

University of Toronto

Faculty of Engineering

Department of Electrical and Computer Engineering

ECE 1779 A1 - Web Development

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: 2022.2.20

Table of contents

Table of contents	1
Introduction	3
User Manual	3
How to use	3
Features Description	3
Home Page	3
Upload a New Pair of Key and Image	4
Image Search	5
Display All the Available Keys Stored in the Database	6
Configure the mem-cache parameters	7
Cache Statistics	8
Development Documentation	9
Features	9
Dependencies	9
Project Structure	10
Web server	10
Memory Cache	11
Implementation Logic	12
Database Design	15
Database Schema	15
Table Photo	15
Table Cache	15
Table Statistics	16
Database Manipulation Model	16
Models.create_tables	16
Models.modify_tables	16
Models.clear_storage	17
API: Web Server	17
photos.upload_image()	17
photos.search_image()	18
photos.display_image_name()	19
cache.cache_config()	19
cache.cache_clear()	20
cache.cache_stats()	20
API: Memcache Server	20
main.put()	20
main.get()	21

main.clear()	21
main.invalidate()	21
main.refresh()	22
Class Memcache	22
Variable	22
Method	22
Memcache.get()	22
Memcache.put()	23
Memcache.replacement()	23
Memcache.lru_replacement()	24
Memcache.random_replacement()	24
Memcache.config()	24
Memcache.num()	25
Memcache.total_size()	25
Memcache.clear()	25
Memcache.invalidate()	25
Memcache.statistics()	25

Introduction

This project shows a web application for saving the key and corresponding image in the local file system and mem-cache. There are five main functions in this application, they are:

- Uploading the key and corresponding image. They will be saved in the local file system
- Searching the image use the given key
- Show all keys which are stored in the current database
- Set the configuration of the mem-cache
- Display the mem-cache's current statistics in last 10 minutes

User Manual

How to use

1. Login AWS with the information provided in credentials.txt
2. Start EC2 instance `ece1779`
3. Start terminal and connect EC2 instance with the command:

```
ssh -i "path/to/keypair.pem" ubuntu@34.203.97.82
```

4. Run the start.sh file with command:

```
./start.sh
```

5. Visit the website by url: <http://34.203.97.82:5000/>
6. After using the web server, terminate it by Ctrl + c
7. If you want to re-run the program, go to step 4

Features Description

Home Page

When users start our application successfully, the homepage will show as Figure 1. There are 5 items on this page which are "Image Upload", "Image Search", "Display All Image Name", "Cache Configuration", and "Cache Statistics". When the user clicks, it automatically redirects to the corresponding page.

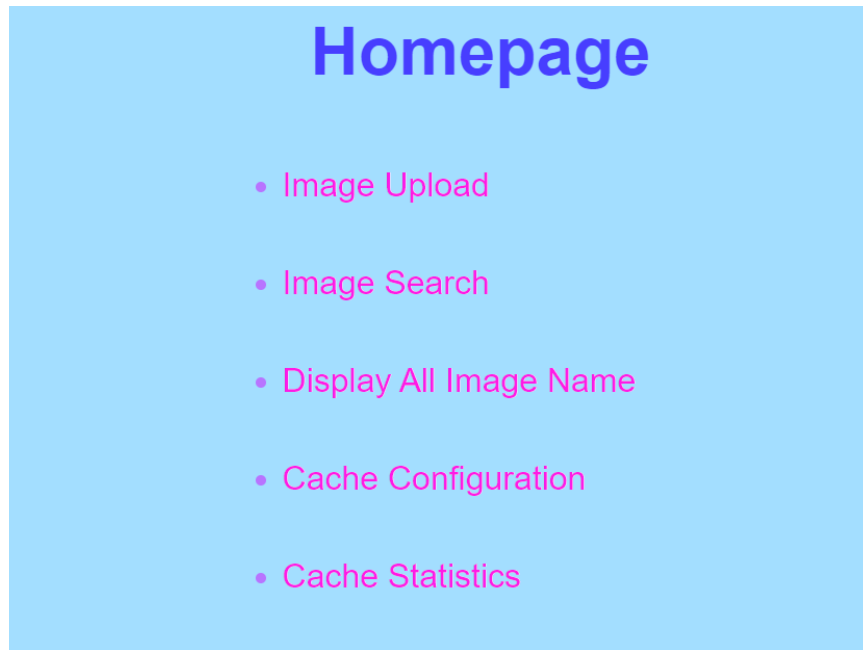


Figure 1. Homepage

Upload a New Pair of Key and Image

When users click "Image Upload", it will go to a page called "upload form" like Figure 2 for uploading an image. Users need to input a key first, then select an image from the local file, then click the "Send" button. If users want to reset everything on this page, they just need to click the "Reset" button. If the user forgets to input the key or select the file, there will be two corresponding upload failure messages displayed on two pages like figure 3 and figure 4. If images are uploaded successfully, there will be a page to tell the user that "Success upload your photo!" like figure 5.

The image shows a light blue rectangular area representing a webpage. At the top center, the words "Upload Form" are written in a large, bold, blue font. Below this, there is a form with the following elements: a label "Key:" followed by a text input field containing the placeholder text "Please input here"; a label "What is the image files to upload? (less than 5MB)" followed by a file selection button labeled "Choose File" and the text "No file chosen"; and two buttons at the bottom, "Send" (green) and "Reset" (red).

Figure 2. Upload a New Pair of Key and Image

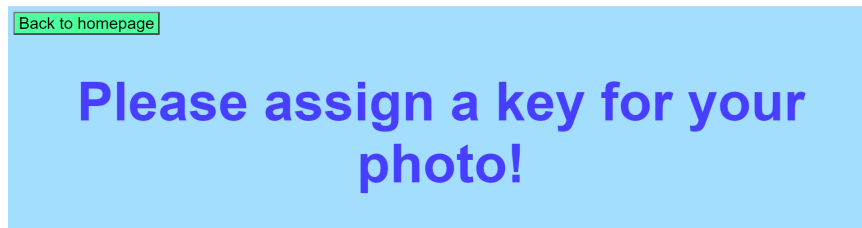


Figure 3. Forget to input the key

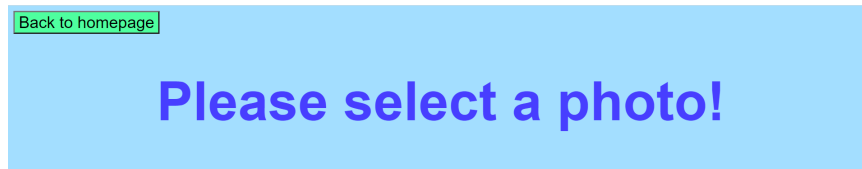


Figure 4. Forget to select a photo

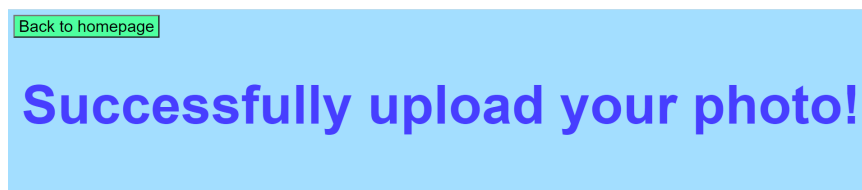


Figure 5. Upload successfully

Image Search

When users click “Image Search”, it will go to a page called “search form” like Figure 6 for users to search the image associated with a given key. Users just need to input the key of an image in the “Photo Name” box and then click the “Send” button. If the user forgets to input the key, there will be a corresponding message displayed on a page to remind the user to input a key as figure 7 shows. If the image is found, then there will be a new page to show the image like figure 8. If the image is not found, there will be a page to tell users “No photo found” as figure 8.

A screenshot of a web page titled "Search Form" in a large, bold, blue font. The page has a light blue background. Below the title, there is a label "Photo Name:" followed by a white text input box with a light gray border and the placeholder text "Please input here". At the bottom of the page, there are two buttons: a green button with the text "Send" and a red button with the text "Reset".

Figure 6. Image Search



Figure 7. Forget to input the key



Figure 8. Find the image

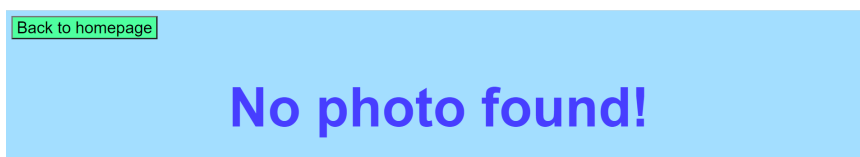


Figure 9. No photo found

Display All the Available Keys Stored in the Database

When users click “Display All Image Name”, it will go to a page called “result” like Figure 10 for showing all image keys stored in the database.

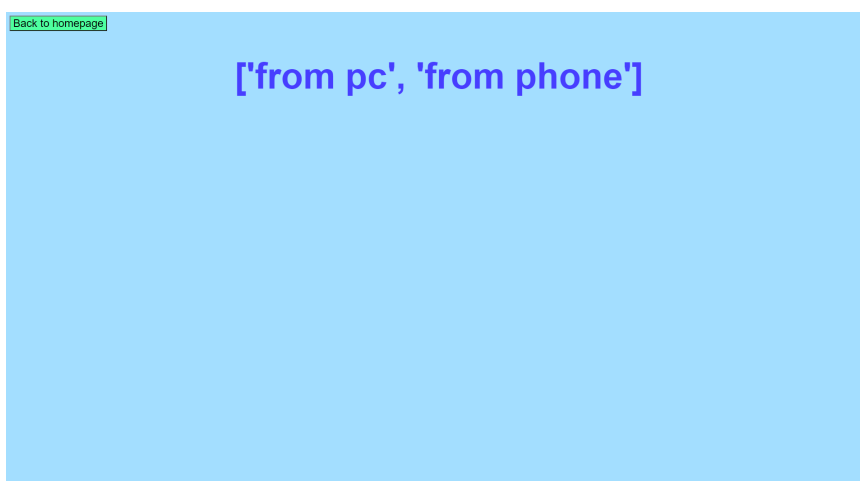
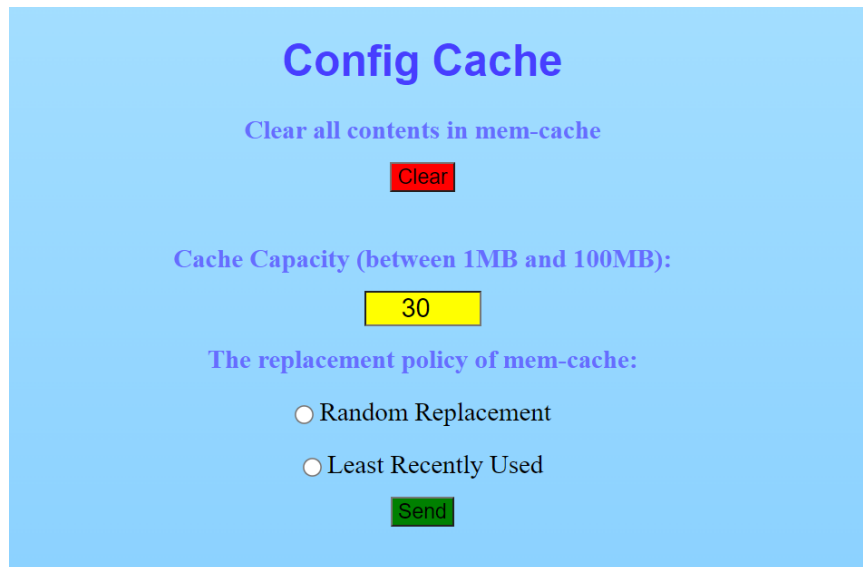


Figure 10. Display all image names

Configure the mem-cache parameters

When users click “Cache Configuration”, it will go to a page called “Config form” like Figure 11 for users to set the configuration of the mem-cache. Clicking the “Clear” button will clear all contents in mem-cache and get a page as Figure 12 shows. Users could set the cache capacity in the capacity box and choose the replacement policy of men-cache. If users finish the form and click the send button, they will get a page as Figure 13 shows.



The image shows a web form titled "Config Cache" on a light blue background. At the top, it says "Clear all contents in mem-cache" with a red "Clear" button below it. Then, it asks for "Cache Capacity (between 1MB and 100MB):" with a yellow input box containing the number "30". Below that, it asks for "The replacement policy of mem-cache:" with two radio button options: "Random Replacement" and "Least Recently Used". At the bottom, there is a green "Send" button.

Figure 11. Config Cache



The image shows a confirmation message on a light blue background. It starts with a green "Back to homepage" link. Below it, the text "Memcache has been cleared" is displayed in large, bold, blue font.

Figure 12. Clear Mem-Cache



The image shows a confirmation message on a light blue background. It starts with a green "Back to homepage" link. Below it, the text "current capacity=30.0MB, replacement policy=1" is displayed in large, bold, blue font.

Figure 13. Finish Config Cache

Cache Statistics

When users click “Cache Statistics”, it will go to a page called “Cache Statistics” like Figure 14 for showing the cache statistics.

Back to homepage					
Cache Statistics					
time	number	size	requests	hitRate	missRate
2022-02-19 16:21:55	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:50	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:45	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:40	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:35	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:30	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:25	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:20	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:15	2	7.630MB	3	33.3333	66.6667
2022-02-19 16:21:10	2	7.630MB	2	0.0	100.0
2022-02-19 16:21:05	1	2.842MB	1	0.0	100.0
2022-02-19 16:21:00	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:55	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:50	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:45	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:40	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:35	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:30	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:25	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:20	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:15	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:10	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:05	1	2.842MB	1	0.0	100.0
2022-02-19 16:20:00	0	0.000MB	0	0.0	0.0

Figure 14. Cache Statistics

Development Documentation

Features

- Upload photos
- Search photos with specific keys
- Display all the keys in database
- Config and clear mem-cache
- Display statistics of mem-cache over the past 10 minutes

Dependencies

Our project runs in a python virtual environment:

- Python
- Virtualenv (you can also use Anaconda)

For complete features, we use packages as follow:

- Flask: A python web framework for development
- Flask-SQLAlchemy: Manipulating MySQL from flask application
- MySQL: RDBMS for data collection
- requests: Send HTTP requests between servers
- uWSGI: A deployment tool for Flask application

```
Flask==2.0.1
Flask-SQLAlchemy==2.5.1
mysqlclient==2.1.0
requests==2.27.1
SQLAlchemy==1.4.31
uWSGI==2.0.20
Werkzeug==2.0.1
```

Project Structure

Web server

We use the Model-View-Controller(MVC) design pattern to develop the web server. Operations related to interacting with databases, 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, cache.py and api.py.

```
web_flask
├── app
│   ├── __init__.py
│   ├── api.py
│   ├── cache.py
│   ├── main.py
│   ├── models
│   │   ├── clear_storage.py
│   │   ├── create_tables.py
│   │   └── modify_tables.py
│   ├── photos.py
│   ├── static
│   │   ├── pictures
│   │   └── style
│   │       ├── config_form_style.css
│   │       ├── mainstyle.css
│   │       ├── returnPage_style.css
│   │       ├── search_form_style.css
│   │       ├── statsstyle.css
│   │       ├── upload_form_style.css
│   │       └── viewstyle.css
│   └── templates
│       ├── base.html
│       ├── config_form.html
│       ├── returnPage.html
│       ├── search_form.html
│       ├── stats.html
│       ├── upload_form.html
│       └── view.html
└── run.py
```

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
│   ├── main.py
│   ├── memcache.py
│   ├── models
│   │   ├── create_tables.py
│   │   └── modify_tables.py
│   ├── timer.py
│   └── total_size.py
└── run.py
```

Implementation Logic

Search photo with specific key

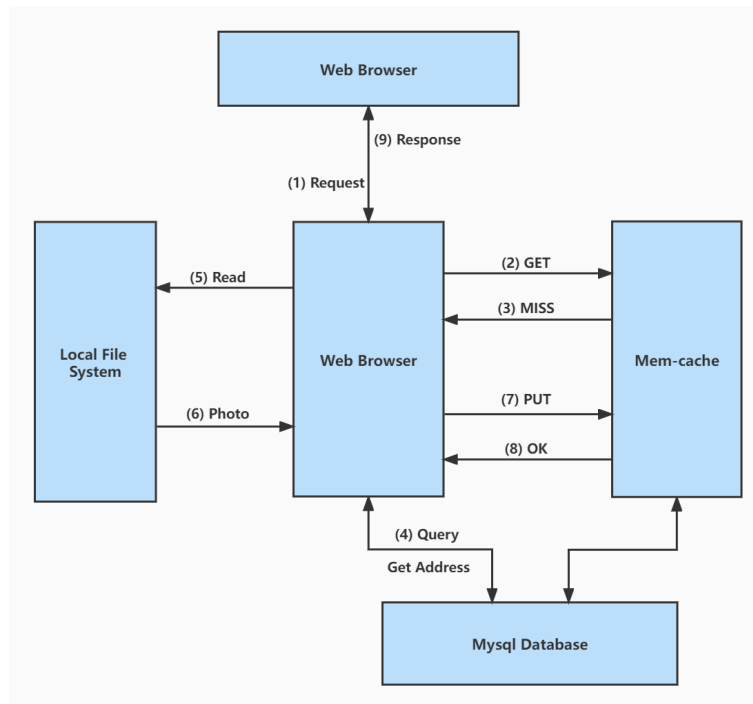


Figure 15

Upload photo

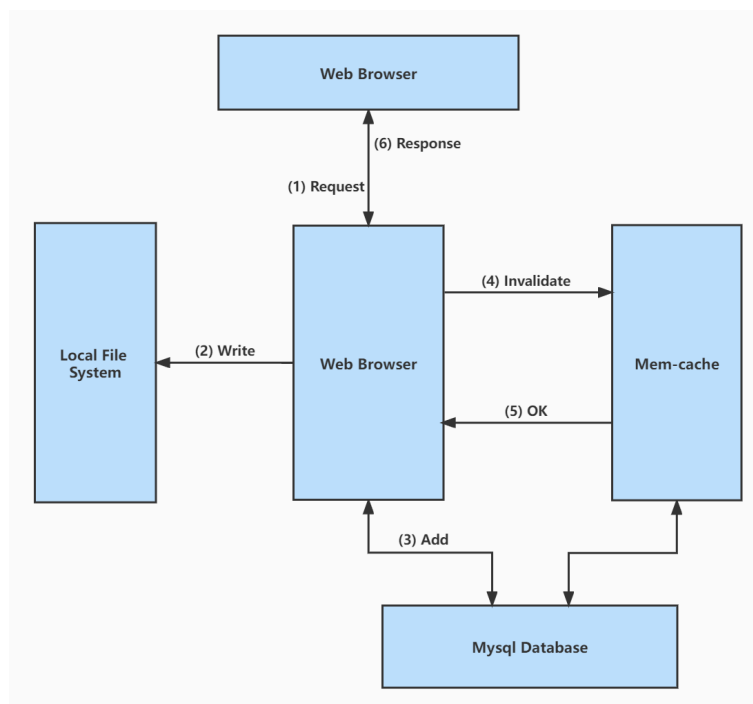


Figure 16

Display all the keys stored in database

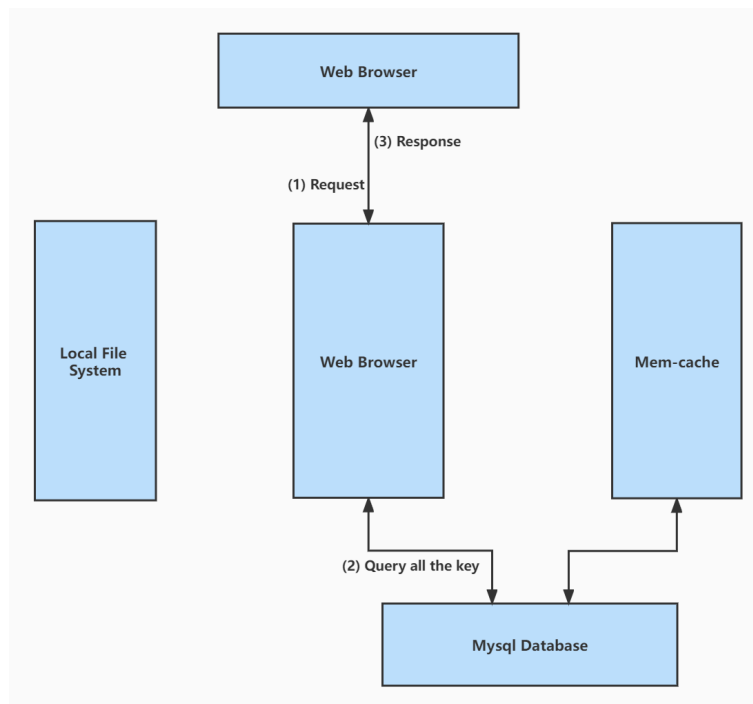


Figure 17

Config cache

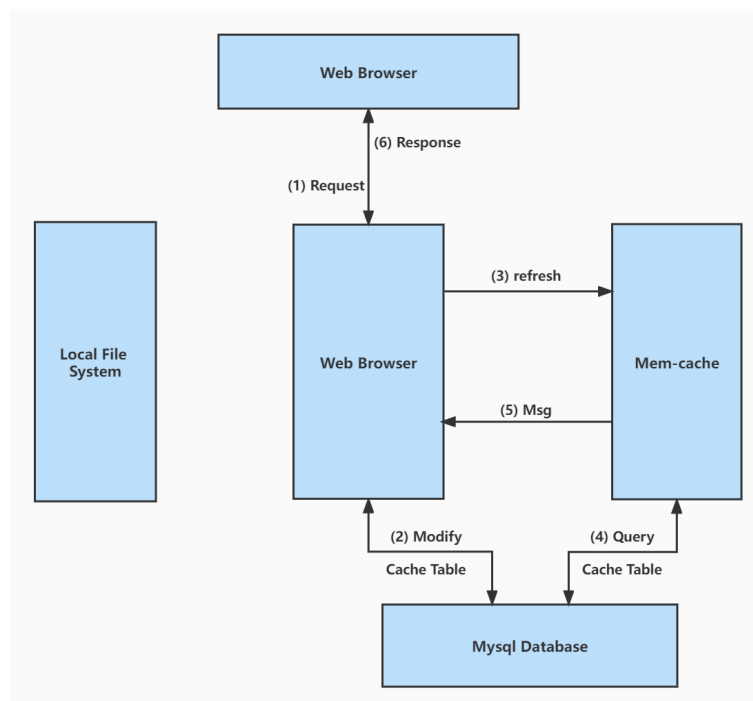


Figure 18

Display statistics over the past 10 minutes

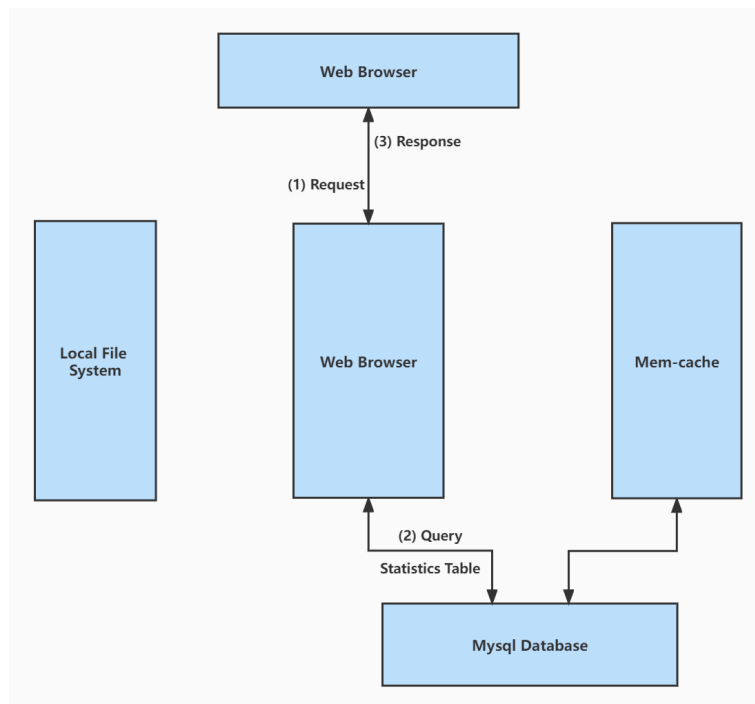
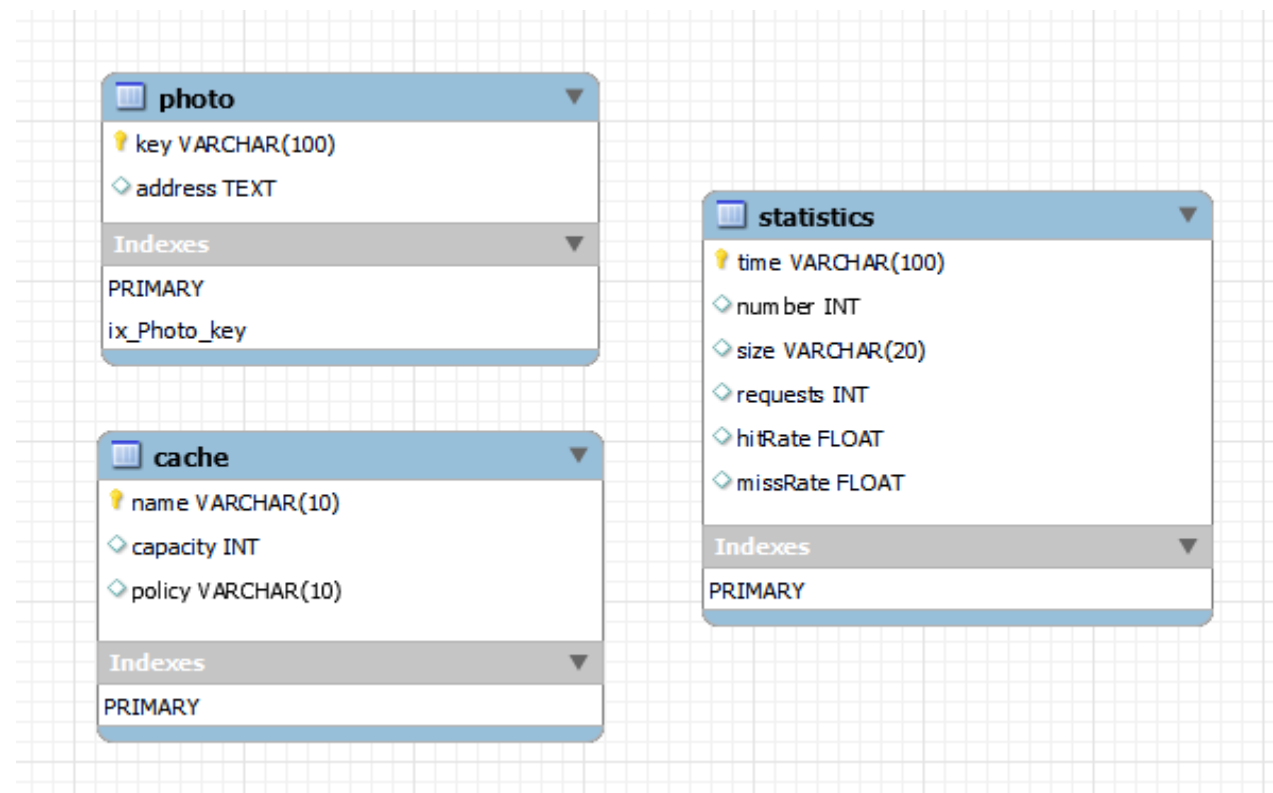


Figure 19

Database Design

Database Schema



We created three tables. Table Photo stores key and address of photos. Table Cache stores configuration information of mem-cache. Table Statistics stores statistics of mem-cache.

Table Photo

Column	Data Type	Description
<u>key</u>	String	The key users specified
address	Text	Address of the image in local file system

Table Cache

Column	Data Type	Description
<u>name</u> (default="local")	String	Name of the mem-cache, in A1 we only have "local"
capacity	Integer	The capacity of mem-cache in MB
policy	String	The replace policy of the mem-cache

Table Statistics

Column	Data Type	Description
<u>time</u>	String	Timestamps
number	Integer	Number of items in mem-cache
size	String	Total size of items in mem-cache in MB
requests	Integer	Number of requests served
hitRate	Float	Hit rate
missRate	Float	Miss rate

Database Manipulation Model

Models.create_tables

Description:

Define the tables in database

Table Class:

Photo(key, address)

Cache(name="local", capacity, policy)

Statistics(time, number, size, requests, hitRate, missRate)

Models.modify_tables

Description:

Operations including query and modification on database

Member function

add_photo():

Input: key, absolute address in file system

Description: add 1 row in Photo table in database

change_photo():

Input: photo, new absolute address in file system

Description: modify 1 row in Photo table in database

search_photo():

Input: key

Return: photo address corresponds to the key

Description: query 1 row in Photo table in database

query_all():

Input: An empty key list

Return: key_list stored all the keys

Description: query all rows in Photo table in database

config():

Input: capacity, policy

Description: modify the configuration in Cache table in database

query_stats():

Input: None

Return: stats_list stored all the statistics of last 10 min

Description: query rows of last 10 min in Statistics table in database

upload_statistics():

Input: time, number, size, requests, hitRate, missRate

Return: None

Description: upload current mem-cache statistics to the database

get_config():

Input: None

Return: capacity, policy

Description: get configuration parameters from the database

Models.clear_storage

Description:

Clear all the photos and data in the database at the initialization stage

API: Web Server

photos.upload_image()

Method: POST

Route: /api/upload_image

Description:

Receive images uploaded from clients, save them to the local file system, store photos' information to the database and send the /invalidate requests to mem-cache.

Parameter

Field	Type	Description
key	String	The key users specified
file	File	The image users uploaded

Response

Type: A web page displayed messages from web server

Type	Reason	Message
Success	/	Successfully upload your photo!
Failed	Empty input	No input!
Failed	Empty key	Please assign a key for your photo!
Failed	Empty file selection	Please select a photo!
Failed	Wrong image type	Please upload photo in jpg, jpeg, gif, png type!
Failed	File size too big	Request entity too large

Automatic testing endpoints:

Relative URL: /api/upload

Function: api.upload()

Response: JSON

- {"success": "true"}
- {"success": "false", "error": {"code": 400 "message": Message listed in the table above}}

photos.search_image()

Method: POST

Route: /api/search

Description:

Receive keys specified by clients, send the /get request to mem-cache. If the photo is not found, query it in the database and retrieve the photo from the local file system. Add key to mem-cache by sending the /put request. Finally, show the image on the web page.

Parameter

Field	Type	Description
key	String	The key users specified

Response

Type: A web page displayed messages/photos from web server

Type	Reason	Message
Success	/	A view page displayed the photo
Failed	Empty key	Please input a key!
Failed	No photo corresponds to the key	No photo found!

Automatic testing endpoints:

Relative URL: /api/key/<key_value>

Function: api.search_by_key()

Response: JSON

- {"success": "true", "content": file contents}
 - {"success": "false", "error": {"code": 400 "message": Message listed in the table above}}
-

photos.display_image_name()

Method: GET, POST

Route: /api/key_display

Description:

Display all the keys on a page.

Parameter: None

Response

Type: A view page with all keys displayed

Automatic testing endpoints:

Relative URL: /api/list_keys

Function: api.list_keys()

Response: JSON

- {"success": "true", "keys": [Array of keys]}
-

cache.cache_config()

Method: POST

Route: /api/cache

Description:

Receive configuration set by clients, save it in the database and send the /refresh request to mem-cache.

Parameter

Field	Type	Description
capacity	Integer	The cache capacity users specified
replace	String	The replace policy users specified

Response

Type: A web page displayed messages from mem-cache server

Type	Reason	Message
Success	/	current capacity: {capacity} MB, replacement policy: {replace}

cache.cache_clear()

Method: POST

Route: /api/cache/clear

Description:

Send the /clear request to mem-cache.

Parameter: None

Response

Type: A web page displayed messages from mem-cache server

Type	Reason	Message
Success	/	Memcache has been cleared

cache.cache_stats()

Method: GET

Route: /api/cache/stats

Description:

Displayed mem-cache statistics of the last 10 min.

Parameter: None

Response

Type: A web page displayed the statistics

API: Memcache Server

main.put()

Method: POST

Route: /put

Description:

Store specified key and its corresponding image content into the mem-cache. Return a message indicating whether or not the insert is successful.

Parameter

Field	Type	Description
key	String	The key of the image
value	String	The content of the image

Return Message

Message Type: JSON {"msg": {message-content}}

Message Content	Type	Description
-----------------	------	-------------

SUCCESS	String	Successfully put into cache
FULL	String	No space for new data

main.get()

Method: POST

Route: /get

Description:

Look up a specified key in the mem-cache. Return the corresponding image content or error message.

Parameter

Field	Type	Description
key	String	The key of the image

Return Message

Message Type: JSON {"value": {return-value}}

Return Value	Type	Description
{image-content}	Base64	Base64-encrypted image content
MISS	String	Can't find the key in the mem-cache

main.clear()

Method: GET, POST

Route: /clear

Description:

Drop all keys and values in the mem-cache.

Parameter: None

Return Message: JSON {"msg": "Memcache has been cleared"}

main.invalidate()

Method: POST

Route: /invalidate

Description:

Drop a specified key and its value from the mem-cache.

Parameter

Field	Type	Description
key	String	The key of the image

Return Message: JSON {"msg": "OK"}

main.refresh()

Method: GET, POST

Route: /refresh

Description:

Read mem-cache configuration values from the database, and reconfigure the mem-cache with these values.

Parameter: None

Return Message: JSON {"msg": {new capacity and replacement policy}}

Class Memcache

Variable

Variable Name	Type	Description
cache	OrderedDict()	cache to store keys and values
capacity	Int	mem-cache's capacity
rpolicy	Int	mem-cache's replacement policy 1 – LRU replacement 0 – random replacement
hit	Int	mem-cache's requests hit number
miss	Int	mem-cache's requests miss number

Method

Memcache.get()

Description:

Get the corresponding value of a specified key and return 1.

Parameter

Field	Type	Description
-------	------	-------------

key	String	The key of the image
-----	--------	----------------------

Return

Return Value	Type	Description
{image-content}	Base64	Base64-encrypted image content
-1	Int	The key is not in the mem-cache

Memcache.put()

Description:

Store the key and its corresponding value to the cache. Return whether it is successful.

Parameter

Field	Type	Description
key	String	The key of the image
value	Base64	Base64-encrypted image content

Return

Return Value	Type	Description
True	Boolean	Successfully put into the cache
False	Boolean	The cache has no space for new data

Memcache.replacement()

Description:

Drop items in the cache under the replacement policy. Return whether the replacement action is successful.

Parameter

Field	Type	Description
threshold	Int	The threshold to stop dropping

Return

Return Value	Type	Description
True	Boolean	Replacement action succeed

False	Boolean	The threshold is too small, failed
-------	---------	------------------------------------

*This method will call lru_replacement() or random_replacement() based on what the replacement policy is.

Memcache.lru_replacement()

Description:

Drop the least recently used (LRU) items in the cache until the cache's size reaches the threshold.

Parameter

Field	Type	Description
threshold	Int	The threshold to stop dropping

Return: None

Memcache.random_replacement()

Description:

Randomly drop items in the cache until the cache's size reaches the threshold.

Parameter

Field	Type	Description
threshold	Int	The threshold to stop dropping

Return: None

Memcache.config()

Description:

Configure the mem-cache properties. Return the mem-cache status after the configuration.

Parameter

Field	Type	Description
capacity	Int	mem-cache's capacity set by users
rpolicy	Int	mem-cache's replacement policy set by users

Return

Field	Type	Description
(capacity, rpolicy)	Tuple(Int, Int)	The mem-cache's capacity and replacement policy after configuration

Memcache.num()

Description:

Return the number of items stored in the mem-cache.

Memcache.total_size()

Description:

Return the total byte size of items stored in the mem-cache.

Memcache.clear()

Description:

Drop all items stored in the mem-cache.

Memcache.invalidate()

Description:

Return a specified key and its corresponding value from the mem-cache.

Parameter

Field	Type	Description
key	String	The key of the image to drop

Return: None

Memcache.statistics()

Description:

Calculate the current statistics of mem-cache and return.

Parameter: None

Return: Tuple(num, size, requests, hit_rate, miss_rate)

Field	Type	Description
num	Int	The number of items in mem-cache
size	Int	The size of items in mem-cache
requests	Int	The number of requests handled
hit_rate	Float	Hit rate
miss_rate	Float	Miss rate