Chavara Public School
An Institution Run by CMI Fathers Pala
Affiliated to Central Board of Secondary Education, Delhi

PROJECT REPORT

Submitted in Fulfillment of Class XII Syllabus Requirement

in

COMPUTER SCIENCE (083)

Topic: Findr

CERTIFICATE

Certified that this is the bonafide record of project work carried out by

_____ Roll No. _____

of Class XII of this school during the year 2024-2025.

Teacher-in-Charge Principal Examiner

Findr

Syon Vijae T
Karthik Bilukumar
Subramanian R
Rithul Nambiar M

Acknowledgement

Apart from the efforts of me, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I express deep sense of gratitude to almighty God for giving me strength for the successful completion of the project.
I express my deep sense of gratitude to the luminary The Principal, Fr. SABU KOODAPPATTU CMI who has been continuously motivating and extending their helping hand to us.

I am extremely grateful to Mr. PRAJEESH M PRABHA, Teacher of Department of Computer Science for his able guidance and useful suggestions, which helped me in completing the project work, in time.

I would also like to thank the School management and all the teaching and non-teaching staff of Chavara Public School who helped me directly or indirectly in the completion of this project. Finally, yet importantly, I would like to express my heartfelt thanks to my beloved parents for their blessings, my friends/group mates for their help and wishes for the successful completion of this project.

The guidance and support received from all the members who contributed and who are contributing to this project, was vital for the success of the project. I am grateful for their constant support and help.

Contents

Introduction

In today's digital age, technology plays a pivotal role in shaping our daily lives and enhancing our capabilities. As part of our Class 12 computer science curriculum, this project aims to explore the practical applications of programming and data management.

The primary goal of this project is to design and implement a feature-rich website where users can post advertisements for items they wish to sell and buy products they are interested in purchasing. By leveraging tools such as Python, SQL, HTML, CSS, Flask and Jinja we will create a responsive and intuitive user interface that enhances the overall user experience.

Key features of the website will include user sign up, secure orders between buyers and sellers, the ability to search for items , and the ability to upload images. Additionally, we will focus on ensuring the security and privacy of user data while maintaining an engaging and straightforward design.

Through this project, we aim to not only gain practical experience in web development but also address the growing demand for convenient online shopping solutions. By combining functionality with an appealing design, we hope to provide users with an efficient platform for buying and selling a variety of goods.

Modules

A module is a file consisting of Python code. It can define functions, classes, and variables and can also include runnable code. Any Python file can be referenced as a module. We have used the following modules in our program

- ***PICKLE*** : Pickle in Python is primarily used in serializing and deserializing a Python object structure. Here it is used for storing the shopping cart in the MySQL tables.

- ***MYSQL CONNECTOR*** : The MySQL module provides the connection between Python and MySQL

- ***Base64*** : Load images from database. Converts binary data into a string of printable ASCII characters. Used to store and display images from the MySQL database

- ***HASHLIB*** : Standard sha256 hashing for password. Provides a helper function for efficient hashing of a file or file-like object. Keeps the users passwords safe in case of a database breach

- ***LOGGING*** : Error messages in console. Used to record information about errors, warnings, and other events that occur during program execution. Only used for development and testing

Functions

- **`init()`**

Establishes connection between python and MySQL.

- **`getcol(col,table)`**

A required column from the given table is returned as a list.

- **`getrow(table,cond,val)`**

A required row from the given table with a given condition, is returned as a list.

- **`getone(col,table,cond,value)`**

Returns a single value from a column in the given with a given condition.

- **`clmheads(table)`**

The column heads of the given table is returned as a list.

- **`increment(col,table)`**

The primary key of successive row is increased by '001' in serialized order.

- **`addcomment(pid,uname,caption)`**

It is used for adding comments for the displayed product.

- **`edit(uname,newdata)`**

It is used for updating or editing userdata.

- **`postcount(uname)`**

It returns us the number of posts uploaded by a user.

- **`signup(name,uname,descrip,email,pwd)`**

It is used for creating a new user account.

- **`checklike(uname,pid)`**

It checks if a post is liked or not.

- **`addlike(uname,pid)`**

It adds a like to a post.

- **`remlike(uname,pid)`**

It removes like, given to a post.

- **`likecount(pid)`**

It returns the number of likes for a post.

- **`getcart(uname)`**

It displays the contents of the cart of the user.

- **`addtocart(uname,pid)`**

It updates serialized cart after addition of product.

- **`remfromcart(uname,pid)`**

It updates serialized cart after removal of product.

- **`clearcart(uname)`**

It clears the cart of the user.

- **`upload(uname,cap,price,img)`**

It is used for uploading the product as a post.

- **`sendrequests(pidlist, uname)`**

It is used to send request for a product.

- **`getrequests(shop)`**

It gets all current requests.

- **`setstatus(rid, status)`**

It sets request status or deletes requests.

- **`banuser(uname)`**

It is used to ban a particular user.

- **`delpost(pid)`**

It is used to delete a post.

Source Code

```
# ------------------------------------------------------------------------#
#                                Findr: The app                           #
# ------------------------------------------------------------------------#
from flask import Flask, render_template, abort, request, redirect, url_for,
 session, flash # web app functionality
from base64 import b64encode # load images from db
from logging import error # error msgs in console
import functions as sql # custom functions from function.py
from pickle import dumps # serializes cart for storage
from hashlib import sha256  # standard sha256 hashing for password
from flask_ipban import IpBan

# ------------------------ initialize flask and mysal --------------------#
sql.init()
app = Flask(__name__, template_folder='blocktemplates', static_folder='static')
app.secret_key = "key"

#Ip Ban
bannedips = ['192.168.1.55', '192.168.1.153']
ip_ban = IpBan(ban_seconds=200)
ip_ban.init_app(app)
ip_ban.block(ip_list=bannedips, permanent=True)

# --------------------------- logout and init page ----------------------#
@app.route("/", methods=['POST', 'GET'])
def init():
    session.clear()
    session['loggedin'] = False
    session['uname'] = 'Guest'
    return redirect(url_for('home'))
```

```
# ------------------------------ signin page --------------------------#
@app.route("/signin", methods=['GET', 'POST'])
def signin():

    if session.get('loggedin'):
        return redirect(url_for('home'))

    else:
        if request.method == 'POST':
            data = request.form
            session['email'] = data.get('email') # stores email in app wide
 session dict

            # check if account exists
            if not session.get('email') in sql.getCol('email', 'userdata'):
                flash('There is no account linked with that email', 'error')
                return redirect(url_for('signup'))

            else:
                # checks hashed pwd with stored hashed pwd
                if sha256(data.get('password').encode()).hexdigest() ==
sql.getOne('pwd',
 'userdata', 'email', data['email']):
                    session['loggedin'] = True
                    session['uname'] = sql.getOne('uname', 'userdata',
 'email', session.get('email'))
                    return redirect(url_for('home'))
                else:
                    flash("Incorrect Password! Try Again", 'error')

    return render_template('signin.html',
                           loggedin=session.get("loggedin"))
```

```
# ------------------------------- signup page --------------------------#
@app.route("/signup", methods=['GET', 'POST'])
def signup():

    if session.get('loggedin'):
        return redirect(url_for('home'))

    else:
        if request.method == 'POST':

            data = request.form
            session['uname'] = data.get('uname')
            session['email'] = data.get('email')


            # check for dupe email
            if session.get('email') in sql.getCol('email', 'userdata'):
                flash("There is already an account linked to that email", 'error')
                return redirect(url_for('signin'))

            # check for dupe username
            elif session.get('uname') in sql.getCol('uname', 'userdata'):
                    flash("Username already taken", 'error')

            else:
                # hash and save pwd and account
                try:
                    error(request.form)
                    password = sha256(data.get('password').encode()).hexdigest()
                    sql.signup(data.get('name'), session.get('uname'),
 data.get('email'), password)
                    session['loggedin'] = True
                    return redirect(url_for('home'))

                except Exception as e:
                    error(e)

        return render_template('signup.html',
                               loggedin=session.get("loggedin"),
                               email = session.get('email'))
```

```
# ---------------------------------- home page ----------------------------#
@app.route("/home", methods=['GET', 'POST'])
def home():
    filt = None
    if request.method == 'POST':
        filt = request.form.get('search')

    return render_template('home.html',
                           loggedin=session.get("loggedin"),
                           name=sql.getOne('name','userdata', 'uname',
 session.get('uname')),
                           products=sql.all(filt),
                           b64encode=b64encode,
                           seller=lambda uname: sql.getOne('name', 'userdata',
 'uname', uname),
                           postno=sql.postcount(session.get('uname')),
                           filter=filt)

# ------------------------------- upload page --------------------------#
@app.route("/upload", methods=['GET', 'POST'])
def upload():

    if session.get('loggedin'):
        if request.method == 'POST':
            data = request.form
            img = request.files['img'].read()

            try:
                sql.upload(session.get('uname'), data.get('caption'),
data.get('price')  ,img)
                flash('Uploaded Succesfully', 'success')

            except Exception as e:
                flash('Upload Failed', 'error')
        return render_template('upload.html',
                               loggedin=session.get("loggedin"))

    else:
        flash("You need to be Logged in to perform this action", 'error')
        return redirect(url_for('signin'))
```

```python
# ------------------------------ product page --------------------------#
@app.route("/<pid>", methods=['GET', 'POST'])
def productpage(pid):


    isliked = sql.checklike(session.get('uname'), pid)

    if pid in sql.getcart(session.get('uname')):
        incart = True
    else:
        incart = False

    if pid not in sql.getCol('pid' ,'productdata'):
        abort(404)
    data = sql.getRow('productdata', 'pid', pid)[0]

    if request.method == 'POST':
        if not session.get('loggedin'):
            flash("You need to be Logged in to perform this action", 'error')
            return redirect(url_for("signin"))

        if request.form.get('comment'):
            sql.addComment(pid, session.get('uname'), request.form.get('comment'))

        elif request.form.get('isliked'):
            if request.form.get('isliked') == "True":
                sql.addlike(session.get('uname'), pid)
                isliked = True

            elif request.form.get('isliked') == "False":
                isliked = False
                sql.remlike(session.get('uname'), pid)

        elif request.form.get('incart'):
            if request.form.get('incart') == 'True':
                sql.addtocart(session.get('uname'), pid)
                return redirect(url_for("cart"))

            elif request.form.get('incart') == 'False':
                sql.remfromcart(session.get('uname'), pid)
                return redirect(url_for('productpage',pid=pid))

        else:
            error("not getting in loop")

    try:
        return render_template('productpage.html',
                               loggedin=session.get("loggedin"),
                               img=b64encode(data[3]).decode('utf-8'),
                               title=data[2],
                               seller=sql.getOne('name', 'userdata', 'uname', data[1]).title(),
                               price=data[4],
                               comments=sql.getRow('commentdata', 'pid', pid),
                               pid=pid,
                               liked=isliked,
                               likecount = sql.likecount(pid),
                               incart=incart)
    except TypeError:
        abort(404)
```

```python
# ------------------------------ edit profile --------------------------#
@app.route("/edit", methods=['GET', 'POST'])
def edit():
    if session.get('loggedin'):

        data = sql.getRow('userdata', 'uname', session.get('uname'))[0][0:5]
        headers = sql.clmnheads('userdata')[0:5]
        if request.method == 'POST':
            try:
                data = [
                    request.form.get('NAME'),
                    session.get('uname'),
                    request.form.get('DECRIP'),
                    request.form.get('EMAIL'),
                    sha256(request.form.get('PWD').encode()).hexdigest(),
                    0,
                    dumps([])]


                if data[4]=="e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855":
                    data[4] = sql.getOne('pwd', 'userdata', 'uname', session.get('uname '))

                sql.edit(session.get('uname'), data)

                flash("Profile edited successfully", 'success')
            except Exception as e:
                error(e)
                flash(f"Process Failed {e}", 'error')
        return render_template('edit.html',
                               loggedin=session.get("loggedin"),
                               dictionary=dict(zip(headers, data)),
                               name=sql.getOne('name','userdata', 'uname', session.get('uname')))

    else:
        flash("You need to be Logged in to perform this action", 'error')
        return redirect(url_for('signin'))
```

```
# ---------------------------------- cart -------------------------------#
@app.route("/cart", methods=['GET', 'POST'])
def cart():
    if session.get('loggedin'):

        cart = []
        total = 0

        for product in sql.getcart(session.get('uname')):
            cart.append(sql.getRow('productdata', 'pid', product)[0])

        for item in cart:
            total+=item[4]

        if request.method == 'POST':
            if request.form.get('buy'):
                sql.sendrequests(sql.getcart(session.get('uname')), session.get('uname'))
                sql.clearcart(session.get('uname'))
                cart = []

            elif request.form.get('delete'):
                sql.remfromcart(session.get('uname'),
request.form.get('delete'))

                for item in cart:
                    if item[0] == request.form.get('delete'):
                        cart.remove(item)


        return render_template('cart.html',
                               loggedin=session.get("loggedin"),
                               details=cart,
                               b64encode=b64encode,
                               name=sql.getOne('name','userdata', 'uname',
session.get('uname')),
                               total=total)
    else:
        flash("You need to be Logged in to perform this action", 'error')
        return redirect(url_for('signin'))
```

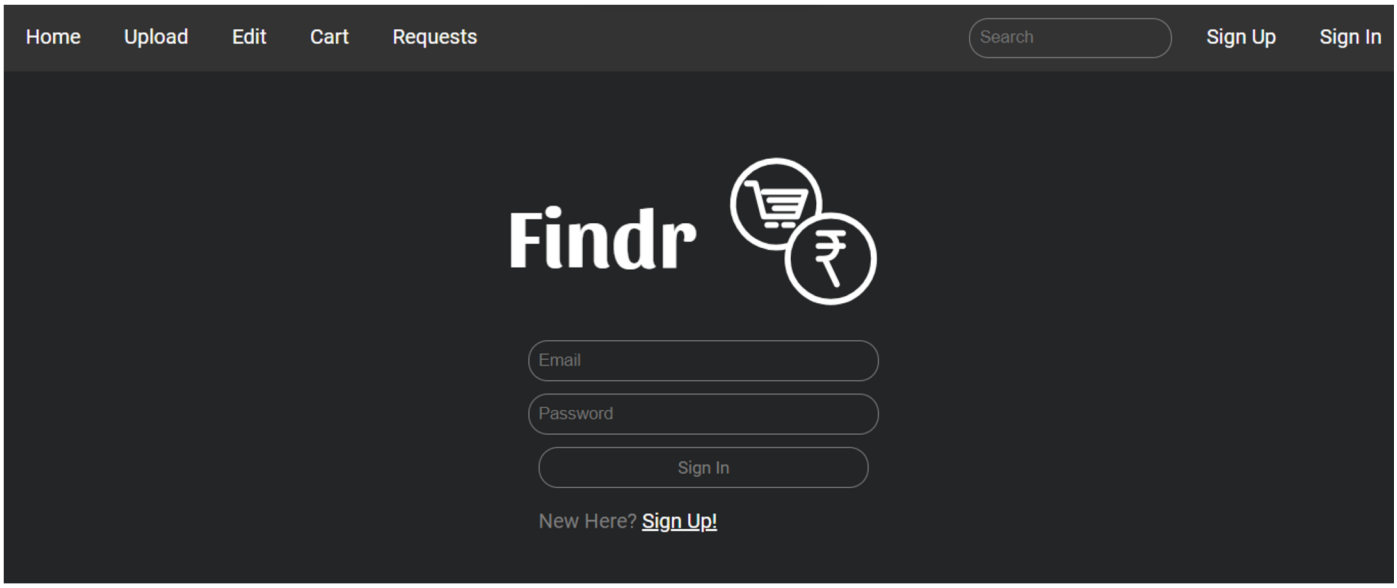
```python
# ------------------------------- requests ----------------------------#
@app.route("/requests", methods=['GET', 'POST'])
def requests():
    if session.get('loggedin'):

        all = sql.getrequests(session.get('uname'))
        requests = []
        for req in all:
            requests.append([req[::2]]+sql.getRow('productdata', 'pid', req[3]))


        if request.method == 'POST':
            if request.form.get('status'):
                status = request.form.get('status').split(',')
                sql.setstatus(status[1], status[0])


        return render_template('requests.html',
                               loggedin=session.get("loggedin"),
                               requests=requests,
                               b64encode=b64encode)

    else:
        flash("You need to be Logged in to perform this action", 'error')
        return redirect(url_for('signin'))


# ------------------------------ Admin Page ------------------------------ #
@app.route("/admin", methods=['GET','POST'])
def admin():
    global bannedips
    if request.method == 'POST':
        if request.form.get('banip'):
            bannedips = request.form.get('banip').split('\r\n')

        if request.form.get('user'):
            sql.banuser(request.form.get('user'))
            flash('User Deleted', 'success')

        if request.form.get('product'):
            sql.delpost(request.form.get('product'))
            flash('Product Deleted', 'success')

    error(bannedips)
    return render_template('admin.html',
                           loggedin=session.get('loggedin'),
                           bannedips='\n'.join(bannedips),
                           uname=session.get('uname'))


# ---------------------------------- 404 ----------------------------------#
@app.errorhandler(404)
def not_found(e):
    return f"Page Not Found!", 404
# ---------------------------------- 403 ----------------------------------#
@app.errorhandler(403)
def not_found(e):
    return f"Go away bad man", 403
if __name__ == '__main__':
    app.run()
```

```
from mysql.connector import connect
from pickle import dumps, loads
from logging import error

#initializes the mysql connector
def init():
        global sql
        global cur
        global uname
        sql = connect(host='localhost', user = 'root', password='1234',
database='syon_project')
        cur = sql.cursor(buffered=True)
        sql.autocommit = True
init()

# ---------------------------------SQL FUNCTIONS ----------------------------#
#returns required column as list
def getCol(col, table):
        cur.execute(f'SELECT {col} from {table}')
        l = []
        for row in cur.fetchall():
                l.append(row[0])
        return l
# ------------------------------------------------------------------------- #

#returns required row as list
def getRow(table, cond, val):
        cur.execute(f'SELECT * FROM {table} WHERE {cond}="{val}";')
        return list(cur.fetchall())
# ------------------------------------------------------------------------- #

#returns single value
def getOne(col, table, cond, val):
        cur.execute(f'SELECT {col} from {table} where {cond}="{val}"')
        return cur.fetchone()[0]
# ------------------------------------------------------------------------- #

#returns column heads as list
def clmnheads(table):
        cur.execute(f"SELECT * FROM {table} where uname='1'")
        return [i[0] for i in cur.description]
# ------------------------------------------------------------------------- #

#returns all items
def all(filter):
        if not filter:
                cur.execute("SELECT * from ProductData")
                return cur.fetchall()
        else:
                cur.execute(f"SELECT * FROM ProductData WHERE CAPTION LIKE
 '%{filter}%' ")
                return cur.fetchall()
# ------------------------------------------------------------------------- #

#autoincrement for primary keys
def increment(col, table):
        cur.execute(f'SELECT {col} from {table};')
        try:
                last = (cur.fetchall()[-1][0])
                return str(last)[0]+f"{int(str(last)[1:])+1 :03d}"
```

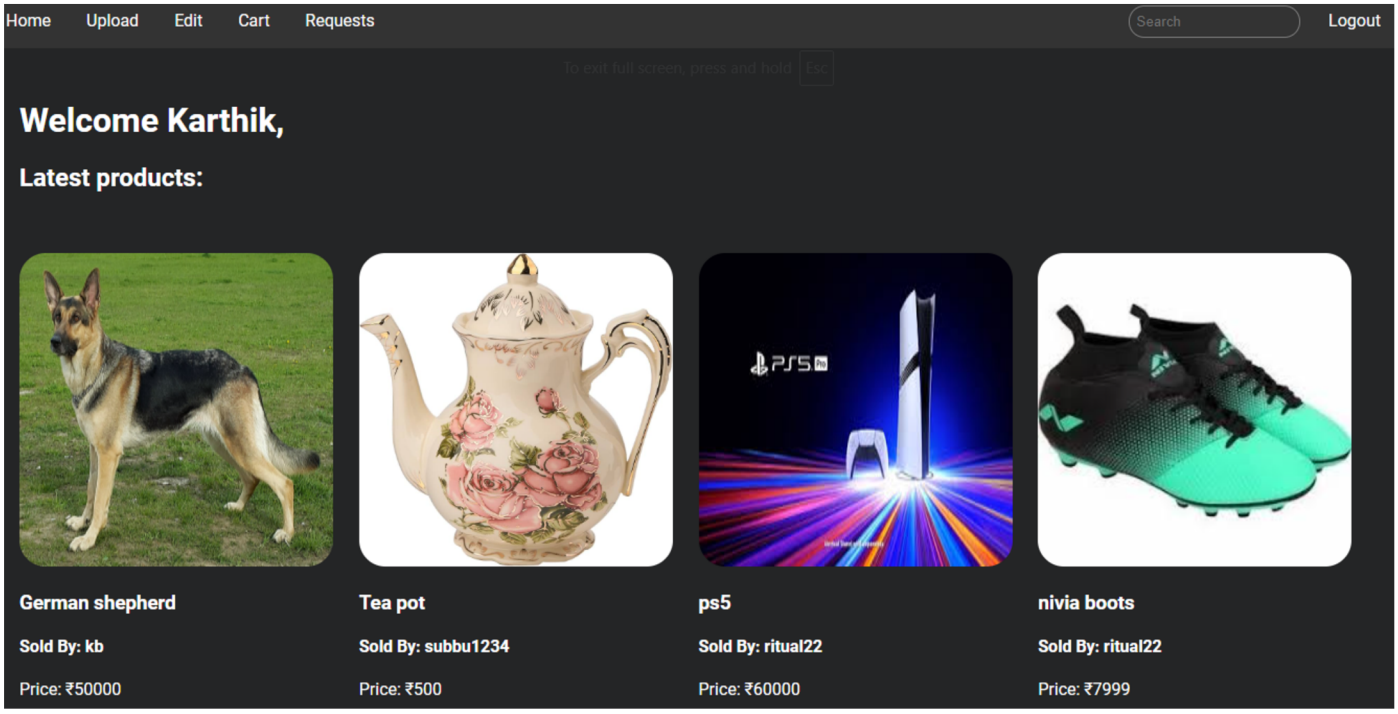
```
# ------------------------------Findr FUNCTIONS-----------------------------#

def addComment(pid, uname, caption): adds comment
        cur.execute('INSERT INTO CommentData VALUES(%s, %s, %s, %s)', (increment('cid', 'commentdata'), pid,
uname, caption))
# ------------------------------------------------------------------------- #


def upload(uname, cap, price, img): #uploads posts
        cur.execute("INSERT INTO ProductData VALUES(%s, %s, %s, %s, %s)", (increment('pid', 'productdata'),
uname, cap, img, price))
        cur.execute("UPDATE UserData SET POSTCOUNT = POSTCOUNT+1 WHERE uname=%s", (uname,))
# ------------------------------------------------------------------------- #


def edit(uname, newdata): #edits userdata
        sql.autocommit = False
        try:
                cur.execute('delete from userdata where uname=%s', (uname,))
                cur.execute('insert into userdata values(%s, %s, %s, %s, %s, %s, %s)', newdata)
        except Exception as e:
                sql.rollback()

        sql.commit()
        sql.autocommit = True
# ------------------------------------------------------------------------- #

def postcount(uname):#returns postcount
        c=0
        for product in all(None):
                if product[1].lower() == uname.lower():
                        c+=1
        return c
# ------------------------------------------------------------------------- #

def signup(name, uname, email, pwd):#creates user
        cur.execute("INSERT INTO UserData VALUES(%s, %s, %s, %s, %s, %s)", (name, uname, email, pwd, 0,
dumps([])))
# ----------------------------- LIKE FUNCTIONS ----------------------------- #

def checklike(uname, pid):#checks if liked
        if pid in getCol('pid', 'LikeData') and getRow('LikeData', 'pid', pid)[0][1] == uname:
                return True
        return False
# ------------------------------------------------------------------------- #

def addlike(uname, pid):#adds Like
        cur.execute("SELECT * FROM LikeData WHERE PID=%s AND UNAME=%s", (pid, uname))
        if not cur.fetchall():
                cur.execute("INSERT INTO LikeData VALUES(%s, %s)", (pid, uname))
# ------------------------------------------------------------------------- #

def remlike(uname, pid):#removes like
        cur.execute("DELETE FROM LikeData WHERE PID=%s AND UNAME=%s", (pid, uname))
# ------------------------------------------------------------------------- #

def likecount(pid):#returns like count
        cur.execute('SELECT * FROM LikeData WHERE PID=%s', (pid,))
        return len(cur.fetchall())
```

```
# ---------------------------- CART FUNCTIONS ---------------------------- #

#returns deserialized cart
def getcart(uname):
        return loads(getOne('CART', 'UserData', 'uname', uname))
# ------------------------------------------------------------------------ #

#updates serialized cart after addition of product
def addtocart(uname, pid):
        cart = getcart(uname)
        cart.append(pid)
        cur.execute("UPDATE UserData SET CART=%s WHERE UNAME=%s", (dumps(cart), uname))
# ------------------------------------------------------------------------ #

#updates serialized cart after removal of product
def remfromcart(uname, pid):
        cart = getcart(uname)
        cart.remove(pid)
        cur.execute("UPDATE UserData SET CART=%s WHERE UNAME=%s", (dumps(cart), uname))
# ------------------------------------------------------------------------ #

#clears cart
def clearcart(uname):
        cur.execute("UPDATE UserData SET CART = %s WHERE UNAME = %s", (dumps([]), uname))
# ---------------------------------- . ---------------------------------- #


# ---------------------------- REQUEST FUNCTIONS ---------------------------- #

#sends requests
def sendrequests(pidlist, uname):
        for pid in pidlist:
                cur.execute("INSERT INTO RequestData VALUES(%s, %s, %s, %s, %s)",
(increment('RID', "RequestData"), getOne('uname','productdata', 'pid', pid), uname, pid, 'Pending'))
# ------------------------------------------------------------------------ #

#gets all current requests
def getrequests(shop):
        cur.execute("SELECT * FROM RequestData WHERE SELLER=%s", (shop,))
        return list(cur.fetchall())
# ------------------------------------------------------------------------ #

#sets request status/deletes requests
def setstatus(rid, status):
        if status == "Delete":
                cur.execute("DELETE FROM RequestData WHERE RID=%s", (rid,))
        else:
                cur.execute("UPDATE RequestData SET STATUS=%s WHERE RID=%s", (status, rid))
# ---------------------------------- . ---------------------------------- #

# ---------------------------- Admin Functions ---------------------------- #

def banuser(uname):
        cur.execute("DELETE FROM UserData WHERE UNAME=%s", (uname,))

def delpost(pid):
        cur.execute("DELETE FROM ProductData WHERE PID=%s", (pid,))
```

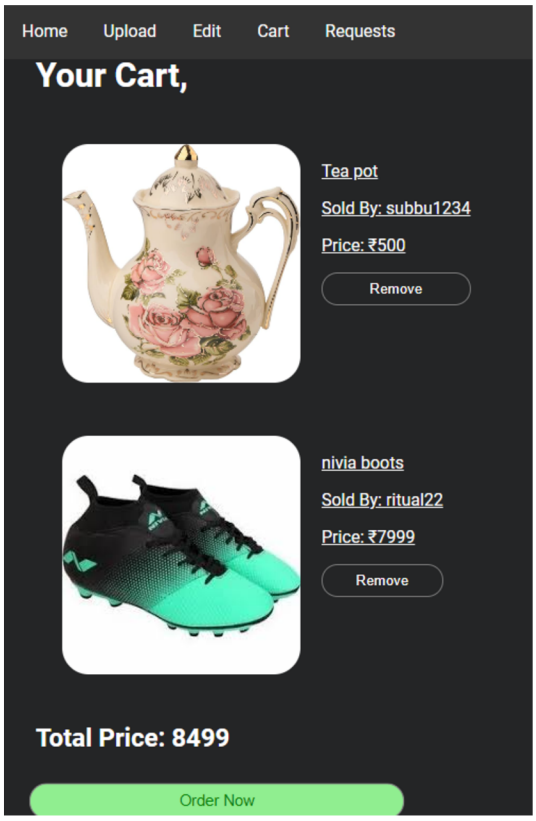
Output



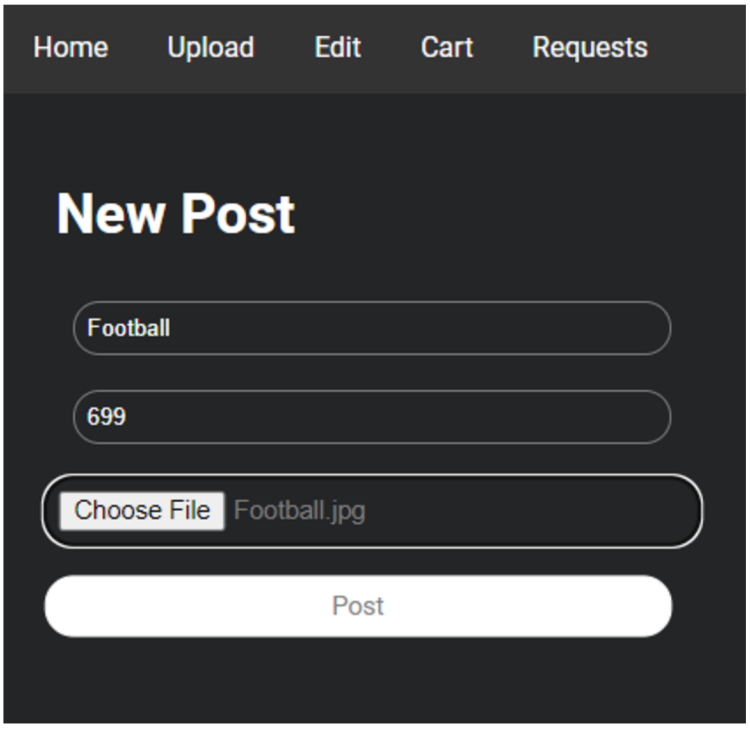
Sign In Page



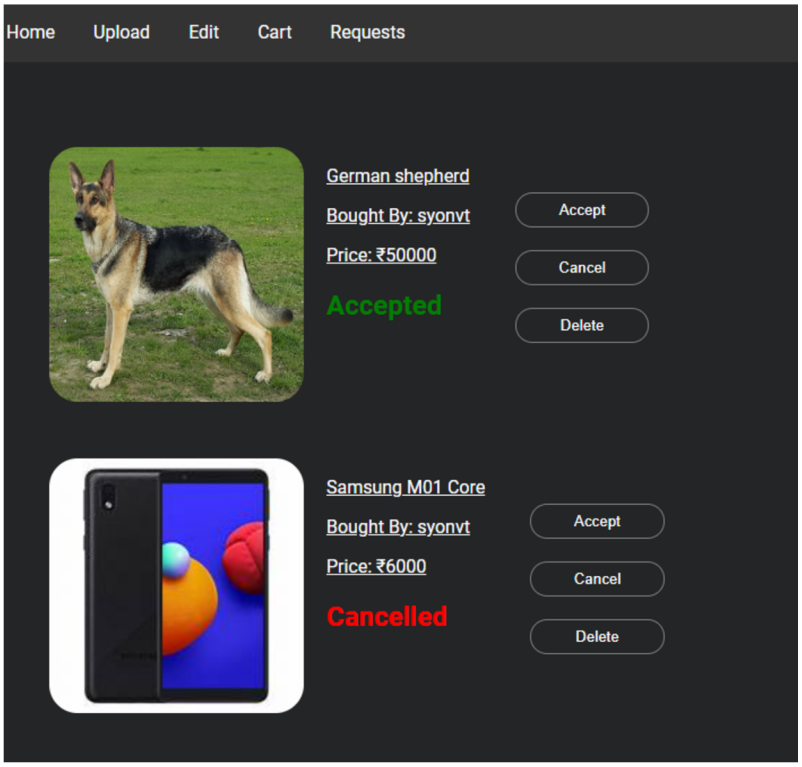
Home Page



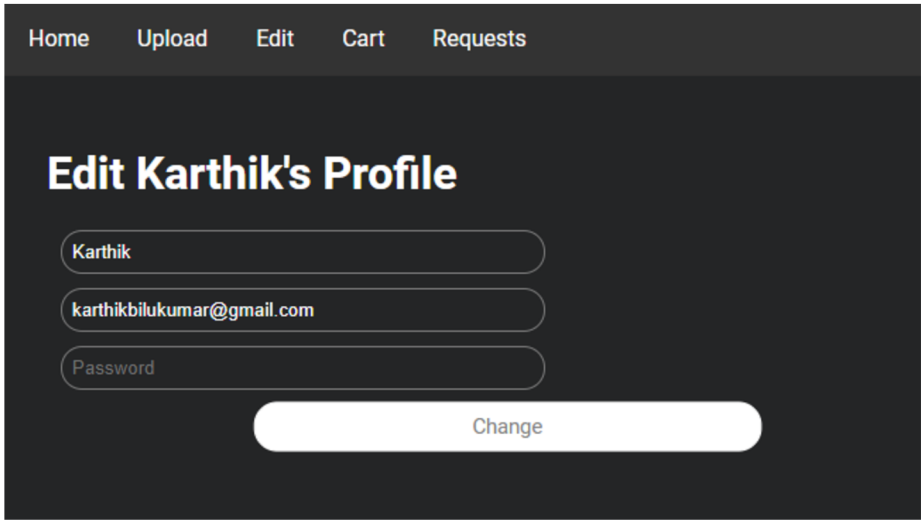
Cart



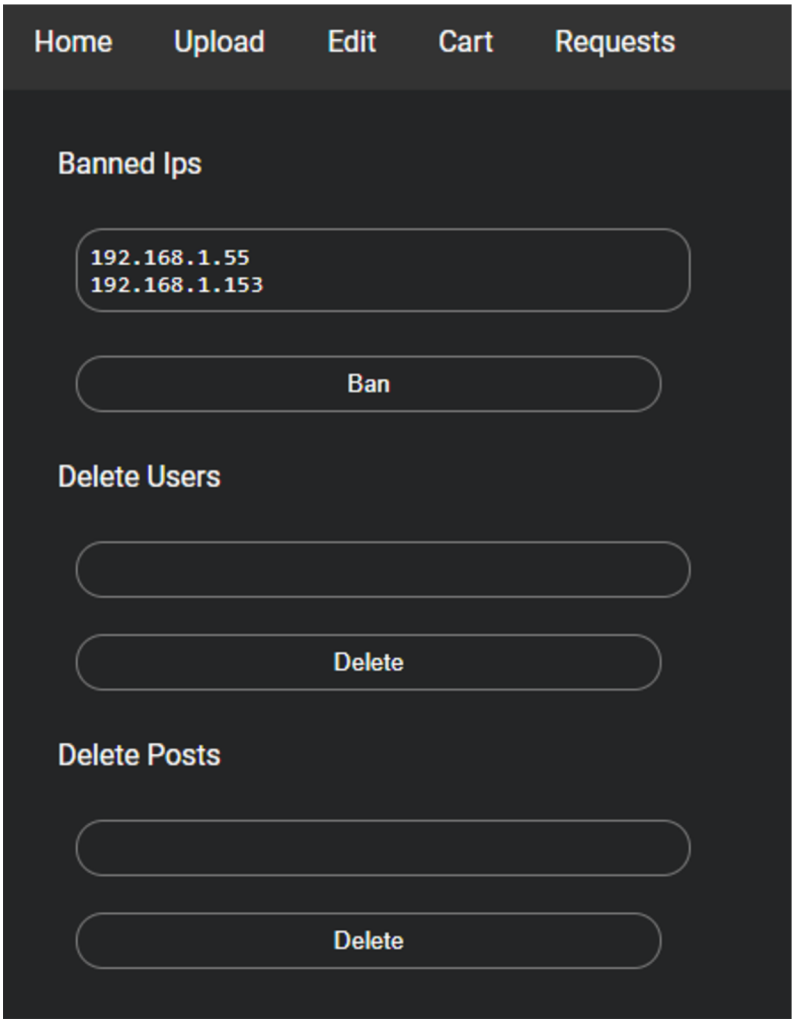
New Post



Request Page



Edit Profile



Admin Page

Bibliography

- **python.org**
- **geeksforgeeks.org**
- **w3schools.com**
- **palletsprojects.com/projects/flask**
- **palletsprojects.com/projects/jinja**


Thank
you!