ECE - 1779

Assignment 1

Web Devlopment

Poroma Banerjee - 1005318764
Anshul Verma - 1004730703

Date due: 17^{th} February 2020

Date handed in: 17^{th} February 2020

Contents

1 Introduction

For this assignment, we developed a simple web application where a user can register, login and upload photos they want to run object detection on. The uploaded photos are then used for object detection via YoloV3 deep net [3]. These uploaded image and the image with objected detected boxes are stored in the server and the image information such as its file path is stored in the database. A user is able to view the images and its objected detected images in the home gallery after they have logged in.

This assignment has provided us with the experience of developing a web application using Python(Flask) and MySQL Workbench. We have deployed and run our application on an Amazon AWS EC2 instance. We have also opened port 5000 for our instance which enables our hosted web-application to be accessed via the internet.

In following sections, we have provided a user manual explaining the features of our web application. Section 3 explains our Relational Database Structure used for the web-application. We discuss the practicality of our application in Section 4. Finally, the deployment of the web application on the Amazon AWS EC2 instance is explained in Section 5.

2 Application User Manual

2.1 Accessing the Web Application

The default page is the login page as shown in Figure 1. It is the page a user will first encounter when accessing our web application. Already registered users will be able to login using their username and password, whereas new users will have the option to register.

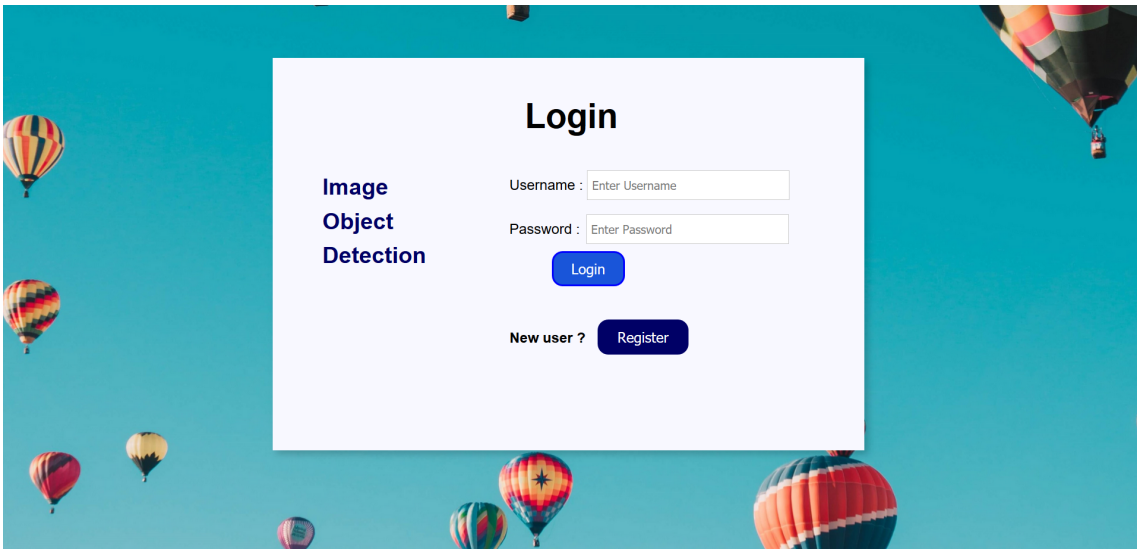


Figure 1: Login page.

The new users can choose the option to register by clicking on the button Register in Figure 2. A user willing to register will be asked to enter a unique (w.r.t names in database) username along with a password. After entering a valid username and password, Section 4, the user will be able to register. A

user is provided with additional options to reset the fields or to go back to the login page.

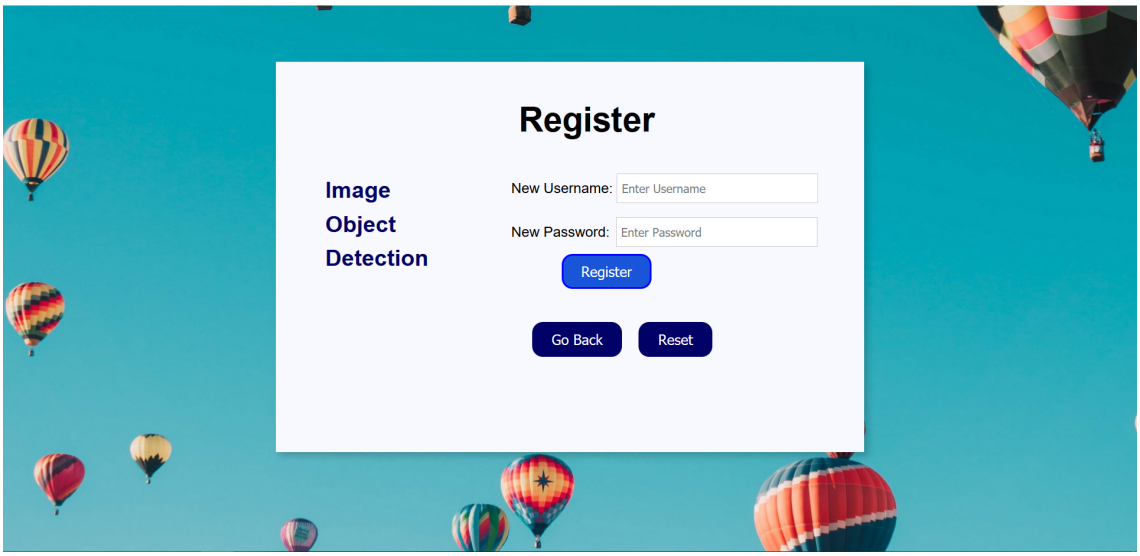


Figure 2: Register Page.

Upon successful registration the user is redirected to the login page with the username pre-filled with their registered username. An information is also flashed notifying their successful registration, like in Figure 3. In case of an error, the user remains in the registration page as is notified on the issue, Figure 4 and Figure 5.

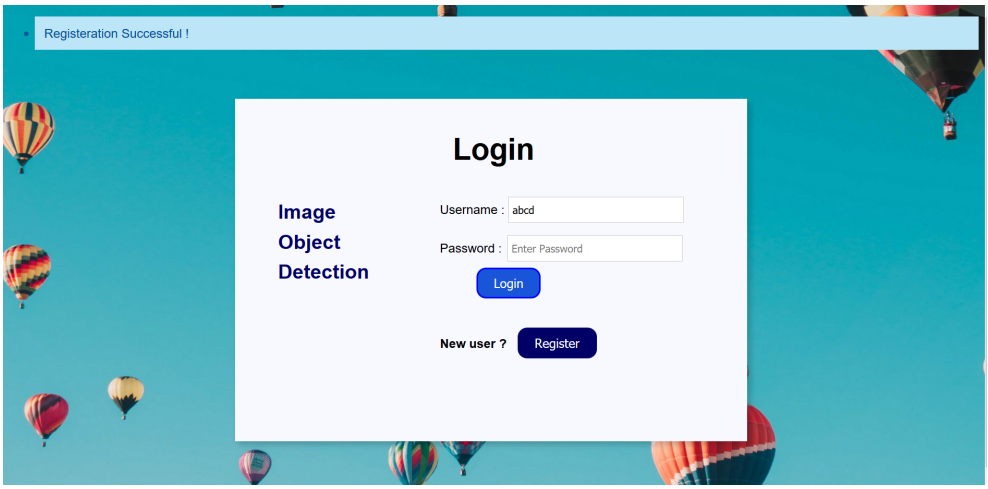


Figure 3: Login page with notification.

We can contrast Figure 4 and Figure 5 to see the style difference between an information and error flash message.

Upon successful login, the user is redirected to their home page with a gallery of the images they have uploaded, Figure 6. The gallery shows the thumbnails of the original photos uploaded by the user and by clicking on the image user can see the full-sized image and its object detected version, Figure 7 The user can also download either the original or the object detected image by clicking on them. If a user has no previous image, their gallery will be empty. Apart from the gallery, The user is provided with two

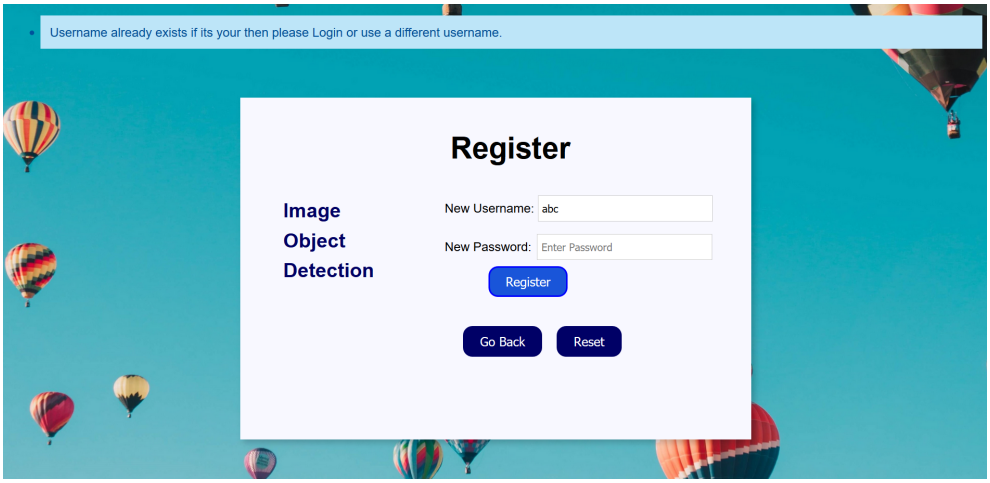


Figure 4: Registration page with notification.

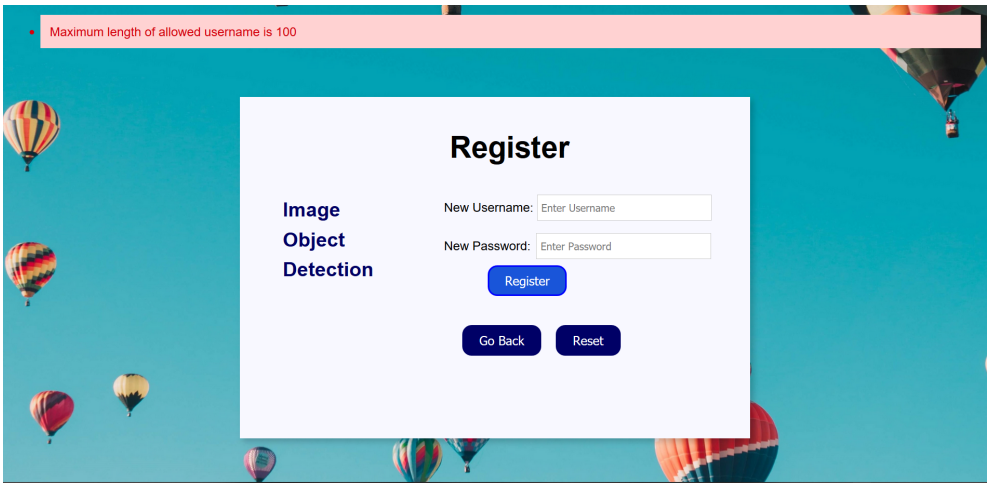


Figure 5: Register page with error.

buttons, upload and logout. Once a user has logged-out, they will need to login again in order to view their gallery.

If a user selects to upload an image they are redirected to the upload page, Figure 8 where they are asked to input their username and password again, along with the file the wish to upload. A user can use another users credentials to upload an image but the image will still be uploaded with the logged in user credentials because he has a session running and is trying to upload a pic. This was done to make sure a logged in user can not upload pics on another users gallery.

Once the user is able to successfully upload an image, the user is redirected to their home page and the newly uploaded image is added to their gallery along with other images, Figure 9. Object detection is performed during uploading process which makes the upload a bit slow, but this ensures that the users do not manually perform this task later.

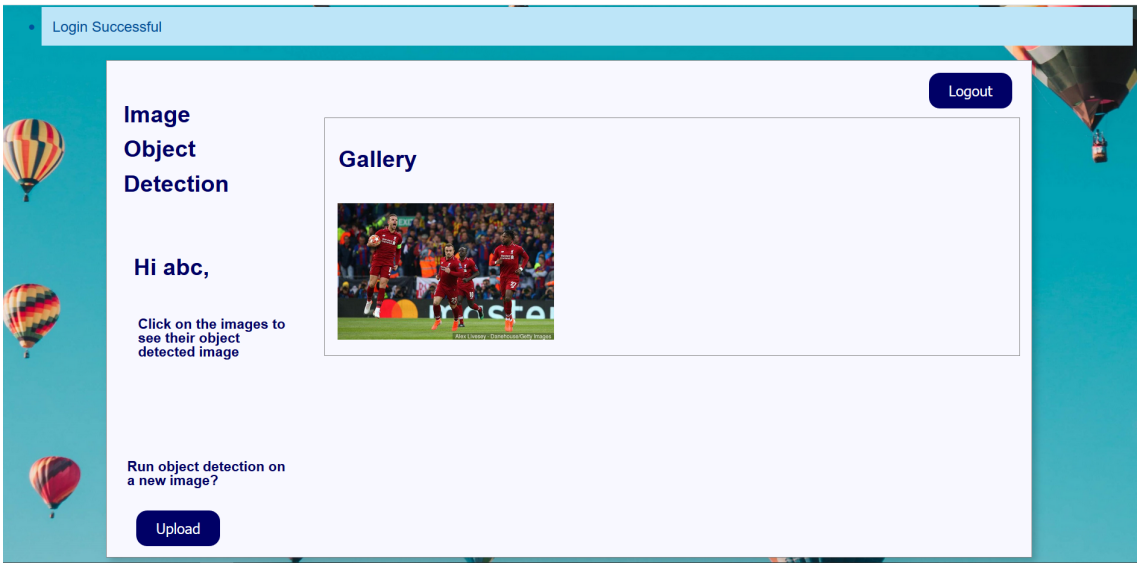
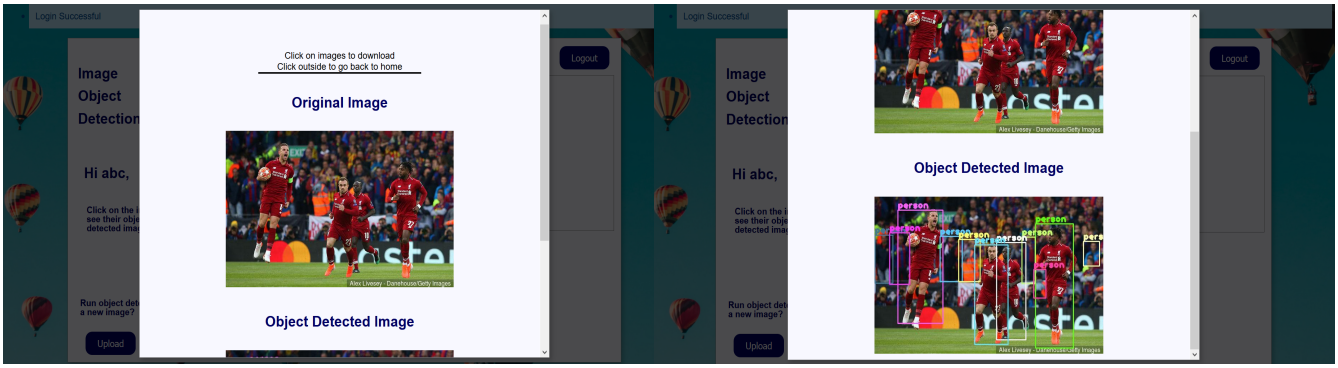


Figure 6: Home page.



(a) (b)

Figure 7: (a) Original Image, (b) Object Detected Image.

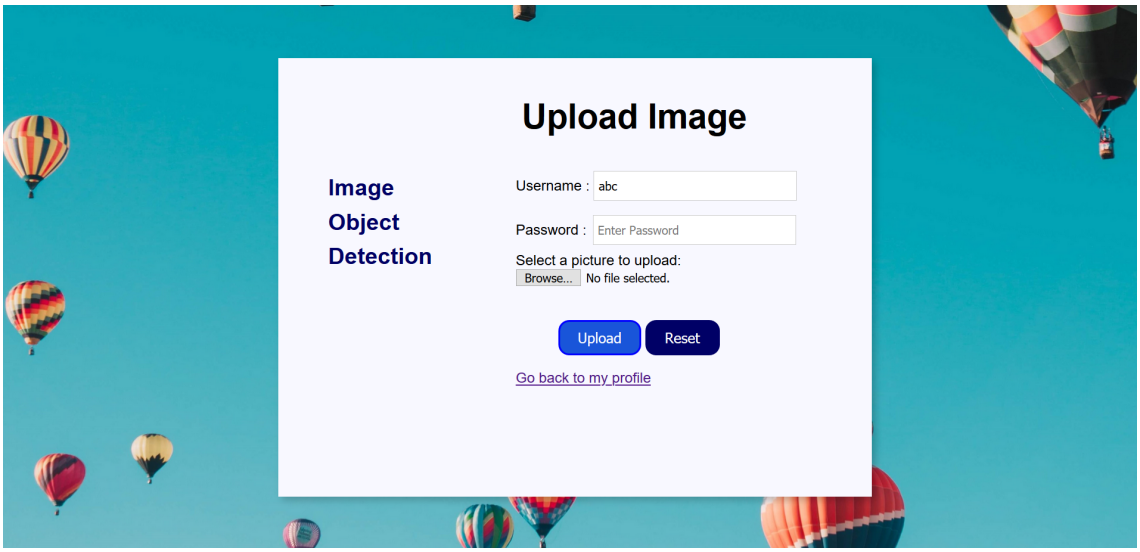


Figure 8: Upload page.

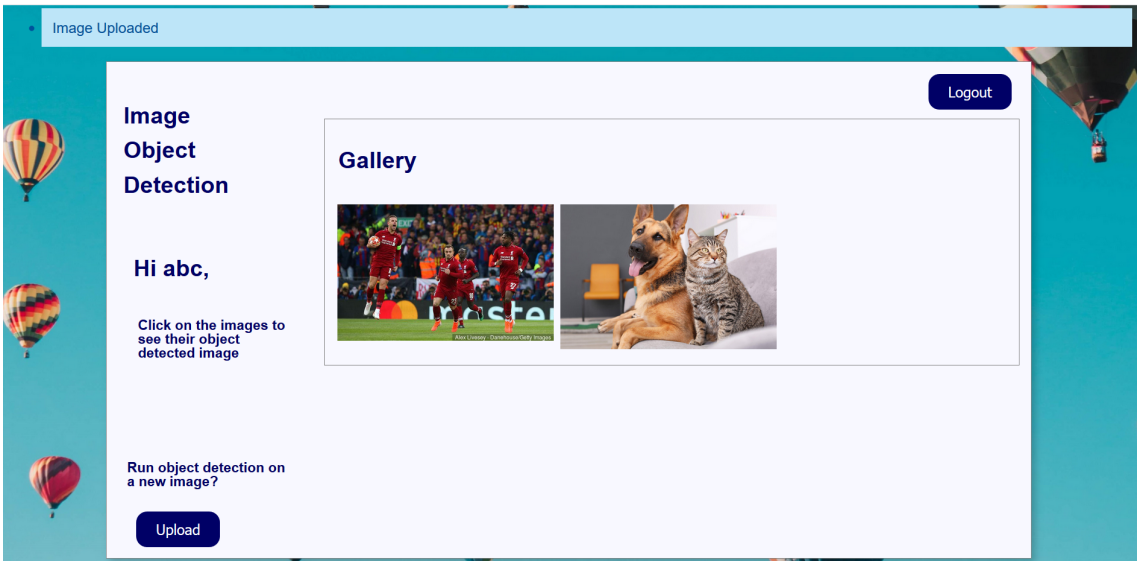


Figure 9: Image Uploaded.

2.2 Rest API

To simplify testing, our application (in addition to your web interface) includes two URL endpoints as APIs that makes it possible for the TA to automatically register users and upload photos. These endpoints should conform to the following interfaces:

Register:

```
relative URL = /api/register
method = POST
POST parameter: name = username, type = string
POST parameter: name = password, type = string
```

Upload:

```
relative URL = /api/upload
enctype = multipart/form-data
method = POST
POST parameter: name = username, type = string
POST parameter: name = password, type = string
POST parameter: name = file, type = file
```

The generator provided in the assignment was checked for the upload page and it works. We have uploaded pics using the generator provided but if more than one pics are to be uploaded then the object-detection is not perfect which we don't know the reason for also the response of the generator is always {} because we didn't know the valid return type of the functions for API extensions to get the response but it works and has been tested.

2.3 Image Object Detection

Our web application has used Yolo-v3 with Open CV for its image object detection. We have modified the original code provided by Sergio Canu [1] to fit our purpose. Yolo v3 [3] is a deep learning algorithm that can detect objects relatively quickly. All the weights, configuration and classes.txt files of the trained network are saved in the project so that the modal can just load in these weights and perform the object detection. The weights of this model were trained on the Coco-dataset [2].

The code was was tested for corner cases and has been modified to show labels within the image even if the box of the detected image is right on the edge of the image.

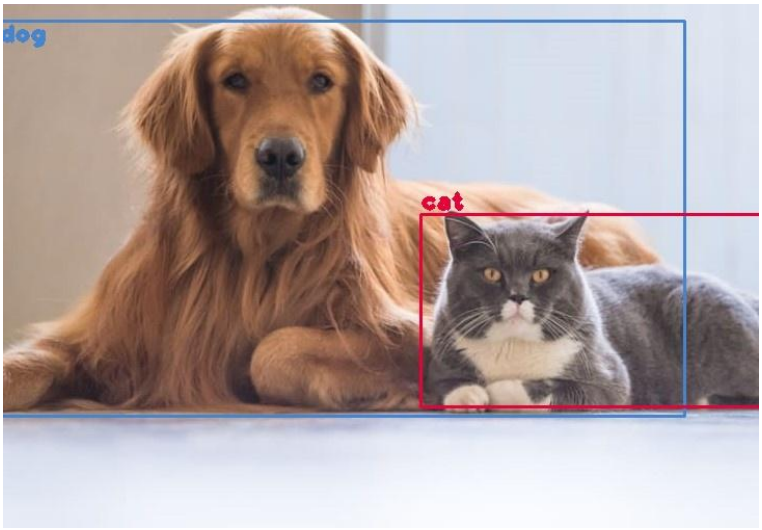


Figure 10: Corner Case Corrected.

3 Database Schema

3.1 MySQL

We have used MySQL Workbench to make our database. MySQL is an open-source Relational Database Management System. It provides visualization of database which helps understanding the database better. It is a GUI database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environment for the MySQL database system.

For the purpose of assignment we built a database(schema) shown in figure 11. The database has three tables. First table stores the user information (Userinfo - Table 1), a table which has all the image information stored in it (imgs - Table 2) and finally a table which stores the userid of a user along with the image id of the images that they have uploaded (user_has_imgs - Table 3).

The user_has_imgs table is essential, as we must know which image is associated with which user. Since the a user has can upload multiple images, therefore the relation between Userinfo and user_has_imgs is one to many. Since any image uploaded has a unique ID, the relation between imgs and user_has_imgs is one to one.

The maximum length of username can be 100 and thus username column is set to be varchar(100).

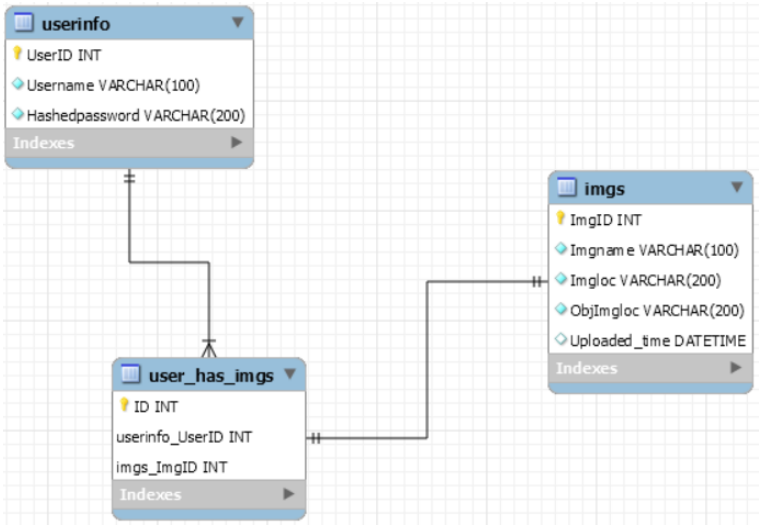


Figure 11: Database ER Diagram.

Table 1: Database Table - Userinfo

Row Name	**Properties**	**Description**
UserID	Primary Key, Unique Index, Auto Increment, Not Null, Data Type - Integer	Primary key of the table Is used as foreign key to create a relational database.
username	Unique Index, Not Null Data Type - String (100 characters)	this element is known to the user (unlike the UserID). we use this data to extract other information of the user such as UserID and password
Hashedpassword	Not Null Data Type - String (200 characters)	Stores the hashed password of the user. Is used to authenticate a user into the gallery page

This way the allowed username will always be accepted in the database. The hashed password will have a fixed character-length of 192 characters regardless of how big the password is. Therefore there is no character limit on the password.

So that all the files are accepted by the database, the filename character length is limited to 50 characters. This ensures that the other coloumns will also be within their range. We make a reasonable assumption that the user has no intention of uploading 10$\hat{2}$4 images of the same file name; which will be required in order to exceed the character length limit.

3.2 Image Storage

For our assignment, we need to store the original image and the obejct detected image uploaded by the user.

We have stored images uploaded by all the users in one directory and the corresponding object detected images in another directory. To differentiate between the uploads of different users we added UserID to

Table 2: Database Table - imgs

Row Name	**Properties**	**Description**
ImgId	Primary Key, Unique Index, Auto Increment, Not Null, Data Type - Integer	Primary key of the table Is used as foreign key to create a relational database.
Imgname	Unique Index, Not Null Data Type - String (100 characters)	This is the image filename uploaded by the user. If a previous file of the same name was uploaded by the user, the filename will get incremented. It is important to maintain the filename for the purpose of download and display
Imgloc	Not Null Data Type - String (200 characters)	Stores the directory location of the image file Is used to obtain the image from its location
ObjImgloc	Not Null Data Type - String (200 characters)	Stores the directory location of the object detected image file Is used to obtain the image from its location

Table 3: Database Table - user_has_imgs

Row Name	**Properties**	**Description**
ID	Primary Key, Not Null, Data Type - Integer	Primary key of the table Is used as foreign key to create a relational database.
userinfo_UserID	Foreign Key, Not Null, Data Type - Integer	Obtains the primary key of the table Userinfo Is used to connect the which user is associated with which image.
imgs_ImgID	Foreign Key, Not Null, Data Type - Integer	Obtains the primary key of the table imgs Is used to connect the which user is associated with which image.

the name of the image before saving it so that it can only be accessed by the user who uploaded it. Along with this a user can also upload different images with same name and the application will give each one of the a unique name so that none of the uploaded images are lost.

However, in the future, we plan on implementing a search option where user can search images using image name. So we plan on modifying the the image saving route to a unique saving route for every user, so that the search becomes easy (as now the search wont have to go through all the uploaded pics by all the users). Also, when a user wishes to download the image, we want the image name to be maintained. Hence, in future we have decided to create different directories for each user and store the images within

those.

Therefore the path to an image follows the following pattern:

```
Original Image Path :
                    \static\uploads\img\<UserID>_<Imagename>
Object Detected Image Path :
                  \static\uploads\obj_img\obj_<UserID>_<Imagename>
```

4 Security and Functionality

Our web application is hosted on Amazon AWS EC2 instance of type t2.small. Hence our web application may have certain performance limitations because of only one worker right noe and we assume that the user is patient for their outputs.

4.1 Security

Since any website that is dependent on input from an unknown user is prone to risks, we have the following security principles to ensure smooth functioning of our website. They have been implemented both on the server and browser side.

- Input fields can not be empty
- Passwords are hashed using salts and saved in the database.
- Images uploaded with the same name are treated as two different images.
- Certain reasonable input restrictions have been made. These are :
  - Image Size should be less than 100 MB
  - Only PNG and JPEG Image files will be allowed to upload
  - Username can't be more than 100 characters
  - Image Filename should be less than 100 characters
- A logged in user can only access to the images uploaded by them. To access the image one must know both the UserID of the user who uploaded the image and the image name itself. The gallery of a user will only be displayed once the user has logged in (hence validating the user).
- A logged in user can not upload images to other user account. (i.e. Even after logging in a user can fill another user's valid credentials in the Upload form and select to upload an image. In this case the image will be uploaded to the logged in users gallery and not on the other user's gallery.)

4.2 Functionality

Our web application is responsive and maintains its integrity in different screen sizes. The interface has been tested on Google Chrome, Microsoft Edge, Firefox and UC Browser. All functions run smoothly. But in Firefox after logging out and going back the user is logged back in. This happens because Firefox maintains cookies and we didn't want to clear cache.

The object detection has been run on various images and performs adequately to the best of our knowledge.

By default a Flask session times-out in 31 days. But for our application we have changed it to 24 hours and our web application can run for multiple users at the same time but it will be slow at the moment because of only one worker.

5 Deployment

5.1 Connecting to the EC2 instance

The Amazon console credentials are provided in the credentials.txt file. The EC2 instance used for the assignment is named assignment1 and the keypair.pem can be used to ssh into the instance. The keypair.pem is also provided and it has read only mode so it can directly be transferred to the .ssh directory and used to ssh into the EC2 instance. The password to connect to the instance via tunnel is 'ece1779pass'

Use the following commands in the terminal to connect to the EC2 instance after starting it.

```
To connect  :
                ssh -i .ssh/keypair.pem ubuntu@<ip of EC2 instance>


To connect via tunnel :
   ssh -i .ssh/keypair.pem ubuntu@<ip of EC2 instance> -L 5901:localhost:5901
```

5.2 Locally Hosting The Webapp

The instance can be accessed by using the keypair file and on its desktop it has a directory named interactive which contains a directory called app holding the files used to run the web-app along with this the instance directory also contains weights, configuration and classes for yolov3 and a venv which has all the essential packages installed to run the web-app. The directory also has run.py file which can be used to locally host the webapp (need to run it after activating venv).

5.3 Hosting The Webapp

The desktop of the instance also has a start.sh which can be run in the command line from desktop ./start.sh. It will host our webapp on the port 5000 which has been made open for the instance. The mode of start.sh is changed to -x so its not needed to bash the file.

Make sure that the database is running before hosting the webapp the password of the database is 'ece1779pass'.

To access the website after hosting it just connect to internet run a browser and go to the url <ip_of_EC2_instance>:5000 and it will direct you to the login page

6 Appendix I - Code Documentation

The following section consists details of the python files that were used and the functions contained within these files.

6.1 config.py

This file consists of database configurations. It also contains the path information for image storage and the permissible file type (PNG, JPG or JPEG) information.

6.2 Hashing_n_Checking.py

This file contains function to hash a password and verify a user entered password with their respective hashed password.

FUNCTIONS:

hash_password(password):

Takes a string password, returns the hashed password along with its salt. The first 64 characters are the salt and the next characters are the hashed password.

returns (salt + pwdhash).decode('ascii')

verify_password(stored_password, provided_password):

Checks if the provided_password matched the hashed password with its salt.

returns pwdhash == stored_password

6.3 objdetect.py

The file consists of the code that performs object detection. The pre-requisite files are "yolo.txt","yolo.cfg", yolo.weights".

FUNCTIONS:

def objdetect(img, ext):

The function inputs the image extension and Image opened with PIL. The image is converted to desired form by using cv2.cvtColor and is then passed through YOLOv3 network for object detection. The Resulting image with boxes for detected objected is then encoded into bytes and is sent to the calling function for saving it using (pillow) PIL.Image. The function returns encoded image its dimensions and a message corresponding to success or failure of object detection process.

returns img_encoded, height, width, msg

6.4 verification.py

This file contains functions to check if the file is of allowed extensions and assigns a unique filename to each uploaded image. It also contains a function used for authenticating users.

FUNCTIONS:

def Unique_Name(cursor, ID, name, i = 0):

Checks if the filename provided by the user is already in the database. If the filename exists then an integer is appended to the filename. The integer is subsequently incremented if more filenames with the same name are uploaded. This function ensures that a unique filename is returned.

returns unique_name

def allowed_file(filename):

The function inputs the filename and checks if it of an allowed extension. This function returns a boolean value determining if the file is of allowed extension or not.

returns True or False

6.5 api_extensions.py

This file the api extensions for registration and uploading.

FUNCTIONS:

def add_user():

Gets the input on being called and tries to add the new user into the database.

returns response

def allowed_file(filename):

Gets the input and tries to upload images for a user.

returns response

6.6 main.py

This file consists of all the routes defined for the webapp which can be visited by the user.
FUNCTIONS:
def connect_to_database():
Connects to the MySQL database.

def get_db():
Gets the MySQL database.

def teardown_db(exception):
Closes the db once the session is closed.

def main():
This function is the login function. It cross checks whether the user input is a valid account and redirects to the home page if its correct.

def register():
the function adds the user input into the database after performing various checks to determine if the user inputs are valid.

def home(username):
Displays the home page (with gallery). Shows all the images uploaded by the currently active user. If the session has timed out, they are redirected to the login page.

def upload(username):
Obtains the file input from the user and adds the image information into the database after checking the user credentials and ensuring that the file input is of a valid form. Once the file is accepted, object detection is run on the file and the objected detected image is saved and the object detected image's file is added to the database. After uploading the user is returned to their home page.

def logout():
Clears session and redirects to the login page.
returns redirect(url_for('main'))

References

[1] Sergio Canu. Yolov3.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[3] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.