----          29-05-2020    15:35           107 Dockerfile
-a-----          06-05-2020    13:58          16848 package-lock.json
-a-----          06-05-2020    13:58           278 package.json
-a-----          08-05-2020    15:57           205 server.js
```

Figure 13 Directory structure

Figure 14 shows the content of the docker file.

```
PS D:\MyDream\Dalhousie\Term 3\Cloud Computing\assignment4> cat dockerfile
FROM node:7
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
CMD node server.js
EXPOSE 3000
```

Figure 14 Docker File

5. Test web service

a) GET request to the endpoint '/jobs995'

This request gets all the job information which is stored in the table jobs995. The HTTP request GET is sent from Postman to test the output of this end point.

Since we have already inserted few records in the table jobs995 using my sql script, the result of the GET request from Postman is shown in Figure 15.

The screenshot shows a Postman interface with a GET request to the URL `https://csci5409-azureapp.azurewebsites.net/jobs995`. The response status is 200 OK, and the response body is a JSON array of three job records. The response is displayed in the 'Body' tab, showing the raw JSON data.

```
[{"jobId995": "job1995", "partId995": 1995, "qty995": 10}, {"jobId995": "job2995", "partId995": 2995, "qty995": 20}, {"jobId995": "job3995", "partId995": 3995, "qty995": 30}]
```

Figure 15 GET request from Postman to retrieve all the jobs

The screenshot shows a web browser displaying the JSON response from the GET request. The response is a JSON array of three job records, identical to the one shown in Figure 15.

```
[{"jobId995": "job1995", "partId995": 1995, "qty995": 10}, {"jobId995": "job2995", "partId995": 2995, "qty995": 20}, {"jobId995": "job3995", "partId995": 3995, "qty995": 30}]
```

Figure 16 Browser which shows the information retrieval

b) GET request to the endpoint '/onejob995'

In this request the 'jobId' and 'partId' are sent as parameters (which are the primary keys in the table jobs995) from the URL to retrieve the information of one record.

In this example job1995, 1995 are sent as query parameters from the URL.

Figure 17 shows the GET request which is sent from Postman with parameters 'job1995' and '1995' to retrieve the record.

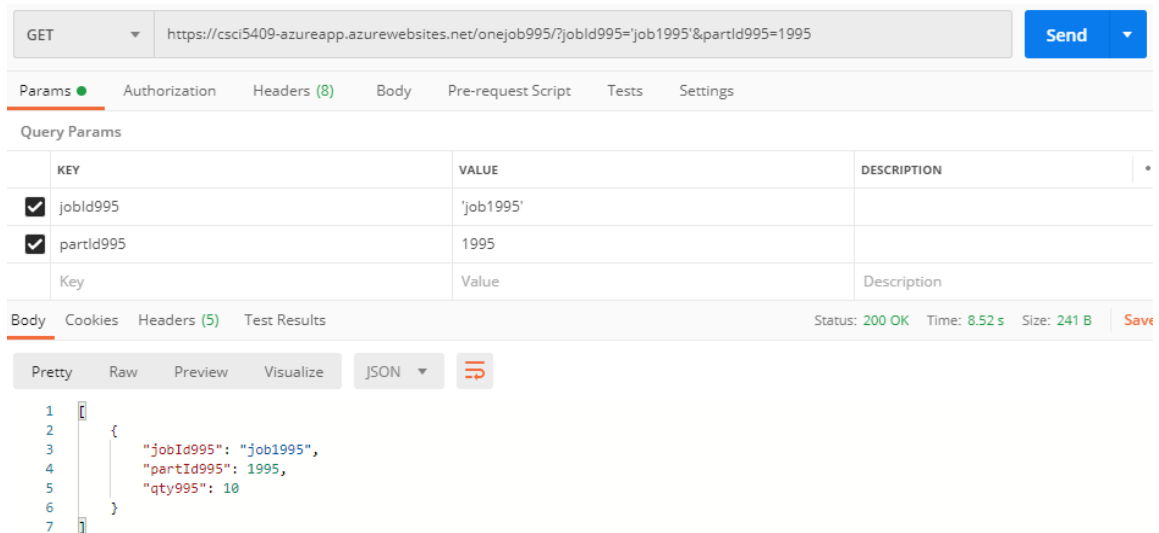


Figure 17 GET request from Postman to retrieve the job information for one record

c) POST request to the endpoint '/jobs995'

In this request (Figure18), the record is inserted into the table if it does not exist. The body of the POST request contains

```

{
  "jobId995": "job6995",
  "partId995": 6995,
  "qty995": 600
}

```

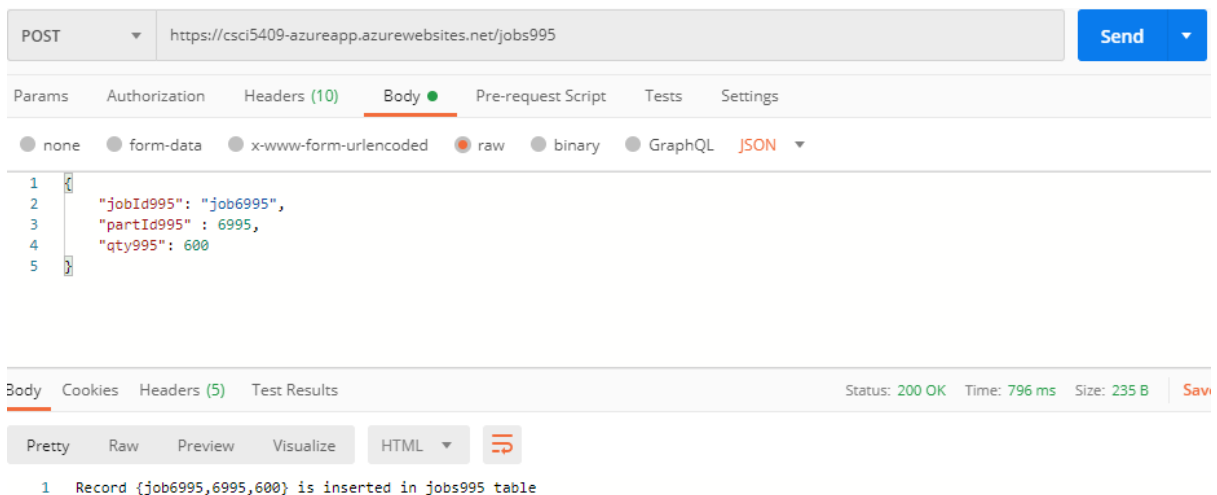


Figure 18 POST request to insert the record in the database

After the record is inserted into the table, a query is executed in My SQL Workbench in order to verify whether the record is inserted. Figure 19 shows the newly inserted record.

13 • `select * from jobs995;`

	jobId995	partId995	qty995
▶	job1995	1995	10
	job2995	2995	20
	job3995	3995	30
	job4995	4995	40
	job5995	5995	50
	job6995	6995	600
*	NULL	NULL	NULL

Figure 19 Retrieval of data from table jobs995 after the record is inserted

d) PUT request to the endpoint '/jobs995'

In this request, the record is updated for the given jobId and partId. The body of PUT method consists of

```
{
  "jobId995": "job1995",
  "partId995": 1995,
  "qty995": 1
}
```

Figure 20 shows that PUT request is sent from Postman to update the record which has job6995 and 6995 as primary keys. The request is to update the column qty to 6.

Response is a successful update with status OK at the bottom.

The screenshot shows a Postman interface for a PUT request to the endpoint `https://csci5409-azureapp.azurewebsites.net/jobs995`. The request body is a JSON object: `{"jobId995": "job6995", "partId995": 6995, "qty995": 6}`. The response status is 200 OK, with a time of 707 ms and a size of 239 B. The response body contains a message: "Record with JobId995 job6995 and partId995 6995 is updated".

Figure 20 PUT request to update the record in the database for given jobId and partId

The screenshot shows a web browser displaying the data from the jobs995 table. The data is presented as a JSON array of objects, each representing a record in the table. The records are: `[{"jobId995": "job1995", "partId995": 1995, "qty995": 10}, {"jobId995": "job2995", "partId995": 2995, "qty995": 20}, {"jobId995": "job3995", "partId995": 3995, "qty995": 30}, {"jobId995": "job4995", "partId995": 4995, "qty995": 40}, {"jobId995": "job5995", "partId995": 5995, "qty995": 50}, {"jobId995": "job6995", "partId995": 6995, "qty995": 6}]`. The record for job6995 and partId995 6995 has been updated with a qty of 6.

Figure 21 Retrieval of data from table jobs995 after the record is updated

References

- [1] Ajlam, "Connect using Node.js - Azure Database for MySQL," Connect using Node.js - Azure Database for MySQL | Microsoft Docs. [Online]. Available: <https://docs.microsoft.com/en-us/azure/mysql/connect-nodejs>. [Accessed: 11-Jun-2020].
- [2] Cryophobia, "Exercise - Create and deploy a web app from a Docker image - Learn," Learn | Microsoft Docs. [Online]. Available: <https://docs.microsoft.com/en-us/learn/modules/deploy-run-container-app-service/5-exercise-deploy-web-app?pivots=csharp>. [Accessed: 11-Jun-2020].
- [3] Btholt, "Deploy Docker containers to Azure App Service from Visual Studio Code," Deploy Docker containers to Azure App Service from Visual Studio Code | Microsoft Docs. [Online]. Available: <https://docs.microsoft.com/en-us/azure/javascript/tutorial-vscode-docker-node-01?tabs=bash>. [Accessed: 11-Jun-2020].
- [4] Dlepow, "Tutorial - Quick container image build - Azure Container Registry," Tutorial - Quick container image build - Azure Container Registry | Microsoft Docs. [Online]. Available: <https://docs.microsoft.com/en-us/azure/container-registry/container-registry-tutorial-quick-task>. [Accessed: 11-Jun-2020].

Tools/Software Used

1. Node.js and npm: <https://nodejs.org/en/>
2. Express : <https://www.npmjs.com/package/express>
3. Nodemon: <https://www.npmjs.com/package/nodemon>
4. Postman: <https://www.postman.com/>
5. Visual Studio code editor: <https://code.visualstudio.com/>
6. Azure cli: <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>

Appendix

A1. Sql script which creates a table with name jobs995 in the database 'csci5409'

```
CREATE TABLE `csci5409`.`jobs995` (  
  `jobId995` VARCHAR(10) NOT NULL,  
  `partId995` INT NOT NULL,  
  `qty995` INT NULL,  
  PRIMARY KEY (`jobId995`, `partId995`));
```

```
insert into jobs995 values('job1995',1995,10);  
insert into jobs995 values('job2995',2995,20);  
insert into jobs995 values('job3995',3995,30);  
insert into jobs995 values('job4995',4995,40);  
insert into jobs995 values('job5995',5995,50);
```

A2. Javascript code used to build the web service

app.js

```
const express = require('express');  
const mysql = require('mysql');
```

```

const app = express();
app.use(express.json());
const bodyParser = require('body-parser');
//setting body-parser
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));

//Creating the connection to the database
const db =mysql.createConnection({
  host: 'myserver-azure.mysql.database.azure.com',
  user: 'csci5409@myserver-azure',
  port: '3306',
  password : 'Cloud@5409',
  database: 'csci5409',
  ssl:true
});
db.connect((err)=>{
  if(err){
    throw err;
  }
  console.log('Connection successful');
});

//Get information for all the jobs
app.get('/jobs995',(req,res)=>{
  let sql = 'select * from jobs995';
  let query = db.query(sql,(err,result)=>{
    if(err){
      throw err;
    }
    if(result.length==0){
      ('Jobs995table does not have any records');
      res.send('Jobs995 table does not have any records');
    }else{
      console.log(result);
      res.send(result);
    }
  });
});

//Get job information for particular jobId995 and partId995
app.get('/onejob995',(req,res)=>{
  let sql = "select * from jobs995 where jobId995 = " +req.query.jobId995+" and partId995 =" +
  req.query.partId995;
  let query=db.query(sql,(err,result)=>{

```

```

    if(err){
        throw err;
    }
    console.log(result);
    if(result.length==0) {
        console.log('Job with jobId ' + req.query.jobId995 + 'and partId ' + req.query.partId995 + 'does
not exist');
        res.status(404).send('Job with jobId '+req.query.jobId995+ ' and partId \'t'
+req.query.partId995 +' was not found');
    }else
        res.send(result);
    });
});
const jsonParser = bodyParser.json();

//Insert the record with job information
app.post('/jobs995',jsonParser,(req,res)=>{
    let select = "select * from jobs995 where jobId995 = '" + req.body.jobId995+ "' and partId995 = " +
req.body.partId995;
    let sql= 'insert into jobs995 SET ?';
    let insertData =
{jobId995:req.body.jobId995,partId995:req.body.partId995,qty995:req.body.qty995};
    let selQuery = db.query(select,(selerror,result)=>{
        if(result.length==0){
            let insertQuery = db.query(sql,insertData,(err,insertResult)=>{
                if(err){
                    throw err;
                }
                res.send('Record {' + insertData.jobId995 + "," + insertData.partId995 + "," + " +
insertData.qty995 + ' } is inserted in jobs995 table');
            });
        }else{
            res.status(404).send('Jobs995 table with jobId ' +insertData.jobId995 +'and partId '
+insertData.partId995 +' already exists');
        }
    });
});

//Update the job information for a given jobId and partId
app.put('/jobs995',jsonParser,(req,res)=>{
    let select = "select * from jobs995 where jobId995 = '" + req.body.jobId995+ "' and partId995 = " +
req.body.partId995;
    let sql = "Update jobs995 SET ? where jobId995 = '" + req.body.jobId995+ "'";
    let updateData = {qty995:req.body.qty995};
    let selQuery = db.query(select,(err,result)=>{

```

```
if(result.length!=0){
  let updateQuery = db.query(sql,updateData,(err,result)=>{
    if(err){
      throw err;
    }
    res.send('Record with JobId995 ' + req.body.jobId995 + ' and partId995 '+
req.body.partId995 +' is updated');
  });
}else{
  res.status(404).send('JobId '+req.body.jobId995+' does not exist');
}
});
});
```

```
module.exports = app;
```

server.js

```
const http=require('http');
const app=require('./app');
const port=process.env.PORT || 3000;
const server =http.createServer(app);
server.listen(port,() => console.log(`Listening to port ${port}...`));
```