

CAB432 Report

Phillipe Sebastiao - n11029935

Timothy Ryan - n11094168

GIF MAKER

Choose File

No file chosen

Enter width

Enter height

Enter duration (seconds)

default fps

Convert To GIF

Contents

| | |
|--|----|
| Introduction | 3 |
| Purpose & Description | 3 |
| Services Used | 3 |
| Use Cases | 4 |
| Technical breakdown | 4 |
| Architecture | 4 |
| Client / Server Demarcation of Responsibilities | 5 |
| Response Filtering / Data Object Correlation | 5 |
| Scaling & Performance | 6 |
| Test plan | 7 |
| Difficulties / Exclusions / Unresolved & Persistent Errors | 8 |
| Extensions | 8 |
| User guide | 9 |
| Appendices | 10 |
| Appendix 1: Code Snippets | 10 |
| Appendix 2: ASG Group Details | 11 |
| Appendix 3: Text Cases | 12 |

Introduction

Purpose & Description

The GIF Maker application provides an ease-of-use interface for converting video files of any type into a GIF. The application allows users to select parameters such as duration, resolution and framerate, to produce high quality GIFs that can be saved locally or accessed for later use. The processing of videos to GIFs is done in an efficient manner completely away from the client, so there is no computational power needed.

Services Used

FFmpeg & Fluent FFMpeg -

<https://www.ffmpeg.org/documentation.html> | <https://www.npmjs.com/package/fluent-ffmpeg>

FFmpeg is a tool that handles processing of video and audio files. It can convert, encode, and transcode multimedia files, enabling format, codec conversions and other advanced capabilities. The Fluent FFMpeg-API package for NodeJS is used to interact with this tool to convert videos into GIFs.

Amazon EC2 - <https://docs.aws.amazon.com/ec2/>

Amazon EC2 is a cloud computing service that offers virtual machines, known as instances, which can be configured and customized according to specific computing needs. Users can choose the instance type, operating system, and software stack to meet their specific application requirements. This flexibility makes EC2 suitable for hosting web applications, running data analysis, and performing various computing tasks in the cloud, providing on-demand computing resources with scalability and control.

Amazon S3 - <https://docs.aws.amazon.com/s3/>

Amazon S3 is a scalable and durable object storage service. It can store data, files, and objects. You can access your stored items from anywhere over the internet. S3 is designed to handle large amounts of data and offers high availability and data redundancy, making it suitable for various use cases like hosting static websites, storing backups, and managing large datasets.

Amazon SQS - <https://docs.aws.amazon.com/sqs/>

Amazon SQS is a distributed message queuing service. It acts as an intermediary between different parts of a software application. When one component has a message to send to another, it places the message in a queue. The receiving component can then retrieve and process messages from the queue. This decoupling of components ensures that messages are reliably transmitted, even if some parts of the application are temporarily unavailable. It's a useful tool for managing communication between different software components in a distributed system.

PM2 - <https://pm2.keymetrics.io/docs/usage/pm2-doc-single-page/>

PM2 is a process manager for JavaScript that can be setup in instances in advance, when the instance is deployed it will run a script. This can be used to start scripts in images, manage the application 24/7 and improve workflow.

Tailwind CSS - <https://tailwindcss.com/>

Tailwind CSS is a utility-first CSS framework that streamlines the process of designing and styling web interfaces. It provides a comprehensive set of CSS classes that correspond to common design patterns and styles.

Use Cases

Use Case 1

As a user, I want to convert a video into a GIF format with an appropriate quality, so that I can create short, animated clips for sharing.

- FFmpeg will convert video files into GIFs with Amazon SQS to queue the GIFs to be converted.

Use Case 2

As a user, I want to convert video files into a GIF and adjust their size, duration and frame rate, so that I can create custom GIFs to use.

- FFmpeg can adjust the size, duration and frame, which will be passed as parameters in SQS.

Use Case 3

As a user, I want to efficiently convert video files into GIFs, so that I can store them in a database for later access.

- S3 will be used to store GIFs and users can access this through the GIF ID.

Technical breakdown

Architecture

The GIF Maker application works by having a front end (client) that the users interact with and upload their videos to. The videos are then sent to an Amazon S3 Bucket with an ID, and a message is sent to Amazon SQS, to be added onto a queue, with the unique ID and parameters selected by the user from the user interface. The workers are EC2 instances scaled by an Auto Scaling Group. They are constantly polling SQS for messages in the queue. After receiving a message, the workers use the ID to pull the video from S3. They then use the parameters from the SQS message to convert the video to a GIF. Once the worker finishes making the GIF, it will upload it back to S3 to replace the original video file, while this is happening the front end will be polling the S3 Bucket for the unique ID. Once the ID is found, it will pull the GIF and display it back to the user.

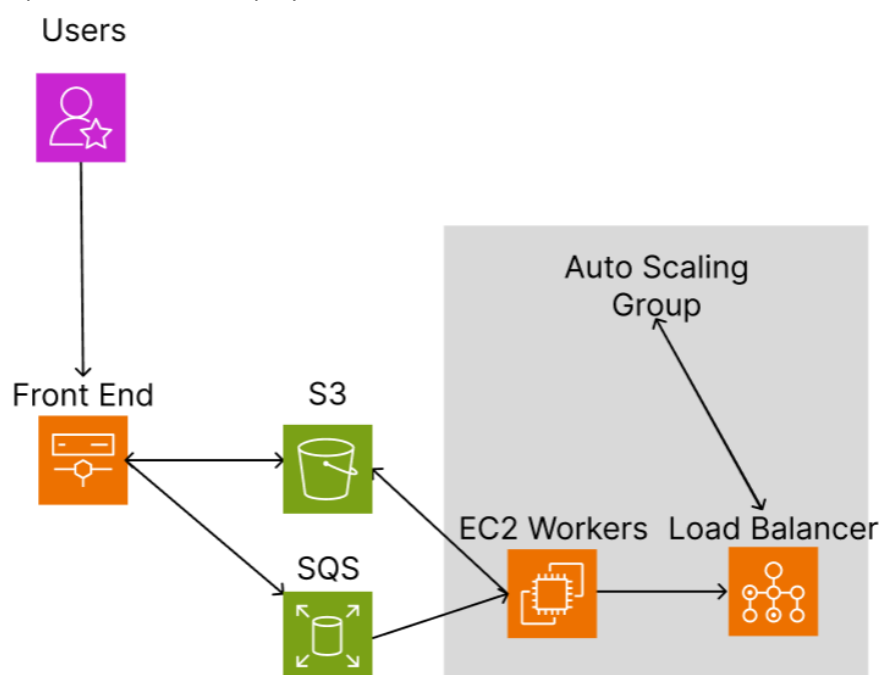


Figure 1: Architecture Diagram

Client / Server Demarcation of Responsibilities

On the front end there are input fields for specifying GIF parameters, these include the width and height, duration and framerate. The user will then click to convert to a GIF which will upload the raw video file to S3 with a unique ID. The parameters, alongside a uniquely generated ID, are passed in the body of a message sent to the SQS queue to be processed by the EC2 workers as seen in [Figure 1](#). The workers are constantly waiting to process a message seen in the example code below:

```
function pollQueue() {
  sqs.receiveMessage(params, (err, data) => {
    if (data.Messages) {
      data.Messages.forEach((message) => {
        processMessage(message);
      });
    }
    pollQueue();
  });
}
```

Passing of these parameters can be seen in [Appendix 1: Figure 1](#). The worker processes the message in the SQS queue and will set a timeout on the message so other workers cannot access the message while the video is being converted using FFmpeg as seen in [Appendix 1: Figure 2](#). The worker downloads the video file using the unique ID passed in the SQS message. The process of converting a GIF is done asynchronously, able to make multiple conversions at once. On conversion the worker will delete the SQS message and replace the video file with the converted GIF. While the video is being processed the front end will continuously poll S3 for a GIF with the unique ID to display to the user. There is an auto scaling group which uses a load balancer to create more instances of workers as the CPU load increases to a target of 50% utilization.

Response Filtering / Data Object Correlation

The main data passed through the application is through the client, being a video tagged with a unique ID uploaded to a S3 bucket and an SQS message tagged with the chosen parameters and unique ID in the body. The code below shows how this data is passed from the frontend with the parameters being passed can be seen in [Appendix 1: Figure 1](#):

```
await uploadVideoToS3(inputVideoFilePath, uniqueID, fileExtension);
await sendSQSMessage(uniqueID, fileExtension, parameters);

res.status(200).redirect(`/gif/${uniqueID}`);
```

The `/gif/:uniqueID` endpoint seen in the above code, is where the client will wait for the specific ID of the GIF to appear in the S3 bucket. When a message is received by the worker, a Read Stream will be opened to grab the video from the S3 bucket to be processed by FFmpeg on the end of the stream seen in the code below:

```
const s3ReadStream = s3.getObject(s3Params).createReadStream();
s3ReadStream.pipe(writeStream);

// Wait for the download to finish.
s3ReadStream.on("end", () => {
  let ffmpegCommand = ffmpeg(videoFilePath);
```

Scaling & Performance

Scaling the EC2 workers was executed by using an Auto Scaling Group seen in the architecture diagram in [Figure 1](#), which only scales the workers to target a 50% CPU utilization. The CPU load comes from the conversion of videos to GIFs. It uses a Load Balancer to manage incoming traffic as seen in [Figure 1](#) and a group size with a maximum of 5, minimum of 1 and a desired capacity of 1 as seen in the architecture diagram in [Appendix 2: Figure 1](#). The new instances that were created from scaling the ASG are only able to grab new messages from SQS, as messages being processed are timed out by the worker that processed them. This means that if a worker has 6 or 7 videos that it is converting and a new worker is created, the load will still be distributed evenly.

In [Figures 2 & 3](#) below it can be seen that as the CPU load increases the desired capacity will scale out to reduce the load on the application to attempt to achieve 50% CPU utilization across the workers. Then as load goes away, there are not as many GIFs to convert, the desired capacity will drop back down to one.

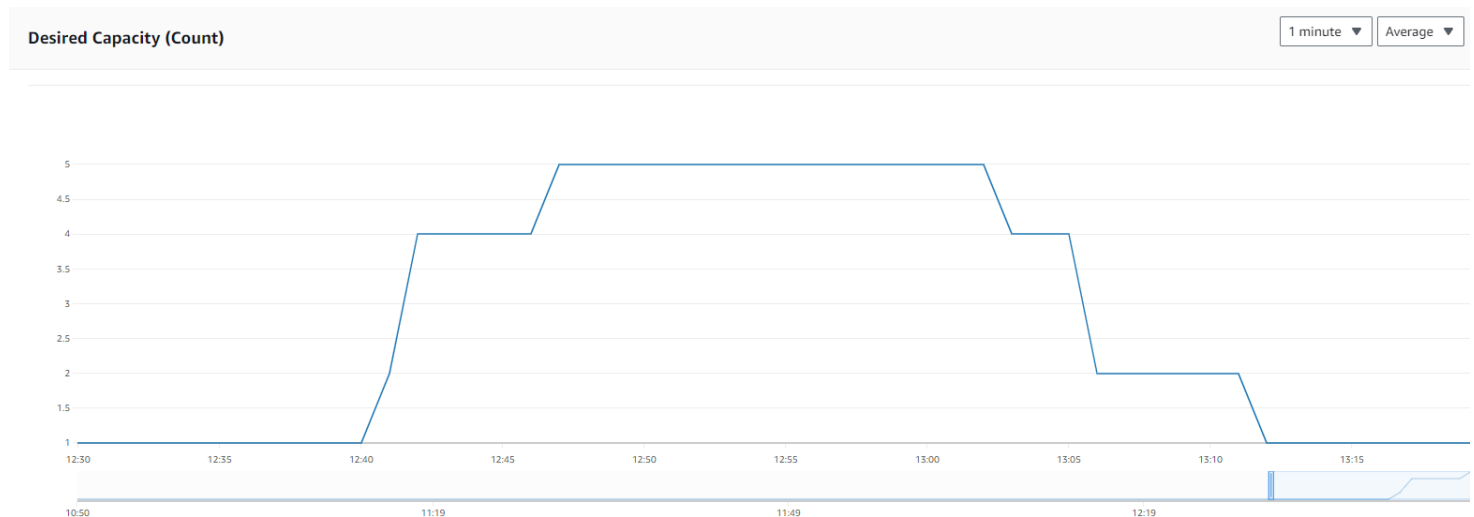


Figure 3: Desired Capacity



Figure 2: CPU Utilization

Test plan

| # | Task | Expected Outcome | Result | Appendix C Figure # |
|----|---|--|--------|----------------------------|
| 1 | GIF Converted | .GIF file appearing in S3 | PASS | 3 |
| 2 | GIF displayed for user | A GIF displaying on the website | PASS | 1 |
| 3 | Large video converting into a small sized GIF | A Gif smaller then the original video displayed on the website | PASS | 5 |
| 4 | Worker instances scaled out | Multiple instances displayed on the EC2 Autoscaling group monitoring tab | PASS | Figure 2 (Not Appendix) |
| 5 | Long video converted into a 5 second GIF | A Gif smaller duration then the original video displayed on the website | PASS | N/A |
| 6 | Upload multiple videos at once | two successfully converted | PASS | 3 |
| 7 | Convert GIF without inputting parameters | GIF displayed with videos default parameters | PASS | 1 |
| 8 | Convert GIF with duration longer than video | GIF displayed with videos duration | PASS | N/A |
| 9 | Create a lot of loads by uploading several large videos | application scales up to deal with the increase in load | PASS | Figure 2 (Not Appendix) |
| 10 | Accessing GIF through unique ID | GIF displayed to user | PASS | 6 |
| 11 | Converting video into a large GIF | Large GIF made | PASS | 1 |
| 12 | Adjust the framerate of the GIF | Visible framerate change | PASS | N/A |
| 13 | SQS message sent | SQS message can be polled | PASS | 7 |

Figure 4: Test Cases

Difficulties / Exclusions / Unresolved & Persistent Errors

Problems with SQS being processed multiple times when the GIF took too long to convert. This was fixed by adding a feature that when a message was processed, a timeout on the SQS messages was applied so multiple wouldn't be processed.

Auto Scaling Groups not using more than 10% of the CPU, this meant that the group wouldn't scale out and create more instances. However when a single instance was run by itself it would easily reach over 90%. The problem was found to be that the auto scaling group was using t2 Micro which would run out of CPU credit balance, changing over to T3 Micro fixed this issue.

Issues with FFmpeg using too much and too little CPU. Using complex filters were attempted to create a GIF with perfect quality, but this was too heavy and would often crash. Depending on the video quality FFmpeg would not produce enough load to force a scale out. A good medium had to be found regarding this with the FFmpeg options.

Some persistent errors include that if too many GIFs are being converted it may bring the instance to a complete stall. This could easily be fixed by setting a limit on how many GIFs can be processed at once but implementing this made scaling out quite difficult. Another error is if the video is never converted there is not much response from the client side to indicate this.

Extensions

Add in features to edit the videos further. This is supported by Ffmpeg and was the original plan to give the users the ability to change the file to any format, as well as edit it in many more different areas. This, however, would not be possible in the time frame with the problems that occurred. Add user accounts for more privacy when uploading GIFs and a page for that specific user to see their converted GIFs. This could be done with an SQL database such as Amazon RDS. Options for higher quality GIFs, this would require more computational power.

User guide

The application is used by selecting a video file using the “Choose File Button”. Then optional parameters can be inputted and clicking the “Convert To GIF” button will begin the GIF conversion process, then when the conversion process is finished the user will be redirected to the `/gif/:uniqueID` endpoint where the GIF will be displayed. These can be seen in *Figure 5* below:

The image shows a web application interface for creating GIFs. At the top, the title "GIF MAKER" is displayed in large, white, serif capital letters against a dark blue background. Below the title, the interface is a lighter blue-grey color. It features a file selection section with a rounded button labeled "Choose File" in purple text and a status indicator "No file chosen" to its right. Below this are four input fields: three text boxes labeled "Enter width", "Enter height", and "Enter duration (seconds)", and a dropdown menu labeled "default fps" with a small downward arrow. At the bottom of the form is a large, rounded purple button with the text "Convert To GIF" in white.

Figure 5: Application

Appendices

Appendix 1: Code Snippets

Figure 1: Parameter Passing

```
const uniqueID = `${Date.now()}`; // Generate a unique video ID
const fileExtension = req.file.originalname.split(".").pop();

// GIF parameters
const parameters = {};
const width = req.body.width;
const height = req.body.height;
const duration = req.body.duration;
const framerate = req.body.framerate;

if (width && height) parameters.size = `${width}x${height}`;
else if (width) parameters.size = `${width}x?`;
else if (height) parameters.size = `?x${height}`;

if (duration) parameters.duration = duration;
if (framerate) parameters.framerate = framerate;
```

Figure 2: FFmpeg Parameter Use To GIF

```
// Wait for the download to finish.
s3ReadStream.on("end", () => {
  console.log("Video downloaded from S3...");
  // Continue with the rest of the processing.
  let ffmpegCommand = ffmpeg(inputFilePath);



  if (parameters.size) ffmpegCommand = ffmpegCommand.size(parameters.size);
  if (parameters.duration)
    ffmpegCommand = ffmpegCommand.setDuration(parameters.duration);
  if (parameters.framerate)
    ffmpegCommand = ffmpegCommand.fps(parameters.framerate);

  console.log(`Converting ${videoID} to GIF...`);
  const outputPath = path.join("./temp", `${videoID}.gif`);
  const s3GIFObjectKey = `${videoID}.gif`;

  ffmpegCommand
    .toFormat("gif")
```

Appendix 2: ASG Group Details

Figure 1: Auto Scaling Group Configuration

| Group details | | |
|--|---|---------------------------|
| Auto Scaling group name n11029935-gifConverter-AutoScaling | Desired capacity 1 | Status - |
| Date created Mon Nov 06 2023 01:33:03 GMT+1000 (Australian Eastern Standard Time) | Minimum capacity 1 | |
| | Maximum capacity 5 | |
| Launch configuration | | |
| Launch configuration n11029935-gif-t3-ssh | AMI ID  ami-093b3e564a4bea74f | Instance type t3.micro |
| Storage (volumes) /dev/sda1 | Security groups  sg-032bd1ff8cf77dbb9 | Key pair name Phillipe |
| View details in the launch configuration console | | |

Appendix 3: Text Cases

Figure 1

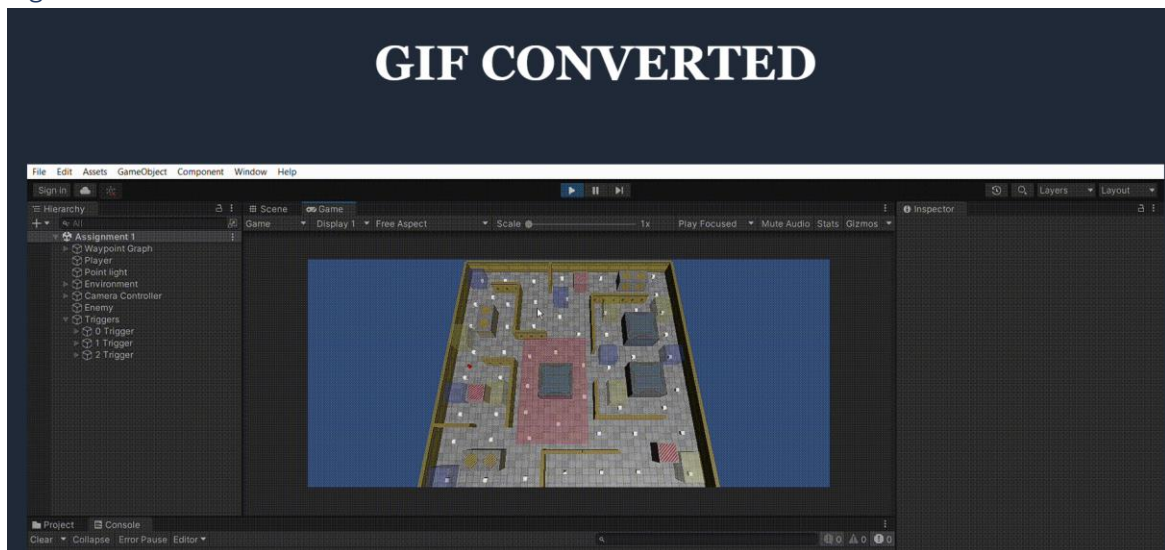


Figure 2

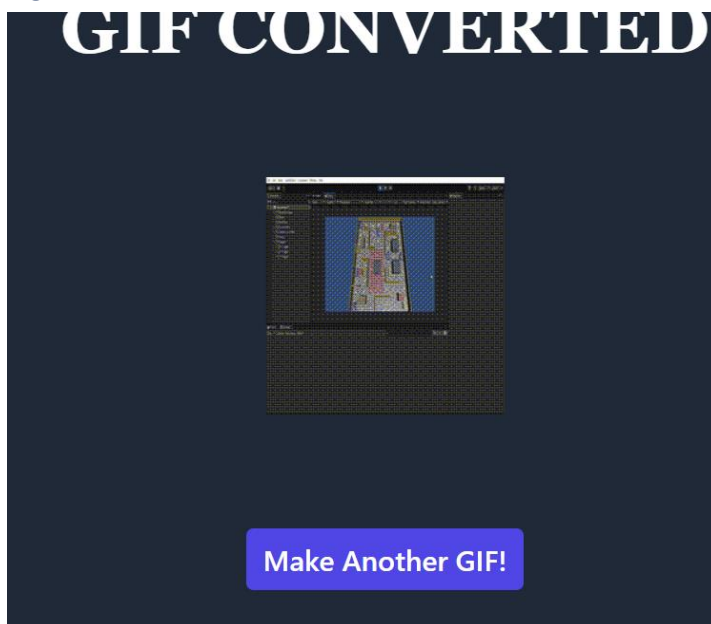


Figure 3

| Name | Type | Last modified |
|-----------------------------------|------|--|
| 1699253834170.gif | gif | November 6, 2023, 17:00:54 (UTC+10:00) |
| 1699253996223.gif | gif | November 6, 2023, 17:00:22 (UTC+10:00) |

Figure 5

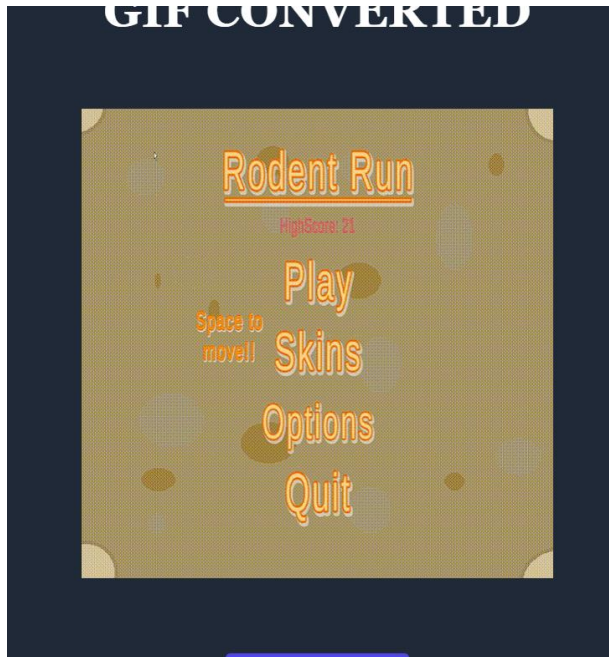


Figure 6

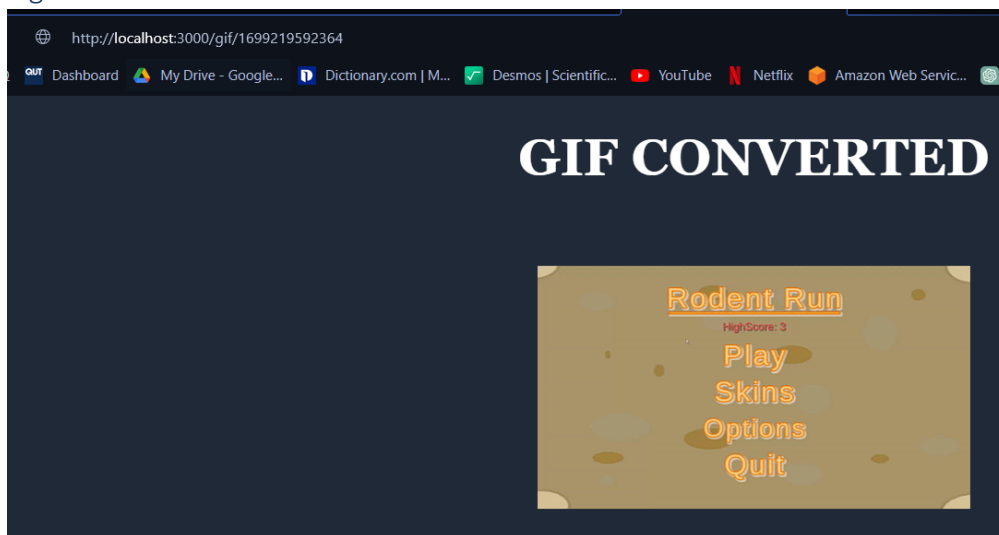


Figure 7

