

# **Artificial Intelligence (CSC16101)**

*Winter Semester (2019-20)*

## **PROJECT REPORT**

BACHELOR OF TECHNOLOGY  
6<sup>th</sup> Semester



*Department of Computer Science and Engineering*

**INDIAN INSTITUTE OF TECHNOLOGY  
(INDIAN SCHOOL OF MINES)  
DHANBAD**

**TITLE: - Garbage Management System**

**OBJECTIVE: - Geo Tracking of Waste in city and Triggering Alerts.**

**ABSTRACT: -**

### **PROBLEM STATEMENT**

Geo Tracking of Waste in city and Triggering Alerts.

#### **Problem in detail, our approach to the problem, our solution:**

Huge amount of wastes lying in different parts of city makes it really hard for the municipal workers to know which places really require urgent cleaning. We are providing a solution with help of machine learning, artificial intelligence and web development which will allow everyone in the city to notify the local municipality of the places which require at most attention. We give every person the ability to just click the images of waste lying around from their mobile phones and upload it to our website. This is the only thing they are required to do. The images they upload will be uploaded along with the “Geotag “, i.e. **we will have the image along with its location**. The image is stored in our database.

Now, we have used a machine learning model and trained it to predict **whether a given image contains garbage or not**. We input the given image to our trained model and based on the result we take further actions. We have used various algorithms to train our learning model and have achieved an accuracy of nearly 94.5%. Now, we show all the locations on our website which are dirty based on the uploaded images from that particular location.

### **WORKING PRINCIPLE:**

#### **1. Data Collection**

- **Tech Stack Used:**
  - Beautiful Soup 4
  - Urllib

We are scrapping images from the internet using python script written with the help of Beautiful soup 4 and Urllib libraries. Beautiful soup is a python library used for pulling data out of HTML and XML files. Urllib is a package consisting of various modules such as urllib.request for opening and reading URLs, urllib.parse for parsing URLs. Here, we are using the parser as urllib.parse and beautiful soup 4 for data extraction.

Following is the code to extract all the images from a website. Libraries required to use image extraction code.

```
] import requests
import os
from tqdm import tqdm
from bs4 import BeautifulSoup as bs
from urllib.parse import urljoin, urlparse
```

The is\_valid() function takes the address of the website from which the images should be extracted, and checks if the website is valid or not.

The `get_all_images()` function grabs all image URLs of a web page.

```
def is_valid(url):  
    parsed = urlparse(url)  
    return bool(parsed.netloc) and bool(parsed.scheme)
```

The `get_all_images()` function grabs all image URLs of a web page.

```
[ ] def get_all_images(url):  
    soup = bs(requests.get(url).content, "html.parser")  
    urls = []  
    for img in tqdm(soup.find_all("img"), "Extracting images"):  
        img_url = img.attrs.get("src")  
        if not img_url:  
            continue  
        img_url = urljoin(url, img_url)  
        try:  
            pos = img_url.index("?")  
            img_url = img_url[:pos]  
        except ValueError:  
            pass  
        if is_valid(img_url):  
            urls.append(img_url)  
    return urls
```

So, we have extracted all the image URLs, now we download image files from the web page in python.

The `download()` does this part for us:

```
def download(url, pathname):  
    if not os.path.isdir(pathname):  
        os.makedirs(pathname)  
    response = requests.get(url, stream=True)  
  
    file_size = int(response.headers.get("Content-Length", 0))  
  
    filename = os.path.join(pathname, url.split("/")[-1])  
  
    progress = tqdm(response.iter_content(1024), f"Downloading {filename}", total=file_size, unit="B", unit_scale=True, unit_divisor=1024)  
    with open(filename, "wb") as f:  
        for data in progress:  
            f.write(data)  
            progress.update(len(data))
```



```
for img in imgs:  
    download(img, path)
```

## 2. Machine Learning Model

- **Tech Stack Used**

- Python
- Keras API
- TensorFlow
- NumPy
- Pandas

Since our project is based on classifying images, we are using Convolution Neural Networks for our machine learning model. The Convolution Neural Networks are also known as ConvNets / CNNs. Three main type of layers are used to build ConvNet Architecture: 1. Convolution Layer 2. Pooling Layer 3. Fully-Connected Layer.

### Pre-processing:

### Libraries required

```
In [1]: import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
import string
import os
from PIL import Image
import glob
import keras
import tensorflow as tf
from pickle import dump, load
from tqdm import tqdm_notebook as tqdm
from time import time
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector, \
                                Activation, Flatten, Reshape, concatenate,
                                Dropout, BatchNormalization
from keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.layers.merge import add
from keras.applications.inception_v3 import InceptionV3

from keras.preprocessing import image
from keras.models import Model
from keras import Input, layers
from keras import optimizers
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

Using TensorFlow backend.
```

We have extracted the features of the images into a vector of numbers. We then resized the input image to size 299\*299.

```
In [0]: def preprocess(image_path):
# Convert all the images to size 299x299 as expected by the inception v3 model
img = image.load_img(image_path, target_size=(299, 299))

x = image.img_to_array(img)
# Add one more dimension
x = np.expand_dims(x, axis=0)

x = preprocess_input(x)
return x
```

The “preprocess” function defined above takes image as input and pre-process the image according to the inception model.

After resizing we encode the image into its feature vector.

```
In [0]: # Function to encode a given image into a vector of size (2048, )
def encode(image):
    image = preprocess(image)
    v = model_new.predict(image)
    v = np.reshape(v, v.shape[1])
    return v
```

The encode () function defined above takes an image as input and returns a vector of size 2048 and it also stores the characteristics of the image.

Using the below code, we encode all the images into a vector representation of size 2048.

```
In [0]: encoded_garbage = a[]
for img in tqdm(garbage):
    encoded_garbage.append(encode(img))

encoded_nongarbage = []
for img in tqdm(nongarbage):
    encoded_nongarbage.append(encode(img))
```

Now the dataset is divided into two sets.

1. Train set – This set is used to train the model.
2. Test set – This set is used to evaluate the model.

We are using train\_test\_split function of sklearn library to split our dataset into train and test sets.

```
In [0]: from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state = 42)
```

We have then normalized the training and test data set.

```
In [0]: X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Since our dataset consist of around 2000 samples, we are using **Image Data Augmentation technique** to increase the dataset.

```
train_datagen= ImageDataGenerator(rescale = 1./255.,
                                   rotation_range=40,
                                   width_shift_range = 0.2,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizaontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1.0/255.)

train_generator = train_datagen.flow_directory(train_dir,
                                                batch_size = 20,
                                                class_mode = 'bianary',
                                                target-size = (150, 150))

validation_generator = test_datagen.flow_from_directory(validation_dir,
                                                         batch_size = 20,
                                                         class_mode = 'binary',
                                                         target_size = (150, 150))
```

For our project we are building two ConvNet architecture.

- i) CNN model build from scratch.
- ii) Model based on Inception-V3

## 2.1 CNN model build from scratch.

We have built a 5-layer Convolution Neural Network for classifying the images. We are using Keras API for implementation. The ConvNet architecture is

**[INPUT – CONV2D – MAX\_POOLING – CONV2D – MAX\_POOLING – DROPOUT – FLATTEN – FULLY CONNECTED – FULLY CONNECTED – FULLY CONNECTED (SIGMOID LAYER) – OUTPUT]**

```

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(4, 4), activation='relu', input_shape=(299, 299, 3), padding = 'SAME'))
model.add(MaxPool2D(padding='SAME', pool_size=(4, 4)))
model.add(Conv2D(filters=64, kernel_size=(4, 4), activation='relu', padding='SAME'))
model.add(MaxPool2D(padding='SAME', pool_size=(4, 4)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))

```

```
[3] model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 299, 299, 32)	1568
max_pooling2d_1 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 64)	32832
max_pooling2d_2 (MaxPooling2D)	(None, 19, 19, 64)	0
dropout_1 (Dropout)	(None, 19, 19, 64)	0
flatten_1 (Flatten)	(None, 23104)	0
dense_1 (Dense)	(None, 1024)	23659520
dense_2 (Dense)	(None, 512)	524800
dense_3 (Dense)	(None, 1)	513
Total params: 24,219,233		
Trainable params: 24,219,233		
Non-trainable params: 0		

fig – CNN model

## 2.2 Model based on Inception-V3

A pre-trained model is preferred when we have less computation resources as well as less data samples. The use of pre-trained model by modifying the bottleneck layer along with the addition of extra sigmoid layers for classification is termed as Transfer learning. There are a lot of pre-trained models that are open-sourced such as VGG-19, Inception-V3. Here, we have used Inception- V3 model in our project.

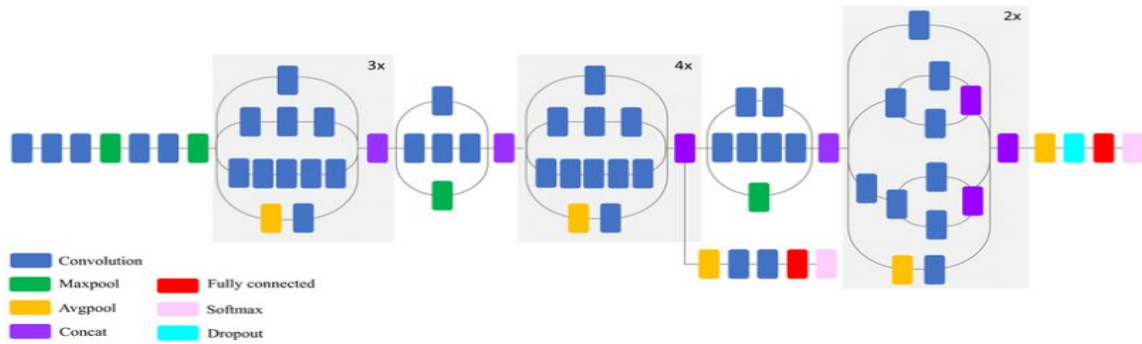


Fig- INCEPTION-V3

The Inception v3 model has been trained on ImageNet dataset. The model attains greater than 78.1% accuracy on the ImageNet dataset in about 170 epochs on each of v2-8, v2-128 and v2-512 configurations. The network is 48 layers deep.

Importing the Inception-v3 model

```
In [0]: model_new = InceptionV3()
model_new = Model(model_new.input, model_new.layers[-2].output)
model_new.load_weights('Models/inception.h5')
```

We have then added a 5 layered Neural Network with **relu** as its activation function which has been fine-tuned with appropriate drop out values.

```
In [0]: model = Sequential()
model.add(Dense(1024, activation='relu', input_dim = 2048))
model.add(Dropout(0.4))
model.add(Dense(1024, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc', f1_m])
```



We have then finally called the `fit_generator()` function with epoch value = 100.

In `fit_generator()`, we do not pass the X and Y directly, instead they are imported from a generator. This is done because it is written in keras documentation that generator is used when we want to avoid duplicate data while using multiprocessing.

```
callbacks = myCallback()
history = model.fit_generator(
    train_generator,
    validation_data = validation_generator,
    steps_per_epoch = 100,
    epochs = 100,
    validation_steps = 50,
    verbose = 2,
    callbacks=[callbacks])
```

The snapshot of last 10 epoch of the training is shown below-

```
Epoch 91/100
1471/1471 [=====] - 0s 175us/step - loss: 0.0116 - acc: 0.9952 - val_lo
ss: 0.1802 - val_acc: 0.9451
Epoch 92/100
1471/1471 [=====] - 0s 173us/step - loss: 0.0058 - acc: 0.9986 - val_lo
ss: 0.1948 - val_acc: 0.9512
Epoch 93/100
1471/1471 [=====] - 0s 179us/step - loss: 0.0013 - acc: 1.0000 - val_lo
ss: 0.2276 - val_acc: 0.9512
Epoch 94/100
1471/1471 [=====] - 0s 181us/step - loss: 0.0062 - acc: 0.9973 - val_lo
ss: 0.1913 - val_acc: 0.9451
Epoch 95/100
1471/1471 [=====] - 0s 183us/step - loss: 0.0094 - acc: 0.9973 - val_lo
ss: 0.1843 - val_acc: 0.9512
Epoch 96/100
1471/1471 [=====] - 0s 151us/step - loss: 0.0043 - acc: 1.0000 - val_lo
ss: 0.2357 - val_acc: 0.9512
Epoch 97/100
1471/1471 [=====] - 0s 160us/step - loss: 0.0029 - acc: 1.0000 - val_lo
ss: 0.2486 - val_acc: 0.9512
Epoch 98/100
1471/1471 [=====] - 0s 175us/step - loss: 0.0036 - acc: 0.9986 - val_lo
ss: 0.2213 - val_acc: 0.9390
Epoch 99/100
1471/1471 [=====] - 0s 191us/step - loss: 0.0017 - acc: 0.9993 - val_lo
ss: 0.2317 - val_acc: 0.9512
Epoch 100/100
1471/1471 [=====] - 0s 163us/step - loss: 0.0077 - acc: 0.9993 - val_lo
ss: 0.2385 - val_acc: 0.9451
```

We have finally achieved the **accuracy of 94.5%** on the validation dataset.

We have built a `predict()` function which takes input as an image, and predicts if the image contains garbage or not.

The predict function:

```
In [0]: def predict(filename):  
        x = plt.imread(filename)  
        plt.imshow(x)  
        v = encode(filename)  
        pred = model.predict(v.reshape(1, v.shape[0]))  
        if pred[0] == 1:  
            return "Garbage in the Image"  
        else:  
            return "Clean city"
```

The Output of the above predict functions:

```
In [0]: predict("dataset/Garbage/p3-jy23-18.jpg")
```

```
Out[0]: 'Garbage in the Image'
```



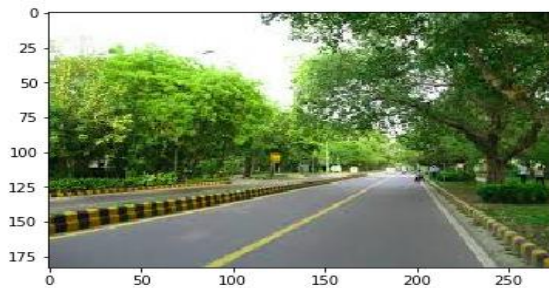
```
In [0]: predict('abcd.jpg')
```

```
Out[0]: 'Garbage in the Image'
```



```
In [28]: predict("image.jpeg")
```

```
Out[28]: 'Clean area'
```



Now, we serve our model through a web-app.

### 3. Website

#### Tech Stack used:

- Front-End:
  - HTML
  - CSS
  - Bootstrap
  - JavaScript
- Back-End:
  - Node.js
  - MongoDB
  - Python
  - TensorFlow
  - Keras
  - NumPy
  - Pandas

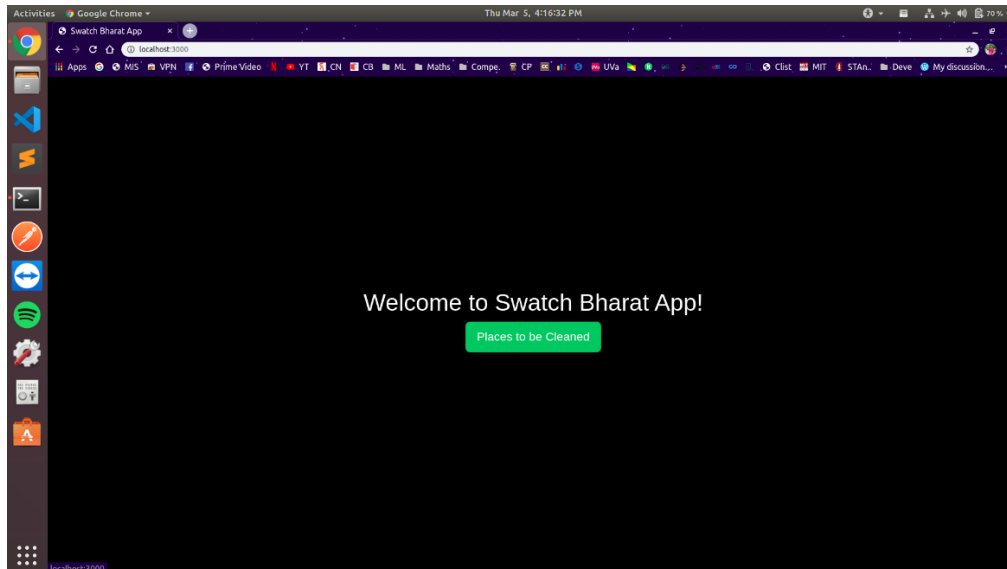
All Node frameworks and packages are as follows:

```
"dependencies": {  
  "body-parser": "^1.19.0",  
  "child_process": "^1.0.2",  
  "connect-flash": "^0.1.1",  
  "ejs": "^3.0.1",  
  "express": "^4.17.1",  
  "express-session": "^1.17.0",  
  "method-override": "^3.0.0",  
  "mongoose": "^5.9.2",  
  "multer": "^1.4.2",  
  "passport": "^0.4.1",  
  "passport-local": "^1.0.0",  
  "passport-local-mongoose": "^6.0.1",  
  "path": "^0.12.7"  
}
```

## Working:

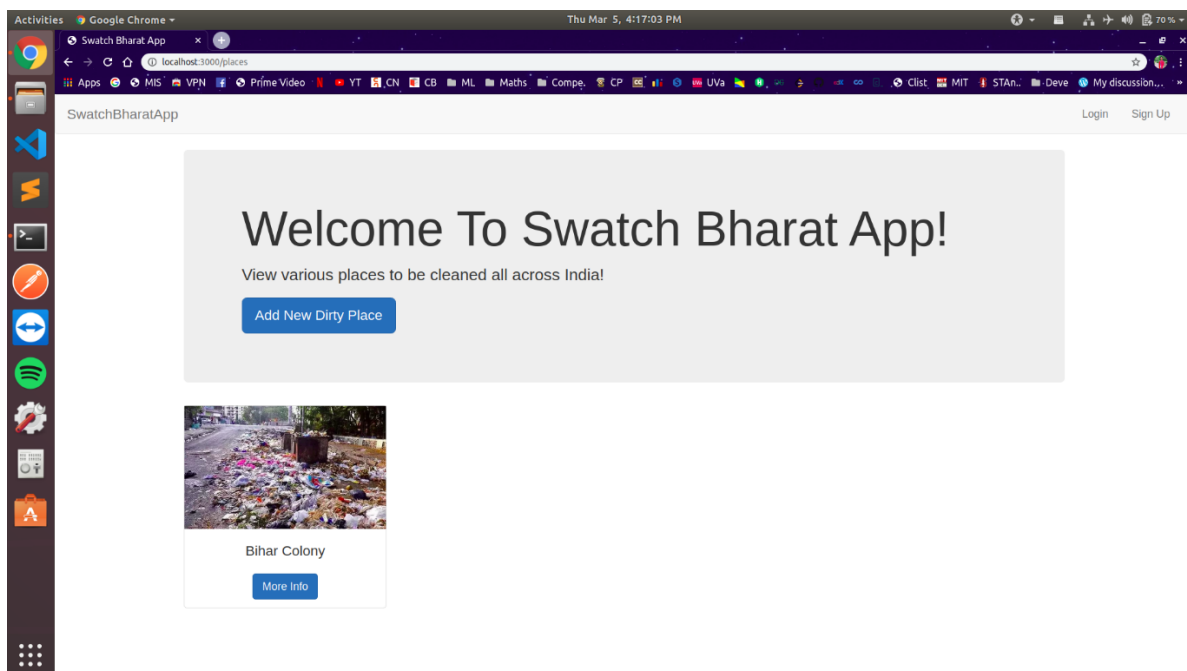
### Landing Page:

The website first lands on the landing page where we get the button to click and to redirect us to main page of the web app.



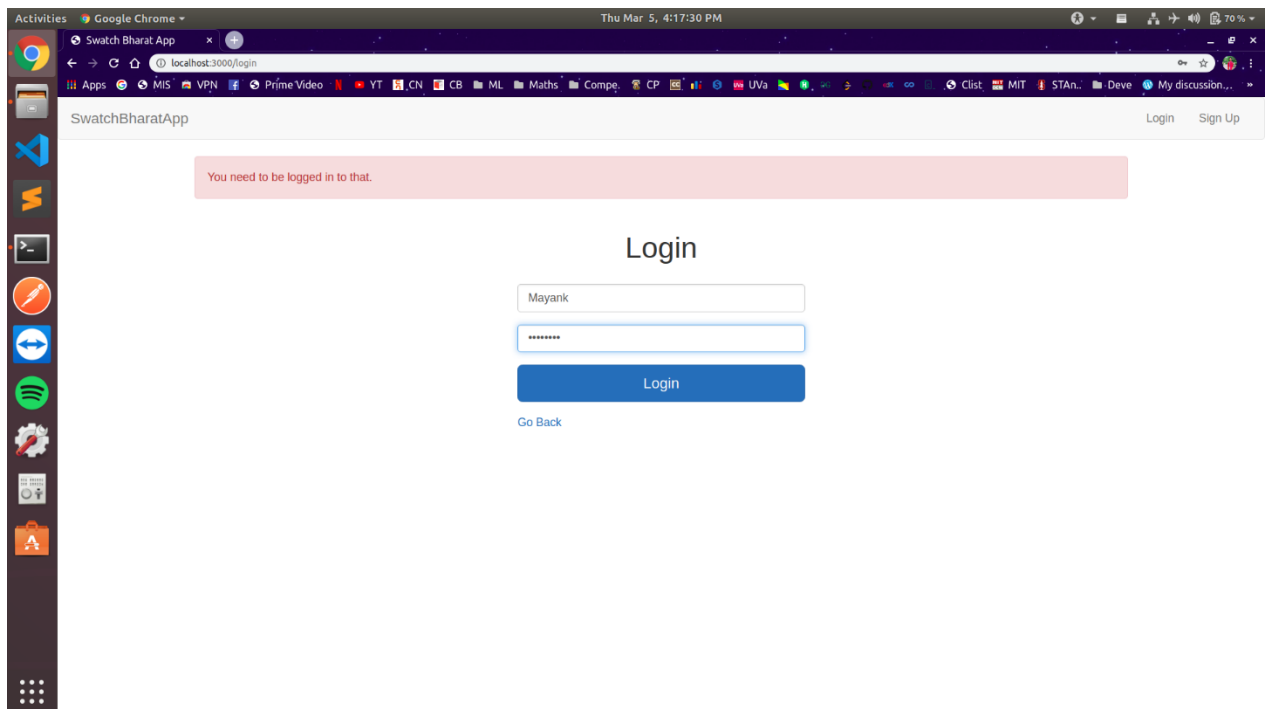
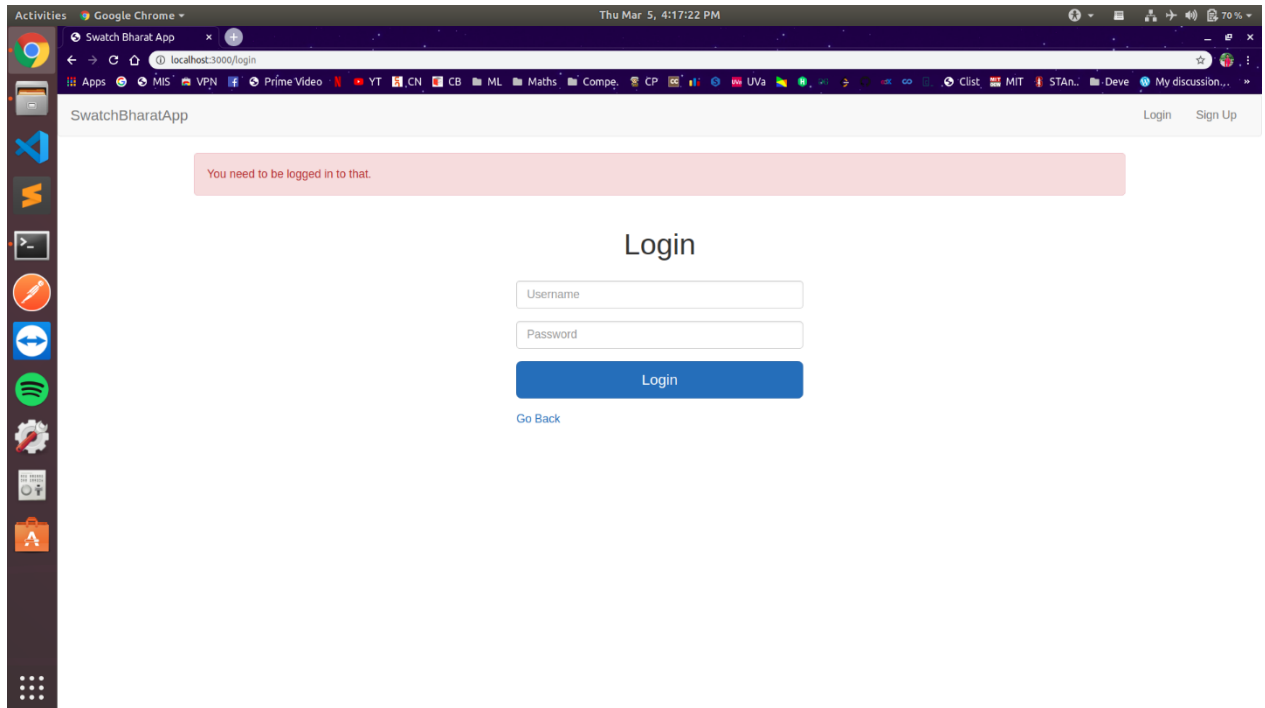
### Main Page:

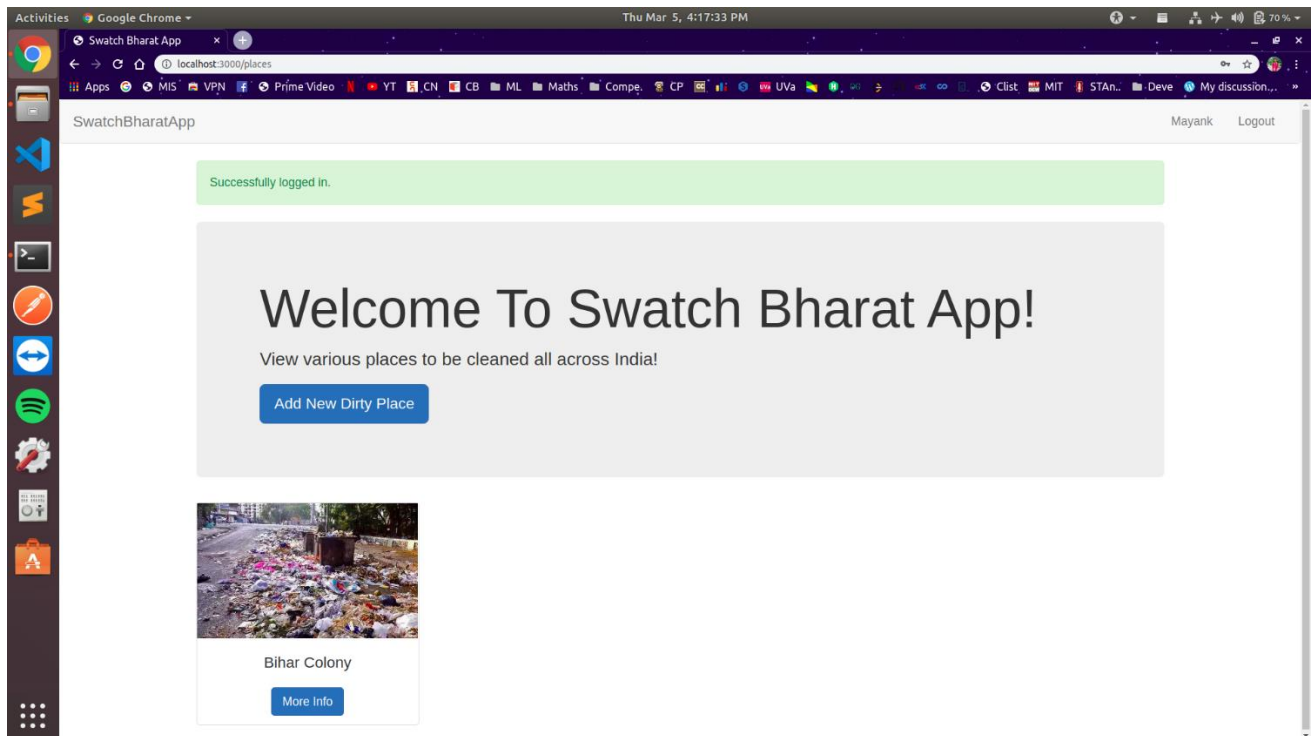
The main page (/places) shows all the dirty places which have been uploaded by different users.



From here we can upload new images to the servers from our local machine using "Add New Dirty Place" button. But for uploading you need to be logged in as a user so every uploaded image can be tracked back to the user.

## Login Page and steps:





After logging in we can upload the images to the server.

When the image is uploaded to the server it is stored in a file at the server and the corresponding path is stored in the database. For a place following schema has been adopted:

```
var placeSchema = new mongoose.Schema({
  name: String,
  image: String,
  description: String,
  comments: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Comment"
    }
  ],
  author: {
    id: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User"
    },
    username: String
  },
  garbageVal: String
});
```

“image stores” the path to the image

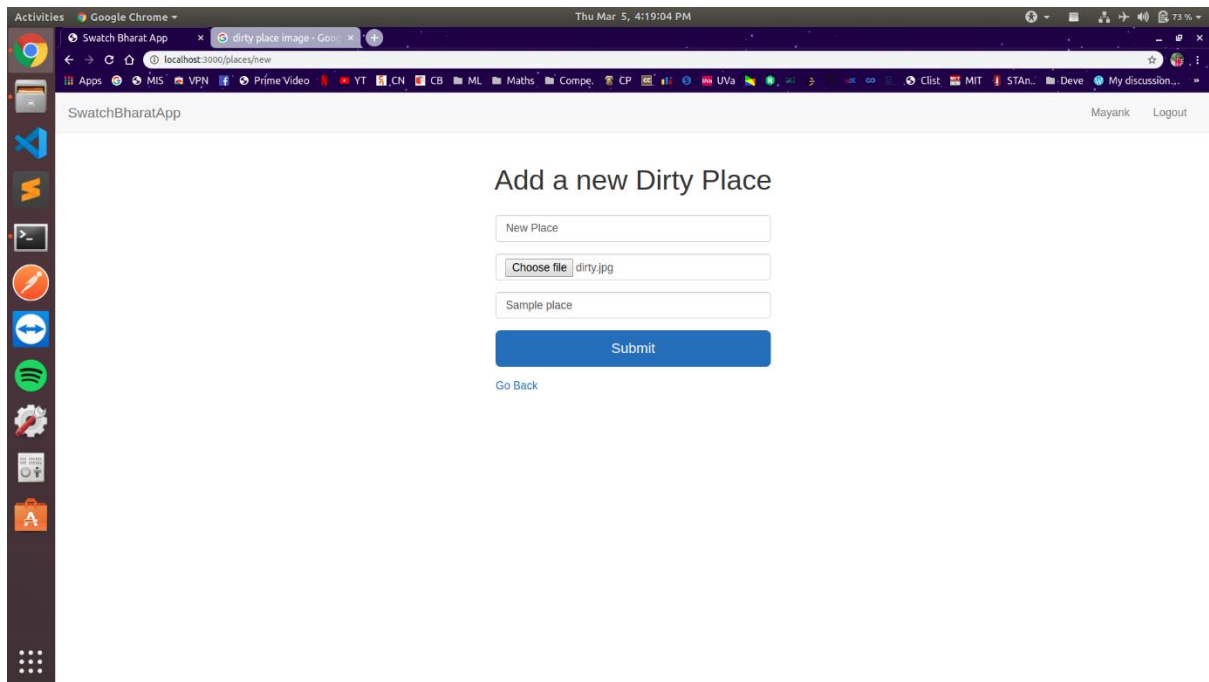
“garbageVal” stores the string “Garbage found” if image is detected by the backend script.

If garbage is detected in the image then the reserved by the server and thus the corresponding entry is shown on the web page.

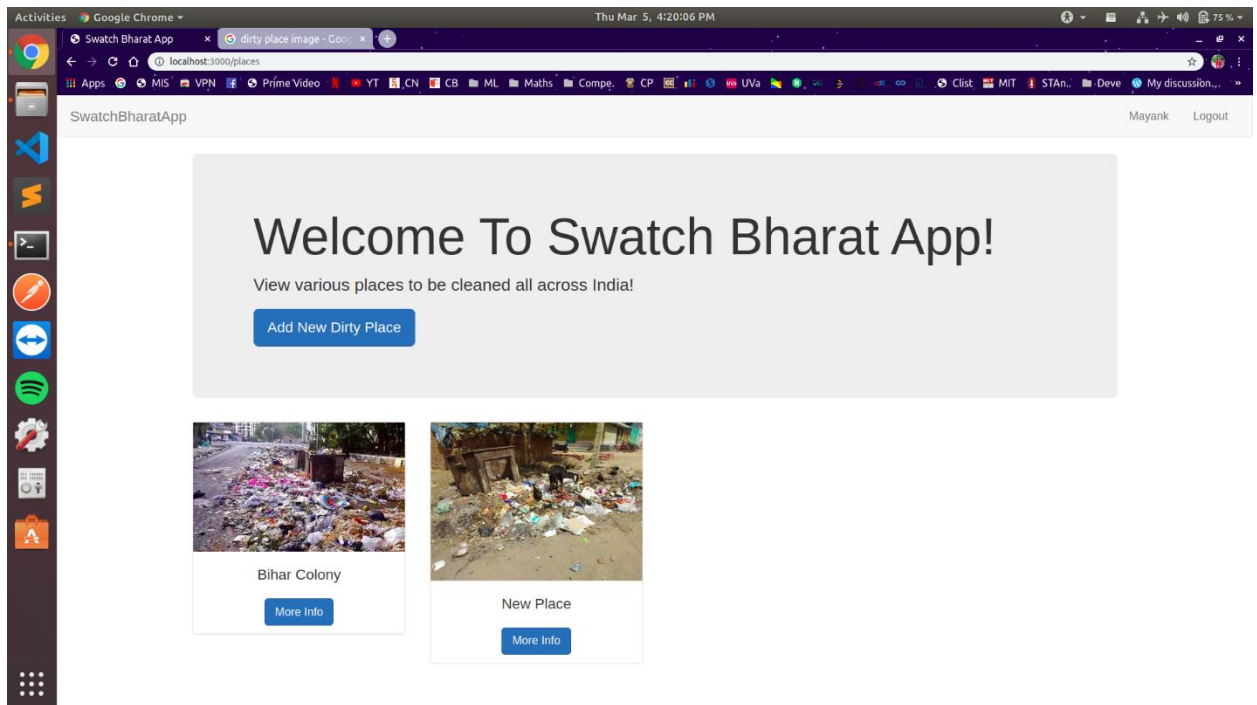
Else the image is discarded and a message is displayed to the user that no garbage was detected.



## Upload page and uploading images:

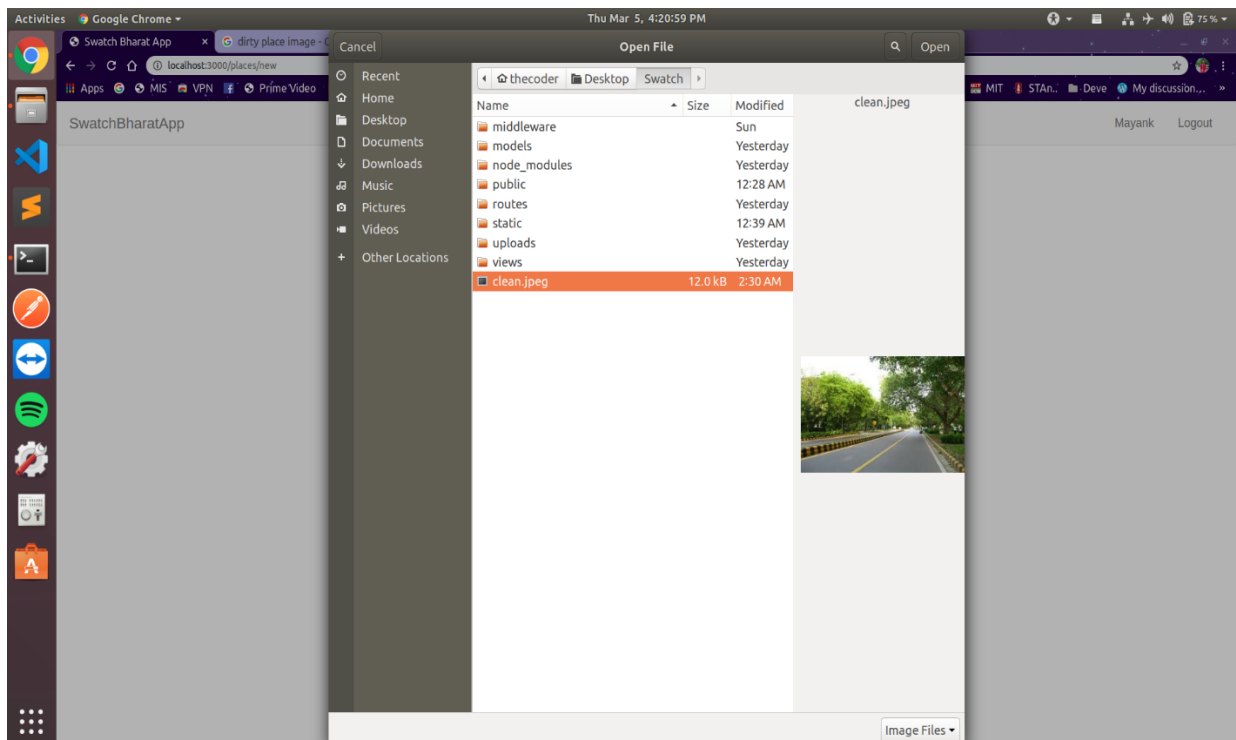


(dirty.jpg)



Since garbage has been found in the image the image with corresponding details are stored in database and corresponding entry is shown on the main page.

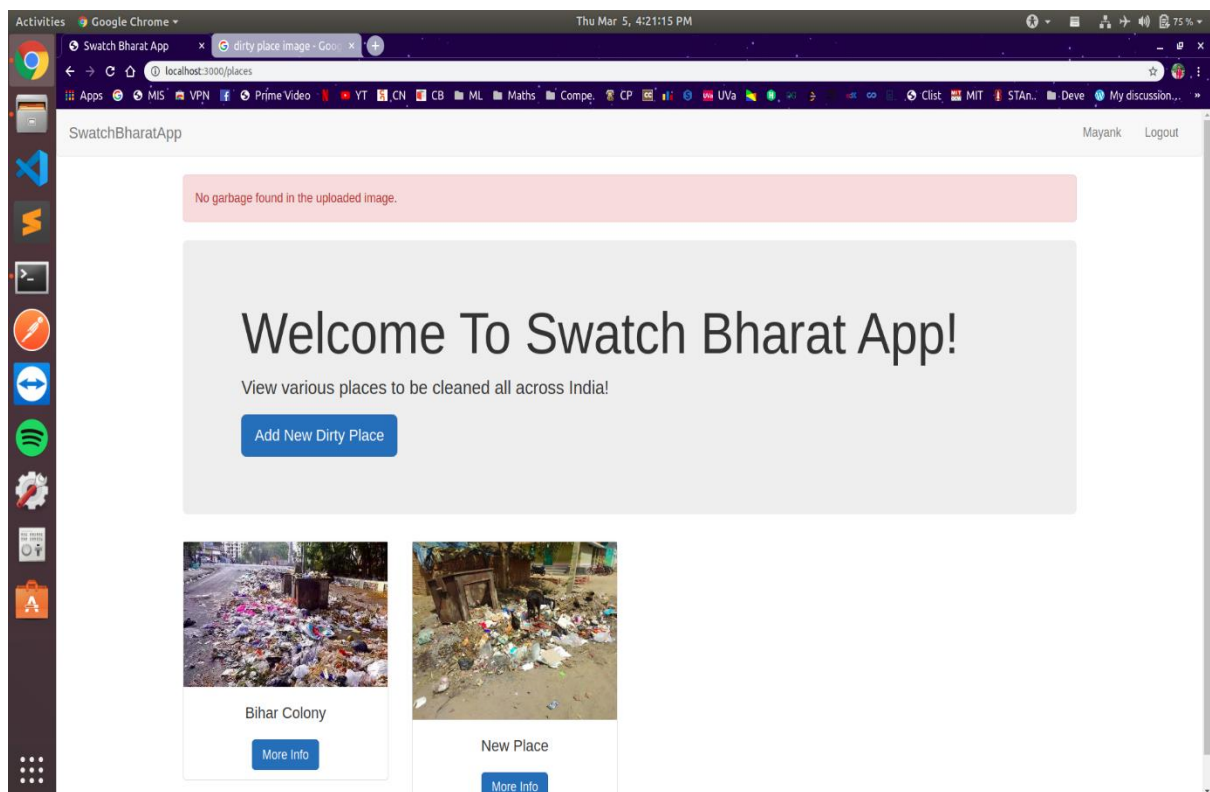
Now let's try uploading an image of a clean area.





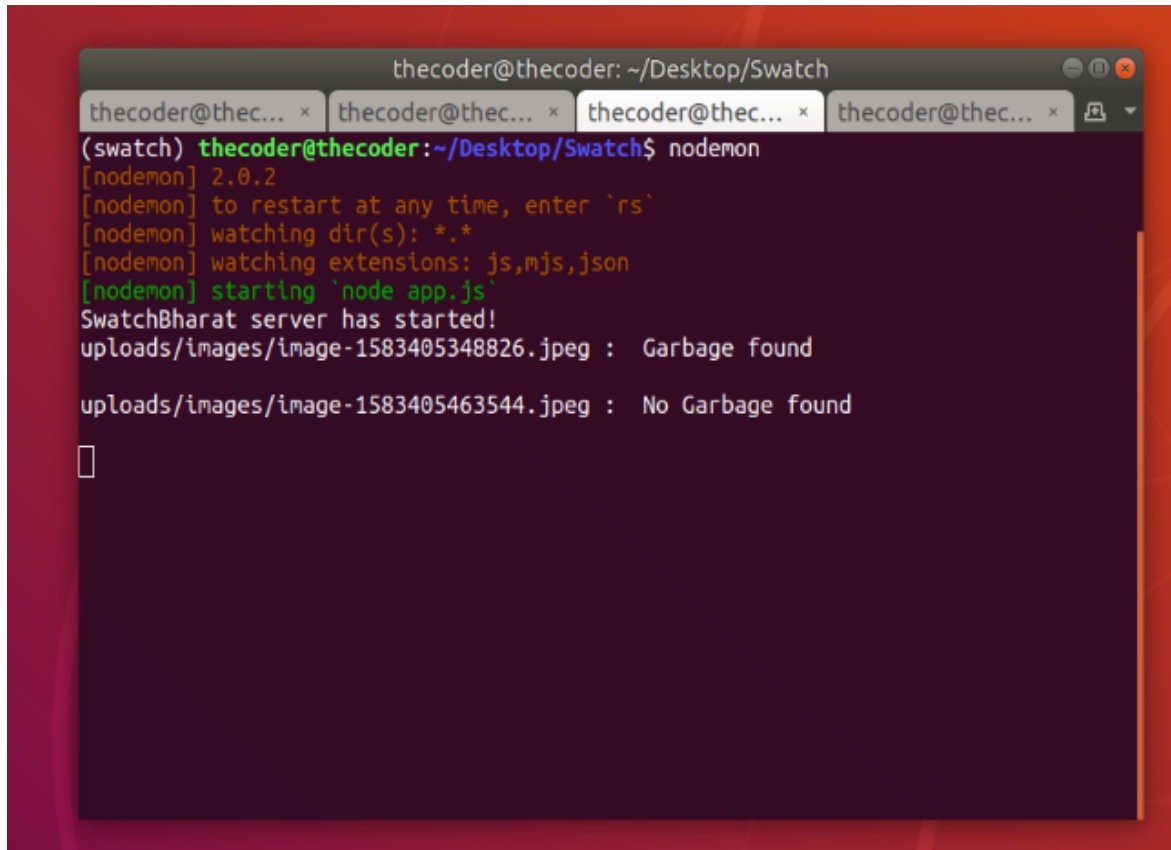


(clean.jpeg)



The error message has been displayed to the user that “No garbage found in the uploaded image”.

**Backend logs for the images were:**

A terminal window titled 'thecoder@thecoder: ~/Desktop/Swatch' with four tabs. The terminal shows the command 'nodemon' being executed. The output includes: '[nodemon] 2.0.2', '[nodemon] to restart at any time, enter `rs`', '[nodemon] watching dir(s): \*.\*', '[nodemon] watching extensions: js,mjs,json', and '[nodemon] starting `node app.js`'. Below this, a message says 'SwatchBharat server has started!'. Then, two file upload checks are shown: 'uploads/images/image-1583405348826.jpeg : Garbage found' and 'uploads/images/image-1583405463544.jpeg : No Garbage found'. The terminal ends with a cursor on a new line.

```
thecoder@thecoder: ~/Desktop/Swatch
thecoder@thec... x thecoder@thec... x thecoder@thec... x thecoder@thec... x
(swach) thecoder@thecoder:~/Desktop/Swatch$ nodemon
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
SwatchBharat server has started!
uploads/images/image-1583405348826.jpeg : Garbage found

uploads/images/image-1583405463544.jpeg : No Garbage found
█
```

The python script running at the server which has been called as a child\_process by node code runs and returns two outputs:

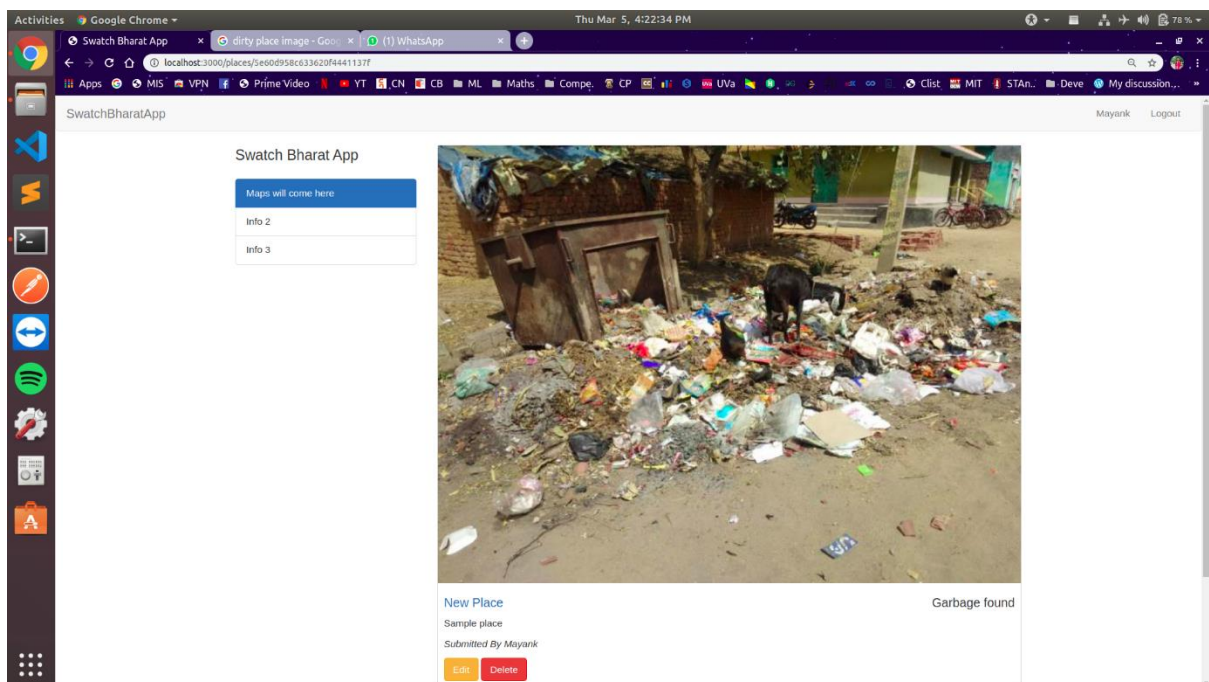
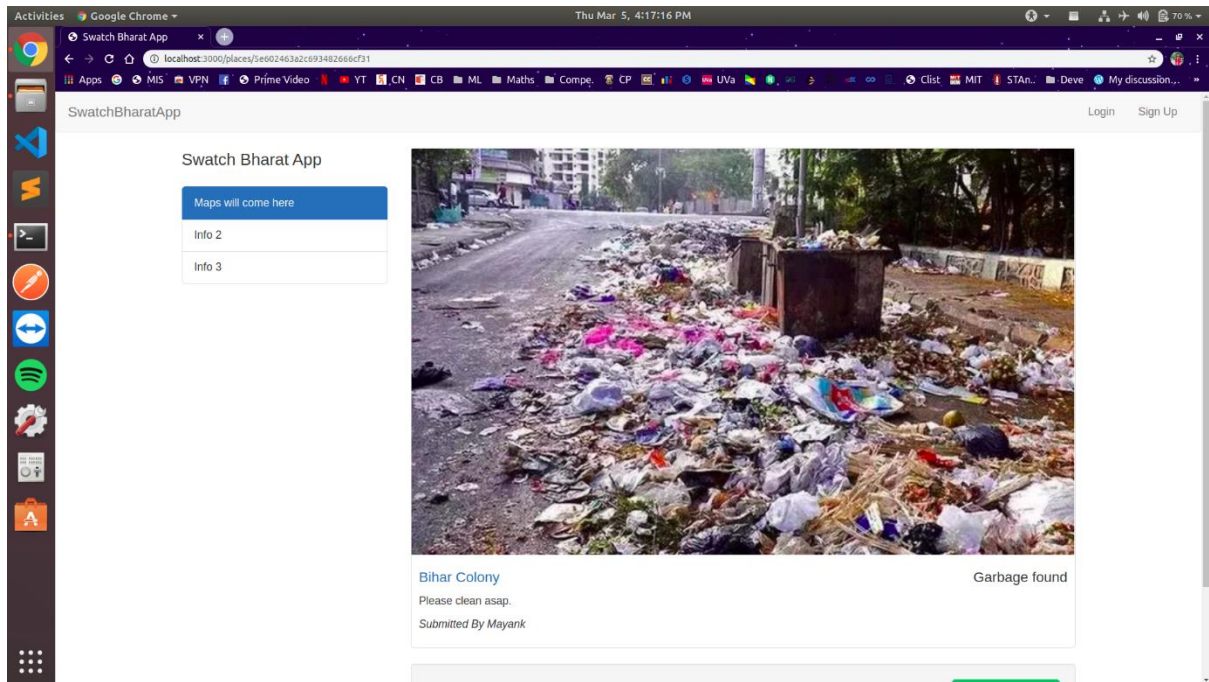
1. Garbage found
2. No Garbage found

These outputs are printed on the console/terminal at the server side and based on these results the node code decides whether to save the image in the server (if dirty) or discard the image and display a message to the user for the same.

**NOTE:** The main page is not rendered until the python script does not returns the output due to which it takes some time to render the main page once you upload an image.

## Show Page

Each entry has a button of “More Info” which redirects to the show page of corresponding place. Major information provided on the show page is the Name of user who uploaded, Image, Name of the place, a “Garbage found” tag.



**Remaining Work/features to be added:**

- Adding a map on the show page showing the location from where the image was uploaded and thus geotagging all the dirty place areas on a larger map. We are currently looking for a free API for the map.
- If map functionality is added the model schema for a place will be updated to have three new attributes namely:
  - Latitude
  - Longitude
  - Location – (this will be the address of the place which will be geotagged using the API.)
- The “Garbage Found” tag will be replaced with the garbage index for which it was originally kept. But garbage index feature is yet to be trained in our machine learning model.
- Some UI improvements.

## REFERENCES:

- **brilliant.org** //for learning the maths behind the algorithms we have used to train the data.
- **google images** // for collecting the dataset of images upon which we have trained our model.
- **hackerearth.com**//for learning the algorithms implementation details.
- **hackernoon.com**// for learning the algorithms.
- **coursera.com** //referred to machine learning and deep learning lectures by Andrew Ng.
- **udemy.com** // referred to implementation details from machine learning and deep learning courses a to z.  
// referred to implementation details from Web Developer Bootcamp
- **software.intel.com** // referred to deep convolutional architecture
- **keras.io** // referred for implementation of machine learning model.
- **tensorflow.org** // referred for implementation of machine learning model
- **blog.cambridgespark.com**//for learning how to link and host our trained model on our website
- **stackoverflow.com** // referred to integrate python script with node

## **GROUP INFORMATION**

### **Team Members and their admission numbers**

- |    |                      |            |
|----|----------------------|------------|
| 1. | Vinit Sharma         | 17JE002922 |
| 2. | Shivam Kumar         | 17JE002842 |
| 3. | Mayank Jain          | 17JE002935 |
| 4. | Utkarsh              | 17JE002849 |
| 5. | Dheeraj Kumar        | 17JE002837 |
| 6. | Ujjwal Mani Tripathi | 17JE002921 |