

## Cloud Computing 2022:

### Assignment 3:




#### Report:

s3879008 : Thomas Lewis

#### a. Links:

Live URL of your project

- <http://54.234.168.110/> (URL is dynamic due to student AWS Credentials resetting each lab)

Instance summary for i-03132df4aab2d683a (My Web Server) <a href="#">Info</a>	
Updated less than a minute ago	
Instance ID	Public IPv4 address
 i-03132df4aab2d683a (My Web Server)	 54.234.168.110   <a href="#">open address</a> 

GitHub Repository URL:

<https://github.com/S3879008/Cloud-A3>

#### b. Summary:

The objective is to directly improve the destiny 2 player experience within the Raid activities available within the live service game, purpose of your project is specified to the learning of the raid mechanics and finding a teammate though LFG (Looking for Group) to complete the 6-player raid.

#### c. Introduction:

i. The motivation of this application is to support a player in learning about a given raid within the destiny 2 game & furthermore search for or make a team to complete the raid they are learning about primarily because the game provides little to no support for this in game as a non-matchmade activity.

ii. Use of the application provides a user with the information pertaining to encounter information related to every available current raid within the game and furthermore allows a user to make a post or search for other user's posts, the posts that users share contain a join code/ id that is directly related to the game join methods and a raid id that is the key a user searches by that are in place as an LFG for raiding.

iii. Destiny 2 the live service game doesn't pertain a match maker for raids, what this means is that everyone that joins must either have your join code or be your friend which can be quite inconvenient for the player making content they paid for inaccessible without external tools.

iv. An application case example would be a new player joining a post or making their own for a raid that they are interested in after or while using the app to learn about the raid of choice and then other users interacting and joining up as they are interested in the same raid to cooperate and complete the encounters described in the app.

v. The central focus on raids allowed me to have an app that contains the relevant information to the game activity and additionally the LFG built into one place as most of the information is only found through social media sharing and user generated content whereas LFG tends to come from bungie, but they don't depict raid solutions (because they are meant to be hard to learn) primary advantage it's all-in-one place.

#### d. Related work:

Bungies LFG: <https://www.bungie.net/en/ClanV2/FireteamSearch>

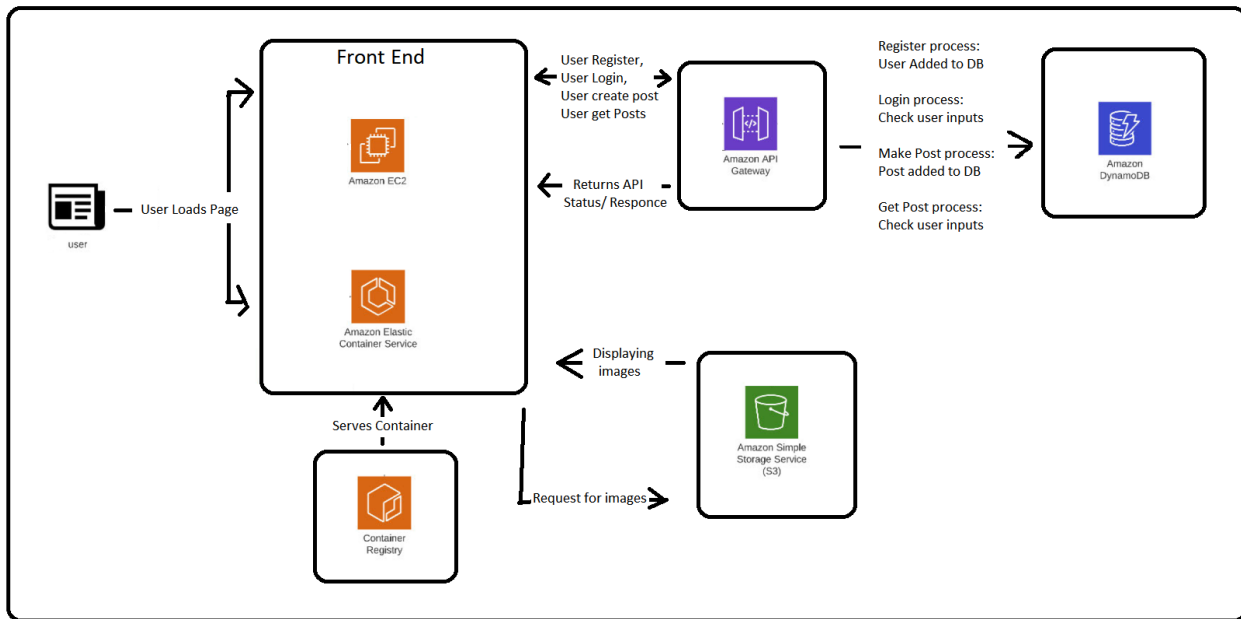
Destinypedia Raid info: <https://www.destinypedia.com/Raid>

100.io LFG: <https://www.the100.io/destiny-2/lfg>

Destiny Fandom Raid info: <https://destiny.fandom.com/wiki/Raid>

#### e. System Architecture

## System Architecture



### f. Developer Manual:

#### **Step 1: Create an Ubuntu EC2 Instance on AWS**

- Log in to AWS Console
- Go to EC2 Section and select Ubuntu (18.4 AMI)
- Select an instance type i.e., T2.micro
- Navigate to the Security Groups Tab
- Allow Inbound Traffic (HTTP: 80), (SSH: 22), (HTTPS: 443)
- Create the Key-Pair (Keep This you only get one)

#### **Step 2: SSH into Ubuntu EC2**

- open a terminal (Putty for Windows or normal CMD for Apple)
- Putty Gen the Key-Pair for Windows or for Apple chmod 400 the Key-Pair
- SSH into the Instance using Putty for Windows or for Apple (`ssh -i <your key name>.pem ubuntu@<Public DNS of your EC2>`)

#### **Step 3: Create Flask (Python) Application Inside EC2**

- Install Python Virtual Environment
  - "\$ sudo apt-get update"
  - "\$ sudo apt-get install python3-venv"
- Activate the Virtual Environment
  - "// Create directory
  - \$ mkdir helloworld
  - \$ cd helloworld"
  - "// Create the virtual environment
  - \$ python3 -m venv venv"
  - "// Activate the virtual environment
  - \$ source venv/bin/activate"

```
// Install Flask
$ pip install Flask
```

- Create a Flask Api  
"\$ sudo nano app.py"

```
// Add this to app.pyfrom flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

```
if __name__ == "__main__":
    app.run()
```

```
// close the sudo nano after saving these changes
```

- Verify its running  
run the following in the SSH terminal to verify that its working  
"python app.py"

You should see a lazy loaded local flask app running  
To stop this running use "Ctrl+C"

#### **Step 4: Run Gunicorn WSGI server to serve the Flask Application**

- Install Gunicorn using  
"\$ sudo pip install gunicorn"

- Run Gunicorn using  
"\$ gunicorn -b 0.0.0.0:8000 app:app"  
(Ctrl + C to exit gunicorn)

#### **Step 5: Use systemd to manage Gunicorn**

- Navigate to "/etc/systemd/system" within your SSH terminal

- create a service file to serve gunicorn  
"\$ sudo nano /etc/systemd/system/helloworld.service"

- add the following to the created file

```
[Unit]
Description=Gunicorn instance for a simple hello world app
After=network.target[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/helloworld
ExecStart=/home/ubuntu/helloworld/venv/bin/gunicorn -b localhost:8000 app:app
Restart=always[Install]
WantedBy=multi-user.target
```

- enable the service

```
"$ sudo systemctl daemon-reload"
```

```
"$ sudo systemctl start helloworld"
```

```
"$ sudo systemctl enable helloworld"
```

What the above dose is call gunicorn to run when the EC2 instance is restarted.

You can check its running using the following

```
"$ curl localhost:8000"
```

### **Step 6: Run Nginx Webserver to route Gunicorn**

- Install Nginx

```
"$ sudo apt-get nginx"
```

- Start Nginx service

```
"$ sudo systemctl start nginx"
```

```
"$ sudo systemctl enable nginx"
```

- edit the default site-available

```
"$ sudo nano /etc/nginx/sites-available/default"
```

- Add the following to the top of the file

```
"upstream flaskhelloworld {  
server 127.0.0.1:8000;  
}"
```

- Then a proxy\_pass at the location/ function

```
"location / {  
proxy_pass http://flaskhelloworld;  
}"
```

- save and close the file

- Restart Nginx

```
"$ sudo systemctl restart nginx"
```

At this point you have a helloworld application working through a flask-based python application hosted in EC2

### **Step 7: Install Boto3**

-Using the following comand you can install the Boto3 API to serve your AWS requests

```
"$ python -m pip install boto3"
```

### **Step 8: AWS Credentials (AWS CLI)**

- navigate to the .aws file

```
"$ cd .aws"
```

- Modify the credentials to match your own (Access key, Secret Access Key, Session Token (if applicable))

```
"$ sudo nano credentials"
```

- Aswell the region (I.e. "us-east-1")  
"\$ Sudo nano config"

### Step 9: Static

- script.js  
// currently unused within the project but is available  
- style.css  
// visual aesthetic is entirely up to you, this is what I used as a base  
"body {  
  
font-family: "helvetica", sans-serif;  
  
text-align: center;  
  
}"

### Step 10: Login & Register

- change index.html to reflect a login page  
"<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<title>CC2022 Assignment Three</title>  
  
<script src="{{ url\_for('static', filename='script.js') }}"></script>  
  
<link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">  
  
</head>  
  
<body>  
  
<h1>Login Page</h1>  
  
<form action="/check\_login/" target="\_self" method="post">  
  
<p>Email:</p>  
  
<input type="text" placeholder="Enter Email" name="email" id="email"> <br>  
  
<p>UserName:</p>  
  
<input type="text" placeholder="Enter Username" name="username" id="username"> <br>  
  
<p>Password:</p>  
  
<input type="text" placeholder="Enter Password" name="password" id="password"><br><br>  
  
<input type="submit">  
  
</form>  
  
<br>  
  
<a href="{{ url\_for('load\_register') }}">Register New Account</a>  
  
</body>

</html>"

- make a register.html

"<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

<link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<h1>Register Page</h1>

<form action="/check\_register/" target="\_self" method="post">

<p>User Email:</p>

<input type="text" placeholder="Enter email" name="email" id="email"> <br>

<p>Username:</p>

<input type="text" placeholder="Enter Username" name="username" id="username"> <br>

<p>Password:</p>

<input type="text" placeholder="Enter Password" name="password" id="password"><br><br>

<input type="submit">

</form>

<br><a href="{{ url\_for('load\_index') }}">Login With Existing Account</a>

</body>

</html>"

- make a main.html template to load from login

"<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

<link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"> </head>

<body>

<h1>Hello, you're logged in</h1>

</body>

</html>"

- create user table in DynamoDB

(attributes= email (String, Primary Key), password (String, Associate Key))

for an example, please refer to the following as a guide

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/getting-started-step-1.html>

- make app.py functions

```
"from flask import Flask, render_template, request
```

```
import boto3
```

```
from boto3.dynamodb.conditions import Key, Attr
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def hello_world():
```

```
    return render_template('index.html')
```

```
@app.route('/load_register/', methods=["GET", "POST"])
```

```
def load_register():
```

```
    return render_template(
```

```
        'register.html')
```

```
@app.route('/load_index/', methods=["GET", "POST"])
```

```
def load_index():
```

```
    return render_template(
```

```
        'index.html')
```

```
def check_email():
```

```
    email = request.form.get('email')
```

```
    username = request.form.get('username')
```

```
    password = request.form.get('password')
```

```
    dynamodb_resource = boto3.resource('dynamodb')
```

```
    table = dynamodb_resource.Table('login')
```

```
    response = table.get_item(
```

```
        Key={
```

```
            'email': email
```

```
        }
```

```
    )
```

```
    try:
```

```

    items = response['Item']
except KeyError:
    items = ""
if items != "":
    if password != items['password']:
        items = ""
return dict(items)

```

```

@app.route('/check_login/', methods=["GET", "POST"])
def check_login():
    emailnow = check_email()
    username = request.form.get('username')
    error = 'error login details dosent exist i.e. password or email'
    if not emailnow:
        return (error)
    else:
        return render_template('main.html', username=username)

```

```

@app.route('/check_register/', methods=["GET", "POST"])
def check_register():
    username = request.form.get('username')
    email = request.form.get('email')
    password = request.form.get('password')
    dynamodb_resource = boto3.resource('dynamodb')
    table = dynamodb_resource.Table('login')
    responce0 = table.get_item(
        Key={
            'email': email
        }
    )
    try:
        items = responce0['Item']
    except KeyError:
        items = ""

```



```

if items == "":

    response1 = table.put_item(

        Item={

            'user_name': username,

            'email': email,

            'password': password

        }

    )

    return render_template('index.html')

else:

    return 'Error: Email Already Exists'

if __name__ == "__main__":

    app.run()

```

At this stage you should have a working login and register assuming you've correctly followed the steps.

### Step 10: S3 Images

- For the following parts come from a tutorial

<https://www.youtube.com/watch?v=cU92Ums-MWs>

- make bucket (Public access)
- make a policy for public object access
- upload objects (images are provided in the app images folder found within my submission)

### Step 11: Main and Pages

- update the main.html to reflect the images and pages we are about to make

```
<h1>User
```

```
Area</h1> <a href="{{ url_for('load_index') }}">Logout</a> <p> Welcome {{username}}</p> <h2>Destiny 2 Raids
</h2> <a href="{{
```

```
url_for('load_VOTD') }}">Vow of The Deciple</a><br> <br> <a href="{{ url_for('load_VOG') }}">Vault of Glass</a><br> <br> <a href="{{
url_for('load_DSC')
```

```
}}">Deep Stone Crypt</a><br> <br> <a
```

```
href="{{ url_for('load_GOS') }}">Garden of Salvation</a><br> <br> <a href="{{ url_for('load_LW') }}">Last Wish</a><br> <br>"

- create the pages for the guide content

lw.html

"<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

<link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"> </head>

<body>

<h1>Last Wish</h1>

<h2>Wish Wall: Wishes</h2>

<br>

<h2>Encounter: Kali the Corrupted</h2>

<br>

<h2>Encounter: Shuro Chi the Corrupted</h2>

<br>

<h2>Encounter: The Vault</h2>

<br>

<h2>Encounter: Riven of a Thousand Voices</h2>

<br>

<h2>Encounter: Queens Walk</h2>

<br>

</body>

</html>"

vog.html

"<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

<link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"> </head>

<body>

<h1>Vault of Glass</h1>

<h2>Encounter: Conflux & Oracle</h2>

<br>

<h2>Encounter: Templar</h2>

<br>

<h2>Encounter: Gatekeeper</h2>

<br>

<h2>Encounter: Atheon</h2>

<br>

</body>

</html>

votd.html

"<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

<link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"> </head>

<body>

<h1>Vow of The Deciple</h1>

<h2>Icon Key For Raid: </h2>

<br>

<h2>Encounter: Acquisition</h2>

<br>

<h2>Encounter: Exhibition</h2>

<br>

<h2>Encounter: Rhulk</h2>

<br>

</body>

</html>

gos.html

"<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

```

<script src="{{ url_for('static', filename='script.js') }}"></script>
<link type="text/css" rel="stylesheet" href="{{ url_for('static', filename='style.css') }}"> </head>
<body>
<h1>Garden of Salvation</h1>
<h2>Encounter: Evade</h2>
<br>
<h2>Encounter: Summon</h2>
<br>
<h2>Encounter: Consecrated Mind</h2>
<br>
<h2>Encounter: Sanctified Mind</h2>
<br>
</body>
</html>”

```

#### dsc.html

```

“<!DOCTYPE html>

<html>

<head>

<title>CC2022 Assignment Three</title>

<script src="{{ url_for('static', filename='script.js') }}"></script>

<link type="text/css" rel="stylesheet" href="{{ url_for('static', filename='style.css') }}"> </head>

<body>

<h1>Deep Stone Crypt</h1>

<h2>Encounter: Entry</h2>

<br>

<h2>Encounter: Security</h2>

<br>
<br>

<h2>Encounter: Replication</h2>

<br>
<br>

<h2>Encounter: Disarmament</h2>

<br>

<h2>Encounter: Abomination</h2>

```

```
<br>
</body>
</html>"
```

- update app.py to allow loading to content pages

```
@app.route('/load_VOTD/', methods=["GET", "POST"])
```

```
def load_VOTD():
    return render_template(
        'votd.html')
```

```
@app.route('/load_VOG/', methods=["GET", "POST"])
```

```
def load_VOG():
    return render_template(
        'vog.html')
```

```
@app.route('/load_DSC/', methods=["GET", "POST"])
```

```
def load_DSC():
    return render_template(
        'dsc.html')
```

```
@app.route('/load_GOS/', methods=["GET", "POST"])
```

```
def load_GOS():
    return render_template(
        'gos.html')
```

```
@app.route('/load_LW/', methods=["GET", "POST"])
```

```
def load_LW():
    return render_template(
        'lw.html')
```

## Step 12: User Posts

- make post table In DynamoDB (attributes= id(String, Key))

- make index for (attributes= raid(String, Key))

to make a index follow the example from AWS

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.Indexes.Creating.html>

- update main.html to reflect a make and get post

```
"<h2>LFG Raid Posts</h2>
```

```
<p>Make A Post</p>
```

```
<form action="/check_post/" target="_self" method="post">
```

```
<p>ID(Join Code): </p>
```

```
<input type="text" placeholder="Enter Your Join Code" name="id" id="id"> <br>
```

```
<p>Raid: </p>
```

```
<input type="text" placeholder="Enter Raid Name" name="raid" id="raid"><br><br>
```

```
<input type="hidden" id="username" name="username" value="{{username}}">
```

```
<input type="submit">
```

```
</form>
```

```
<h3>Find A Post<h3>
```

```
<form action="/get_post/" target="_self" method="post">
```

```
<p>Raid: </p>
```

```
<input type="text" placeholder="Enter Raid Name" name="raid" id="raid"><br><br>
```

```
<input type="hidden" id="username" name="username" value="{{username}}">
```

```
<input type="submit">
```

```
</form>
```

```
<h3>Results: </h3>
```

```
<p> {{items}} </p>"
```

- update app.py to call functions

```
"@app.route('/check_post/', methods=["GET", "POST"])
```

```
def check_post():
```

```
    id = request.form.get('id')
```

```
    raid = request.form.get('raid')
```

```
    username = request.form.get('username')
```

```
    dynamodb_resource = boto3.resource('dynamodb')
```

```
    table = dynamodb_resource.Table('post')
```

```
    response = table.put_item(
```

```
        Item={
```

```
        'id': id,
```

```
        'raid': raid
```

```
    }
```

)

```
return render_template('main.html', username = username)
```

```
@app.route('/get_post/', methods=["GET","POST"])
```

```
def get_post():
```

```
    username = request.form.get('username')
```

```
    raid = request.form.get('raid')
```

```
    if raid != "":
```

```
        dynamodb_resource = boto3.resource('dynamodb')
```

```
        table = dynamodb_resource.Table('post')
```

```
        response = table.query(
```

```
            IndexName='raid-index',
```

```
            KeyConditionExpression=Key('raid').eq(raid))
```

```
        try:
```

```
            items = response['Items']
```

```
        except KeyError:
```

```
            items = "Error: No Raid Posts Available"
```

```
    if raid == "":
```

```
        items = "Error: Null Input"
```

```
    return render_template('main.html', username = username, items = items)"
```

### Step 13: You're Done

- You now have a working LFG and Raid Guide for Destiny 2 from a python Flask based app within a EC2 instance

### g. A small user manual:

A quick overview of how to use your application.

### Register

# Register Page

User Email:

Username:

Password:

[Login With Existing Account](#)

- A form that allows you to become a user to access the application
- will require you make a username, password and provide an email
- It is recommended that you make your username the same as your Bungie join ID (Your Bungie username), this will simplify the utilization of searching and making a posts.

## Login

# Login Page

Email:

UserName:

Password:

[Register New Account](#)

- using the credentials, you've created in login you can access the app
- this takes you to the main page

## Navigation of Content



## Main Page:

---

**User Area**  
[Logout](#)  
Welcome Yikes(Adept)#1125

**Destiny 2 Raids**  
[View of The Disciple](#)  
  
[Vault of Glass](#)  
  
[Dance of the Nine](#)  
  
[Garden of Salvation](#)  
  
[Last Wish](#)  


**LFG Raid Posts**  
Make A Post  
ID(Join Code):  
  
Raid:  
  
  
Find A Post  
Raid:  
  
  
Results:

- the main page presents you with the options to a user area, view raid guides and make or search for a LFG post

## User Area:

# User Area

[Logout](#)

Welcome Yikes(Adept)#1125

- the user area will reflect a logout option and your own username used in the id for posts.

## Raid guide Example: Vow of the Disciple

## Vow of The Deciple



- Raid Guides have a link with a logo.

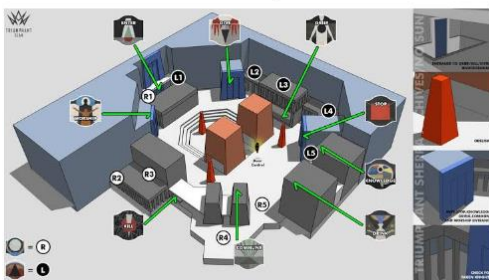
## Content Example: Vow of the Disciple

### Vow of The Deciple

Icon Key For Raid:



### Encounter: Acquisition



- these present you with all the relevant information to the raid you selected from the main page as encounter references the further you scroll down.

## Making Post

# LFG Raid Posts

Make A Post

ID(Join Code):

Raid:

- these require you to provide your join id for your bungie account & the raid you're doing as this is how people will be able to find and join you through the app

## Query Posts

### Find A Post

Raid:

Results:

```
[{'raid': 'Last Wish', 'id': 'Yikes(Adept)#1125'}]
```

- searching by the raid name you will be able to see players that have made a post you can join to participate in a raid, in this example I've searched for Last Wish.
- by using the id result we can join the player associated to the post in game using the /join command
- using my example again this would appear in game as "/join Yikes(Adept)#1125" allowing you to join me for the last wish raid I've posted.

## h. References:

### Python based EC2 Web App Guide

Medium. 2022. *Step-by-step visual guide on deploying a Flask application on AWS EC2*. [online] Available at: <https://medium.com/techfront/step-by-step-visual-guide-on-deploying-a-flask-application-on-aws-ec2-8e3e8b82c4f7> [Accessed 16 May 2022].

### Raid Images (Guide Content)

Reddit.com. 2022. [online] Available at: <https://www.reddit.com/r/destiny2/> [Accessed 16 May 2022].

### Seal Images for Raid Icons

Light.gg. 2022. *Seals - Destiny 2 Legend - light.gg*. [online] Available at: <https://www.light.gg/db/legend/616318467/seals/> [Accessed 16 May 2022].

### AWS Make a DynamoDB Table

2022. [online] Available at: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/getting-started->

[step-1.html](#)> [Accessed 16 May 2022].

### **Static images for a website**

Youtube.com. 2022. [online] Available at: <<https://www.youtube.com/watch?v=cU92Ums-MWs>> [Accessed 16 May 2022].

### **AWS Make a Index**

2022. [online] Available at:

<<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.Indexes.Creating.html>>

[Accessed 16 May 2022].