

Kalan Assignment 2

by Kalan Hughes

Submission date: 12-Jan-2022 03:52PM (UTC+0000)

Submission ID: 168638492

File name: 17861_Kalan_Jay_Hughes_Kalan_Assignment_2_382342_731442063.docx (399.31K)

Word count: 5983

Character count: 36344

Assignment 2

Kalan Hughes

1902299

Bachelor of Science: Computing (Web Development)

Cloud-Based Web Services

14 January 2022

Table of Contents

Cloud Computing Report	3
Virtualisation and Containerisation	6
Development Evaluation.....	8
eCommerce Application.....	14
YouTube Video Link.....	14
Database Login Details.....	14
References	15
Appendix.....	17
Service.js:	17
pgdbgame.js:	20
server.js:.....	27
index.html:	28
Addgameform.html:.....	30
Addgameform.js:.....	33
Readgamespage.html:	34
Readgamespage.js:	36
Thanks.html:	37
Checkout.html:.....	38

Cloud Computing Report

This report is going to contain information regarding cloud computing. The report will explain what cloud computing is and it will evaluate the benefits and drawbacks of deploying applications to the cloud. The cost implications of deploying an application to the cloud will also be talked about. Some examples and scenarios will be used when talking about deploying applications to the cloud. The reliability and scalability of cloud computing will be discussed, and the importance of these principles will be talked about.

Cloud computing is a service that allows you to have access to applications, resources, servers, storage, and much more through the internet [1]. This service is provided through cloud service providers who use remote data centres to host all of these resources. The cloud service providers allow you to use these resources in exchange for money. They could either charge you monthly for a subscription fee or they could charge you depending on the amount of resources you use. There are different types of cloud computing [2]. The first type of cloud computing is public cloud. Public cloud is when third-party cloud service providers provide the resources through the internet. A public cloud can be accessed through a web browser. All of the resources in a public cloud are managed by the cloud service provider [2]. The second type of cloud computing is a private cloud. A private cloud is when a single organisation uses the cloud computing resources. Unlike a public cloud, a private cloud can be physically located in the organisations on-site data centre. Organisations are able to pay cloud service providers to host their private cloud. A private cloud means that all of the resources are only available through a private network [2]. The final type of cloud computing is a hybrid cloud. A hybrid cloud allows a public and private cloud to communicate by using technology that allows the two clouds to share data and applications. A hybrid cloud is beneficial as it allows for more flexibility because you use both public, and private clouds [2]. These are the three different types of clouds available through cloud computing. There are four different types of cloud services available. The first cloud service is Infrastructure as a service (IaaS). This is the most basic cloud service. This allows you to rent cloud resources from a cloud service provider [2]. The second cloud service is Platform as a service (PaaS). This cloud service provides an on-demand environment which allows you to manage software applications. It allows you to easily develop, test, and deliver applications without having to set up any of the resources needed for developing an application [2]. The third cloud service is Serverless computing. This allows you to develop applications without having to worry about managing the servers. The cloud service provider manages the server for you. The serverless architecture is event-driven and highly scalable. This means that it only uses resources when a specific function or trigger occurs [2]. The final cloud service is Software as a service (SaaS). This allows you to access software applications on demand over the internet. The cloud service providers manage the software applications which means that you do not have to worry about the architecture or maintenance of the software. The applications are accessed over the internet, typically with a web browser [2]. These are all of the cloud services available.

There are some benefits of deploying an application to the cloud. One benefit of deploying an application to the cloud is that it can save money [3]. This is because it means that you do not have to buy the hardware, software, and servers needed to develop and deploy an application [2]. You only need to pay for the cloud service which can help to save a lot of money over time. The second benefit of deploying an application to the cloud is security [4]. This is because many cloud service providers have policies and technologies that can help to strengthen the security of the application [2]. The third benefit of deploying an application to the cloud is accessibility [3]. This is because you can access the application and the services over the internet. This means that you do not have to

have a fixed device which you use to access the application and services. This can help to improve flexibility as you are able to work on the application regardless of where you are [3]. The final benefit of deploying an application to the cloud is disaster recovery [4]. This is because the application will be stored on the cloud which helps to keep it safe from any disaster like fires. If the application was stored on-site, it is at risk of being destroyed in a disaster. However, if it is stored on the cloud, it is safe from any disaster which can prevent you from losing any data relating to the application. These are some of the benefits of deploying an application to the cloud.

There are also some drawbacks of deploying an application to the cloud. One drawback of deploying an application to the cloud is downtime [5]. This is a drawback because if your cloud service provider goes down for maintenance, you will be unable to access the application. This could prevent you from being able to manage the application. This could be bad if you needed to access the application at that moment when the cloud service provider went down for maintenance. Another drawback of deploying an application to the cloud is the resources. This is because if any of the resources have bugs or issues, you have to wait for the cloud service provider to update the resources to fix the issues. This is because you are unable to update the resources yourself. This could result in a lot of time being wasted while you wait for the cloud service provider to fix the issues. The final drawback of deploying an application to the cloud is inflexibility [5]. This is because if you wanted to change your cloud service provider, you would have to transfer all of the data and the application to the new cloud service provider. This could be difficult because the cloud service provider may not have an easy way of transferring the data and the application to the new cloud service provider. This means that you would have to spend a lot of time trying to transfer everything which could result in a lot of time being wasted. These are some of the drawbacks of deploying an application to the cloud.

There are also cost implications when deploying applications to the cloud. Depending on the size of the application being deployed to the cloud, the price for the necessary cloud services may increase. For example, if your application only required 1 core, 0.75 GB of RAM, and 20 GB of temporary storage, it would cost you £12.0481 per month [6]. However, if your application required 8 cores, 14 GB of RAM, and 2,040 GB of temporary storage, it would cost you £385.5364 per month [6]. This shows that you may have to pay more per month depending on the requirements of the application. This is an important implication to be aware of because depending on the size of the application, it may end up being cheaper to deploy your application from an on-site data centre. However, depending on the size of the application, it may end up being cheaper to deploy your application on the cloud. An example of when it would be cheaper to deploy your application on the cloud, is when your application is small and does not require a lot of resources. This is because it would only cost you £12.0481 per month [6]. However, if you wanted to deploy your small application from an on-site data centre, you would have to spend money for the servers, and the application architecture which would end up costing a lot more money in the long run. An example of when it would be cheaper to deploy your application from an on-site data centre is when your application requires a lot of resources. This is because once you have spent money on the servers and the applications architecture, you would only have to spend money on maintenance and upkeep. However, if you wanted to deploy your large application on the cloud, it could cost up to £385.5364 per month [6]. This could end up costing more in the long run which is why it would be cheaper to deploy it from an on-site data centre. This is why it is important to consider the cost implications when deciding how you would like to deploy your application.

Two of the most important principles in cloud computing are reliability and scalability [7]. Reliability is important in cloud computing because you need to make sure that the cloud service provider that

you are using is reliable. This is because you need to make sure that their resources are always available [7]. If the resources are not available, you are not going to be able to get work done which could result in time being wasted. The cloud service provider also needs to be reliable in terms of security [7]. This is to ensure that any data that you transfer over the cloud is secure. If the cloud service provider is not reliable in terms of security, the data that you transfer over the cloud could get stolen which could have a lot of consequences. Finally, you need to ensure that the cloud service provider is reliable in terms of downtime [7]. This is because you need to know when the cloud service provider is going to have downtime so you can schedule your work around the downtime. If the cloud service provider is not reliable with their downtime, they could have downtime at random times. This could result in the cloud service provider having downtime when you are working which could result in you not being able to do any work.

Scalability is also important in cloud computing because you need to make sure that the cloud service provider you are using allows for scalability. This is because you need to make sure that their services are convenient [7]. This means that the cloud service provider needs to allow you to increase or decrease any resources when necessary. This can help to make sure that you always have the necessary number of resources. The cloud service provider also needs to be scalable in terms of cost [7]. This is to ensure that the amount of money you pay changes depending on the amount of cloud resources you are using. This can help to prevent you from paying too much money for resources you are not using. Finally, you need to ensure that the cloud service provider is scalable in terms of flexibility [7]. This is to ensure that you can respond quickly to changing demands. This can help to prevent time from being wasted. These are the reasons why reliability and scalability are important principles in cloud computing.

You need to consider the reliability and scalability when deciding if you want to deploy an application through the cloud or through an on-site data centre. If you decided to deploy the application to the cloud, it would not be as reliable as an on-site data centre as there may be unexpected downtime. However, it would be more scalable than an on-site data centre as the cloud has a variety of different resources that you can use on demand. If you wanted to deploy the application through an on-site data centre it would be more reliable than the cloud. This is because you would be aware of when maintenance needs to be carried out so you could plan your work around it. However, it would be less scalable than the cloud as you would not have the same number of resources that the cloud provides on demand. This concludes the report on cloud computing.

Virtualisation and Containerisation

This section is going to talk about virtualisation. It is also going to explain the fundamental differences between virtual machines and containers. Finally, there will be a section which talks about the driving forces behind the adoption of Docker.

Virtualisation allows you to divide the elements of a single computer into multiple virtual computers by using software to create an abstraction layer over the computer's hardware [8]. Each of the virtual computers, also known as virtual machines, run their own operating systems. Virtualisation allows for more efficiency when utilising a physical computer's hardware. Virtualisation is the technology which drives the cloud computing economics [8]. This is because it allows cloud service providers to provide their users with existing physical computer hardware. These resources can also be scaled depending on what the user needs for their workload [8]. One benefit of virtualisation is faster provisioning [8]. This is because provisioning virtual machines to run applications is a lot faster than buying and installing the applications. Another benefit of virtualisation is minimal downtime [8]. This is because you are able to run multiple virtual machines alongside each other. This means that if one of them were to fail or had any problems, you could quickly switch over to another virtual machine. A hypervisor coordinates the virtual machines on the software layer. Hypervisors are needed to ensure that the virtual machines and the underlying physical hardware each have the necessary physical resources needed to execute [8]. There are two types of hypervisors. The first type is type 1 hypervisors. This type commonly appears in virtual server scenarios. The second type is type 2 hypervisors. This type is commonly used on endpoint devices to run alternative operating systems [8]. There are many different types of virtualisations. Some of these are desktop virtualisation, network virtualisation, and storage virtualisation [8].

There are some fundamental differences between virtual machines and containers. These fundamental differences will be discussed in this section. Virtual machines and containers are both used to create virtual environments for developing applications and software [9]. There are some fundamental differences in the way that they operate. A virtual machine is an operating system which shares the physical resources of a server. A virtual machine consists of several layers. The hypervisor is the layer that enables the virtualisation [9]. A virtual machine contains everything that is needed to run an application. This means that it is self-contained as it has its own infrastructure. Each virtual machine is isolated from the host operating system. Virtual machines can also have different operating systems from the host operating system [9]. A container is not dependent on the operating system. It is an environment that can run applications [9]. The user is able to create many workloads on a single operating system instance because the container isolates the application from the host by virtualising it [9]. The kernel of the host operating system is needed to run the different functions of an application which is separated into containers [9]. Each one of the containers runs isolated tasks. The host machine cannot be harmed, and it cannot come into conflict with other applications which are running in separate containers [9]. You are able to create a template of an environment when working inside a container. The container can run a snapshot of the system at a particular time which can help to provide consistency in the application [9]. The container shares the host's kernel to run all of the individual applications within the container. Each container requires bins, libraries, and some runtime components [9]. These are some of the fundamental differences between virtual machines and containers.

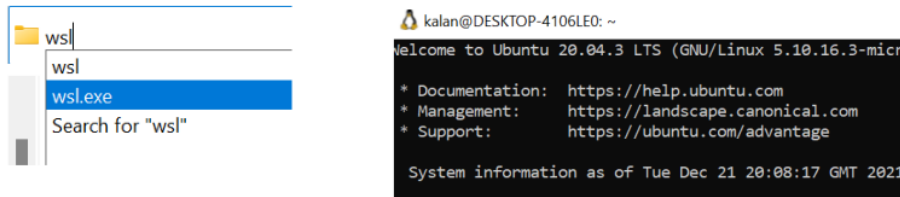
This section is going to talk about the driving forces behind the adoption of Docker. One reason why Docker was adopted is because virtual machine hypervisors need a lot of system requirements to

function [10]. Containers use shared operating systems which make them a lot more efficient than hypervisors when using system resources. Using containers allows you to have up to six times the number of server application instances as you could have on a virtual machine using the same hardware [10]. This is one reason why Docker was adopted, as it uses containers. Containers also allow developers to integrate their code into a shared repository. This allows them to deploy their code quickly and efficiently. This is another reason why Docker was adopted as it allows developers to easily run an application as a self-sufficient container which can virtually run anywhere [10]. Another reason why Docker was adopted, is because Docker containers are easy to deploy in a cloud [10]. An additional reason why Docker was adopted is because it made containers easier and safer to deploy and use than other approaches [10]. Docker also partnered with other container powers such as Google and Red Hat which allowed for containers to be standardised [10]. These are some of the driving forces behind the adoption of Docker. This concludes the section on virtualisation and containerisation.

Development Evaluation

This section is going to evaluate the development process of producing node.js applications that can be run in Docker. This will be done by providing a step-by-step guide of the development process and each step will be evaluated to show if the step is easy or not. There will also be illustrations provided in this section.

The first step of producing node.js applications that can be run in Docker, is to open WSL. This is an easy step as you can open WSL from the file explorer.



The next step is to install node.js. First, you need to enable the NodeSource repository and then you need to install node.js and npm. The nodejs package will install both of them. This is easy to do as it is only two lines of code. Below are the two lines of code.

```
curl -fsSL https://deb.nodesource.com/setup\_16.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

You can then check the version of node by entering 'node --version'. You can also check the version of Node Package Manager by entering 'npm --version'.

```
kalan@DESKTOP-4106LE0:~$ node --version  
v16.13.0  
kalan@DESKTOP-4106LE0:~$ npm --version  
8.1.0
```

The next step is to install the print module. This will allow you to display things on the screen. This is a simple step as it is only one line of code. That line of code is displayed below.

```
npm install print
```

You can check for any outdated dependencies to ensure that they are all up to date. You should also update all of the modules to ensure that you are using the latest version. This is easy to do as it is only two lines of code.

```
npm outdated  
npm update
```

The next step is to make a directory called expressjs. This is a simple step as it is only one line of code to make a new directory. That line is below.

```
kalan@DESKTOP-4106LE0:~$ mkdir expressjs_
```

The next step is to put three files into this directory. These files are called '.dockerignore, docker-compose.yml, and Dockerfile'. This is a simple step as these files can be found on the internet.



The next step is to create a new directory inside of expressjs. This new directory is called nodeweb service. This is an easy step as it is only one line of code.

```
kalan@DESKTOP-4106LE0:~$ cd expressjs/  
kalan@DESKTOP-4106LE0:~/expressjs$ mkdir nodeweb service
```

You then need to create a package.json file inside of the nodeweb service directory. This is an easy step as it is only one line of code. It is also easy because instead of answering all of the questions in the file, you can just type '--yes' to use the default values.

```
kalan@DESKTOP-4106LE0:~/expressjs/nodeweb service$ npm init
```

The express.js framework is going to be used so it needs to be installed. This is an easy step as it is one line of code which can be seen below.

```
kalan@DESKTOP-4106LE0:~/expressjs/nodeweb service$ npm install express express-validator cors pg pg-format
```

The next step is to add some commands to the package.json file to run the service. This is an easy step because only one line is being replaced.

```
{  
  "name": "nodeweb service",  
  "version": "1.0.0",  
  "description": "node.js web service example",  
  "main": "service.js",  
  "scripts": {  
    "exec": "node service.js",  
    "start-mount": "./node_modules/.bin/nodemon -L service.js",  
    "start": "./node_modules/.bin/nodemon service.js"  
  },  
  "author": "Kalan Hughes",  
  "license": "ISC",  
  "dependencies": {  
    "cors": "^2.8.5",  
    "express": "^4.17.1",  
    "express-validator": "^6.13.0",  
    "pg": "^8.7.1",  
    "pg-format": "^1.0.4"  
  },  
  "devDependencies": {  
    "nodemon": "^2.0.14"  
  }  
}
```

The next step is to create a file called service.js inside of the nodeweb service directory. Inside of the file, four lines of code will be written. This is an easy step as the four lines of code are short and easy to write.

```
const express = require('express');
const service = express();
const port = 1339;
service.listen(port, ()=>console.log('Example web service listening at http://localhost:${port}'));
```

Once the file has been created, you need to compile and run the service. This is an easy step as it is only one line of code. It will then show you where the webservice is listening at.

```
kalan@DESKTOP-4106LE0:~/expressjs/nodeweb service$ npm run start
```

```
Example web service listening at http://localhost:1339
```

The next step is to add nodemon to the package.json file. Nodemon is a module that allows you to monitor the code for changes to restart the service. This is an easy step as you only have to type three lines of code in the scripts section of the file.

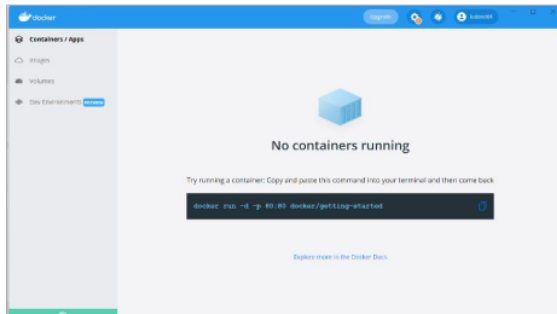
```
"scripts": {
  "exec": "node service.js",
  "start-mount": "./node_modules/.bin/nodemon -L service.js",
  "start": "node_modules/.bin/nodemon service.js"
}
```

You now need to enter some basic routes to the service.js file. This is an easy step as each route only requires you to write three lines of code and each line is short.

```
service.get('/', (req, res) => {
  res.send('Welcome!');
});

service.get('/hi', (req, res) => {
  res.send('Hello!');
});
```

Once you have added the basic routes, you then need to install Docker. This is an easy step as there are many tutorials on the internet which show you how to correctly install Docker. Once Docker is installed, you can type 'docker --version' to make sure that you have the latest version of Docker.



```
kalan@DESKTOP-4106LE0:~/expressjs/nodeweb-service$ docker --version
Docker version 20.10.10, build b485636
```

Once you have installed Docker, you need to run the webservice. To do this, you need to go into the directory containing the docker-compose file. One you are in the directory, you need to enter 'docker-compose up'. This will then start the webservice and it will tell you what port the service is running on. This is an easy step to do as it only requires you to write one line of code.

```
kalan@DESKTOP-4106LE0:~/expressjs$ docker-compose up
[+] Running 9/9
  Network expressjs_dbnetwork      Created
  Container expressjs-mysql-1      Created
  Container expressjs-webserver-1  Created
  Container expressjs-phpmyadmin-1 Created
  Container expressjs-postgres-1   Created
  Container expressjs-mongodb-1    Created
  Container expressjs-adminer-1    Created
  Container expressjs-web-service-1 Created
  Container expressjs-mongo-express-1 Created
```

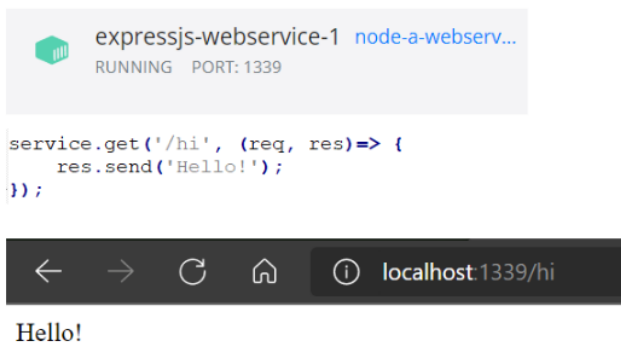
```
expressjs-web-service-1 | [nodemon] 2.0.14
expressjs-web-service-1 | [nodemon] to restart at any time, enter `rs`
expressjs-web-service-1 | [nodemon] watching path(s): *.*
expressjs-web-service-1 | [nodemon] watching extensions: js,mjs,json
expressjs-web-service-1 | [nodemon] starting `node service.js`
```

```
expressjs-web-service-1 | Example web service listening at http://localhost:1339
```

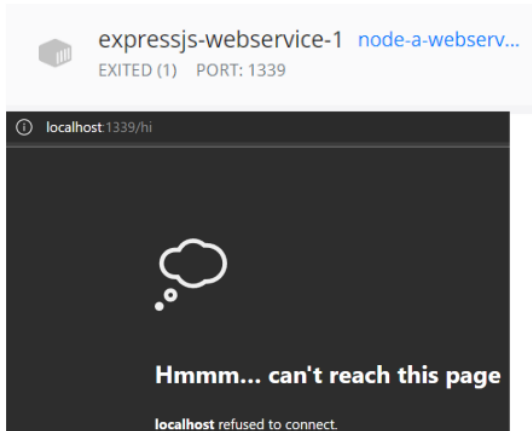
Once the webservice is running, you can go to the browser and type 'http://localhost:1339'. It will then display the route that you created in the service.js file.



You can also type 'http://localhost:1339/hi' into the browser and it will display the other route that you created in the service.js file.



If the webservice is not running, it will not display the service.js file in the browser.



In conclusion, the development process for producing node.js applications that can be run in Docker is relatively simple. However, it may be difficult for new users who are not familiar with node.js or Docker. This is because the syntax may be confusing for new users as it is very easy to mistype the syntax. It may also confuse the user as some of the characters in the syntax look similar which may result in the user writing the code wrong. Overall, the development process is easy for people who have experience with node.js and Docker but it can be difficult for beginners to produce node.js applications that can be run in Docker.

eCommerce Application

YouTube Video Link

Below is the link for the YouTube video which demonstrates the application, database, and files that were created when developing the application.

<https://youtu.be/ZbFWPBwjGL0>

Database Login Details

System: PostgreSQL

Server: postgres

Username: docker

Password: password

Database: mydb

References

- [S. Vennam, "Cloud Computing," 18 August 2020. [Online]. Available:
1 <https://www.ibm.com/cloud/learn/cloud-computing>. [Accessed 8 December 2021].
]
- ["What is cloud computing?," [Online]. Available: <https://azure.microsoft.com/en-gb/overview/what-is-cloud-computing/#cloud-deployment-types>. [Accessed 8 December 2021].
]
- [J. Pascal, "Benefits and Challenges of Deploying Platforms in the Cloud," 14 February 2020.
3 [Online]. Available: <https://www.intraway.com/blog/benefits-challenges-deploying-platforms-in-the-cloud/#:~:text=%20Benefits%20of%20Deploying%20Platforms%20in%20the%20Cloud,stored%20in%20Cloud%20applications%20from%20anywhere...%20More%20>. [Accessed 8 December 2021].
- ["12 Benefits of Cloud Computing," [Online]. Available:
4 <https://www.salesforce.com/products/platform/best-practices/benefits-of-cloud-computing/>.
] [Accessed 8 December 2021].
- [S. Ward, "The Disadvantages of Cloud Computing," 20 November 2019. [Online]. Available:
5 <https://www.thebalancesmb.com/disadvantages-of-cloud-computing-4067218#:~:text=The%20inflexibility%20of%20some%20cloud%20apps%20can%20be,allow%20easy%20transfer%20of%20information%20into%20other%20systems..> [Accessed 8 December 2021].
- ["Cloud Services pricing," [Online]. Available: https://azure.microsoft.com/en-gb/pricing/details/cloud-services/?OCID=AID2200274_SEM_fd27b4eb46c71fd370d1d55a7ece8bff:G:s&ef_id=fd27b4eb46c71fd370d1d55a7ece8bff:G:s&msclkid=fd27b4eb46c71fd370d1d55a7ece8bff#pricing. [Accessed 8 December 2021].
- [L. C. Team, "Cloud computing 101: The interrelationship of scalability, reliability, and availability,"
7 [Online]. Available: <https://www.lucidchart.com/blog/reliability-availability-in-cloud-computing>.
] [Accessed 9 December 2021].
- ["Virtualization," [Online]. Available: <https://www.ibm.com/topics/virtualization>. [Accessed 18
8 December 2021].
]
- ["Containers vs Virtual Machines (VMs): What's the Difference?," 15 April 2019. [Online].
9 Available: <https://phoenixnap.com/kb/containers-vs-vms#:~:text=Virtual%20machines%20are%20commonly%20used%20for%20demanding%20applications%2C,and%20caching%20services%2C%20network%20daemons%2C%20and%20small%20databases..> [Accessed 18 December 2021].

[S. J. Vaughan-Nichols, "What is Docker and why is it so darn popular?," 21 March 2018. [Online].
1 Available: <https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/>.
0 [Accessed 18 December 2021].
]

Appendix

Service.js:

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

```
const cors = require('cors');
```

```
const service = express();
```

```
const db = require("./pgdbgame");
```

```
const port = 1339;
```

```
service.use(express.json());
```

```
service.use(express.urlencoded({extended:true}));
```

```
let corsOptions = {
```

```
  origin: "*",
```

```
  methods: "GET,HEAD,PUT,PATCH,POST,DELETE",
```

```
  allowedHeaders:"Authorization, Origin, Content-Type, Accept, X-Requested-With",
```

```
  maxAge: 0
```

```
};
```

```
service.use(cors(corsOptions));
```

```
service.get('/', (req,res) => {
```

```
  res.send("Welcome!");
```

```
});
```

```
service.get('/hi', (req,res) => {
```

```
  res.send("Hello!");
```

```
});
```

```
service.get('/hello/:name', (req,res) => {
```

```
  res.send(`Hello ${req.params.name}`);
```

```
});
```

```
service.get('/error', (req,res) => {  
    res.status(403).end();  
});
```

```
service.post('/game', async (req, res) => {  
    await db.Insert({  
        Name: req.body.name,  
        Quantity: req.body.quantity,  
        Platform: req.body.platform  
    }, res);  
});
```

```
service.get('/game', async (req, res) => {  
    if(typeof req.query.name === "string")  
        await db.SelectByName(req.query.name, res);  
    else  
        await db.Select(res);  
});
```

```
service.get('/game/:id', async (req, res) => {  
    await db.SelectById(parseInt(req.params.id), res);  
});
```

```
service.delete('/game/:id', async (req, res) => {
```

```
    await db.Delete(parseInt(req.params.id), res);  
  });
```

```
service.listen(port, () => console.log(`Example web service listening at http://localhost:${port}`));
```

pgdbgame.js:

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

```
const {Client, Connection} = require('pg');
```

```
const format = require('pg-format');
```

```
function GetClient()
```

```
{
```

```
    return new Client({
```

```
        host: "postgres",      // Service name in docker-compose
```

```
        port: 5432,
```

```
        user: "docker",
```

```
        password: "password",
```

```
        database: "mydb"
```

```
    });
```

```
}
```

```
Insert = async (game, res) =>
```

```
{
```

```
    let client = GetClient();
```

```
    console.log(`Adding new game named ${game.Name}`);
```

```
    try
```

```
    {
```

```
        await client.connect();
```

```
        let query = format('INSERT INTO "Game"("Name", "Quantity", "Platform")  
VALUES(%L, %L, %L) RETURNING *',
```

```
            game.Name,
```

```
            game.Quantity,
```

```
            game.Platform);
```

```
        let results = await client.query(query);
```

```
        if(results.rowCount > 0)
```

```

        {
            console.log('New record added');
            res.status(201).send(results.rows[0]);
        }
        else
        {
            res.status(409).end();
        }
    }
    catch(error)
    {
        console.log(error);
        res.status(403).send(error);
    }
    finally
    {
        await client.end();
    }
};

```

Select = async (res) =>

```

{
    let client = GetClient();
    console.log(`Reading all the games details`);
    try
    {
        await client.connect();

        let query = 'SELECT * FROM "Game"';
        let results = await client.query(query);
    }
}

```

```

        if(results.rows.length > 0)
            console.log(`Read ${results.rows.length} records`)
        else
            console.log(`Database is empty`);

        res.send(results.rows);
    }
    catch(error)
    {
        console.log(error);
        res.status(403).send(error);
    }
    finally
    {
        await client.end();
    }
};

SelectById = async (id, res) =>
{
    let client = GetClient();
    console.log(`Looking for the game with id ${id}`);
    try
    {
        await client.connect();

        let query = format('SELECT * FROM "Game" WHERE "Id"=%L', id);
        let results = await client.query(query);

        if(results.rows.length > 0)
        {

```

```

        console.log(`Found ${results.rows.length} records`);
        res.send(results.rows[0]);
    }
    else
    {
        console.log(`Record not found for Id=${id}`);
        res.status(409).end();
    }
}
catch(error)
{
    console.log(error);
    res.status(403).send(error);
}
finally
{
    await client.end();
}
};

```

SelectByName = async (name, res) =>

```

{
    let client = GetClient();
    console.log(`Looking for all the games named ${name}`);
    try
    {
        await client.connect();

        let query = format('SELECT * FROM "Game" WHERE "Name"=%L', name);
        let results = await client.query(query);
    }
}

```

```

        if(results.rows.length > 0)
            console.log(`Found ${results.rows.length} records`);
        else
            console.log(`Records not found for ${name}`);

        res.send(results.rows);
    }
    catch(error)
    {
        console.log(error);
        res.status(403).send(error);
    }
    finally
    {
        await client.end();
    }
};

```

```

Delete = async (id, res) =>
{
    let client = GetClient();
    console.log(`Delete the game with id = ${id}`);
    try
    {
        await client.connect();

        let query = format('DELETE FROM "Game" WHERE "Id"=%L', id);
        let results = await client.query(query);

        if(results.rowCount > 0)
        {

```

```

        console.log('Record deleted');
        res.send({});
    }
    else
    {
        console.log(`Failed to delete record for Id = ${id}`);
        res.status(409).end();
    }
}
catch(error)
{
    console.log(error);
    res.status(400).send(error);
}
finally
{
    await client.end();
}
};

Patch = async (game, res) =>
{
    let client = GetClient();
    console.log(`Modifying the game with Id = ${game.Id}`);
    try
    {
        //let query: string = 'UPDATE "Game" SET ';
        let queryArgs = [];
        let values = [];
        await client.connect();
        if(game.Name != null) { values.push(game.Name); queryArgs.push(`"Name"= %L`); }
    }

```

```

        if(game.Quantity != null) { values.push(game.Quantity);
queryArgs.push("Quantity"=%L'); }

        if(game.Platform != null) { values.push(game.Platform);
queryArgs.push("Platform"=%L'); }

        values.push(game.Id); // Id of record to change

        let query = format.withArray('UPDATE "Game" SET ' + queryArgs.join(",") + ' WHERE
"Id"=%L',

                                                                    values);

        let results = await client.query(query);

        if(results.rowCount > 0)
        {
            console.log('Modified the record');
            res.send({});
        }
        else
        {
            console.log(`Failed to modify record for Id = ${game.Id}`);
            res.status(409).end();
        }
    }
    catch(error)
    {
        console.log(error);
        res.status(403).send(error);
    }
    finally
    {
        await client.end();
    }
};

```

```
module.exports = { Insert, Select, SelectById, SelectByName, Delete, Patch };
```

`server.js:`

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

```
const path = require('path');
```

```
const Axios = require('axios');
```

```
const server = express();
```

```
const port = 80;
```

```
let dirname = __dirname;
```

```
server.use( express.json());
```

```
server.use( express.urlencoded({extended:true}));
```

```
server.use( express.static( path.join(dirname, '/public')));
```

```
server.post('/addgame', async (req, res) => {
```

```
    try
```

```
    {
```

```
        let containerName = "webservice";
```

```
        let serviceResponse = await
```

```
            Axios.post(`http://${containerName}:1339/game`,{
```

```
                name: req.body["name-textbox"],
```

```
                quantity: req.body["quantity-numberbox"],
```

```
                platform: req.body["platform-textbox"]
```

```
            });
```

```
            console.log('Service response = ${serviceResponse.status}');
```

```
        res.redirect('thanks.html');
    }
    catch(error)
    {
        console.log(error);
        res.status(403).end();
    }
});

server.listen(port, () => console.log(`Example web server listening at http://localhost:${port}`));
```

[index.html](#):

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<link rel="stylesheet" href="themes/kalan1.min.css" /> <!-- These stylesheets and scripts allow
themes and features to be added to the webpage -->
```

```
<link rel="stylesheet" href="themes/jquery.mobile.icons.min.css" />
```

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile.structure-
1.4.5.min.css" />
```

```
<script src="jquery-3.6.0.js"></script>
```

```
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
```

```
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
```

```
<style>
```

```
#pageheading {
```

```
text-align: center;
```

```
font-size: 80px;
```

```
}
```

```
#mybutton.ui-btn { background-color : #79b9fb}
```

```
.maintext{
```

```
background-color: #99CCFF;
```

```
text-align: center;
```

```
}
```

```
</style> <!--This style tag allows things on the webpage to be styled in a certain way-->
```

```
<title>The Video Games Shop</title> <!-- This adds a title to the webpage-->
```

```
<meta charset="utf-8" />
```

```
</head>
```

```
<body>
```

```
<h1 id="pageheading">The Video Games Shop</h1> <!-- This adds a heading to the webpage-->
```

```
<div data-role="main" class="maintext"> <!-- This adds a navigation menu to the webpage-->
```

```
<h2>Navigation Menu</h2>
```

```
<a href="addgameform.html" id="mybutton" class="ui-btn ui-btn-inline">Games Page</a> <!--Adds  
a button that takes you to a different page when clicked-->
```

```
<a href="readgamespage.html" id="mybutton" class="ui-btn ui-btn-inline">Shopping Cart</a>
```

```
</div><br><br>
```

```
<div class="maintext"> <!--First paragraph on webpage-->
```

```
<h2> About The Website</h2>
```

```
<p>This website will allow you to purchase a variety of video games. On the Games Page, you can  
enter
```

```
the games that you want to purchase. You can then go to the Shopping Cart to view the games that  
you have
```

```
put in your cart. You can then checkout to purchase the games in your cart. We hope that the  
website provides
```

you with the games that you are looking for.</p>

</div>

</body>

</html>

Addgameform.html:

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="themes/kalan1.min.css" /> <!-- These stylesheets and scripts allow themes and features to be added to the webpage -->

<link rel="stylesheet" href="themes/jquery.mobile.icons.min.css" />

<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile.structure-1.4.5.min.css" />

<script src="jquery-3.6.0.js"></script>

<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>

<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>

<style>

#pageheading {

text-align: center;

font-size: 80px;

}

#mybutton.ui-btn { background-color : #79b9fb}

.maintext{

background-color: #99CCFF;

text-align: center;

```
}
```

```
</style> <!--This style tag allows things on the webpage to be styled in a certain way-->
```

```
<title>Games Page</title> <!-- This adds a title to the webpage-->
```

```
<meta charset="utf-8" />
```

```
</head>
```

```
<body>
```

```
<h1 id="pageheading">Games Page</h1> <!-- This adds a heading to the webpage-->
```

```
<div data-role="main" class="maintext"> <!-- This adds a navigation menu to the webpage-->
```

```
<h2>Navigation Menu</h2>
```

```
<a href="index.html" id="mybutton" class="ui-btn ui-btn-inline">Go To Previous Page</a> <!--Adds a button that takes you to a different page when clicked-->
```

```
</div><br><br>
```

```
<div class="maintext"> <!--First paragraph on webpage-->
```

```
<h2> Available Video Games & Platforms</h2>
```

```
<p><b>Game -</b> Spider-Man<br><b>Platforms -</b> PS4/PS5</p>
```

```
<p><b>Game -</b> God of War<br><b>Platforms -</b> PS4/PS5</p>
```

```
<p><b>Game -</b> Halo Infinite<br><b>Platforms -</b> Xbox One/Xbox Series X</p>
```

```
<p><b>Game -</b> Call of Duty: Vanguard<br><b>Platforms -</b> PS4/PS5/Xbox One/Xbox Series X</p>
```

```
<p><b>Game -</b> Battlefield 2042<br><b>Platforms -</b> PS4/PS5/Xbox One/Xbox Series X</p>
```

```
<p><b>Game -</b> Ghost of Tsushima<br><b>Platforms -</b> PS4/PS5</p>
```

```
<p><b>Game -</b> Fifa 22<br><b>Platforms -</b> PS4/PS5/Xbox One/Xbox Series X</p>
```

```
<p><b>Game -</b> Rocket League<br><b>Platforms -</b> PS4/PS5/Xbox One/Xbox Series X</p>
```

```
<p><b>Game -</b> Death Stranding<br><b>Platforms -</b> PS4/PS5</p>
```

```
<p><b>Game -</b> Far Cry 6<br><b>Platforms -</b> PS4/PS5/Xbox One/Xbox Series X</p>
```

```
<p><b>Game -</b> Guardians of the Galaxy<br><b>Platforms -</b> PS4/PS5/Xbox One/Xbox Series X</p><br>
```

```
</div>
```

```
<br><br>
```

```
<div class="maintext"> <!--This is where the user enters the games they want-->
```

```
<form action="addgame" method="POST">
```

```
<label for="name-textbox"><b>Game Name:</b></label>
```

```
<input id="name-textbox" name="name-textbox" type="text" />
```

```
<br/><label for="quantity-numberbox"><b>Quantity:</b></label>
```

```
<input id="quantity-numberbox" name="quantity-numberbox"
```

```
type="number" min="1" max="150" value="1" />
```

```
<br/><label for="platform-textbox"><b>Platform:</b></label>
```

```
<input id="platform-textbox" name="platform-textbox" type="text" />
```

```
<br/><br/><br/><input id="add-button" type="button" value="Add Game To Shopping Cart"/> <!--  
Buttons that add games to the cart-->
```

```
<br/><input type="submit" value="Add Game To Shopping Cart & Submit"/>
```

```
</form>
```

```
</div>
```

```
<Script src="/js/addgameform.js"></script>
```

```
</body>
```

```
</html>
```

Addgameform.js:

```
$(function ()
{
    $.ajaxSetup({cache: false});

    $("#add-button").click(AddGameClick);
});

function AddGameClick(e)
{
    let json = {
        name: $("#name-textbox").val(),
        quantity: parseInt($("#quantity-numberbox").val()),
        platform: $("#platform-textbox").val()
    };

    $.ajax({
        cache: false,
        type: "POST",
        url: 'http://localhost:1339/game',
        data: JSON.stringify(json),
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: (json, status, req) => console.log("Added game"),
        error: (req, status, error) => console.log("Failed to add game")
    });

    e.preventDefault();

    return false;
}
```

Readgamespage.html:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<link rel="stylesheet" href="themes/kalan1.min.css" /> <!-- These stylesheets and scripts allow  
themes and features to be added to the webpage -->
```

```
<link rel="stylesheet" href="themes/jquery.mobile.icons.min.css" />
```

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile.structure-  
1.4.5.min.css" />
```

```
<script src="jquery-3.6.0.js"></script>
```

```
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
```

```
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
```

```
<style>
```

```
#pageheading {  
text-align: center;  
font-size: 80px;  
}
```

```
#mybutton.ui-btn { background-color : #79b9fb}
```

```
.maintext{  
background-color: #99CCFF;  
text-align: center;  
}
```

```
</style> <!--This style tag allows things on the webpage to be styled in a certain way-->
```

```
<title>Shopping Cart Page</title> <!-- This adds a title to the webpage-->
```

```
<meta charset="utf-8" />
```

```
</head>
```

```
<body>

<h1 id="pageheading">Shopping Cart Page</h1> <!-- This adds a heading to the webpage-->


<div data-role="main" class="maintext"> <!-- This adds a navigation menu to the webpage-->

<h2>Navigation Menu</h2>

<a href="index.html" id="mybutton" class="ui-btn ui-btn-inline">Go To Previous Page</a> <!--Adds a
button that takes you to a different page when clicked-->

</div><br><br>


<div class="maintext"> <!--This is where the user enters the games they want-->


<form action="" method="POST">

<input id="read-button" type="button" value="View All Games In Shopping Cart"/> <!--Adds a
button that shows all of the games in the shopping cart when clicked-->

</form>


<div id="all-games">Waiting...</div><br><br> <!--This is where the games in the shopping cart are
displayed-->

</div>


<a href="checkout.html" id="mybutton" class="ui-btn ui-btn-inline">Click To Checkout</a> <!--Adds
a button that takes you to a different page when clicked-->


<Script src="/js/readgamespage.js"></script> <!-- Script linked to readgamespage.js file -->

</body>

</html>
```

Readgamespage.js:

```
$(function ()
{
    $.ajaxSetup({cache: false});
    $("#read-button").click(ReadGameClick);
});

function ReadGameClick(e)
{
    $.ajax({
        cache: false,
        type: "GET",
        url: 'http://localhost:1339/game',
        data: "",
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: ReadAllResult,
        error: (req, status, error) => console.log("Failed to read games")
    });

    e.preventDefault();
    return false;
}

function ReadAllResult(json, status, req)
{
    let html = "";
    for(let s in json)
    {
        let game = json[s];
        html += `Id: ${game.Id} Name: ${game.Name} `;
    }
}
```

```

        html += `Quantity: ${game.Quantity} Platform: ${game.Platform}<br/>`;
    }
    $("#all-games").html( html );
}

```

Thanks.html:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<link rel="stylesheet" href="themes/kalan1.min.css" /> <!-- These stylesheets and scripts allow
themes and features to be added to the webpage -->
```

```
<link rel="stylesheet" href="themes/jquery.mobile.icons.min.css" />
```

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile.structure-
1.4.5.min.css" />
```

```
<script src="jquery-3.6.0.js"></script>
```

```
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
```

```
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
```

```
<style>
```

```
#pageheading {
```

```
text-align: center;
```

```
font-size: 80px;
```

```
}
```

```
#mybutton.ui-btn { background-color : #79b9fb}
```

```
</style> <!--This style tag allows things on the webpage to be styled in a certain way-->
```

```
<title>Added to Cart Page</title> <!-- This adds a title to the webpage-->
```

```
<meta charset="utf-8" />
```

```
</head>
```

```

<body>
<h1 id="pageheading">Added to Cart Page</h1> <!-- This adds a heading to the webpage-->

<div data-role="main" class="maintext"> <!-- This adds a navigation menu to the webpage-->
<h2>Navigation Menu</h2>
<a href="addgameform.html" id="mybutton" class="ui-btn ui-btn-inline">Go To Previous Page</a>
<!--Adds a button that takes you to a different page when clicked-->

</div><br><br>

<p style="font-size:30px;"><b>You have added a new game to your shopping cart!</b></p> <!--First
paragraph on webpage-->

</body>
</html>

```

[Checkout.html:](#)

```

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="themes/kalan1.min.css" /> <!-- These stylesheets and scripts allow
themes and features to be added to the webpage -->

<link rel="stylesheet" href="themes/jquery.mobile.icons.min.css" />

<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile.structure-
1.4.5.min.css" />

<script src="jquery-3.6.0.js"></script>

<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>

<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>

<style>
#pageheading {

```

```
text-align: center;
font-size: 80px;
}
```

```
#mybutton.ui-btn { background-color : #79b9fb}
```

```
</style> <!--This style tag allows things on the webpage to be styled in a certain way-->
```

```
<title>Checkout Page</title> <!-- This adds a title to the webpage-->
```

```
<meta charset="utf-8" />
```

```
</head>
```

```
<body>
```

```
<h1 id="pageheading">Checkout Page</h1> <!-- This adds a heading to the webpage-->
```

```
<div data-role="main" class="maintext"> <!-- This adds a navigation menu to the webpage-->
```

```
<h2>Navigation Menu</h2>
```

```
<a href="readgamespage.html" id="mybutton" class="ui-btn ui-btn-inline">Go To Previous Page</a>
```

```
<!--Adds a button that takes you to a different page when clicked-->
```

```
</div><br><br>
```

```
<p style="font-size:30px;"><b>You have successfully checked out. Your items will be with you soon!</b></p> <!--First paragraph on webpage-->
```

```
</body>
```

```
</html>
```

Kalan Assignment 2

GRADEMARK REPORT

FINAL GRADE

84/100

GENERAL COMMENTS

Instructor

Cloud Computing (40): 37

Virtual & Container (15): 15

Eval Dev (25): 20

eComm (20): 12

Feedback:

=====

Nice discussion of cloud computing.

Very good discussion of VMs and containers,.

I wanted more of your evaluation of the process.

Provide a basic ecommerce site using in class code. OK demo of web services.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31
