

**Atypical:Stock Image Site**

**MINOR PROJECT - II**

**Submitted in Partial Fulfilment of the Requirements for the Award of the Degree of**

**MASTER OF COMPUTER APPLICATIONS**

**(M.C.A.)**

**BY**

**Vaibhav Baweja**

**(35335304421)**

**MCA – 2nd SEMESTER**



**BHARATI VIDYAPEETH’S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT (BVICAM)**

**(GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY)**

**NEW DELHI**

**July, 20****22**

**CANDIDATE’S DECLARATION**

I hereby declare that the work which is being presented in this project work entitled “**Atypical:Stock Image site**” in partial fulfilment of the requirements for the award of the degree of **Master in Computer Applications at Bharati Vidyapeeth’s Institute of Computer Applications and Management (BVICAM), New Delhi** is an authentic record of my own work carried out during the period January 2021 to April 2022 under the supervision and guidance of **Dr Sunil Pratap Singh(Associate Professor, BVICAM)**.

I have not submitted the matter embodied in this project work anywhere for the award of any degree or diploma.

**Vaibhav Baweja**

Enroll. No. 35335304421

**ACKNOWLEDGEMENT**

It is my proud privilege to express my profound gratitude to the entire management of Bharati Vidyapeeth’s Institute of Computer Applications and Management and teachers of the institute for providing me with the opportunity to avail the excellent facilities and infrastructure. The knowledge and values inculcated have proved to be of immense help at the very start of my career. Special thanks to Hon’ble Founder, Bharati Vidyapeeth, Pune for having provided us an excellent infrastructure at BVICAM.

I am grateful to **Dr Sunil Pratap Singh** (Project Guide, BVICAM, New Delhi) for his astute guidance, constant encouragement and sincere support for this project work.

I feel pride and privileged in expressing my deep sense of gratitude to all those who have helped me in presenting this assignment. I would be failing in my endeavour if I do not place my acknowledgement**.**

**Vaibhav Baweja**

Enroll. No. 35335304421

**Abstract**

Atypical is a Web App where Users can download pictures publically for free without any hassle. It’s made for people like Graphic Designers, Artists, Content Creators who can search images which need in their work and can download them for free. Images can be posted by anyone after creating an account on Atypical by Signing up. It’s clean and elegant design helps people to use it conveniently. People can find images by going through the Categories assigned right after the image is uploaded to the server. It’s responsiveness enables it to be platform independent which simply means Users can conviniently access the whole site with their Computers and Smartphones. Users can upvote and downvote images which they like or dislike for giving feedback to the person who uploaded it.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | |  |
| **CHAPTER 1: INTRODUCTION** | | | |
| 1.1 | | General Description under study |  |
| 1.2 | | Objectives of Study |  |
| 1.3 | | Scope of project |  |
| 1.4  1.5 | | Methodology used for data collection  Methodology used for analysis, design and development |  |
|  | |  |  |
| **CHAPTER 2: SYSTEM REQUIREMENT ANALYSIS** | | | |
| 2.1 | | Description of process(es) that are undertaken for software development | ? |
| 2.2 | | Input and Output Processes | ? |
| 2.3 | | Data Elements that are to be incorporated as required by the User | ? |
| 2.4 | | Procedures/rules/mathematical relationships used for converting input into output | ? |
| 2.5 | | The controls (for I/O & access), security needs, validation rules and codes used for data elements | ? |
|  | |  |  |
| **CHAPTER 3: Systems Design** | | | |
| 3.1 | | Physical Design |  |
| 3.2 | | Block Diagram |  |
| 3.3  3.4  3.5 | | Input and Output Processes  DFD Diagram |  |
| 3.6  3.7 | | ER Diagram  Processing Logic  Interface Design  3.8 Database and file Design |  |
| **CHAPTER 4: Systems Development** | | | |
| 4.1 | | Program Development |  |
| 4.2 | Testing and Debugging  **CHAPTER 5: Summary and Conclusions**  5.1 Summary  5.2 Objectives  5.3 Scope  5.4 Atypical’s Limitations, Bugs and Scope for Future Development: | |  |
|  | |  |  |

**CHAPTER 1 - INTRODUCTION**

**1. General Description of the System under Study:**

Atypical is a Web App where Users can download pictures publicly for free without any hassle. It’s made for people like Graphic Designers, Artists, Content Creators who can search images which need in their work and can download them for free. Images can be posted by anyone after creating an account on Atypical by Signing up. It’s clean and elegant design helps people to use it conveniently. People can find images by going through the Categories assigned right after the image is uploaded to the server. It’s responsiveness enables it to be platform independent which simply means Users can conveniently access the whole site with their Computers and Smart phones. Users can up-vote and down-vote images which they like or dislike for giving feedback to the person who uploaded it.

**2. Objectives of Study:**

The objective of Atypical is to give Artists and Photographers a platform to share their images to the world so that they can download them for free. Images are Categorized right after they are uploaded which enables everyone to discover their content in no time. People need to have an account on Atypical to Upload Images. That account also gives them ability to Up-vote and Down-vote images for feedback. In this application, all the Uploads are handled by Everypixel’s API which categorizes images and makes them search-able. Main features of Atypical are listed below:

* Responsive Web Design for making it easily accessible from devices like PCs and Smart Phones
* Dynamic Loading of Content for convenience and automated pagination
* Uses MongoDB which is the most popular NoSQL Database for storing Accounts and Images
* Automated Categorization of Image Uploads and key-wording them for making Images Search-able using Everypixel’s API
* Masonry Layout for Easy Navigation
* User Feedback in the form of Up-votes and Down-votes
* Image Compression with Anti-Aliasing for storing Image data
* Users can change their profile picture, email address, password and username along with their post’s description

**3. Scope of project:**

Atypical is meant to provide a platform for Photographer, Artists or people who want to share their Pictures with the world for free even if they don’t get credit for it. It will also provide people a platform where they can find any kind of image which they want, whether it’s an image of a dog or moon, or dog on moon. Atypical’s design makes it easy for everyone to use it, doesn’t matter if they are using it on their PCs or their Smart phones. It also gives the ability to give feedback to Owner by up-voting and down-voting images which enables the artists analyze how good they are becoming on taking pictures or creating designs.

**4. Methodology used for Data Collection**

Primary Data: For the proposed system, the data has been collected from User Feedback and personal Interviews along for getting user preference and taste of best UI and Design idea.

Secondary Data: It is data gathered from studies, surveys, or experiments that have been run by other people or for research. Atypical is inspired from existing stock image sites like Unsplash, EyeEm and ShutterStock along with a Social Media Platform named Instagram.

**5. Methodology used for Analysis, Design & Development**

Agile Model is used for the development of the proposed system. The main aim of using this model is to add more features in the existing modules to increase project reliability and usability. Using this model, we can adapt to the changing requirements of the user which helps in developing the project in relatively small amount of time. The next increment implements user suggestions plus some additional requirements in the previous increment. The process is repeated until the project is completed.

**5.1 Hardware specification**

* RAM: 8GB RAM
* CPU: Intel Core i5 Processor (7th Gen)
* GPU: NVIDIA GEFORCE 940MX
* HDD: 1 TB
* SSD: 500GB

**5.2 Software Specification**

* HTML
* CSS
* JavaScript
* jQuery
* Ajax
* Boostrap 4
* CroppieJS
* MomentJS
* Python 3
* Flask
* MongoDB

**6. Theoretical Description:**

* User Can Post or React to Existing Posts only if they are logged in and verified
* Images can be downloaded by Clicking on the Image/Post using Flask Packages which performs compression and conversion simultaneously in the Backend
* All the Images uploaded are automatically Categorized and Organized using the EveryPixel API
* Users can up-vote and down-vote images for giving feedback just like Facebook’s Like Button with an addition of Dislike Button.
* Images are made searchable using EveryPixel’s Keywording API during the time of upload

**Chapter 2 Systems Requirement Analysis**

**2.1 Description of process(es) that are undertaken for software development**

* **Sign Up:** Users can create an account on Atypical which is required for posting images and giving feedback to other users. An Account is only needed if someone wants to post an Image or give feedback. An Account is not needed for searching images via Search or Categories, and downloading Images from the site.
* **Login:** Users can use this project for finally uploading their pictures and get them available for everyone. Users can use their Username or Email whichever they find convenient for logging in. Users need to login for Posting Images, Changing their Profile Picture, Image Description and
* **Upload:** This process allows everyone to share their pictures via the site. It stores Images ‘Uploaded’ by the User in the Database and Categorizes them automatically so that they are available for everyone. They can also use hashtags in their post to make their images search-able.
* **Post Management:** Users can edit description of the images they uploaded.
* **Account Management:** Users can change their username, email address, password and Profile Picture anytime.
* **Timeline:** Users can checkout latest images posted by other users directly from homepage. Posts are loaded using AJAX which automates pagination to give infinite scroll effect.
* **Categories:** Users can go through numerous categories for searching any kind of Images available on Atypical. Categories are assigned as soon as images are posted on Atypical via Everypixel’s Keywording API. Users can go through either All Categories or Any Individual Categories.
* **Search:** Users can search for images by using this feature. This feature uses the Hash Tags present in the Posts assigned during Upload via Everypixel’s Keywording API. These tags are stored permanently in the Image Meta Data.
* **Download:** Users can download Images they want by clicking on the Image. Images are downloaded and their Download Count is increased in the Backend and the change is then finally reflected in User Profile.
* **Feedback:** Users can give feedback to other users by clicking on Up-vote and Down-vote button available on Images. Difference of Up-votes and Down-votes are added to the Total Up-votes received by the User.
* **Profile:** Users can check the Image Owner’s profile for his specific posts. User profile displays User’s Profile Picture followed by his/her Total Number of Uploads, Total Number of his/her Images Downloads by other Users, and Total Number of Up-votes he/she received on accumulated on his Images.

**2.2** **Input and Output Processes**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Process** | **Input** | **Output** |
| 1 | Signup | name, email, password, gender, age | Signed Up Successfully! |
| 2 | Login | username or email, password | User Profile is Displayed |
| 3 | Upload | image, description ,session | Image is Displayed if session is valid else Redirect to Homepage with Error Message |
| 4 | Edit Image Description | description, session | Updated Description if session is valid else Redirect to Homepage with Error Message |
| 5 | Search (Navbar) | Query | Search Results are Displayed |
| 6 | Search (Homepage) | Query | Search Results are Displayed |
| 7 | Update User Account | username, email, password, about, session | Updated User Account Info if session is valid else Redirect to Homepage with Error Message |
| 8 | Download Image | click on image | Image is Downloaded |
| 9 | Timeline | mouse scroll | Images are loaded via AJAX |
| 10 | Profile | mouse scroll | Images are loaded via AJAX |

**2.3 Data Elements that are to be incorporated as required by the User**

|  |  |  |
| --- | --- | --- |
| **Collection** | **Fields** | **Type** |
| User | \_id | ObjectID |
| Email | String |
| Username | String |
| totalUploads | Integer |
| Password | String |
| Gender | String |
| current\_sessions | List |
| Age | Integer |
| profilePicture | String |
| About | String |
| totalDownloads | Integer |
| totalUpvotes | Integer |
| Images | \_id | ObjectID |
| userID | ObjectID |
| categories | List |
| image | String |
| totalUpvotes | Integer |
| totalDownloads | Integer |
| description | String |
| image\_id | String |
| upvotes | List |
| downvotes | List |
| tags | List |
| Categories | \_id | ObjectID |
| category | string |
| number\_of\_images | Integer |

**2.4 Procedures/rules/mathematical relationships used for converting input into output**

**2.4.1. Signup:** User enters name email, password, gender, age is stored into Database after checking if they are valid, not empty and does not contain any invalid character and then flash message is displayed if they validate successfully which is Signed Up Successfully! Password is stored after hashing in SHA512 Salted for security reasons.

**2.4.2 Login:** User can Enter his/her username or email and password for logging in, if password User Profile is Displayed. Password is checked via built-in password checker from Werkzeug Security which is a 3rd party Security Library for Python. He is assigned a session which is encrypted with Fernet Technique.

**2.4.3 Upload:** User can upload his/her Image on Atypical after logging in, with or without Image Description, his/her login session which is encrypted using valid else Redirect to Homepage with Error Message

**2.4.4 Edit Image Description:** User can update his post’s description, session in the backend is verified if it is valid and only then Description is Updated. If it isn’t valid then User is Redirected to Homepage with an Error Message.

**2.4.5 Search (Navbar and Homepage):** User can search for anything which is received as query. That query is then filtered by removing Special Characters, then they word in the query are split into words and those words are then compared with Hashtags in the posts of the user. The Most Relevant search results on the basis of hashtags are then displayed.

**2.4.6 Update Account Info:** Users can update their username, email, password or about info of their account after logging in, when they click on update button, their session is verified and then their Personal Information is Updated after several validations on the individual fields. If session is invalid then User is redirected to Homepage with Error Message.

**2.4.7 Download Image:** When a user clicks on Image Post, the request is sent to the server to get high quality version of the Image and then Image is Downloaded right after updating the Number of Downloads attribute in Image and Total Downloads of Uploader’s Images.

**2.4.8 Timeline:** When User scrolls the screen till the bottom, jQuery is used to make AJAX for getting Images from the server. These Images are loaded in Masonry Layout so that they look good and fill the whole screen. These Images are firstly compressed for rendering on the Atypical for reducing browser load. AJAX requests are sent for specific pages like User Profile, Timeline and Each Individual Categories.

**2.4.9 Profile:** When User scrolls the screen till the bottom, jQuery is used to make AJAX for getting Images from the server and images of that specific user account are retrieved. These Images are rendered in Masonry Layout. AJAX requests are validated if they contain the query of a specific length and does not contain any invalid characters then only the result is returned.

**2.4.10 Categories:** When a user clicks on any Category, that category is loaded and fetched to the user same as Timeline and Profile Page using AJAX request specific to the category.

**2.5 The controls (for I/O & access), security needs, validation rules and codes used for data elements**

|  |  |  |
| --- | --- | --- |
| **Form** | **Field** | **Validation Rule** |
| Registration | name | InputRequired(), checkForJunk() -> Raises Validation Error when special character is used |
| gender | InputRequired() |
| age | InputRequired() |
| email | InputRequired(), Email() |
| password | InputRequired(), Length(min=6, max=16), StrongPassword() -> Must Contain atleast 1 Upper Case, 1 Special Character and 1 Number |
| Login | username, email | InputRequired(), Length(min=4, max=50), |
|  | password | InputRequired(), Length(min=6, max=16) |
| Search (Navbar and Homepage) | query | checkForJunk() |
| Image Upload | image, session | FileRequired(), FileAllowed([‘jpg’,’jpeg’,’png’]), Session Verification , image verification using Pillow |
| description, session | Session Verification |
| Edit Image Description | description, session | Session Verification |
| Feedback  (Up-vote, Down-vote) | session | Session Verification |
| Update Account Info | username, session | InputRequired(),  Session Verification |
| email, session | InputRequired(), Email(),  Session Verification |
| password, session | InputRequired(), Length(min=6, max=16), StrongPassword(),  Session Verification |
| about, session | InputRequired(), checkForJunk(),  Session Verification |

**Chapter 3: Systems Design**

**3.1 Physical Design**

Atypical contains many modules for proper being a fully functional Stock Image Site. Modules are Signup, Login, Upload, Post Management, Account Management, Timeline, Categories, Search, Download, Feedback and Profile.

Users can create an account on Atypical which is required for posting images and giving feedback to other users. An Account is only needed if someone wants to post an Image or give feedback. An Account is not needed for searching images via Search or Categories, and downloading Images from the site.

They can use this project for finally uploading their pictures and get them available for everyone. Users can use their Username or Email whichever they find convenient for logging in. Users need to login for Posting Images, changing their Profile Picture, Image Description andUpload process allows everyone to share their pictures via the site.

It stores Images ‘Uploaded’ by the User in the Database and Categorizes them automatically so that they are available for everyone. They can also use hashtags in their post to make their images search-able.

Users can edit description of the images they uploaded, can change their username, email address, password and Profile Picture anytime. can checkout latest images upload by other users directly from homepage. Posts are loaded using AJAX which automates pagination to give infinite scroll effect.

Users can go through numerous categories for searching any kind of Images available on Atypical. Categories are assigned as soon as images are posted on Atypical via Everypixel’s Keywording API.

Users can go through either All Categories or Any Individual Categories. Users can search for images by using this feature. This feature uses the Hash Tags present in the Posts assigned during Upload via Everypixel’s Keywording API. These tags are stored permanently in the Image Meta Data. Users can download Images they want by clicking on the Image.

Images are downloaded and their Download Count is increased in the Backend and the change is then finally reflected in User Profile. Up-vote and Down-vote button available on Images. Difference of Up-votes and Down-votes are added to the Total Up-votes received by the User.

They can check the Image Owner’s profile for his specific posts. User profile displays User’s Profile Picture followed by his/her Total Number of Uploads, Total Number of his/her Images Downloads by other Users, and Total Number of Up-votes he/she received on accumulated on his Images.

**3.1.1 Block Diagram**

Block Diagram is a Diagram of a system, in which the principal parts or functions are represented by blocks connected by lines, which show the relationship of the blocks. They are heavily used in engineering world in hardware design, electric design, software design and process flow diagrams. The Block diagram is typically used for a higher level, less detailed description aimed more at understanding the overall concepts and less at understanding the details of implementation.

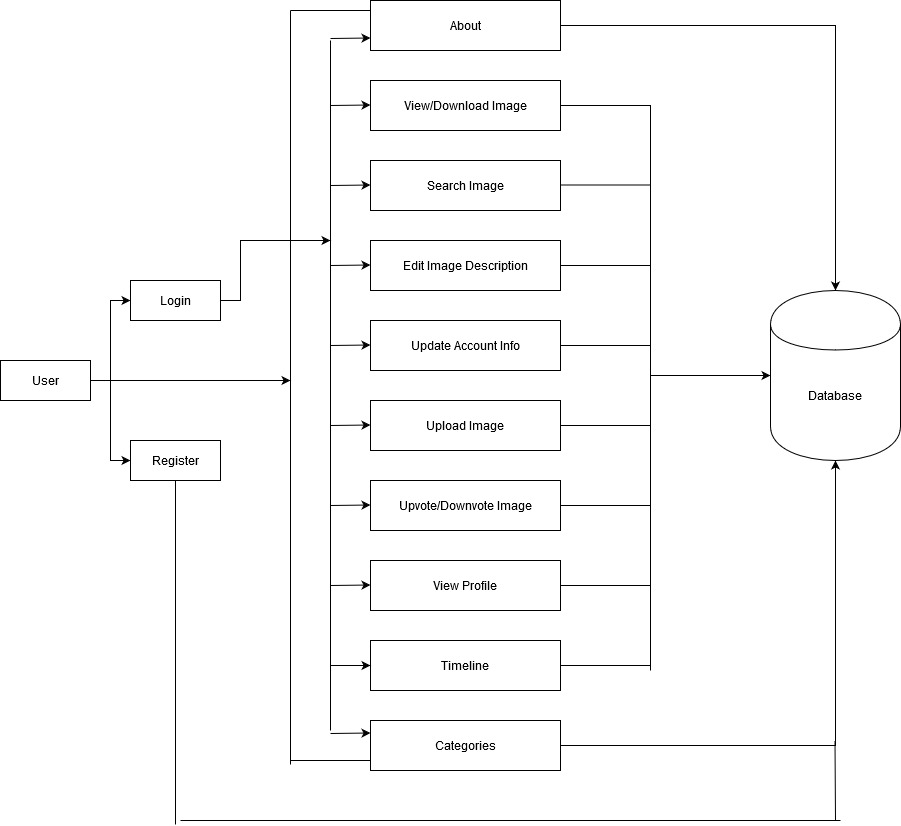
****

Fig 3.1.1. Block Diagram of Atypical

**3.1.2 Input and Output Processes**

Atypical uses a lot of validation for a better security without slowing down the site so that Users can have premium experience while using the Web App for either sharing their pictures, downloading pictures which are available publicly, giving feedback on pictures posted by other Users or browsing the categories or timeline for discovering endless number of Images available on Atypical.

The Following Table contains the list of Each Process taking some input and returning Output to the User after performing some validations behind the scenes.

Table 3.1.2. I/O Processes

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Process** | **Input** | **Output** |
| 1 | Signup | name, email, password, gender, age | Signed Up Successfully! |
| 2 | Login | username or email, password | User Profile is Displayed |
| 3 | Upload | image, description ,session | Image is Displayed if session is valid else Redirect to Homepage with Error Message |
| 4 | Edit Image Description | description, session | Updated Description if session is valid else Redirect to Homepage with Error Message |
| 5 | Search (Navbar) | Query | Search Results are Displayed |
| 6 | Search (Homepage) | Query | Search Results are Displayed |
| 7 | Update User Account | username, email, password, about, session | Updated User Account Info if session is valid else Redirect to Homepage with Error Message |
| 8 | Download Image | click on image | Image is Downloaded |
| 9 | Timeline | mouse scroll | Images are loaded via AJAX |
| 10 | Profile | mouse scroll | Images are loaded via AJAX |

**3.1.3 Use Case Diagram**

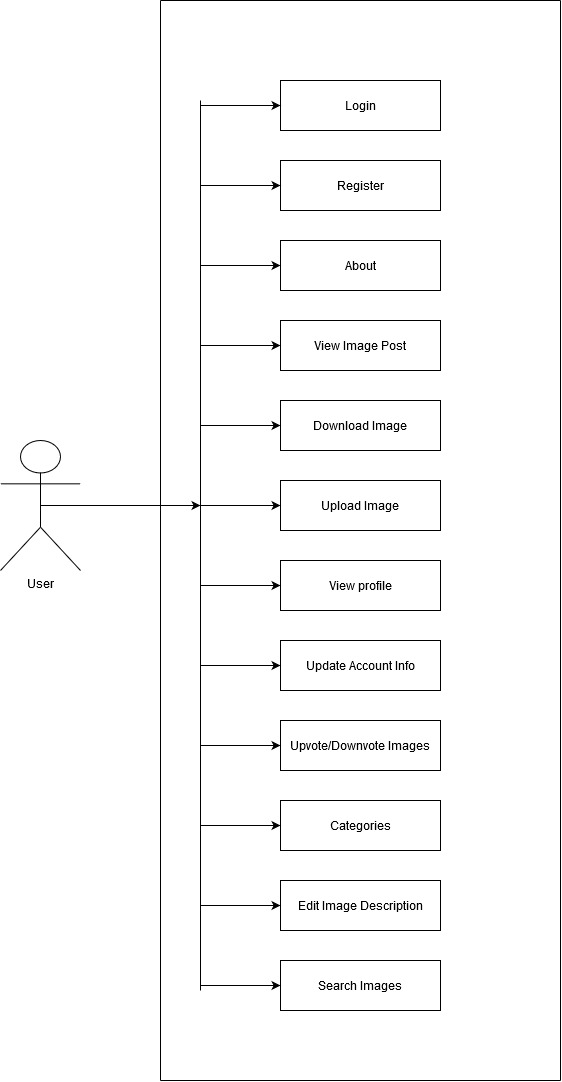


Fig 3.1.3. Use Case Diagram of Atypical

**3.1.4 DFD Diagram**

DFD (data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see 2 levels in data flow diagram, which are: 0-level DFD and 1-level DFD.

1. **Level 0 DFD**

It is also known as context diagram. It’s designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as single bubble with input and output data indicated by incoming/outgoing arrows.

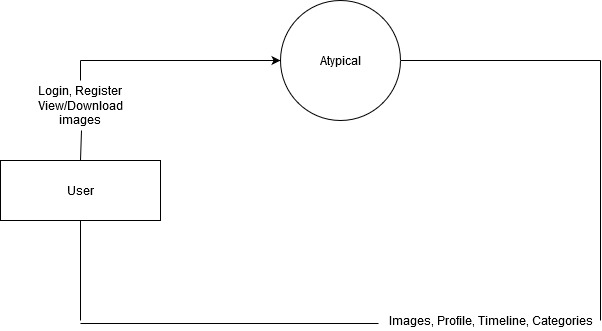
****

Fig 3.1.4 (a) LEVEL 0 DFD

**(b) Level 1 DFD**

In 1-level DFD, context diagram is decomposed into multiple bubbles/processes.in this level we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.

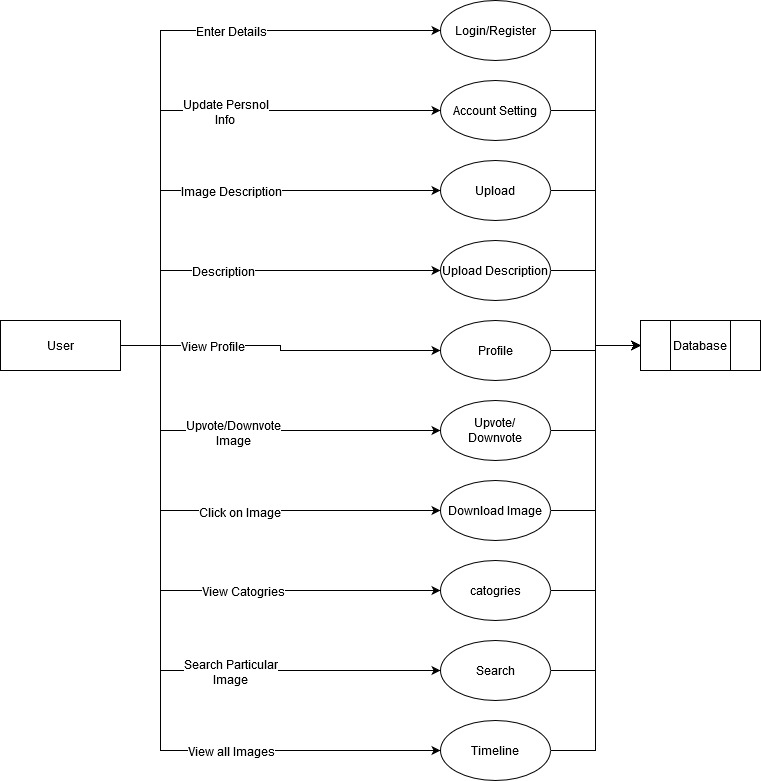


Fig 3.1.4 (b) LEVEL 1 DFD

**3.1.5 ER Diagram**

Atypical’s ER Diagram explains the relationship between each and every Entity and its attributes with one another. It explains which Entities are weak, multivalued or dependent attributes. In ER Diagram below, Image is a weak entity which depends on User Entity and Category is Another Weak Entity which depends on Images. In other words, there won’t be any Images without Users and There won’t be any Categories without Images. User and Image have 1-n (One to Many) relationship which means 1 User can have many Images. On the other hand, Image and Category have m-n (Many to Many) relationship as One Image can Have Many Images and Many Categories can have Many Images.

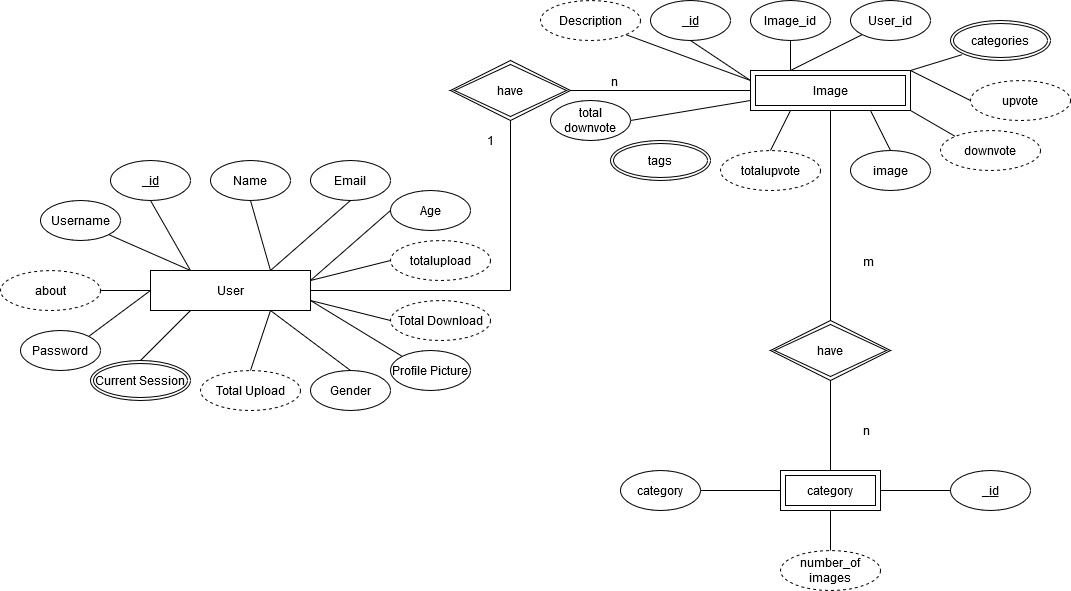


Fig 3.1.5 ER Diagram of Atypical

**4. Processing Logic:**

**4.1. Flow Chart**

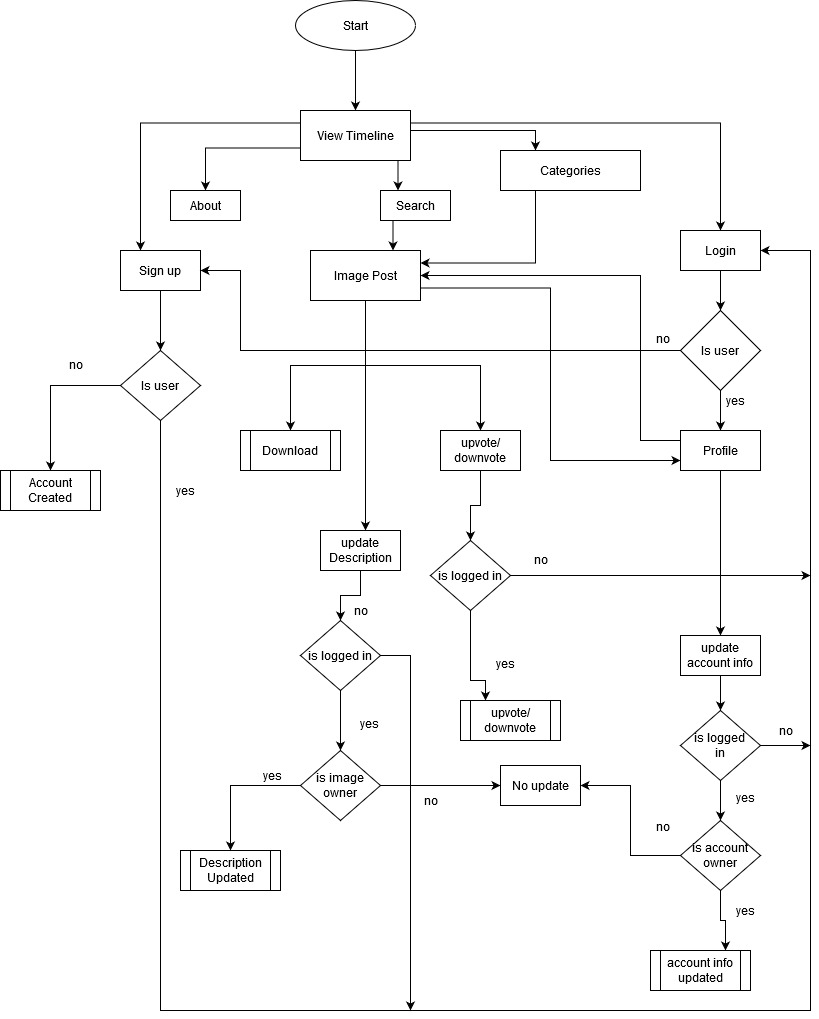


Fig 4.1 Flow Chart of Atypical

**5. Interface Design**:

**(a) Input Design (Form or Screen)**

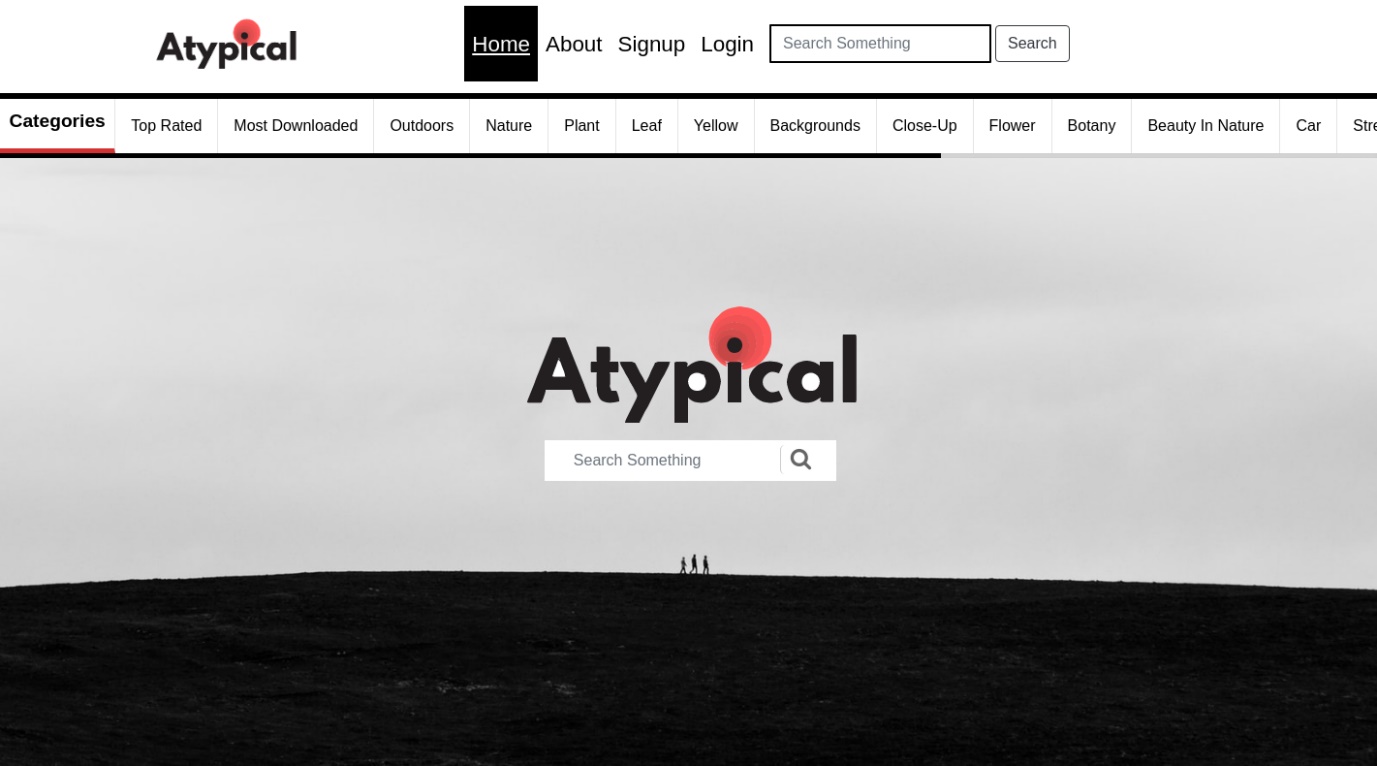
****

Image of Atypical Homepage (index.html)

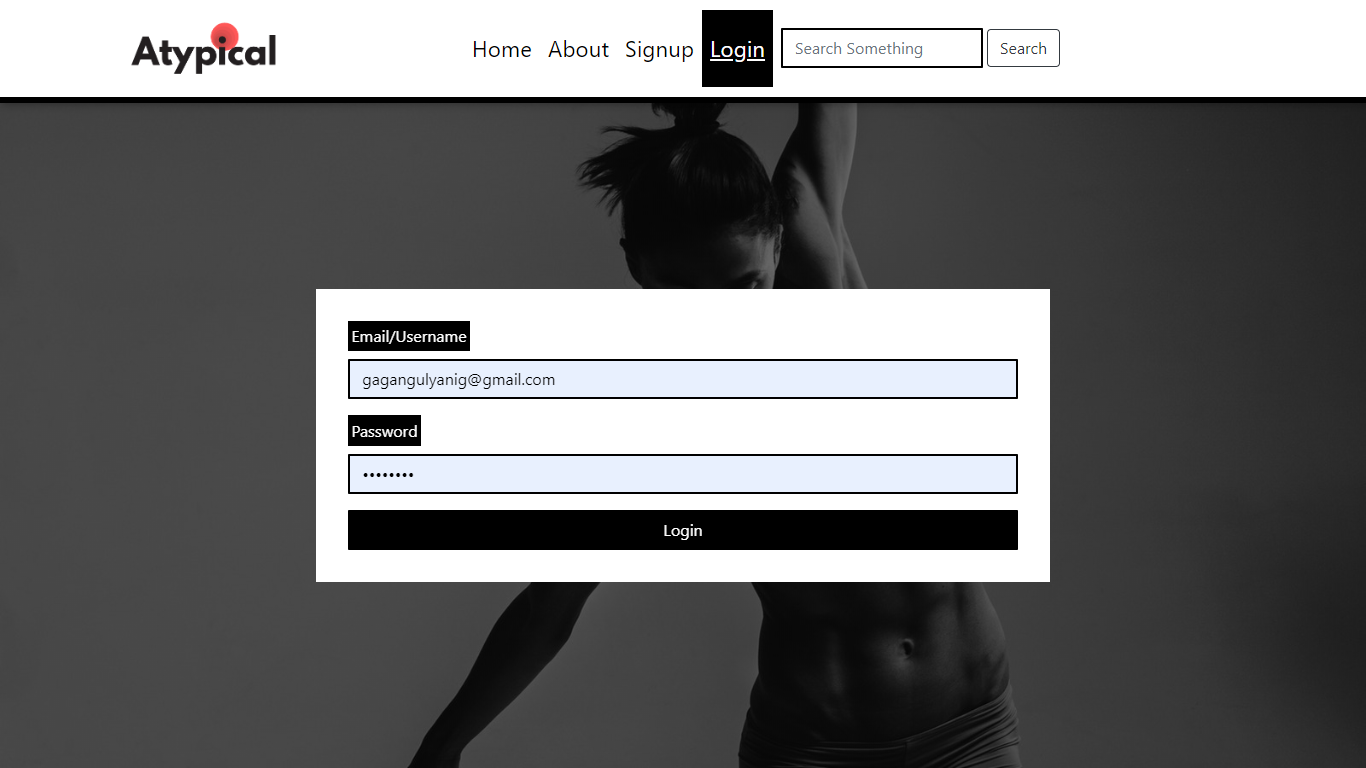


Image of Atypical (login.html)

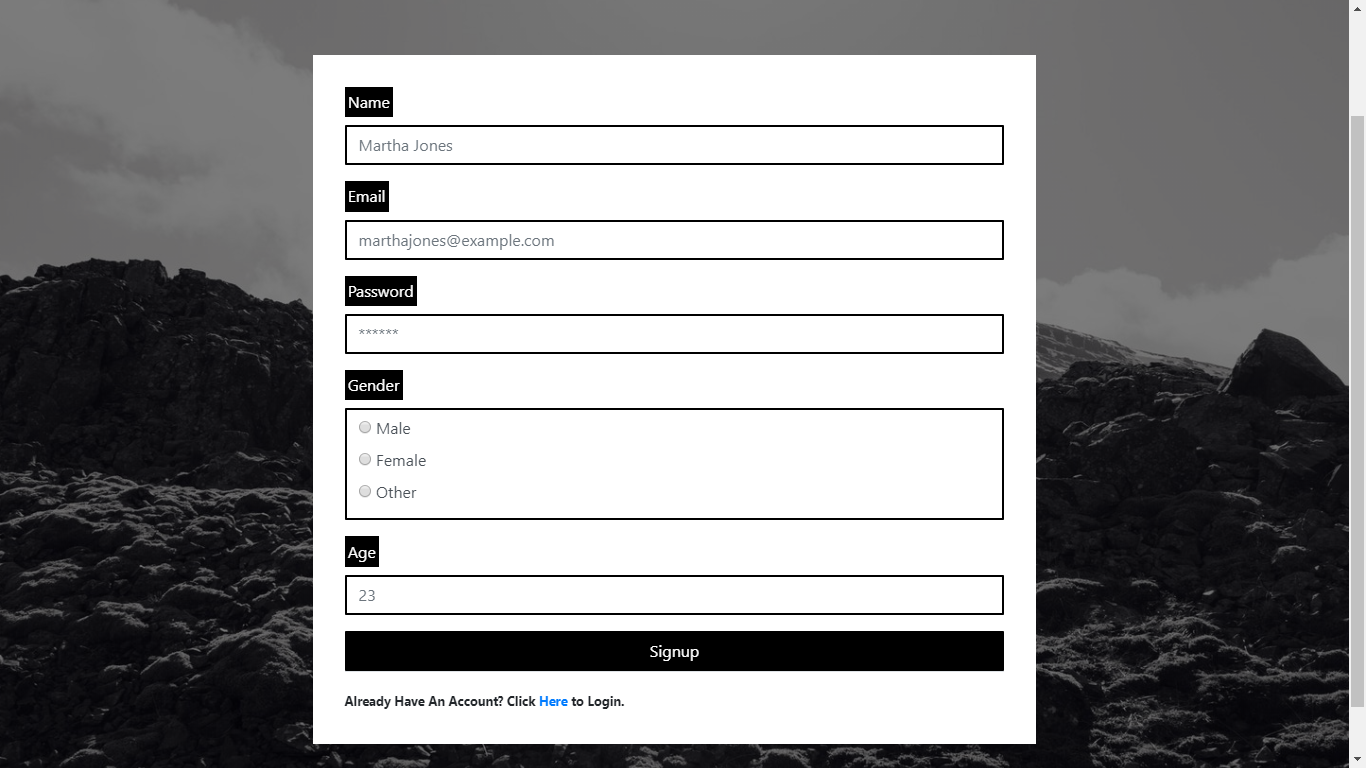


Image of Atypical (Signup.html)

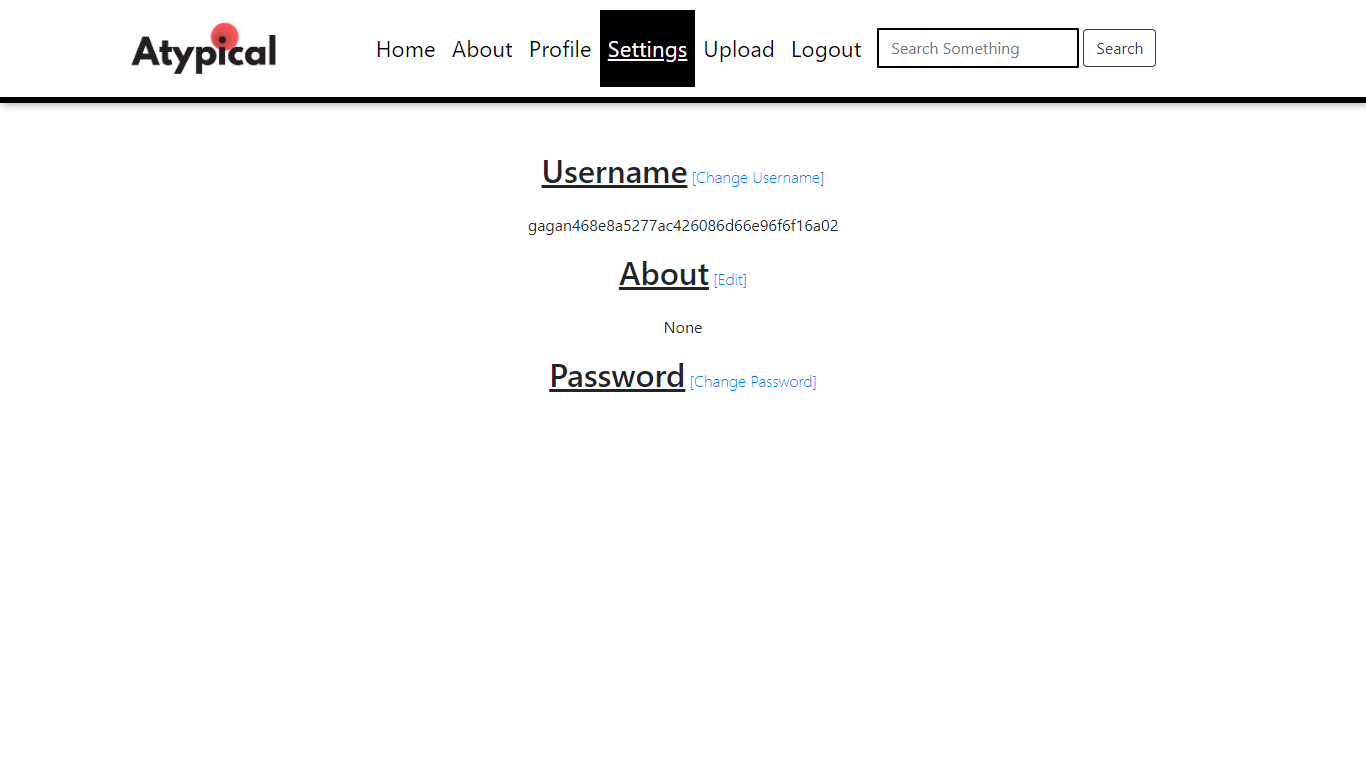


Image of Atypical (settings.html)

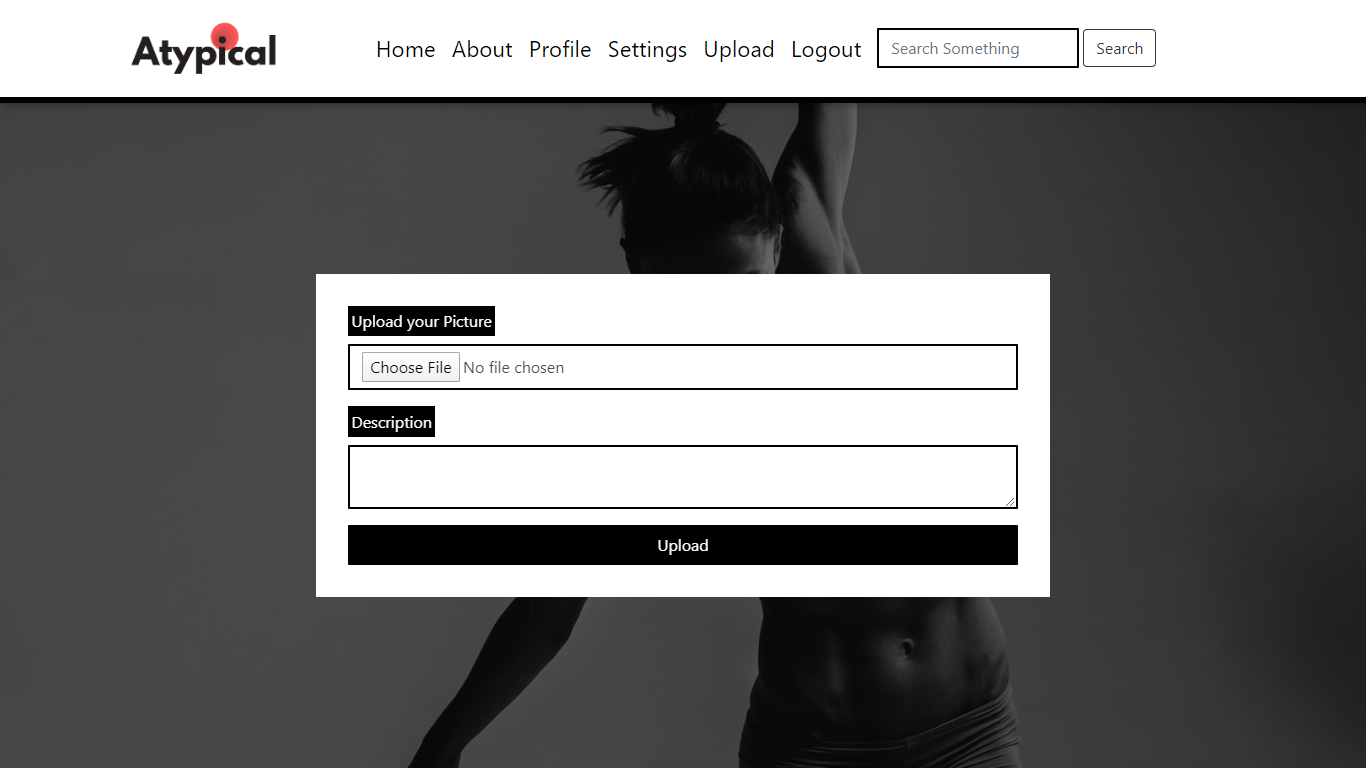


Image of Atypical (upload.html)

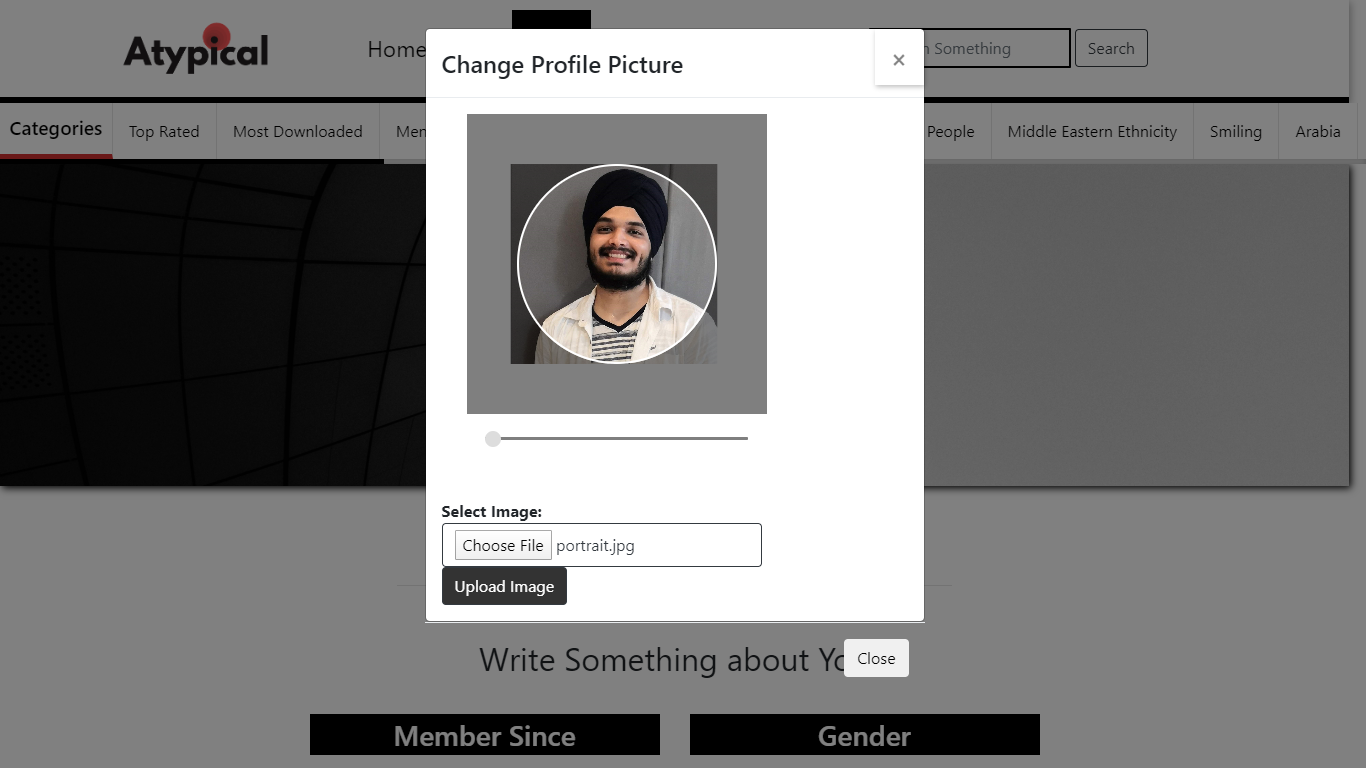


Image of Atypical (profile.html)

1. **Output Design:**

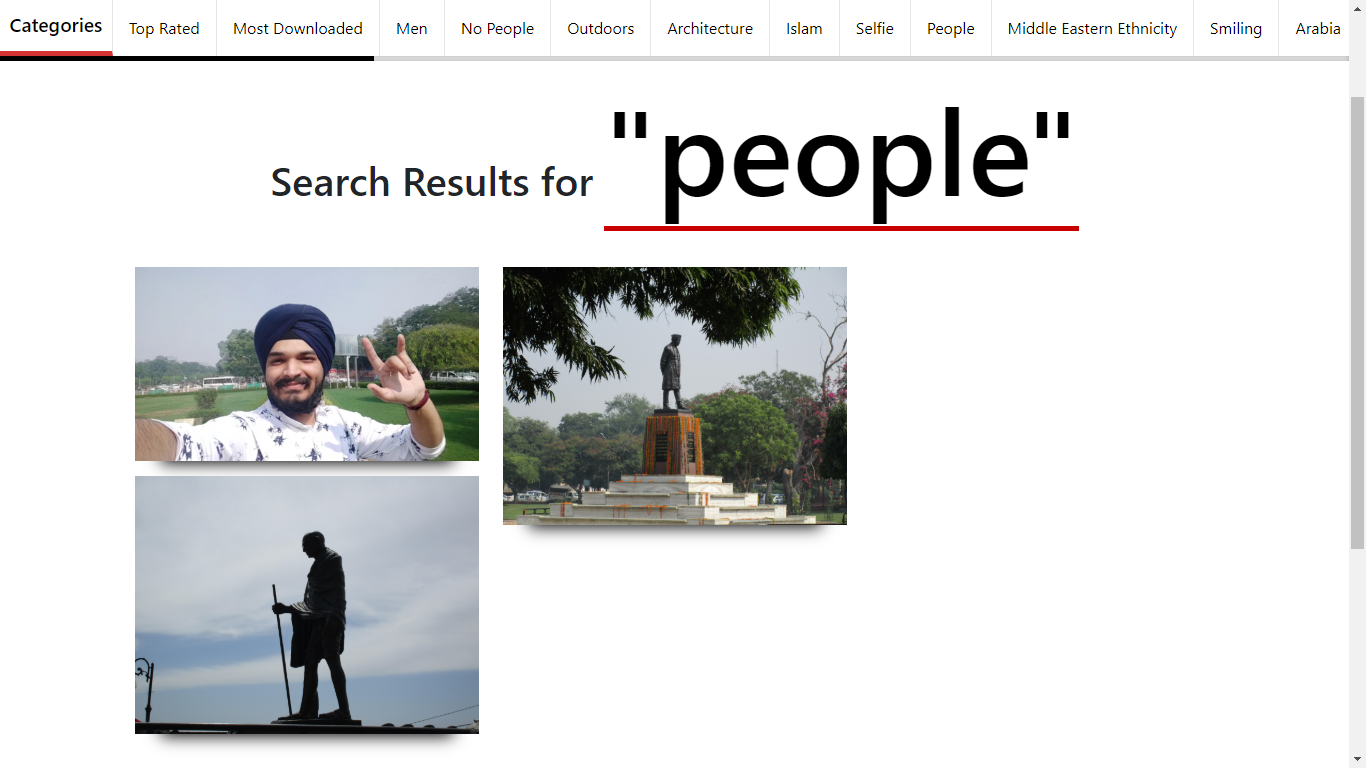


Image of Atypical (search.html)

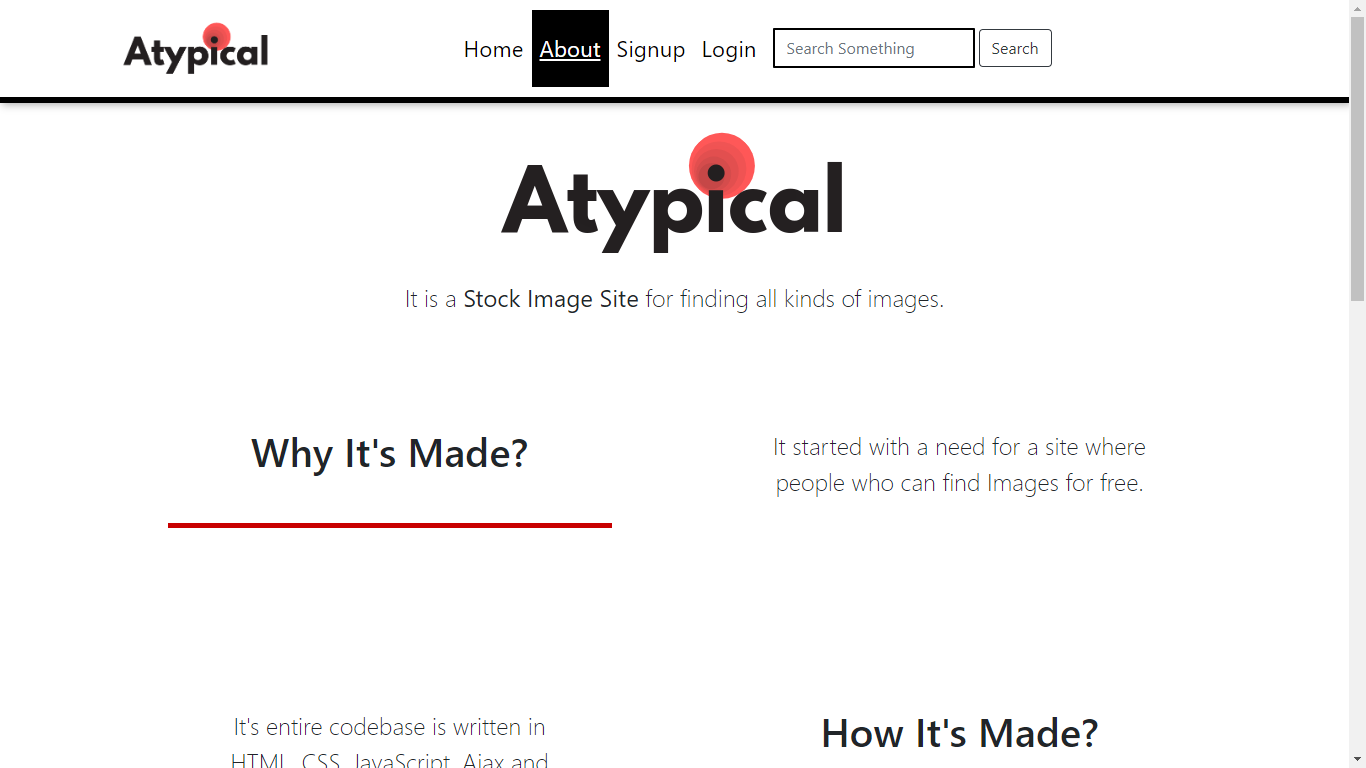


Image of Atypical (about.html)

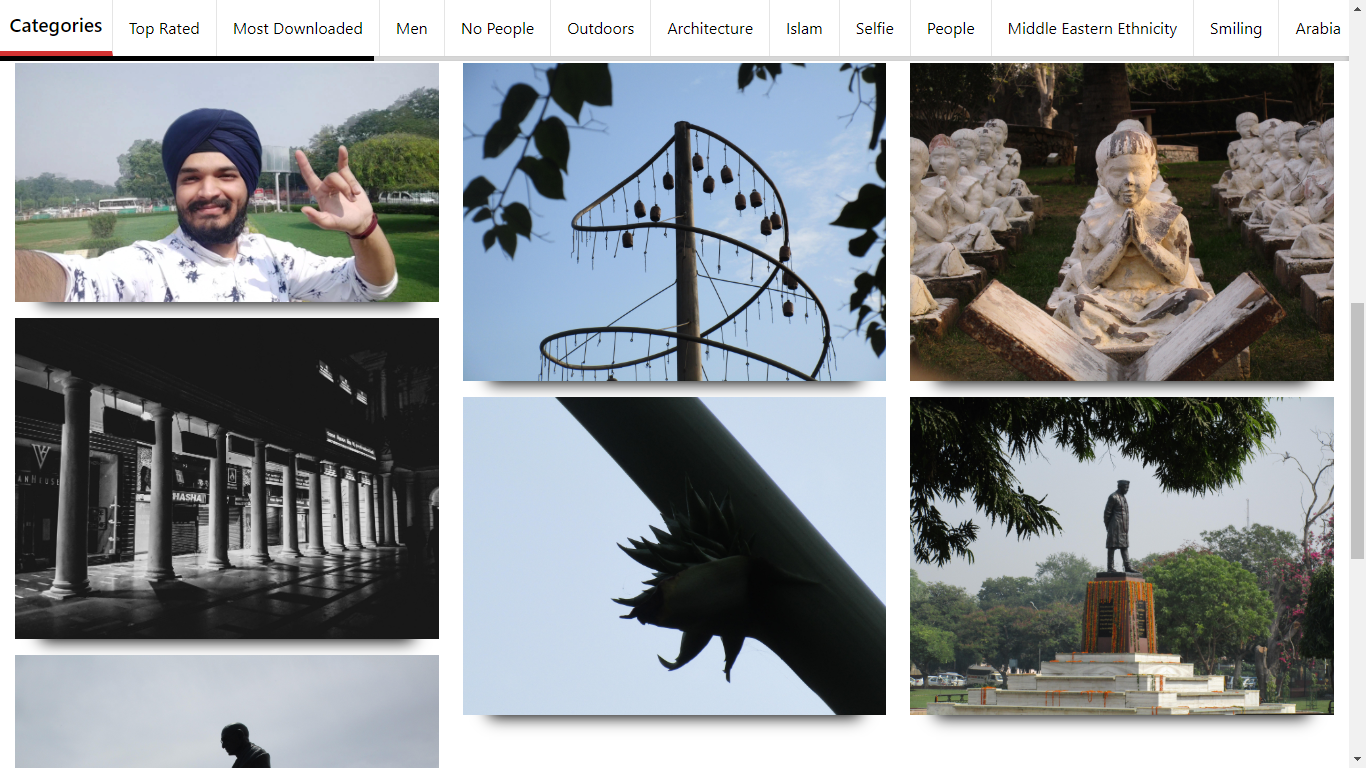


Image of Atypical (profile.html)

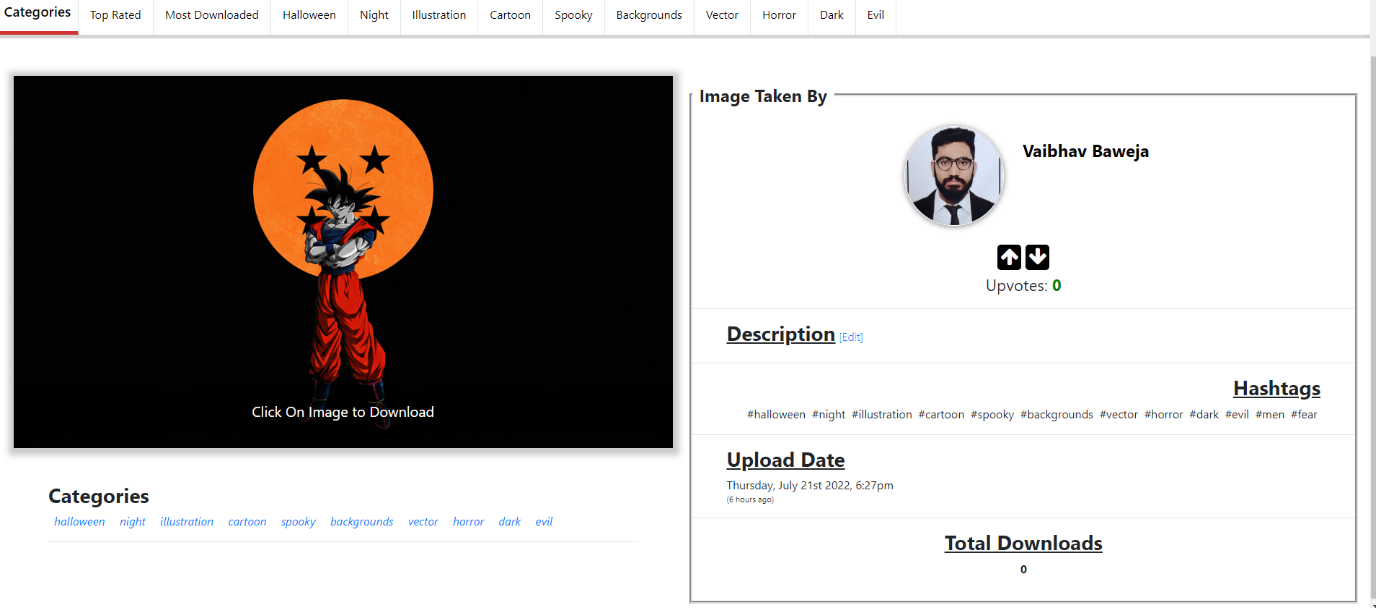
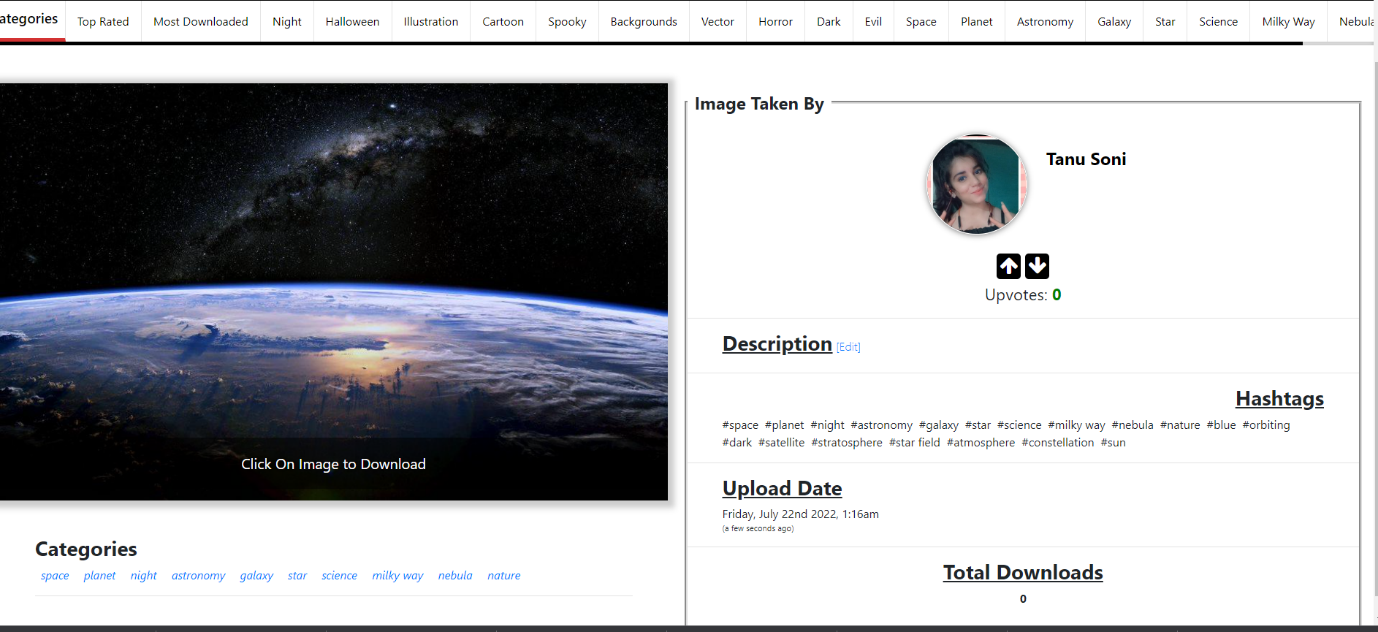


Image of Atypical (post.html)



**6. Database & File Design:**

The following is the Technology Stack Atypical consists of:

* HTML
* CSS
* JavaScript
* jQuery
* CroppieJS
* MomentJS
* Animate.css
* Ajax
* Boostrap 4
* Font Awesome
* Python
* Flask
* MongoDB

Table 6: Fields and Types in each Collection

|  |  |  |  |
| --- | --- | --- | --- |
| **Collection** | **Field Name** | **Type** | **Description** |
| User | \_id | ObjectID | Primary Key assigned by MongoDB |
| Email | String | User’s Email Address |
| Username | String | User’s Username |
| totalUploads | Integer | Count of Total Images Uploaded by the User |
| Password | String | User’s Password |
| Gender | String | User’s Gender |
| current\_sessions | List | User’s Login Sessions |
| Age | Integer | User’s Age |
| profilePicture | String | User’s Profile Picture |
| About | String | User’s About Info |
| totalDownloads | Integer | Count of Total Images of User downloaded by Everyone |
| totalUpvotes | Integer | Difference of Total Upvotes and Downvotes on Images of User by Everyone |
| Images | \_id | ObjectID | Primary Key Assigned by MongoDB |
| userID | ObjectID | \_id of the User who posted the image |
| categories | List | Categories assigned to the Image |
| image | String | Image file in the form of a string containing Base64 Image |
| totalUpvotes | Integer | Difference of Total Upvotes and Downvotes on Image of User by Everyone |
| totalDownloads | Integer | Count of Total Number of Image Downloads by everyone |
| description | String | Description of Image |
| image\_id | String | Unique ID of Image |
| upvotes | List | List containing \_id’s of Users who Upvoted the image |
| downvotes | List | List containing \_id’s of Users who Downvoted the image |
| tags | List | List containing tags for making the image searchable. Alias of tags is Hashtags for the User’s Convenience |
| Categories | \_id | ObjectID | Primary Key Assigned by MongoDB |
| category | string | Category of Images |
| number\_of\_images | Integer | Number of Images containing specific Category |

**Chapter 4: Systems Development**

**4. Purpose:**

**(a) Program Development:**

The following is the source code of Atypical’s Routes which are used by Flask and User Models for Handling User Collection in Python Programming Language.

**routes.py**

========================================================

from flask import (Flask, render\_template,

request, session, redirect, url\_for,

flash, jsonify, send\_file)

from flask\_bootstrap import Bootstrap

from flask\_moment import Moment

from forms import (SignupForm, LoginForm,

UploadForm)

from urllib.parse import (unquote, unquote\_plus)

from models.users import User

from models.database import Database

from models.categories import Category

from models.search import Hashtags, words

from models.images import Image as ImageC

from uuid import uuid4

from string import punctuation

from datetime import datetime

from io import BytesIO

from PIL import Image, ImageFile

from os.path import exists

import base64

import re

app = Flask(\_\_name\_\_)

Database.initialize('Atypical')

@app.context\_processor

def inject\_stage\_and\_region():

categories = Category.getAllCategories()

categories = [cat for cat in categories]

return dict(categories=categories)

app.config['SECRET\_KEY'] = '{}'.format(uuid4().hex)

ImageFile.LOAD\_TRUNCATED\_IMAGES = True

@app.before\_request

def make\_session\_permanent():

session.permanent = True

def getCUserData():

secret\_cookie = session.get('\_cu')

return User.verifySession(secret\_cookie)

app.jinja\_env.globals.update(getCUserData=getCUserData)

bootstrap = Bootstrap(app)

app.config['BOOTSTRAP\_SERVE\_LOCAL'] = True

moment = Moment(app)

def compressImage(base64str, lossless=False, size=(720, 720), quality=35):

img\_data = base64.b64decode(base64str)

f = open('temp', 'wb+')

f.write(img\_data)

f.seek(0)

image = Image.open(f)

if not lossless:

image.thumbnail(size, Image.ANTIALIAS)

image = image.convert('RGB')

buffered = BytesIO()

image.save(buffered, format="JPEG", optimize=True, quality=quality)

encodedImg = base64.b64encode(buffered.getvalue()).decode()

f.close()

del img\_data

del f

del buffered

del image

return encodedImg

@app.route("/")

def index():

if exists('models/config\_api.json'):

return render\_template('index.html')

return render\_template('missing\_config.html')

@app.route("/profile/<string:username>")

@app.route("/profile/")

def profile(username=None):

if session.get('\_cu', None) is None and username is not None:

if any(char in punctuation.replace('\_', '') for char in username):

flash('Invalid Username!')

return redirect(url\_for('index'))

else:

current\_usr = {}

elif username is None and session.get('\_cu', None):

return redirect(f'/profile/{getCUserData().get("username")}', 302)

elif username and session.get('\_cu', None):

# if getCUserData().get('username') == username:

current\_usr = getCUserData()

dataToDisplay = User.getUserByUsername(username)

if dataToDisplay:

return render\_template('profile.html',

userData=dataToDisplay,

current\_usr=current\_usr,

current\_time=datetime.utcnow())

else:

return redirect('/'), 404, {"Refresh": "2; url=/"}

@app.route("/post/<string:img\_id>", methods=['GET'])

@app.route("/post/<string:vote>", methods=['POST'])

@app.route("/post/<string:updDisc>", methods=['PUT'])

@app.route("/post/<string:remove>", methods=['DELETE'])

@app.route("/post/")

def post(img\_id=None, vote=None, updDisc=None, remove=None):

if request.method == 'GET':

if session.get('\_cu', None) is None and img\_id is not None:

if any(char in punctuation.replace('\_', '') for char in img\_id):

flash('Invalid img\_id!')

return redirect(url\_for('index'))

else:

current\_usr = {}

elif img\_id is None:

return redirect(url\_for('index'))

elif img\_id and session.get('\_cu', None):

current\_usr = getCUserData()

upvoted = downvoted = False

dataToDisplay = User.getImagebyImgID(img\_id)

if session.get('\_cu'):

if current\_usr.get('\_id') in dataToDisplay.get('upvotes'):

upvoted = True

if current\_usr.get('\_id') in dataToDisplay.get('downvotes'):

downvoted = True

if dataToDisplay:

return render\_template('post.html',

ImgData=dataToDisplay,

current\_usr=current\_usr,

getUserData=User.getUserByID,

current\_time=datetime.utcnow(),

upvoted=upvoted,

downvoted=downvoted)

else:

flash('Invalid Post ID')

return redirect(url\_for('index'))

elif request.method == 'POST':

img\_id = request.get\_data(as\_text=True).split('=')[1]

if len(img\_id) != 32:

return jsonify({'error': 'Invalid Request!'})

if vote == 'upvote':

result = User.vote(session.get('\_cu'), img\_id=img\_id)

else:

result = User.vote(session.get('\_cu'), img\_id=img\_id,

upvote=False)

if result is True:

return jsonify({'success': True})

else:

if result == -1:

flash('User who posted this image not found!')

flash('Upvoting/Downvoting and Commenting is Di\

sabled for this Post!')

return jsonify({'error': '-1'})

elif result == -2:

flash('Please Login to Upvote/Downvote/Comment')

return jsonify({'error': '-2'})

elif result == -3:

flash('Invalid Post ID')

return jsonify({'error': '-3'})

elif request.method == 'PUT':

desc = request.json

if desc:

usr = getCUserData()

img\_id = desc.get('img\_id')

description = desc.get('desc')

if any(char in img\_id for char in punctuation):

flash('Invalid Post ID')

return jsonify({'error': '-1'})

if len(img\_id) != 32:

flash('Invalid Post ID')

return jsonify({'error': '-2'})

if usr:

img = User.getImagebyImgID(img\_id)

if img.get('userID') == usr.get('\_id'):

if User.changeDescription(description, img\_id):

return jsonify({'success': '1'})

else:

return jsonify({'error': '-1'})

else:

flash('Invalid Request!')

return jsonify({'error': -2})

else:

flash('Please Login to Make Changes!')

return jsonify({'error': -2})

else:

return jsonify({'error': 'no json'})

else:

return jsonify({'error': 'invalid request'})

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

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

def displayImages():

if request.method != 'POST':

return redirect('/', 302)

else:

data = request.json

print(data)

if not data.get('skip') is None:

skip = data.get('skip', 0)

cat = data.get('category')

username = data.get('username')

search\_query = data.get('search')

# print(search\_query)

usr = {}

isProfile = False

if 'int' in str(type(skip)) and skip < 10000:

if cat and cat.replace(' ', '').isalpha() and len(cat) < 100:

imgs = ImageC.GetImgsByCategory(skip\_=skip,

limit\_=10,

category=cat)

elif username and len(username) < 100:

usr = User.getUserByUsername(username)

if usr:

imgs = ImageC.GetAllImages(query={

'userID': usr.get('\_id')},

skip\_=skip, limit\_=10, sortField='\_id')

isProfile = True

elif search\_query:

query = words(search\_query)

squery=[]

for q in range(len(query)):

squery.append(

{'tags': {'$all': query[:q+1]}}

)

imgs = list(ImageC.GetAllImages(

query = {

"$or": squery

},

skip\_=skip,

limit\_=10

))

print(f'images for search : {squery}')

else:

imgs = ImageC.GetAllImages(skip\_=skip, limit\_=10)

final\_imgs = []

for img in imgs:

description = img.get('description')

hasProfilePicture = True

if not usr:

usr = User.getUserByID(img.get('userID'))

if not isProfile:

description = None

if usr:

if not usr.get('profilePicture'):

hasProfilePicture = False

if usr.get('gender') == 'M':

usr.update({

'profilePicture': url\_for('static',

filename='\

images/profile/profile\_male.jpg')

})

else:

usr.update({

'profilePicture': url\_for('static',

filename='\

images/profile/profile\_female.jpg')

})

# print(img.get('img\_id'))

final\_imgs.append({

'img\_id': img.get('img\_id'),

'category': img.get('categories'),

'name': usr.get('name').title(),

'profilePicture': str(usr.get('profilePicture')),

'description': description,

'isProfile': isProfile,

'created\_at': Database.created\_at(img.get('\_id')),

'hasProfilePicture': hasProfilePicture})

usr = None

del img

del imgs

# print(final\_imgs[0]['profilePicture'])

return jsonify({'images': final\_imgs, 'endCursor': skip + 10})

else:

return jsonify({'error': 'something wrong with skip value'})

else:

return jsonify({'error': 'invalid request'})

@app.route('/images/<string:img\_id>.jpg')

@app.route('/images/display/<string:img\_id>.jpg')

def get\_image(img\_id):

if any(char in img\_id for char in punctuation):

flash('Invalid Post ID')

return redirect('/', 302)

if len(img\_id) != 32:

flash('Invalid Post ID')

return redirect('/', 302)

img = User.getImagebyImgID(img\_id)

if img:

image\_binary = base64.b64decode(img.get('image'))

if "/images/display/" not in request.path:

usr = User.getUserByID(img.get('userID'))

if usr:

usr.update({'totalDownloads': usr.get('totalDownloads') + 1})

User.updateUserInfo(user=usr, \_id=usr.get('\_id'))

ImageC.updateImgData(img\_id=img\_id, Download=True)

else:

print('Error: Couldn\'t find User..')

# image\_binary = read\_image(pid)

# response = make\_response(image\_binary)

# response.headers.set('Content-Type', 'image/jpeg')

if img.get('name'):

return send\_file(

BytesIO(image\_binary),

mimetype='image/jpeg',

as\_attachment=True,

attachment\_filename=f"{img.get('name').title()}\'s Picture - \

Atypical.jpeg")

return jsonify({'error': 'Image Corrupt!'})

else:

return jsonify({'error': 'Image Not Found!'})

@app.route("/settings/")

def settings():

if session.get('\_cu'):

dataToDisplay = getCUserData()

return render\_template('settings.html',

userData=dataToDisplay,

current\_time=datetime.utcnow())

else:

flash('Please Login to Access Settings Page!')

return redirect('/login', 302)

@app.route("/categories")

@app.route('/category/<string:category>')

def Categories(category=None):

categories = User.getCategories()

if category is None:

# print(categories)

for cat in categories:

image = cat.get('image')

cat.update({'image': compressImage(image)})

print(categories[0].get('category'))

return render\_template('category.html',

categories=categories,

enumerate=enumerate)

if any(char in punctuation for char in category):

flash('Invalid Category!')

print(category)

return redirect('/categories', 302)

cats = User.getCategories(category)

if cats:

imgs = [i for i in cats]

else:

flash('Category not found!')

return redirect('/')

if imgs:

return render\_template('imgByCat.html',

category=category)

else:

flash('Category not Found!')

return redirect('/categories', 302)

@app.route('/search')

def imageSearch():

query = " ".join(words(request.args.get('q')))

return render\_template('search.html', query=query)

@app.route("/about")

def about():

return render\_template('about.html')

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

def login():

form = LoginForm()

if request.method == 'POST':

if session.get('\_cu', None) == None:

form = LoginForm()

# If Post Method is used..

if form.validate() == False:

# if form validation fails..

return render\_template('login.html', form=form)

else:

# Regular Expression for checking email syntax

isEmail = re.compile(r"[^@]+@[^@]+\.[^@]+")

if isEmail.fullmatch(form.username.data):

# flash('Email detected!')

result = User.checkUser(

username=form.username.data.lower(),

password=form.password.data,

email=True)

else:

result = User.checkUser(

username=form.username.data.lower(),

password=form.password.data)

if result:

new\_session = User.createSession(result)

session['\_cu'] = new\_session

return redirect('/profile/')

elif result is False:

flash('Invalid Password! Try Again..')

else:

flash('User Does not Exist!')

flash('Please Check your Email Address or Username \

and Try Again..')

return redirect(url\_for('login'))

else:

return redirect(url\_for('index'))

elif request.method == 'GET':

return render\_template('login.html', form=form)

@app.route('/upload/profilePicture', methods=['POST'])

def updateProfilePicture():

current\_session = session.get('\_cu')

if current\_session:

img = request.get\_data(as\_text=True)

# as\_text=True

if img:

img = str(unquote(img))[str(unquote(img)).find('=') + 1:]

image = str(unquote(img)).split(',')[-1]

img

try:

img\_data = base64.b64decode(image)

f = open('temp', 'wb+')

f.write(img\_data)

f.seek(0)

image = Image.open(f)

image.verify()

except (IOError, SyntaxError):

flash('Picture is Corrupt!')

flash('Please Upload Another Picture or Try Again!')

return jsonify({'error': 'Image is Corrupt!'})

else:

f.seek(0)

image = Image.open(f)

image = image.convert('RGB')

image = image.resize((900, 900), Image.ANTIALIAS)

buffered = BytesIO()

image.save(buffered, format="JPEG", optimize=True, quality=75)

encodedImg = base64.b64encode(buffered.getvalue()).decode()

result = User.changeProfilePicture(

\_cu=current\_session,

profilePicture=encodedImg)

f.truncate()

f.close()

flash('Profile Picture has been Updated!')

del img\_data

del f

del buffered

del image

del encodedImg

if result:

return jsonify({'success': 'Profile Picture Updated!'})

else:

return jsonify({'error': 'User Not Found!'})

return jsonify({'result': 'success'})

else:

return jsonify({'error': 'Invalid Args!', 'args': request.args})

else:

return jsonify({'error': 'Invalid Session!'})

@app.route('/api', methods=['POST', 'PUT', 'DELETE'])

def api():

current\_session = session.get('\_cu')

if current\_session:

data = request.json

if data:

print(data)

result = None

if data.get('fieldType', '') == 'username':

usr = getCUserData()

if usr:

uname = data.get('data')

if len(uname) > 100:

flash('Username too Long!')

return jsonify({'error': 'failed'})

if not uname.replace('\_', '').isalnum():

flash('Username Must Not Contain Special Characters')

return jsonify({'error': 'failed'})

uname = uname.replace(' ', '\_')

result = User.changeUsername(

usr.get('\_id'), uname)

# print(result)

flash('Username Changed Successfully!')

else:

flash('Please Login to Make these Changes..')

return redirect('/', 302)

elif data.get('fieldType', '') == 'about':

usr = getCUserData()

if usr:

about = data.get('data')

about = about.replace('\n', '<br>')

result = User.changeAbout(usr.get('\_id'),

about)

# print(result)

flash('About You updated Successfully Successfully!')

else:

flash('Please Login to Make these Changes..')

return redirect('/', 302)

elif data.get('fieldType', '') == 'password':

usr = getCUserData()

if usr:

result = User.changePassword(

usr.get('\_id'), data.get('data'))

print(result)

flash('Password Changed Successfully!')

else:

flash('Please Login to Make these Changes..')

return redirect('/', 302)

else:

return(jsonify({'success': False}))

if result:

return(jsonify({'success': True}))

else:

return(jsonify({'success': False}))

else:

print('data not found')

return redirect('/')

else:

print('invalid request')

return('<h1>Invalid Request Sent to The Server</h1>', 404)

@app.route('/remove/profilePicture', methods=['POST'])

def removeProfilePicture():

current\_session = session.get('\_cu')

if current\_session:

json\_data = request.get\_data(as\_text=True)

response = json\_data.split('=')

if 'true' in response and 'removeProPic' in response:

User.changeProfilePicture(

\_cu=current\_session,

profilePicture=None)

flash('Profile Picture has been Removed!')

return jsonify({'profilePicture': 'removed'})

else:

return jsonify({'error': 'invalid request'})

else:

return jsonify({'error': 'Invalid Session!'})

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

def upload():

if User.sessionCreatedAt(session.get('\_cu', None)):

form = UploadForm()

if request.method == 'POST':

if form.validate() == False:

flash('Validation Failed!')

return redirect(url\_for('upload'))

else:

image = form.photo.data

try:

img = Image.open(image)

img.verify()

except (IOError, SyntaxError):

flash('Image is not Valid!')

else:

img = Image.open(image)

buffered = BytesIO()

img.save(buffered, format="JPEG",

optimize=True, quality=75)

fileObj = buffered.getvalue()

encodedImg = base64.b64encode(fileObj).decode()

# compressing image for API

size = img.size

img.resize((size[0]//10, size[1]//10), Image.ANTIALIAS)

img.save(buffered, format="JPEG",

optimize=True, quality=75)

fileObj = buffered.getvalue()

tags, api\_data = Hashtags(fileObj)

min\_score = api\_data['keywords'][9]['score'] # 10th highest score

cats = [i.get('keyword').lower() for i in api\_data['keywords'] if i.get('score') >= min\_score]

tags = [i.lower() for i in tags]

img\_id = str(uuid4().hex)[::-1]

User.uploadImage(session.get('\_cu'),

categories=cats,

image=encodedImg,

description=form.description.data,

hashtags=tags,

img\_id=img\_id)

del img

del fileObj

flash('Image Uploaded Successfully!')

return redirect(f'/post/{img\_id}', 302)

elif request.method == 'GET':

return render\_template('upload.html', form=form)

else:

if session.get('\_cu'):

flash('Invalid Session Cookies..')

flash('Please Login Again..')

session.pop('\_cu', None)

else:

flash('You are not Logged In!')

flash('Please Login to Upload a Picture!')

return redirect(url\_for('login'))

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

def signup():

form = SignupForm()

if request.method == 'POST':

if form.validate() == False:

return render\_template('signup.html', form=form)

else:

result = User.checkUser(

username=form.email.data,

password=form.password.data,

signUp=True,

email=True)

if result:

flash('User Already Exists!')

flash('Please Login or Use different email address..')

return redirect(url\_for('signup'))

else:

newUser = User(name=form.name.data.lower(),

email=form.email.data.lower(),

username="{}{}".format(

form.name.data.split()[0].lower(),

uuid4().hex),

password=form.password.data,

age=form.age.data,

gender=form.gender.data)

newUser.saveUser()

flash('Account Created Successfully! Now go to the login page\

to Login into your account.')

return redirect(url\_for('index'))

elif request.method == 'GET':

session['username'] = None

return render\_template('signup.html', form=form)

@app.route("/logout")

def logout():

cu = session.pop('\_cu', None)

if cu:

User.removeSession(cu)

flash('You are Logged out!')

else:

flash('You were not Logged In!')

return redirect(url\_for('index'))

@app.errorhandler(404)

def page\_not\_found(e):

return render\_template('404.html'), 404

@app.errorhandler(500)

def internal\_server\_error(e):

return render\_template('500.html'), 500

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True,

host='0.0.0.0')

**users.py**

from models.database import Database

from models.images import Image

from models.categories import Category

from models.encryption\_ import Encrypt, Decrypt

from uuid import uuid4

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from datetime import datetime

import re

class User(object):

COLLECTION = "Users"

def \_\_init\_\_(self,

name,

email,

username,

password,

age,

gender,

current\_sessions=[],

coverPhoto=None,

profilePicture=None,

about=None,

totalUploads=0,

totalDownloads=0,

totalUpvotes=0,

\_id=None):

self.name = name

self.totalUploads = totalUploads

self.username = username

self.email = email

self.gender = gender

self.password = User.set\_password(password)

self.coverPhoto = coverPhoto

self.current\_sessions = current\_sessions

self.age = age

self.profilePicture = profilePicture

self.about = about

self.totalDownloads = totalDownloads

self.totalUpvotes = totalUpvotes

self.\_id = \_id

@staticmethod

def set\_password(password):

return generate\_password\_hash(password, method='pbkdf2:sha512')

@staticmethod

def checkUser(username, password, signUp=False, email=False):

"""

This function takes username and password

Returns if one of the following is the Case

Case 1 : if user is not found = None

Case 2 : if user is found

a) if password is correct = True

b) if password is incorrect = False

"""

if email:

result = Database.find(collection=User.COLLECTION,

query={"email": username})

else:

result = Database.find(collection=User.COLLECTION,

query={"username": username})

if signUp:

if result:

return User.getUser(result)

return result

if result:

if check\_password\_hash(result.get("password"), password):

return result

else:

return False

def saveUser(self):

"""

Inserts User into the collection

Returns -1 if User Exists in Users Collection

"""

if User.checkUser(username=self.username,

password=self.password, signUp=True):

return False

Database.insert(collection=User.COLLECTION,

data=self.toJson())

return True

def getUserInfo(self):

"""

Converts User object to Json Object

Adds 'created\_at' attribute by using '\_id'

Returns Json Object

"""

userData = self.toJson()

userData['created\_at'] = Database.created\_at(self.\_id)

return userData

@staticmethod

def createSession(jsonObj):

"""

Creates a new user session and updates current user document

in User Collection

Returns New User Session

"""

newSession = Encrypt(str(datetime.utcnow()))

newSession = [i.decode('utf-8') for i in newSession]

if jsonObj['current\_sessions'] is None:

jsonObj['current\_sessions'] = newSession

else:

jsonObj['current\_sessions'].append(newSession)

Database.update(collection=User.COLLECTION,

query={'\_id': jsonObj['\_id']}, update\_query=jsonObj)

# print('updating Database for new session')

return newSession[0]

@staticmethod

def getUserBySession(current\_session, logout=False):

"""

Finds User using Current Session

If Found:

Returns User Object

Else:

Returns None

"""

result = Database.find\_multi(

collection=User.COLLECTION,

SearchField='current\_sessions', items=[current\_session])

if result:

return User.getUserByID(result.get('\_id'), logout=logout)

@staticmethod

def verifySession(current\_session):

return Database.find\_multi(collection=User.COLLECTION,

SearchField='current\_sessions',

items=[current\_session])

@staticmethod

def getUserByID(\_id, logout=False):

usr = Database.find(collection=User.COLLECTION, query={'\_id': \_id})

if usr:

usr.pop('password', None)

# if not logout:

# usr.pop('current\_sessions', None)

return usr

@staticmethod

def sessionCreatedAt(current\_session):

"""

Decrypts the session for cookie

if session is real,

Returns datetime object

else

Returns None

"""

result = User.verifySession(current\_session)

if result:

result = result.get('current\_sessions')

current\_session\_encoded = current\_session.encode('utf-8')

key = 0

for k in result:

if current\_session in k:

key = k[1]

break

if key:

time = Decrypt(enMessage=current\_session\_encoded,

key=key.encode('utf-8'))

return datetime.strptime(time, '%Y-%m-%d %H:%M:%S.%f')

@staticmethod

def removeSession(current\_session):

"""

Removes a user session provided from the argument

and updates current user document in User Collection

Returns None

"""

usr = User.getUserBySession(current\_session, logout=True)

for session in usr.get('current\_sessions'):

if current\_session in session:

ele = session

usr['current\_sessions'].remove(ele)

Database.update(collection=User.COLLECTION,

query={'\_id': usr.get('\_id')}, update\_query=usr)

def toJson(self):

"""

Converts User object to json

(mostly for saving and accessing user)

"""

return {"name": self.name,

"email": self.email,

"username": self.username,

"totalUploads": self.totalUploads,

"password": self.password,

"coverPhoto": self.coverPhoto,

"gender": self.gender,

"current\_sessions": self.current\_sessions,

"age": self.age,

"profilePicture": self.profilePicture,

"about": self.about,

"totalDownloads": self.totalDownloads,

"totalUpvotes": self.totalUpvotes}

@staticmethod

def getUserByUsername(username, usr=None):

if not usr:

usr = Database.find(collection=User.COLLECTION,

query={'username': username})

if usr:

usr.pop('password', None)

usr.pop('current\_sessions', None)

usr.update({'createdAt': Database.created\_at(usr['\_id'])})

print('user found!')

return usr

print('user not found!')

return None

return usr

@staticmethod

def getImages(\_id):

"""

Returns Images uploaded by the User

"""

return Image.GetImgsByUserID(\_id)

@staticmethod

def getImagebyImgID(img\_id):

"""

Returns Image uploaded by the User

By Using Image ID

"""

img = Image.GetByImgID(img\_id)

if img:

userID = img.get('userID')

usr = User.getUserByID(userID)

if usr:

img.update(

{'name': usr.get('name'),

'gender': usr.get('gender'),

'profilePicture': usr.get('profilePicture'),

'created\_at': Database.created\_at(img.get('\_id'))})

return img

@staticmethod

def vote(\_cu, img\_id, inc=1, upvote=True):

img = Image.GetByImgID(img\_id)

if img:

print('image found')

userID = img.get('userID')

voter = User.verifySession(\_cu)

if voter:

print('voter found')

usr = User.getUserByID(userID)

if usr:

print('OP found')

votes = len(img.get('upvotes')) - len(img.get('downvotes'))

# subtracting current post's upvotes from User's total upvotes

usr.update(

{'totalUpvotes': usr.get('totalUpvotes') - votes})

if upvote:

val = 'upvotes'

val2 = 'downvotes'

else:

val = 'downvotes'

val2 = 'upvotes'

if voter.get('\_id') in img.get(val2):

print(f'voter has already {val2} the image')

img.get(val2).remove(voter.get('\_id'))

img.update(

{val2: img.get(val2)})

print(f'un{val2}ed the image')

if voter.get('\_id') in img.get(val):

print(f'voter has already {val} the image')

img.get(val).remove(voter.get('\_id'))

img.update(

{val: img.get(val)})

print(f'un{val} the image')

else:

print(f'previous state of img upvotes: {img.get("upvotes")}')

print(f'previous state of img downvotes: {img.get("downvotes")}')

votes = img.get(val)

votes.append(voter.get('\_id'))

img.update({val: votes})

print(f'Adding {voter.get("\_id")} to img')

print(f'current state of img upvotes: {img.get("upvotes")}')

print(f'current state of img downvotes: {img.get("downvotes")}')

votes = len(img.get('upvotes')) - len(img.get('downvotes'))

# subtracting current post's upvotes from User's total upvotes

usr.update(

{'totalUpvotes': usr.get('totalUpvotes') + votes})

img.update({'totalUpvotes': votes})

Database.update(collection=Image.COLLECTION,

query={'img\_id': img.get('img\_id')},

update\_query=img)

Database.update(collection=User.COLLECTION,

query={'\_id': usr.get('\_id')},

update\_query=usr)

return True

return -1

return -2

return -3

@staticmethod

def getCategories(cat =None):

return Image.GetImgCategory(cat = cat)

@staticmethod

def updateUserInfo(user, \_id):

return Database.update(collection=User.COLLECTION,

query={'\_id': \_id}, update\_query=user)

@staticmethod

def uploadImage(\_cu,

categories,

image, img\_id,

description="", hashtags=[]):

usr = User.verifySession(\_cu)

if usr:

hashtags\_ = re.findall(r"#(\w+)", description)

if hashtags\_:

hashtags\_ = [hashtag.lower() for hashtag in hashtags\_]

hashtags\_.extend(hashtags)

img = Image(userID=usr.get('\_id'),

categories=categories, image=image,

description=description, tags=hashtags\_,

img\_id=img\_id)

img.saveImg()

Category.updateCategories(categories, inc = 1)

# Category.updateCategory(cat)

User.updateUploads(\_cu)

return True

return 0

@staticmethod

def deleteUser(\_id):

"""

Deletes User from Users Collection

"""

return Database.delete(collection=User.COLLECTION,

query={"\_id": \_id})

@staticmethod

def deleteAllUsers():

"""

Deletes All Users from Users Collection

Returns Number of Users Deleted

"""

return Database.delete\_all(collection=User.COLLECTION,

query={})

@classmethod

def getUser(cls, json):

"""

Takes json object as input and

Returns User Object

"""

return cls(\*\*json)

@staticmethod

def getUsers():

"""

Returns all users present in the

Users Collection

"""

return Database.find\_all(collection=User.COLLECTION,

query={})

@staticmethod

def updateUploads(\_cu, val=1):

usr = User.verifySession(\_cu)

if usr:

totalUploads = usr.get('totalUploads')

totalUploads += val

usr.update({'totalUploads': totalUploads})

Database.update(collection=User.COLLECTION,

query={"\_id": usr.get('\_id')},

update\_query=usr)

return True

@staticmethod

def changePassword(\_id, password):

usr = User.getUserByID(\_id)

if usr:

usr.update({'password': User.set\_password(password)})

Database.update(collection=User.COLLECTION,

query={"\_id": usr.get('\_id')},

update\_query=usr)

return True

else:

print(usr)

@staticmethod

def changeUsername(\_id, username):

usr = User.getUserByID(\_id)

if usr:

usr.update({'username': username})

Database.update(collection=User.COLLECTION,

query={"\_id": usr.get('\_id')},

update\_query=usr)

return True

@staticmethod

def changeAbout(\_id, about):

usr = User.getUserByID(\_id)

if usr:

usr.update({'about': about})

print(usr)

Database.update(collection=User.COLLECTION,

query={"\_id": usr.get('\_id')},

update\_query=usr)

return True

else:

print(usr)

@staticmethod

def changeCoverPhoto(\_cu, coverPhoto):

usr = User.verifySession(\_cu)

if usr:

usr.update({'coverPhoto': coverPhoto})

Database.update(collection=User.COLLECTION,

query={"\_id": usr.get('\_id')},

update\_query=usr)

return True

else:

print(usr)

@staticmethod

def changeProfilePicture(\_cu, profilePicture):

usr = User.verifySession(\_cu)

if usr:

usr.update({'profilePicture': profilePicture})

Database.update(collection=User.COLLECTION,

query={"\_id": usr.get('\_id')},

update\_query=usr)

return True

else:

print(usr)

@staticmethod

def changeDescription(description, img\_id):

hashtags = re.findall(r"#(\w+)", description)

if hashtags:

hashtags = [hashtag.lower() for hashtag in hashtags]

print('calling updateImgData')

return Image.updateImgData(img\_id, description, hashtags)

if \_\_name\_\_ == "\_\_main\_\_":

Database.initialize('Atypical')

**(b) Testing & Debugging:**

**1. Unit Tests:** Unit Tests have been made for assuring each and every component works as expected.

**2. Integration Tests:** Every component is Integrated then tested again for expected Output.

**3.** **Regression Tests:**  As Agile Model is being followed for building Atypical, many new features have been applied as needed and these features are then tested. Then it is integrated with existing components and tested again for expected output.

**Test Reports:**

1. Web App is tested with different kinds of Inputs and Output is Observed to be expected.
2. Collections and Logs are constantly checked while debugging and testing to detect any bugs and anomalies in the process while either updating or removing existing components or adding a new component in Atypical.
3. Database schema is made also maintained for preventing any unwanted behaviour.

**Chapter 5: Summary and Conclusions**

**5.1 Summary**

Atypical is a Web App where Users can download pictures publicly for free without any hassle. It’s made for people like Graphic Designers, Artists, Content Creators who can search images which need in their work and can download them for free. Images can be posted by anyone after creating an account on Atypical by Signing up.

It’s clean and elegant design helps people to use it conveniently. People can find images by going through the Categories assigned right after the image is uploaded to the server.

Atypical’s responsiveness enables it to be platform independent which simply means Users can conveniently access the whole site with their Computers and Smart phones. Users can up-vote and down-vote images which they like or dislike for giving feedback to the person who uploaded it.

**5.2 Objectives**

The objective of Atypical is to give Artists and Photographers a platform to share their images to the world so that they can download them for free. Images are Categorized right after they are uploaded which enables everyone to discover their content in no time. People need to have an account on Atypical to Upload Images.

That account also gives them ability to Up-vote and Down-vote images for feedback. In this application, all the Uploads are handled by Everypixel’s API which categorizes images and makes them search-able. Main features of Atypical are listed below:

* Responsive Web Design for making it easily accessible from devices like PCs and Smart Phones
* Dynamic Loading of Content for convenience and automated pagination
* Uses MongoDB which is the most popular NoSQL Database for storing Accounts and Images
* Automated Categorization of Image Uploads and key-wording them for making Images Search-able using Everypixel’s API
* Masonry Layout for Easy Navigation
* User Feedback in the form of Up-votes and Down-votes
* Image Compression with Anti-Aliasing for storing Image data
* Users can change their profile picture, email address, password and username along with their post’s description

**5.3. Scope**

Atypical is meant to provide a platform for Photographer, Artists or people who want to share their Pictures with the world for free even if they don’t get credit for it.

It will also provide people a platform where they can find any kind of image which they want, whether it’s an image of a dog or moon, or dog on moon. Atypical’s design makes it easy for everyone to use it, doesn’t matter if they are using it on their PCs or their Smart phones.

Atypical also gives the ability to give feedback to Owner by up-voting and down-voting images which enables the artists analyze how good they are becoming on taking pictures or creating designs.

**5.4. Atypical’s Limitations, Bugs and Scope for Future Development:**

* It still lags performance which is needed for is important when it comes to becoming a platform where large number of Users use the site. Algorithms for Storing and Retrieving data can be optimized.
* Atypical also needs work on the User Interface as it is static, even though it uses JavaScript for loading for some processes but in order to be 100% responsive and Dynamic, it needs to use a Frontend Framework like ReactJS, AngularJS or VueJS.
* The Image Compression algorithm and Categorization can be further optimized.
* It can also be become a social media platform where Artists can connect to each other and send messages to each other. Atypical is made scalable so implementation of the features is possible.
* Atypical can also be expanded furthermore for not just Stock Images but Videos, Music and Games too which are made available for everyone free of cost.