

ECOM 5416

Parallel and Distributed Systems

Project 1

Web Development












Group member:

- Abdelrhman Abo deba`a 120180865
- Mohammed hamdan 120180318
- Tariq Hijazi 120180562

20/10/2022

Contents

01		Introduction
02		Usage
2.1		Insert page
2.2		Search page
2.3		All keys page
2.4		Edit-Mem page
2.5		Mem-cache page
03		Overall Architecture
04		Mem - Cache



0.1 Introduction

In Project 1 for ECOM 5416 :Parallel and Distributed Systems, we build a website that can upload photos from users with key. The website It will work AWS EC2 , with Flask as the backbone framework and MySQL as the database.

This document will provide a comprehensive review of our website

Section 2 will introduce the usage and navigation of the website

Section 3 will illustrate how the website code is architected , as well as how modularization is achieved

Section 4 will describe each code module in detail mem-cache .

Requirement To start Local

Use: Python 3

```
pip install Flask
pip install mysql-connector-python
pip install requests
pip install APScheduler
```

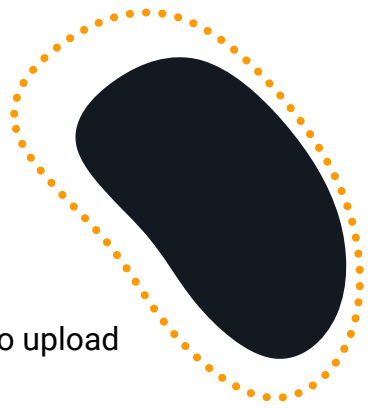
Use: **Web Server**

consists mostly of the Apache HTTP Server, MySQL Database

Once started, our website can be accessed either locally <http://localhost:5000>

0.2 Usage

Once connected to the website, users will see a insert page , A page to upload a new pair of key and image .



2.1 Insert page

A page to upload a new pair of key and image:

- The key should be used to uniquely identify its image , when click "Choose file".
- Click "Submit" , The web front end should store the image in the local file system, and add the key to the list of known keys in the database.
- A subsequent upload with the same key will replace the image stored previously.

A screenshot of a web application interface. At the top, there is a navigation bar with five links: 'Insert', 'Search', 'All keys', 'Edit-Mem', and 'Mem-Cache'. The 'Insert' link is highlighted. Below the navigation bar, the main content area has a title 'insert image' with a blue vertical bar to its left. Under the title, there are two input fields. The first is labeled 'Key:' and contains the text 'key'. The second is labeled 'img:' and contains a 'Choose File' button and the text 'No file chosen'. Below these fields is a blue 'submit' button.

Insert *Search* *All keys* *Edit-Mem* *Mem-Cache*

insert image

Key:

img:

 No file chosen

Insert

Search

All keys

Edit-Mem

Mem-Cache

Successfully , The image has been added

insert image

Key:

key

img:

Choose File No file chosen

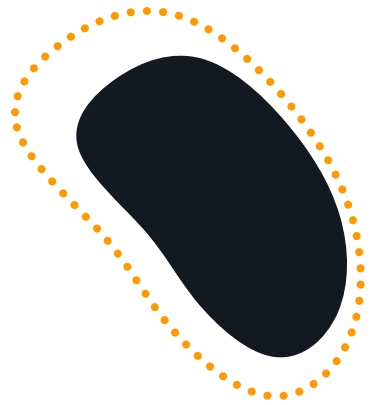
submit

2.2 Search page

A page that shows an image associated with a given key.

When the user put key and click "Get" , **GET** request that If the image associated with the key is present in the mem-cache or database, it will appear as Figure 2.2-1

if it does not exist, a flash message will appear ' **Invalid key** ' as Figure 2.2-2



Insert *Search* *All keys* *Edit-Mem* *Mem-Cache*

Get image

key

Get

Key:

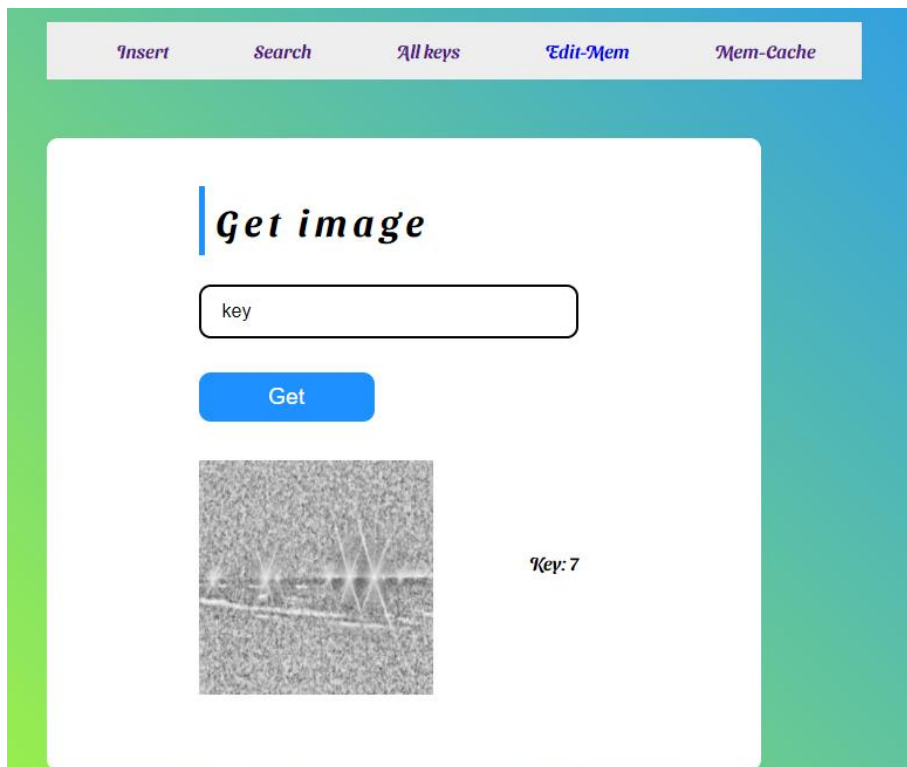


Figure 2.2 -1

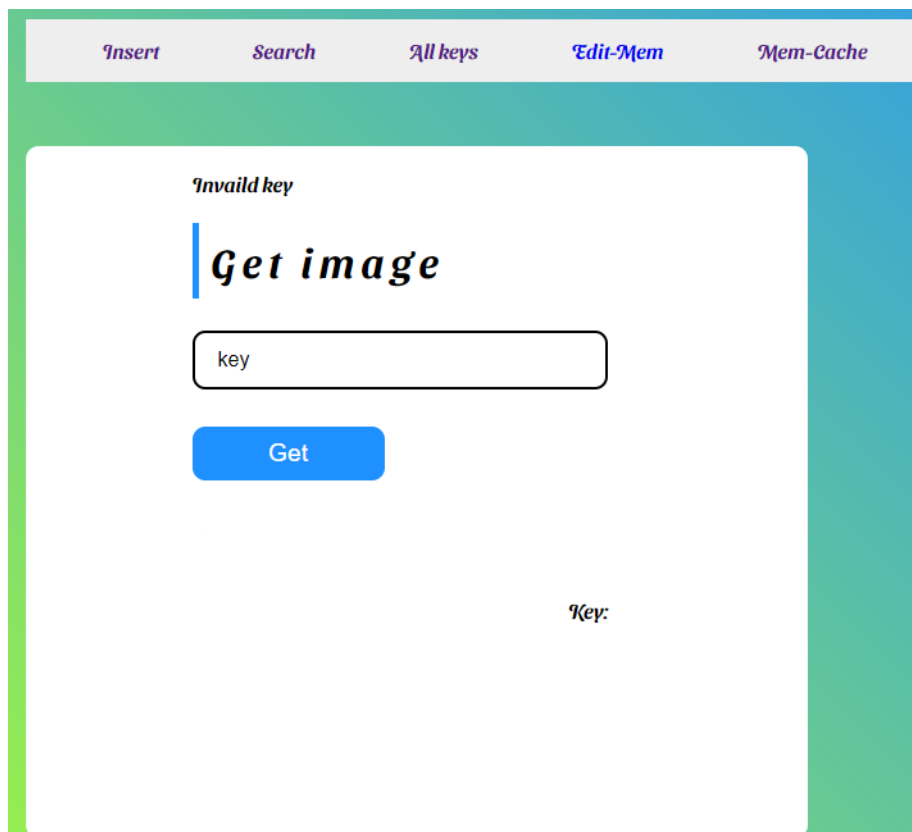
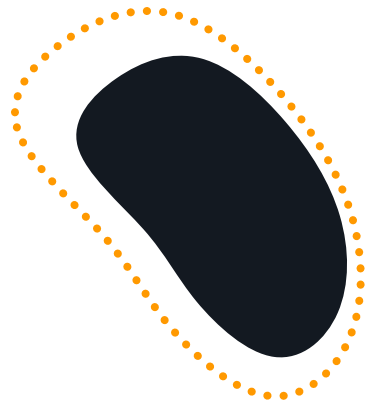


Figure 2.2 -2

2.3 All Keys page

A page that displays all the available keys stored in the database.



Insert

Search

All keys

Edit-Mem

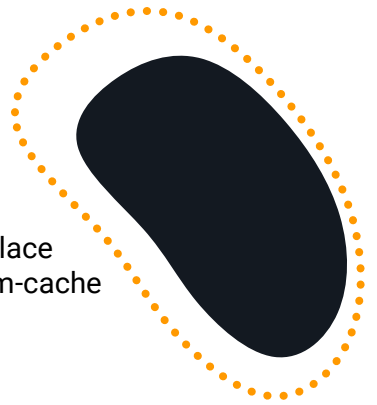
Mem-Cache

All keys

Key
1
12
2
3
4
7
test1

2.4 Edit-Mem page

A section to configure the mem-cache parameters (e.g. capacity in MB, replace policy) as well as clear the cache. When the user choose configuration mem-cache then click "Submit" , if need clear the cache click "clear".



Insert *Search* *All keys* *Edit-Mem* *Mem-Cache*

Memory-Cache Configuration:-

☒ *Random Replacement*
☐ *Least Recently Used*

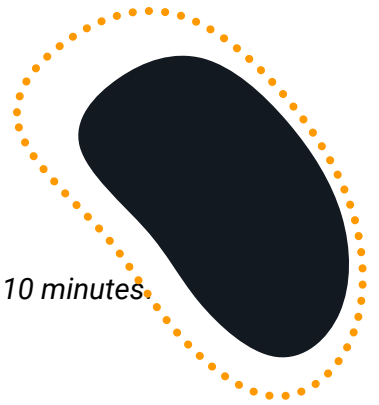
Memory Capacity

3/4 MB +/-

submit

clear

2.5 Mem-Cache page



Display a page with the current statistics for the mem-cache over the past 10 minutes.

The statistics (number of items in cache, total size of items in cache, number of requests served, miss rate and hit rate).

*Insert**Search**All keys**Edit-Mem**Mem-Cache*

Mem-Cache Statistics:-

Number of items 0

Total size of items: 0

Number of request: 0

Miss Rate: 0

Hit Rate: 0

0.3 Overall Architecture

The website is responsible for handling all user requests and queries when they click buttons or view pages of our website. It is implemented using the Flask framework in Python 3 . The website is designed in such a way that it not only responds to requests correctly, but is also less error-prone to programming mistake .

3.1 Request Handling

Request handler functions are directly tied to handling specific HTTP requests. The data is then passed into the logic functions form . And finally, the website redirects users or renders corresponding templated HTML pages.

when running , appears Status Code as 200 , 304 , 302

200 successful processing of the request

302 redirection is created on the site

```
@app.route("/keys",methods=["GET"])
def viewall():
    if request.method == 'GET':
        return render_template("keys.html",keys = keys)
```

```

@app.route("/upload-image", methods=["GET", "POST"])

def upload_image():
    if request.method == 'GET':
        return render_template("upload.html")

```

```

C:\Users\Lenovo\Desktop\Cloud>flask run
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [19/Oct/2022 11:43:01] "GET /upload-image HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2022 11:43:01] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Oct/2022 11:43:10] "POST /upload-image HTTP/1.1" 302 -
127.0.0.1 - - [19/Oct/2022 11:43:10] "GET /upload-image HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2022 11:43:10] "GET /static/css/style.css HTTP/1.1" 304 -

```

3.2 Error Handling

```

@app.errorhandler(500)
def server_error(e):
    app.logger.error("server error")
    flash("error ! already exists")

    return redirect(request.url)

```

3.2 Persistent Storage

The website is using both MySQL database , mem-cache and filesystem for persistent storage , The file system is only used for storing images uploaded by the user .

MySQL database stores key, and photo data. **The diagram below** shows the schema of the database. The website application has a separate module for querying and updating the MySQL database , The mem-cache uses the database and it will be explained later.

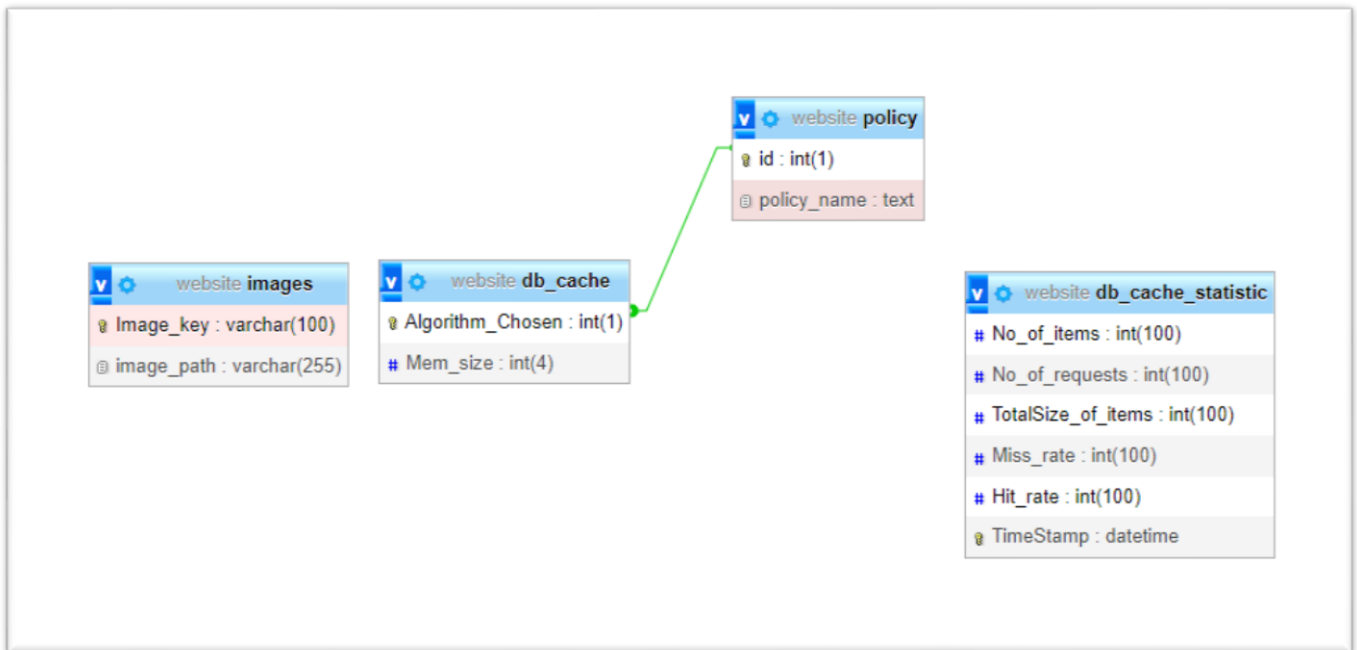


Figure 3.2-1 : Database schema.

- Connect Flask to a Database

```
from flask_mysql import MySQL

mysql = MySQL(app)
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = ''
app.config['MYSQL_DB'] = 'website'
```

Example to execute query get all key

```
@app.route("/keys",methods=["GET"])
def viewall():
    if request.method == 'GET':

        cur = mysql.connection.cursor()
        # column image_key from table images
        cur.execute(''SELECT `image_key` FROM `images` '')
        keys = cur.fetchall()

        return render_template("keys.html",keys = keys)
```

- The mem-cache uses the database

Example to execute query insert mem-cache statistic

```
# function used in mem-cache

def interval_task():
    with app.app_context():
        ct = datetime.datetime.now()
        cursor = mysql.connection.cursor()

        cursor.execute('' INSERT INTO db_cache_statistic VALUES(%s,%s,%s,%s,%s,%s,%s)'',
            (len(memory_cache),
            sys.getsizeof(memory_cache),
            requests[0],
            miss[0],
            hit[0],
            ct))

        mysql.connection.commit()
        cursor.close()
```

0.4 Mem - Cache

a mem-cache is an in-memory cache application implemented.

Starter code for a Flask-based mem-cache. It uses a global variable to store data in memory as a **Python dictionary** and includes two endpoints (get, put) that return json.

Figure 1 shows the workflow of a data **GET** request that is not on the mem-cache

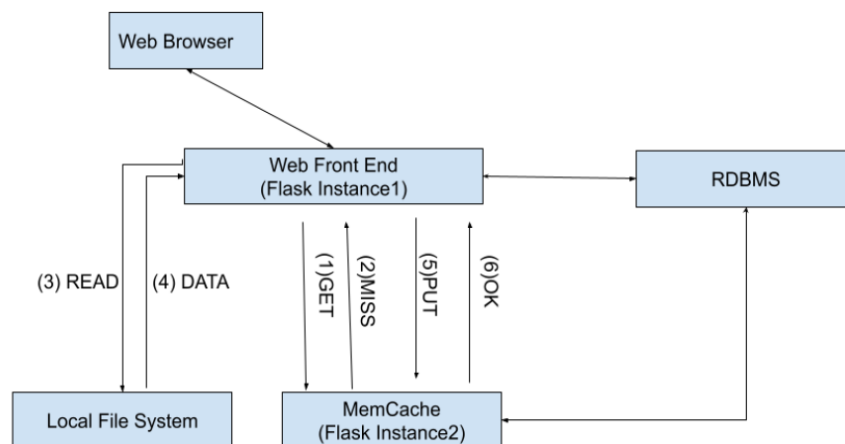


Fig 1. Workflow of a data GET request with a cache miss

Figure 2 shows the workflow of a PUT request:

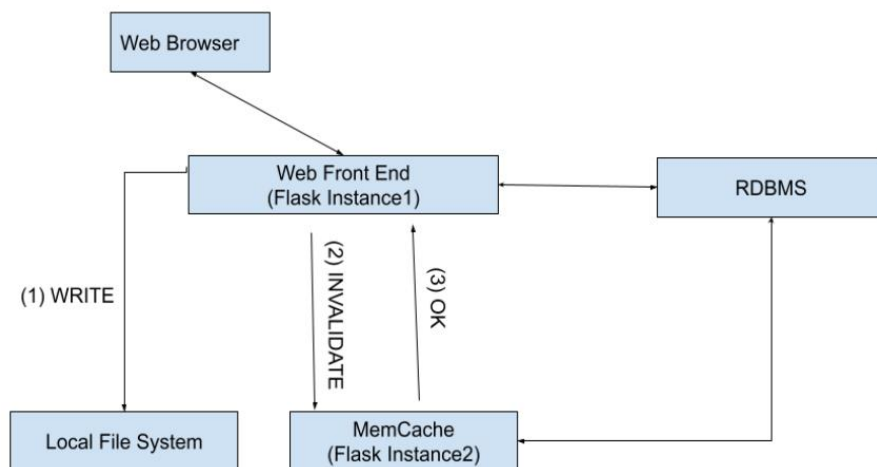


Fig 2. Workflow of a PUT request

LRU Cache in Python using OrderedDict

Time Complexity : $O(1)$

```
from collections import OrderedDict

class LRUCache:
    # initialising capacity
    def __init__(self, capacity: int):
        self.cache = OrderedDict()
        self.capacity = capacity

    # we return the value of the key
    # that is queried in  $O(1)$  and return -1 if we
    # don't find the key in our dict / cache.
    # And also move the key to the end
    # to show that it was recently used.
    def get(self, key: int) -> int:
        if key not in self.cache:
            return -1
        else:
            self.cache.move_to_end(key)
            return self.cache[key]

    # first, we add / update the key by conventional methods.
    # And also move the key to the end to show that it was recently used.
    # But here we will also check whether the length of our
    # ordered dictionary has exceeded our capacity,
    # If so we remove the first key (least recently used)
    def put(self, key: int, value: int) -> None:
        self.cache[key] = value
        self.cache.move_to_end(key)
        if len(self.cache) > self.capacity:
            self.cache.popitem(last = False)

# RUNNER
# initializing our cache with the capacity of 1 MB
cache = LRUCache(1)
```


Use **BackgroundScheduler** runs in a thread inside existing application

Job 1 : The mem-cache should store its statistics every 5 seconds.

Job 2 : Display a page with the current statistics for the mem-cache over the past 10 minutes

```
from apscheduler.schedulers.background import BackgroundScheduler

scheduler = BackgroundScheduler()
scheduler.start()

@app.before_first_request
def before_first_request():
    scheduler.add_job(func=interval_task, trigger="interval", seconds=5)
    scheduler.add_job(func=delay_Mins, trigger="interval", minutes=10)
```

The time taken get item from database **vs** mem-cache

```
C:\Users\Lenovo\Desktop\Cloud>flask run
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [19/Oct/2022 19:08:43] "POST /item HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2022 19:08:43] "GET /display/img.png HTTP/1.1" 302 -
127.0.0.1 - - [19/Oct/2022 19:08:43] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Oct/2022 19:08:43] "GET /static/images/img.png HTTP/1.1" 304 -
3.989696502685547 ms
127.0.0.1 - - [19/Oct/2022 19:09:17] "POST /item HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2022 19:09:17] "GET /display/bruh.gif HTTP/1.1" 302 -
127.0.0.1 - - [19/Oct/2022 19:09:17] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Oct/2022 19:09:17] "GET /static/images/bruh.gif HTTP/1.1" 304 -
127.0.0.1 - - [19/Oct/2022 19:09:18] "GET /display/bruh.gif HTTP/1.1" 302 -
127.0.0.1 - - [19/Oct/2022 19:09:18] "GET /static/images/bruh.gif HTTP/1.1" 200 -
0.0 ns
```