University of Toronto

Faculty of Engineering

Department of Electrical and Computer Engineering

ECE 1779 A3

Serverless Computing

Group Members

Peiqi Li pq.li@mail.utoronto.ca

Kaiduo Wang kaiduo.wang@mail.utoronto.ca

Tianyi Wang tywing.wang@mail.utoronto.ca

Date Submitted: 2022.4.17

Application Description **3**

How to use **4**

For Users(Web-front-end): 4

Homepage 4

Upload Images 5

Searching Image 6

Deleting Image 7

Display all image name 8

Extracting text from image 9

Finding celebrity 10

Facial analysis 11

For Developers (Manager_app): 12

Homepage 12

Display charts 13

Display Lambda 14

Delete application data 15

Clear memcache 15

Development Documentation **16**

Project Structure 16

Auto Scaler 16

Metrics Sender 16

Web server 16

Memory Cache 17

Manager App 17

Web Server 18

Upload image 18

Delete image 18

Show image 19

Textract image 19

Celebrity Detection in image 19

Facial Analysis of image 19

Manager App 19

Show Charts 19

Delete application data 20

Clear memcache data 20

Memcache 20

Insert Data 20
Search Data 21
Drop Data 21
Clear Memcache 21
Auto Replacement 21
Memcache Configuration 21
Publish Cloudwatch Custom Metrics 21
Lambda Function 21
Metrics Sender 21
Auto Scaler 22

Cost Model 23
Assumption 23
Cost Model 23
Predict 24

Application Description

Our application consists of two parts. The **web front end** part is **user-oriented** and the **manager app** is **developer-oriented**.

For users, our application provides two main services which are **image storage** and **image recognition**. For image storage, users could upload, search, and delete any image they want only with a key. Besides, all keys of photos could also be displayed and users can click each of them to view the picture. As for image recognition, our application could identify the text in the picture, get some celebrity's name through facial recognition, and judge the gender, age, and emotion in the picture.

For maintainers of the web front end, we designed a manager app to easily **screen the status** of the web server and functions deployed on Lambda from charts shown in the manager app. Besides, they could **manually clear photo information in S3, dynamodb and memcache**.

Our applications are deployed by **zappa** on **AWS Lambda**. They are **serverless** applications which enjoy **high availability** and **fault tolerance**.

How to use

For Users(Web-front-end):

Homepage

When users open our application, the homepage is like figure 1 shows.

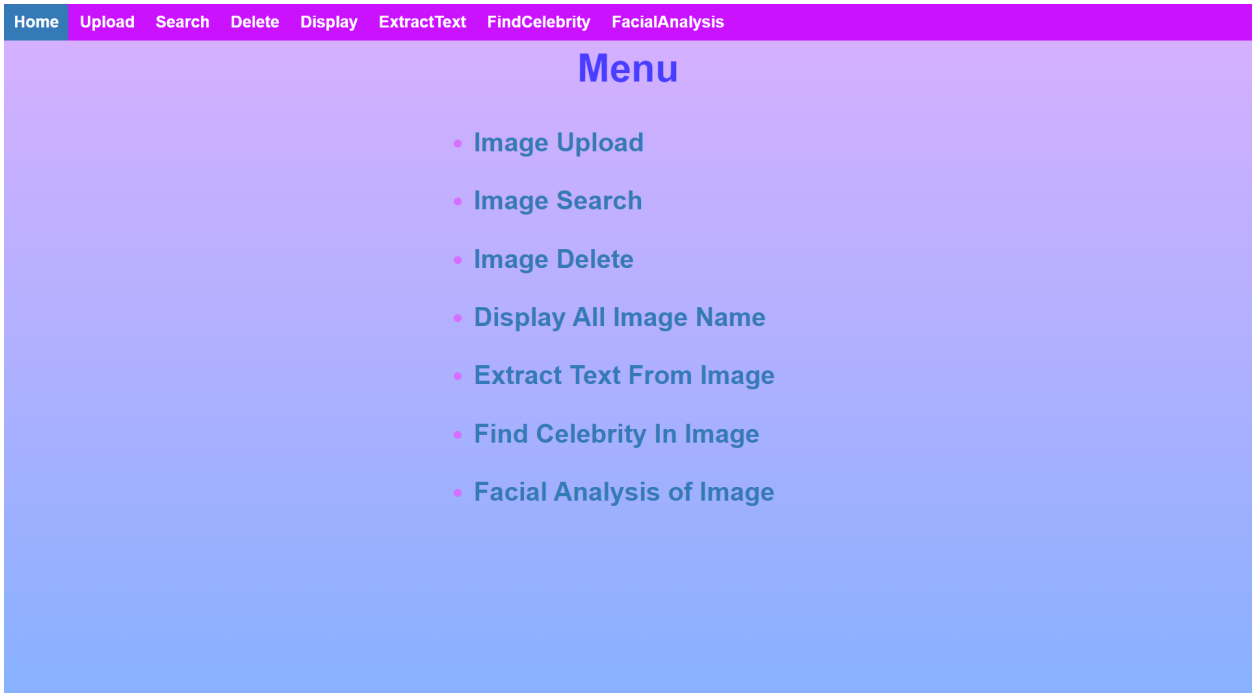

Figure 1. Homepage

Upload Images

For uploading images, please input the key and select an image you want to upload on the uploading page as shown in figure 2. If you upload successfully, there will be successful feedback as shown in figure 3.

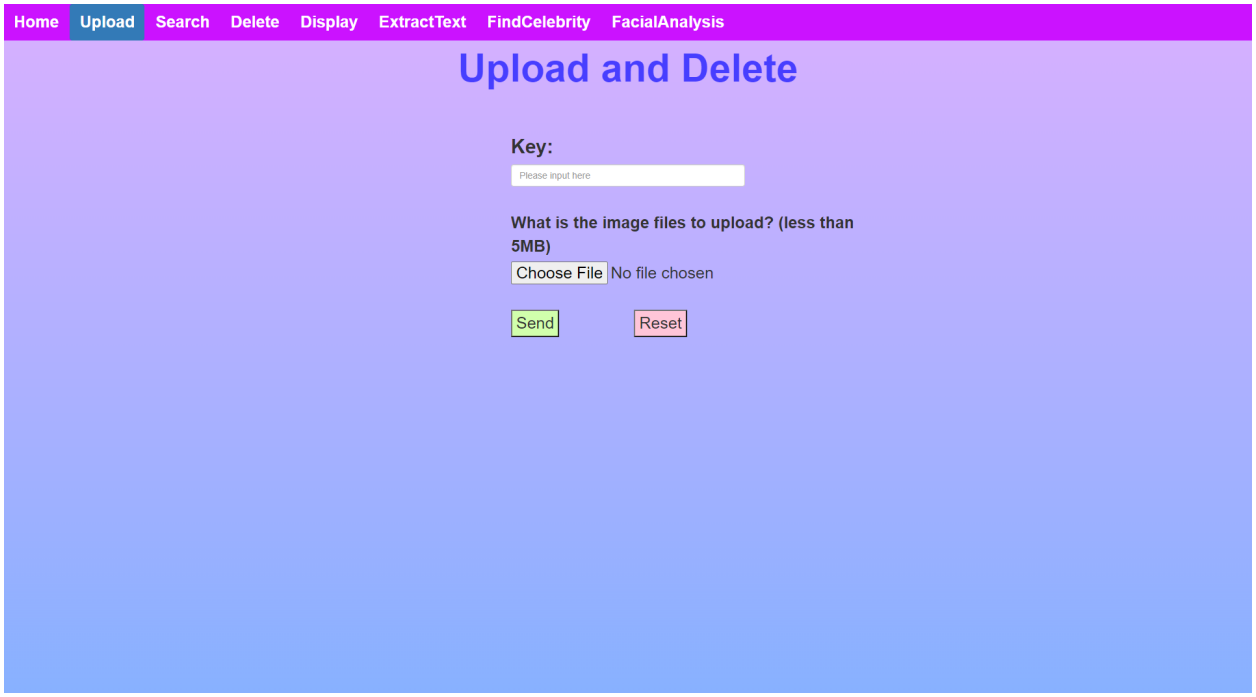

Figure 2. Upload page

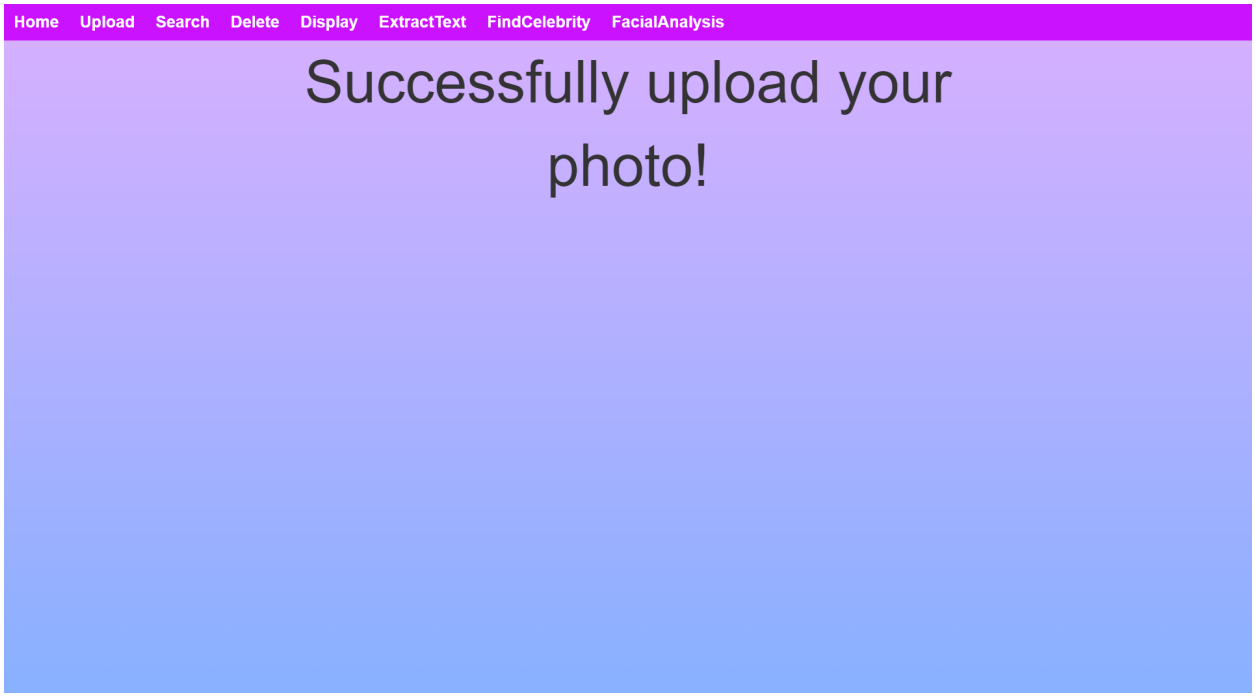

Figure 3. Upload success

Searching Image

For searching an image, please input the key of the image you want to upload on the searching page as shown in figure 4. The result will be shown on a return page as figure 5 shows.

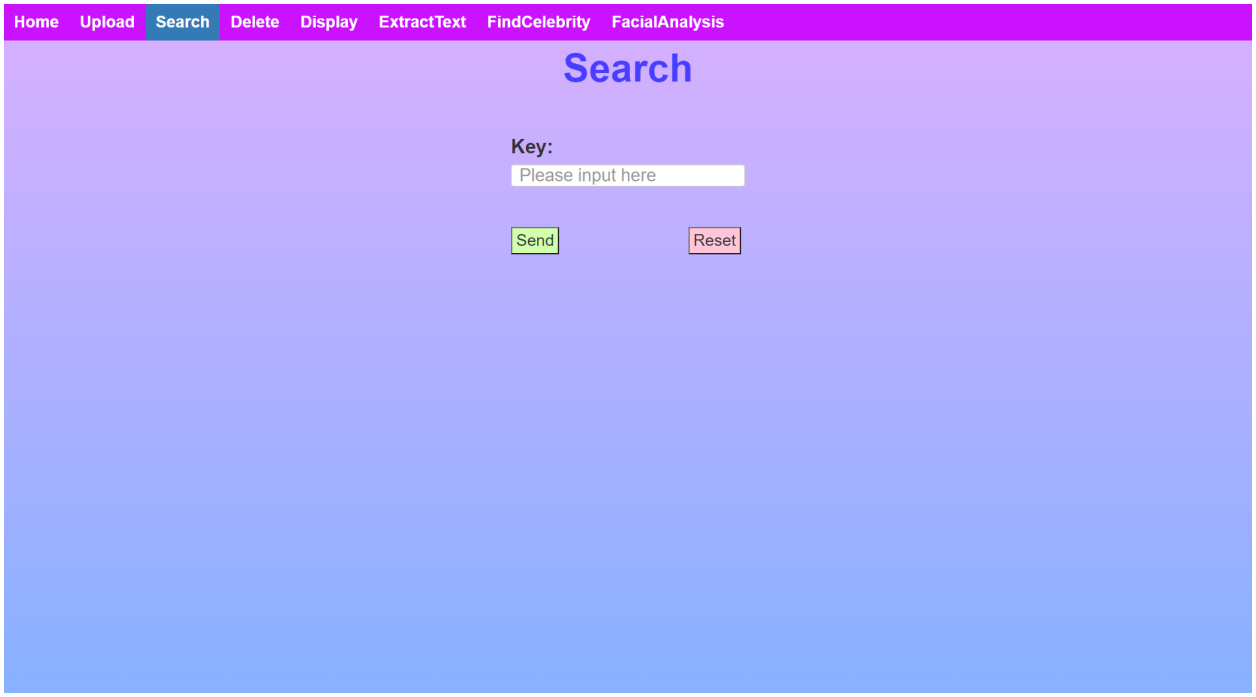

Figure 4. Searching page

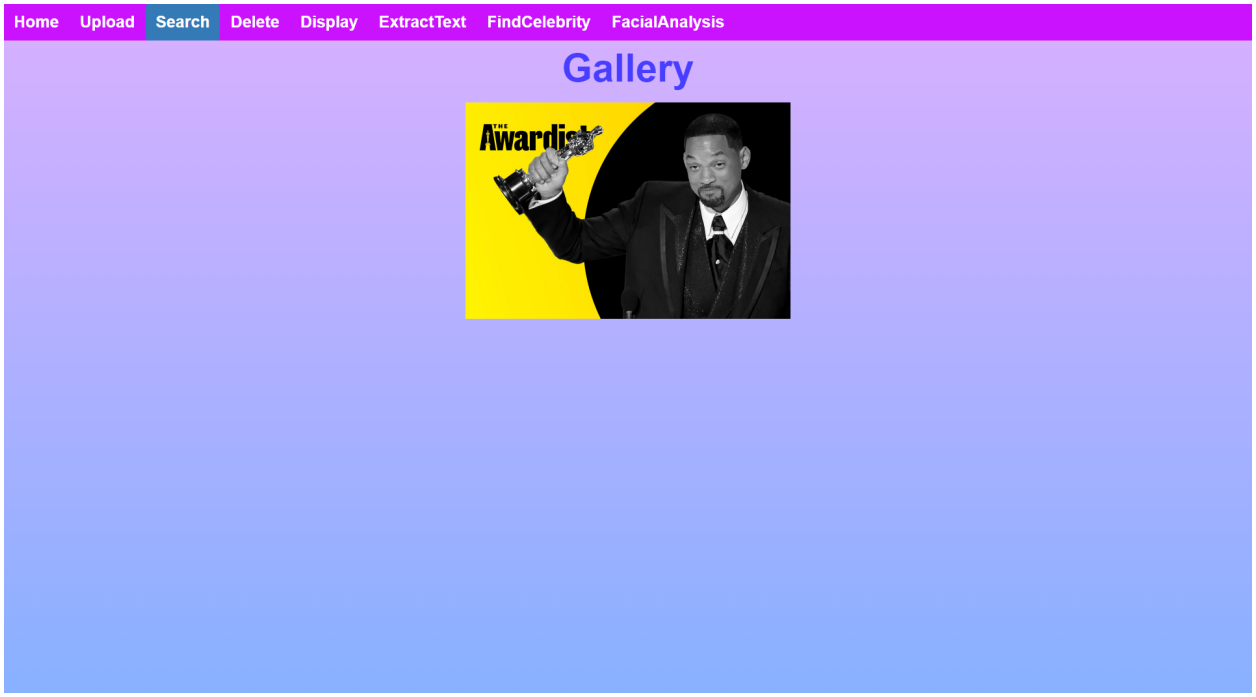

Figure 5. Search success

Deleting Image

For deleting an image, please input the key of the image you want to delete on the deleting page as shown in figure 5. If you delete successfully, there will be successful feedback as shown in figure 6.

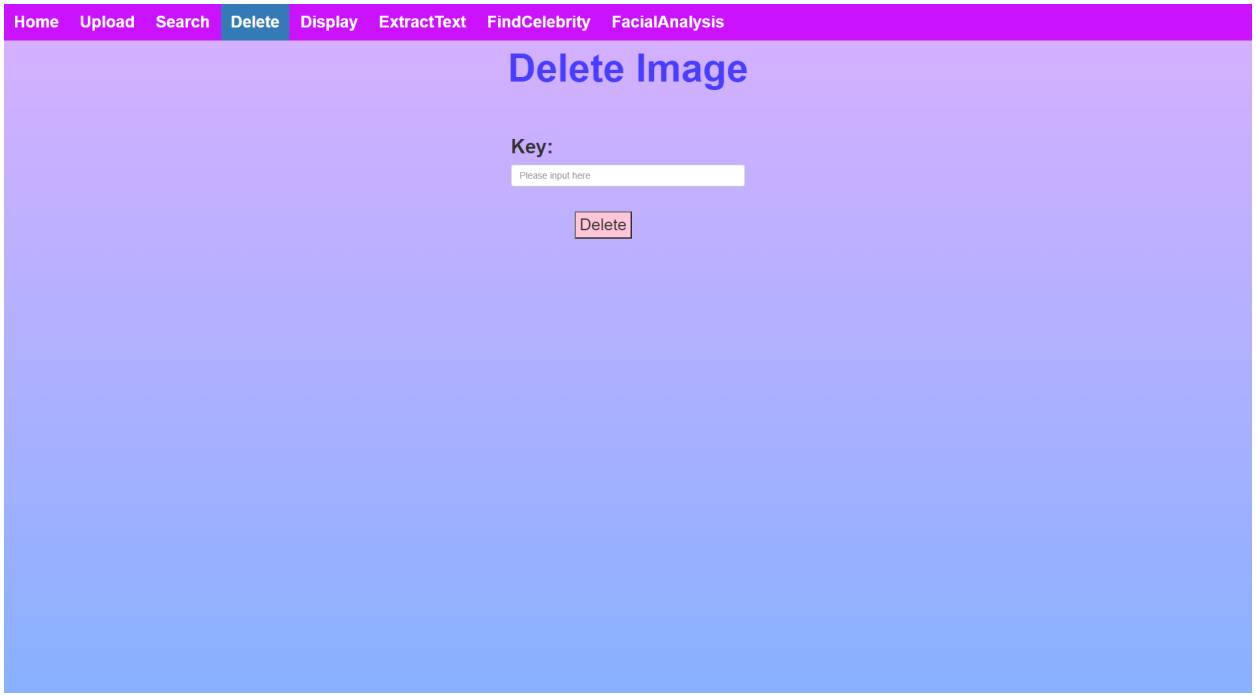

Figure 6. Deleting page

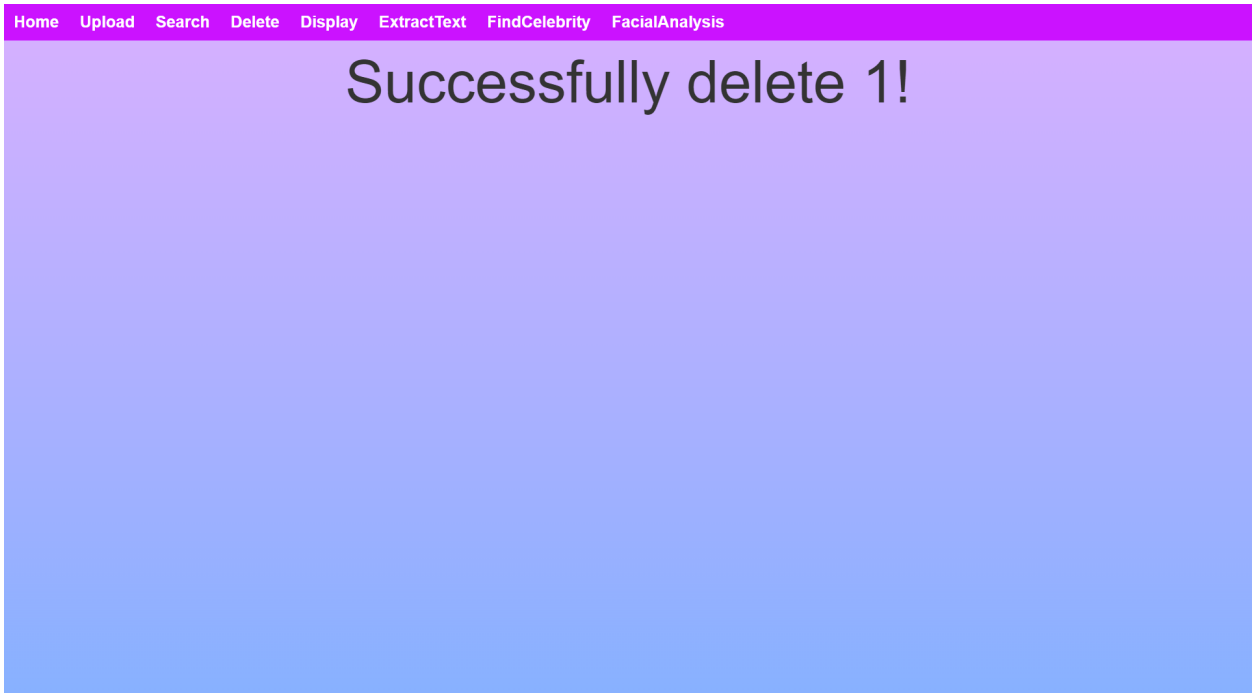

Figure 7. Delete success

Display all image name

For displaying all image names, please click on the corresponding item and the result will show like figure 8 which will show all keys of the image you upload. If you click each item, it will show the corresponding image as figure 9 shows.

Figure 8. Display all image names

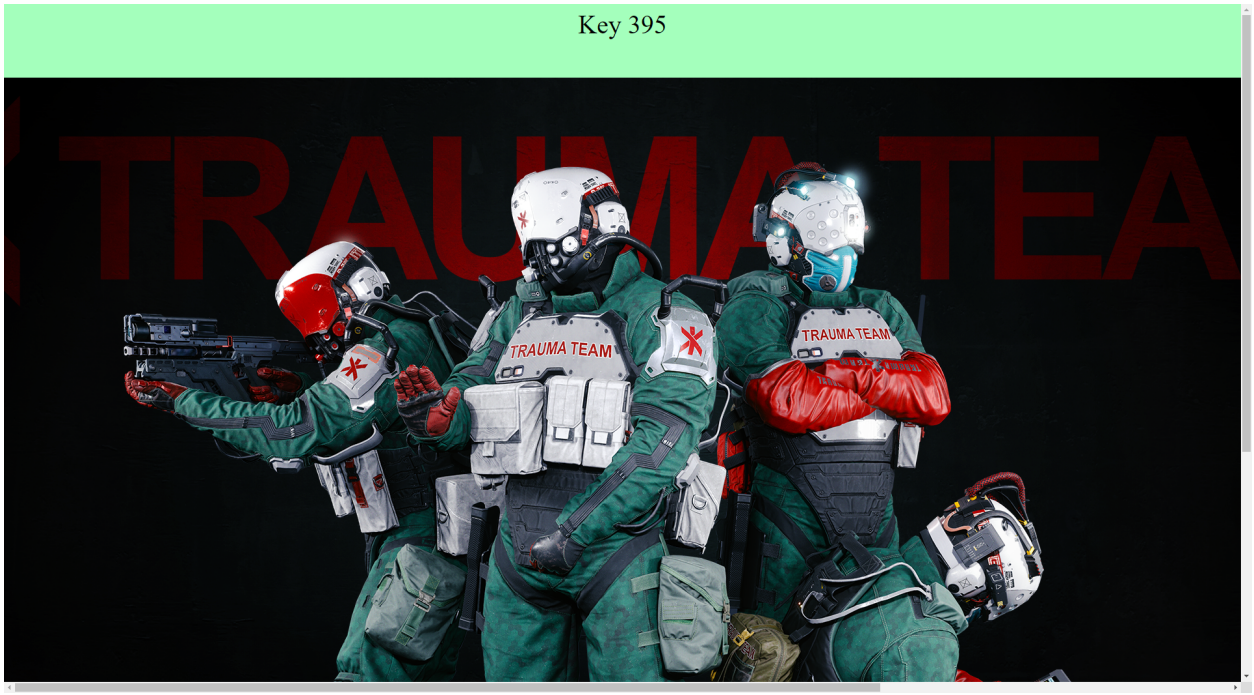

Figure 9. Check each item

Extracting text from image

For extracting text from an image, please input the key of the image you want to extract text on the text-extracting page as shown in figure 10. There will be a result page as shown in figure 11.

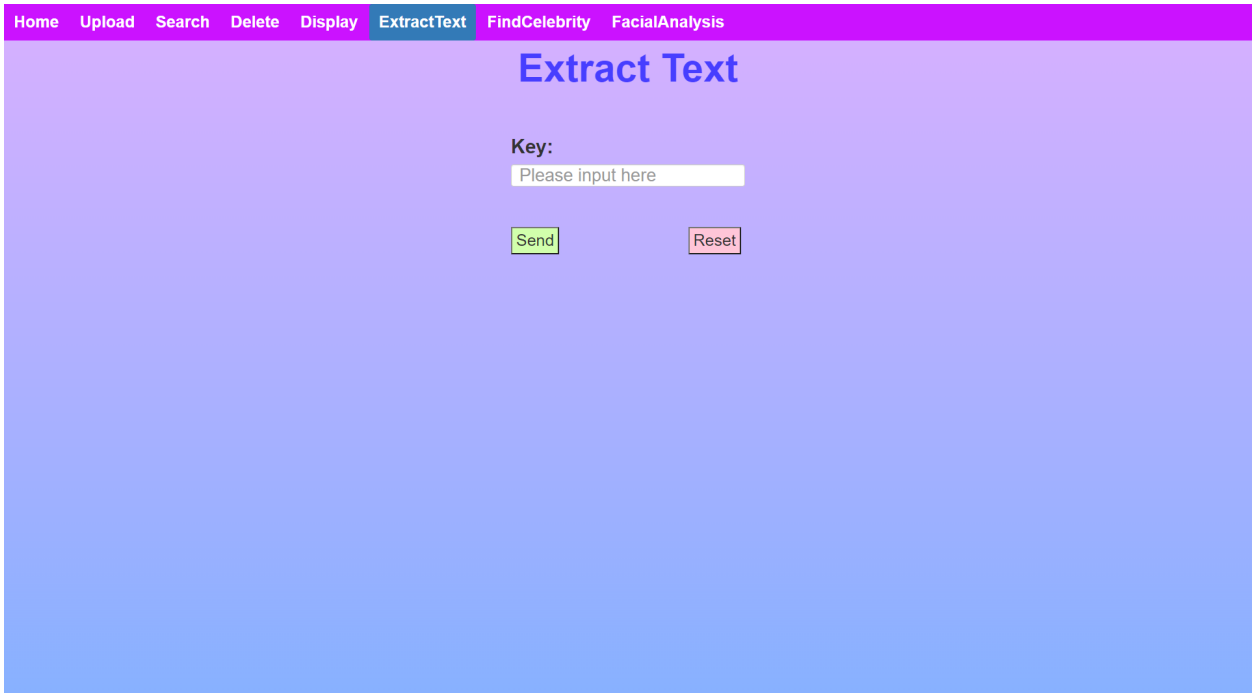

Figure 10. Extract text

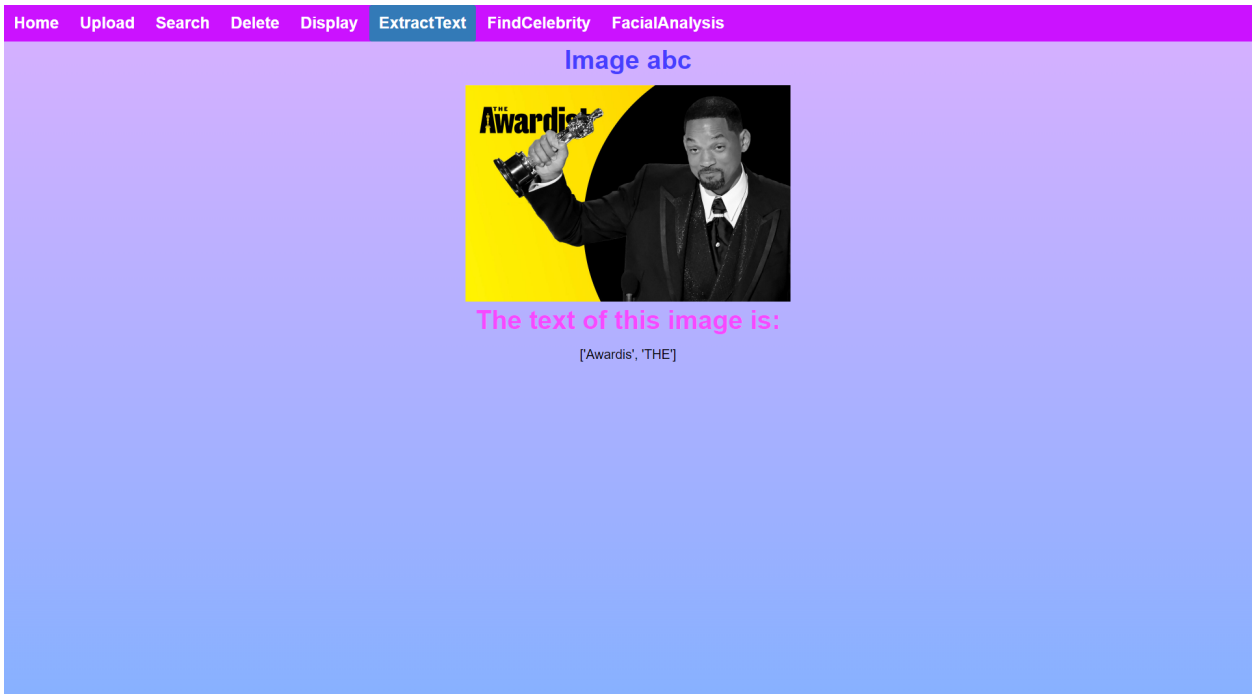

Figure 11. Extract success

Finding celebrity

For finding the celebrity in an image, please input the key of the image on the finding celebrity page as shown in figure 12. There will be a result page as shown in figure 13.

Figure 12. Finding celebrity

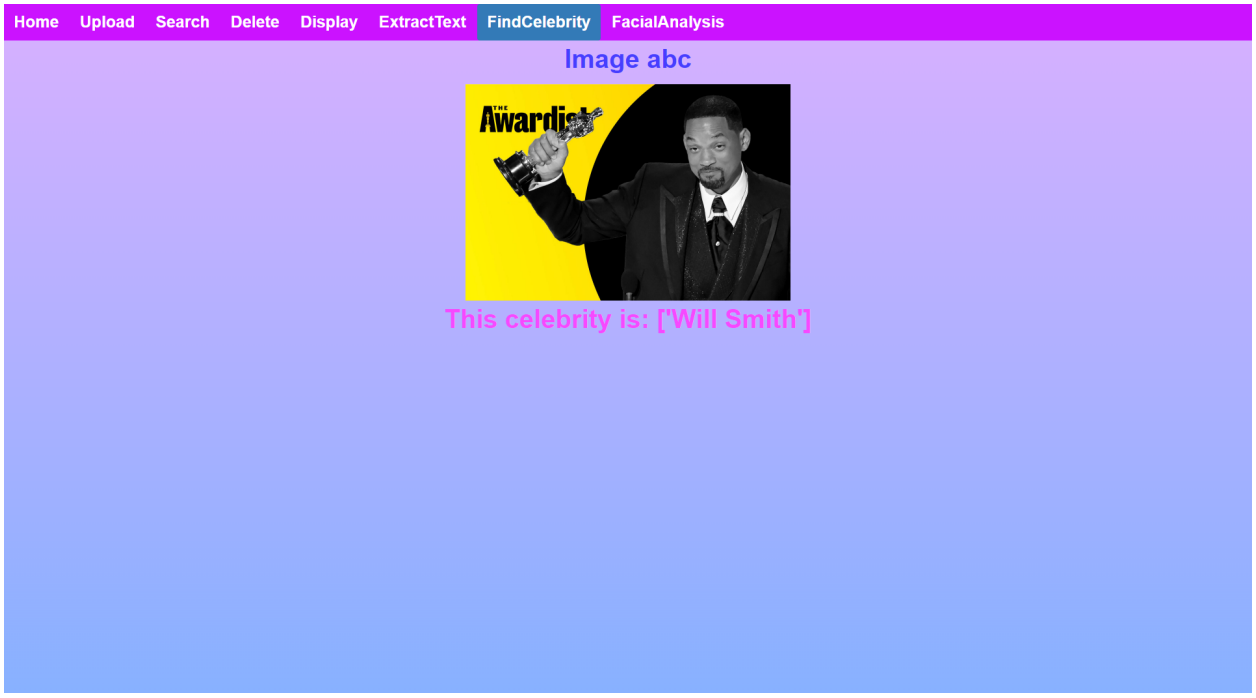

Figure 13. Find success

Facial analysis

For doing the facial analysis of an image, please input the key of the image on the facial analysis page as shown in figure 14. There will be a result page as shown in figure 15.

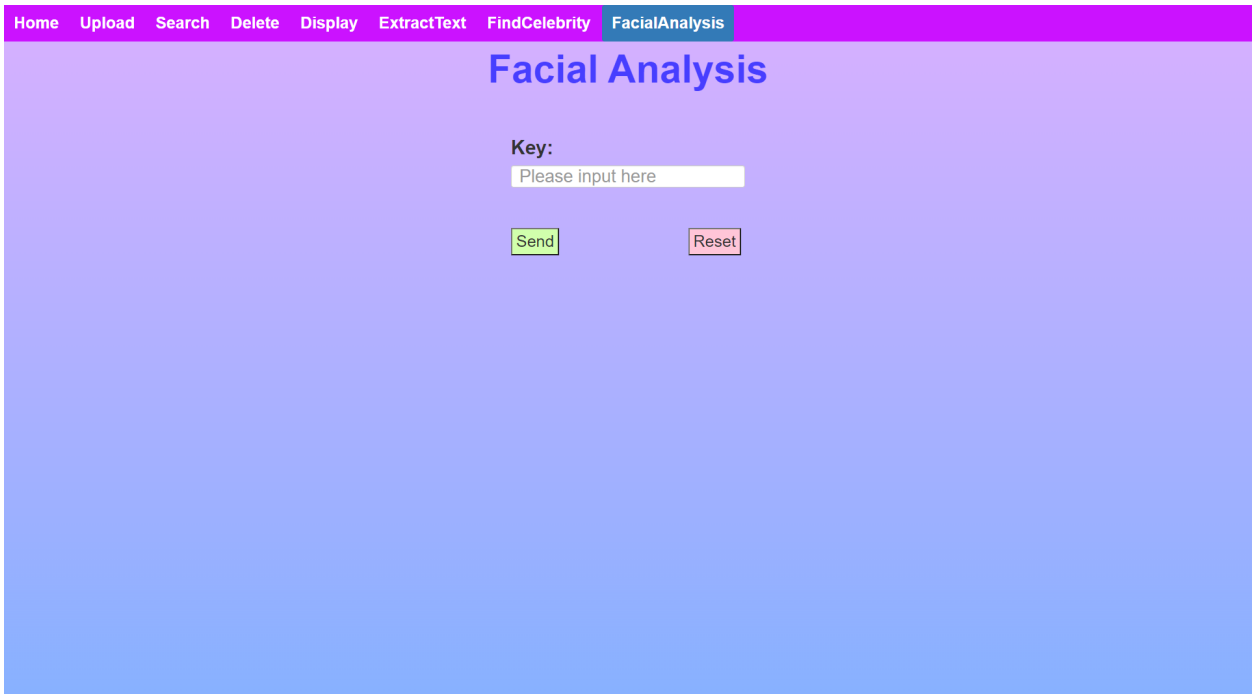

Figure 14. Facial Analysis

Figure 15. Analysis success

For Developers (Manager_app):

Homepage

When developers open Manager_app, the homepage is like figure 16 shows.

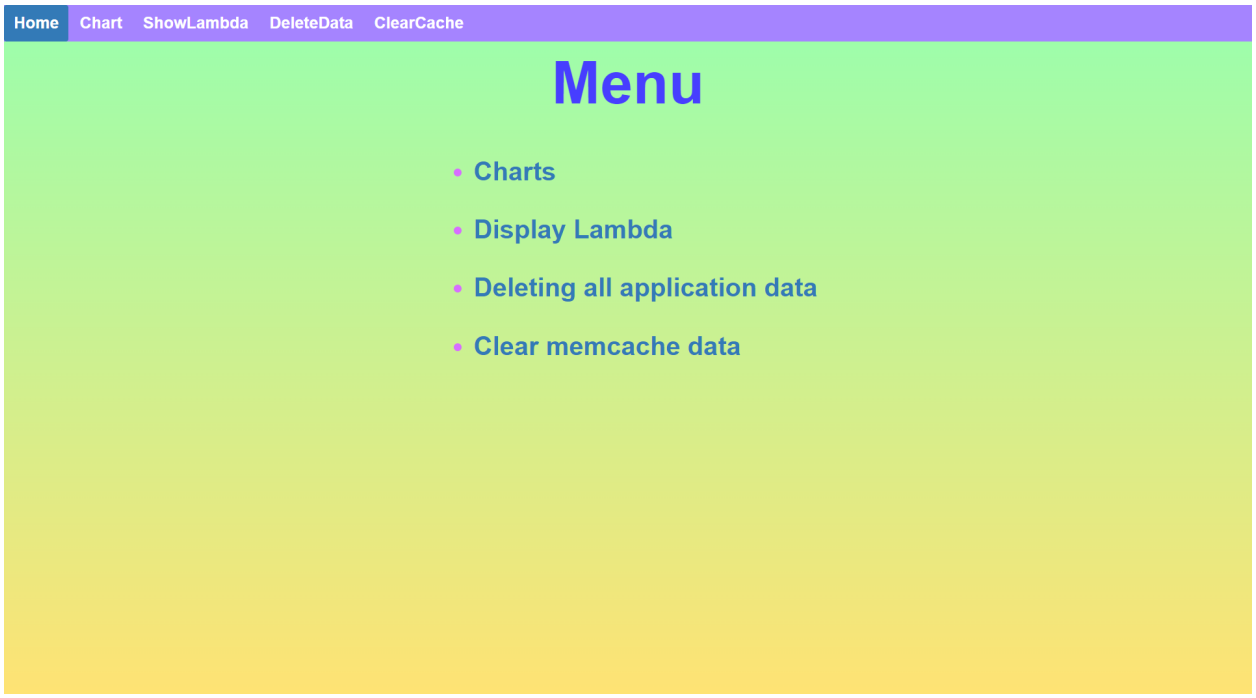

Figure 16. Homepage

Display charts

For displaying charts, please click on the corresponding item and the result will show figures 17 and 18 which will show five charts and they are: work numbers, miss rate & hit rate, request number, item number, and total size.

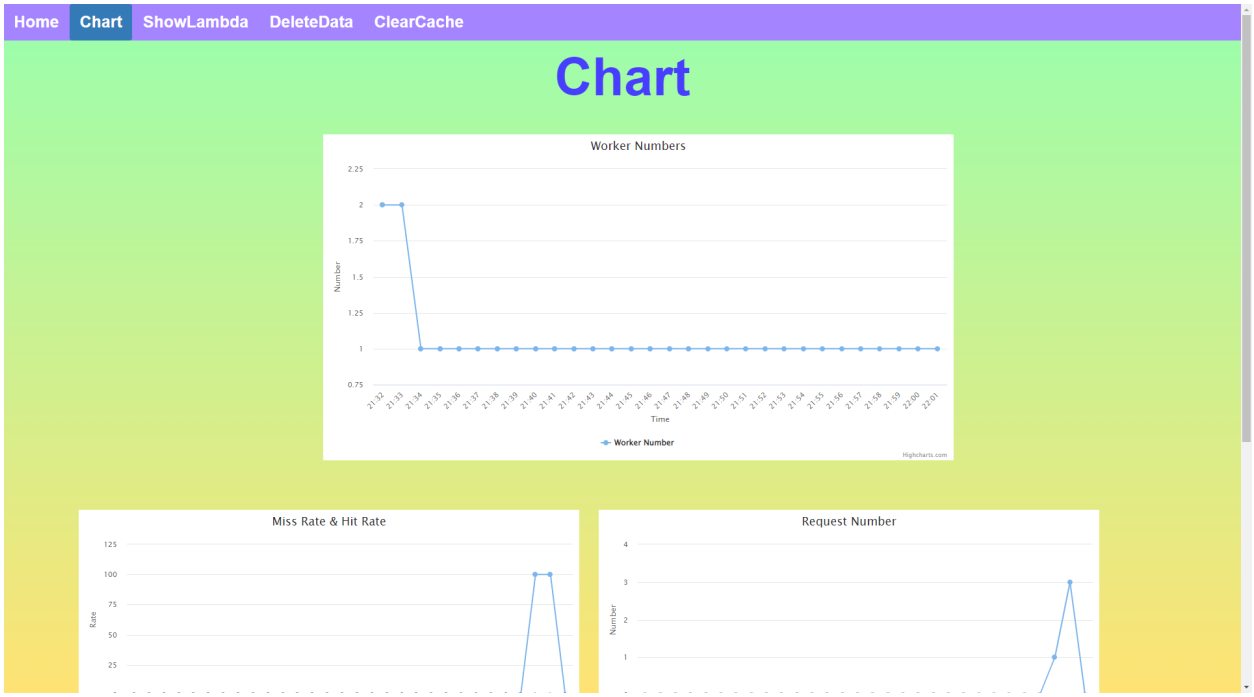

Figure 17. Displaying charts

Figure 18. Displaying charts

Display Lambda

For displaying Lambda, please click on corresponding item and the result will show like figure 19 which will show two charts and they are error number and request duration.

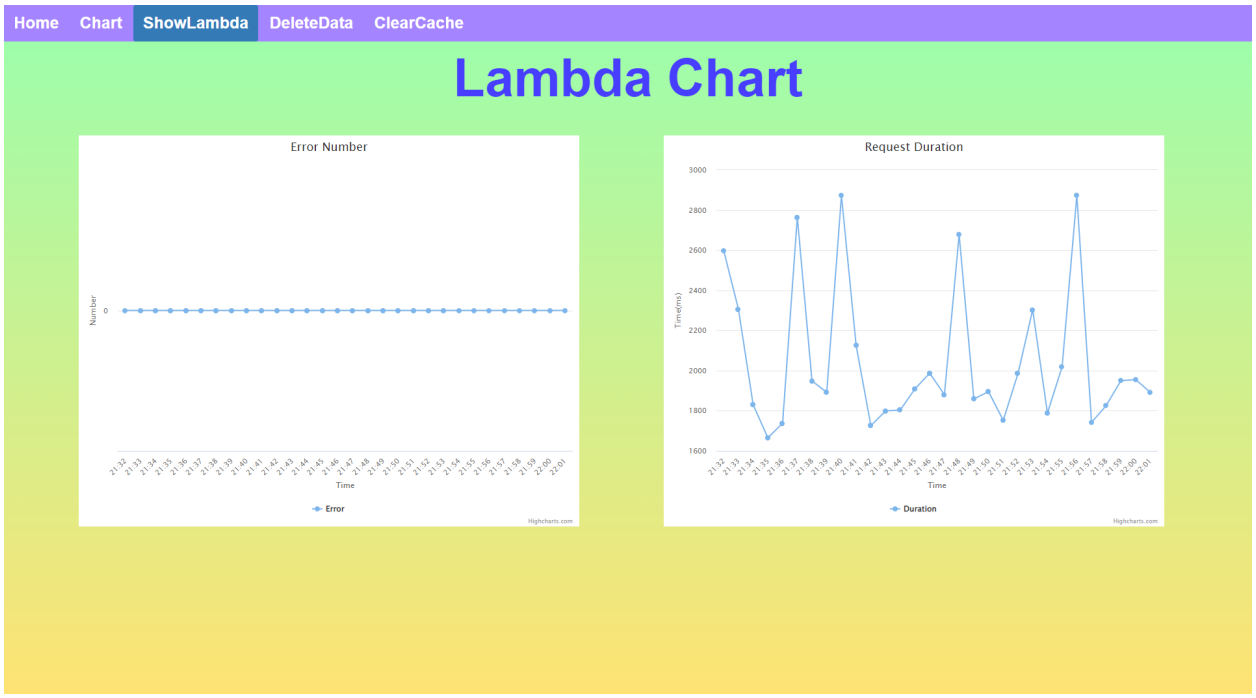

Figure 19. Display Lambda

Delete application data

For deleting application data, please click on the delete button on figure 20.

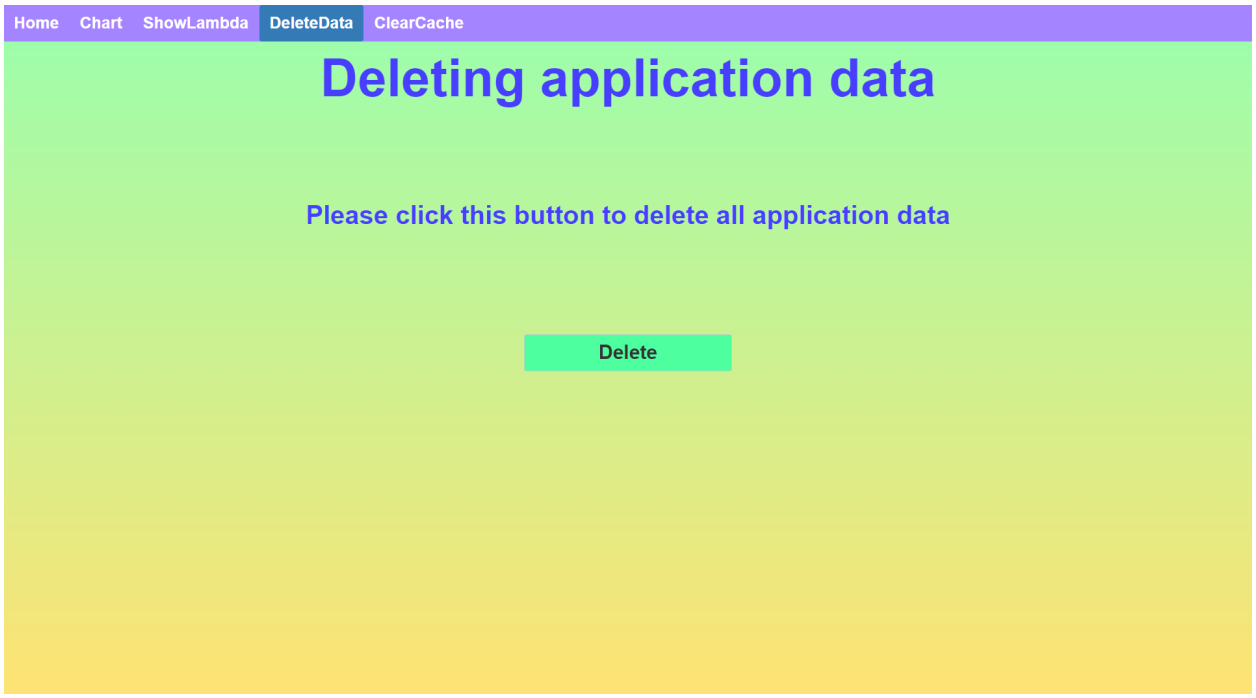

Figure 20. Deleting page

Clear memcache

For clearing memcache, please click on the clear button on figure 21.

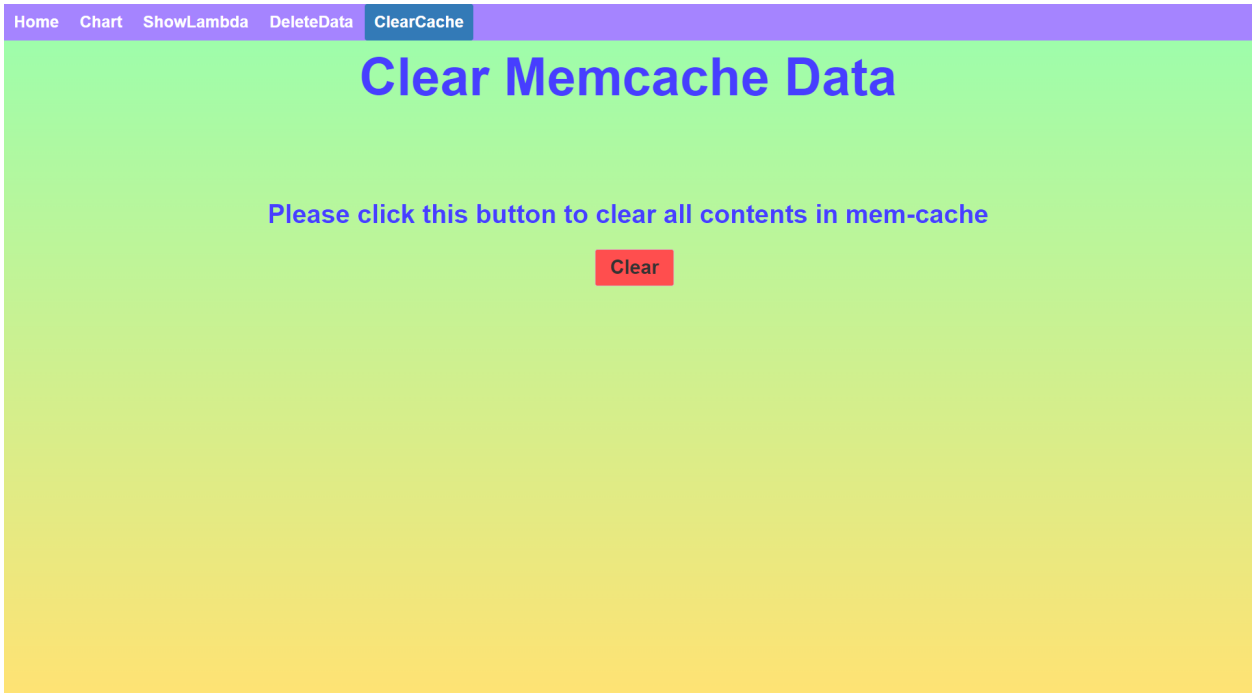

Figure 21. Clear memcache

Development Documentation

Project Structure

Auto Scaler

It's an independent program running in **lambda**, which checks the memory usage **at 4:00 am EST every day** and auto scales the number of memcache nodes.

Metrics Sender

It's another independent program running in **lambda**, which calculates and publishes the aggregate CloudWatch metric of all working nodes **every minute**.

Web server

We use the Model-View-Controller(MVC) design pattern to develop the web server. Operations related to interacting with dynamodb, like creating tables, querying data and modifying data, are handled in Model. View is responsible for presenting data and providing user interface at the front end. All the HTMLs are put in the template folder. Controller is used to handle clients' requests and interact with Model and View. We divided it into photos.py, reko.py and init_cache.py .

```
web_flask
├── app
│   ├── __init__.py
│   ├── __pycache__
│   ├── config.py # basic configuration information
│   ├── init_cache.py # initialize cache node when web starts
│   ├── main.py
│   ├── models
│   │   ├── __pycache__
│   │   ├── clear_storage.py # clear old contents when web starts
│   │   ├── create_tables.py # create tables in dynamodb
│   │   └── modify_tables.py # modify and query info in dynamodb
│   ├── photos.py # upload, delete and show images
│   ├── reko.py # interface of Machine Learning function
│   ├── static
│   │   └── style
│   └── templates
│       ├── base.html
```

```
│        ├── celebrity_form.html
│        ├── delete_form.html
│        ├── displayKey.html
│        ├── facial_form.html
│        ├── returnCele.html
│        ├── returnFacial.html
│        ├── returnPage.html
│        ├── returnText.html
│        ├── search_form.html
│        ├── textract_form.html
│        ├── upload_form.html
│        └── view.html
├── run.py
└── zappa_settings.json
```

Memory Cache

The mem-cache module serves as a storage buffer of the web server, which would speed up the image lookup process. It supports cache data modification and cache property configuration. Also, it will calculate statistics data periodically and upload it to the database.
The structure of the mem-cache module is as follow:

```
memcache
├── memcache_app
│    ├── __init__.py
│    ├── config.py  # basic configuration information
│    ├── aws.py  # Operations related to AWS are encapsulate here
│    ├── main.py # memcache server api
│    ├── memcache.py # memcache class
│    ├── models # functions to interact with the dynamodb
│    │    └── modify_tables.py
│    ├── timer.py # periodically publish CloudWatch metrics
│    └── total_size.py # a module to calculate the memcache item size
└── run.py
```

Manager App

This app is responsible for showing charts of memcache pool, charts of lambda status and clear data and cache.

```
manager_app
├── app
│   ├── __init__.py  # initialization
│   ├── aws.py  # Operations related to AWS are encapsulate here
│   ├── config.py  # basic configuration information
│   ├── configure.py  # clear data and cache
│   ├── display.py  # display charts
│   ├── main.py
│   ├── models  # Operations related to dynamodb
│   │   └── modify_tables.py
│   ├── static
│   │   └── Bootstrap
│   └── templates
│       ├── base.html
│       ├── clear.html
│       ├── delete.html
│       ├── returnPage.html
│       ├── showChart.html
│       └── showLambda.html
├── zappa_settings.json
└── run.py
```

Web Server

Upload image

When the **upload_image()** function in **photo.py** receives the key and image uploaded by users, it will save the file to **S3** and add {key, address} information to **dynamodb**. After that, the **invalidate request** will be routed to the corresponding **memcache** node in the **hash ring**.

Delete image

When the **delete_image()** function in **photo.py** receives the key to be deleted, it will first check if the key information exists in **dynamodb**. If the key exists, the **invalidate request** will be routed to the corresponding **memcache** node in the **hash ring** and the corresponding image in **S3** will also be deleted. Finally, key information will be removed from **dynamodb**.

Show image

When users click the **list_key url** at the front end, the **display_image_name()** function in **photo.py** will query all the information in the **Photo Table in dynamodb** and return the key list to the front end.
After that, if users **click a certain key** in the key list displayed at the front end, the javascript function **myFunction()** in **displayKey.html** will send a **show request** to the server and call the **show_image()** function in **photo.py**. This function will retrieve a certain photo from **memcache** or **S3** and show the image at the front end. Besides, if the photo is not saved in the memcache, a **put request** will be sent to the **routed memcache** and the memcache will save that image.

Textract image

When the **textract_image()** function in **reko.py** receives the key uploaded, it will first determine whether the image exists by querying the key in **dynamodb**. If the image exists, **detect_text()** function in **aws rekognition** will be called and extract the text in the image from **S3**. The result will be displayed line by line. If no text is detected, a reminder will be sent to the front end.

Celebrity Detection in image

When the **celebrity_image()** function in **reko.py** receives the key uploaded, it will first determine whether the image exists by querying the key in **dynamodb**. If the image exists, **recognize_celebrities()** function in **aws rekognition** will be called and detect celebrities in the image from **S3**. The result will be displayed in a list if more than one celebrity is shown in this image. If no celebrity is detected, a reminder will be sent to the front end.

Facial Analysis of image

When the **facial_image()** function in **reko.py** receives the key uploaded, it will first determine whether the image exists by querying the key in **dynamodb**. If the image exists, **detect_faces()** function in **aws rekognition** will be called and analyze facial information in the image from **S3**. The result includes age range, gender and emotion of the detected face. If no face is detected, a reminder will be sent to the front end.

Manager App

Show Charts

When users send http requests to show charts, the **display_charts()** function in display.py will be called.

The **display.py** file is responsible for getting statistics uploaded by **lambda** function and delivering NumberofWorkers, MissRate, HitRate, NumberofItems, TotalSize, NumberofRequests and timestamps to showChart.html.
showChart.html uses javascript and highchart tools to draw those charts and display them.

Show Lambda Status

When users send http requests to show charts, the **display_lambda()** function in display.py will be called.
The function will call the lambda_monitor_data() function in aws.py, which is responsible for getting statistics uploaded by the lambda function and delivering error number and request duration and timestamps to showLambda.html.
showLambda.html uses javascript and highchart tools to draw those charts and display them.

Delete application data

When users send requests at the front end, the **delete_data()** function in the **configure.py** file will be called. This function will clear **photo table** in dynamodb and delete all the files in **S3 /images** folder

Clear memcache data

When users send requests at the front end, the **clear_cache()** function in the **configure.py** file will be called. This function will first query the **MemNode table** in dynamodb and get public ips of all the running memcache instances and send clear requests to all the ips.

Memcache

The memcache module serves as a storage buffer of the web server, which would speed up the image lookup process. It supports cache data modification and cache property configuration. Also, it will calculate statistics data periodically and publish it to CloudWatch.
Our project implemented a distributed memcache pool. The number of memcache nodes in the pool is adjusted dynamically by a periodical Auto Scaler function to achieve the load balance.

Insert Data

Memcache nodes enable users to insert data. Each piece of data is inserted as the **(key, value)** pattern.

Search Data

Users can look up data in the memcache by passing a key to it. If the key is presented in the memcache, its corresponding value will be returned. Or, memcahe will return ‘miss’.

Drop Data

Users can drop data from the memcache by passing a key to it. The corresponding key and value will be permanently deleted from the memcache.

Clear Memcache

Users can clear the whole memcache node. All keys and values will be permanently deleted from the memcache.

Auto Replacement

When the total size of data in the memcache exceeds the max capacity, the memcache will automatically drop the data. By default, the replacement policy is the **Least Recently Used (LRU)**, and the **Random** policy is an alternative choice.

Memcache Configuration

Memcache enables users to configure its **capacity** and **replacement policy**.

Publish Cloudwatch Custom Metrics

This function is achieved by our self-defined **AWSClient** class (based on boto3). By using the AWSClient object as a handler, we are able to interact with AWS CloudWatch, so as to publish and get CloudWatch metrics efficiently. In our application, the memcache node would call **AWSClient.put_statistics()** every 5 seconds to publish its statistics to CloudWatch.

Lambda Function

Our project implemented AWS Lambda service to deploy two background programs (MetricsSender and AutoScaler) that run periodically. This would significantly reduce the burden of our application. Two lambda functions are as follow:

Metrics Sender

Description:

Metrics Sender is a background function that periodically generates aggregate CloudWatch metrics of all running memcache nodes.
Each time it is activated, the function will first acquire a list of all working memcache node, and then retrieve their CloudWatch metrics in the past 1 minute. Then it will calculate an aggregate metric based on the collected data and publish it to CloudWatch. The aggregate metrics published by auto monitor are displayed as a chart on the web page.

Trigger:
The Metrics Sender function is scheduled to run **every one minute**.

Auto Scaler

Description:
Auto Scaler is a background function that can automatically control the size of the memcache pool to achieve the load balance.
Each time it is activated, it will first check the **current load** of the memcache nods, which is defined as:

$$load = total_size / (worker_num * max_capacity)$$

If current load is **larger than 75%**, the Auto Scaler will **grow** the memcache nodes, similarly, if current load is **less than 25%**, the Auto Scale will **shrink** the memcache nodes.

Trigger:
The Auto Scaler function is scheduled to run at **4:00 am EST everyday**.

Cost Model

Assumption

In our assumption, each user would upload five images with an average size of 2Mb every day. They would search their image 10 times, do text extract once, find the celebrity once, and do facial analysis once a day.

Cost Model

By using the AWS calculator, we get the cost model of one-month costs for 10 users, 1000 users, and 1000000 users:

	10 Users	1000 Users	1000000 Users
EC2	6.65 USD	82.00 USD	16400.00 USD
S3	0.14 USD	2.15 USD	2896.20 USD
DynamoDB	0.81 USD	3.06 USD	2500.56 USD
Lambda	0.00 USD	0.63 USD	139.59 USD
Rokognition	0.89 USD	90.00 USD	35950.00 USD
API Gateway	0.02 USD	2.43 USD	2220.00 USD
EventBridge	0.04USD	0.04 USD	0.04 USD
Total	8.55 USD	180.31 USD	60106.39 USD

Predict

Six-month cost for 10 users, 1000 users, and 1000000 users:

	10 Users	1000 Users	1000000 Users
EC2	39.90 USD	492.00 USD	98400.00 USD
S3	0.84 USD	12.90 USD	17377.20 USD
DynamoDB	4.86 USD	18.36 USD	15003.36 USD
Lambda	0.00 USD	3.78 USD	837.54 USD
Rokognition	5.34 USD	540.00 USD	215700.00 USD
API Gateway	0.12 USD	14.58 USD	13320.00 USD
EventBridge	0.24 USD	0.24 USD	0.24USD
Total	51.30 USD	1081.86 USD	360638.34 USD