

Boston University

Electrical & Computer Engineering

EC464 Capstone Senior Design Project

User’s Manual

KartyKlub:

A Mario Kart 64 Analytics Suite

Submitted to

Dan Ryan of ByteLight

183 State St

Boston, MA 02109

by

|  |  |
| --- | --- |
|  | Team #15  The Karty Boyz |

|  |  |
| --- | --- |
| Michael Abed | [mgabed@bu.edu](mailto:mgabed@bu.edu) |
| Jonathan Bell | [jbell@bu.edu](mailto:jbell@bu.edu) |
| Johan Mickos | [jmickos@bu.edu](mailto:jmickos@bu.edu) |
| Josh Navon | [navonj@bu.edu](mailto:navonJ@bu.edu) |
| Mark O[’Brien](mailto:mgabed@bu.edu) | [mark0b@bu.edu](mailto:mark0b@bu.edu) |

Karty Klub

Table of Contents

Executive Summary 3

1. Introduction 4

2. System Overview & Installation 5

2.1 Overview Block Diagram 5

2.2 User Interface 6

2.3 Physical Description 8

2.4 Installation Setup and Support 8

2.4.1 Hardware Setup 8

2.4.2 VLC Setup 9

3. Operation 10

3.1 Normal Operation 10

3.1.1 Uploading a Session Video 10

3.1.2 Querying for Data 10

3.1.3 Advanced Querying for Data 11

3.2 Abnormal Operation 12

3.2.1 Crashing OpenCV & Idle Virtual Machines 12

3.2.2 Undefined File Content 12

3.2.3 XHR Errors 12

3.3 Safety Issues 13

4. Technical Background 14

4.1 Video Processing 14

4.2 Audio Processing 14

4.3 Storage and Querying 15

5. Cost Breakdown 16

6. Appendix 17

6.1 Appendix A – Specification 17

Video & Audio Processing 17

Database, Web Interface, & Cloud Technologies 17

Website 18

6.2 Appendix B – Team Information 18

Michael Abed 18

Jonathan Bell 18

Johan Mickos 18

Josh Navon 18

Mark O’Brien 19

# Executive Summary

The customer, a group of avid Mario Kart 64 enthusiasts, has no way to track their progress as racers. They wish to have a system from which they can gain insights into how they play, or review previous race tapes. The proposed solution is twofold: a video and audio processing suite to detect the events of their races, and a website where they can view their races and ask questions about how they’re racing. To use this solution the users will upload videos of their race sessions to the website which will run the processing suite on their video. When that has finished they will be able to review races or gain insight into what happened through a custom designed query language. This will all be made possible by leveraging cloud computing on the Amazon Web Services (AWS) platform. As a result, this system will elastically scale to however much Mario Kart they are able to play.

# 1. Introduction

Dan Ryan, the customer, is an avid Mario Kart aficionado. A few days a week he plays Mario Kart with friends. He wishes to have a way to track his statistics from played races, both in order to understand the underlying mechanics better, and to keep a leaderboard for him and his friends. Mr. Ryan has tasked us with creating a queriable database that stores race events. An event in Mario Kart is an action from a predefined list of possible actions. Every action has an associated sprite that is displayed each time the action is performed. An example is a character picking up an item. The sprite associated with this action is the item box appearing at the top of the specific character’s window.

Utilizing a web application hosted on AWS elastic beanstalk, the user will be able to upload a Mario Kart 64 session video, including multiple races, to process. Once the session video has been uploaded, jobs are created to split the video into individual races. Once split, more jobs are created so that processing may begin. Processing is done on both race video and audio. While the video processing engine detects key actions in gameplay, the audio processing engine will detect exciting moments in gameplay. The first exciting moment in audio is when the sound level goes above the average noise level in the room by a certain amount. The second is a detector for the phrase “tag it,” which utilizes an utterance detector in addition to the Google Speech API. Once the events have been processed, they are sent to a database. The database stores the events, and is accessible through the web application through our querying api. The user can also request individual race videos he has uploaded to watch at a later date.

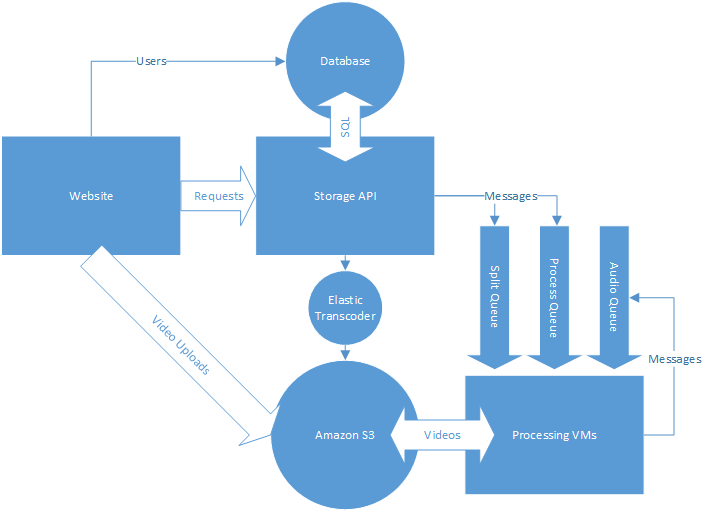
It is also worth noting one limitation regarding video codecs. We have experienced an issue with AVI files recorded from VLC Player. The root cause of the issue hasn’t been tracked down, but the analytics suite will occasionally experience a segmentation fault. For this reason, it is advised that users not upload AVI files recorded using VLC player. A workaround we present for the user is to transcode the video to another common video format. This can be done using a variety of tools, one of which being FFMpeg.

It is also advised that users speak loudly when saying the “tag it” and “watch this” commands. This is less a shortcoming of our system and more of aspect related to speech recognition. Typically, speech recognition software requires the user to speak directly into the microphone being used. Thus, we recommend that users account for the increased distance between them and the microphone.

In the following sections, the setup and operation of the Mario Kart 64 analytics suite will be described in detail. Then, there will be a thorough description of each component of the suite on a technical level. Finally, we outline the cost of running and maintaining the suite.

# 2. System Overview & Installation

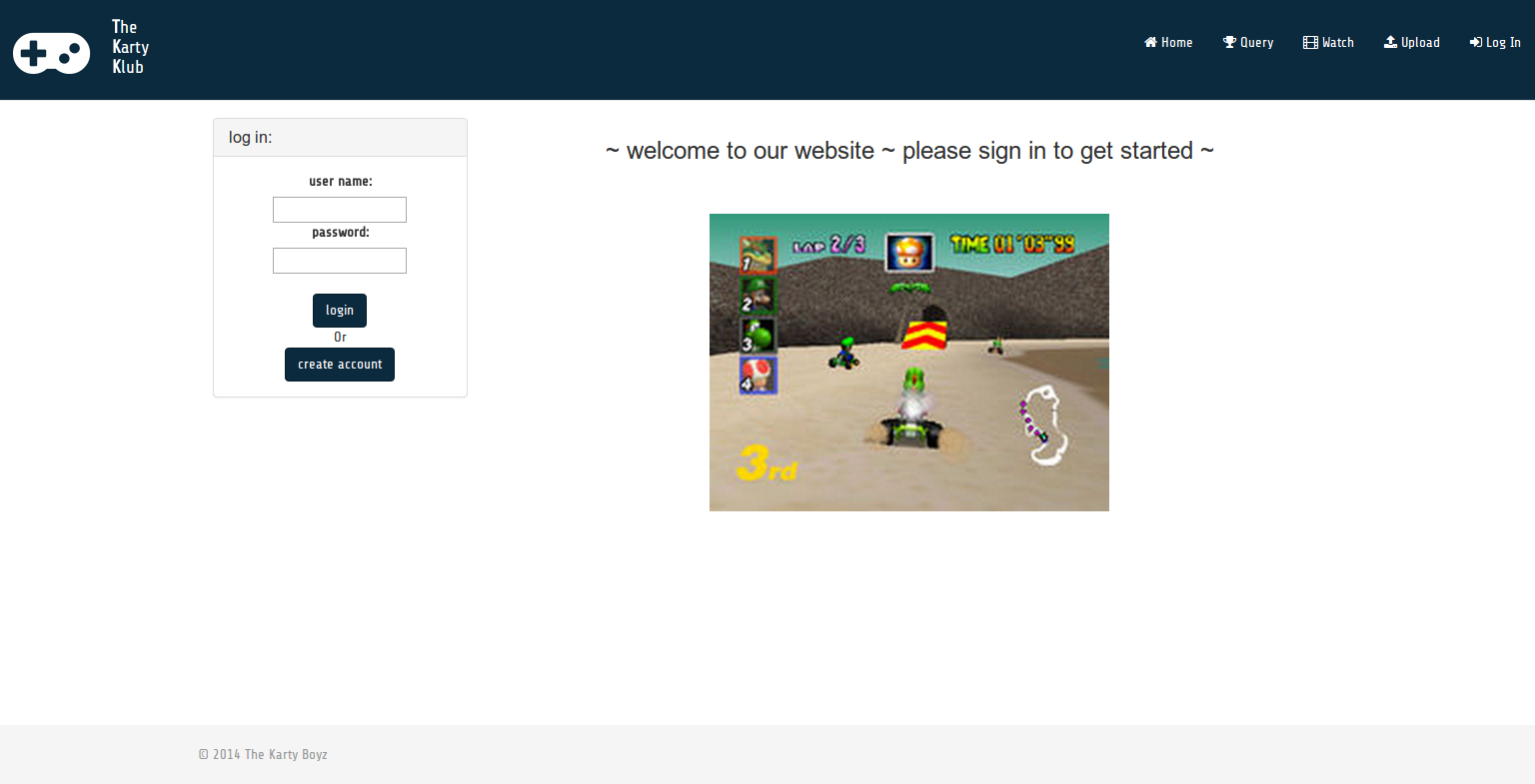
## 2.1 Overview Block Diagram



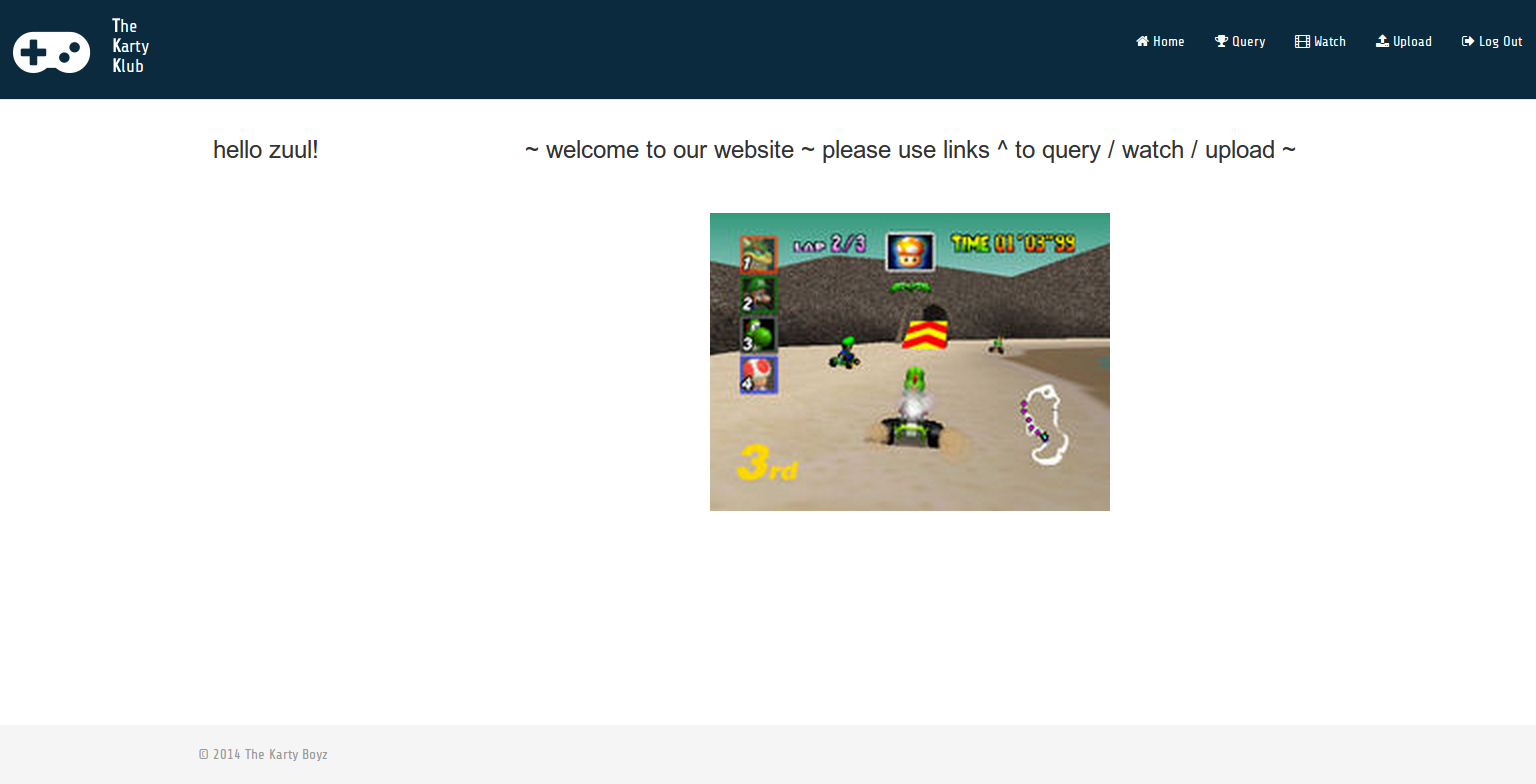
*Figure 2.1: Representation of component interaction*

The system gets inputs from the user at the website. Videos get uploaded directly to Amazon S3 while other requests get routed to the Storage API. The Storage API passes messages to the processing VMs via Amazon SQS queues. The processing VMs get videos from S3 and process them for events, updating the Storage API with the races and events detected. When processing is done, the user can submit queries to the website.

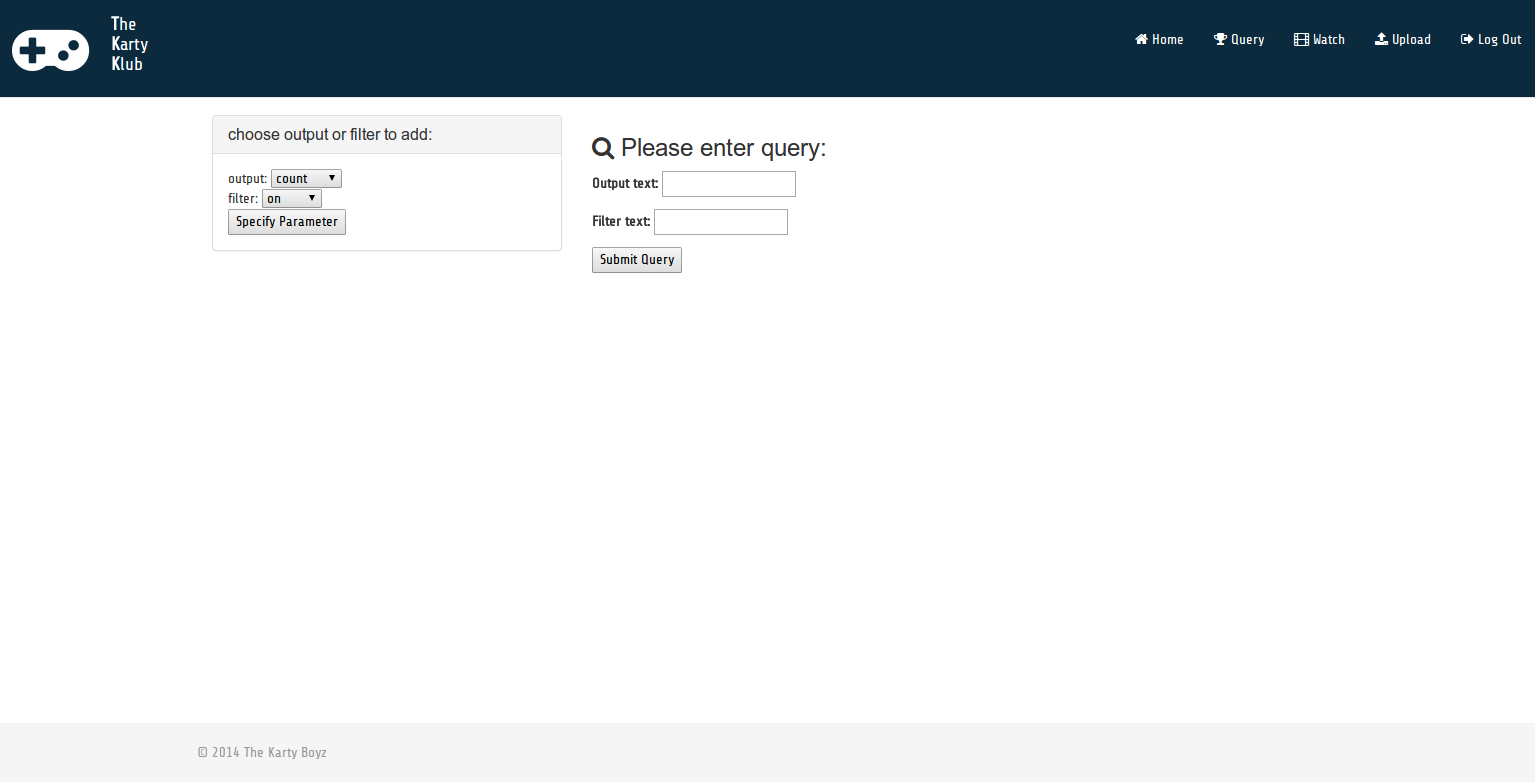
## 2.2 User Interface



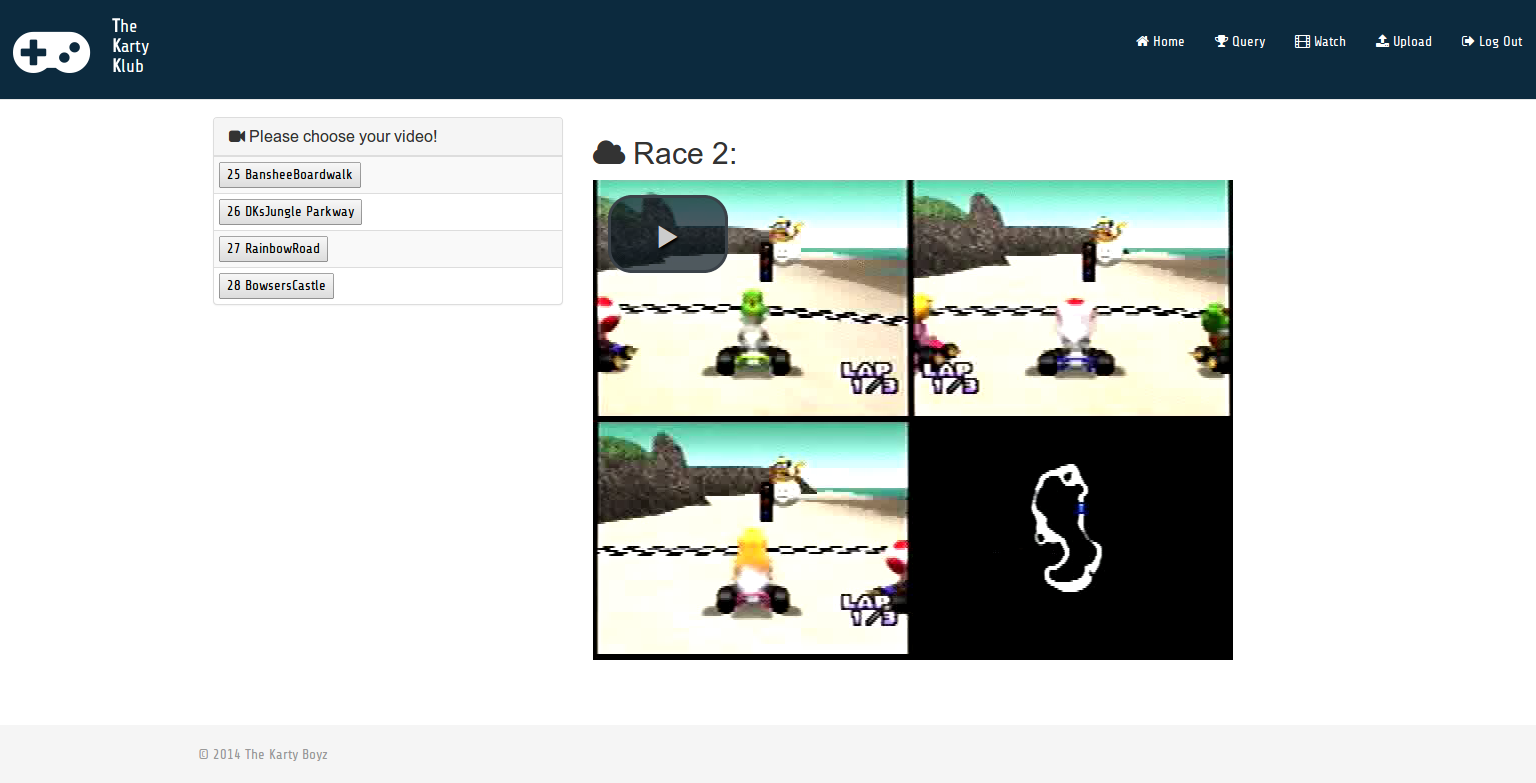
*Figure 2.2: Homepage when not logged in. Upon arriving at the website prior to logging in, the user will be greeted by our home page, prompting the user to sign in. Note that the page links at the top of the homepage will be unavailable until user sign-in.*



*Figure 2.3: Homepage after login. Upon signing into the website, the user will be taken to the modified home page. The homepage displayed after sign-in occurs greets the user and prompts him or her to navigate to different pages on the website.*



*Figure 2.4: Query page. This is where the user will query race statistics. Using the custom query language, KQL, the user can find out relevant race statistics, and improve his or her gameplay strategy.*

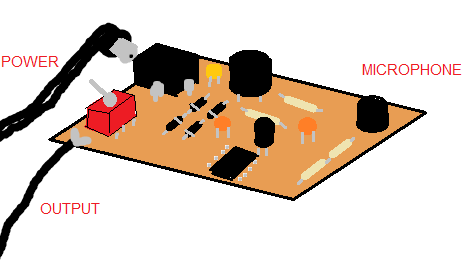


*Figure 2.6: Watch page. the user can reference past races here and watch them. After races are finished processing, the user will be able to view each race video in an embedded video player.*



*Figure 2.6: Upload page. The user will upload session video on this page. Upon choosing a video file to upload, the user must click on the “Start Job” button to start processing on the session video.*

## 2.3 Physical Description



*Figure 2.7: Physical depiction of the microphone module. This is what the PCB microphone module will look like.*

## 2.4 Installation Setup and Support

In order to use KartyKlub, the user will need the following:

* Nintendo 64 Game Console
* Mario Kart 64 Game Cartridge
* 2 RCA cable splitters, 1 female end split to 2 male ends
* KartyKlub Microphone Module
* 9VDC Power supply with 5.5mm tip diameter and 2.1mm inner hole diameter.
* USB Composite Video capture card
* Computer with internet connection and VLC installed

### 2.4.1 Hardware Setup

Once these things are all present, the user connects the yellow video output and one audio output (red or white makes no difference) to the two RCA splitters. For each of these splitters connect one output to the television and one to the USB Composite video capture card. For the non-split channel of audio connect it only to the television. The unconnected audio channel of the capture card is then connected to the output of the microphone module. Next plug in the 9V power supply and plug in the microphone module, making sure switch is in the off position. Now that the microphone is plugged in to both the capture card and power the switch may be turned on. Finally connect the capture card to your computer install it’s corresponding device drivers and open VLC.

### 2.4.2 VLC Setup

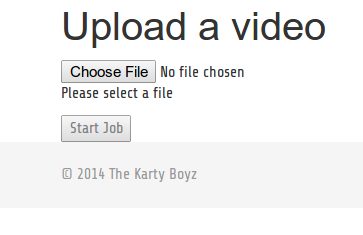
To set up VLC to record your gameplay, click on the “Media” drop-down menu and select “Open Capture Device”. In the menu that opens make sure that “DirectShow” is the selected Capture Mode and select your video and audio devices which will probably be called “USB Grabber” and “USB Audio Interface” but may vary depending on your specific capture device and operating system. Click the play button to begin viewing the video coming from your device. (**WARNING**: If computer is too close to microphone and VLC’s sound is turned on, feedback will occur.) To start recording a video of this capture stream click the button with the red circle in the menu below the video. You are now ready to begin recording a session of Mario Kart 64 races to be processed with the KartyKlub Analytics Suite. Record as many races as you want in the same video. Finally go to the KartUp website, sign up for a free account and upload the video.

# 3. Operation

## 3.1 Normal Operation

Under normal circumstances, the user will have access to a Nintendo 64, the video game Mario Kart 64, the included microphone module, and a way of capturing the audio and video streams into a computer-readable format. Here is an overview of how the user would normally proceed:

### 3.1.1 Uploading a Session Video

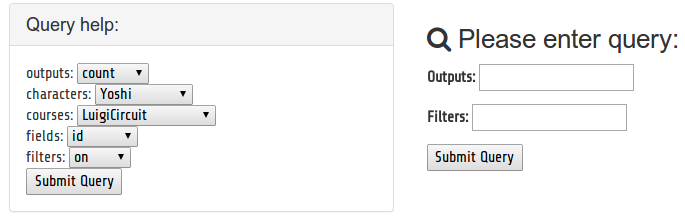
1. The capture device is prepared to record both the race data and the environmental audio
2. A Mario Kart 64 session video is recorded into a standard video format (mp4, mov, etc.)
3. The user visits the KartyKlub website through a web browser
4. In the menu-bar of the website, the “Upload” link is clicked
5. The user chooses the recorded session video under “Choose file”
6. Once uploading has completed, the user will select “Start Job” to begin session processing

*Figure 3.1: Uploading a session video*

### 3.1.2 Querying for Data

If the user does not have access to the necessary hardware or if he simply wishes to view data gathered from previously processed Mario Kart 64 races, he can query for interesting results:

1. The user visits the KartyKlub website in a web browser
2. In the menu-bar of the website, the “Query” link is clicked
3. Based on the sample queries provided on the page, the user can construct a unique query to fetch interesting Mario Kart 64 data



*Figure 3.2: Querying for Mario Kart data*

### 3.1.3 Advanced Querying for Data

Part of the power of this project is the ability to run arbitrary queries against the database with the use of our custom query language KQL (‘kay-quel’). This language allows one with knowledge of what the database looks like to use almost the entire power of SQL in a safe and compact manner.

The core piece of this language is the selector. A selector is just a reference to any type of event that has happened. This could be something like “Item” or “Lap” which would refer to all events that involved items or laps. You could also be more specific though. “Get.GreenShell” would refer to all of the events where somebody got a green shell. Likewise “New.3” would refer to all of the new laps that were lap 3. The full list of selectors and sub-selectors can be found on the website in the drop down menu.

Selectors can be used in both outputs and filters. Outputs as you might guess control what gets output in the query results, and filters control what data can be output but do not themselves put things in the output.

Outputs and filters are called by giving the name of the output/filter and then providing it arguments. The default filter is “where” and the default filter, and “out” is the default output. The outputs take an additional field argument which corresponds to what information about the event you want to know. Some useful filters are “on” and “with”.

Some simple queries

* out Item course -- this will output the name of the course on each item event
* out Item.GreenShell place -- this will output the place where each green shell was received
* out Time: on KoopaTroopaBeach -- will output the lap times on KoopaTroopaBeach
* Time: with Toad, on KoopaTroopaBeach -- toads lap times on Koopa Beach

As the last query shows, multiple filters can be applied by listing them with commas. This is also true of outputs. Keep in mind that the same selector in different portions of the query will always refer to the same set of events. If you use a subselector you must use the selector/subselector combo every time you refer to it or you will wind up referring to different things.

Another useful filter to be aware of is the “by” filter. This lets you group your query by a certain field on a selector. Some more advanced queries using this mechanism. A more complicated query that uses these mechanisms is:

Item place, count Item, Item: by Item, by Item place

This will filter for all Item events, and aggregate them first by what the item was, and then by the place the player was in when they got the item. The outputs will output the place, the number of items of this type in this place, and the item name. In other words, a breakdown of item frequencies by place. This query could allow us to look at the rubber banding mechanic, where people further back get better items than those in first.

Some filters and outputs take additional arguments. These are the top/bottom outputs and the lap/place/more than/less than filters. These all take an additional number argument.

* top/bottom <n> Selector field-- sort by a selectors field and only output n of them
* lap/place <n> Selector -- restrict this selector to only the ones on a certain lap or place
* more than/less than <n> Selector -- count the number here and only output if that count is more/less than n.

These let us improve on some of the features provided by Mario Kart 64

* top 10 Time: on KoopaTroopaBeach
* top 10 Time: lap 3 Time, with Toad, on KoopaTroopaBeach
* top 10 Finish timestamp: on KoopaTroopaBeach

Here we can get lap times on specific courses and limit them to only specific characters. And we can get more than the single lap time and 5 finish times that the game records.

## 3.2 Abnormal Operation

### 3.2.1 Crashing OpenCV & Idle Virtual Machines

There are a few cases where the above instructions fail to generate the expected outcome. Specifically, OpenCV is known to crash on certain .avi files generated by VLC’s device capture functionality. As the project currently stands, this can only be detected and amended by users with access to the EC2 component of the AWS Console. This is due to the fact that upon a crash, the script running the timeout loop also crashes, preventing the running virtual machine from terminating itself. To detect and amend this:

1. The user must SSH into the VM and run htop (or top) in order to monitor running processes and CPU usage
2. If no instance of the dispatcher.py --daemon script is running, it is an indicator that the program has crashed.
3. The user can now, through the AWS Console, terminate the specific virtual machine with the IP used in step 1

Conversely, the user could decide to write a cron job which would automate the process of self-termination upon crashing.

### 3.2.2 Undefined File Content

Currently, the upload script on the website does no filetype checking, meaning the user can upload any type of file. As no error-checking is done, this will get pipelined all the way to OpenCV, which will attempt to read the file as a video and crash. To fix this, please read section 3.2.1 regarding OpenCV crashing.

### 3.2.3 XHR Errors

Due to unknown reasons, the file upload page sometimes returns XHR errors while attempting to upload the given file. Fortunately, they only occur a fraction of the time. When they do, the user simply needs to re-upload the file. If the XHR errors are persistent for a given file, the user will have to convert the file into another format.

## 3.3 Safety Issues

From a cybersecurity perspective, the project currently contains a few vulnerabilities. As mentioned in section 3.2.2, the user can upload any file type to our product. A malicious user could technically upload malicious code which would get pipelined through all layers of abstraction. It is possible that this could result in unexpected and unwanted behavior.

Similarly, the database does not attempt to use any sort of authentication, allowing malicious users to perform a number of attacks on the database, ranging from DOS/DDOS to injecting invalid or corrupt data.

# 4. Technical Background

## 4.1 Video Processing

The core of the video processing back-end can be boiled down to two important concepts: image recognition and parallel processing.

Image recognition is a challenging but important problem in the field of computer vision. It deals with analyzing an image’s shapes, colors, edges, and other features in order to determine its similarity to a known. The problem arises when one wishes to recognize items in an image that is scaled, skewed, or otherwise distorted from the known object. Various algorithms have been implemented to bypass this, from pixel-averaging to machine learning, each with its own additional computational overhead. The solution implemented in our product is known as a normalised sum of squared differences (NSSD).

The second core concept used in the video processing is that of parallel processing. Parallel processing makes use of the fact that separate cores can execute in parallel, thus reducing the time it takes to complete a task. Because video processing is a resource-heavy task, it is split up in order to maximize work being done and reduce time needed to process a video.

## 4.2 Audio Processing

The audio processing that is done on the captured external gameplay audio consists of two parts. First there is a general detection of exciting moments and then a detection of certain keyphrases.

For both of these goals in audio detection, it first must be known where the speech band is at a higher than average level for a specific race, to determine when sections of speech and silence occur during the race. By bandpass filtering audio to the bandwidth in which speech primarily exists and then calculating the Teager Energy of 30 ms windows of this audio, each with 20 ms overlapping with the last and next, a measurement of speech energy as a function of time is obtained.

Using this speech energy measurement, exciting moments are rather simply detected using a relatively high threshold on this energy. This threshold locates moments where energy is above and below a certain level. If any of these moments of high energy extends over a certain number of windows, an exciting moment will be detected.

In order to detect keyphrases being spoken, google’s un-official speech recognition module is accessed through url requests, however this tool can only handle up to 3 seconds of audio at a time in flac format. To separate the audio into sections that google can process, the same speech energy measurement is used with a lower threshold value in order locate all moments where speech may be occurring. These moments are then grouped into buffers which are then converted to a flac file and a request to google is made. The responses from these requests contain the hypothesis and confidence of the hypothesis. Finally only those responses which contain keyphrases with confidence greater than a certain amount are chosen as detected keyphrases.

## 4.3 Storage and Querying

The storage consists primarily of a REST API backed up by a PostgreSQL database. The API responds to normal HTTP requests with JSON data associated with them and responds with JSON. This includes a number of simple operations: creating/modifying/listing races, sessions, and events. In addition, it provides a number of additional API endpoints that allow for the website to get different aggregations of data. For instance, one endpoint gives tags for a race and another gives races for a user. These allow for substantial decoupling between the database and the website.

The other main component of this module is query evaluation. An endpoint exists to accept queries, parses these with the query parser and then walks the parse tree to build up a SQL query. This is implemented using SQLAlchemy’s query objects. One of these gets modified at every step of the query evaluation to create the final query.

# 5. Cost Breakdown

The primary cost of this project is running servers for it on Amazon Web services. There are a few fixed costs per month and a number of costs dependent on usage. The following assumptions are made in this cost calculation.

1. There are 2 t1.micro instances running the website and storage API running at all times
2. Users upload 50GB of video files every month
3. These include 4 hours of race footage
4. Once transcoded the videos take an additional 20GB in reduced redundancy storage
5. These files represent 15 sessions of playing Mario Kart
6. Each session has 4 races in it
7. They stream 10 of their races after uploading, representing 2GB of streaming bandwidth

From these assumptions we can estimate what the monthly price would be

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Quantity** | **Description** | **Cost / Unit** | **Unit** | **Total** |
| 1 | 2 | t1.micro instances | $0.02 / hour | 744 hours | $29.76 |
| 2 | 20 | c3.xlarge spot instances | $0.10 / hour (peak) | 1.2 hours | $2.50 |
| 3 | 1 | S3 Video Storage | $0.03 / GB | 50 GB | $1.50 |
| 4 | 1 | S3 Encoded Video Storage | $0.24 / GB | 25 GB | $0.60 |
| 5 | 1 | Elastic Transcoder | $0.015 / min | 240 min | $3.60 |
| 6 | 1 | Bandwidth | $0.12 / GB | 2 GB | $0.24 |
| 7 | 1 | Misc Request Charges | $0.01 / million | 1 million | $0.01 |
|  |  |  |  | **Total** | **$38.11** |

This usage is for the first month of running this site. Since videos will never be deleted you can expect to see the cost grow by the cost of hosting an additional 75 GB of videos each month. This represents an increase in cost of about $2.10 every month. This cost is relatively modest but could result in a significant expense over time.

Another thing to consider is that the number of spot instances is highly volatile and depends on both user usage and spot request usage. The peak cost is $0.10/hour, but could be as low as $0.01/hour and depending on the market. Usage of these is also dependent on user usage. If users upload lots of videos this will cost more. The current estimate is based on what we’ve seen.

# 6. Appendix

## 6.1 Appendix A – Specification

### Video & Audio Processing

As the project currently stands, the processing/detection suite can deliver:

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Proposed Criteria** | **Actual Performance** |
| Event detection false positives | 5% | 0% |
| Event detection rate | 90% | Estimated 100% (actual may vary) |
| Types of Events Detected | Changing places  Game begin/end  Lap changes  Collisions  Acquiring item  Shortcut  Boost | Changing places  Game begin/end  Lap changes  ~~-Collisions~~  Acquiring item  Shortcut  ~~-Boost~~  +Reverse & falling off map |
| Exciting moments | Tag durations where volume is over normalized threshold | Same as proposed |
| Speech recognition for “Tag it!” and “Watch this!” | False negative rate < 3%  False positive rate < 15% | False negative rate estimated 20%  False positive rate 0% |
| Race processing | Process in less than 2x race duration | Process in ~7x race duration |

### Database, Web Interface, & Cloud Technologies

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Proposed Criteria** | **Actual Performance** |
| Data Querying | Statistics can be queried in an easy (non-SQL) manner | Same as proposed |
| Data Storage | All Mario Kart events can be saved | Same as proposed |
| Scalable Infrastructure | Cloud scales number of VMs based on load | Same as proposed |

### Website

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Proposed Criteria** | **Actual Performance** |
| Video Upload | User can upload their videos | Same as proposed |
| Query Database | User can ask database questions in non-SQL format | Same as proposed |
| Watch Video | User can stream videos on race | Same as proposed |

## 6.2 Appendix B – Team Information

### Michael Abed

Michael grew up playing Mario Kart, starting with the original Mario Kart for the Super Nintendo, and continuing up through the version of Mario Kart Wii. While his favorite Mario Kart is Double Dash, Mario Kart 64 will always have a special place in his heart as he has fond memories of playing Mario Kart 64 with his parents and soundly trouncing them on Moo Moo Farm and Yoshi Valley. Michael will be working at Akamai as a Systems Engineer after he leaves BU.

### Jonathan Bell

Jonathan’s first console was an N64, and Mario Kart 64 was his first competitive game. He repeatedly crushed his brother in the majority of races. This seeded his passion of pwning n00bz in video games, which continues to this day. After graduating from Boston University, Jonathan will be joining the Cloud Computing Initiative at the Hariri institute as a software engineer.

### Johan Mickos

Johan was born and raised in Finland and spent the first 11 years of his life there playing video games on his beloved PlayStation 1. The first piece software he ever wrote was a Pascal script to automate tedious tasks in the MMORPG RuneScape, and he’s loved programming ever since. After graduating at Boston University he plans to complete his military service obligation in Finland and find employment as a software engineer at a Silicon Valley startup.

### Josh Navon

Playing Mario Kart 64 as a child, Josh has always had a deep love for the game. That love quickly turned to complete obsession during his freshman year at Boston University, when he would play every Mario Kart 64 Grand Prix twice a day. After achieving Master rank D on Luigi Raceway, he realized his dream to work on Mario Kart 64 for his senior design project.

After graduating from Boston University, Josh plans to find employment in the field of software engineering. Along with searching for employment, he also plans on learning a bit about self-modifying code, and how to code it on the assembly level.

### Mark O’Brien

Never owning his own copy of Mario Kart 64, Mark found himself a vagabond at a young age, traveling from friend’s house to friend’s house honing his skills. After the Great Kart Massacre of 1999 (broken Nintendo 64), his career as a kart racer was set aside until the fire was rekindled during his senior year at Boston University’s College of Engineering.

Mark’s post-graduation plan’s consist of working full time at a non-engineering related job for the summer while focusing his free time on searching for a suitable engineering job or internship and developing his own digital electric guitar effects.