|  |
| --- |
| Cloud Project (1) |
|  |

Purpose:

The idea of the project is to create a storage web application to upload images and give a key to each of these images and relied on memory )key-value( cache to speed up the process of restoring images and displaying it , so that the image path is stored in the database and the image itself is stored in the cache, Some settings have been made in terms of adjusting the cache capacity, giving the ability to refresh the cache, and setting the replacement policy for new images on the cache which :

### A - Random Replacement

Randomly selects a key and discards it to make space when necessary. This algorithm does not require keeping any information about the access history.

**B - Least Recently Used**

Discards the least recently used keys first. This algorithm requires keeping track of what was used when, if one wants to make sure the algorithm always discards the least recently used key.

Store Image:

description: this route is used to add elements (key,image) to the cache and uploads folder and we save the key and path on database

input: key -> the key to the image we want to add

image -> the image we want to add

output: home page with image added successfuly message to notify that the process is done

@app.route('/get', methods =["POST"])

def get():

    image\_key = request.form.get("Key")

    cacheResult = cache.get(image\_key)

    if(cacheResult != None):

        flash(f'image for key {image\_key}')

        return render\_template("twocolumn1.html",image\_value= cacheResult.decode('utf-8'))

    cursor = db.cursor()

    cursor.execute(f'SELECT \* FROM key\_image WHERE image\_key = %s', (image\_key,))

    image\_p= cursor.fetchall()

    cursor.close()

    if image\_p:

        im = Image.open(os.path.join('static/uploads',image\_p[0][1]))

        im =im.convert('RGB')

        data = io.BytesIO()

        im.save(data, "JPEG")

        encoded\_img\_data = base64.b64encode(data.getvalue())

        cache.put(key= image\_key, image= encoded\_img\_data)

        flash(f'image for key {image\_key}')

        return render\_template("twocolumn1.html",image\_value= encoded\_img\_data.decode('utf-8'))

    else:

        flash('key doesn\'t exist !!')

        im = Image.open('static/images/notfound.png')

        im =im.convert('RGB')

        data = io.BytesIO()

        im.save(data, "JPEG")

        encoded\_img\_data = base64.b64encode(data.getvalue())

        return render\_template("twocolumn1.html",image\_value= encoded\_img\_data.decode('utf-8'))

Get Image:

description: this route is used to get image and show it on the html

page if it is on cache and if it wasn't we get it's path from database

and show it but if it doesn't exist the user will be notified

input: key -> the key to the image we want to get

output: html page contains the image if it exist else the user will be

notified

@app.route('/get', methods =["POST"])

def get():

    image\_key = request.form.get("Key")

    cacheResult = cache.get(image\_key)

    if(cacheResult != None):

        flash(f'image for key {image\_key}')

        return render\_template("twocolumn1.html",image\_value= cacheResult.decode('utf-8'))

    cursor = db.cursor()

    cursor.execute(f'SELECT \* FROM key\_image WHERE image\_key = %s', (image\_key,))

    image\_p= cursor.fetchall()

    cursor.close()

    if image\_p:

        im =Image.open(os.path.join('static/uploads',image\_p[0][1]))

        im =im.convert('RGB')

        data = io.BytesIO()

        im.save(data, "JPEG")

        encoded\_img\_data = base64.b64encode(data.getvalue())

        cache.put(key= image\_key, image= encoded\_img\_data)

        flash(f'image for key {image\_key}')

        return render\_template("twocolumn1.html",image\_value= encoded\_img\_data.decode('utf-8'))

    else:

        flash('key doesn\'t exist !!')

        im = Image.open('static/images/notfound.png')

        im =im.convert('RGB')

        data = io.BytesIO()

        im.save(data, "JPEG")

        encoded\_img\_data = base64.b64encode(data.getvalue())

        return render\_template("twocolumn1.html",image\_value= encoded\_img\_data.decode('utf-8'))

Get All Keys:

First Function :

description: this method is used to get items from array to

use it in pagination when showing the keys

input: keys -> list of elements we want to show in pagination

offset -> where to start getting the elements from the list

per\_page -> number of elements we want to show per page

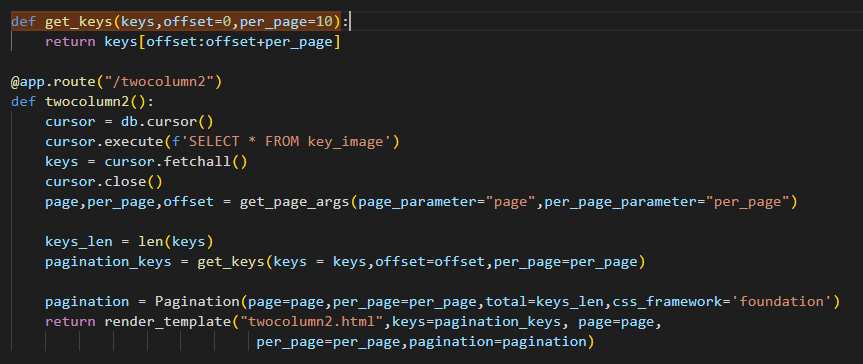
output: list of selected elements from the original list

Second Function :

description: this route is used to show keys page in pages

input: None

output: show keys html page



Statistics:

description: this route is used to get cache configuration page and show

its the cache configuration from database

input: None

output: cache configuration html page

@app.route("/threecolumn")

def threecolumn():

    cursor = db.cursor()

    cursor.execute(f'SELECT \* FROM cache ORDER BY created\_at DESC LIMIT 1')

    stats= cursor.fetchall()

    cursor.close()

    items = cache.count()

    requsts = cache.requsts

    size = cache.sizeMB()

    miss\_rate = cache.missRate()

    hit\_rate = cache.hitRate()

    if(stats != None and len(stats) != 0):

        items = stats[0][1]

        requsts = stats[0][2]

        size = stats[0][3]

        miss\_rate = stats[0][4]

        hit\_rate = stats[0][5]

    return render\_template("threecolumn.html", items=items, requsts=requsts, size=size, miss\_rate=miss\_rate, hit\_rate=hit\_rate)

Cache Configuration:

description: this route return the configuration page to show it and be able

to change cache size and clear its content when we want.

we get the configuration from database if it has been set else

we use default values

input: None

output: configuration html page

@app.route("/onecolumn")

def onecolumn():

    cursor = db.cursor()

    cursor.execute(f'SELECT \* FROM cache\_configuration')

    config = cursor.fetchall()

    cursor.close()

    policy = 0

    capacity = 2

    if(config is not None and len(config) != 0):

        capacity = config[0][0]

        policy = config[0][1]

    cursor = db.cursor()

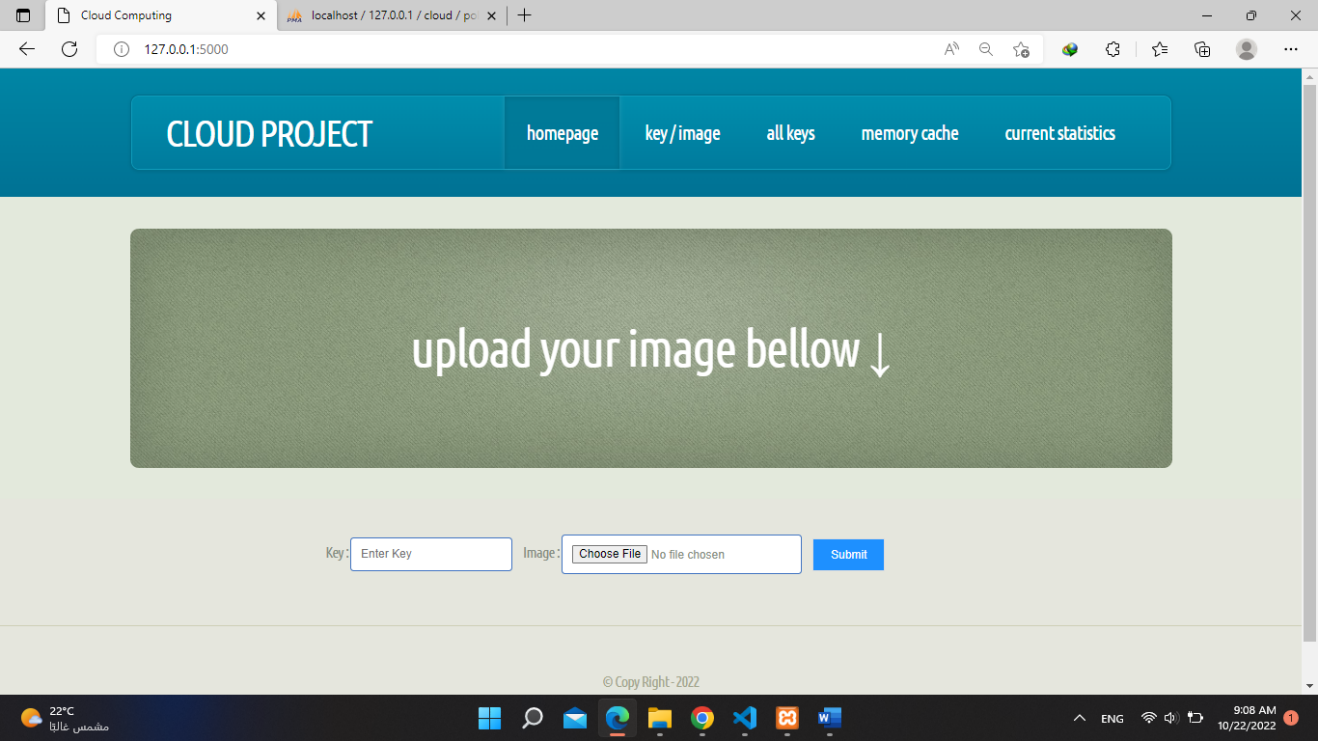
    cursor.execute(f'SELECT \* FROM policy\_type WHERE id = {policy}')

    policy = cursor.fetchall()[0][1]

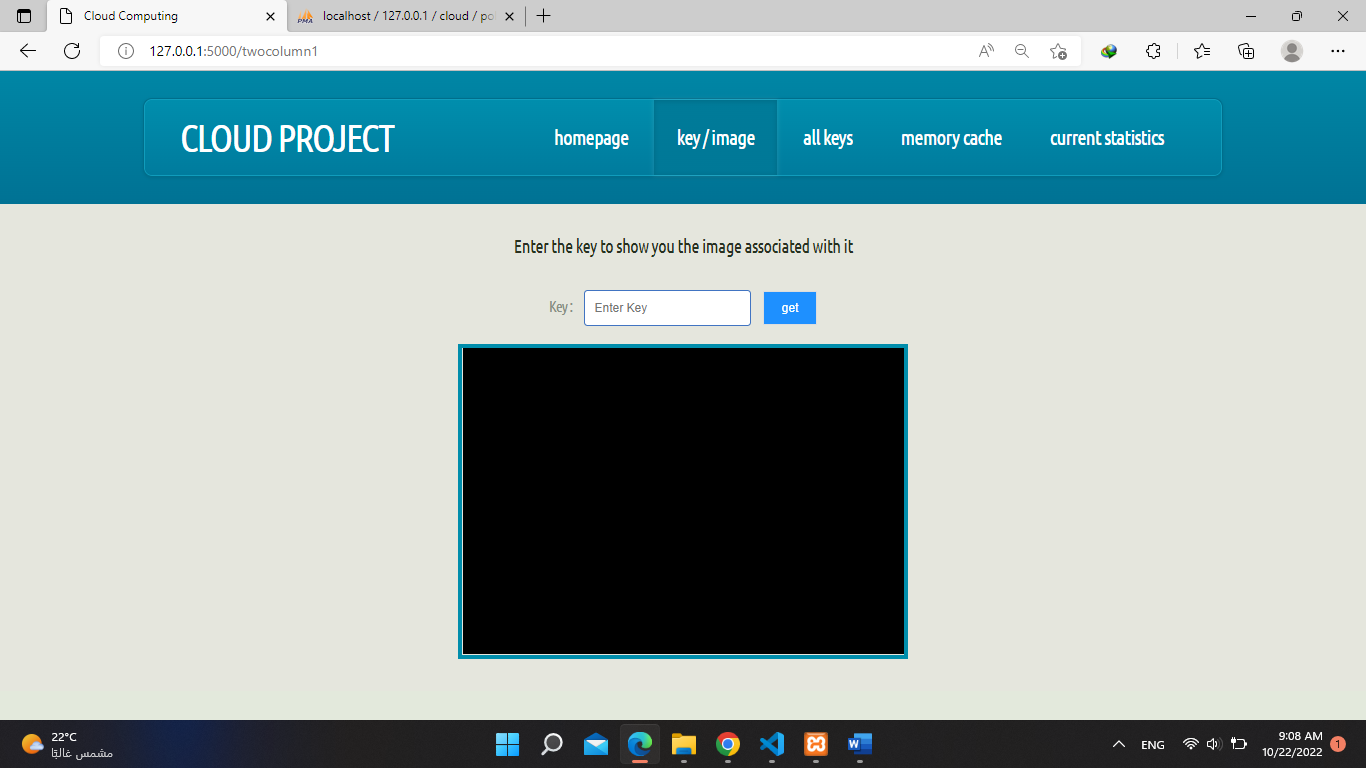
    cursor.close()

    return render\_template("onecolumn.html", policy = policy, capacity = capacity)

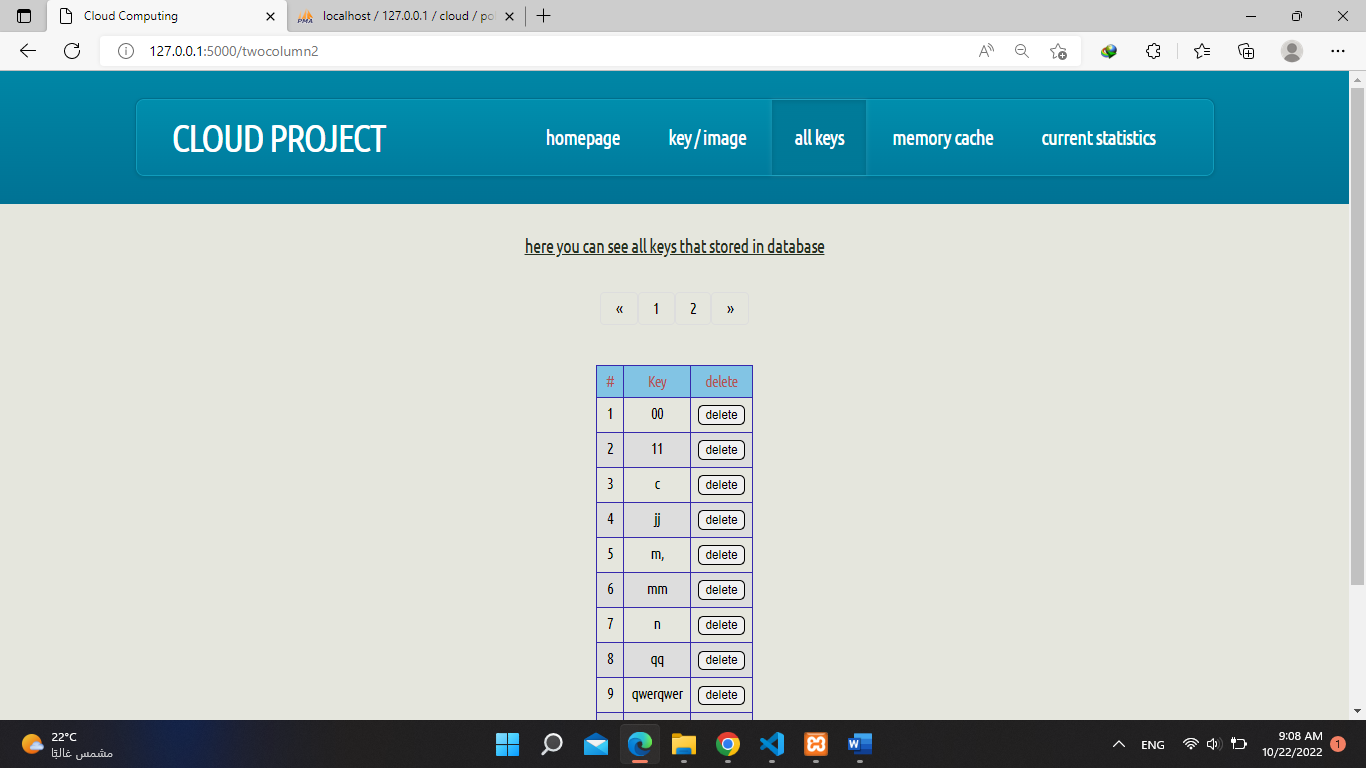
Pages documentation:



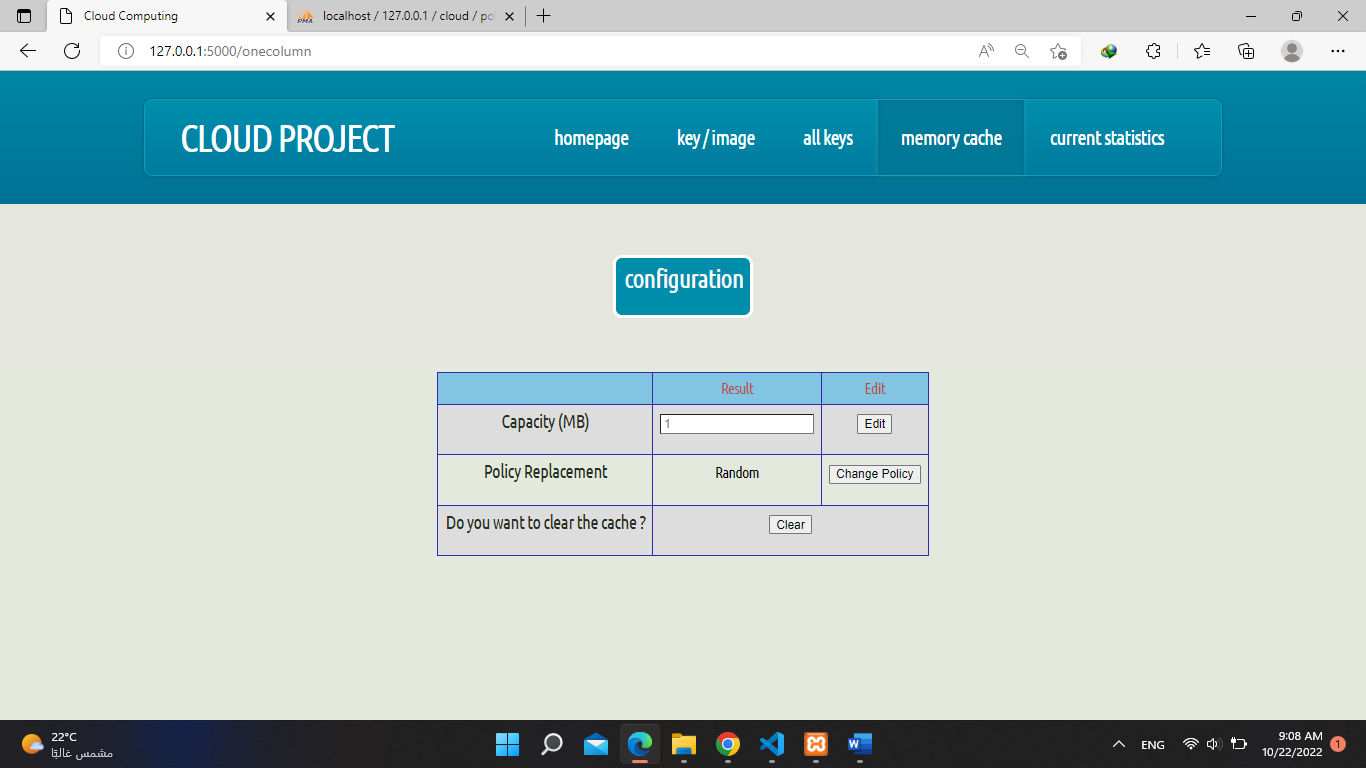
This page is the main page where you add the key and the image to store them in the cache and the database, you enter the key and upload your image and press submit to upload it.



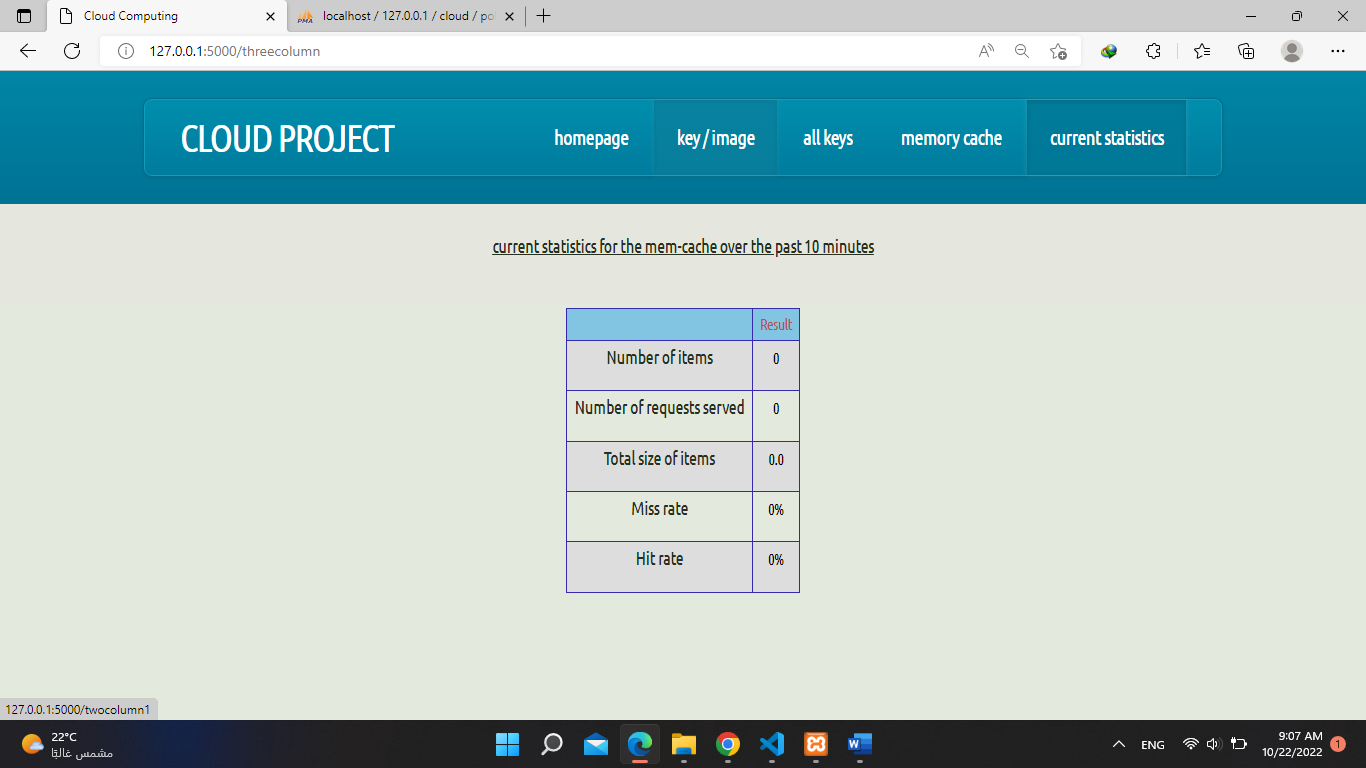
This page is where we get the images to display them, you can search for the image using it’s key and the result will be shown if the image exists else it will notify the user that it doesn’t exit



This page is where you can see all the keys stored in database, you can delete keys by pressing delete to the button on the right of the table



This page where you see the cache configuration and you can edit it’s size and you can change the deleting policy and you can clear the cache contents.



This page where you can see the statistics of the cache every 10 minutes ,

You can see number of items in the cache now, number of requests served, total size of items, miss rate and hit rate.